

Java Programming/Print version

Wikibooks.org

February 15, 2012

Contents

1	INTRODUCTION	3
1.1	ARE YOU NEW TO PROGRAMMING?	3
1.2	PROGRAMMING WITH JAVA™	4
2	ABOUT THIS BOOK	5
2.1	WHO SHOULD READ THIS BOOK?	5
2.2	HOW CAN YOU PARTICIPATE	6
2.3	NECESSARY PREREQUISITES	7
3	HISTORY	13
3.1	THE GREEN TEAM	13
3.2	RESHAPING THOUGHT	14
3.3	THE DEMISE OF AN IDEA, BIRTH OF ANOTHER	17
3.4	RECENT HISTORY	17
3.5	VERSIONS	17
3.6	CITATIONS	20
4	THE JAVA PLATFORM	21
4.1	JAVA RUNTIME ENVIRONMENT (JRE)	21
4.2	JAVA DEVELOPMENT KIT (JDK)	25
4.3	SIMILAR CONCEPTS	29
5	GETTING STARTED	33
5.1	GETTING STARTED	33
6	COMPILATION	35
6.1	COMPILING TO BYTECODE	35
6.2	AUTOMATIC COMPILATION OF DEPENDENT CLASSES	36
6.3	PACKAGES, SUBDIRECTORIES, AND RESOURCES	36
6.4	FILENAME CASE	37
6.5	COMPILER OPTIONS	37
6.6	ADDITIONAL TOOLS	37
6.7	JBUILDER	38
6.8	JCREATOR	38
6.9	ECLIPSE	38
6.10	NETBEANS	39
6.11	BLUEJ	39
6.12	KAWA	39
6.13	ANT	40
6.14	THE JIT COMPILER	42

7	EXECUTION	43
7.1	JSE CODE EXECUTION	43
7.2	J2EE CODE EXECUTION	46
7.3	JINI	49
8	UNDERSTANDING A JAVA PROGRAM	51
8.1	THE DISTANCE CLASS: INTENT, SOURCE, AND USE	51
8.2	DETAILED PROGRAM STRUCTURE AND OVERVIEW	52
8.3	COMMENTS IN JAVA PROGRAMS	60
9	SYNTAX	61
9.1	UNICODE	62
9.2	LITERALS	63
9.3	BLOCKS	64
9.4	WHITESPACES	65
9.5	REQUIRED WHITESPACE	65
9.6	INDENTATION	66
10	STATEMENTS	67
10.1	WHAT EXACTLY ARE STATEMENTS?	67
10.2	WHERE DO YOU FIND STATEMENTS	67
10.3	VARIABLES	67
10.4	DATA TYPES	68
10.5	WHOLE NUMBERS AND FLOATING POINT NUMBERS	68
10.6	ASSIGNMENT STATEMENTS	69
11	CLASSES, OBJECTS AND TYPES	71
11.1	OBJECTS AND CLASSES	71
11.2	INSTANTIATION AND CONSTRUCTORS	71
11.3	TYPE	72
11.4	MULTIPLE CLASSES IN A JAVA FILE	73
11.5	EXTERNAL LINKS	74
12	PACKAGES	75
12.1	JAVA PACKAGE / NAME SPACE	75
12.2	WILDCARD IMPORTS	76
12.3	IMPORTING PACKAGES FROM .JAR FILES	76
12.4	CLASS LOADING / NAME SPACE	77
13	NESTED CLASSES	79
13.1	NEST A CLASS INSIDE A CLASS	79
13.2	NEST A CLASS INSIDE A METHOD	80
13.3	ANONYMOUS CLASSES	80
14	ACCESS MODIFIERS	83
14.1	ACCESS MODIFIERS	83
15	METHODS	85
15.1	METHOD DEFINITION	85
15.2	METHOD OVERLOADING	85

15.3	METHOD OVERRIDING	87
15.4	PARAMETER PASSING	88
15.5	FUNCTIONS	89
15.6	RETURN PARAMETER	89
15.7	SPECIAL METHOD, THE CONSTRUCTOR	91
15.8	STATIC METHODS	92
15.9	EXTERNAL LINKS	93
16	PRIMITIVE TYPES	95
17	TYPES	97
17.1	DATA TYPES IN JAVA	97
17.2	JAVA AS HYBRID LANGUAGE	98
17.3	EXAMPLES OF TYPES	98
17.4	ARRAY TYPES	99
17.5	PRIMITIVE DATA TYPES	100
17.6	DATA CONVERSION (CASTING)	101
17.7	AUTOBOXING/UNBOXING	101
18	JAVA.LANG.STRING	103
18.1	JAVA.LANG.STRING	103
18.2	USING STRINGBUFFER/STRINGBUILDER TO CONCATENATE STRINGS	104
18.3	COMPARING STRINGS	105
18.4	SPLITTING A STRING	106
18.5	CREATING SUBSTRINGS	107
18.6	MODIFYING STRING CASES	107
18.7	SEE ALSO	108
19	ARRAYS	109
19.1	INTRO TO ARRAYS	109
19.2	ARRAY FUNDAMENTALS	109
19.3	TWO-DIMENSIONAL ARRAYS	110
19.4	MULTIDIMENSIONAL ARRAY	111
20	DATA AND VARIABLES	113
20.1	STRONG TYPING	114
20.2	CASE CONVENTIONS	114
20.3	SCOPE	114
21	GENERIC	117
21.1	WHAT ARE GENERICS?	117
21.2	INTRODUCTION	118
21.3	NOTE FOR C++ PROGRAMMERS	119
21.4	CLASS<T>	120
21.5	VARIABLE ARGUMENT	121
21.6	WILDCARD TYPES	122
22	DEFINING CLASSES	125
22.1	FUNDAMENTALS	125

23 CREATING OBJECTS	129
23.1 INTRODUCTION	129
23.2 CREATING OBJECT WITH THE new KEYWORD	129
23.3 CREATING OBJECT BY CLONING AN OBJECT	130
23.4 CREATING OBJECT RECEIVING FROM A REMOTE SOURCE	132
24 INTERFACES	135
24.1 INTERFACES	135
24.2 EXTERNAL LINKS	136
25 USING STATIC MEMBERS	137
25.1 WHAT DOES STATIC MEAN?	137
25.2 WHAT CAN IT BE USED FOR?	137
25.3 DANGER OF STATIC VARIABLES	138
25.4 EXTERNAL LINKS	138
26 DESTROYING OBJECTS	139
26.1 FINALIZE()	139
27 OVERLOADING METHODS AND CONSTRUCTORS	141
28 ARRAYS	143
28.1 INTRO TO ARRAYS	143
28.2 ARRAY FUNDAMENTALS	143
28.3 TWO-DIMENSIONAL ARRAYS	144
28.4 MULTIDIMENSIONAL ARRAY	145
29 COLLECTION CLASSES	147
29.1 INTRODUCTION TO COLLECTIONS	147
29.2 GENERICS	148
29.3 COLLECTION OR MAP	149
29.4 SET OR LIST OR QUEUE	151
29.5 MAP CLASSES	158
29.6 THREAD SAFE COLLECTIONS	159
29.7 CLASSES DIAGRAM (UML)	159
29.8 EXTERNAL LINKS	161
30 THROWING AND CATCHING EXCEPTIONS	163
30.1 EXCEPTION ARGUMENTS	164
30.2 CATCHING AN EXCEPTION	165
30.3 EXCEPTION HANDLERS	165
30.4 EXCEPTION CLASSES IN THE JCL	166
30.5 CATCH CLAUSES	168
30.6 EXAMPLE OF HANDLING EXCEPTIONS	169
30.7 APPLICATION EXCEPTIONS	170
30.8 RUNTIME EXCEPTIONS	171
30.9 KEYWORD REFERENCES	172
30.10 MINIMIZE THE USE OF THE KEYWORD 'NULL' IN ASSIGNMENT STATEMENTS	173
30.11 MINIMIZE THE USE OF THE NEW TYPE[INT] SYNTAX FOR CREATING ARRAYS OF OBJECTS	173

30.12	CHECK ALL REFERENCES OBTAINED FROM 'UNTRUSTED' METHODS	174
30.13	COMPARING STRING VARIABLE WITH A STRING LITERAL	174
30.14	SEE ALSO	175
31	LINKS	177
31.1	EXTERNAL REFERENCES	177
31.2	EXTERNAL LINKS	177
32	LICENSE	181
33	GNU FREE DOCUMENTATION LICENSE	183
34	AUTHORS	185
	LIST OF FIGURES	193

1 Introduction

The beautiful thing about learning is nobody can take it away from you

Learning a computer programming language is AKIN TO¹ a toddler's first few steps towards an upright walk. You stumble and you fall but when you start walking, you take upon it as SECOND NATURE². Learning to program is similar in many ways to learning to walk. However once you start programming, you never cease to evolve. Learn one programming language and you know them all.

This book is a concentrated effort in trying to get you to take your baby steps towards programming in the Java™ programming language. Although this book is primarily meant to act as a learning aid for beginners, it can equally be helpful as a reference manual and guide for the intermediate and experienced programmers. The book is laid out in such a manner that with each successive chapter, the complexity of programming constructs increase. Thus, programmers are constantly challenged to develop their skills with a progressive learning curve.

1.1 Are you new to programming?



Figure 1: This stunning image of the sunset on planet Mars wouldn't have been possible without Java.

If you have chosen Java as your first programming language, be assured that Java happens to also be the first choice for educational programs in many universities and colleges. Its simple and intuitive SYNTAX³ (that's grammar for programming languages) helps beginners feel at ease with complex programming constructs in no time.

But be aware, Java is not just any *basic* programming language. In fact, NASA used Java as the driving force (quite literally) behind its Mars Rover missions. Computers, mobile phones, set-top boxes, and even digital television sets can all be programmed using the Java programming

1 [HTTP://EN.WIKTIONARY.ORG/WIKI/AKIN](http://en.wiktionary.org/wiki/akin)

2 [HTTP://EN.WIKTIONARY.ORG/WIKI/SECOND%20NATURE](http://en.wiktionary.org/wiki/second%20nature)

3 [HTTP://EN.WIKTIONARY.ORG/WIKI/SYNTAX](http://en.wiktionary.org/wiki/syntax)

language. Robots, air traffic control systems and the self-checkout barcode scanners in your favourite supermarkets are all being programmed in Java.

1.2 Programming with Java™

By now, you might truly be able to grasp the power of the Java programming language – there is still lots more you can do with Java. Not every programmer gets to program applications that take unmanned vehicles onto other planets. Software that we encounter in our daily life is somewhat humble in that respect. Software programs and applications written in Java however cover vast area of the computing ecosphere. Here are just some examples of the ubiquitous nature of Java applications in real-life:

- [OPENOFFICE.ORG](http://en.wikipedia.org/wiki/OpenOffice.org)⁴, a desktop office management suite that rivals the Microsoft Office suite has been written in Java.
- The popular building game [MINECRAFT](http://en.wikipedia.org/wiki/Minecraft)⁵ is also written in Java.
- Online browser-based games like [RUNESCAPE](http://en.wikipedia.org/wiki/Runescape)⁶, a 3D massively multi-player online role playing game (MMORPG), run on graphics routines, 3D rendering and networking capabilities powered by the Java programming language.
- Two of the world's renowned digital video recorders, [TiVo](http://en.wikipedia.org/wiki/TiVo)⁷ and [BSkyB's SKY+](http://en.wikipedia.org/wiki/Sky%2B)⁸ use built-in live television recording software to record, rewind and play your favourite television shows. These applications make extensive use of the Java programming language.

The above mentioned application illustrates the reach and ubiquity of Java applications. Here's another fact – almost 80% of mobile phone vendors adopt Java as their primary platform for the development of applications. The second most widely used mobile-based operating system, [ANDROID](http://en.wikipedia.org/wiki/Android)⁹, uses Java as one of its key application platforms – developers are encouraged to develop application for Android in the Java programming language.

1.2.1 What can Java *not* do?

Well, to be honest, there is nothing that Java can not do. You can program just about anything in Java. This book would however get you acquainted with the basics of the language in order for you to create that masterpiece of a software you dream of creating one day.

People have written operating systems in Java, they have programmed robots in the language and managed power stations and conveyor belts in factories; thus, you can do anything and everything with Java. Your applications and code need not fear any bounds because with Java, possibilities are endless – all you need is *creativity*.

4 [HTTP://EN.WIKIPEDIA.ORG/WIKI/OPENOFFICE.ORG](http://en.wikipedia.org/wiki/OpenOffice.org)

5 [HTTP://EN.WIKIPEDIA.ORG/WIKI/MINECRAFT](http://en.wikipedia.org/wiki/Minecraft)

6 [HTTP://EN.WIKIPEDIA.ORG/WIKI/RUNESCAPE](http://en.wikipedia.org/wiki/Runescape)

7 [HTTP://EN.WIKIPEDIA.ORG/WIKI/TiVo](http://en.wikipedia.org/wiki/TiVo)

8 [HTTP://EN.WIKIPEDIA.ORG/WIKI/SKY%2B](http://en.wikipedia.org/wiki/Sky%2B)

9 [HTTP://EN.WIKIPEDIA.ORG/WIKI/ANDROID%20%28OPERATING%20SYSTEM%29](http://en.wikipedia.org/wiki/Android%20%28operating%20system%29)

2 About This Book

The **Java Programming** Wikibook is a shared effort in amassing a comprehensive guide of the complete Java platform - from programming advice and tutorials for the desktop computer to programming on mobile phones. The information presented in this book has been conceptualised with the combined efforts of various AUTHORS AND CONTRIBUTORS¹, and anonymous editors.

The primary purpose of this book is to teach the Java programming language to an audience of beginners, but its progressive layout of tutorials increasing in complexity, it can be just as helpful for intermediate and experienced programmers. Thus, this book is meant to be used as:

- a collection of tutorials building upon one another in a progressive manner;
- a guidebook for efficient programming with the Java programming language; and,
- a comprehensive manual resource for the advanced programmer.

This book is intended to be used in conjunction with various other online resources, such as:

- the JAVA PLATFORM API DOCUMENTATION²;
- the OFFICIAL JAVA WEBSITE³; and,
- active Java communities online, such as JAVA.NET⁴ and JAVARANCH⁵, etc.

2.1 Who should read this book?

Every thing you would need to know to write computer programs would be explained in this book. By the time you finish reading, you will find yourself proficient enough to tackle just about anything in Java and programs written using it. This book serves as the first few stepping stones of many you would need to cross the unfriendly waters of computer programming. We have put a lot of emphasis in structuring this book in a way that lets you start programming from scratch, with Java as your preferred language of choice. This book is designed for you if any one of the following is true.

- You are relatively new to programming and have heard how easy it is to learn Java.
- You had some BASIC⁶ or PASCAL⁷ in school, and have a grasp of basic programming and logic.
- You already know and have been introduced to programming in earlier versions of Java.

1 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FAUTHORS%20%26%20CONTRIBUTORS](http://en.wikibooks.org/wiki/Java%20Programming%2FAuthors%20%26%20Contributors)

2 [HTTP://DOWNLOAD.ORACLE.COM/JAVASE/7/DOCS/API/OVERVIEW-SUMMARY.HTML](http://download.oracle.com/javase/7/docs/api/overview-summary.html)

3 [HTTP://WWW.ORACLE.COM/US/TECHNOLOGIES/JAVA/INDEX.HTML](http://www.oracle.com/us/technologies/java/index.html)

4 [HTTP://HOME.JAVA.NET](http://home.java.net)

5 [HTTP://WWW.JAVARANCH.COM](http://www.javaranch.com)

6 [HTTP://EN.WIKIPEDIA.ORG/WIKI/BASIC](http://en.wikipedia.org/wiki/BASIC)

7 [HTTP://EN.WIKIPEDIA.ORG/WIKI/PASCAL%20%28PROGRAMMING%20LANGUAGE%29](http://en.wikipedia.org/wiki/Pascal%20%28Programming%20Language%29)

- You are an experienced developer and know how to program in other languages like C++⁸, VISUAL BASIC⁹, PYTHON¹⁰ or RUBY¹¹.
- You've heard that Java is great for web applications and web services programming.

Although, this book is generally meant to be for programmers who are beginning to learn programming; it can be highly beneficial for intermediate and advanced programmers who may have missed up on some vital information. By the end of this book, you would be able to solve any complicated problem and tackle it using the best of your learnt skills in Java. Once you finish, you are also encouraged to take upon ambitious programming projects of your own as you certainly would be able to do that as well.

This book assumes that the reader has no prior knowledge of programming in Java, or for that matter, any object-oriented programming language. The book makes it easier to understand development methodology as one reads through the book with practical examples and exercises at the end of each topic and module. Although, if you are a complete beginner, we would suggest that you move slowly through this book and practice each exercise at your pace.

2.2 How can you participate

Content is constantly being updated and enhanced in this book as is the nature of wiki-based content. This book is therefore in a constant state of evolution. Any Wikibooks users can participate in helping this book to a better standard as both a reader, or an author/contributor.

2.2.1 As a reader

If you are interested in reading the content present in this book, we encourage you to:

- share comments about the technical accuracy, content, or organisation of this book by telling the authors in the **Discussion** section for each page. You can find the link **Discussion** on each page in this book leading you to appropriate sections for discussion.
- leave a signature when providing feedback, writing comments, or giving suggestion on the **Discussion** pages. This can be achieved by appending -- ~~~~ to your messages. Do *not* add your signatures to the **Book** pages, they are only meant for the **Discussion** pages.
- review pages – at the bottom of every page of this book, you are likely to find controls that help you review and give feedback for pages. Content on each page should be rated according to the four Wikibooks quality metrics: *Reliability*, *Completeness*, *Neutrality* and *Presentation*.
- share news about the Java Programming Wikibook with your family and friends and let them know about this comprehensive Java guide online.
- become a contributing author, if you think that you have information that could fill in some missing gaps in this book.

8 [HTTP://EN.WIKIPEDIA.ORG/WIKI/C%2B%2B](http://en.wikipedia.org/wiki/C%2B%2B)

9 [HTTP://EN.WIKIPEDIA.ORG/WIKI/VISUAL%20BASIC](http://en.wikipedia.org/wiki/VISUAL%20BASIC)

10 [HTTP://EN.WIKIPEDIA.ORG/WIKI/PYTHON%20%28PROGRAMMING%20LANGUAGE%29](http://en.wikipedia.org/wiki/PYTHON%20%28PROGRAMMING%20LANGUAGE%29)

11 [HTTP://EN.WIKIPEDIA.ORG/WIKI/RUBY%20%28PROGRAMMING%20LANGUAGE%29](http://en.wikipedia.org/wiki/RUBY%20%28PROGRAMMING%20LANGUAGE%29)

2.2.2 As an author or contributor

If you intent on writing content for this book, you need to do the following:

- When writing content for this book, you can always pose as an anonymous author, however we recommend you sign-in into the Wikibooks website when doing so. It becomes easier to track and acknowledge changes to certain parts parts of the book. Furthermore, the opinions and views of logged-in users are given precedence over anonymous users.
- Once you have started contributing content for this book, make sure that you add your name to THE AUTHORS AND CONTRIBUTORS LIST¹².
- BE BOLD¹³ and try to follow the CONVENTIONS¹⁴ for this Wikibook. It is important that the conventions for this book be followed to the letter to make content consistent and reliable throughout.

2.3 Necessary prerequisites

2.3.1 Installing the Java platform

For in-depth information, see the chapter on THE JAVA PLATFORM¹⁵.

In order to make use of the content in this book, you would need to follow along each and every tutorial rather than simply reading through the book. But to do so, you would need access to a computer with the **Java platform** installed on it – the Java platform is the basic prerequisite for *running* and *developing* Java code, thus it is divided into two essential software:

- the **Java Runtime Environment (JRE)**, which is needed to *run* Java applications and applets; and,
- the **Java Development Kit (JDK)**, which is needed to *develop* those Java applications and applets.

However as a developer, you would only require the JDK which comes equipped with a JRE as well. Given below are installation instruction for the JDK for various operating systems:

2.3.2 For Windows

Download instructions

Some Windows based systems come built-in with the JRE, however for the purposes of writing Java code by following the tutorials in this book, you would require the JDK nevertheless. To acquire the latest JDK (version 7), you can manually [DOWNLOAD THE JAVA SOFTWARE¹⁶](#) from the Oracle website.

12 [HTTP://EN.WIKIBOOKS.ORG/WIKI/..%2FAuthors%20%26%20Contributors](http://en.wikibooks.org/wiki/..%2FAuthors%20%26%20Contributors)

13 [HTTP://EN.WIKIBOOKS.ORG/WIKI/Be%20Bold](http://en.wikibooks.org/wiki/Be%20Bold)

14 [HTTP://EN.WIKIBOOKS.ORG/WIKI/..%2FConventions](http://en.wikibooks.org/wiki/..%2FConventions)

15 [Chapter 4 on page 21](#)

16 [HTTP://WWW.ORACLE.COM/TECHNETWORK/JAVA/JAVASE/DOWNLOADS/INDEX.HTML](http://www.oracle.com/technetwork/java/javase/downloads/index.html)

For the convenience of our readers, the following table presents direct links to the latest JDK for the Windows operating system.



Operating system	Setup Installer	License
Windows x86	DOWNLOAD ¹⁷	ORACLE BINARY CODE LICENSE AGREEMENT ¹⁸
Windows x64	DOWNLOAD ¹⁹	ORACLE BINARY CODE LICENSE AGREEMENT ²⁰

You must follow the instructions for the setup installer wizard step-by-step with the default settings to ensure that Java is properly installed on your system. Once the setup is completed, it is highly recommended to restart your Windows operating system.

If you kept the default settings for the setup installer wizard, you JDK should now be installed at **C:\Program Files\Java\jdk1.7.0_01**. You would require the location to your **bin** folder at a later time – this is located at **C:\Program Files\Java\jdk1.7.0_01\bin**

Updating environment variables (Optional)

In order for you to start using the JDK compiler utility with the Command Prompt, you would need to set the environment variables that points to the **bin** folder of your recently installed JDK. Follow the steps below to permanently include your Java platform to your environment variables.

1. For Windows XP, click **Start > Control Panel > System**.
For Windows 2000, click **Start > Settings > Control Panel > System**.
For Windows Vista or Windows 7, click **Start > Control Panel > System and Maintenance > System**.
Alternatively, you can also press  +  to open the **Run** dialog. With the dialog open, type the following command at the prompt:

```
rundll32 shell32.dll,Control_RunDLL sysdm.cpl
```
2. Navigate to the **Advanced** tab on the top, and select **Environment Variables...**
3. Under **System variables**, select the variable named **Path** and click **Edit...**

¹⁷ [HTTP://DOWNLOAD.ORACLE.COM/OTN-PUB/JAVA/JDK/7U1-B08/JDK-7U1-WINDOWS-I586.EXE](http://download.oracle.com/otn-pub/java/jdk/7u1-b08/jdk-7u1-windows-i586.exe)

¹⁸ [HTTP://WWW.ORACLE.COM/TECHNETWORK/JAVA/JAVASEBUSINESS/DOCUMENTATION/JAVA-SE-BCL-LICENSE-430205.HTML](http://www.oracle.com/technetwork/java/javasebusiness/documentation/java-se-bcl-license-430205.html)



¹⁹ [HTTP://DOWNLOAD.ORACLE.COM/OTN-PUB/JAVA/JDK/7U1-B08/JDK-7U1-WINDOWS-X64.EXE](http://download.oracle.com/otn-pub/java/jdk/7u1-b08/jdk-7u1-windows-x64.exe)

²⁰ [HTTP://WWW.ORACLE.COM/TECHNETWORK/JAVA/JAVASEBUSINESS/DOCUMENTATION/JAVA-SE-BCL-LICENSE-430205.HTML](http://www.oracle.com/technetwork/java/javasebusiness/documentation/java-se-bcl-license-430205.html)

4. In the **Edit System Variable** dialog, go to the **Variable value** field. This field is a list of directory paths separated by semi-colons (;).
5. To add a new path, append the location of your JDK **bin** folder separated by a semi-colon (;).
6. Click **OK** on every opened dialog to save changes and get past to where you started.

Start writing code

Once you have successfully installed the JDK on your system, you are ready to program code in the Java programming language. However, to write code, you would need a decent text editor. Windows comes with a default text editor by default – **Notepad**. In order to use notepad to write code in Java, you need to follow the steps below:

1. Click **Start** › **All Programs** › **Accessories** › **Notepad** to invoke the application. Alternatively, you can also press  +  to open the **Run** dialog. With the dialog open, type the following command at the prompt:

`notepad`
2. Once the **Notepad** application has fired up, you can use the editor to write code for the Java programming language.

The problem with **Notepad** however is that it does not support developer-friendly features, such as SYNTAX HIGHLIGHTING²¹ and CODE COMPLETION²². These features are a vital part of the exercise of writing code. Nevertheless, there are a variety of different open-source editors available as alternatives to **Notepad** that support these features.

For the purposes of the tutorials in this book, the most recommended editor is the **Notepad++**, a free and open-source fully integrated text editor that supports syntax highlighting and code completion. You need to DOWNLOAD THE LATEST VERSION OF NOTEPAD++²³ in order to start writing code with the editor.

²¹ [HTTP://EN.WIKIPEDIA.ORG/WIKI/SYNTAX%20HIGHLIGHTING](http://en.wikipedia.org/wiki/Syntax%20highlighting)

²² [HTTP://EN.WIKIPEDIA.ORG/WIKI/CODE%20COMPLETION](http://en.wikipedia.org/wiki/Code%20completion)

²³ [HTTP://NOTEPAD-PLUS-PLUS.ORG](http://notepad-plus-plus.org)

Note:

Amongst others, there are many editors available online that are specifically designed to code Java applications. Such editors have countless other features that facilitate programming with Java, e.g., **DEBUGGING**^a and application design interfaces, etc. Text editors that have comprehensive features and utilities to facilitate programmers are called **Integrated Development Environments** or **IDEs**. Java programmers often recommend the two most widely used IDEs for Java programming needs – these are:

- NetBeans – [DOWNLOAD HERE](#)^b and [READ THE INSTALLATION INSTRUCTIONS HERE](#)^c.
- Eclipse – [Download 32-BIT HERE](#)^d and [64-BIT HERE](#)^e. You can [READ THE INSTALLATION INSTRUCTIONS HERE](#)^f.

a [HTTP://EN.WIKIPEDIA.ORG/WIKI/DEBUGGING](http://en.wikipedia.org/wiki/Debugging)

b [HTTP://DOWNLOAD.NETBEANS.ORG/NETBEANS/7.0.1/FINAL/BUNDLES/NETBEANS-7.0.1-ML-WINDOWS.EXE](http://download.netbeans.org/netbeans/7.0.1/final/bundles/netbeans-7.0.1-ml-windows.exe)

c [HTTP://NETBEANS.ORG/COMMUNITY/RELEASES/70/INSTALL.HTML](http://netbeans.org/community/releases/70/install.html)

d [HTTP://WWW.ECLIPSE.ORG/DOWNLOADS/DOWNLOAD.PHP?FILE=/TECHNOLOGY/EPP/DOWNLOADS/RELEASE/INDIGO/SR1/ECLIPSE-JAVA-INDIGO-SR1-WIN32.ZIP&R=1](http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/indigo/SR1/eclipse-java-indigo-SR1-win32.zip&r=1)

e [HTTP://WWW.ECLIPSE.ORG/DOWNLOADS/DOWNLOAD.PHP?FILE=/TECHNOLOGY/EPP/DOWNLOADS/RELEASE/INDIGO/SR1/ECLIPSE-JAVA-INDIGO-SR1-WIN32-X86_64.ZIP&R=1](http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/indigo/SR1/eclipse-java-indigo-SR1-win32-x86_64.zip&r=1)

f [HTTP://WIKI.ECLIPSE.ORG/ECLIPSE/INSTALLATION](http://wiki.eclipse.org/Eclipse/Installation)

2.3.3 For Linux

Installation using Terminal

Downloading and installing the Java platform on Linux machines (in particular Ubuntu Linux) is very easy and straight-forward. To use the terminal to download and install the Java platform, follow the instructions below.

1. For Ubuntu, go to **Application** › **Accessories** › **Terminal**.
Alternatively, you can press **Alt** + **F2** to open the **Run Application** window. At the prompt, type `xterm` or `gnome-terminal` to open the **Terminal** window.
2. At the prompt, write the following:

```
$ sudo apt-get install openjdk-7-jdk openjdk-7-jre openjdk-7-doc
```

3. All Java software should be installed and instantly available now.

Download instructions

Alternatively, you can manually [DOWNLOAD THE JAVA SOFTWARE](#)²⁴ from the Oracle website.

²⁴ [HTTP://WWW.ORACLE.COM/TECHNETWORK/JAVA/JAVASE/DOWNLOADS/INDEX.HTML](http://www.oracle.com/technetwork/java/javase/downloads/index.html)

For the convenience of our readers, the following table presents direct links to the latest JDK for the Linux operating system.

Operating system	RPM	Tarball	License
Linux x86	DOWNLOAD ²⁵	DOWNLOAD ²⁶	ORACLE BINARY CODE LICENSE AGREEMENT ²⁷
Linux x64	DOWNLOAD ²⁸	DOWNLOAD ²⁹	ORACLE BINARY CODE LICENSE AGREEMENT ³⁰

Start writing code

The most widely available text editor on Gnome desktops is the **Gedit** application, while on the KDE desktops, one can find **Kate**. However unlike Notepad on Windows, both these editors support syntax highlighting and code completion and therefore are sufficient for our purposes.

However, if you require a robust and standalone text-editor like the Notepad++ editor on Windows, you would require the use of the minimalistic editor loaded with features – **SciTE**. Follow the instructions below if you wish to install **SciTE**:

1. For Ubuntu, go to **Application › Accessories › Terminal**.
Alternatively, you can press **Alt + F2** to open the **Run Application** window. At the prompt, type `xterm` or `gnome-terminal` to open the **Terminal** window.
2. At the prompt, write the following:

```
$ sudo apt-get install scite
```
3. You should now be able to use the Scite editor for your programming needs.

²⁵ [HTTP://DOWNLOAD.ORACLE.COM/OTN-PUB/JAVA/JDK/7U1-B08/JDK-7U1-LINUX-I586.RPM](http://download.oracle.com/otn-pub/java/jdk/7u1-b08/jdk-7u1-linux-i586.rpm)

²⁶ [HTTP://DOWNLOAD.ORACLE.COM/OTN-PUB/JAVA/JDK/7U1-B08/JDK-7U1-LINUX-I586.TAR.GZ](http://download.oracle.com/otn-pub/java/jdk/7u1-b08/jdk-7u1-linux-i586.tar.gz)

²⁷ [HTTP://WWW.ORACLE.COM/TECHNETWORK/JAVA/JAVASEBUSINESS/DOCUMENTATION/JAVA-SE-BCL-LICENSE-430205.HTML](http://www.oracle.com/technetwork/java/javasebusiness/documentation/java-se-bcl-license-430205.html)

²⁸ [HTTP://DOWNLOAD.ORACLE.COM/OTN-PUB/JAVA/JDK/7U1-B08/JDK-7U1-LINUX-X64.RPM](http://download.oracle.com/otn-pub/java/jdk/7u1-b08/jdk-7u1-linux-x64.rpm)

²⁹ [HTTP://DOWNLOAD.ORACLE.COM/OTN-PUB/JAVA/JDK/7U1-B08/JDK-7U1-LINUX-X64.TAR.GZ](http://download.oracle.com/otn-pub/java/jdk/7u1-b08/jdk-7u1-linux-x64.tar.gz)

³⁰ [HTTP://WWW.ORACLE.COM/TECHNETWORK/JAVA/JAVASEBUSINESS/DOCUMENTATION/JAVA-SE-BCL-LICENSE-430205.HTML](http://www.oracle.com/technetwork/java/javasebusiness/documentation/java-se-bcl-license-430205.html)

Note:

Amongst others, there are many editors available online that are specifically designed to code Java applications. Such editors have countless other features that facilitate programming with Java, e.g., [DEBUGGING^a](#) and application design interfaces, etc. Text editors that have comprehensive features and utilities to facilitate programmers are called **Integrated Development Environments** or **IDEs**. Java programmers often recommend the two most widely used IDEs for Java programming needs – these are:

- NetBeans – [DOWNLOAD HERE^b](#) and [READ THE INSTALLATION INSTRUCTIONS HERE^c](#).
- Eclipse – [Download 32-BIT HERE^d](#) and [64-BIT HERE^e](#). You can [READ THE INSTALLATION INSTRUCTIONS HERE^f](#).

a [HTTP://EN.WIKIPEDIA.ORG/WIKI/DEBUGGING](http://en.wikipedia.org/wiki/Debugging)

b [HTTP://DOWNLOAD.NETBEANS.ORG/NETBEANS/7.0.1/FINAL/BUNDLES/NETBEANS-7.0.1-ML-LINUX.SH](http://download.netbeans.org/netbeans/7.0.1/final/bundles/netbeans-7.0.1-ml-linux.sh)

c [HTTP://NETBEANS.ORG/COMMUNITY/RELEASES/70/INSTALL.HTML](http://netbeans.org/community/releases/70/install.html)

d [HTTP://WWW.ECLIPSE.ORG/DOWNLOADS/DOWNLOAD.PHP?FILE=/TECHNOLOGY/EPP/DOWNLOADS/RELEASE/INDIGO/SR1/ECLIPSE-JAVA-INDIGO-SR1-LINUX-GTK.TAR.GZ&R=1](http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/indigo/SR1/eclipse-java-indigo-SR1-linux-gtk.tar.gz&r=1)

e [HTTP://WWW.ECLIPSE.ORG/DOWNLOADS/DOWNLOAD.PHP?FILE=/TECHNOLOGY/EPP/DOWNLOADS/RELEASE/INDIGO/SR1/ECLIPSE-JAVA-INDIGO-SR1-LINUX-GTK-X86_64.TAR.GZ&R=1](http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/indigo/SR1/eclipse-java-indigo-SR1-linux-gtk-x86_64.tar.gz&r=1)

f [HTTP://WIKI.ECLIPSE.ORG/ECLIPSE/INSTALLATION](http://wiki.eclipse.org/Eclipse/Installation)

2.3.4 For Mac OS

2.3.5 For Solaris

3 History

On 23 May 1995, John Gage, the director of the Science Office of the Sun Microsystems along with Marc Andreessen, co-founder and executive vice president at Netscape announced to an audience of SunWorld™ that Java technology wasn't a myth and that it was a reality and that it was going to be incorporated into Netscape Navigator.¹

At the time the total number of people working on Java was less than 30.³ This team would shape the future in the next decade and no one had any idea as to what was in store. From being the mind of an unmanned vehicle on Mars to the operating environment on most of the consumer electronics, e.g. cable set-top boxes, VCRs, toasters and also for personal digital assistants (PDAs).⁵ Java has come a long way from its inception. Let's see how it all began.

3.1 The Green team



Figure 2: James Gosling, architect and designer of the compiler for the Java technology

1 JAVA TECHNOLOGY: THE EARLY YEARS². Sun Microsystems . Retrieved 9 May 2008

3 JAVA TECHNOLOGY: THE EARLY YEARS⁴. Sun Microsystems . Retrieved 9 May 2008

5 HISTORY OF JAVA⁶. Lindsey, Clark S. . Retrieved 7 MAY^{[HTTP://EN.WIKIPEDIA.ORG/WIKI/7%20MAY](http://en.wikipedia.org/wiki/7%20MAY)} 2008^{[HTTP://EN.WIKIPEDIA.ORG/WIKI/2008](http://en.wikipedia.org/wiki/2008)}

Behind closed doors, a project was initiated in December of 1990⁷, whose aim was to create a programming tool that could render obsolete the C and C++ programming languages. Engineer Patrick Naughton had become extremely frustrated with the state of Sun's C++ and C APIs (application programming interfaces) and tools. While he was considering to move towards NEX⁸, he was offered a chance to work on new technology and the "Stealth Project" was started, a secret nobody but he knew.

This Stealth Project was later named the "Green Project" when James Gosling and Mike Sheridan joined Patrick.⁹ Over the period of time that the Green Project teathed, the prospects of the project started becoming clearer to the engineers working on it. No longer was its aim to create a new language far superior to the present ones, but it aimed to target the language to devices other than the computer.

Staffed at 13 people, they began work in a small office on Sand Hill Road in Menlo Park, California. This team would be called "Green Team" henceforth in time. The project they underwent was chartered by Sun Microsystems to anticipate and plan for the "next-wave" in computing. For the team, this meant at least one significant trend, that of the convergence of digitally controlled consumer devices and computers.¹¹

3.2 Reshaping thought

The team started thinking of replacing C++ with a better version, a faster version, a responsive version. But the one thing they hadn't thought of, as of yet, was that the language they were aiming for, had to be developed for an EMBEDDED SYSTEM¹³ with limited resources. An **embedded system** is a computer system scaled to a minimalistic interface demanding only a few functions from its design. For such a system, C++ or any successor would seem too large as all the languages at the time demanded a larger footprint than what was desired. And, other than this, the language lacked some other important features as well. The team thus had to think in a different way to go about solving all these problems.

Co-founder of Sun Microsystems, Bill Joy, envisioned a language combining the power of Mesa and C in a paper he wrote for the engineers at Sun named *Further*. Gathering ideas, Gosling began work on enhancing C++ and named it "C++ ++ --", a pun on the evolutionary structure of the language's name. The ++ and -- meant, *putting in* and *taking out stuff*. He soon abandoned the name and called it **Oak**¹⁴ after the tree that stood outside his office.

7 [HTTP://EN.WIKIPEDIA.ORG/WIKI/1990](http://en.wikipedia.org/wiki/1990)

8 [HTTP://EN.WIKIPEDIA.ORG/WIKI/NEXT](http://en.wikipedia.org/wiki/NEXT)

9 JAVA TECHNOLOGY: THE EARLY YEARS¹⁰. Sun Microsystems . Retrieved 9 May 2008

11 JAVA TECHNOLOGY: THE EARLY YEARS¹². Sun Microsystems . Retrieved 9 May 2008

13 [HTTP://EN.WIKIPEDIA.ORG/WIKI/EMBEDDED%20SYSTEM](http://en.wikipedia.org/wiki/Embedded%20system)

14 JAVA TECHNOLOGY: THE EARLY YEARS¹⁵. Sun Microsystems . Retrieved 9 May 2008

Table 1: Who's who of the Java tech- nology <small><ref name="CITEREFEarlyYearsSun1"/></small> Has worked for GT (Green Team), FP (FirstPer- son) and JP (Java Prod- ucts Group) Name	GT	FP	JP	Details
Lisa Friendly		✓ Yes	✓ Yes	FirstPerson employee and member of the Java Prod- ucts Group Science Office (Director), Sun Microsys- tems
John Gage				Lead engineer and key ar- chitect of the Java technol- ogy
James Gosling	✓ Yes	✓ Yes	✓ Yes	Co-founder and VP, Sun Microsys- tems; Princi- pal designer of the UC Berkeley, ver- sion of the UNIX® OS
Bill Joy				Java Products Group em- ployee, author of The Java FAQ1
Jonni Kanerva		✓ Yes		FirstPerson employee and member Java Products Group
Tim Lindholm		✓ Yes	✓ Yes	

Table 1: Who's who of the Java tech- nology <small><ref name="CITEREFEarlyYearsSun1"/></small> Has worked for GT (Green Team), FP (FirstPer- son) and JP (Java Prod- ucts Group)	GT	FP	JP	Details
Name Scott McNealy				Chairman, President, and CEO of Sun Microsystems
Patrick Naughton	✓ Yes	✓ Yes		Green Team member, FirstPerson co-founder
George Paolini				Corporate Marketing (Director), Sun's Java Software Divi- sion
Kim Polese		✓ Yes		FirstPerson product mar- keting
Lisa Poulson				Original di- rector of public re- lations for Java technol- ogy (Burson- Marsteller)
Wayne Rosing		✓ Yes		FirstPerson President
Eric Schmidt				Former Sun Microsystems Chief Tech- nology Officer
Mike Sheri- dan	✓ Yes			Green Team member

3.3 The demise of an idea, birth of another

By now, the work on Oak had been significant but come the year 1993, people saw the demise of set-top boxes, interactive TV and the PDAs. A failure that completely ushered the inventors' thoughts to be reinvented. Only a miracle could make the project a success now. And such a miracle awaited anticipation.

NATIONAL CENTER FOR SUPERCOMPUTING APPLICATIONS¹⁶ (NCSA) had just unveiled its new commercial web browser for the internet the previous year. The focus of the team, now diverted towards where they thought the "next-wave" of computing would be – the internet. The team then divulged into the realms of creating the same embeddable technology to be used in the web browser space calling it AN APPLE¹⁷ – *a small application*. The team now needed a proper identity and they decided on naming the new technology they created Java ushering a new generation of products for the internet boom. A by-product of the project was a cartoon named "DUKE¹⁸" created by Joe Parlang which became its identity then.

Finally at the SunWorldTM conference, Andreessen unveiled the new technology to the masses. Riding along with the explosion of interest and publicity in the Internet, Java quickly received widespread recognition and expectations grew for it to become the dominant software for browser and consumer applications.¹⁹

3.4 Recent history

Initially Java was owned by Sun Microsystems, but later it was released to open source; the term Java was a trademark of Sun Microsystems. Sun released the source code for its HotSpot Virtual Machine and compiler in November 2006, and most of the source code of the class library in May 2007. Some parts were missing because they were owned by third parties, not by Sun Microsystems. The released parts were published under the terms of the GNU GENERAL PUBLIC LICENSE²¹, a free software license.

3.5 Versions

Unlike C and C++, Java's growth is pretty recent. Here, we'd quickly go through the development paths that Java took with age.

16 [HTTP://EN.WIKIPEDIA.ORG/WIKI/NATIONAL%20CENTER%20FOR%20SUPERCOMPUTING%20APPLICATIONS](http://en.wikipedia.org/wiki/National%20Center%20for%20Supercomputing%20Applications)

17 [HTTP://EN.WIKIPEDIA.ORG/WIKI/APPLET](http://en.wikipedia.org/wiki/AppleT)

18 [HTTP://EN.WIKIPEDIA.ORG/WIKI/DUKE%20%28MASCOT%29](http://en.wikipedia.org/wiki/Duke%20%28mascot%29)

19 HISTORY OF JAVA²⁰. Lindsey, Clark S.. Retrieved 7 MAY[^]{[HTTP://EN.WIKIPEDIA.ORG/WIKI/7%20MAY%202008](http://en.wikipedia.org/wiki/7%20May%202008)[^]{[HTTP://EN.WIKIPEDIA.ORG/WIKI/2008](http://en.wikipedia.org/wiki/2008)}

21 [HTTP://EN.WIKIPEDIA.ORG/WIKI/GNU%20GENERAL%20PUBLIC%20LICENSE](http://en.wikipedia.org/wiki/GNU%20General%20Public%20License)

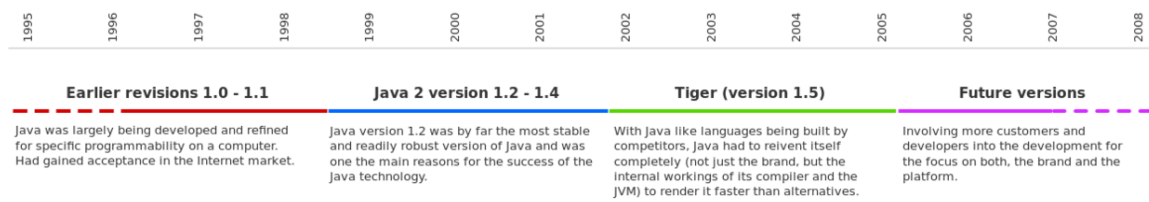


Figure 3: Development of Java over the years. From version 1.0 to version 1.7, Java has displayed a steady growth.

