

Introduction to Eclipse



Overview

- Eclipse Background
- Obtaining and Installing Eclipse
- Creating a Workspaces / Projects
- Creating Classes
- Compiling and Running Code
- Debugging Code
- Sampling of Features
- Summary

What is Eclipse?

- Eclipse started as a proprietary IBM product (IBM Visual age for Smalltalk/Java)
 - Embracing the open source model IBM opened the product up
- Open Source
 - It is a general purpose open platform that facilitates and encourages the development of third party plug-ins
- Best known as an Integrated Development Environment (IDE)
 - Provides tools for coding, building, running and debugging applications
- Originally designed for Java, now supports many other languages
 - Good support for C, C++
 - Python, PHP, Ruby, etc...

Prerequisites for Running Eclipse

- Eclipse is written in Java and will thus need an installed JRE or JDK in which to execute
 - JDK recommended

Eclipse on GL

- This years coordinated release (known as Ganymede) of the Eclipse IDE for Java Developers has been installed on GL
 - From any of the Linux machines in the labs simply run the command **eclipse**

Obtaining Eclipse

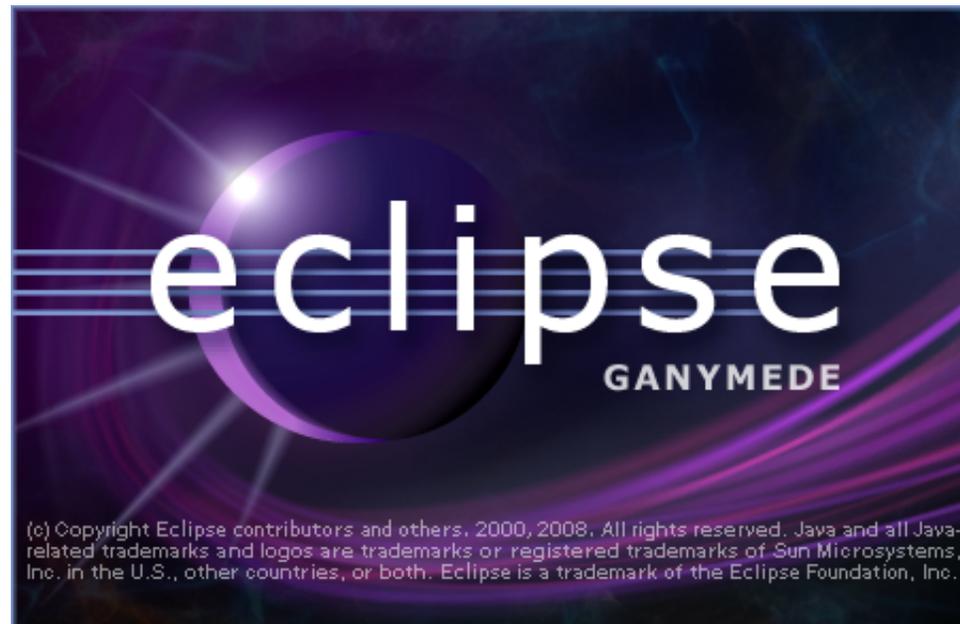
- Eclipse can be downloaded from...
 - <http://www.eclipse.org/downloads/packages/>
 - Be sure to grab “Eclipse IDE for Java Developers”
- Eclipse comes bundled as a zip file (Windows) or a tarball (all other operating systems)
 - Some versions of Linux (i.e. Fedora, Ubuntu) offer Eclipse in their respective repositories and can be downloaded using the appropriate tool (i.e. yum, apt-get)

Installing Eclipse

- Simply unwrap the zip file to some directory where you want to store the executables
- On windows
 - I typically unwrap the zip file to C:\eclipse\
 - C:\eclipse\eclipse.exe
 - I then typically create a shortcut on my desktop to the eclipse executable
- Under Linux
 - I typically unwrap to /opt/eclipse/

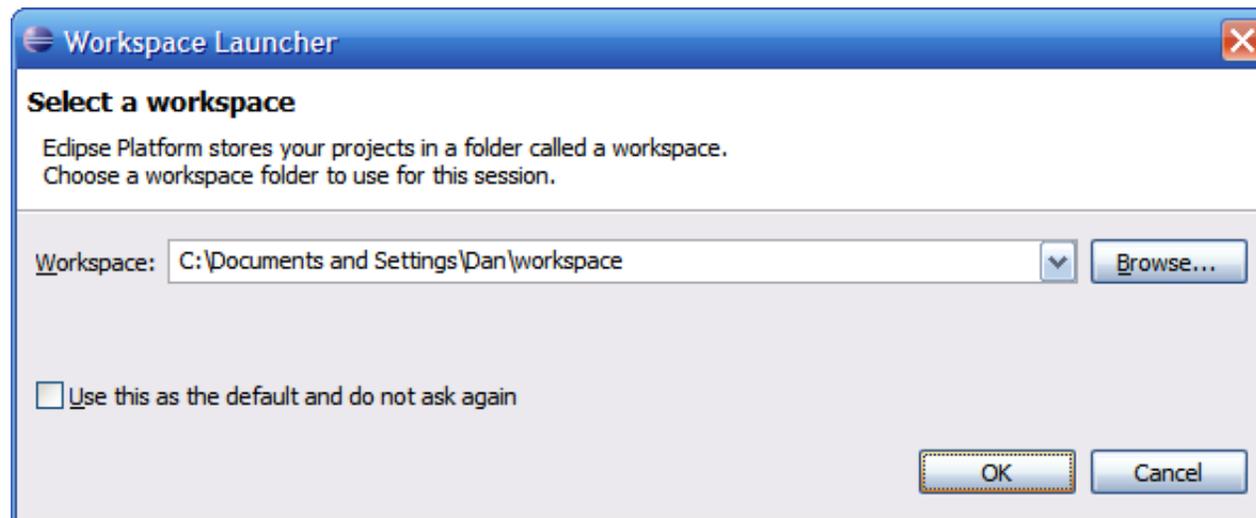
Launching Eclipse

- Once you have the environment setup, go ahead and launch eclipse
- You should see the following splash screen...



Selecting a Workspace

- In Eclipse, all of your code will live under a workspace
- A workspace is nothing more than a location where we will store our source code and where Eclipse will write out our preferences
- Eclipse allows you to have multiple workspaces – each tailored in its own way
- Choose a location where you want to store your files, then click OK

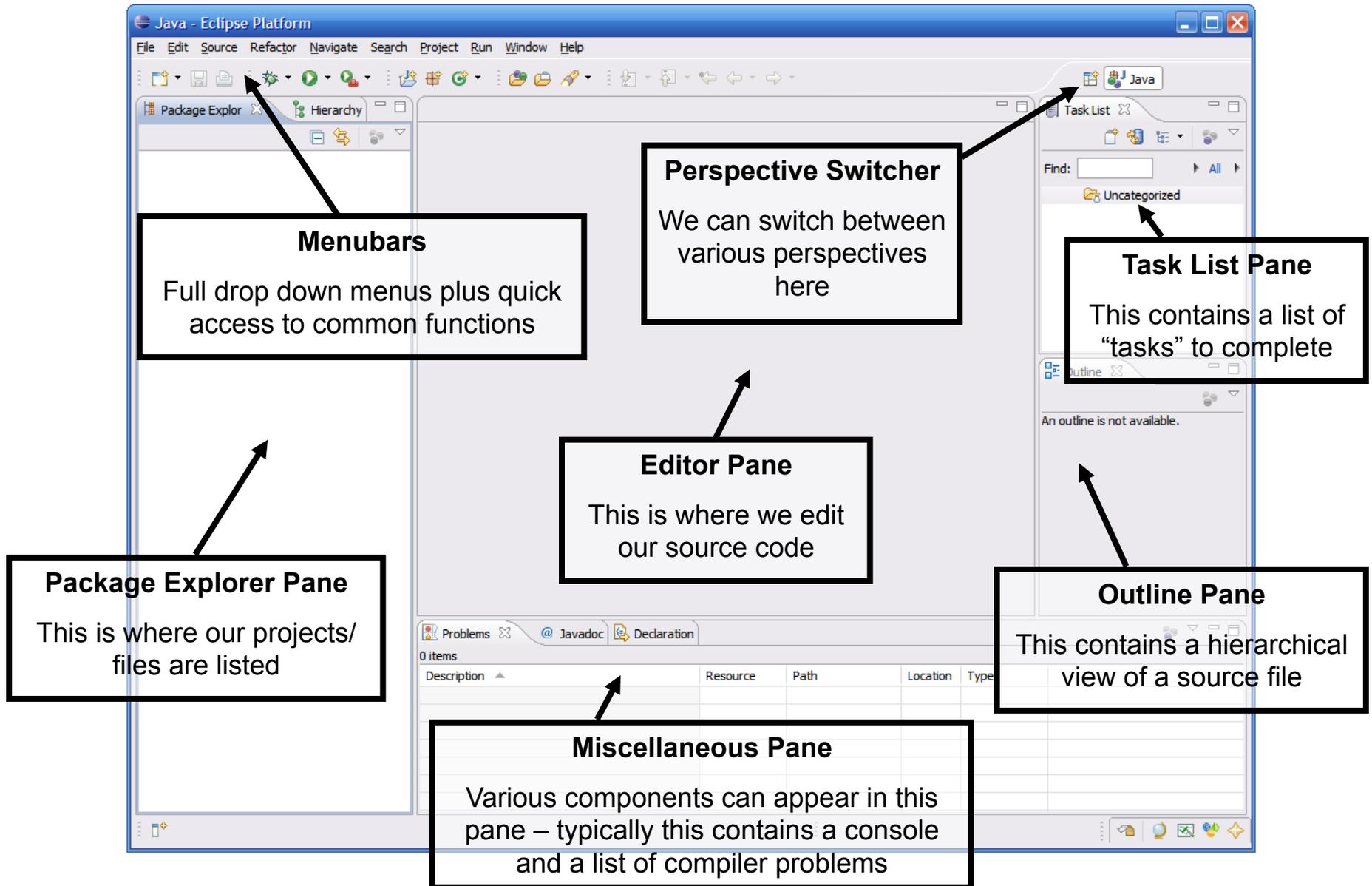


Welcome to Eclipse

- The first time you launch Eclipse, you will be presented with a welcome screen
- From here you can access an overview to the platform, tutorials, sample code, etc...
- Click on the arrow on the right to get to the actual IDE

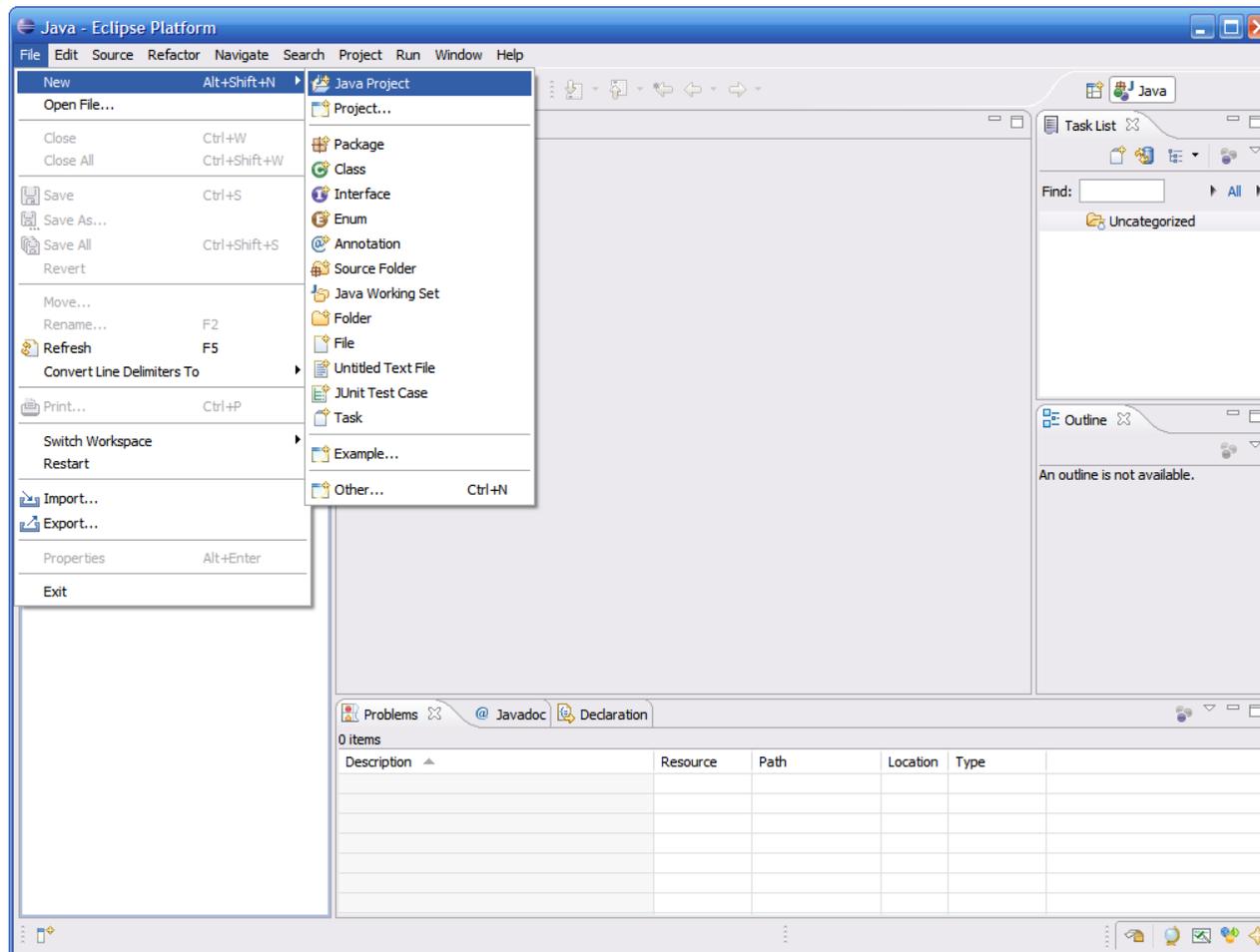


Eclipse IDE Components



Creating a New Project

- All code in Eclipse needs to live under a project
- To create a project: File → New → Java Project



Creating a New Project (continued)

New Java Project

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

Project name:

Contents

Create new project in workspace

Create project from existing source

Directory:

JRE

Use default JRE (Currently 'jre1.6.0_05') [Configure JREs...](#)

Use a project specific JRE:

Use an execution environment JRE:

Project layout

Use project folder as root for sources and class files

Create separate folders for sources and class files [Configure default...](#)

Working sets

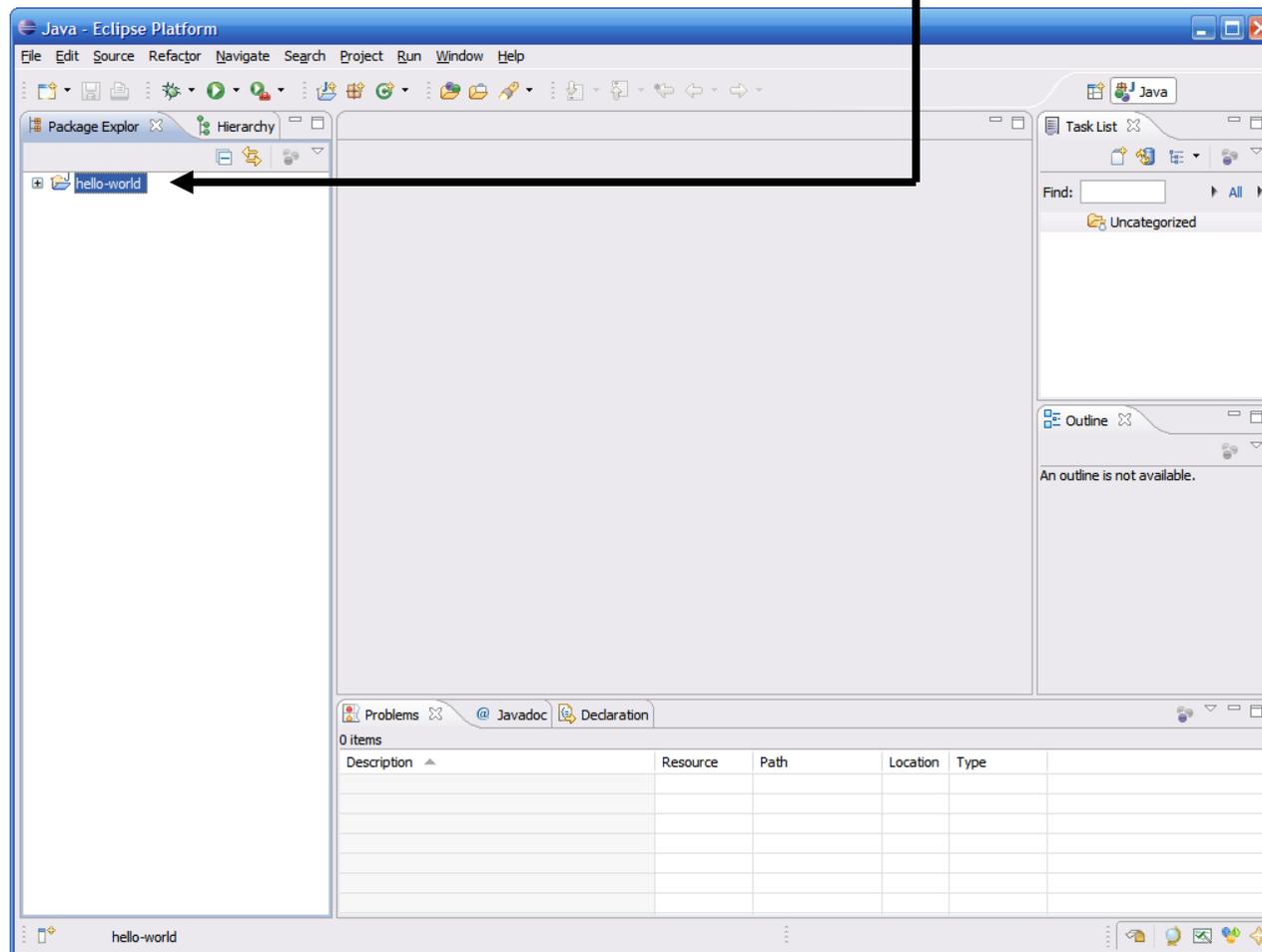
Add project to working sets

Working sets:

- Enter a name for the project, then click Finish

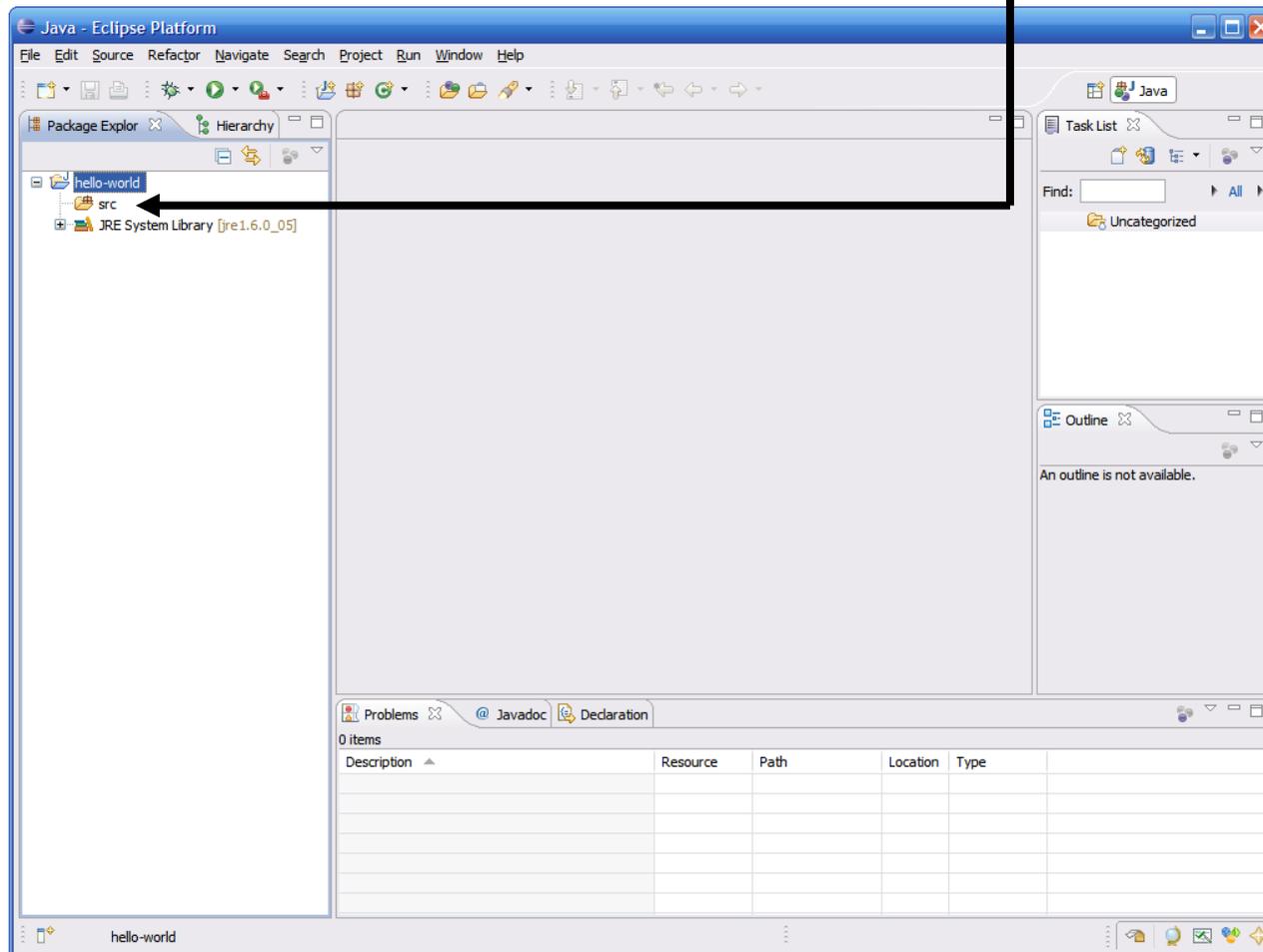
Creating a New Project (continued)

- The newly created project should then appear under the Package Explorer



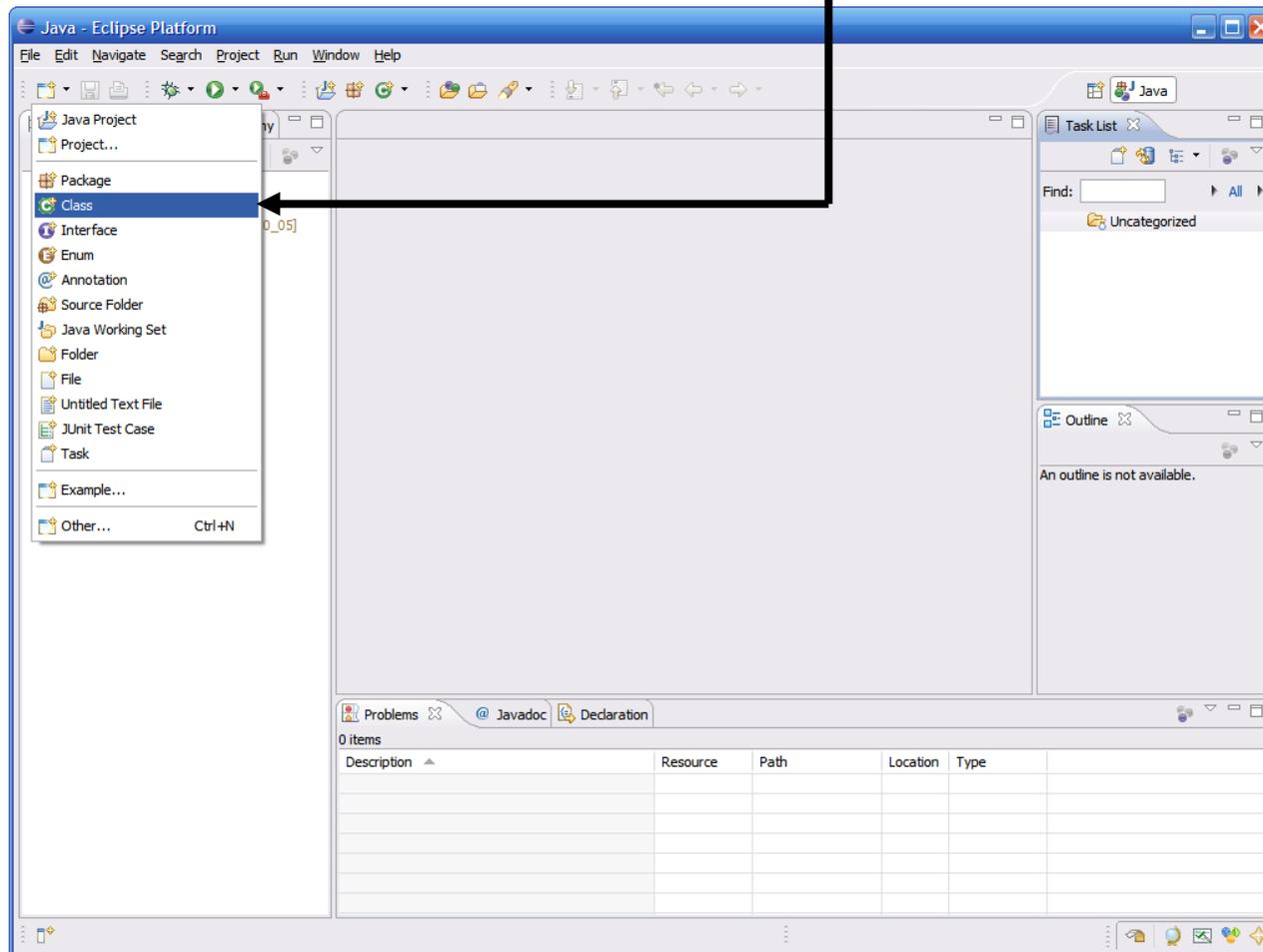
The src folder

- Eclipse automatically creates a folder to store your source code in called src



Creating a Class

- To create a class, simply click on the New button, then select Class



Creating a Class (continued)

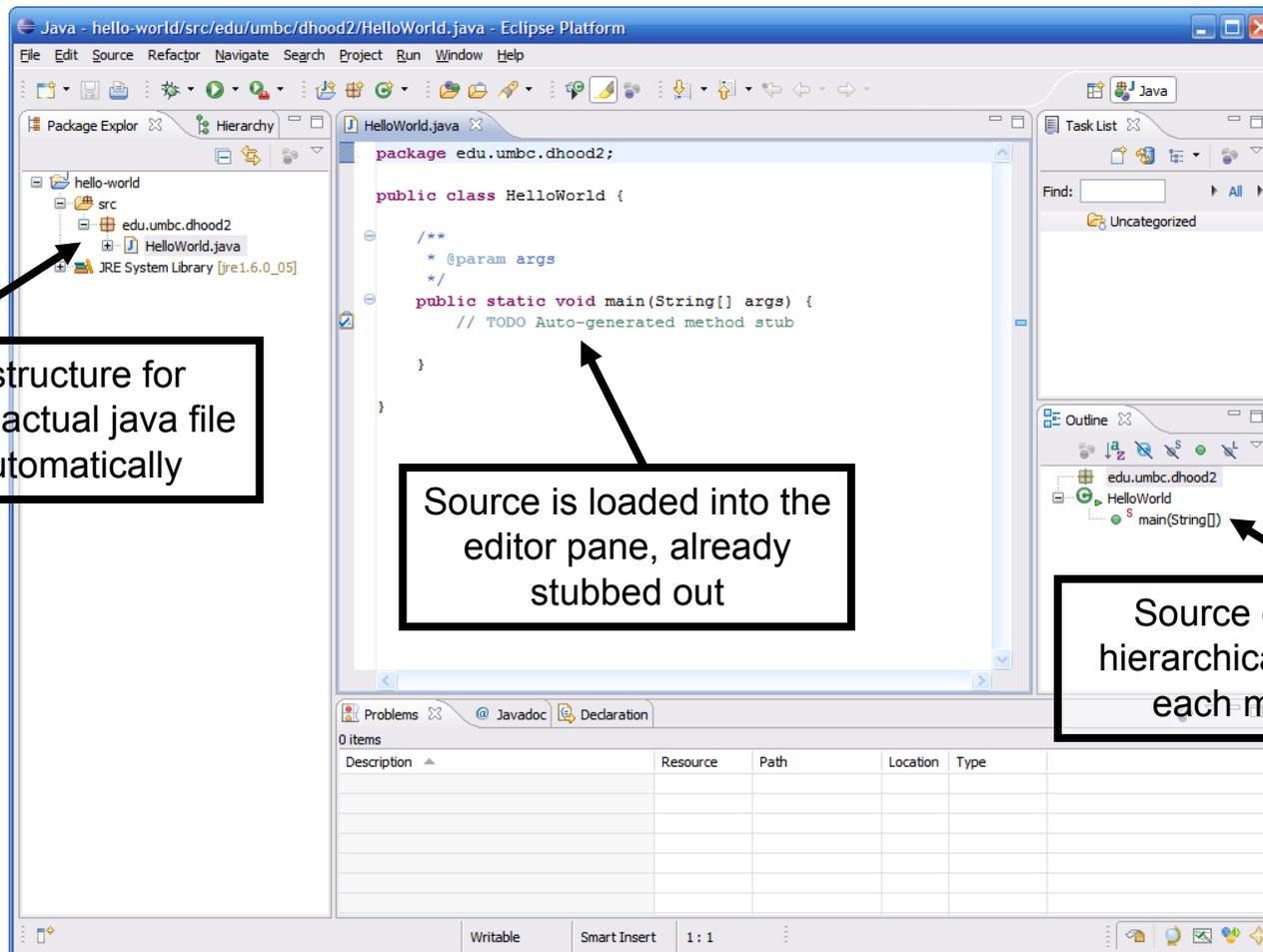
The screenshot shows the 'New Java Class' dialog box with the following configuration:

- Source folder: hello-world/src
- Package: edu.umbc.dhood2
- Endosing type: (empty)
- Name: HelloWorld
- Modifiers: public, default, private, protected, abstract, final, static
- Superclass: java.lang.Object
- Interfaces: (empty)
- 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

- This brings up the new class wizard
- From here you can specify the following...
 - Package
 - Class name
 - Superclass
 - Whether or not to include a main
 - Etc...
- Fill in necessary information then click Finish to continue

The Created Class

- As you can see a number of things have now happened...



Directory structure for package and actual java file created automatically

Source is loaded into the editor pane, already stubbed out

Source displayed in a hierarchical fashion listing each method name

Compiling Source Code

- One huge feature of Eclipse is that it automatically compiles your code in the background
 - You no longer need to go to the command prompt and compile code directly
- This means that errors can be corrected when made
 - We all know that iterative development is the best approach to developing code, but going to shell to do a compile can interrupt the normal course of development
 - This prevents going to compile and being surprised with 100+ errors

Example Compilation Error

- This code contains a typo in the println statement...

Packages/Classes with errors are marked with a red X

Often Eclipse may have suggestions on how to fix the problem – if so, a small light bulb will be displayed next to the line of offending code

Error underlined with red squiggly line (just like spelling errors in many word processors)

Position in file is marked with a red line – 1 click allows you to jump to line with error

Methods with errors are marked with a red X

The Problems tab will contain a tabular representation of all errors across all files of all open projects

```
package edu.umbc.dhood2;

public class HelloWorld {

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

        System.out.printn("Hello World!");

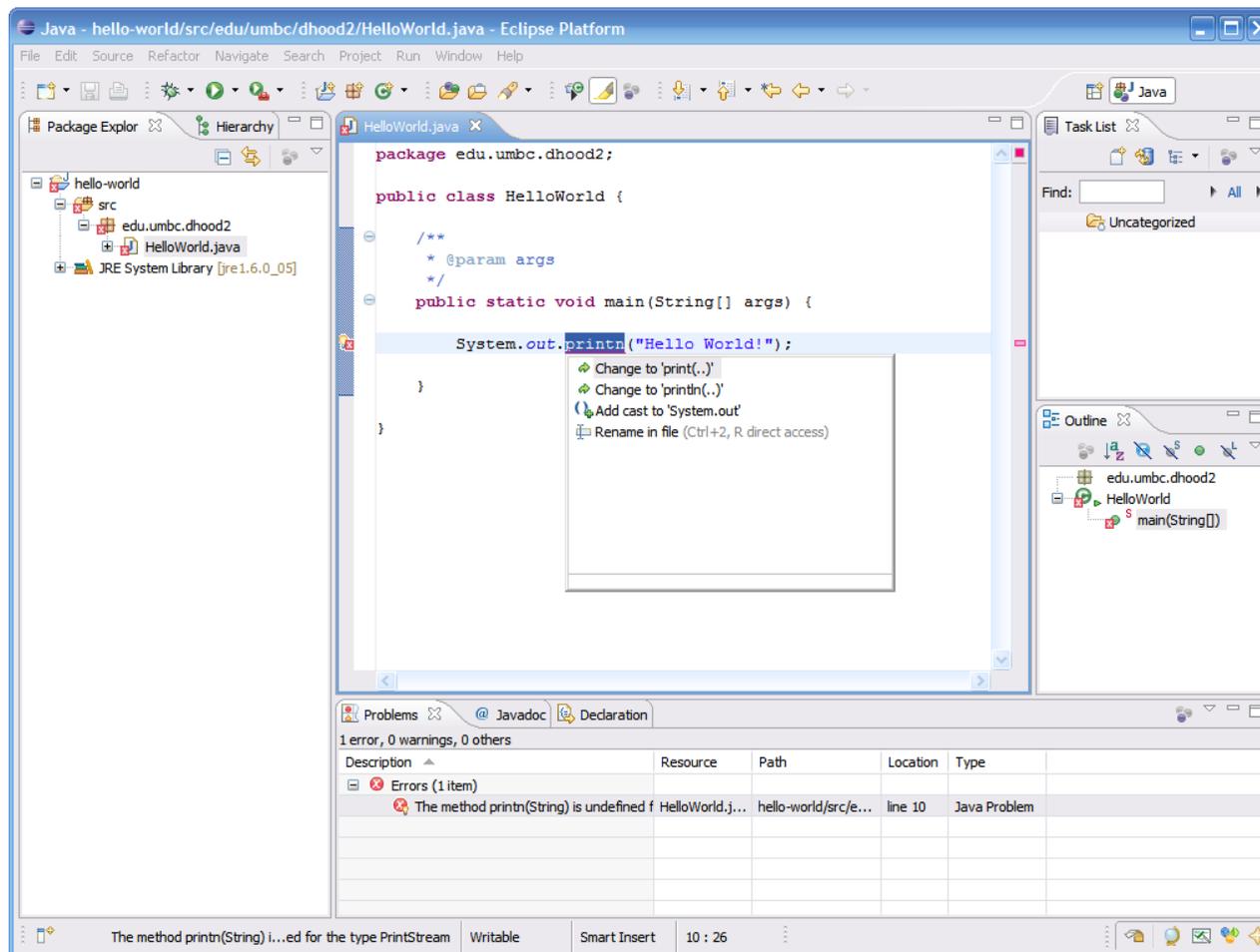
    }

}
```

Description	Resource	Path	Location	Type
Errors (1 item)				
The method println(String) is undefined for HelloWorld.j...	hello-world/src/e...	hello-world/src/e...	line 10	Java Problem

