



OOP

Circus of Plates | Game

11.12.2018

Team

Ahmed Mohamed Abdelhameed Elzeny 06

Ahmed Abdallah Waheeb 04

Abdelfattah Mohamed 24

Abdelrahman Kamal 23

Ahmed Elsayed Molahez 03

Introduction

This project was assigned programming 2 (OOP) course by prof. Dr Khaled Nagi on Wednesday, December 12th, 2018 due on Monday, December 24th, 2018

The Report is delivered to the teaching assistants: Eng. Shehab K. Elsemmany, Eng. Paula B.Bassily.

Overview

A single player-game in which a clown carries two stacks of plates, Dices or Cups and there are a set of colored shapes queues that fall and he tries to catch them, if he manages to collect three consecutive shapes of the same color, then they are vanished and his score increases. The game has Three levels/difficulties which varies the speed of the falling shapes, the score you get when you successfully land 3 shapes of the same color, and the time given to the User.

Extra Features

- Extra Design Patterns (more than the 10 required)
- Realistic game-play (as much as possible)
- Extra game features (making it more interesting)
- Sound interaction with game events

Design Patterns

1. Singleton

We use it in many places where one condition is needed which is that u need only one instance of this class only to be made so whenever u ask for an instance u get the same object of this class such as in the pool and the sound and the logger.

```
15
16
17 private MovingPool() {
18 }
19
20
21 public static MovingPool getInstance() {
22     if (objectPool == null) {
23         objectPool = new MovingPool();
24     }
25     return objectPool;
```

```
15
16
17 private Sound() {
18 }
19
20 public static Sound getInstance() {
21     return sound;
22 }
23
24 public void stopSound() {
```

```
11
12 public class GameLogger {
13     private static GameLogger instance;
14
15     public static GameLogger getInstance(){
16
17         if(GameLogger.instance==null){
18             GameLogger.instance = new GameLogger();
19         }
20
21         return GameLogger.instance;
22     }
23 }
24
25
```

2. Factory

We use the factory in many places too,, when we deal with a reference from the interface and we have many options to instantiate this option to (many classes that implement the same interface)

