

CSE 331 Software Design and Implementation

Handout T2: *Editing, Compiling, Running, and Testing Java Programs*

This handout describes how to perform common Java development tasks in the Allen Center software labs with Eclipse and the command-line.

Contents:

- [Starting Eclipse](#)
 - [Starting Eclipse on Linux](#)
 - [Starting Eclipse on Windows](#)
 - [Setting the JDK in Eclipse on Windows](#)
 - [Eclipse generics errors configuration](#)
- [Opening Files: Managing Multiple Files](#)
- [Creating New Files](#)
 - [New Java files](#)
 - [New Text files](#)
- [Editing Java Source Files](#)
 - [Autocompletion](#)
 - [Organizing Imports](#)
 - [Viewing Documentation](#)
 - [Eclipse](#)
- [Running Automated Tasks with Ant](#)
 - [Command-line](#)
 - [Eclipse](#)
- [Compiling Java Source Files](#)
 - [Command-line](#)
 - [Eclipse](#)
 - [Compiling With Ant in Eclipse](#)
- [Running Java Programs](#)
 - [Command-line](#)
 - [Eclipse](#)
- [Testing Java Programs with JUnit](#)
 - [Running JUnit Tests](#)
 - [Command-line](#)
 - [Eclipse](#)
 - [CSE 331 JUnit Framework](#)

Starting Eclipse

You should have already performed the initial setup described in [Tools: Initial Setup](#). If you are working on your own computer, you should have [installed Eclipse](#). (Students who are working through the handouts in the recommended order have already done both steps!)

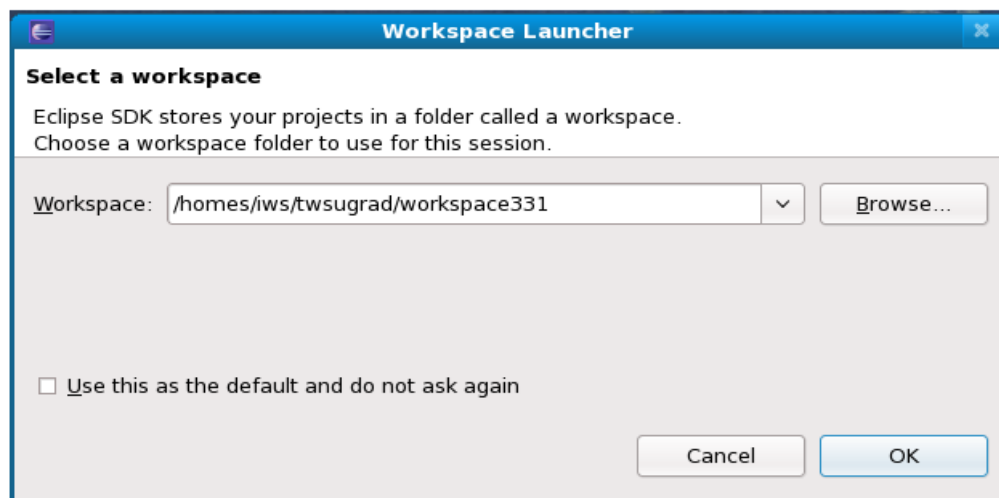
- [Starting Eclipse on Linux](#)
- [Starting Eclipse in Windows](#)

Starting Eclipse on Linux

Type this at the prompt to start Eclipse on Linux:

```
eclipse &
```

Eclipse will start up, display a splash screen, and then show a workspace selection dialog:



Eclipse is asking you which workspace folder to use for this session. In response, type:

```
/homes/iws/YourUserName/workspace331
```

