IBM Software Group

IBM® TXSeries® for Multiplatforms V6

Introduction to TXSeries server programming in Java™

Updated October 25, 2006

This presentation will introduce you to Java server programming, one of the object oriented programming languages supported by TXSeries

Goals

- Provide an overview of object-oriented programming under TXSeries
- Understand the behavior of Java virtual machine under TXSeries
- Performance considerations
- Provide an overview of JCICS services
- JCICS class hierarchy

The goal of this presentation is to provide an overview of object-oriented programming under TXSeries, and help you understand the behavior of Java Virtual machine, also referred to as JVM under TXSeries. It will outline certain performance considerations and provide an overview of JCICS servers and the class hierarchy.

Agenda

- Introduction to Object oriented programming under TXSeries
  - Introduction to Server programming in Java
    - TXSeries and Java virtual machine
    - Performance considerations
- JCICS
  - What is JCICS?
  - JCICS Services
  - JCICS Programming considerations
  - JCICS class hierarchy

The agenda is to first introduce the topic of object oriented programming and TXSeries Java server programming. JCICS will also be defined and described.

Section

Introduction to object oriented programming under TXSeries

This section covers support provided by TXSeries for writing server applications using Java, and outline some performance considerations.

Object oriented programming under TXSeries

- TXSeries provides the flexibility of writing server programs using object-oriented programming style. With this support, you can:
  - Perceive TXSeries services as objects
  - Reusable classes
  - Create applications using object-oriented architecture.
  - Leverage use of existing application classes and tools provided by the language environment.

TXSeries provides the flexibility of writing server programs in both modular and object-oriented programming languages. With object-oriented programming, programmers can perceive TXSeries services as *objects*. TXSeries provides support for object-oriented programming in C++ and Java, allowing you to:

Perceive TXSeries services as objects.

Design and create reusable classes that can be used in and outside the TXSeries environment.

Design and create new modern TXSeries server applications using an object-oriented architecture. and

Effectively use the strengths of various existing application classes and tools that are provided with the object-oriented language.

Server programming in Java - Introduction

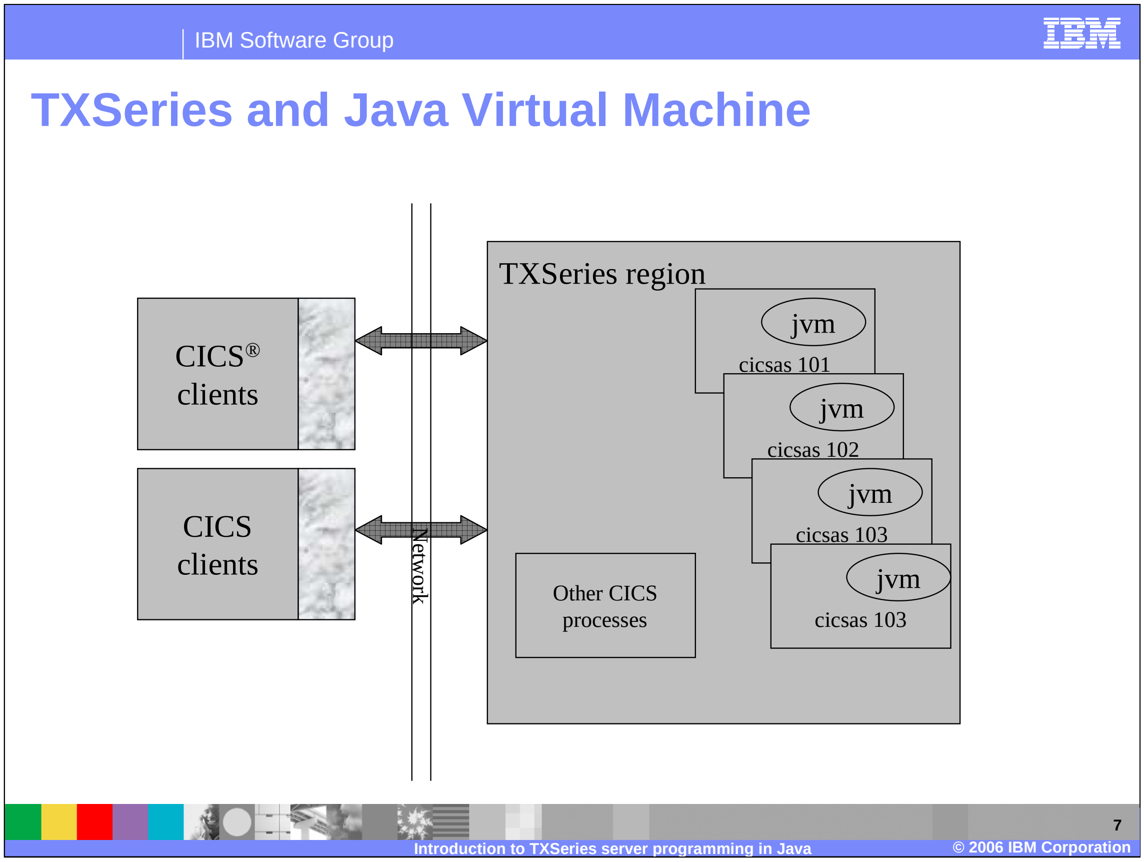
- Java language platform:
  - system-independent
  - object-oriented
  - designed to be portable and architecture neutral.
- Programmers use classes or objects instead of coding EXEC CICS statements to access a TXSeries service
- TXSeries provides various functional classes for programmers to perform relevant TXSeries functions

