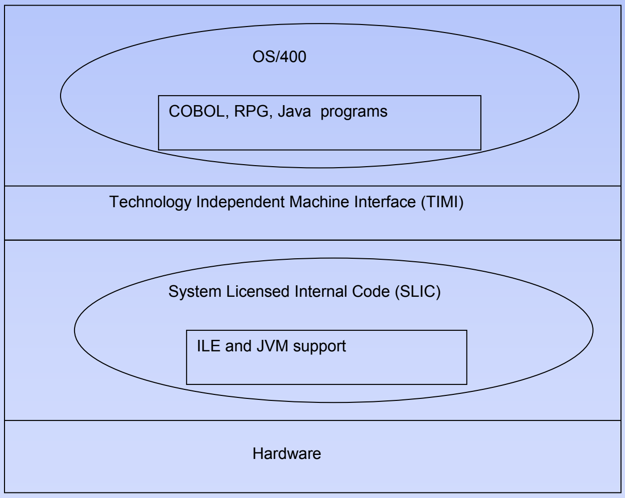
WebSphere Development Studio Client for iSeries: Creating Java Applications

Diane Wolff
and
Larry Riggins
Virginia Western Community College

Java on the iSeries


OS/400
COBOL, RPG, Java programs
Technology Independent Machine Interface (TIMI)
System Licensed Internal Code (SLIC)
ILE and JVM support
Hardware

IBM WebSphere Development Studio for iSeries (WDSc)

- In OS/400 V5R1, many of the programs for application development were bundled together
  - previously sold separately
- New package, WebSphere Development Studio for iSeries (WDSc), consolidates all key iSeries development tools for host and workstation

IBM WebSphere Development Studio for iSeries (WDSc)

- Includes:
  - WebSphere Studio – used for Web development
  - VisualAge for Java – the IBM Java development platform
  - CODE/400
  - VisualAge RPG
  - IBM Distributed debugger

IBM WebSphere Development Studio for iSeries (WDSc)

- Tools were integrated, updated, and in some tools totally redone
- The Java tool received major changes
  - now using Eclipse -- the “accepted” open source development tool as its basis
- CODE/400 became simply CODE (CoOperative Development Environment)

IBM WebSphere Development Studio for iSeries (WDSc)

- The WebSphere test environment for Web design was enhanced
- Enhancements were made in RPG IV, ILE C/C++, and COBOL
- Major parts of the ADTS (Application Development Tool Set) were integrated

IBM WebSphere Development Studio for iSeries (WDSc)

- New technologies were added
  - IBM WebFacing tool – a wizard to give legacy applications a GUI Java-based Web front
  - Struts – standard Java tools for code re-use
  - New Web component wizards
  - Web Services
  - XML
- Integration was the key!!!

Virginia Western COMMUNITY COLLEGE

IBM WebSphere Development Studio for iSeries (WDSc)

- The Advanced Edition adds support for
  - Enterprise Java Beans
  - J2EE
  - Test cases
  - The portal toolkit
- NOTE: There is a related version WebSphere Device Developer for developing Java apps for handhelds

Virginia Western COMMUNITY COLLEGE

WDSc Packaging

WebSphere Development Studio Advanced

C/C++ COBOL RPG ADTS

WebSphere Development Studio Client for iSeries

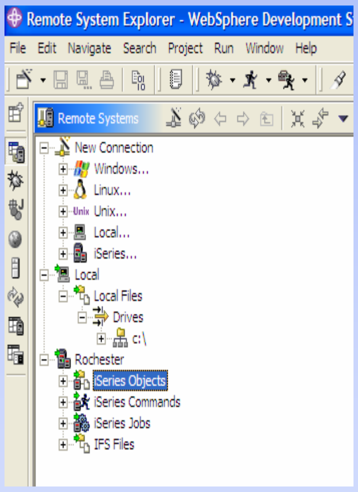
- Remote System Explorer
- iSeries Projects
- Web Tools for iSeries
- Java Tools for iSeries
- IBM WebFacing Tool
- Classical Tools (VARPG, CODE)
- WebSphere Studio Site Developer or Application Developer

Eclipse is Based on Perspectives

This is the Remote Systems Explorer perspective

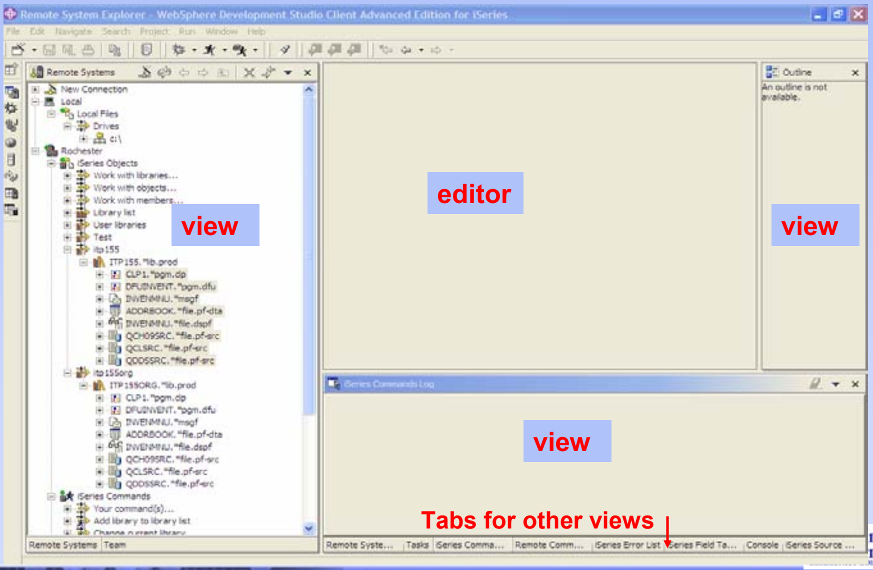
Remote Systems Explorer perspective →
Debug perspective →
Java perspective →
Web perspective →
Server perspective →
Install/update perspective →
iSeries projects perspective →
WebFacing perspective →

Click on the one you want. Add new ones with Window → Open perspective



Perspectives are created using multiple views

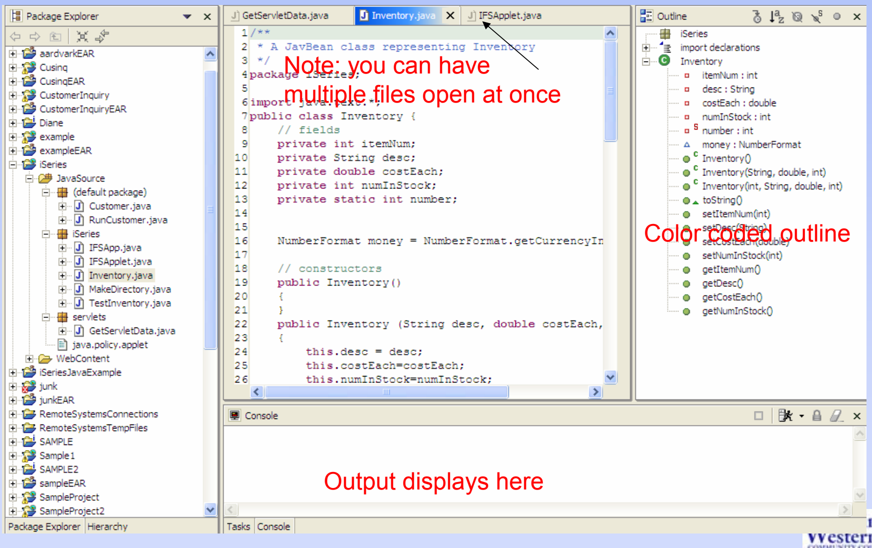
(different views for different perspectives)


Remote System Explorer - WebSphere Development Studio Client Advanced Edition for iSeries
File Edit Navigate Search Project Run Window Help
Remote Systems
Outline
An outline is not available.
iSeries Commands Log
view
editor
view
view
Tabs for other views
Remote Systems Team
Remote Syste... Tasks iSeries Comma... Remote Comm... iSeries Error List iSeries Field Ta... Console iSeries Source ...

Views in the Perspective

- WDSc provides default set of views and layout for any given perspective
- Rearrange, add, or close certain views to customize the arrangement
- We will need the perspectives in the next set of slides.

