6.170 Laboratory in Software Engineering
Fall 2005
Problem Set 0: Introduction to 6.170
Due: **Thursday**, September 15, 2005 at 1:00pm

Handout PS0

Contents:

Welcome to 6.170. You may notice that this Problem Set is some 20-odd pages in length. **DON'T PANIC!** In view that some of you may be new to Java, we have added quite a few screenshots and detailed instructions so that you will be able to complete this Problem Set with as much help as you need. Most of the problems consist of following instructions to set up the configuration of your Eclipse IDE (Integrated Development Environment) and some simple exercises to help you become familiar with the tools that you will use for the rest of the course.

For the first problem set only, we encourage you to work in pairs, collaborating as much as you like, without constraint. If you know Java forwards and backwards, feel free to help a fellow student. If Java is all new to you, ask another student for help, along with the LAs, the TAs, and (if you corner them) the lecturers. However, you will submit your problem set and be graded independently. We urge you not to let your buddy "help" you by doing everything for you; this will just make your life harder on the next problem set.

Initial Setup

You must do this step FIRST, even if you are working from home

Before doing any work (problem sets, labs, etc.) for 6.170, you need to set up your 6.170 environment on Athena. (Once you set up your environment, you will not need to do it again this semester.) Even if you choose to work from home, you will ultimately be submitting your files on Athena filesystem, so you must perform this step. On dialup, or while logged in at an Athena workstation, type the following commands at your athena`%` prompt:

```
add 6.170
student-setup.pl
```

If you see the message **6.170 setup complete**, then you are done. **You must logout and login again for the changes to take effect.** This script will add some lockers and set some environment variables that are required for 6.170. These changes will be documented in your `~/.environment` file. If you see an error message when running `student-setup.pl`, then contact the course staff for assistance.

Introduction

In this assignment, you will learn how to edit, run and test Java code. We will introduce several tools to help you with these tasks, namely Eclipse, CVS, and JUnit. These are tools commonly used by developers both in academia and in industry, so your familiarity with them will be a valuable skill.

Though we provide some instructions on how to complete 6.170 assignments on your own computer with your own tools, **you are strongly encouraged to complete this assignment first on Athena using exactly the tools we describe here**. If you plan to work on a home computer, you should do the problem set on Athena and *also* on your home computer since there will be slight differences in the setup. See the Working at Home handout for tips on getting set up. If you plan to work on your own laptop, you may wish to bring the laptop to an Athena cluster and do the problem set there, since LAs will be available to help you out if you run into problems. Please note that if you choose to use Eclipse on your own machine instead of on an Athena workstation, and you have problems, then you are again generally on your own, however, some assistance may be possible if you are running Linux, Windows XP, or MacOS X.4

If you get stuck on something in this problem set, you should first check the FAQs/corrections for the problem set. Next, check the online forum to see if someone else had earlier posted a similar question. If you can't find what you need, ask an LA or TA in person. Finally, if you cannot locate a TA or LA, post to the online forum.

Note: If you would like to get more practice with Java, then we recommend walking through Sun's Java Tutorial. If possible, try to complete Problem Set 0 first so you can use the tools we describe here when doing the examples in Sun's tutorial.

Problem 1: Running Eclipse for the First Time on Athena

After running the `student-setup.pl` script described in **Initial Setup**, run the following command from the `athena%` prompt:

```
eclipse
```

It is extremely important that you run Eclipse on Athena using exactly this command so that you run the correct version of Eclipse with the correct workspace. If you are working from home, please note that you will need to install Eclipse version 3.1m4 or greater since Java 1.5 semantics were not supported until release 3.1m4.

If this is your first time running Eclipse, you may need to click on the "Workbench" icon on the initial Eclipse screen. Subsequent runs of Eclipse should bring up the Workbench screen automatically.

Then, at the command line, you should see the following:

```
Please input workspace directory [/mit/<user path>/workspace]?
```
where *`<user path>`* corresponds to your Athena locker. In response, type:
```
/mit/<username>/6.170
```
where *`<username>`* is your Athena username. (**Note:** Do not enter "`~/6.170`". Eclipse does not recognize the '~' character. You have been warned.) This directory should have been created for you by `student-setup.pl`. It has special permissions that lets the 6.170 staff read this directory, so you should use it as your workspace when using Eclipse for 6.170. Note that when you start Eclipse in the future on Athena, the startup prompt will be:
```
Please input workspace
directory[/mit/<username>/6.170]?
```
in which case you can just hit **Enter** because the default value for the workspace is correct. Now that you have Eclipse up and running, you can proceed to Problem 2.

There is nothing to turn in for this problem.

Note: Eclipse is quite a memory- and computational-intensive application, so you may wish to avoid the SunBlade 100's on Athena and use the other machines instead.
Also, be sure that you close Eclipse before logging out of an Athena terminal. In the past, some students have left themselves logged into an Athena terminal without shutting down Eclipse properly and then switched to an Athena machine on another platform (say switching from a Linux machine to a Sun machine). When this happened, the conflicting

operating systems would end up clobbering the student's workspace data, potentially causing the student to lose a lot of work if the data were not committed to CVS. When this happens, Eclipse can be set right again by clearing out the `~/6.170/.metadata` directory, but then you will have to check out your code from Eclipse again and reset all of your preferences. Avoid this hassle by closing Eclipse properly. Note also that when you switch between Linux and Sun machines on Athena, the Eclipse configuration settings are different. As a result, you will find that your projects are no longer visible. This is not a problem. Please see instructions on how to import an existing project in the Q and A Section.

Problem 2: Checking Out From CVS

The first tool that we will introduce is CVS which stands for *Concurrent Versions System*. CVS works by storing a central **repository** containing the most recent version of your files. You can create a personal and local copy of these files by **checking out** these files into your local directory.

You should do your work and make changes to the files in your checked out copy of the repository in your local directory. If, in the meantime, new versions of the files have been put into CVS, then you can perform an **update** to merge those changes from the central copy into your copy of the files. For this problem set, you will not need to worry about performing an update, since you will be the only person accessing your repository. In future projects, as you work in groups, CVS will be handy for keeping your files synchronized with the rest of the group.

CVS allows you to **commit** the current version of your code to the repository while you are working on it. Some of the advantages of CVS are:

1. CVS allows you to work at multiple locations and to synchronize changes between them automatically. For example, if you decide to work on Athena and on your home machine, you will want access to the same set of files and you want a way to automatically duplicate changes made on one machine to the other.
2. CVS allows you to revert to a previous version of your code that you committed.
3. Your CVS repository serves as a backup copy of your code. In practice, it is also a good idea to periodically make a backup copy of your CVS repository, but you will not have to worry about that in this course.
4. CVS enables multiple people to work on the same file at the same time. As you will be the only person working on your problem sets, this may not seem like an important feature now, but it will be critical when your team begins its final project.

The default location of your CVS repository for 6.170 is on Athena at:

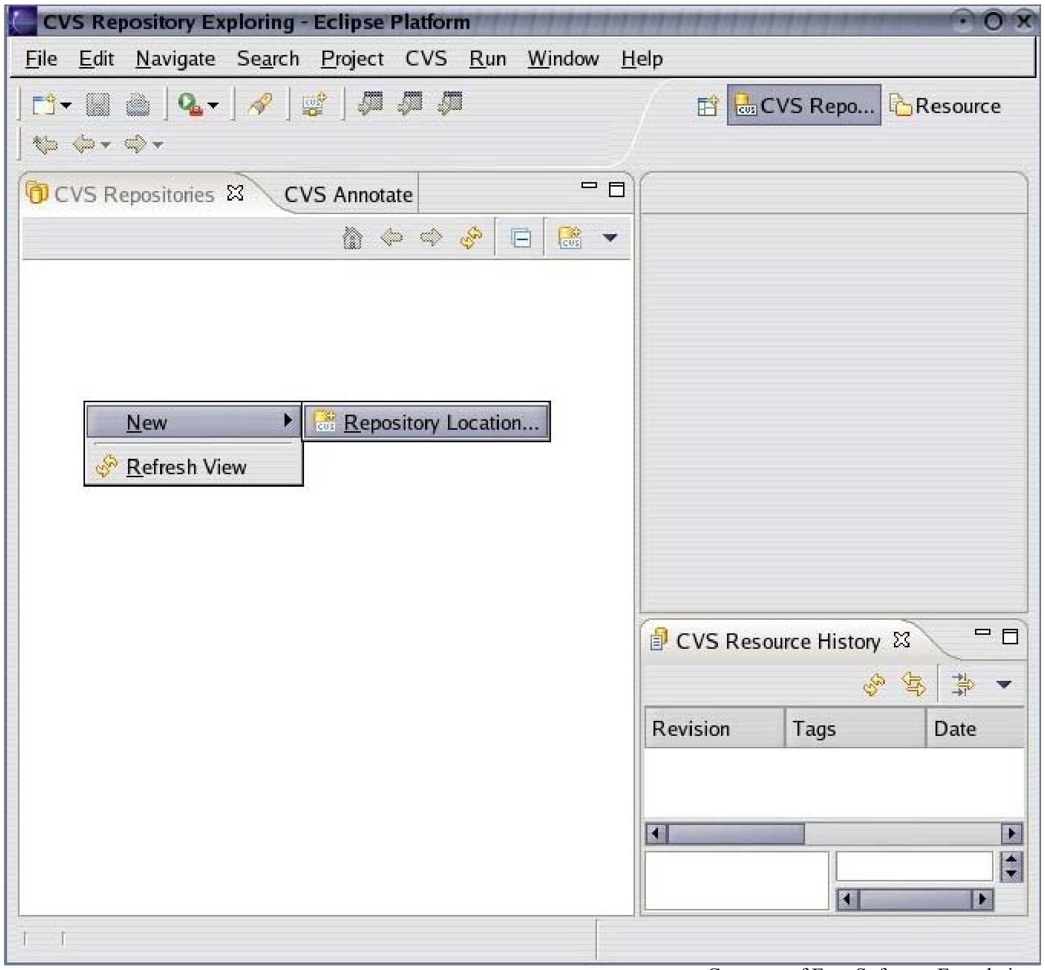
```
~/6.170/cvsroot
```

This repository was created by the `student-setup.pl` script. By running that script, you have granted the 6.170 staff read/write access to this directory. Please do not change this permission because we will be downloading problem set source code directly to this directory for your convenience and we will also be uploading your problem set solutions from there. If you are not familiar with CVS, you may want to glance at a CVS overview.

Using CVS From Eclipse on Athena

We will now show you how to use Eclipse to check out CVS modules with an example. We have created a CVS module called **lib6170**. This module contains all the libraries that you need to compile the code for this and future problem sets. You only have to check this out once (if you work from home, you will need to check this out again with Eclipse on your home computer or laptop).

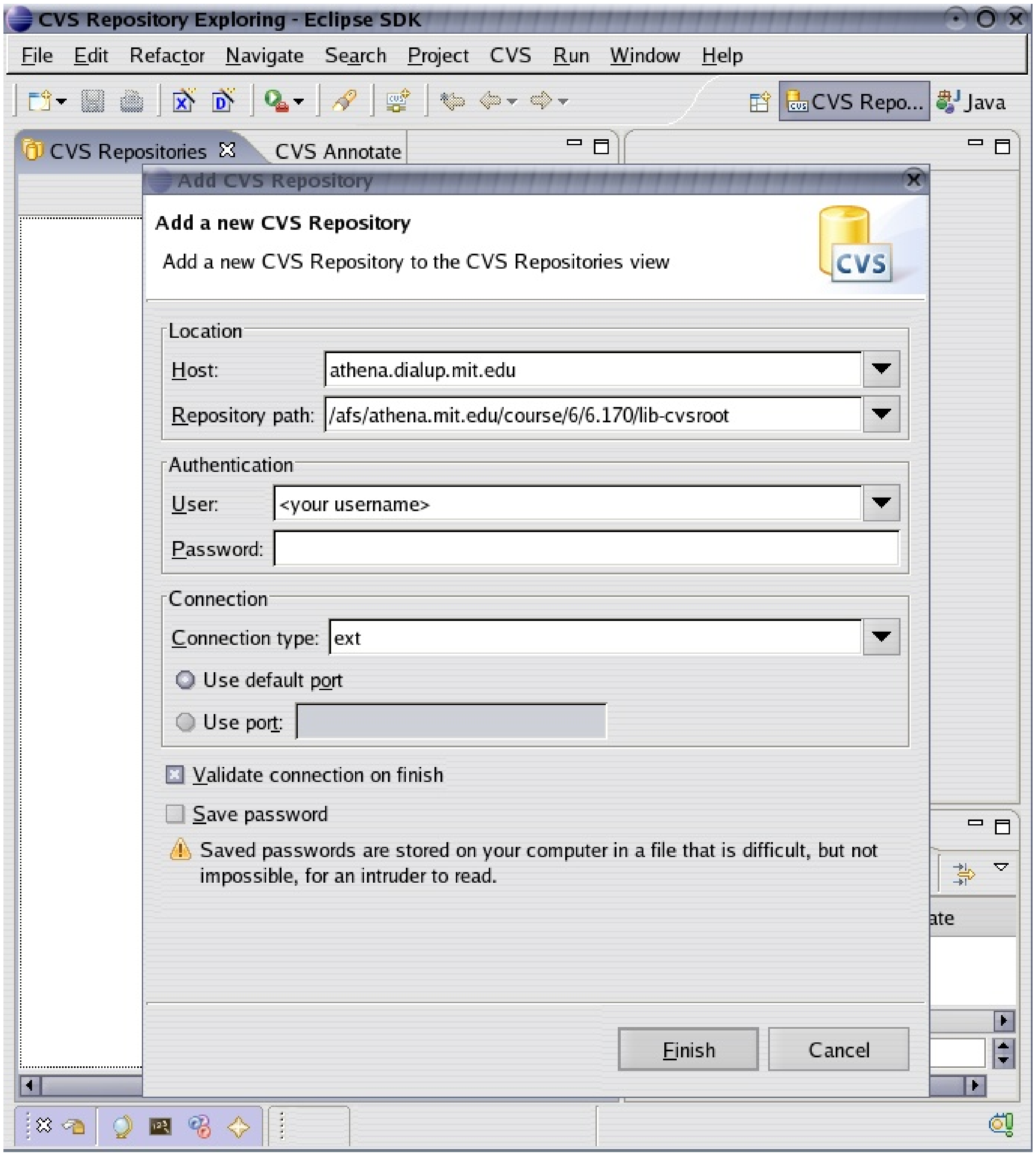
- Start Eclipse by typing `eclipse` in an Athena terminal as described earlier.
- Select **Window** >> **Open Perspective** >> **Other...** >> **CVS Repository Exploring**
- Right-click in the `CVS Repositories' Window, and select **New** >> **Repository Location...**



Courtesy of Free Software Foundation.