3.5.1 Initial Release (versions 1.0 and 1.1)

Introduced in 1996 for the SOLARIS²², WINDOWS²³, MAC OS²⁴ Classic and LINUX²⁵, Java was initially released as the Java Development Kit 1.0 (JDK 1.0). This included the Java runtime (the virtual machine and the class libraries), and the development tools (e.g., the Javac compiler). Later, Sun also provided a runtime-only package, called the Java Runtime Environment (JRE). The first name stuck, however, so usually people refer to a particular version of Java by its JDK version (e.g., JDK 1.0).

3.5.2 Java 2 (version 1.2)

Introduced in 1998 as a quick fix to the former versions, version 1.2 was the start of a new beginning for Java. The JDKs of version 1.2 and later versions are often called *Java 2* as well. For example, the official name of JDK 1.4 is *The Java(TM) 2 Platform, Standard Edition version 1.4*.

Major changes include:

- Rewrite the event handling (Add Event Listeners)
- Change Thread synchronizations
- Introduction of the JIT-Just in time compilers

3.5.3 Kestrel (Java 1.3)

3.5.4 Merlin (Java 1.4)

Java 1.4 has improved programmer productivity by expanding language features and available APIs

- Assertion
- Regular Expression

²² [HTTP://EN.WIKIPEDIA.ORG/WIKI/SOLARIS%20OPERATING%20ENVIRONMENT](http://en.wikipedia.org/wiki/Solaris%20operating%20environment)

²³ [HTTP://EN.WIKIPEDIA.ORG/WIKI/MICROSOFT%20WINDOWS](http://en.wikipedia.org/wiki/Microsoft%20Windows)

²⁴ [HTTP://EN.WIKIPEDIA.ORG/WIKI/MAC%20OS](http://en.wikipedia.org/wiki/Mac%20OS)

²⁵ [HTTP://EN.WIKIPEDIA.ORG/WIKI/LINUX](http://en.wikipedia.org/wiki/Linux)

- XML processing
- Cryptography and Secure Socket Layer (SSL)
- Non-blocking I/O(NIO)
- Logging

3.5.5 Tiger (version 1.5.0; Java SE 5)

Released in September 2004

Major changes include:

- **GENERIC²⁶** - Provides compile-time type safety for collections :and eliminates the drudgery of casting.
- **AUTOBOXING/UNBOXING²⁷** - Eliminates the drudgery of manual conversion between primitive types (such as int) and wrapper types (such as Integer).
- **Enhanced for** - Shorten the for loop with Collections use.
- **Static imports** - Lets you import all the static part of a class.
- **ANNOTATION²⁸/Metadata** - Enabling tools to generate code and deployment descriptors from annotations in the source code. This leads to a "declarative" programming style where the programmer says what should be done and tools emit the code to do it. Annotations can be inspected through source parsing or by using the additional reflection APIs added in Java 5.
- **JVM Improvements** - Most of the run time library is now mapped into memory as a memory image, as opposed to being loaded from a series of class files. Large portion of the runtime libraries will now be shared among multiple JVM instances.

(from [HTTP://JAVA.SUN.COM/FEATURES/2003/05/BLOCH_QA.HTML²⁹](http://java.sun.com/features/2003/05/bloch_qa.html))

3.5.6 Mustang (version 1.6.0; Java SE 6)

Released on 11 December 2006.³⁰

What's New in Java SE 6:

- **Web Services** - First-class support for writing XML web service client applications.
- **Scripting** - You can now mix in JavaScript technology source code, useful for prototyping. Also useful when you have teams with a variety of skill sets. More advanced developers can plug in their own scripting engines and mix their favorite scripting language in with Java code as they see fit.

²⁶ Chapter 21 on page 117

²⁷ [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FPRIMITIVE%20TYPES%23AUTOBOXING%2FUNBOXING](http://en.wikibooks.org/wiki/Java%20Programming%2FPrimitive%20Types%23Autoboxing%2Funboxing)

²⁸ [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FANNOTATION](http://en.wikibooks.org/wiki/Java%20Programming%2FAnnotation)

²⁹ [HTTP://JAVA.SUN.COM/FEATURES/2003/05/BLOCH_QA.HTML](http://java.sun.com/features/2003/05/bloch_qa.html)

³⁰ JAVA PLATFORM STANDARD EDITION 6³¹. Sun Microsystems . Retrieved 9 May 2008

- Database - No more need to find and configure your own JDBC database when developing a database application! Developers will also get the updated JDBC 4.0, a well-used API with many important improvements, such as special support for XML as an SQL datatype and better integration of Binary Large Objects (BLOBs) and Character Large Objects (CLOBs) into the APIs.
- More Desktop APIs - GUI developers get a large number of new tricks to play like the ever popular yet newly incorporated SwingWorker utility to help you with threading in GUI apps, JTable sorting and filtering, and a new facility for quick splash screens to quiet impatient users.
- Monitoring and Management - The really big deal here is that you don't need do anything special to the startup to be able to attach on demand with any of the monitoring and management tools in the Java SE platform.
- Compiler Access - Really aimed at people who create tools for Java development and for frameworks like JavaServer Pages (JSP) or Personal Home Page construction kit (PHP) engines that need to generate a bunch of classes on demand, the compiler API opens up programmatic access to javac for in-process compilation of dynamically generated Java code. The compiler API is not directly intended for the everyday developer, but for those of you deafened by your screaming inner geek, roll up your sleeves and give it a try. And the rest of us will happily benefit from the tools and the improved Java frameworks that use this.
- Pluggable Annotations allows programmer to write annotation processor so that it can analyse your code semantically before javac compiles. For example, you could write an annotation processor that verifies whether your program obeys naming conventions.
- Desktop Deployment - At long last, Java SE 6 unifies the Java Plug-in technology and Java WebStart engines, which just makes sense. Installation of the Java WebStart application got a much needed makeover.
- Security - Java SE 6 has simplified the job of its security administrators by providing various new ways to access platform-native security services, such as native Public Key Infrastructure (PKI) and cryptographic services on Microsoft Windows for secure authentication and communication, Java Generic Security Services (Java GSS) and Kerberos services for authentication, and access to LDAP servers for authenticating users.
- The -lities: Quality, Compatibility, Stability - Bug fixes ...

3.5.7 Dolphin (version 1.7.0; Java SE 7)

Anticipated for 2010

3.6 Citations

4 The Java Platform

As one of the necessary prerequisites for using this book to program in the Java programming language, we asked you download and install the **Java platform**. The **Java platform** is the name given to the computing platform from Oracle that helps users to *run* and *develop* Java applications. The platform does not just enable a user to run and develop Java application, but also features a wide variety of tools that can help developers work efficiently with the Java programming language.

The platform consists of two essential software:

- **the Java Runtime Environment (JRE)**, which is needed to *run* Java applications and applets; and,
- **the Java Development Kit (JDK)**, which is needed to *develop* those Java applications and applets. If you have installed the JDK, you should know that it comes equipped with a JRE as well. So, for all the purposes of this book, you would only require the JDK.

In this section, we would explore in further detail what these two software components of the Java platform do.

4.1 Java Runtime Environment (JRE)

Any piece of code written in the Java programming language can be run on any operating system, platform or architecture – in fact, it can be run on any device that supports the Java platform. Before Java, this amount of ubiquity was very hard to achieve. If a software was written for a Unix-based system, it was impossible to run the same application on a Windows system – in this case, the application was native only to Unix-based systems.

A major milestone in the development of the Java programming language was to develop a special runtime environment that would execute any Java application *independent* of the computer's operating system, platform or architecture.

The **Java Runtime Environment (JRE)** sits on top of the machine's operating system, platform and architecture. If and when a Java application is run, the JRE acts as a liaison between the underlying platform and that application. It interprets the Java application to run in accordance with the underlying platform, such that upon running the application, it looks and behaves like a native application. The part of the JRE that accomplishes this complex liaison agreement is called the **Java Virtual Machine (JVM)**. We will discuss the JVM in detail later.

Figure 1 : Java applications can be *written once* and *run anywhere*. This feature of the Java platform

is commonly abbreviated to **WORA** in formal Java texts.

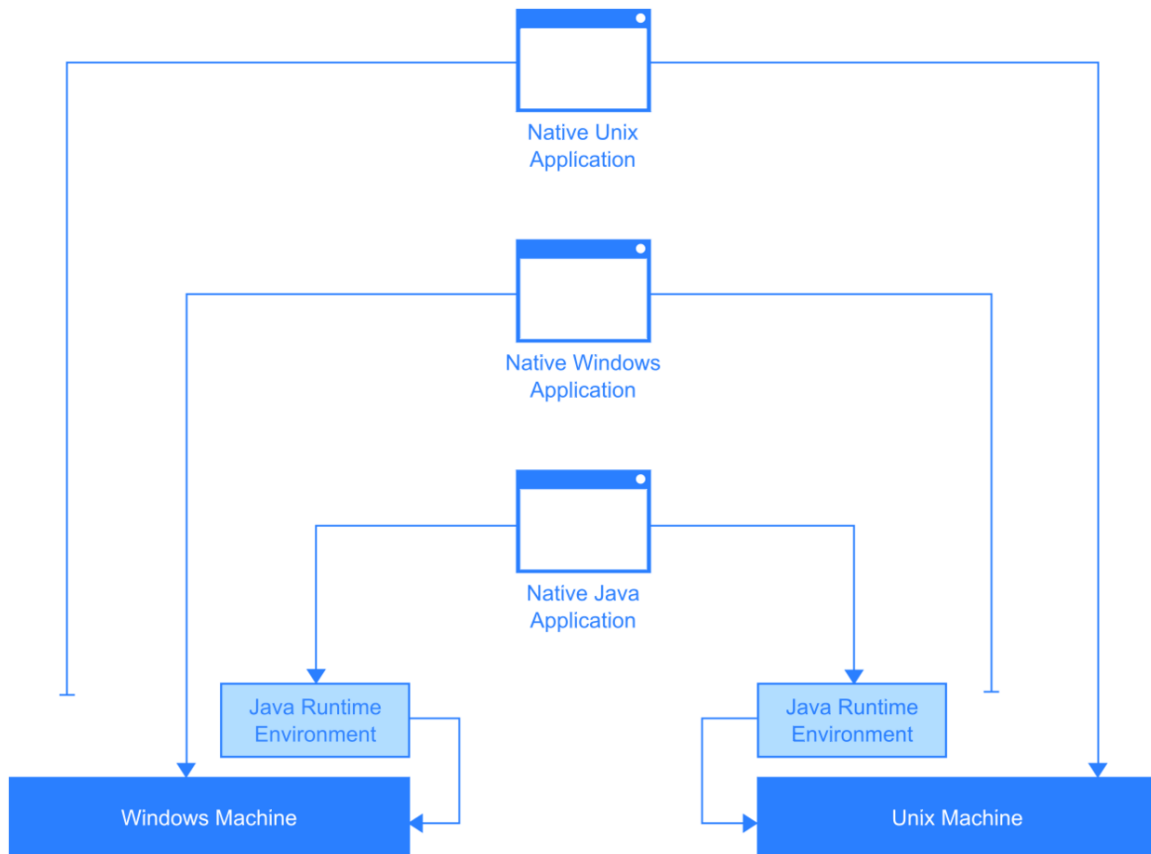


Figure 4

4.1.1 Executing native Java code (or *byte-code*)

Native Java applications are preserved in a special format called the `BYTE-CODE`¹. Byte-code remains the same, no matter what hardware architecture, operating system, or software platform it is running under. On a file-system, Java byte-code resides in files that have the `.class` (also known as a *class file*) or the `.jar` (also known as a *Java archive*) extension. To run byte-code, the JRE comes with a special tool (appropriately named `java`).

Suppose your byte-code is called `SomeApplication.class`. If you want to execute this Java byte-code, you would need to use the following command in Command Prompt (on Windows) and Terminal (on Linux or Mac OS):

```
java SomeApplication.class
```

If you want to execute a Java byte-code with a `.jar` extension (say, `SomeApplication.jar`), you would need to use the following command in Command Prompt (on Windows) and Terminal (on Linux or Mac OS):

```
java -jar SomeApplication.jar
```

¹ [HTTP://EN.WIKIPEDIA.ORG/WIKI/JAVA%20BYTECODE](http://en.wikipedia.org/wiki/Java%20bytecode)

Note:

Not all Java class files or Java archives are executable. Therefore, the **java** tool would only be able to execute files that are executable. Non-executable class files and Java archives are simply called *class libraries*.

4.1.2 Do you have a JRE?

Most computers come with pre-installed copy of the JRE. If your computer doesn't have a JRE, then the above commands would not work. You can always check what version of the JRE is installed on the computer by writing the following command in Command Prompt (on Windows) and Terminal (on Linux or Mac OS):

```
java -version
```

4.1.3 Java Virtual Machine (JVM)

Quite possibly, the most important part of the JRE is the **Java Virtual Machine (JVM)**. The JVM acts like a *virtual* processor enabling Java applications to be run on the local system. Its main purpose is to interpret (*read* translate) the received byte-code and make it appear as native code. The olden Java architecture used this process of **interpretation** to execute Java byte-code. Even though the process of interpretation brought the WORA principle to diverse machines, it had a drawback – it consumed a lot of time and clocked the system processor intensively to load an application.

Figure 2 : A JVM interpreter *translates* the byte-code line-by-line to make it appear as if a native application is being executed.

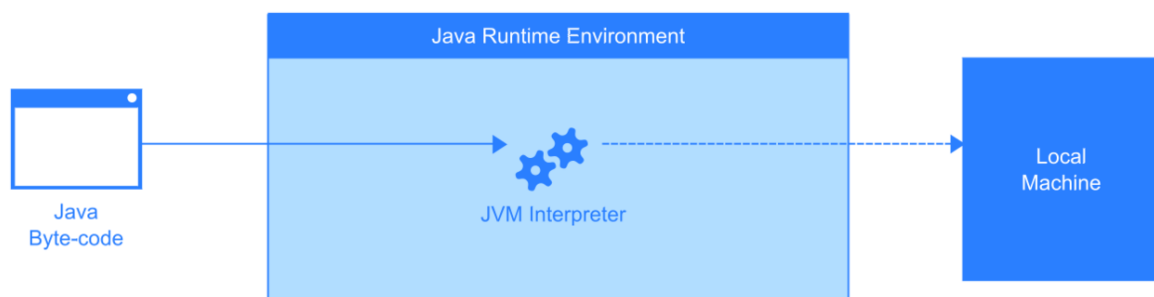


Figure 5

Just-in-Time Compilation

The new versions of the JRE (since version 1.2) features a more robust JVM. Instead of interpreting byte-code, it down-right converts the code straight into equivalent native code for the local system. This process of conversion is called **JUST-IN-TIME COMPILATION²** or **JIT-compilation**.

² [HTTP://EN.WIKIPEDIA.ORG/WIKI/JUST-IN-TIME%20COMPILATION](http://en.wikipedia.org/wiki/Just-in-time%20compilation)

This process only occurs when the byte-code is executed for the first time. Unless the byte-code itself is changed, the JVM uses the compiled version of the byte-code on every successive execution. Doing so saves a lot of time and processor effort, allowing applications to execute much faster at the cost of a *small* delay on first execution.

Figure 3 : A just-in-time compiler only compiles the byte-code to equivalent native code at first execution. Upon every successive execution, the JVM merely uses the already *compiled* native code to optimize performance.

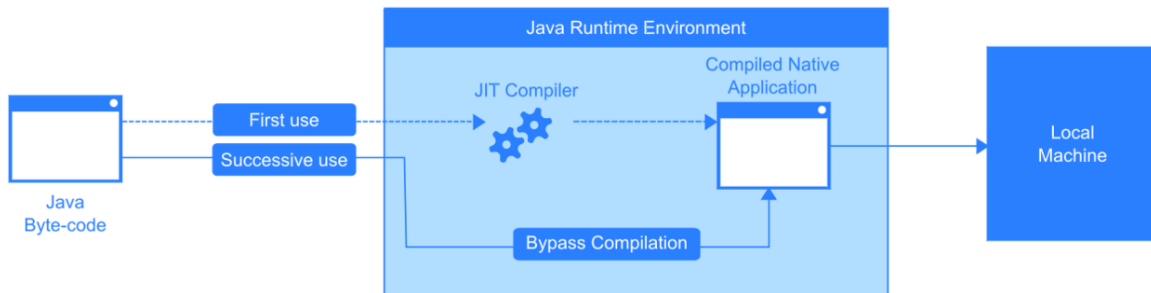


Figure 6

Native optimization

The JVM is an intelligent *virtual* processor. It has the ability to identify areas within the Java code itself that can be optimized for faster and better performance. Based on every successive run of your Java applications, the JVM would optimize it to run even better.

Note:

There are portions of Java code that do not require to be JIT-compiled at runtime, e.g., the Reflection API; therefore, code that uses such functions are not necessarily fully compiled to native code.

Was JVM the first *virtual machine*?

Java was not the first virtual-machine-based platform, though it is by far the most successful and well-known. Previous uses for virtual machine technology primarily involved EMULATORS³ to aid development for not-yet-developed hardware or operating systems, but the JVM was designed to be implemented entirely in software, while making it easy to efficiently port an implementation to hardware of all kinds.

3 [HTTP://EN.WIKIPEDIA.ORG/WIKI/EMULATORS](http://en.wikipedia.org/wiki/emulators)

4.2 Java Development Kit (JDK)

The JRE takes care of running the Java code on multiple platforms, however as developers, we are interested in writing pure code in Java which can then be converted into Java byte-code for mass deployment. As developers, we do *not* need to write Java byte-code, rather we write the code in the Java programming language (which is quite similar to writing C or C++ code).

Upon downloading the JDK, a developer ensures that their system has the appropriate JRE and additional tools to help with the development of applications in the Java programming language. Java code can be found in files with the extension **.java**. These files are called *Java source files*. In order to convert the Java code in these source files to Java byte-code, you need to use the **Java compiler** tool installed with your JDK.

4.2.1 The Java compiler

The **Java compiler** tool (named **javac** in the JDK) is the most important utility found with the JDK. In order to compile a Java source file (*say*, `SomeApplication.java`) to its respective Java byte-code, you would need to use the following command in Command Prompt (on Windows) and Terminal (on Linux or Mac OS):

```
javac SomeApplication.java
```

This command would convert the `SomeApplication.java` source file into its equivalent Java byte-code. The resultant byte-code would exist in a newly created file named `SomeApplication.class`. This process of converting Java source files into their equivalent byte-codes is known as *compilation*.

Figure 4 : The basic Java compilation process

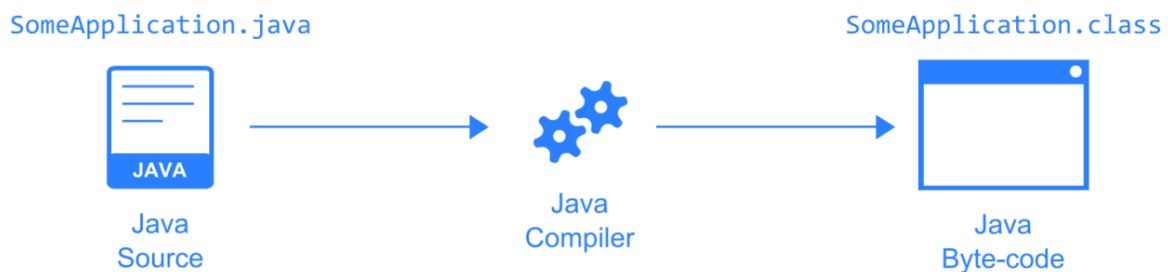


Figure 7

There are a huge array of tools available with the JDK that will all be explained in due time as you progress with the book. For the reader's convenience, these tools are listed below in order of their usage:

4.2.2 Applet development

- **appletviewer** – Java applets require a particular environment to execute. Typically, this environment is provided by a browser with a Java plug-in, and a web server serving the applet.

However, during development and testing of an applet it might be more convenient to start an applet without the need to fiddle with a browser and a web server. In such a case, Sun's appletviewer from the JDK can be used to run an applet.

4.2.3 Annotation processing

For more about annotation processing, READ THIS⁴

In Java 1.5 (alias Java 5.0) Sun added a mechanism called annotations. Annotations allow the addition of meta-data to Java source code, and even provide mechanisms to carry that meta-data forth into a compiled class files.

- **apt** – An annotation processing tool which digs through source code, finds annotation statements in the source code and executes actions if it finds known annotations. The most common task is to generate some particular source code. The actions apt performs when finding annotations in the source code are not hard-coded into apt. Instead, one has to code particular annotation handlers (in Java). These handlers are called annotation processors. The most difficult thing with apt is that Sun decided to use a whole set of new terminology. apt can simply be seen as a source code preprocessor framework, and annotation processors are typically just code generators.

4.2.4 Integration of non-Java and Java code

- **javah** – A Java class can call native, or non-Java, code that has been prepared to be called from Java. The details and procedures are specified in the JNI (Java Native Interface). Commonly, native code is written in C (or C++). The JDK tool javah helps to write the necessary C code, by generating C header files and C stub code.

4.2.5 Class library conflicts

- **extcheck** – This tool appeared first with Java 1.5. It can be used prior to the installation of a Java extension into the JDK or JRE environment. It checks if a particular Jar file conflicts with an already installed extension.

4.2.6 Software security and cryptography tools

The JDK comes with a large number of tools related to the security features of Java. Usage of these tools first requires study of the particular security mechanisms. The tools are:

- **keytool** – To manage keys and certificates
- **jarsigner** – To generate and verify digital signatures of JARs (Java ARchives)
- **policytool** – To edit policy files
- **kinit** – To obtain Kerberos v5 tickets

⁴ [HTTP://JAVA.SUN.COM/J2SE/1.5.0/DOCS/GUIDE/APT/GETTINGSTARTED.HTML](http://java.sun.com/j2se/1.5.0/docs/guide/apt/gettingstarted.html)

- **klist** – To manage Kerberos credential cache and key table
- **ktab** – To manage entries in a key table

4.2.7 The Java archiver

- **jar** – (short for Java archiver) is a tool for creating Java archives or jar files - a file with .jar as the extension. A Java archive is a collection of compiled Java classes and other resources which those classes may require (such as text files, configuration files, images) at runtime. Internally, a jar file is really a .ZIP FILE⁵.

4.2.8 The Java debugger

- **jdb** – (short for Java debugger) is a command-line console that provides a DEBUGGING⁶ environment for Java programs. Although you can use this command line console, IDE's normally provide easier to use debugging environments.

4.2.9 Documenting code with Java

As programs grow large and complex, programmers need ways to track changes and to understand the code better at each step of its evolution. For decades, programmer have been employing the use of special programming constructs called comments - regions that help declare user definitions for a code snippet within the source code. But comments are prone to be verbose and incomprehensible, let alone be difficult to read in applications having hundreds of lines of code.

- **javadoc** – Java provides the user with a way to easily publish documentation about the code using a special commenting system and the javadoc tool. The javadoc tool generates documentation about the application programming interface (API) of a set of user-created Java classes. javadoc reads source file comments from the .java source files and generates HTML documents that are easier to read and understand without looking at the code itself.
- **javap** – Where Javadoc provide a detailed view into the API and documentation of a Java class, the javap tool prints information regarding members (constructors, methods and variables) in a class. In other words, it lists the class' API and/or the compiled instructions of the class. javap is a formatting disassembler for Java bytecode.

4.2.10 The native2ascii tool

native2ascii is an important, though underappreciated, tool for writing properties files -- files containing configuration data -- or resource bundles -- files containing language translations of text.

⁵ [HTTP://EN.WIKIPEDIA.ORG/WIKI/ZIP%20%28FILE%20FORMAT%29](http://en.wikipedia.org/wiki/ZIP%20%28FILE%20FORMAT%29)

⁶ [HTTP://EN.WIKIPEDIA.ORG/WIKI/DEBUGGING](http://en.wikipedia.org/wiki/DEBUGGING)

Such files can contain only ASCII and Latin-1 characters, but international programmers need a full range of character sets. Text using these characters can appear in properties files and resource bundles only if the non-ASCII and non-Latin-1 characters are converted into Unicode escape sequences (\uXXXX notation).

The task of writing such escape sequences is handled by `native2ascii`. You can write the international text in an editor using the appropriate character encoding, then use `native2ascii` to generate the necessary ASCII text with embedded Unicode escape sequences. Despite the name, `native2ascii` can also convert from ASCII to native, so it is useful for converting an existing properties file or resource bundle back to some other encoding.

`native2ascii` makes most sense when integrated into a build system to automate the conversion.

4.2.11 Remote Method Invocation (RMI) tools

4.2.12 Java IDL and RMI-IIOP Tools

4.2.13 Deployment & Web Start Tools

4.2.14 Browser Plug-In Tools

4.2.15 Monitoring and Management Tools / Troubleshooting Tools

With Java 1.5 a set of monitoring and management tools have been added to the JDK, in addition to a set of troubleshooting tools.

The monitoring and management tools are intended for monitoring and managing the virtual machine and the execution environment. They allow, for example, monitoring memory usage during the execution of a Java program.

The troubleshooting tools provide rather esoteric insight into aspects of the virtual machine. (Interestingly, the Java debugger is not categorized as a troubleshooting tool.)

All the monitoring and management and troubleshooting tools are currently marked as "experimental" (which does not affect `jdb`). So they might disappear in future JDKs.

4.2.16 Java class libraries (JCL)

In most modern operating systems, a large body of reusable code is provided to simplify the programmer's job. This code is typically provided as a set of `DYNAMICALLY LOADABLE LIBRARIES`⁷ that applications can call at runtime. Because the Java platform is not dependent on any specific operating system, applications cannot rely on any of the existing libraries. Instead, the Java platform provides a comprehensive set of standard class libraries, containing much of the same reusable functions commonly found in modern operating systems.

⁷ [HTTP://EN.WIKIPEDIA.ORG/WIKI/LIBRARY%20%28COMPUTER%20SCIENCE%29%23DYNAMIC%20LINKING](http://en.wikipedia.org/wiki/Library%20%28computer%20science%29%23dynamic%20linking)

The Java class libraries serve three purposes within the Java platform. Like other standard code libraries, they provide the programmer with a well-known set of functions to perform common tasks, such as maintaining lists of items or performing complex string parsing. In addition, the class libraries provide an abstract interface to tasks that would normally depend heavily on the hardware and operating system. Tasks such as network access and file access are often heavily dependent on the native capabilities of the platform. The Java `java.net` and `java.io` libraries implement the required native code internally, then provide a standard interface for the Java applications to perform those tasks. Finally, some underlying platforms may not support all of the features a Java application expects. In these cases, the class libraries can either emulate those features using whatever is available, or provide a consistent way to check for the presence of a specific feature.

4.3 Similar concepts

The success of the Java platform and the concepts of the WRITE ONCE, RUN ANYWHERE⁸ principle has led to the development of similar frameworks and platforms. Most notable of these is the Microsoft's .NET FRAMEWORK⁹ and its open-source equivalent MONO¹⁰.

4.3.1 The .NET framework

The .NET framework borrows many of the concepts and innovations of Java – their alternative for the JVM is called the COMMON LANGUAGE RUNTIME (CLR)¹¹, while their alternative for the byte-code is the COMMON INTERMEDIATE LANGUAGE (CIL)¹². In fact, the .NET platform had an implementation of a Java-like language called VISUAL J#¹³ (formerly known as J++¹⁴).

J# is normally not supported with the JVM because instead of compiling it in Java byte-code, the .NET platform compiles the code into CIL, thus making J# different from the Java programming language. Furthermore, because J# implements the .NET Base Class Libraries (BCL) instead of the Java Class Libraries, J# is nothing more than a non-standard extension of the Java programming language. Due to the lack of interest from developers, Microsoft had to withdraw their support for J#, and focused on a similar programming language – C#.

To compare Java with C#, read COMPARISON OF C# AND JAVA¹⁵.

8 [HTTP://EN.WIKIPEDIA.ORG/WIKI/WRITE%20ONCE%2C%20RUN%20ANYWHERE](http://en.wikipedia.org/wiki/Write%20once%2C%20run%20anywhere)

9 [HTTP://EN.WIKIPEDIA.ORG/WIKI/.NET%20FRAMEWORK](http://en.wikipedia.org/wiki/.NET%20framework)

10 [HTTP://EN.WIKIPEDIA.ORG/WIKI/MONO%20%28SOFTWARE%29](http://en.wikipedia.org/wiki/Mono%20%28software%29)

11 [HTTP://EN.WIKIPEDIA.ORG/WIKI/COMMON%20LANGUAGE%20RUNTIME](http://en.wikipedia.org/wiki/Common%20language%20runtime)

12 [HTTP://EN.WIKIPEDIA.ORG/WIKI/COMMON%20INTERMEDIATE%20LANGUAGE](http://en.wikipedia.org/wiki/Common%20intermediate%20language)

13 [HTTP://EN.WIKIPEDIA.ORG/WIKI/J%20SHARP](http://en.wikipedia.org/wiki/J%20sharp)

14 [HTTP://EN.WIKIPEDIA.ORG/WIKI/J%20PLUS%20PLUS](http://en.wikipedia.org/wiki/J%20plus%20plus)

15 [HTTP://EN.WIKIPEDIA.ORG/WIKI/COMPARISON%20OF%20C%20SHARP%20AND%20JAVA](http://en.wikipedia.org/wiki/Comparison%20of%20C%20sharp%20and%20Java)

4.3.2 Third-party compilers targeting the JVM

The word Java, by itself, usually refers to the Java programming language which was designed for use with the Java platform. Programming languages are typically outside of the scope of the phrase "platform". However, Oracle does not encourage the use of any other languages with the platform, and lists the Java programming language as a core part of the Java 2 platform. The language and runtime are therefore commonly considered a single unit.

There are cases where you might want to program using a different language (say, PYTHON¹⁶) and yet be able to generate Java byte-code (instead of the Python compiled code) to be run with the JVM. Many third-party programming language vendors provide compilers that can compile code written in their language to Java byte-code. For instance, Python developers can use JYTHON¹⁷ compilers to compile Python code to the Java byte-code format (as illustrated below).

Figure 5 : Third-party JVM-targeted compilation for non-Java source compilation to Java byte-code. Illustrated example shows Python source being compiled to both Python compiled code and Java byte-code.

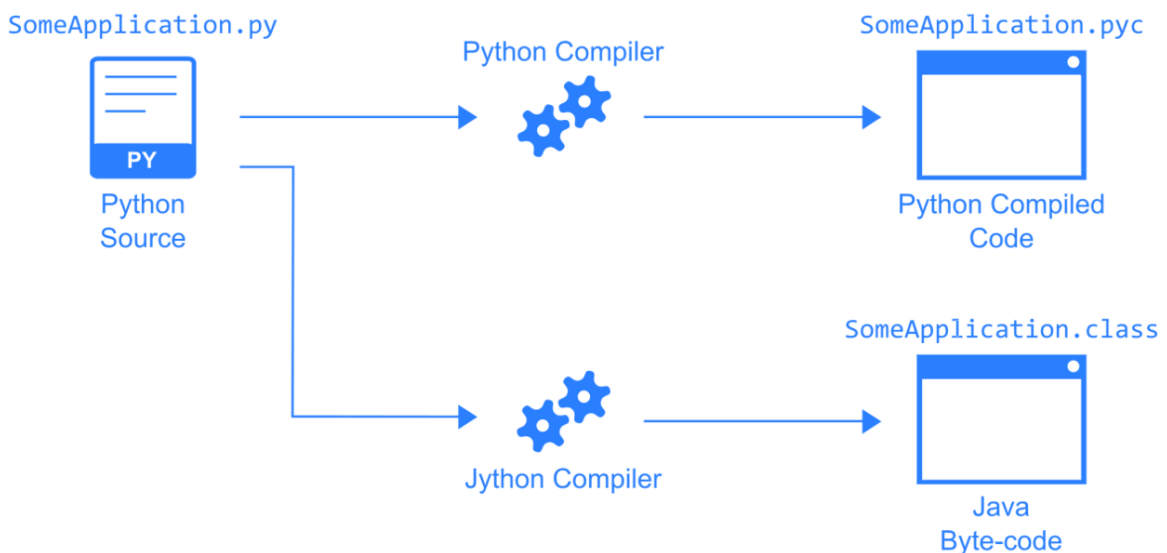


Figure 8

Of late, JVM-targeted third-party programming and scripting languages have seen tremendous growth. Some of these languages are also used to extend the functionalities of the Java language itself. A few examples include the following:

- GROOVY¹⁸
- PIZZA¹⁹
- GJ²⁰ (Generic Java) – later officially incorporated into Java SE 5.

16 [HTTP://EN.WIKIPEDIA.ORG/WIKI/PYTHON%20%28PROGRAMMING%20LANGUAGE%29](http://en.wikipedia.org/wiki/Python%20%28programming%20language%29)

17 [HTTP://EN.WIKIPEDIA.ORG/WIKI/JYTHON](http://en.wikipedia.org/wiki/Jython)

18 [HTTP://EN.WIKIPEDIA.ORG/WIKI/GROOVY%20%28PROGRAMMING%20LANGUAGE%29](http://en.wikipedia.org/wiki/Groovy%20%28programming%20language%29)

19 [HTTP://EN.WIKIPEDIA.ORG/WIKI/PIZZA%20%28PROGRAMMING%20LANGUAGE%29](http://en.wikipedia.org/wiki/Pizza%20%28programming%20language%29)

20 [HTTP://EN.WIKIPEDIA.ORG/WIKI/GENERIC%20JAVA%20%28PROGRAMMING%20LANGUAGE%29](http://en.wikipedia.org/wiki/Generic%20Java%20%28programming%20language%29)

- NETREXX²¹

²¹ [HTTP://EN.WIKIPEDIA.ORG/WIKI/NETREXX](http://en.wikipedia.org/wiki/NETREXX)

5 Getting Started

1. REDIRECT JAVA PROGRAMMING/GETTING STARTED¹

5.1 Getting Started

With Java, it is as easy as pie to get started. However, if you encounter difficulty in getting started with Java, you may find useful help and tips in this chapter. Our authors have tried their best to include as much useful content as possible in this chapter for your reference. But, nevertheless, if problems persist, please consult the authors using the [DISCUSSION](#)² page above.

This section is a quick start to using Java: installing Java software, compiling and running programs, and some small sample programs to illustrate the basics of getting started with Java.

CATEGORY:JAVA PROGRAMMING³

¹ [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FGETTING%20STARTED](http://en.wikibooks.org/wiki/Java%20Programming%2FGetting%20Started)

² [HTTP://EN.WIKIBOOKS.ORG/WIKI/TALK%3AJAVA%20PROGRAMMING%2FGETTING%20STARTED](http://en.wikibooks.org/wiki/Talk%3AJava%20Programming%2FGetting%20Started)

³ [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/Category%3AJava%20Programming)

6 Compilation

We have already discussed COMPILATION BASICS¹. Here's a recap of the concepts we'd seen earlier and some additional details.

6.1 Compiling to bytecode

In Java, programs are not compiled into executable files; they are compiled into BYTECODE² (as discussed EARLIER³), which the JVM then executes at runtime. Java source code is compiled into bytecode when we use the `javac` compiler. The bytecode gets saved on the disk with the file extension `.class`. When the program is to be run, the bytecode is converted, using the JUST-IN-TIME(JIT) COMPILER⁴. The result is machine code which is then fed to the memory and is executed.

So Java has four step compilation:

- compiler will check the syntactical error
- create byte-code
- create a blank *.class file
- merge that byte code to that blank *.class file

The Java classes/Byte Codes are compiled to machine code and loaded into memory by the JVM when needed the first time. This is different than other languages like C/C++ where the whole program had to be compiled to machine code and linked to create an executable file, before the program could start.

JIT compilers compile byte-code once and the compiled machine code are re-used again and again, to speed up execution. Early Java compilers compiled the byte-code to machine code each time it was used, but more modern compilers cache this machine code for reuse on the machine. Even then, java's JIT compiling was still faster than an "interpreter-language", where code is compiled from **high level language**, instead of from byte-code each time it was used.

1 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FJAVA%20PROGRAMMING%20ENVIRONMENT](http://en.wikibooks.org/wiki/Java%20Programming%2FJava%20Programming%20Environment)

2 [HTTP://EN.WIKIPEDIA.ORG/WIKI/BYTECODE](http://en.wikipedia.org/wiki/Bytecode)

3 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FJAVA%20PROGRAMMING%20ENVIRONMENT%23THE%20BYTECODE](http://en.wikibooks.org/wiki/Java%20Programming%2FJava%20Programming%20Environment%23The%20Bytecode)

4 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FJAVA%20PROGRAMMING%20ENVIRONMENT%23THE%20JIT%20COMPILER](http://en.wikibooks.org/wiki/Java%20Programming%2FJava%20Programming%20Environment%23The%20JIT%20Compiler)

6.2 Automatic Compilation of Dependent Classes

In Java, if you have used any reference to any other java object, then the class for that object will be automatically compiled, if that was not compiled already. These automatic compilations are nested, and this continues until all classes are compiled that are needed to run the program. It is usually enough to compile only the high level class, since all the dependent classes will be automatically compiled.

```
javac ... MainClass.java
```

However, you can't rely on this feature if your program is using reflection to create objects, or you are compiling for servlets or a "jar" package. In these cases you should list these classes for explicit compilation.

```
javac ... MainClass.java, ServletOne.java, ...
```

The best way is to use a build tool to build your application. The build tool would check all the needed dependencies and compile only the needed class for the build. The ANT⁵ tool is the best and the most popular build tool currently available. Using ANT⁶ you would build your application from the command line by typing:

```
ant build.xml
```

The xml file contains all the information needed to build the application.

Note: In rare cases, your code may appear to compile correctly but the program behaves as if you were using an old copy of the source code (or otherwise reports errors during runtime.) When this occurs, you may need to clean your compilation folder by either deleting the class files or using the Clean command from the IDE.

The next most popular way to build applications are using an IDE. IDE stands for *Integrated Development Environment*, examples of which are listed below.

6.3 Packages, Subdirectories, and Resources

Each Java top level class belongs to a package (covered in the chapter about PACKAGES⁷). This may be declared in a `package` statement at the beginning of the file; if that is missing, the class belongs to the unnamed package.

For compilation, the file must be in the right directory structure. A file containing a class in the unnamed package must be in the current/root directory; if the class belongs to a package, it must be in a directory with the same name as the package.

5 Chapter 6.13 on page 40

6 Chapter 6.13 on page 40

7 Chapter 12 on page 75

The convention is that package names and directory names corresponding to the package consist of only lower case letters.

Example:

Top level package. A class with this package declaration `package example;` has to be in a directory named `example`

Example:

Subpackages. A class with this package declaration `package org.wikibooks.en;` has to be in the `org/wikibooks/en` directory.

Java programs often contain non-code files such as images and properties files. These are referred to generally as 'resources' and stored in directories local to the classes in which they're used. For example, if the class `com.example.ExampleApp` uses the `icon.png` file, this file could be stored as `/com/example/resources/icon.png`. These resources present a problem when a program is compiled, because `javac` does not copy them to wherever the `.class` files are being compiled to (see above); it is up to the programmer to move the resource files and directories. *See also the section on how to automate this using `ant`⁸, below.*

6.4 Filename Case

The Java source file name must be the same as the public class name, the file contains. There can be only one public class defined per file. The Java class name is case sensitive, as is the source file name.

The naming convention for the class name is for it to start with a capital letter.

6.5 Compiler Options

6.5.1 Debugging and Symbolic Information

6.6 Additional Tools

6.6.1 IDEs

This section contains a little about the different IDEs available and their strengths and weaknesses.

⁸ Chapter 6.13 on page 40

6.7 JBuilder

JBuilder is a IDE with proprietary source code, sold by BORLAND⁹. One of the advantages in integration with together, a modeling tool.

6.8 JCreator

There's info at: <http://www.apcomputerscience.com/ide/jcreator/index.htm>

6.9 Eclipse

Eclipse is a free IDE, plus a developer tool framework that can be extended for a particular development need. IBM was behind this free software development and it replaced IBM Visual Age tool. The idea was to create a standard look and feel that can be extended. The extendibility has distinguished Eclipse from other IDE tools. Eclipse also meant to compete with Microsoft Visual Studio tools. Microsoft tools give a standard way of developing code in the Microsoft world. Eclipse gives a similar standard way of developing code in the Java world, with big success so far. With the online error checking only, coding can be speed up by at least 50%(coding does not include programming).

The goal for Eclipse are twofold:

- Give a standard IDE for developing code
- Give a starting point, and the same look and feel for all other more sophisticated tools built on Eclipse

IBM's WSAD, and later IBM Rational Software Development Platform are built on Eclipse.

Standard Eclipse features:

- Standard window management (perspectives, views, browsers, explorers, ...)
- As you type error checking (immediate error indications, ...)
- As you type help window (type ., or <ctrl> space, ...)
- Automatic build (changed source code automatically compiled, ...)
- Built in debugger (full featured GUI debugger)
- Source code generation (getters and setters, ...)
- Searches (for implementation, for references, ...)
- Code refactoring (global reference update, ...)
- Plug-in-based architecture (be able to build tools that integrate seamlessly with the environment and other tools)
- ...

For more information see;

- ECLIPSE ¹⁰.

⁹ [HTTP://EN.WIKIBOOKS.ORG/WIKI/BORLAND](http://en.wikibooks.org/wiki/Borland)

¹⁰ [HTTP://WWW.ECLIPSE.ORG/](http://www.eclipse.org/)

- [PLUGINCENTRAL](#)¹¹

6.10 NetBeans

The NetBeans IDE is a free, open-source Integrated Development Environment for software developers. The IDE runs on many platforms including Windows, Linux, Solaris, and the MacOS. It is easy to install and use straight out of the box. The NetBeans IDE provides developers with all the tools they need to create professional cross-platform desktop, enterprise, web and mobile applications.

