# EECS 3311 B - Lab 03
# Design Report

Amir Mohamad - 217 387 762

TA - Alireza Naeiji

Oct 8, 2021

## Contents

# 1 Introduction

`SHAPESORT` is a Java program that simply sorts a set of shapes based on their surface area using concepts of Object Oriented Design (OOD) such as polymorphism, inheritence, encapsulation, and abstraction. During this project I didn't really face many challanges. The only issue that I faced was with using the Java `awt` package when drawing the shapes onto a `JPanel`. This was easily overcome in a few minutes by reading the Java API documentation. In this design document, we will cover the design of this system with resect to the front end and the backend. We will overview the general design and then analyze the specific implementation.

# 2 System Design

## 2.1 Overview

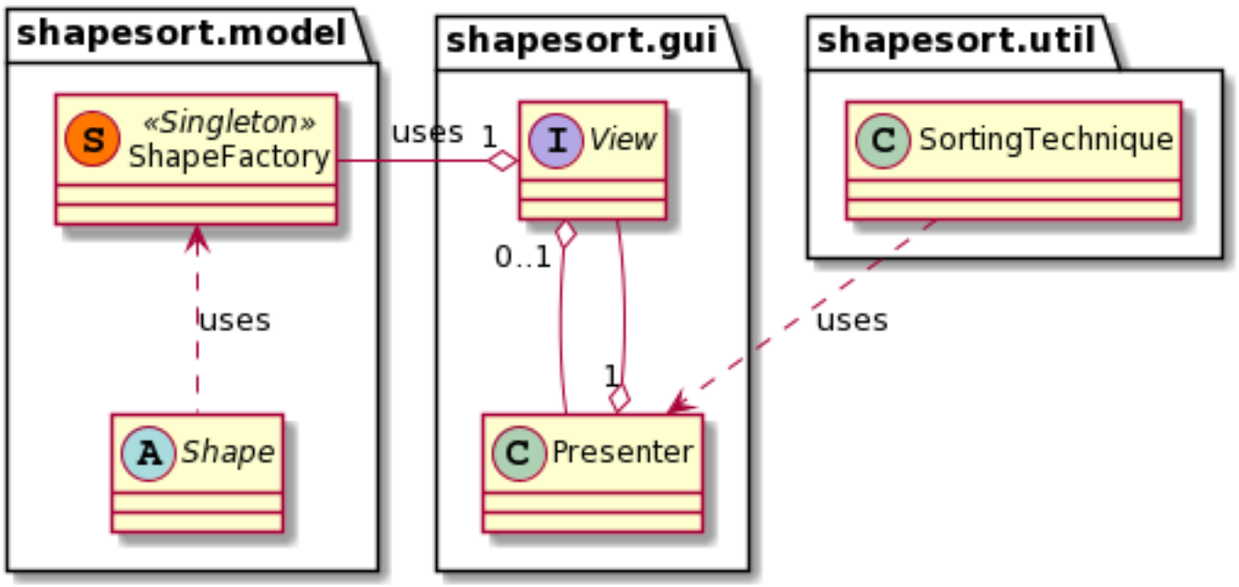


Figure 1: The general system design of `SHAPESORT`

# 3 Front end

## 3.1 Overview

## 3.2 Class Diagrams

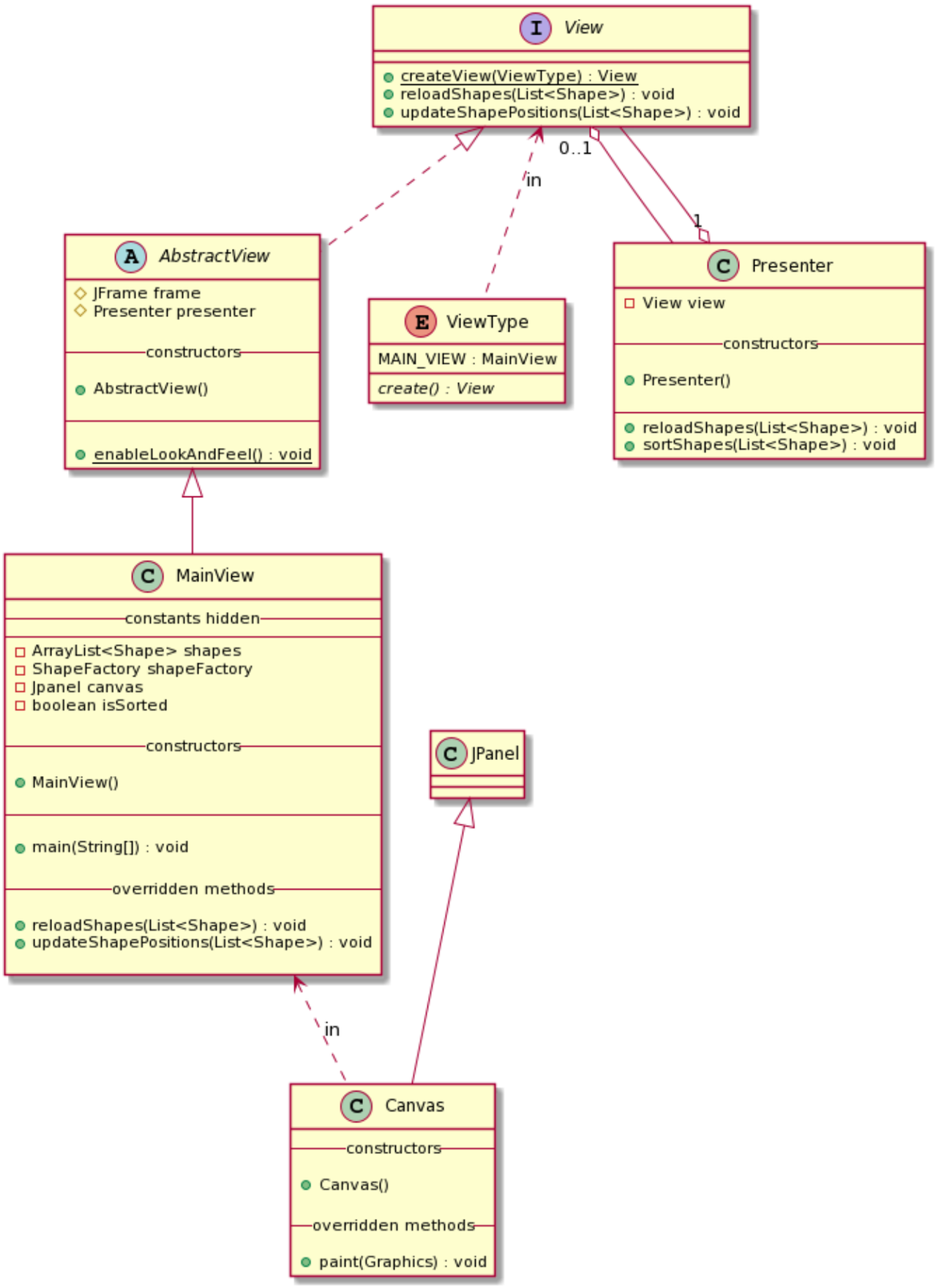


Figure 2: The class diagram of SHAPESORT's front end model

#### 3.2.0.1 View

#### 3.2.0.2 AbstractView

#### 3.2.0.3 Presenter

#### 3.2.0.4 MainView

#### 