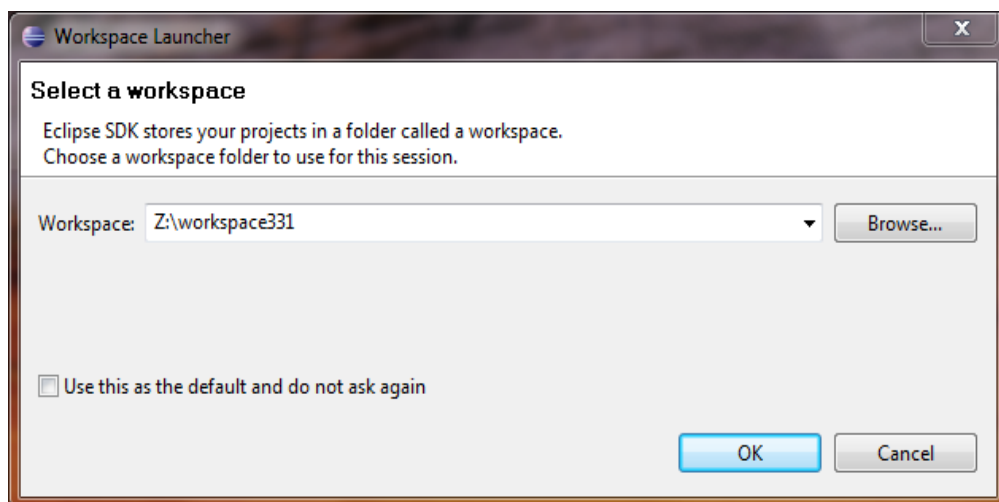
where *YourUserName* is your UW CSE username. (**Note:** Do **not** enter "`~/workspace331`" for this step. Eclipse does not recognize the '~' character.)

If Eclipse shows the welcome screen, containing only the text "Welcome to the Eclipse IDE for Java Developers" on a pretty background, switch to the code editor by going to Window > Open Perspective > Other... and selecting `Java (default)`.

Starting Eclipse on Windows

From the start menu, goto: **All Programs » DEV TOOLS & LANGUAGES » Eclipse » eclipse**.

Eclipse will start up, display a splash screen, and then show a workspace selection dialog:



Eclipse is asking you which workspace folder to use for this session. In response, type:

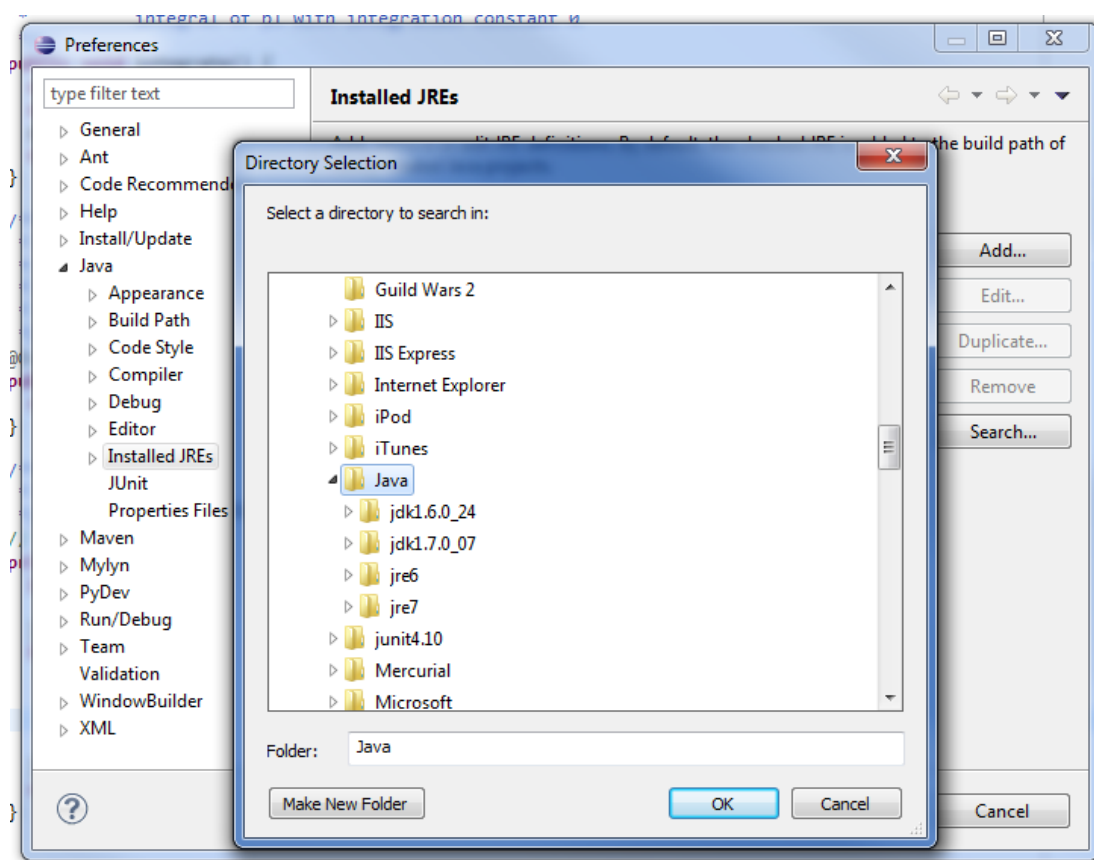
```
Z:\workspace331
```

This directory will work on the lab Windows computers; you may need to adjust it if you are working on your personal computer.

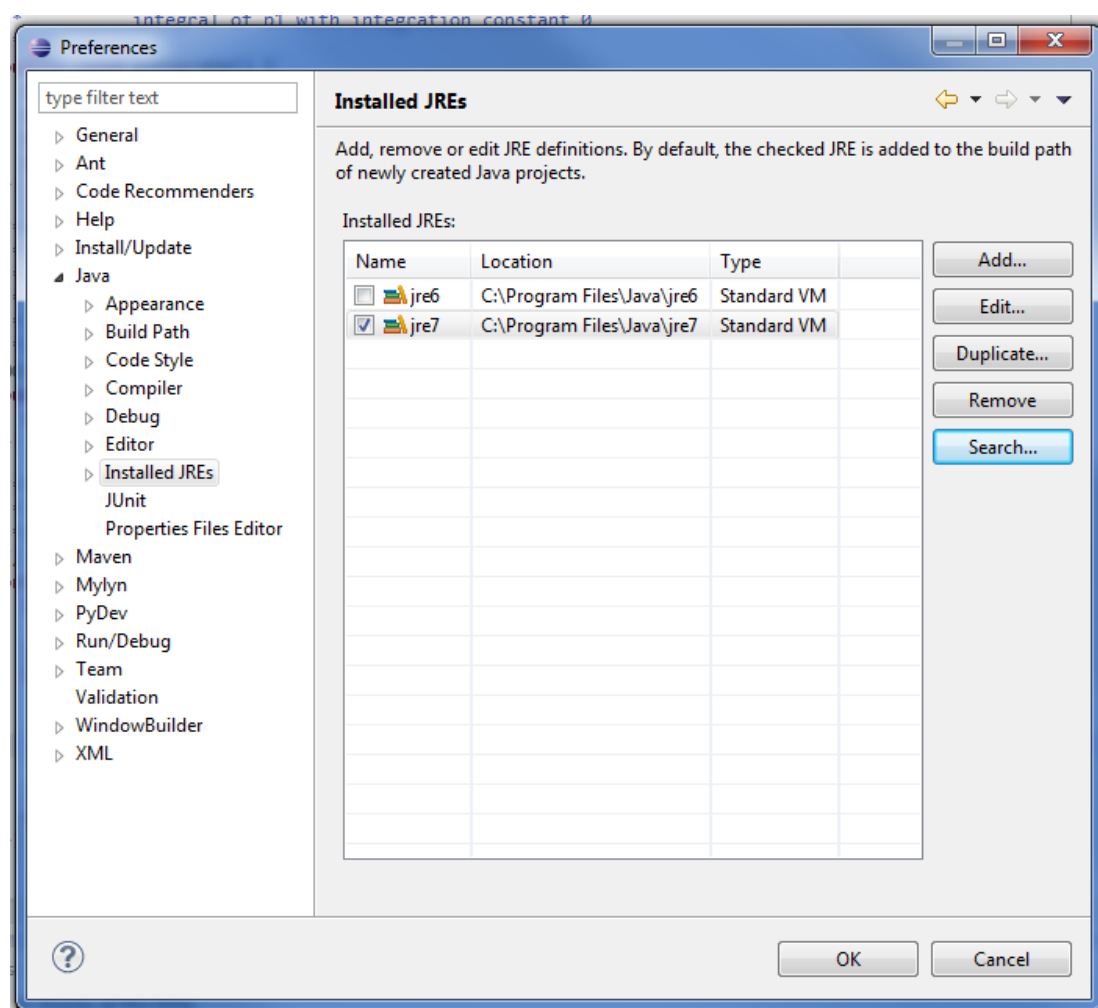
Setting the JDK in Eclipse on Windows

If you are using one of the Windows instructional machines and would like to use the Ant buildfile, you need to tell Eclipse to use the Java JDK (development kit) instead of the JRE (runtime environment).