More info can be found at <http://www.netbeans.org/products/ide/>

6.11 BlueJ

BlueJ is an IDE that includes templates and will compile and run the applications for you. BlueJ is often used by classes because it is not necessary to set classpaths. BlueJ has it's own sets of Library's and you can add your own under preferences. That sets the classpath for all compilations that come out of it to include those you have added and the BlueJ libraries.

BlueJ offers an interesting GUI for the creation of packages and programs. Classes are represented as boxes with arrows running between them to represent inheritance/implementation or if one is constructed in another. BlueJ adds all those classes (the project) into the classpath at compile time.

[BLUEJ HOMESITE](#)¹²

6.12 Kawa

Kawa was developed by Tek-Tools. It is basically a Java editor which does not include wizards, and GUI tools. It is best suited to experienced Java programmers in small and midsize development teams.

The latest version is 4.0, you can [DOWNLOAD IT](#).¹³ For more info. see [KAWA FROM TEK-TOOLS](#)¹⁴. See a [JAVAWORLD ARTICLE](#)¹⁵

It looks that there is no new development for Kawa.

¹¹ [HTTP://WWW.ECLIPSEPLUGINCENTRAL.COM/](http://www.eclipseplugincentral.com/)

¹² [HTTP://WWW.BLUEJ.ORG](http://www.bluej.org)

¹³ [HTTP://WWW.OLABS.COM/KAWA/](http://www.olabs.com/kawa/)

¹⁴ [HTTP://WWW.TEK-TOOLS.COM/KAWA/](http://www.tek-tools.com/kawa/)

¹⁵ [HTTP://WWW.JAVAWORLD.COM/JAVAWORLD/JW-06-2000/JW-0602-IW-KAWA.HTML](http://www.javaworld.com/javaworld/JW-06-2000/JW-0602-IW-KAWA.HTML)

6.13 Ant

For comprehensive information about all aspects of Ant, please see the ANT WIKIBOOK¹⁶.

Ant is a build management tool designed to replace `make` as the tool for automated builds of large Java applications. Like Java, and unlike `make`, Ant is designed to be platform independent.

Building a Java application requires certain tasks to be performed. Those tasks may include not only compiling the code, but also copying files, packaging the program into a `jar` file, running tests and so on. Some of these tasks may depend upon others having been done previously (not creating a `jar` unless the program has been compiled, for instance). It might also be a good idea to not execute all tasks every time the program is compiled -- e.g. to only compile changed source files. **Ant makes all of these things easy.**

The tasks and their dependencies are defined in a `build.xml` file, generally kept in the root directory of the java project. Ant parses this file and executes the tasks therein. Below we give an example `build.xml` file.

Ant tool is written in Java and is open source, so it can be extended if there is a task you'd like to be done during the build that is not in the pre-defined tasks list. It is very easy to hook your ant task code to the other tasks: your code only needs to be in the classpath, and the Ant tool will load it at runtime. For more information about writing your own Ant tasks, please see the project website at <http://ant.apache.org/>.

Example build.xml file.

xml Source

```
<?xml version="1.0"?>
<project name="ExampleApp" basedir="." default="main">

  <property name="source.dir" value="source" />
  <property name="libraries.dir" value="libraries" />
  <property name="build.dir" value="build" />
  <property name="classes.dir" value="${build.dir}/classes" />
  <property name="dist.dir" value="${build.dir}/dist" />
  <property name="main-class" value="com.example.ExampleApp"/>

  <path id="classpath">
    <fileset dir="${libraries.dir}" includes="**/*.jar"/>
  </path>

  <target name="clean">
    <delete dir="${build.dir}"/>
  </target>
```

¹⁶ [HTTP://EN.WIKIBOOKS.ORG/WIKI/PROGRAMMING%3AAPACHE%20ANT](http://en.wikibooks.org/wiki/Programming%3AApache%20Ant)

```

<target name="compile">
  <mkdir dir="${classes.dir}"/>
  <javac srcdir="${source.dir}" destdir="${classes.dir}"
classpathref="classpath" />
  <!-- Copy all resources to the build directory (all non-java files); see
the
      section 'Packages, Subdirectories, and Resources' above for more
information. -->
  <copy todir="${classes.dir}">
    <fileset dir="${src.dir}" excludes="**/*.java" />
  </copy>
</target>

```

```

<target name="build" depends="compile">
  <mkdir dir="${dist.dir}"/>
  <copy todir="${dist.dir}/lib" flatten="true">
    <path refid="classpath" />
  </copy>
  <path id="dist.classpath">
    <fileset dir="${dist.dir}/lib" includes="*.jar" />
  </path>
  <manifestclasspath property="dist.manifest.classpath"
jarfile="${dist.dir}/${ant.project.name}.jar">
    <classpath refid="dist.classpath" />
  </manifestclasspath>
  <jar destfile="${dist.dir}/${ant.project.name}.jar" >
    <zipfileset dir="${classes.dir}" />
    <manifest>
      <attribute name="Class-Path" value="${dist.manifest.classpath}"/>
      <attribute name="Main-Class" value="${main-class}" />
    </manifest>
  </jar>
</target>

```

```

<target name="run-build" depends="build">
  <java jar="${dist.dir}/${ant.project.name}.jar" fork="true">
    <classpath>
      <path refid="classpath"/>
      <path location="${dist.dir}/${ant.project.name}.jar"/>
    </classpath>
  </java>
</target>

```

```

<target name="run" depends="compile">
  <java classname="${main-class}" >
    <classpath>
      <path refid="classpath"/>
      <pathelement location="${classes.dir}" />
    </classpath>
  </java>
</target>

```

```
<target name="clean-build" depends="clean,build"/>

<target name="main" depends="clean,run"/>

</project>
```

6.14 The JIT compiler

The standard JIT compiler runs *on demand*. When a method is called repeatedly, the JIT compiler analyzes the bytecode and produces highly efficient machine code, which runs very fast. The JIT compiler is smart enough to recognize when the code has already been compiled, so as the application runs, compilation happens only as needed. As Java applications run, they tend to become faster and faster, because the JIT can perform runtime profiling and optimization to the code to meet the execution environment. Methods or code blocks which do not run often receive less optimization; those which run often (so called *hotspots*) receive more profiling and optimization.

7 Execution

There are various ways in which Java code can be executed. A complex Java application usually uses third party APIs or services. In this section we list the most popular ways a piece of Java code may be packed together and/or executed.

7.1 JSE code execution

Java language first edition came out in the client-server era. Thick clients were developed with rich GUI interfaces. Java first edition, JSE (Java Standard Edition) had/has the following in its belt:

- GUI capabilities (AWT, Swing)
- Network computing capabilities (RMI¹)
- Multi-tasking capabilities (Threads)

With JSE the following Java code executions are possible:

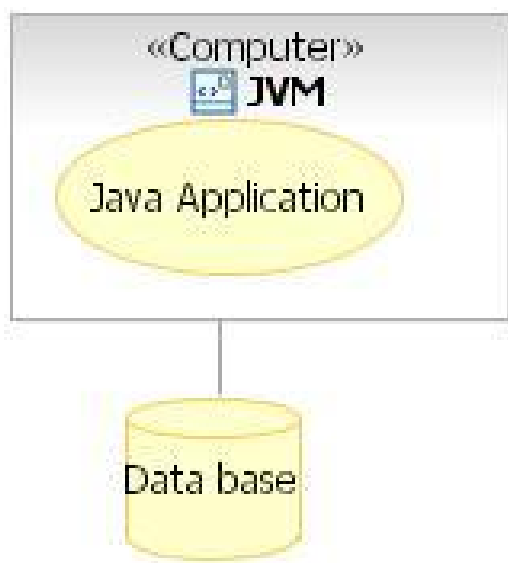


Figure 9: Figure 1: Stand alone execution

Stand alone Java application

¹ [HTTP://EN.WIKIPEDIA.ORG/WIKI/JAVA%20REMOTE%20METHOD%20INVOCATION](http://en.wikipedia.org/wiki/Java%20Remote%20Method%20Invocation)

(Figure 1) Stand alone application refers to a Java program where both the user interface and business modules are running on the same computer. The application may or may not use a database to persist data. The user interface could be either AWT or Swing.

The application would start with a `main()` method of a Class. The application stops when the `main()` method exits, or if an exception is thrown from the application to the JVM. Classes are loaded to memory and compiled as needed, either from the file system or from a *.jar file, by the JVM.

Invocation of Java programs distributed in this manner requires usage of the command line. Once the user has all the class files, he needs to launch the application by the following command line (where Main is the name of the class containing the `main()` method.)

```
java Main
```

Java 'jar' class libraries

Utility classes, framework classes, and/or third party classes are usually packaged and distributed in Java ' *.jar' files. These 'jar' files need to be put in the CLASSPATH of the java program from which these classes are going to be used.

If a jar file is executable, it can be run from the command line:

```
java -jar Application.jar
```

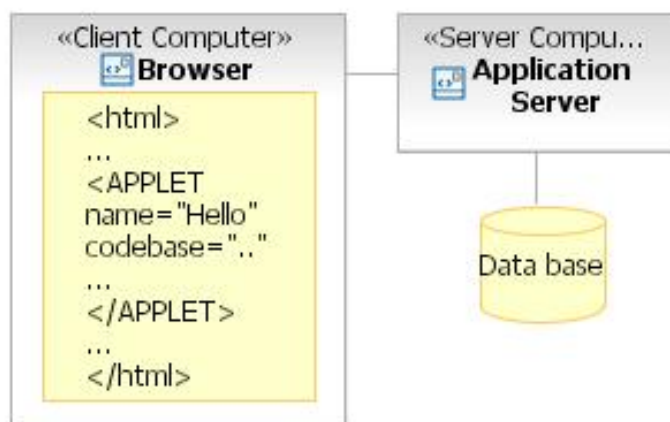


Figure 10: Figure 2: Applet Execution

Java Applet code

(Figure 2) Java Applets are Java code referenced from HTML² pages, by the `<APPLET>` tag. The Java code is downloaded from a server and runs in the client browser JVM. Java has built-in support to render applets in the browser window.

² [HTTP://EN.WIKIPEDIA.ORG/WIKI/HTML](http://en.wikipedia.org/wiki/HTML)

Sophisticated GUI clients were found hard to develop, mostly because of download time, incompatibilities between browser JVM implementations, and communication requirements back to the server. Applets are rarely used today, and are most commonly used as small, separate graphic-like animation applets. The popularity of Java declined when Microsoft withdrew its Java support from INTERNET EXPLORER³ default configuration, however, the plugin is still available as a free download from JAVA.COM⁴.

More information can be found about applets at the APPLET CHAPTER⁵, in this book. Also, Wikipedia has an article about JAVA APPLETS⁶.

Client Server applications

The client server applications consist of a front-end, and a back-end part, both running on a separate computer. The idea is that the business logic would be on the back-end part of the program, which would be reused by all the clients. Here the challenge is to achieve a separation between front-end user interface code, and the back-end business logic code.

The communication between the front-end and the back-end can be achieved by two ways.

- One way is to define a data communication PROTOCOL⁷ between the two tiers. The back-end part would listen for an incoming request. Based on the PROTOCOL⁸ it interprets the request and sends back the result in data form.
- The other way is to use JAVA REMOTE INVOCATION⁹ (RMI). With the use of RMI, a remote object can be created and used by the client. In this case Java objects are transmitted across the network.

More information can be found about client-server programming, with sample code, at the CLIENT SERVER CHAPTER¹⁰ in this book.

Web Applications

For applications needed by lots of client installations, the client-server model did not work. Maintaining and upgrading the hundreds or thousands of clients caused a problem. It was not practical. The solution to this problem was to create a unified, standard client, for all applications, and that is the BROWSER¹¹.

3 [HTTP://EN.WIKIPEDIA.ORG/WIKI/INTERNET%20EXPLORER](http://en.wikipedia.org/wiki/Internet%20Explorer)

4 [HTTP://JAVA.COM/](http://java.com/)

5 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FAPPLETS](http://en.wikibooks.org/wiki/Java%20Programming%2FApplets)

6 [HTTP://EN.WIKIPEDIA.ORG/WIKI/JAVA%20APPLET](http://en.wikipedia.org/wiki/Java%20Applet)

7 [HTTP://EN.WIKIPEDIA.ORG/WIKI/PROTOCOL%20%28COMPUTING%29](http://en.wikipedia.org/wiki/Protocol%20%28Computing%29)

8 [HTTP://EN.WIKIPEDIA.ORG/WIKI/PROTOCOL%20%28COMPUTING%29](http://en.wikipedia.org/wiki/Protocol%20%28Computing%29)

9 [HTTP://EN.WIKIPEDIA.ORG/WIKI/JAVA%20REMOTE%20METHOD%20INVOCATION](http://en.wikipedia.org/wiki/Java%20Remote%20Method%20Invocation)

10 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FCLIENT%20SERVER](http://en.wikibooks.org/wiki/Java%20Programming%2FClient%20Server)

11 [HTTP://EN.WIKIPEDIA.ORG/WIKI/WEB%20BROWSER](http://en.wikipedia.org/wiki/Web%20Browser)

Having a standard client, it makes sense to create a unified, standard back-end service as well, and that is the APPLICATION SERVER¹².

Web Application is an application that is running in the APPLICATION SERVER¹³, and it can be accessed and used by the BROWSER¹⁴ client.

There are three main area of interest in Web Applications, those are:

- The WEB BROWSER¹⁵. This is the container of rendering HTML text, and running client scripts
- The HTTP¹⁶ PROTOCOL¹⁷. Text data are sent back and forth between Browser and the Server
- The WEB SERVER¹⁸ to serve static content, APPLICATION SERVER¹⁹ to serve dynamic content and host EJB²⁰s.

Wikipedia also has an article about WEB APPLICATION²¹.

7.2 J2EE code execution

As the focus was shifting from reaching GUI clients to thin client applications, with Java version 2, Sun introduced J2EE (Java 2 Extended Edition). J2EE added :

- COMPONENTS BASE ARCHITECTURE²², (Servlet, JSP, EJB Containers)

With J2EE the following Java component executions are possible:

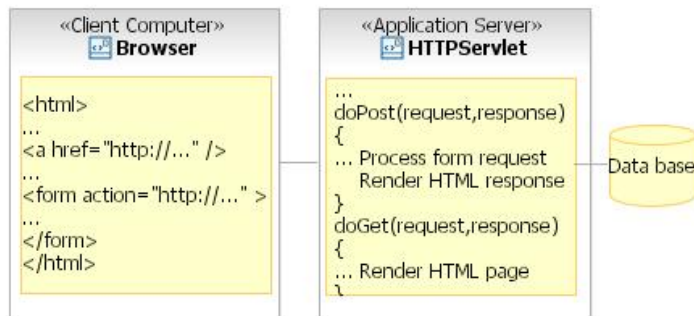


Figure 11: Figure 3: Servlet Execution

Java Servlet code

- 12 [HTTP://EN.WIKIPEDIA.ORG/WIKI/APPLICATION%20SERVER](http://en.wikipedia.org/wiki/Application%20server)
- 13 [HTTP://EN.WIKIPEDIA.ORG/WIKI/APPLICATION%20SERVER](http://en.wikipedia.org/wiki/Application%20server)
- 14 [HTTP://EN.WIKIPEDIA.ORG/WIKI/WEB%20BROWSER](http://en.wikipedia.org/wiki/Web%20browser)
- 15 [HTTP://EN.WIKIPEDIA.ORG/WIKI/WEB%20BROWSER](http://en.wikipedia.org/wiki/Web%20browser)
- 16 [HTTP://EN.WIKIPEDIA.ORG/WIKI/HYPertext%20TRANSFER%20PROTOCOL](http://en.wikipedia.org/wiki/Hypertext%20transfer%20protocol)
- 17 [HTTP://EN.WIKIPEDIA.ORG/WIKI/PROTOCOL%20%28COMPUTING%29](http://en.wikipedia.org/wiki/Protocol%20%28computing%29)
- 18 [HTTP://EN.WIKIPEDIA.ORG/WIKI/WEB%20SERVER](http://en.wikipedia.org/wiki/Web%20server)
- 19 [HTTP://EN.WIKIPEDIA.ORG/WIKI/APPLICATION%20SERVER](http://en.wikipedia.org/wiki/Application%20server)
- 20 [HTTP://EN.WIKIPEDIA.ORG/WIKI/ENTERPRISE%20JAVA BEAN](http://en.wikipedia.org/wiki/Enterprise%20java%20bean)
- 21 [HTTP://EN.WIKIPEDIA.ORG/WIKI/WEB%20APPLICATION](http://en.wikipedia.org/wiki/Web%20application)
- 22 [HTTP://EN.WIKIPEDIA.ORG/WIKI/SOFTWARE%20COMPONENTRY](http://en.wikipedia.org/wiki/Software%20componentry)

(Figure 3) Java got its popularity with server side programming, more specifically with J2EE²³ servlets. Servlets are running in a simple J2EE framework to handle client HTTP²⁴ requests. They are meant to replace CGI PROGRAMMING²⁵ for web pages rendering dynamic content.

The servlet is running in a so called SERVLET-CONTAINER/WEB CONTAINER²⁶. The servlet's responsibility is to:

- Handle the request by doing the business logic computation,
- Connecting to a database if needed,
- Create HTML to present to the user through the browser

The HTML output represents both the presentation logic and the results of the business computations. This represents a huge problem, and there is no real application relying only on servlets to handle the presentation part of the responsibility. There are two main solutions to this:

- Use a template tool (Store the presentation part in an HTML file, marking the areas that need to be replaced after business logic computations).
- Use JSP (See next section)

Wikipedia also has an article about SERVLETS²⁷.

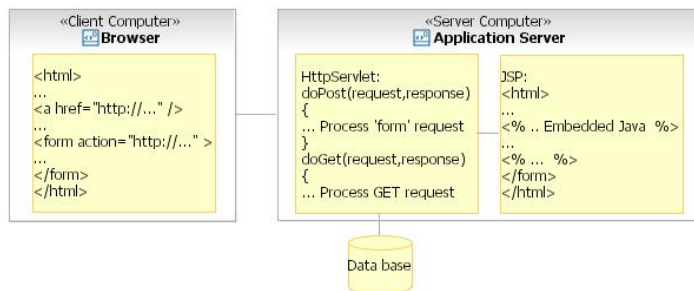


Figure 12: Figure 4: Jsp Execution

Java Server Pages (JSP) code

(Figure 4) JSP is an HTML file with embedded Java code inside. The first time the JSP is accessed, the JSP is converted to a Java Servlet. This servlet outputs HTML which has inside the result of the business logic computation. There are special JSP tags that helps to add data dynamically to the HTML. Also JSP technology allows to create custom tags.

Using the JSP technology correctly, business logic computations should not be in the embedded Java part of the JSP. JSP should be used to render the presentation of the static and dynamic data. Depending on the complexity of the data, 100% separation is not easy to achieve. Using custom

23 [HTTP://EN.WIKIPEDIA.ORG/WIKI/JAVA%20EE](http://en.wikipedia.org/wiki/Java%20EE)

24 [HTTP://EN.WIKIPEDIA.ORG/WIKI/HYPertext%20TRANSFER%20PROTOCOL](http://en.wikipedia.org/wiki/Hypertext%20Transfer%20Protocol)

25 [HTTP://EN.WIKIPEDIA.ORG/WIKI/COMMON%20GATEWAY%20INTERFACE](http://en.wikipedia.org/wiki/Common%20Gateway%20Interface)

26 [HTTP://EN.WIKIPEDIA.ORG/WIKI/WEB%20CONTAINER](http://en.wikipedia.org/wiki/Web%20Container)

27 [HTTP://EN.WIKIPEDIA.ORG/WIKI/JAVA%20SERVLET](http://en.wikipedia.org/wiki/Java%20Servlet)

tags, however may help to get closer to 100%. This is advocated also in MVC²⁸ architecture (see below).

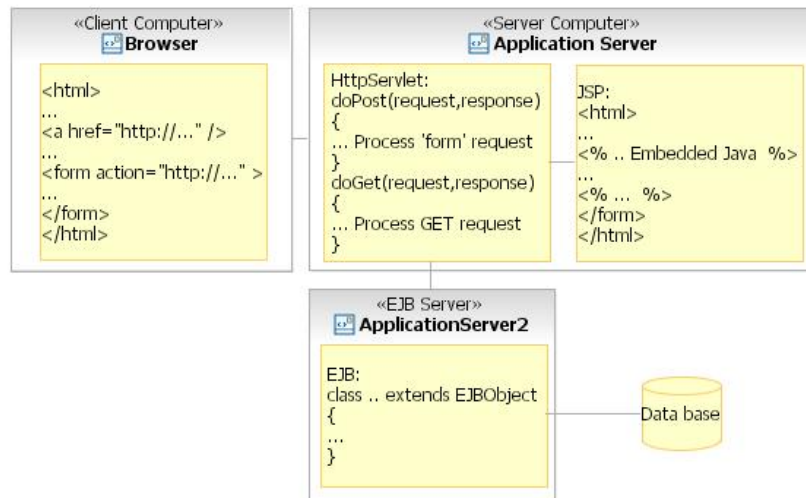


Figure 13: Figure 5: EJB Execution

EJB code

(Figure 5) In the 1990s, with the client server computing, a trend started, that is to move away from Mainfram computing. That resulted in many small separate applications in a Company/Enterprise. Many times the same data was used in different applications. A new philosophy, "Enterprise Computing", was created to address these issues. The idea was to create components that can be reused throughout the Enterprise. The Enterprise Java Beans (EJBs) were supposed to address this.

An **EJB** is an application component that runs in an EJB container. The client accesses the EJB modules through the container, never directly. The container manages the life cycle of the EJB modules, and handles all the issues that arise from network/enterprise computing. Some of those are SECURITY/ACCESS CONTROL²⁹, OBJECT POOLING³⁰, TRANSACTION MANAGEMENT³¹,

EJBs have the same problems as any reusable code: they need to be generic enough to be able to be reused and the changes or maintenance of EJBs can affect existing clients. Many times EJBs are used unnecessarily when they are not really needed. An EJB should be designed as a separate application in the enterprise, fulfilling one function.

²⁸ [HTTP://EN.WIKIPEDIA.ORG/WIKI/MODEL-VIEW-CONTROLLER](http://en.wikipedia.org/wiki/Model-View-Controller)

²⁹ [HTTP://EN.WIKIPEDIA.ORG/WIKI/ACCESS%20CONTROL](http://en.wikipedia.org/wiki/Access%20control)

³⁰ [HTTP://EN.WIKIPEDIA.ORG/WIKI/OBJECT%20POOL](http://en.wikipedia.org/wiki/Object%20pool)

³¹ [HTTP://EN.WIKIPEDIA.ORG/WIKI/TRANSACTION%20PROCESSING](http://en.wikipedia.org/wiki/Transaction%20processing)

MVC Paradigm Diagram

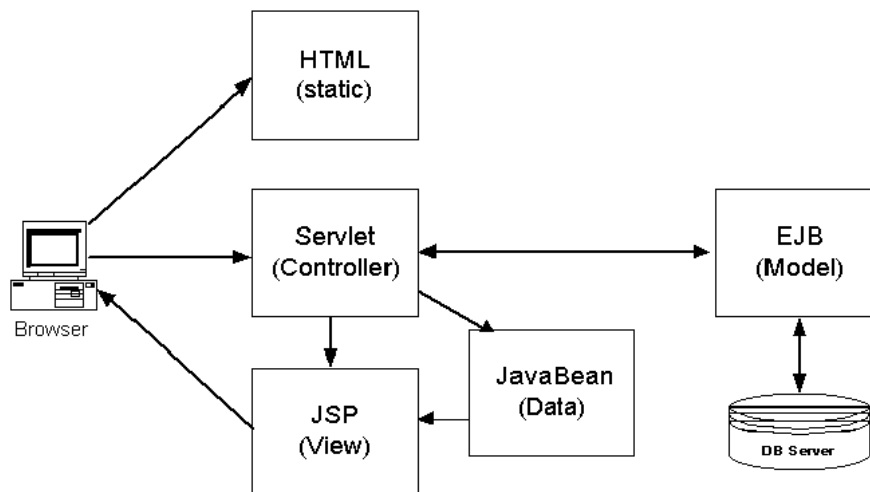


Figure 14: Figure 6: MVC Execution

Combine J2EE components to create an MVC architecture

This leads us to the three layers/tiers as shown in (Figure 6).

In modern web applications, with lots of static data and nice graphics, how the data is presented to the user became very important and usually needs the help of a graphic artist.

To help programmers and graphic artists to work together, the separation between data, code, and how it is presented became crucial.

- The **view** (User Interface Logic) contains the logic that is necessary to construct the presentation. This could be handled by JSP technology.
- The servlet acts as the **controller** and contains the logic that is necessary to process user events and to select an appropriate response.
- The business logic (**model**) actually accomplishes the goal of the interaction. This might be a query or an update to a database. This could be handled by EJB technology.

For more information about MVC, please see MVC³².

7.3 Jini

After J2EE Sun had a vision about the next step of network computing. That is JINI³³. The main idea is that in a network environment, there would be many independent services and con-

³² [HTTP://EN.WIKIPEDIA.ORG/WIKI/MODEL-VIEW-CONTROLLER](http://en.wikipedia.org/wiki/Model-View-Controller)

³³ [HTTP://EN.WIKIPEDIA.ORG/WIKI/JINI](http://en.wikipedia.org/wiki/Jini)

sumers. Jini would allow these services/consumers to interact dynamically with each other in a robust way. The basic features of Jini are:

- **No user intervention** is needed when services are brought on or offline. (In contrast to EJBs where the client program has to know the server and port number where the EJB is deployed, in Jini the client is *supposed to find*, to discover, the service in the network.)
- **Self healing** by adapting when services (consumers of services) come and go. (Services periodically need to renew a lease to indicate that they are still available.)
- Consumers of JINI services do not need prior knowledge of the service's implementation. The **implementation is downloaded dynamically** and run on the consumer JVM, without configuration and user intervention. (For example, the end user may be presented with a slightly different user interface depending upon which service is being used at the time. The implementation of the user interface code would be provided by the service being used.)

A minimal Jini network environment consists of:

- One or more **services**
- A **lookup-service** keeping a list of registered services
- One or more **consumers**

Jini is not widely used at the current writing (2006). There are two possible reasons for it. One is Jini a bit complicated to understand and to set it up. The other reason is that Microsoft pulled out from Java, which caused the industry to turn to the use of proprietary solutions.

CATEGORY:JAVA PROGRAMMING³⁴

³⁴ [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/Category%3AJAVA%20Programming)

8 Understanding a Java Program

This article presents a small Java program which can be run from the console. It computes the distance between two points on a plane. You need not understand the structure and meaning of the program just yet; we will get to that soon. Also, because the program is intended as a simple introduction, it has some room for improvement, and later in the module we will show some of these improvements. But let's not get too far ahead of ourselves!

8.1 The Distance Class: Intent, Source, and Use

This class is named *Distance*, so using your favorite editor or Java IDE, first create a file named `Distance.java`, then copy the source below and paste it into the file and save the file.

```
public class Distance
{
    private java.awt.Point point0, point1;

    public Distance(int x0, int y0, int x1, int y1)
    {
        point0 = new java.awt.Point(x0, y0);
        point1 = new java.awt.Point(x1, y1);
    }

    public void printDistance()
    {
        System.out.println("Distance between " + point0 + " and " + point1
            + " is " + point0.distance(point1));
    }

    public static void main(String[] args)
    {
        Distance dist = new Distance(
            intValue(args[0]), intValue(args[1]),
            intValue(args[2]), intValue(args[3]));
        dist.printDistance();
    }

    private static int intValue(String data)
    {
        return Integer.parseInt(data);
    }
}
```

At this point, you may wish to review the source to see how much you might be able to understand. While perhaps not being the most literate of programming languages, someone with understanding of other procedural languages such as C, or other OO languages such as C++ or C#, will be able to understand most if not all of the sample program.

Once you save the file, `COMPILE`¹ the program:

```
javac Distance.java
```

(If the `javac` command fails, review the [JAVA INSTALLATION INSTRUCTIONS](#)².)

To run the program, you supply it with the x and y coordinates of two points on a plane. (For this version of `Distance`, only integer points are supported.) The command sequence is

```
java Distance  $x_0$   $y_0$   $x_1$   $y_1$ 
```

to compute the distance between the points (x_0, y_0) and (x_1, y_1)

For example, the command

```
java Distance 0 3 4 0
```

will compute the distance between the points $(0,3)$ and $(4,0)$ and print the following:

```
Distance between java.awt.Point[x=0,y=3] and java.awt.Point[x=4,y=0] is 5.0
```

The command

```
java Distance -4 5 11 19
```

will compute the distance between the points $(-4,5)$ and $(11,19)$:

```
Distance between java.awt.Point[x=-4,y=5] and java.awt.Point[x=11,y=19] is  
20.518284528683193
```

We'll explain this strange looking output, and also show how to improve it, later.

8.2 Detailed Program Structure and Overview

As promised, we will now provide a detailed description of this Java program. We will discuss the syntax and structure of the program and the meaning of that structure.

¹ Chapter 6 on page 35

² [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FINSTALLATION](http://en.wikibooks.org/wiki/Java%20Programming%2FInstallation)

8.2.1 Introduction to Java Syntax

The *syntax* of a Java class is the characters and symbols and their structure used to code the class using Unicode characters. A fuller treatment of the syntax elements of Java may be found at SYNTAX³. We will provide here only enough description of the syntax to grasp the above program.

Java programs consist of a sequence of tokens. There are different kinds of tokens. For example, there are word tokens such as `class` and `public` which represent KEYWORDS⁴ - special words with reserved meaning in Java. Other words (non keywords such as `Distance`, `point0`, `x1`, and `printDistance`) are *identifiers*. Identifiers have many different uses in Java but primarily they are used as names. Java also has tokens to represent numbers, such as `1` and `3`; these are known as LITERALS⁵. STRING LITERALS⁶, such as `"Distance between "`, consist of zero or more characters embedded in double quotes, and OPERATORS⁷ such as `+` and `=` are used to express basic computation such as addition or String concatenation or assignment. There are also left and right braces (`{` and `}`) which enclose BLOCKS⁸. The body of a class is one such block. Some tokens are punctuation, such as periods `.` and commas `,` and semicolons `;`. You use WHITESPACE⁹ such as spaces, tabs, and newlines, to separate tokens. For example, whitespace is required between keywords and identifiers: `publicstatic` is a single identifier with twelve characters, not two Java keywords.

8.2.2 Declarations and Definitions

Sequences of tokens are used to construct the next building blocks of Java classes: declarations and definitions. A class declaration provides the name and visibility of a class. For our example,

```
public class Distance
```

is the class declaration. It consists (in this case) of two keywords, `PUBLIC`¹⁰ and `CLASS`¹¹ followed by the identifier `Distance`.

This means that we are defining a class named `Distance`. Other classes, or in our case, the command line, can refer to the class by this name. The `public` keyword is an ACCESS MODIFIER¹² which declares that this class and its members may be accessed from other classes. The `class` keyword, obviously, identifies this declaration as a class. Java also allows declarations of INTERFACES¹³ and (as of Java 5) ANNOTATIONS¹⁴.

3 Chapter 9 on page 61

4 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FKEYWORDS](http://en.wikibooks.org/wiki/Java%20Programming%2FKeywords)

5 Chapter 9.2 on page 63

6 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FSTRING%20LITERALS](http://en.wikibooks.org/wiki/Java%20Programming%2FString%20Literals)

7 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FOPERATORS](http://en.wikibooks.org/wiki/Java%20Programming%2FOperators)

8 Chapter 9.3 on page 64

9 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FWHITESPACE](http://en.wikibooks.org/wiki/Java%20Programming%2FWhitespace)

10 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FKEYWORDS%2FPUBLIC](http://en.wikibooks.org/wiki/Java%20Programming%2FKeywords%2FPublic)

11 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FKEYWORDS%2FCLASS](http://en.wikibooks.org/wiki/Java%20Programming%2FKeywords%2FClass)

12 Chapter 14 on page 83

13 Chapter 24 on page 135

14 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FANNOTATIONS](http://en.wikibooks.org/wiki/Java%20Programming%2FAnnotations)

The class declaration is then followed by a block (surrounded by curly braces) which provides the class's definition. The definition is the implementation of the class - the declaration and definitions of the class's members. This class contains exactly six members, which we will explain in turn.

1. Two field declarations, named `point0` and `point1`
2. A constructor declaration
3. Three method declarations

Example: Instance Fields

The declaration

```
private java.awt.Point point0, point1;
```

declares two INSTANCE FIELDS¹⁵. Instance fields represent named values that are allocated whenever an instance of the class is constructed. When a Java program creates a `Distance` instance, that instance will contain space for `point0` and `point1`. When another `Distance` object is created, it will contain space for its *own* `point0` and `point1` values. The value of `point0` in the first `Distance` object can vary independently of the value of `point0` in the second `Distance` object.

This declaration consists of:

1. The `PRIVATE`¹⁶ access modifier, which means these instance fields are not visible to other classes.
2. The type of the instance fields. In this case, the type is `java.awt.Point`. This is the class `Point` in the `java.awt` package.
3. The names of the instance fields in a comma separated list.

These two fields could also have been declared with two separate but more verbose declarations,

```
private java.awt.Point point0;  
private java.awt.Point point1;
```

Since the types of these fields is a reference type (i.e. a field that *refers to* or can hold a *reference to* an object value), Java will implicitly initialize the values of `point0` and `point1` to `null` when a `Distance` instance is created. The `null` value means that a reference value does not refer to an object. The special Java literal, `null` is used to represent the null value in a program. While you can explicitly assign null values in a declaration, as in

```
private java.awt.Point point0 = null;  
private java.awt.Point point1 = null;
```

it is not necessary and most programmers omit such default assignments.

¹⁵ [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FINSTANCE%20FIELDS](http://en.wikibooks.org/wiki/Java%20Programming%2FInstance%20Fields)

¹⁶ [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FKEYWORDS%2FPRIVATE](http://en.wikibooks.org/wiki/Java%20Programming%2FKeywords%2FPrivate)

Example: Constructor

A `CONSTRUCTOR`¹⁷ is a special method in a class which is used to construct an instance of the class. The constructor can perform initialization for the object, beyond that which the Java VM does automatically. For example, Java will automatically initialize the fields `point0` and `point1` to null.

Below is the constructor for this class. It consists of five parts:

1. The optional `ACCESS MODIFIER(S)`¹⁸.
In this case, the constructor is declared `public`
2. The constructor name, which must match the class name exactly: `Distance` in this case.
3. The constructor `PARAMETERS`¹⁹.
The parameter list is required. Even if a constructor does not have any parameters, you must specify the empty list `()`. The parameter list declares the type and name of each of the method's parameters.
4. An optional **throws** clause which declares the exceptions that the constructor may throw. This constructor does not declare any exceptions.
5. The constructor body, which is a Java `BLOCK`²⁰ (enclosed in `{}`). This constructor's body contains two statements.

```
public Distance(int x0, int y0, int x1, int y1) { point0 = new
java.awt.Point(x0, y0); point1 = new java.awt.Point(x1, y1); }
```

This constructor accepts four parameters, named `x0`, `y0`, `x1` and `y1`. Each parameter requires a parameter type declaration, which in this example is **int** for all four parameters. Java integer values are signed, 32 bit twos complement integers. The parameters in the parameter list are separated by commas.

The two assignments in this constructor use Java's **new operator** to allocate two `java.awt.Point` objects. The first allocates an object representing the first point, `(x0, y0)`, and assigns it to the `point0` instance variable (replacing the null value that the instance variable was initialized to). The second statement allocates a second `java.awt.Point` instance with `(x1, y1)` and assigns it to the `point1` instance variable.

This is the constructor for the `Distance` class. `Distance` implicitly extends from `java.lang.Object`. Java inserts a call to the super constructor as the first executable statement of the constructor if there is not one explicitly coded. The above constructor body is equivalent to the following body with the explicit super constructor call:

```
{
    super();
    point0 = new java.awt.Point(x0, y0);
    point1 = new java.awt.Point(x1, y1);
}
```

17 Chapter 15.7 on page 91

18 Chapter 14 on page 83

19 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FPARAMETERS](http://en.wikibooks.org/wiki/Java%20Programming%2FParameters)

20 Chapter 9.3 on page 64

While it is true that this class could be implemented in other ways, such as simply storing the coordinates of the two points and computing the distance as $\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$, this class instead uses the existing `java.awt.Point` class. This choice matches the abstract definition of this class: to print the distance between two points on the plane. We take advantage of existing behavior already implemented in the Java platform rather than implementing it again. We will see later how to make the program more flexible without adding much complexity, because we choose to use object abstractions here. However, the key point is that this class uses information hiding. That is, *how* the class stores its state or how it computes the distance is hidden. We can change this implementation without altering how clients use and invoke the class.

Example: Methods

Methods are the third and most important type of class member. This class contains three *methods* in which the behavior of the `Distance` class is defined: `printDistance()`, `main()`, and `intValue()`

The `printDistance()` method

The `printDistance()` method prints the distance between the two points to the standard output (normally the console).

```
public void printDistance()
{
    System.out.println("Distance between " + point0
                      + " and " + point1
                      + " is " + point0.distance(point1));
}
```

This INSTANCE METHOD²¹ executes within the context of an implicit `Distance` object. The instance field references, `point0` and `point1`, refer to instance fields of that implicit object. You can also use the special variable `this` to explicitly reference the current object. Within an instance method, Java binds the name `this` to the object on which the method is executing, and the type of `this` is that of the current class. The body of the `printDistance` method could also be coded as

```
System.out.println("Distance between " + this.point0
                  + " and " + this.point1
                  + " is " + this.point0.distance(this.point1));
```

to make the instance field references more explicit.

This method both computes the distance and prints it in one statement. The distance is computed with `point0.distance(point1)`; `distance()` is an instance method of the `java.awt.Point` class (of which `point0` and `point1` are instances. The method operates on `point0` (binding `this` to the object that `point0` refers to during the execution of the method)

²¹ [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FINSTANCE%20METHODS](http://en.wikibooks.org/wiki/Java%20Programming%2FInstance%20Methods)

and accepting another `Point` as a parameter. (Actually, it is slightly more complicated than that, but we'll explain later.) The result of the `distance()` method is a double precision floating point number.

This method uses the syntax

```
"Distance between " + this.point0
+ " and " + this.point1
+ " is " + this.point0.distance(this.point1)
```

to construct a `String` to pass to the `System.out.println()`. This expression is a series of `STRING CONCATENATION`²² methods which concatenates `Strings` or the `String` representation of primitive types (such as `doubles`) or objects, and returns a long string. For example, the result of this expression for the points (0,3) and (4,0) is the `String`

```
"Distance between java.awt.Point[x=0,y=3] and java.awt.Point[x=4,y=0] is 5.0"
```

which the method then prints to `System.out`.

In order to print, we invoke the `println()`. This is an instance method from `java.io.PrintStream`, which is the type of the static field `out` in the class `java.lang.System`. The Java VM binds `System.out` to the standard output stream when it starts a program.

The main() method

The `main()` method is the main entry point which Java invokes when you start a Java program from the command line. The command

```
java Distance 0 3 4 0
```

instructs Java to locate the `Distance` class, put the four command line arguments into an array of `String` values, then pass those arguments the `public static main(String[])` method of the class. (We will introduce arrays shortly.) Any Java class that you want to invoke from the command line or desktop shortcut must have a `main` method with this signature.

```
public static void main(String[] args)
{
    Distance dist = new Distance(
        intValue(args[0]), intValue(args[1]),
        intValue(args[2]), intValue(args[3]));
    dist.printDistance();
}
```

The `main()` method invokes the final method, `intValue()`, four times. The `intValue()` takes a single string parameter and returns the integer value represented in the string. For example, `intValue("3")` will return the integer 3.

²² [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FSTRING%20CONCATENATION](http://en.wikibooks.org/wiki/Java%20Programming%2FString%20Concatenation)

The `intValue()` method

The `intValue()` method delegates its job to the `Integer.parseInt()` method. The main method could have called `Integer.parseInt()` directly; the `intValue()` method simply makes the `main()` method slightly more readable.

```
private static int intValue(String data)
{
    return Integer.parseInt(data);
}
```

This method is `PRIVATE`²³ since, like the fields `point0` and `point1`, it is part of the internal implementation of the class and is not part of the external programming interface of the `Distance` class.

Static vs. Instance Methods

Both the `main()` and `intValue()` methods are `STATIC METHODS`²⁴. The `static` keyword tells the compiler to create a single memory space associated with the class. Each individual object instantiated has its own private state variables and methods but use the same **static** methods and members common to the single class object created by the compiler when the first class object is instantiated or created. This means that the method executes in a static or non-object context - there is no implicit separate instance available when the static methods run from various objects, and the special variable `this` is not available. As such, static methods cannot access instance methods or instance fields (such as `printDistance()` or `point0`) directly. The `main()` method can only invoke the instance method `printDistance()` method via an instance reference such as `dist`.

8.2.3 Data Types

Most declarations have a data type. Java has several categories of data types: reference types, primitive types, array types, and a special type, `void`.

Reference Types

A *reference type* is a Java data type which is defined by a Java class or interface. Reference types derive this name because such values *refer to* an object or contain a *reference to* an object. The idea is similar to pointers in other languages like C.

Java represents sequences of character data, or `STRING`²⁵, with the reference type `java.lang.String` which is most commonly referred to as `String`. *String literals*, such as `"Distance between "` are constants whose type is `String`.

²³ [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FKEYWORDS%2FPRIVATE](http://en.wikibooks.org/wiki/Java%20Programming%2FKeywords%2FPRIVATE)

²⁴ [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FSTATIC%20METHODS](http://en.wikibooks.org/wiki/Java%20Programming%2FStatic%20Methods)

²⁵ Chapter 18 on page 103

This program uses three separate reference types:

1. `java.lang.String` (or simply `String`)
2. `Distance`
3. `java.awt.Point`

For more information see chapter : `JAVA PROGRAMMING/CLASSES, OBJECTS AND TYPES`²⁶.

Primitive Types

In addition to object or reference types, Java supports `PRIMITIVE TYPES`²⁷. The primitive types are used to represent Boolean, character, and numeric values. This program uses only one primitive type explicitly, `int`, which represents 32 bit signed integer values. The program also implicitly uses `double`, which is the return type of the `distance()` method of `java.awt.Point`. `double` values are 64 bit IEEE floating point values. The `main()` method uses integer values 0, 1, 2, and 3 to access elements of the command line arguments. The `Distance()` constructor's four parameters also have the type `int`. Also, the `intValue()` method has a return type of `int`. This means a call to that method, such as `intValue(args[0])`, is an expression of type `int`. This helps explain why the main method cannot call

```
new Distance(args[0], args[1], args[2], args[3]) // this is an error
```

Since the type of the `args` array element is `String`, and our constructor's parameters must be `int`, such a call would result in an error because Java cannot automatically convert values of type `String` into `int` values.

