Lab: Introduction to Java™ Programming

Student Exercises

Kim Button
John Eberhard

IBM Rochester

Session 33LA (403692)

Lab: Introduction to Java™ Programming

Goal

In this lab, you will work through a series of exercises that provide you with an opportunity to create some simple Java™ programs in a variety of environments. You will explore the differences between an application, applet, and servlet, as well as learn how to use some Java development tools.

Tools

This lab uses primitive tools. The most basic editor and command line compiler are used to create Java source code and compile your Java programs. You will use Windows® Notepad to edit your Java source code and the Java Development Kit (JDK, J2SE, SDK) to compile and run. The JDK is installed on the lab PCs.

It is common for Java developers to use these tools everyday. However, there are some very good integrated development environments (IDEs) that make programming easier. They provide syntax highlighting, point-and-click compile and run, integrated help text and much more. IBM WebSphere Development Studio Client is one such example. IDEs are not used in this lab because the lab would turn into an exercise in learning the tool instead of writing some Java code.

Setup

During this lab you will need a userid and password for the IBM System i5™ server and you will need to know the name of the system. The userid will be JAVAxx and the password is JAVAxx1 where xx is the number 01 to 99 (two digit number). This information will be given to you by your instructor. Please fill in this information below so that you have it for reference during the lab.

The **name** of my System i5 is: __________________________
My System i5 **userid** is: __________________________
My System i5 **password** is: __________________________
The System i5 **directory** is: /JAVA/**Javaxx** (Javaxx is your userid)
The PC **directory** is: C:\JAVA
The PC **hostname** is: __________________________

Hints

The most important tip that you should remember while performing the following exercises is that the Java language is **HIGHLY** case sensitive. Please be extra careful to double check the case of all code that you enter to ensure that you have input your code correctly.

Exercise 1: Create and run a simple Java application on Windows®

What This Exercise Is About

In this exercise you will create and run a simple Java application on Windows.

What You Should Be Able To Do

At the end of this exercise, you should be able to:

- Edit Java source code
- Compile the Java source code into a Java class file
- Run the Java program from the command line
- Know how to print text to the screen in a Java application

Introduction

Let's create a simple Java application. This application will print out the current date and time.

Exercise Instructions

Open An Editor

The Java Development Kit comes with all the tools you need for Java development except for a source code editor. You can use any editor you prefer. We will use Windows Notepad for this lab.

1. Open an MS-DOS prompt (in Windows 2000 and XP). Select the Start->Programs->Accessories->Command Prompt menu
2. Create a directory for your work. Type the following command in the MS-DOS prompt:

   ```
   mkdir c:\JAVA
   ```

3. Make this directory the current directory:

   ```
   cd c:\JAVA
   ```

4. Use Notepad to begin editing the Java source file. We will call the source file CurrentInfo.java. (Remember: Java is case-sensitive!)

   ```
   notepad CurrentInfo.java
   ```

   When Notepad asks you if you want to create a new file, click on **Yes**.

Now you are ready to start coding...

Enter And Compile The Source

Type in the following source code. Please note the punctuation as well as the capitalization. Both are very important elements of Java's syntax. Unlike some other languages, Java is not column specific so indentation does not matter.

```
import java.util.Date;

public class CurrentInfo {
   public static void main(String[] args)
   {
      System.out.println("Current date and time: " + new Date());
   }
}
```

Hint: The class name ("CurrentInfo" here) must match the name of the source file ("CurrentInfo.java") exactly - with the same case.

1. Save the file using the **File->Save** menu.
2. Set your **PATH** environment variable. In order to run a java program, you need to set your path environment variable to point to the Java Development Kit (JDK) which is installed on the lab PC. Type the following command in the MS-DOS prompt:

   ```
   SET PATH=%PATH%;c:\j2sdk1.4.2_09\bin;c:\jsdk2.0\bin
   ```

   ****NOTE**: This sets your PATH environment variable for the current MS-DOS prompt only. If you open a new MS-DOS prompt you need to reset your PATH in order to run a Java program.

3. In addition to setting up the path, you also need to tell Java where to find compiled class files at runtime. This requires you to set the **CLASSPATH** environment variable. Type the following command in the MS-DOS prompt:

   ```
   SET CLASSPATH=.;c:\jsdk2.0\bin;c:\jsdk2.0\servlet.jar
   ```

   ****NOTE**: This sets your CLASSPATH environment variable for the current MS-DOS prompt only. If you open a new MS-DOS prompt you need to reset your CLASSPATH in order to run a java program.

4. Now it is time to compile your source code into Java byte codes using the javac compiler. javac comes with the Java Development Kit (JDK) which is installed on the lab PC. Type the following command in the MS-DOS prompt (making sure you're in the c:\JAVA directory):

   ```
   javac CurrentInfo.java
   ```

5. If the source code compiles successfully, you will be returned to the c:\JAVA> prompt. No news is good news! If there are errors in your source code, they will be displayed.

6. If you have any errors in your source code, go back to your Notepad window, fix the errors, save the file, and try to compile again. Common errors are misspellings, inconsistent capitalization, or punctuation mistakes.

7. After you compile with no errors, type **`dir`** in the MS-DOS prompt. You will notice that there is a new file called CurrentInfo.class. This file contains the Java byte codes generated by the compiler.

Run The Application

Once you are able to successfully compile CurrentInfo.java, it is time to run it.

1. At the MS-DOS prompt, type:

   ```
   java CurrentInfo
   ```

   This starts a Java Virtual Machine (JVM), loads the Java byte codes contained in CurrentInfo.class, and runs the program.

2. Verify that you see the correct current date and time output from your program:

   Example Output:

   ```
   Current date and time: Fri Aug 25 12:38:39 CDT 2000
   ```

(Leave the MS-DOS window open. You will use it in other exercises.)

Exercise 2: Run a simple Java application on a System i5 Server

What This Exercise Is About

This exercise will demonstrate the "write once, run anywhere" ability of Java. In this exercise you will run the CurrentInfo Java application on the System i5 server.

What You Should Be Able To Do

At the end of this exercise, you should be able to:

- FTP Java class files from the PC to the integrated file system of a System i5 server
- Run Java programs on the System i5 server

Introduction

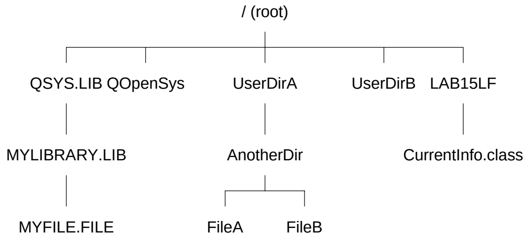
In the previous exercise you created a Java program and ran it on the PC. Now you will run the same program on the System i5 server.

Some System i5 Specifics

The Integrated File System on the System i5 server

To understand how Java programs are stored and run on the System i5 server, knowledge about the System i5's integrated file system is helpful. The integrated file system is a part of i5/OS that supports stream I/O and file management similar to UNIX and PC file systems**.** Key features of the integrated file system include a hierarchical directory structure and support for storing information in stream files. Benefits of the integrated file system include fast access to i5/OS data and efficient handling of stream data including images and audio. All Java programs that reside on the System i5 server are stored in the integrated file system. Libraries and files are accessible via the QSYS.LIB directory. An example of a directory tree is shown below:

System i5 Directory Tree Structure



Exercise Instructions

Use the emulator to sign-on to the System i5 server

In order to run a Java program on the System i5 server, the class file must be in the integrated file system of the System i5 server. First, you need to create a directory in the integrated file system as a place to copy the class file that you compiled in the previous exercise.