We deal with that interface till a point that we decide what object of this interface we need so we assign it to this reference. Such as in the strategy chooser when the user chooses the game mode or the shape (image) chooser when the load image process starts.

```

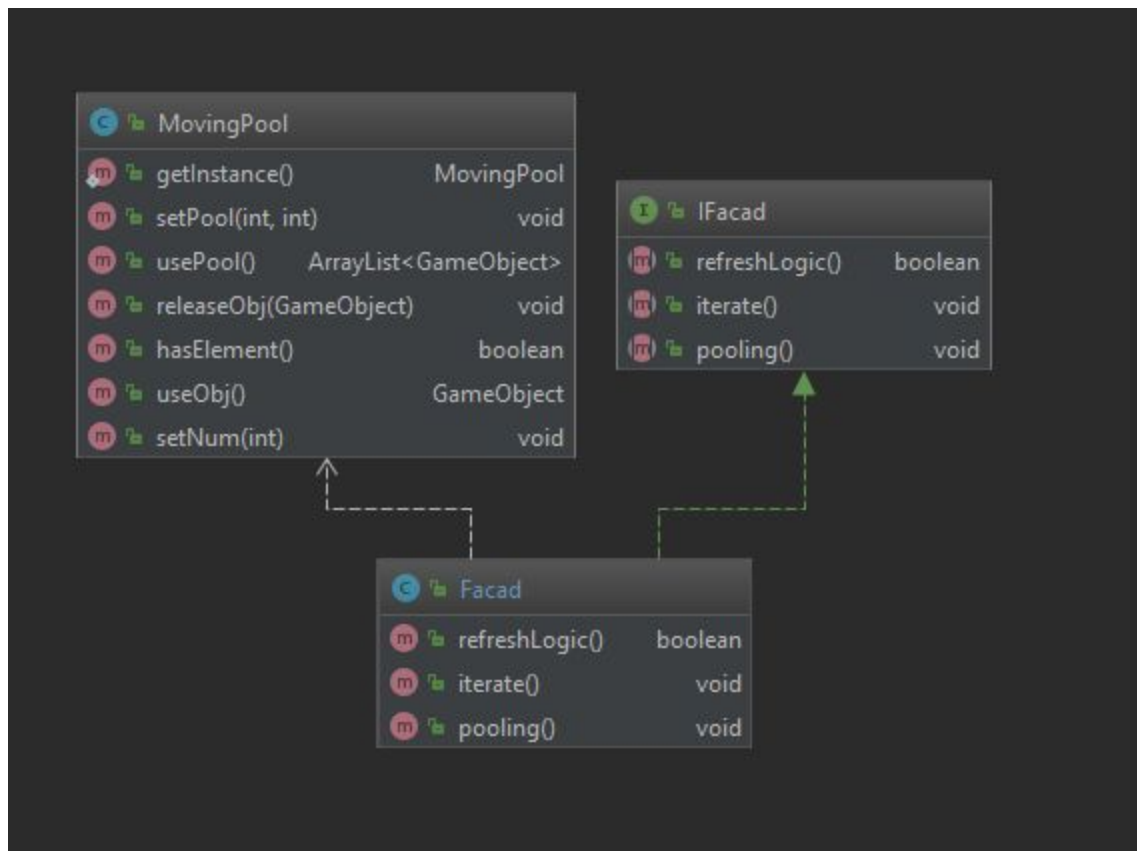
50 if (num1 == 1) {
51     if(classMap.get(color+"Plate")==null){
52         try {
53             classMap.put(color+"Plate", listofClasses.get(2).getConstructor(String.class).newInstance(color));
54             shape = (IShape) listofClasses.get(2).getConstructor(String.class).newInstance(color);
55         } catch (InstantiationException | IllegalAccessException | IllegalArgumentException
56                | InvocationTargetException | NoSuchMethodException | SecurityException e) {
57             // TODO Auto-generated catch block
58             e.printStackTrace();
59         }
60     }else{shape = (IShape) classMap.get(color+"Plate"); }
61 } else if (num1 == 2) {
62     if(classMap.get(color+"Cup")==null) {
63         try {
64             classMap.put(color+"Cup", listofClasses.get(0).getConstructor(String.class).newInstance(color));
65             shape = (IShape) listofClasses.get(0).getConstructor(String.class).newInstance(color);
66         } catch (InstantiationException | IllegalAccessException | IllegalArgumentException
67                | InvocationTargetException | NoSuchMethodException | SecurityException e) {
68             // TODO Auto-generated catch block
69             e.printStackTrace();
70         }
71     }else{shape = (IShape) classMap.get(color+"Cup"); }
72 } else {
73     if (classMap.get(color + "Dice") == null) {
74         try {
75             classMap.put(color+"Dice", listofClasses.get(6).getConstructor(String.class).newInstance(color));
76             shape = (IShape) listofClasses.get(6).getConstructor(String.class).newInstance(color);
77         } catch (InstantiationException | IllegalAccessException | IllegalArgumentException
78                | InvocationTargetException | NoSuchMethodException | SecurityException e) {
79             // TODO Auto-generated catch block
80             e.printStackTrace();
81         }
82     }else{shape = (IShape) classMap.get(color+"Dice"); }
83 }
84 return shape;
85 }

```

```
public void setStrategy(int x){  
    if(x==1){  
        strategy=new easy();  
    }else if(x==2){  
        strategy = new normal();  
    }else if(x==3){  
        strategy = new hard();  
    }  
}
```