Java's primitive types are `boolean`, `byte`, `char`, `short`, `int`, `long`, `float` and `double`, each of which are also Java language keywords.

Array Types

Java supports `ARRAYS`²⁸, which are aggregate types which have a fixed element type (which can be any Java type) and an integral size. This program uses only one array, `String[] args`. This indicates that `args` has an array type and that the element type is `String`. The Java VM constructs and initializes the array that is passed to the `main` method. See `ARRAYS`²⁹ for more details on how to create arrays and access their size.

The elements of arrays are accessed with integer indices. The first element of an array is always element 0. This program accesses the first four elements of the `args` array explicitly with the indices 0, 1, 2, and 3. (This program does *not* perform any input validation, such as verifying that the user passed at least four arguments to the program. We will fix that later.)

26 Chapter 11 on page 71

27 Chapter 16 on page 95

28 Chapter 28 on page 143

29 Chapter 28 on page 143

void

`VOID`³⁰ is not a type in Java; it represents the absence of a type. Methods which do not return values are declared as `void` methods.

This class defines two `void` methods:

```
public static void main(String[] args) { ... }  
public void printDistance() { ... }
```

8.3 Comments in Java programs

See [HERE](#)³¹ for more information on that important topic.

³⁰ [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FKEYWORDS%2FVOID](http://en.wikibooks.org/wiki/Java%20Programming%2Fkeywords%2Fvoid)

³¹ [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA_PROGRAMMING%2FUNDERSTANDING_A_JAVA_PROGRAM%2FJAVADOC_AND_OTHER_COMMENTS](http://en.wikibooks.org/wiki/Java_Programming%2FUnderstanding_a_Java_Program%2FJavadoc_and_other_comments)

9 Syntax

Java derives much of its *syntax* from the C¹ programming language: basic assignment statement syntax, expressions, control flow statements and blocks, etc. will be very familiar to C programmers.

Unicode

Java source code are built by Unicode characters.

Tokens

Java programs consist of a sequence of different kinds of tokens. For example, there are word tokens such as **class** and **public** which are KEYWORDS².

KEYWORDS³

Those are special words with reserved meaning in Java. Those words can not be used by the programers to name identifiers.

Identifiers

Other words (non keywords) are identifiers. Identifiers have many different uses in Java but primarily they are used as names, class names, method names, and variable names... .

LITERALS⁴

Java also has tokens to represent numbers, such as 1 and 3; these are known as LITERALS⁵.

String LITERALS⁶, such as "http://en.wikibooks.org/Java_Programming", consist of zero or more characters embedded in double quotes.

OPERATORS⁷

And OPERATORS⁸ such as + and = are used to express basic computation such as addition or String concatenation or assignment.

BLOCKS⁹

1 [HTTP://EN.WIKIBOOKS.ORG/WIKI/PROGRAMMING%3AC](http://en.wikibooks.org/wiki/Programming%3AC)
2 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FKEYWORDS](http://en.wikibooks.org/wiki/Java%20Programming%2FKeywords)
3 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FKEYWORDS](http://en.wikibooks.org/wiki/Java%20Programming%2FKeywords)
4 [Chapter 9.2 on page 63](#)
5 [Chapter 9.2 on page 63](#)
6 [Chapter 9.2 on page 63](#)
7 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FOPERATORS](http://en.wikibooks.org/wiki/Java%20Programming%2FOperators)
8 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FOPERATORS](http://en.wikibooks.org/wiki/Java%20Programming%2FOperators)
9 [Chapter 9.3 on page 64](#)

There are also left and right braces ({ and }) which enclose BLOCKS¹⁰. The body of a class is one such block.

STATEMENTS¹¹

A Block contains one or more Java STATEMENT(S)¹², separated by semicolons. A statement is the smallest building block of Java.

Separators

Some tokens are punctuation, such as periods . and commas , and semicolons ; .

WHITESPACE¹³

You use WHITESPACE¹⁴ such as spaces, tabs, and newlines, to separate tokens. For example, whitespace is required between keywords and identifiers: `publicstatic` is a single identifier with twelve characters, not two Java keywords.

COMMENTS¹⁵

Comments are not part of the executing code. Comments are used to document the code.

9.1 Unicode

Most Java program text consists of ASCII¹⁶ characters, but any Unicode character can be used as part of identifier names, in comments, and in character and string literals. UNICODE ESCAPE SEQUENCES¹⁷ may also be used to express a Unicode character.

For example, π (which is the Greek Lowercase Letter **pi**) is a valid Java identifier.

```
double  $\pi$  = Math.PI;
```

and in a string literal

```
String pi = "string";
```

π may also be represented in Java as the *Unicode escape sequence* `\u03C0`. Thus, the following is a valid, but not very readable, declaration and assignment:

10 Chapter 9.3 on page 64

11 Chapter 10 on page 67

12 Chapter 10 on page 67

13 Chapter 9.4 on page 65

14 Chapter 9.4 on page 65

15 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FSYNTAX%2FCOMMENTS](http://en.wikibooks.org/wiki/Java%20Programming%2FSyntax%2FComments)

16 [HTTP://EN.WIKIPEDIA.ORG/WIKI/ASCII](http://en.wikipedia.org/wiki/ASCII)

17 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FSYNTAX%2FUNICODE%20ESCAPE%20SEQUENCES](http://en.wikibooks.org/wiki/Java%20Programming%2FSyntax%2FUnicode%20Escape%20Sequences)

```
double \u03C0 = Math.PI;
```

The following demonstrate the use of Unicode escape sequences in other Java syntax:

```
Declare Strings pi and quote which contain \u03C0 and \u0027
respectively:
```

```
String pi = string;
String quote = string;
```

Note that a Unicode escape sequence functions just like any other character in the source code. E.g., `\u0022` (double quote, ") needs to be quoted in a string just like ".

```
Declare Strings doubleQuote1 and doubleQuote2 which both contain "
(double quote):
```

```
String doubleQuote1 = string;
String doubleQuote2 = string;
```

```
"\u0022" doesn't work since "" doesn't work.
```

See [UNICODE ESCAPE SEQUENCES](#)¹⁸ for full details.

CATEGORY:JAVA PROGRAMMING¹⁹

9.2 Literals

Java Literals are syntactic representations of boolean, character, numeric, or string data. Literals provide a means of expressing specific values in your program. For example, in the following statement, an integer variable named `count` is declared and assigned an integer value. The literal `0` represents, naturally enough, the value zero.

```
int count = 0;
```

The following method call passes a String literal `string` the boolean literal `true` and the special null value `null` to the method `parse()`:

¹⁸ [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FSYNTAX%2FUNICODE%20ESCAPE%20SEQUENCES](http://en.wikibooks.org/wiki/Java%20Programming%2FSyntax%2FUnicode%20Escape%20Sequences)

¹⁹ [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/Category%3AJava%20Programming)

```
List items = parse(string, true, null);
```

- **BOOLEAN LITERALS**²⁰
- **NUMERIC LITERALS**²¹
 - **CHARACTER LITERALS**²²
 - **INTEGER LITERALS**²³
 - **FLOATING POINT LITERALS**²⁴
- **STRING LITERALS**²⁵
- **null**²⁶

9.3 Blocks

Java has a concept called block that is enclosed between the { and } characters, called curly braces. A block executed as a single statement, and can be used where a single statement is accepted.

After a block is executed all local variables defined inside the block is discarded, go out of scope.

```
{
...
// -- This is a block ---
}
```

Blocks can be nested:

```
{
...
{
// -- This is a nested block ---
}
}
```

CATEGORY:JAVA PROGRAMMING²⁷

20 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FLITERALS%2FBOOLEAN%20LITERALS](http://en.wikibooks.org/wiki/JAVA%20PROGRAMMING%2FLITERALS%2FBOOLEAN%20LITERALS)

21 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FLITERALS%2FNUMERIC%20LITERALS](http://en.wikibooks.org/wiki/JAVA%20PROGRAMMING%2FLITERALS%2FNUMERIC%20LITERALS)

22 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FLITERALS%2FNUMERIC%20LITERALS%2FCHARACTER%20LITERALS](http://en.wikibooks.org/wiki/JAVA%20PROGRAMMING%2FLITERALS%2FNUMERIC%20LITERALS%2FCHARACTER%20LITERALS)

23 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FLITERALS%2FNUMERIC%20LITERALS%2FINTEGER%20LITERALS](http://en.wikibooks.org/wiki/JAVA%20PROGRAMMING%2FLITERALS%2FNUMERIC%20LITERALS%2FINTEGER%20LITERALS)

24 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FLITERALS%2FNUMERIC%20LITERALS%2FFLOATING%20POINT%20LITERALS](http://en.wikibooks.org/wiki/JAVA%20PROGRAMMING%2FLITERALS%2FNUMERIC%20LITERALS%2FFLOATING%20POINT%20LITERALS)

25 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FLITERALS%2FSTRING%20LITERALS](http://en.wikibooks.org/wiki/JAVA%20PROGRAMMING%2FLITERALS%2FSTRING%20LITERALS)

26 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FLITERALS%2FNUL](http://en.wikibooks.org/wiki/JAVA%20PROGRAMMING%2FLITERALS%2FNUL)

27 [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/CATEGORY%3AJAVA%20PROGRAMMING)

9.4 Whitespaces

Whitespace in Java is used to separate the tokens in a Java source file. Whitespace is required in some places, such as between ACCESS MODIFIERS²⁸, TYPE NAMES²⁹ and Identifiers, and is used to improve readability elsewhere.

Wherever whitespace is required in Java, one or more whitespace characters may be used. Wherever whitespace is optional in Java, zero or more whitespace characters may be used.

Java whitespace consists of the

- space character ' ' (0x20),
- the tab character (hex 0x09),
- the form feed character (hex 0x0c),
- the line separators characters newline (hex 0x0a) or carriage return (hex 0x0d) characters.

Line separators are special whitespace characters in that they also terminate line comments, whereas normal whitespace does not.

Other Unicode space characters, including vertical tab, are not allowed as whitespace in Java.

9.5 Required Whitespace

Below is the declaration of an **abstract** method taken from a Java class

```
public abstract Distance distanceTo(Destination dest);
```

Whitespace is required between **public** and **abstract**, between **abstract** and `Distance`, between `Distance` and `distanceTo`, and between `Destination` and `dest`.

However, the following is not legal:

```
publicabstractDistance distanceTo(Destination dest);
```

because whitespace is required between keywords and identifiers. The following is lexically valid

```
publicabstractDistance distanceTo(Destination dest);
```

but means something completely different: it declares a method which has the return type `publicabstractDistance`. It is unlikely that this type exists, so the above would result in a semantic error.

²⁸ Chapter 14 on page 83

²⁹ Chapter 17 on page 97

9.6 Indentation

Java ignores all whitespace in front of a statement. As this, these two code snippets are identical for the compiler:

```
public static void main(String[] args) {
    printMessage();
}

void printMessage() {
    System.out.println("Hello World!");
}
```

```
public static void main(String[] args) {
printMessage();
}

void printMessage() {
System.out.println("Hello World!");
}
```

However, the first one's style (with whitespace) is preferred, as the readability is higher. (The method body is easier to distinguish from the head, even at a higher reading speed.)

CATEGORY:JAVA PROGRAMMING³⁰

CATEGORY:JAVA PROGRAMMING³¹

³⁰ [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/Category%3AJAVA%20Programming)

³¹ [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/Category%3AJAVA%20Programming)

10 Statements

Now, that we have the Java platform on our systems and have run the first program successfully, we are geared towards understanding how programs are actually made. As we have already discussed. A program is a set of instructions – very simple tasks provided to a computer. These instructions are called **statements** in Java. Statements can be anything from a single line of code to a complex mathematical equation. This section helps you understand what statements are and how they work.

10.1 What exactly are statements?

Statements are a single instruction in a program – a single unit of code. Consider the following line:

Listing 1.1: A simple assignment statement.

```
int age = 24;
```

This line is a simple instruction that tells the system to initialize a variable and set its value as 24. Within that simple definition, we talked about initialization of a variable and setting its value. This all might sound a bit too technical, but it will make sense as you read ahead.

10.2 Where do you find statements

Java, in the same style as C and C++, places statements within functions (or methods). The function in turn is placed within a class declaration. If the above statement was the only one in the program, it would look similar to this:

```
public class MyProgram
{
    public static void main (String[] args)
    {
        int age = 24;
    }
}
```

The class declaration and function declaration will be described in the upcoming chapters.

10.3 Variables

Christmases are usually exciting, birthdays too. And the one thing that makes them exciting are: you get presents. Think of a present you've ever gotten. A

tiny (or big, if you're lucky) box with your name on it. Now think of variables as something similar. They are tiny little boxes in the computer's memory that save something within themselves. This something within them is called a value.

Note:

A **variable** is an identifier to a value in the system's memory.

10.4 Data types

Take a look at the code in Listing 1.1. Here the variable we have just created is `age`. The word `int` tells us what is inside the `age` variable. `int` actually stands for integer – a number. On the right to this variable is the value of the variable which is the number 24. Just like `int`, we can use `byte`, `short`, `long`, `double`, `float`, `boolean`, `char` or `String`. All these tell us what type of data is within a variable. These are hence called **data types**.

We explored the statement in Listing 1.1, where the variable held an integer within in. Let's put another type of data within it. Let's say, the number 10.5. The code would look something like this:

Listing 1.2: Putting a number with decimal point inside an integer variable.

```
int age = 10.5;
```

This is actually wrong. By definition (if you have been awake throughout your mathematics lectures) you'd know that integers are whole numbers: 0, 1, 2, all the way up to infinity. Anything with a decimal point is not an integer, hence the statement by virtue of it is wrong.

What would make it right is when you assign a certain type to the variable that would accept numbers with decimal points – numbers with decimal points are called **floating points**.

10.5 Whole numbers and floating point numbers

The data types that one can use for whole numbers are `byte`, `short`, `int` and `long` but when it comes to floating point numbers, we use `float` or `double`. Now that we know that, we can modify the code in Listing 1.2 as:

Listing 1.3: The correct way to assign a type to floating point variables

```
double age = 10.5;
```

Why not `float`, you say? Well, there are several reasons why not. 10.5 can be anything – a `float` or a `double` but by a certain rule, it is given a certain type. This can be explained further by looking at the table below.

Data type	Values accepted	Declaration
<code>byte</code>	Any number between -128 and 127.	<code>byte b = 123;</code>
<code>short</code>	Any number between -32,768 and 32,767.	<code>short s = 12345;</code>

Data type	Values accepted	Declaration
int	Any number between -2,147,483,648 and 2,147,483,647.	int i = 1234567;
long	Any number between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807.	long l = 1234567890L;
float	Extremely large numbers beyond the scope of discussion here.	float f = 123.4567f;
double	Extremely large numbers beyond the scope of discussion here. The only difference between double and float is the addition of an f as a suffix after the float value.	double d = 1234.56789;

The above table only list the number data types. We will look at the others as we go on. So, did you notice why we used a `double` in listing 1.3, and not a `float`? The answer is pretty simple. If we'd used a `float`, we would have to append the number with a `f` as a suffix, so `10.5` should be `10.5f` as in:

Listing 1.4: The correct way to define floating point numbers of type `float`.

```
float age = 10.5f;
```

10.6 Assignment statements

Up until now, we've assumed the creation of variables as a single statement. In essence, we assign a value to those variables, and that's just what it is called. When you assign a value to a variable in a statement, that statement is called an **assignment statement**. Did you notice one more thing? The semicolon (;). It's at the end of each statement. A clear indicator that a line of code is a statement is its termination with an ending semicolon. If one was to write multiple statements, it is usually done on each separate line ending with a semicolon. Consider the example below:

Listing 1.5: Multiple assignment statements.

```
int a = 10;
int b = 20;
int c = 30;
```

You do not necessarily have to use a new line to write each statement. Just like English, you can begin writing the next statement where you ended the first one as depicted below:

Listing 1.6: Multiple assignment statement on the same line.

```
int a = 10; int b = 20; int c = 30;
```

However, the only problem with writing such code is, it's very difficult to read it back. It doesn't look that intimidating at first, but once you've got a significant amount of code, it's usually better to organize it in a way that makes sense. It would look more complex and incomprehensible written as it is in Listing 1.6.

Now that we have looked into the anatomy of a simple assignment statement, we can look back at what we've achieved. We know that...

- A statement is a unit of code in programming.
- If we are assigning a variable a value, the statement is called an assignment statement.
- An assignment statement include three parts: a data type, variable name (also called an identifier) and the value of a variable. We will look more into the nature of identifiers and values in the section titled IDENTIFIERS, LITERALS AND EXPRESSIONS¹ later.

Now, before we more on to the next topic, you need to try and understand what the code below does.

Listing 1.7: Multiple assignment statements with expressions

```
int firstNumber = 10;
int secondNumber = 20;
int result = firstNumber + secondNumber;
```

The first two statements are pretty much similar to those in Listing 1.5 but with different variable names. The third however is a bit interesting. We've already talked of variables as being similar to gift boxes. Think of your computer's memory as a shelf where you put all those boxes. Whenever you need a box (or variable), you call its identifier (that's the name of the variable). So calling the variable identifier `firstNumber` gives you the number 10, calling `secondNumber` would give you 20 hence when you add the two up, the answer should be 30. That's what the value of the last variable `result` would be. The part of the third statement where you add the numbers, i.e., `firstNumber + secondNumber` is called an **expression** and the expression is what decides what the value is to be. If it's just a plain value, nothing fancy, then it's called a **literal**.

With the information you have just attained, you can actually write a decent Java program that can sum up values. To learn more, continue reading.

¹ [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FIDENTIFIERS%2C%20LITERALS%20AND%20EXPRESSIONS](http://en.wikibooks.org/wiki/JAVA%20PROGRAMMING%2FIDENTIFIERS%2C%20LITERALS%20AND%20EXPRESSIONS)

11 Classes, Objects and Types

11.1 Objects and Classes

An **object** is composed of **members** and **methods**. The members, also called *data members*, *characteristics*, *attributes*, or *properties*, describe the object. The methods generally describe the actions associated with a particular object. Think of an object as a noun, its members as adjectives describing that noun, and its methods as the verbs that can be performed by or on that noun.

For example, a sports car is an object. Some of its members might be its height, weight, acceleration, and speed. An object's members just hold data about that object. Some of the methods of the sports car could be "drive", "park", "race", etc. The methods really don't mean much unless associated with the sports car, and the same goes for the members.

The blueprint that lets us build our sports car object is called a **class**. A class doesn't tell us how fast our sports car goes, or what color it is, but it does tell us that our sports car will have a member representing speed and color, and that they will be say, a number and a word (or hex color code), respectively. The class also lays out the methods for us, telling the car how to park and drive, but these methods can't take any action with just the blueprint - they need an object to have an effect.

In Java, a class is located in a file similar to its own name. If you want to have a class called `SportsCar`, its source file needs to be `SportsCar.java`. The class is created by placing the following in the source file:

```
public class SportsCar
{
    /* Insert your code here */
}
```

The class doesn't do anything yet, as you will need to add methods and member variables first.

11.2 Instantiation and Constructors

In order to get from class to object, we "build" our object by **instantiation**. Instantiation simply means to create an **instance** of a class. Instance and object are very similar terms and are sometimes interchangeable, but remember that an instance refers to a *specific object*, which was created from a class.

This instantiation is brought about by one of the class's methods, called a **constructor**. As its name implies, a constructor builds the object based on the blueprint. Behind the scenes, this

means that computer memory is being allocated for the instance, and values are being assigned to the data members.

In general there are four constructor types: default, non-default, copy, and cloning.

A **default constructor** will build the most basic instance. Generally, this means assigning all the members values like null, zero, or an empty string. Nothing would stop you, however, from your default sports car color from being red, but this is generally bad programming style. Another programmer would be confused if your basic car came out red instead of say, colorless.

A **non-default constructor** is designed to create an object instance with prescribed values for most, if not all, of the object's members. The car is red, goes from 0-60 in 12 seconds, tops out at 190mph, etc.

A **copy constructor** is not included in the Java language, however one can easily create a constructor that do the same as a copy constructor. It's important to understand what it is. As the name implies, a copy constructor creates a new instance to be a duplicate of an already existing one. In Java, this can be also accomplished by creating the instance with the default constructor, and then using the assignment operator to equivocate them. This is not possible in all languages though, so just keep the terminology under your belt.

Java has the concepts of **cloning object**, and the end results are similar to copy constructor. Cloning an object is faster than creation with the new keyword, because all the object memory is copied at once to destination cloned object. This is possible by implementing the Cloneable interface, which allows the method Object.clone() to perform a field-by-field copy.

11.3 Type

When an object is created, a reference to the object is also created. The object can not be accessed directly in Java, only through this object reference. This object reference has a **type** assigned to it. We need this type when passing the object reference to a method as a parameter. Java does strong type checking.

Type is basically a list of features/operations, that can be performed through that object reference. The object reference type basically is a contract that guarantees that those operations will be there at run time.

When a car is created, it comes with a list of features/operations listed in the user manual that guarantees that those will be there when the car is used.

When you create an object from a class by default its type is the same as its class. It means that all the features/operations the class defined are there and available, and can be used. See below:

```
(new ClassName()).operations();
```

You can assign this to a variable having the same type as the class:

```
ClassName objRefVariable = new ClassName();  
objRefVariable.operations();
```

You can assign the created object reference to the class super class, or to an interface the class implements:

```
SuperClass objectRef = new ClassName(); // features/operations list are defined
by the SuperClass class
..
Interface inter = new ClassName(); // features/operations list are defined by the
interface
```

In the car analogy, the created car may have different **Type** of drivers. We create separate user manuals for them, Average user manual, Power user manual, Child user manual, or Handi-capped user manual. Each type of user manual describes only those features/operations appropriate for the type of driver. The Power driver may have additional gears to switch to higher speeds, that are not available to other type of users...

When the car key is passed from an adult to a child we replacing the user manuals, that is called **Type Casting**.

In Java, casts can occur in three ways:

- up casting: going up in the inheritance tree, until we reach the **Object**
- up casting: to an interface the class implements
- down casting: until we reach the class the object was created from

Type and **Type Casting** will be covered in more details later at 'JAVA PROGRAMMING/TYPES¹' module.

11.4 Multiple classes in a Java file

Normally, a Java file can have one and only one **public** Java class. However, a given file can contain additional non-public classes.

```
public class OuterClass
{
    ...
}
class AdditionalClass
{
    ...
}
```

Because they have the "package (default)" access specifier, the 'AdditionalClass' can be accessed only in the same package.

These "additional" classes compile to separate ".class" bytecode files when compiled, just as if they were in separate source files. However, including multiple classes in one file may increase the difficulty in examining the structure of a given application.

¹ Chapter 17 on page 97

11.5 External links

- [CONSTRUCTORS, INTERACTIVE JAVA LESSON](#)²

[CATEGORY:JAVA PROGRAMMING](#)³

[DE:JAVA STANDARD: CLASS](#)⁴ [ES:PROGRAMACIÓN EN JAVA/CLASES](#)⁵ [FR:PROGRAMMATION JAVA/LES CLASSES EN JAVA](#)⁶ [IT:JAVA/CLASSI E OGGETTI](#)⁷ [NL:PROGRAMMEREN IN JAVA/KLASSEN](#)⁸
[PT:JAVA/INTRODUÇÃO ÀS CLASSES](#)⁹

2 [HTTP://JAVALESSONS.COM/CGI-BIN/FUN/JAVA-TUTORIALS-MAIN.CGI?SES=A0789&CODE=CTR&SUB=FUN](http://javalessons.com/cgi-bin/fun/java-tutorials-main.cgi?ses=a0789&code=ctr&sub=fun)

3 [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/Category%3AJava%20Programming)

4 [HTTP://DE.WIKIBOOKS.ORG/WIKI/JAVA%20STANDARD%3A%20CLASS](http://de.wikibooks.org/wiki/Java%20Standard%3A%20Class)

5 [HTTP://ES.WIKIBOOKS.ORG/WIKI/PROGRAMACI%F3N%20EN%20JAVA%2FCLASES](http://es.wikibooks.org/wiki/Programaci%F3n%20en%20Java%2FClases)

6 [HTTP://FR.WIKIBOOKS.ORG/WIKI/PROGRAMMATION%20JAVA%2FLES%20CLASSES%20EN%20JAVA](http://fr.wikibooks.org/wiki/Programmation%20Java%2FLes%20Classes%20en%20Java)

7 [HTTP://IT.WIKIBOOKS.ORG/WIKI/JAVA%2FCLASSI%20E%20OGGETTI](http://it.wikibooks.org/wiki/Java%2FClassi%20e%20oggetti)

8 [HTTP://NL.WIKIBOOKS.ORG/WIKI/PROGRAMMEREN%20IN%20JAVA%2FKLASSEN](http://nl.wikibooks.org/wiki/Programmeren%20in%20Java%2FKlassen)

9 [HTTP://PT.WIKIBOOKS.ORG/WIKI/JAVA%2FINTRODU%7E3O%20%20E0S%20CLASSES](http://pt.wikibooks.org/wiki/Java%2FIntrodu%7E3o%20%20e0s%20Classes)

12 Packages

12.1 Java Package / Name Space

Usually a Java application is built by many developers and it is common that third party modules/classes are integrated. The end product can easily contain hundreds of classes. Class name collision is likely to happen. To avoid this a Java class can be put in a "name space". This "name space" in Java is called the **package**.

The Java package needs to be unique across Vendors to avoid name collisions. For that reason Vendors usually use their domain name in reverse order. That is guaranteed to be unique. For example a company called 'Your Company Inc.', would use a package name something like this: `com.yourcompany.yourapplicationname.yourmodule.YourClass`.

To put a class in a package, the `package` keyword is used at the top of each class file. For Example,

```
package com.yourcompany.yourapplication.yourmodule;
```

When we want to reference a Java class that is defined outside of the current package name space, we have to specify which package name space that class is in. So we could reference that class with something like `com.yourcompany.yourapplication.yourmodule.YourClass`. To avoid having to type in the package name each time when we want to reference an outside class, we can declare which package the class belongs to by using the **import** Java keyword at the top of the file. For Example,

```
import com.yourcompany.yourapplication.yourmodule.YourClass;
```

Then we can refer to that class by just using the class name *YourClass*.

In rare cases it can happen that you need to reference two classes having the same name in different packages. In those cases, you can not use the `import` keyword for both classes. One of them needs to be referenced by typing in the whole package name. For Example,

```
package com.mycompany.myapplication.mymodule;
...
import com.yourcompany.yourapplication.youmodule.SameClassName;
...
SameClassName yourObjectRef = new SameClassName();
com.hercompany.herapplication.hermodule.SameClassName herObjectRef =
    new com.hercompany.herapplication.hermodule.SameClassName();
```

The Java package has one more interesting characteristic; the package name corresponds where the actual file is stored on the file system. And that is actually how the compiler and the class loader find the Java files on the file system. For example, the class *com.yourcompany.yourapplication.yourmodule.YourClass*, is stored on the file system in the corresponding directory : *com/yourcompany/yourapplication/yourmodule/YourClass*. Because of this, package names should be lowercase, since in some operating systems the directory names are not case sensitive.

12.2 Wildcard imports

It is possible to import an entire package, using an asterisk:

```
import javax.swing.*;
```

While it may seem convenient, it may cause problems if you make a typographical error. For example, if you use the above import to use `JFrame`, but then type `JFram frame=new JFram();`, the Java compiler will report an error similar to "Cannot find symbol: `JFram`". Even though it seems as if it was imported, the compiler is giving the error report at the first mention of `JFram`, which is half-way through your code, instead of the point where you imported `JFrame` along with everything else in `javax.swing`.

If you change this to `import javax.swing.JFrame;` the error will be at the import instead of within your code.

Furthermore, if you `import javax.swing.*;` and `import java.util.*;`, and `javax.swing.Queue` is later added in a future version of Java, your code that uses `Queue` (`java.util`) will fail to compile. This particular example is fairly unlikely, but if you are working with non-Sun libraries, it may be more likely to happen.

12.3 Importing packages from .jar files

If you are importing library packages or classes that reside in a `.jar` file, you must ensure that the file is in the current classpath (both at compile- and execution-time). Apart from this requirement, importing these packages and classes is the same as if they were in their full, expanded, directory structure.

Example:

To compile and run a class from a project's top directory (that contains the two directories `/source` and `/libraries`) you could use the following command:

```
javac -classpath libraries/lib.jar source/MainClass.java
```

And then to run it, similarly:

```
java -classpath libraries/lib.jar source/MainClass
```

(The above is simplified, and demands that `MainClass` be in the default package, or a package called `'source'`, which isn't very desirable.)

12.4 Class Loading / Name Space

A fully qualified class name consists of the package name plus the class name. For example, the fully qualified class name of `HashMap` is `java.util.HashMap`. Sometime it can happen that two classes have the same name, but they can not belong to the same package, otherwise it would be the same class. It can be said that the two classes with the same name are in different name spaces. In the above example, the `HashMap` class is in the `java.util` name space.

Let be two `Customer` classes with different name spaces (in different packages):

- `com.bluecompany.Customer`
- `com.redcompany.Customer`

when we need to use both classes in the same program file, we can use the **import** keyword only for one of the classes. For the other we need to use the fully qualified name.

The runtime identity of a class in Java 2 is defined by the fully qualified class name and its defining class loader. This means that the same class, loaded by two different class loaders, is seen by the Virtual Machine as two completely different types.

FR:PROGRAMMATION JAVA/EXTENSIONS¹ IT:JAVA/PACKAGE²

¹ [HTTP://FR.WIKIBOOKS.ORG/WIKI/PROGRAMMATION%20JAVA%2FEXTENSIONS](http://fr.wikibooks.org/wiki/Programmation%20Java%2FExtensions)

² [HTTP://IT.WIKIBOOKS.ORG/WIKI/JAVA%2FPACKAGE](http://it.wikibooks.org/wiki/JAVA%2FPackage)

13 Nested Classes

In Java you can define a class inside an other class.

A class can be nested:

- inside another class,
- or inside a method

13.1 Nest a class inside a class

When a class is declared inside another class, the nested class' access modifier can be **public**, **private**, **protected** or package (default).

```
public class OuterClass
{
    private String outerInstanceVar;

    public class InnerClass
    {
        public void printVars()
        {
            System.out.println( "Print Outer Class Instance Var.:" + outerInstanceVar);
        }
    }
}
```

The inner class has access to the enclosing class instance's variables and methods, even private ones, as seen above. This makes it very different from the nested class in C++, which are equivalent to the "static" inner classes, see below.

An inner object has a reference to the outer object. The nested object can only be created with a reference to the 'outer' object. See below.

```
public void testInner()
{
    ...
    OuterClass outer = new OuterClass();
    OuterClass.InnerClass inner = outer.new InnerClass();
    ...
}
```

(When in a non-static method of the outer class, you can directly use `new InnerClass()`, since the class instance is implied to be `this`.)

You can directly access the reference to the outer object from within an inner class with the syntax `OuterClass.this`; although this is usually unnecessary because you already have access to its fields and methods.

Inner classes compile to separate ".class" bytecode files, usually with the name of the enclosing class, followed by a "\$", followed by the name of the inner class. So for example, the above inner class would typically be compiled to a file named "OuterClass\$InnerClass.class".

13.1.1 Static inner class

An inner class can be declared *static*. A static inner class has no enclosing instance, and therefore cannot access instance variables and methods of the outer class. You do not specify an instance when creating a static inner class. This is equivalent to the inner classes in C++.

13.2 Nest a class inside a method

These inner classes, also called *local classes*, cannot have access modifiers, like local variables, since the class is 'private' to the method. The inner class can be only **abstract** or **final**.

```
public class OuterClass
{
    public void method()
    {
        class InnerClass
        {
        }
    }
}
```

In addition to instance variables of the enclosing class, local classes can also access local variables of the enclosing method, but only ones that are declared *final*. This is because the local class instance might outlive the invocation of the method, and so needs its own copy of the variable. To avoid problems with having two different copies of a mutable variable with the same name in the same scope, it is required to be *final*, so it cannot be changed.

13.3 Anonymous Classes

In Java a class definition and its instantiation can be combined into a single step. By doing that the class does not require a name. Those classes are called anonymous classes. An anonymous class can be defined and instantiated in contexts where a reference can be used, and it is a nested class to an existing class. Anonymous class is a special case of the local class to a method, above; and hence they also can use *final* local variables of the enclosing method.

Anonymous classes are most useful to subclass and upcast to an 'Adapter Class' or to an interface.

```

public interface ActionListener
{
    public void click();
}
...
ActionListener clk = new ActionListener()
{
    public void click()
    {
        // --- implementation of the click event ---
        ...
        return;
    }
};

```

In the above example the class that implements the `ActionListener` is **anonymous**. The class is defined where it is instantiated.

The above code is harder to read than if the class explicitly defined, so why use it? If many implementations are needed for an interface and those classes are used only in one particular place, using **anonymous** class makes sense.

The following example uses anonymous class to implement an action listener.

```

import java.awt.*;
import java.awt.event.*;
import java.io.Serializable;
class MyApp implements Serializable
{
    BigObjectThatShouldNotBeSerializedWithAButton bigOne;
    Button aButton = new Button();
    MyApp()
    {
        aButton.addActionListener( new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                System.out.println("Hello There");
            }
        }
    );
}
}

```

The following example does the same thing, but it names the class that implements the action listener.

```

import java.awt.*;
import java.awt.event.*;
import java.io.Serializable;
class MyApp implements Serializable
{
    BigObjectThatShouldNotBeSerializedWithAButton bigOne;
    Button aButton = new Button();
    // --- Nested class to implement the action listener ---
    class MyActionListener implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {

```

```
        System.out.println("Hello There");
    }
}
MyApp()
{
    aButton.addActionListener( new MyActionListener() );
}
}
```

Using **anonymous** classes is especially preferable when you intend to use many different classes that each implement the same Interface.

CATEGORY:JAVA PROGRAMMING¹

¹ [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/Category%3AJAVA%20Programming)

14 Access Modifiers

14.1 Access modifiers

You surely would have noticed by now, the words **public**, **protected** and **private** at the beginning of class's method declarations used in this book. These keywords are called the **access modifiers** in the Java language syntax, and can be defined as...

Quote:

.. keywords that help set the visibility and accessibility of a class, its member variables, and methods.

The following table shows what Access Modifiers are appropriate for classes, nested classes, member variables, and methods:

	Class	Nested class	Method, or Member variable	Interface	Interface method signature
public	visible from anywhere	same as its class	same as its class	visible from anywhere	visible from anywhere
protected	N/A	its class and its subclass	its class and its subclass, and from its package	N/A	N/A
package (default)	only from its package	only from its package	only from its package	N/A	N/A, default is public
private	N/A	only from its class	only from its class	N/A	N/A

Points to ponder:

Note that Interface method visibility is `PUBLIC`^a by default. You do not need to specify the access modifier it will default to `PUBLIC`^b. For clarity it is considered a good practice to put the `PUBLIC`^c keyword.

The same way all member variables defined in the Interface by default will become `STATIC`^d `FINAL`^e once inherited in a class.

a [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FKEYWORDS%2FPUBLIC](http://en.wikibooks.org/wiki/Java%20Programming%2FKeywords%2FPublic)
b [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FKEYWORDS%2FPUBLIC](http://en.wikibooks.org/wiki/Java%20Programming%2FKeywords%2FPublic)
c [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FKEYWORDS%2FPUBLIC](http://en.wikibooks.org/wiki/Java%20Programming%2FKeywords%2FPublic)
d [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FKEYWORDS%2FSTATIC](http://en.wikibooks.org/wiki/Java%20Programming%2FKeywords%2FStatic)
e [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FKEYWORDS%2FFINAL](http://en.wikibooks.org/wiki/Java%20Programming%2FKeywords%2FFinal)

If a class has public visibility, the class can be referenced by anywhere in the program. If a class has package visibility, the class can be referenced only in the package where the class is defined. If a class has private visibility, (it can happen only if the class is defined nested in an other class) the class can be accessed only in the outer class.

If a variable is defined in a public class and it has public visibility, the variable can be reference anywhere in the application through the class it is defined in. If a variable has package visibility, the variable can be referenced only in the same package through the class it is defined in. If a variable has private visibility, the variable can be accessed only in the class it is defined in.

If a method is defined in a public class and it has public visibility, the method can be called anywhere in the application through the class it is defined in. If a method has package visibility, the method can be called only in the same package through the class it is defined in. If a method has private visibility, the method can be called only in the class it is defined in.

1

¹ [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3A](http://en.wikibooks.org/wiki/Category%3A)

15 Methods

15.1 Method Definition

A method is an operation on a particular object. An object is an instance of a class. When we define a class we define its member variables and its methods. For each method we need to give a name, we need to define its input parameters and we need to define its return type. We also need to set its visibility(private, package, or public). If the method throws an Exception, that needs to be declared as well. The syntax of method definition is:

```
class MyClass
{
    ...
    public ReturnType methodName( ParamOneType param1, ParamTwoType param2 ) throws ExceptionName
    {
        ReturnType retType;
        ...
        return retType;
    }
    ...
}
```

We can declare that the method does not return anything using the **void** java keyword. For example:

```
private void methodName( String param1, String param2 )
{
    ...
    return;
}
```

When the method returns nothing, the **return** keyword at the end of the method is optional. The **return** keyword can be used anywhere in the method, when the execution flow reach the **return** keyword, the method execution is stopped and the execution flow returns to the caller method.

15.2 Method Overloading

For the same class we can define two methods with the same name. However the parameter types and/or the number of parameters must be different for those two methods. In the java terminology, this is called **method overloading**. It is useful to use method overloading when we need to do something different based on a parameter type. For example we may have the operation : runAroundThe. We can define two methods with the same name, but different input

parameter type:

```
public void runAroundThe( Building block )
{
    ...
}
public void runAroundThe( Park park )
{
    ...
}
```

Related terminology is the **method signature**. In java the method signature contains method name and the input parameter types. The java compiler takes the signature for each method and makes sure that each method signature is unique for a class. For example the following two method definitions are valid:

```
public void logIt( String param, Error err )
{
    ...
}
public void logIt( Error err, String param )
{
    ...
}
```

Because the type order is different. If both input parameters were type String, that would be a problem since the compiler would not be able to distinguish between the two:

```
public void logIt( String param, String err )
{
    ...
}
public void logIt( String err, String param )
{
    ...
}
```

The compiler would give an error for the following method definitions as well:

```
public void logIt( String param )
{
    ...
}
public String logIt( String param )
{
    String retType;
    ...
    return retValue;
}
```

Note, the return type is not part of the unique signature. Why not? The reason is that a method can be called without assigning its return value to a variable. This feature came from C and C++.

So for the call:

```
{
  logIt( msg );
}
```

the compiler would not know which method to call.

15.3 Method Overriding

Obviously a method signature has to be unique inside a class. The same method signature can be defined in different classes. If we define a method that exist in the super class then we override the super class method. The terminology for this is **method overriding**. This is different from method overloading. Method overloading happens with methods with the same name different signature. Method overriding happens with same name, same signature between inherited classes.

The return type can cause the same problem we saw above. When we override a super class method the return type also must be the same. In fact if that is not the same, the compiler will give you an error.

Method overriding is related dynamic linking, or runtime binding. In order for the Method Overriding to work, the method call that is going to be called can not be determined at compilation time. It will be decided at runtime, and will be looked up in a table.

```
{
1 MyClass obj = new SubOfMyClass();
2
3 MyClass obj = new MyClass();
4
5 obj.myMethod(); // -- During compilation, it is not known what reference the
'obj' has, MyClass or SubOfMyClass
}
```

In the above example 'obj' reference has the type MyClass on both line 1 and line 3. However the 'obj' reference points two different objects. On line 1 it references SubOfMyClass object, on line 3 it references MyClass object. So on line 5 which method will be called, method define in MyClass, or the method that defined in its subclasses. Because the 'obj' reference can point to object and all its sub object, and that will be known only at runtime, a table needs to be kept with all the possible method address to be called.

Also another rule is that when you do an override, the visibility of the new method that overrides the super class method can not be reduced. The visibility can be increased, however. So if the super class method visibility is public, the override method can not be package, or private.

In the case of the exception the override method may throw can be the same as the super class or it can be one of that exception inherited class. So the common rule is that the override method must throw the same exception or it is any of its subclasses.

NOTE: A common mistake to think that if we can override methods, we could also override member variables. This is not the case, as member variables are not overridden.

```
{
1 MyClass obj = new SubOfMyClass();
2
3 MyClass obj = new MyClass();
4
5 String var = obj.myMemberVar; // -- The myMemberVar is defined in the
  MyClass object
}
```

In the above example, it does not count what object the 'obj' reference points to, because it was declared MyClass type on both line 1 and line 3, the variable in the MyClass object will be referenced. In real examples we rarely use public variables, but if you do keep in mind that Java does not support variable overriding.

15.4 Parameter Passing

We can pass in all the primitive data types or any object references to a method. An object cannot be passed to a method, only its references. All parameters (those are primitive types and object references) are passed by value. In other words if you change the passed in parameter values inside the method, that will have no effect on the original variable that was passed in. When you pass in an object reference to a method and then you change that inside the method, that will have no effect on the original object reference. However if you modify the object itself, that will stay after the method returns. Think about the object reference as a pointer to an object. If you change the object the reference points at, that will be permanent. For example:

```
1 {
2   int var1 = 10;
3   int var2 = 20;
4   ...
5   myMethod( var1, var2 );
6   ...
7   System.out.println( "var1="+var1 +"var2="+var2 ); // -- The variable values
  did not change
8 }
9 ...
10 void myMethod( int var1, int var2 )
11 {
12   ...
13   var1 = 0;
14   var2 = 0;
15   ...
16 }
```

On line 7 the value of var1 is 10 and the value of var2 is 20. When the variables were passed in to the methods their values were copied. This is called passing the parameter by value. In java we do not represent an object directly, we represent an object through an **object reference**. You can think of an object reference as a variable having the address of the object. So the object reference passed in by value, but the object itself is not. For example:

```
1 {
2   MyObjOne obj = new MyObjOne();
3   obj.setName("Christin");
4   ...
5   myMethod( obj );
6   String name = obj.getName(); // --- The name attribute was changed to
'Susan' inside the method
7 }
8 void myMethod( MyObjOne obj )
9 {
10  obj.setName("Susan");
11  ...
12  obj = new MyObjOne();
13  obj.setName("Sonya");
14  ...
15 }
```

On line 2, we created an object, on line 3 we set its name property to 'Christin'. On line 5 we called the `myMethod(obj)`. Inside the method, we changed the name to 'Susan' through the passed in object reference. So that change will stay. Note however that after we reassigned the `obj` reference to a new object, that is no effect whatsoever on the passed in object.

