Case Study

Rowan Pilon

041049454

Due December 8th, 2023

For CST8284

**Task 1 & 2:** 1. Create a Junit test case in eclipse and write down the complete code with the output. 2. Use any three assertion methods and create a Java Programming code to generate the correct outputs.

I decided to do both these tasks in the same class/test class. The class looks like the following:

A screen shot of a computer

Description automatically generated

It is a very simple and basic number calculator-style class, which will use a method, either addition or subtraction that will take two numbers as parameters. These two numbers are stored in class-level variables and then calculated according to the method chosen. The result can either be printed to show the result using the result function, or reset using the reset function. Pictured below is my test class for this calculator program.

A screenshot of a computer program

Description automatically generated

I started by creating the test class and selecting all the methods that I wanted to test from the calculator class. I then made sure all the junit functions I wanted were properly imported and started on creating test methods. For both the calculation methods, I created local variables to represent the parameters that would be used when invoking the actual method. And I used an assertion method to ensure that the class-level variable c, which represents the result of the most recent equation, was equal to the sum of its parts, which in this case is the two local variables, named a and b. Next, for the method that prints the result, I used the assertTrue assertion method to ensure that the c variable is valid when made into an int and subsequently printed. For the final method, the reset method, I used three separate assertSame methods to ensure all three class-level variables were equal to one another, the only way in which this is possible is if they are all equal to zero, which means they have been reset, no methods have yet to be invoked, or a method was invoked with only zeroes in the constructor.

**Task 3: Case Study:** You must develop java applications using suitable coding and output. The following concepts must be included in your project:

Class Object Encapsulation Inheritance Overriding Overloading Polymorphism I/O files and GUI.

For my case study, I decided to create a simulation-style game which rolls two dice and tells you the total. In the next few images, I will capture the source code and then various shots of the output. There will be a short description of the image underneath each one.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generatedThe above image shows the first 38 lines of the rolltwo.java class. Contained in this are the imported utilities, the class declaration and its default constructor declaration, as well as a method named startermethod(int c) which is used for outputting the value of the most recent dice roll total to a file, which in this case is a file called testcase.txt.

This is the second half the file rolltwo.java. It contains the method used for the majority of the program, startermethod(). Startermethod begins by initializing the jpanel, jframe, random, and jbutton objects that will be used within it. The method then performs some actions on the GUI for the button that will be used to allow a user to interact with the gui and roll the dice. ActionListener is then added to the button object, which gives it the ability to execute code when the button is pressed, this method is called actionPerformed. The ActionListener method actionPerformed(ActionEvent e) is overridden in order to suit our needs, it starts by disabling the button, and then sets the two dice variables to random numbers between 1 and 6, and then gets the total using the total variable. Directly after that, a JOptionPane opens up, and displays the total of the dice roll, and then finally, the startermethod(int c) is called, outputting the total that we just obtained, the button is set back to true and the method ends. After that is the main method, which uses SwingUtilities to safely execute an instance of the rolltwo(), which calls its constructor, which invokes its methods. I will now show what the output looks like.

A screenshot of a computer

Description automatically generated

Upon executing rolltwo.java, this window opens. ^

A computer screen with a message

Description automatically generated

When the button is pressed, a popup appears which displays the result of the diceroll.

A screenshot of a computer

Description automatically generated

And here is the text file testcase.txt, located in my eclipse workspace.

That is all the output generated by this program. Pertaining to the requirements that need to be met for this test case, I will do my best to attempt to explain where each one was implemented.

**Class** – the file rolltwo.java is a class, and is used for this test case

**Object** – objects are used all throughout the application, a random object is used to generate the output for the dicerolls, objects are created for every JFrame utility, filewriter uses an object to call its method, and so on and so forth.

**Encapsulation** – the three meaningful global variables are set to private, meaning that they are only accessible within the class of rolltwo, the majority of important methods are private and can only be accessed by and used upon instances of the rolltwo class.

**Inheritance** – rolltwo inherits the JFrame class, which allows it to create objects of the JFrame class, use its methods or overload them.

**Overriding** – everytime ‘@override’ appears, it is an instance of a method being overridden, for the majority of the rolltwo application, these come from an object of the ActionListener class. But there is one in the main method which comes from an instance of the class Runnable, a default Java class which contains the blank method run(), since we added to it, it has been overridden.

**Overloading** – the methods startermethod() and startermethod(int c) are examples of an overloaded method, since it has different parameters. These two methods are presented in rolltwo.java

**Polymorphism** – since there is method overriding and method overloading, which are both instances of polymorphism, it is present within this test case.

**I/O Files**- there is an instance of the FileWriter class which is responsible for writing the output of every button press to a file located outside of the class. This is an instance of an I/O operation.

**GUI** – this class is equipped with an-albeit simple-gui. It consists of a handfully of JFrame objects such as a jbutton, a jframe, and a jpanel.

On top of these mentioned criteria, I feel there may be a few other java principles present in this application, such as the usage of try-catch error handling, usage of the String non-primitive data type, as well as a few other I may of missed.

I forgot to display my imports at the top of the file, as eclipse automatically minimized them. So here is an image of them

A screen shot of a computer program

Description automatically generated