1. Start the 5250 emulator using the icon located on the Windows desktop.
2. Sign-on to the System i5 server using the user id and password assigned by the instructor.
3. Create a directory called /JAVA/JAVAxx, where xx is a number assigned by the instructor. This is the directory where you will copy your Java class file.

```
MKDIR '/JAVA/JAVAxx'
```

Use FTP to copy the Java class file from the PC to the System i5 server

You will now copy the Java class file from the PC to the System i5 server. Java programs are stored in class files. You will FTP your class file from the PC to the System i5 server. (Another option is to map a network drive using IBM System i5 Access for Windows to copy the file. To keep it simple you will use FTP in this lab). From the **MS-DOS prompt on your PC**, enter the following commands where ***system*** is the name of your System i5 server and ***Javaxx*** is your userid.

1. `ftp` ***`system`***
2. ... Enter your System i5 userid and password when prompted
3. `cd /`
4. `cd /JAVA/`***`Javaxx`***
5. `bin`
6. `put CurrentInfo.class`
7. `quit`

These FTP commands copied CurrentInfo.class to your directory on the System i5. If you are curious about what each of these FTP commands did, study the response text issued after each one.

HINT: Notice that the Windows file system uses backslashes (\) and the System i5 integrated file system uses forward slashes (/).

Run the Java application on the System i5 server.

The Java class file is now on the System i5 server. Let's run the Java application on the System i5 server.

1. The first step is to tell Java the location of your program. Java uses the CLASSPATH environment variable to find Java programs. You will tell it to look in **/JAVA/Javaxx**. *Environment variable names* (i.e., CLASSPATH) *are case sensitive* so make sure you type CLASSPATH in uppercase. Enter the following on the **i5/OS** command line:

   ```
   ADDENVVAR CLASSPATH '/JAVA/Javaxx'
   ```

2. Run the application on the System i5 server:

   ```
   java CurrentInfo
   ```

   Notice that this is the exact same command you used when you ran the application on Windows in an MS-DOS prompt.

3. Verify that you see the correct output from your program:

   ```
   Current date and time: Fri Aug 25 12:54:06 GMT+00:00 2000
   ```

4. Java applications run in a "Java shell" on the System i5 server. This Java shell looks a lot like the command entry screen. Press F3 to exit the Java shell and return to the command line.

(Leave the emulator open. You will use it in other exercises.)

Exercise 3: Manipulate Java Program Objects

What This Exercise Is About

Java class files contain byte codes. All platforms that support Java must convert these byte codes into machine instructions. The System i5 server has several ways of doing this -- interpreted (the byte codes are converted each time the Java application is run), static (the byte codes are converted once and saved), or a mixture of both (this is called the just-in-time compiler, or the JIT). The JIT compiler is the preferred method of running Java applications on the i5/OS because it offers the best performance.

Regardless of the method of execution, every time a Java application is run on the i5/OS, it must have an attached **i5/OS Java Program** object. This object contains an expanded version of the class file and some pre-verification information that allow faster run time processing. A Java Program (*JVAPGM) object can be created explicitly by the user, or, if no such object is found at runtime, the Java Virtual Machine will create one for you.

In this exercise, we will be running a Java application using the i5/OS JIT and will examine the **CRTJVAPGM**, **DSPJVAPGM** and **DLTJVAPGM** commands on the System i5 server.

What You Should Be Able To Do

At the end of this exercise, you should be able to:

- Use **CRTJVAPGM** to decrease initial start up time
- Use **DSPJVAPGM** to analyze how a Java program was created
- Use **DLTJVAPGM** to delete an i5/OS Java program

Introduction

In order to make Java run faster on the System i5 server, Java class files (or JAR or ZIP files) are pre-processed to create an i5/OS Java Program object. The Java Program object is a pre-verified, expanded version of the Java class file, and it is stored attached to the Java class file. The i5/OS command **CRTJVAPGM** creates the Java Program. The original byte codes still exist in the integrated file system and the i5/OS Java Program is not visible in the integrated file system. Only commands like **CRTJVAPGM**, **DSPJVAPGM** and **DLTJVAPGM** allow you to interact with the Java Program. The i5/OS keeps track of the two versions (byte codes and i5/OS machine instructions) and discards the i5/OS Java Program if the byte codes are changed (for example if the program is recompiled or if you move a new copy from a client).

The i5/OS Java Program is created in one of two ways: by explicitly using the **CRTJVAPGM** command on a class or jar file, or when the file is first used. In this exercise you will explicitly create a Java Program to help reduce initial start up time for Java applications. **DSPJVAPGM** displays the attributes of a class or jar file in integrated file system. **DLTJVAPGM** will destroy the Java program that is associated with a class or jar file in the integrated file system (leaving the class or jar file intact).

Exercise Instructions

DSPJVAPGM, CRTJVAPGM

We will work with a small sample program that will examine a web page and determine which character occurs most frequently.

1. On the i5/OS command line, change the current directory to be the location of the class files:

   ```
   cd '/Javaxx'
   ```

   `**NOTE:  Javaxx is your user id.`

2. Now you can investigate the Java program associated with the SimpleProgram.jar file that we will be using in this exercise. You can examine the attributes of the Java program using the DSPJVAPGM command. To do this, type:

   ```
   DSPJVAPGM CLSF(SimpleProgram.jar)
   ```

   You should see the message “No Java program associated with the file.”

   Recall from the introduction that there are two ways that a Java program object can be created: either the object will be created automatically by the JVM the first time that a class file is used, or the object can be explicitly created by a call to CRTJVAPGM.

3. To explicitly create the Java program object, run the following command:

   ```
   CRTJVAPGM CLSF(SimpleProgram.jar)
   ```

   You should see the message “Java program created for “SimpleProgram.jar.””

4. To verify that a Java program object was successfully created, type:

   ```
   DSPJVAPGM CLSF(SimpleProgram.jar)
   ```

   Notice the various fields. You can see that this Java program contains information about a single Java class. Notice also that the optimization level reads 10. This is the default optimization level **when a Java program is created with the CRTJVAPGM command**.

5. Press Enter to return to the i5/OS command line.

6. Before we can run our Java application, we need to update the CLASSPATH environment variable as follows:

   ```
   CHGENVVAR CLASSPATH VALUE('/Javaxx:/Javaxx/SimpleProgram.jar')
   ```

   Note that any jar file that you want to reference must be explicitly listed in the CLASSPATH. Also notice that multiple jar files or directories are separated in the CLASSPATH using a colon.

7. To run the Java application, use the following command:

   ```
   JAVA CLASS(SimpleProgram) PARM('http://www.google.com')
   ```

   If you have an Internet connection, the above command will attempt to connect to the web page (www.google.com) and determine which character occurs the most on that page. Otherwise, the program will reference a static version of a web page stored in the jar file SimpleProgram.jar. You can try running the command with different parameters to see what happens. Always press the Enter key or F3 after each run to return to the i5/OS command line.

   Since the Java program that we attached to SimpleProgram.jar was created with optimization level 10 (and we took all of the defaults on the Java command), this application is running in mixed mode. That means that we are using both the JIT compiler and some of the statically compiled methods stored in the Java program.

DLTJVAPGM

If you delete a class file from the integrated file system, any associated Java program will be deleted as well. In comparison, DLTJVAPGM will remove the associated Java program, leaving the class file intact. Let's try it out now on the SimpleProgram.jar file.

1. Delete the associated Java program for SimpleProgram.jar:

   ```
   DLTJVAPGM CLSF(SimpleProgram.jar)
   ```

2. Your Java program should now be deleted. Let's use **DSPJVAPGM** to double check:

   ```
   DSPJVAPGM CLSF(SimpleProgram.jar)
   ```

   You should receive a message saying "No Java program associated with the file." The next time you run SimpleProgram, a new Java program will be created automatically by the Java Virtual Machine.

3. Let's verify that this actually happens. Type the following command:

   ```
   JAVA CLASS(SimpleProgram) PARM('http://www.yahoo.com')
   ```

The first message that you should see in the Java shell display is "Attaching Java program to /***Javaxx***/SimpleProgram.jar." Press Enter to exit the Java shell.