Java Perspective



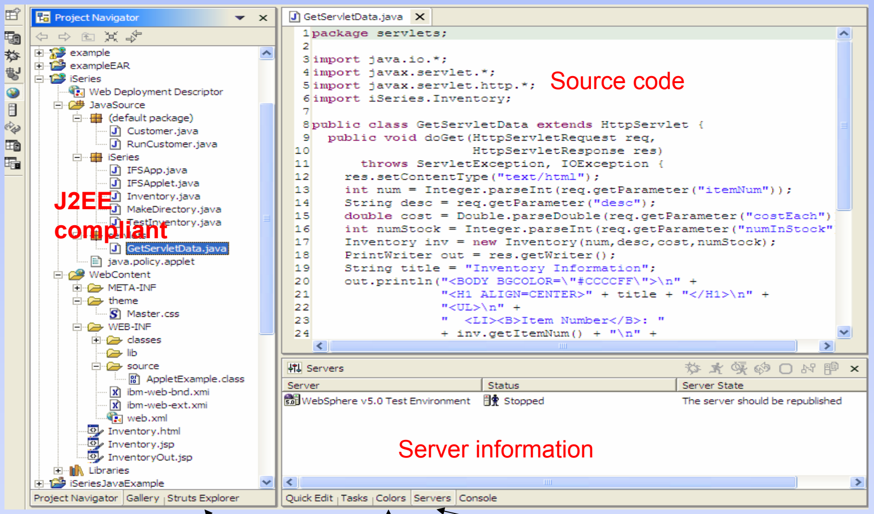
```java
/**
 * A JavBean class representing Inventory
 */
package iSeries;

import java.text.*;
public class Inventory {
    // fields
    private int itemNum;
    private String desc;
    private double costEach;
    private int numInStock;
    private static int number;


    NumberFormat money = NumberFormat.getCurrencyIn

    // constructors
    public Inventory()
    {
    }
    public Inventory (String desc, double costEach,
    {
        this.desc = desc;
        this.costEach=costEach;
        this.numInStock=numInStock;
```

Note: you can have multiple files open at once

Color coded outline

Output displays here

Web Perspective



```java
package servlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import iSeries.Inventory;

public class GetServletData extends HttpServlet {
  public void doGet(HttpServletRequest req,
                    HttpServletResponse res)
      throws ServletException, IOException {
    res.setContentType("text/html");
    int num = Integer.parseInt(req.getParameter("itemNum"));
    String desc = req.getParameter("desc");
    double cost = Double.parseDouble(req.getParameter("costEach")
    int numStock = Integer.parseInt(req.getParameter("numInStock"
    Inventory inv = new Inventory(num,desc,cost,numStock);
    PrintWriter out = res.getWriter();
    String title = "Inventory Information";
    out.println("<BODY BGCOLOR=\"#CCCCFF\">\n" +
                "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
                "<UL>\n" +
                "  <LI><B>Item Number</B>: "
                + inv.getItemNum() + "\n" +
```

Server	Status	Server State
WebSphere v5.0 Test Environment	Stopped	The server should be republished

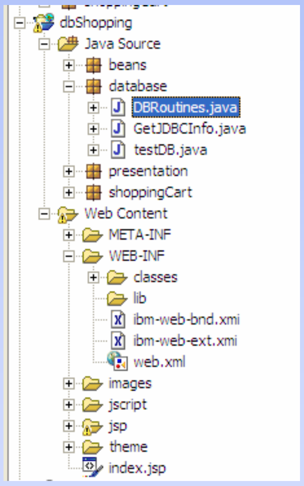
J2EE compliant

Source code

Server information

Banners, bullets   Strut info   Web colors   Server information

Web Perspective (cont)



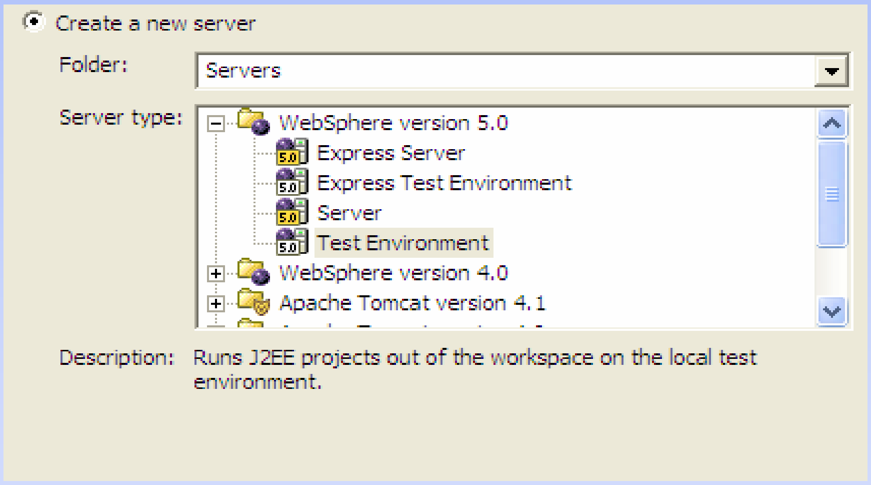
```
dbShopping
  Java Source
    beans
    database
      DBRoutines.java
      GetJDBCInfo.java
      testDB.java
    presentation
    shoppingCart
  Web Content
    META-INF
    WEB-INF
      classes
      lib
      ibm-web-bnd.xmi
      ibm-web-ext.xmi
      web.xml
    images
    jscript
    jsp
    theme
    index.jsp
```

← When you open a Web app, the correct file structure for J2EE standards is automatically created for you.

← Automatically packages the application in an .ear file using J2EE specs

← Provides the ability to debug and test the Web app in a totally remote WebSphere Test Environment

Integrated Test Environment



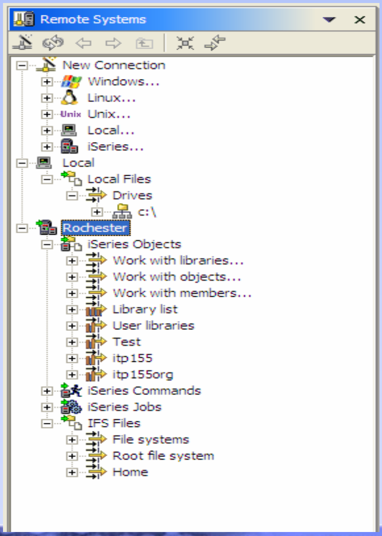
(●) Create a new server

Folder: Servers

Server type:
- WebSphere version 5.0
  - Express Server
  - Express Test Environment
  - Server
  - Test Environment
- WebSphere version 4.0
- Apache Tomcat version 4.1

Description: Runs J2EE projects out of the workspace on the local test environment.

Remote Systems Explorer Perspective

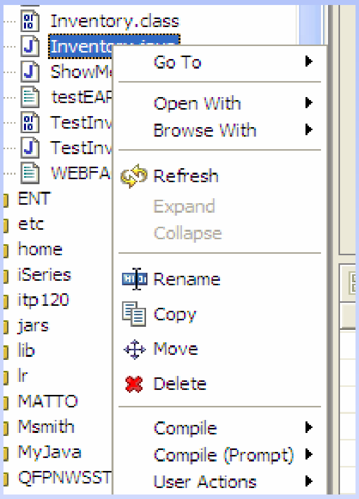


← Can make connections to various systems

←Can transfer from local drives to IFS

← The iSeries IFS provides transfer from Java perspective directly to the iSeries IFS

