Estimated time: 45 minutes

WebSphere eXtreme Scale:
Introduction to ObjectGrid

What this exercise is about

The objective of this lab is to help you understand how to access and work with data that is stored in an ObjectGrid, which is one of the new technologies provided with WebSphere Extended Deployment version 6.1.

Lab requirements

- ObjectGrid technology does not require a WebSphere Extended Deployment or WebSphere Application Server environment to run. It can run in any J2SE 1.4.2 or higher environment. The WebSphere eXtreme Scale installer places the necessary JAR files in <WAS_HOME>/optionalLibraries/ObjectGrid and <WAS_HOME>/lib when installed as an extension to WebSphere Application Server or <WXS_HOME>/ObjectGrid when installed stand-alone.
    - The instructions in this lab exercise use the Eclipse IDE for Java Developers release 3.3 as a development environment for the sample code. The lab can be completed using any Java IDE, such as Rational Application Developer. The steps required may vary slightly in some places, but the overall process is the same. You must make sure that your IDE is configured to run your project using a 1.4.2 or higher JVM.

      The examples can also be run from the command line under any Java 1.4.2 or higher JVM. You must ensure that objectgrid.jar and objectgridSamples.jar are in the Java classpath.

What you should be able to do

At the end of this lab you should be able to:

- Programmatically create an ObjectGrid instance
- Configure the ObjectGrid instance programmatically or based on XML files
- Understand the basic methods for accessing ObjectGrid data
- Understand the different copy modes available on an ObjectGrid

Introduction

ObjectGrid technology implements a high performance, transactional cache that can be accessed by applications as a Map-like object. This data can be updated in real-time and provides very high performance to applications.

In this lab exercise you will first create a simple ObjectGrid and learn the basics of accessing and modifying data stored in the grid using Map-like methods. You will then explore a more comprehensive sample application that is provided with WebSphere Extended Deployment. This sample will illustrate how ObjectGrids can be created and associated with plug-in classes (such as custom data Loaders or Evictors) using XML configuration files.

Finally, this lab refers to several other sample applications that are provided with WebSphere Extended Deployment. These samples are extensively documented in their source code and provide examples of many other ObjectGrid concepts, including the different copy modes, MapEventListeners, and accessing data from a separate process.

Exercise instructions

Some instructions in this lab are Windows® operating-system specific. If you plan on running the lab on an operating-system other than Windows, you will need to run the appropriate commands, and use appropriate files (.sh or .bat) for your operating system. The directory locations are specified in the lab instructions using symbolic references, as follows:

Reference Variable	Windows Location	AIX® or UNIX® Location
<WXS_HOME>	C:\WebSphere\eXtremeScale	/usr/WebSphere/eXtremeScale /opt/WebSphere/eXtremeScale
<LAB_FILES>	C:\LabfilesXD	/tmp/LabfilesXD
<TEMP>	C:\temp	/tmp

Note for Windows users: When directory locations are passed as parameters to a Java program such as EJBdeploy or wsadmin, it is necessary to replace the backslashes with forward slashes to follow the Java convention. For example, replace C:\LabFilesXD\ with C:/LabFilesXD/

Part 1: Environment setup

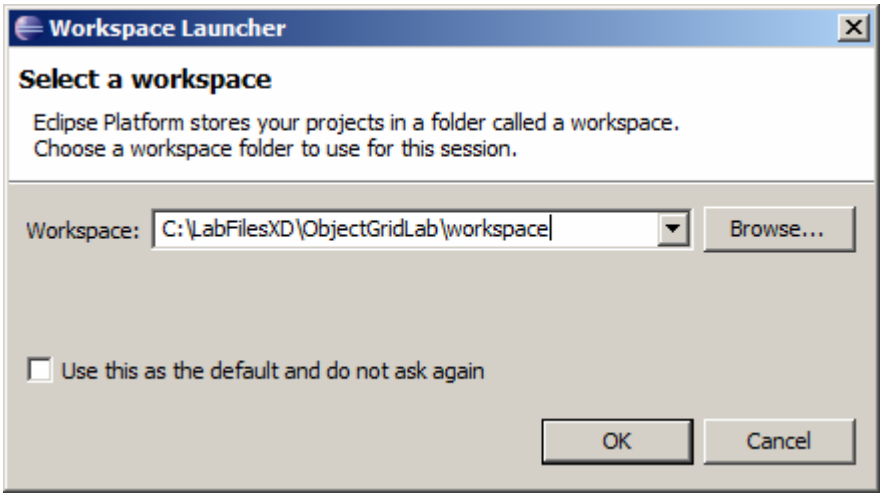
In this section, you will import the ObjectGrid sample code that is provided with WebSphere Extended Deployment into an Eclipse project, and make the stand-alone ObjectGrid runtime libraries available to that project, so that the sample applications can be run directly from the Eclipse interface.