Today Java is perceived as a popular system-independent object-oriented language platform. The Java language is designed to be portable

and architecture-neutral, and the byte code generated by compilation requires a machine-specific interpreter for execution. TXSeries provides support to run Java server applications in a transactional environment and provides a range of facilities to access TXSeries services as objects in Java. Thus, Java programming skills can be effectively used in writing modern server applications.

Using Java for writing TXSeries server applications does not require programmers to code or to know about EXEC CICS statements. Instead, they

use *classes* or *objects* that are supplied by TXSeries to access a service. This method of accessing the services eliminates the need for translating the server Java program. TXSeries provides various functional classes for programmers to perform relevant TXSeries functions. In the next section, various functional classes will be discussed.



As with any system, it is important to understand how TXSeries runs your Java program in a transactional environment. TXSeries uses the Java Virtual Machine (JVM) runtime support to run your Java server applications. This section describes the fundamentals of how TXSeries uses the Java runtime to run your applications.

As illustrated in the picture, TXSeries initializes the JVM on the application server process that runs your Java server application. The JVM are

loaded and initialized only if a request comes to an application server that requires a Java server program to be run. Further requests to run a Java server program on the same application server process continue to use the JVM that was initialized earlier. If the request happens to go to another application server process, then a new JVM is loaded and initialized.