15.5 Functions

In java, functions (methods really) are just like in C++ except that they must be declared inside a class and objects are passed by reference automatically. You cannot create pointers to a function but Java has events which really are function pointers under the hood for when you need that type of functionality.

```
int a_function(double d)
{
    return (int)d;
}
```

15.6 Return Parameter

So as we can see, a method may or may not return a value. If the method does not return a value we use the **void** java keyword. Same as the parameter passing, the method can return a primitive type or an object reference. So a method can return only one value. What if you want to return more than one value from a method. You can always pass in an object reference to the method, and let the method modify the object properties. The modified values can be considered as an output value from the method. However better option, and cleaner if you create an Object array inside the method, assign the return values and return the array to the caller. You could have a problem however, if you want to mix primitive data types and object references as the output values from the method. There is a better approach. Defines special return object with the needed return values. Create that object inside the method, assign the values and return the reference to this object. This special object is "bound" to this method and used only for returning values, so do not use a public class. The best way is to use a nested class, see example

below:

```
public class MyObject
{
    ...

    /** Nested object is for return values from 'getPersonInfoById' method */
    public static class ReturnObj
    {
        private int    age;
        private String name;

        public void setAge( int val )
        {
            this.age = val;
        }
        public int  getAge()
        {
            return age;
        }

        public void setName( String val )
        {
            name = val;
        }
        public String getName()
        {
            return name;
        }
    } // --- End of nested class defination ---

    /** Method using the nested class to return values */
    public ReturnObj getPersonInfoById( int ID )
    {
        int    age;
        String name;
        ...
        // --- Get the name and age based on the ID from the database ---
        ...
        ReturnObj ret = new ReturnObj();
        ret.setAge( age );
        ret.setName( name );

        return ret;
    }
}
```

In the above example the 'getPersonInfoById' method returns an object reference that contains both values the name and age. See below how you may use that object:

```
{
    ...
    MyObject obj = new MyObject();
    MyObject.ReturnObj person = obj.getPersonInfoById( 102 );

    System.out.println( "Person Name=" + person.getName() );
    System.out.println( "Person Age =" + person.getAge() );
    ...
}
```


15.7 Special method, the Constructor

There is a special method for each class that will be executed each time an object is created from that class. That is the **Constructor**. Constructor does not have a return value and its name is the same as the class name. The Constructor can be overloaded; you can define more than one constructor with different parameters. For example:

```
public class MyClass
{
    private String memberField;

    /**
     * MyClass Constructor, there is no input parameter
     */
    public MyClass()
    {
        ...
    }

    /**
     * MyClass Constructor, there is one input parameter
     */
    public MyClass( String param1 )
    {
        memberField = param1;
        ...
    }
}
```

In the above example we defined two constructors, one with no input parameter, and one with one input parameter. You may ask which constructor will be called. Its depends how the object is created with the **new** keyword. See below:

```
{
    ...
    MyClass obj1 = new MyClass();           // The constructor with no input
parameter will be called
    MyClass obj2 = new MyClass("Init Value"); // The constructor with one input
param. will be called
    ...
}
```

In the above example we created two objects from the same class, or we can also say that obj1 and obj2 both have the same type. The difference between the two is that in the first one the memberVar field is not initialized, in the second one that is initialized to 'Init Value'. obj1, and obj2 contains the reference to the object. Each class must have a constructor. If we do not define one, the compiler will create a default so called empty constructor automatically.

```
public class MyClass
{
    /**
     * MyClass Empty Constructor
     */
    public MyClass()
    {
```

```
    }  
}
```

The Constructor is called automatically when an object is created with the **new** keyword. A constructor may also be called from an other constructor, see below:

```
public class MyClass  
{  
    private String memberField;  
  
    /**  
     * MyClass Constructor, there is no input parameter  
     */  
    public MyClass()  
    {  
        MyClass("Default Value");  
    }  
  
    /**  
     * MyClass Constructor, there is one input parameter  
     */  
    public MyClass( String param1 )  
    {  
        memberField = param1;  
        ...  
    }  
}
```

In the above example, the constructor with no input parameter calls the other constructor with the default initial value. This gives an option to the user, to create the object with the default value or create the object with a specified value.

15.8 Static Methods

We defined methods above as operations on an object. Static methods are defined inside a class, but they are not an operation on an object. No object needs to be created to execute a static method, they are simply global functions, with input parameters and a return value.

```
public class MyObject  
{  
    static public String myStaticMethod()  
    {  
        ...  
        return("I am a static method");  
    }  
}
```

Static methods can be referenced anywhere prefixed by the class name. See below:

```
{  
    ...  
}
```

```

// --- Call myStaticMethod ---
System.out.println( "Output from the myStaticMethod:" + MyObject.myStaticMethod() );
...
}

```

You can write a non object oriented program by using only static methods in java. Because java evolved from the C programming language, static methods are left over from a non object oriented language.

You write static methods the same way you do normal methods, the only difference is that you cannot reference any member variables or any object methods. Static methods can reference only static variables and call only other static methods. However you can create an object and use it inside a static method.

```

public class MyObject
{
    public String memberVar;

    static private String memberStaticVar;

    static public String myStaticMethod()
    {
        memberVar = "Value"; --> ERROR Cannot reference member var.

        memberStaticVar = "Value"; // --- This is okay, static vars. can be used

        // --- Create an object ---
        MyObject obj = new MyObject();
        obj.memberVar = "Value"; // --- This is okay since an object is created --
        ...
        return("I am a static method");
    }
}

```

15.9 External links

- [ABSTRACT METHODS, INTERACTIVE JAVA LESSON¹](http://javalessons.com/cgi-bin/fun/java-tutorials-main.cgi?ses=ao789&code=abs&sub=fun)

¹ [HTTP://JAVALESSONS.COM/CGI-BIN/FUN/JAVA-TUTORIALS-MAIN.CGI?SES=AO789&CODE=ABS&SUB=FUN](http://javalessons.com/cgi-bin/fun/java-tutorials-main.cgi?ses=ao789&code=abs&sub=fun)

16 Primitive Types

1. [redirect JAVA PROGRAMMING/TYPES](#)¹

¹ [Chapter 17 on page 97](#)

17 Types

Data Types (or simply **Types**) in Java are a way of telling what certain data *is*. It is seen as a way of declaring allowed values for certain data, the structure of such data and operations associated with it. Any data, be it numeric or a sentence of words, can have a different data type. For instance, just to define different types of numbers in Java, there are about six simple types available to programmers – some define whole numbers (integers numbers) and others define numbers with a decimal values (floating point numbers). Java also gives freedom to programmers to create complex and customizable data types. We will deal with complex data types in later chapters.

17.1 Data Types in Java

Java is considered a **strongly typed** programming language in that it is obligatory for all data, expressions and declarations within the code to have a certain type associated with it. This is either declared or inferred and the Java language only allows programs to run if they adhere to type constraints.

As we have discussed above, you can have types that define a number, or types that define textual content within your program. If you present a numeric type with data that is not numeric, say textual content, then such declarations would violate Java's type system. This gives Java the unique ability of **type safety**. Java checks if an expression or data is encountered with an incorrect type or none at all. It then automatically flags this occurrence as an error at compile time. Most type-related errors are caught by the Java compiler, hence making a program more secure and safe once compiled completely and successfully.

In the Java language, there are three broad categories of data types:

- PRIMITIVES¹
- OBJECT REFERENCE TYPES²
- ARRAYS³

In the following section, we will discuss these three categories in detail.

17.1.1 Primitives

Primitives are the most basic data types available within the Java language. These types serve as the building blocks of data manipulation in Java. Such types serve only one purpose – containing pure, simple values of a kind. Because these data types are defined into the Java type system by

1 [HTTP://EN.WIKIBOOKS.ORG/WIKI/%23PRIMITIVES](http://en.wikibooks.org/wiki/%23Primitives)

2 [HTTP://EN.WIKIBOOKS.ORG/WIKI/%23OBJECT%20REFERENCE%20TYPES](http://en.wikibooks.org/wiki/%23Object%20Reference%20Types)

3 [HTTP://EN.WIKIBOOKS.ORG/WIKI/%23ARRAYS](http://en.wikibooks.org/wiki/%23Arrays)

default, they come with a number of operations predefined. You can not define a new operation for such primitive types. In the Java type system, there are three further categories of primitives:

- **NUMERIC PRIMITIVES**⁴These primitive data types hold only numeric data. Operations associated with such data types are those of simple arithmetic (addition, subtraction, etc.) or of comparisons (is greater than, is equal to, etc.)
- **TEXTUAL PRIMITIVES**⁵These primitive data types hold characters (which can be alphabets or even numbers), but unlike numbers, they do not have operations that serve arithmetic purposes. Rather, operations associated with such types are those of textual manipulation (comparing two words, joining characters to make words, etc.)
 - **BOOLEAN AND NULL PRIMITIVES**⁶

17.1.2 Object Reference Types

17.1.3 Arrays

17.2 Java as hybrid language

On the one hand, Java is a **strongly typed language** in that all expressions and declarations have types (either declared or inferred) and the language only allows programs which adhere to the type constraints. Therefore, most type errors are caught and detected by the Java compiler. Some type errors can still occur at runtime because Java supports a cast operation which is a way of changing the type of one expression to another. However, Java performs run time type checking when doing such casts, so an incorrect type cast will cause a runtime exception rather than succeeding silently and allowing data corruption.

On the other hand, Java is also known as a **hybrid language**. While supporting object oriented (OO) programming, Java is not a pure OO language like SMALLTALK⁷ or RUBY⁸. Instead, Java offers both object types and **PRIMITIVE TYPES**⁹. Primitive types are used for boolean, character, and numeric values and operations. This allows relatively good performance when manipulating numeric data, at the expense of flexibility. For example, you cannot subclass the primitive types and add new operations to them.

17.3 Examples of Types

Below are two examples of Java types and a brief description of the allowed values and operations for these types. Additional details on each are available in other modules.

17.3.1 Example: int

The **PRIMITIVE TYPE**¹⁰ **int** represents a signed 32 bit integer value. The allowed data values for **int** are the integers between -2147483648 to 2147483647 inclusive.

4 [HTTP://EN.WIKIBOOKS.ORG/WIKI/%2FNUMERIC%20PRIMITIVES](http://en.wikibooks.org/wiki/%2FNUMERIC%20PRIMITIVES)

5 [HTTP://EN.WIKIBOOKS.ORG/WIKI/%2FTEXTUAL%20PRIMITIVES](http://en.wikibooks.org/wiki/%2FTEXTUAL%20PRIMITIVES)

6 [HTTP://EN.WIKIBOOKS.ORG/WIKI/%2FOther%20PRIMITIVES](http://en.wikibooks.org/wiki/%2FOther%20PRIMITIVES)

7 [HTTP://EN.WIKIBOOKS.ORG/WIKI/PROGRAMMING%3ASmalltalk](http://en.wikibooks.org/wiki/PROGRAMMING%3ASmalltalk)

8 [HTTP://EN.WIKIBOOKS.ORG/WIKI/Ruby%20Programming](http://en.wikibooks.org/wiki/Ruby%20Programming)

9 Chapter 16 on page 95

10 Chapter 16 on page 95

The set of operations that may be performed on `int` values includes integer arithmetic such as `+`, `-`, `*`, `/`, `%`, comparison operations (`==`, `!=`, `<`, `>`, `<=`, `>=`), assignments (`=`, `++`, `--`, `+=`, `-=`), bit-wise operations such as logical and, logical or, logical xor, negation (`&`, `|`, `^`, `~`), bit shift operations (`<<`, `>>`, `>>>`), CONVERSIONS¹¹ to other numeric types and PROMOTION¹² to other integer types.

For example, to declare a **private** integer instance field named `length`, one would use the declaration `private int length;`

17.3.2 Example: String

You use **class** and **interface** definition in Java to define new types. Class and interface types are associated with object references also sometime referred to as *Reference types*. An object reference has two main attributes:

- Its **type** associated with a class or an interface
- A java object it references, that is created by instantiating a class

The **String** class is one such example. String values are a sequence of 0 or more Unicode characters. The **null** reference is another valid value for a String expression.

The operations on a String reference variable are those available for all reference types, such as comparison operations `==`, `!=` and assignment `=`.

The allowed operations on String object, however are the set of methods in the `java.lang.String` class, which includes `length()`, `toString()`, `toLowerCase()`, `toUpperCase()`, `compareTo(String anotherString)` and more... .

In addition, String objects also inherit the set of operations from the base class that String extends from, which is `java.lang.Object`. These operations include methods such as `equals()`, `hashCode()`, `wait()`, `notifyAll()`, and `getClass()`. **private String** `name` = "Marry Brown"; In the above example the *name* object reference's attributes are:

- **Type is : String**
- The **referenced object** is also : **String**

Both the `java.lang.String` class methods and `java.lang.Object` class methods are available for the object reference *name*. **private Object** `name` = "Marry Brown"; In the above example the *name* object reference's attributes are:

- **Type is : Object**
- The **referenced object** is : **String**

Only the `java.lang.Object` class methods are available for the object reference *name*.

17.4 Array Types

Arrays in Java are represented as a built-in Array object. As with object types, they behave as a reference but have a few differences allowing them to allow easy access to sub elements.

When one declares an array, the data type is changed to include square brackets. (While you can instead place these square brackets next to the variable name, this is not recommended.)

To create an array, you will need to use the new operator to have it create the Array with the specified number of elements:

```
/* Declares an array named data1, and has it assigned to an array with 25
```

¹¹ Chapter 16 on page 95

¹² Chapter 16 on page 95

```

elements. */
private int[] data1 = new int[25];

/* Declares an array named data2 that contains these three elements. */
private int[] data2 = new int[] { 1, 99, -2 }

```

Arrays are described in more detail in the Arrays section.

17.5 Primitive Data Types

The Java primitive data types contain pure values, no operations. It is basically data types similar to what other non-object-oriented languages have.

There are arrays of primitive types in Java; but because they are not objects, primitive values can not be put in a collection.

For this reason object wrappers are defined in JDK 'java.lang.*' package for all the primitive types.

	Size (bits)	Example	Minimum Value	Maximum Value	Wrapper Class
Integer types					
byte	8	byte b = 65;	-128	127	Byte
char	16	char c = 'A'; char c = 65;	0	$2^{16}-1$	Character
short	16	short s = 65;	-2^{15}	$2^{15}-1$	Short
int	32	int i = 65;	-2^{31}	$2^{31}-1$	Integer
long	64	long l = 65L;	-2^{63}	$2^{63}-1$	Long
Floating-point types					
float	32	float f = 65f;			Float
double	64	double d = 65.55;			Double
Other					
boolean	1	boolean b = true;	--	--	Boolean
void	--	--	--	--	Void

The types **short**, **int**, **long**, **float**, and **double** are usually used in arithmetic operations; **byte** and **char** can also be used, but less commonly.

The character type **char** is used for text processing. The type **byte** is commonly used in binary file input output operations.

String objects representing literal character strings in Java, in the java.lang.* package. java.lang.String is not a primitive type, but instead is a special class built into the Java language. For further info, see **String**.

17.6 Data Conversion (Casting)

Data conversion (casting) can happen between two primitive types. There are two kinds:

- Implicit : casting operation is not required; the magnitude of the numeric value is always preserved. However, *precision* may be lost when converting from integer to floating point types
- Explicit : casting operation required; the magnitude of the numeric value may not be preserved

Example for implicit casting: `int i = 65; long l = i; // --- int is converted to long, casting is not needed`

Example for explicit casting: `long l = 656666L; int i = (int) l; // --- long is converted to int, casting is needed`

The following table shows the conversions between primitive types, it shows the casting operation for explicit conversions:

	from byte	from char	from short	from int	from long	from float	from double	from boolean
to byte	-	(byte)	(byte)	(byte)	(byte)	(byte)	(byte)	N/A
to char		-	(char)	(char)	(char)	(char)	(char)	N/A
to short		(short)	-	(short)	(short)	(short)	(short)	N/A
to int				-	(int)	(int)	(int)	N/A
to long					-	(long)	(long)	N/A
to float						-	(float)	N/A
to double							-	N/A
to boolean	N/A	N/A	N/A	N/A	N/A	N/A	N/A	-

17.7 Autoboxing/unboxing

Autoboxing/unboxing

Autoboxing and unboxing, language features since Java 1.5, make the programmer's life much easier when it comes to working with the primitive wrapper types. Consider this code fragment:

```
int age = 23; Integer ageObject = new Integer(age);
```

Primitive wrapper objects were Java's way of allowing one to treat primitive data types as though they were objects. Consequently, one was expected to 'wrap' one's primitive data type with the corresponding primitive wrapper object, as shown above.

Autoboxing

Since Java 1.5, one may write as below and the compiler will automatically create the wrap object. The extra step of wrapping the primitive is no longer required. It has been 'automatically boxed up' on your behalf.

Points to ponder:

Keep in mind that the compiler still creates the missing wrapper code, so one doesn't really gain anything performance-wise. Consider this feature a programmer convenience, not a performance booster.

```
int age = 23; Integer ageObject = age;
```

Unboxing

Uses the same process in reverse. Study the following code for a moment. The **if** statement requires a **boolean** primitive value, yet it was given a Boolean wrapper object. No problem!

Java 1.5 will automatically 'unbox' this.

```
Boolean canMove = new Boolean(true);
```

```
if ( canMove ) { System.out.println( "This code is legal in Java 1.5" ); }
```

CATEGORY:JAVA PROGRAMMING¹³

DE:JAVA_STANDARD: PRIMITIVE DATENTYPEN¹⁴ FR:PROGRAMMATION JAVA/TYPES DE BASE¹⁵

IT:JAVA/TIPI DI DATI¹⁶ PT:JAVA/TIPOS DE DADOS PRIMÁRIOS¹⁷

¹³ [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/Category%3AJAVA%20PROGRAMMING)

¹⁴ [HTTP://DE.WIKIBOOKS.ORG/WIKI/JAVA_STANDARD%3A%20PRIMITIVE%20DATENTYPEN](http://de.wikibooks.org/wiki/JAVA_STANDARD%3A%20PRIMITIVE%20DATENTYPEN)

¹⁵ [HTTP://FR.WIKIBOOKS.ORG/WIKI/PROGRAMMATION%20JAVA%2FTYPES%20DE%20BASE](http://fr.wikibooks.org/wiki/Programmation%20Java%2FTypes%20de%20base)

¹⁶ [HTTP://IT.WIKIBOOKS.ORG/WIKI/JAVA%2FTIPI%20DI%20DATI](http://it.wikibooks.org/wiki/JAVA%2FTipi%20di%20dati)

¹⁷ [HTTP://PT.WIKIBOOKS.ORG/WIKI/JAVA%2FTIPOS%20DE%20DADOS%20PRIM%20%C3%A1RIOS](http://pt.wikibooks.org/wiki/JAVA%2FTipos%20de%20dados%20prim%C3%A1rios)

18 java.lang.String

UNKNOWN TEMPLATE
ifeq: Types
none ← TYPES¹

Java Programming
String

UNKNOWN TEMPLATE
ifeq: Classes, Objects and
Types
none CLASSES, OBJECTS
AND TYPES² →

18.1 java.lang.String

String is a special class built into the Java language defined in the `java.lang` package.

The **String** class represents character strings. String literals in Java programs, such as "abc", are implemented as instances of this class.

For example:

```
String str = "This is string literal";
```

On the right hand side a String object is created represented by the string literal. Its object reference is assigned to the `str` variable.

Strings are *immutable*; that is, they cannot be modified once created. Whenever it looks as if a String object was modified, a new String was actually created and the old one was thrown away.

The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings. For example:

```
String str = "First part" + " second part";  
// --- Is the same as:  
String str = "First part second part";
```

Integers will also be converted to String after the (+) operator:

```
String str = "Age=" + 25;
```

¹ Chapter 17 on page 97

² Chapter 11 on page 71

Each Java object has the `String toString()` inherited from the **Object** class. This method provides a way to convert objects into `Strings`. Most classes override the default behavior to provide more specific (and more useful) data in the returned `String`.

The `String` class provides a nice set of methods for string manipulation. Since `String` objects are immutable, all methods return a new `String` object. For example:

```
name = name.trim();
```

The `trim()` method returns a copy of the string with leading and trailing whitespace removed. Note that the following would do nothing useful:

```
name.trim(); // wrong!
```

This would create a new trimmed string and then throw it away. Study the `String` class and its methods carefully. `Strings` are ubiquitous in Java; it will serve you well to know how to manipulate them skillfully.

18.2 Using `StringBuffer`/`StringBuilder` to concatenate strings

Remember that `String` objects are immutable objects. Once a `String` is created, it can not be modified, takes up memory until garbage collected. Be careful of writing a method like this :

```
public String convertToString(Collection<String> coll)
{
    String str = "";
    for(String oneElem : coll) // loops through every element in coll
    {
        str = str + oneElem + " ";
    }
    return str;
}
```

On the (+) operation a new **String** object is created at each iteration. Suppose `coll` contains the elements ["Foo", "Bar", "Bam", "Baz"]. The method creates five `Strings` ("", "Foo ", "Foo Bar ", "Foo Bar Bam ", and "Foo Bar Bam Baz") even though only last one is actually useful.

Instead use `STRINGBUFFER`³, as shown below, where only one `STRINGBUILDER`⁴ object is created:

3 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FAPI%2FJAVA.LANG.STRINGBUFFER](http://en.wikibooks.org/wiki/Java%20Programming%2FAPI%2FJava.Lang.StringBuffer)

4 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FAPI%2FJAVA.LANG.STRINGBUILDER](http://en.wikibooks.org/wiki/Java%20Programming%2FAPI%2FJava.Lang.StringBuilder)

To avoid unnecessary memory use like this, use the `STRINGBUFFER`⁵ or `STRINGBUILDER`⁶ class. They provide similar functionality to **Strings**, but store their data in a mutable way. Also because object creation is time consuming, using `StringBuffer` or `StringBuilder` produces much faster code.

```
public String convertToString(Collection<String> coll)
{
    

StringBuilder buf = new StringBuilder();
        for(String oneElem : coll) // loops through every element in coll
        {
            buf.append(oneElem);
            buf.append(" ");
        }
        return buf.toString();


}
```

`StringBuilder` was introduced in Java 5. Unlike `StringBuffer`, `StringBuilder` isn't thread safe, so you can't use it in more than one thread (see the chapter on `CONCURRENCY`⁷). However, because it doesn't have to worry about synchronization, `StringBuilders` are faster.

18.3 Comparing Strings

Comparing strings is not as easy as it may first seem. We cannot just use a simple equality statement such as:

```
if(myString == "Hello World!") //Can't Use this.
{
    

System.out.println("Match Found");


}
```

To test for equality, use the `equals(Object)` method inherited by every class and defined by `String` to return **true** if and only if the object passed in is a `String` containing the exact same data.

```
String greeting = "Hello World!";
if(greeting.equals("Hello World!")) { //true
    

// ...


}
```

5 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FAPI%2FJAVA.LANG.STRINGBUFFER](http://en.wikibooks.org/wiki/Java%20Programming%2FAPI%2FJava.Lang.StringBuffer)

6 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FAPI%2FJAVA.LANG.STRINGBUILDER](http://en.wikibooks.org/wiki/Java%20Programming%2FAPI%2FJava.Lang.StringBuilder)

7 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FCONCURRENTPROGRAMMING](http://en.wikibooks.org/wiki/Java%20Programming%2FConcurrentProgramming)

```
}  
if(greeting.equals("hello world!")) { //false  
    // ...  
}
```

To order `String` objects, use the **`compareTo()`** method, which can be accessed wherever we use a `String` datatype. Let's take a look at an example:

```
String myString = "Hello World!";  
//...  
if(myString.compareTo("Hello World!") == 0 )  
{  
    System.out.println("Match Found");  
}
```

This snippet of code is comparing the `String` variable `myString` to "Hello World". The `compareTo` method returns a negative, zero, or positive number if the parameter is less than, equal to, or greater than the object on which it is called. If *myString* was to be different, even in the slightest manner we will get a value above or below 0 depending on the exact difference. The result is negative if this `String` object lexicographically precedes the argument string. The result is a positive integer if this `String` object lexicographically follows the argument string. Take a look at the [JAVA API](#)⁸ for more details.

18.4 Splitting a String

Sometimes it is useful to split a string into separate strings, based on a *regular expression*. (For more information on regular expressions, see [REGEX](#)⁹.) The **`String`** class has a `split()` method, since Java 1.4, that will return a `String` array.

See the following example:

```
String person = "Brown, John:100 Yonge Street, Toronto:(416)777-9999";  
...  
String[] personData = person.split( ":" );  
...  
String name      = personData[0];  
String address  = personData[1];  
String phone     = personData[2];
```

8 [HTTP://JAVA.SUN.COM/J2SE/1.4.2/DOCS/API/JAVA/LANG/STRING.HTML#COMPARETO \(JAVA.LANG.STRING\)](http://java.sun.com/j2se/1.4.2/docs/api/java/lang/String.html#compareTo(java.lang.String))

9 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%2FREGEX](http://en.wikibooks.org/wiki/JAVA%2FREGEX)

An other useful application could be to 'split' the String text based on the 'new line' character, so you could process the text line by line.

18.5 Creating substrings

It may also be sometimes useful to create **substrings**, or strings using the order of letters from an existing string. This can be done in two methods.

The first method involves creating a substring out of the characters of a string from a given index to the end.

For example:

```
String str    = "coffee";
String substr = str.substring(3);
```

In this example, `substr` would return "fee". As previously discussed, the index of the first character in a string is 0. By counting from there, it is apparent that the character in index 3 is the second "f" in "coffee". This is known as the `beginIndex`. All characters from the `beginIndex` until the end of the string will be copied into the new substring.

The second method involves a user-defined `beginIndex` and `endIndex`. For example:

```
String str = "supporting";
String substr = str.substring(3,7);
```

The string returned by `substr` would be "port". Please note that the `endIndex` is **not** inclusive. This means that the last character will be of the index `endIndex-1`. Therefore, in this example, every character from index 3 to index 6, inclusive, was copied into the substring.

Note: "Substring" is considered to be one word. This is why the method name does not seem to follow the common syntax of Java. It is easy to mistake the method `substr()` for `subStr()` (which does not exist and would return with a syntax error on compilation). Just remember that this style only applies to methods or other elements that are made up of more than one word.

18.6 Modifying String cases

The String Class also allows for the modification of cases. The two methods that make this possible are `toLowerCase()` and `toUpperCase()`. These methods are useful, for example, in the typical programming classroom assignment of evaluating whether or not a string is a palindrome.

```
String a = "WIKIBOOKS";
String b = "wikipedia";
```

In this example, a call to `a.toLowerCase()` would return a result of "wikibooks", and `b.toUpperCase()` would return "WIKIPEDIA".

18.7 See also

- [JAVA API: JAVA.LANG.STRING](#)¹⁰
- [JAVA API: JAVA.LANG.STRINGBUFFER](#)¹¹
- [JAVA API: JAVA.LANG.STRINGBUILDER](#)¹²
- [JAVA PROGRAMMING/API/JAVA.LANG.STRINGBUFFER](#)¹³
- [JAVA PROGRAMMING/API/JAVA.LANG.STRINGBUILDER](#)¹⁴

¹⁰ [HTTP://JAVA.SUN.COM/J2SE/1.5.0/DOCS/API/JAVA/LANG/STRING.HTML](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html)

¹¹ [HTTP://JAVA.SUN.COM/JAVASE/6/DOCS/API/JAVA/LANG/STRINGBUFFER.HTML](http://java.sun.com/javase/6/docs/api/java/lang/StringBuffer.html)

¹² [HTTP://JAVA.SUN.COM/JAVASE/6/DOCS/API/JAVA/LANG/STRINGBUILDER.HTML](http://java.sun.com/javase/6/docs/api/java/lang/StringBuilder.html)

¹³ [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FAPI%2FJAVA.LANG.STRINGBUFFER](http://en.wikibooks.org/wiki/JAVA%20PROGRAMMING%2FAPI%2FJAVA.LANG.STRINGBUFFER)

¹⁴ [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FAPI%2FJAVA.LANG.STRINGBUILDER](http://en.wikibooks.org/wiki/JAVA%20PROGRAMMING%2FAPI%2FJAVA.LANG.STRINGBUILDER)

19 Arrays

19.1 Intro to Arrays

An **array** is similar to a table of data, keyed by number. In Java an array is an object like all other objects. Look at the following program:

```
import java.util.*;

public class ArrayExample {

    static Scanner input = new Scanner(System.in);

    public static void main(String[] args) {
        int numNames = getInt("Number of names?");
        String[] names = new String[numNames];
        for (int i = 0; i < names.length; i++) {
            names[i] = getString("Enter name #" + (i+1));
        }
        for (int i = 0; i < names.length; ++i) {
            System.out.println(names[i]);
        }
    }

    public static int getInt(String prompt) {
        System.out.print(prompt + " ");
        int integer = input.nextInt();
        input.nextLine(); // Get rid of this line
                          // so that getString won't read it
        return integer;
    }

    public static String getString(String prompt) {
        System.out.print(prompt + " ");
        return input.nextLine();
    }
}
```

Copy the code and compile it. The program will ask you to enter some names then reprints the names in order. It demonstrates three major aspects of arrays: how to define an array, how to set data, and how to access it. The code `String[] names = new String[numNames];` tells Java to create an array of size `numNames` that will store Strings. To set data, use `names[x] = data` where `x` is the index to access. Note that all Java arrays start at 0 and go to (array size - 1). Thus, if you dimension an array to size 10, the highest index is 9.

19.2 Array Fundamentals

- To create an array, use the syntax **`DataType[] variable = new DataType[ArraySize]`**. Alternatively, if you know the data beforehand, you can write **`DataType[] variable = {item 1, item 2,...item n}`**

- All elements of the array will be automatically initialized with the *default value* for that datatype. This is *false* for booleans, *0* for all numeric primitive types, and *null* for all reference types. So for example, the previous note created an array of `DataType` references, all of which are initialized to *null*.
- To access an item, use the syntax `variable[i]` where *i* is the index
- To set an item, use the syntax `variable[i] = data`
- To find the length of an array, use the syntax `variable.length`

19.3 Two-Dimensional Arrays

A two-dimensional array is represented by an array of arrays. Because an array is also an object, like any other object having the **Object** as the super class, it can be used to create an array where the element of the array is also an array. In this way an array with any number of dimensions can be created. Here are examples of two dimensional arrays with initializer blocks:

```
String[][] twoDimArray = {"a", "b", "c", "d", "e"},
                        {"f", "g", "h", "i", "j"},
                        {"k", "l", "m", "n", "o"};

int[][] twoDimIntArray = { { 0, 1, 2, 3, 4},
                          {10, 11, 12, 13, 14},
                          {20, 21, 22, 23, 24}};
```

Note that the above "twoDimArray" is equivalent to the following more verbose code:

```
String[][] twoDimArray = new String[3][];
for(int i = 0; i < twoDimArray.length; i++) {
    twoDimArray[i] = new String[5];
    for(int j = 0; j < twoDimArray[i].length; j++)
        twoDimArray[i][j] = "" + i + j;
}
```

In the above example we defined an array which has three elements, each element contains an array having 5 elements. We could create the array having the 5 elements first and use that one in the initialize block.

```
String[] oneDimArray = { "00", "01", "02", "03", "04" };
String[][] twoDimArray = { oneDimArray ,
                          {"10", "11", "12", "13", "14"},
                          {"20", "21", "22", "23", "24"} };
```

Since they are arrays of array references, these multi-dimensional arrays can be "jagged" (i.e. subarrays can have different lengths), or the subarray reference can even be null. Consider:

```
String[][] weirdTwoDimArray = {"10", "11", "12"},
                              null,
                              {"20", "21", "22", "23", "24"};
```

Note that the length of a two-dimensional array is the number of one-dimensional arrays it contains. In the above example, `weirdTwoDimArray.length` is 3, whereas `weirdTwoDimArray[2].length` is 5.

19.4 Multidimensional Array

Going further any number of dimensional array can be defined.

```
<elementType>[] [] ... [] <arrayName>  
or  
<elementType><arrayName>[] [] ... []
```

DE:JAVA_STANDARD: DATENSTRUKTUREN¹ ES:PROGRAMACIÓN EN JAVA/ARRAYS²
FR:PROGRAMMATION JAVA/TABLEAUX³ NL:PROGRAMMEREN IN JAVA/ARRAYS⁴ PT:JAVA/VETORES⁵

1 [HTTP://DE.WIKIBOOKS.ORG/WIKI/JAVA_STANDARD%3A%20DATENSTRUKTUREN](http://de.wikibooks.org/wiki/JAVA_STANDARD%3A%20DATENSTRUKTUREN)
2 [HTTP://ES.WIKIBOOKS.ORG/WIKI/PROGRAMACION%20EN%20JAVA%2FARRAYS](http://es.wikibooks.org/wiki/PROGRAMACION%20EN%20JAVA%2FARRAYS)
3 [HTTP://FR.WIKIBOOKS.ORG/WIKI/PROGRAMMATION%20JAVA%2FTABLEAUX](http://fr.wikibooks.org/wiki/PROGRAMMATION%20JAVA%2FTABLEAUX)
4 [HTTP://NL.WIKIBOOKS.ORG/WIKI/PROGRAMMEREN%20IN%20JAVA%2FARRAYS](http://nl.wikibooks.org/wiki/PROGRAMMEREN%20IN%20JAVA%2FARRAYS)
5 [HTTP://PT.WIKIBOOKS.ORG/WIKI/JAVA%2FVETORES](http://pt.wikibooks.org/wiki/JAVA%2FVETORES)

20 Data and Variables

A variable in Java can store two kinds of variables:

- JAVA PRIMITIVE TYPE VALUES¹
- a reference to a JAVA OBJECT²

Java's primitive types are

- integers (whole numbers, declared as **byte**, **short**, **int**, or **long**; only **int** need be of interest to a beginner)
- floating-point numbers (decimal numbers, declared as **float** or **double**; only **float** need be of interest at first)
- characters (declared as **char**, representing one character like 'A' or ';')
- **boolean** (holding only *true* or *false* as values)

In addition, the Java language has special features for its String class, and strings can be treated very much like primitives for many purposes.

As in most languages, a variable is declared to be a particular type of data; the syntax for a declaration is:

```
variabletype variablename;
```

To store a value in a variable, a program statement like

```
variablename = data;
```

And can reference the variable (and use the data stored in it) by its name.

For example, to create an **int** primitive type value, named **year** that stores 2007;

```
year = 2007;
```

To access the data in *year*, use the variable in place of the number.

```
System.out.println(year);
```

Produces

¹ Chapter 16 on page 95

² Chapter 23 on page 129

2007

20.1 Strong Typing

Variables in Java are *strongly typed*, which means that the compiler checks the type of a variable matches the type of data stored in that variable. If you declare a variable to hold a `String`, for instance, you cannot assign an integer value to that variable. Some languages (such as C) define an interpretation of such a statement and use that interpretation without any warning; others (such as PL/I) define a conversion for almost all such statements and perform the conversion to complete the assignment. Strong typing is intended to prevent mistakes made by unwittingly assigning the wrong kind of value to a variable, and catching those mistakes when the program is compiled rather than waiting to find it when the program is running.

20.2 Case Conventions

Java is case-sensitive. A method called `myMethod` is completely separate from a method called `myMethod`. Be careful!

By convention, most identifiers that includes more than one word uses CAMEL CASE³. Classes begin with a capital letter; methods and variables do not. (Constructors have to start with a capital, because they must have the same name as the class.) Package names use lowercase, and do not use camel case. Thus:

```
package org.wikibooks.samplecode;

class CaseConventions {
    int variable;
    int multipleWordVariable;

    CaseConventions(String id) {
    }

    void method() {
    }

    void longMethodName() {
    }
}
```

20.3 Scope

Java uses **block scope**, which means that a variable is "un-defined" (and becomes useless) at the end of the block in which it is defined. A *block* is any section of code within curly braces. Common blocks include class definitions, methods and constructors, `if/else` blocks, and `for`,

³ [HTTP://EN.WIKIPEDIA.ORG/WIKI/CAMELCASE](http://en.wikipedia.org/wiki/CamelCase)

while, and do-while loops.

```
class BlockScope {
    int classScope; // valid in all of class BlockScope

    BlockScope(int param) { // param is valid only in this constructor
        int localVariable = 0; // valid only in this constructor
    }

    void someMethod() {
        int local = 42; // valid only in this method

        if(local > 0) {
            boolean positive = true; // valid only within the if block
        } else {
            // positive is not defined here!
        }
    }
}
```

There are three basic kinds of scope for variables in Java:

- local variable, declared within a method in a class, valid for (and occupying storage only for) the time that method is executing. Every time the method is called, a new copy of the variable is used.
- instance variable, declared within a class but outside any method. It is valid for and occupies storage for as long as the corresponding object is in memory; a program can instantiate multiple objects of the class, and each one gets its own copy of all instance variables. This is the basic data structure rule of Object-Oriented programming; classes are defined to hold data specific to a "class of objects" in a given system, and each instance holds its own data.
- static variable, declared within a class as *static*, outside any method. There is only one copy of such a variable no matter how many objects are instantiated from that class.

CATEGORY:JAVA PROGRAMMING⁴

⁴ [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/Category%3AJAVA%20Programming)

21 Generics

Generics were added to the Java language syntax in version 1.5. This means that code using Generics will not compile with Java 1.4 and less.

Java was long criticized for the need to explicitly type-cast an element when it was taken out of a "container/collection" class. There was no way to enforce that a "collection" class contains only one type of object (e.g., to forbid *at compile time* that an `Integer` object is added to a `Collection` that should only contain `Strings`). This is now possible since Java 1.5.

In the first couple of years of Java evolution, Java did not have a real competitor. This has changed by the appearance of Microsoft C#. With Generics Java is better suited to compete against C#. Similar constructs to Java Generics exist in other languages, see [W:GENERIC PROGRAMMING](#)¹ for more information.

21.1 What are Generics?

Generics are so called because this language feature allows methods to be written generically, with no foreknowledge of the type on which they will eventually be called upon to carry out their behaviors. A better name might have been **type parameter argument**. Because, it is basically that, to pass a `Type` as a parameter to a class at creation time.

When an object is created, parameters can be passed to the created object, through the constructor. Now with Generics, we can also pass in `Types`. The type-place-holders will be replaced with the specified type, before the object is created.

Type parameter arguments can be set:

for a class

When an object is created from that class the type-parameter-argument will be replaced with the actual `Type`.

```
public class Person<T>
{
    private Person<T> person;
    ...
}
...
// --- Create an Employee person ---
Person<Employee> emplPerson = new Person<Employee>();
...
// --- Create a Customer person ---
Person<Customer> custPerson = new Person<Customer>();
```

¹ [HTTP://EN.WIKIPEDIA.ORG/WIKI/GENERIC%20PROGRAMMING](http://en.wikipedia.org/wiki/Generic%20programming)

for a method

Just like class declarations, method declarations can be generic--that is, parameterized by one or more type parameters.

```
public static <T> void assign( Person<T> person, T obj )
{
    person.setPerson( obj );
}
```

use of generics is optional

For backwards compatibility with pre-Generics code, it is okay to use generic classes without the generics type specification thing (<T>). In such a case, when you retrieve an object reference from a generic object, you will have to manually typecast it from type Object to the correct type. The compiler should also warn about unsafe operations.

21.2 Introduction

Java is a strongly typed language. That's one of the reasons why it is so easy to use. Many potential problems are caught by the compiler. One area where Java was criticized was regarding the container objects. Container objects are objects that contain other objects. Before Generics were introduced there was no way to ensure that a container object contains only one type of objects. When an object was added to a container, it was automatically cast to Java **Object**. When it was taken out an explicit cast was needed. Normally an explicit cast is checked by the compiler.

```
String st = "This is a String";
...
Integer integer = (Integer) st; // --- Compilation Error --
```

But in the case of container classes, the compiler was not able to catch an invalid type casting.

```
1 Collection collString = new ArrayList();
2 collString.add( "This is a String" );
...
3 Integer integer = (Integer) collString.get(0); // --- No Compilation Error; RunTime CastException
```

Just looking at line 3, we do not know what type of objects *collString* contains. If that contains Integers then the code is fine.

The below code using Generic:

```
Collection<String> collString = new ArrayList<String>();
collString.add( "This is a String" );
...
Integer integer = (Integer) collString.get(0); // --- Compilation Error
```

collString is a container object, that can contain only *String* objects, nothing else, so when we get out an element it can be casted only to class that normally a String can be casted.

With Generics, Java strict type checking can be extended to container objects. Using Generics with container classes, gives an impression that a new container type is created, with each different type parameter. Before Generics:

```
Collection collCustomer = new ArrayList();
collCustomer.add( new Customer() );
...
Collection collObject = collCustomer; // --- No problem, both collObject and collCustomer have the same type
```

With generics:

```
Collection<Customer> collCustomer = new ArrayList<Customer>();
collCustomer.add( new Customer() );
...
Collection<Object> collObject = collCustomer; // --- Compilation Error
```

Both *collObject* and *collCustomer* have the same type, **BUT it is against the Generic rule**, that is *collCustomer* can contain only Customer objects, and *collObject* can contain only Object object. So there is an additional check to the normal type checking, the type of the parameter type has to be matched too.

21.3 Note for C++ programmers

Java Generics are similar to C++ Templates in that both were added for the same reason. The syntax of Java Generic and C++ Template are also similar.

There are some differences however. The C++ template can be seen as a kind of macro, that generates code before compilation. The generated code depends on how the Template class is referenced. The amount of code generated depends on how many different types of classes are created from the Template. C++ Templates do not have any run-time mechanisms. The compiler creates normal code to substitute the template, similar to any 'hand-written' code.

In contrast, Java Generics are built into the language. The same Class object handles all the Generic type variations. No additional code is generated, no matter how many Generic objects are created with different type parameters. For example.

```
Collection<String> collString = new ArrayList<String>();
Collection<Integer> collInteger = new ArrayList<Integer>();
```

There is only one Class object created. In fact, at runtime, both these objects appear as the same type (both *ArrayList*'s). The generics type information is erased during compilation (type erasure). This means, for example, that if you had function that takes *Collection<T>* as an argument, and that collection happened to be empty, your function would have no way of instantiating another T object, because it doesn't know what T was.

The Class **class** itself is generic since Java 1.5.

```

public final class Class<T> extends Object
    implements Serializable, GenericDeclaration, Type, AnnotatedElement
{
    ...
}

```

The **T** type here represents the type that is handed to the `Class` object. The **T** type will be substituted with the class being loaded.

