October 23, 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

```
    /***
    * This is the game screen to put things on.
    * Also, it works as separate thread to create a game loop.
    */
    public class MainJPanel extends JPanel implements Runnable, KeyListener {
        ...
    }

    //create thread
    public class MainWindow extends JFrame {

            private MainJPanel contentPanel;
            private Thread gameThread;

            public void createThread() {
                try {

                    gameThread = new Thread(contentPanel);
                } catch (Exception e) {
                 e.printStackTrace();
                }
            }


            public void startThread() {
                gameThread.start();
            }

    ...

    }
```

2.1.2 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

2.1.3 Game Loop with 60 FPS

```
    // MainJPanel class override run method
    @Override
    public void run() {
        //FPS = 60
        double fpsMS = 1000000000 / FPS;
        double deltaTime = 0;
        long lastTime = System.nanoTime();
        long currentTime;
        long counter = 0;
        int numberOfDraw = 0;

        //game will continue until you exit
        while (true) {
            currentTime = System.nanoTime();
            deltaTime += (currentTime - lastTime) / fpsMS;
            counter += (currentTime - lastTime);
            lastTime = currentTime;

            if (deltaTime >= 1) {
                //update
                //repaint
                deltaTime--;
                numberOfDraw++;
            }

            if (counter >= 1000000000) {
                currentFPS = numberOfDraw;
                numberOfDraw = 0;
                counter = 0;
            }
        }
    }
```

2.2 Drawing Components

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

```
    /***
     * Paint main character(circle) on screen using Graphics object.
     * @param g Graphics object
     */
    public void draw(Graphics g) {

        Graphics2D g2 = (Graphics2D) g;
        g2.setStroke(new BasicStroke(3f));
        g2.setColor(Color.decode("#f84545")); // red color
        g2.fill(new Ellipse2D.Double(150, cor.getyStart(), 30, 30));
    }
```

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;

```java
/**
 * Update position of road
 * Stones are simple rectangles
 * */
public void updateRoad(int distance) {
    for (RoadStone stone : stoneList) {
        stone.setX(stone.getX() - distance);
    }
    RoadStone stone2 = stoneList.get(0);
    if (stone2.getX() + 50 < 0) {
        stone2.setX(stoneList.get(stoneList.size() - 1).getX() + 50);
        stoneList.add(stone2);
        stoneList.remove(0);
    }
}
```