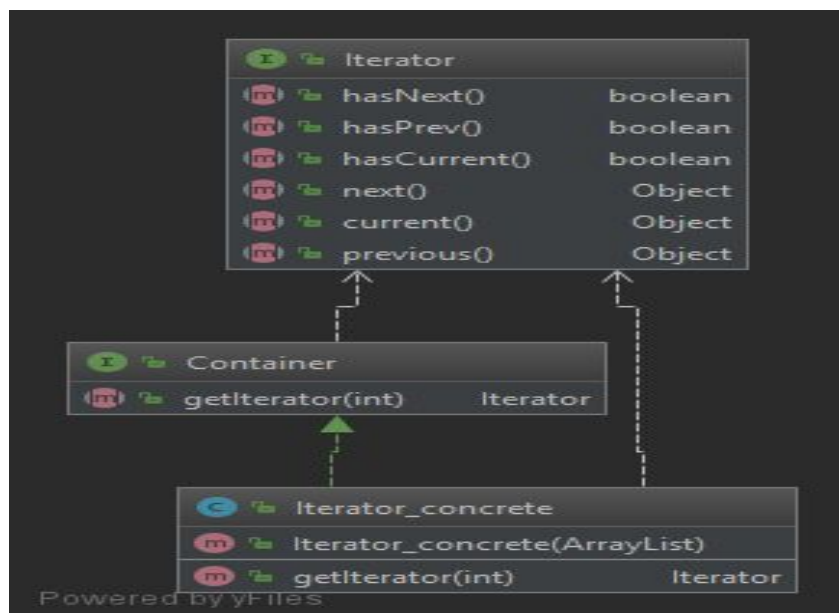
3. Pool

We use the pool design pattern when there is a pool of resources (objects) that can be used under certain conditions (like time limit and max number and so on) its purpose is to limit the use of the memory and don't waste it ,, so when you use an object u remove it from the pool as it's no longer available for others to use it and when u finish using it u release that object and return it back to the pool so it can be used again,, here our pool is the moving objects so we can use objects from it to set it in the movings and whenever the user captures it's still being used only when the user collects 3 shapes or drop a shape this shape is returned to the pool so it can be reused