- In Eclipse, goto **Window » Preferences** to open the Preferences dialog.
- In the left pane of Preference dialog, select **Java » Installed JREs**
- Click the **Search...** button in the Installed JREs pane
- In the directory selection window that appears, select **C:\ » Program Files » Java** as shown in the screenshot below.



- Click **OK**
- In the Installed JREs list, check the box next to jdk1.7.0_04 (or any version that starts with jdk1.7) and click **OK**



- g. If Eclipse shows the welcome screen, containing only the text "Welcome to the Eclipse IDE for Java Developers" on a pretty background, switch to the code editor by going to **Window > Open Perspective > Other...** and selecting **Java (default)**.

Eclipse generics errors configuration

We expect your code to not have any generics-related problems. For example, the following code is unacceptable:

```
List myList = new ArrayList();
myList.add("foo");
```

The generic type `List` of `myList` should be parametrized, for instance, to `String` by replacing the first line with `List<String> myList = new ArrayList<String>();` Note that `List<String> myList = new ArrayList();` is also incorrect.

By default, Eclipse shows generics problems as warnings (indicated by yellow lines and markers). You can configure Eclipse to instead issue errors (indicated by red lines and markers) for these problems. Doing so will help you remember to write acceptable generics code.

To make this configuration, go to **Windows » Preferences** and select **Java » Compiler » Errors/Warnings**. Under **Generic types**, change the value of **Unchecked generic type operation** to **Error**.

(Note that there is another setting named **Usage of a raw type** that is set to **Ignore** by default. We recommend leaving this option disabled or set simply to **Warn** because it is specific to the Eclipse compiler and checks for more stringent requirements than required

by the Java language specification. Thus, "Usage of raw type" may complain about issues, that while providing insight about your code, is not checked by Oracle's `javac`, which is our official standard for compilation errors and warnings in this class.)

Opening Files; Managing Multiple Files

Switch to the "Java" perspective in Eclipse if you're not already in it (**Window » Open Perspective » Other... » Java**).