21.4 Class<T>

Since Java 1.5, the class `java.lang.Class` is generic. It is an interesting example of using genericness for something other than a container class.

For example, the type of `String.class` is `Class<String>`, and the type of `Serializable.class` is `Class<Serializable>`. This can be used to improve the type safety of your **reflection code**.

In particular, since the `newInstance()` method in `Class` now returns a `T`, you can get more precise types when creating objects reflectively.

Now we can use the `newInstance()` method to return a new object with exact type, without casting.

```

Customer cust = Utility.createAnyObject(Customer.class); // - No casting
...
public static <T> T createAnyObject(Class<T> cls)
{
    T ret = null;
    try
    {
        ret = cls.newInstance();
    }
    catch (Exception e)
    {
        // --- Exception Handling
    }
    return ret;
}

```

And the above code without Generics:

```

Customer cust = (Customer) Utility.createAnyObject(Customer.class); // - Casting is needed
...
public static Object createAnyObject(Class cls)
{
    Object ret = null;
    try
    {
        ret = cls.newInstance();
    }
    catch (Exception e)
    {
        // --- Exception Handling
    }
}

```

```

    return ret;
}

```

Get exact type when getting JavaBean property, using reflection

See the following code where the method will return the exact type of the Java Bean property, based on how it will be called.

```

// --- Using reflection, get a Java Bean property by its name ---
public static <T> T getProperty(Object bean, String propertyName)
{
    if (bean == null ||
        propertyName == null ||
        propertyName.length() == 0)
    {
        return null;
    }
    // --- Based on the property name build the getter method name ---
    String methodName = "get" +
        propertyName.substring(0,1).toUpperCase() +
        propertyName.substring(1);
    T property = null;
    try
    {
        java.lang.Class c = bean.getClass();
        java.lang.reflect.Method m = c.getMethod(methodName, null);
        property = (T) m.invoke(bean, null);
    }
    catch (Exception e)
    {
        // --- Handle exception --
    }
    return property;
}

```

21.5 Variable Argument

Using Generics, it is very easy to define a method with a variable number of arguments. Before generics, this was not possible in Java—if a likewise feature was needed, this was mostly done by passing an array. The only requirement for using a variable number of arguments using Generics is that the arguments in the list must have the same type.

The following code illustrates a method that can be called with a variable number arguments:

```

/**
 * Method using variable-length argument list
 * @param <T>
 * @param args
 */
public static <T> List<T> makeAList(T... args)
{
    List<T> argList = new ArrayList<T>();
    for (int i = 0; i < args.length; i++)
    {
        argList.add(args[i]);
    }
    // List<T> argList = Arrays.asList(args);
}

```

```
    return argList;
}
```

The above method can be called with a variable number of arguments, for example:

```
List<String> list1 = makeAList("One", "Two", "Three");
List<String> list2 = makeAList("One", "Two", "Three", "Four");
```

In the above example calls, the arguments must be of type `String`. If we write `<? extends Object>` instead of `T`, then we can pass *any* kind of objects, regardless of their type:

```
List<? extends Object> list3 = makeAList("One", 10, new StringBuffer(), new
LinkedList());
```

Note: the number 10 in the above code will be converted (autoboxed) to `Integer`.

See also:

```
java.util.Arrays.asList(T... a)
```

21.6 Wildcard Types

As we have seen above, generics give the impression that a new container type is created with each different type parameter. We have also seen that in addition to the normal type checking, the type parameter has to match as well when we assign generics variables.

In some cases this is too restrictive. What if we would like to relax this additional checking? What if we would like to define a collection variable that can hold any generic collection, regardless of the parameter type it holds?

Wildcard

The wildcard type is represented by the character `<?>`, and pronounced **Unknown**, or **Any-Type**. This **Unknown** type matches anything, if it is used only by itself. Any-Type can be expressed also by `<? extends Object>`. Any-Type includes Interfaces, not only Classes.

So now we can define a collection whose element type matches anything. See below:

```
Collection<?> collUnknown;
```

Note that we can not add anything to this collection. We can only take out elements of type **Object** from it. So what is the use of this variable if we can not add anything to the collection it represents? The use of this new construct will be clear when you want to create a generic method that takes any collection.


```

public static void printElements( Collection<?> anycoll )
{
    Iterator<?> iter = anycoll.iterator();
    while ( iter.hasNext() )
    {
        System.out.print( iter.next() );
    }
}

```

Wildcard for a specific type of classes

"<? extends **ClassName**>" specifies a restriction on the types of classes that may be used.

For example, to create a collection that may only contain "Serializable" objects, specify:

```

Collection<? extends Serializable> serColl = new ArrayList<String>();

```

The above code is valid because, the **String** class is serializable. Use of a class that is not serializable would cause a compilation error.

The following collection can only contain objects that extend the class **Animal**.

```

class Dog extends Animal
{
    ...
}
...
// --- Create "Animal Collection" variable ---
Collection<? extends Animal> animalColl = new ArrayList<Dog>();

```

"<? super **ClassName**>" specifies a restriction on the types of classes that may be used.

For example, to declare a **Comparator** that can compare **Dogs**, you use

```

Comparator<? super Dog> myComparator;

```

Now suppose you define a comparator that can compare **Animals**:

```

class AnimalComparator implements Comparator<Animal>
{
    int compare(Animal a, Animal b) { //...
    }
}

```

Since **Dogs** are **Animals**, you can use this comparator to compare **Dogs** also. Comparators for any superclass of **Dog** can also compare **Dog**; but comparators for any strict subclass cannot.

```
Comparator<Animal> myAnimalComparator = new AnimalComparator();

static int compareTwoDogs(Comparator<? super Dog> comp, Dog dog1, Dog dog2) {
    return comp.compare(dog1, dog2);
}
```

The above code is valid because, the `Animal` class is a supertype of the `Dog` class. Use of a class that is not a supertype would cause a compilation error.

CATEGORY:JAVA PROGRAMMING²

FR:PROGRAMMATION JAVA/TYPES GÉNÉRIQUES³

² [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/Category%3AJava%20Programming)

³ [HTTP://FR.WIKIBOOKS.ORG/WIKI/PROGRAMMATION%20JAVA%20TYPES%20G%E9N%E9RIQUES](http://fr.wikibooks.org/wiki/Programmation%20Java%20Types%20G%C3%A9n%C3%A9riques)

22 Defining Classes

22.1 Fundamentals

Every class in Java can be composed of the following elements

- **fields** - Fields are variables that hold data specific to each object. For example, an employee might have an ID number. (They are also called member variables.) There is one field for each object of a class.
- **member methods** - Member methods perform operations on an object. For example, an employee might have a method to issue his paycheck or to access his name.
- **static fields** - Static fields are common to any object of the same class. For example, a static field within the Employee class could keep track of the last ID number issued. Only one static field exists for one class.
- **static methods** - Static methods are methods that do not affect a specific object.
- **other classes** - Sometimes it is useful to contain a class within another one if it is useless outside of the class or should not be accessed outside the class.
- **Constructors** - A special method that generates a new object.
- **Parameterized types** - Since 1.5, 'parameterized types' can be assigned to a class during definition. The 'parameterized types' will be substituted with the types specified at the class's instantiation. It is done by the compiler. It is similar to the C language macro '#define' statement, where a preprocessor evaluates the macros.

```
public class Employee // This defines the Employee class.
{ // The public modifier indicates
    that
        // it can be accessed by any other class
        private static int nextID; // Define a static field. Only one copy of
        this will exist, // no matter how many Employees are created.
```

```
private int myID;           // Define fields that will be stored
private String myName;     // for each Employee. The private modifier
                             // indicates that
                             // only code inside the Employee class can
                             // access it.

public Employee(String name) // This is a constructor. You can pass a
name to the constructor
{
    myName = name;         // and it will give you a newly created
    myID = nextID;         // Employee object.
    nextID++;             // Automatically assign an ID to the object
                             // Increment the ID counter
}

public String getName()    // This is a member method that returns the
{
    return myName;        // Employee object's name.
                             // Note how it can access the private field
                             // myName.
}

public int getID()        // This is another member method.
{
    return myID;
}

public static int getNextID() // This is a static method that returns the
next ID
{
    return nextID;        // that will be assigned if another Employee
                             // is created.
}
}
```

The following Java code

```
public class EmployeeList {
    public static void main(String[] args) {

        System.out.println(Employee.getNextID());

        Employee a = new Employee("John Doe");
        Employee b = new Employee("Jane Smith");
        Employee c = new Employee("Sally Brown");

        System.out.println(Employee.getNextID());

        System.out.println(a.getID() + " : " + a.getName());
        System.out.println(b.getID() + " : " + b.getName());
        System.out.println(c.getID() + " : " + c.getName());
    }
}
```

would produce this output:

```
0
3
0 : John Doe
1 : Jane Smith
2 : Sally Brown
```

CATEGORY:JAVA PROGRAMMING¹

¹ [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/Category%3AJAVA%20PROGRAMMING)

23 Creating Objects

23.1 Introduction

Before a Java object can be created the class byte code must be loaded from the file system (with .class extension) to memory. This process of locating the byte code for a given class name and converting that code into a Java CLASS¹ class instance is known as class loading. There is one CLASS² created for each type of Java class.

All objects in java programs are created on heap memory. An object is created based on its class. You can consider a class as a blueprint, template, or a description how to create an object. When an object is created, memory is allocated to hold the object properties. An object reference pointing to that memory location is also created. To use the object in the future, that object reference has to be stored as a local variable or as an object member variable.

The Java Virtual Machine (JVM), keeps track of the usage of object references. If there are no more reference to the object, the object can not be used any more and becomes garbage. After a while the heap memory will be full of unused objects. The JVM collects those garbage objects and frees the memory they allocated, so the memory can be reused again when a new object is created. See below a simple example:

```
{
  // --- Create an object ---
  MyObject obj = new MyObject();

  // --- Use the object ---
  obj.printMyValues();
}
```

The `obj` contains the object reference pointing to an object created from the `MyObject` class. The `obj` object reference is in scope inside the `{}`. After the `}` the object becomes garbage. Object references can be passed in to methods, object references can be returned from methods.

23.2 Creating object with the new keyword

99% of new objects are created using the `new` keyword.

1 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FAPI%2FJAVA.LANG.CLASS](http://en.wikibooks.org/wiki/Java%20Programming%2FAPI%2FJava.Lang.Class)

2 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FAPI%2FJAVA.LANG.CLASS](http://en.wikibooks.org/wiki/Java%20Programming%2FAPI%2FJava.Lang.Class)

```
{
  // --- Create an 'MyObject' for the first time the application started ---
  MyObject obj = new MyObject();
}
```

When an object from the `MyObject` class is created for the first time. The JVM searches the file system for the definition of the class, that is the Java byte code. The file has the extension of `*.class`. The `CLASSPATH` environment variable contains locations where Java classes are stored. The JVM is looking for the `'MyObject.class'` file. Depending on which package the class belongs to, the package name will be translated to a directory path.

When the `'MyObject.class'` file is found, the JVM's class loader loads the class in memory, and creates a `Class` object. The JVM stores the code in memory, allocates memory for the **static** variables, and executes any static initialize block. Memory is not allocated for the object member variables at this point, memory will be allocated for them when an instance of the class, an object, is created.

There is no limit on how many objects from the same class can be created. Code and **static** variables are stored only once, no matter how many objects are created. Memory is allocated for the object member variables when the object is created. Thus, the size of an object is determined not by its code's size but by the memory it needs for its member variables to be stored.

23.3 Creating object by cloning an object

Cloning is not automatically available to classes. There is some help though, as all Java objects inherit the `protected Object clone()` method. This base method would allocate the memory and do the bit by bit copying of the object's states.

You may ask why we need this clone method. Couldn't I create a constructor and just passing in the same object, and do the copying variable by variable? Lets see:

```
public class MyObject
{
  private int memberVar;
  ...
  MyObject( MyObject obj )
  {
    this.memberVar = obj.memberVar;
    ...
  }
  ...
}
```

You might think that accessing the `private memberVar` variable of `obj` would fail but as this is in the same class this code is legal. The `clone()` method copies the whole object's memory in one operation. This is much faster than using the `new` keyword. Object creation with the **new** keyword is expensive, so if you need to create lots of objects with the same type, performance will be better if you create one object and clone new ones from it. See below a factory method that will return a new object using cloning.


```

HashTable _cacheTemplate = new HashTable;
...
/** Clone Customer object for performance reason */
public Customer createCustomerObject()
{
    // --- See if a template object exists in our cache ---
    Customer template = _cacheTemplate.get( "Customer" );
    if ( template == null )
    {
        // --- Create template ---
        template = new Customer();
        _cacheTemplate.put( "Customer", template );
    }
    return template.clone();
}

```

Now, lets see how to make the Customer object cloneable.

- First the Customer class needs to implement the Cloneable Interface.
- Override and make the clone() method **public**, as that is **protected** in the Object class.
- Call the super.clone() method at the beginning of your clone method.
- Override the clone() method in all the subclasses of Customer.

```

public class Customer implements Cloneable
{
    ...
    public Object clone() throws CloneNotSupportedException
    {
        Object obj = super.clone();

        return obj;
    }
}

```

In the above example we used cloning for speed up object creation.

An other use of cloning could be to take a snapshot of an object that can change in time. Lets say we want to store Customer objects in a collection, but we want to disassociate them from the 'live' objects . So before adding the object, we clone them, so if the original object changes from that point forward, the added object won't. Also lets say that the Customer object has a reference to an Activity object that contains the customer activities. Now we are facing a problem, it is not enough to clone the Customer object, we also need to clone the referenced objects. The solution:

- Make the Activity class also cloneable
- Make sure that if the Activity class has other 'changeable' object references, those has to be cloned as well, as seen below
- Change the Customer class clone() method as follows:

```

public class Customer implements Cloneable
{
    Activity _activity;
    ...
    public Customer clone() throws CloneNotSupportedException
    {
        Customer clonedCustomer = (Customer) super.clone();

        // -- Clone the object referenced objects ---
        if ( _activity != null )

```

```
        {
            clonedCustomer.setActivity( (Activity) _activity.clone() );
        }
        return clonedCustomer;
    }
}
```

Note that only mutable objects needs to be cloned. References to unchangeable objects such as String be used in the cloned object without worry.

23.4 Creating object receiving from a remote source

When an object is sent through a network, the object needs to be **recreated** at the receiving host.

Object Serialization

The term Object Serialization refers to the act of converting the object to a byte stream. The byte stream can be stored on the file system, or can be sent through a network.

At the later time the object can be re-created from that stream of bytes. The only requirement is that the same class has to be available at both times, when the object is serialized and also when the object is re-created. If that happens in different servers, then the same class must be available on both servers. Same class means that exactly the same version of the class must be available, otherwise the object won't be able to be re-created. This is a maintenance problem to those applications where java serialization is used to persist object or sent the object through the network.

When a class is modified, there could be a problem re-creating those objects that were serialized using an earlier version of the class.

Java has built in support for serialization, using the Serializable interface; however, a class must first implement the Serializable interface.

By default, a class will have all of its fields serialized when converted into a data stream (with TRANSIENT³ fields being skipped.) If additional handling is required beyond the default of writing all fields, you need to provide an implementation for two methods:

```
private void writeObject(java.io.ObjectOutputStream out) throws IOException;
private void readObject(java.io.ObjectInputStream in) throws IOException,
ClassNotFoundException;
private void readObjectNoData() throws ObjectStreamException;
```

If the object needs to write or provide a replacement object during serialization, it needs to implement the following two methods, with any access specifier:

```
Object writeReplace() throws ObjectStreamException;
Object readResolve() throws ObjectStreamException;
```

3 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FKEYWORDS%2FTRANSIENT](http://en.wikibooks.org/wiki/Java%20Programming%2FKeywords%2FTransient)

Normally, a minor change to the class can cause the serialization to fail. You can still allow the class to be loaded by defining the serialization version id:

```
private static final long serialVersionUID = 42L;
```

CATEGORY:JAVA PROGRAMMING⁴

⁴ [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/Category%3AJAVA%20Programming)

24 Interfaces

24.1 Interfaces

Java does not allow you to create a subclass from two classes. There is no multiple inheritance. The major benefit of that is that all java objects can have a common ancestor. That class is called **Object**. All java classes can be up-casted to **Object**. Example:

```
class MyObject
{
    ...
}
```

When you type the above code, it actually means the following:

```
class MyObject extends Object // -- The compiler adds 'extends Object'. if not specified
{
    ...
}
```

So it can be guaranteed that certain methods are available in all java classes. This makes the language simpler.

To mimic multiple inheritance, java offers *interfaces*, which are similar to abstract classes. In interfaces all methods are abstract by default, without the *abstract* key word. Interfaces have no implementation and no variables, but constant values can be defined in interfaces - however, a single class can implement as many interfaces as required.

```
public interface MyInterface
{
    public static final String CONSTANT = "This value can not be changed";

    public String methodOne(); // This method must be implemented by the class that implements this interface
    ...
}

...

public class MyObject implements MyInterface
{
    // Implement MyInterface interface
    public String methodOne()
    {
        ...
    }
}
```

24.2 External links

- [INTERFACES, INTERACTIVE JAVA LESSON¹](#)
- [INTERFACES 2, INTERACTIVE JAVA LESSON²](#)

[CATEGORY:JAVA PROGRAMMING³](#)

[FR:PROGRAMMATION JAVA/INTERFACES⁴](#)

¹ [HTTP://JAVALESSONS.COM/CGI-BIN/FUN/JAVA-TUTORIALS-MAIN.CGI?SES=AO789&CODE=IF1&SUB=FUN](http://javalessons.com/cgi-bin/fun/java-tutorials-main.cgi?ses=ao789&code=if1&sub=fun)

² [HTTP://JAVALESSONS.COM/CGI-BIN/FUN/JAVA-TUTORIALS-MAIN.CGI?SES=AO789&CODE=IF2&SUB=FUN](http://javalessons.com/cgi-bin/fun/java-tutorials-main.cgi?ses=ao789&code=if2&sub=fun)

³ [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/Category%3AJava%20Programming)

⁴ [HTTP://FR.WIKIBOOKS.ORG/WIKI/PROGRAMMATION%20JAVA%2FINTERFACES](http://fr.wikibooks.org/wiki/Programmation%20Java%2FInterfaces)

25 Using Static Members

25.1 What does static mean?

When you declare a

- method, or
- member variable

static, it is independent of any particular, but rather it is shared among all instances of a class. To access a static method or member variable, no instance needs to be created.

The **static** keyword is used to declare a method, or member variable static.

25.2 What can it be used for?

- Static variables can be used as data sharing amongst objects of the same class. For example to implement a counter that stores the number of objects created at a given time can be defined as so:

```
public AClass
{
    static private int counter;
    ...
    public AClass()
    {
        ...
        counter += 1;
    }
    ...
    public int getNumberOfObjectsCreated()
    {
        return counter;
    }
}
```

The `counter` variable is incremented each time an object is created.

Public static variable should not be used, as these become GLOBAL variables that can be accessed from everywhere in the program. Global constants can be used, however. See below:

```
static public final String CONSTANT_VAR = "Const";
```

- Static methods can be used for utility functions or for functions that do not belong to any particular object. For example:

```
public Match
{
    ...
    public static int addTwoNumbers( int par1, int par2 )
    {
        return par1 + par2;
    }
}
```

25.3 Danger of static variables

Use static variables only for:

- data sharing (be careful)
- defining global constants

Use static methods for:

- utility functions

Using static variables and/or method for other purposes goes against object orientation, and will make your code either harder to read or maintain.

25.4 External links

- [STATIC, INTERACTIVE JAVA LESSON¹](http://javalessons.com/cgi-bin/fun/java-tutorials-main.cgi?ses=a0789&code=stc&sub=fun)

[CATEGORY:JAVA PROGRAMMING²](http://en.wikibooks.org/wiki/Category%3AJAVA%20Programming)

¹ [HTTP://JAVALESSONS.COM/CGI-BIN/FUN/JAVA-TUTORIALS-MAIN.CGI?SES=A0789&CODE=STC&SUB=FUN](http://javalessons.com/cgi-bin/fun/java-tutorials-main.cgi?ses=a0789&code=stc&sub=fun)

² [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/Category%3AJAVA%20Programming)

26 Destroying Objects

Unlike in many other object-oriented programming languages, Java performs automatic garbage collection - any unreferenced objects are automatically erased from memory - and prohibits the user from manually destroying objects.

26.1 finalize()

When an object is garbage-collected, the programmer may want to manually perform cleanup, such as closing any open input/output streams. To accomplish this, the `finalize()` method is used. Note that `finalize()` should never be manually called, except to call a super class' `finalize` method from a derived class' `finalize` method. Also, we can not rely on when the `finalize()` method will be called. If the java application exits before the object is garbage-collected, the `finalize()` method may never be called.

```
protected void finalize() throws Throwable
{
    try {
        doCleanup();           // Perform some cleanup. If it fails for some
reason, it is ignored.
    } finally {
        super.finalize(); //Call finalize on the parent object
    }
}
```

The garbage-collector thread runs in a lower priority than the other threads. If the application creates objects faster than the garbage-collector can claim back memory, the program can run out of memory.

The `finalize` method is required only if there are resources beyond the direct control of the Java Virtual Machine that need to be cleaned up. In particular, there is no need to explicitly close an `OutputStream`, since the `OutputStream` will close itself when it gets finalized. Instead, the `finalize` method is used to release either native or remote resources controlled by the class.

CATEGORY:JAVA PROGRAMMING¹

¹ [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/Category%3AJAVA%20Programming)

27 Overloading Methods and Constructors

If two methods of a class (whether both declared in the same class, or both inherited by a class, or one declared and one inherited) have the same name but different signatures, then the method name is said to be overloaded. This fact causes no difficulty and never of itself results in a compile-time error. There is no required relationship between the return types or between the throws clauses of two methods with the same name but different signatures.

Methods are overridden on a signature-by-signature basis.

If, for example, a class declares two public methods with the same name, and a subclass overrides one of them, the subclass still inherits the other method. In this respect, the Java programming language differs from C++.

When a method is invoked, the number of actual arguments and the compile-time types of the arguments are used, at compile time, to determine the signature of the method that will be invoked. If the method that is to be invoked is an instance method, the actual method to be invoked will be determined at run time, using dynamic method lookup.

CATEGORY:JAVA PROGRAMMING¹

¹ [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/Category%3AJAVA%20Programming)

28 Arrays

28.1 Intro to Arrays

An **array** is similar to a table of data, keyed by number. In Java an array is an object like all other objects. Look at the following program:

```
import java.util.*;

public class ArrayExample {

    static Scanner input = new Scanner(System.in);

    public static void main(String[] args) {
        int numNames = getInt("Number of names?");
        String[] names = new String[numNames];
        for (int i = 0; i < names.length; i++) {
            names[i] = getString("Enter name #" + (i+1));
        }
        for (int i = 0; i < names.length; ++i) {
            System.out.println(names[i]);
        }
    }

    public static int getInt(String prompt) {
        System.out.print(prompt + " ");
        int integer = input.nextInt();
        input.nextLine(); // Get rid of this line
                          // so that getString won't read it
        return integer;
    }

    public static String getString(String prompt) {
        System.out.print(prompt + " ");
        return input.nextLine();
    }
}
```

Copy the code and compile it. The program will ask you to enter some names then reprints the names in order. It demonstrates three major aspects of arrays: how to define an array, how to set data, and how to access it. The code `String[] names = new String[numNames];` tells Java to create an array of size *numNames* that will store Strings. To set data, use `names[x] = data` where *x* is the index to access. Note that all Java arrays start at 0 and go to (array size - 1). Thus, if you dimension an array to size 10, the highest index is 9.

28.2 Array Fundamentals

- To create an array, use the syntax **DataType[] variable = new DataType[ArraySize]**. Alternatively, if you know the data beforehand, you can write **DataType[] variable = {item 1, item 2,...item n}**

- All elements of the array will be automatically initialized with the *default value* for that datatype. This is *false* for booleans, *0* for all numeric primitive types, and *null* for all reference types. So for example, the previous note created an array of `DataType` references, all of which are initialized to *null*.
- To access an item, use the syntax **variable[i]** where *i* is the index
- To set an item, use the syntax **variable[i] = data**
- To find the length of an array, use the syntax **variable.length**

28.3 Two-Dimensional Arrays

A two-dimensional array is represented by an array of arrays. Because an array is also an object, like any other object having the **Object** as the super class, it can be used to create an array where the element of the array is also an array. In this way an array with any number of dimensions can be created. Here are examples of two dimensional arrays with initializer blocks:

```
String[][] twoDimArray = {"a", "b", "c", "d", "e"},
                        {"f", "g", "h", "i", "j"},
                        {"k", "l", "m", "n", "o"};

int[][] twoDimIntArray = { { 0, 1, 2, 3, 4},
                          {10, 11, 12, 13, 14},
                          {20, 21, 22, 23, 24}};
```

Note that the above "twoDimArray" is equivalent to the following more verbose code:

```
String[][] twoDimArray = new String[3][];
for(int i = 0; i < twoDimArray.length; i++) {
    twoDimArray[i] = new String[5];
    for(int j = 0; j < twoDimArray[i].length; j++)
        twoDimArray[i][j] = "" + i + j;
}
```

In the above example we defined an array which has three elements, each element contains an array having 5 elements. We could create the array having the 5 elements first and use that one in the initialize block.

```
String[] oneDimArray = { "00", "01", "02", "03", "04" };
String[][] twoDimArray = { oneDimArray ,
                          {"10", "11", "12", "13", "14"},
                          {"20", "21", "22", "23", "24" } };
```

Since they are arrays of array references, these multi-dimensional arrays can be "jagged" (i.e. subarrays can have different lengths), or the subarray reference can even be null. Consider:

```
String[][] weirdTwoDimArray = {"10", "11", "12"},
                              null,
                              {"20", "21", "22", "23", "24"};
```

Note that the length of a two-dimensional array is the number of one-dimensional arrays it contains. In the above example, `weirdTwoDimArray.length` is 3, whereas `weirdTwoDimArray[2].length` is 5.

28.4 Multidimensional Array

Going further any number of dimensional array can be defined.

```
<elementType>[][]...[] <arrayName>  
or  
<elementType><arrayName>[][]...[]
```

DE:JAVA_STANDARD: DATENSTRUKTUREN¹ ES:PROGRAMACIÓN EN JAVA/ARRAYS²
FR:PROGRAMMATION JAVA/TABLEAUX³ NL:PROGRAMMEREN IN JAVA/ARRAYS⁴ PT:JAVA/VETORES⁵

1 [HTTP://DE.WIKIBOOKS.ORG/WIKI/JAVA_STANDARD%3A%20DATENSTRUKTUREN](http://de.wikibooks.org/wiki/JAVA_STANDARD%3A%20DATENSTRUKTUREN)
2 [HTTP://ES.WIKIBOOKS.ORG/WIKI/PROGRAMACION%20EN%20JAVA%2FARRAYS](http://es.wikibooks.org/wiki/PROGRAMACION%20EN%20JAVA%2FARRAYS)
3 [HTTP://FR.WIKIBOOKS.ORG/WIKI/PROGRAMMATION%20JAVA%2FTABLEAUX](http://fr.wikibooks.org/wiki/PROGRAMMATION%20JAVA%2FTABLEAUX)
4 [HTTP://NL.WIKIBOOKS.ORG/WIKI/PROGRAMMEREN%20IN%20JAVA%2FARRAYS](http://nl.wikibooks.org/wiki/PROGRAMMEREN%20IN%20JAVA%2FARRAYS)
5 [HTTP://PT.WIKIBOOKS.ORG/WIKI/JAVA%2FVETORES](http://pt.wikibooks.org/wiki/JAVA%2FVETORES)

29 Collection Classes

Collections are a group of objects bound together by a common characteristic. Java has various built-in classes to support the collection of objects, either of the same type or general. The most basic construct to work with is the *array* of which you will come to know of in a SECTION¹ later in this chapter.

29.1 Introduction to Collections

The most basic collection interface is called `Collection`. This interface gives the user a generic usage of a collection.

```
import java.util.Collection; // Interface
import java.util.ArrayList; // Implementation
import java.util.HashSet;   // Implementation
...
Collection coll1 = new ArrayList();
Collection coll2 = new HashSet();
...
< Use coll1 & coll2 >
```

In the above there are two collections. The usage of the collections are the same, the implementations are different. If the existing collection implementations do not meet your needs, you can write your version of the implementation. Your version of the implementation just needs to implement the same `java.util.Collection` interface, then you can switch to using your implementation and the code that is using the collection does not need to be changed.

```
import java.util.Collection;
import com.yourcomp.util.YourCollectionImpl;
...
Collection coll1 = new YourCollectionImpl();
Collection coll2 = new YourCollectionImpl();
...
< Use coll1 & coll2 >
```

The Java JDK collection implementations are quite powerful and good, so it is unlikely that you will need to write your own.

All collections contain object references. Because of that, if the object is changed after it was put in the collection, the object that is 'in' the collection also 'changes'. The object is not really in the collection, only the object reference is. It is not guaranteed that the objects 'inside' the collections won't change. This is an issue only if you put an actively used object in the collection.

¹ Chapter 28 on page 143

In that case when you are adding an object that could change any time you need to make a copy or clone of the object. A new object will be created and its reference will be put in the collection. In that case there will be no object references outside of the collection, so the objects 'inside' the collection can only be changed if we take out an object reference from the collection.

Aside from the `java.util.Collection` interface, the Java JDK has the `java.util.Map` interface as well. This defines key value mappings. Implementations of the Map interface do not contain collections of objects. Instead they contain collections of key->value mappings.

```
import java.util.Map;
import java.util.Hashtable;
...
Map map = new Hashtable();
...
map.put( key, value );
```

All collections need to have the same basic operations. Those are:

- Adding element(s) to the collection
- Removing element(s) from the collection
- Obtaining the number of elements in the collection
- Listing the contents of the collection, (Iterating through the collection)

Before selecting a particular collection implementation, ask the following question:

- Can my collection contain the same elements, i.e. are duplicates allowed?
- Can my collection contain the **null** element?
- Should the collection maintain the order of the elements? Is the order important in any way?
- How do you want to access an element? By index, key or just with an iterator?
- Does the collection need to be synchronized?
- From a performance perspective, which one needs to be faster, updates or reads?
- From a usage perspective, which operation will be more frequent, updates or reads?

Once you know your needs, you can select an existing implementation. But first decide if you need a 'Collection', or a 'Map'.

29.2 Generics

Since JDK version 1.5 an enhancement to the type system of the Java language has been added. It is called *Generics*. Most often *Generics* will be used with the collection classes.

parameterized type <E>

All collection implementations since 1.5 now have one 'parameterized type <E>' added. The 'E' refers to an *Element* type. When a collection is created the actual 'Element type' will replace the E.

Objects put into a collection are upcasted to **Object** class. It means that you need to cast the object reference back when you get an element out from the collection. It also means that **you need to know** the type of the object when you taking it out. For this reason we usually add objects

of the same type to a collection. If a collection contains different types of objects, we will have difficulty finding out the type of the objects obtained from a collection at run time.

With the use of the 'parameterized type <E>', the 'Element-type' that can be put into the collection can be specified when the collection object is created.

```
Collection<Integer> ageList = new ArrayList<Integer>();
ageList.add( new Integer(46) ); // --- Integer can be added
ageList.add( "50" ); // --- Compilation error, ageList can have only Integers
inside
```

`ageList` is a collection that can contain only `Integer` objects as elements. No casting is required when we take out an element.

```
Integer age = ageList.get(0);
```

For more information about Generics, please go to [JAVA PROGRAMMING/GENERICS²](#).

29.3 Collection or Map

The Java JDK contains two high level Interfaces:

- `java.util.Collection`
- `java.util.Map`

Implementations for those interfaces are not interchangeable. Collections are used for collecting Java objects. Maps are used for mapping key/value pairs.

29.3.1 Collection

Use the `Collection` interface if you need to keep related (usually the same type of) objects together in a collection where you can:

- Search for a particular element
- List the elements
- Maintain and/or change the order of the elements by using the collection basic operations (Add, Remove, Update,..)
- Access the elements by an index number

The advantages of using the `Collection` interface are:

- Gives a generic usage, as we talked about above, it is easy to switch implementation
- It makes it easy to convert one type of collection to an other.

The `Collection` interface defines the following basic operations:

- `boolean add(E o);` -- (using Element type E)

- `boolean addAll(Collection c);`
- `boolean remove(Object o);`
- `boolean removeAll(Collection c);`
- `boolean retainAll(Collection c);`

The methods above return **true** if the collection has changed due to the operation. Note that in `addAll()` we can add any type of collection. This is the beauty of using the `Collection` interface. You can have a `LinkedList` and just call the `addAll(list)` method, passing in a list. You can pass in a `Vector`, an `ArrayList`, a `HashSet`, a `TreeSet`, a `YourImpOfCollection`, ... All those different type of collections will be **magically** converted to a `LinkedList`.

Lets have a closer look at this *magic*. The conversion is easy because the `Collection` interface defines a standard way of looping through the elements. The following code is a possible implementation of `addAll()` method of the `LinkedList`.

```
import java.util.Collection
import java.util.Iterator
...
public String addAll( Collection coll )
{
    int sizeBefore = _linkList.size();
    Iterator iter = coll.iterator();
    while( iter.hasNext() )
    {
        _linkList.add( iter.next() );
    }
    if ( sizeBefore > _linkList.size() )
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

The above code just iterates through the passed in collection and adds the elements to the link list. You do not have to do that, since that is already defined. What you might need to code for is to loop through a 'Customer' collection:

```
import java.util.Collection
import java.util.Iterator
import java.yourcompany.Customer
...
public String printCustomerNames( Collection customerColl )
{
    StringBuffer buf = new StringBuffer();

    Iterator iter = customerColl.iterator();
    while( iter.hasNext() )
    {
        Customer cust = (Customer) iter.next();
        buf.append( cust.getName() );
        buf.append( "\n" );
    }
    return buf.toString();
}
```

Notice two things:

- The above code will work for all type of collections.
- We have to know the type of objects inside the collection, because we call a method on it.

29.3.2 Map

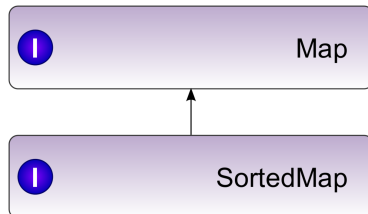


Figure 17: Figure 1:Map Interfaces

A map, sometimes also called an *Associated Array* or a *Dictionary*, can be thought of as an array where the index need not be an integer.

Use the Map interface if you need to keep related objects together in a Map where you can:

- Access an element by a key object
- Map one object to other

java.util.Map<K,V>

maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value. The Map interface provides three collection views, which allow a map's contents to be viewed as a set of keys, collection of values, or set of key-value mappings. The key is usually a non-mutable object. The value object however can be a mutable object.

java.util.SortedMap<K,V> : same as the Map interface, plus the keys in the Map are sorted.

29.4 Set or List or Queue

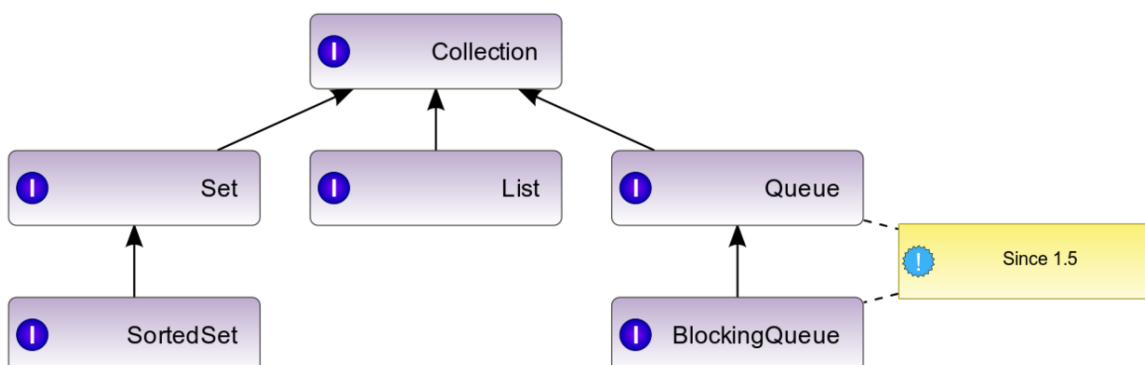


Figure 18: Figure 2:

There is no direct implementation for the `java.util.Collection` interface. The `Collection` interface has five sub interfaces. Those are:

`java.util.Set<E>`

contains unique elements, so duplicates not allowed. It is similar to a mathematical Set.

`java.util.List<E>` : elements are put in the list in a certain order, and can be accessed by an index. Duplicates are allowed, the same element can be added to a list.

`java.util.SortedSet<E>` : same as the `Set` interface; plus the elements in the `SortedSet` are sorted

`java.util.Queue<E>` : queues provide additional insertion, extraction, and inspection operations. There are FIFO (first in, first out) and LIFO (last in, first out) queues.

`java.util.BlockingQueue<E>` : waits for the queue to become non-empty when retrieving an element, and waits for space to become available in the queue when storing an element. Best used for producer-consumer queues.

29.4.1 Set

The basic implementation of the `Set` interface is the `HashSet`.

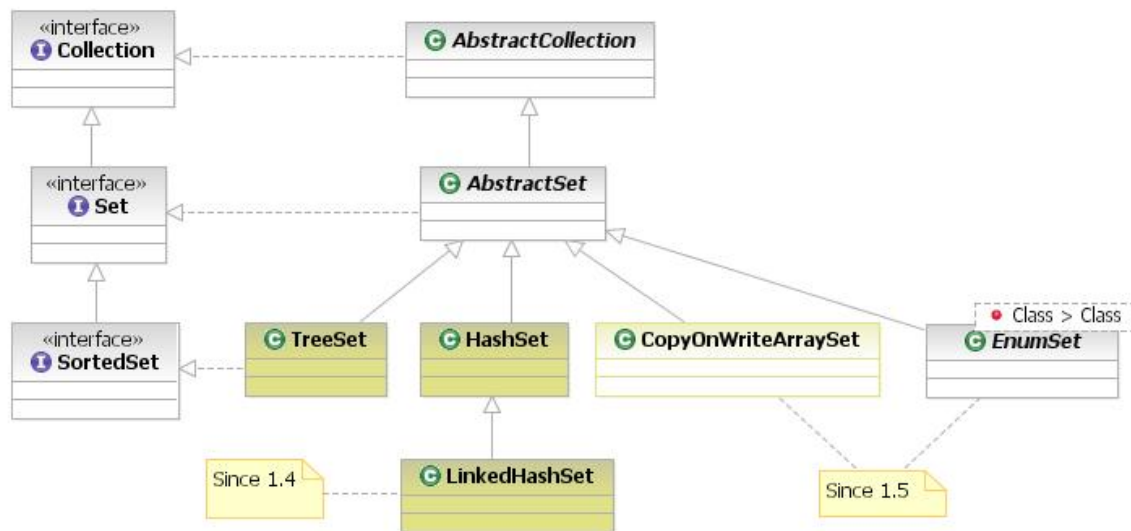


Figure 19

`java.util.TreeSet<E>`

Elements are sorted, not synchronized. **null** not allowed

`java.util.HashSet<E>` : Not synchronized. Allows the **null** elements

`java.util.CopyOnWriteArraySet<E>` : Thread safe, a fresh copy is created during modification operation. Add, update, delete are expensive.

`java.util.EnumSet<E extends Enum<E>>` : All of the elements in an enum set must come from a single enum type that is specified, explicitly or implicitly, when the set is created. Enum sets are represented internally as bit vectors.

`java.util.LinkedHashSet<E>` : Same as `HashSet`, plus defines the iteration ordering, which is the order in which elements were inserted into the set.

Detecting duplicate objects in Sets

`Set` cannot have duplicates in it. You may wonder how duplicates are detected when we are adding an object to the `Set`. We have to see if that object exists in the `Set` or not. It is not enough to check the object references, the objects' values have to be checked as well.

To do that, fortunately, each java object has the `boolean equal(Object obj)`³, method available inherited from **Object**. You need to override it. That method will be called by the `Set` implementation to compare the two objects to see if they are equal or not.

There is a problem, though. What if I put two different type of objects to the `Set`. I put an Apple and an Orange. They can not be compared. Calling the `equal()`⁴ method would cause a `ClassCastException`. There are two solutions to this:

- **Solution one** : Override the `int hashCode()`⁵ method and return the same values for the same type of objects and return different values for different type of objects. The `equal()`⁶ method is used to compare objects only with the same value of `hashCode`. So before an object is added, the `Set` implementation needs to:
 - find all the objects in the `Set` that has the same `hashCode` as the candidate object `hashCode`
 - and for those, call the `equal()`⁷ methods passing in the candidate object
 - if any of them returns `true`, the object is not added to the `Set`.
- **Solution two** : Create a super class for the Apple and Orange, let's call it Fruit class. Put Fruits in the `Set`. You need to do the following:
 - Do not override the `equals()` and `hashCode()` methods in the Apple and Orange classes
 - Create `appleEquals()` method in the Apple class, and create `orangeEquals()` method in the Orange class
 - Override the `hashCode()` method in the Fruit class and return the same value, so the `equals()` is called by the `Set` implementation
 - Override the `equals()` method in the Fruit class for something like this.