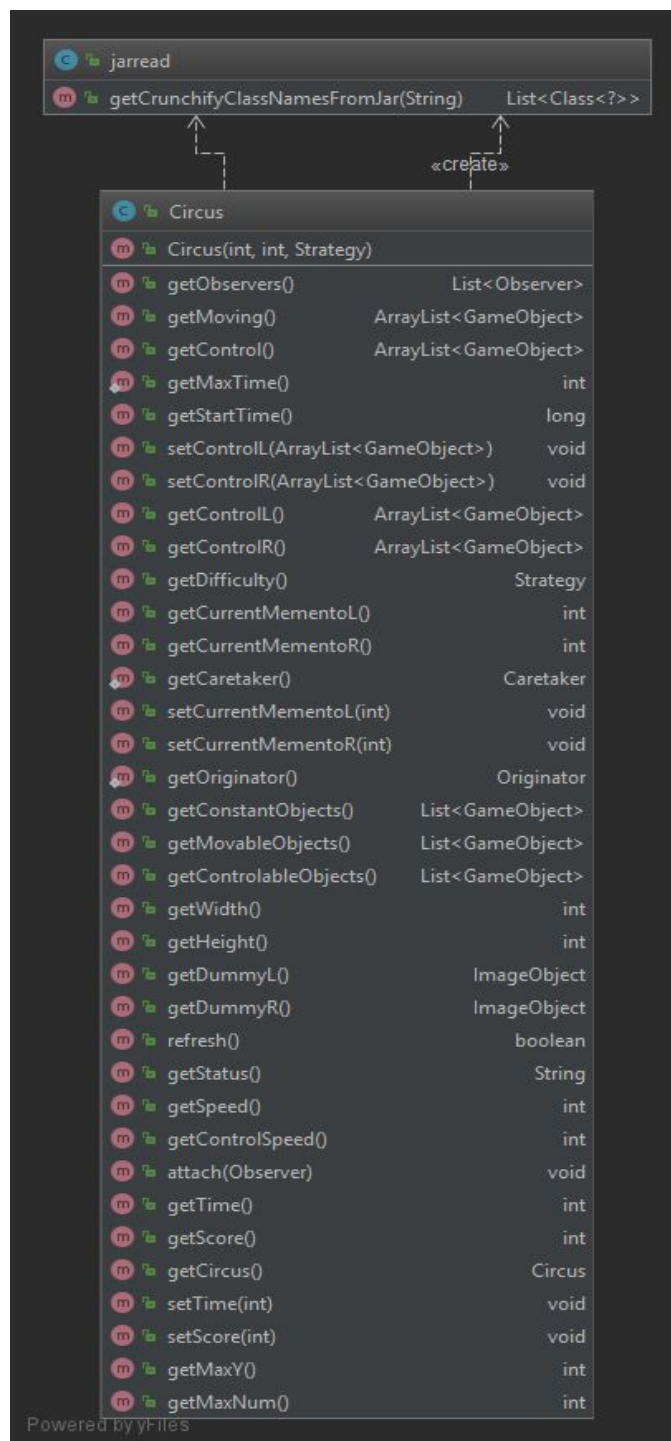
4. Iterator,

1. It's used to iterate on any array list used in project as the concrete class is based on array lists



5. Dynamic Linkage

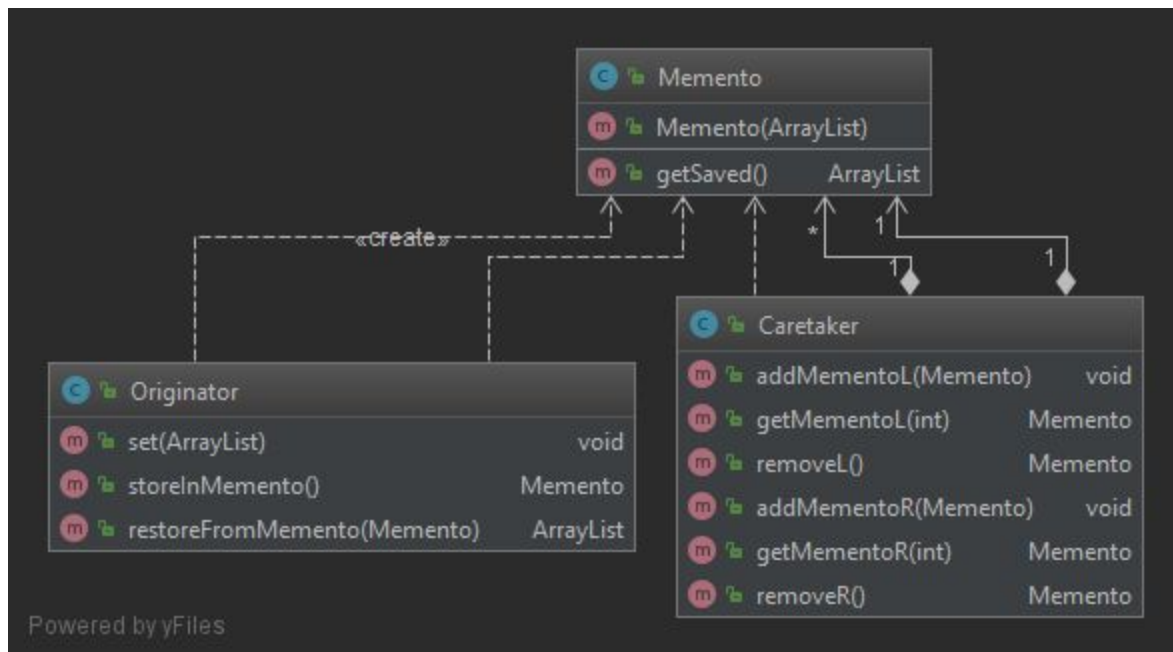
- 1) Load all image object of the game dynamically before the game is begun.
- 2) Load all image from resources folder from jar file.



6. Snapshot

1) Save state of right and left stack at every intersection with last plate in stack.