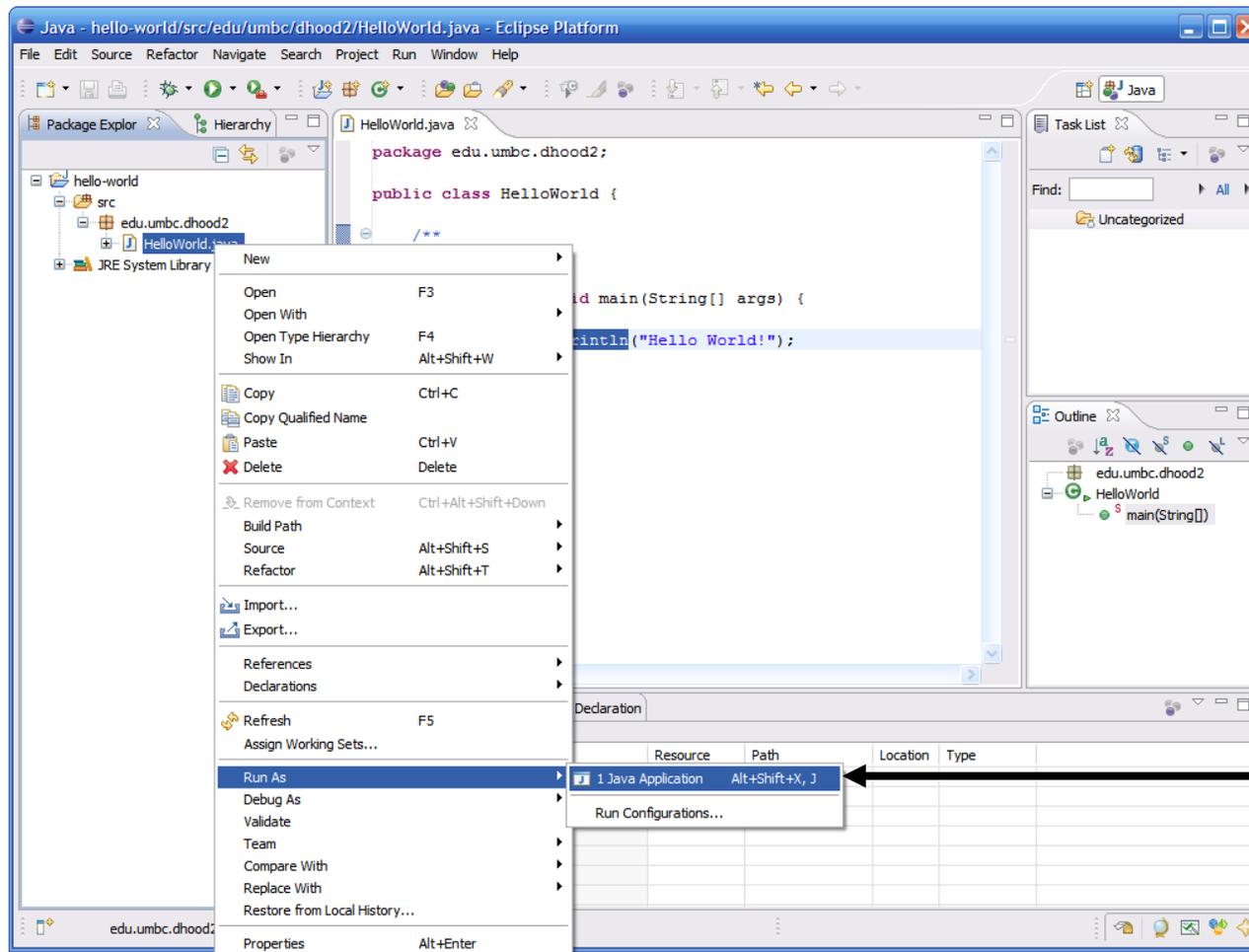
Example Compilation Error (continued)

- When clicking on the light bulb, Eclipse suggests changing println to either print or println



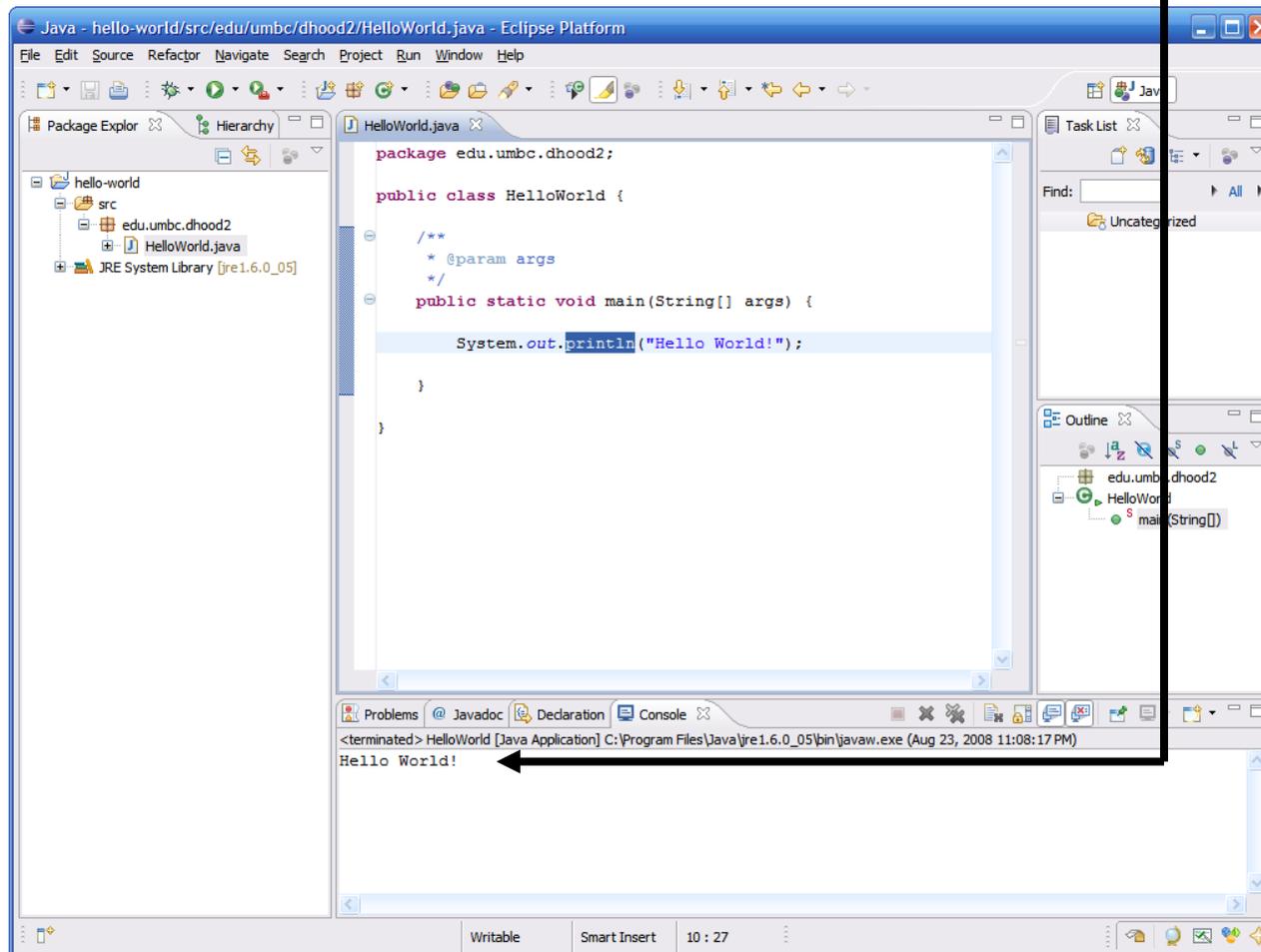
Running Code

- An easy way to run code is to right click on the class and select Run As → Java Application



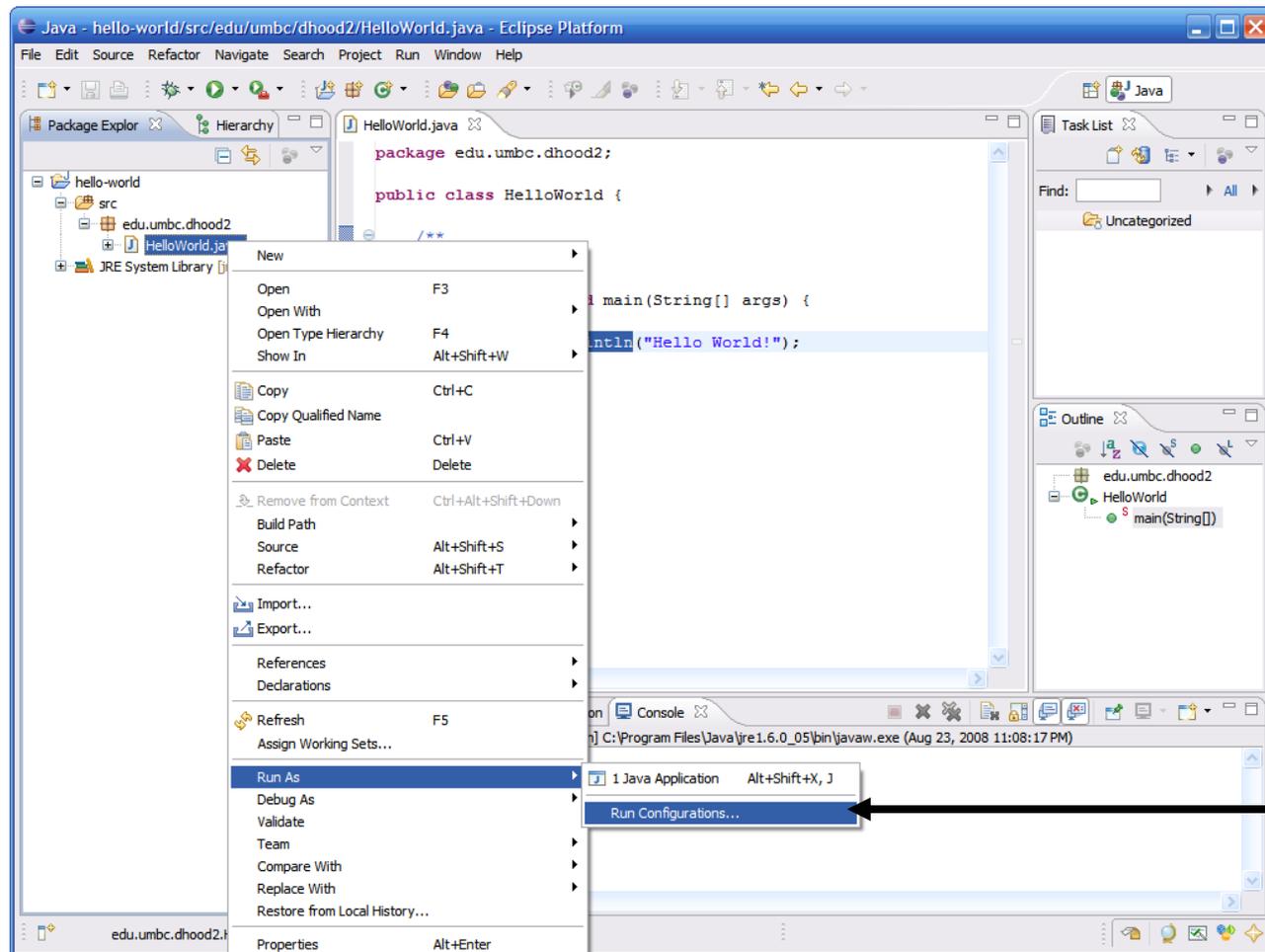
Running Code (continued)

- The output of running the code can be seen in the Console tab in the bottom pane

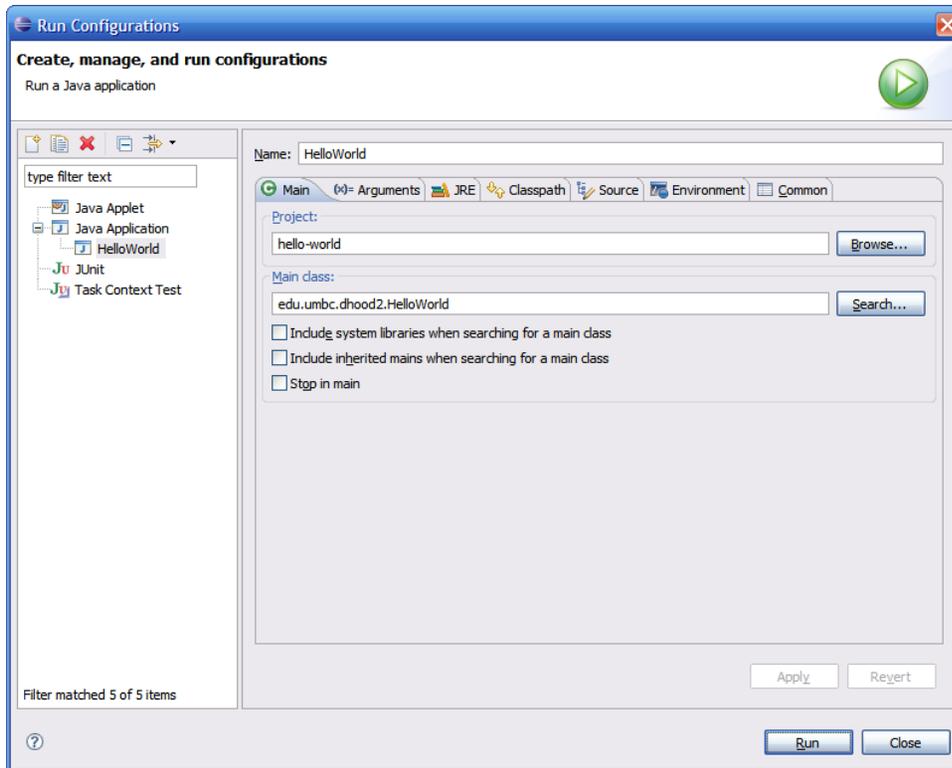


Run Configuration

- Advanced options for executing a program can be found by right clicking the class then clicking Run As → Run...



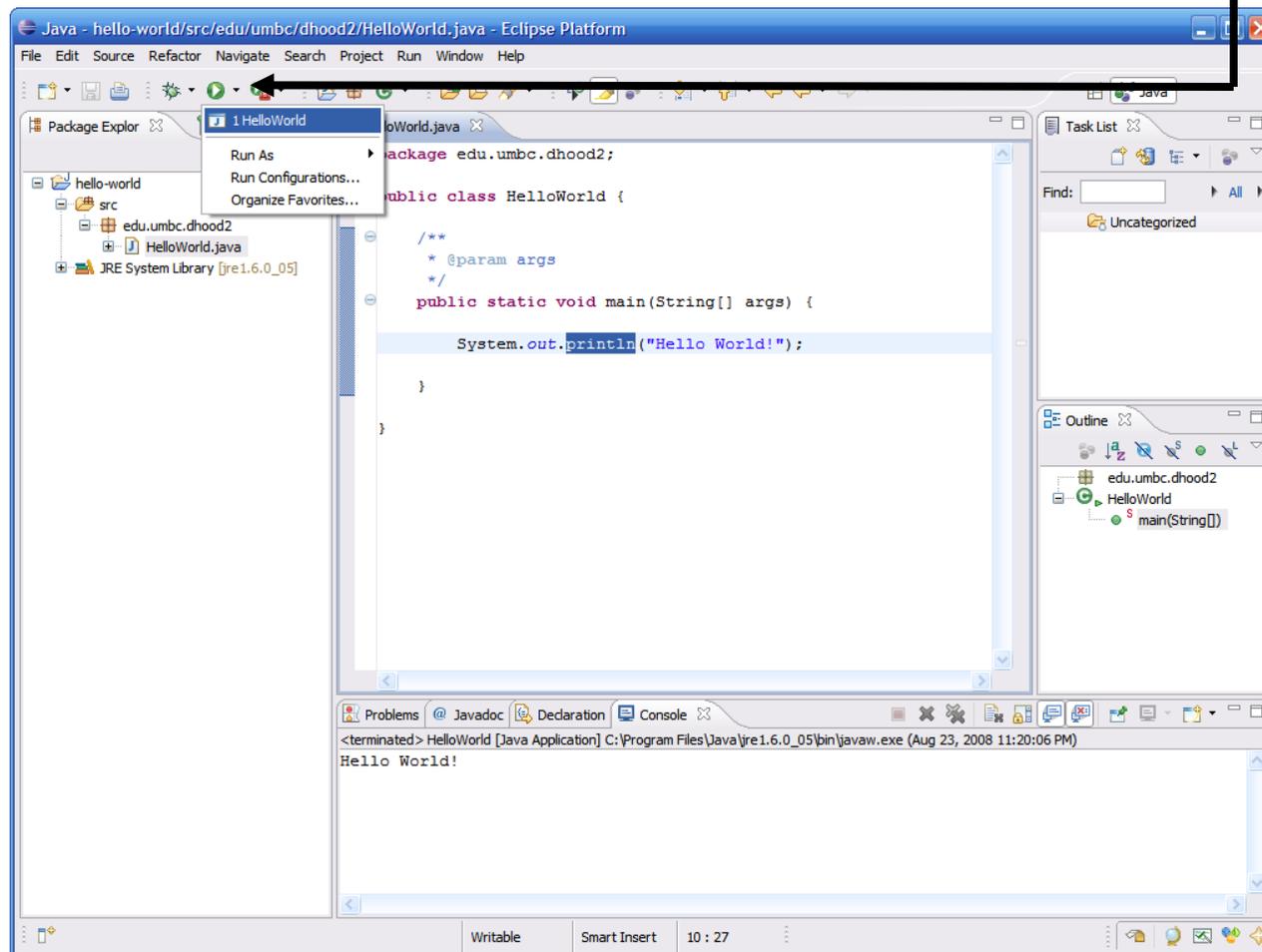
Run Configuration (continued)



- Here you can change/add any of the following:
 - JVM arguments
 - Command line arguments
 - Classpath settings
 - Environment variables
 - Which JVM to use