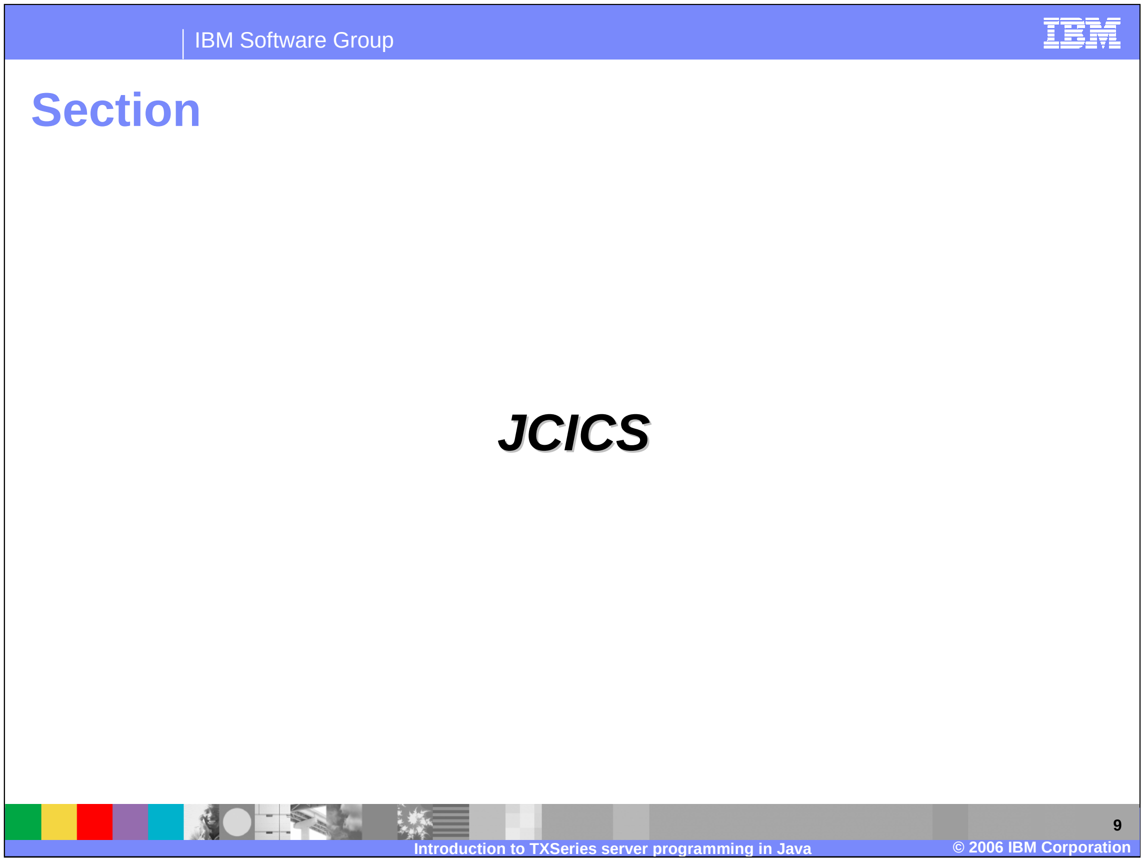
TXSeries supplies a language runtime for Java called *cicsprJAVA*. This runtime is responsible for loading and initializing the JVM and enabling Java server applications to run under the TXSeries environment. The cicsprJAVA runtime is dynamically loaded on to the application server process upon a request to run a Java server program. The runtime is not unloaded after the server program execution completes. Instead it is reused when another request comes to run a Java server program on the same application server process.

Performance considerations

- TXSeries does not cache Java server programs
  - Resident attribute of PD entry not applicable
  - Caching takes place within the JVM
- JVM are not shared across application server processes
  - JVM is initialized only once within a process
  - JVM is destroyed when the application server process terminates
  - Every application server has its own instance of JVM
  - Having low IdleTimeOut can affect performance

TXSeries does not cache Java server programs, which means that the program definitions attribute *Resident* is not applicable for Java server programs. The caching takes place within the JVM, as Java server programs are directly interpreted by the Java runtime. Because there are multiple application server processes running, there can be multiple JVMs initialized and used within the region, meaning that an application server process cannot share its JVM resources with another. So when a request to run a Java server program is made, an application server process initializes the Java runtime, loads the application classes, and runs them to completion. When the same request is made again, and the request is assigned to a different application server process, then the new application server process initializes a new JVM, loads the application classes, and runs them to completion. However, if the second request runs in the same application server as the first request, JVM is not loaded again, and therefore application classes might have been cached within the JVM.

Another factor that can affect the performance of the system is the use of the *IdleTimeout* region definitions (RD) parameter of the region. This is because as the *IdleTimeout* is reached for an application server process waiting for some work, the application server process is terminated, and its corresponding JVM is also terminated. This affects the performance because the JVM and the relevant application classes must be reloaded onto a new application server process.