____ 1. Create a new workspace in Eclipse

__ a. Launch Rational Application Developer V7.

1) From a Windows Command line, type "**C:\eclipse\eclipse.exe**"

__ b. When prompted to select a workspace, enter **C:\LabFilesXD\ObjectGridLab\workspace.**



__ c. Click **OK**.

__ d. Close the **Welcome** tab to go to the workbench.

____ 2. Create a new Java project in Rational Application Developer.

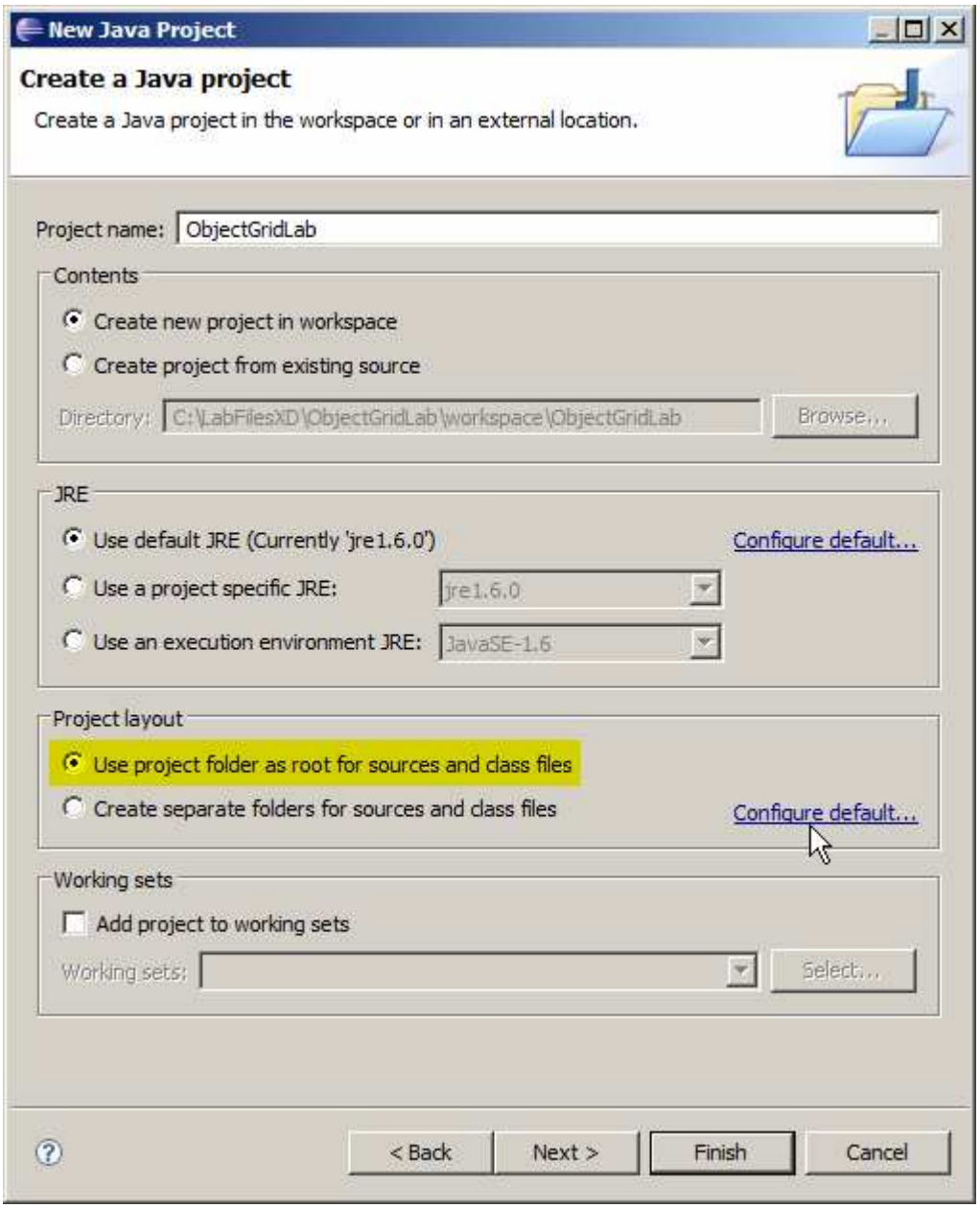
__ a. From the menu, select **File → New → Project**…

__ b. In the New Project window, expand **Java**, and then highlight **Java Project**.

__ c. Click **Next.**

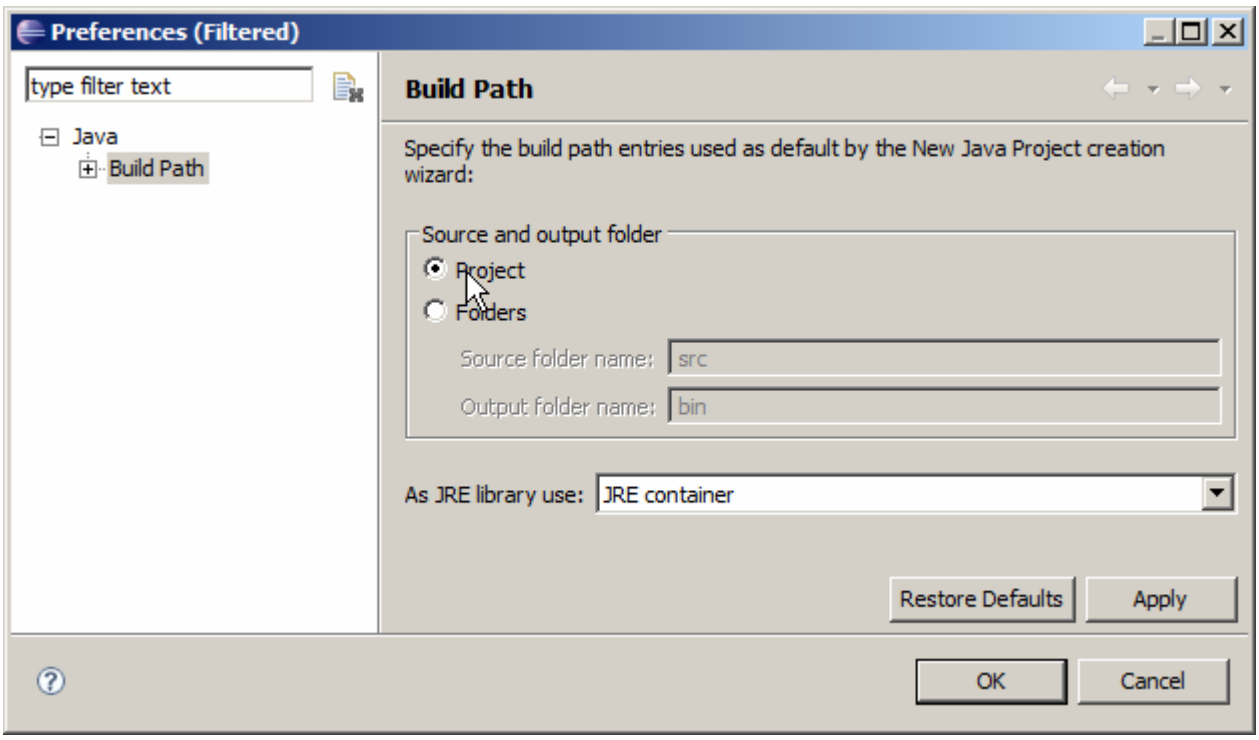
__ d. In the Create New Java Project window, name the project **ObjectGridLab**