Right-Click on a .java in IFS

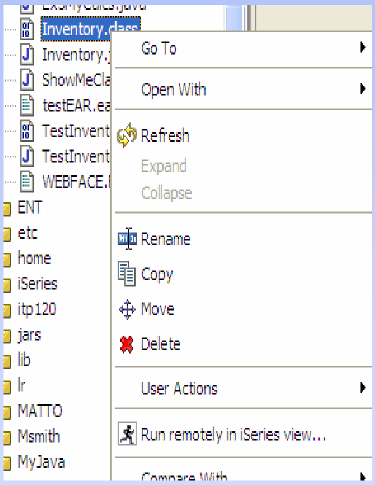


← Can open and edit the Java code that is on the iSeries

← Can copy, move etc. Java programs that are on the iSeries

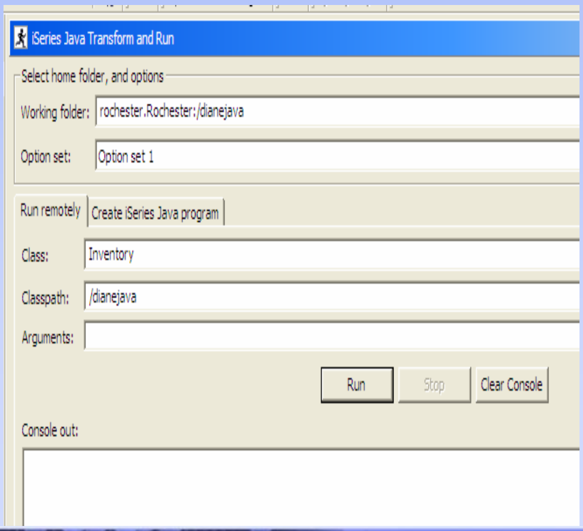
← Can compile on the iSeries from WDSc

Right-Click on a .class in IFS



← **Can run the Java program on the iSeries from WDSc**

Options for Running a Java Program


iSeries Java Transform and Run

Select home folder, and options

Working folder: rochester.Rochester:/dianejava

Option set: Option set 1

Run remotely | Create iSeries Java program

Class: Inventory

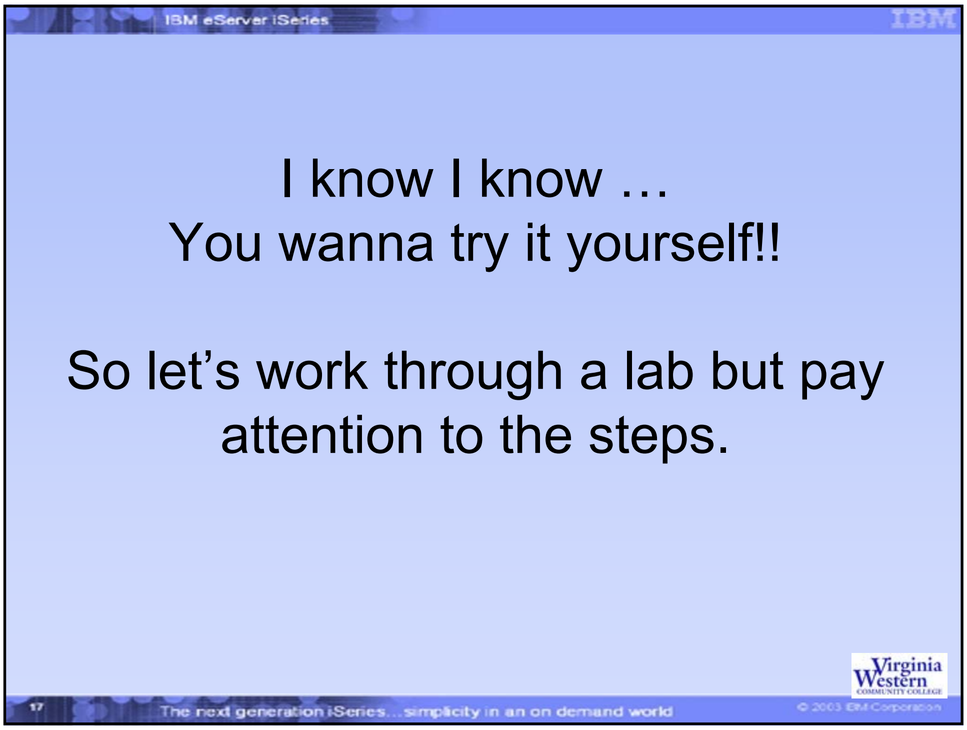
Classpath: /dianejava

Arguments:

← **Can define option sets (different parameters for different applications)**

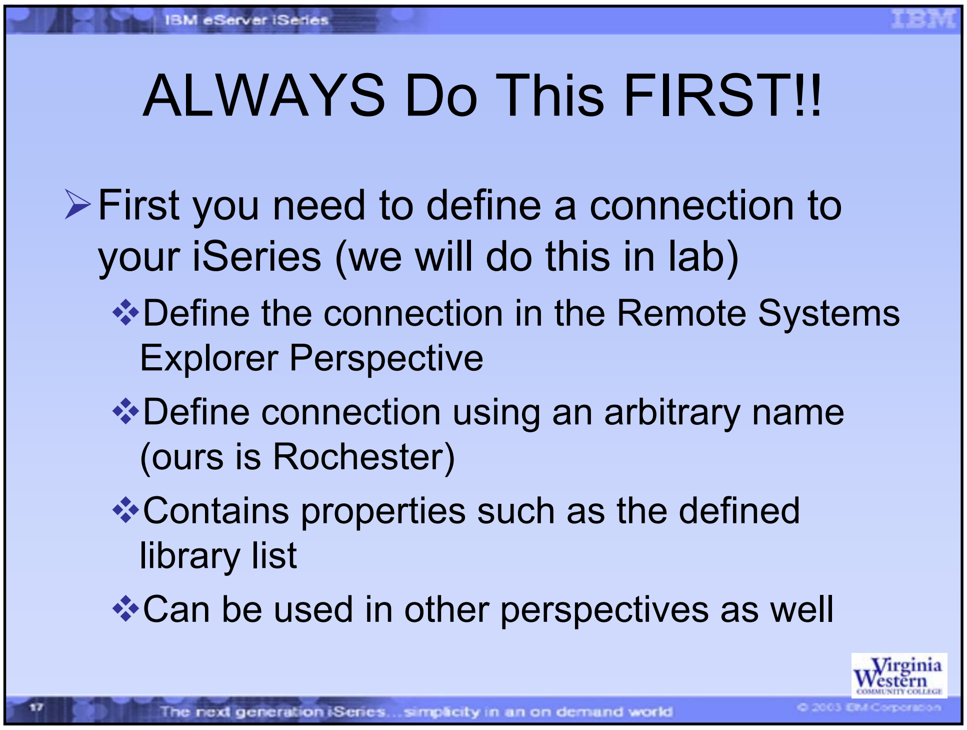
← **Can change classpath and arguments before running**

← **Output shows here in the Console out:**



I know I know …
You wanna try it yourself!!

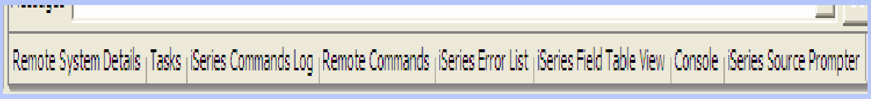
So let’s work through a lab but pay attention to the steps.



ALWAYS Do This FIRST!!

- ➢ First you need to define a connection to your iSeries (we will do this in lab)
  - ❖ Define the connection in the Remote Systems Explorer Perspective
  - ❖ Define connection using an arbitrary name (ours is Rochester)
  - ❖ Contains properties such as the defined library list
  - ❖ Can be used in other perspectives as well

Some most Useful Views



•Remote System Details – details of the remote system

•Tasks – use to remind yourself of tasks you must complete

•iSeries Commands Log – keeps track of equivalent CL commands

•Remote Commands – launches a command shell in a remote system

•iSeries Error List - ID, message, severity, line, and location of iSeries errors

•iSeries Field Table View – displays name, record, type, length and text

•Console – output to the "screen"

•iSeries Source prompter – prompts for iSeries commands

Summary

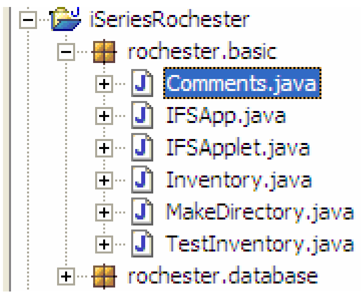
- WDSc is a powerful tool for application development
- Great for constructing a complete end-to-end J2EE Java application and testing it before deploying to a Web Application server
- It is THE IDE of choice for Java developers (no bias on my part! ☺)

WDSc Java Lab
Summer School IBM 2004
Riggins/Wolff

NOTE: these are lab sections from our modules for our Java classes

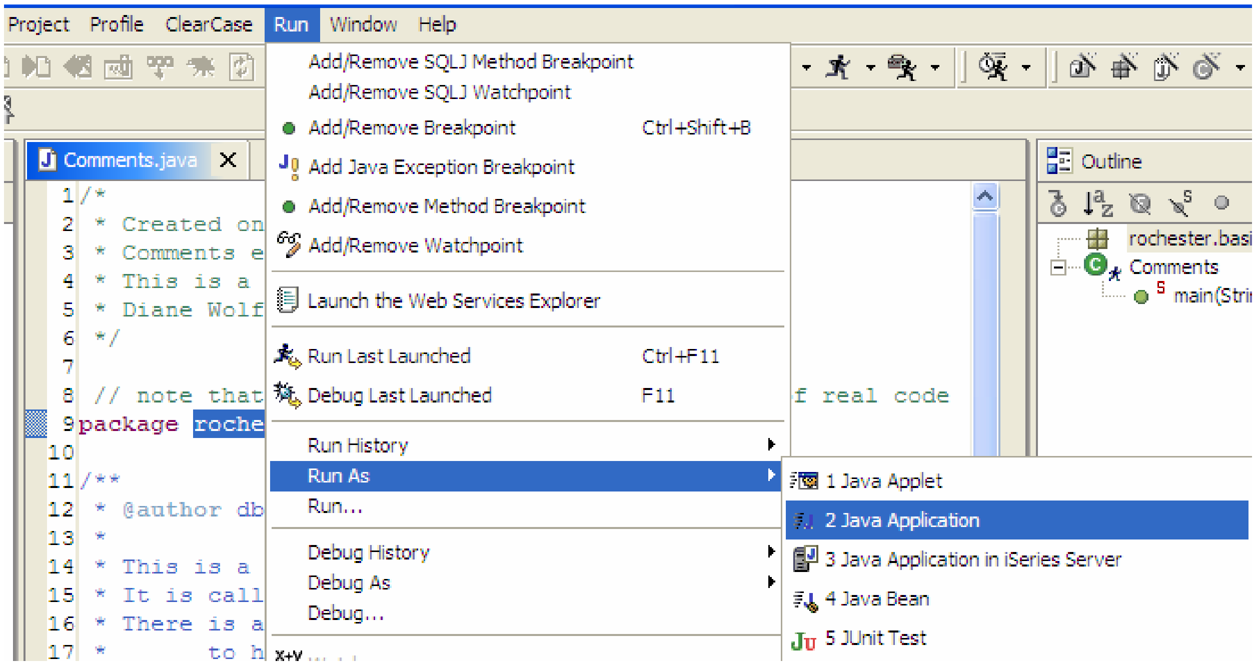
Part 1 . Running A Simple Program using WDSc and the iSeries

1. Start WDSc and import our jar file.
   a. Follow the instructions in the Word document named **JavaWDSc.doc**.
   b. Write the name of the folder where your workspace will be and click **OK**. (Write the folder name somewhere it will be easy to locate if you need to refer to it!!!).
   c. After WDSc opens, check that you are in the Java perspective. Look for the icon in the vertical icon list that looks like .
   d. Import the **rochester.jar** jar file.
      i. First create a **Project.** Click on the icon on the menu bar, name the project **iSeriesRochester**, and click on **Finish**.
      ii. Import Rochester.jar into this project. Click on iSeries in the **Package Explorer perspective** and choose **File → Import → Zip file → Next.** Browse for the location of the zip file (we will give you instructions on this location) and click **Open** and **Finish**.
2. Look at the code in Comments.java.
   a. Expand by clicking the + sign in front of the package named **rochester.basic**.