- Fill in the fields as follows, and as shown below:
  Host: **athena.dialup.mit.edu**
  Repository path: **/afs/athena.mit.edu/course/6/6.170/lib-cvsroot**
  User: *<your username>*
  Password: *leave this field blank!*
  Connection type: **ext**

- Click **Finish**


CVS Repository Exploring - Eclipse SDK
File Edit Refactor Navigate Search Project CVS Run Window Help
CVS Repo... Java
CVS Repositories
CVS Annotate
Add CVS Repository

Add a new CVS Repository

Add a new CVS Repository to the CVS Repositories view

Location

Host: athena.dialup.mit.edu

Repository path: /afs/athena.mit.edu/course/6/6.170/lib-cvsroot

Authentication

User: <your username>

Password:

Connection

Connection type: ext

Use default port

Use port:

[✓] Validate connection on finish

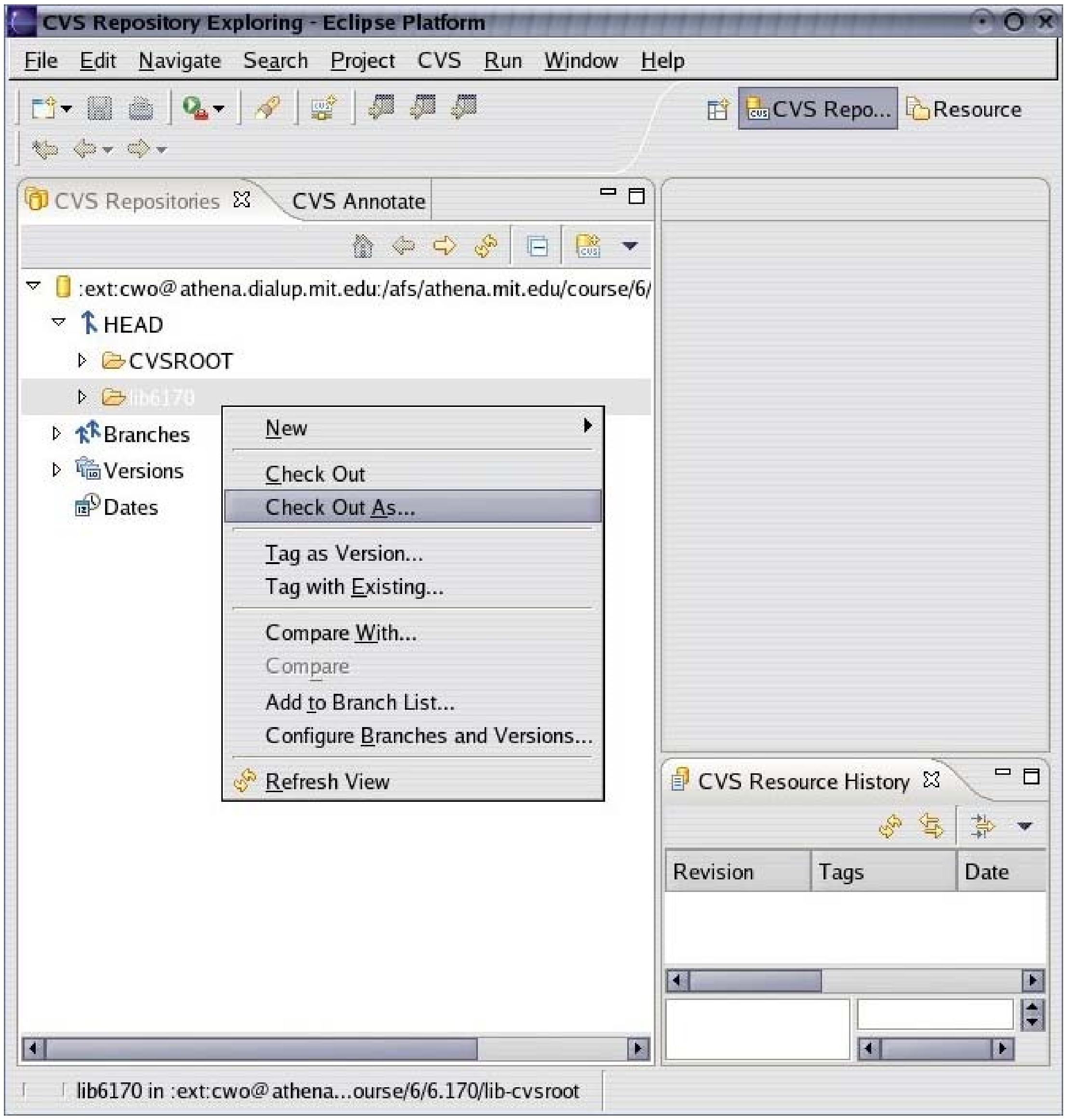
[ ] Save password

Saved passwords are stored on your computer in a file that is difficult, but not impossible, for an intruder to read.

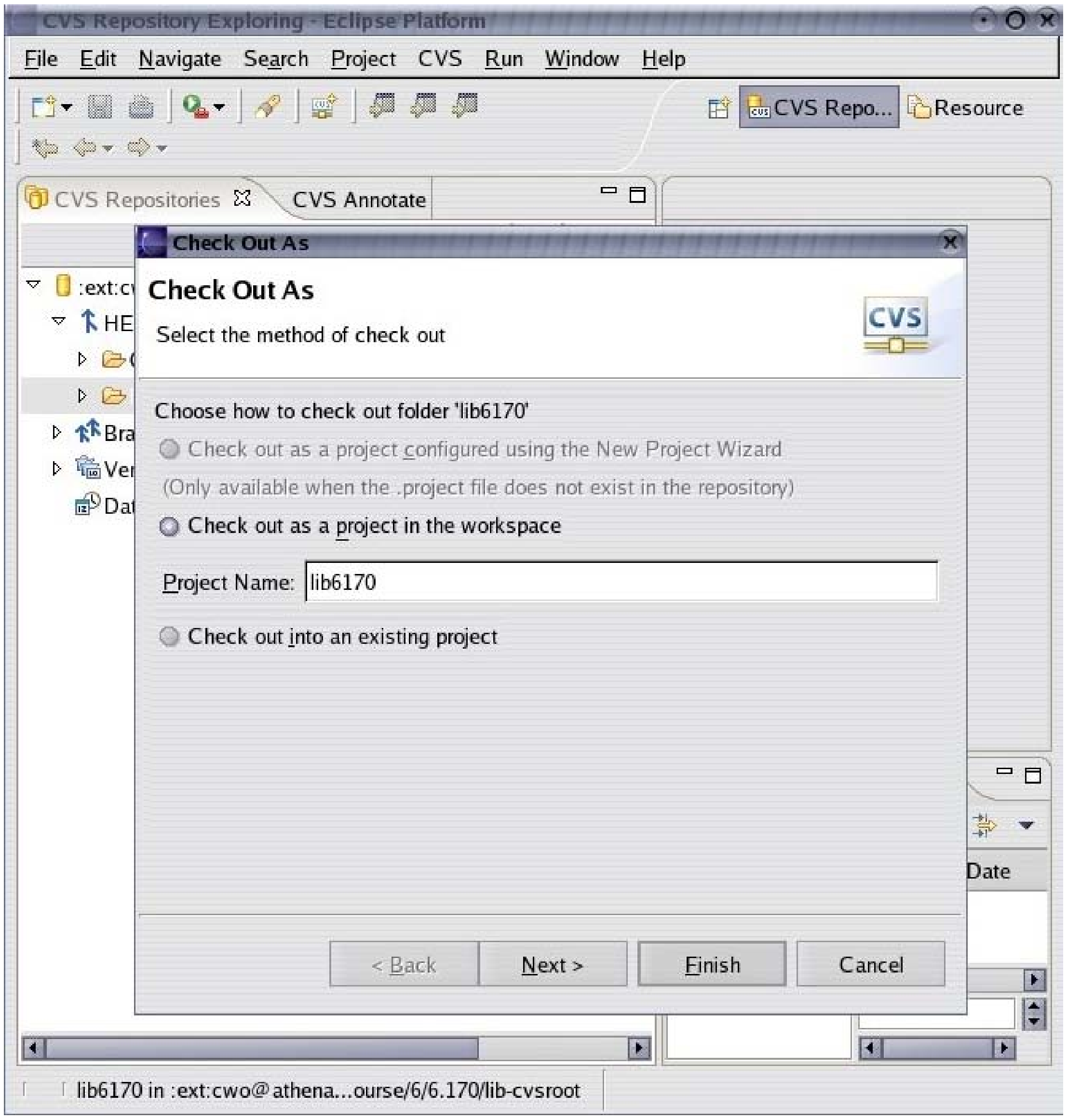
Courtesy of Free Software Foundation.

- Expand the CVS Repository directory structure, in the CVS Repositories Perspective Window. Expand **HEAD**, so that `lib6170' is visible.

- Right-click "lib6170," and select **Check Out As...**


CVS Repository Exploring - Eclipse Platform
File Edit Navigate Search Project CVS Run Window Help
CVS Repo... Resource
CVS Repositories CVS Annotate
:ext:cwo@athena.dialup.mit.edu:/afs/athena.mit.edu/course/6/
HEAD
CVSROOT
lib6170
Branches
Versions
Dates
New
Check Out
Check Out As...
Tag as Version...
Tag with Existing...
Compare With...
Compare
Add to Branch List...
Configure Branches and Versions...
Refresh View
CVS Resource History
Revision
Tags
Date
lib6170 in :ext:cwo@athena...ourse/6/6.170/lib-cvsroot
Courtesy of Free Software Foundation.

- You should see the option "Check out as project in the workspace. Project name: lib6170". Click **Finish** to continue.


CVS Repository Exploring - Eclipse Platform
File Edit Navigate Search Project CVS Run Window Help
CVS Repo...
Resource
CVS Repositories
CVS Annotate
:ext:c
HE
Bra
Ver
Dat
Date
< Back
Next >
Finish
Cancel

Check Out As

Select the method of check out

Choose how to check out folder 'lib6170'

Check out as a project configured using the New Project Wizard

(Only available when the .project file does not exist in the repository)

[✓ Check out as a project in the workspace]

Project Name: lib6170

Check out into an existing project

lib6170 in :ext:cwo@athena...ourse/6/6.170/lib-cvsroot

Courtesy of Free Software Foundation.

More Practice with Checking out Modules from CVS

Next, you must check out the files that you need for this problem set. These files have already been uploaded into your CVS repository which resides in `~/6.170/cvsroot`. Using the same steps as you did for `lib6170' above, add a new repository location using the following information: (**Note:** You need to specify a DIFFERENT repository location.)

Host: **athena.dialup.mit.edu**
Repository path: **/mit/<*your username*>/6.170/cvsroot**
User: *your username*
Password: *leave this field blank!*
Connection type: **ext**

Once this repository has been added, click on the second **ext:<username>** icon and expand **HEAD**. Right click on **ps0** and choose **Check Out As**. When asked for a name, type **ps0**. This module contains all the source files required for this problem.

Using CVS From Other Environments

We encourage you to try Eclipse for this problem set. In the future, if you prefer to use Emacs on the Athena, we also provide instructions for retrieving problem sets from the command line.