```
public boolean equals( Object obj )
{
```

3 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FAPI%2FJAVA.LANG.OBJECT%23EQUALS%28%29%20METHOD](http://en.wikibooks.org/wiki/Java%20Programming%2FAPI%2FJava.Lang.Object%23Equals%28%29%20Method)

4 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FAPI%2FJAVA.LANG.OBJECT%23](http://en.wikibooks.org/wiki/Java%20Programming%2FAPI%2FJava.Lang.Object%23)

5 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FAPI%2FJAVA.LANG.OBJECT%23HASHCODE%28%29%20METHOD](http://en.wikibooks.org/wiki/Java%20Programming%2FAPI%2FJava.Lang.Object%23HashCode%28%29%20Method)

6 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FAPI%2FJAVA.LANG.OBJECT%23](http://en.wikibooks.org/wiki/Java%20Programming%2FAPI%2FJava.Lang.Object%23)

7 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FAPI%2FJAVA.LANG.OBJECT%23](http://en.wikibooks.org/wiki/Java%20Programming%2FAPI%2FJava.Lang.Object%23)

```
boolean ret = false;
if ( this instanceof Apple &&
    obj instanceof Apple )
{
    ret = this.appleEquals(obj);
}
else if ( this instanceof Orange &&
         obj instanceof Orange )
{
    ret = this.orangeEquals(obj);
}
else
{
    // --- Can not compare Orange to Apple ---
    ret = false;
}
return ret;
}
}
```

Note:

- only the objects that have the same hashCode will be compared.
- you are responsible to override the `equal()` and `hashCode()` methods. The default implementations in `Object` won't work.
- Only override the `hashCode()`⁸ method if you want to eliminate value duplicates.
- Do not override the `hashCode()`⁹ method if you know that the values of your objects are different, or if you only want to prevent adding the exactly same object.
- Beware that the `hashCode()`¹⁰ may be used in other collection implementations, like in a `Hashtable` to find an object fast. Overriding the default `hashCode()` method may affect performance there.
- the default hashCodes are unique for each object created, so if you decide not to override the `hashCode()` method, there is no point overriding the `equal()` method, as it won't be called.

SortedSet

The `SortedSet` interface extends the `Set` interface. All elements in the `SortedSet` must implement the `Comparable` interface, furthermore all elements must be mutually comparable.

Note that the ordering maintained by a sorted set must be consistent with `equals` if the sorted set is to correctly implement the `Set` interface. This is so because the `Set` interface is defined in terms of the `equals` operation, but a sorted set performs all element comparisons using its `compare` method, so two elements that are deemed equal by this method are, from the standpoint of the sorted set, equal.

8 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FAPI%2FJAVA.LANG.OBJECT%23HASHCODE%28%29%20METHOD](http://en.wikibooks.org/wiki/Java%20Programming%2FAPI%2FJava.Lang.Object%23hashCode%28%29%20Method)

9 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FAPI%2FJAVA.LANG.OBJECT%23HASHCODE%28%29%20METHOD](http://en.wikibooks.org/wiki/Java%20Programming%2FAPI%2FJava.Lang.Object%23hashCode%28%29%20Method)

10 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FAPI%2FJAVA.LANG.OBJECT%23HASHCODE%28%29%20METHOD](http://en.wikibooks.org/wiki/Java%20Programming%2FAPI%2FJava.Lang.Object%23hashCode%28%29%20Method)

The SortedSet interface has additional methods due to the sorted nature of the 'Set'. Those are:

- `E first()`; -- returns the first element
- `E last()`; -- returns the last element
- `SortedSet headSet(E toElement)`; -- returns from the first, to the exclusive toElement
- `SortedSet tailSet(E fromElement)`; -- returns from the inclusive fromElement to the end
- `SortedSet subSet(E fromElement, E toElement)`; -- returns elements range from fromElement, inclusive, to toElement, exclusive. (If fromElement and toElement are equal, the returned sorted set is empty.)

29.4.2 List

The List has the following implementations:

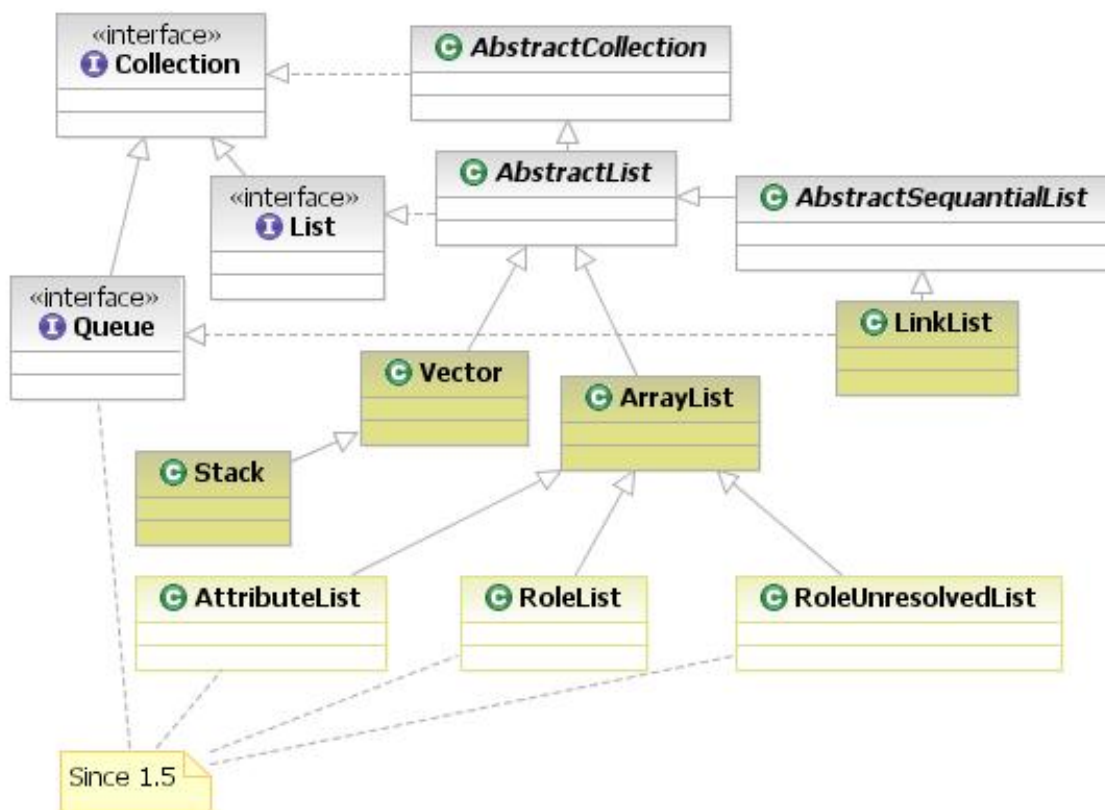


Figure 20

`java.util.Vector<E>`

Synchronized, use in multiple thread access, otherwise use ArrayList

`java.util.Stack<E>` : It extends class Vector with five operations that allow a vector to be treated as a stack. It represents a last-in-first-out (LIFO) stack of objects.

`java.util.ArrayList<E>` : Non-synchronized, use in single thread environment, otherwise use Vector

`java.util.LinkedList<E>` : Non-synchronized, update operation is faster than other lists, easy to use for stacks, queues, double-ended queues.

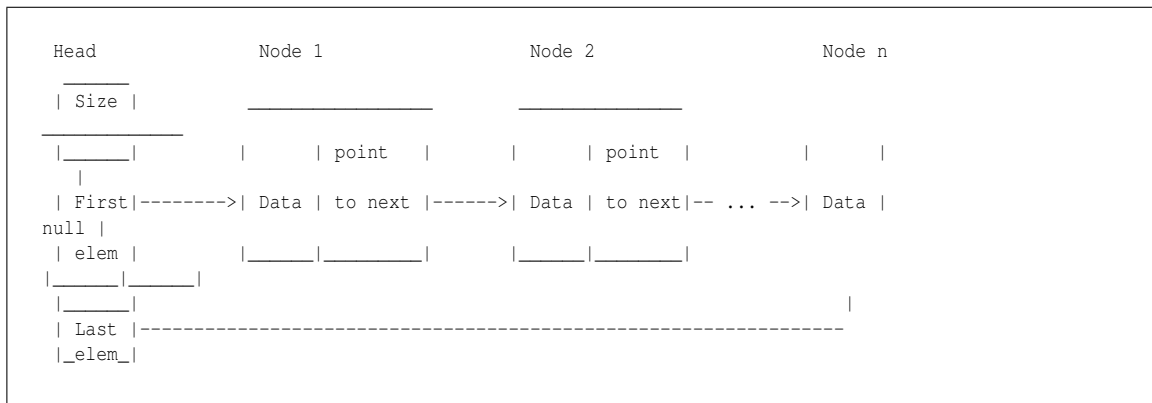
`javax.management.AttributeList<E>` : Represents a list of values for attributes of an MBean. The methods used for the insertion of Attribute objects in the AttributeList overrides the corresponding methods in the superclass ArrayList. This is needed in order to insure that the objects contained in the AttributeList are only Attribute objects.

`javax.management.relation.RoleList<E>` : A RoleList represents a list of roles (Role objects). It is used as parameter when creating a relation, and when trying to set several roles in a relation (via 'setRoles()' method). It is returned as part of a RoleResult, to provide roles successfully retrieved.

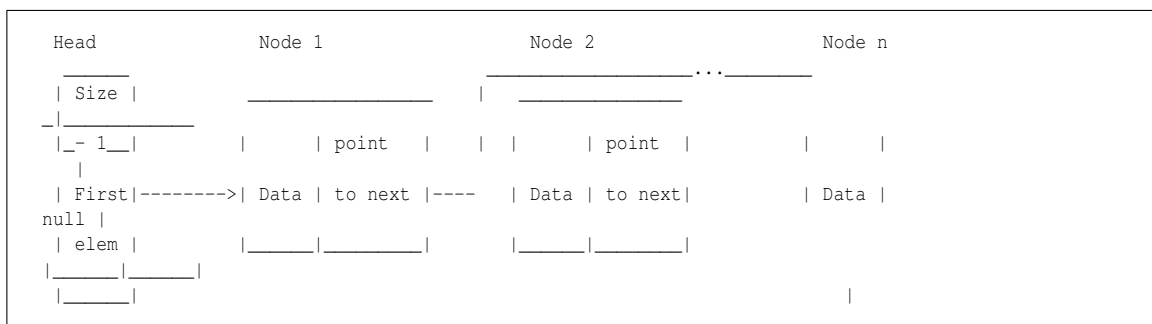
`javax.management.relation.RoleUnresolvedList<E>` : A RoleUnresolvedList represents a list of RoleUnresolved objects, representing roles not retrieved from a relation due to a problem encountered when trying to access (read or write to roles).

The basic implementation of the List interface is the ArrayList. The ArrayList is not synchronized, not thread safe. Vector is synchronized, and thread safe. Vector is slower, because of the extra overhead to make it thread safe. When only one thread is accessing the list, use the ArrayList. Whenever you insert or remove an element from the list, there are extra overhead to reindex the list. When you have a large list, and you have lots of insert and remove, consider using the LinkedList.

The name LinkedList implies a special data structure where the elements/nodes are connected by pointers. To remove an element from the link list the pointers need to be rearranged.



After removing Node 2:



```
| Last |-----|
|_elem_|
```

29.4.3 Queue

The Queue interface adds the following operations to the Collection interface:

E element()	Retrieves, but does not remove, the head of this queue. This method differs from the peek method only in that it throws an exception if this queue is empty
boolean offer(E o)	Inserts the specified element into this queue, if possible.
E peek()	Retrieves, but does not remove, the head of this queue, returning null if this queue is empty
E poll()	Retrieves and removes the head of this queue, or null if this queue is empty
E remove()	Retrieves and removes the head of this queue. This method differs from the poll method in that it throws an exception if this queue is empty.

- **java.util.PriorityQueue<E>**
orders elements according to an order/priority specified at construction time, null element is not allowed.
- **java.util.concurrent.ArrayBlockingQueue<E>**
: orders elements FIFO;
synchronized, thread safe.
- **java.util.concurrent.SynchronousQueue<E>**
: each put must wait for a take, and vice versa, does not have any internal capacity, not even a capacity of one, an element is only present when you try to take it; you cannot add an element (using any method) unless another thread is trying to remove it

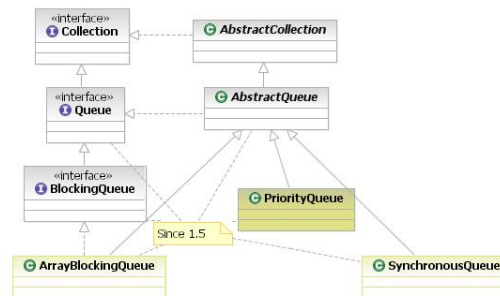


Figure 21

29.5 Map Classes

The Map interface has the following implementations:

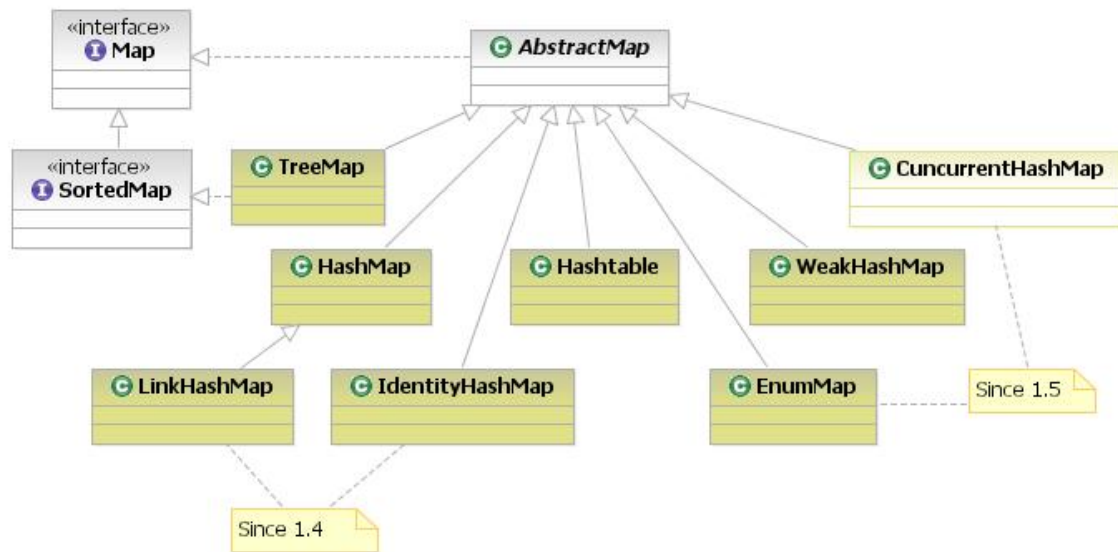


Figure 22

`java.util.TreeMap<E>`

guarantees that the map will be in ascending key order, sorted according to the natural order for the key's class, not-synchronized.

`java.util.HashMap<E>` : is roughly equivalent to `Hashtable`, except that it is unsynchronized and permits nulls

`java.util.concurrent.ConcurrentHashMap` : same `Hashtable`, plus retrieval operations (including `get`) generally do not block, so may overlap with update operations (including `put` and `remove`).

`java.util.Hashtable<E>` : Synchronized, null can not be used as key

`java.util.WeakHashMap<E>` : entry in a `WeakHashMap` will automatically be removed when its key is no longer in ordinary use. Non-synchronized.

`java.util.LinkedHashMap<E>` : This linked list defines the iteration ordering, which is normally the order in which keys were inserted into the map (insertion-order). Note that insertion order is not affected if a key is re-inserted into the map.

`java.util.IdentityHashMap` : This class implements the `Map` interface with a hash table, using reference-equality in place of object-equality when comparing keys (and values). In other words, in an `IdentityHashMap`, two keys `k1` and `k2` are considered equal if and only if `(k1==k2)`. (In normal `Map` implementations (like `HashMap`) two keys `k1` and `k2` are considered equal if and only if `(k1==null ? k2==null : k1.equals(k2))`.) Not-synchronized.

`java.util.EnumMap` : All of the keys in an enum map must come from a single enum type that is specified, explicitly or implicitly, when the map is created. Enum maps are represented internally as arrays. This representation is extremely compact and efficient. Not-synchronized.

29.6 Thread Safe Collections

It is also called Concurrent Collections. Most of the popular collection classes have implementations for both single thread and multiple thread environments. The non-synchronized implementations are always faster. You can use the non-synchronized implementations in multiple thread environments, when you make sure that only one thread updating the collection at any given time.

A new Java JDK package was introduced at Java 1.5, that is `java.util.concurrent`. This package supplies a few Collection implementations designed for use in multi-threaded environments.

The following table list all the synchronized collection classes:

	synchronized	non-synchronized
List	<code>java.util.Vector</code>	<code>java.util.ArrayList</code>
	<code>java.util.Stack</code>	<code>java.util.LinkedList</code>
	<code>java.util.concurrent.CopyOnWriteArrayList</code>	<code>java.util.TreeSet</code>
Set		<code>java.util.HashSet</code>
		<code>java.util.LinkHashSet</code>
	<code>java.util.concurrent.CopyOnWriteArraySet</code>	<code>java.util.TreeMap</code>
Map	<code>java.util.Hashtable</code>	<code>java.util.HashMap</code>
	<code>java.util.concurrent.ConcurrentHashMap</code>	<code>java.util.LinkedHashMap</code>
		<code>java.util.IdentityHashMap</code>
		<code>java.util.EnumMap</code>

29.7 Classes Diagram (UML)

The following UML class diagram shows the **Collection** interfaces and their implementations.

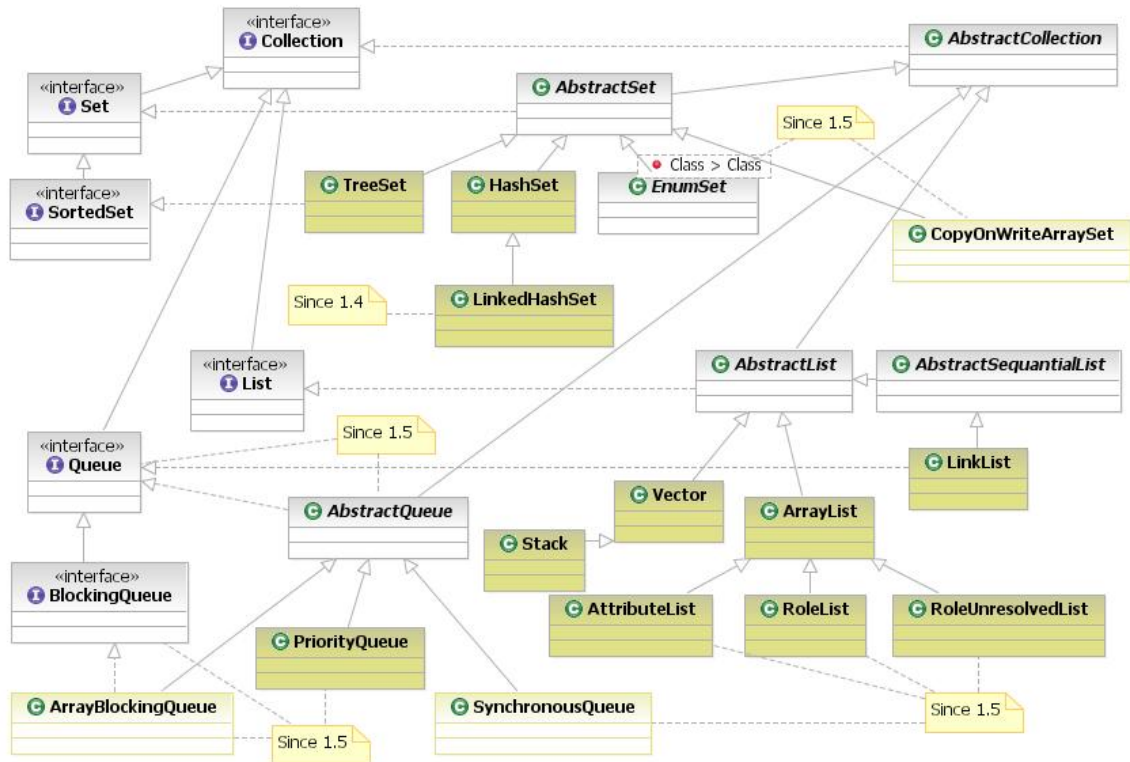


Figure 23

The following UML class diagram shows the **Map** interfaces and their implementations.

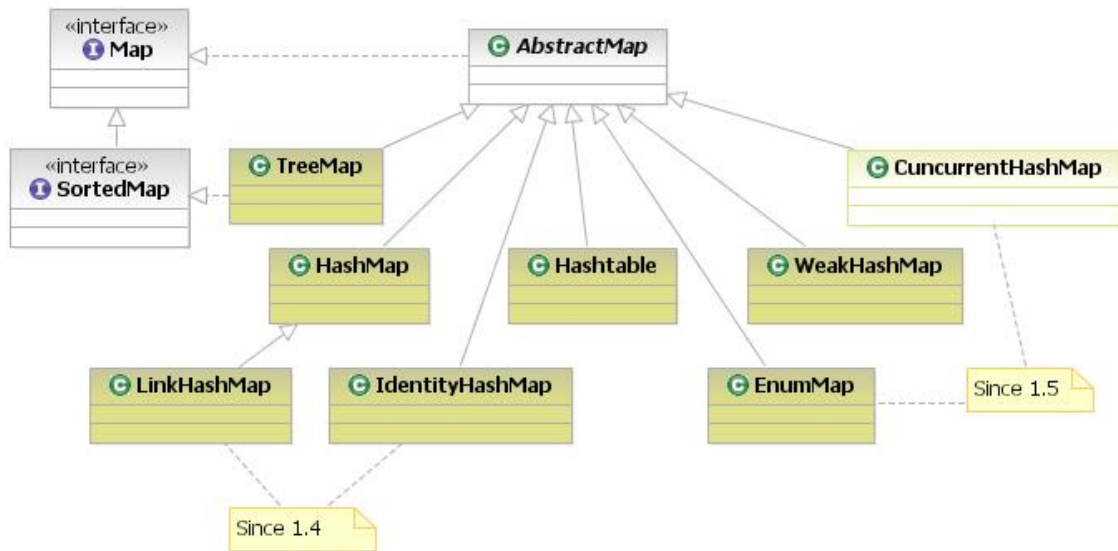


Figure 24

29.8 External links

- [JAVA TUTORIAL ON COLLECTIONS](#)¹¹

[CATEGORY:JAVA PROGRAMMING](#)¹²

¹¹ [HTTP://JAVA.SUN.COM/DOCS/BOOKS/TUTORIAL/COLLECTIONS/INTERFACES/COLLECTION.HTML](http://java.sun.com/docs/books/tutorial/collections/interfaces/collection.html)

¹² [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/Category%3AJAVA%20Programming)

30 Throwing and Catching Exceptions

Language compilers are adept at pointing out most of the erroneous code in a program, however there are some errors that only become apparent when the program is executed. Consider the code in Listing 1.1; here, the program defines a method **divide** that does a simple division operation taking two integers as parameter arguments and returning the result of their division. It can safely be assumed that when the **divide(4, 2)** statement is called, it would return the number **2**. However, consider the next statement, where the program relies upon the provided command line arguments to generate a division operation. What if the user provides the number zero (**0**) as the second argument. We all know that division by zero is impossible, but the compiler couldn't possibly have anticipated the user providing zero as an argument.

Listing 1.1: A simple division operation.

```
///  
public class ExceptionTutorial01  
{  
    public static int divide(int a, int b)  
    {  
        return a / b;  
    }  
    public static void main(String[] args)  
    {  
        System.out.println( divide(4, 2) );  
        if(args.length > 1)  
        {  
            int arg0 = Integer.parseInt(args[0]);  
            int arg1 = Integer.parseInt(args[1]);  
            System.out.println( divide(arg0, arg1) );  
        }  
    }  
}  
} /* Output for: java ExceptionTutorial01 1 0  
2  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at ExceptionTutorial01.divide(ExceptionTutorial01.java:6)  
    at ExceptionTutorial01.main(ExceptionTutorial01.java:15)  
*///:~
```

Such *exceptional code* that results in erroneous interpretations at program runtime usually results in errors that are called *exceptions* in Java. When the Java interpreter encounters an exceptional code, it halts execution and displays information about the error that occurs. This information is known as a **stack trace**. The **stack trace** in the above example tells us more about the error, such as the thread - "main" - where the exception occurred, the type of exception - `java.lang.ArithmeticException`, a comprehensible display message - `/ by zero`, and the exact methods and the line numbers where the exception may have occurred.

30.1 Exception arguments

In the code above, the exception object for `java.lang.ArithmeticException` was generated by the Java interpreter itself. However, there are times when you would need to explicitly create your own exceptions. As with any object in Java, you always create exceptions on the heap using **new**, which allocates storage and calls a constructor. There are two constructors in all standard exceptions:

1. The default constructor; and,
2. A constructor taking a string argument so that you can place pertinent information in the exception.

Listing 1.2: Instance of an exception object with the default constructor.

```
new Exception();
```

Listing 1.3: Instance of an `Exception` object by passing string in constructor.

```
new Exception("Something unexpected happened");
```

This string can later be extracted using various methods, as you'll see.

The keyword **throw** produces a number of interesting results. After creating an exception object with **new**, you give the resulting reference to **throw**. The object is, in effect, "returned" from the method, even though that object type isn't normally what the method is designed to return. A simplistic way to think about exception handling is as a different kind of return mechanism, although you get into trouble if you take that analogy too far. You can also exit from ordinary scopes by throwing an exception. In either case, an exception object is returned, and the method or scope exits.

Listing 1.4: Throwing an exception results in an unexpected return from the method.

```
throw new Exception();
```

Anything after the **throw** statement would not be executed, unless the *thrown* exception is HANDLED¹ properly. Any similarity to an ordinary return from a method ends here, because *where* you return is some place completely different from where you return for a normal method call, i.e., you end up in an appropriate exception handler that might be far away - many levels on the call stack - from where the exception was thrown.

In addition, you can throw any type of **Throwable**, which is the exception root class. Typically, you'll throw a different class of exception for each different type of error. The information about the error is represented both inside the exception object and implicitly in the name of the exception class, so someone in the bigger context can figure out what to do with your exception. Often, the only information is the type of exception, and nothing meaningful is stored within the exception object.

¹ [HTTP://EN.WIKIBOOKS.ORG/WIKI/%23EXCEPTION%20HANDLERS](http://en.wikibooks.org/wiki/%23Exception%20handlers)

30.2 Catching an exception

To see how an exception is caught, you must first understand the concept of a *guarded region*. This is a section of code that might produce exceptions and is followed by the code to handle those exceptions.

30.2.1 The `try` block

If you are inside a method and you throw an exception (or another method that you call within this method throws an exception), that method will exit in the process of throwing. If you don't want a **throw** to exit the method, you can set up a special block within that method to capture the exception. This is called the *try block* because you "try" your various method calls there. The **try** block is an ordinary scope preceded by the keyword **try**.

Listing 1.5: A basic try block.

```
try
{
    // Code that might generate exceptions
}
```

If you were checking for errors carefully in a programming language that didn't support exception handling, you'd have to surround every method call with setup and error-testing code, even if you call the same method several times. With exception handling, you put everything in a **try** block and capture all the exceptions in one place. This means your code is much easier to write and read because the goal of the code is not confused with the error checking.

30.3 Exception handlers

Of course, the thrown exception must end up some place. This "place" is the *exception handler*, and there's one for every exception type you want to catch. Exception handlers immediately follow the **try** block and are denoted by the keyword **catch**:

Listing 1.6: Exception handling with catch blocks.

```
try
{
    // Suppose the code here throws two exceptions:
    // NullPointerException and NumberFormatException,
    // then each is handled in a separate catch block.
}
catch(NullPointerException ex)
{
    // Exception handling code for the NullPointerException
}
catch(NumberFormatException ex)
{
    // Exception handling code for the NumberFormatException
}
// etc...
```

Using the syntax in Listing 1.6, we can write the potentially problematic division code in Listing 1.1 as under.

Listing 1.7: Catching 'division by zero' errors.

```
int result = 0;
try
{
    result = a / b;
}
catch (ArithmeticException ex)
{
    result = 0;
}
return result;
```

Using the code in Listing 1.7, the program would now not abruptly terminate but continue its execution whilst returning zero for a division by zero operation. I know it's not the correct answer, but hey, it saved your program from terminating abnormally.

30.4 Exception classes in the JCL

The box 1.1 below talks about the various exception classes within the `java.lang` package of the JCL.

Box 1.1: The Java exception classes

- **Throwable**

The Throwable class is the superclass of all errors and exceptions in the Java language. Only objects that are instances of this class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java throw statement.

- A throwable contains a snapshot of the execution stack of its thread at the time it was created. It can also contain a message string that gives more information about the error. Finally, it can contain a cause: another throwable that caused this throwable to get thrown. The cause facility is new in release 1.4. It is also known as the chained exception facility, as the cause can, itself, have a cause, and so on, leading to a "chain" of exceptions, each caused by another

- **Error**

An Error indicates serious problems that a reasonable application should not try to catch. Most such errors are abnormal conditions. **Exception** : The class Exception and its subclasses are a form of Throwable that indicates conditions that a reasonable application might want to catch. Also this is the class that a programmer may want to extend when adding business logic exceptions. **RuntimeException** : RuntimeException is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine. A method is not required to declare in its throws clause any subclasses of RuntimeException that might be thrown during the execution of the method but not caught.

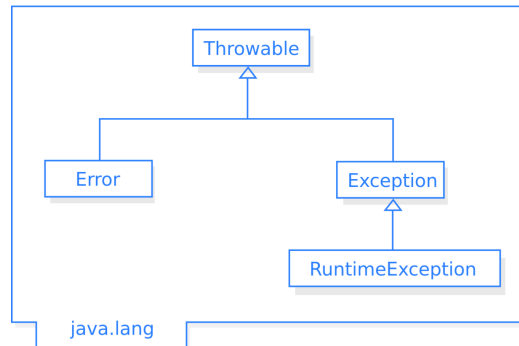


Figure 25

Figure 1.1: The exception classes and their inheritance model in the JCL.

30.5 Catch clauses

There are three distinct types of catch clauses used by Java programmers. We have already explored the first rule in the code listings above. Given below are all three clauses; we will explain all three in greater detail in the next sections.

1. The *catch-list* clause
2. The *catch-any* clause
3. The *multi-catch* clause

30.5.1 The *catch-list* clause

Using the **catch-list** clause, you provide one handle for every one type of exception. So, for instance if a *guarded block* generates a **NullPointerException** and an **ArithmeticException**, you have to provide two exception handling **catch blocks** - one for each exception. Refer to code listing 1.6 for an example.

30.5.2 The *catch-any* clause

There are times when too many blocks are just verbose, especially when all exceptions are handled in a similar manner. What you need here is a **catch-any** clause, where all exceptions are handled in a single **catch block**. Because all exceptions in Java are the sub-class of `java.lang.Exception` class, you can have a single **catch block** that catches an exception of type **Exception** only. Hence the compiler is fooled into thinking that this block can handle any exception. Using this, the code in Listing 1.6 can be rewritten as follows:

Listing 1.8: The *catch-any* clause.

```
try
{
    // ...
}
catch(Exception ex)
{
    // Exception handling code for ANY exception
}
```

Note:

You can also use the `java.lang.Throwable` class here, since **Throwable** is the parent class for the *application-specific* **Exception** classes. However, this is discouraged in Java programming circles. This is because **Throwable** happens to also be the parent class for the *non-application specific* **Error** classes which are not meant to be handled explicitly as they are catered for by the JVM itself.

30.5.3 The *multi-catch* clause

As of JDK 7, Java added another convenient feature in their exception handling routines - the **multi-catch** clause. This is a combination of the previous two **catch** clauses and let's you handle

exceptions in a single handler while also maintaining their types. So, instead of being boxed into a parent Exception super-class, they retain their individual types.

Listing 1.9: The multi-catch clause.

```
try
{
    // ...
}
catch (NullPointerException | NumberFormatException ex)
{
    // Exception handling code specifically for the NullPointerException
    // and the NumberFormatException
}
```

30.6 Example of handling exceptions

Let's examine the following code:

```
public void methodA() throws SomeException
{
    //methodbody
}
public void methodB() throws CustomException, AnotherException
{
    //Methodbody
}
public void methodC()
{
    methodB();
    methodA();
}
```

In the code sample, methodC is invalid. Because methodA and methodB pass (or throw) exceptions, methodC must be prepared to handle them. This can be handled in two ways: a **try - catch** block, which will handle the exception within the method and a **throws** clause which would in turn throw the exception to the caller to handle. The above example will cause a compilation error, as Java is very strict about exception handling. So the programmer forced to handle any possible error condition at some point.

A method can do two things with an exception. Ask the calling method to handle it by the **throws** declaration. Or handle the exception inside the method by the **try-catch** block.

To construct a throws declaration, add `throws ExceptionName` (additional exceptions can be added with commas). To construct a **try - catch** block, use the following syntax

```
try
{
    // Guarded region that will catch any exception that
    // occurs within this code.
}
catch (Exception ex)
{
    // Caught exceptions are handled here
}
finally
{
    // This class is optional. Code here is executed
```

```
// regardless of exceptions being thrown
}
```

The original code can be modified to work correctly in multiple ways. For example, the following:

```
public void methodC() throws CustomException, SomeException
{
    try
    {
        methodB();
    }
    catch(AnotherException e)
    {
        // Handle caught exceptions.
    }
    methodA();
}
```

The AnotherException from methodB will be handled locally, while CustomException and SomeException will be thrown to the caller to handle it.

30.7 Application Exceptions

Application Exception classes should extend the `java.lang.Exception` class. Some of the JDK classes also throw exception objects inherited from `java.lang.Exception`. If any of those Exception object is thrown, **it must be caught by the application** some point, by a catch-block. The compiler will enforce that there is a catch-block associated with an exception thrown, if the thrown exception object is inherited from `java.lang.Exception` and it is not the `java.lang.RuntimeException` or its inherited objects. However, `java.lang.RuntimeException` or its inherited objects, can be caught by the application, but that is not enforced by the compiler.

Lets see what is the catching criteria for a catch block to catch the "thrown" exception.

A catch-block will "catch" a thrown exception if and only if:

- the thrown exception object is the same as the exception object specified by the catch-block
- the thrown exception object is the subtype of the exception object specified by the catch-block

```
try
{
    throw new Exception("This will be caught below");
}
catch(Exception ex)
{
    // The "thrown" object is the same what is specified at the catch-block
}
try
{
    throw new NullPointerException("This will be caught below");
}
catch(Exception e)
{
    // NullPointerException is subclass of the Exception class.
}
```

There can be more than one catch-block for a try-block. The catching blocks evaluated sequentially one by one. If a catch-block catch the exception, the others will not be evaluated.

Example:

```
try
{
    throw new NullPointerException("This will be caught below");
}
catch(Exception e)
{
    // The NullPointerException thrown in the code above is caught here...
}
catch(NullPointerException e)
{
    // ..while this code is never executed and the compiler returns an error
}
```

Because `NullPointerException` is subclass of the `Exception` class. All `NullPointerException`s will be caught by the first catch-block.

Instead the above code should be rewritten as follows:

```
try
{
    throw new NullPointerException("This will be caught below");
}
catch(NullPointerException e)
{
    // The above NullPointerException will be caught here...
}
catch(Exception e)
{
    // ..while other exception are caught here.
}
```

30.8 Runtime Exceptions

The `java.lang.RuntimeException` exception class is inherited from `java.lang.Exception`. It is a special exception class, because catching this exception class or its subclasses are not enforced by the Java compiler.

runtime exception

Runtime exceptions are usually caused by data errors, like arithmetic overflow, divide by zero, Runtime exceptions are not business related exceptions. In a well debugged code, runtime exceptions should not occur. Runtime exceptions should only be used in the case that the exception could be thrown by and only by something hard-coded into the program. These should not be able to be triggered by the software's user(s).

30.8.1 NullPointerException

`NullPointerException` is a `RuntimeException`. In Java, a special **null** can be assigned to an object reference. `NullPointerException` is thrown when an application attempts to use an object reference, having the **null** value. These include:

- Calling an instance method on the object referred by a null reference.
- Accessing or modifying an instance field of the object referred by a null reference.

- If the reference type is an array type, taking the length of a null reference.
- If the reference type is an array type, accessing or modifying the slots of a null reference.
- If the reference type is a subtype of Throwable, throwing a null reference.

Applications should throw instances of this class to indicate other illegal uses of the null object.

```
Object obj = null;
obj.toString(); // This statement will throw a NullPointerException
```

The above code shows one of the pitfall of Java, and the most common source of bugs. No object is created and the compiler does not detect it. NullPointerException is one of the most common exceptions thrown in Java.

30.8.2 Why do we need null?

The reason we need it is because many times we need to create an object reference, before the object itself is created. Object references cannot exist without a value, so we assign the **null** value to it.

```
public Customer getCustomer()
{
    Customer customer = null;
    try
    {
        ...
        customer = createCustomer();
        ...
    }
    catch(Exception ex)
    {
        ...
    }
    return customer;
}
```

In the above code we want to create the Customer inside the try-block, but we also want to return the object reference to the caller, so we need to create the object reference outside of the try-block, because of the scoping rule in Java. This is one of the pitfall of Java.

30.9 Keyword references

- TRY²
- CATCH³
- THROWS⁴
- THROW⁵

CATEGORY:JAVA PROGRAMMING⁶

2 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FKEYWORDS%2FTRY](http://en.wikibooks.org/wiki/JAVA%20PROGRAMMING%2FKEYWORDS%2FTRY)
3 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FKEYWORDS%2FCATCH](http://en.wikibooks.org/wiki/JAVA%20PROGRAMMING%2FKEYWORDS%2FCATCH)
4 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FKEYWORDS%2FTHROWS](http://en.wikibooks.org/wiki/JAVA%20PROGRAMMING%2FKEYWORDS%2FTHROWS)
5 [HTTP://EN.WIKIBOOKS.ORG/WIKI/JAVA%20PROGRAMMING%2FKEYWORDS%2FTHROW](http://en.wikibooks.org/wiki/JAVA%20PROGRAMMING%2FKEYWORDS%2FTHROW)
6 [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/CATEGORY%3AJAVA%20PROGRAMMING)

This page describes some techniques for preventing `NullPointerException`.

It does not describe general techniques for how you should program Java. It is of some use, to make you more aware of null values, and to be more careful about generating them yourself.

Note that this list is not complete - there are no rules for preventing `NullPointerException` entirely in Java, because the standard libraries have to be used, and they can cause `NullPointerException`s. Also, it is possible to observe an uninitialised final field in Java, so you can't even treat a final field as being completely trusted during the object's creation.

A good approach is to learn how to deal with `NullPointerException`s first, and become competent with that. These suggestions will help you to cause less `NullPointerException`s, but they don't replace the need to know about `NullPointerException`s.

30.10 Minimize the use of the keyword 'null' in assignment statements

This means not doing things like:

```
String s=null;
while (something)
    if (something2)
        s="yep";

if (s!=null)
    something3(s);
```

You can replace this with:

```
boolean done=false;

while (!done && something)
    if (something2)
    {
        done=true;
        something3("yep");
    }
```

You might also consider replacing null with "" in the first example, but default values bring about bugs caused by default values being left in place. A `NullPointerException` is actually better, as it allows the runtime to tell you about the bug, rather than just continue with a default value.

30.11 Minimize the use of the new `Type[int]` syntax for creating arrays of objects

An array created using `new Object[10]` has 10 null pointers. That's 10 more than we want, so use collections instead, or explicitly fill the array at initialisation with:

```
Object[] objects={"blah",5,new File("/usr/bin")};
```

or:

```
Object[] objects;  
objects=new Object[]{"blah",5,new File("/usr/bin")};
```

30.12 Check all references obtained from 'untrusted' methods

Many methods that can return a reference can return a null reference. Make sure you check these. For example:

```
File file=new File("/etc");  
File[] files=file.listFiles();  
if (files!=null)  
{  
    stuff  
}
```

`File.listFiles()` can return null if `"/etc"` is not a directory.

You can decide to trust some methods not to return null, if you like, but that's an assumption you're making. Some methods that don't specify that they might return null, actually do, instead of throwing an exception.

30.13 Comparing string variable with a string literal

When you compare a variable with a string literal, always put the string literal first. For example do:

```
if ( "OK".equals( state ) )  
{  
    ...  
}
```

and do not do:

~~<strike>~~