2.3.1 Implemented Animation

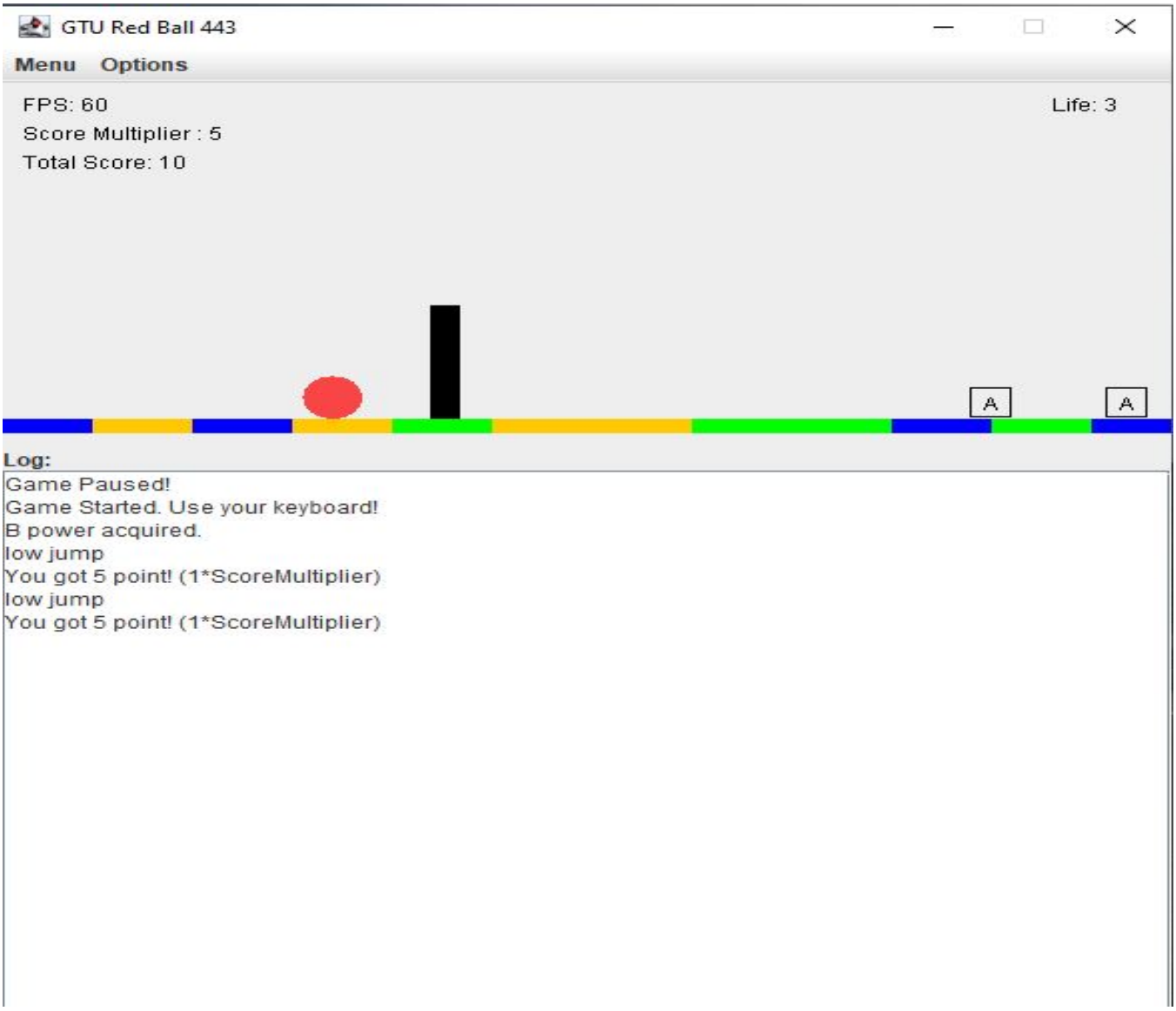

Figure 1: Picture from Game

2.4 Design Patterns

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

2.4.1 Strategy Design Pattern

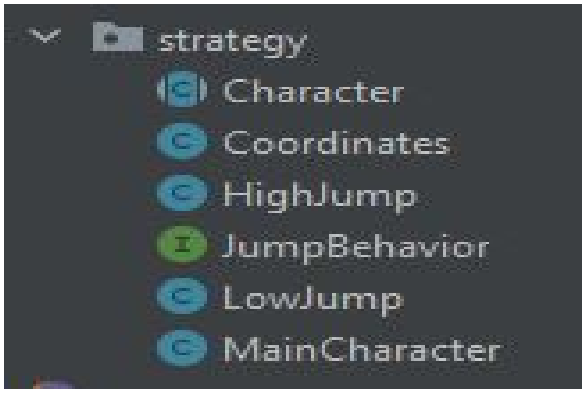

Figure 2: Classes in Strategy Design Pattern

Character is abstract class and JumpBehavior is an interface.