__ e. Ensure **"Use project folder as root for sources and class files"** is selected under Project layout



__ f. Click **Configure default…** under **Project layout**

__ g. Verify that the **Project** radio button is selected, as shown below. If it is not, select it.



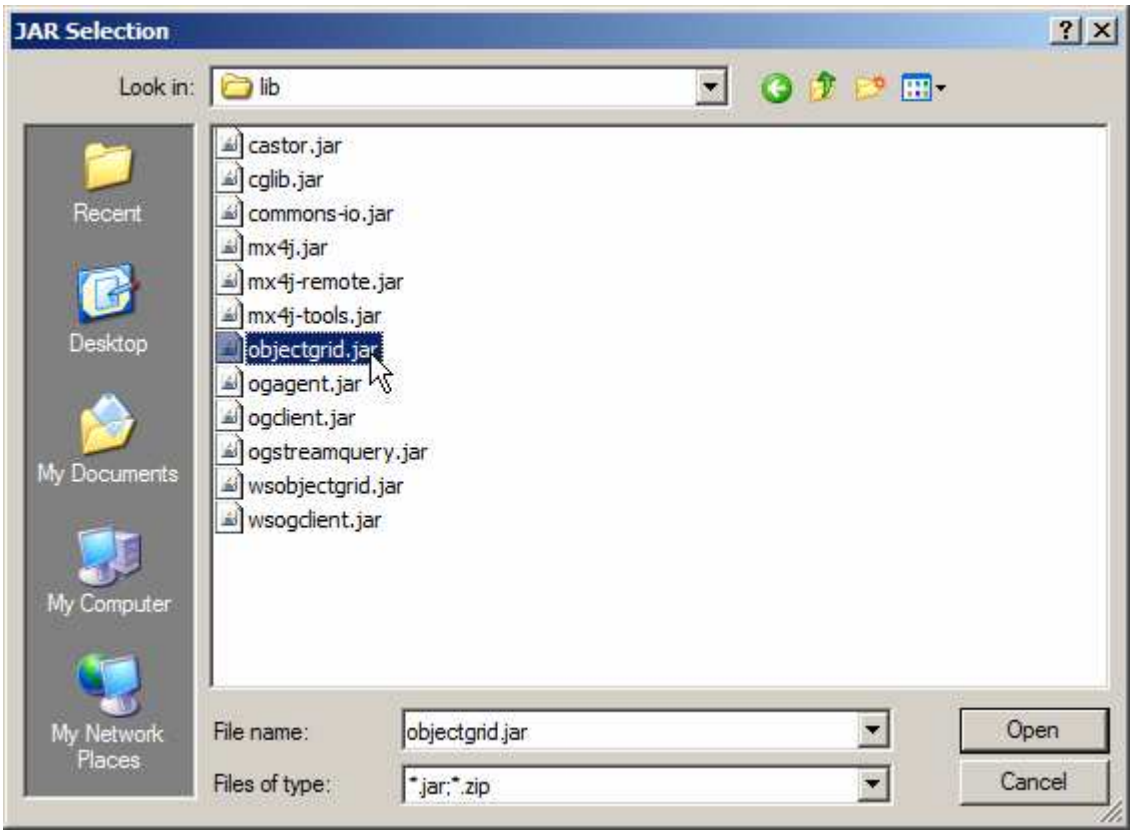
__ h. Click **OK** to close the Preferences window.