- Checking out from CVS

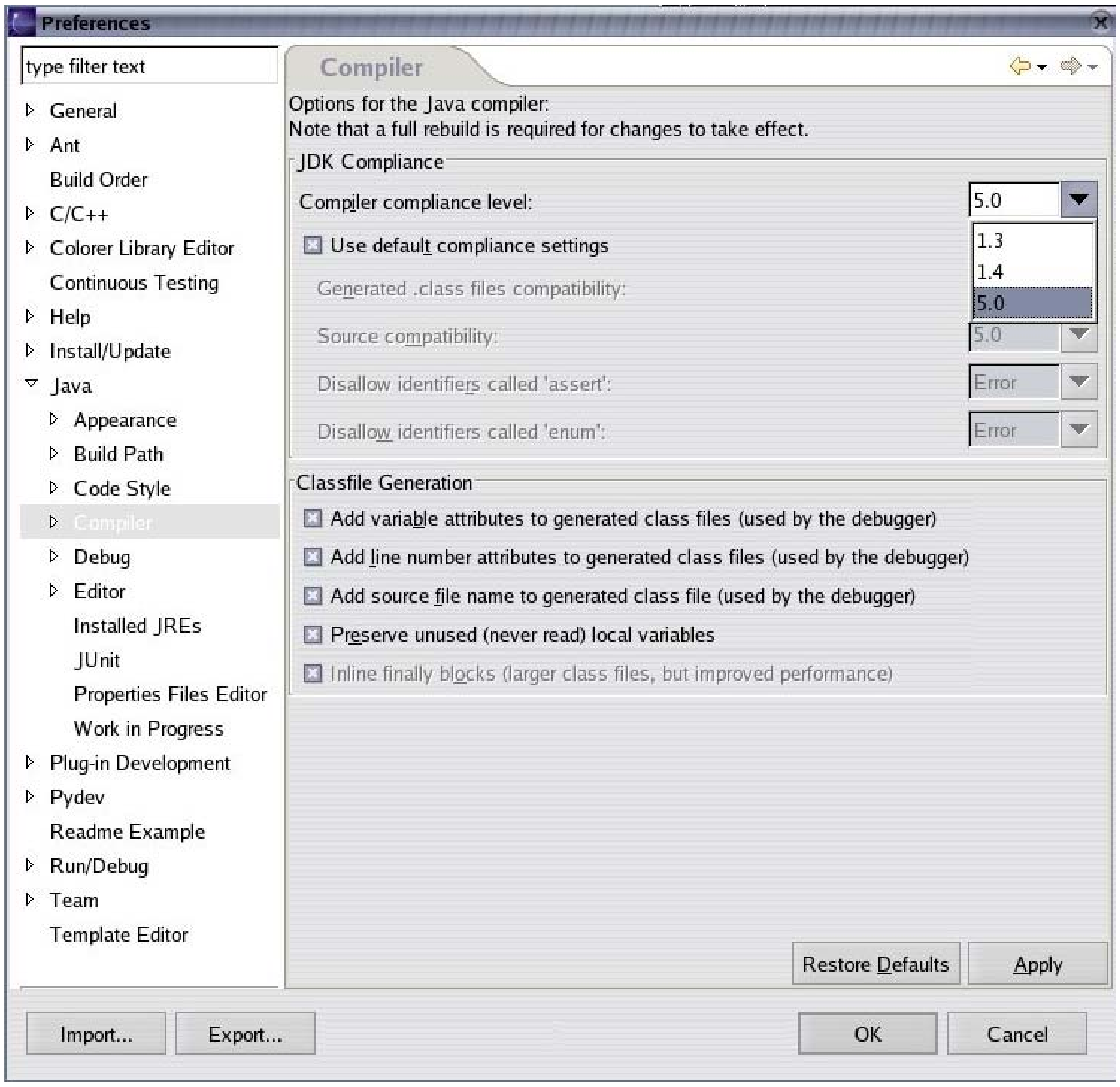
Eclipse is particularly well-suited for working on a home computer running Windows or Linux. There is also a version for MacOS X but Java 1.5 is only availabe on MacOS X.4 Tiger. If you prefer to work at home, here are instructions for retrieving problem sets with Eclipse and CVS:

- Using Eclipse At Home

Editing Java Code in Eclipse

Before we begin, we must first set Eclipse to use Java 5 by default (which is not the current case).

- Select from the top menu **Window >> Preferences**
- In the left pane, expand the heading **Java** and click on the heading **Compiler**
- Across from where it says **Compiler Compliance Level** select **5.0** and click **OK**
- A window may appear that says a **A full rebuild is required ... Do the full build now?** Simply click **No** since no project has been built yet.



Compiler

Options for the Java compiler:
Note that a full rebuild is required for changes to take effect.

JDK Compliance

Compiler compliance level: 5.0 (1.3 / 1.4 / 5.0)

[✓] Use default compliance settings

Generated .class files compatibility:

Source compatibility: 5.0

Disallow identifiers called 'assert': Error

Disallow identifiers called 'enum': Error

Classfile Generation

[✓] Add variable attributes to generated class files (used by the debugger)

[✓] Add line number attributes to generated class files (used by the debugger)

[✓] Add source file name to generated class file (used by the debugger)

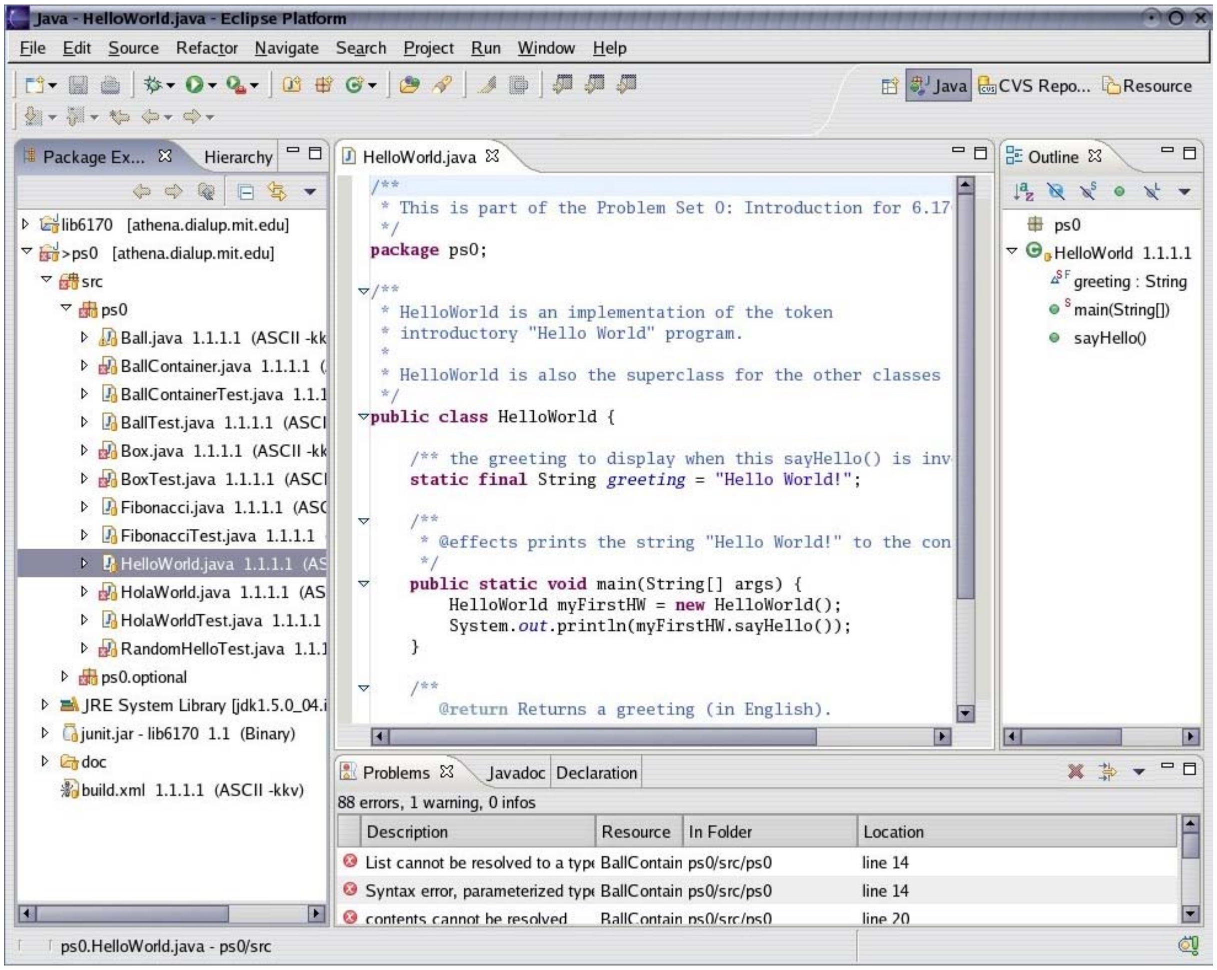
[✓] Preserve unused (never read) local variables

[✓] Inline finally blocks (larger class files, but improved performance)

Courtesy of The Eclipse Foundation.

Next, we will edit and compile Java files with Eclipse. After you have completed the problem set, you will turn in your answers by committing `ps0` back into your repository and running the validate script.

- Select from the top menu **Window** >> **Open Perspective** >> **Other...** >> **Java**
- Explore the directory structure in the Package Explorer window to find the Java source code. The code is available in *ps0/src/ps0/*. Double-click on *HelloWorld.java* to view its source code.


Java - HelloWorld.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Java
CVS Repo...
Resource
Package Ex...
Hierarchy
lib6170 [athena.dialup.mit.edu]
>ps0 [athena.dialup.mit.edu]
src
ps0
Ball.java 1.1.1.1 (ASCII -kk
BallContainer.java 1.1.1.1 (
BallContainerTest.java 1.1.1
BallTest.java 1.1.1.1 (ASCI
Box.java 1.1.1.1 (ASCII -kk
BoxTest.java 1.1.1.1 (ASCI
Fibonacci.java 1.1.1.1 (ASC
FibonacciTest.java 1.1.1.1
HelloWorld.java 1.1.1.1 (AS
HolaWorld.java 1.1.1.1 (AS
HolaWorldTest.java 1.1.1.1
RandomHelloTest.java 1.1.1
ps0.optional
JRE System Library [jdk1.5.0_04.i
junit.jar - lib6170 1.1 (Binary)
doc
build.xml 1.1.1.1 (ASCII -kkv)
HelloWorld.java
Outline
ps0
HelloWorld 1.1.1.1
greeting : String
main(String[])
sayHello()
Problems
Javadoc
Declaration
ps0.HelloWorld.java - ps0/src

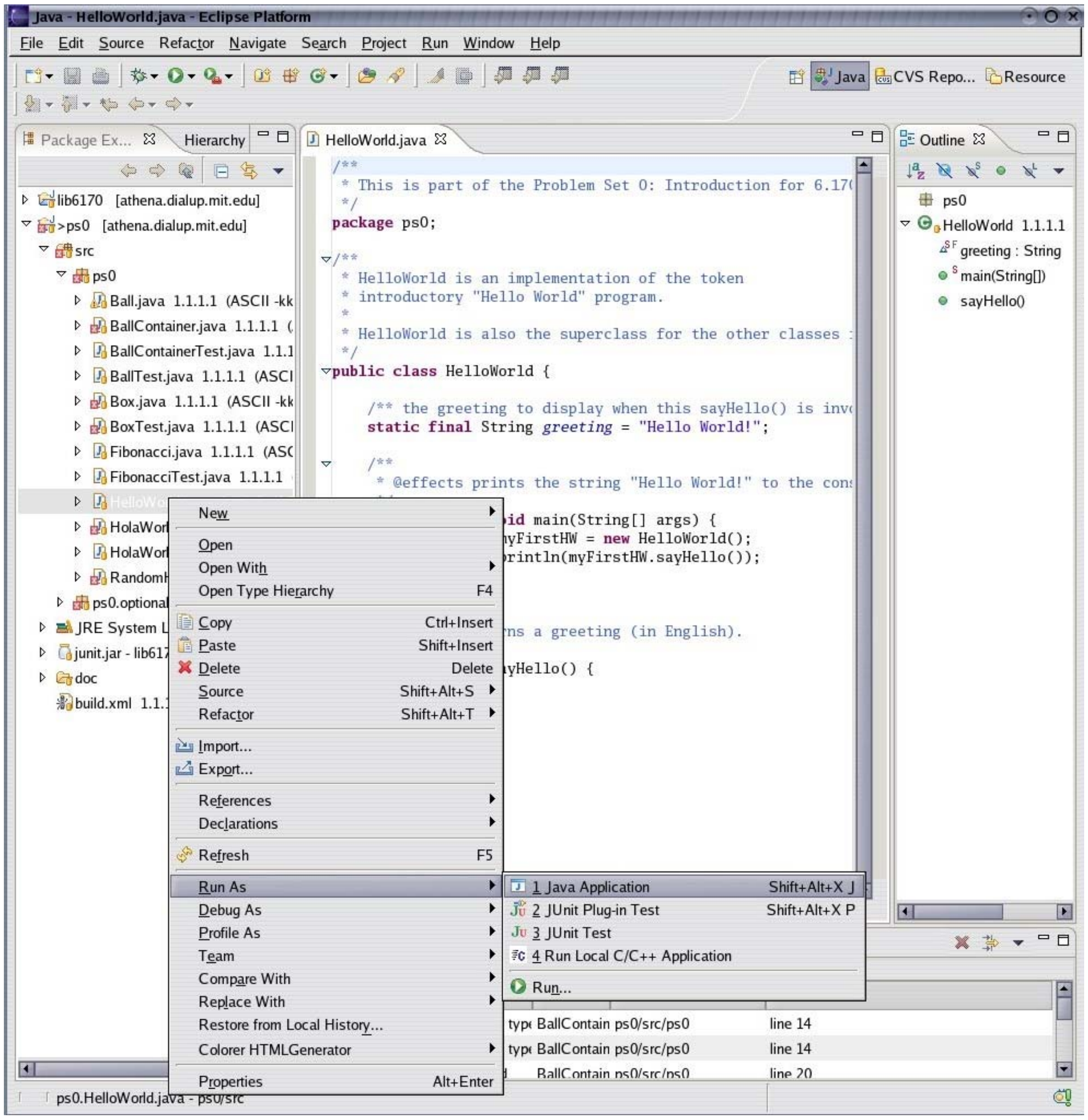
```
/**
 * This is part of the Problem Set 0: Introduction for 6.17
 */
package ps0;

/**
 * HelloWorld is an implementation of the token
 * introductory "Hello World" program.
 *
 * HelloWorld is also the superclass for the other classes
 */
public class HelloWorld {

    /** the greeting to display when this sayHello() is inv
    static final String greeting = "Hello World!";

    /**
     * @effects prints the string "Hello World!" to the con
     */
    public static void main(String[] args) {
        HelloWorld myFirstHW = new HelloWorld();
        System.out.println(myFirstHW.sayHello());
    }

    /**
      @return Returns a greeting (in English).
```