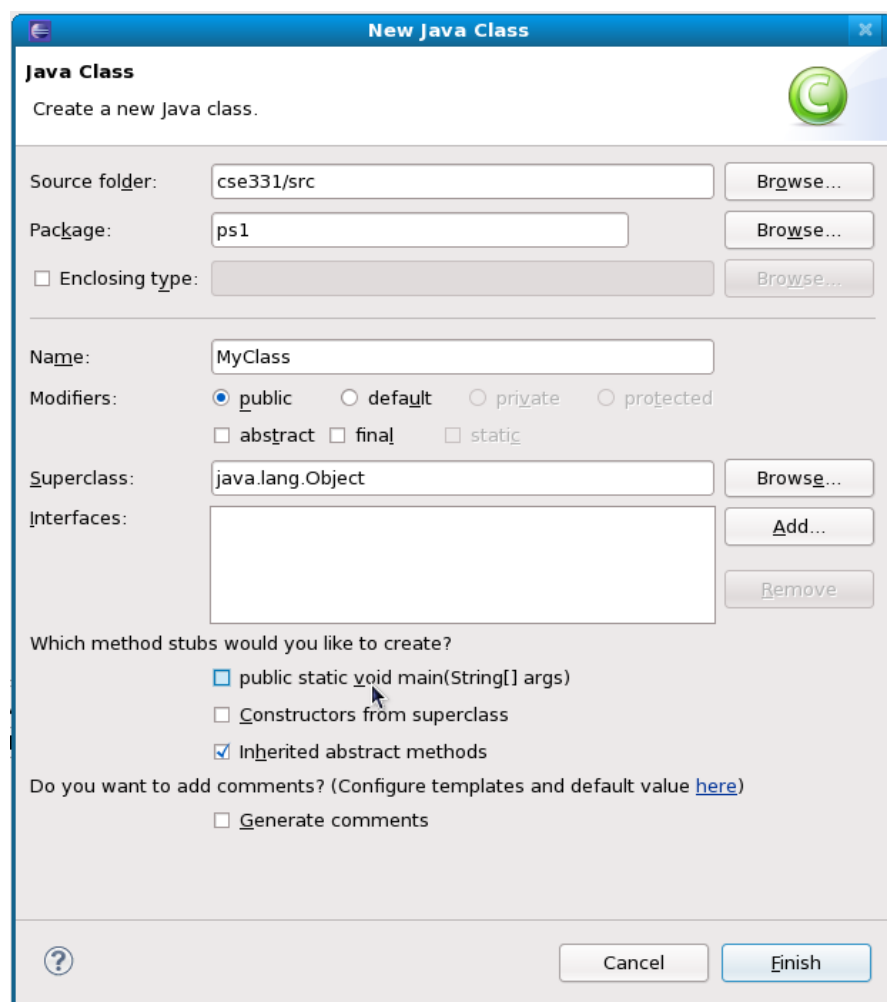
You can open multiple files in Eclipse by double-clicking each of them from the Package Explorer pane. Once you have multiple files open, you can switch between them quickly by holding down **Ctrl** and hitting **F6** to bring up a dropdown box of open files, and then using the arrow keys to select the file whose editor you wish to bring into focus. You can also navigate through different files using the tabs on top of the editor pane.

Creating New Files

New Java files

To create a new Java source file (with the `.java` extension), select from the top menu **File » New » Class**. A window will pop up, asking you details about the class. Leave the source folder as **cse331/src**, and select a package (e.g. `hw1`). Choose a name for your class (e.g. `MyClass`) Type this name in the "Name" field and click **Finish**.

(If you want your new class to be executable, it will need a `main` method. Eclipse can generate that automatically for you if you check the appropriate checkbox in the **New Java Class** screen.)



New Text files

There is a similar procedure for creating new non-Java files such as text files. Select **File » New » File**. In the resulting dialog box, choose the parent directory of your new file and type the desired filename. If you want to create a new directory, you can do so by appending the directory name in front of your desired filename. For example, if you want to create a file `problem0.txt` in the directory `hw1/answers`, but the `answers` directory does not yet exist, you can choose `hw1` as the parent directory, and then type `answers/problem0.txt` as the file name, and Eclipse will create the new directory and file for you.

Editing Java Source Files

Here are some useful actions that you can perform when editing Java code:

- [Autocompletion](#)
- [Organizing Imports](#)
- [Viewing Documentation](#)

Autocompletion

Autocompletion is the ability of an editor to guess what you are typing after you type out only part of a word. 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.

Eclipse continuously parses your Java files as you are editing, so it is aware of the names

of variables, methods, etc... that you have declared thus far.

CTRL+Space can be used to autocomplete most things inside the Eclipse Java editor. For example, if you have declared a variable named `spanishGreeting` in the current class, and have typed the letters `spanishGree` in a subsequent line, Eclipse can infer that you mean to type `spanishGreeting`. To use this feature, press **CTRL+Space** while your cursor is at the right of the incomplete name. You should see `spanishGree` expand to `spanishGreeting`.

Eclipse can also help you autocomplete method names. Suppose you have a variable `myList` of type `List`, and you want to call the method `clear`. Begin typing "`myList.`" - at this point, a pop-up dialog will display a list of available methods for the type `List`, and you can select the appropriate method with the arrow keys. You can force the popup to appear with **CTRL+Space**.

Organizing Imports

You can press **CTRL+SHIFT+o** to *organize* your imports in a Java file. Eclipse will remove extraneous `import` statements and try to infer correct ones for types that you refer to in your code but have not yet been imported. (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 which one to import.)

Viewing Documentation

Although you can directly browse the [Java 7 API](#) and other documentation at the Oracle website, it is often useful to be able to cross-reference parts of your code with the appropriate documentation from within your editor.

