

OCTOBER 27, 2021

**GTU Department of Computer Engineering
CSE443 Object Oriented Analysis and Design
Fall 2021 - Homework 1 Report**

**Akif Kartal
171044098**

1 Problem Definition

The problem is to implement a **2D side-scrolling video game** with the help of strategy and decorator design patterns.

2 Solution

The homework was finished **fully** as expected in homework pdf file.

2.1 Game Loop

In order to make a game with java, there must be a game loop. In state of the art, there are more than one way to create a game loop. But must preferred one is game loop using **thread**.

2.1.1 Creating a Thread

```
1  /**
2   * This is the game screen to put things on.
3   * Also, it works as separate thread to create a game loop.
4   */
5   public class MainJPanel extends JPanel implements Runnable, KeyListener {
6       ...
7   }
8
9   //create thread
10  public class MainWindow extends JFrame {
11
12      private MainJPanel contentPanel; //Game area
13      private Thread gameThread;
14
15      public void createThread() {
16          try {
17
18              gameThread = new Thread(contentPanel);
19          } catch (Exception e) {
20              e.printStackTrace();
21          }
22      }
23
24
25      public void startThread() {
26          gameThread.start();
27      }
28
29      ...
30
31  }
```

2.1.2 General Structure with Threads

In this game, playing area was implemented by using second thread. General game window structure is following

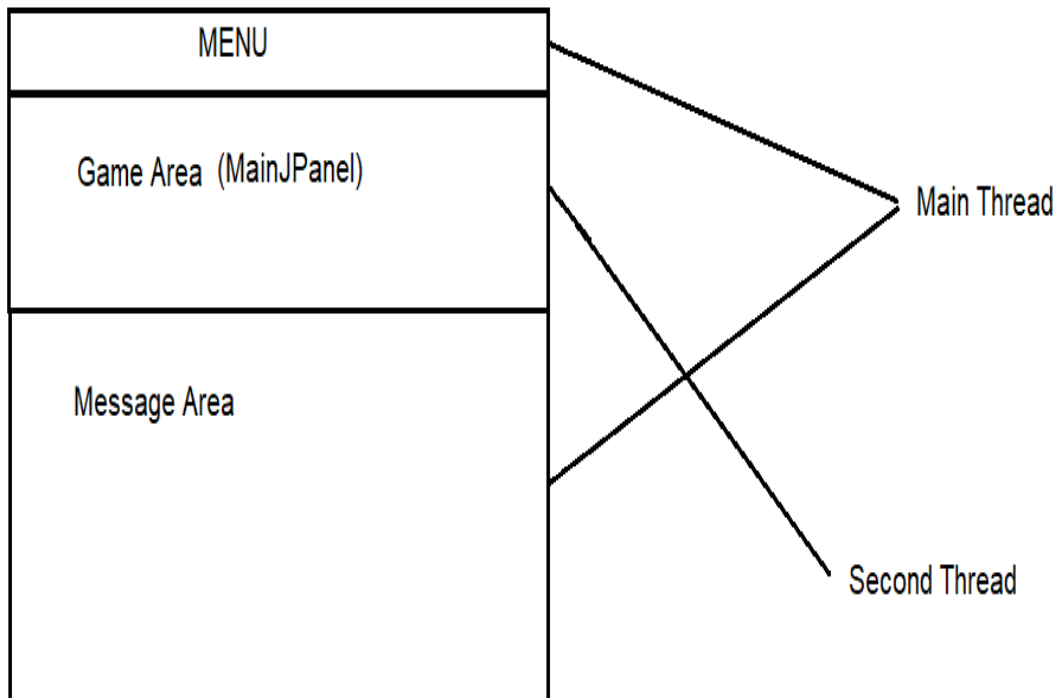


Figure 1: Game Window Structure

2.1.3 FPS

One of the challenging part setting of the fps. In order to set FPS default to 60, I have used "System.nanoTime()" function in java. Because;

1 second = 1.000.000.000 nanoseconds.