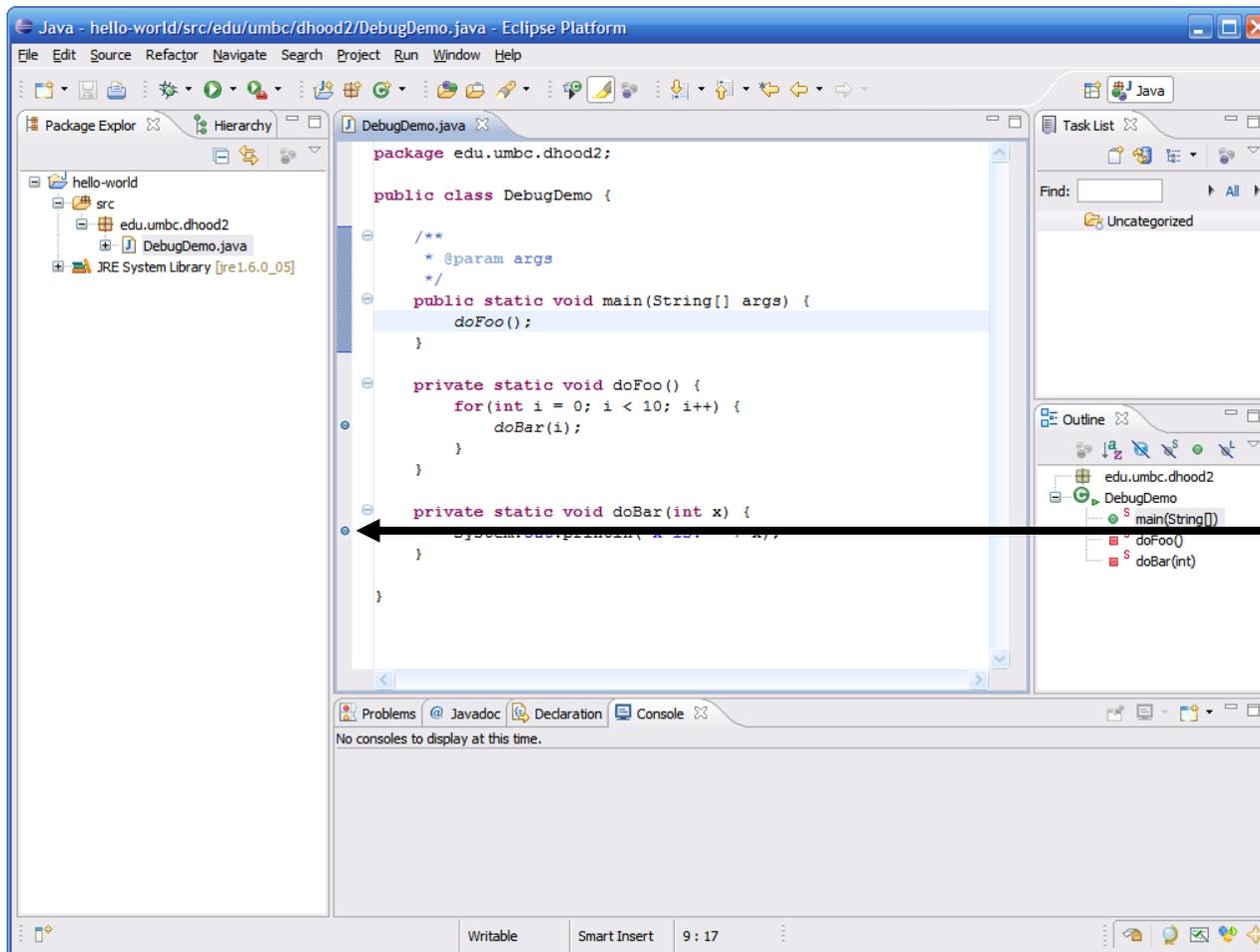
Re-Running Code

- After you run the code a first time, you can re-run it just by selecting it from the run drop down menu



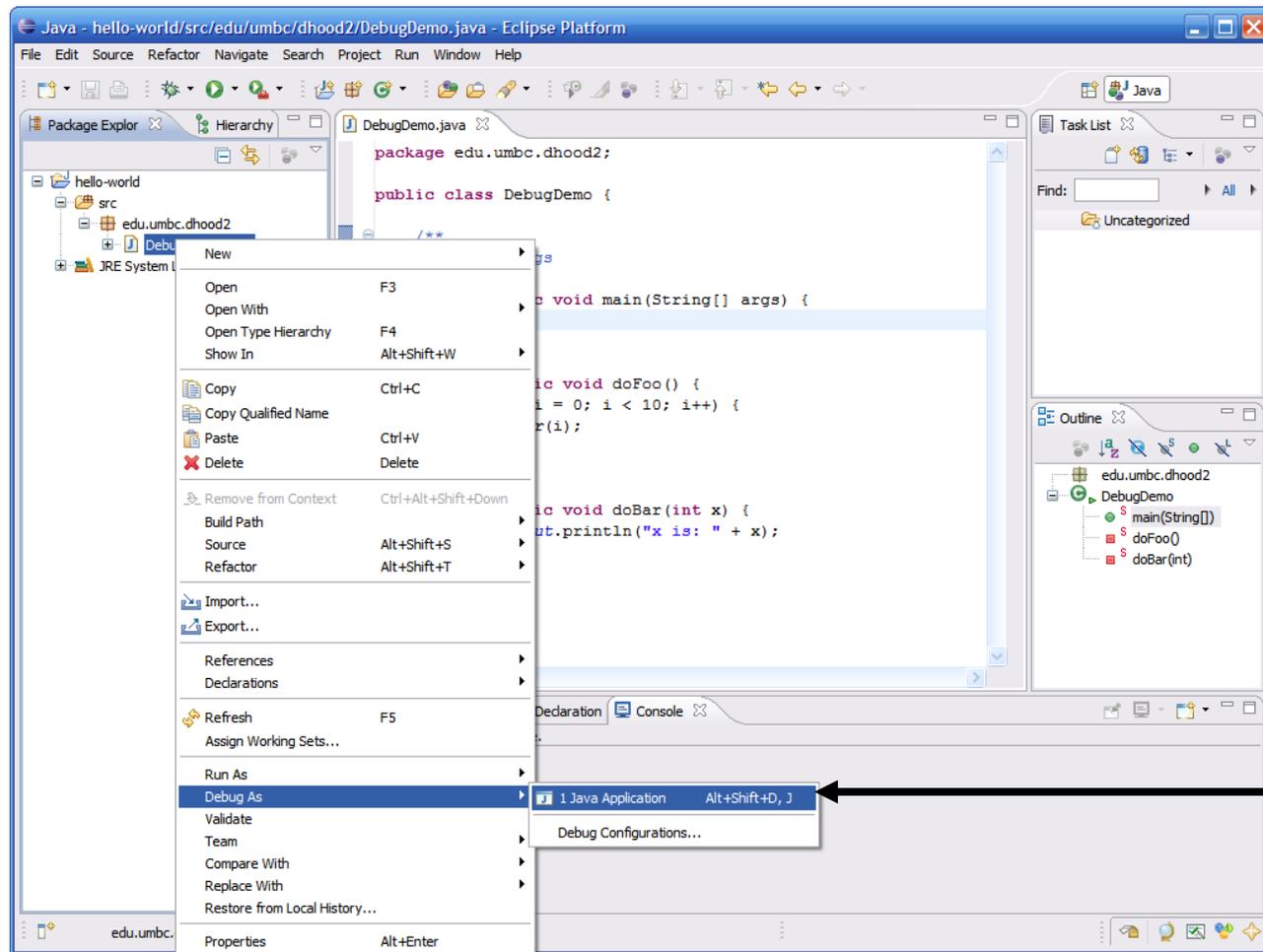
Debugging Code

- Eclipse comes with a pretty good built-in debugger
- You can set break points in your code by double clicking in the left hand margin – break points are represented by these blue bubbles



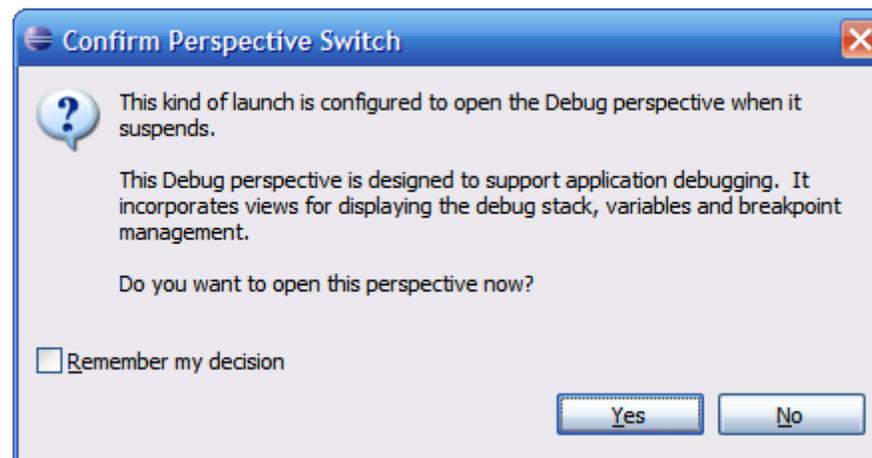
Debugging Code (continued)

- An easy way to enter debug mode is to right click on the class and select Debug As → Java Application



Debugging Code (Continued)

- The first time you try to debug code you will be presented with the following dialog



- Eclipse is asking if you want to switch to a perspective that is more suited for debugging, click Yes
- Eclipse has many perspectives based on what you are doing (by default we get the Java perspective)

Debug Perspective

Note new Debug perspective – click Java to return to normal

These buttons allow you to step through the code

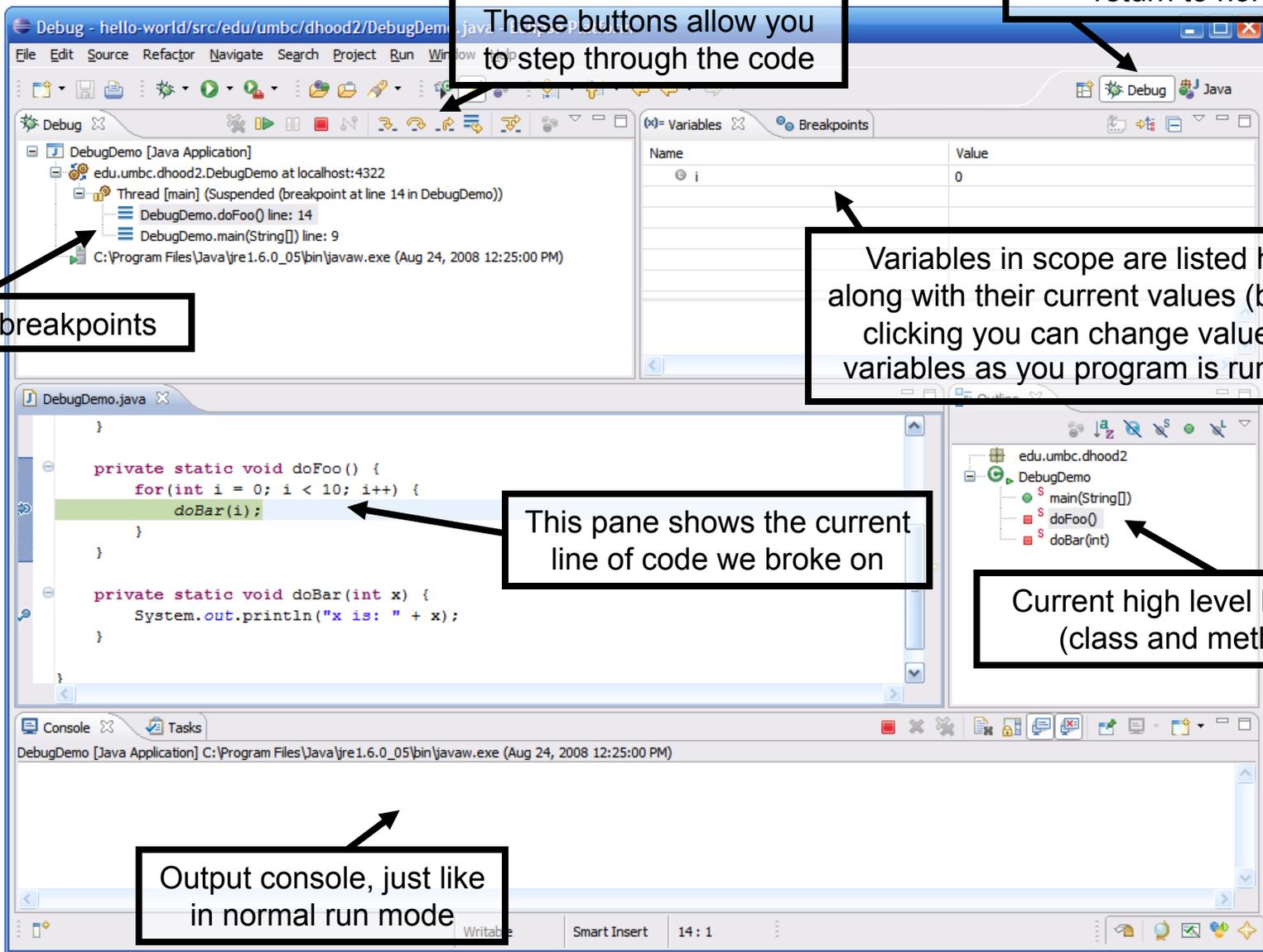
List of breakpoints

Variables in scope are listed here along with their current values (by right clicking you can change values of variables as you program is running)

This pane shows the current line of code we broke on

Current high level location (class and method)

Output console, just like in normal run mode

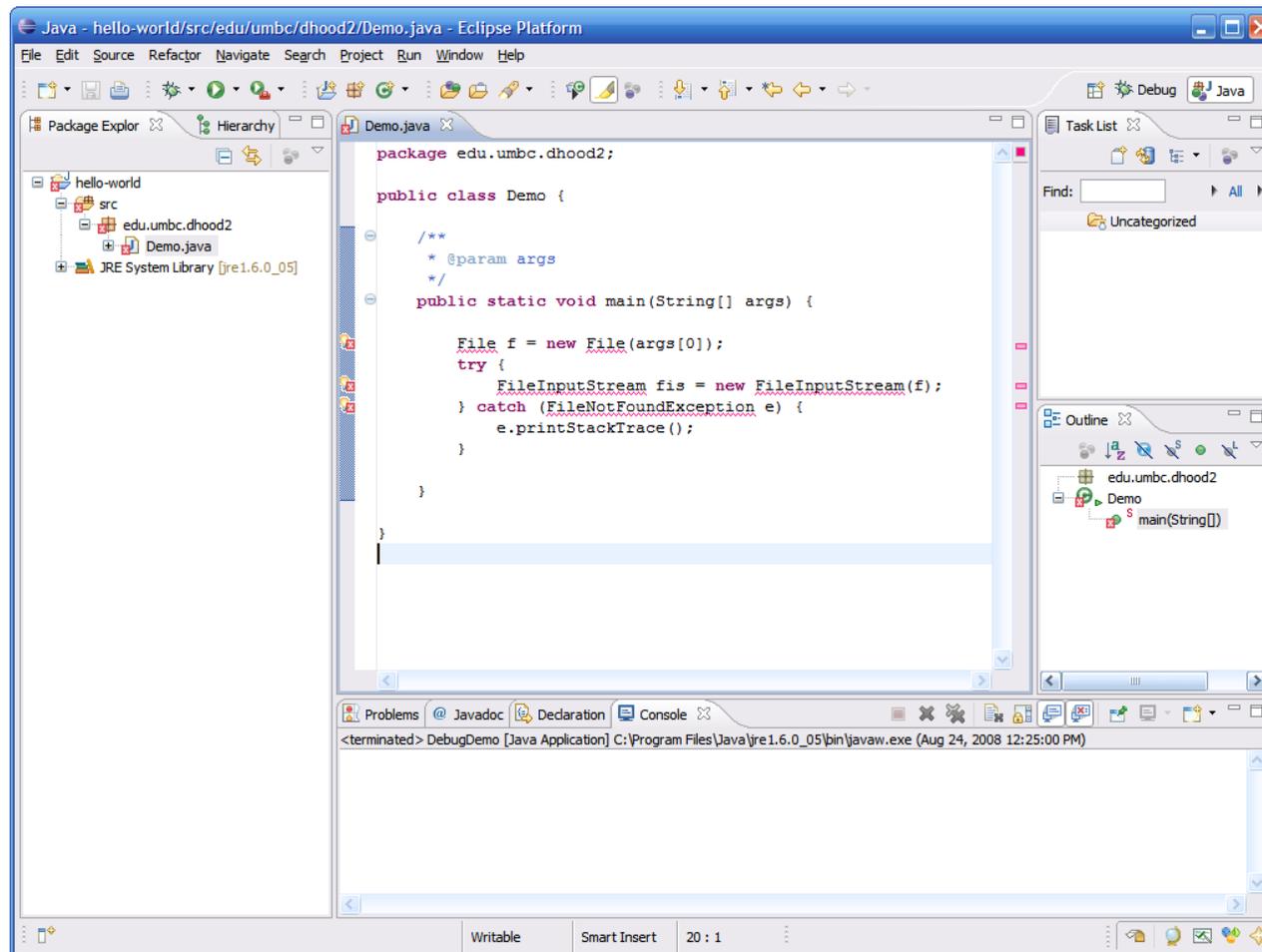


Sampling of Some Other Features

- Import organization
- Context assist
- Javadoc assist
- Getter/Setter generation
- Add unimplemented methods
- Exception handling
- Reminders
- Local history

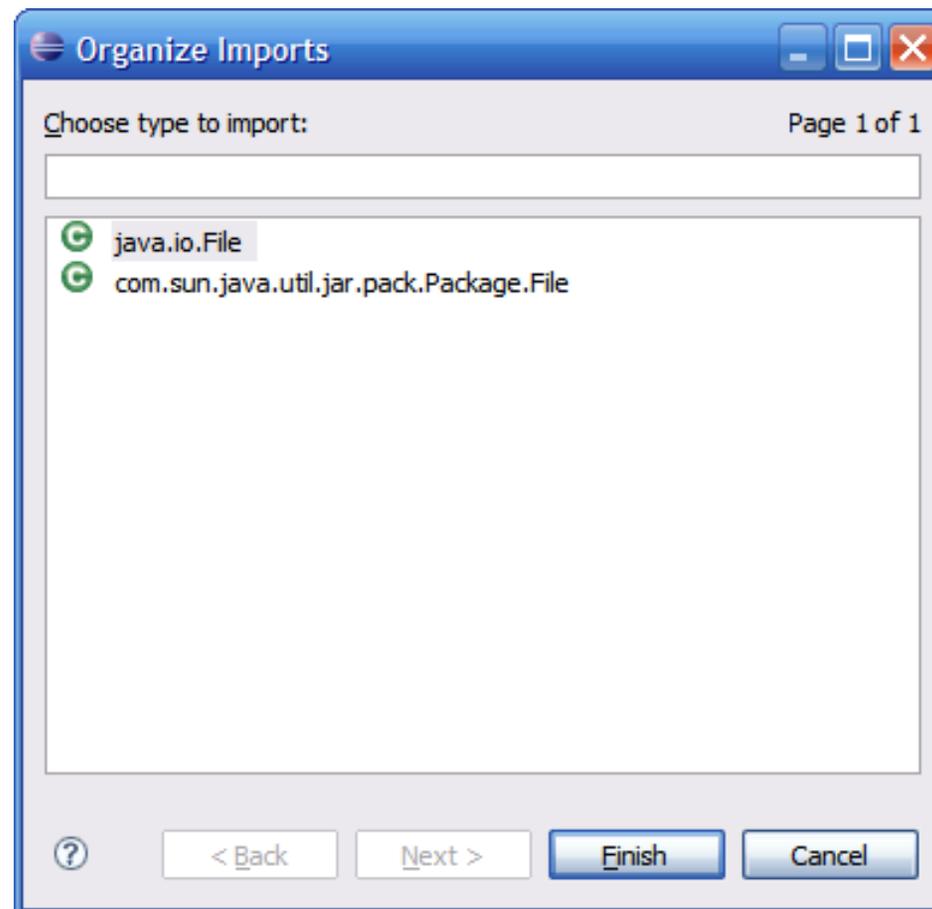
Import Organization

- Eclipse can automatically include import statements for any classes you are using, just press Control + Shift + o (letter o)



