1 Fundamentals of Test-First Design

1.1 Setting up the Eclipse IDE

We will use the Eclipse IDE throughout the week. In the introductory classes the *tester* library can be used with any Java IDE, but later in the week as we move to more advanced material, we will use some Eclipse plugins that for now are not available with other IDEs.

Setting up your workspace

Start by setting up a directory for your Eclipse projects. Typically you will have one project per lab session, plus an on-going project you will work on during the week. We assume you call it *EclipseWorkspace*.

Create another directory at the same level (though this is a convention, not a requirement) where you will keep all library (JAR) files, call it *EclipseJars*. For the rest of the Lab/term we will refer to these two directories as *EclipseWorkspace* and *EclipseJars*.

If you do not have Eclipse on your laptop, download and install Eclipse from the **http://www.eclipse.org/**.

Start Eclipse. It should ask you where you want your workspace to be... so enter (or click `Browse` and navigate to) the workspace directory you created. Feel free to check the "*Use this as the default...*" box. Click OK.

Once Eclipse starts, close the annoying *Welcome* screen if it comes up.

Tabbing setup

There's a few settings you may want to change the way before you start — mainly to use spaces instead of tabs, and reduce the indentation level.

Select `Preferences` under the `Window` menu and change the following settings.

1. Type "`tab`" in the search box at the upper-left to minimize the available selections.

2. Select "`Text Editors`" on the left. Make sure the "`Insert spaces for tabs`" check-box is checked.

3. Select "`Formatter`" on the left. Click the "`Edit...`" button, then choose the "`Indentation`" tab at the top. Change the "`Tab policy`" to "`Spaces only`".

4. When you say OK it will force you to create a name for the profile... you can just say "`Mine`" or some other descriptive name.

Project setup

The instructions for getting started with the Eclipse project are at the *javalib* web site:

http://www.ccs.neu.edu/javalib/SetupGuides/Eclipse.html

We suggest that you save all libraries we will need in one directory, named **EclipseJars**. To simplify the downloading of all libraries, we have bundled all of them in the file **EclipseJars.zip**. Download this file and unzip it at the same level as your *Eclipse* workspace. You find this file at:

http://www.ccs.neu.edu/home/vkp/HtDC/Workshops/Summer2011/EclipseJars.zip

Most of these libraries can also be downloaded from the *Downloads* section of the *javalib* website:

http://www.ccs.neu.edu/javalib

When downloading single *jar* files on the Windows platform, make sure you do not download it as *WinZip* file, instead select the *All Files* option.

1.2 Unit test design with the *tester* library

The sample code from the *javalib* web site defines several classes, one interface, and a number of methods.

Read the code, look at how the tests are designed, add errors and see how the failed tests are reported.

At the end of the `Examples` class add the method:

```
// run the program/tests directly
public static void main(String[] argv){
  Examples e = new Examples();

  // run the test report displaying all tests
  // successes and failures, and
  // displaying all data
  Tester.runReport(e, true, true);
}
```

This variant of the tester invocation gives the user the choice of whether to display all tests, not just failed ones, and whether to display all data. We provide an option for novices that avoids the introduction of the `public`

`static void main`, but more seasoned programmer may want the choice of how the reports should be generated, and avoid setting up the configurations for each project.

The tests in the `Examples` class are written in the functional style (no `void` methods). However, using this style, once the first test of the compound *and* expression fails, the remaining tests are not evaluated. While our students do not see the `void` methods for several weeks, we will allow their use in the test suites.

Change all test methods to `void` and run the tests again.

1.3 Understanding Data: Designing Methods

The goal of this lab, now that you have master the logistics, is to understand the importance of systematic design of data and its impact on the design of methods. Select one of the three options, then discuss the other options with those that worked them out. Make sure you follow the design recipes when designing data and when designing methods.

1. Extend the definition of shapes by removing the `abstract` class `AShape` and adding a new class `Combo` that combines tow shapes, a top one and a bottom one.

   Rewrite all methods as needed, add the methods for the `Combo` variant, together, of course, with the appropriate tests.

   Then design the methods with the following purpose statements:

   - produce the larger of the two shapes: this and the given one
   - produce a shape that looks the same as this one but is twice the size

   Think what would be needed to test the second method without having the `tester` library.

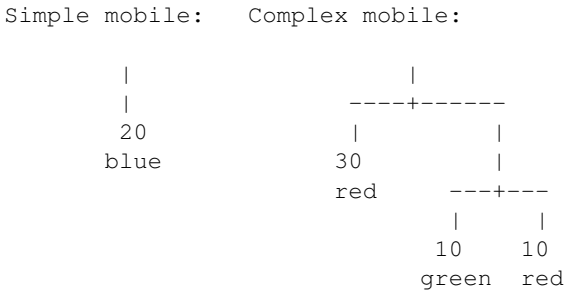
2. Import the lecture code for the files and directories problem into a new project. Make sure you can run it.

   - design the method `largestFile` that produces the size of the largest file in a directory (including all subdirectories).
   - design the method that produces a list of all files of the given `type` in a directory and all its subdirectories.

Think what would be needed to test the second method without having the `tester` library.

3. Start a new project. Design the data representation of a mobile. A mobile is either a simple ball (we know its color and weight) hanging on a string of some length, or it is a string of some length that has at the end a horizontal strut (we know the length of the left and right part from the place where it is suspended) with a mobile hanging from each end of the strut. (Use just a `String` to represent the color.)

   Here are some examples:



   Design the following methods for your mobiles:

   - Design a method that will produce the total height of this mobile. Each ball has height 10.
   - Design a method that will produce a new mobile where every red ball is replaced with a yellow one.
   - Design the method that determines whether the mobile is balanced. The mobile is balanced when the weight of the left mobile multiplied by the length of the left strut equals the weight of the right mobile multiplied by the length of the right strut, and every mobile hanging from each strut is also balanced.

   Think what would be needed to test the second and the third method without having the `tester` library.