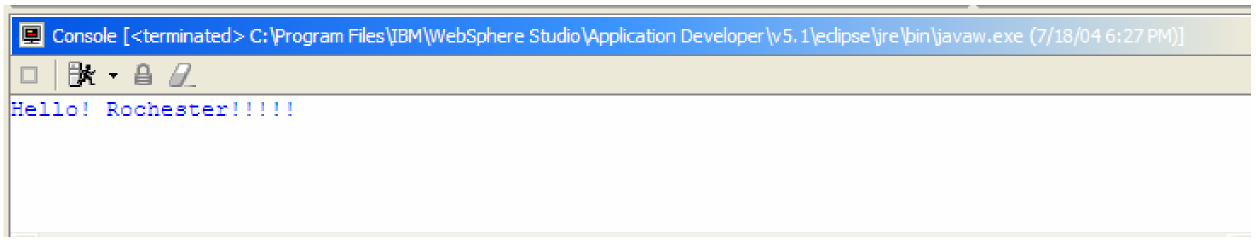
```
iSeriesRochester
  rochester.basic
    Comments.java
    IFSApp.java
    IFSApplet.java
    Inventory.java
    MakeDirectory.java
    TestInventory.java
  rochester.database
```

   b. Double-click on the program named Comments. View the code. Note -- the package name must be the top line of the code. Double-click on the program name on the top tab to make the code display using the full screen. Double-click on the program name again to display it in default view size.
3. Run the Comments program in my .jar file. View the output.
   a. Make certain that Comments.java is selected in the Package Explorer.
   b. Select **Run → Run as → Java application**


Project
Profile
ClearCase
Run
Window
Help
Comments.java
Outline
rochester.basi
Comments
main(Stri
Add/Remove SQLJ Method Breakpoint
Add/Remove SQLJ Watchpoint
Add/Remove Breakpoint Ctrl+Shift+B
Add Java Exception Breakpoint
Add/Remove Method Breakpoint
Add/Remove Watchpoint
Launch the Web Services Explorer
Run Last Launched Ctrl+F11
Debug Last Launched F11
Run History
Run As
Run...
Debug History
Debug As
Debug...
1 Java Applet
2 Java Application
3 Java Application in iSeries Server
4 Java Bean
5 JUnit Test

```
/*
 * Created on
 * Comments e
 * This is a
 * Diane Wolf
 */

// note that                                   f real code
package roche

/**
 * @author db
 *
 * This is a
 * It is call
 * There is a
 *         to h
```

You will see the output of this simple Java program show up in the Console view at the bottom of the screen.


Console [<terminated> C:\Program Files\IBM\WebSphere Studio\Application Developer\v5.1\eclipse\jre\bin\javaw.exe (7/18/04 6:27 PM)]

```
Hello! Rochester!!!!!
```

4. Let’s see where the program was saved.
   a. Go to the folder that was created in step 1. (This is folder name that you were to write down and store in case you needed to refer to it.)
   b. Browse to this location.
   c. Note that in the workspace, there is a folder created named rochester that has a folder in it named basic (it also has one that is named database) and in the basic folder is both Comments.java and Comments.class. The .java file is the source code and the .class file is the compiled code. The Windows folder structure mimics the Java package structure. Note that a package named **a.b.c** would be in the workspace. The workspace is organized so that the folder named **a** has a folder in it named **b**. This folder **b** has a folder in it named **c** that contains the java code.
5. Next we will use Comments.java, export this file to the iSeries, and run it on the iSeries. Java programs on the iSeries are stored in the IFS (integrated file structure) files. There will be one folder in the iSeries IFS already created for you under the root file system with your username as the folder name.
6. Sending our file to the IFS.
   a. Using the Java perspective, click Comments.java.
   b. Right-click and select **Copy**.

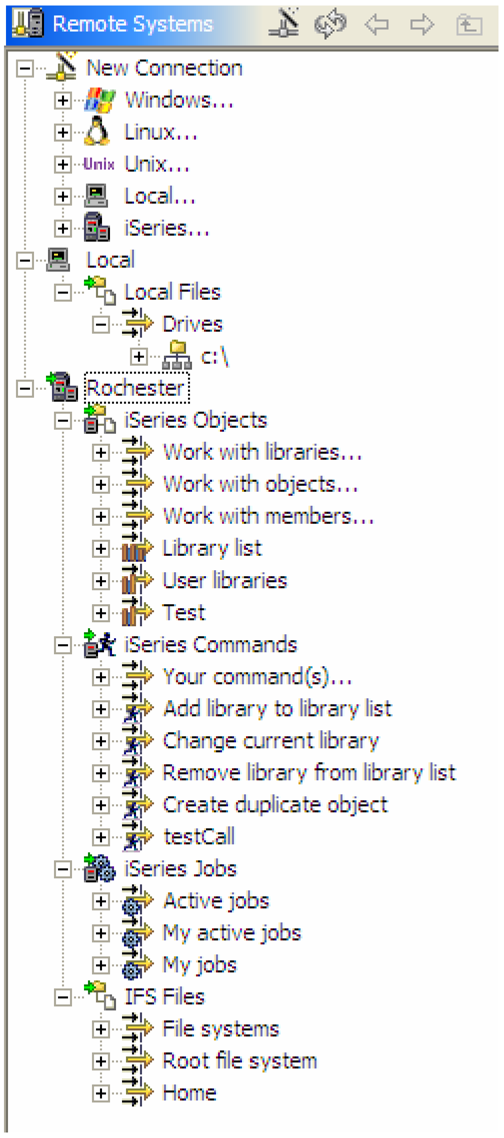
c. Open the Remote System Explorer perspective (if there is not an icon in the left margin for it, select **Windows → Open Perspective → Remote System Explorer**).

d. You will need a connection to the iSeries from this perspective.

7. Connect to the iSeries

   a. To make a connection to the iSeries, refer to the **iSeriesWDSCConnection.doc** file. We will assume that you have the software installed and the connection created for the remainder of this exercise.

8. After making the connection, your workbench should appear as the graphic below:



← Note you can make a Windows, Linux, Unix, local, or iSeries connection

← Use this area to get to your files on your local drive

← Our connection was named Rochester

← You can choose to work with libraries, objects, members or your library list

← Work with your iSeries commands here

← Work with your iSeries jobs here

← Work with the iSeries IFS files here
This is where our Java programs are stored. You will have a folder under the root file system with your username.

9. The connection that we created in the illustration above was named Rochester. You should see the connection that you defined on your workbench.

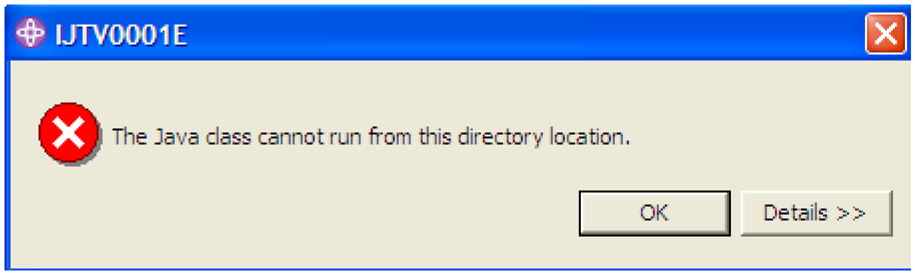
10. Next we will move Comments.java to the IFS, compile, and execute the program. NOTE: in the Java perspective, when you choose **Run → Run as.. → Java Application** it both compiles and executes. On the iSeries, you must perform these steps separately.

11. Running our program.

   a. Open the Java perspective, right-click on **Comments.java**, and select **Copy**.
   b. Open the Remote Systems Explorer (click on the icon).
   c. Open the **IFS** folder near the bottom, open the **root file system,** and open your folder (*yourusername*).
   d. Right-click on your folder and choose **Paste**. The files will be copied to the IFS file structure of the iSeries. Note that we have successfully transferred these files from the Windows machine to the remote iSeries. Double-click on the file in the IFS. Note that the editor Window opens and you can edit the file while it is on the iSeries as well. Close the opened window (**X** top right of the Window Comments.java ✕).

12. Compiling the program.
   a. Now right-click on the Comments.java file.
   b. Choose **Compile → javac**. The file will be compiled and the Comments1.class file will appear in the folder.
   c. You should now have two files in the folder (you may need to right-click and choose to Refresh the output), the source file (.java) and the byte code compiled file (.class).