88 errors, 1 warning, 0 infos

Description	Resource	In Folder	Location
List cannot be resolved to a type	BallContain	ps0/src/ps0	line 14
Syntax error, parameterized type	BallContain	ps0/src/ps0	line 14
contents cannot be resolved	BallContain	ps0/src/ps0	line 20

Courtesy of The Eclipse Foundation.

- Now you will run *HelloWorld*. To do this, right click on `HelloWorld.java' in the Package Explorer. Then select **Run As >> Java Application.** Note that you will see a message saying *"Errors exist in a required project. Continue launch?"*. You can click **Ok** and ignore this message for now. There are intentional errors in the source code for this problem set which you will deal with in later problems. The output `Hello World!` should appear in the `Console' window. There is nothing to turn in for this problem.


Java - HelloWorld.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Java
CVS Repo...
Resource
Package Ex...
Hierarchy
HelloWorld.java
Outline
lib6170 [athena.dialup.mit.edu]
>ps0 [athena.dialup.mit.edu]
src
ps0
Ball.java 1.1.1.1 (ASCII -kk
BallContainer.java 1.1.1.1 (
BallContainerTest.java 1.1.1
BallTest.java 1.1.1.1 (ASCI
Box.java 1.1.1.1 (ASCII -kk
BoxTest.java 1.1.1.1 (ASCI
Fibonacci.java 1.1.1.1 (ASC
FibonacciTest.java 1.1.1.1
HelloWo
HolaWor
HolaWor
Random
ps0.optional
JRE System L
junit.jar - lib617
doc
build.xml 1.1.1
ps0
HelloWorld 1.1.1.1
greeting : String
main(String[])
sayHello()
New
Open
Open With
Open Type Hierarchy F4
Copy Ctrl+Insert
Paste Shift+Insert
Delete Delete
Source Shift+Alt+S
Refactor Shift+Alt+T
Import...
Export...
References
Declarations
Refresh F5
Run As
Debug As
Profile As
Team
Compare With
Replace With
Restore from Local History...
Colorer HTMLGenerator
Properties Alt+Enter
1 Java Application Shift+Alt+X J
2 JUnit Plug-in Test Shift+Alt+X P
3 JUnit Test
4 Run Local C/C++ Application
Run...
ps0.HelloWorld.java - ps0/src

```
/**
 * This is part of the Problem Set 0: Introduction for 6.170
 */
package ps0;

/**
 * HelloWorld is an implementation of the token
 * introductory "Hello World" program.
 *
 * HelloWorld is also the superclass for the other classes :
 */
public class HelloWorld {

    /** the greeting to display when this sayHello() is invo
    static final String greeting = "Hello World!";

    /**
     * @effects prints the string "Hello World!" to the cons
                  id main(String[] args) {
                  yFirstHW = new HelloWorld();
                  rintln(myFirstHW.sayHello());

                  ns a greeting (in English).

                  yHello() {
```

```
typ BallContain ps0/src/ps0    line 14
typ BallContain ps0/src/ps0    line 14
    BallContain ps0/src/ps0    line 20
```

Courtesy of The Eclipse Foundation.

Problem 3: Working with Java in Eclipse

`HolaWorld` has errors and does not compile. Double-click `HolaWorld.java` in Package Explorer to find these simple errors and fix them so that `HolaWorld` builds and runs correctly.

Notice that the sections that cause compilation problems are underlined in red. There is also a red X on the file icon for `HolaWorld.java` in Package Explorer, as well as a red rectangles to the right of the scrollbar in the Java editor to indicate where you need to scroll to find the compilation errors in the file (this is more helpful in longer Java files). Use these visual aides to help find the compilation errors in your code. (You can also find a list of all of the compilation errors in your project by looking at the Problems tab at the bottom of the window. If the Problems tab is not visible, then you can open it again by doing **Window >> Show View >> Problems**.)

Once you have found a compilation error, you can move the mouse over it and a description of the error will pop up as a tooltip. You can also mouse over either of the red icons in the margin of the Java editor to get this information.

Once you have fixed the broken `HolaWorld.java` file, use the steps you followed in Problem 2 to run it. If you ran `HolaWorld` successfully, you will see both English and Spanish greetings in the Console window. For this problem, you need to turn in a version of `HolaWorld.java` that compiles without errors and prints the correct (Spanish) greeting. Instructions for turning in files will be given at the end of this problem set.

Eclipse Tip:Autocompletion

Using autocompletion will reduce the amount of typing that you have to do as well as the number of spelling mistakes, thereby increasing your efficiency.

In Eclipse, **CTRL+Space** can be used to autocomplete most things inside the Java editor. For example, place the cursor at the right of `spanishGree` and then press **CTRL+Space.** You should see `spanishGree` expand to `spanishGreeting`. Then press **CTRL+S** to save the file. You will see that the compilation error disappears upon saving your changes.

CTRL+Space can also help you autocomplete method names. Again, place the cursor to the right of `world.` on line 26. Eclipse knows that `world` is of type `HolaWorld`, so when you press **CTRL+Space** after the period, it pops up a list of methods that `HelloWorld` has. Since we want to use the method `sayHello()`, type **s** to start writing the method name as you normally would. Now the list of available methods has been reduced to one item because `sayHello()` is the only method in `HelloWorld` that starts with **s**. Now with `sayHello()` as the only option in the list, you can press **Enter** to complete the method name.

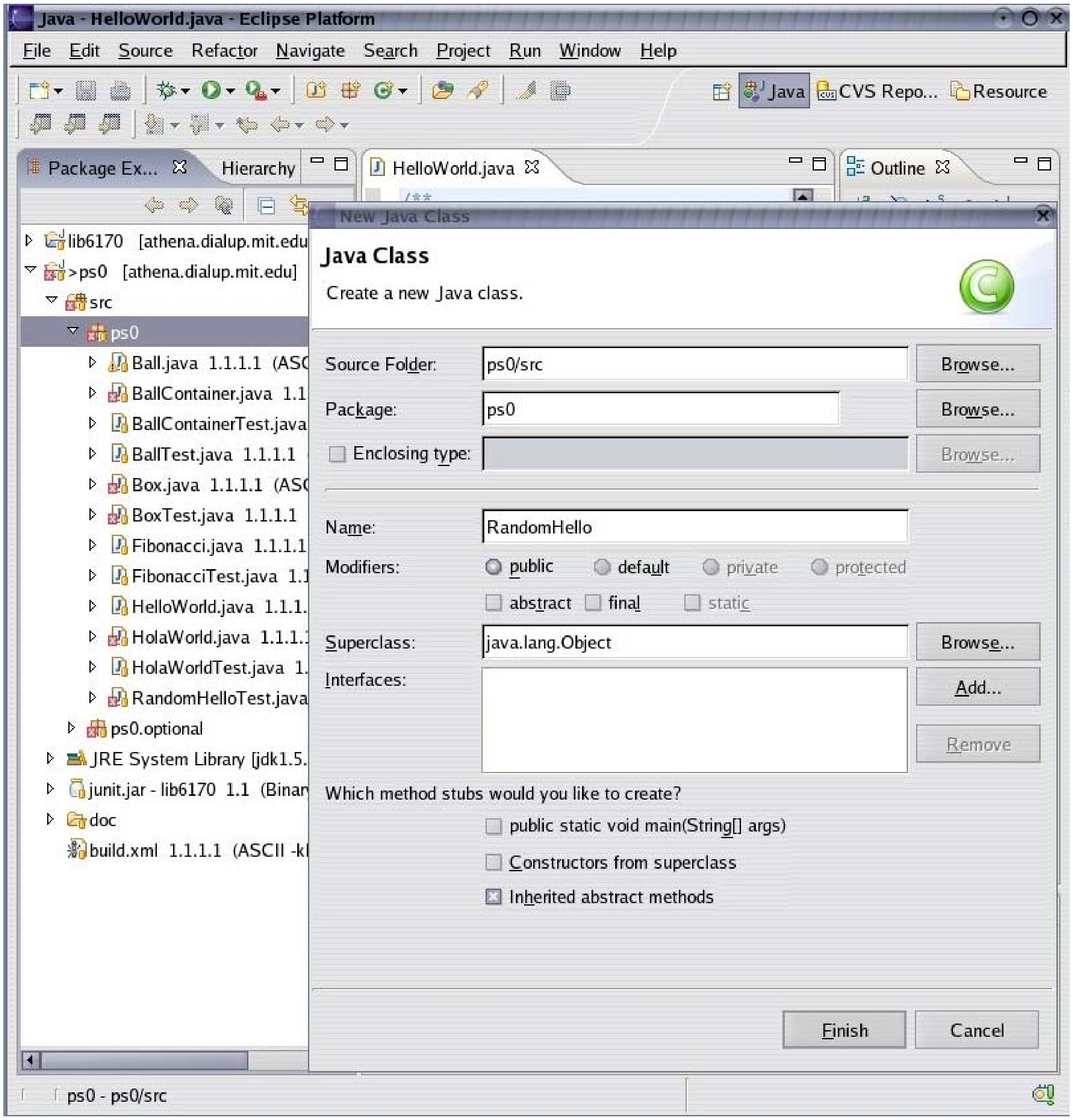
If you are working in Emacs, you should check out the Emacs autocompletion features in the assignments section.

Problem 4: Warm-Up Exercise -- RandomHello

For this problem, you will create your first Java class. The class's `main` method will randomly choose, and then print to the console, one of five possible greetings. The text of the 5 possible greetings is up to you.

First create a new class. Select from the top menu **File >> New >> Class**. A window will pop up, asking you details about the class. Use `ps0/src` as the **Source Folder**. Your class should be in the `ps0` package, and you should call it `RandomHello.` Type this name in the "Name" field and click **Finish**.

In order to run `RandomHello`, you need to give it a `main` method whose signature is `public static void main(String[] argv)` (which is identical to the signature of the `main` method in `HelloWorld` and `HolaWorld`). When creating `RandomHello`, you can check the appropriate checkbox and have Eclipse create a stub of the main method for you.


Java - HelloWorld.java - Eclipse Platform

New Java Class

Java Class

Create a new Java class.

Source Folder: ps0/src

Package: ps0

Enclosing type: [ ]

Name: RandomHello

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

Courtesy of The Eclipse Foundation.

You may use the following as a skeleton for your class `RandomHello.java`:

```
package ps0;

/**
 * RandomHello selects a random greeting to display to the user.
 */
public class RandomHello {

        /**
         * @effects uses a RandomHello object to print
         * a random greeting to the console
         */
        public static void main(String[] argv) {

                RandomHello randomHello = new RandomHello();
                 System.out.println(randomHello.sayHello());
        }

        /**
         * @return a random greeting from a list of five different
greetings.
         */
        public String sayHello() {
                // YOUR CODE GOES HERE
        }
}
```

Please note that this skeleton is meant only to serve as a starting point for students who are not familiar with Java. If you have some Java programming experience, you are free to organize `RandomHello` as you see fit.

No need to reinvent the Wheel

Instead of writing your own random number generator to decide which greeting will be written to the console, you should take advantage of Java's Random class. (This is a good example of the adage "Know and Use the Libraries" as described in Chapter 7 of Joshua Bloch's *Effective Java*).

