211lab06_S11

Lab 06 - Simple Java Classes and JUnit Testing

Recommended Reading:

Note: Some of the code used in the following reading differ from that used in class!

- Principles of Object Oriented Programming (Connexions Course)
  - Introduction
    - Abstraction
    - Objects and Classes
    - Object Relationships
    - UML Diagrams
  - Polymorphism in Action
    - Union Design Pattern: Inheritance and Polymorphism
  - Labs (info in this section may be a bit dated as it has not been updated for the latest version of DrJava.)
    - DrJava
    - Unit Testing with JUnit in DrJava
  - Resources
    - Java Syntax Primer

1. Writing Simple Java Classes (45 min)

First steps

1. If you have not done it already, download the latest DrJava to your Desktop. Use this link. Download the "latest developement" release!
2. Create a `lab06` directory within your Comp211 directory.
3. **Set your DrJava "Language Level" to "Functional"** (aka "Elementary"). The Elementary language level simplifies the real Java syntax to facilitate the transition from Scheme to Java.
4. Create a class to calculate the areas of rectangles in DrJava by following the steps below. We will begin the process of writing a class that represents a rectangle given its width and height. We will write the class in several small steps. At each step, we will compile the code to ensure that everything is syntactically correct. By compiling the code at each small step, we hope to avoid seeing a large number of error messages that can be rather intimidating.
   - In the Definitions pane (upper right pane), type the following:

```
class Rectangle {
    double width;
    double height;
}
```

     Note the placement of the curly braces: the opening brace is on the same line as the class name, while the closing brace lines up with the beginning of the class definition on a new line. This is the *de-facto* Java coding style. The two lines beginning with `double` define fields within the class, much like the fields of a structure in Scheme.
     The word `double` is a the name for the principal real number type in Java. So the `width` and `height` fields of a `Rectangle` have type `double`. The `Rectangle` class is not very interesting because we cannot do much with it. However, it is syntactically correct and we can compile it.
     Aside: The equivalent Scheme code for class Rectangle is something like:

```
define-struct Rectangle(width height)
```

   - Save the file in your `Comp211/lab06` folder under the default name `Rectangle`. The file will be saved as a `.dj0` file, which is a DrJava Elementary level file.
   - Click the "Compile All" button in DrJava. The file should compile with no errors.
   - In the Interactions pane (bottom pane), type the following:

```
Rectangle r = new Rectangle(5, 10);
```

     This code defines a variable called `r` of type `Rectangle` and binds it to a new object belonging to class `Rectangle`.
   - There is not much we can do with the `Rectangel` class yet, but we can print the string representations of `Rectangle` objets. In the Interactions pane, type `r` and hit Enter. This action will print the string representation of object bound to `calc`. In the DrJava Elementary language level, the string representation of the `Rectangle` object `r` is {"Rectanlge(5.,10.)"}. DrJava automatically strips the quotation marks off string representations when it prints them. Conventional Java would return a much more cryptic string representation for a `Rectangle` object.
   - Take a look at your `Comp211/lab07` directory and you will see several new files. The compiler automatically created a file called `Rectangle.class` containing the compiled cJava bytecode ready execution on the Java Virtual Machine (JVM). The DrJava Interactions

Pane is a convenient user interface to a Java Virtual Machine. When you refer to a Java class in the Interactions pane, DrJava automatically loads the byte code for that class. When you compile the file `Rectangle.dj0` using DrJava, DrJava generates a corresponding conventional Java source file called `Rectangle.java` and invokes the Java compiler to translate this Java source file to bytecode.

5. Now we are ready to add to `Rectangle` a method to compute the area of a rectangle. Change the definition of the `Rectangle` class to read:

```
class Rectangle {
    double width;
    double height;

    // The only new lines:
    double area() {
       return width * height;
    }
}
```

- Compile All: the `Rectangle` file is automatically saved and recompiled.
- To create an instance of class `Rectangle`, type the following in the Interactions pane:

```
Rectangle r = newRectangle(4,5);
```

Note: the "Compile All" command resets the Interactions pane erasing all extant definitions of names bound to objects. So when we compiled our new definition for the `Rectangle` class, we destroyed our first binding of `r` in the Interactions pane.

- To "ask" the `Rectangle` object bound to `r2` to compute its area, type

```
r2.area();
```

in the Interactions pane.

- Experiment with the DrJava Interactions pane to create some new rectangles and computer their areas.

Defining Right Triangles and Circles

Suppose we want to define a class representing a right triangle given the length of the two sides forming the right angle, which we call `base` and `height` . What code should we write? Use the DrJava `New` command to create a new document. In this document define a class called `RightTriangle` with fields of type `double` called `base` and `height`, and a method

```
double area()
```

to compute the area of this right triangle. Save and compile this file. Interact with it in the Interactions pane.

- Now suppose we are asked to define a class that can compute the area of a circle. How would you do it? Write a class called `Circle` that has one double field, radius, and a method to compute the area.

2. Unit Testing with JUnit (35 min)

Extreme Programming (XP) is a software development methodology that is "very hot" these days and is currently embraced by the software industry. One key element of XP, called "unit testing", is the idea of writing test code for functions (or procedures or object methods) before the functions (or procedures or object methods) are even written. (Don't ask me how to write test code for the test code themselves! The whole thing is "kinda" suspiciously recursive with no well-defined base cases.)