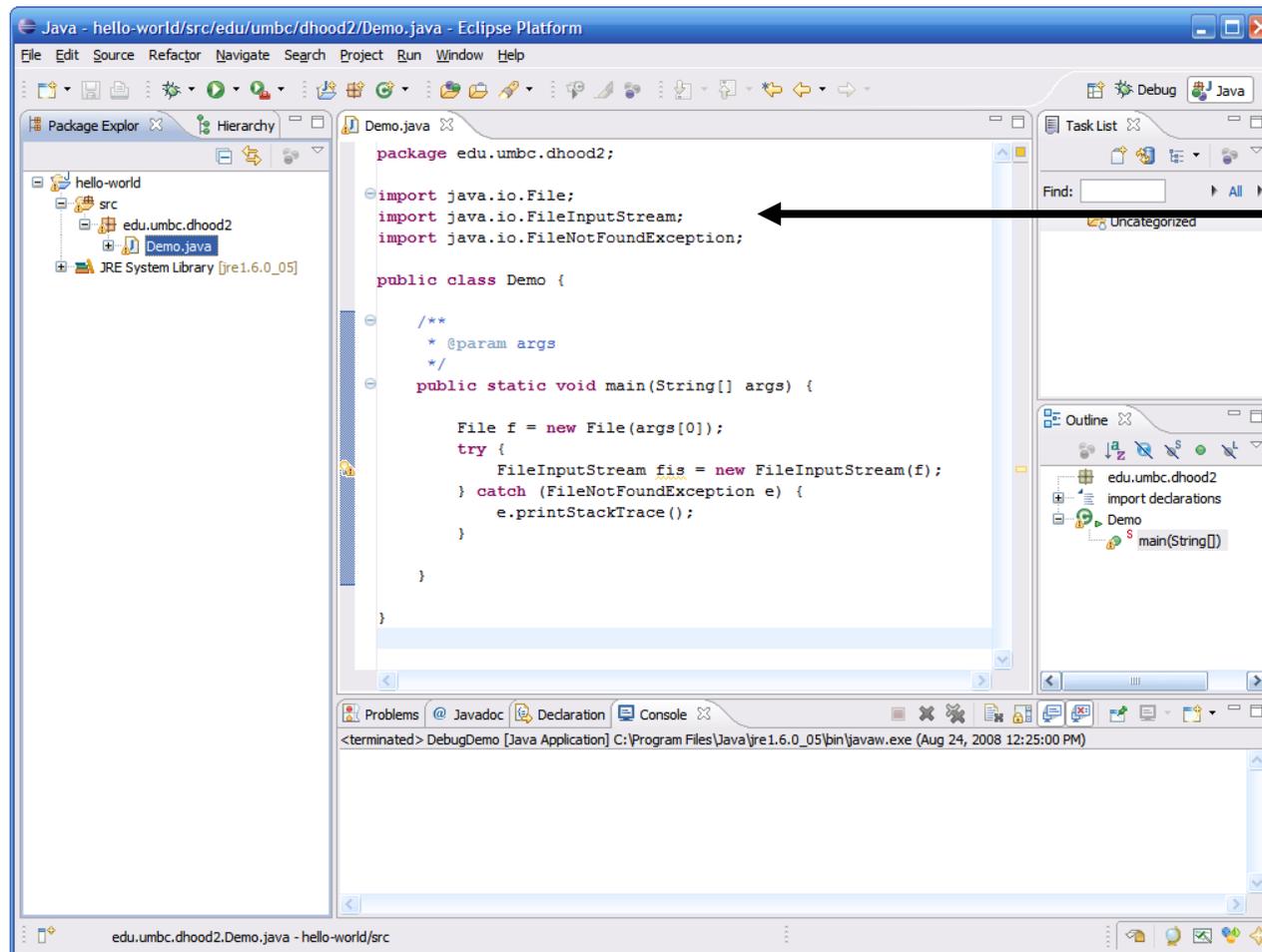
Import Organization (continued)

- If the class is ambiguous (more than one in the API) then it will ask you to select the correct one



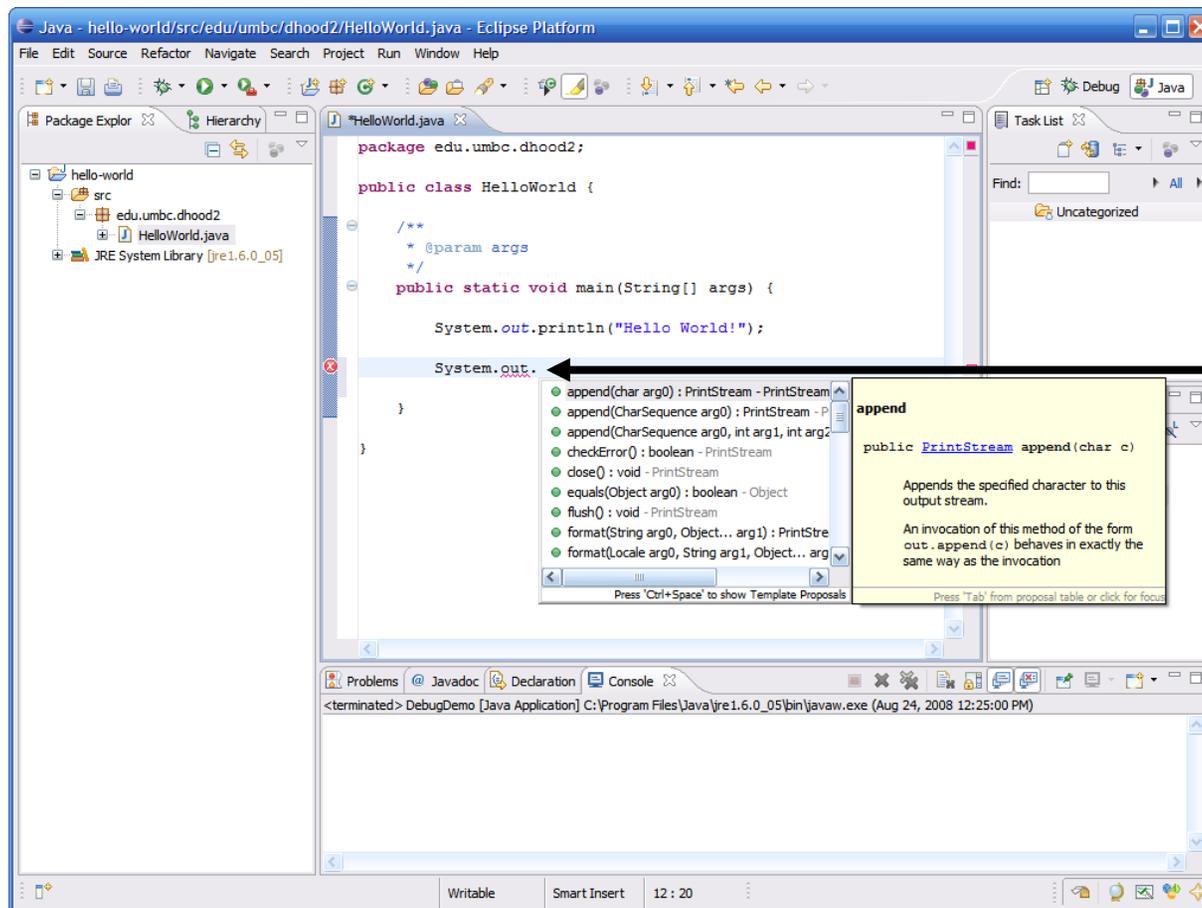
Import Organization (continued)

- Import statements automatically included and organized
 - You can organize imports to clean them up at any time



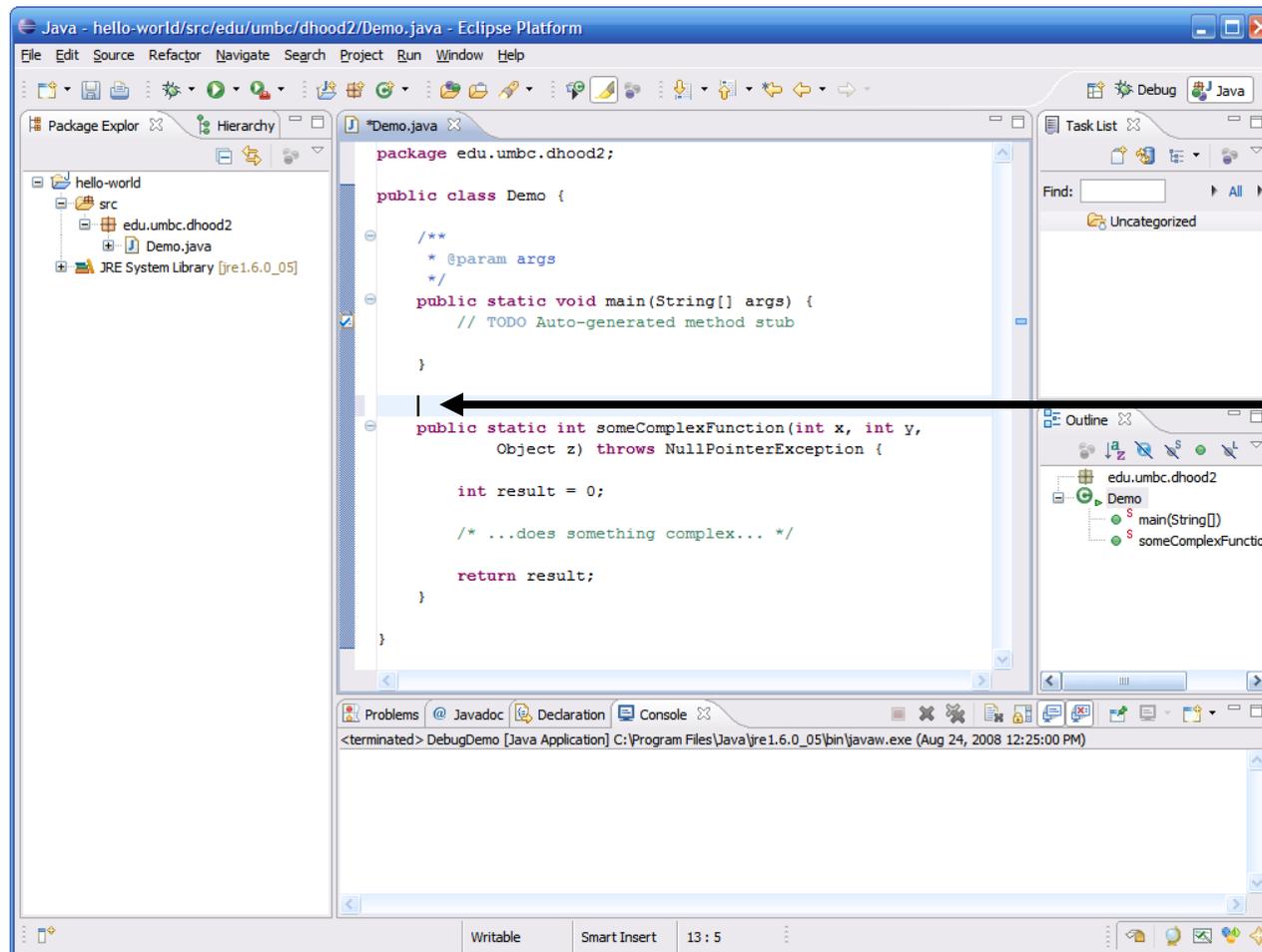
Context Assist

- If you are typing and press a “.” character and pause a second, Eclipse will show you a list of all available methods for the class
 - Prevents having to browse javadocs to see what methods are available
 - Get context assist at any time by pressing Control + Space



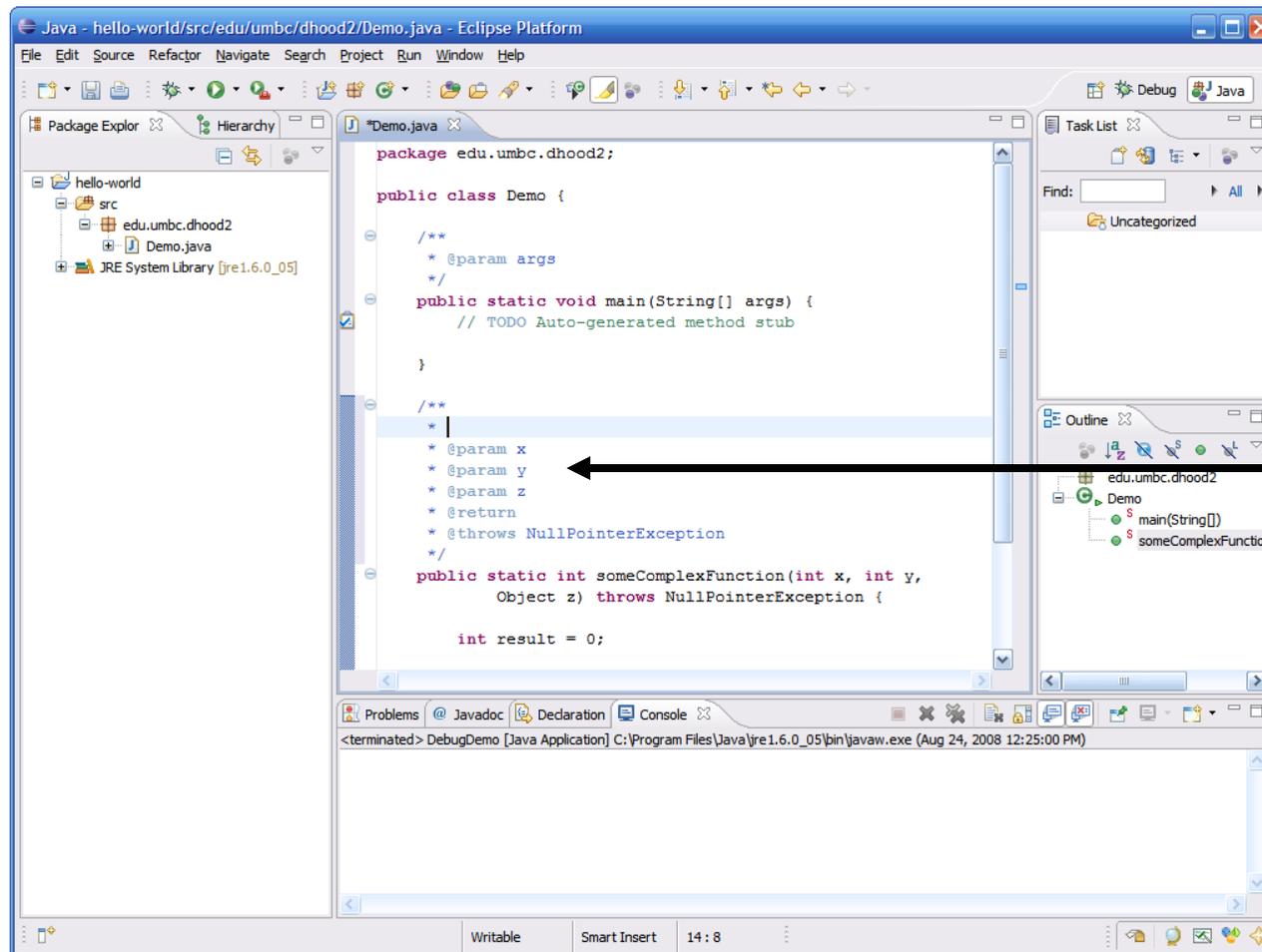
Javadoc Assist

- Eclipse can also help generate javadoc comments for you, simply place the cursor before the method and then type “/**” then Enter



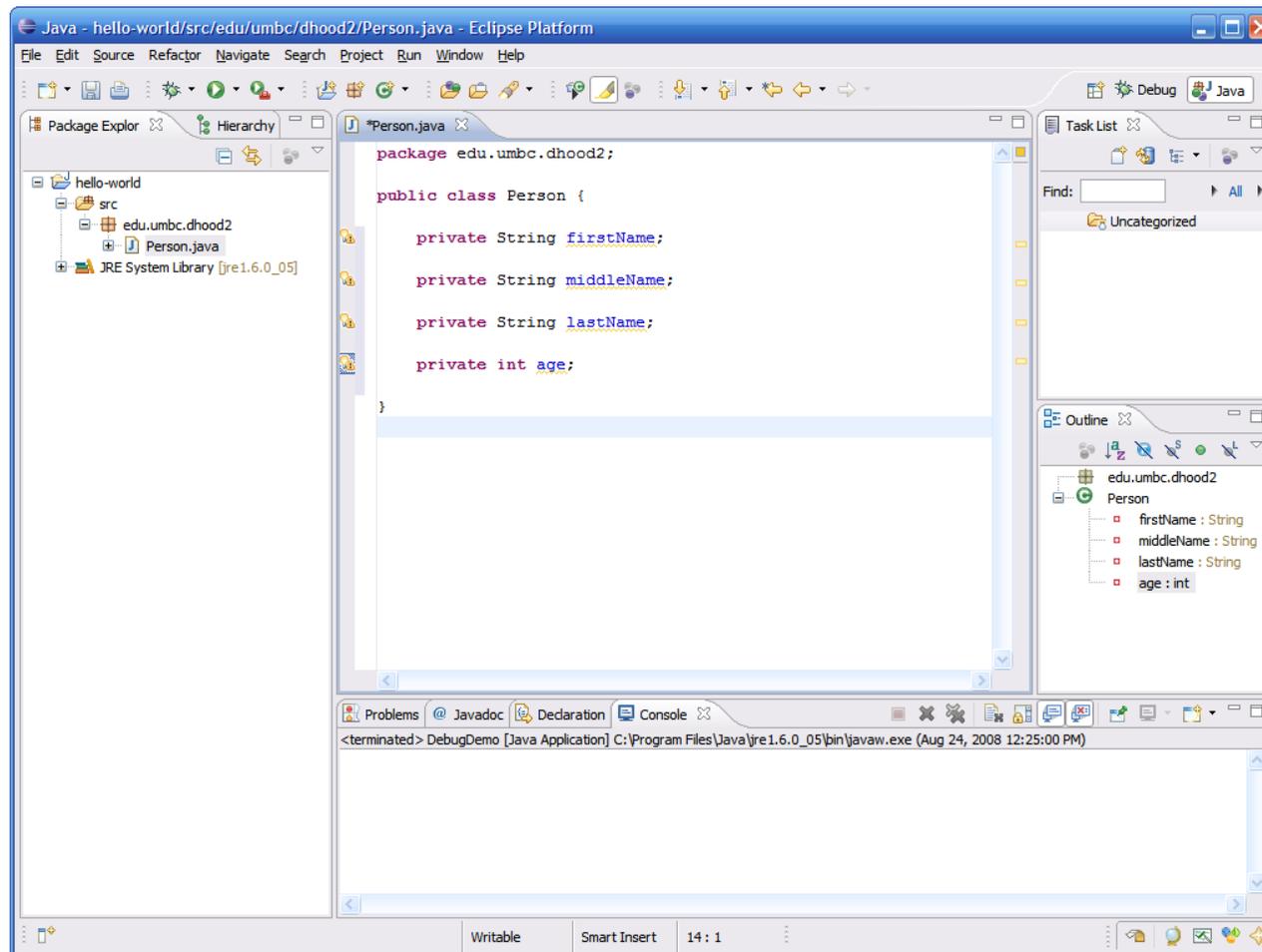
Javadoc Assist (continued)