Then we will divide nano time with 60;

$1.000.000.000 / 60 = 16.666.666,66$ nanoseconds. = 0.01666 second = 60 FPS

2.1.4 Game Loop with 60 FPS

```

1  // MainJPanel class override run method
2  @Override
3  public void run() {
4      //FPS = 60
5      double fpsMS = 1000000000 / FPS;
6      double deltaTime = 0;
7      long lastTime = System.nanoTime();
8      long currentTime;
9      long counter = 0;
10     int numberOfDraw = 0;
11
12     //game will continue until you exit
13     while (true) {
14         currentTime = System.nanoTime();
15         deltaTime += (currentTime - lastTime) / fpsMS;
16         counter += (currentTime - lastTime);
17         lastTime = currentTime;
18
19         if (deltaTime >= 1) {

```

```

20         //update
21         //repaint
22         deltaTime--;
23         numberOfDraw++;
24     }
25
26     if (counter >= 1000000000) {
27         currentFPS = numberOfDraw;
28         numberOfDraw = 0;
29         counter = 0;
30     }
31 }
32 }

```

2.2 Drawing Components

In order to draw things on screen I used **Graphics class** in java. For example;

```

1  /**
2   * Paint main character(circle) on screen using Graphics object.
3   * @param g Graphics object
4   */
5  public void draw(Graphics g) {
6
7      Graphics2D g2 = (Graphics2D) g;
8      g2.setStroke(new BasicStroke(3f));
9      g2.setColor(Color.decode("#f84545")); // red color
10     g2.fill(new Ellipse2D.Double(150, cor.getyStart(), 30, 30));
11 }

```

2.3 Creating Animation

In order to create simple animation which is background that moves, while the character always remains at a fixed spot on the screen, I draw **colored rectangles** and combine them to create a simple road. Then, I just change **x position** of them and with the **repaint()** function they create simple animation. For example;

```

1  /**
2   * Update position of road
3   * Stones are simple rectangles
4   * */
5  public void updateRoad(int distance) {
6      for (RoadStone stone : stoneList) {
7          stone.setX(stone.getX() - distance);
8      }
9      RoadStone stone2 = stoneList.get(0);
10     if (stone2.getX() + 50 < 0) {
11         stone2.setX(stoneList.get(stoneList.size() - 1).getX() + 50);
12         stoneList.add(stone2);
13         stoneList.remove(0);
14     }
15 }

```

2.3.1 Implemented Animation



Figure 2: Picture from Game

2.4 Design Patterns

This game was implemented by using strategy and decorator design patterns.

2.4.1 Strategy Design Pattern

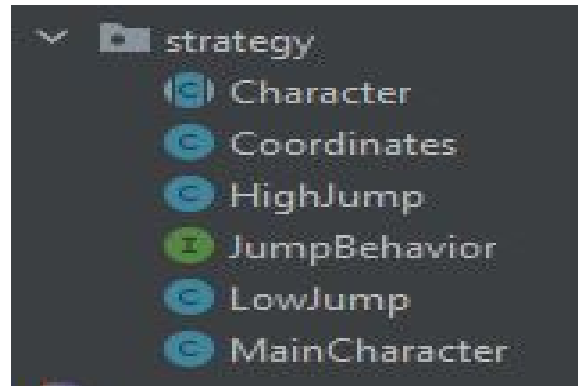


Figure 3: Classes in Strategy Design Pattern

Character is abstract class and JumpBehavior is an interface.