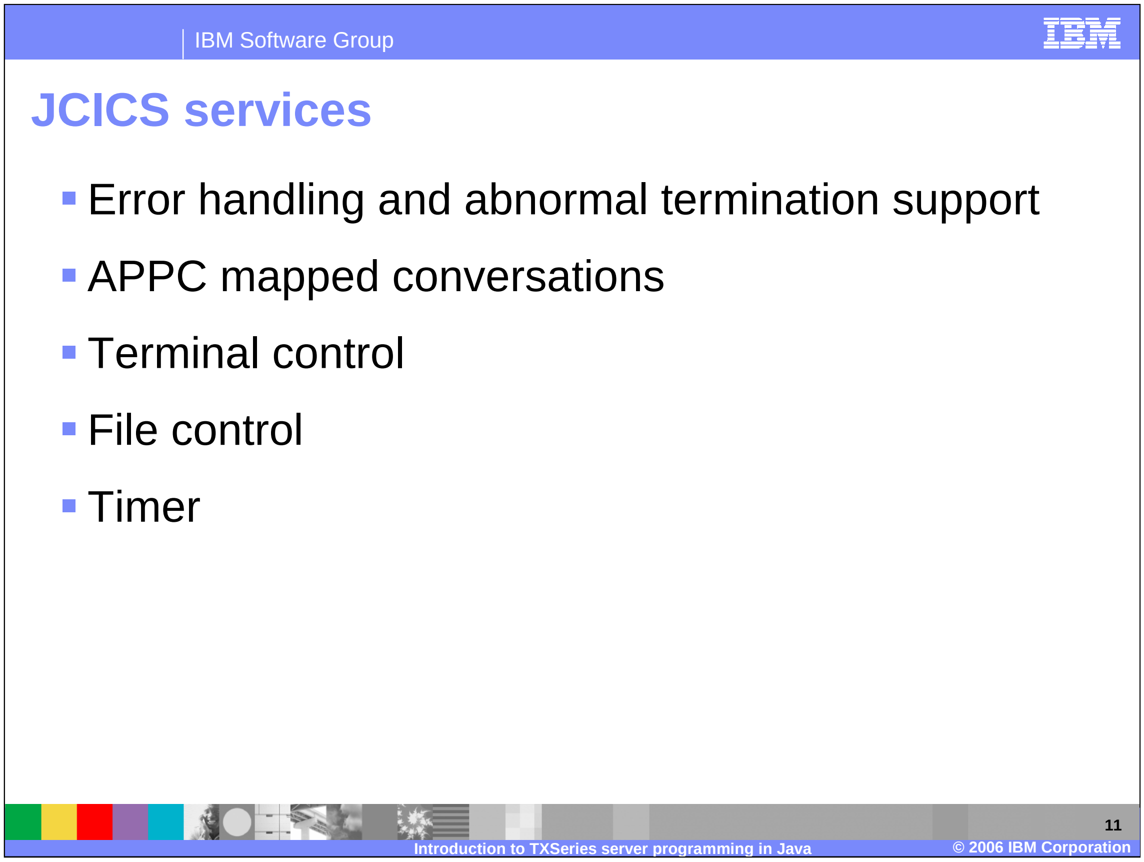
This section covers the concept of JCICS and the services offered by JCICS.

JCICS

1. JCICS is a fundamental programming element to write Java-based server applications in a TXSeries environment.
2. TXSeries supplies the JCICS application programming interfaces (API), which are basically Java classes that can be instantiated in a Java program to access a given service

JCICS is a fundamental programming element to write Java-based server applications in a CICS environment. TXSeries supplies the JCICS application

programming interfaces, or APIs, which are Java classes that can be instantiated in a Java program to access a given service.



Error handling and abnormal termination support services allow you to control the execution of the task, such as to cancel a task, issue a force abend to the class, and to handle conditions and abends that are thrown by TXSeries services. These services are located in the *Task* Java class.

Advanced Program-to-Program Communication mapped conversations provide a complete Java class to enable APPC-mapped conversations. These services are located in the *AttachInitiator*, *Conversation*, and *ConversationPrincipalFacility* Java classes.

Terminal control services are limited to only two APIs that are equivalent to SEND CONTROL and SEND TEXT CICS commands. These services are available in the Task Java class.

File control services provide APIs to access VSAM files. The supported file types are key-sequenced data set (KSDS), entry-sequenced data set (ESDS), and relative record data set (RRDS). These services are available in *File*, *KeyedFile*, *KSDS*, *ESDS*, and *RRDS* Java class.

Timer services provide a means to start or cancel tasks. Other time-related services such as ASKTIME, FORMATTIME, and DELAY are not supported. These services are available in the *StartRequest* Java class.

JCICS services (cont.)

- Program execution
- Logical unit of work
- Serialization
- Temporary storage queue (TSQ)
- Transient data queues (TDQ)

Program execution services enable you to call another program in the same logical unit of work. Data can be passed and received through

COMMAREA. These services are located in the *Program* class.

Logical unit of work services enable you to commit or roll back the task's work. These services are located in the Task class.

Serialization services provide the means to handle shared access to resources. These services are located in the *SynchronizationResource* class.