Type the following into the body of your `sayHello()` method:

```
 Random randomGenerator = new Random();
```

This line creates a random number generator. In Eclipse, your code may be marked with a red underline, indicating an error. This is because the `Random` class lies in a package that has not yet been imported (`java.lang` and `ps0` are the only packages that are implicitly imported). Java libraries are organized as packages and you can only access Java classes in packages that are imported (this is similar to #include in C). To explicitly import `java.util.Random` and avoid the compilation error, you can add the following line under the line `package ps0;` at the top of your file (after the `package ps0;` declaration):

```
 import java.util.Random;
```

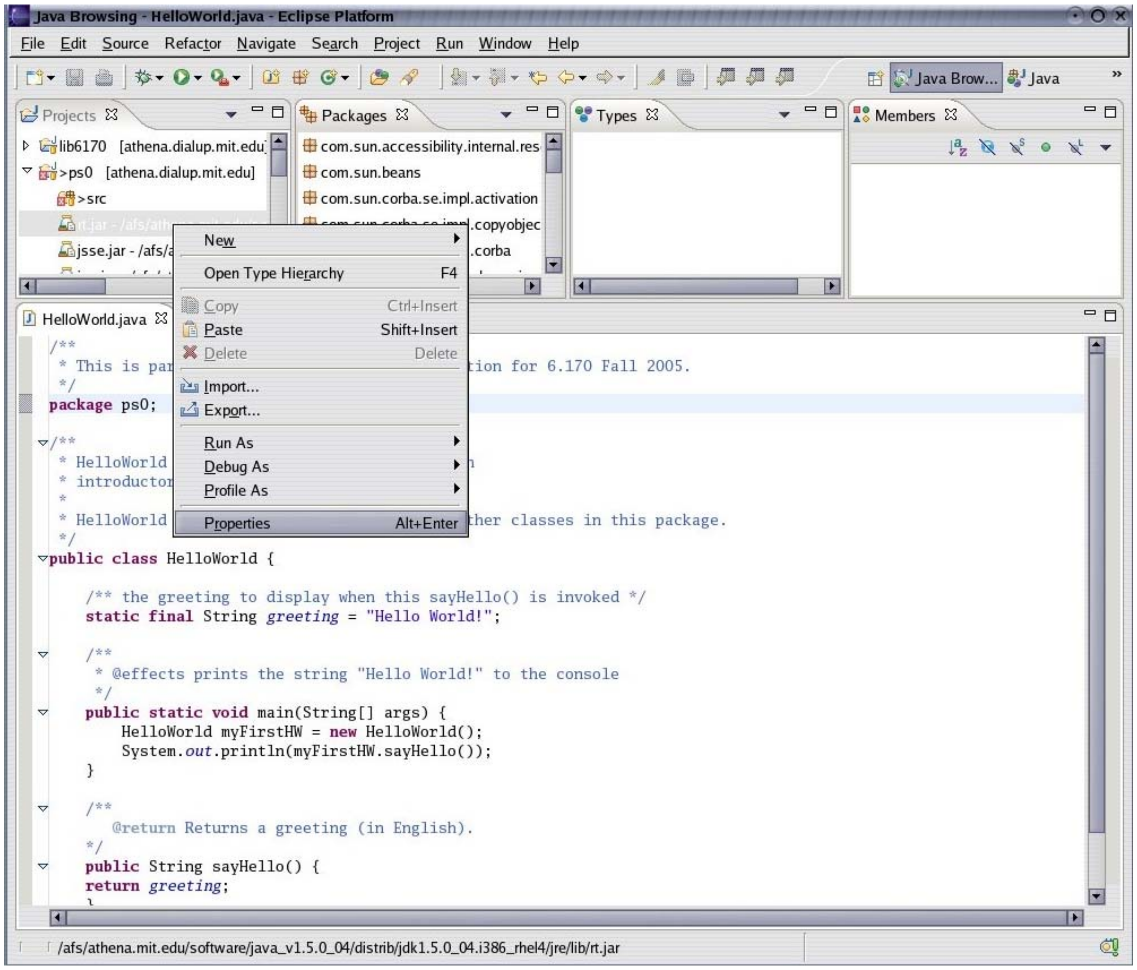
The above line will import the class `Random` into your file. If you are using Eclipse, you can also hit **CTRL-SHIFT-O** to *organize* your imports. Because there is only one class named `Random`, Eclipse can figure out that you mean to import `java.util.Random` and will add the above line of code automatically. (If the name of the class that needs to be imported is ambiguous – for example, there is a `java.util.List` as well as a `java.awt.List` – then Eclipse will prompt you to choose the one to import.)

Viewing documentation for source code

You can also use Eclipse to view the documentation for available Java classes, which is automatically generated from comments in Java source code using a tool called **javadoc**. You will learn about how to generate your own javadoc files later in the course. For now, we will show you how to view them in Eclipse.

The first tiem you do this, however, you need to tell Eclipse where to look for the external Java documentation files. (You will only ahve to do this once). To accomplish this do the following:

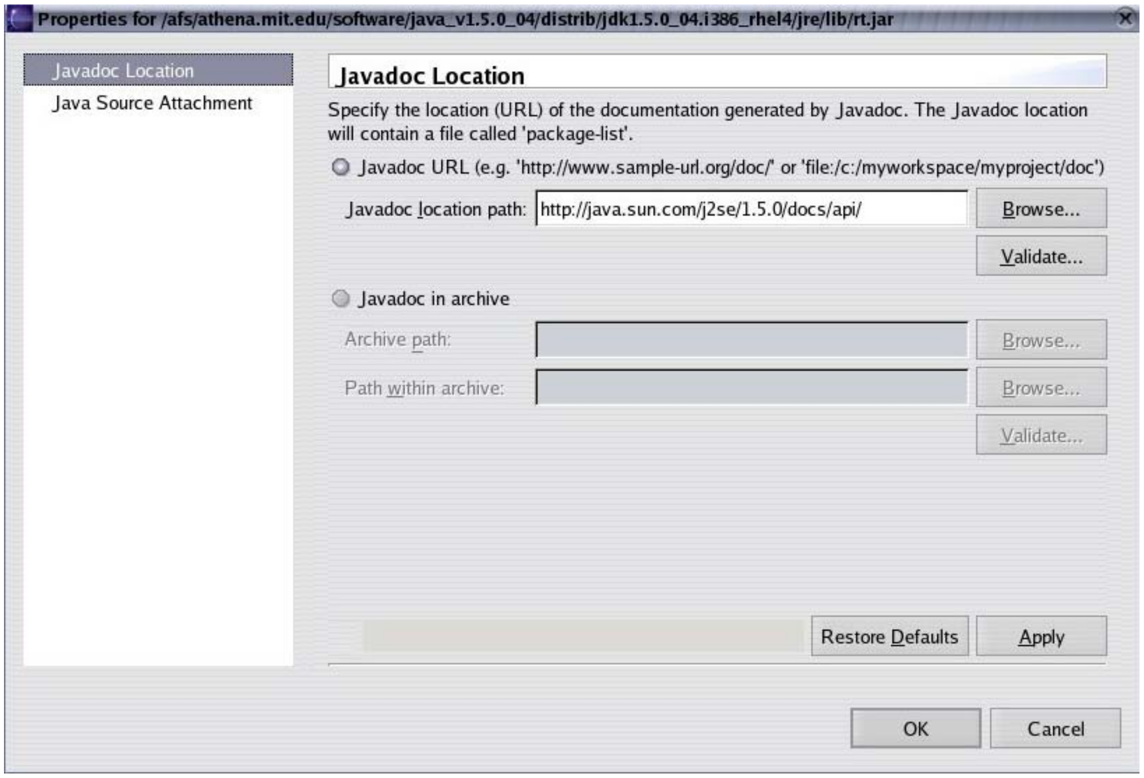
- Switch to the Java Browsing perspective, by choosing **Window >> Open Perspective >> Other... >> Java Browsing**.
- In the *Projects* window, expand `ps0` to find *ps0/rt.jar*. Right-click on `rt.jar`, and choose **Properties**.



```
/**
 * This is par                     tion for 6.170 Fall 2005.
 */
package ps0;

/**
 * HelloWorld                      n
 * introductor
 *
 * HelloWorld                      ther classes in this package.
 */
public class HelloWorld {

    /** the greeting to display when this sayHello() is invoked */
    static final String greeting = "Hello World!";

    /**
     * @effects prints the string "Hello World!" to the console
     */
    public static void main(String[] args) {
        HelloWorld myFirstHW = new HelloWorld();
        System.out.println(myFirstHW.sayHello());
    }

    /**
       @return Returns a greeting (in English).
    */
    public String sayHello() {
    return greeting;
```

/afs/athena.mit.edu/software/java_v1.5.0_04/distrib/jdk1.5.0_04.i386_rhel4/jre/lib/rt.jar

Courtesy of The Eclipse Foundation.


Properties for /afs/athena.mit.edu/software/java_v1.5.0_04/distrib/jdk1.5.0_04.i386_rhel4/jre/lib/rt.jar

Javadoc Location

Specify the location (URL) of the documentation generated by Javadoc. The Javadoc location will contain a file called 'package-list'.

Javadoc URL (e.g. 'http://www.sample-url.org/doc/' or 'file:/c:/myworkspace/myproject/doc')

Javadoc location path: http://java.sun.com/j2se/1.5.0/docs/api/

Javadoc in archive

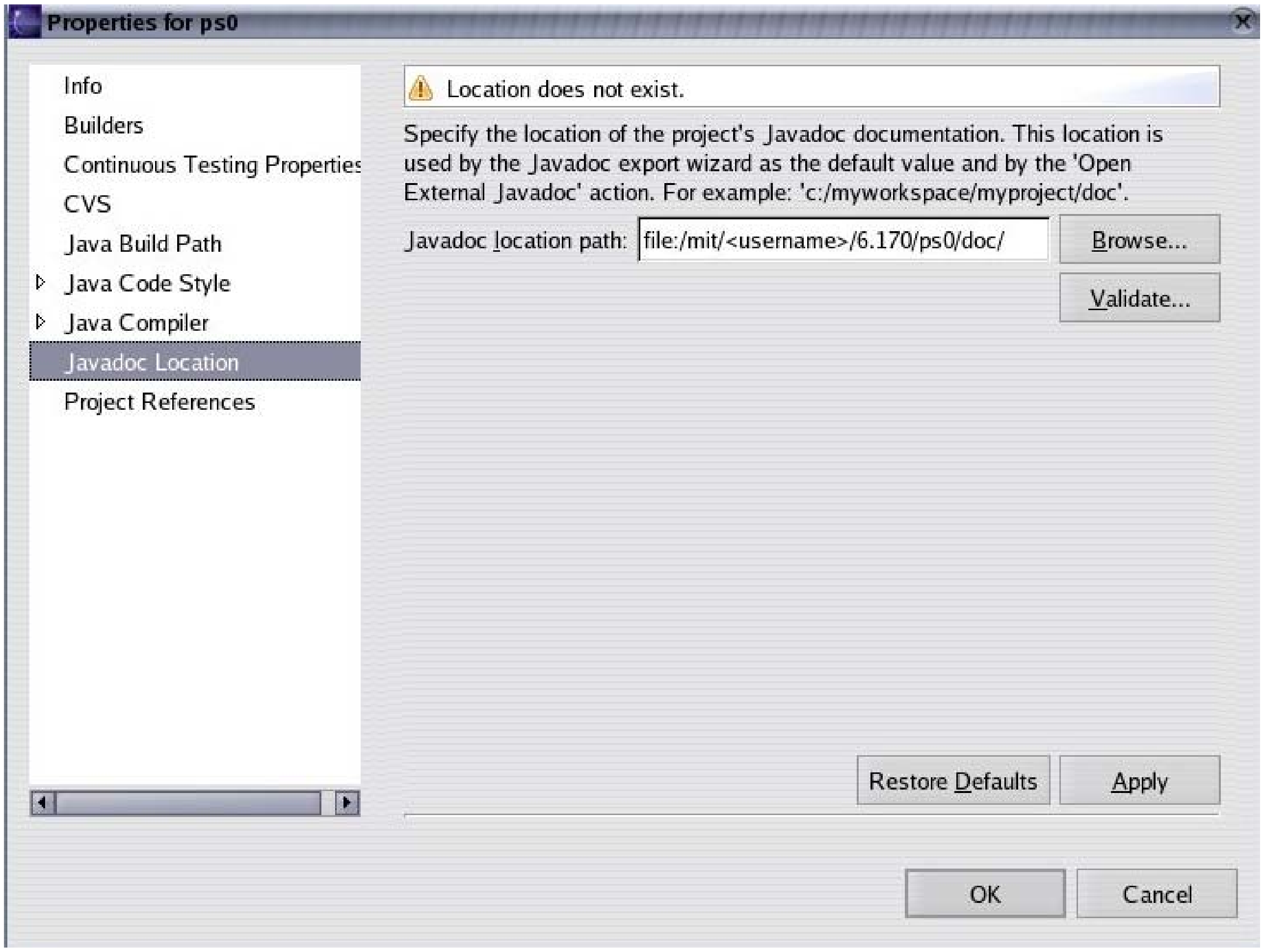
Archive path:

Path within archive:

Courtesy of The Eclipse Foundation.

