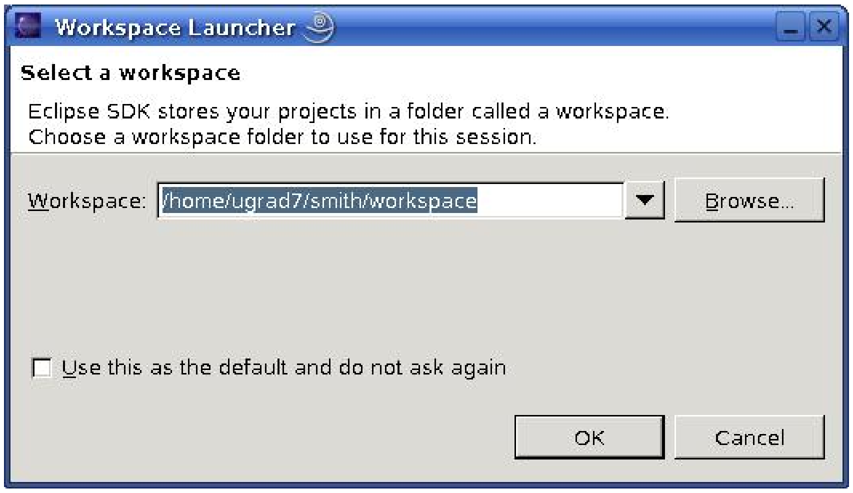
Eclipse Tutorial

Step 1: Starting Eclipse (on CS Linux machines)

In order to use Eclipse on CS Linux machines you need to source /local/config/cshrc.eclipse to get it into your path. You should add "source /local/config/cshrc.eclipse" to your .cshrc file to have Eclipse in your path every time you log in to your CS account. Start Eclipse by typing "eclipse &".

The first screen that comes up after opening Eclipse is the Workspace Launcher. This screen allows you to choose where the projects you create will be saved. You may use the default, or specify a different location.


Workspace Launcher

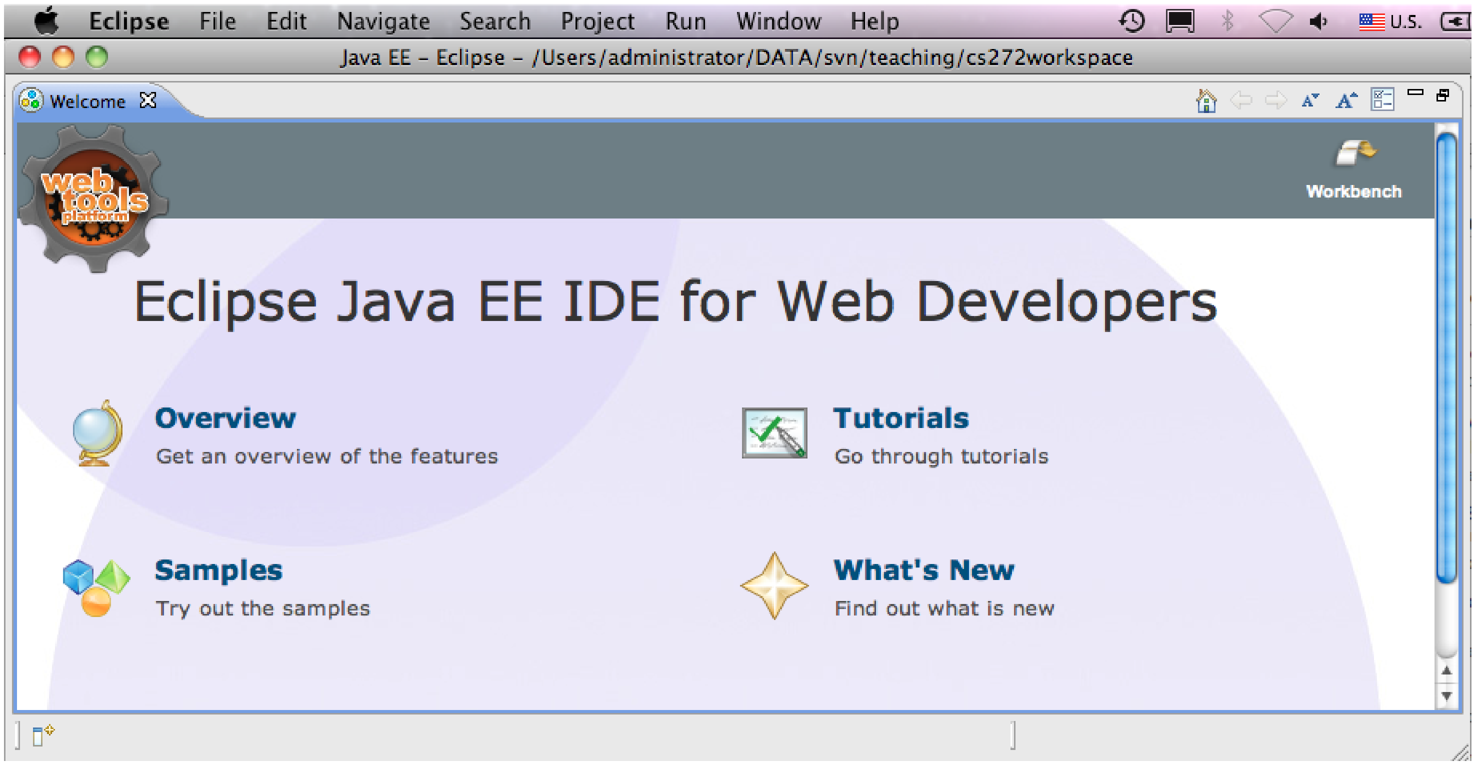
Select a workspace

Eclipse SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: /home/ugrad7/smith/workspace

☐ Use this as the default and do not ask again

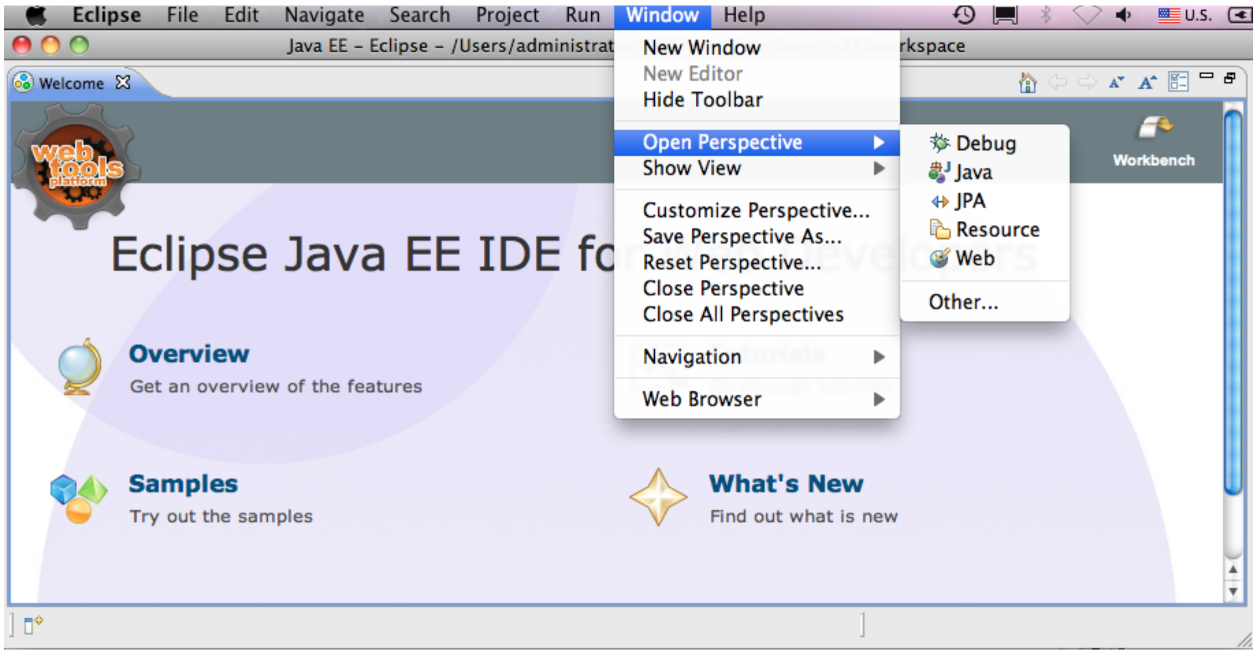
After selecting a directory you should see this window


Java EE - Eclipse - /Users/administrator/DATA/svn/teaching/cs272workspace

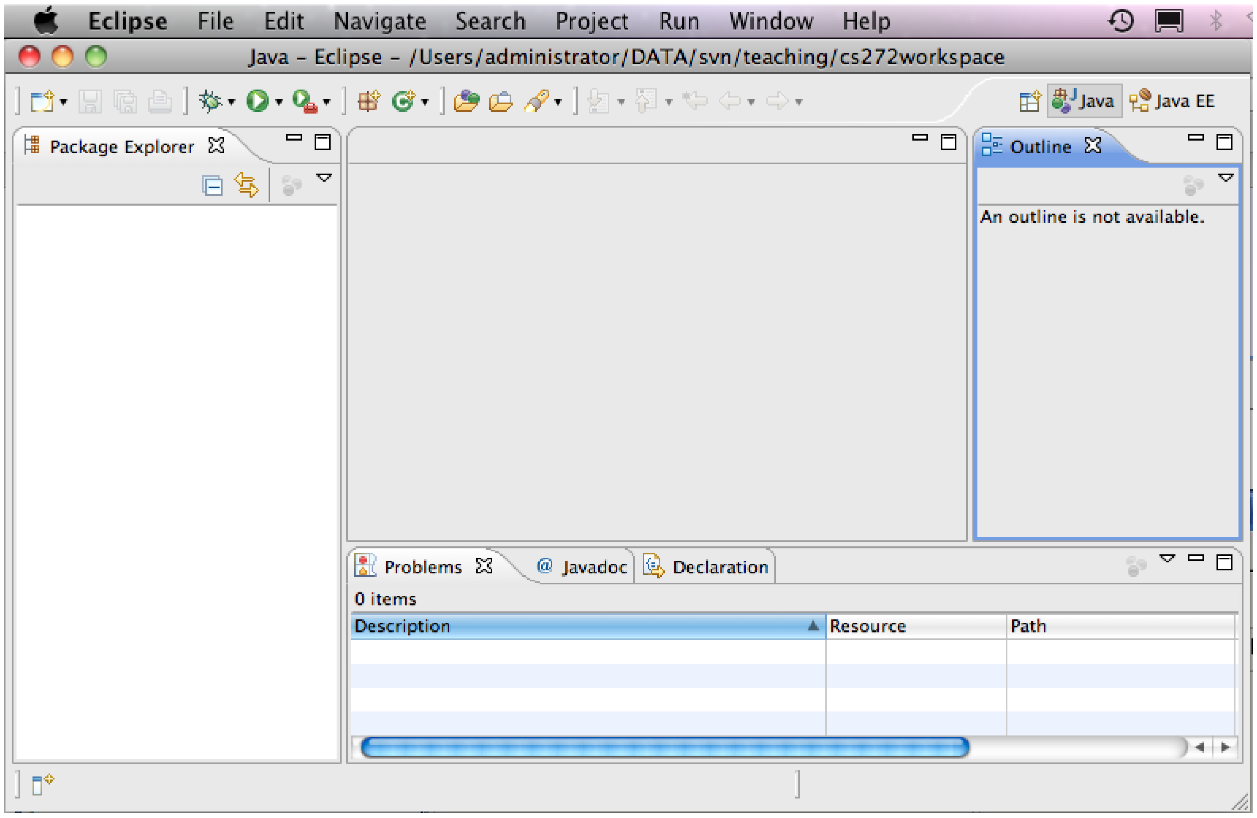
Eclipse Java EE IDE for Web Developers

Step 2: Choosing a Perspective

A perspective defines the initial set and layout of views in the window. Perspectives control what appears in certain menus and toolbars. For example, a Java perspective contains the views that you would commonly use for editing Java source files, while the Debug perspective contains the views you would use for debugging Java programs. You may switch perspectives, but you need to specify an initial perspective for a workspace. To create Java programs, set the Java perspective by choosing *Window, Open Perspective, Java* from the main menu, as shown below:



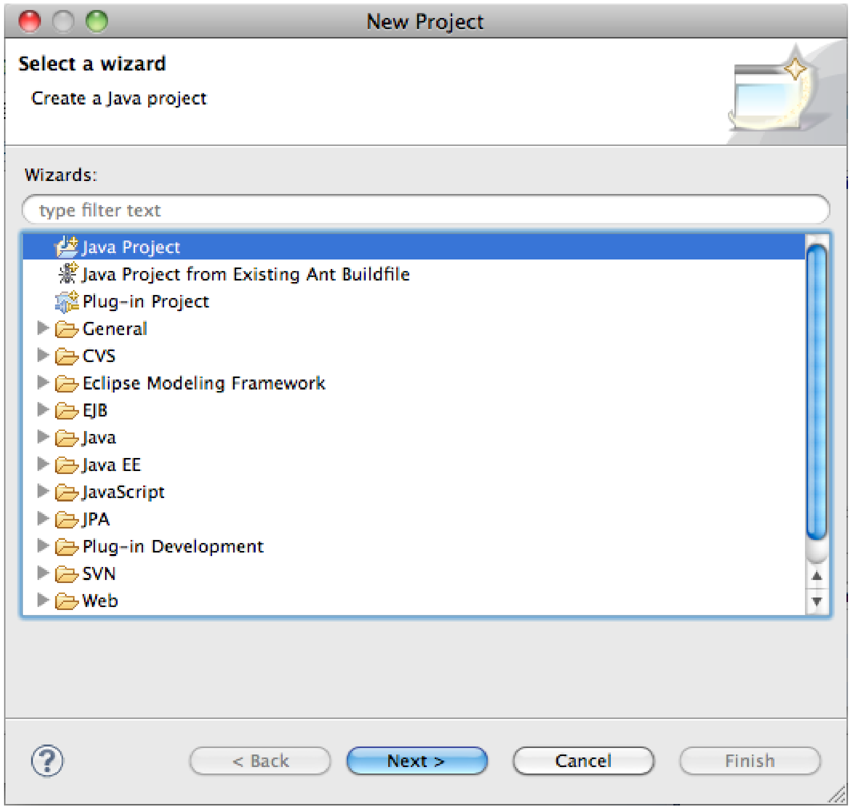
Close the Welcome window and you will see the Eclipse user interface displayed according to the perspective:



Eclipse File Edit Navigate Search Project Run Window Help
Java – Eclipse – /Users/administrator/DATA/svn/teaching/cs272workspace
Java
Java EE
Package Explorer
Outline
An outline is not available.
Problems
Javadoc
Declaration
0 items
Description
Resource
Path

Step 3: Creating a New Java Project

To create a project, choose *File, New, Project* to display New Project wizard, as shown below:


New Project

Select a wizard

Create a Java project

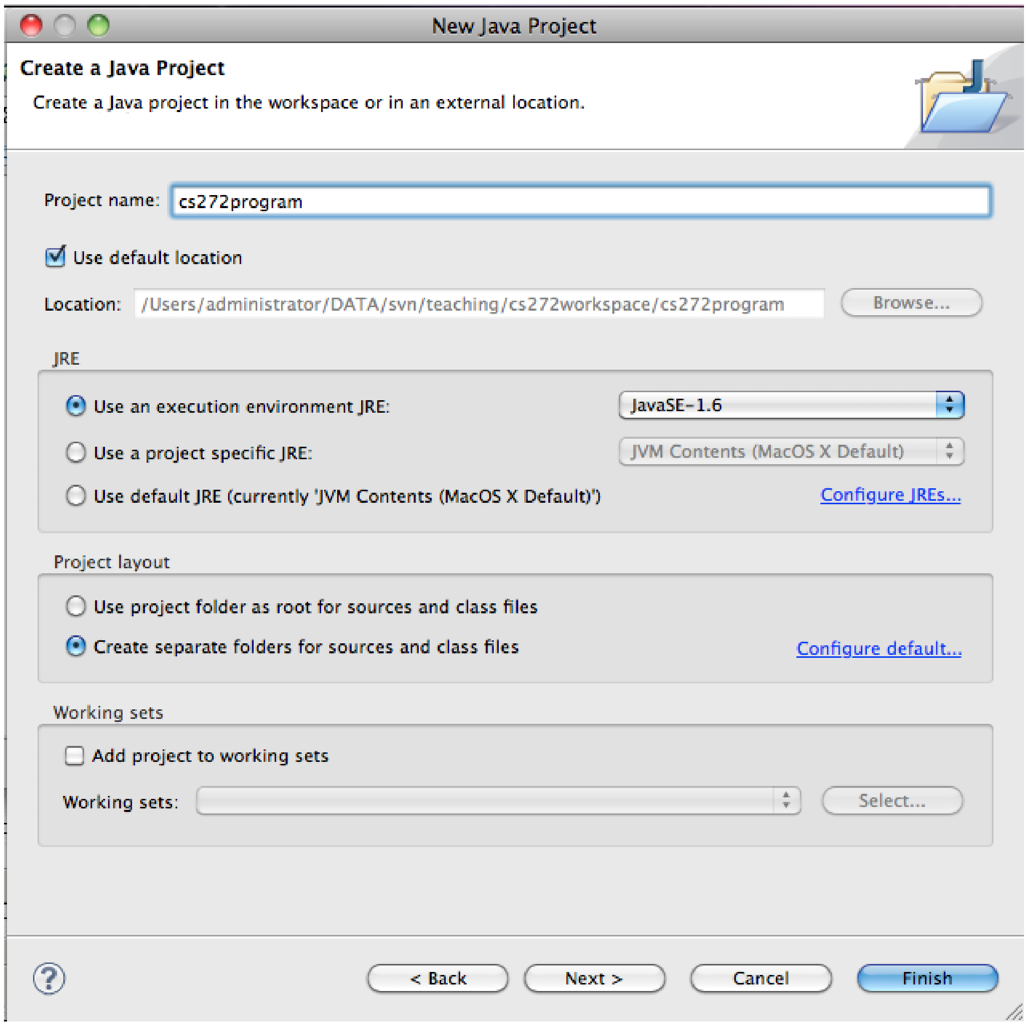
Wizards:

type filter text

- Java Project
- Java Project from Existing Ant Buildfile
- Plug-in Project
- General
- CVS
- Eclipse Modeling Framework
- EJB
- Java
- Java EE
- JavaScript
- JPA
- Plug-in Development
- SVN
- Web

< Back | Next > | Cancel | Finish

Select *Java Project* and click *Next* to display New Java Project wizard, as shown below. Type *cs272program* in the Project name field. Make sure that you selected the options "*Create new project in workspace*" and "*Use project folder as root for sources and class files*". Click *Finish* to create the project.


New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: cs272program

[✓] Use default location

Location: /Users/administrator/DATA/svn/teaching/cs272workspace/cs272program   Browse...

JRE

(●) Use an execution environment JRE: JavaSE-1.6

( ) Use a project specific JRE: JVM Contents (MacOS X Default)

( ) Use default JRE (currently 'JVM Contents (MacOS X Default)')   Configure JREs...

Project layout

( ) Use project folder as root for sources and class files

(●) Create separate folders for sources and class files   Configure default...

Working sets

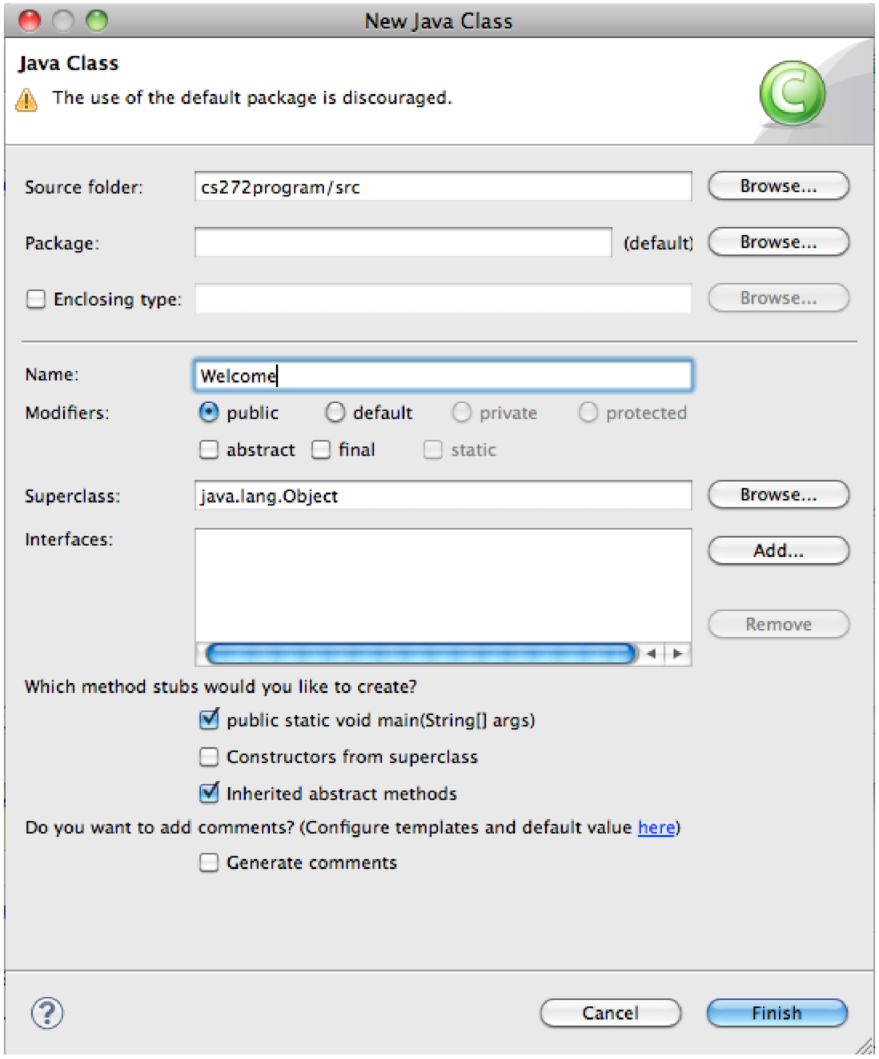
[ ] Add project to working sets

Working sets: Select...