The test code becomes in effect the "documentation and specification" of the behavior of the functions (or procedures or object methods) in code form!

Each time a function (or procedure or object method) is modified, we require that it passes its existing test suite. Each time a test code for a function (or procedure or object method) is modified, the corresponding function (or procedure or object method) may have to be revised to satisfy the new specification. Tests and code must be developed in tandem.

JUnit (http:-www.junit.org) (www.junit.org) is an open source framework developed to support the above concept of unit testing for Java programs. This tutorial will lead you through a simple example of how to write unit test code in developing a (simple) Java program with only one class.

Step 0.

Run DrJava with Elementary Language level. We will do all development in DrJava since it has very nicely integrated JUnit with its development environment.

Step 1.

Suppose we want to model the notion of a smart person who knows his/her birthday and can compute the number of months till his/her next birthday given the current month. For our purpose, a month can simply be represented by an integer, which in Java is called int. We start by writing "stub" code for the class Person that is to represent our smart person.

```
class Person {
  /**
   * Computes the number of months till the next birthday given the current month.
   */
  int nMonthTillBD(int currentMonth) {
    // todo
  }
}
```

Notice in the above there is really no concrete code. As a matter of fact, the above would not compile. Now we must abandon everything and start writing test code for nMonthTillBD(...).

Step 2.

DrJava is very nice to you and will create a test stub class for you if you know what to click:

- Go to the File menu and select New Junit Test Case...
- Create a stub test case called Test_Person. DrJava will create a stub class that looks like the following

```
/** * A JUnit test case class.
 * Every method starting with the word "test" will be called when running
 * the test with JUnit. */
class Test_Person extends TestCase {
  /** * A test method.
   * (Replace "X" with a name describing the test. You may write as
   * many "testSomething" methods in this class as you wish, and each
   * one will be called when running JUnit over this class.)
   */
  void testX() {
  }
}
```

At this point, do not worry about what "extends TestCase" means in the above. DrJava is re-using the class TestCase from the JUnit framework in some clever way without you having to deal with all the details of how to use an existing class from some other files.

If you have the "stable release, v. 20100913-r5387, you may receive a compile error. To remedy this, at the top of the Test_Person file, add the line

import junit.framework.TestCase;

Or simply download and replace with a later version, e.g. the latest development version.

- Compile this test class only (by clicking on Tools/Compile current document), and save the class in the same directory as Person. The Test_Person class should compile, though the Person class does not.

- Change the name of the stub method testX() in the above to test_nMonthTillBD()and add code to make it look like the code below.

```
class Test_Person extends TestCase {
 void test_nMonthTillBD() {
   Person peter = new Person(9);// a person born in September.
   assertEquals("Calling nMonthTillBD(2).", 7, peter.nMonthTillBD(2));
   assertEquals("Calling nMonthTillBD(9).", 0, peter.nMonthTillBD(9));
   assertEquals("Calling nMonthTillBD(12).", 9, peter.nMonthTillBD(12));
 }
}
```

Note that in the code for test_nMonthTillBD(), we have decided that we only need to know the birth month of a person in order to compute the number of months till the next birth day. As a result, we instantiate a Person object by passing only the birth month: new Person(9).

Also the test covers three cases:

- the current month is less than the birth month

- the current month is equal to the birth month
- the current month is greater than the birth month.

The

```
assertEquals
```

method comes from the class TestCase and takes in three parameters:

- the first parameter is a String
- the second parameter is the expected result
- the third parameter is the actual result of the computation you are testing.

Most of the time, your test code will call on the assertEquals method to test for equality between the result of the computation you are testing and the expected result.

When you compile the above code in DrJava, it won't compile. You will have to go in and fix the code for Person to make the test code compile.

Step 3.

Fix the code for Person until Test_Person compiles! (See the code below.)

First add a int field called _bMonth and compile. What happens?

Now add the statement return 0; to the body of the method nMonthTillBD(...) and compile.

```
class Person {
   int _bMonth;

   /**
    * Computes the number of months till the next birthday.
    */
   int nMonthTillBD(int currentMonth) {
      return 0;// todo
   }
}
```

Step 4.

After you have cleaned up your code for Person as shown in the above, you should be able to compile Test_Person. With Test_Person open, click on the Test button in DrJava tool bar.

What do you see? Something has failed! The formula for the number of months till the next birth day seems to be the culprit. We will pretend ignorance and fix the "bug" in two steps.

Step 5.1

Change the formula to

```
   int nMonthTillBD(int currentMonth) {
       return _bMonth - currentMonth; // todo
   }
```

Compile all and test again.

Still we have errors.

Step 5.2

Change the formula to

```
   int nMonthTillBD(int currentMonth) {
      return(_bMonth - currentMonth + 12) % 12;// todo
   }
```

Compile all and test again. Now everything should pass! You may now remove the TO DO comment from the code of nMonthTillBD.

In XP programming, only after a method has passed its unit test that you are allowed to proceed to another one.

3. Additional Reading:

Lecture notes on Object-Oriented Programming using Java by Dung X. Nguyen and Stephen B. Wong: http://cnx.org/content/col10213/latest
Here is another link for a more detailed discussion of JUnit testing uisng DrJava: http://cnx.rice.edu/content/m11707/latest/
The following link contains a more elaborate discussion on JUnit testting and a more involved example of testing: http://junit.sourceforge.net/doc/testinfected/testing.htm.