SYSC 2004 Object-Oriented Software Development

Lab 2

Lab 2:

Objective

The objective of this lab is for you to develop and test a Java class using the BlueJ environment. You'll learn how to edit the source code for a class, test it interactively using the object bench and inspector, and run automated test cases.

Getting Started

1. Download file `heater.zip` from cuLearn. Save the file somewhere on your M: drive or in C:/Temp. (The M: drive is recommended, because C:/Temp is erased when you log out.)
2. Right-click on the `heater.zip` folder and select Extract All... to extract all the files into a folder called `heater`.
3. Launch BlueJ (for example, by double-clicking on the BlueJ icon on the desktop or selecting it via the Start icon). Note that double-clicking on any of the files in `figures` or `house` will **not** launch BlueJ.
4. As you complete each part of the lab, get it checked by a TA. **Note: For this and all future labs, as with lab #1, if the TAs are busy, and you have no questions or problems, then feel free to continue working through the lab and get it checked at the end or when the TAs are less busy.**

Developing Class `Heater`

- Part 1:
    1. From the menu bar, select Project > Open Project... An Open Project dialogue box will appear. Browse to the folder where your `heater` project is stored. Select the `heater` folder and click the Open button. A class diagram containing classes `Heater` and `HeaterTest` will appear. For now, you can ignore `HeaterTest`.
    2. You have been provided with a very incomplete implementation of class `Heater`. Double-click on the `Heater` class icon. The Java source for the class will appear in an editor window. To learn what the finished class will model, read the comment at the start of the source code. Two fields, called `temperature` and `increment`, and two constants (`INITIAL_TEMPERATURE` and

DEFAULT_INCREMENT) have been defined for you. There are two constructors, but they don't contain any code. Methods `warmer()`, `cooler()` and `setIncrement()` have empty bodies. Method `temperature()` always returns 0. This isn't very useful, but the method was defined this way so that the class will compile without errors.

3. Begin modifying class `Heater` by adding your name to the list of authors. In the comment at the beginning of the class, edit the line:

   ```
   @author Add your name here.
   ```

   Edit the line:

   ```
   @version 1.02 September 9th, 2012
   ```

   changing the class' version number to 1.03, and the date to today's date.

4. Edit the body of the constructor that has an empty parameter list (the one with signature `public Heater()`) so that it looks like this:

   ```
   public Heater()
   {
       temperature = INITIAL_TEMPERATURE;
       increment = DEFAULT_INCREMENT;
   }
   ```

   The first statement initializes the `temperature` field with the value 15. Notice that statement uses the symbolic constant `INITIAL_TEMPERATURE`, not the "magic number" 15. The second statement initializes the `increment` field to 5. Once again, we use a symbolic constant (`DEFAULT_INCREMENT`), not the magic number 5.

   Do not write any code for the constructor that has two parameters; in other words, the body of the constructor that has the signature:

   ```
   public Heater(int minTemp, int maxTemp)
   ```

   should remain empty. Click the `Compile` button (either the one in the upper left corner of the editor window or the one to the left of the class diagram in the main BlueJ window). If you get any errors, edit the constructor and recompile the class until it compiles without errors.

5. Edit the body of method `temperature()` to return the heater's current temperature setting. *Make sure that you replace the method's comment with one*

that describes what the method does. Compile the class. If you get compile-time errors, edit the method until it compiles without errors.

6. Now you'll interactively test the modified `Heater` class. Right-click on the `Heater` class icon. A pop-up menu will appear. Create a new `Heater` object by selecting new Heater() from the menu. A Create Object dialogue box will appear. Click OK. A red box with rounded corners will appear in the object bench towards the bottom of the BlueJ window. This box represents the newly created `Heater` object whose name (identity) is `heater1`.
7. Right-click on `heater1` (i.e., right-click on the red box the represents the `Heater` object, not the icon that represents the `Heater` class). A pop-up menu will appear, listing the `Heater` object's methods. Select *Inspect* from the pop-up menu. An Object Inspector window will appear. Use the inspector to verify that the constructor you wrote for Step 4 correctly initialized the `temperature` and `increment` fields to 15 and 5, respectively. If necessary, correct the class by editing the constructor, recompiling and retesting.
8. Predict the value that will be returned by `temperature()` when this method is invoked on the newly created `Heater` object. Right-click on `heater1` and interactively invoke `temperature()` by selecting this method from the menu. When the Method Result dialogue box appears, verify that method returns the correct value. If necessary, correct the class by editing `temperature()`, recompiling and retesting.
9. Get Part 1 checked by a TA.