- For the *Javadoc location*, enter **http://java.sun.com/j2se/1.5.0/docs/api/**, and press **OK**. (This information may already appear, if so, just click **OK**.)
- Now return to the Java Perspective (**Window** >> **Open Perspective** >> **Java**), and go back to editing `RandomHello`. Place your cursor on `Random`, and press **SHIFT-F2**. A web browser window will appear, showing documentation for the `Random` class.

To configure Eclipse to recognize the Java documentation files for this problem set's source code, right click on the project name (**"ps0"**) in the Package Explorer pane and click **"Properties"**. Select **"Javadoc Location"** in the left pane. The location is **"file:/mit/<username>/6.170/ps0/doc/"**. (Note: the **"file:"** portion is important, since the location is expected to be recognizable by a web browser.) You can either type in this location or click **"Browse..."** to browse to the **"doc"** directory in the ps0 project. After setting the Javadoc location path, click **OK**.



Properties for ps0

Location does not exist.

Specify the location of the project's Javadoc documentation. This location is used by the Javadoc export wizard as the default value and by the 'Open External Javadoc' action. For example: 'c:/myworkspace/myproject/doc'.

Javadoc location path: file:/mit/<username>/6.170/ps0/doc/

Courtesy of The Eclipse Foundation.

It is now possible to generate Javadoc documentation for your source code by clicking on the **Project** pull-down menu at the top and choosing **Generate Javadoc**. This opens a window that allows you to customize what kind of documentation is generated. You can simply click on **Finish**.

If you are working in Emacs, you can find instructions on how you can view Java documentation within Emacs in the assignments section.

Using `java.util.Random`

Find the documentation for `nextInt(int n)` and read it. You don't have to understand all the details of its behavior specification, only that it returns a random number from 0 to n-1. You should use this method to choose your greeting.

One way to choose a random greeting is using an array. This approach might look something like:

```
String[] greetings = new String[5];
greetings[0] = "Hello World";
greetings[1] = "Hola Mundo";
greetings[2] = "Bonjour Monde";
greetings[3] = "Hallo Welt";
greetings[4] = "Ciao Mondo";
```

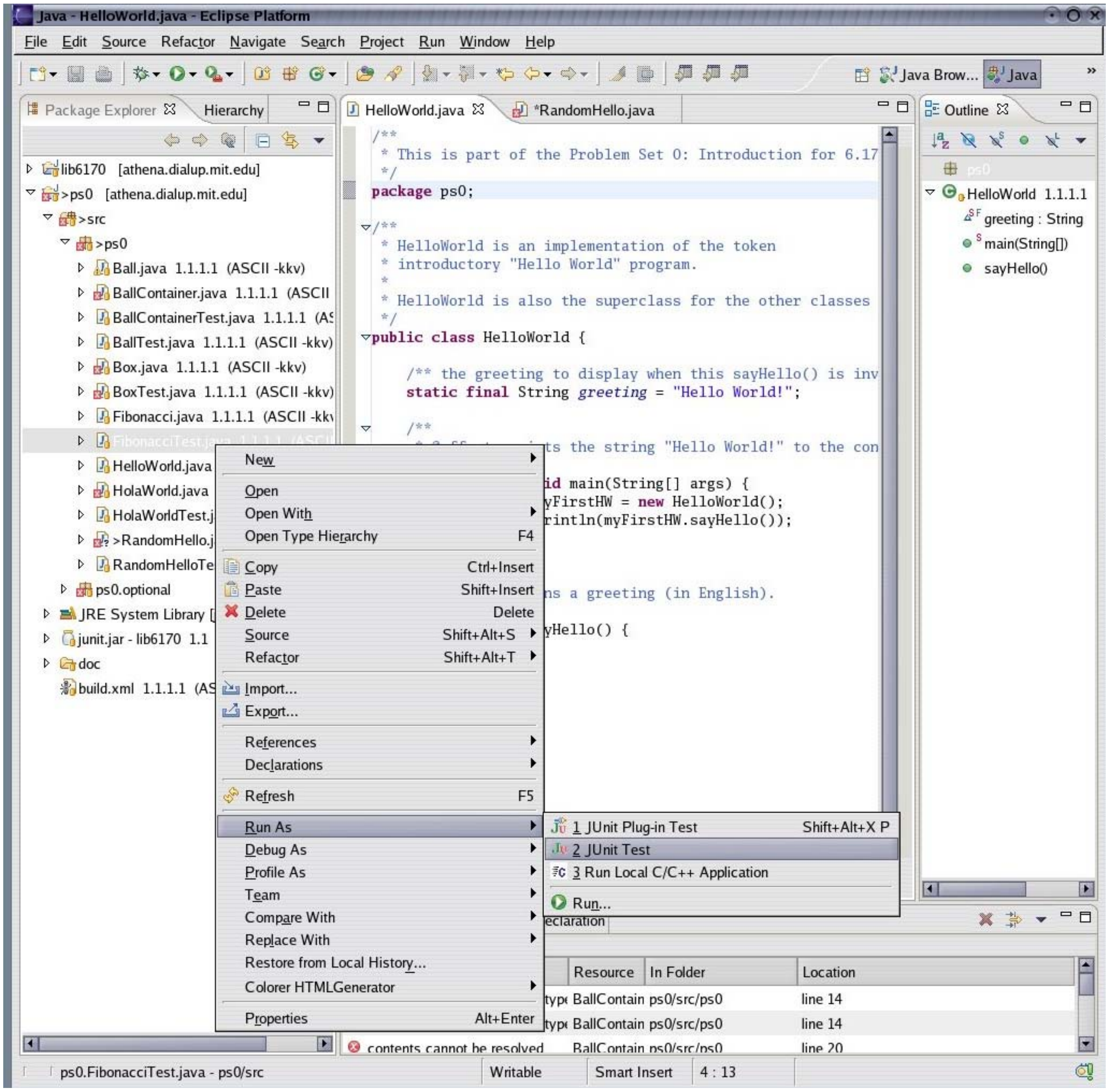
Looking at the `main` method, the System.out.println() method prints a String to the console. In this case the String is provided by the method `sayHello()` which returns one of five Strings chosen randomly from the `greetings[]` array. See `HelloWorld.sayHello()` for a simple example of how to return a String.

If you haven't used Java before, or are having trouble with the language or syntax, a good reference is The Java Tutorial from Sun's website. When you are finished writing your code and it compiles, run it several times to ensure that all 5 greetings can be displayed. Congratulations – you have written and compiled your first Java class!

Problem 5: Testing Java Code with JUnit

Part of your job as a software engineer is to verify that the software you produce works according to its specification. One form of verification is testing. JUnit is a framework for creating *unit* tests in Java. A unit test is a test for verifying that a specific method in a class conforms to its specifications. You will learn more about unit testing in the next problem set. In this problem, we will provide you with a quick overview of how JUnit works with a simple example.

Open both `Fibonacci.java` and `FibonacciTest.java`. From the comments, you can see that `FibonacciTest` is a test of the `Fibonacci` class. In Package Explorer, right-click on `FibonacciTest.java` and select **Run As >> JUnit Test**.

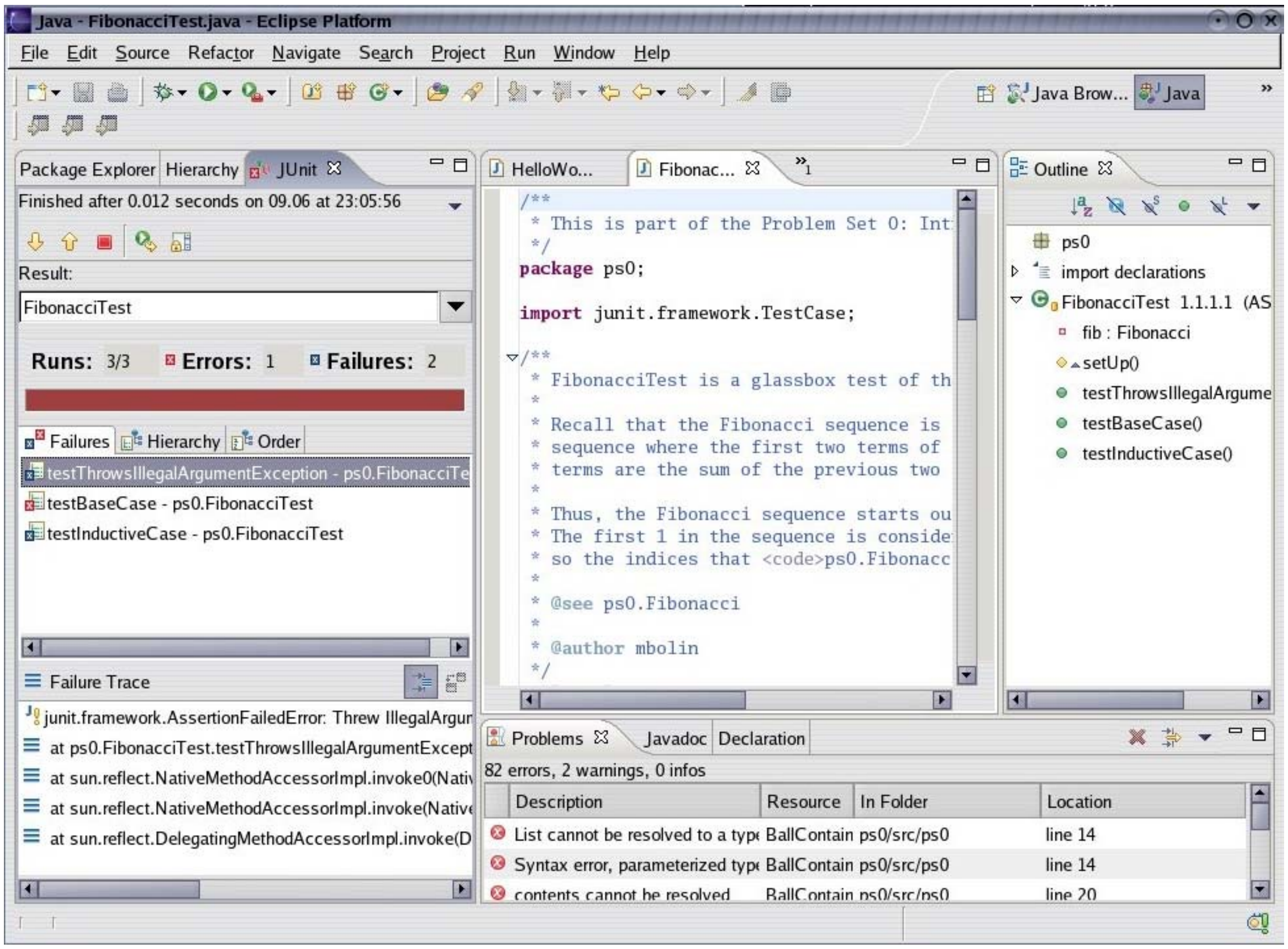


```
/**
 * This is part of the Problem Set 0: Introduction for 6.17
 */
package ps0;

/**
 * HelloWorld is an implementation of the token
 * introductory "Hello World" program.
 *
 * HelloWorld is also the superclass for the other classes
 */
public class HelloWorld {

    /** the greeting to display when this sayHello() is inv
    static final String greeting = "Hello World!";

    /**
                   ts the string "Hello World!" to the con
                   id main(String[] args) {
                   yFirstHW = new HelloWorld();
                   rintln(myFirstHW.sayHello());

                   ns a greeting (in English).

                   yHello() {
```

```
typ BallContain ps0/src/ps0   line 14
typ BallContain ps0/src/ps0   line 14
contents cannot be resolved   BallContain ps0/src/ps0   line 20
```

Courtesy of The Eclipse Foundation.

A window with a menacing red bar will appear, indicating that not all of the tests in `FibonacciTest` completed successfully. The top pane displays the list of tests that failed, while the bottom pane shows the Failure Trace for the highlighted. The first line in the Failure Trace should display an error message that explains why the test failed (it is the responsibility of the author of the test code to produce this error message).


Java - FibonacciTest.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Java Brow... Java
Package Explorer Hierarchy JUnit
HelloWo... Fibonac...
Outline
Result:
Failures Hierarchy Order
Failure Trace
Problems Javadoc Declaration

Finished after 0.012 seconds on 09.06 at 23:05:56

FibonacciTest

Runs: 3/3 Errors: 1 Failures: 2

testThrowsIllegalArgumentException - ps0.FibonacciTe
testBaseCase - ps0.FibonacciTest
testInductiveCase - ps0.FibonacciTest

```
junit.framework.AssertionFailedError: Threw IllegalArgur
at ps0.FibonacciTest.testThrowsIllegalArgumentExcept
at sun.reflect.NativeMethodAccessorImpl.invoke0(Nati
at sun.reflect.NativeMethodAccessorImpl.invoke(Native
at sun.reflect.DelegatingMethodAccessorImpl.invoke(D
```

```
/**
 * This is part of the Problem Set 0: Int
 */
package ps0;

import junit.framework.TestCase;

/**
 * FibonacciTest is a glassbox test of th
 *
 * Recall that the Fibonacci sequence is
 * sequence where the first two terms of
 * terms are the sum of the previous two
 *
 * Thus, the Fibonacci sequence starts ou
 * The first 1 in the sequence is conside
 * so the indices that <code>ps0.Fibonacc
 *
 * @see ps0.Fibonacci
 *
 * @author mbolin
 */
```

ps0
import declarations
FibonacciTest 1.1.1.1 (AS
fib : Fibonacci
setUp()
testThrowsIllegalArgume
testBaseCase()
testInductiveCase()

82 errors, 2 warnings, 0 infos

Description	Resource	In Folder	Location
List cannot be resolved to a typ	BallContain	ps0/src/ps0	line 14
Syntax error, parameterized typ	BallContain	ps0/src/ps0	line 14
contents cannot be resolved	BallContain	ps0/src/ps0	line 20

Courtesy of The Eclipse Foundation.