< Back | Next > | Cancel | Finish

Step 4: Creating a Program in the Project

Now you can create a program in the project by choosing *File, New, Class* to display the New Java Class wizard as shown below. Type *Welcome* in the Name field. Check the option *public static void main(String[] args)*.


New Java Class

Java Class

The use of the default package is discouraged.

Source folder: cs272program/src

Package: (default)

Enclosing type:

Name: Welcome

Modifiers: [✓ public] default private protected

abstract final static

Superclass: java.lang.Object

Interfaces:

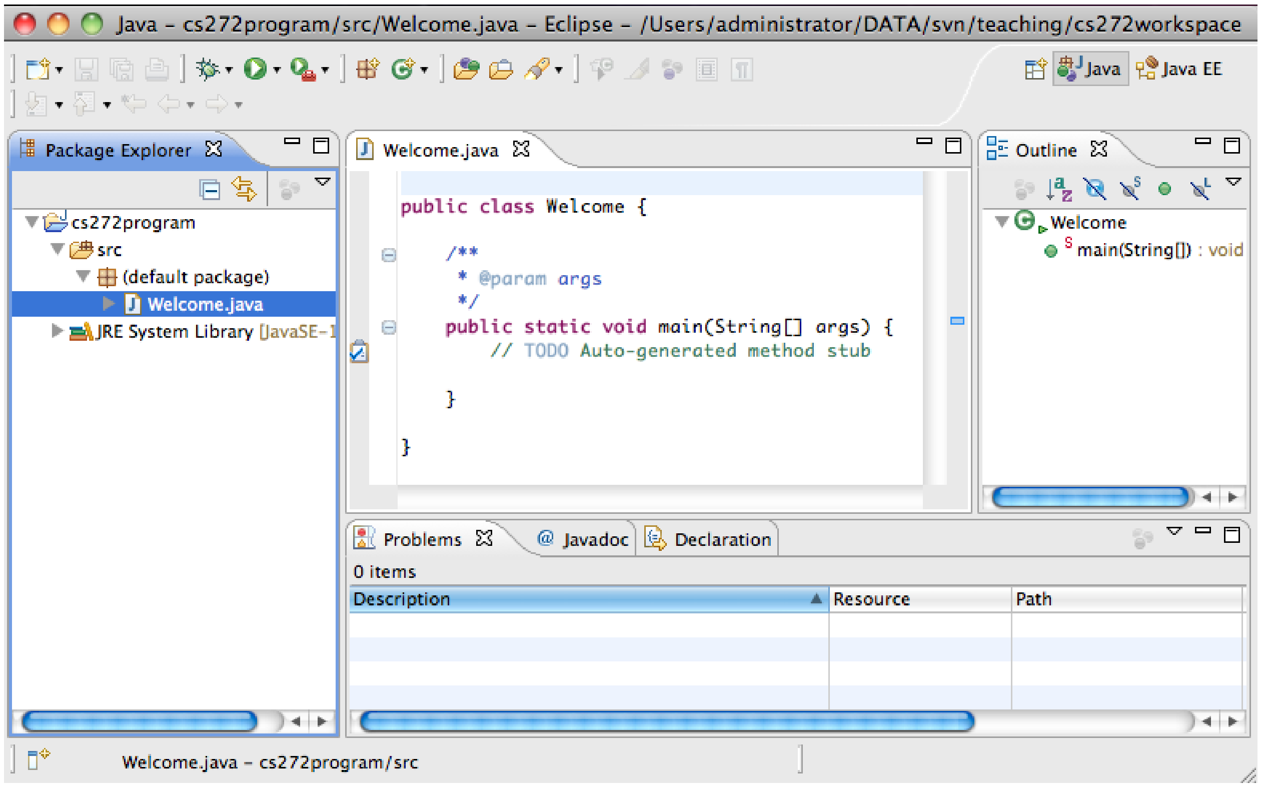
Which method stubs would you like to create?

- [✓] public static void main(String[] args)
- [ ] Constructors from superclass
- [✓] Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

- [ ] Generate comments

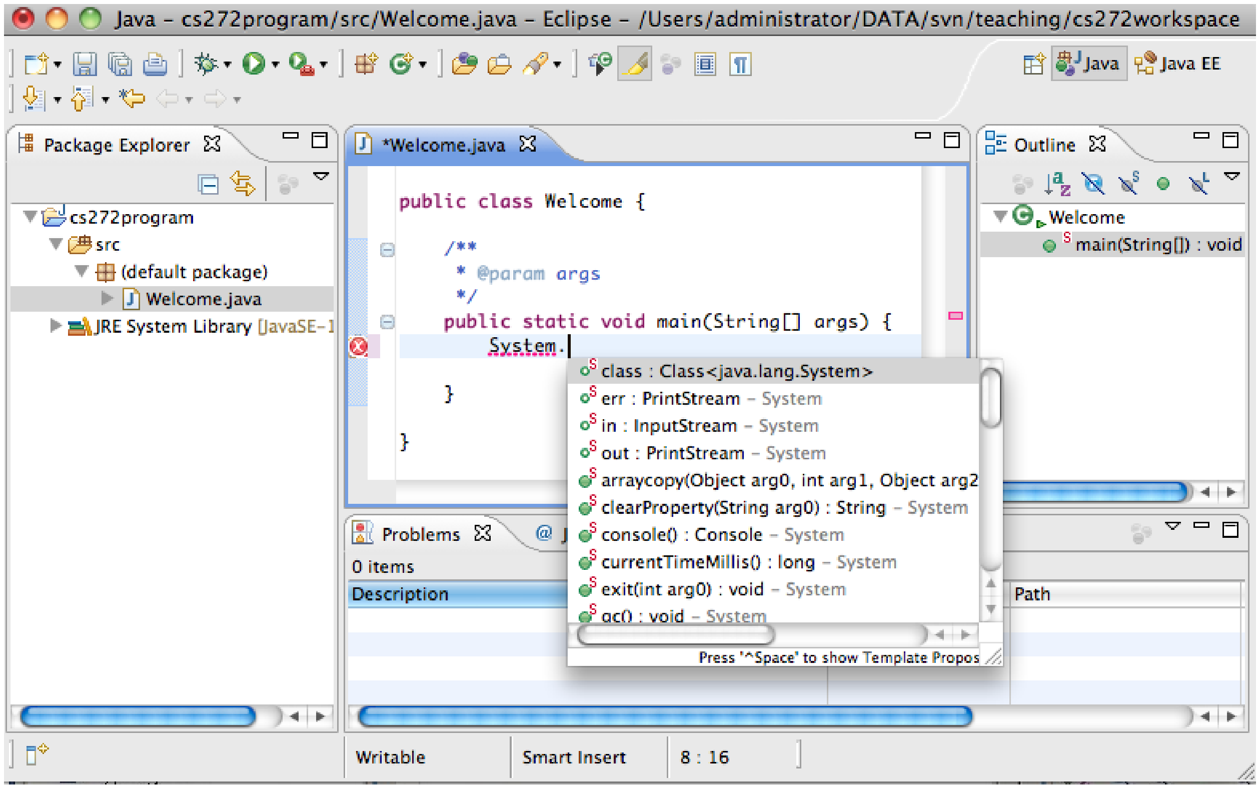
Click *Finish* to generate the template for the source code *Welcome.java*, as shown below:



```
public class Welcome {

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

    }

}
```

Type *System.out.println("Welcome to Java");* in the main method.

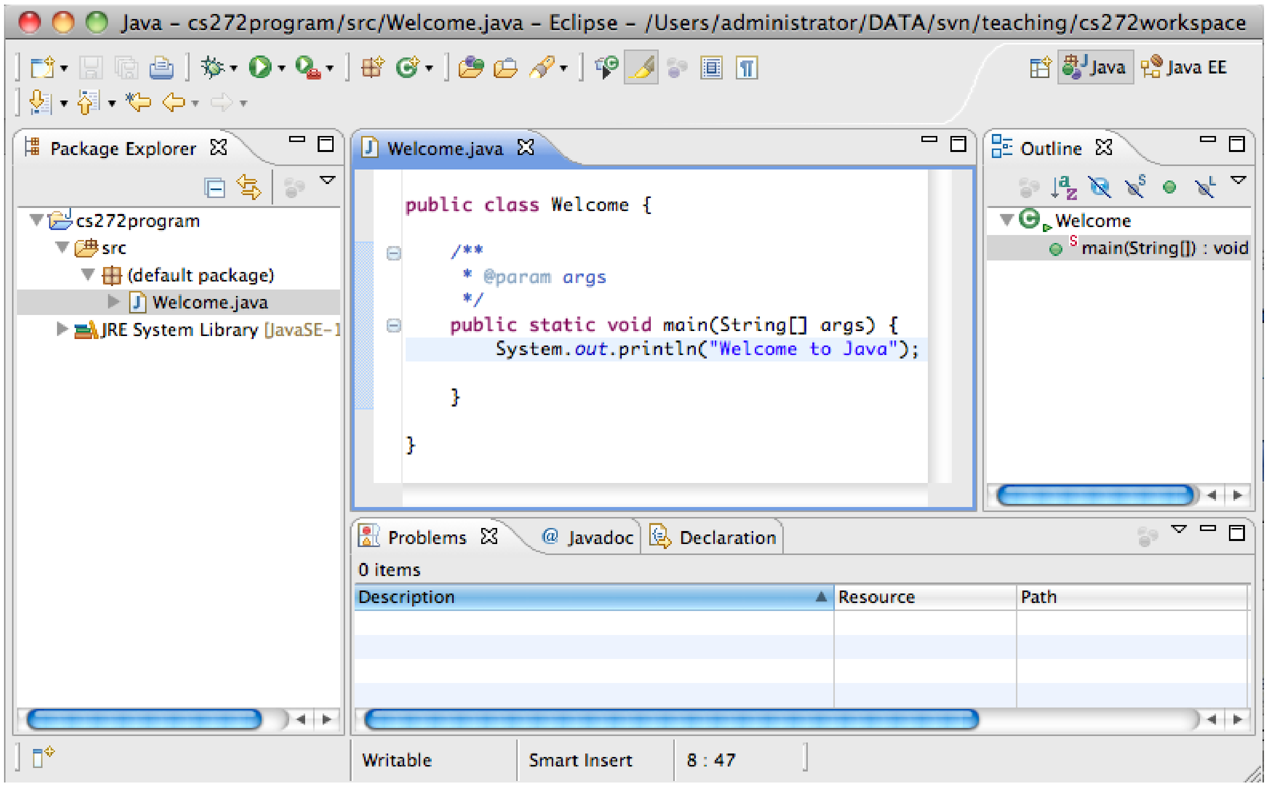
Note: As you type, the code completion assistance may automatically come up to give you suggestions for completing the code. For instance, when you type a dot (.) after *System* and pause for a second, Eclipse displays a pop up menu with suggestions to complete the code, as shown below. You can then select the appropriate item from the menu to complete the code.