```
if ( state.equals( "OK" ) )  
{  
    ...  
}
```

~~</strike>~~

If the 'state' variable is null, you get a `NullPointerException` in the second example, but not in the first one.

CATEGORY:JAVA PROGRAMMING⁷

30.14 See also

- W:JAVA PLATFORM⁸
- W:JAVA API⁹
- W:JAVA VIRTUAL MACHINE¹⁰
- GCC¹¹ (includes a Java to machine language compiler)
- COMPARISON OF JAVA TO C++¹².
- W:JINI¹³
- ECLIPSE¹⁴ IDE¹⁵ <http://eclipse.org/>
- W:NETBEANS¹⁶ (Another open source IDE)
- W:OPTIMIZATION OF JAVA¹⁷

⁷ [HTTP://EN.WIKIBOOKS.ORG/WIKI/CATEGORY%3AJAVA%20PROGRAMMING](http://en.wikibooks.org/wiki/Category%3AJAVA%20PROGRAMMING)

⁸ [HTTP://EN.WIKIPEDIA.ORG/WIKI/JAVA%20PLATFORM](http://en.wikipedia.org/wiki/JAVA%20PLATFORM)

⁹ [HTTP://EN.WIKIPEDIA.ORG/WIKI/JAVA%20API](http://en.wikipedia.org/wiki/JAVA%20API)

¹⁰ [HTTP://EN.WIKIPEDIA.ORG/WIKI/JAVA%20VIRTUAL%20MACHINE](http://en.wikipedia.org/wiki/JAVA%20VIRTUAL%20MACHINE)

¹¹ [HTTP://EN.WIKIPEDIA.ORG/WIKI/GNU%20COMPILER%20COLLECTION](http://en.wikipedia.org/wiki/GNU%20COMPILER%20COLLECTION)

¹² [HTTP://EN.WIKIPEDIA.ORG/WIKI/COMPARISON%20OF%20JAVA%20TO%20CPLUSPLUS](http://en.wikipedia.org/wiki/COMPARISON%20OF%20JAVA%20TO%20CPLUSPLUS)

¹³ [HTTP://EN.WIKIPEDIA.ORG/WIKI/JINI](http://en.wikipedia.org/wiki/JINI)

¹⁴ [HTTP://EN.WIKIPEDIA.ORG/WIKI/ECLIPSE%20%28COMPUTING%29](http://en.wikipedia.org/wiki/ECLIPSE%20%28COMPUTING%29)

¹⁵ [HTTP://EN.WIKIPEDIA.ORG/WIKI/OPEN%20SOURCE%20INTEGRATED%20DEVELOPMENT%20ENVIRONMENT](http://en.wikipedia.org/wiki/OPEN%20SOURCE%20INTEGRATED%20DEVELOPMENT%20ENVIRONMENT)

¹⁶ [HTTP://EN.WIKIPEDIA.ORG/WIKI/NETBEANS](http://en.wikipedia.org/wiki/NETBEANS)

¹⁷ [HTTP://EN.WIKIPEDIA.ORG/WIKI/OPTIMIZATION%20OF%20JAVA](http://en.wikipedia.org/wiki/OPTIMIZATION%20OF%20JAVA)

31 Links

31.1 External References

- [JAVA CERTIFICATION PREPARATION GUIDES¹](#)
- [JAVA CERTIFICATION MOCK EXAMS² 500+ questions with exam simulator \(this is the older 1.4 version of the exam\)](#)
- [JAVA LANGUAGE SPECIFICATION, 3RD EDITION³.](#)
- [THINKING IN JAVA⁴](#)
- [JAVA 5 SDK DOCUMENTATION⁵](#)
- [JAVA 5 SDK DOCUMENTATION IN CHM FORMAT⁶](#)
- [JAVA 5 API DOCUMENTATION⁷](#)
- [THE JAVA TUTORIAL⁸](#)
- [Sun Developer Network NEW TO JAVA CENTER⁹](#)
- [A SIMPLE JAVA TUTORIAL¹⁰](#)
- [TWO SEMESTERS OF COLLEGE-LEVEL JAVA LECTURES--FREE¹¹](#)
- [JAVA LESSONS - INTERACTIVE JAVA PROGRAMMING TUTORIALS BASED ON EXAMPLES¹²](#)
- [JAVA TUTORIALS FOR KIDS AND ADULTS¹³](#)

31.2 External links

- [JAVA CERTIFICATION MOCK EXAMS¹⁴ 500+ questions with exam simulator](#)
- [SWINGWIKI¹⁵ - Open documentation project containing tips, tricks and best practices for Java Swing development](#)
- [JAVATIPS¹⁶ - Blog project containing best JAVA tips and tricks](#)

1 [HTTP://WWW.EPRACTIZELABS.COM/](http://www.epractizelabs.com/)
2 [HTTP://WWW.CERTIFICATION4CAREER.COM](http://www.certification4career.com)
3 [HTTP://JAVA.SUN.COM/DOCS/BOOKS/JLS/THIRD_EDITION/HTML/J3TOC.HTML](http://java.sun.com/docs/books/jls/third_edition/html/j3TOC.html)
4 [HTTP://WWW.MINDVIEW.NET/BOOKS/TIJ/](http://www.mindview.net/books/tij/)
5 [HTTP://JAVA.SUN.COM/J2SE/1.5.0/DOCS/INDEX.HTML](http://java.sun.com/j2se/1.5.0/docs/index.html)
6 [HTTP://WWW.ZEUSEDIT.COM/FORUM/VIEWTOPIC.PHP?T=10](http://www.zeusedit.com/forum/viewtopic.php?t=10)
7 [HTTP://JAVA.SUN.COM/J2SE/1.5.0/DOCS/API/INDEX.HTML](http://java.sun.com/j2se/1.5.0/docs/api/index.html)
8 [HTTP://JAVA.SUN.COM/DOCS/BOOKS/TUTORIAL/INDEX.HTML](http://java.sun.com/docs/books/tutorial/index.html)
9 [HTTP://JAVA.SUN.COM/DEVELOPER/ONLINETRAINING/NEW2JAVA/INDEX.HTML](http://java.sun.com/developer/onlineTraining/new2java/index.html)
10 [HTTP://WWW.ALNAJA7.NET/PROGRAMMER/393/ITCS-393.HTM](http://www.alnaja7.net/programmer/393/ITCS-393.htm)
11 [HTTP://CURMUDGEON99.GOOGLEPAGES.COM/](http://curmudgeon99.googlepages.com/)
12 [HTTP://JAVALESSONS.COM](http://javalessons.com)
13 [HTTP://WWW.KIDWARESOFTWARE.COM](http://www.kidwaresoftware.com)
14 [HTTP://WWW.CERTIFICATION4CAREER.COM](http://www.certification4career.com)
15 [HTTP://WWW.SWINGWIKI.ORG](http://www.swingwiki.org)
16 [HTTP://WWW.AKKIDI.COM](http://www.akkidi.com)

- [FREE JAVA/ ADVANCED JAVA BOOKS](#)¹⁷
- [FREE JAVA AND J2EE eBooks](#)¹⁸
- [JAVA BOOKS AVAILABLE FOR FREE DOWNLOADS](#)¹⁹
- [ROEDY GREEN'S JAVA & INTERNET GLOSSARY](#)²⁰ A comprehensive reference that's also an excellent starting point for beginners
- [C2: JAVA LANGUAGE](#)²¹
- [NETBEANS IDE](#)²²
- [ECLIPSE IDE](#)²³
- [ZEUS FOR WINDOWS IDE](#)²⁴
- [OFFICIAL JAVA HOME SITE](#)²⁵
- [ORIGINAL JAVA WHITEPAPER](#)²⁶
- [COMPLETE JAVA PROGRAMMING TUTORIALS](#)²⁷
- [JAVAPASSION, JAVA COURSE](#)²⁸ - The Javapassion Site, Java Course, driven by Sang Shin from Sun
- [BEANSHELL](#)²⁹ Interpreted version
- [THE JAVA LANGUAGE SPECIFICATION, THIRD EDITION](#)³⁰ "This book attempts a complete specification of the syntax and semantics of the language."
- [THE JAVA VIRTUAL MACHINE SPECIFICATION, SECOND EDITION](#)³¹ and [AMENDMENTS](#)³²
- [A PURE JAVA DESKTOP](#)³³
- [JAVAPEDIA PROJECT](#)³⁴
- [Bruce Eckel Thinking in Java Third edition](#) -- [HTTP://WWW.MINDVIEW.NET/BOOKS/TIJ/](http://www.mindview.net/books/tij/)³⁵ (Bruce has an C/C++ free book available on-line too)
- [JAVAGAMEDEVELOPMENT](#)³⁶ Daily news and articles on Java Game Development
- [JAVA CERTIFICATIONS SITE\(SCJP,SCWCD,SCBCD,JAVA 5.0,SCEA\)](#)³⁷
- [JAVA PROGRAMMING FAQs AND TUTORIALS](#)³⁸
- [MORE RESOURCES](#)³⁹
- [JAVA LESSONS](#)⁴⁰

17 [HTTP://WWW.FREEBOOKCENTRE.NET/JAVATECH/JAVACATEGORY.HTML](http://www.freebookcentre.net/javatech/javacategory.html)

18 [HTTP://WWW.BESTEBOOKSWORLD.COM/DEFAULT.ASP?CAT=55](http://www.bestebooks.com/default.asp?cat=55)

19 [HTTP://WWW.TECHBOOKSFORFREE.COM/JAVA.SHTML](http://www.techbooksforfree.com/java.shtml)

20 [HTTP://WWW.MINDPROD.COM/JGLOSS/JGLOSS.HTML](http://www.mindprod.com/jgloss/jgloss.html)

21 [HTTP://C2.COM/CGI/WIKI?JAVALANGUE](http://c2.com/cgi/wiki?javalangue)

22 [HTTP://WWW.NETBEANS.ORG](http://www.netbeans.org)

23 [HTTP://WWW.ECLIPSE.ORG](http://www.eclipse.org)

24 [HTTP://WWW.ZEUSEDIT.COM/JAVA.HTML](http://www.zeusedit.com/java.html)

25 [HTTP://JAVA.SUN.COM](http://java.sun.com)

26 [HTTP://JAVA.SUN.COM/DOCS/WHITE/LANGENV/](http://java.sun.com/docs/white/langenv/)

27 [HTTP://WWW.ROSEINDIA.NET/JAVA/](http://www.roseindia.net/java/)

28 [HTTP://WWW.JAVAPASSION.COM/JAVAINTRO/](http://www.javapassion.com/javaintro/)

29 [HTTP://WWW.BEANSHELL.ORG](http://www.beanshell.org)

30 [HTTP://JAVA.SUN.COM/DOCS/BOOKS/JLS/THIRD_EDITION/HTML/J3TOC.HTML](http://java.sun.com/docs/books/jls/third_edition/html/j3toc.html)

31 [HTTP://JAVA.SUN.COM/DOCS/BOOKS/VMSPEC/2ND-EDITION/HTML/VMSPECTOC.DOC.HTML](http://java.sun.com/docs/books/vmspec/2nd-edition/html/vmspectoc.doc.html)

32 [HTTP://JAVA.SUN.COM/DOCS/BOOKS/VMSPEC/2ND-EDITION/JVMS-CLARIFY.HTML](http://java.sun.com/docs/books/vmspec/2nd-edition/jvms-clarify.html)

33 [HTTP://WWW.JDISTRO.COM/](http://www.jdistro.com/)

34 [HTTP://WIKI.JAVA.NET/BIN/VIEW/JAVAPEDIA/](http://wiki.java.net/bin/view/Javapedia/)

35 [HTTP://WWW.MINDVIEW.NET/BOOKS/TIJ/](http://www.mindview.net/books/tij/)

36 [HTTP://JAVAGAMEDEVELOPMENT.NET](http://javagamedevelopment.net)

37 [HTTP://WWW.JAVABEAT.NET](http://www.javabeat.net)

38 [HTTP://WWW.APL.JHU.EDU/~{ }HALL/JAVA/FAQS-AND-TUTORIALS.HTML](http://www.apl.jhu.edu/~{ }hall/java/faqs-and-tutorials.html)

39 [HTTP://FINDSHELL.COM](http://findshell.com)

40 [HTTP://WWW.LANDOFCODE.COM/JAVA/](http://www.landofcode.com/java/)

- [ONLINE JAVA TUTORIAL](#)⁴¹
- [FULL JAVA TUTORIAL](#)⁴² - A collection of free premium programming tutorials
- [JAVA CERTIFICATION PRACTICE TESTS AND ARTICLES](#)⁴³
- [KODE JAVA - LEARN JAVA PROGRAMMING BY EXAMPLES](#)⁴⁴
- [GAMES PROGRAMMING WIKI](#)⁴⁵ - Java tutorials and lessons based on game programming
- [WIKIJAVA](#)⁴⁶ - Examples and tutorials in Java
- [DOWNLOAD FREE JAVA EBOOKS FROM 83 EBOOKS COLLECTION](#)⁴⁷ - Free Java Ebooks to download from ebookslab.info
- [DOWNLOAD FREE SUN CERTIFIED DEVELOPER FOR JAVA WEB SERVICES](#)⁴⁸ - Free Java Ebooks to download from ebooks.mzwriter.net
- [CODE CONVENTIONS FOR THE JAVA PROGRAMMING LANGUAGE](#)⁴⁹ - At SUN
- [JAVA LESSONS AT LEOLOL](#)⁵⁰ - A collection of introductory lessons to Java
- [JAVA EXERCISES AT LEOLOL](#)⁵¹ - A collection of Java exercises with sample solutions

Newsgroups:

- [\[news:comp.lang.java comp.lang.java\]](#) ([GOOGLE'S WEB INTERFACE](#)⁵²)

41 [HTTP://COMPUTER.FREEONLINEBOOKSTORE.ORG/SHOWBOOK.PHP?SUBCATEGORYID=17](http://computer.freeonlinebookstore.org/showbook.php?subcategoryId=17)

42 [HTTP://WWW.MESHPLEX.ORG/WIKI/JAVA/INTRODUCTION_TO_JAVA](http://www.meshplex.org/wiki/java/introduction_to_java)

43 [HTTP://WWW.UCERTIFY.COM/VENDORS/SUN.HTML](http://www.ucertify.com/vendors/sun.html)

44 [HTTP://WWW.KODEJAVA.ORG/](http://www.kodejava.org/)

45 [HTTP://GPWIKI.ORG/](http://gpwiki.org/)

46 [HTTP://WWW.WIKIJAVA.ORG/](http://www.wikijava.org/)

47 [HTTP://WWW.EBOOKSLAB.INFO/DOWNLOAD-FREE-JAVA-EBOOKS](http://www.ebookslab.info/download-free-java-ebooks)

48 [HTTP://EBOOKS.MZWRITER.NET/E-BOOKS/SHARE_EBOOK-310-220-SUN-CERTIFIED-DEVELOPER-FOR-JAVA-WEB-](http://ebooks.mzwriter.net/e-books/share_ebook-310-220-sun-certified-developer-for-java-web-)

49 [HTTP://JAVA.SUN.COM/DOCS/CODECONV/HTML/CODECONVTOC.DOC.HTML](http://java.sun.com/docs/codeconv/html/codeconvtoc.doc.html)

50 [HTTP://WWW.LEOLOL.COM/DRUPAL/TUTORIALS/JAVA/LESSONS](http://www.leolol.com/drupal/tutorials/java/lessons)

51 [HTTP://WWW.LEOLOL.COM/DRUPAL/TUTORIALS/JAVA/EXERCISES](http://www.leolol.com/drupal/tutorials/java/exercises)

52 [HTTP://GROUPS.GOOGLE.COM/GROUPS?GROUP=COMP.LANG.JAVA](http://groups.google.com/groups?group=comp.lang.java)

32 License

33 GNU Free Documentation License

1. REDIRECT WIKIBOOKS:GNU FREE DOCUMENTATION LICENSE¹

DE:JAVA² DEUTSCH J.SE³

¹ [HTTP://EN.WIKIBOOKS.ORG/WIKI/GNU%20FREE%20DOCUMENTATION%20LICENSE](http://en.wikibooks.org/wiki/GNU%20Free%20Documentation%20License)

² [HTTP://DE.WIKIBOOKS.ORG/WIKI/JAVA](http://de.wikibooks.org/wiki/JAVA)

³ [HTTP://DE.WIKIBOOKS.ORG/WIKI/JAVA%20STANDARD](http://de.wikibooks.org/wiki/JAVA%20STANDARD)

34 Authors

Edits	User
2	1WHEEL ¹
1	A. B. 10 ²
3	AVERMA ³
1	ADRILEY ⁴
1	ADDPS4CAT ⁵
35	ADRIGNOLA ⁶
3	ALAINR345 ⁷
1	ALEXANDER.ORLOV ⁸
4	ANONYMOUS DISSIDENT ⁹
2	ANTIDRUGUE ¹⁰
1	APHONIK ¹¹
6	ARLEN22 ¹²
1	ARSENALFAN ¹³
225	ARUNREGINALD ¹⁴
7	ASHMAILIT ¹⁵
2	AZ1568 ¹⁶
1	B3T ¹⁷
1	BRUTE ¹⁸
2	BASTIE ¹⁹
1	BENO1000 ²⁰
2	BENPAGE26 ²¹

1	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:1WHEEL
2	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:A._B._10
3	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:AVERMA
4	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:ADRILEY
5	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:ADDPS4CAT
6	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:ADRIGNOLA
7	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:ALAINR345
8	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:ALEXANDER.ORLOV
9	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:ANONYMOUS_DISSIDENT
10	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:ANTIDRUGUE
11	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:APHONIK
12	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:ARLEN22
13	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:ARSENALFAN
14	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:ARUNREGINALD
15	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:ASHMAILIT
16	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:AZ1568
17	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:B3T
18	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:BRUTE
19	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:BASTIE
20	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:BENO1000
21	HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:BENPAGE26

1 BIGDUNC²²
 1 BRENTP²³
 1 CARSRACBOT²⁴
 1 CECLAUSON²⁵
 4 COLFULUS²⁶
 1 COLINDAVEN²⁷
 1 COOLBOY1234²⁸
 1 CSPURRIER²⁹
 2 DALLAS1278³⁰
 8 DAN POLANSKY³¹
 5 DARKLAMA³²
 5 DARKXXXXILLUSION³³
 8 DAVIDBOURGUIGNON³⁴
 6 DAVIDCARY³⁵
 2 DERBETH³⁶
 1 DEVOURER09³⁷
 1 DICKDICKDICK³⁸
 6 DIRK HÜNNIGER³⁹
 3 DIRK GENTLY⁴⁰
 95 DJB⁴¹
 2 DMONEGO⁴²
 1 DRAWDE83⁴³
 1 ELI VERBUYST⁴⁴
 4 ELLMIST⁴⁵
 720 ERVINN⁴⁶

22 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:BIGDUNC](http://en.wikibooks.org/w/index.php?title=User:BIGDUNC)
 23 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:BRENTP](http://en.wikibooks.org/w/index.php?title=User:BRENTP)
 24 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:CARSRACBOT](http://en.wikibooks.org/w/index.php?title=User:CARSRACBOT)
 25 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:CECLAUSON](http://en.wikibooks.org/w/index.php?title=User:CECLAUSON)
 26 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:COLFULUS](http://en.wikibooks.org/w/index.php?title=User:COLFULUS)
 27 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:COLINDAVEN](http://en.wikibooks.org/w/index.php?title=User:COLINDAVEN)
 28 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:COOLBOY1234](http://en.wikibooks.org/w/index.php?title=User:COOLBOY1234)
 29 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:CSPURRIER](http://en.wikibooks.org/w/index.php?title=User:CSPURRIER)
 30 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:DALLAS1278](http://en.wikibooks.org/w/index.php?title=User:DALLAS1278)
 31 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:DAN_POLANSKY](http://en.wikibooks.org/w/index.php?title=User:DAN_POLANSKY)
 32 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:DARKLAMA](http://en.wikibooks.org/w/index.php?title=User:DARKLAMA)
 33 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:DARKXXXXILLUSION](http://en.wikibooks.org/w/index.php?title=User:DARKXXXXILLUSION)
 34 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:DAVIDBOURGUIGNON](http://en.wikibooks.org/w/index.php?title=User:DAVIDBOURGUIGNON)
 35 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:DAVIDCARY](http://en.wikibooks.org/w/index.php?title=User:DAVIDCARY)
 36 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:DERBETH](http://en.wikibooks.org/w/index.php?title=User:DERBETH)
 37 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:DEVOURER09](http://en.wikibooks.org/w/index.php?title=User:DEVOURER09)
 38 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:DICKDICKDICK](http://en.wikibooks.org/w/index.php?title=User:DICKDICKDICK)
 39 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:DIRK_H%C3%BCNNIGER](http://en.wikibooks.org/w/index.php?title=User:DIRK_H%C3%BCNNIGER)
 40 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:DIRK_GENTLY](http://en.wikibooks.org/w/index.php?title=User:DIRK_GENTLY)
 41 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:DJB](http://en.wikibooks.org/w/index.php?title=User:DJB)
 42 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:DMONEGO](http://en.wikibooks.org/w/index.php?title=User:DMONEGO)
 43 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:DRAWDE83](http://en.wikibooks.org/w/index.php?title=User:DRAWDE83)
 44 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:ELI_VERBUYST](http://en.wikibooks.org/w/index.php?title=User:ELI_VERBUYST)
 45 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:ELLMIST](http://en.wikibooks.org/w/index.php?title=User:ELLMIST)
 46 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:ERVINN](http://en.wikibooks.org/w/index.php?title=User:ERVINN)

- 14 EXABYTE⁴⁷
- 6 EXPLANATOR⁴⁸
- 3 FELIPEOCHOA0918⁴⁹
- 3 FISHPI⁵⁰
- 1 FKEREKI⁵¹
- 1 FLATIPAC⁵²
- 12 FORAGE⁵³
- 2 FRATERM⁵⁴
- 2 FREDMARANHAO⁵⁵
- 1 FRIGOTONI⁵⁶
- 11 FTIERCEL⁵⁷
- 1 FUNNYMAN3595⁵⁸
- 2 GOLLOXP⁵⁹
- 2 GROKUS⁶⁰
- 1 GRUNNY⁶¹
- 1 GUANACO⁶²
- 5 GUANGPU.HUANG⁶³
- 1 HMPERSON1⁶⁴
- 2 HAGINDAZ⁶⁵
- 1 HERMIONE1980⁶⁶
- 1 HETHRIR⁶⁷
- 1 HETHRIRBOT⁶⁸
- 6 HIGHLAND⁶⁹
- 1 HRODULF⁷⁰
- 2 IMACWIN95⁷¹

-
- 47 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:EXABYTE](http://en.wikibooks.org/w/index.php?title=User:EXABYTE)
 - 48 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:EXPLANATOR](http://en.wikibooks.org/w/index.php?title=User:EXPLANATOR)
 - 49 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:FELIPEOCHOA0918](http://en.wikibooks.org/w/index.php?title=User:FELIPEOCHOA0918)
 - 50 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:FISHPI](http://en.wikibooks.org/w/index.php?title=User:FISHPI)
 - 51 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:FKEREKI](http://en.wikibooks.org/w/index.php?title=User:FKEREKI)
 - 52 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:FLATIPAC](http://en.wikibooks.org/w/index.php?title=User:FLATIPAC)
 - 53 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:FORAGE](http://en.wikibooks.org/w/index.php?title=User:FORAGE)
 - 54 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:FRATERM](http://en.wikibooks.org/w/index.php?title=User:FRATERM)
 - 55 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:FREDMARANHAO](http://en.wikibooks.org/w/index.php?title=User:FREDMARANHAO)
 - 56 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:FRIGOTONI](http://en.wikibooks.org/w/index.php?title=User:FRIGOTONI)
 - 57 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:FTIERCEL](http://en.wikibooks.org/w/index.php?title=User:FTIERCEL)
 - 58 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:FUNNYMAN3595](http://en.wikibooks.org/w/index.php?title=User:FUNNYMAN3595)
 - 59 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:GoLLOXP](http://en.wikibooks.org/w/index.php?title=User:GoLLOXP)
 - 60 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:GROKUS](http://en.wikibooks.org/w/index.php?title=User:GROKUS)
 - 61 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:GRUNNY](http://en.wikibooks.org/w/index.php?title=User:GRUNNY)
 - 62 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:GUANACO](http://en.wikibooks.org/w/index.php?title=User:GUANACO)
 - 63 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:GUANGPU.HUANG](http://en.wikibooks.org/w/index.php?title=User:GUANGPU.HUANG)
 - 64 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:HMPERSON1](http://en.wikibooks.org/w/index.php?title=User:HMPERSON1)
 - 65 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:HAGINDAZ](http://en.wikibooks.org/w/index.php?title=User:HAGINDAZ)
 - 66 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:HERMIONE1980](http://en.wikibooks.org/w/index.php?title=User:HERMIONE1980)
 - 67 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:HETHRIR](http://en.wikibooks.org/w/index.php?title=User:HETHRIR)
 - 68 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:HETHRIRBOT](http://en.wikibooks.org/w/index.php?title=User:HETHRIRBOT)
 - 69 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:HIGHLAND](http://en.wikibooks.org/w/index.php?title=User:HIGHLAND)
 - 70 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:HRO%C3%B0ULF](http://en.wikibooks.org/w/index.php?title=User:HRO%C3%B0ULF)
 - 71 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:IMACWIN95](http://en.wikibooks.org/w/index.php?title=User:IMACWIN95)

2 IMAGINATIONAC⁷²
 1 ITSBORIN⁷³
 1 J36MILES⁷⁴
 7 JACKPOTTE⁷⁵
 1 JAMES.SUTHERLAND⁷⁶
 33 JGUK⁷⁷
 1 JIMMYATIC⁷⁸
 1 JK33⁷⁹
 7 JOMEGAT⁸⁰
 2 JONATHAN WEBLEY⁸¹
 1 JPKOTTA⁸²
 8 JSONSTEIN⁸³
 2 KAATIEP⁸⁴
 1 KEJIA⁸⁵
 1 KENJ0418⁸⁶
 1 KRIAK⁸⁷
 4 KRISCHIK⁸⁸
 1 LCAWTE⁸⁹
 1 LESATH⁹⁰
 1 LMDELGADO⁹¹
 1 LUKE101⁹²
 1 LUÍS VITÓRIO CARGNINI⁹³
 6 MALFIST⁹⁴
 2 MATRIXFROG⁹⁵
 14 MATTYLAWS⁹⁶

72 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:IMAGINATIONAC](http://en.wikibooks.org/w/index.php?title=User:IMAGINATIONAC)
 73 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:ITSBORIN](http://en.wikibooks.org/w/index.php?title=User:ITSBORIN)
 74 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:J36MILES](http://en.wikibooks.org/w/index.php?title=User:J36MILES)
 75 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:JACKPOTTE](http://en.wikibooks.org/w/index.php?title=User:JACKPOTTE)
 76 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:JAMES.SUTHERLAND](http://en.wikibooks.org/w/index.php?title=User:JAMES.SUTHERLAND)
 77 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:JGUK](http://en.wikibooks.org/w/index.php?title=User:JGUK)
 78 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:JIMMYATIC](http://en.wikibooks.org/w/index.php?title=User:JIMMYATIC)
 79 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:JK33](http://en.wikibooks.org/w/index.php?title=User:JK33)
 80 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:JOMEGAT](http://en.wikibooks.org/w/index.php?title=User:JOMEGAT)
 81 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:JONATHAN_WEBLEY](http://en.wikibooks.org/w/index.php?title=User:JONATHAN_WEBLEY)
 82 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:JPKOTTA](http://en.wikibooks.org/w/index.php?title=User:JPKOTTA)
 83 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:JSONSTEIN](http://en.wikibooks.org/w/index.php?title=User:JSONSTEIN)
 84 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:KAATIEP](http://en.wikibooks.org/w/index.php?title=User:KAATIEP)
 85 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:KEJIA](http://en.wikibooks.org/w/index.php?title=User:KEJIA)
 86 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:KENJ0418](http://en.wikibooks.org/w/index.php?title=User:KENJ0418)
 87 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:KRIAK](http://en.wikibooks.org/w/index.php?title=User:KRIAK)
 88 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:KRISCHIK](http://en.wikibooks.org/w/index.php?title=User:KRISCHIK)
 89 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:LCAWTE](http://en.wikibooks.org/w/index.php?title=User:LCAWTE)
 90 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:LESATH](http://en.wikibooks.org/w/index.php?title=User:LESATH)
 91 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:LMDELGADO](http://en.wikibooks.org/w/index.php?title=User:LMDELGADO)
 92 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:LUKE101](http://en.wikibooks.org/w/index.php?title=User:LUKE101)
 93 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:LU%C3%ADs_VIT%C3%B3RIO_CARGNINI](http://en.wikibooks.org/w/index.php?title=User:LU%C3%ADs_VIT%C3%B3RIO_CARGNINI)
 94 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:MALFIST](http://en.wikibooks.org/w/index.php?title=User:MALFIST)
 95 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:MATRIXFROG](http://en.wikibooks.org/w/index.php?title=User:MATRIXFROG)
 96 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:MATTYLAWS](http://en.wikibooks.org/w/index.php?title=User:MATTYLAWS)

- 1 MAXBOWSHER⁹⁷
- 1 MESSI⁹⁸
- 3 METABOHEMIAN⁹⁹
- 1 MHAYES¹⁰⁰
- 1 MIKE.LIFEGUARD¹⁰¹
- 63 MIKM¹⁰²
- 2 MKN¹⁰³
- 1 MS2GER¹⁰⁴
- 3 MSHONLE¹⁰⁵
- 1 MSTENTA¹⁰⁶
- 2 N313T3¹⁰⁷
- 1 OLDGEEK¹⁰⁸
- 1 PADDU¹⁰⁹
- 22 PANIC2K4¹¹⁰
- 1 PENGO¹¹¹
- 3 PHILIP¹¹²
- 2 PITEL¹¹³
- 2 POWEROID¹¹⁴
- 15 QUITEUNUSUAL¹¹⁵
- 1 RDEV¹¹⁶
- 1 RAKESH KPN¹¹⁷
- 1 RALPHCOOK¹¹⁸
- 10 RAPPO¹¹⁹
- 1 RAVICHANDAR84¹²⁰
- 1 RAYKIDDY¹²¹

-
- 97 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:MAXBOWSHER](http://en.wikibooks.org/w/index.php?title=User:MaxBowsHer)
 - 98 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:MESSI](http://en.wikibooks.org/w/index.php?title=User:Messi)
 - 99 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:METABoHEMIAN](http://en.wikibooks.org/w/index.php?title=User:MetaBohemian)
 - 100 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:MHAYES](http://en.wikibooks.org/w/index.php?title=User:Mhayes)
 - 101 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:MIKE.LIFEGUARD](http://en.wikibooks.org/w/index.php?title=User:Mike.LifeGuard)
 - 102 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:MIKM](http://en.wikibooks.org/w/index.php?title=User:MIKM)
 - 103 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:MKN](http://en.wikibooks.org/w/index.php?title=User:MKN)
 - 104 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:MS2GER](http://en.wikibooks.org/w/index.php?title=User:Ms2Ger)
 - 105 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:MSHONLE](http://en.wikibooks.org/w/index.php?title=User:Mshonle)
 - 106 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:MSTENTA](http://en.wikibooks.org/w/index.php?title=User:Mstenta)
 - 107 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:N313T3](http://en.wikibooks.org/w/index.php?title=User:N313T3)
 - 108 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:OLDGEEK](http://en.wikibooks.org/w/index.php?title=User:OldGeek)
 - 109 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:PADDU](http://en.wikibooks.org/w/index.php?title=User:Paddu)
 - 110 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:PANIC2K4](http://en.wikibooks.org/w/index.php?title=User:Panic2k4)
 - 111 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:PENGO](http://en.wikibooks.org/w/index.php?title=User:Pengo)
 - 112 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:PHILIP](http://en.wikibooks.org/w/index.php?title=User:Philip)
 - 113 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:PITEL](http://en.wikibooks.org/w/index.php?title=User:Pixel)
 - 114 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:POWEROID](http://en.wikibooks.org/w/index.php?title=User:Poweroid)
 - 115 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:QUITEUNUSUAL](http://en.wikibooks.org/w/index.php?title=User:QuiteUnusual)
 - 116 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:RDEV](http://en.wikibooks.org/w/index.php?title=User:RDev)
 - 117 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:RAKESH_KPN](http://en.wikibooks.org/w/index.php?title=User:Rakesh_KPN)
 - 118 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:RALPHCOOK](http://en.wikibooks.org/w/index.php?title=User:RalphCook)
 - 119 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:RAPPO](http://en.wikibooks.org/w/index.php?title=User:Rappo)
 - 120 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:RAVICHANDAR84](http://en.wikibooks.org/w/index.php?title=User:Ravichandar84)
 - 121 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=USER:RAYKIDDY](http://en.wikibooks.org/w/index.php?title=User:RayKiddy)

- 7 RECENT RUNES¹²²
- 2 REMI¹²³
- 11 RICKY CLARKSON¹²⁴
- 1 ROBERT HORNING¹²⁵
- 3 SBJOHNNY¹²⁶
- 1 SAILWINHEIN¹²⁷
- 6 SAMWILSON¹²⁸
- 1 SAVH¹²⁹
- 1 SEANJA¹³⁰
- 4 SHAHIDSIDD¹³¹
- 2 SHAUNW¹³²
- 1 SHENME¹³³
- 68 SIGMA 7¹³⁴
- 1 SNARIUS¹³⁵
- 1 SPOCK431¹³⁶
- 8 SPONGEBOB88¹³⁷
- 16 SPOON!¹³⁸
- 1 STEPHANVANINGEN¹³⁹
- 4 SUNDAR22IN¹⁴⁰
- 3 SUNNYCHAN¹⁴¹
- 7 SUPERFLY JON¹⁴²
- 1 SWIFT¹⁴³
- 1 TALKTOATISH¹⁴⁴
- 2 TANMINIVAN¹⁴⁵
- 1 TAROSE.TREVOR¹⁴⁶

-
- 122 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:RECENT_RUNES](http://en.wikibooks.org/w/index.php?title=User:Recent_Runes)
 - 123 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:REMI](http://en.wikibooks.org/w/index.php?title=User:REMI)
 - 124 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:RICKY_CLARKSON](http://en.wikibooks.org/w/index.php?title=User:Ricky_Clarkson)
 - 125 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:ROBERT_HORNING](http://en.wikibooks.org/w/index.php?title=User:Robert_Horning)
 - 126 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:SBJOHNNY](http://en.wikibooks.org/w/index.php?title=User:SBJohnny)
 - 127 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:SAILWINHEIN](http://en.wikibooks.org/w/index.php?title=User:Sailwinhein)
 - 128 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:SAMWILSON](http://en.wikibooks.org/w/index.php?title=User:SamWilson)
 - 129 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:SAVH](http://en.wikibooks.org/w/index.php?title=User:SAVH)
 - 130 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:SEANJA](http://en.wikibooks.org/w/index.php?title=User:Seanja)
 - 131 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:SHAHIDSIDD](http://en.wikibooks.org/w/index.php?title=User:Shahidsidd)
 - 132 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:SHAUNW](http://en.wikibooks.org/w/index.php?title=User:ShaunW)
 - 133 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:SHENME](http://en.wikibooks.org/w/index.php?title=User:Shenme)
 - 134 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:SIGMA_7](http://en.wikibooks.org/w/index.php?title=User:Sigma_7)
 - 135 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:SNARIUS](http://en.wikibooks.org/w/index.php?title=User:Snarius)
 - 136 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:SPOCK431](http://en.wikibooks.org/w/index.php?title=User:SPOCK431)
 - 137 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:SPONGEBOB88](http://en.wikibooks.org/w/index.php?title=User:Spongebob88)
 - 138 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:SPOON%21](http://en.wikibooks.org/w/index.php?title=User:Spoon%21)
 - 139 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:STEPHANVANINGEN](http://en.wikibooks.org/w/index.php?title=User:Stephanvaningen)
 - 140 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:SUNDAR22IN](http://en.wikibooks.org/w/index.php?title=User:Sundar22in)
 - 141 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:SUNNYCHAN](http://en.wikibooks.org/w/index.php?title=User:SunnyChan)
 - 142 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:SUPERFLY_JON](http://en.wikibooks.org/w/index.php?title=User:Superfly_Jon)
 - 143 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:SWIFT](http://en.wikibooks.org/w/index.php?title=User:Swift)
 - 144 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:TALKTOATISH](http://en.wikibooks.org/w/index.php?title=User:Talktoatish)
 - 145 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:TANMINIVAN](http://en.wikibooks.org/w/index.php?title=User:Tanminivan)
 - 146 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:TAROSE.TREVOR](http://en.wikibooks.org/w/index.php?title=User:Tarose.Trevor)

- 1 THARENTHEL¹⁴⁷
- 1 THEDAVEROSS¹⁴⁸
- 3 THEPHILWELLS¹⁴⁹
- 2 UNV¹⁵⁰
- 1 UCERTIFY¹⁵¹
- 1 VINAY H¹⁵²
- 2 WEBAWARE¹⁵³
- 1 WHITEKNIGHT¹⁵⁴
- 6 WIKIWIZARD¹⁵⁵
- 2 WIKIALT¹⁵⁶
- 1 WIKIMI-DHIANN¹⁵⁷
- 1 WILLEM SOUWER¹⁵⁸
- 1 WISEEYES¹⁵⁹
- 1 WUR-DENE¹⁶⁰
- 2 WUTZOFANT¹⁶¹
- 2 YMS¹⁶²
- 6 YUUKI MAYUKI¹⁶³
- 9 ZEROONE¹⁶⁴
- 1 Володимир Груша¹⁶⁵
- 1 砲火 万物の靈長¹⁶⁶

-
- 147 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:THARENTHEL](http://en.wikibooks.org/w/index.php?title=User:Tharenthel)
 - 148 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:THEDAVEROSS](http://en.wikibooks.org/w/index.php?title=User:TheDAVEROSS)
 - 149 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:THEPHILWELLS](http://en.wikibooks.org/w/index.php?title=User:ThePhilWells)
 - 150 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:UNV](http://en.wikibooks.org/w/index.php?title=User:UNV)
 - 151 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:UCERTIFY](http://en.wikibooks.org/w/index.php?title=User:UCERTIFY)
 - 152 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:VINAY_H](http://en.wikibooks.org/w/index.php?title=User:VINAY_H)
 - 153 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:WEBAWARE](http://en.wikibooks.org/w/index.php?title=User:WEBAWARE)
 - 154 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:WHITEKNIGHT](http://en.wikibooks.org/w/index.php?title=User:WHITEKNIGHT)
 - 155 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:WIKIWIZARD](http://en.wikibooks.org/w/index.php?title=User:WIKIWIZARD)
 - 156 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:WIKIALT](http://en.wikibooks.org/w/index.php?title=User:WIKIALT)
 - 157 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:WIKIMI-DHIANN](http://en.wikibooks.org/w/index.php?title=User:WIKIMI-DHIANN)
 - 158 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:WILLEM_SOUWER](http://en.wikibooks.org/w/index.php?title=User:WILLEM_SOUWER)
 - 159 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:WISEEYES](http://en.wikibooks.org/w/index.php?title=User:WISEEYES)
 - 160 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:WUR-DENE](http://en.wikibooks.org/w/index.php?title=User:WUR-DENE)
 - 161 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:WUTZOFANT](http://en.wikibooks.org/w/index.php?title=User:WUTZOFANT)
 - 162 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:YMS](http://en.wikibooks.org/w/index.php?title=User:YMS)
 - 163 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:YUUKI_MAYUKI](http://en.wikibooks.org/w/index.php?title=User:YUUKI_MAYUKI)
 - 164 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:ZEROONE](http://en.wikibooks.org/w/index.php?title=User:ZEROONE)
 - 165 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:%D0%92%D0%BE%D0%BB%D0%BE%D0%B4%D0%B8%D0%BC%D0%B8%D1%80_%D0%93%D1%80%D1%83%D1%88%D0%B0](http://en.wikibooks.org/w/index.php?title=User:%D0%92%D0%BE%D0%BB%D0%BE%D0%B4%D0%B8%D0%BC%D0%B8%D1%80_%D0%93%D1%80%D1%83%D1%88%D0%B0)
 - 166 [HTTP://EN.WIKIBOOKS.ORG/W/INDEX.PHP?TITLE=User:%E7%A0%B2%E7%81%AB_%E4%B8%87%E7%89%A9%E3%81%AE%E9%9C%8A%E9%95%B7](http://en.wikibooks.org/w/index.php?title=User:%E7%A0%B2%E7%81%AB_%E4%B8%87%E7%89%A9%E3%81%AE%E9%9C%8A%E9%95%B7)

List of Figures

- GFDL: Gnu Free Documentation License. <http://www.gnu.org/licenses/fdl.html>
- cc-by-sa-3.0: Creative Commons Attribution ShareAlike 3.0 License. <http://creativecommons.org/licenses/by-sa/3.0/>
- cc-by-sa-2.5: Creative Commons Attribution ShareAlike 2.5 License. <http://creativecommons.org/licenses/by-sa/2.5/>
- cc-by-sa-2.0: Creative Commons Attribution ShareAlike 2.0 License. <http://creativecommons.org/licenses/by-sa/2.0/>
- cc-by-sa-1.0: Creative Commons Attribution ShareAlike 1.0 License. <http://creativecommons.org/licenses/by-sa/1.0/>
- cc-by-2.0: Creative Commons Attribution 2.0 License. <http://creativecommons.org/licenses/by/2.0/>
- cc-by-2.0: Creative Commons Attribution 2.0 License. <http://creativecommons.org/licenses/by/2.0/deed.en>
- cc-by-2.5: Creative Commons Attribution 2.5 License. <http://creativecommons.org/licenses/by/2.5/deed.en>
- cc-by-3.0: Creative Commons Attribution 3.0 License. <http://creativecommons.org/licenses/by/3.0/deed.en>
- GPL: GNU General Public License. <http://www.gnu.org/licenses/gpl-2.0.txt>
- PD: This image is in the public domain.
- ATTR: The copyright holder of this file allows anyone to use it for any purpose, provided that the copyright holder is properly attributed. Redistribution, derivative work, commercial use, and all other use is permitted.
- EURO: This is the common (reverse) face of a euro coin. The copyright on the design of the common face of the euro coins belongs to the European Commission. Authorised is reproduction in a format without relief (drawings, paintings, films) provided they are not detrimental to the image of the euro.
- LFK: Lizenz Freie Kunst. <http://artlibre.org/licence/lal/de>
- CFR: Copyright free use.
- EPL: Eclipse Public License. <http://www.eclipse.org/org/documents/epl-v10.php>

List of Figures

1	NASA	PD
2	PETER CAMPBELL ¹⁶⁷	GFDL
3	USER:ARUNREGINALD ¹⁶⁸	GFDL
4	ARUN REGINALD ¹⁶⁹	cc-by-sa-3.0
5	ARUN REGINALD ¹⁷⁰	cc-by-sa-3.0
6	ARUN REGINALD ¹⁷¹	cc-by-sa-3.0
7	ARUN REGINALD ¹⁷²	cc-by-sa-3.0
8	ARUN REGINALD ¹⁷³	cc-by-sa-3.0
9		
10		
11		
12		
13		
14		
15	ARUN REGINALD ¹⁷⁴	GFDL
16	ARUN REGINALD ¹⁷⁵	GFDL
17	Original version created by B:USER:ERVINN ¹⁷⁶ , SVG version created by MYSELF ¹⁷⁷	cc-by-sa-2.5
18	Original version created by B:USER:ERVINN ¹⁷⁸ , SVG version created by MYSELF ¹⁷⁹	cc-by-sa-2.5
19		
20		
21		
22		
23		
24		
25	ARUNREGINALD ¹⁸⁰	GFDL

¹⁶⁷ [HTTP://EN.WIKIBOOKS.ORG/WIKI/USER%3APETER%20CAMPBELL](http://en.wikibooks.org/wiki/User%3APeter%20Campbell)

¹⁶⁸ [HTTP://EN.WIKIBOOKS.ORG/WIKI/USER%3AARUNREGINALD](http://en.wikibooks.org/wiki/User%3AARUNREGINALD)

¹⁶⁹ [HTTP://EN.WIKIBOOKS.ORG/WIKI/USER%3AARUNREGINALD](http://en.wikibooks.org/wiki/User%3AARUNREGINALD)

¹⁷⁰ [HTTP://EN.WIKIBOOKS.ORG/WIKI/USER%3AARUNREGINALD](http://en.wikibooks.org/wiki/User%3AARUNREGINALD)

¹⁷¹ [HTTP://EN.WIKIBOOKS.ORG/WIKI/USER%3AARUNREGINALD](http://en.wikibooks.org/wiki/User%3AARUNREGINALD)

¹⁷² [HTTP://EN.WIKIBOOKS.ORG/WIKI/USER%3AARUNREGINALD](http://en.wikibooks.org/wiki/User%3AARUNREGINALD)

¹⁷³ [HTTP://EN.WIKIBOOKS.ORG/WIKI/USER%3AARUNREGINALD](http://en.wikibooks.org/wiki/User%3AARUNREGINALD)

¹⁷⁴ [HTTP://EN.WIKIBOOKS.ORG/WIKI/USER%3AARUNREGINALD](http://en.wikibooks.org/wiki/User%3AARUNREGINALD)

¹⁷⁵ [HTTP://EN.WIKIBOOKS.ORG/WIKI/USER%3AARUNREGINALD](http://en.wikibooks.org/wiki/User%3AARUNREGINALD)

¹⁷⁶ [HTTP://EN.WIKIBOOKS.ORG/WIKI/USER%3AERVINN](http://en.wikibooks.org/wiki/User%3AERVINN)

¹⁷⁷ [HTTP://EN.WIKIBOOKS.ORG/WIKI/USER%3AMIKM](http://en.wikibooks.org/wiki/User%3AMIKM)

¹⁷⁸ [HTTP://EN.WIKIBOOKS.ORG/WIKI/USER%3AERVINN](http://en.wikibooks.org/wiki/User%3AERVINN)

¹⁷⁹ [HTTP://EN.WIKIBOOKS.ORG/WIKI/USER%3AMIKM](http://en.wikibooks.org/wiki/User%3AMIKM)

¹⁸⁰ [HTTP://EN.WIKIBOOKS.ORG/WIKI/USER%3AARUNREGINALD](http://en.wikibooks.org/wiki/User%3AARUNREGINALD)