4. Now you can use DSPJVAPGM to examine the attributes of the new Java program:

```
DSPJVAPGM CLSF(SimpleProgram.jar)
```

5. Notice that, this time, the Java program was created with optimization level *INTERPRET. This is the default **when a Java program is created by the Java Virtual Machine**. If a Java program is created with optimization level *INTERPRET (either explicitly by the user or implicitly by the JVM), then those applications will be run using only the JIT compiler. Java program objects created with *INTERPRET are best suited to the JIT compiler because they contain all of the information that the JIT needs but they are faster to create and take up less space than Java program objects created at other optimization levels.

   Technically, this most recent run of SimpleProgram was actually slower than the first time we ran it. The first time we ran, the Java program object already existed, but this time, the object did not exist, so the JVM created it for us. This added some extra start up time, but since this is such a small application, the amount of time saved by running the CRTJVAPGM ourselves was negligible. However, for large applications, creating a Java program can be a fairly expensive operation, and so explicitly running CRTJVAPGM yourself can significantly improve your first time start up costs.

(Leave the emulator open. You will use it in other exercises.)

Exercise 4: Create and run a simple Java servlet

What This Exercise Is About

Servlets are Java programs that run in response to a request for a web page. When you type a URL in your browser window, the browser requests a web page from the web server. In many cases, that web page is simply a file sent to your browser. In the case of a servlet, the web server runs the servlet. The output of the servlet is a web page. Servlets allow you to easily generate web pages as they are requested - you can include up-to-the-minute information like inventory, stock price, or status information. In this exercise you will create a servlet which outputs the current date and time.

What You Should Be Able To Do

At the end of this exercise, you should be able to:

- Create a servlet
- Compile a servlet with javac
- Run the servletrunner test server
- Test a servlet with servletrunner

Introduction

When you create a Java servlet, you usually extend the *javax.servlet.http.HttpServlet* class, which is part of the Java Servlet Development Kit (JSDK). This is not part of the JDK, but rather it is a Java standard extension. This means that it is standardized, but is a separate download from java.sun.com. Many Java development tools include it. The javax.servlet.http.HttpServlet class does most of the work. Your class will modify its behavior only when needing to define the output of the servlet. In this case your class will output the current date and time in a browser.

Exercise Instructions

Edit and Compile The Source

1. Use Notepad to create CurrentInfoServlet.java. In the MS-DOS prompt type:

   ```
   notepad CurrentInfoServlet.java
   ```

   When Notepad asks you if you want to create a new file, click on **Yes**.

2. Enter the code as follows:

   ```
   import java.io.*;
   import java.util.Date;
   import javax.servlet.http.*;

   public class CurrentInfoServlet extends HttpServlet
   {
       public void doGet(HttpServletRequest req, HttpServletResponse res)
       throws IOException
       {
           res.setContentType("text/html");
           PrintWriter out = res.getWriter();
           out.println("Current date and time: " + new Date());
       }
   }
   ```

   Notice that there is a bit more to this program than the others that you wrote. The web server calls the servlet's doGet() method everytime a client requests it. You overrode the doGet() method to define what your servlet does every time it is called. First you declare that the response will be HTML text. The PrintWriter is used for any of your servlet's output. The output of the servlet can then be any HTML text (including tags). In this case, we just printed some plain text with the current date and time.

3. Save the file using the **File->Save** menu.
4. It is time again to compile your source code into Java byte codes using the javac compiler. Type the following command in the MS-DOS prompt:

   ```
   javac CurrentInfoServlet.java
   ```

5. If the source code compiles successfully, you will be presented again with the MS-DOS prompt. If there are errors in your source code, they will be displayed.
6. If you have any errors in your source code, go back to your Notepad window, fix the errors, save the file, and try to compile again.

Run the servletrunner test server

servletrunner is a test server which comes with the Java Servlet Development Kit (JSDK). It is not really a web server, but it is helpful when testing servlets. We will use it to test our servlet in action.

1. In your MS-DOS prompt, start servletrunner, telling it to use your lab directory as the location for servlets:

   ```
   servletrunner -d C:\JAVA
   ```

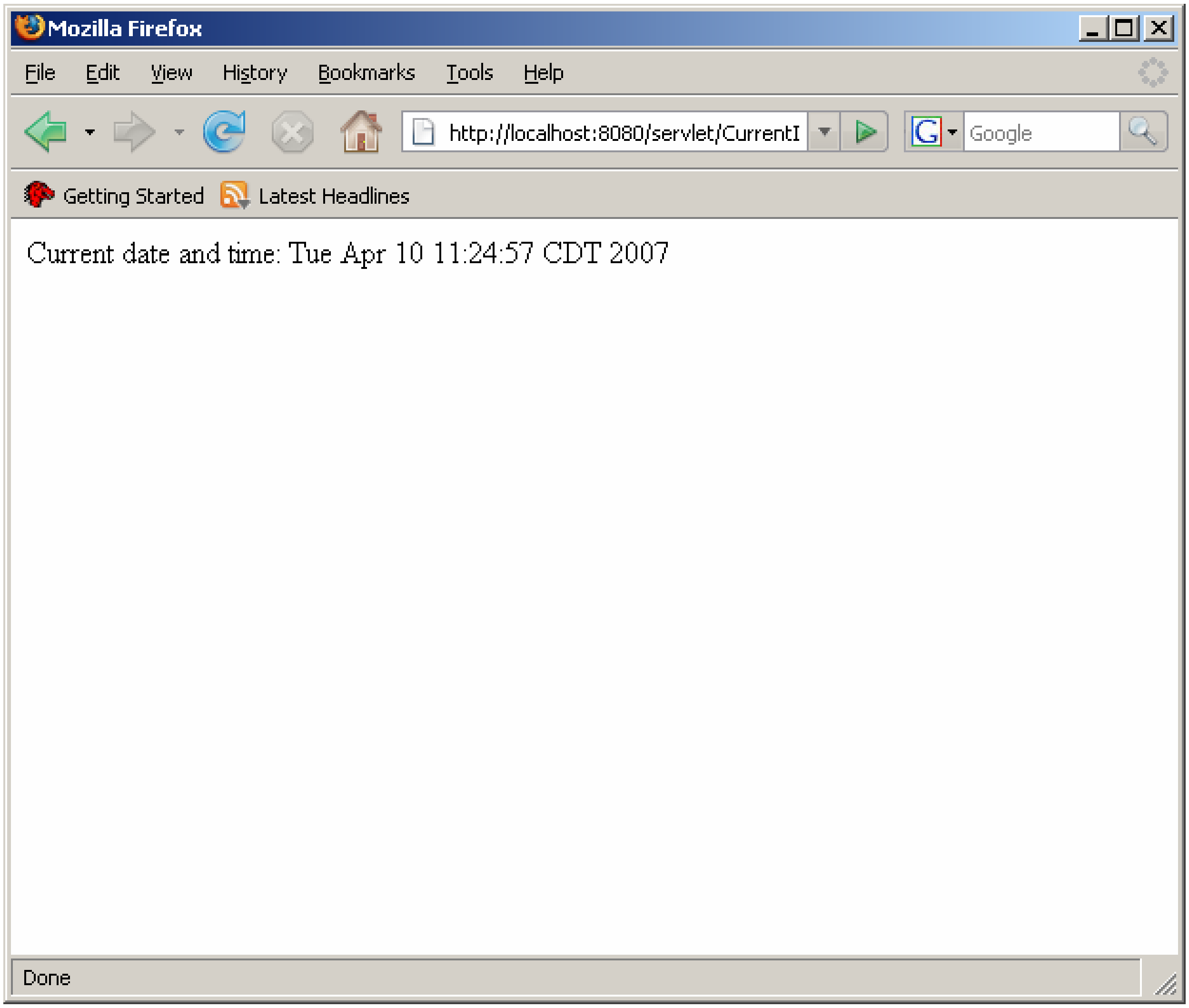
   The tool will print information about its running state. After you see this information, servletrunner is ready to use.