As shown in the figure above, if you click on the first failure `testThrowsIllegalArgumentException()`, the bottom pane will automatically switch to the appropriate error message. You can see from the first line of the failure trace that `Fibonacci.java` threw an `IllegalArgumentException` for the argument zero, when it shouldn't have thrown any exception (you will have to scroll the pane to the right to see this). Once you've figured out the problem in `Fibonacci.java` that caused this error and fixed it, you can rerun the JUnit test by clicking on the green icon with the arrow pointing to the right in the JUnit pane (just above "Result").

Use the information in the Failure Trace box to help you debug `Fibonacci` and keep a record of what you did to debug `Fibonacci` as you will have to answer questions about your debugging experience in the next problem. After you have fixed all the problems in `Fibonacci`, you should see a bright green bar instead of a red one when you run `FibonacciTest`.

Note: Now that you have a few files open, try holding down **Ctrl** and hitting **F6** to bring up a dropdown box of open files. You can use the arrow keys to select the file whose editor you wish to bring into focus.

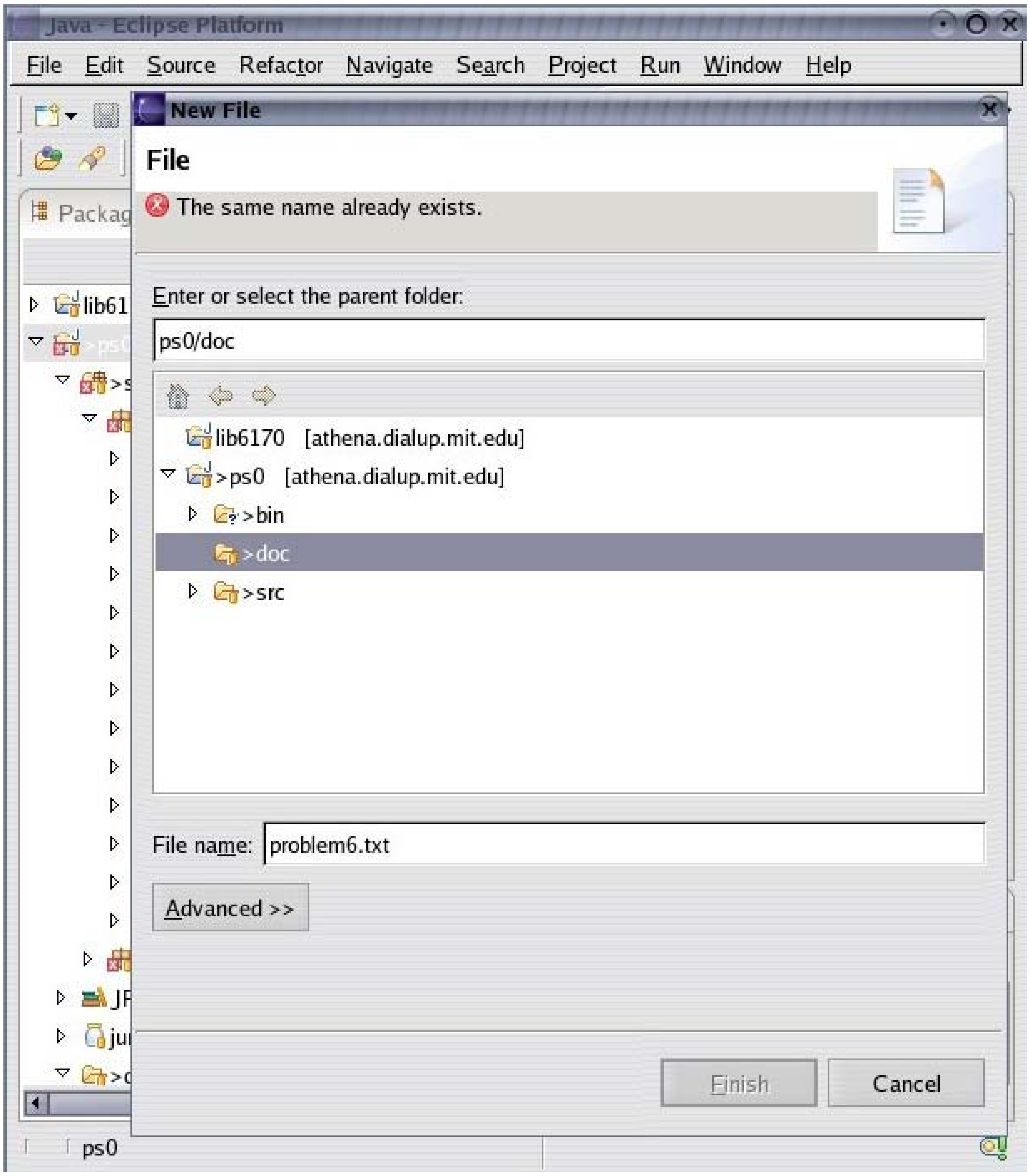
Now that you have learned to run JUnit, you should also check out the JUnit tests that we wrote for `HolaWorld` and `RandomHello` in Problems 3 and 4 earlier. They are called `HolaWorldTest.java` and `RandomHelloTest.java` respectively. Please ensure that your modified code passes these tests before you turn in your problem set.

Problem 6: Answering Questions About the Code

In this problem, you will create a text file in your Eclipse project and write the answers to some questions in it. Most design projects that you will be assigned will require you to submit some sort of response or write-up in addition to your code. Follow these instructions to create a text file with answers to the following questions:

1. Why did Fibonacci fail the **testThrowsIllegalArgumentException** test? What (if anything) did you have to do to fix it?
2. Why did Fibonacci fail the **testBaseCase** test? What (if anything) did you have to do to fix it?
3. Why did Fibonacci fail the **testInductiveCase** test? What (if anything) did you have to do to fix it?

To create the text file, select from the top menu **File >> New >> File**. A window will pop up, asking for the name and location of the file. Select `ps0/doc` as the **parent folder**, and name the file `problem6.txt`. (All documentation files should go in the project's `doc/` directory, unless specified otherwise.)


Java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
New File
File
Packag
lib61
Enter or select the parent folder:
lib6170 [athena.dialup.mit.edu]
>ps0 [athena.dialup.mit.edu]
>bin
>doc
>src
Advanced >>
Finish
Cancel
ps0

The same name already exists.

Enter or select the parent folder: ps0/doc

File name: problem6.txt

Courtesy of The Eclipse Foundation.

Problem 7: Getting a Real Taste of Java -- Balls and Boxes

Until now, we have only been introducing tools. In this problem, we will delve into a real programming exercise. If you are not familiar with Java, we recommend walking through Sun's Java Tutorial.

The intention of this problem is to give you a sense of what Java programming entails and to demonstrate the use of Eclipse and the JUnit testing tool. If you have never programmed in Java. you may find this problem somewhat challenging. Do not be discouraged, if you spend effort getting this problem right now, you will likely have less trouble later in the course. This is why we encourage you to work in pairs. The intention is that students who have Java experience can help those who have little or no experience to get up to speed. Nevertheless, if you get stuck, please do not hesitate to seek help from one of the TAs or LAs.

As you work on this problem, you should also record your answers to the various question in a file called `problem7.txt` in the project's `doc/` directory.

a. Warm-Up: Creating a Ball

As a warm up exercise, take a look at `Ball.java`. A **Ball** is a simple object that has a capacity.

- What is wrong with `Ball.java`? Please fix the problems with `Ball.java` and document your work in `problem7.txt`.

If you pay attention to the warnings in Eclipse, you should be able to find at least one of the bugs without referring to the JUnit results.

b. Using Pre-Defined Data Structures

Next, we want to create an class called **BallContainer**. As before, skeleton code is provided (see `BallContainer.java`). A BallContainer is a container for Balls. BallContainer must support the following methods and your task is to fill in the code that will implement all these methods correctly:

1. `add(Ball)`
2. `remove(Ball)`
3. `getCapacity()`
4. `size()`
5. `clear()`
6. `contains(Ball)`

One of the nice things about Java is that it has many libraries and pre-defined data structures that you can simply use without have to write your own from scratch. One of the intentions of this problem is to expose you to the use of pre-defined Java data-structures. In BallContainer, we use a `java.util.LinkedList` to keep track of the balls. If you open `BallContainer.java` in Eclipse, you will notice that there is a warning message associated with the line:

```
contents = new LinkedList();
```

It should say something about an **"Unsafe type operation"**. If you are not familiar with Java 1.5, you may be puzzled by this warning. Since Java 1.5, the Java Collections framework is strongly typed so instead of defining a generic `LinkedList`, you should define a typed `LinkedList`. Please modify this line to remove this warning for the constructor statement.

Before you proceed to implement the required method, please take a moment to read through the documentation for `LinkedList`. Some of the methods that you are required to implement simply require you to call the appropriate predefined methods for `LinkedList`.

Hint: Place your cursor on `LinkedList`, and press **SHIFT-F2**. A web browser window will appear, showing documentation for the `LinkedList` class.

Most of the methods that you are required to implement are quite simple. Before you start coding, please take time to think about the following questions (you need to turn in your answers):

- There are two obvious approaches for implementing `getCapacity()`:
    1. Every time `getCapacity()` is called, go through all the Balls in the `LinkedList` and add up the capacities. (**Hint:** If you choose this approach, you will probably want to use an `Iterator` to extract Balls from the `LinkedList`. You can see an example of how `Iterator` is used in `BoxTest.java`.)
    2. Keep track of the total capacity of the Balls in `BallContainer` whenever Balls are added and removed. This obviates the need to perform any computations.
- Which approach do you think is the better one? Why?

c. Implementing Algorithms

By the time you are done with `BallContainer`, you should be somewhat comfortable with Java, so in this problem, we want you to do a little more design and thinking and a little less coding. Your final task in this problem is to create a class called **Box**. A Box is also a container for Balls. The key difference between a `Box` and a `BallContainer` is that a Box has only finite capacity. Once a box is full, we cannot put in more Balls. The size (capacity) of a Box is defined when the constructor is called:

```
public Box(double capacity);
```

Since a `Box` is really a `BallContainer` with some extra properties, it makes sense to say that in fact a `Box` is a type of `BallContainer`, which is why the `Box` is defined to `extend BallContainer`. This is called *Inheritance*. Don't worry too much about this; you will learn more about this later in the course. For now, it suffices to know that what this means is that `Box` automatically has all the methods and properties of `BallContainer`. When you look in `Box.java`, you will realize that `Box` has only two methods defined in `Box.java`:

1. `add(Ball)`
2. `getBallsFromSmallest()`

Depending on your implementation of `getCapacity()` in Problem 7b, you may also need to implement a different `remove(Ball)` method for **Box** as well. If so, please do so. There is no need to change your implementation of `BallContainer` for this problem. You can also make other changes to Ball, BallContainer or Box, but you should explicitly describe your changes in `problem7.txt` and explain why the changes were made.

As before, your task is to implement these two methods.

Before you start working on `getBallsFromSmallest()`, you may wish to check out the documentation on `Iterator` (Place your cursor on `Iterator`, and press **SHIFT-F2**.) Also, take some time to answer the following questions (which you need to turn in):

1. There are many ways to implement `getBallsFromSmallest()`. Brainstorm with your buddy and come up with at least two ways. Briefly describe them.
2. Which of the above ways do you think is the best? Why?

There is no single **correct** answer. The whole point of this exercise is help you fight that urge to code up the first thing that comes to mind, but instead spend a little more time thinking before you start coding. Remember: **More thinking, less coding**.