__ i. Back in the New Java Project window, click **Next**.

Object Grid

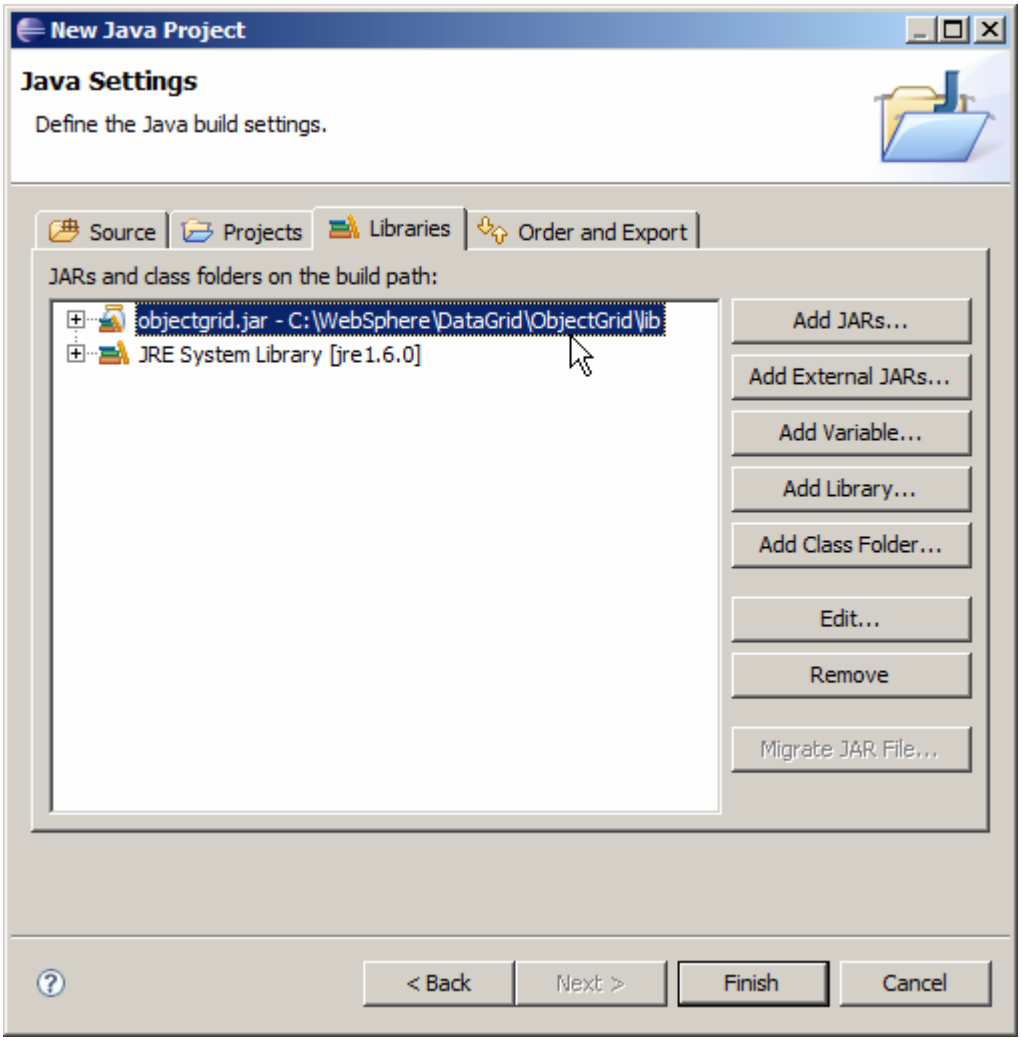
__ j. Click the **Libraries** tab and then click **Add External JARs…**

__ k. In the JAR Selection window, browse to **<WXS_HOME>\ObjectGrid\lib** and highlight **objectgrid.jar**, as shown below.



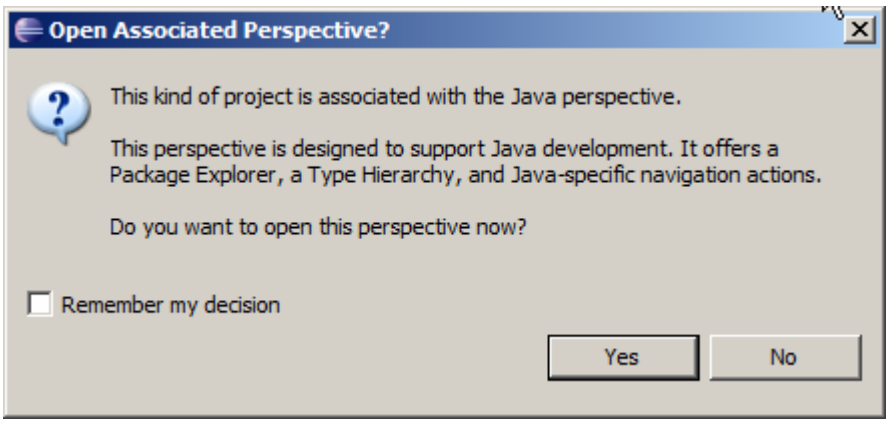
__ l. Click **Open**.

__ m. Verify that **objectgrid.jar** now appears under the Libraries tab, as shown below.



__ n. Click **Finish**.

__ o. You may be presented with a window asking if you want to switch to the Java Perspective. If asked, click **Yes**.



Object Grid

____ 3. Import sample programs.

__ a. Right click on **ObjectGridLab** and select **Import** …

__ b. In the "Select an import source" window, expand **General** and select **Archive file** and click **Next.**

__ **c.** Browse to **<WXS_HOME>\ObjectGrid\samples\objectgridSamples.jar**

__ d. Click "Open"

__ e. Click "Finish"

> Depending on which version of the JDK you are using with Eclipse, some of the samples may have annotation or other errors or warnings in the Problems pane of the IDE. For this lab you can safely ignore these errors.