- Eclipse will automatically generate a javadoc header for the method all stubbed out with the parameters, return type and exceptions



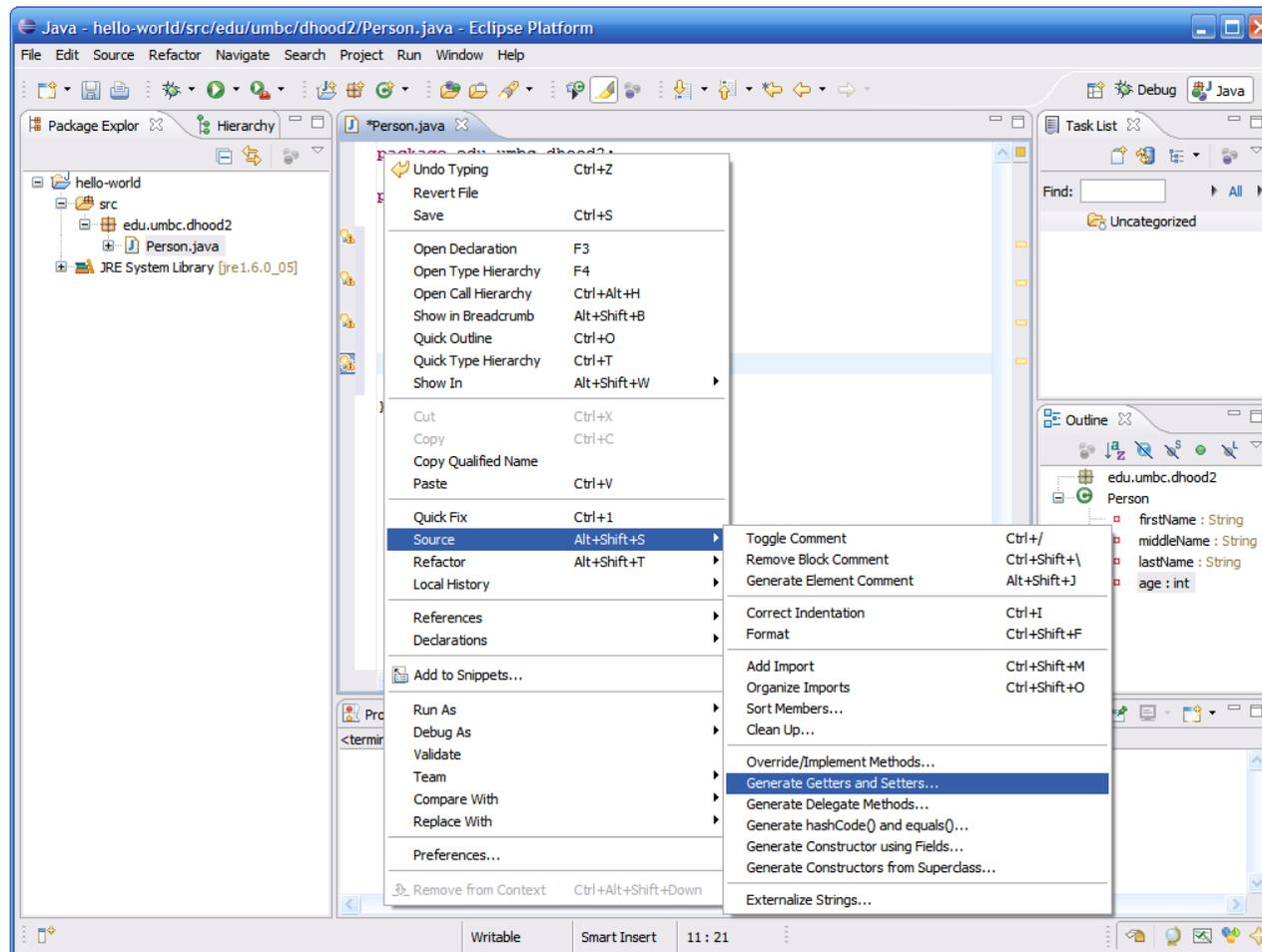
Getter/Setter Generation

- Eclipse can automatically generate getters and setters for member of a class...



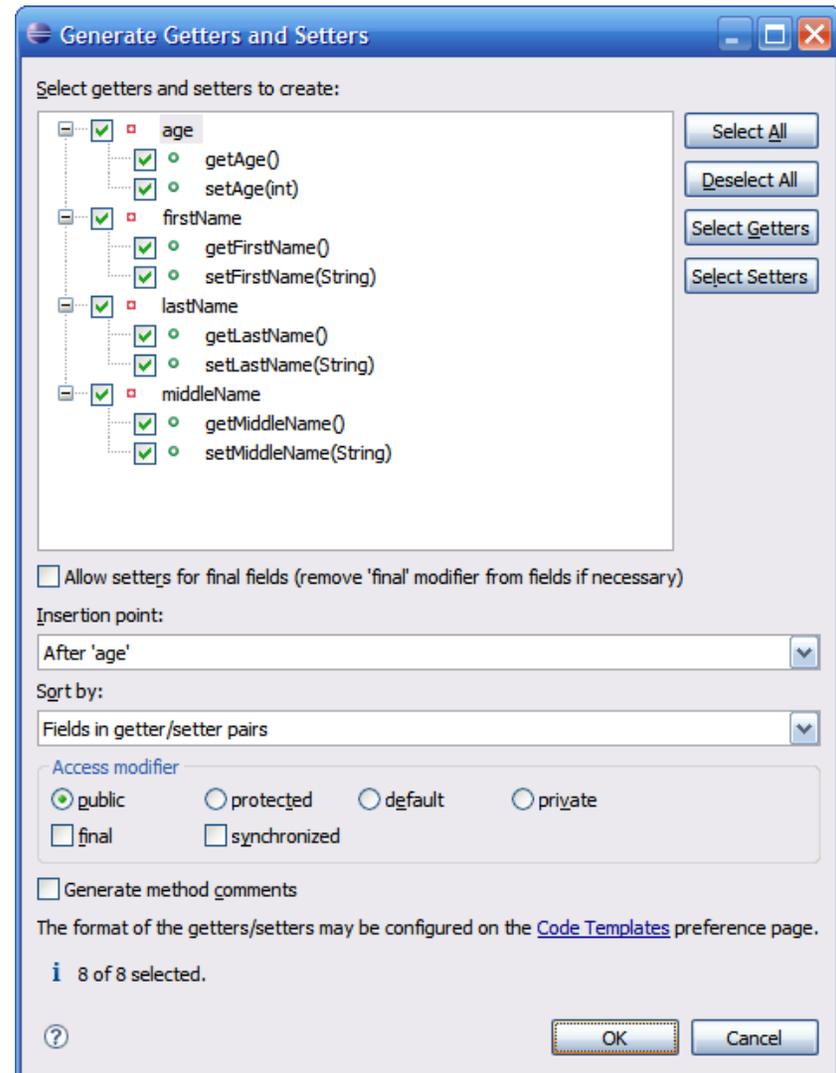
Getter/Setter Generation (continued)

- To generate getters and setters, right click in the main pane, then select Source → Generate Getters and Setters



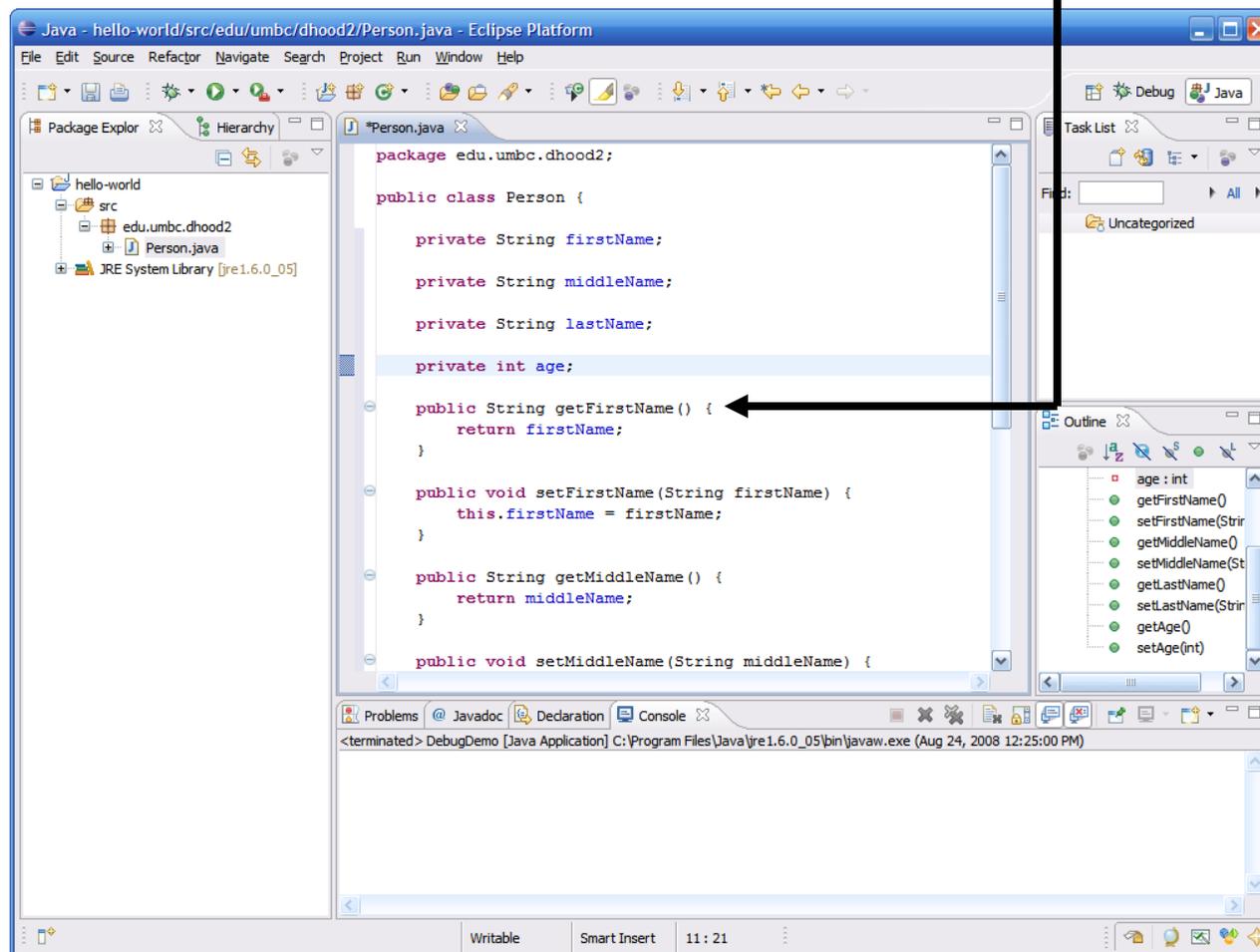
Getter/Setter Generation (continued)

- Here you can selectively choose members for which to generate getters and setters



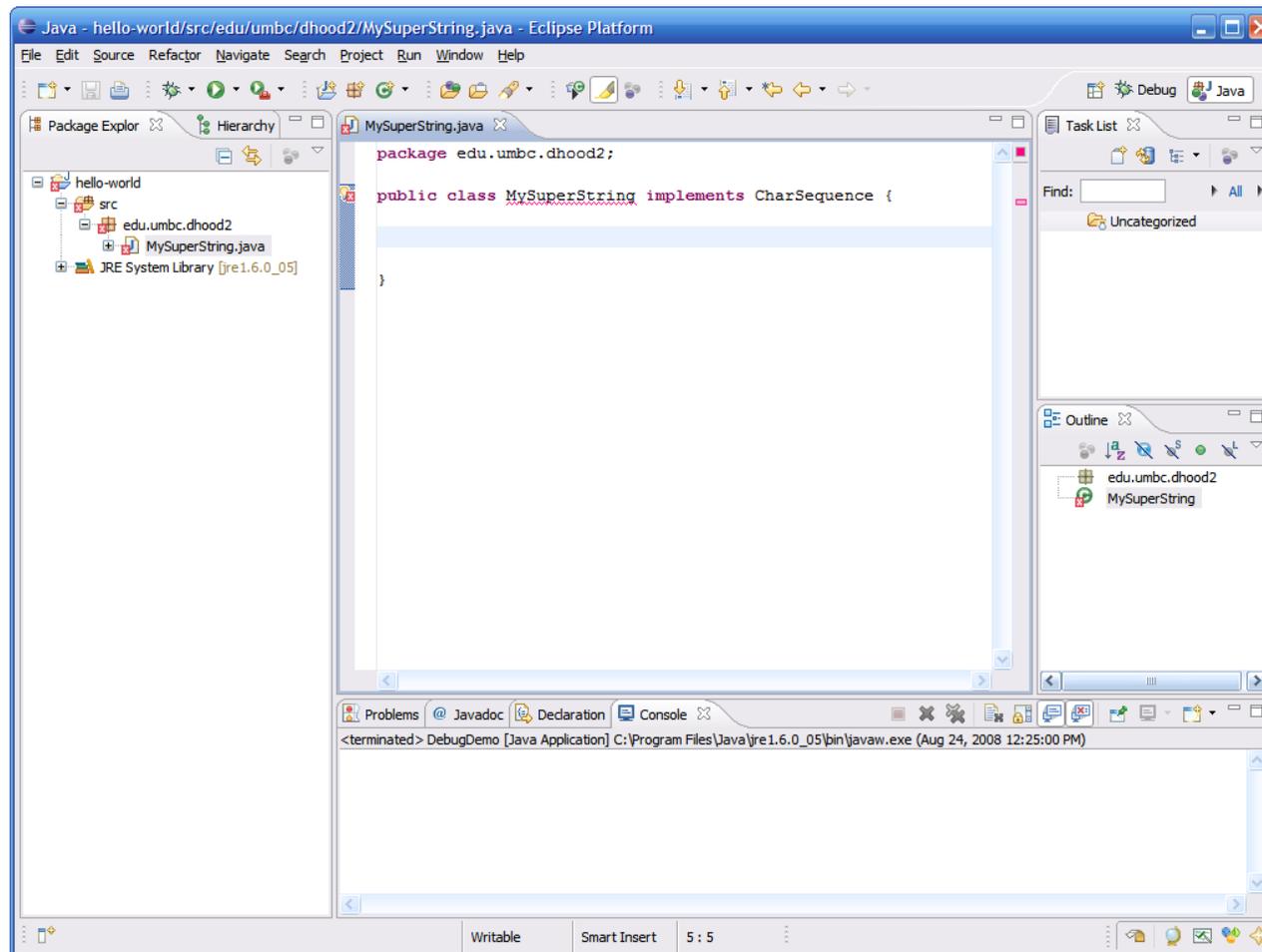
Getter/Setter Generation (continued)

- Eclipse will then automatically generate the code for the getters and setters



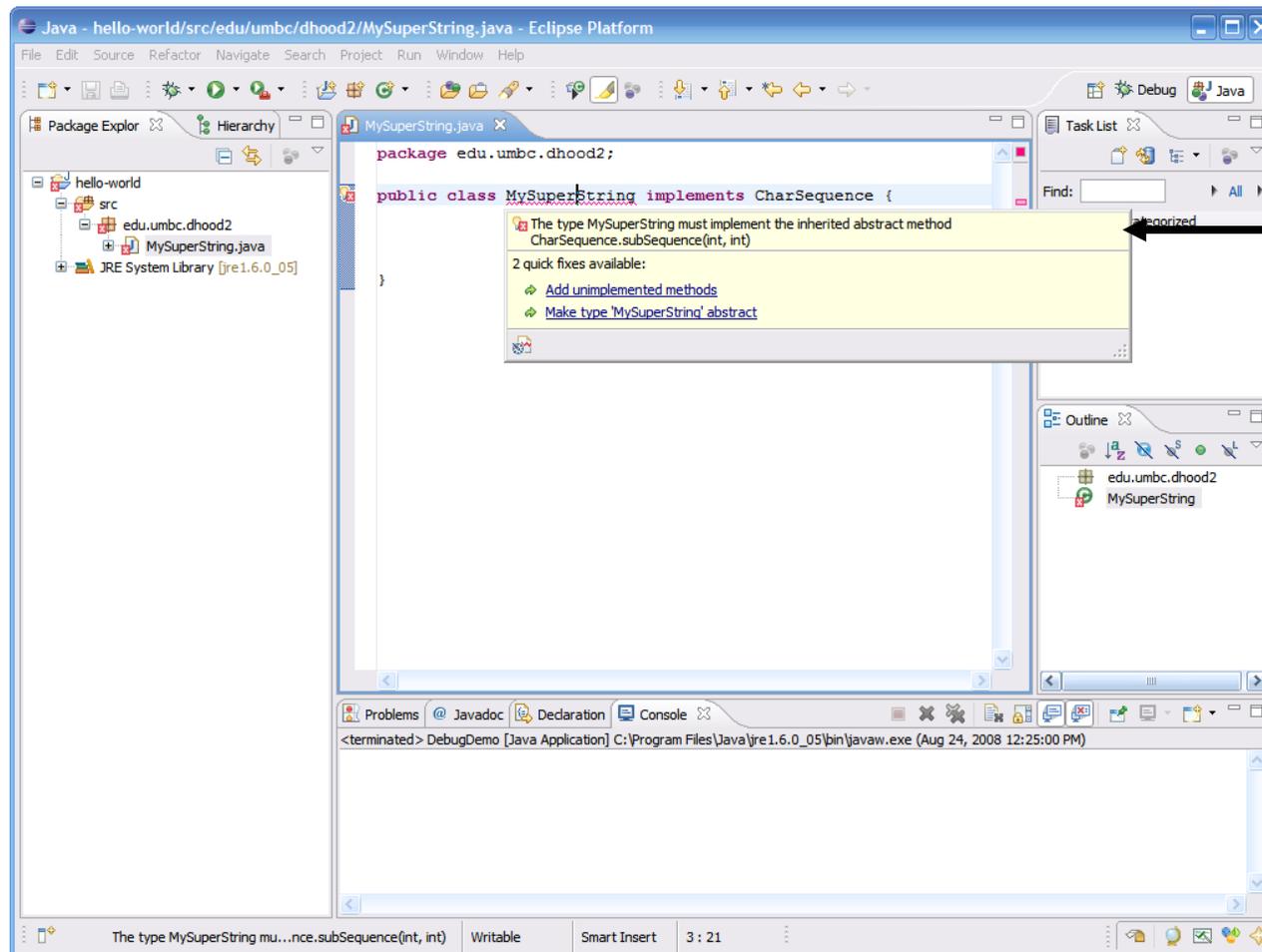
Add Unimplemented Methods

- Eclipse can also stub out methods that need to be present as a result of implementing an interface...