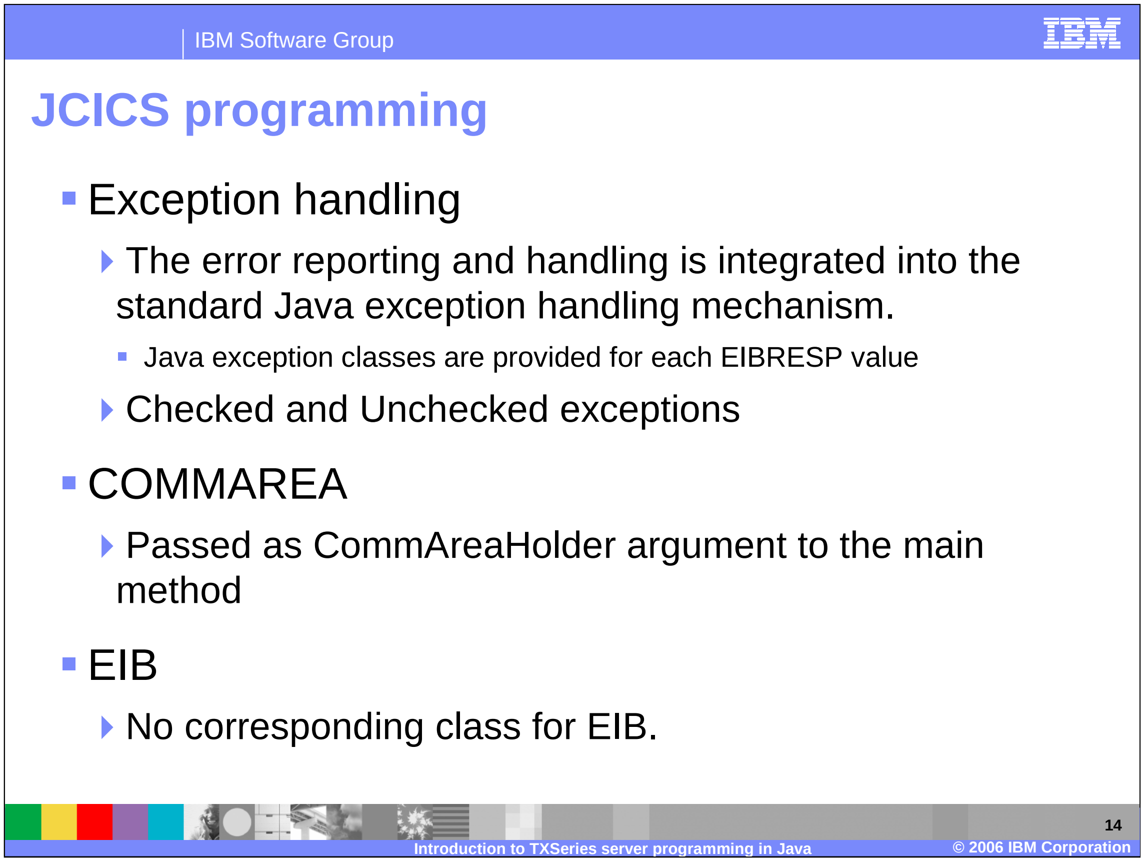
Temporary storage queue (TSQ) services provide the means to share data across multiple transactions. These services are located in the *TSQ* class.

Transient data queues (TDQ) services provide the ability to access transient data queues. These services are located in the *TDQ* class.