Problem 8: Turning In Your Problem Set

Throughout this course, you will turn in the solutions for the problem sets by checking them into your CVS repository. All the relevant files should be included, for example:

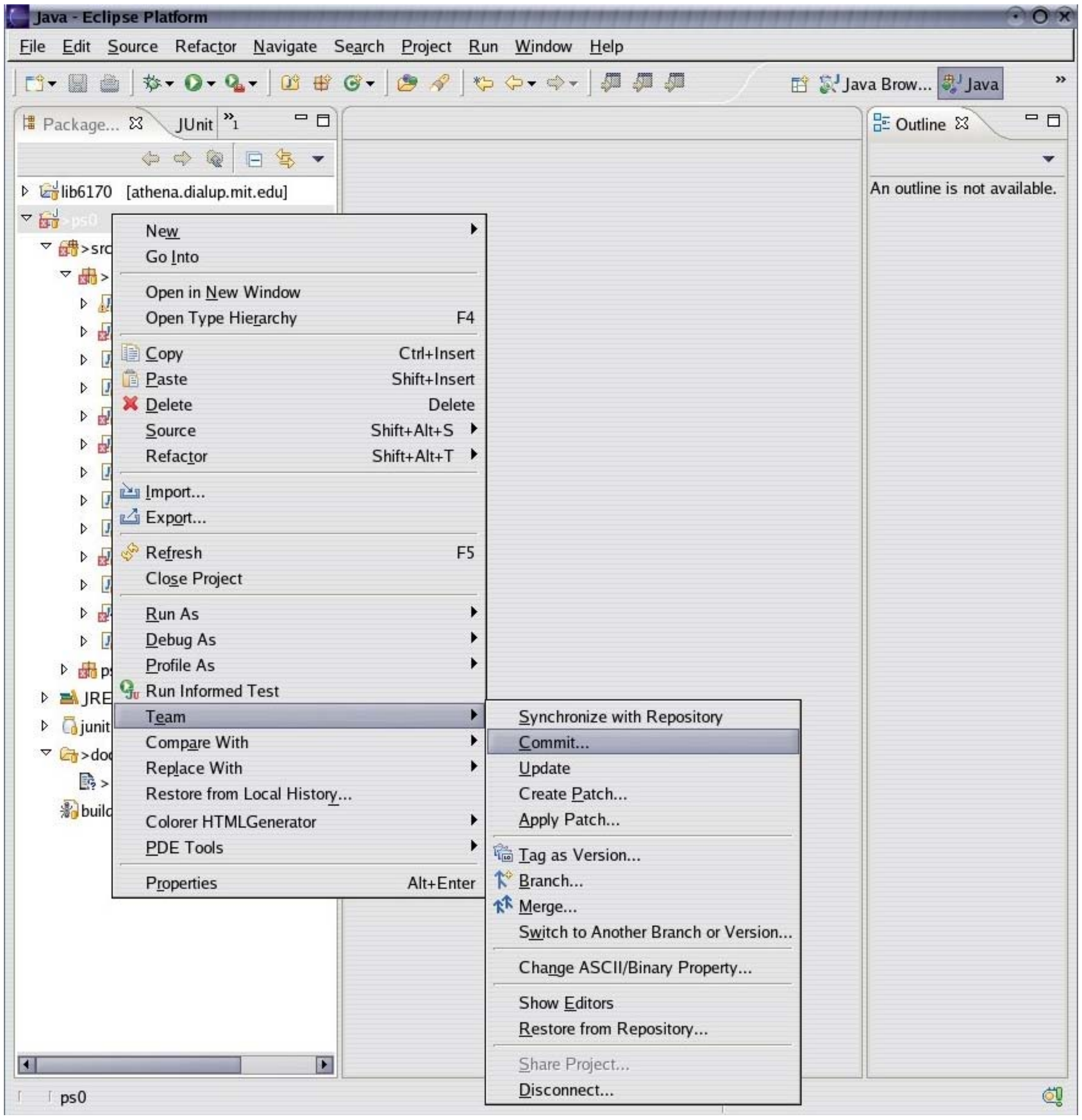
- Java source files we provide, and to which you make changes.
- Java source code you write.
- Text files, drawings, or documentation of your code.

Each problem set will indicate exactly what you need to turn in a Section entitled "What to Turn In"

If you add a new file, you must explicitly add to your CVS repository. Just adding the file to the repository is not enough; you need to **commit** the file. Furthermore, after making changes to a file, you must commit it so that we, the staff, can collect your most up-to-date version.

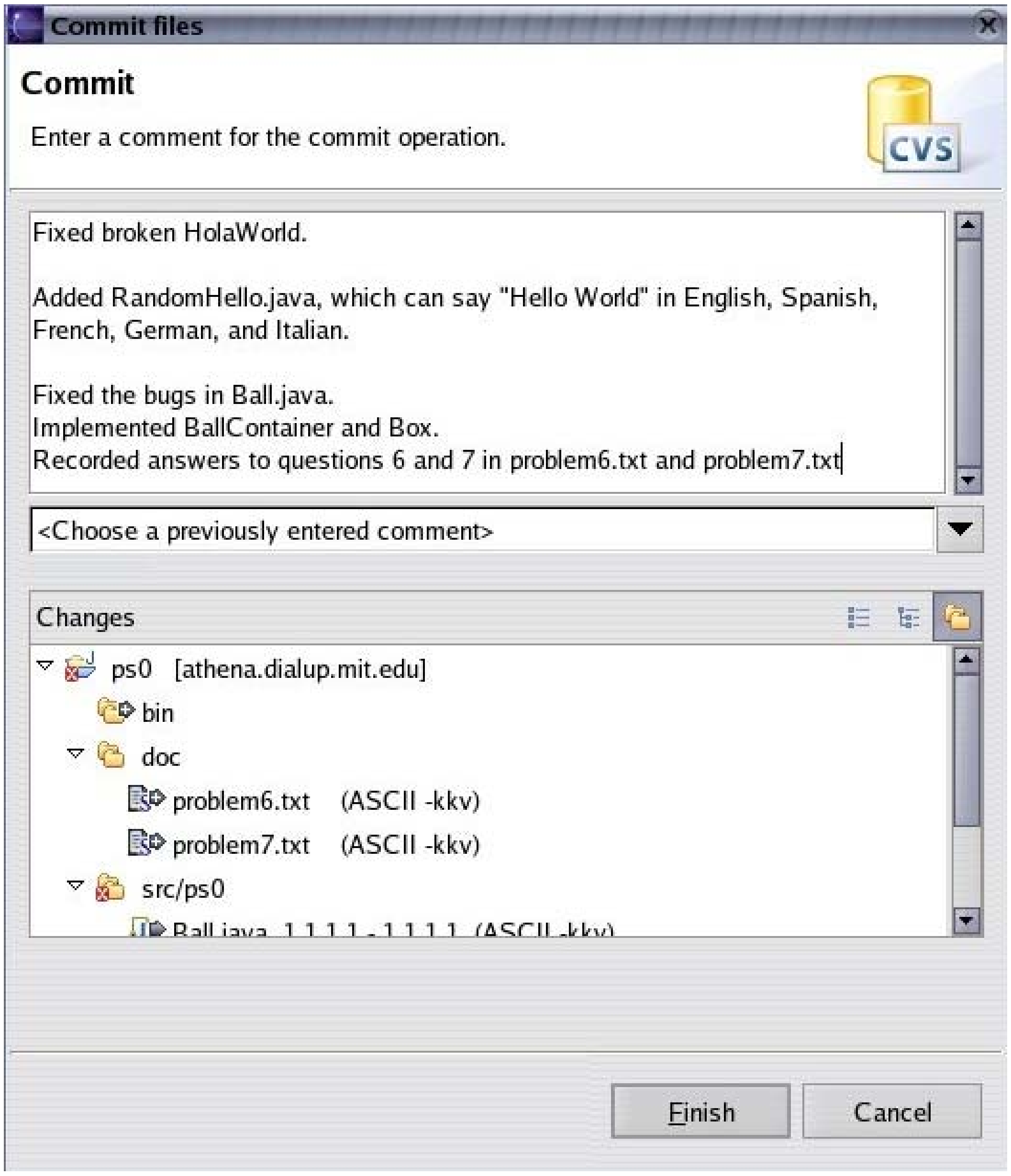
Since files like `RandomHello.java`, `problem6.txt` and `problem7.txt` are not in your CVS repository, you must add them. You also want to commit the changes you made to `HolaWorld.java`. Instructions for doing so follow:

- In the Package Explorer window, right-click on the top-most `ps0` in the directory structure (the one representing the entire project). Choose **Team >> Commit...**



Courtesy of The Eclipse Foundation.

- With every commit to CVS, you are allowed to append a comment describing changes you made since the last commit. If you ever need to look at old versions of your code, your commit messages will remind you of what happened between versions.
  Enter a descriptive commit comment, and click **Finish**. The lower panel shows you the list of files that you have modified and that will be committed to the CVS repository.


Commit files

Commit

Enter a comment for the commit operation.

```
Fixed broken HolaWorld.

Added RandomHello.java, which can say "Hello World" in English, Spanish,
French, German, and Italian.

Fixed the bugs in Ball.java.
Implemented BallContainer and Box.
Recorded answers to questions 6 and 7 in problem6.txt and problem7.txt
```

<Choose a previously entered comment>

Changes

```
ps0   [athena.dialup.mit.edu]
  bin
  doc
    problem6.txt    (ASCII -kkv)
    problem7.txt    (ASCII -kkv)
  src/ps0
    Ball.java  1.1.1.1 - 1.1.1.1  (ASCII -kkv)
```

Courtesy of Free Software Foundation.

After completing the above steps, all your code and materials to turn in should be in your CVS repository. The final step to turn in is to *validate* your solutions.
Please note: You can not run the "validate6170" script remotely by connecting to an Athena machine. To run the script, you have to be at an **actual Athena machine**. That's because the validate6170 script basically invokes javac (to look for compilation errors) and java (to look for some basic runtime errors). And the remote Athena machines do not have enough memory to load the Java VM.

The validate script performs general sanity checks on your solutions. The validate script will also test your code against our suite of public tests. At the `athena%` prompt, run:

```
validate6170 ps0
```

If validation was successful, you should see output that looks something like

```
athena% validate6170 ps0

Checking your cvs repository... done.

Compiling every Java source file... done.

Running junit... Pass=(100%) Fail=(0%) Error=(0%)

Note: this validation script can not guarantee that
your problem set is bug-free or 100% complete.
```

If there is an error, the validate script should provide some information about what is wrong.

```
athena% validate6170 ps0

Checking your cvs repository... done.

Compiling every Java source file... error occurred!
  src/ps0/HolaWorld.java:26:  expected
          System.out.println(world.);
                                  ^
  src/ps0/HolaWorld.java:26: ')' expected
          System.out.println(world.);
                                   ^
  2 errors
```

This error would indicate that there was a problem compiling one of the files (HolaWorld.java). If the file compiles fine for you, then you just have to make sure that it has been properly committed.

Run the `validate6170` program as many times as necessary until there are no failures or errors. You have now successfully turned in your first 6.170 problem set. We encourage you to give the Optional Tutorial Problems a try.

What to Turn In

Your TA should be able to find the following when he checks your problem set out of CVS:

- A version of `HolaWorld.java` with no compilation errors that works as described in Problem 3.
- A file called `RandomHello.java` in the `ps0` package that prints out one of five random messages when its `main` method is executed.
- A version of `Fibonacci.java` that passes the three tests in `FibonacciTest.java`. (Note that you should **NOT** edit `FibonacciTest.java` to accomplish this task.)
- A version of `Ball.java`, `BallContainer.java` and `Box.java` that pass their respective JUnit tests. (Again, you should not modify the JUnit tests, though you are most welcome to read the source code to understand what they test for).

- Files called `problem6.txt` and `problem7.txt` in the `doc/` directory that contains answers to the questions in Problems 6 and 7.

Optional Tutorial Problems

If you have not had much experience with Java, we recommend that you do these extra Optional Problems (source code for these problems can be found in the `ps0.optional` package that came with your `ps0` checkout). If you do not know Java, 6.170 will require you to learn it quite quickly. The optional problems, although not required, will help you become more comfortable with Java.

Hints

We strongly encourage you to use the 6.170 Online Forum.

Errata

1. You can not run the "**validate6170**" script remotely by connecting to an Athena machine. To run the script, you have to be at an **actual Athena machine**. That's because the validate6170 script basically invokes javac (to look for compilation errors) and java (to look for some basic runtime errors). And the remote Athena machines do not have enough memory to load the Java VM.

Q & A

This section will list clarifications and answers to common questions about problem sets. We'll try to keep it as up-to-date as possible, so this should be the first place to look (after carefully rereading the problem set handout and the specifications) when you have a problem.

> **Q:** Do I have to use Athena? Can I work at home?
> **A:** You can do either. You should look at the Problem Set Procedure Guide
> **Q:** Eclipse is too slow. What can I do?
> **A:** You have a few options. One is to try to find a faster machine. The HP Compaq P4s, SunBlade 1500s, and IBM IntelliStation E Pros in the Student Center are relatively fast. Otherwise, you may wish to use Emacs or some other text editor (such as Vim) instead of Eclipse.
> **Q:** Can I switch between running Eclipse on an Athena Sun versus running Eclipse on an Athena Linux?
> **A:** Eclipse is very picky about its workspace files. Switching from Linux to Sun and back has caused troubles for many people. Even switching from a new

version to an old version causes troubles. Here are the tips that the other TA's and LA's suggest:
(1)To be safe, either **stay exclusively on Sun, or exclusively on Linux**. You'd probably prefer Linux, since Eclipse runs much faster on Linux.
(2)To be safe, make sure you **commit your code to CVS** at the end of every session. Your code doesn't have to be perfect; you can commit your code as many times as you like before the deadline. This has many benefits:

- If Eclipse does clobber your profile, you can always delete the "6.170/.workspace" directory, then get your code out of CVS **without losing any work**.
- If you've made a big programming error, you can **revert** back to an earlier version of your Java source codes, and compare them to find out where the bug may be, for example.