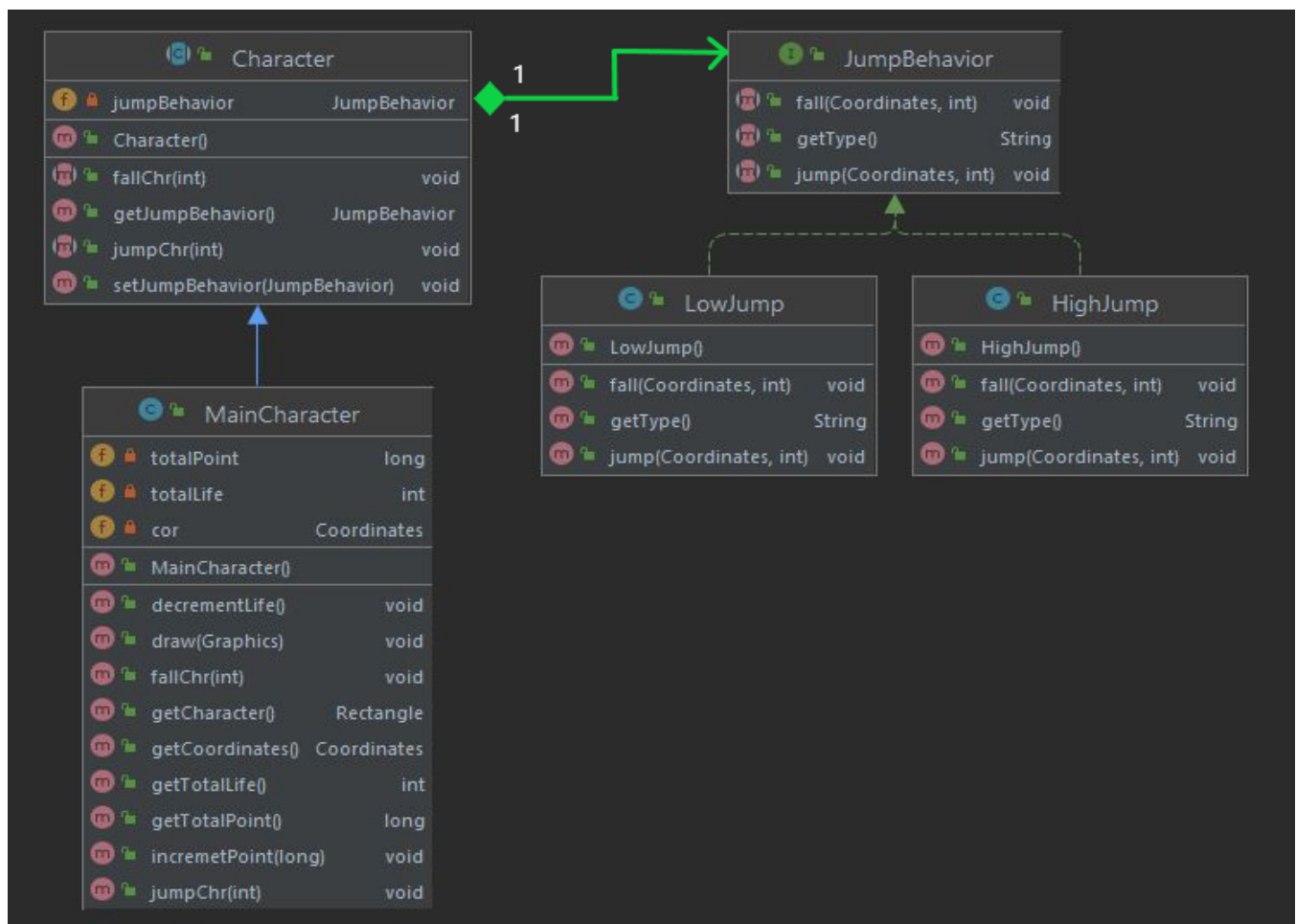


Figure 4: Simple Class Diagram for Strategy Design Pattern

2.4.2 Decorator Design Pattern

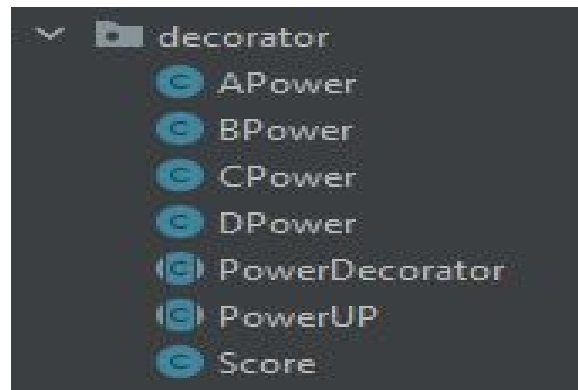


Figure 5: Classes in Decorator Design Pattern

PowerUP and PowerDecorator are abstract classes.