Part 2: Creating a simple ObjectGrid application

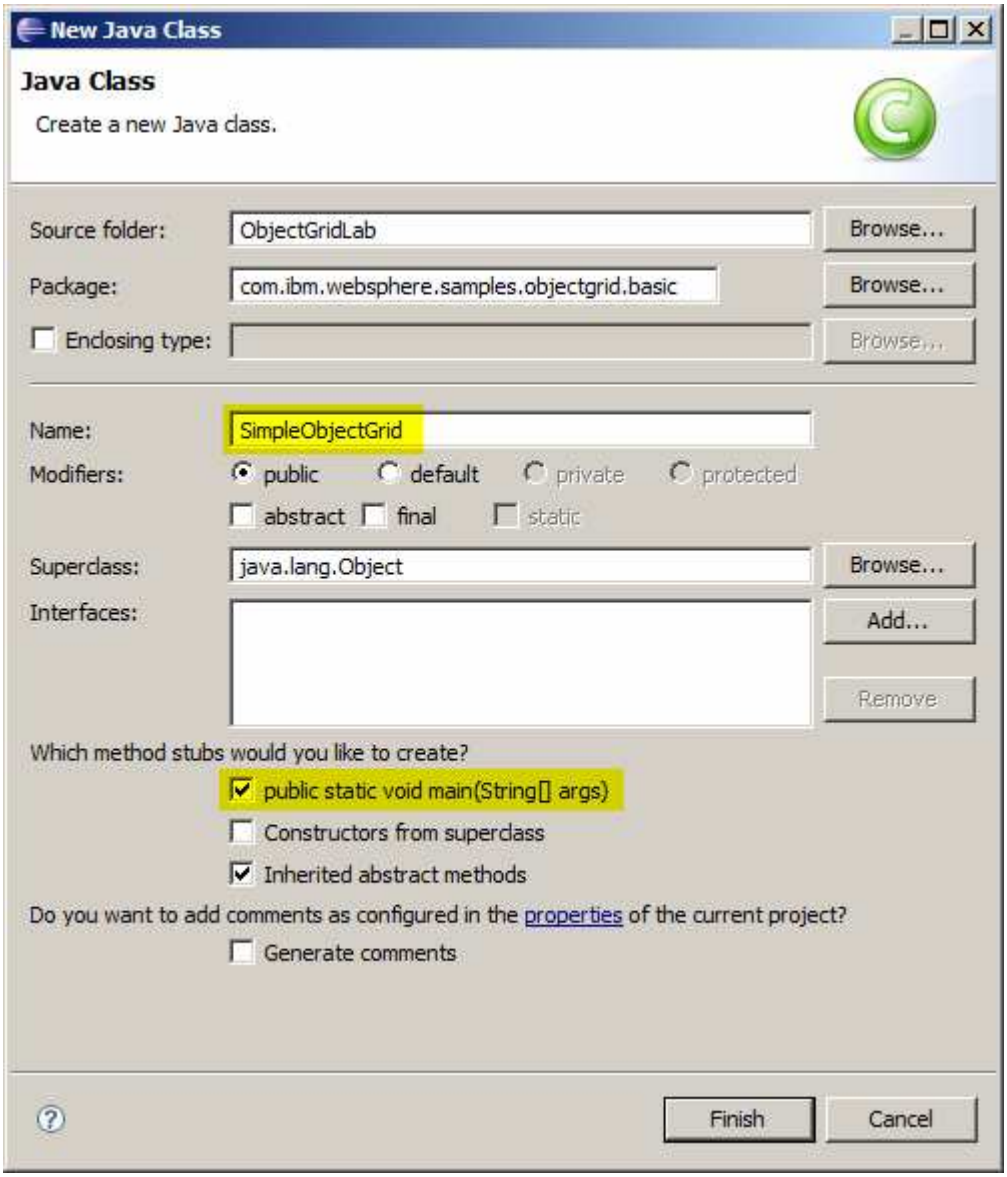
In this section, you will create a simple Java application that demonstrates how to instantiate an ObjectGrid, and how the Maps stored in an Object grid support all of the expected Map-like functions in a transactional fashion.

____ 1. Create a new Java class named **SimpleObjectGrid**.

__ a. In the Package Explorer pane of Rational Application Developer, expand **ObjectGridLab** and right-click on the package **com.ibm.websphere.samples.objectgrid.basic**. Then select **New → Class**.

__ b. In the New Java Class window, enter the name **SimpleObjectGrid**, as shown below.

__ c. Check the **public static void main(String[] args)** check box.



Object Grid

__ d. Click **Finish**.

____ 2. Import the package com.ibm.websphere.objectgrid.*, so that your code will have access to the ObjectGrid data types.

__ a. Find the line that reads `package com.ibm.websphere.samples.objectgrid.basic;`

__ b. Type this line below it:

```
import com.ibm.websphere.objectgrid.*;
```

____ 3. Edit the main() method so that it can throw an Exception. For the purposes of this example, this will keep you from having to worry about exception handling.

__ a. Find the line that reads `public static void main(String[] args) {`

__ b. Change it to read `public static void main(String[] args) throws Exception {`

____ 4. Inside the main() method, initialize a new ObjectGrid, using an ObjectGridManager.

__ a. Type this line of code inside the main() method:

```
ObjectGridManager OGMgr = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = OGMgr.createObjectGrid("testGrid");
```

____ 5. Next, define a map to hold your data. This map will hold (key, value) pairs of objects, just like any other map-like object. An ObjectGrid can contain many maps, but in this case you will define only one. A Map implementation can define a Loader that specifies a hardened store for the data in the map, or an Evictor, which defines an eviction policy for the map. This default map defines neither. The initialize() method indicates that configuration is complete, and the ObjectGrid is ready to use.

__ a. Type this line of code below the previous line:

```
og.defineMap("testMap");
og.initialize();
```

____ 6. In order to access data in the Object Grid, you must create a session. A session gives the Object Grid its transactional nature, with begin(), rollback(), and commit() methods. Sessions are not thread safe, and should be accessed by only one thread at a time.