Java - cs272program/src/Welcome.java - Eclipse - /Users/administrator/DATA/svn/teaching/cs272workspace
Java
Java EE
Package Explorer
cs272program
src
(default package)
Welcome.java
JRE System Library [JavaSE-1
*Welcome.java
Outline
Welcome
main(String[]) : void
class : Class<java.lang.System>
err : PrintStream - System
in : InputStream - System
out : PrintStream - System
arraycopy(Object arg0, int arg1, Object arg2
clearProperty(String arg0) : String - System
console() : Console - System
currentTimeMillis() : long - System
exit(int arg0) : void - System
gc() : void - System
Press '^Space' to show Template Propos
Problems
0 items
Description
Path
Writable
Smart Insert
8 : 16

```
public class Welcome {

    /**
     * @param args
     */
    public static void main(String[] args) {
        System.

    }

}
```

Step 5: Compiling and Running a Program

By default, your source code is dynamically compiled as you type. For example, if you forgot to type the semicolon (;) to end the statement, as shown below, you will see the red wriggly line in the editor pointing to the error:

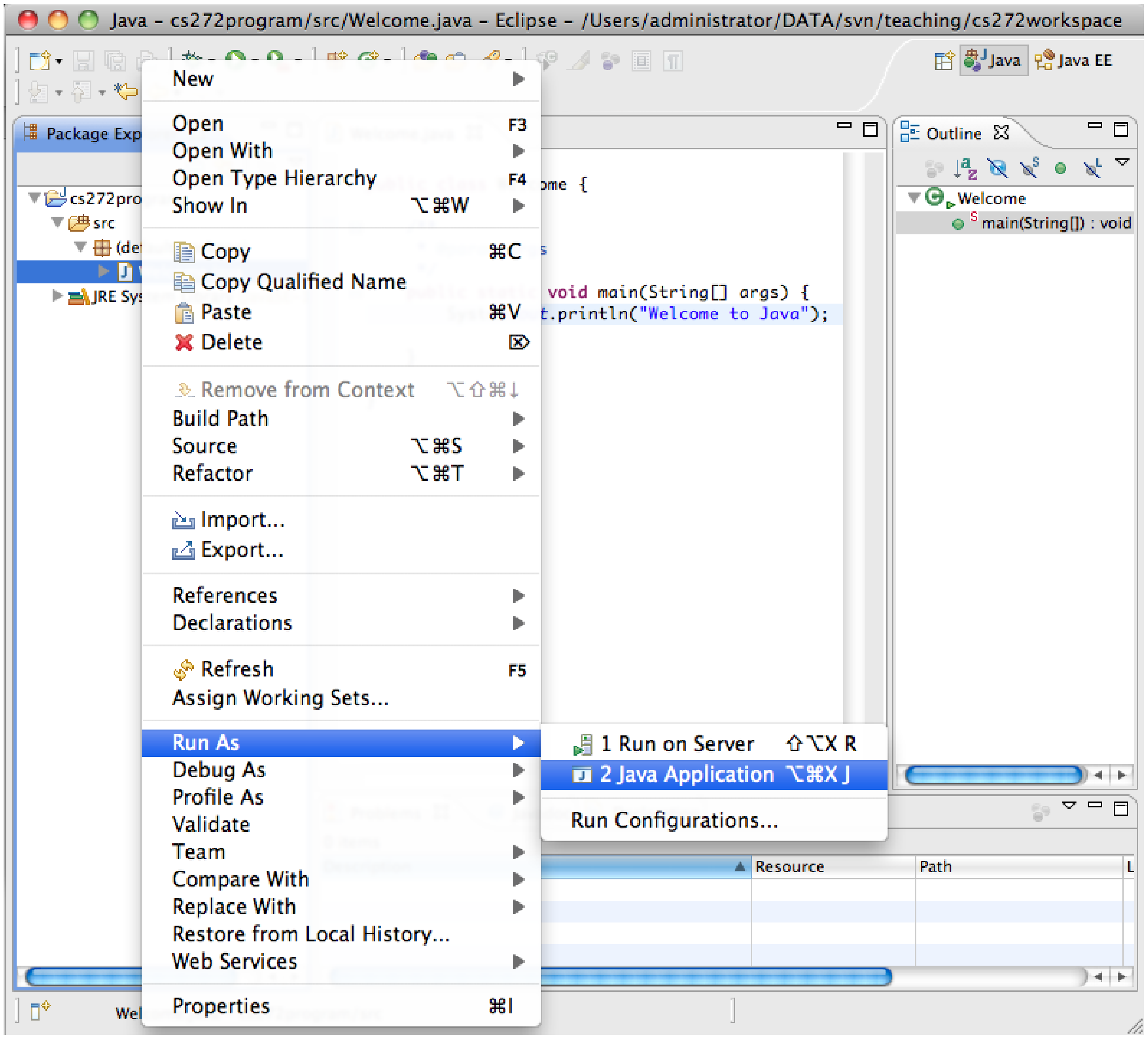


```
public class Welcome {

    /**
     * @param args
     */
    public static void main(String[] args) {
        System.out.println("Welcome to Java");

    }

}
```

0 items

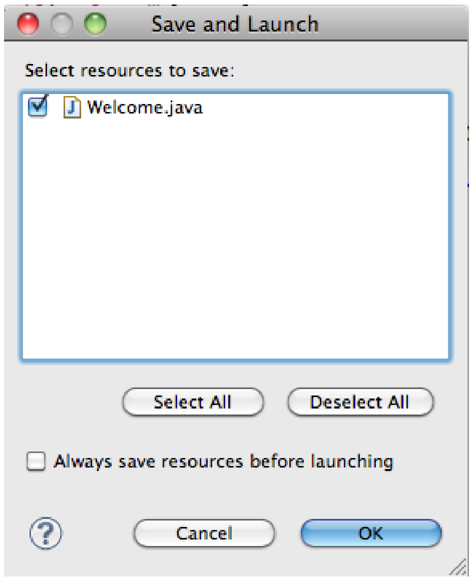
Description	Resource	Path

To run the program, right-click the class in the project to display a context menu, as shown below. Choose *Run as, Java Application* in the context menu to run the class.

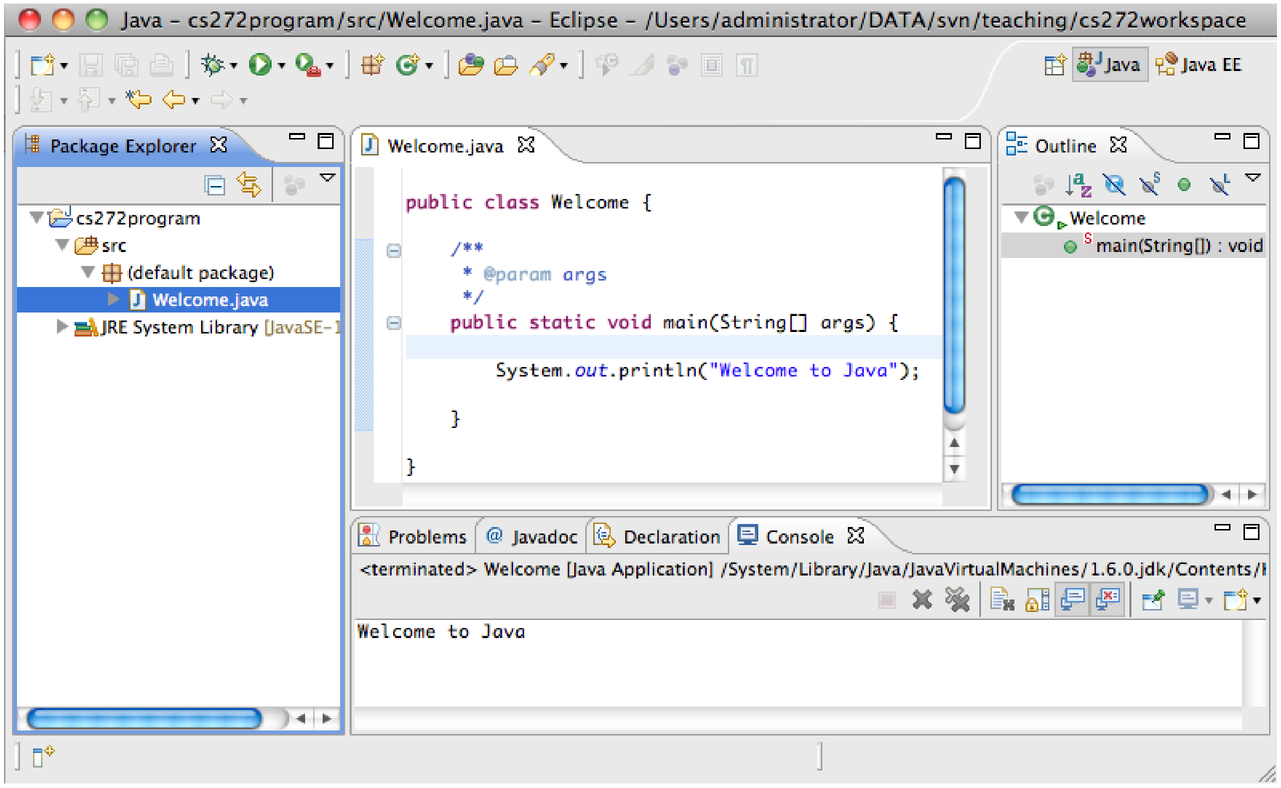


```
ome {
void main(String[] args) {
t.println("Welcome to Java");
```

Before executing the program a window appears allowing you to save your program if it was not saved yet:


Save and Launch
Select resources to save:
☑ Welcome.java
Select All
Deselect All
☐ Always save resources before launching
Cancel
OK

The output is displayed in the Console pane, as shown below:



```
public class Welcome {

    /**
     * @param args
     */
    public static void main(String[] args) {

        System.out.println("Welcome to Java");

    }

}
```

```
<terminated> Welcome [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/I
Welcome to Java
```

Another way to run a program is to select *Run, Run as, Java Application* from the main menu.

Step 6: Compile and Run Java Applications from the Command Line

You can also compile and run program standalone directly from the operating system. Here are the steps to run the *Welcome* application from the prompt.