13. Running the program
   a. Right click on the **Comments.class** file and choose **Run remotely in iSeries view**.
   b. You will get a dialog box that looks like


IJTV0001E

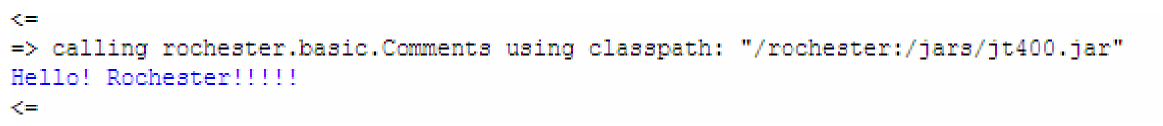
The Java class cannot run from this directory location.

   c. Read the details. Remember from the beginning of this lab, that this class is in the package itp120mod1 and therefore needs to be in a folder named itp120mod1.
   d. Right-click on your folder name and choose **New → Folder**. Name the folder **rochester**.
   e. Right-click on this folder that you just created and add a folder inside named **basic.**
   f. Click Finish.
   g. Now right-click on Comments.class and choose **Copy**.
   h. Right-click on the folder that you created (the **basic** folder that is inside of the **Rochester** folder) and choose **Paste**. You will get a copy of Comments.class in the folder.

i. Right-click on Comments.class (the one that is in the correct folder for the matching package structure) and choose **Run remotely in iSeries view.** The **iSeries Java Transform and Run** view will show at the bottom.

j. Press the **Run** button. (If you do not see it, double-click on the blue toolbar to make that view full-size.

k. You will see the output of the Java program in the **Console out.** It should look like the following:



```
<=
=> calling rochester.basic.Comments using classpath: "/rochester:/jars/jt400.jar"
Hello! Rochester!!!!!
<=
```

14. Right-click on this folder that you created and choose **Show in Table**. Note the table view. You can click on any title to sort by that field. You can rename, copy, move, and delete in the table view just as you do in the outline view.

15. ALL RIGHT!! You now have run a Java program both remotely and on the iSeries!!!!!!!

Part II Using the IBM Toolbox for Java

16. Now we will look at a few toolbox examples.

a. Make certain that you are back in the Java perspective. Look for the icon in the vertical icon list that looks like .

b. Open up the Rochester.basic package.

c. Double-click on the **MakeDirectory.java** file. The Java editor will open and you can see the Java code for this program. Read the comments in the code to get an understanding of the program. This program will make a directory on the IFS file system of the iSeries (you must have permission to do this) and writes a Java file to the directory.

d. With this file highlighted in the Project Navigator, choose **Run → Run as → Java application** from the workbench menu. The program will launch.

e. You will see a window for signing on to the server. It will ask you for your System (put in the IP address of your iSeries machine – we will provide you with this data), your UserID, and your Password.

f. Fill these in and click **OK** and the program should run. The Console view at the bottom should say "Successful!" if the connection was made and the file written. If you get an "IO Exception occurred" message, check your connection to the iSeries (the program could not make a connection.). NOTE: the program will create a folder on the IFS named MyJava. It will not check to see if such a folder already exists (it will just replace it) so if you run this multiple times or multiple people run this file, keep this in mind.

g. You will probably have to put jt400.jar in the classpath for this project.

i. Right-click on the project and choose **Properties**.

ii. Go to **Java Build Path** and choose the **Libraries** tab.

        iii. Click to **Add external jar** and browse to C:\JavaDrivers\jt400.jar.
        iv. Click **OK** to add it to your classpath.

    h. Double-click on **IFSApp.java** and look at the code. Run the program. The program will launch and you will see a window for signing on to the server. It will ask you for your System (put in the IP address of your iSeries machine), your UserID, and your Password. Fill these in and click **OK** and the program should run. The Console at the bottom will show whether the file is readable and/or writeable and the contents of the file. If the Console view is too small, you can double-click on its top bar to make it full screen size. Double-clicking again on the top bar puts it back to normal size.
    i. Repeat the instructions above with **IFSApplet.java**. In this case select **Run → Run as → Java Applet**. The default applet window is too small for the output so before filling in your System IP, drag the bottom right corner of the applet viewer to make it larger in both directions. Run as above. Then click **Exit** to end the applet. This size can be adjusted when applets are run in Web pages but we will not concern ourselves with that at this time.

17. Next you will look at a JavaBean.
    a. Double click on **Inventory.java**. This is a JavaBean. This program does not launch – it only defines the description of an Inventory JavaBean instance.
    b. The program that runs is **TestInventory.java** so double-click on this and look at the code.
    c. Then run it with **Run → Run as → Java application.** Since this is not using the iSeries, you will not see the panel for iSeries information. The console view will list information about bean instances of the Inventory class. Remember - if the Console view is too small, you can double-click on its top bar to make it full screen size. Double-clicking again puts it back to normal size.

18. We will now transfer those to the remote iSeries IFS file system.
    a. Open the Java perspective and under the **rochester.basic** package, highlight both the **Inventory.java** and the **InventoryTest.java** files (hold down the Ctrl key between selections.)
    b. Right-click and choose **Copy**.
    c. Open the Remote System Explorer perspective and click on your folder (you used this above).
    d. Right-click and choose **Paste**. The files will be copied to the IFS file structure of the iSeries. Note that we have successfully transferred these files from the Windows machine to the remote iSeries.
    e. Open up your folder on the IFS on the iSeries and right-click on the Inventory.java file.
    f. Choose **Compile → javac**. The files will be compiled and the Inventory.class file will appear in the folder.
    g. Repeat this step for InventoryTest.java. You should now have four files in the folder (you may need to right-click and choose to Refresh the output), the two source files (.java) and the two byte code compiled files (.class).

h. Remember what we learned about folders and package names and move these compiled versions to the correct folder on the IFS (into the basic folder that is in the rochester folder).

i. Right click on the **InventoryTest.class** file and choose **Run remotely in iSeries view**. The **iSeries Java Transform and Run** view will show at the bottom.

j. Press the **Run** button. (If you do not see it, double-click on the blue toolbar to make that view full-size. You will see the output of the Java program in the **Console out.**

k. Repeat the above step to compile and run your First.java file that we created by running the MakeDirectory.java class. This should be in the IFS file system in a folder named **MyJava**.

l. Compile and run the program. The output will show up in the Console at the bottom.

m. Right-click on **MyJava** and choose **Show in Table**. Note the table view. You can click on any title to sort by that field. You can rename, copy, move, and delete in the table view just as you do in the outline view.

Part III Database Access with Java

19. Make certain that you are back in the Java perspective. Look for the icon in the vertical icon list that looks like .

20. Open up the Rochester.database package

21. Run the ShowMeClassPath.java program in the rochester.database package to see the classpath for your applications.

22. To connect to any database, three things are required (after you put the jar file in the classpath). You must know the name of the Driver (this is supplied from the driver supplier), you must know the address of the database server, and you must have permissions and a userid and a password. I wrote a generic Java program named GetJDBCInfo that will extract the first two if you indicate which database you want to use. Look at the method named getDriver (). For the iSeries, the driver is the program `com.ibm.as400.access.AS400JDBCDriver`. Look at the method getURL(). The URL for our iSeries is `"jdbc:as400:164.106.231.17"`. We need to change this to the IP address for the iSeries that we are using during this class.

23. We will create the Coffee and Supplier tables from the SUN tutorial (http://java.sun.com/docs/books/tutorial/jdbc/basics/index.html -- I will follow it somewhat but alter the code as needed). Run Ex1CreateTables.java. If it runs successfully, you should see the output (what you type in is green below but use your assigned user name and password):

```
Which database?(0)oracle(1)iSeries(2)SQL Server(OLD)(5)SQL Server (NEW) 1
Username: yourusername
Password: yourusername
com.ibm.as400.access.AS400JDBCDriver
jdbc:as400:164.106.231.17
Known drivers that are registered:
AS/400 Toolbox for Java JDBC Driver
Connection successful!
Coffee Table didn't exist
```

```
Supplier table didn't exist
Tables created
Closing the connection in finally
```

24. If it works, **congratulations!!!!!!!** If not, if you get a red error about not finding the drivers, check your work on step 2 above. If you still get the error, make certain you can get to the Internet from your computer. Do you have firewalls that prevent you from hitting the iSeries?
25. Switch to the RemoteExplorer perspective and bring up your library on the iSeries (not on the IFS).
    a. Go to the RemoteExplorer perspective .
    b. Double-click on **Work with Libraries**.
    c. You will be able to view the library filter screen.
    d. For the **Library:** entry, enter *yourusername* and click **Next**. On the next screen, name your filter *yourusername* and click **Finish**.
    e. An entry under iSeries objects for *yourusername* should be displayed.
    f. See if these physical files have been added to your library.
26. Now we need to add data to the two physical database files.
    a. Run Ex2PopulateTables.java.
    b. Note carefully the output about what classes are really executing. Although we refer to Connection, we can NOT make an instance of this since it is an interface. So that can not be the class that is actually running. You can always find out what class is actually being used by `con.getClass().getName()`. Note how each piece of data is obtained. This java program is not great since you use the column names and types that are returned, and you may not know that.
27. Run Ex4DisplayTablesGeneric and compare it to Ex5DisplayTables. It is more advantageous to do it generically.
28. Now run Ex6Join. Essentially you can use any valid SQL statement. Look at the API for DatabaseMetaData (used to find out information about the underlying database) and ResultSetMetaData (used to get information about the result set that is returned). This second one can be used to make the output generic so you do not need to know the column names or types.
29. Prepared statements run much faster. Run Ex7PreparedStatements. Note how prepared statements work.
30. Very often you will want to use stored procedures. They run even faster.
    a. Run StoredProcCreate_IBM to create a stored procedure. Understand the code.
    b. When creating stored procedures, you need a SQL statement similar to:

    ```
    createString  = "create procedure updateco language sql update
    coffees set price = price *2 where sup_id = 101";
    ```

    c. Run StoredProcCall_IBM to run the stored procedure.
    d. When running a stored procedure that has parameters, you need a call similar to:

    ```
    ca = con.prepareCall("{call updateco2 (150)}");
    ```

31. Now let's see how to port some of these programs to the IFS.
    a. Right-click on CreateTables.java and hold down the CTRL-key while choosing PopulateTables.java, GetJDBCInfo.java, and Keyboard.java.
    b. Choose **Copy**.
    c. Open up your folder on the IFS right-click, and paste these in with **Paste**.
    d. Using our previous methods, compile the Keyboard and GetJDBCInfo classes.
    e. Right-click on your folder and create a folder named itp120mod6. Right-click on that folder and create a folder named database.
    f. Move the compiled versions of Keyboard and GetJDBCInfo into this folder.
    g. Compile the CreateTables and GetJDBCInfo source files.
    h. Move the compiled versions of these two into the database folder also.
    i. Right-click on the CreateTables.class that was moved to the folder and choose to **Run Remotely on iSeries view**.
    j. Double-click the toolbar to make it full screen.
    k. We have to put the jt400.jar file in the classpath before we run this.
    l. I have the jt400.jar in a folder named jars in the root IFS directory.
32. Double-click on **Advanced Options**.
    a. Click on the **ENVIRONMENTAL** tab.
    b. In the textbox for **Variable:** type in CLASSPATH.
    c. For the textbox for **Value:** type in /jars/jt400.jar.
    d. Click **Append to List**, click **SAVE**, and click **OK**.
    e. Note that this was filled in for option 1. You can store a variety of classpaths for running different programs.
33. Now click the **RUN** button. CreateTables should run. Fill in **1** to choose the iSeries and fill in your username and password. The program should complete successfully.
34. Now from the IFS, right-click on PopulateTables.class (in the database folder) and choose to **Run Remotely on iSeries view**. Double-click the toolbar to make it full screen. Choose the **RUN** button. Fill in **1** to choose the iSeries and fill in your username and password. The program should complete successfully. You will see the output of populating the COFFEES and SUPPLIERS table.
35. Note – when running Java on the iSeries, there are two items that you need to consider: what optimization level you are using and which JDBC driver you choose. We used the default optimization (level 10) and the iSeries native JDBC driver (not the one found in the jt400.jar that we put in the jars folder). FYI – to change the optimization, there is choice on the **Transform** tab that was in the **Advanced Options** that we saw in step 18 above. Often you use 10 or 20 to develop and 30 or 40 for deployed applications. Also – it should be using the native JDBC driver. When making the connection,

```
connection = DriverManager.getConnection(url+";user="+uid +";password="+ pass
+ ";driver=native");
```

will use the native driver when running th program on the iSeries and the jt400.jar JDBC driver when running on the client.

```
connection = DriverManager.getConnection(url, uid, pass);
```

will always use the JDBC drive that is in the jt400.jar file.

Part IV Server-Side Web Development

36. Web examples are stored in .ear files – not .jar files. Included is all of the java plus the images, JavaServerPages, html files, etc. We will import **rochester.ear**. The example will have:

    Inventory.html – an HTML file with FORM data used to send information about an inventory item to the Web. When the FORM is submitted, it calls the GetServletData java servlet that processes the data and returns an HTML file back to the user.

    GetServletData.java – a servlet called by Inventory.html that returns the data back to a Web page.

    Inventory.java – a JavaBean class used to represent inventory items

    Inventory.jsp – similar to the Inventory.html file except it is a JavaServer Page. When the FORM is submitted, it calls the JSP InventoryOut.jsp.

    InventoryOut.jsp – retrieves data from the JSP page named Inventory.jsp and creates an Inventory JavaBean instance from the data. It then retrieves the data fields from this bean and prints them out.

37. Importing the Web application.
    a. Open up the WebSphere Development Studio Client for iSeries (take the default location for the workspace). Then open the Web perspective ● if it is not already opened (**Window → Open Perspective → Web**. If you do not see Web as a choice, click **Other** and choose it from here).
    b. Make certain that the **Project Navigator** tab is selected in the left window.
    c. Then do **File → Import → EAR file → Next**.
    d. On the next screen click the Browse button to browse to find where you located the file **rochester.ear** and double-click on it. The EAR file name and the Project name should now be filled in for you. Note that a default location for storing this project is also created. This can be changed but we will leave it as it is.
    e. Click **Finish**.
    f. You should see two new folders in your Project Navigator – **iSeriesRochesterWeb** and **rochester**.

38. Click on the + signs in front of each folder to investigate the contents. Inside of the **iSeriesRochesterWeb** folder, you will see the programs described above. The Java files are located in the package named servlets under JavaSource. These packages are located in the **Java Source** folder. The html and JSP files are located under **WebContent.**

39. Let's look at a Web servlet and JavaServer Page. These files must execute in a Web application server. The Client has a Web application server built in called the WebSphere Test Environment. This makes testing Web applications very nice since you do not have to export them to the real Web application server to debug and test.

40. Let us view and run the servlet first.
    a. The servlet class will be called from an HTML file named **Inventory.html**. This file is found under WebContent.

b. Double-click on this file and study the code. Note that the FORM action calls **servlet/servlets.GetServletData**. When the submit button on the form is pressed, the form data (any data filled into the text boxes on the form) are forwarded to the **GetServletData.java** program (find it under the Java Source in the package named servlets.)

c. Double-click on the **GetServletData.java** file to see the construct of a servlet. When the **SUBMIT** button on the html page is pressed, it executes this servlet and calls the doGet method that retrieves the data and returns an HTML file that was created dynamically by the servlet.

d. Servlets must be run on the Web application server. To do this, right-click on **Inventory.html** and choose **Run on server**.

e. If this has not been done before, you need to define a new server instance to use. Click on **Create a new server** (the name does not matter) and choose **Test Environment → Finish**. This process will take several minutes as the server is created, opened, and launched. The Web application is bundled into an ear file that is placed on the server and run. You can follow these steps by watching the Console at the bottom.

f. After the server is launched, the Web page will show in the center in the internal Web browser (which is an instance of Internet Explorer.)

g. Fill in test information (it does not matter what you enter but make certain to use valid data) and click the **SUBMIT** button.

h. The form will call the servlet from the action attribute (servlet/servlets.GetServletData) and run that program which will create an HTML page and return it to the Web browser.

41. Now we will do a similar application with JavaServer Pages.

    a. Look at the code for **Inventory.jsp** and **InventoryOut.jsp**.

    b. Right-click on **Inventory.jsp** and choose **Run on server.**

    c. Since a server has previously been created (for the above example), it will be available for use. Choose the server that you previously created.

    d. The file will launch and look similar to the HTML version.

    e. Fill in the information and press the **SUBMIT** button.

    f. This form action calls **InventoryOut.jsp** which will retrieve data from **Inventory.jsp** and creates an Inventory JavaBean instance from the data. It then retrieves the data fields from this bean and prints the values of the datafields. In this case, the programmer chose to display the output in a table instead of as bulleted items.

42. The applications from this chapter have been executed inside of the WebSphere Development Studio Client for iSeries. Once you get your code debugged and running in this environment on the Test server, you can move your .ear file to a WebSphere Application Server for production deployment.

ITP120 General Laboratory Instructions
(JavaWDSC.doc)

You will need to install WDSC. The general workstation requirements are:

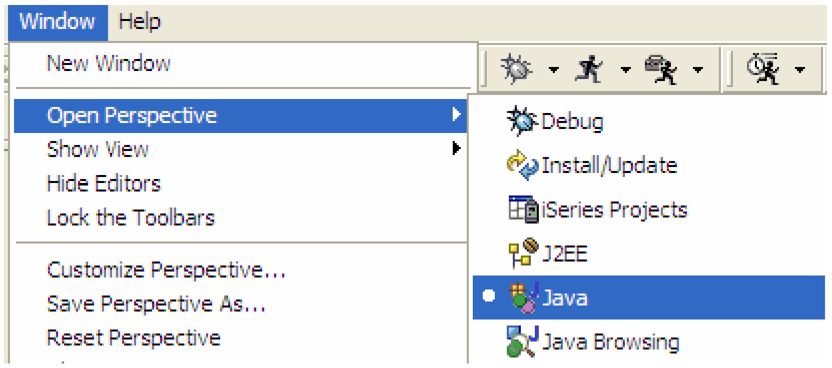
Windows 2000 or XP
Microsoft Internet Explorer 5.5 or higher
500 MHz process or faster recommended
512 MB RAM or more recommended
2.2 GB hard drive or larger recommended
CD ROM drive

The software will be provided on CDs for you. It will autoboot and self-install. Take the defaults for WDSC. More details for installation can be found in the redbooks for the software. Contact me if you need help

---

Starting WDSC

1. We will assume that you have successfully installed WDSC.
2. Choose **Start → Programs → IBM WebSphere Studio → Development Studio Client for iSeries**. You will be asked where you would like to store your workspace (where your projects are stored). Note the default location and take it (click **OK**).
3. Wait for WDSC to open up. Look at the icons along the left edge. If you see one that looks like double-click it. This is the Java perspective. If you do not see one, from the tool bar, choose **Window → Open Perspective → Java.**



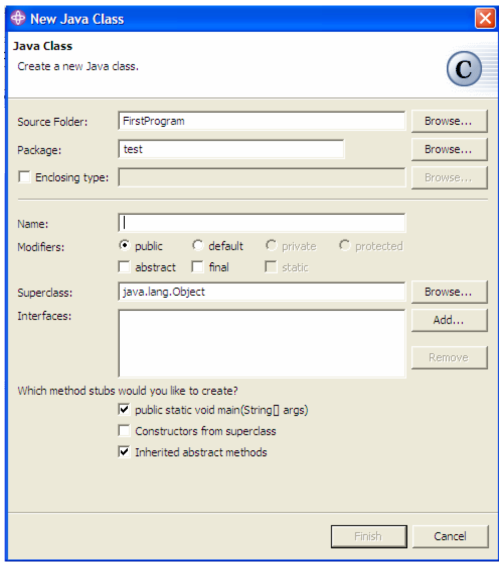
4. Every Java program (or class) must be contained in a package. A package is a Java concept that puts "similar" programs together. For WDSC, every package must be in a project. Although this is not a Java concept, it is used to keep similar packages together. Therefore, when creating Java applications from scratch, we need to first create a project – then a package – then the class or classes.
5. Java applications are imported and exported from WDSC as .jar files. A .jar file is an "intelligent" .zip file. For our concern at this point, they are essentially the

same (.jar files can even be opened with WinZip or other zipping software). WDSC will do the jarring and unjarring for you.