Note that you need to [generate the api docs locally](#) before you can view docs for classes in the assignment.

Eclipse

To view the documentation of a class that is referred to in your code, place your cursor over the class's name, and press **SHIFT+F2**. A web browser window will be opened to the class's documentation page. If the class is provided by Java, the documentation page will be on Oracle's website.

For your own classes, you will need to tell Eclipse where to find their documentation. To do so, right click on the project name (e.g. `cse331`) in the Package Explorer pane and click **"Properties"**. Select **"Javadoc Location"** in the left pane. Select the location, e.g. **"file:/homes/iws/YourUserName/workspace331/cse331/doc/"**. (Note: the **"file:"** portion is important, since the location is expected to be recognizable by a web browser.) After setting the Javadoc location path, click **OK**.

Running Automated Tasks with Ant

Ant is a tool that can be used to automate many common tasks, such as compiling, testing, and running code. It is also used to [validate a CSE 331 assignment submission](#). The instructions for Ant are stored in a "buildfile" named `build.xml` in each assignment's directory.

The buildfile specifies a set of **targets** that it supports, such as [build](#) and [test](#). Note that

the "help" target will output information about the supported targets in our buildfile.

Important: the [validate](#) target is only supported on `attu`.

- [Command-line](#)
- [Eclipse](#)


Command-line

To run Ant for assignment N from the command line, do the following:

```
cd ~/workspace331/cse331/src/hwN
ant target
```

Eclipse

To run Ant in Eclipse, right click `cse331/src/hwN/build.xml` in the **Package Explorer**, where N is the desired assignment number. Now right click **Run As » Ant Build...** and in the resulting window, go to **Targets** and select the desired target(s).

There is a button near the left-hand side of the Eclipse toolbar that looks like  which will re-run the last ant target (or other external tool) that you ran. The drop down button will let you easily re-run a number of ant targets.

Compiling Java Source Files

You must compile your source code before running it. The `javac` compiler is used to transform Java programs into bytecode form, contained in a *class file*. Class files are recognized by their `.class` extension. The bytecode in class files can be executed by the `java` interpreter.

- [Command-line](#)
- [Eclipse](#)

Command-line

To compile all source files for a particular assignment, change your current directory to `~/workspace331/cse331/src/hwN/`, where N is the desired assignment number. Now type on the command-line:

```
ant build
```

This will run an Ant script that uses the instructions denoted in `build.xml` to compile all the `.java` files into corresponding `.class` files. Note that if one or more of your files do not compile, you will receive error messages and no `.class` files will be generated for the files that do not compile properly.

If you would like to manually compile files without the use of an Ant script, you can use `javac` to compile one or more source files into class files for execution by the Java interpreter. The following commands:

```
cd ~/workspace331/cse331/src
javac -Xlint -d . -g [more options] hwN/file1.java hwN/file2.java ...
```


will generate class files `hwN/file1.class`, `hwN/file2.class`, etc., for each specified source file. Type `"man javac"` at the `attu` prompt for more information on `javac` options. You should almost always use the `-g` option, which will provide improved debugging output, and also the `-Xlint` option, which provides stricter compiler warnings.

Eclipse

Eclipse is set up by default to automatically recompile your code every time you save. Classes with compile errors are marked in the **Package Explorer** with red cross marks. The compile errors, as well as compile warnings, also appear in the **Problems** view (usually situated at the bottom panel of Eclipse).

If your file is saved and Eclipse says that it does not compile but you believe that it should, make sure that all of the files on which your file depends are saved and compiled. If that does not work, try refreshing your project or using **Project » Rebuild Project** to force Eclipse to recognize the latest versions of everything.

Compiling With Ant in Eclipse

To compile with Ant in Eclipse, right click `cse331/hwN/build.xml` in the **Package Explorer**, where `N` is the desired assignment number. Now right click **Run As » Ant Build...** and in the resulting window, go to **Targets** and select **build** in the list of targets.

Running Java Programs

Once you have compiled your source code into class files, you can execute it with the Java interpreter.