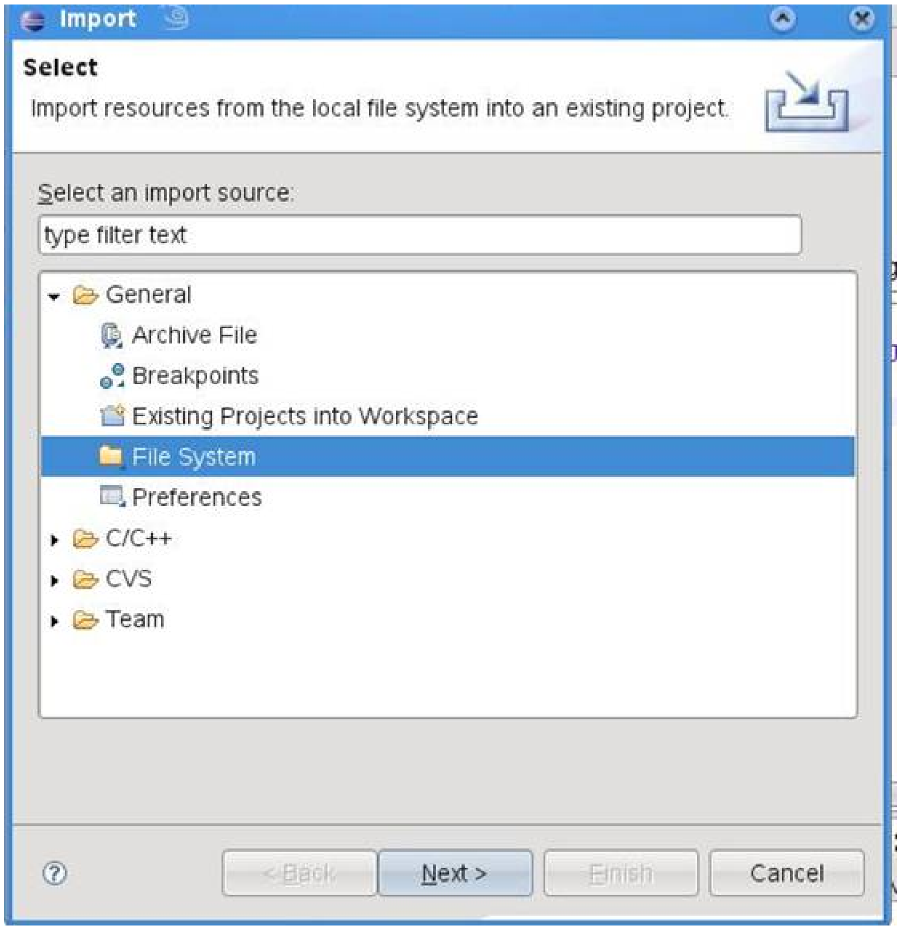
1. Type **cd workspace/cs272program** to change the directory to **workspace/cs272program**.
2. Type **java Welcome** to run the program. A sample run of the output looks like the

   following:`% java Welcome`
   ```
   Welcome to Java
   ```
3. You may also compile the program using the **javac** command at the prompt:

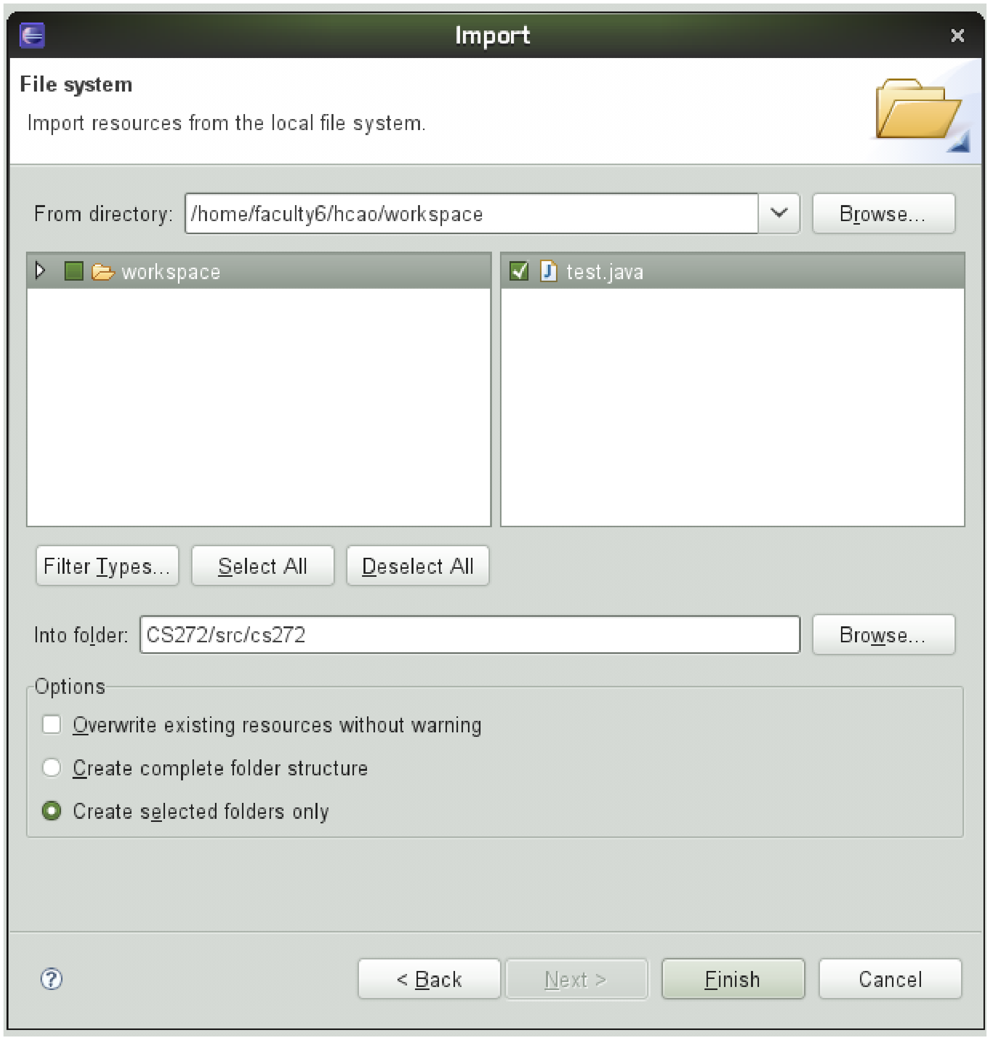
   ```
   % javac Welcome.java
   ```

Step 7: Importing Existing Resources

Another way to add files to a project is to import them from the local file system. This is done when you already have files and want to incorporate them into the project. First, create a new project as described earlier. To practice, create a new project with the name *showtime*. Then, select *File, Import* from the main menu. The import wizard appears and gives you many different ways to import resources, we are interested in importing from the file system.



Choose *File system* in the window and click *Next*. The window that appears allows you to select a directory to retrieve files from and a directory to place files in. You may type in the name of the directory or click the *Browse* button to search for it. Type */home/faculty5/ipivkina/cs272* in the From directory field. Click on the left panel to have the directory appear on the left panel as shown below. After you select the directory you need to select files which you want to retrieve from it. Click on *cs272* on the left panel to get a list of files and select file *ShowCurrentTime.java*:


Import

File system

Import resources from the local file system.

From directory: /home/faculty6/hcao/workspace

workspace

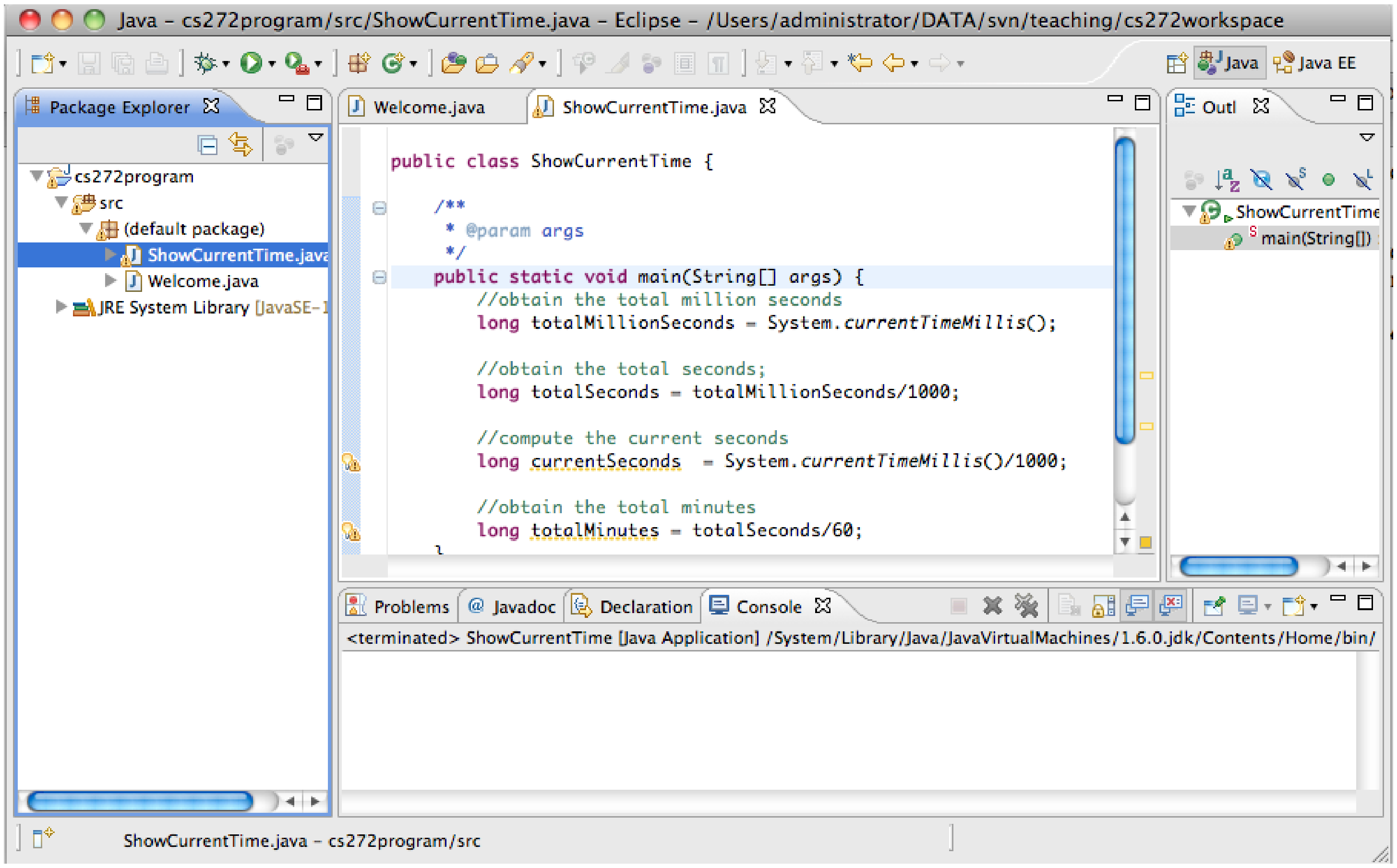
✓ test.java

Into folder: CS272/src/cs272

Options

- ☐ Overwrite existing resources without warning
- ○ Create complete folder structure
- ● Create selected folders only

Make sure that the Into folder field has the name of the directory of the project in it (*showtime*). After you select the file(s) click the *Finish* button. To see the contents of a file you imported double click on your project name in the left panel, then double click the *(default package)* under your project name in the left panel, then double click on the name of the file.