__ a. Enter this line of code beneath the previous line:

```
Session sess = og.getSession();
```

____ 7. You can then get a reference to the Map within the scope of the session. ObjectMaps are also not thread safe, and each thread must get its own map.

__ a. Enter the line of code below the previous line to obtain a reference to the ObjectMap:

```
ObjectMap testMap = sess.getMap("testMap");
```

____ 8. Now that you have access to the Map, you can insert data into it. In this case, you will insert a string that represents a name. Note that any changes you make to the map are not written into the data structure until you call the commit() method.

__ a. Enter this block of code below the previous line:

```
System.out.println("Putting the string \"John Doe\" into the map,
using keyword \"name\".");
sess.begin();
testMap.insert("name", "John Doe");
sess.commit();
```


____ 9. Now that there is data in the map, write a block of code that will access the object, print out its value. By using the getForUpdate() method, you are creating a write-lock on the object, indicating that you intend to change it, and other threads cannot change the object until you release the lock.

__ a. Enter this block of code below the previous block:

```
sess.begin();
String person = (String) testMap.getForUpdate("name");


System.out.println("Value retrieved from the map: " + person);
System.out.println("Changing John\'s last name, and updating the
map");
System.out.println("...");
```

____ 10. Now change the last name in the string object from “Doe” to “Cartwright”, and put the object back into the map.

__ a. Enter this block of code below the previous block:

```
person = "John Cartwright";
testMap.put("name", person);
sess.commit();
```

____ 11. Finally, write code that will get the value from the map once again, and print it, showing that the name has in fact been changed in the map itself after being committed.

__ a. Enter these lines of code below the previous lines:

```
String newPerson = (String) testMap.get("name");
System.out.println("Retrieved value from the updated map: " +
newPerson);
```


Object Grid

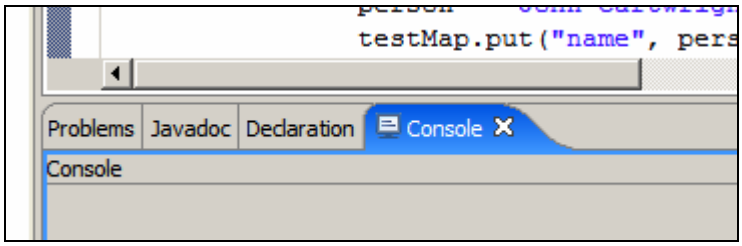
Note: The program you have just written creates a basic ObjectGrid, and illustrates how you can access the grid in the scope of a transaction.

____ 12. Save your new source code.

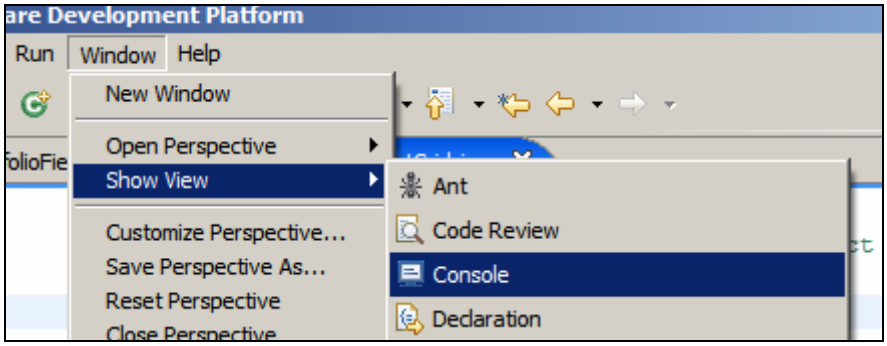
__ a. Select **Save** from the **File** menu, or click the “Save” icon (which looks like a floppy disk) in the toolbar.

__ b. Correct any errors that may be identified, then Save again.

____ 13. If the Console view does not appear in the lower pane:



__ a. From the menu bar, select **Window → Show View → Console**.



____ 14. Run your program from within Rational Application Developer and verify the output.

__ a. In the Project Explorer pane, right-click on **SimpleObjectGrid.java**, and select **Run As → Java Application**

__ b. You may see a dialog indicating “Errors exist in a required project.” Click Proceed to allow the program to run.

__ c. If there are no errors in your code, you will see the output of your program displayed in the Console area in the lower right of the window. Verify that you see this output, showing that your code was able to access and manipulate the data in the map:

```
Putting the string "John Doe" into the map, using keyword "name".
Value retrieved from the map: John Doe
Changing John's last name, and updating the map
...
Retrieved value from the updated map: John Cartwright
```

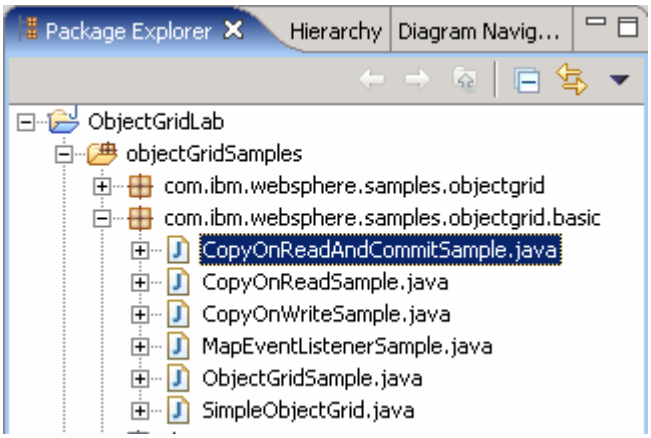
Note: if your code does not run correctly, or the IDE displays errors that you cannot solve, a correct version of this program can be found in C:\LabFilesXD\ObjectGridLab\solution. You can import this file into your workspace, or copy and paste its contents into your existing source file.