6. **ALL** Java programs are contained in a Java class (we will discuss the details of this later) and the source code (the "English-like language") will be stored in a file named ClassName.java where ClassName is the name of the class. When the program is compiled (or converted to byte code – one level above machine language – see page 40 in your text), it must be stored in a file named ClassName.class.

Creating a Java Application from Scratch

1. Complete steps 1-3 above to open up WDSC.
2. Find the icons on your tool bar that look like .
3. First we must create a Java package. Click on the first icon above (the one with the J – this is the icon for creating a new Java package.)
4. On the window that comes up, give it a unique package name and click **Finish**.
5. Next we must create a package. A package s a group of related Java classes. Single-click on your project name in the left **Package Explorer** and then double-click on the icon that looks like the second one from the left in #2 above (it looks like a package!!). You will see a screen where the project name is filled in and it is waiting for you to supply a package name. **Package names in Java should be all small letters – no spaces!** Type in the package name and press **Finish**.
6. Now we will create a Java application (a class). Single click on your package name in the left **Package Explorer** and then double-click on the icon that looks like the far right one from #2 above (it has the letter C on it for class). You will see the New Java Class screen like below (it will have your project and package name instead of mine).


New Java Class

Java Class

Create a new Java class.

Source Folder: FirstProgram

Package: test

Enclosing type: [ ]

Name:

Modifiers: [✓ public] default private protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[✓] public static void main(String[] args)

[ ] Constructors from superclass

[✓] Inherited abstract methods

7. Fill in the name for your Java class. **Java classes MUST be named correctly. Typically use only letters (maybe with a number at the end) and if the name**

is made up of more than one word, capitalize each word in the name. For instance, the following class names are correct: MyFirstProgram, Test, Test2, Hello, DatabaseProgram --- but not myFirstProgram, test, Databaseprogram.

8. If you want your program to execute, leave the default check mark in front of the choice to add the "public static void main" stub (we will discuss this later) and if it is only a description of an object but not executable, uncheck this open. Click **Finish**.
9. Double-click on your class name in the left **Package Explorer.** You are ready to work on your application.

Adding a New Java Class to an Existing Package

Since the project and package are already present in the **Package Explorer** on the left, just follow steps 6-9 above.

Adding an Existing Java Class (from outside of WDSc) to an Existing Package

1. The project and package will already be present in the **Package Explorer** on the left.
2. Single-click on this project name in the left **Package Explorer** and then from the menu, choose **File → Import →File system → Next.** On the next screen, browse to find the location of the .java files and click on it. Click **Open**. Put a check mark in front of the Java files that you want to bring in (you can bring in more than one at a time). You will see the import screen. The "From directory:" and the "Into folder:" should be filled in. Click **Finish.**
3. The Java classes that you added now probably need to be slightly changed. The first line of a Java file needs to be the name of the package. Since you brought this file in from an external source, the package name is probably wrong. Double-click on the file and change (or add if it does not have a package statement) the very top line to be **package xxx;** where xxx is the name of the package that you imported these files into.

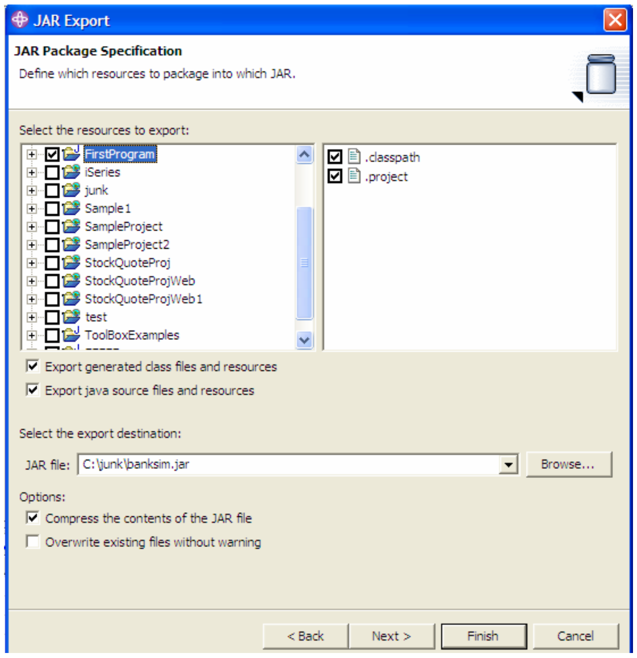
Importing an Existing Application from a .jar File

1. Complete steps 1-3 at the very top of this document to open up WDSC.
2. Find the icons on your tool bar that look like .
3. First we must create a Java project. Click on the first icon above (the one with the J – this is the icon for creating a new Java project).
4. On the window that comes up, give it a unique project name and click **Finish**.
5. Single-click on this project name in the left **Package Explorer** and then from the menu, choose **File → Import →Zip file → Next.** (Even though it is a jar file, we import it as a zip). On the next screen, browse to find the location of the jar file and click on it. Click **Open**. You will see the import screen. The "From zip file:" and the "Into folder:" should be filled in. Click **Finish.**

6. Double-click on your class name in the left **Package Explorer.** You are ready to work with your application.

Exporting a Application into a .jar File

1. After working on your application, you may want to export it from WDSC into a .jar file (this is how you will be submitting your work to me). Click on the project name that you want to export in the left **Package Explorer** on the left.
2. From the menu, choose **File → Export → jar file → Next**.
3. You will see a screen similar to the one below.



JAR Package Specification

Define which resources to package into which JAR.

Select the resources to export:

- [✓] FirstProgram
- [ ] iSeries
- [ ] junk
- [ ] Sample1
- [ ] SampleProject
- [ ] SampleProject2
- [ ] StockQuoteProj
- [ ] StockQuoteProjWeb
- [ ] StockQuoteProjWeb1
- [ ] test
- [ ] ToolBoxExamples

- [✓] .classpath
- [✓] .project

- [✓] Export generated class files and resources
- [✓] Export java source files and resources

Select the export destination:

JAR file: C:\junk\banksim.jar

Options:

- [✓] Compress the contents of the JAR file
- [ ] Overwrite existing files without warning

4. **MAKE CERTAIN THAT THERE IS A CHECK MARK I N FRONT OF "Export java source files and resources."**
5. Below the "Select the export destination", either type in the name of the jar file that you want to create, or browse for the location and type in the name. Click **Finish**. If this is an application that you will send to me, **NAME THE JAR FILE (ex dwolffmodx.jar except use the first letter of your name followed by your last name. x = which module. So if your name is Sam Smith, and you are sending me the lab for module 3, it would be ssmithmod3.jar. The package name should also be ssmithmod3 – we will see how to change it.)**
6. So, for instance, if Sam has an a: drive and want to store his module 2 jar file here, type in a:ssmithmod2.jar. **MAKE CERTAIN THE EXTENSION IS .jar**.
7. Click **Finish** and wait for the export to finish.

REMEMBER ---- IMPORT ZIP----EXPORT JAR!!!!

Changing Package Names

If I supply you a jar file and you are to make changes to it and send it back to me as your assignment, you will need to change the package name to follow the rules above. For instance, it you are working on module 3, the file I supply will be named itp120mod3.jar and the package name will be itp120mod3. You need to send it back to me with the package name as given in the instructions above. So, if your name is Mike Miller, you will need to change the package name to mmilermod3 and send it to me as mmilermod3.jar

To change a package name after importing it, in the **Package Explorer**, right-click on the package name and choose **Refactor → Rename** and the Rename window will come up. Fill in your new package name and click **OK**. The package name will be changed both on the workbench and in the code for every class in the package.

Running Java Applications

For a Java application to run, it must have a method with the construct of :

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

We will discuss this later, but this is the method that runs first. This is created by putting the check mark in front of this choice when creating the class (see instructions above). To run this class, single click on this class name in the left **Package Explorer.** Then from the tool bar, choose **Run → Run as → Java Application**. The program will run and any Console output will show at the bottom of the screen in the Console view.