3.2.0.5 ViewType

#### 3.2.0.6 Canvas

# 4 Back end

## 4.1 Overview

## 4.2 Class Diagrams

### 4.2.1 Shape Model

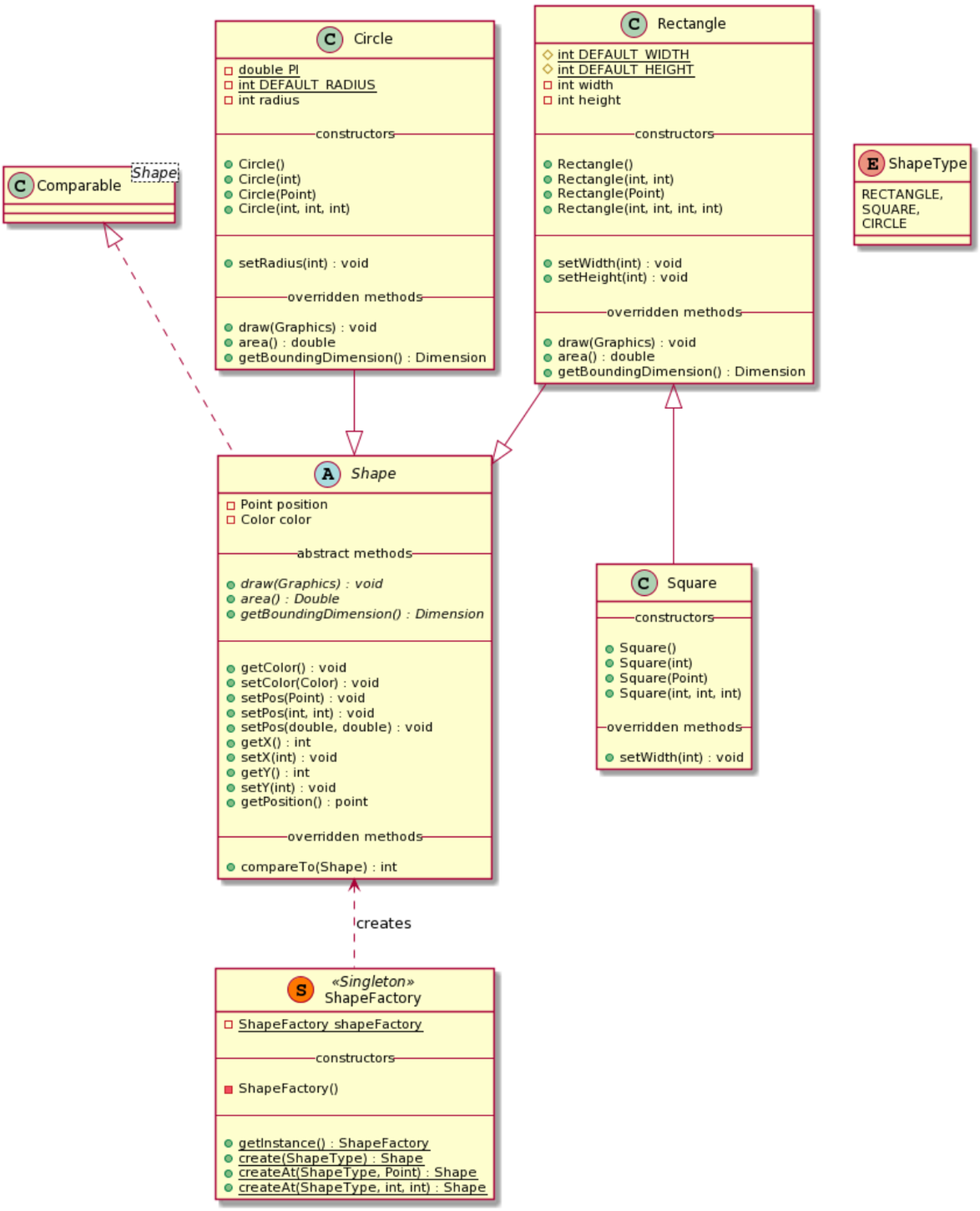


Figure 3: The class diagram of SHAPESORT's shape model

#### 4.2.1.1 Shape

#### 4.2.1.2 Rectangle

#### 4.2.1.3 Square

#### 4.2.1.4 Circle

#### 4.2.1.5 ShapeType

#### 4.2.2 Utility

##### 4.2.2.1 ColorUtility

##### 4.2.2.2 SwingUtility

##### 4.2.2.3 SortingTechnique

# 5 Alternative Design

to-do

# 6 Implementation

## 6.1 Sorting Algorithm

The algorithm used to sort the shapes based on the surface area is Quicksort.