   ```
   servletrunner starting with settings:
   port = 8080
   backlog = 50
   max handlers = 100
   timeout = 5000
   servlet dir = C:\JAVA
   document dir = .\examples
   servlet propfile = C:\JAVA\servlet.properties
   ```

2. Bring up a browser window. You can use the icon for Mozilla Firefox located on the desktop. Enter the following URL.

   ```
   http://localhost:8080/servlet/CurrentInfoServlet
   ```

   The page should look like the following:


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/servlet/CurrentI
Google
Getting Started
Latest Headlines
Done

Current date and time: Tue Apr 10 11:24:57 CDT 2007

When the browser requests the page, the servlet runs on the web server and its output is sent to the browser window. Press the **Reload** button a few times and notice that the servlet runs each time you load the page.

Note: Normally the web server will be running on a central location and clients all over the network will be accessing it. In our small example, the client and server are running on the same machine.

****NOTE**: To stop the servletrunner press CTRL-C. In order to pick up any changes made to a class, the servletrunner must be restarted

Exercise 5: Use Qshell on a System i5 server

What This Exercise Is About

Introduce you to QShell on the System i5 server.

What You Should Be Able To Do

At the end of this exercise, you should be able to use QShell to compile and run Java programs.

Introduction

Many developers use the command line to develop, debug, and run Java programs in Windows and UNIX. The home for Java programs on the i5/OS is the Java shell. In one of the previous exercises, you used the i5/OS **java** command to run a Java application. You may have noticed that while running the program you were switched to the Java shell. You pressed F3 to exit the shell and return to the i5/OS command prompt. Another way to get into this Java shell is by running Qshell. Qshell is a full function shell environment, but for the purpose of this lab it is simply a convenient place to run Java programs. You can navigate the file system and compile and run Java programs in Qshell.

Exercise Instructions

Using QShell

From the i5/OS command prompt, enter **QSH**. This starts the shell interpreter. If you are used to Windows or UNIX, the shell has a familiar look and feel. Do the following from the shell:

1. Set up the CLASSPATH environment variable. Enter

```
export CLASSPATH=/JAVA/Javaxx
```

2. Change directory to the directory containing your lab exercises. Enter:

```
cd /JAVA/Javaxx
```

3. Display the contents of the directory. Qshell is like a UNIX shell. Enter:

```
ls
```

4. Run your Java program. Enter:

```
java CurrentInfo
```

5. Press F3 when you are ready to exit Qshell

Exercise 6: Create a simple web page

What This Exercise Is About

Before we are able to create a Java applet, you will need a web page in which to embed it. In this lab you will create a web page using HyperText Markup Language (HTML).

What You Should Be Able To Do

At the end of this exercise, you should be able to create a web page, and then view it using a browser.

Introduction

Web pages are written using a scripting language called HyperText Markup Language (HTML). Learning HTML is comparable to learning another programming language. Since our page only has a little text and formatting, we will enter the HTML tags by hand. HTML editing tools are available which allow WYSIWYG (What-You-See-Is-What-You-Get) editing of web pages.

Why are you creating a web page in a Java programming lab? In the next exercise you will create a Java applet. Java applets *always* run embedded in a web page in the context of a web browser. In order to run your Java applet, you need to have a web page for it.

Exercise Instructions

Our page will contain only a header and a hypertext link. The link may not work if the PCs in the lab are not connected to the web.

Create The Web Page

1. In the MS-DOS prompt, edit a new file called MyPage.html. This will contain the HTML which describes the contents of your web page:

```
notepad MyPage.html
```

2. When Notepad asks you if you want to create a new file, click on **Yes**.
3. Type in the following HTML:

```
<HTML>
<BODY>
<H1>My Web Page</H1>
<P>Click
<A HREF="http://www.ibm.com/java">
here</A> for information about Java.
</BODY>
</HTML>
```

4. Save the file using the **File->Save** menu.

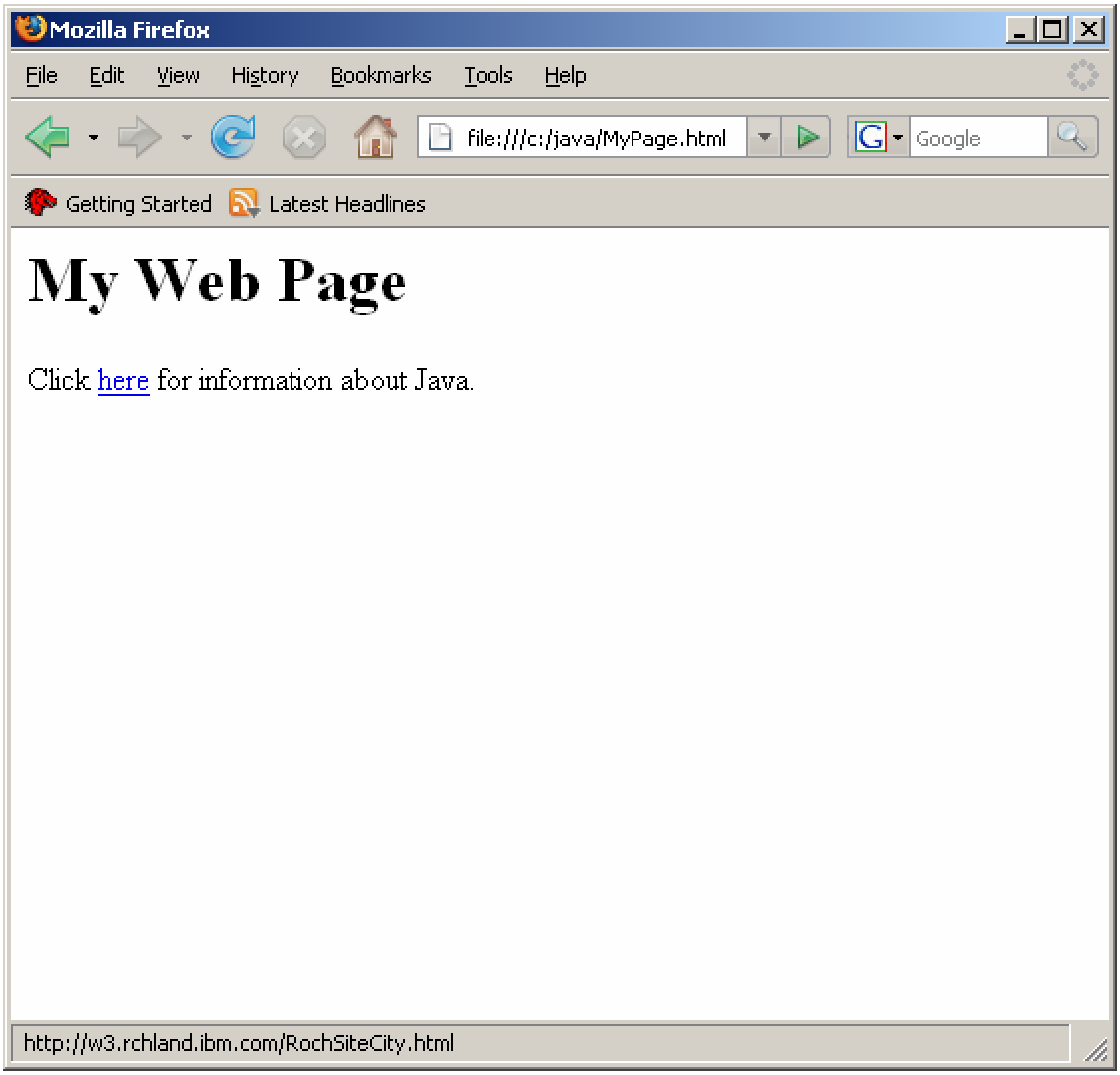
View The Web Page

Let’s see what your web page looks like in the browser.

