Part 1: Introduction

This software project aims at creating an application that displays an interface with two buttons, each with their own functionality. The first button generates six shapes that can be squares, rectangles, or circles. The second button sorts the six shapes based on their surface and displays the shapes on the interface.

The challenges associated to the software project include innovating an efficient design for the software project, using appropriate sorting algorithms in terms of space and time complexity, and presentation of various design patterns of the software project via UML diagrams.

There are many concepts used throughout the software project. The first one is object-oriented design (OOD). Object-oriented design allows defining software object’s states and behaviours, and the way these objects interact to achieve the desired achievements. In other words, Object-Oriented Design specifies software objects and the way they collaborate to satisfy requirements. OOD utilizes objects which comprise data and methods (features of an object) ,and enables reusability and maintainability being grouped into classes. Another concept used in the software project is Object-Oriented Design Principles (OOD Principles), which consists of abstraction, encapsulation, polymorphism, inheritance. Abstraction prevents all the detailed data and methods of a class to be known by other classes to ease their interactions. Encapsulation keeps an object’s state and methods inside a class, disabling direct interaction with other classes’ objects. Inheritance consists of creating a child and a parent class where the child class can reuse the parent state and methods without changing them. Polymorphism gives the ability of a method to take different shapes (having several methods with the same name but different parameters). Lastly, the concept of design patterns is used in the software project. The design pattern identifies the classes and instances, their roles, the dependencies between them, and the distribution of classes responsibilities. There are 23 design patterns in three categories, creational, structural, and behavioural design patters. Creational design patterns focus on instantiation of objects and allow decoupling a system from the way its objects are instantiated, composed, and represented. The Creational design patterns used in this software project include Singleton and Factory. The Structural design patterns focus on the composition of classes and objects to design larger structures and achieve new functionality. Lastly, the behavioural design pattern focusses on the communication between objects and on the assignment of responsibilities between objects through composition.

This report is structured in parts, following the introduction, the report explains in further detail the design of the software project, the implementation of the software project and a conclusion of the report.

Part 2: Design

Diagram

Description automatically generated

Elements of the UML class diagram consists of 8 classes. The client class, which is the main control unit of the software project calls instantiates the Shape Factory, Merge Sort and Draw classes. It also acts as the main JFrame, with the control buttons in the initial JPanel and the JPanel of shapes from Draw class. The next class in the diagram is the Shape Factory class, which plays the role in instantiating the six random shapes of the Shape object, varying between circle, square and rectangle. The Shape abstract class is the generic class of a shape, it stores the common values and methods of a shape. It’s child classes (circle, rectangle and square) inherit these methods and values from the shape class. The Draw class is the one that extends JPanel to draw the shapes generated from shape factory. Lastly, the Merge Sort class uses merge sort algorithm to sort the Array List of shapes based on their surface through the Comparable interface.

In the design of this software system, I have used Object Oriented Design Principles which include inheritance, abstraction, and encapsulation. Throughout the design, encapsulation has been used almost everywhere, since most classes have their own set of states and methods. It disables the direct interaction among other classes, only assessable through mutator methods. These methods are the ‘get’ methods utilized to get the attributes of the shape. The concept of having child classes like circle, square, and rectangle extend from the shape abstract class also follow the design principle of inheritance. These child classes inherit all the methods and states of the Shape abstract class. The principle of abstraction is demonstrated using shape abstract class. The abstract class allows inner implementation of the class to be hidden from other classes, and it prevents the class from being instantiated.

Part 3: Implementation

The algorithm used in sorting the shapes is the Merge Sort algorithm. This algorithm is preferred as it takes O(nLogn) time complexity with n inputs. It’s a famous algorithm which follows the divide and conquer method. It recursively splits the list or array into half until it cannot be split anymore. Then it merges the elements based on its order into a new list. In the case of this project, the algorithm merges the shapes into a new list according to their surface area, compared using the comparable interface.   
I chose to implement the first class diagram. First, I started off with creating the Shapes, which is implementing the Shape abstract class and its child classes. Following the inheritance, I was able to make a ArrayList of the type Shape in the Shape Factory class. Since the sub class of shapes all share common states and methods, I was able to group hem into Shape and compare them using the comparable interface. Second, I implemented the Shape Factory which loops through the number of shapes required for the project which is six, and in each iteration the class instantiates a random shape with random values and computes its coordinates based on previous shapes in the list. It also generates random values of red, blue, and green for its color parameters. The Shape Factory also has a method which returns the generated shapes to the main client class. This method uses deep copy to return a list with new shape objects. Next, I’ve implemented the Client class which contains the JFrame unit, along with buttons and JPanel. The initial JPanel acts as the empty interface with two buttons, once the load button is selected, the client class calls the Shape Factory class and displays the shapes using the Draw class. The draw class uses the method paint component inherited from the JFrame and determines the type of shape to use the right fill method. Lastly, the merge sort class uses merge sort to sort the list of Shapes and resets the shapes coordinates according to their order in the array list. This prevents the shapes from clustering when displaying the shapes. The merge sort uses the comparable interface of the shape to compare surface areas of shapes and align them in order.

The tools used during the implementation is Eclipse IDE for Eclipse Committers version 2020-06.

Snapshot of the interface after clicking on ‘Load Shapes’

Graphical user interface

Description automatically generated with medium confidence

Part 4: Conclusion

Adapting the merge sort algorithm as a sorting technique class to sort the list of shapes, while using the comparable interface went well overall. In addition, the instantiation of the shapes and the implementation of the shape factory class went well.

Making the buttons was the more challenging task of the project. Unlike JavaScript, there is no onClick method, and had to learn the use of Action listener.

I have learned the use of java swing library in java and some of Java’s GUI.

Top recommendations would be providing resources for tools such as JButton, JFrame, etc. as they were not covered well in the lecture, going into further detail of other design patterns as not many were discussed.