JCICS services – Not supported

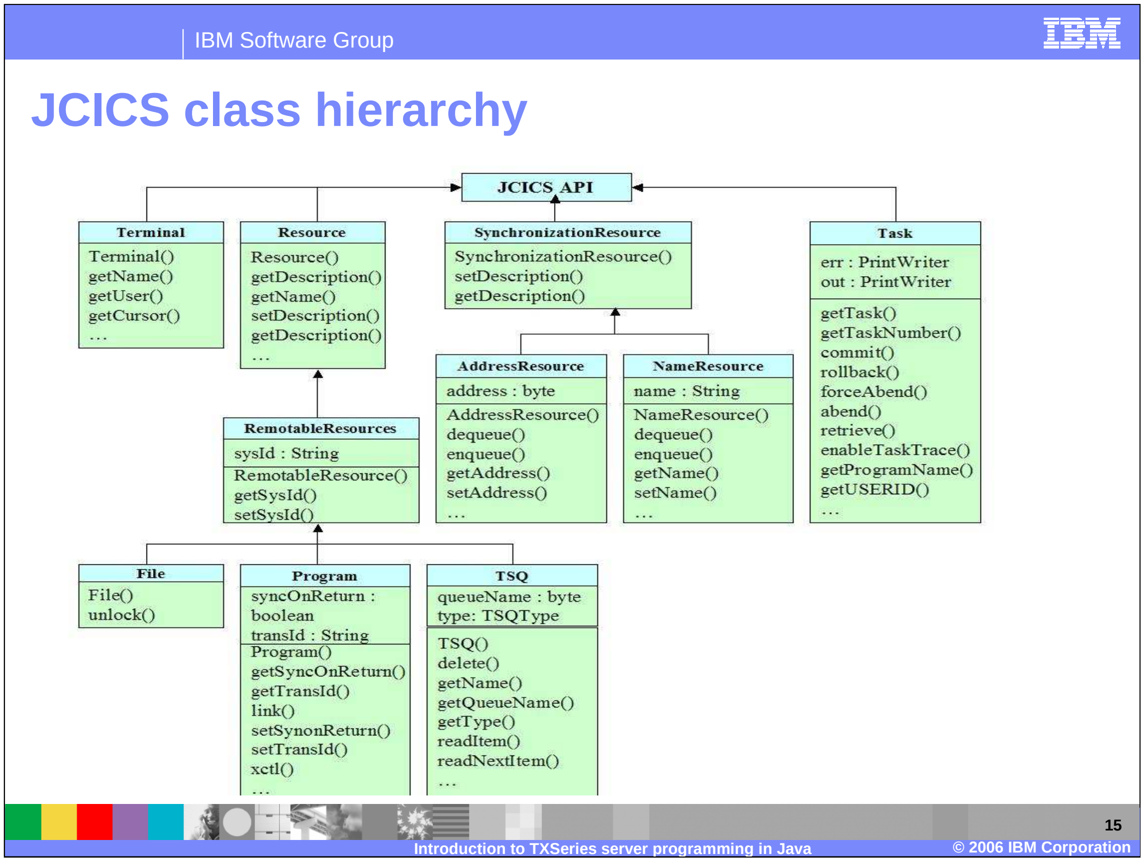
- DUMP
- JOURNAL
- Storage
- BMS
- APPC unmapped conversations
- Authentication
- Inquire services for:
  - Programs, Files, Journals and Statistics

The services listed on this slide are *not* supported by JCICS under TXSeries.



It is always necessary to handle error conditions in your code, as anything can go wrong within the application. The error reporting and handling in JCICS is integrated into the standard Java exception handling mechanism. In other languages, TXSeries indicates the success or failure of a CICS command through EXEC interface block (EIB) variables called *EIBRESP* and *EIBRESP2*. In Java, these EIBRESP codes are mapped to Java exception classes. For each EIBRESP value that can occur in CICS, there is one corresponding Java exception class. The exceptions are generally classified as *checked exceptions* and *unchecked exceptions*. When you call a method that can throw a checked exception, you are requested to either handle the exception in your method, or to declare your own method as throwing that exception. However, unchecked exceptions need not necessarily be handled or declared. They usually represent conditions that are so common (or so rare) that it is impractical to be forced to handle them.

In CICS, there are two primitive structures that are used: COMMAREA and EXEC interface block, or EIB. The COMMAREA is automatically passed into a program using the CommAreaHolder argument to the main method, and there is no class that can be instantiated to access the EIB block. However, certain fields such as Transaction ID (eibtrnid) and Task Number (eibtaskn) can be accessed through Task.getTransactionName() and Task.getTaskNumber() respectively.



The Figure in this slide illustrates the JCICS class hierarchy.

Summary

- Provide an overview of object-oriented programming under TXSeries
- Understand the behavior of Java virtual machine under TXSeries
- Performance considerations
- Provide an overview of JCICS services
- JCICS class hierarchy

In summary, this presentation covered TXSeries programming in Java, including the JVM, performance considerations, and the JCICS services and class hierarchy.

Trademarks, copyrights, and disclaimers