1. Start Mozzila Firefox using the icon on the desktop.
2. Open your web page using the **File->Open File** menu, then enter

   ```
   C:\JAVA\MyPage.html
   ```

   Your page should appear in the main browser window:



My Web Page

Click here for information about Java.

http://w3.rchland.ibm.com/RochSiteCity.html

Exercise 7: Create and run a simple Java applet

What This Exercise Is About

Applets are Java programs that are dynamically downloaded from a web server and run inside a browser. In this exercise you will update your Java program so it runs as an applet as well as an application.

What You Should Be Able To Do

At the end of this exercise, you should be able to:

- Create an applet
- Compile an applet with javac
- Add an applet inside of an existing HTML page
- Test a web page with an applet

Introduction

One of the powerful features of Java is the variety of environments available in which to run a program. In exercise 1 you created a Java application. In this exercise you will update this application so it can also run as an applet. In practice, Java code is often separated between the business logic and the presentation. You can write the business logic once, then easily display data as an application, applet, or servlet.

When you create a Java applet, you must extend the *java.applet.Applet* class, which is part of the Java class library available in every Java environment. This means you are going to use the *java.applet.Applet* class as a reusable part. java.applet.Applet will do most of the work. Your class will modify its behavior only where appropriate. In this case your class will print out the current date and time in a browser.

Exercise Instructions

Edit and Compile The Source

1. Use Notepad to edit CurrentInfo.java. (You may still have the window up from a previous exercise.)

```
notepad CurrentInfo.java
```

2. Update the source code as follows. **The changes to make are in bold**. Please note the punctuation as well as the capitalization.

```
import java.applet.Applet;
import java.awt.Label;
import java.util.Date;

public class CurrentInfo extends Applet
{
   public static void main(String[] args)
   {
      System.out.println("Current date and time: " + new Date());
   }

   public void init()
   {
      add(new Label("Current date and time: " + new Date()));
   }
}
```

3. Save the file using the **File->Save** menu.
4. It is time again to compile your source code into Java byte codes using the javac compiler. Type the following command in the MS-DOS prompt that you used earlier:

```
javac CurrentInfo.java
```

5. If the source code compiles successfully, you will be presented again with the MS-DOS prompt. If there are errors in your source code, they will be displayed.
6. If you have any errors in your source code, go back to your Notepad window, fix the errors, save the file, and try to compile again.

Insert The Applet Into Your Web Page

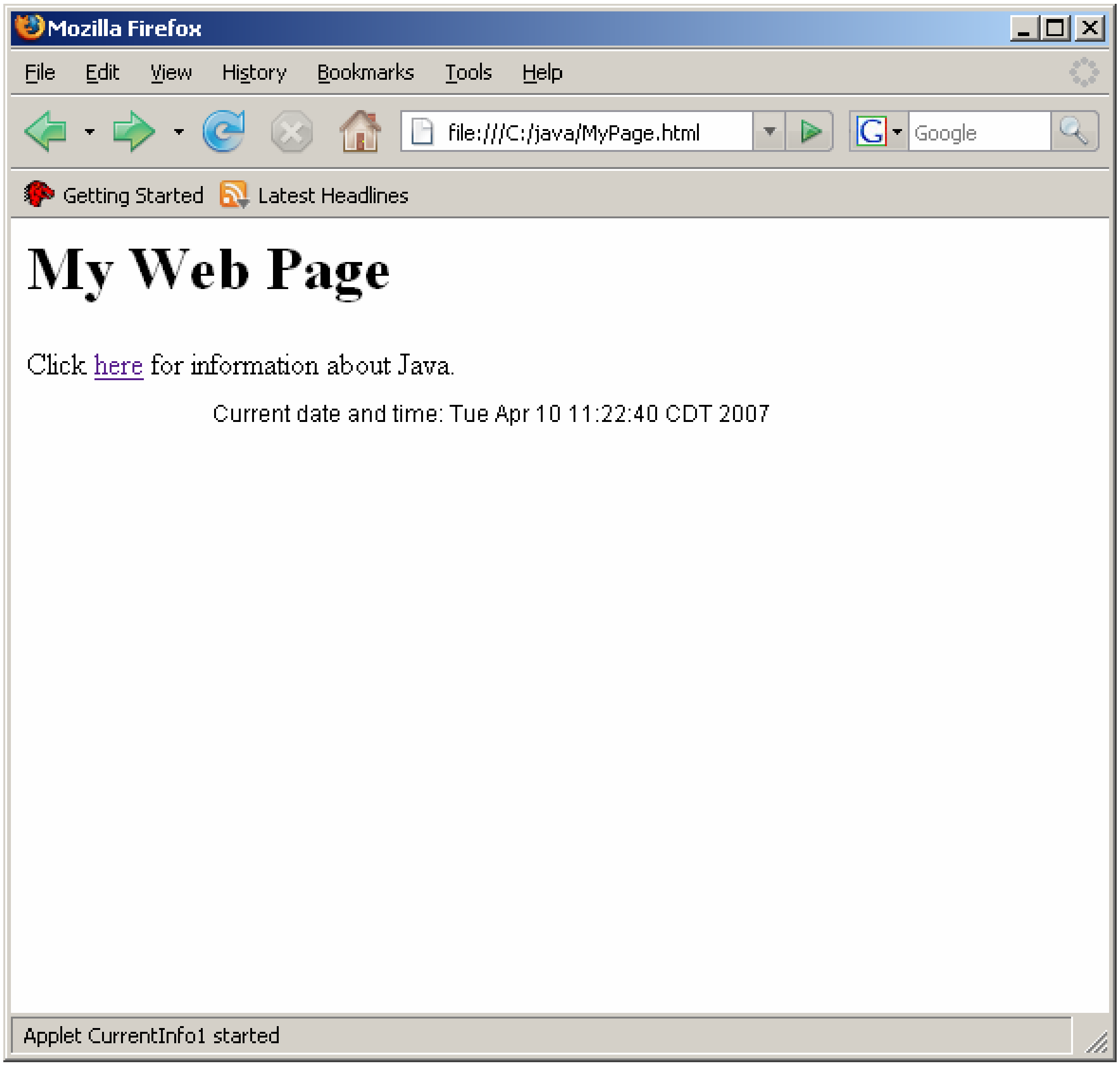
1. Use Notepad to edit MyPage.html. (You may still have the window up from a previous exercise.)

```
notepad MyPage.html
```

2. Update the HTML as follows. **The changes to make are in bold**. These lines will result in your applet being embedded as part of the web page.

```
<HTML>
<BODY>
<H1>My Web Page</H1>
<P>Click
<A HREF="http://www.ibm.com/java">
here</A> for information about Java.
<BR>
<APPLET CODE="CurrentInfo.class" WIDTH=500 HEIGHT=300>
</APPLET>
</BODY>
</HTML>
```

3. Save the file using the **File->Save** menu.
4. If the browser window is still up from the previous exercise, click the reload button in the toolbar. Otherwise, use the instructions from the previous exercise to start and load the page. The page should look like the following:



My Web Page

Click here for information about Java.

Current date and time: Tue Apr 10 11:22:40 CDT 2007

Applet CurrentInfo1 started

When the browser processes the APPLET tag it starts its internal Java Virtual Machine and automatically runs the CurrentInfo class. During applet processing, the init() method is called which is when our class defines its user interface.

Conclusion

In this lab, you had the opportunity to code simple Java programs in many environments. You saw a Java application, a Java applet, and a Java servlet. In addition, you ran a Java application on a System i5 server and learned about i5/OS Java program optimization.

Your next step is to delve deeper into the Java language itself. There are many books, classes, and online tutorials to help you with this. As with any programming language, the key to learning Java is to use it. Choose a simple project that will help you (a checkbook register, to-do list, fantasy football, etc) or your team (vacation scheduler, problem tracker, etc) and work on it a little at a time. As you add features, you can learn new Java skills, such as File I/O, network access, or graphical user interfaces.