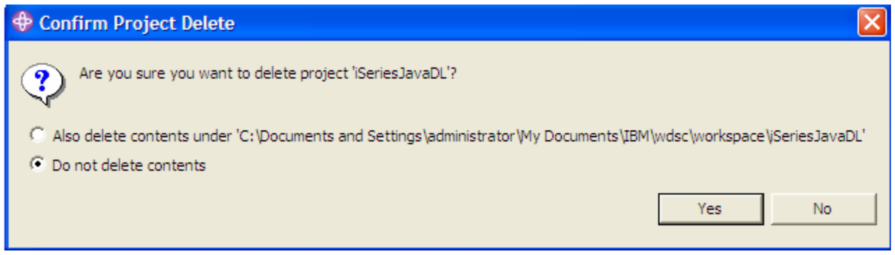
Editing or Creating Java Code

Whether you start your own class by the methods described above, or you want to edit one of my programs, double click on the class name in the left **Package Explorer** and the code will show up in the center editor window. Make your changes as you see fit. After you have made any changes, the name on the tab at the top of the editor window will have an * to indicate that changes have been made. Hold down the CTRL while hitting the S (for save) to save the file. The changes will be saved and the * will go away. If there are errors in the code, the line where the error occurs will be underlined red either as you type it or when you save it. Hover your cursor over it (or over the yellow light bulb in the left margin) to see help on the error. Correct the errors and save again. After all of the errors are gone, run the application as above.

Note – as you are typing Java code, you may want the editor view to fill the whole screen so you can see more lines of code at once. To do so, double-click on the program name on the tab at the top of the editor window (actually you can click anywhere on the top area) and to make it back to normal size, double-click there again.

Deleting Projects from WDSC

After you export and save your .jar file, you may want to delete the project from the WDSC workbench **Package Explorer** to keep this area neat (and to make WDSC execute faster). To do so, right-click on the project name and choose **Delete**. You will see the dialog box



Are you sure you want to delete project 'iSeriesJavaDL'?

- ○ Also delete contents under 'C:\Documents and Settings\administrator\My Documents\IBM\wdsc\workspace\iSeriesJavaDL'
- [✓ Do not delete contents]

Most times you want to choose the top one which removes the contents both from the workbench and the actual file system on your computer (**make certain you have a good .jar copy before you do this!!**). If you do not, when you try to import it next time, it will indicate that that project already exists.

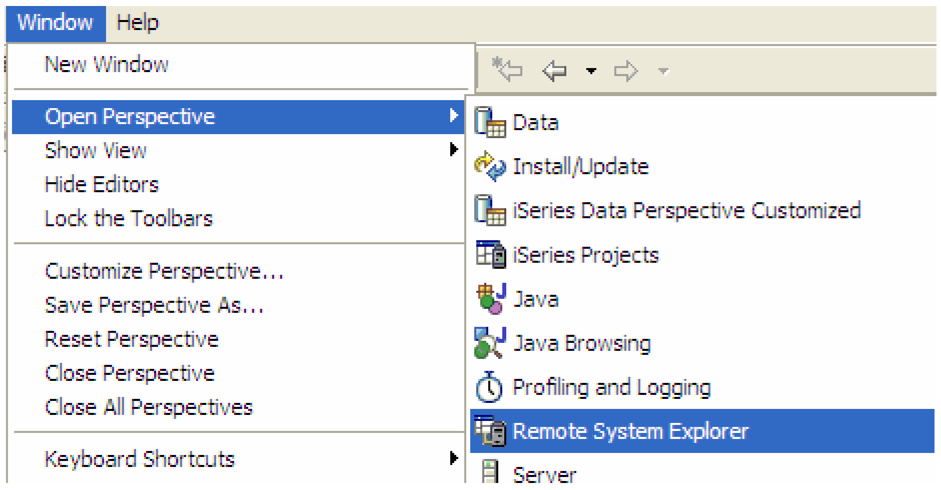
Starting an iSeries Connection in WebSphere Development Studio for iSeries **(iSeriesWDSCConnection.doc)**

After you have installed the software, click **Start → Programs → IBM WebSphere Studio → Development Studio Client for iSeries**. You will be asked where you would like to store your workspace. Note the default and click **OK**. Wait for the integrated development environment to open.

If you are using the program to access your iSeries, you will need to define a connection. After you have opened the workbench, you need to make certain that you are in the Remote System Explorer perspective so that you can define a connection to a server. Check the title bar of your workbench, which indicates the perspective you are in.

If it does not show the words **Remote System Explorer**, then check to see if the icon for this perspective is in the left margin. If not:

1. Click **Window → Open Perspective → Other**.
2. Click **Remote System Explorer**, and click **OK**.



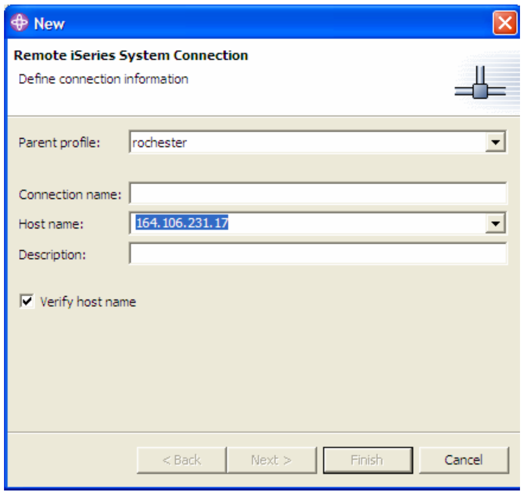
The first time you connect to an iSeries server, you need to define a profile. All connections, filters, and filter pools (collections of filters) belong to profiles. Profiles help you partition data when you have a lot of connections or filter pools. You use profiles to group connections, share connections, or keep them private, and they help you partition data if you have a lot of connections or filter pools.

Your first profile will be for your local workstation connected to the iSeries. As you complete the steps for your first connection, you can decide whether to use your initial profile, or a team profile so that you can share resources and information with other people.

To configure your first connection:

1. Switch to the Remote System Explorer perspective by clicking on the icon in the left margin.

2. In the Remote Systems view, **New Connection** is automatically expanded to show the various remote systems types you can connect to through the Remote System Explorer. Expand **iSeries** to configure a connection to an iSeries server.
3. If you are defining your first connection, you will see the **New - Name personal profile** dialog. Enter a name for your first profile and click **Next**. You will see the following dialog box.



New
Remote iSeries System Connection
Define connection information
< Back
Next >
Finish
Cancel

Parent profile: rochester

Connection name:

Host name: 164.106.231.17

Description:

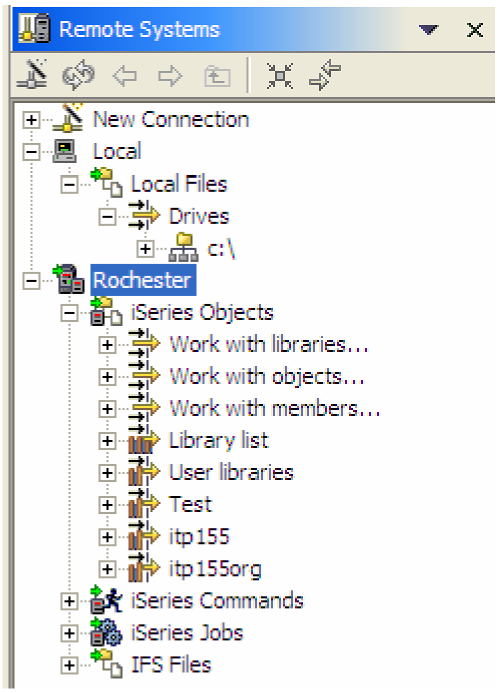
☑ Verify host name

4. Click the **Parent profile** drop-down list.
   - To keep your files private, select the default profile you created in the previous step.
   - To share your resources, select the Team profile.
5. Enter a connection name. This name displays in your tree view and must be unique to the profile. Use alphanumeric characters for your connection name; you might encounter problems if, for example, your connection name contains DBCS characters. But it does not matter what name you choose.
6. In the **Host Name** field, enter the name or TCP/IP address of your remote iSeries server, for example, PROD400. We need to use 164.106.231.17.
7. (Optional) Enter a **Description**; this appears in the Properties view after the connection is created.
8. Click **Finish** to define your system.

Note: If you are running the iSeries server remotely, to check your port number, right-click your connection or subsystem from the Remote Systems view and select **Properties**. Click **Subsystem** to see the relevant information. You will see that the port is "0," which means that your Remote System Explorer communications server will pick any free port on the iSeries server. You can specify a specific port number if you need to, for example, to work with a firewall.

After you create a connection to an iSeries server, you can easily connect and disconnect. To connect:

1. In the Remote Systems view, expand your new connection to reveal your subsystems.



2. (Optional) If this is the first time you are connecting to the remote server, right-click one of the subsystems, such as iSeries Objects, and select **Verify Connection**. Enter your user ID and password for the iSeries server to verify the connection. This action checks if you are missing any PTFs.
3. To connect to any of the subsystems, right-click one of them, such as iSeries Objects, and select **Connect**. You can also click the plus beside any of the subsystems to connect.
4. Enter your user ID and password when prompted and click **OK**. Select **Save password** if you want the workbench to remember your password. See the related link for more information about passwords.

You can monitor and change the properties of your connection in the Properties view of the Remote System Explorer perspective. Some values are read-only, and you can change others, such as the description or the server name. Although each Remote System Explorer subsystem maintains its own list of properties, three properties (connected or disconnected, port, and user ID) are shared among all iSeries subsystems. If you change any of these properties in one subsystem, the other subsystems reflect the change. Select a subsystem and check the Properties view to see the shared properties for all of your subsystems. For example, the **Connected** value is **Yes** or **No** for all of your subsystems under one connection.