Part 3: Understanding an application that uses ObjectGrid

In this section you will examine CopyOnReadAndCommitSample.java, one of the samples provided with WebSphere Extended Deployment, which demonstrates how to create an ObjectGrid based on an XML configuration file.

____ 1. Open the first sample application, CopyOnReadAndCommitSample.java

__ a. In the Package Explorer (far left), expand **ObjectGridLab → com.ibm.websphere.samples.objectgrid.basic**, as shown below:



__ b. Double-click **CopyOnReadAndCommitSample.java**

____ 2. Explore the sample code, and notice that it is creating an ObjectGrid based on an XML configuration file.

__ a. At the bottom of the file, find the **main()** method, which will be called when you run this class.

1) Note that it is calling the **initialize()** method, and passing it the path of an XML file.

__ b. Scroll about halfway up the file, and note that the **initialize()** method is creating an ObjectGrid by calling the **ObjectGridManager.createObjectGrid()** method. This method reads an XML file, and returns the named ObjectGrid, as defined by the XML file.

____ 3. Open this XML file, sample1.xml.

__ a. In the Project Explorer pane, expand **META-INF**

__ b. Double-click on **sample1.xml**.

____ 4. Examine the file, to see how an ObjectGrid is defined.

__ a. Locate this section:

```
<ObjectGrid name="copyonreadandcommit">
      <bean id="TransactionCallback"
      className="com.ibm.websphere.samples.objectgrid.HeapTransaction
      Callback" />
      <BackingMap name="employees" readOnly=”false”
      plugin-collection-ref="employees" preloadMode="false"
      lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" />
</ObjectGrid>
```


Object Grid

__ b. Note that this section defines an ObjectGrid named “copyonreadandcommit”, and specifies a backingMap named “employees”. It also specifies that the grid will use an optimistic locking strategy and the COPY_ON_READ_AND_COMMIT copy mode. This copy mode specifies that a new copy of the object will be created for every read and every commit action. This is the safest mode of operation.

__ c. Next locate this section in the file:

```
<backingMapPluginCollection id="employees">
    <bean id="Loader" className=
              "com.ibm.websphere.samples.objectgrid.HeapCacheLoader">
       <property name="preLoadClassName" type="java.lang.String"
       value="com.ibm.websphere.samples.objectgrid.EmployeePreload"
       description="name of class that implements HeapPreloadData
       interface" />
    </bean>
    <bean id="ObjectTransformer" className=
          "com.ibm.websphere.samples.objectgrid.
          EmployeeObjectTransformer" />
    <bean id="OptimisticCallback"
    className="com.ibm.websphere.samples.objectgrid.EmployeeOptimisti
    cCallbackImpl" />
</backingMapPluginCollection>
```

__ d. Note that this section defines a set of BackingMap plug-ins named “employees”. This collection defines a Loader implementation class, and ObjectTransformer and OptimisticCallback implementations.

__ e. Note also that the earlier definition of the ObjectMap has an attribute (**plugin-collection-ref**) that associates it with this plug-in collection, meaning that each of these plug-in classes is associated with the grid named “copyonreadandcommit”.

____ 5. Further examine the source code of CopyOnReadAndCommitSample.java, to gain an understanding of how the COPY_ON_READ_AND_COMMIT mode works. It ensures that the application never has a direct reference to the actual value object in the map, but instead always works with a copy.

__ a. Locate this block of code inside the **showCopyOccursOnCommit()** method. It retrieves an employee record from the map, and prints data about the employee.

```
Employee emp = (Employee) map.get( key );
System.out.println("");
System.out.println("dump of employee 5 prior to manager change: " +
          emp.dumpSelf() );
```

__ b. Note that this block of code immediately below it then changes the employee’s manager, and commits the change to the map. It also prints the new contents of the employee object to the console:

```
emp.setManagerNumber(2);
map.put( key, emp );
// Commit the transaction.
session.commit();
System.out.println(
     "dump of employee number 5 after changing manager to 2: " +
     emp.dumpSelf() );
```

__ c. What happens next is the important part. Locate the next line of code, which changes the manager attribute of the “**emp**” object that was retrieved from the map back to 5:

```
emp.setManagerNumber(5);
```

NOTE: Because the COPY_ON_READ_AND_COMMIT mode is being used, a copy of the "emp" object was made when the commit() method was called. This means that while the manager number of the "emp" object was just changed by the last line of code, the manager number of the corresponding object stored in the map is unchanged, since they are in fact different objects.

__ d. Locate the line of code below, which illustrates the behavior explained by the above note. When this value is printed to the console, you will see that the value of daffy.dumpSelf() indicates that the manager number is still 2, proving that a copy was made at the time of the commit().

```
session.begin();
Employee daffy = (Employee) map.get( key );
session.commit();
System.out.println( "Notice manager number is still 2 since \
     "COPY_ON_READ_AND_COMMIT is used: " + daffy.dumpSelf() );
```

____ 6. Run the CopyOnReadAndCommitSample application to see its behavior demonstrated.

__ a. In the Project Explorer pane, right-click on **CopyOnReadAndCommitSample.java** and select **Run As→Java Application** from the menu. If you see a dialog indicating "Errors exist in a required project." Click **Proceed** to allow the program to run.