Java - cs272program/src/ShowCurrentTime.java - Eclipse - /Users/administrator/DATA/svn/teaching/cs272workspace

```
public class ShowCurrentTime {

    /**
     * @param args
     */
    public static void main(String[] args) {
        //obtain the total million seconds
        long totalMillionSeconds = System.currentTimeMillis();

        //obtain the total seconds;
        long totalSeconds = totalMillionSeconds/1000;

        //compute the current seconds
        long currentSeconds  = System.currentTimeMillis()/1000;

        //obtain the total minutes
        long totalMinutes = totalSeconds/60;
```

```
<terminated> ShowCurrentTime [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/
```

Step 8: Debugging in Eclipse

The debugger utility is integrated in Eclipse. You can pinpoint bugs in your program with the help of the Eclipse debugger. The Eclipse debugger enables you to set breakpoints and execute programs line by line. As your program executes, you can watch the values stored in variables, observe which methods are being called, and know what events have occurred in the program.

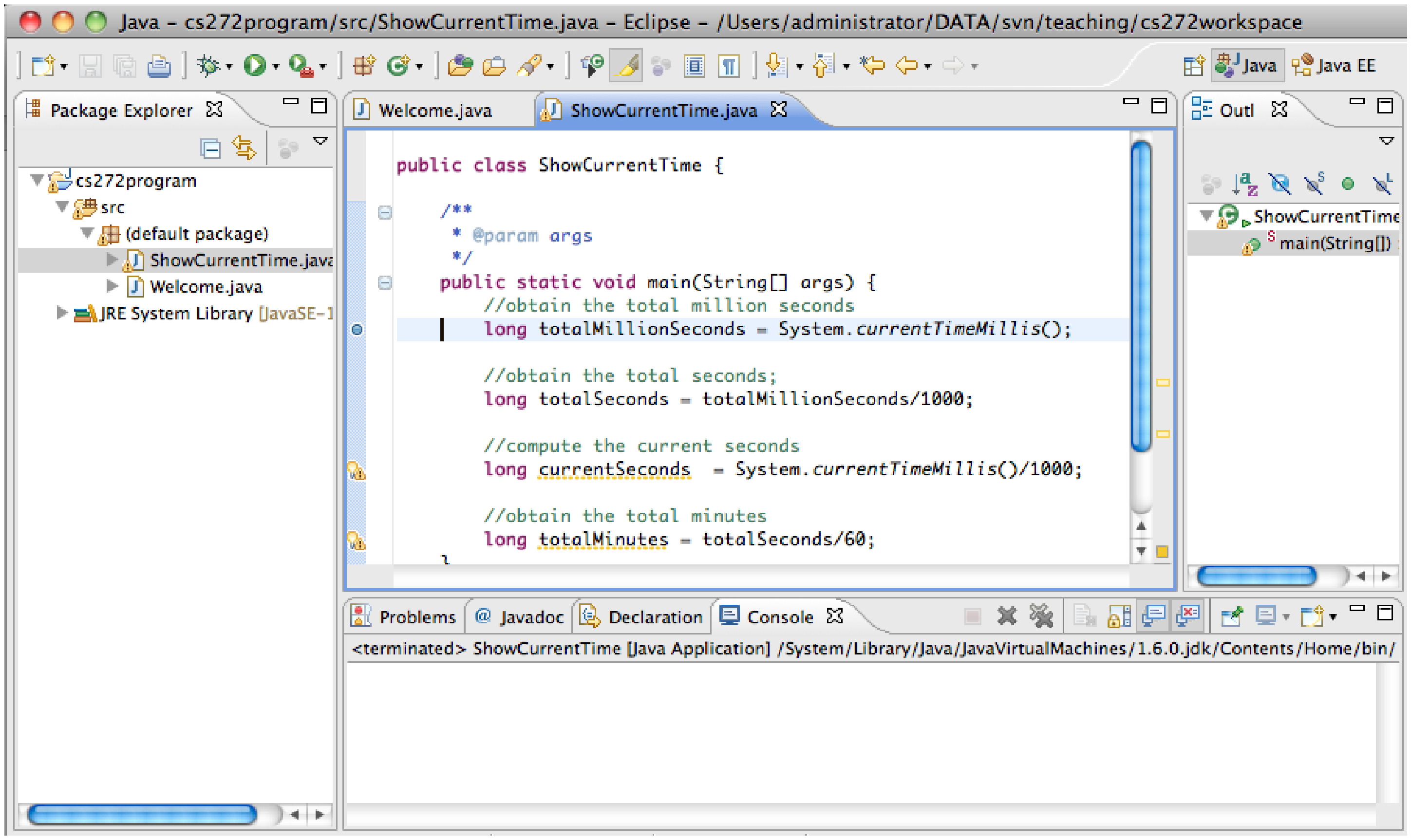
To demonstrate debugging, let us use example that displays the current time. The source code for ShowCurrentTime.java can be obtained from the directory /home/faculty5/ipivkina/cs272/
You may either import the file ShowCurrentTime.java or create a new class named ShowCurrentTime and type in the code yourself.

8.1 Setting breakpoints

You can execute a program line by line to trace it, but this is time-consuming if you are debugging a large program. Often, you know that some parts of the program work fine. It makes no sense to trace these parts when you only need to trace the lines of code that are likely to have bugs. In cases of this kind, you can use breakpoints.

A *breakpoint* is a stop sign placed on a line of source code that tells the debugger to pause when this line is encountered. The debugger executes every line until it encounters a breakpoint, so you can trace the part of the program at the breakpoint. Using the breakpoint, you can quickly move over the sections you know work correctly and concentrate on the sections causing problems.

There are several ways to set a breakpoint on a line. One way is to click on the line on which you want to put a breakpoint and then choose *Run, Toggle Line Breakpoint* from the main menu. You will see the line highlighted, and a breakpoint mark on the left of the line:



```
public class ShowCurrentTime {

    /**
     * @param args
     */
    public static void main(String[] args) {
        //obtain the total million seconds
        long totalMillionSeconds = System.currentTimeMillis();

        //obtain the total seconds;
        long totalSeconds = totalMillionSeconds/1000;

        //compute the current seconds
        long currentSeconds  = System.currentTimeMillis()/1000;

        //obtain the total minutes
        long totalMinutes = totalSeconds/60;
```

```
<terminated> ShowCurrentTime [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/
```

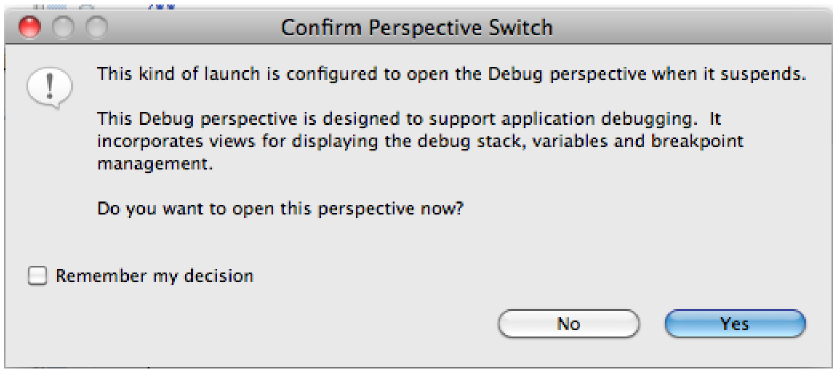
To remove a breakpoint, you may right-click on the breakpoint mark on the left of the line and select *Toggle Breakpoint* from the pop up menu.

As you debug your program, you can set as many breakpoints as you want, and can remove breakpoints at any time during debugging. The project retains the breakpoints you have set when you exit the project. The breakpoints are restored when you reopen it.

8.2 Starting the Debugger

There are several ways to start the debugger. A simple way is shown below:

1. Set a breakpoint at the first statement in the main method in the Source Editor.
2. Click on *ShowCurrentTime.java* . Choose *Run, Debug As, Java Application* from the main menu to start debugging. You will first see the Confirm Perspective Switch dialog, as shown below:



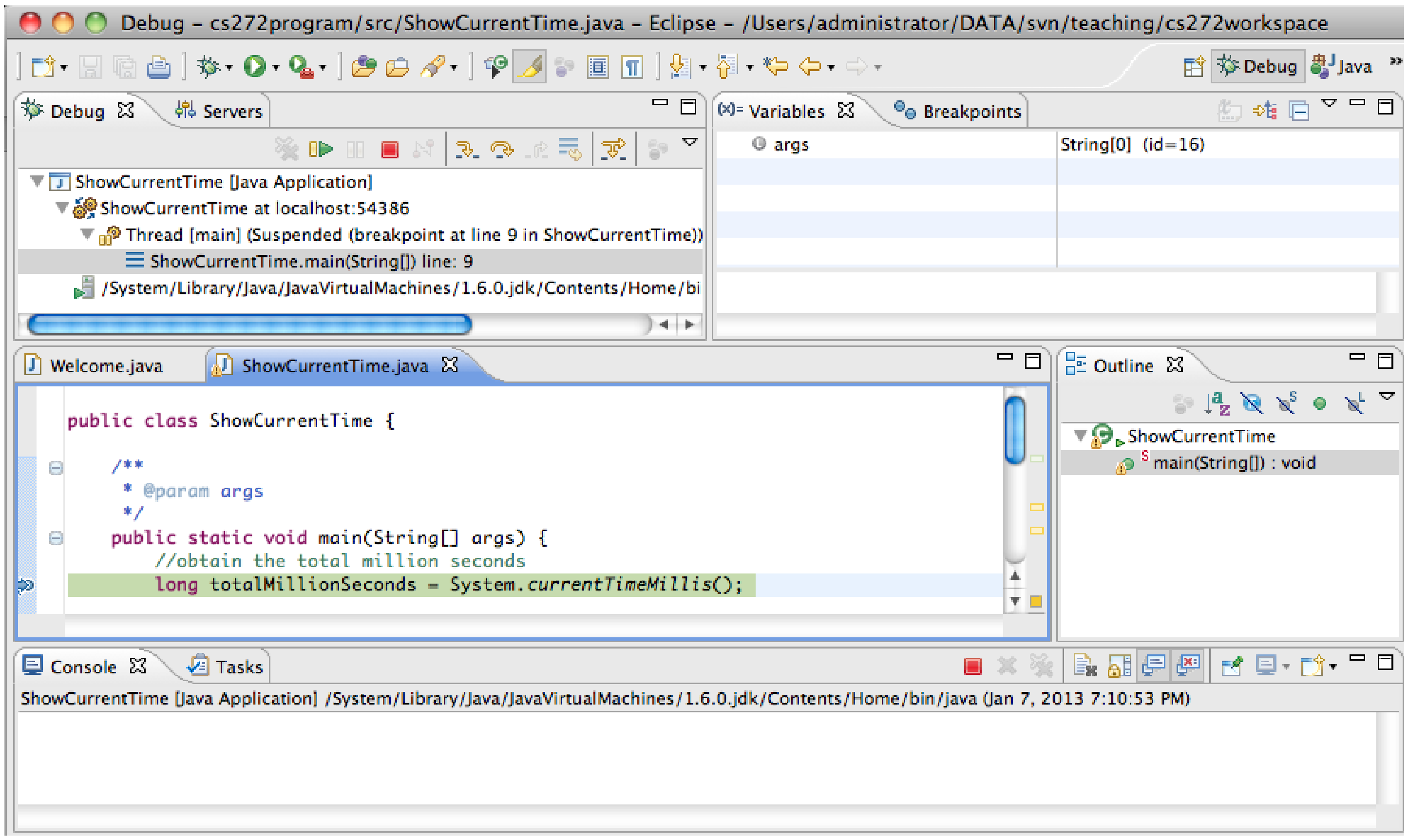
This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

   Click *Yes* to switch to the Debug perspective.