### 6.1.1 Pseudocode Code

```
1  fun quickSort(Object[] shapes, int left, int right)
2      if (left >= right)
3          return
4
5      Shape pivot = (Shape) shapes[(left + right) / 2]
6      int index = partition(shapes, left, right, pivot)
7      quickSort(shapes, left, index - 1)
8      quickSort(shapes, index, right)
9
10
11 fun partition(Object[] shapes, int left, int right, Shape pivot)
12     while (left <= right)
13         while (((Shape) shapes[left]).compareTo(pivot) < 0)
14             left++
15
16         while (((Shape) shapes[right]).compareTo(pivot) > 0)
17             right--
18
19         if (left <= right)
20             if (((Shape) shapes[right]).area() != ((Shape) shapes[left]).area())
21                 swap(left, right, shapes)
22             left++
23             right--
24
25     return left
26
27
28 fun swap(int left, int right, Object[] shapes)
29     Shape leftShape = (Shape) shapes[left]
30     Shape rightShape = (Shape) shapes[right]
31     // swap shape positions also
32
33     shapes[left] = shapes[right]
34     shapes[right] = leftShape
```

## 6.2 Execution

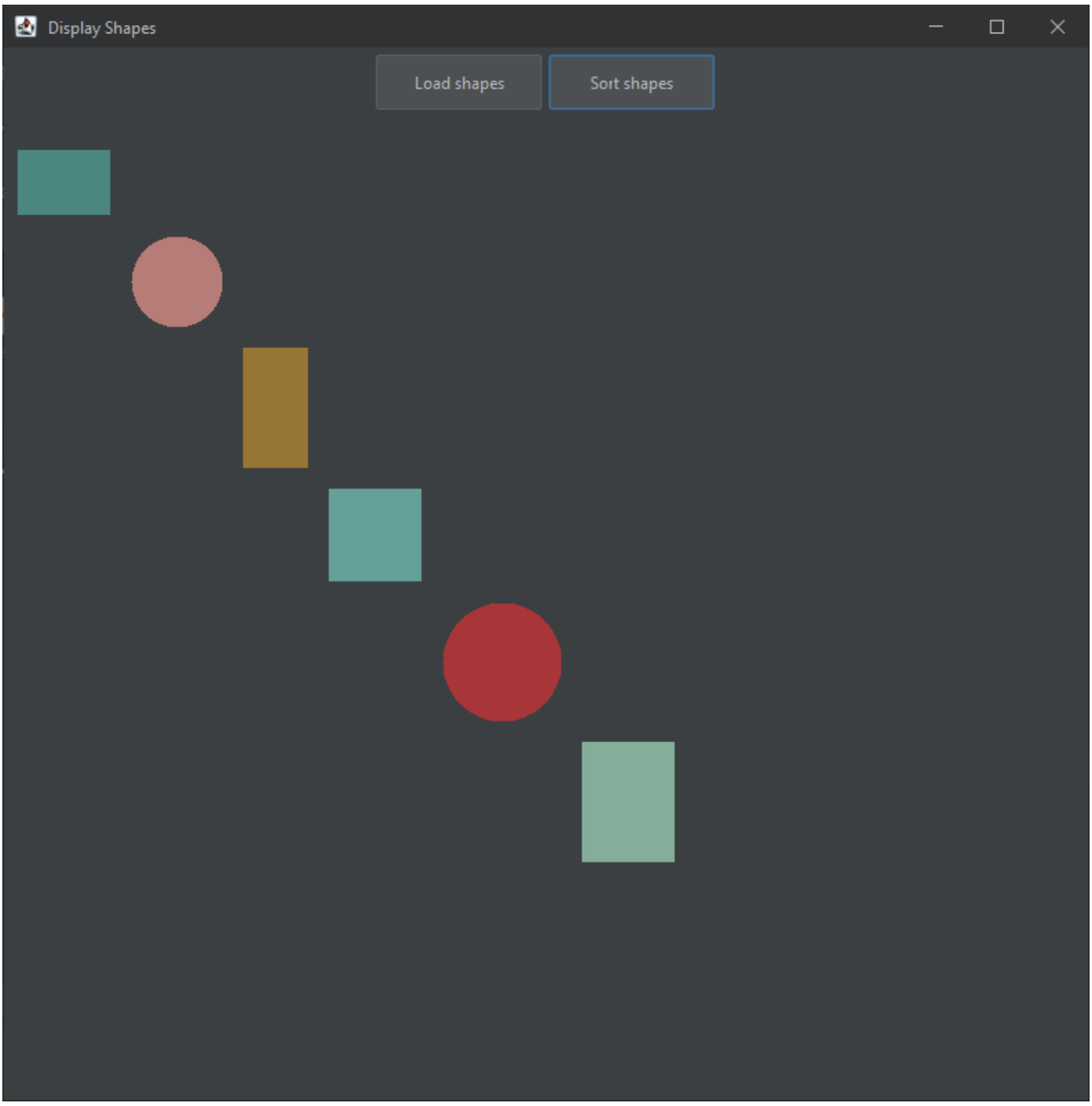


Figure 4: `SHAPESORT`'s main interface

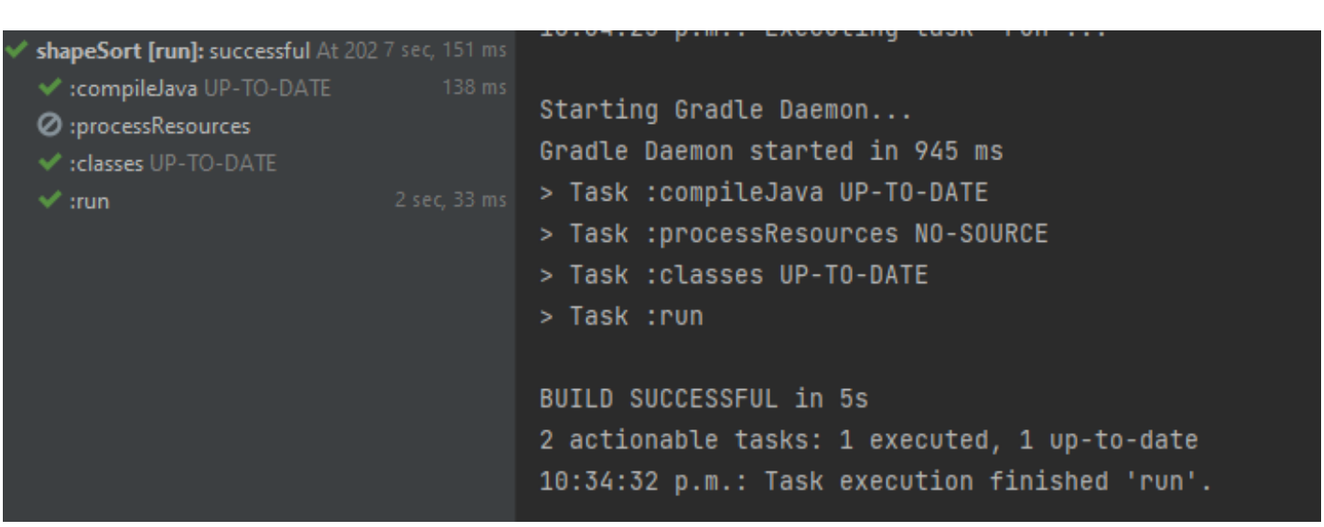


Figure 5: The success execution status from the `gradle` run task

# 7 Tools

## 7.1 Source Code

IntelliJ to implement Java classes and vscode (LaTeX) for documentation

## 7.2 Build

Gradle (to manage dependencies) and Java16

# 8 Conclusion

This project was successful because I followed through with the four step design process to complete `SHAPESORT`. Overall, this software project was successful. The only thing that might have been percived as going wrong is not understanding how to use some classes from the Java `awt` package. During this project I learned how to implement a project using several design patterns. The recommendations I would give someone to help complete this project effectively is to learn how to read Java's API documentation. This is curutial to help understand how to use certain classes to achieve the desired functionality.