Appendix A: Bonus Exercise

Bonus Exercise 1: Create and run a simple Java application using an Integrated Development Environment

What This Exercise Is About

Integrated development environments (IDEs) make programming easier by assisting programmers while developing applications. In this exercise you will create and run a simple Java application using the Eclipse IDE.

What You Should Be Able To Do

At the end of this exercise, you should be able to:

- Create a Java project in Eclipse
- Create a Java application using Eclipse
- Run a Java application in Eclipse

Introduction

Integrated development environments often provide syntax highlighting, point and click compile and run, integrated help text and more. While many IDEs are typically designed for a specific programming language, there are many other multiple language IDEs available. Eclipse is an open source, multi-language, and multi-platform IDE. In exercise 1, you created a Java application. In this exercise, you will create and run that same Java application using the Eclipse IDE.

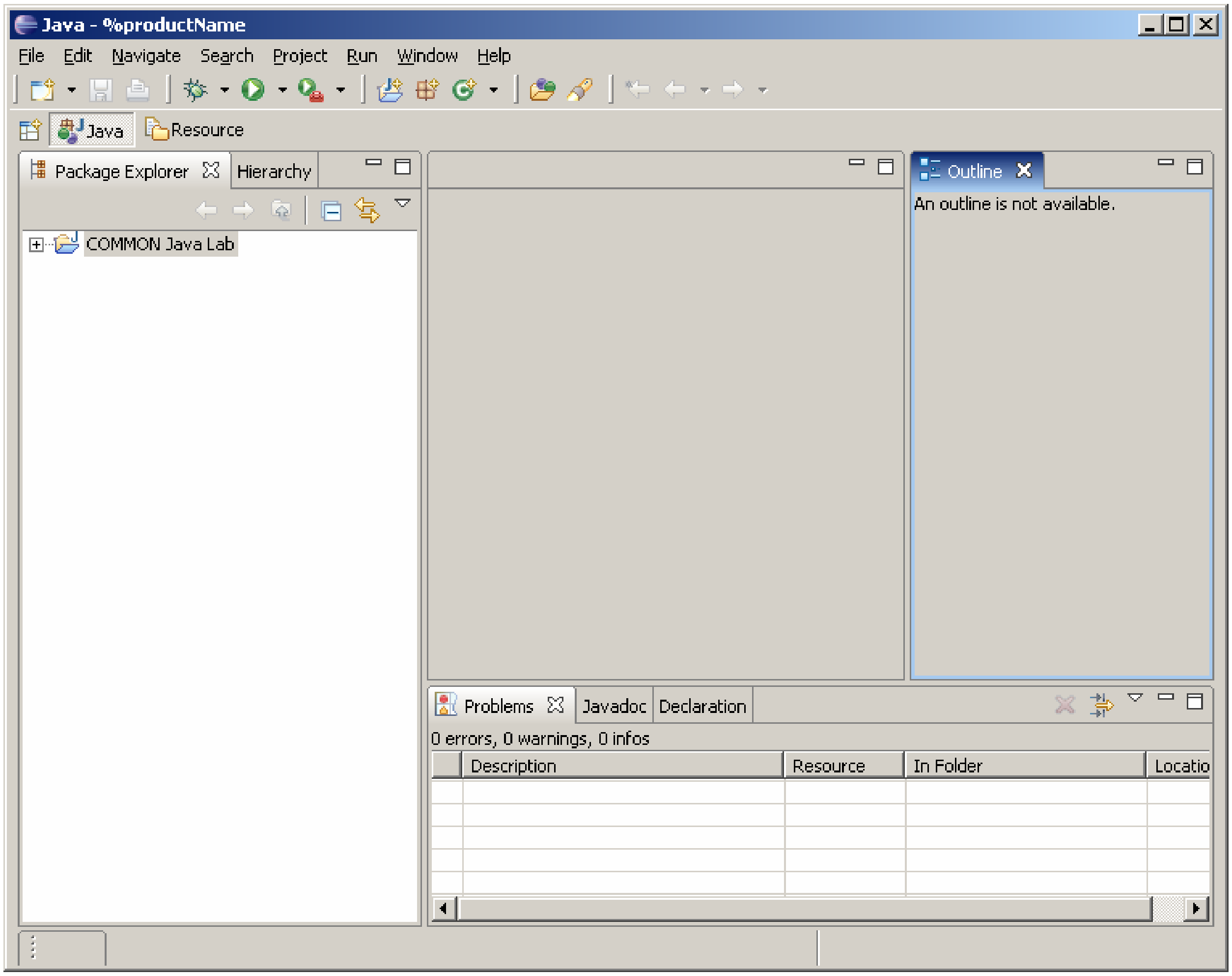
This exercise is not intended to teach you all about the IDE. It is provided to demonstrate how an IDE can assist you in developing Java applications.

Note: In order to use Eclipse on your PC, a Java Runtime Environment (JRE) needs to be installed. Eclipse and a JRE have already been installed for you in this lab. You can download Eclipse for your own use from http://www.eclipse.org.

Exercise Instructions

Start the Eclipse IDE and Create a Java Project

1. Start Eclipse. You can use the icon for Eclipse located on the desktop.
2. Eclipse should start and the Workspace Launcher window should be displayed. Enter **C:\JAVA** for the name of your workspace.
3. Click the OK button in the Workspace Launcher window. The Eclipse IDE should be displayed. If the Welcome Screen is displayed, you may close it for this exercise.
4. Create a Java Project using the **File -> New -> Project** menu.
5. In the New Project window, make sure **Java Project** is highlighted and click the Next button. The New Java Project window should be displayed.
6. In the New Java Project window, enter **COMMON Java Lab** in the **Project name** field and click the Finish button. Click the Yes button in the Open Associated Perspective dialog. The Java perspective will be displayed and it should look similar to this:

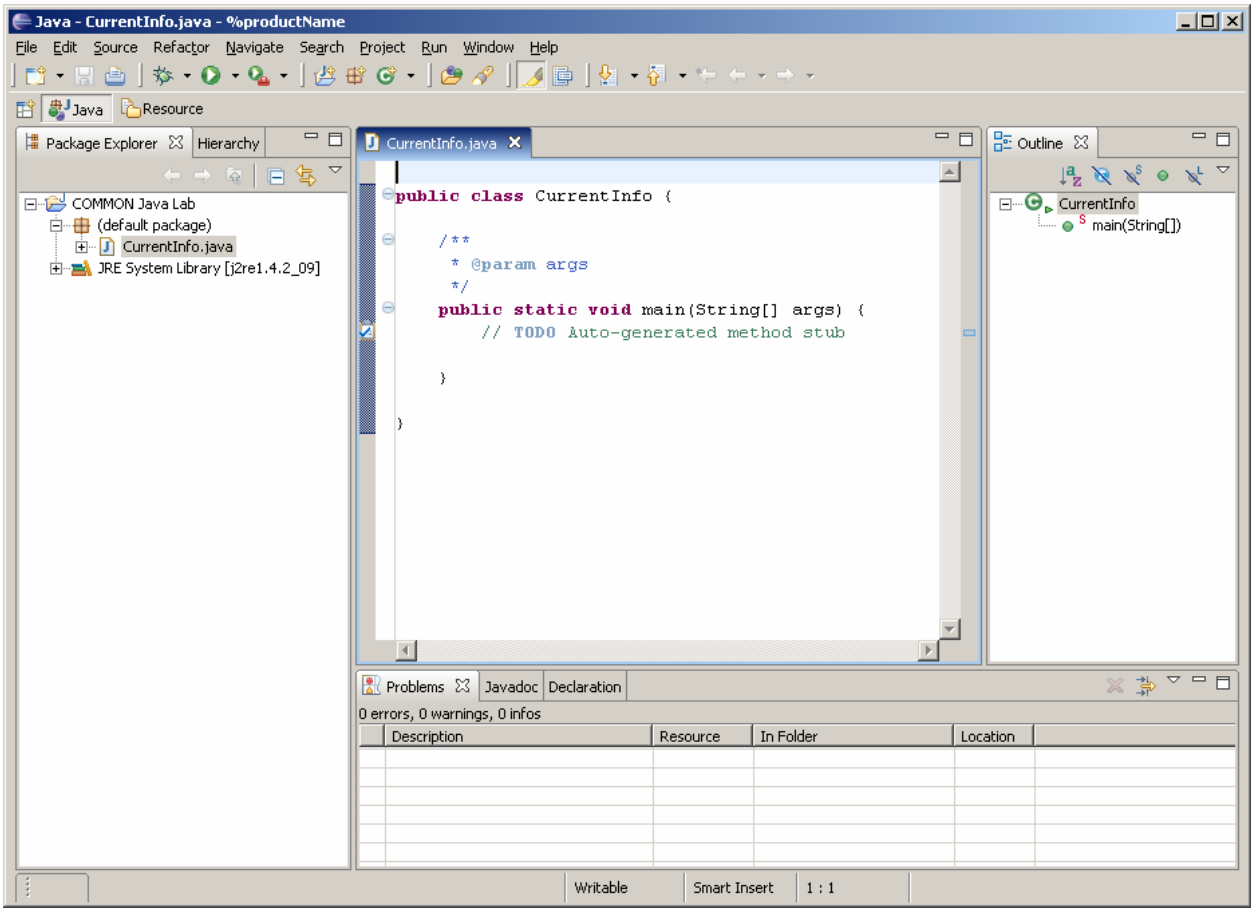


An outline is not available.

0 errors, 0 warnings, 0 infos

Create a Java source file

1. Create a Java source file using the **File -> New -> Class** menu. The New Java Class window will be displayed.
2. Enter **CurrentInfo** in the **Name** field and make sure to select the checkbox next to the public static void main(String[] args) method in the method stub section. Click the Finish button.
3. A CurrentInfo.java class will be created with an empty main method. Your window should like the following:



```
public class CurrentInfo {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

0 errors, 0 warnings, 0 infos

Edit the Java source file

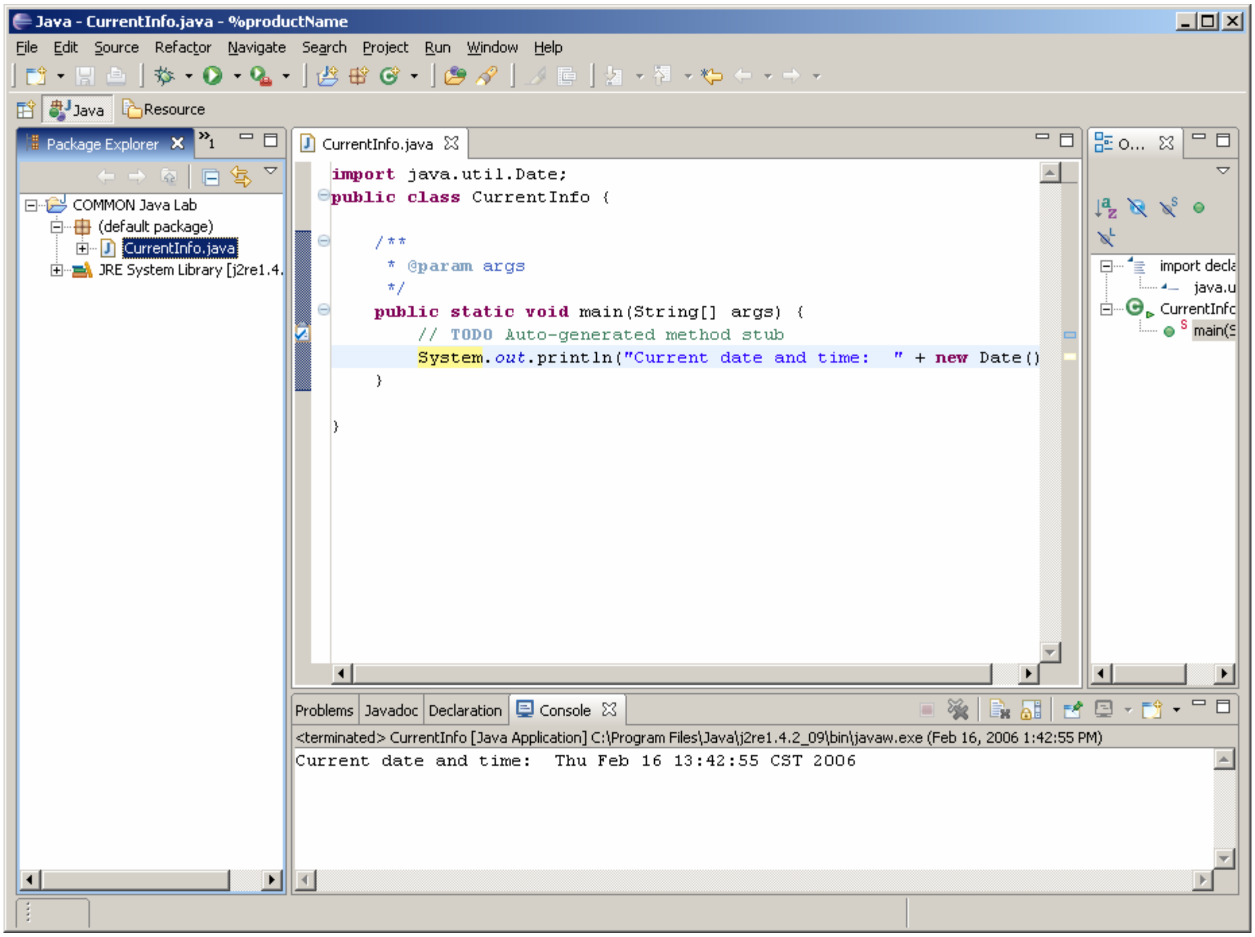
1. Add the **System.out.println("Current date and time: " + new Date());** statement inside of the main method.
2. Take note of the syntax highlighting and how the IDE assists you in writing your statements.
3. Notice how the word Date is underlined in red and a light bulb with a red x appears next to the line. This indicates there is an error on the line. If you move your mouse pointer over the word Date, you will see a box that indicates that "Date cannot be resolved to a type." This is because we have not imported the java.util.Date class into our java source file.
4. Move you mouse to the light bulb with the red x and left click on it. Two little pop-up windows appear with some options that can help resolve the error on the line. Select the **Import 'Date' (java.util)** item by double clicking on it. This will add the **import java.util.Date;** statement to the top of the java source file. The light bulb with the red x to the left of the line should now be gone. Your java source file should look like the following:

```
import java.util.Date;

public class CurrentInfo {

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		System.out.println("Current date and time:  " + new Date());
	}
}
```

5. Save the file by using the **File -> Save** menu. If any red x's appear to the left of a line of code you still have an error. Fix the error and try to save the file again.

Run the Application

1. In the Package Explorer window on the left, right-click on CurrentInfo.java. A menu should appear.
2. Run the application by using the **Run As -> Java Application** menu.
3. The application will run in the Console window at the bottom of the IDE. It should look similar to the following:



```
import java.util.Date;
public class CurrentInfo {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Current date and time:  " + new Date()
    }