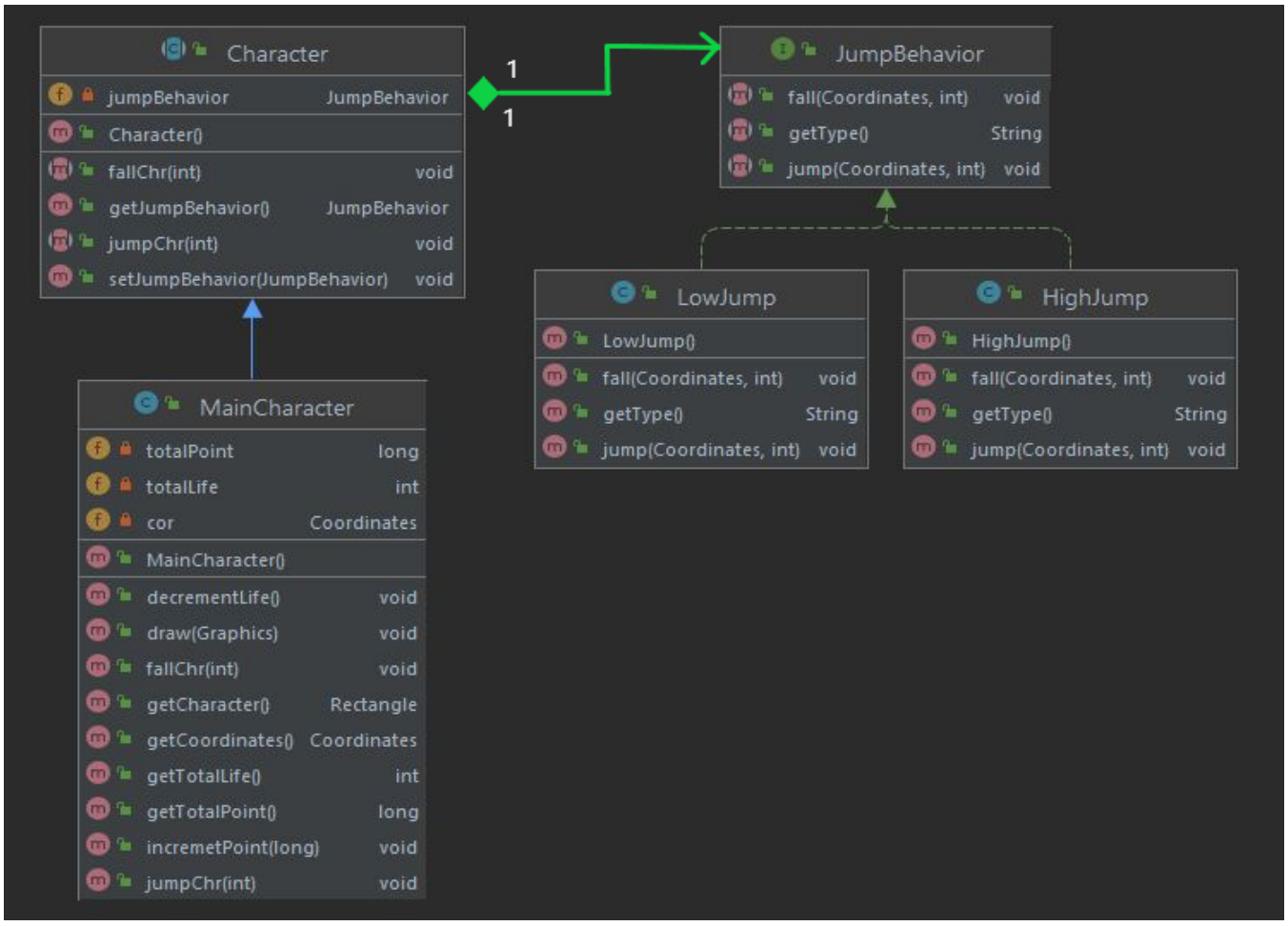

Figure 3: Simple Class Diagram for Strategy Design Pattern

2.4.2 Decorator Design Pattern

Figure 4: Classes in Decorator Design Pattern

PowerUP and PowerDecorator are abstract classes.