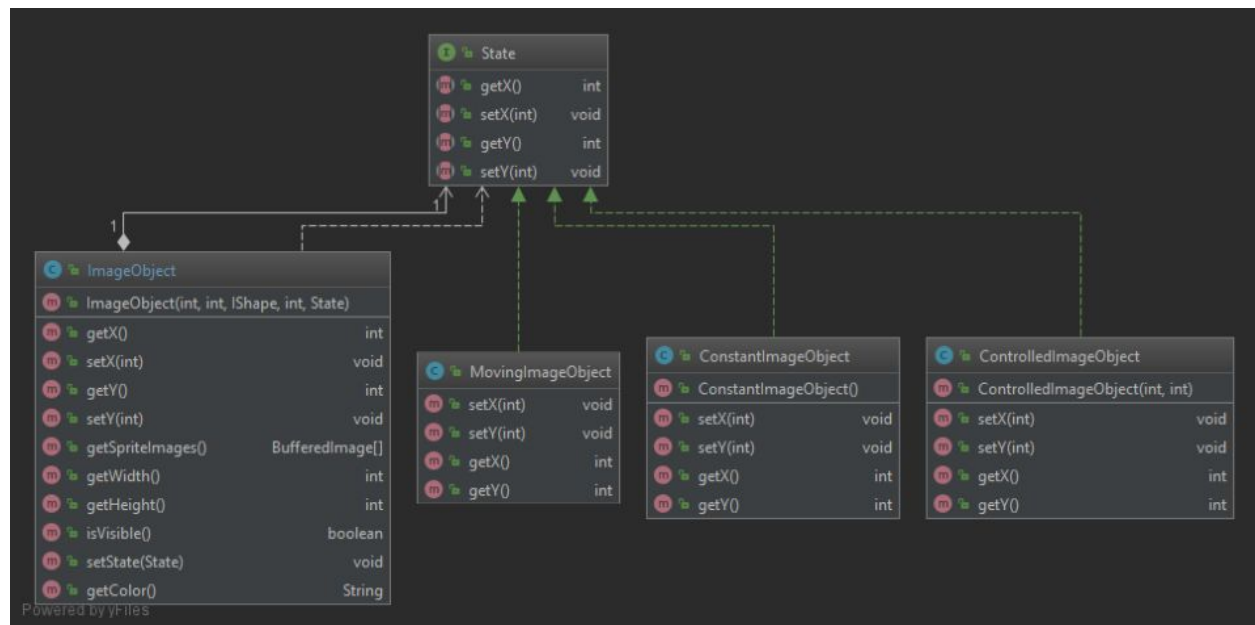
2) Remove last three states if there are three shapes consecutive.



7. State

We use the state design pattern to decide the state of the image (constant or moving or controlled) it's also used to change the state of the shape

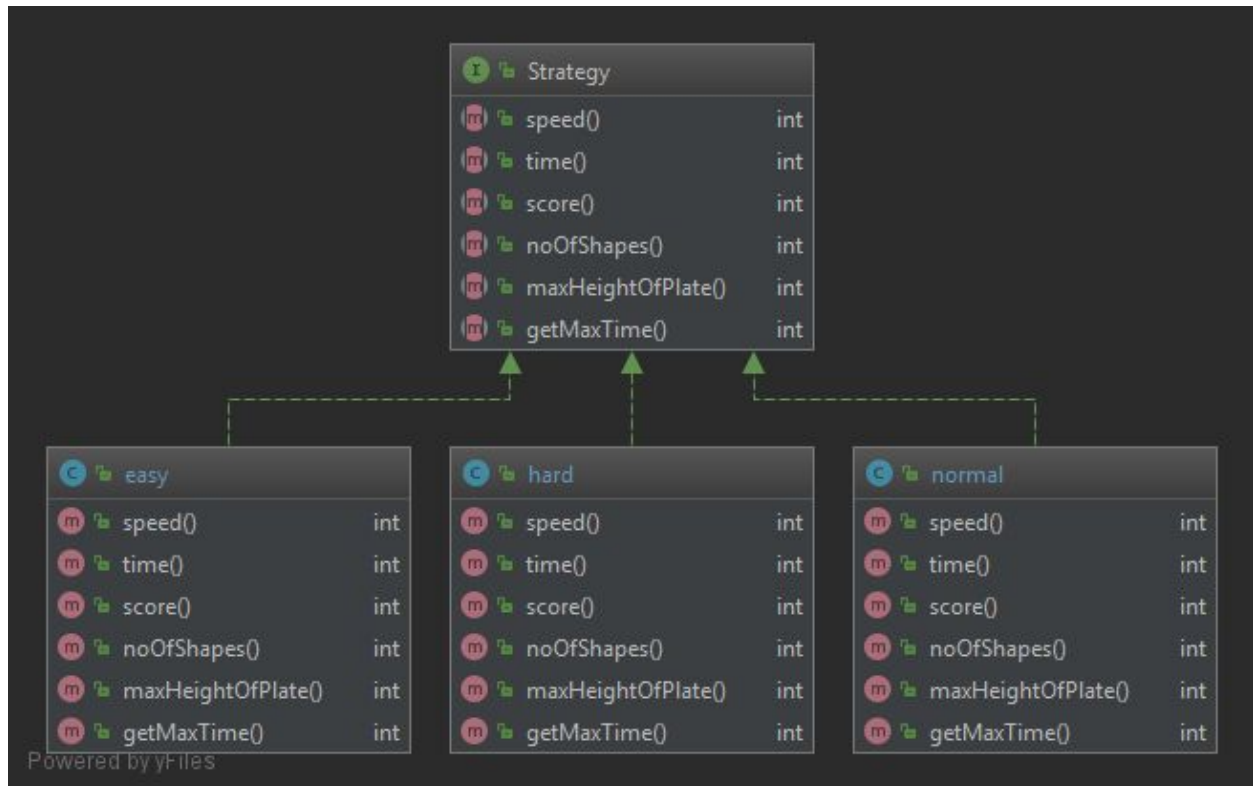
from state to state when the user catches shape(moving to controlled) or when he collects 3 shapes of the same color(control to moving) when a shape fall off (control to moving)and so on