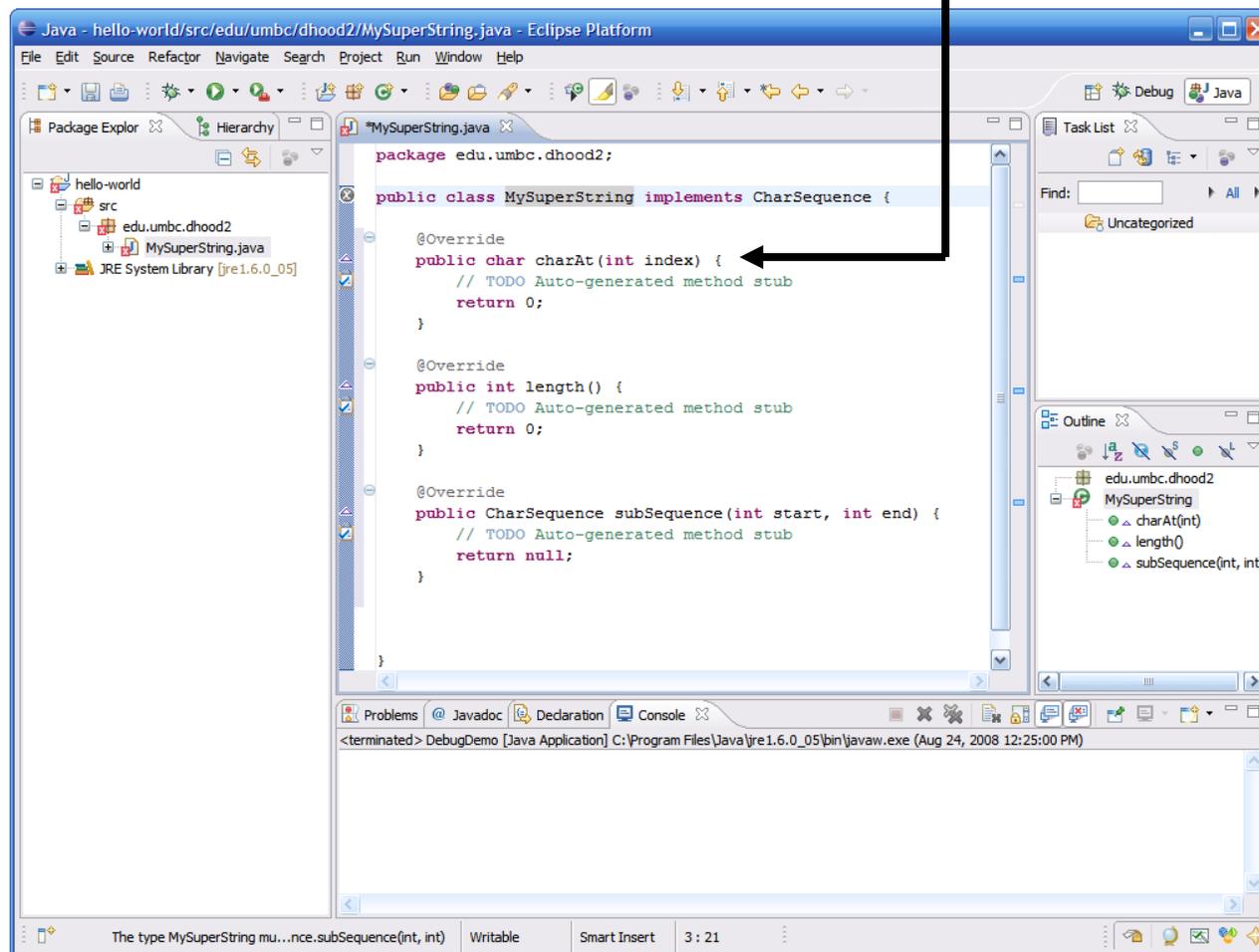
Add Unimplemented Methods (continued)

- You can use the quick fix light bulb to add the interfaces unimplemented methods to the class



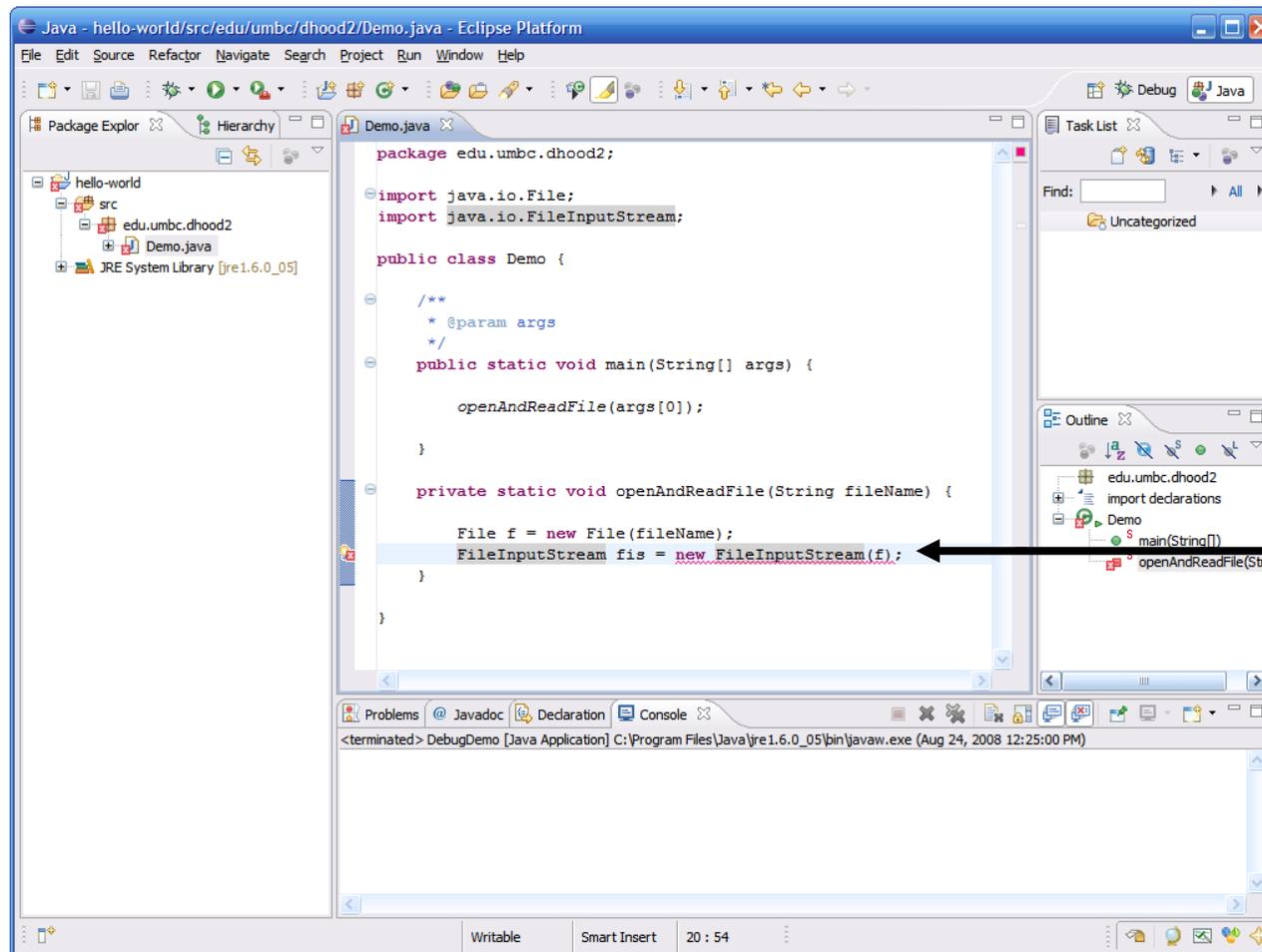
Add Unimplemented Methods (continued)

- Again Eclipse will go ahead and stub out the method for us



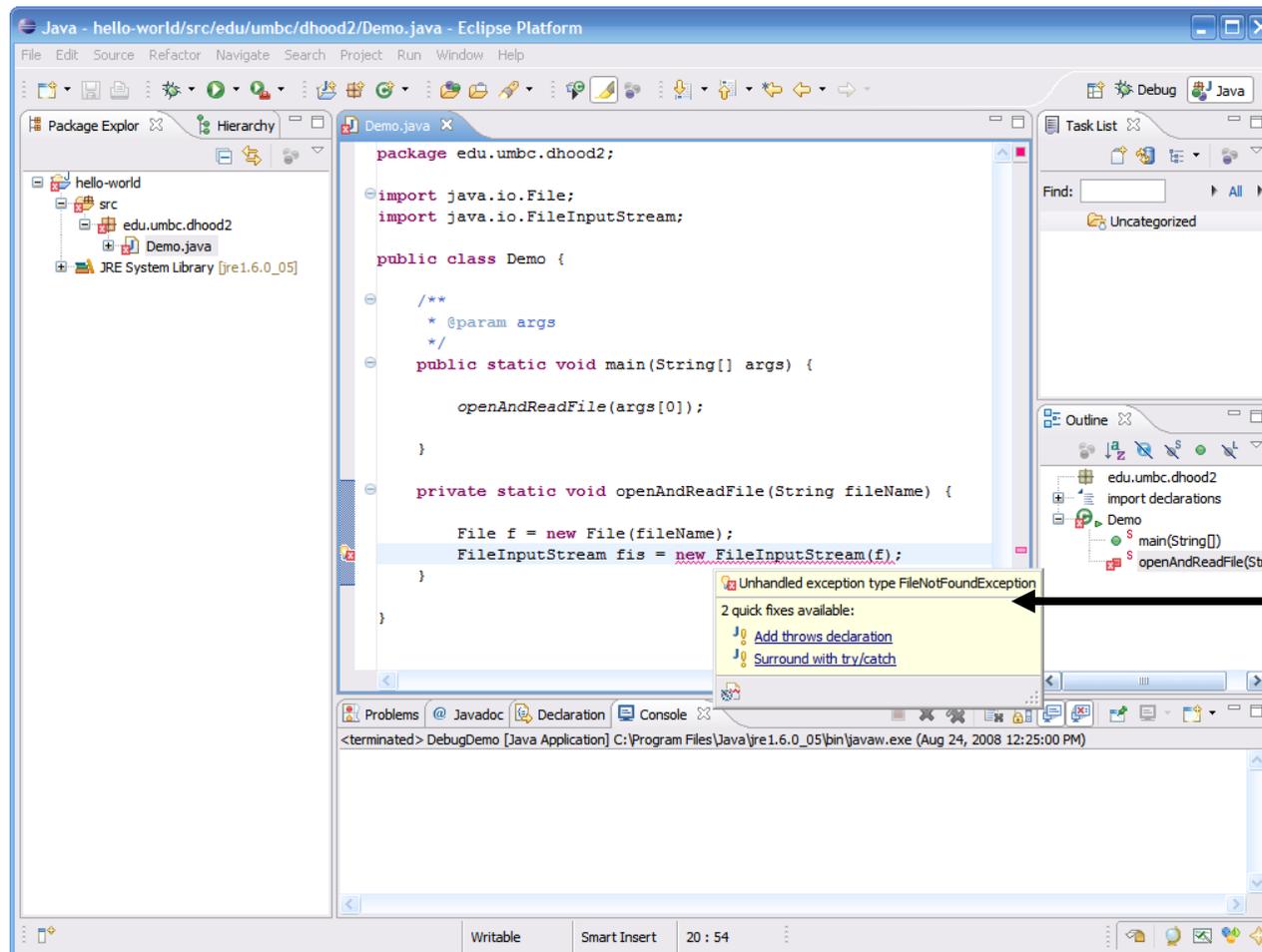
Exception Handling

- Eclipse will also pickup on unhandled exceptions



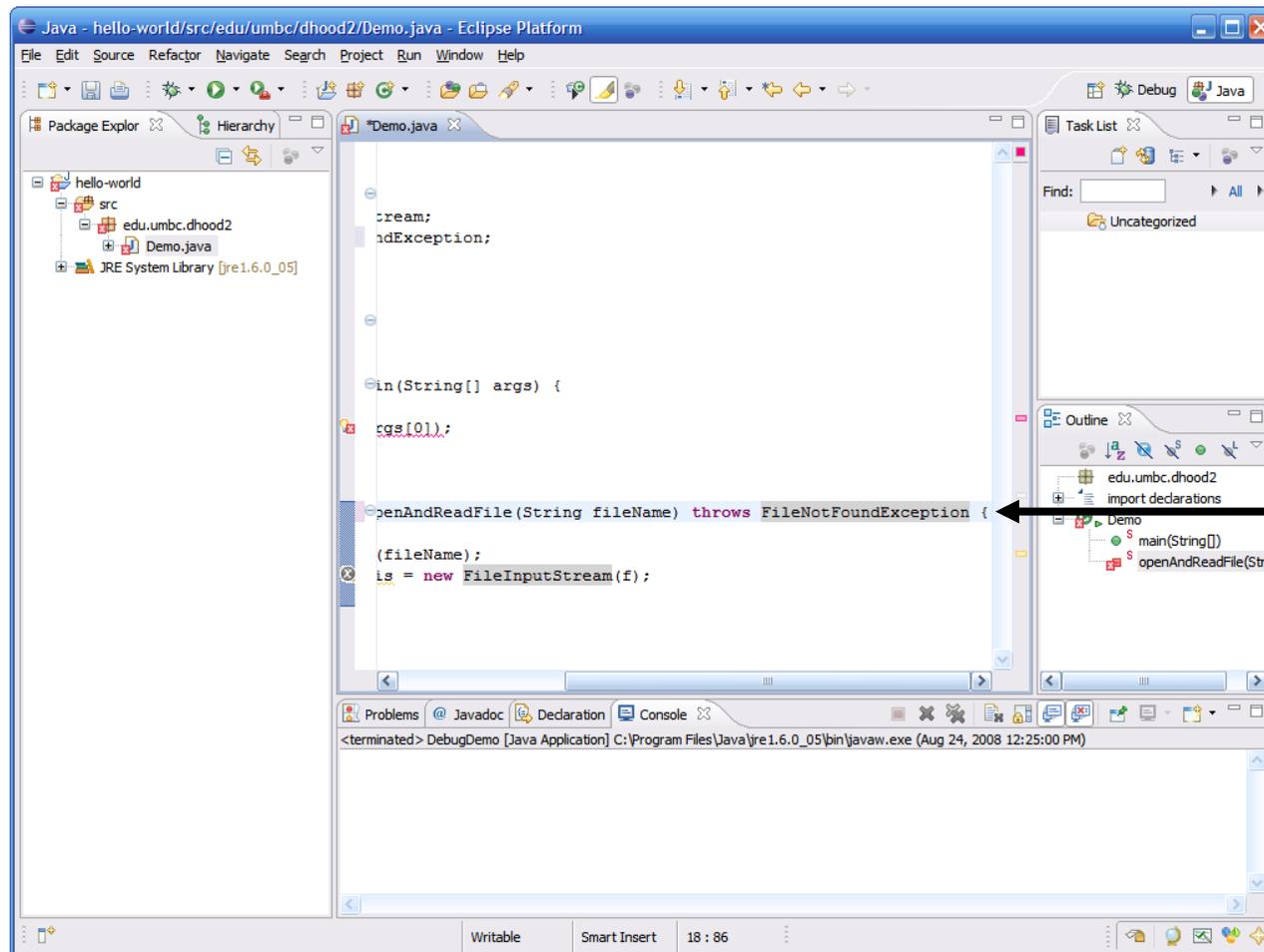
Exception Handling (continued)

- By clicking on the quick fix light bulb, Eclipse can suggest what to do to handle the exception



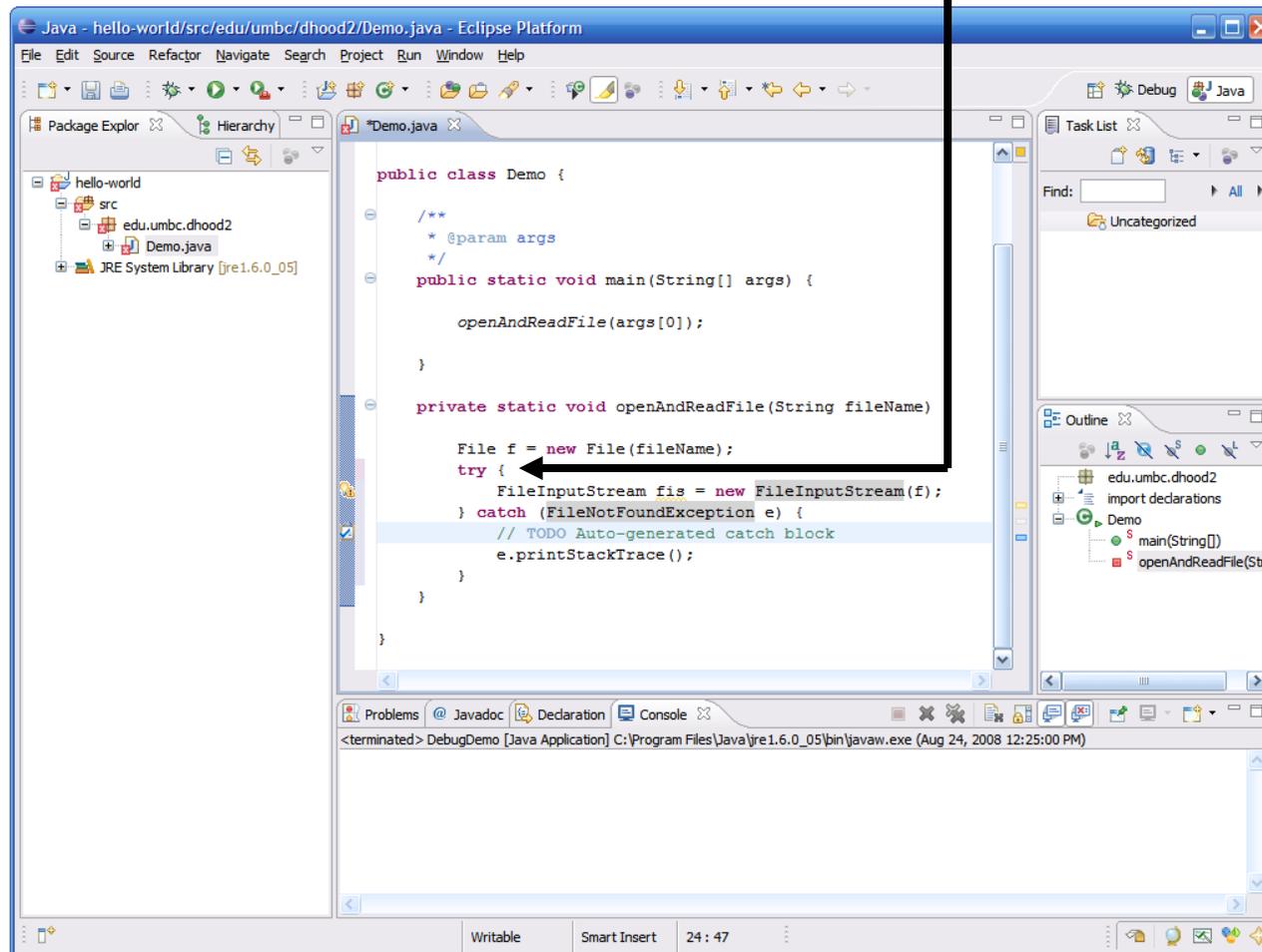
Exception Handling (continued)