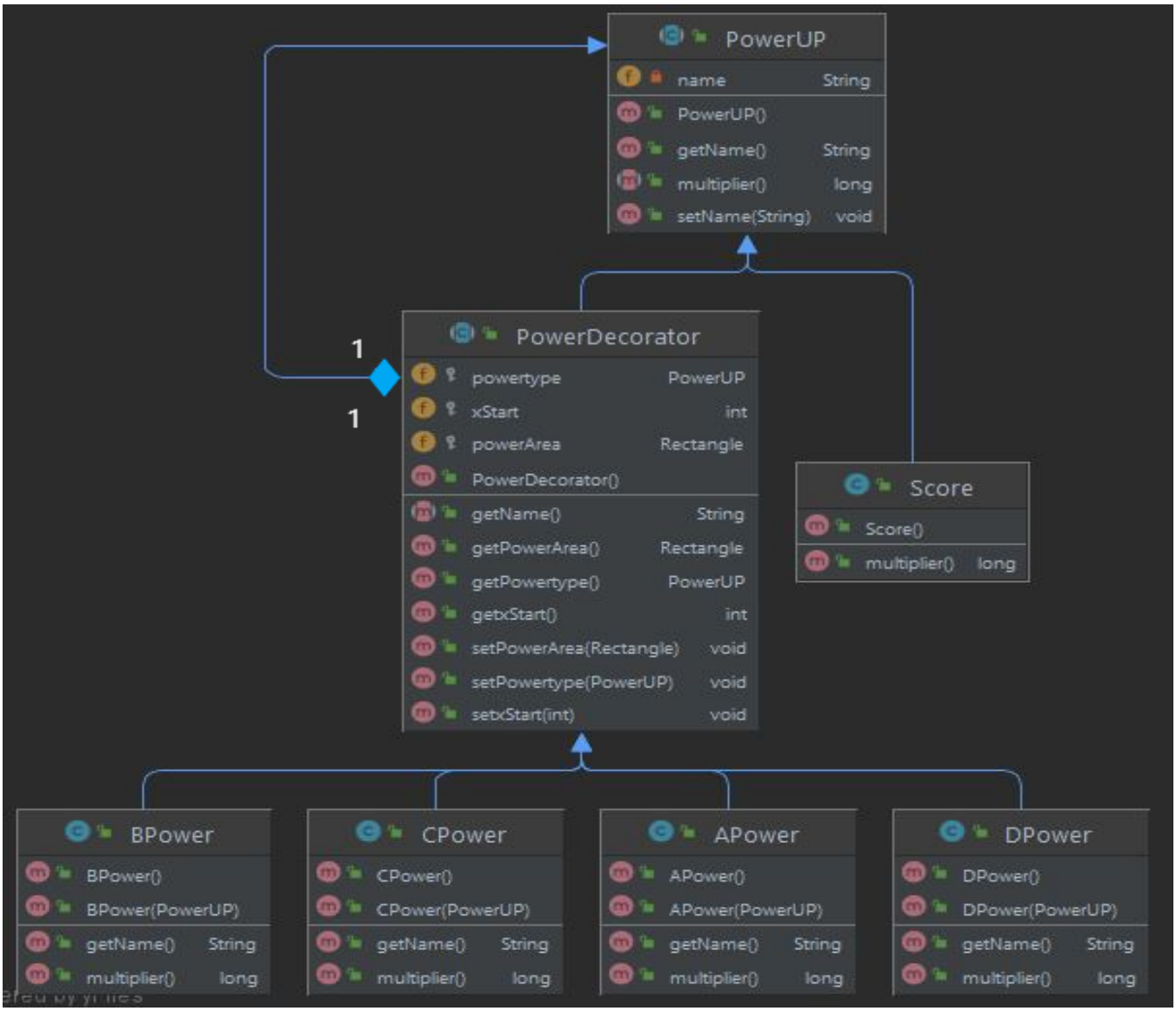

Figure 5: Simple Class Diagram for Decorator Design Pattern

3 Class Diagrams

In order to see class diagrams check class diagrams folder.

4 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.