- Part 2
    1. Edit the body of method `warmer()`. (*Remember to edit the method's comment, too!*) This method should increase the heater's temperature by the value stored in field `increment`. Compile the class, then interactively test your method by creating an instance of class `Heater` and opening an inspector on the object. Note the values that are currently stored in the object's fields. Predict how the object's state will change when you invoke `warmer()` on the object. Which field(s) will be assigned new values when the method is executed? What will those values be? Interactively invoke `warmer()` and use the inspector to check your predictions. What value should now be returned by `temperature()`? Test your hypothesis by interactively invoking this method and determining if the method returns the expected value. If necessary, correct the class and retest it before moving to the next step.
    2. Edit the comment and body of method `cooler()`. This method should decrease the heater's temperature by the value stored in field `increment`. Interactively test this method the same way you did in the previous step, except this time you will invoke `cooler()` on the `heater` object.
    3. Get Part 2 checked by a TA.

- Part 3:
    1. Interactive testing of classes can quickly become tedious. Fortunately, a popular *test framework* called JUnit is built into BlueJ. This framework allows us to automate class testing. When you have a test class in your project, you should see the "Run Tests" botton on the left side of the BlueJ window. You can also select "Run Tests" under "Testing" in the "Tools" menu, or you can select one or all tests by right-clicking on the green test class.
    2. Class `HeaterTest` contains a *suite* of *test cases* that test that class `Heater`. Compile both classes by clicking the `Compile` button to the left of the class diagram. Right-click on the `HeaterTest` class icon. Select Test All from the pop-up menu to run all the test cases. (Another way of doing this is to click the Run Tests button to the left of the class diagram.) A Test Results dialogue box will appear, listing the test cases that were executed. The green check-marks to the left of `HeaterTest.testConstructor`, `HeaterTest.testWarmer` and `HeaterTest.testCooler` indicates that these three test cases, which test `Heater()`, `temperature()`, `warmer()` and `cooler()`, passed without problems. The x's to the left of the other test cases indicates that they failed. This shouldn't be a surprise, because they test behaviour you haven't yet implemented.
    3. Now we'll add some "bells and whistles" to `Heater`. Modify your `Heater` class to define two new integer fields called `min` and `max`. The visibility modifier of both of these fields must be `private`. These fields store the minimum and maximum temperature settings of the heater. Modify the body of the constructor `Heater()` (the one you wrote in Step 4 or Part1) to initialize `min` and `max` with the values 0 and 100, respectively. (Remember to define symbolic constants to represent these values). Compile `Heater` and run all the tests. The number of tests that pass and fail shouldn't change; i.e., 3 test cases should pass and there should be 5 failures. If there are problems, test `Heater` class interactively and use the inspector to help you determine what the problem is, then correct the class and run all the tests.
    4. Write the body of the **two-parameter** constructor (the one with signature `public Heater(int minTemp, int maxTemp)`) so that it initializes `min` and `max` with the values of parameters `minTemp` and `maxTemp`, respectively. This constructor should also set the values of `temperature` and `increment` to 15 and 5, respectively (once again, use the symbolic constants `INITIAL_TEMPERATURE` and `DEFAULT_INCREMENT`, not magic numbers). Run all the tests. If your two-parameter constructor is correct, there should now be a green check-mark beside `HeaterTest.testTwoParameterConstructor`. The three tests that previously passed should still pass. (There will also be a green check mark beside `HeaterTest.testZeroAndNegativeIncrement`. Ignore this.) If there are problems, test your `Heater` class interactively and use the inspector to help you determine what the problem is, then correct the class and run all the tests.

5. Get Part 3 checked by a TA.

- Part 4:
    1. Modify `warmer()` so that it leaves the heater's temperature setting unchanged if the temperature would otherwise be raised to a value greater than `max`. For example, suppose the current values of `temperature`, `increment` and `max` are 20, 5, and 23, respectively. Invoking `warmer()` should leave the temperature at 20, because the current temperature plus the increment equals 25, which is greater than the maximum value, 23. Run all the tests. `HeaterTest.testMax` should now pass. If there are problems, test your `Heater` class interactively and use the inspector to help you determine what the problem is (e.g., before and after you execute `warmer()`, what values are stored in the objects fields? Which ones are incorrect?) Repeat this step until `testMax` passes.
    2. Modify `cooler()` so that it leaves the heater's temperature setting unchanged if the temperature would be lowered to a value less than `min`. Run all the tests. `HeaterTest.testMin` should now pass. If there are problems, test your `Heater` class interactively and use the inspector to help you determine what the problem is (e.g., before and after you execute `cooler()`, what values are stored in the objects fields? Which ones are incorrect?) Repeat this step until `testMin` passes.
    3. Write the comment and body for method `setIncrement()`. This method has a single integer parameter, which should assigned to field `increment`. Run all the tests. Make sure that `HeaterTest.testSetIncrement` passes before you move to the next step. By the way, `HeaterTest.testZeroAndNegativeIncrement` should now fail - this will be the next thing you'll correct.
    4. Modify `setIncrement()` so that the `increment` field is modified only if the value of parameter `newIncrement` is positive. If method is passed 0 or a negative value, it should return without modifying the state of the heater object. Run all the tests. If everything is o.k., you should see green check-marks beside all of the test cases. If there are any failures, test your class interactively and use the inspector to help you determine what the problems is. Edit the class and retest it until all the test cases pass.
    5. Get Part 4 checked by a TA.