- Eclipse can automatically add a “throws declaration” to the method signature



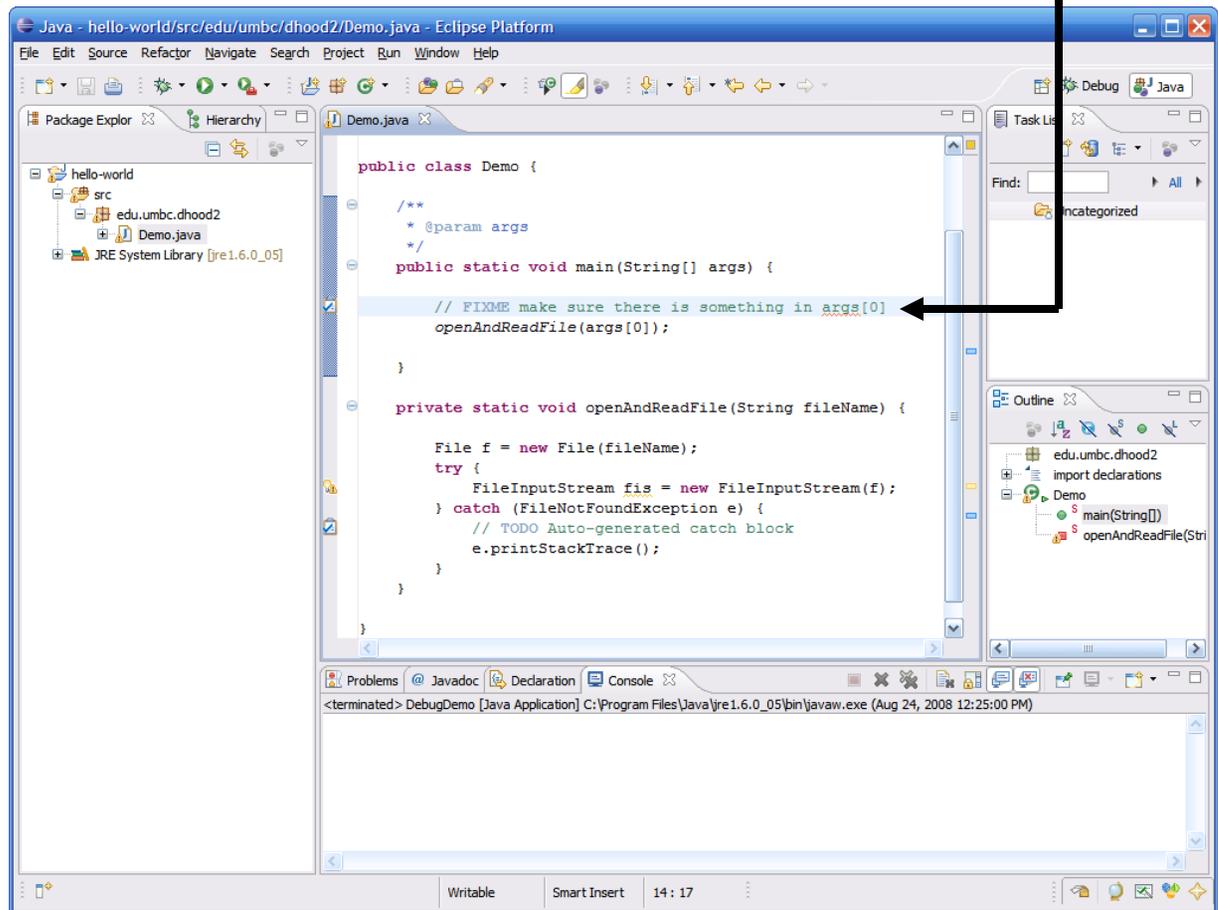
Exception Handling (continued)

- Alternately, Eclipse can also wrap the code inside a try/catch block



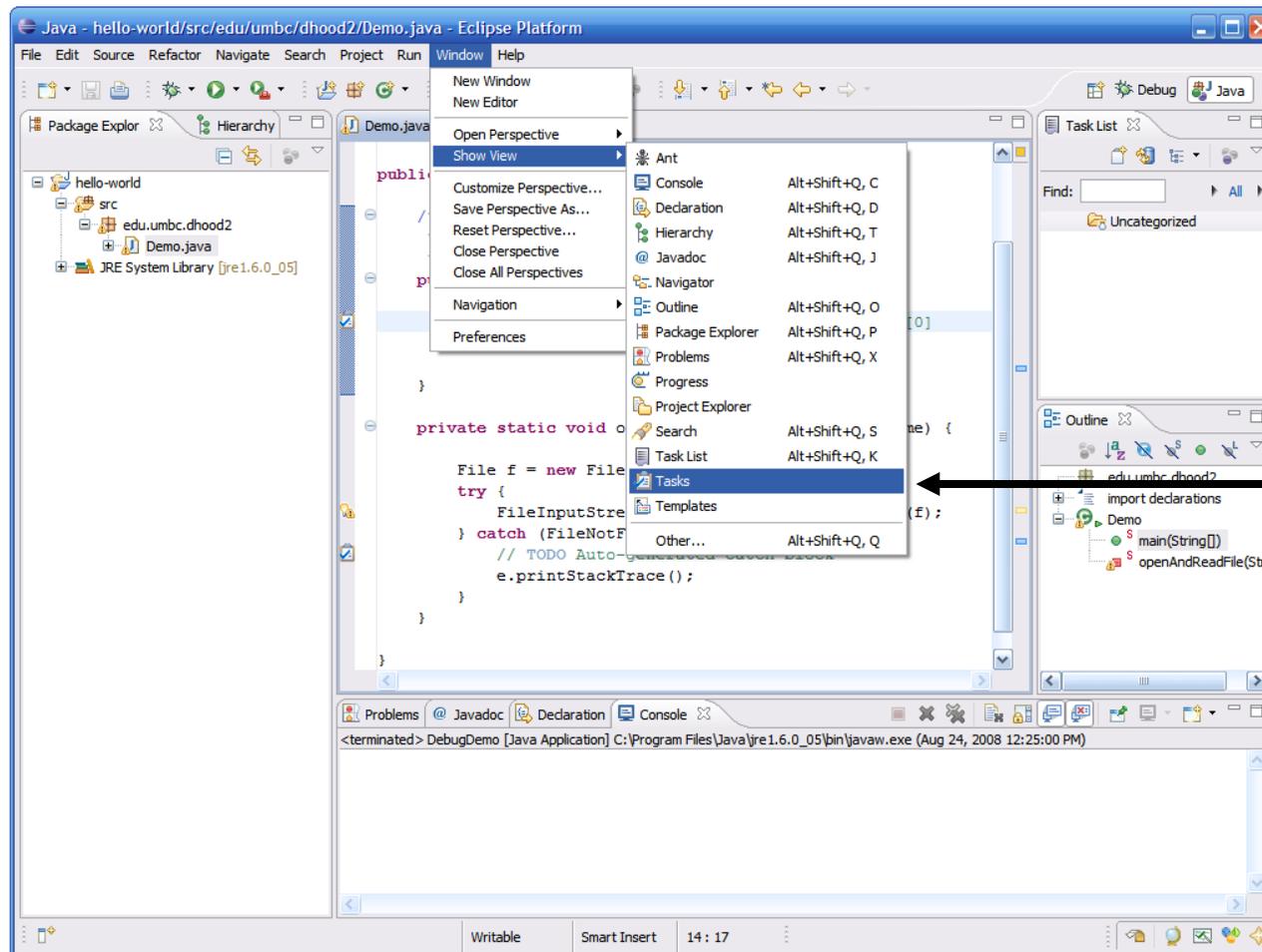
Tasks

- Eclipse allows you to insert reminders into your code and stores them for you to come back and revisit them
- Eclipse recognizes the following tags inside comments...
 - TODO
 - FIXME
 - XXX
- You can even add your own custom tasks through the preferences menu



Tasks (continued)

- To add a table of all reminders in all of your source code you can add the Tasks view by clicking on Window → Show View → Tasks



Tasks (continued)

- This neatly displays all tasks in a tabular form

The screenshot shows the Eclipse IDE interface. The main editor displays the source code for `Demo.java`. Two tasks are visible in the code: a red squiggly line under `args[0]` in the `main` method and a red squiggly line under the `catch` block in the `openAndReadFile` method.

The **Task List** window on the right shows a search field and a list of tasks under the **Uncategorized** group. The **Outline** window shows the class structure with `main(String[])` and `openAndReadFile(String)` methods.

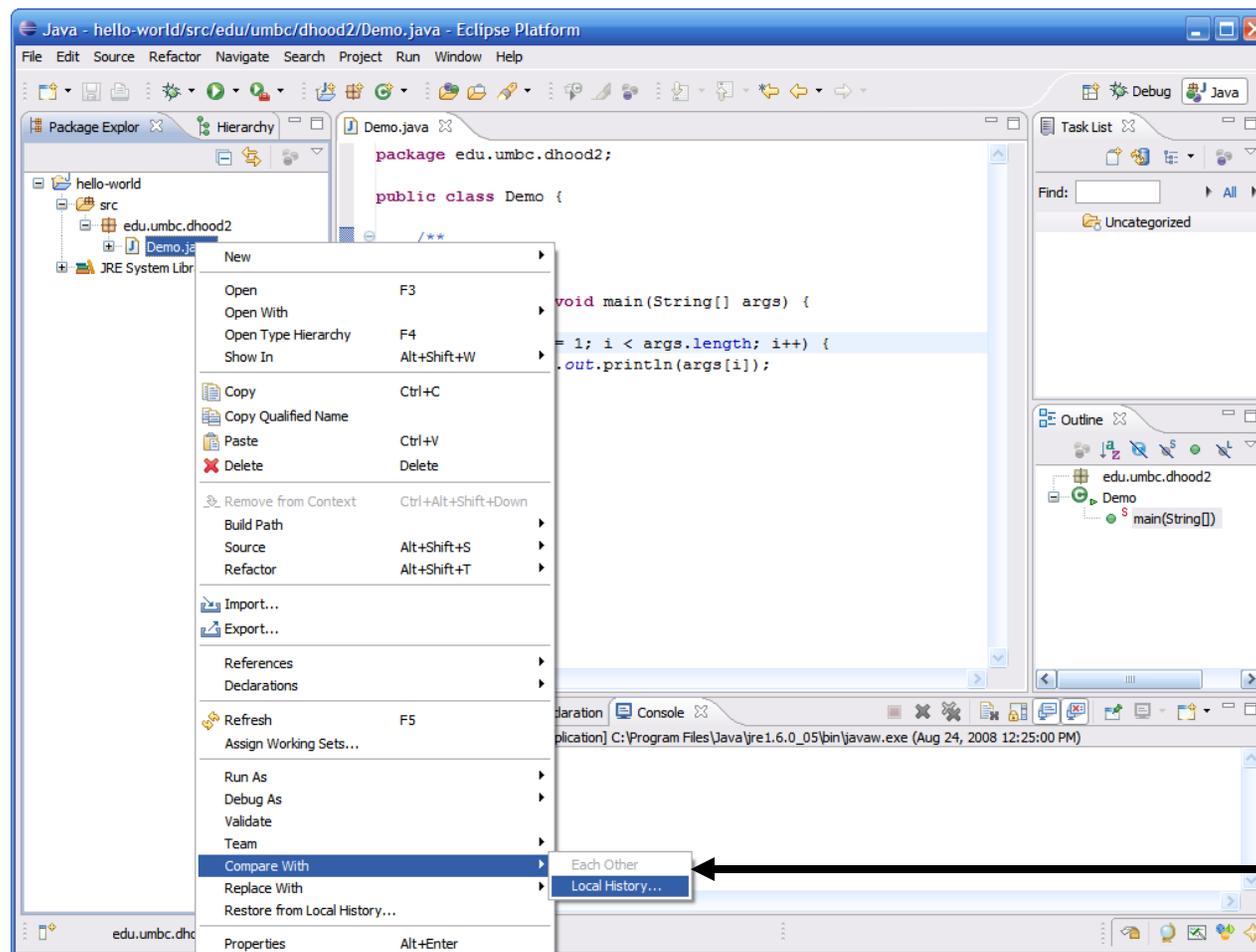
The **Tasks** window at the bottom displays a table of tasks:

	Description	Resource	Path	Location	Type
!	FIXME make sure there is something in args[0]	Demo.java	hello-world/src/e...	line 14	Java Task
	TODO Auto-generated catch block	Demo.java	hello-world/src/e...	line 25	Java Task

An arrow points from the **Tasks** window to the `args[0]` task in the code editor.

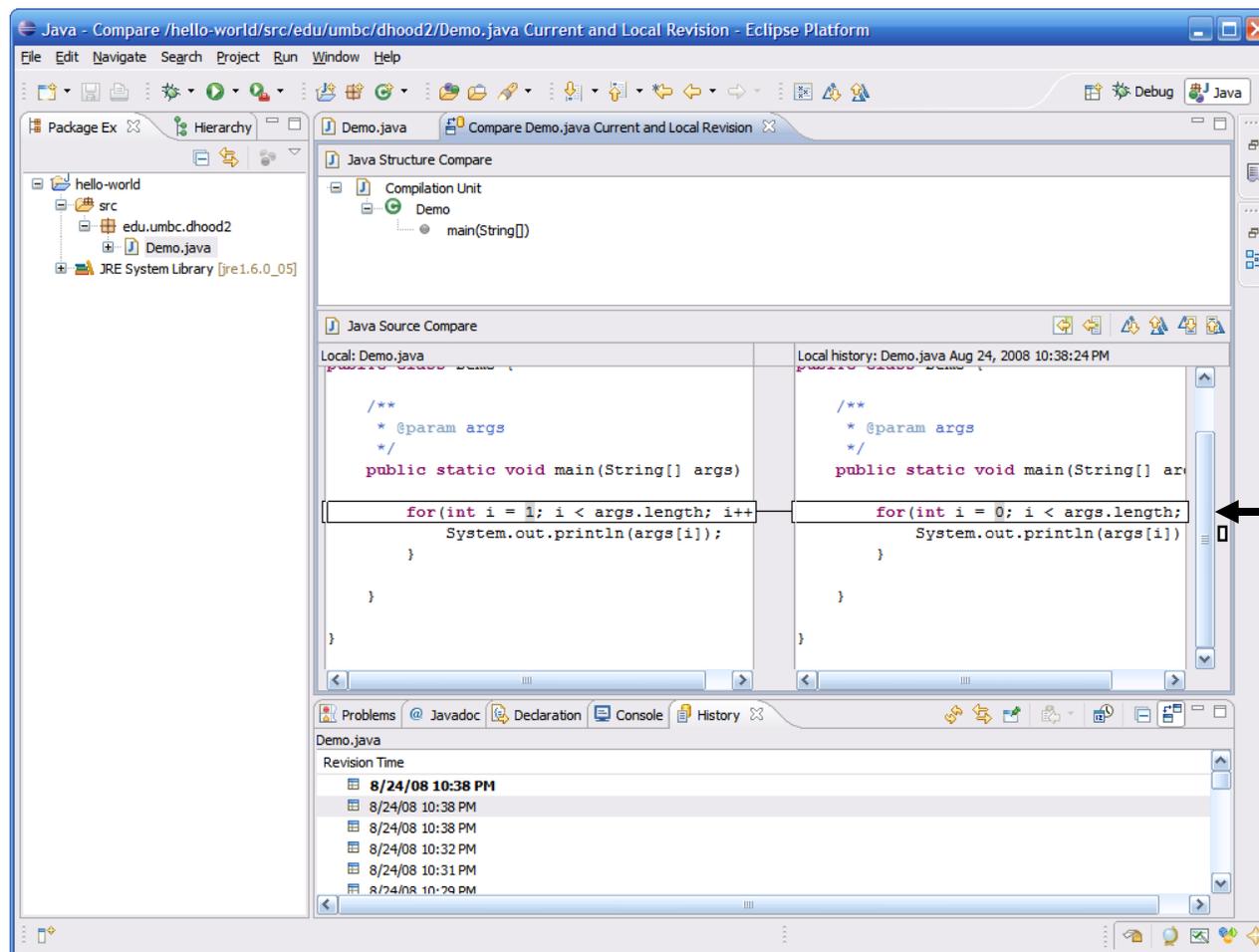
Local History

- Eclipse maintains a local history of file revisions which can be accessed by right clicking on the class, then selecting Compare With → Local History...



Local History (continued)

- Previous saved revisions are displayed in the History pane, double click a revision to view in the built-in diff viewer



Summary

- Benefits
 - Code completion
 - Faster code/compile/run cycles (real time)
 - Open source (free)
 - Extensible (plugins)
- Disadvantages
 - Pretty heavyweight
 - Requires JRE
 - Learning Curve