}
```

```
<terminated> CurrentInfo [Java Application] C:\Program Files\Java\j2re1.4.2_09\bin\javaw.exe (Feb 16, 2006 1:42:55 PM)
Current date and time:  Thu Feb 16 13:42:55 CST 2006
```

4. To re-run the program you can simply click the Run icon (green circle with white arrow) in the Toolbar.
5. Leave Eclipse open for use during the next exercise.

Bonus Exercise 2: Create and Generate Javadoc™ for a Java class

What This Exercise Is About

This exercise demonstrates how to create Javadoc™ comments for a class and its methods and then generate and publish the Javadoc for such a class. In this exercise, you will build on the application you created in Bonus Exercise 1 by adding Javadoc comments to the class file and expanding the auto-generated Javadoc comments of the main method. You will then run the javadoc command to generate the Javadoc for the class and then display it using a web browser.

What You Should Be Able To Do

At the end of this exercise, you should be able to:

- Create Javadoc comments for a class, field and method
- Run the javadoc command to generate the Javadoc
- Display the Javadoc you created using a web browser

Introduction

Javadoc is the mechanism Java uses to provide thorough information to Java developers about classes and APIs available for their use. Javadoc is displayed in HTML format with hypertext links to other classes and APIs. This exercise uses Bonus Exercise 1 as a prerequisite. You will edit the Javadoc comments in the class, save the source, and then run the javadoc command to generate the Javadoc for the class. You will then use a web browser to display the Javadoc you just created for your class.

This exercise does not generate Javadoc at the package level, however, generating Javadoc at the package level is similar when using the correct options and arguments of the javadoc command.

Note: The Javadoc tool is one of the Java SDK tools. If you are able to compile a Java class on a system, then the SDK is installed on that system and the Javadoc tool would be installed.

Exercise Instructions

Edit the class' Javadoc comments

1. You see the following lines of comments above the main method of the Currentlnfo.java class you created in Bonus Exercise 1.

   ```
   /**
    * @param args
    */
   ```

   Edit it to look like this.

   ```
   /**
    * This is the main method of the CurrentInfo class.  It represents the main
    * program entry for this application.  It displays the current date and time at the
    * system console.
    *
    * @param args array of Strings - the main program arguments
    */
   ```

   Note: All Javadoc comments start with /** and end with */

2. Edit the class Javadoc comments by inserting the following block right above the class declaration statement.

   ```
   /**
    * A basic Java class with only a <code>main</code> method.  <br> This class
    * displays the current date and time.
    *
    * @author [Type your name here]
    * @version 5.4
    * @since 4.1
    */
   ```

   The class declaration statement is

   ```
   public class CurrentInfo {
   ```

   Notice the impeded <br> and <code> html tags. You can customize the way your documentation looks in the web browser by impeding various html tags. Javadoc tags start with the @ sign, eg @auther, @version and @since.

3. Save your changes using the **File -> Save** menu.

4. Your source file should look like the following:

```
import java.util.Date;

/**
 * A basic Java class with only a <code>main</code> method.  <br> This class
 * displays the current date and time.
 *
 * @author [Type your name here]
 * @version 5.4
 * @since 4.1
 */

public class CurrentInfo {

	/**
	 * This is the main method of the CurrentInfo class.  It represents the main
	 * program entry for this application.  It displays the current date and time at the
	 * system console.
	 *
	 * @param args array of Strings - the main program arguments
	 */

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		System.out.println("Current date and time: " + new Date());
	}

}
```

Run the javadoc command to generate your Javadoc API documentation

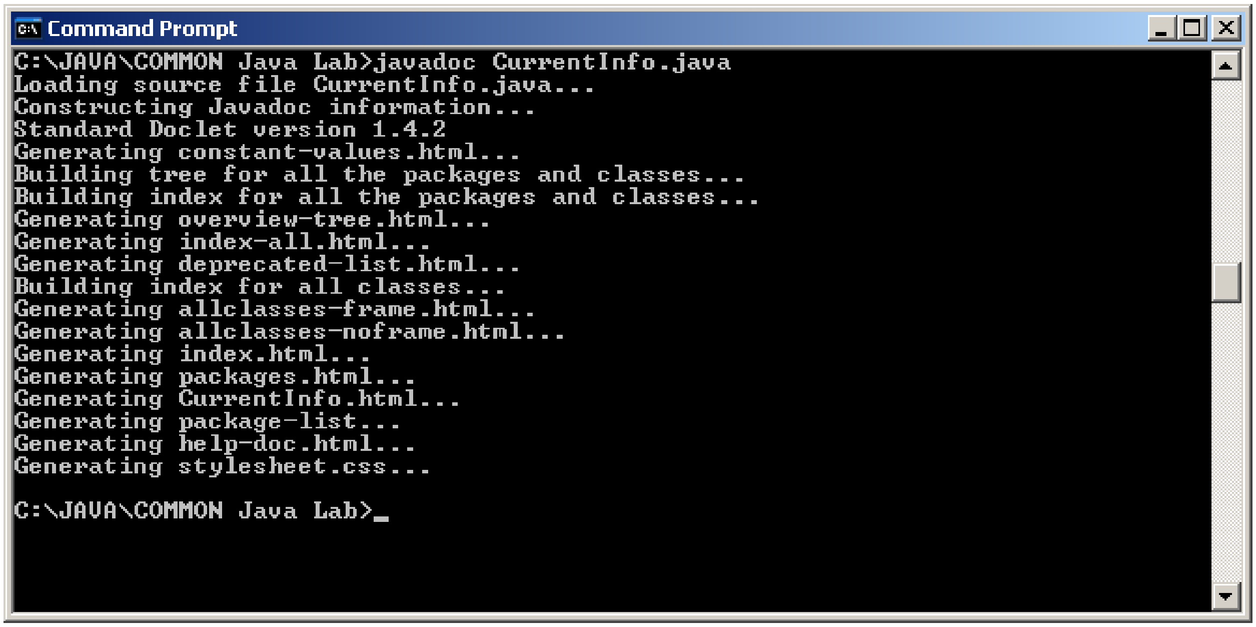
1. Before you run the tool, open Windows® Explorer. Select the Start->All Programs->Accessories->Windows Explorer. Type “C:\JAVA\COMMON Java Lab” in the address bar to open that directory. Observe there is no .html file in there. Leave this window open, you will use it later.
2. Open a MS-DOS prompt if one is not still open from a previous exercise. Select the Start->All Programs->Accessories->Command Prompt and change the current directory to C:\JAVA\COMMON Java Lab.

```
cd c:\JAVA\COMMON Java Lab
```

3. Generate Javadoc for the CurrentInfo.java source file. (Remember: Java is case-sensitive!) Note: If you opened a new MS-DOS prompt make sure to set your PATH environment variable like you did in Exercise 1.

   ```
   javadoc CurrentInfo.java
   ```

   The result of the javadoc command should look similar to this:



```
C:\JAVA\COMMON Java Lab>javadoc CurrentInfo.java
Loading source file CurrentInfo.java...
Constructing Javadoc information...
Standard Doclet version 1.4.2
Generating constant-values.html...
Building tree for all the packages and classes...
Building index for all the packages and classes...
Generating overview-tree.html...
Generating index-all.html...
Generating deprecated-list.html...
Building index for all classes...
Generating allclasses-frame.html...
Generating allclasses-noframe.html...
Generating index.html...
Generating packages.html...
Generating CurrentInfo.html...
Generating package-list...
Generating help-doc.html...
Generating stylesheet.css...

C:\JAVA\COMMON Java Lab>_
```

Display the Javadoc in a Web browser

1. Go to the Windows® Explorer window you opened in the last instruction and make sure it still displays the contents of C:\JAVA\COMMON Java Lab.
2. Notice the many .html files and the resources directory that were added due to running the javadoc command.
3. Locate the index.html file and double click on it to open it with your default web browser.
4. Notice that the main method name in the Method Summary table is hyperlinked and that only the first sentence of the method description is displayed in the table cell.
5. Click on the name of the main method. The detail should now be displayed. Notice that all of the Javadoc comments you typed in the file are displayed in HTML format including all of the method and parameter descriptions.
6. When finished, close all open windows. You are now done with the lab work.