The user interface for Debug perspective is shown below:


Debug – cs272program/src/ShowCurrentTime.java – Eclipse – /Users/administrator/DATA/svn/teaching/cs272workspace
Debug
Java
Debug
Servers
Variables
Breakpoints
Welcome.java
ShowCurrentTime.java
Outline
Console
Tasks

```
ShowCurrentTime [Java Application]
  ShowCurrentTime at localhost:54386
    Thread [main] (Suspended (breakpoint at line 9 in ShowCurrentTime))
      ShowCurrentTime.main(String[]) line: 9
  /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bi
```

args	String[0] (id=16)

```java
public class ShowCurrentTime {

    /**
     * @param args
     */
    public static void main(String[] args) {
        //obtain the total million seconds
        long totalMillionSeconds = System.currentTimeMillis();
```

```
ShowCurrentTime
  main(String[]) : void
```

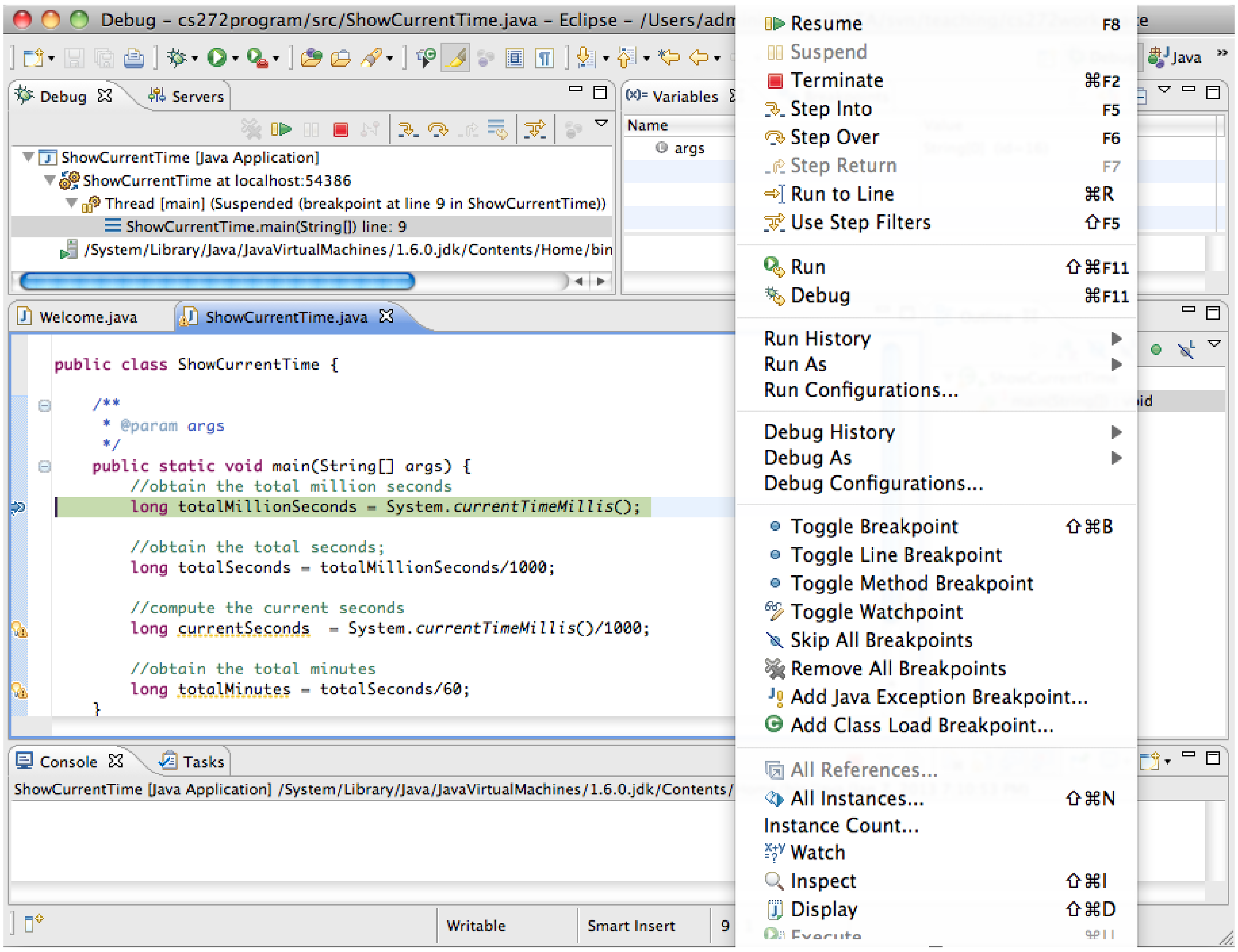
```
ShowCurrentTime [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Jan 7, 2013 7:10:53 PM)
```

8.3 Controlling Program Execution

The program pauses at the first line in the *main* method. This line, called the *current execution point*, is highlighted in green. The current execution point marks the nest line of source code to be executed by the debugger.

When the program pauses at the execution point, you can issue debugging commands to control the execution of the program. You can also inspect or modify the values of variables in the program.

When Eclipse is in the debugging mode, the following toolbar buttons for debugging are displayed in the Debug window, as shown below. The toolbar button commands also appear in the *Run* menu:



```
ShowCurrentTime [Java Application]
  ShowCurrentTime at localhost:54386
    Thread [main] (Suspended (breakpoint at line 9 in ShowCurrentTime))
      ShowCurrentTime.main(String[]) line: 9
  /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin
```

```
public class ShowCurrentTime {

    /**
     * @param args
     */
    public static void main(String[] args) {
        //obtain the total million seconds
        long totalMillionSeconds = System.currentTimeMillis();

        //obtain the total seconds;
        long totalSeconds = totalMillionSeconds/1000;

        //compute the current seconds
        long currentSeconds  = System.currentTimeMillis()/1000;

        //obtain the total minutes
        long totalMinutes = totalSeconds/60;
    }
```

```
ShowCurrentTime [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/
```

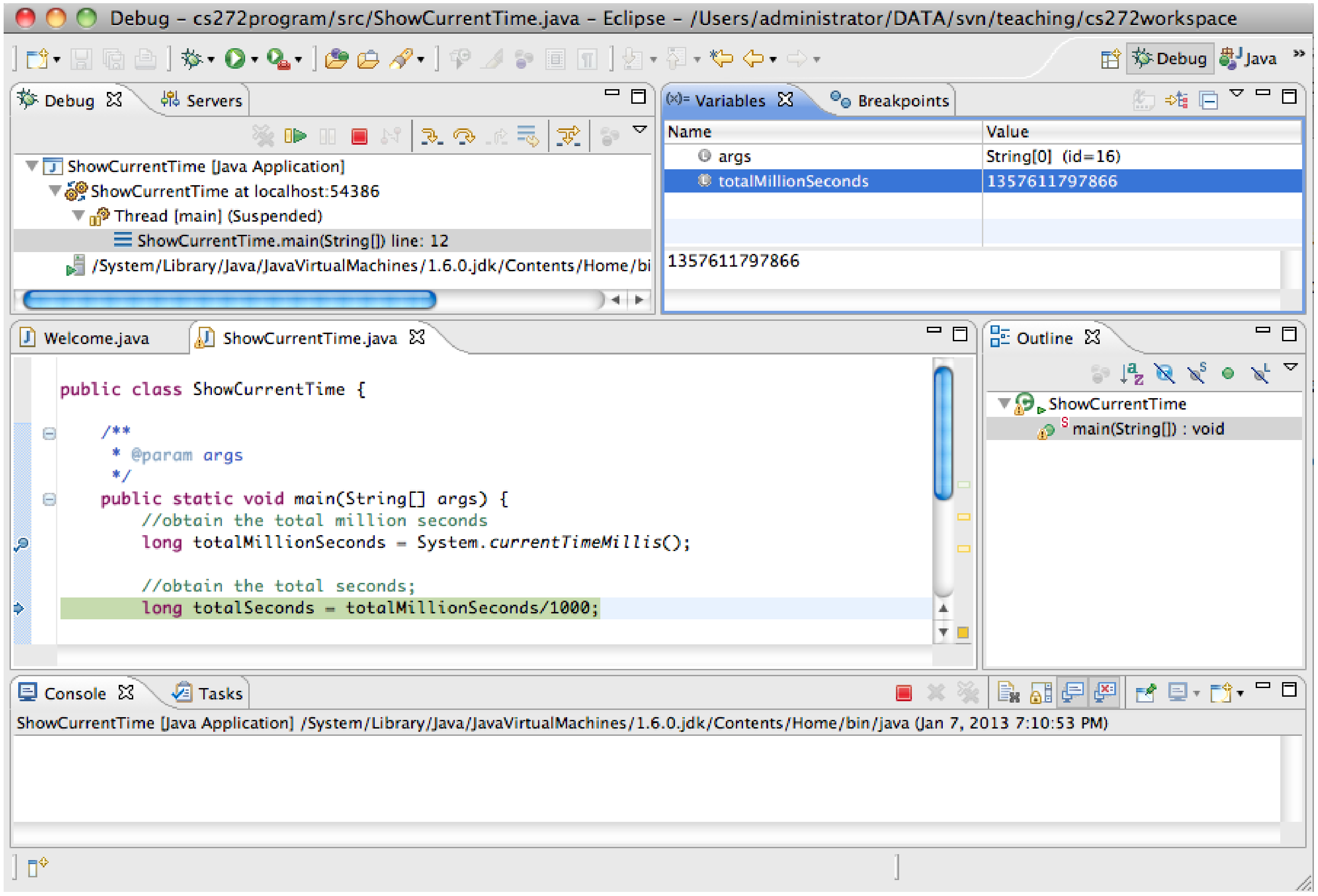
The following are the commands for controlling program execution:

- **Resume** resumes the execution of a paused program.
- **Suspend** temporarily stops execution of a program.
- **Terminate** ends the current debugging session.
- **Step Into** executes a single statement or steps into a method.
- **Step Over** executes a single statement. If the statement contains a call to a method, the entire method is executed without stepping through it.
- **Step Return** executes all the statements in the current method and returns to its caller.
- **Run to Line** runs the program, starting from the current execution point, and pauses and places the execution point on the line of code containing the cursor, or at a breakpoint.