- [Command-line](#)
- [Eclipse](#)

Command-line

Typically, to run a program you will just type `ant` on the command line, possibly with a more specific target: `ant target`. However, you can also invoke the Java virtual machine directly via the `java` program.

Here is how to run Java programs from the command-line:

```
cd ~/workspace331/cse331/src
java -ea hwN.theClassYouWantToRun
```

(The `-ea` flag enables assertions.)

For example, if you wish to run the `PolyGraph` class from `hw1`, you would run:


```
java -ea hw1.PolyGraph
```

Note that you do not include `.java` or `.class` at the end of the class name.

Eclipse

To run a program, right click on the Java source file containing the `main()` method and

choose **Run As... » Java Application**.

There is also a button near the left-hand side of the Eclipse toolbar that looks like  which will re-run the last application (or JUnit test, see below) that you ran.

Testing Java Programs with JUnit

JUnit is the testing framework that you will be using for writing and running tests.

For more information, visit:

- <http://junit.org>, the official web site.
- [JUnit Cookbook](#), a brief tutorial.
- [JUnit API](#)

Running JUnit Tests

You can run JUnit from several different environments:

- [Command-line](#)
- [Eclipse](#)

Command-line

As mentioned above, the `test` Ant target can be used to test `SpecificationTests` and `ImplementationTests`.

To run JUnit without using Ant, change to the `~/workspace331/cse331/src` directory and invoke `java org.junit.runner.JUnitCore` for the text interface with the name of the test class as a parameter. For example, if I wanted to run `RatTermTest` from `hw1`, I could type:

```
cd ~/workspace331/cse331/src
java org.junit.runner.JUnitCore hw1.test.RatTermTest
```

Eclipse

JUnit is integrated with Eclipse, so you can run the test suite from within the IDE.

- First, select the test you want to run from the **Package Explorer** (usually the leftmost pane).
- From the Eclipse menu at the top of the screen, select **Run » Run As » JUnit Test**
- The JUnit GUI should pop up in place of the Package Explorer momentarily and run all the tests. You can double-click on failed tests to jump to the code for that test. When you're done inspecting the JUnit results, close the JUnit pane to go back to the Package Explorer.

If you are not working at UW CSE, you might have to explicitly add the `junit-4.11.jar` library using **Project » Properties » Java Build Path » Libraries » Add External JARs** and then telling Eclipse where your copy of `junit-4.11.jar` is located.

CSE 331 JUnit Framework

Because your JUnit tests will likely have different class and method names than those of

your classmates, there needs to be a standardized way of accessing every student's tests. Thus, each assignment comes with the JUnit test classes `hwN.test.SpecificationTests` and `hwN.test.ImplementationTests`. You will load all the JUnit tests you wrote in one of these two test suites.

`hwN.test.SpecificationTests`, as its name suggests, should contain only *specification tests* - that is, those tests that check only for features implied by the specification. Consequently, your specification tests should be valid tests for any other person's code that claims to satisfy the same specification, even if that implementation is inherently very different.

Conversely, `hwN.test.ImplementationTests` should contain *implementation tests* - that is, those tests that test only details that are specific to your implementation.

As an example, suppose you were implementing the following specification:

```
/** Frobs the blarghnik.
 * @requires b != null
 */
public void frob(Blarghnik b);
```

A *specification test* should never pass in a null parameter to this method - this would violate the specified pre-condition. However, your particular implementation might check for the null parameter and throw a `NullPointerException`. Your *implementation test* can safely exercise this case by passing in null.

Similarly, an iterator specification which does not specify the order in which elements are returned indicates that no specific order should be assumed in a *specification test*. Your implementation may happen to keep elements in a sorted list, and so your *implementation test* may wish to check that the elements returned by the iterator are sorted.

Before you submit each assignment, you should ensure that your code passes both the `SpecificationTests` and `ImplementationTests` test suites. We have provided a convenient way to run these tests in each assignment's build file in `~/workspace331/cse331/src/hwN/build.xml`. The `test` target will run `hwN.test.SpecificationTests` and `hwN.test.ImplementationTests`. The `validate` target runs these tests on a fresh copy of your code it checks out from your repository.

For problems or questions regarding this page, contact: cse331-staff@cs.washington.edu.