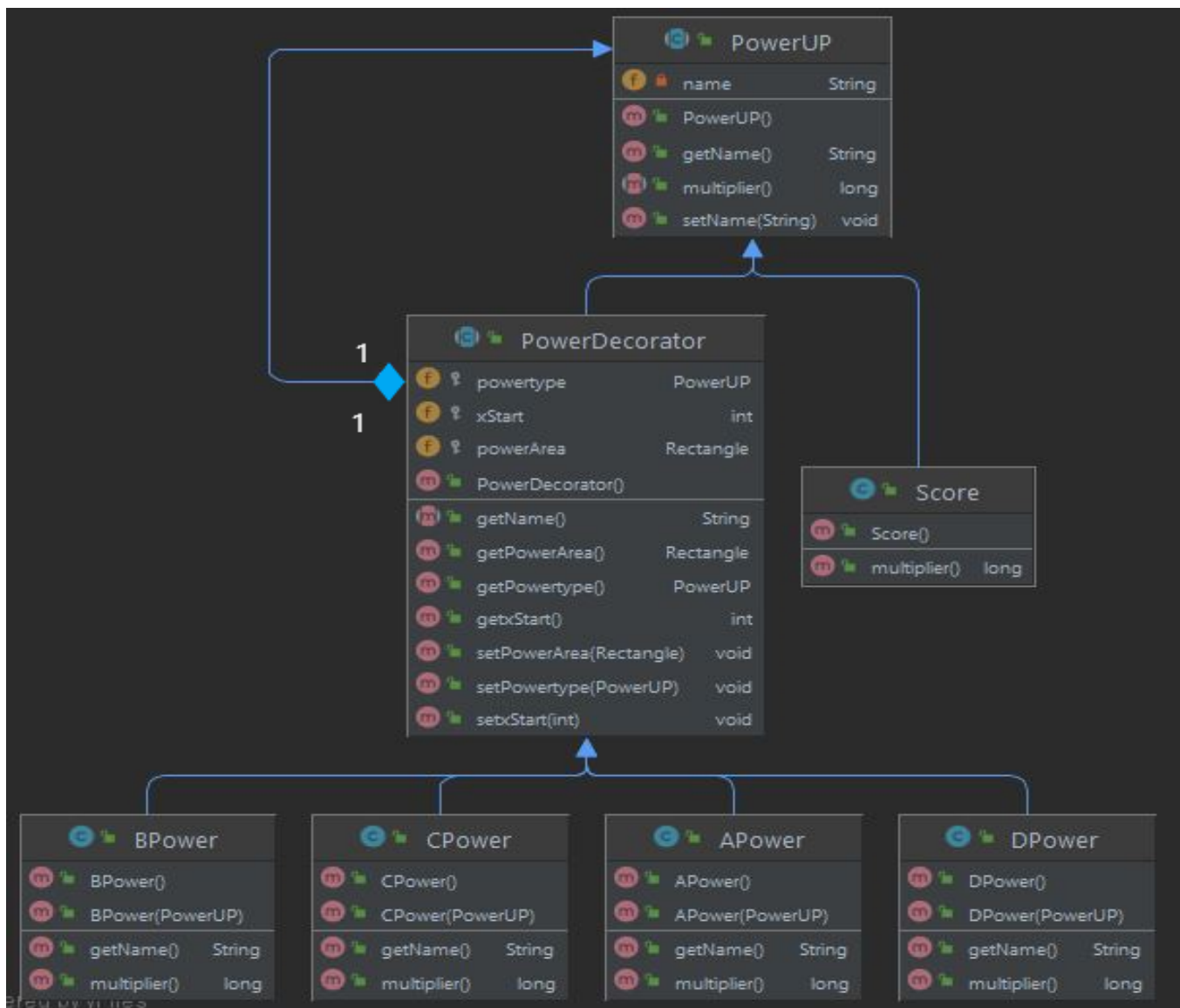


Figure 6: Simple Class Diagram for Decorator Design Pattern

3 Class Diagrams

In order to see full class diagram check class diagrams folder.

4 Java version

Project default java version is 17. Therefore, executable(jar file) may not be working, if java 17 is not installed.

5 References that was used

- ▶ Head First Design Patterns, 2nd Edition.
- ▶ Online sources to learn java gui.
- ▶ Eclipse editor to implement java gui.
- ▶ IntelliJ IDEA editor to create class diagrams.