**CIS163 Lab 5**

**Exceptions, Polymorphism, and UML lab**

**Preparation:**

Bring your book to lab.

Attended class and stayed up to date with class material

**Objectives** (after completing the lab you will be able to do:)

* Create and handle custom exceptions
* Know the difference between checked and unchecked exceptions
* Create a UML class diagram from code provided.
* Write Skelton code given a UML diagram.
* Apply polymorphic concepts to provided code.

**Activities:**

Part 1:

1. In your StopWatch project, create a new exception “SWIllegalArgumentException” that is a descendant of an IllegalArgumentException
   1. Is the SWIllegalArgumentException a checked or unchecked exception?
2. Replace all IllegalArgumentExceptions thrown in StopWatch.java with SWIllegalArgumentException
   1. At this point, your code should still work without an issue. Do your JUnit tests that check for an IllegalArgumentException still work? Why or why not?
3. Make all methods that throw the SWIllegalArgumentException be checked by adding “throws SWIllegalArgumentException” in the method header
   1. At this point, you will see a bunch of errors come up in your GUI code. You will fix these in the next step.
4. Refactor your GUI code to catch the SWIllegalArgumentExceptions and handle them appropriately. Your GUI should still work as expected when this step is done
   1. As part of this step, be sure to remove all code that throws in exception in any of the GUI classes that is not then caught somewhere else. Remember, this is how you get “red lines” in your console! Depending on when and where you threw your exceptions in your GUI code, you may have to do a little extra refactoring to properly handle this change.
5. If you threw any other errors in your StopWatch class (e.g., NumberFormatException), you can repeat the above steps for some extra exception handling practice

Part 2:

1. Download the Lab5.zip to your local system (I suggest your desktop)
2. Import the file into eclipse, this is tricky so watch the instructor if needed.
   1. Create a package called “lab5”
   2. Right click on the package and click on “Import”
   3. Choose to import from an archive file (typically top option)
   4. Navigate to Lab5.zip
   5. Select to import all files
3. Once you have successfully downloaded/installed/executed the program, follow these steps:

**(PLEASE COMPLETE EACH STEP COMPLETELY BEFORE MOVING TO THE NEXT STEP.)**

1. Create an interface with one abstract method

public interface PayInterface {

public abstract double pay();

}

1. Remove the abstract method from the Staffmember class, and instead implement the PayInterface.
2. Run the program now using an interface. Does it work as expected?
3. Using the debugger, execute the program and use the step into function to watch how the program executes, i.e., step by step through the code. Does it work as expected?
4. In this step, change the code, that is, rewrite the code for StaffMember that would allow you to remove the cast statements (see below) from the Staff class constructor method.

~~((Executive)~~staffList[0]).awardBonus (500.00);

~~((Hourly)~~staffList[3]).addHours (40);

1. For your last change to the Firm.java program. Create a SuperExecutive class that extends Executive. Add an SuperExecutive to the staffList array found in the Staff class constructor as the 7th person. See code below for add a 7th person in the staffist array.

staffList[6] = new SuperExectutive ("Roger", "123 main Street","555-5555");

1. Be sure to create the pay and toString methods inside the SuperExecutive class.
2. Run the program now. Does it work as expected?
3. Let’s say you didn’t know the exact contents of the staffList array in the Staff class constructor. How would you be able to call awardBonus to any Executive or addHours to any Hourly in the staffList array (if they were in the array)? Write the code to accomplish this.
4. Finally, for this lab, create a complete UML diagram of this program.