__ b. Note this output that appears in the Console panel, illustrating the behavior that was just described, that is, the value in the map is unchanged by an action on the copy of it that was created:

```
resourcePath: META-INF/sample1.xml
objectgridUrl:
file:/C:/LabFilesXD/ObjectGridLab/workspace/ObjectGridLab/META-
INF/sample1.xml

the following maps are defined in the copyonreadandcommit instance:
[employees]
EmployeeOptimisticCallback returning version object for employee = Perry
Cheng, version = 0
EmployeeOptimisticCallback returning version object for employee = Hao Lee,
version = 0
EmployeeOptimisticCallback returning version object for employee = Ken
Huang, version = 0
EmployeeOptimisticCallback returning version object for employee = Jerry
Anderson, version = 0
EmployeeOptimisticCallback returning version object for employee = Kevin
Bockhold, version = 0
EmployeeOptimisticCallback returning version object for employee = Kevin
Bockhold, version = 0

dump of employee 5 prior to manager change: employee name = Kevin Bockhold
department = ABC   employee id = 5   manager id = 1   optimistic seqno = 0
EmployeeOptimisticCallback returning version object for employee = Kevin
Bockhold, version = 0
EmployeeOptimisticCallback updating employee = Kevin Bockhold with new
version = 1
EmployeeOptimisticCallback returning version object for employee = Kevin
Bockhold, version = 1
dump of employee number 5 after changing manager to 2: employee name =
Kevin Bockhold   department = ABC   employee id = 5   manager id = 2
optimistic seqno = 1
```


Object Grid

```
calling setManagerNumber(5) after commit has occured.
EmployeeOptimisticCallback returning version object for employee = Kevin
Bockhold, version = 1

Notice manager number is still 2 since COPY_ON_READ_AND_COMMIT is used:
employee name = Kevin Bockhold   department = ABC   employee id = 5
manager id = 2   optimistic seqno = 1
```

Part 4: Further exploring the ObjectGrid samples

The com.ibm.websphere.samples.objectgrid.basic package that you imported into your workspace contains several more examples. While this exercise will not discuss any of them in detail, you are encouraged to explore them on your own, to gain a better understanding of the ObjectGrid API. The source code of each of the samples is well documented, and will guide you in understanding how the sample works.

You can also find information about each of these samples in **SamplesGuide.htm**, which you can find in the **doc** directory of the extracted ObjectGridSamples.jar file.

____ 1. **CopyOnReadSample.java** illustrates the COPY_ON_READ copy mode. In this mode, a copy is made when an object is read from the map, but unlike the previous example, a copy is not made when the commit() method is called. While this mode performs better than the previous mode, you must ensure that a thread never uses a reference to the object after it is committed, or data integrity will be compromised,

____ 2. **CopyOnWriteSample.java** is a slightly more complicated sample that describes how to use the COPY_ON_WRITE copy mode. This mode offers performance that can be better than the previous two modes, but also offers the data integrity of the COPY_ON_READ. To accomplish this, the application must always use a pre-defined interface to access the value object in the map, rather than a concrete implementation class. This allows the ObjectGrid to return a dynamic proxy to the object, rather than a direct reference to the object, and copy the object only when the data changes. Based on the typical frequency of reads compared to writes, this mode will often perform better, since it minimizes the number of copies made.

____ 3. **MapEventListenerSample.java** demonstrates how you can implement the com.ibm.websphere.objectgrid.MapEventListener interface to enable your application to receive notifications of certain events, like the completion of pre-loading data into the map, or the eviction of an object. Also of interest, the XML file used in this sample (**sample4.xml**) specifies that data should be pre-loaded into the Map asynchronously, so that the application can potentially access the Map before all of the data has been loaded into it. This sample demonstrates the benefit of a MapEventListener by demonstrating how an application can wait for data pre-loading to complete before accessing the map.

____ 4. **ObjectGridSample.java** is the most comprehensive example, and exemplifies the most functionality. It uses **objectgrid-definition.xml** to create several maps and highlights many ObjectGrid concepts, including Loaders, TransactionCallback, OptimisticCallback, and ObjectTransformers. It also shows how to work with data within the maps. This code is the basis of the ObjectGridSample.ear application, which you can find in the installableApps directory and deploy to your server.

Reminder: the source code for each of these examples contains a wealth of information in the comments. Look there to understand how each of these additional samples works, and to gain a better understanding of the ObjectGrid concepts that they demonstrate.

Object Grid

What you did in this exercise

In this lab exercise you were exposed to various features and capabilities of the ObjectGrid API through writing and exploring source code.

In **Part 1**, you configured Rational Application Developer for use as the runtime environment for the sample applications. You may have also configured Eclipse or a command-line Java Runtime Environment instead.

In **Part 2**, you created a simple Java class that demonstrated the basics of creating and working with an ObjectGrid. In addition to creating the grid, you learned how sessions are used to give the grid a transactional nature, and how the data can be accessed and manipulated using standard Map-like methods.

In **Part 3**, you explored the CopyOnReadAndCommit.java sample file that is provided with WebSphere Extended Deployment. You learned how it relies on an XML configuration file to define how the ObjectGrid should be created and what plug-in classes should be associated with it. You also learned that the default copy mode, COPY_ON_READ_AND_COMMIT is the safest copy mode, because it ensures that an application never has a direct reference to a value object in the ObjectGrid.

In **Part 4**, you explored some of the other sample code that is provided with WebSphere Extended Deployment. These other samples highlighted the other copy modes, described how to use the MapEventListener interface to receive notifications of Map events, and described using many of the extensible features of ObjectGrid.

Solution instructions

If you were unable to create the Java application in Part 2, a completed version of the code is available in C:\LabFilesXD\ObjectGridLab\solution\SimpleObjectGrid.java