8.4 Examining and Modifying Variables

Among the most powerful features of an integrated debugger is its capability to examine the values of variables, array items, and objects, or the values of the parameters passed in a method call. You can also modify a variable value if you want to try a new value to continue debugging without restarting the program.

To demonstrate it, choose *Run, Step Over* to execute one line in the source code, and you will see the value for *totalMilliseconds* in the Variables pane, as shown below:


Debug - cs272program/src/ShowCurrentTime.java - Eclipse - /Users/administrator/DATA/svn/teaching/cs272workspace

```
ShowCurrentTime [Java Application]
  ShowCurrentTime at localhost:54386
    Thread [main] (Suspended)
      ShowCurrentTime.main(String[]) line: 12
  /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bi
```

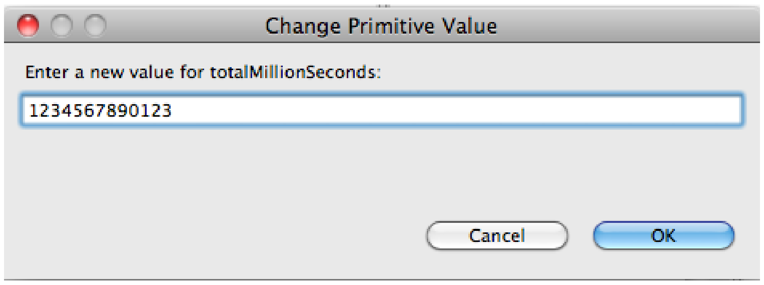
Name	Value
args	String[0] (id=16)
totalMillionSeconds	1357611797866

1357611797866

```java
public class ShowCurrentTime {

    /**
     * @param args
     */
    public static void main(String[] args) {
        //obtain the total million seconds
        long totalMillionSeconds = System.currentTimeMillis();

        //obtain the total seconds;
        long totalSeconds = totalMillionSeconds/1000;
```

```
ShowCurrentTime [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Jan 7, 2013 7:10:53 PM)
```

To change the value in *totalMilliseconds*, right-click on *totalMilliseconds* and select *Change Value* from the pop up menu. The Change Primitive Value dialog box will appear, as shown below. You can now set a new value for *totalMilliseconds*.


Change Primitive Value

Enter a new value for totalMillionSeconds:

1234567890123

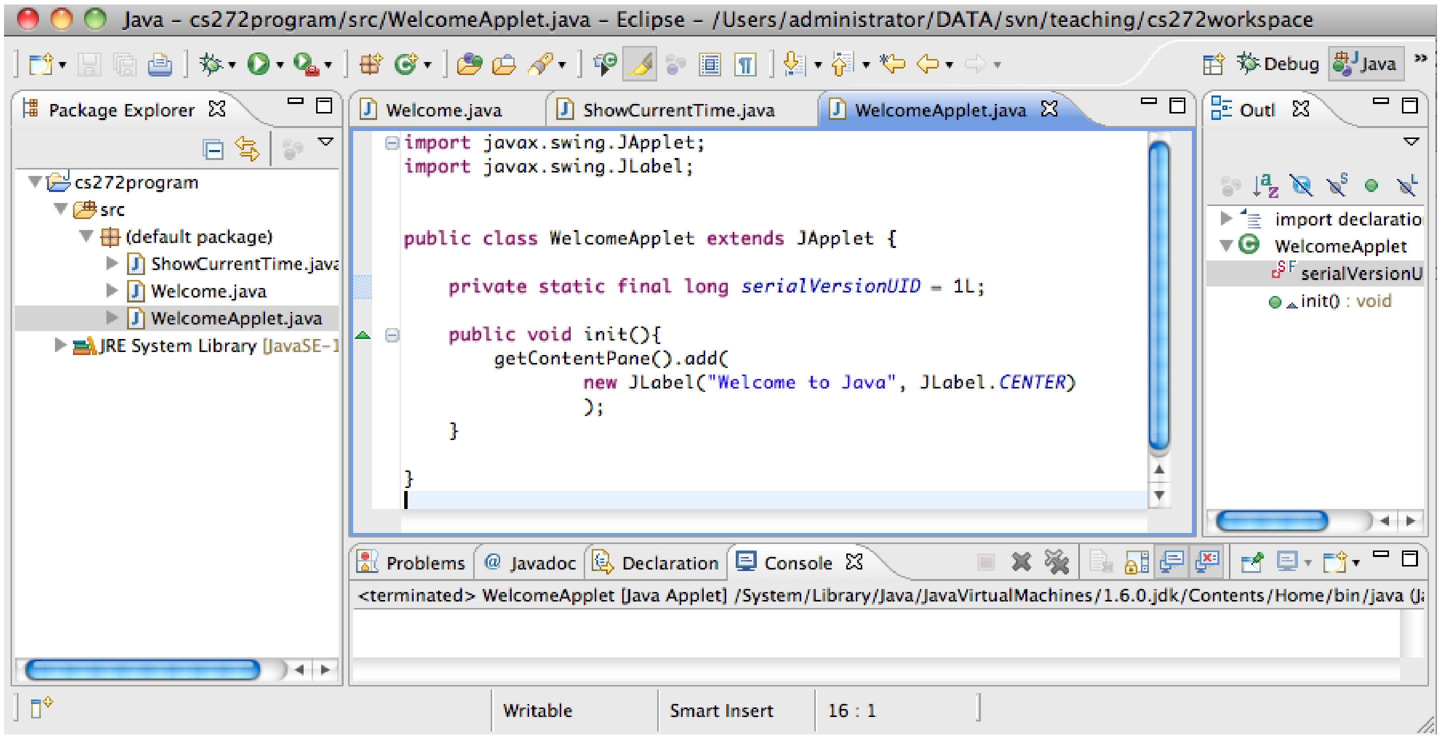
Change the value for *totalMilliseconds* to *1234567890123*. Then, choose *Run, Resume* to resume execution of the program. What time is displayed in the Output window?

The debugger is an indispensable, powerful tool that boosts your productivity. It may take you some time to become familiar with it, but the effort will pay off in the long run.

After finishing debugging, you may switch to the Java perspective by choosing *Window, Open Perspective, Java* from the main menu.

Step 9: Creating and Testing Java Applets

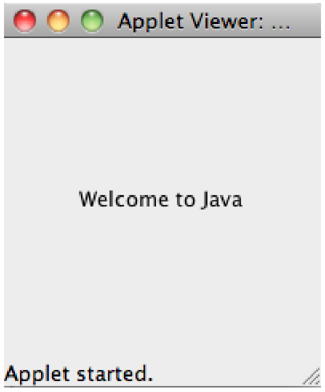
You can create a Java applet in the same way you create a Java application. For example, you can create the WelcomeApplet class, as shown below:


Java - cs272program/src/WelcomeApplet.java - Eclipse - /Users/administrator/DATA/svn/teaching/cs272workspace

```
import javax.swing.JApplet;
import javax.swing.JLabel;


public class WelcomeApplet extends JApplet {

    private static final long serialVersionUID = 1L;

    public void init(){
        getContentPane().add(
                new JLabel("Welcome to Java", JLabel.CENTER)
                );
    }

}
```

```
<terminated> WelcomeApplet [Java Applet] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (J
```

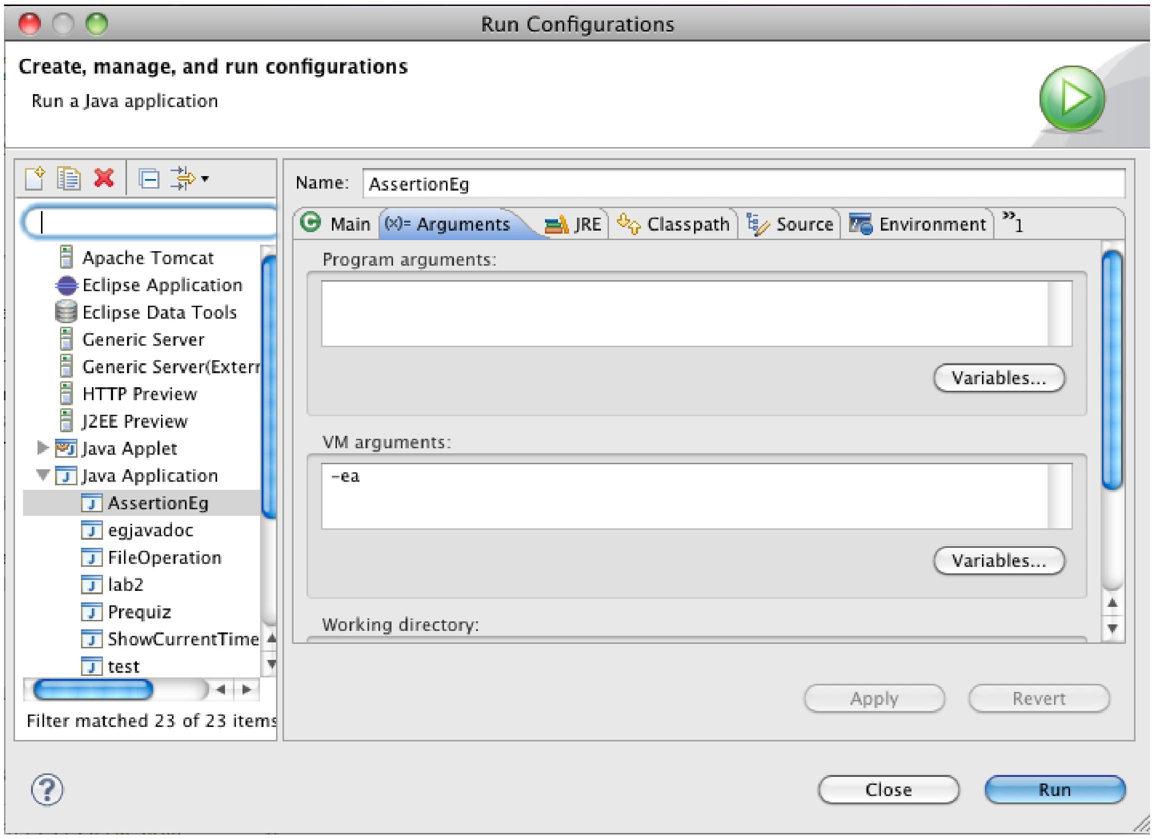
To run an applet, choose *Run, Run As, Java Applet* from the main menu. Eclipse automatically creates an HTML file to contain the applet and invokes the appletviewer utility to run the applet, as shown below.



Many times we need to turn on assertions during debugging a program. E.g.,

```
public static int maxOf2(int a, int b) throws AssertionError
{
        int answer = 1;
        if (a>b) answer = a;
        else answer =b;

        assert((answer==a)||(answer==b)): "maxOf2 answer is not equal to one of the arguments";

        return answer;
}
```

To turn on the assertions: right click the class that implements this method, Then, click "Run as/Run Configurations". You will see the following window Click the "Arguments" tab and add "-ea" to the VM arguments box.



Name: AssertionEg

Program arguments:

VM arguments:

```
-ea
```