8. Strategy

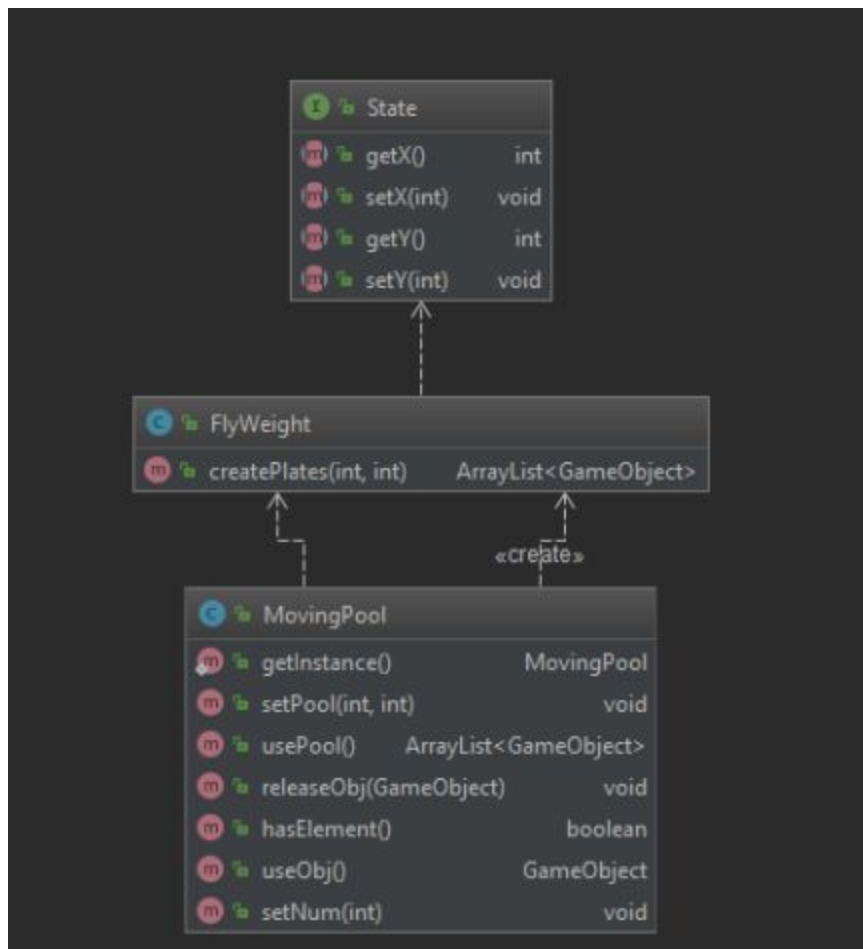
- 1) the game speed and the control speed are increased
- 2) the max time is reduced

- 3)the number of shapes falling is increased
- 4)the score given when he collects 3 objects is reduced
- 5)the bonus time given when he collects 3 objects is reduced
- 6)the max height the plates can reach before he lose is reduced

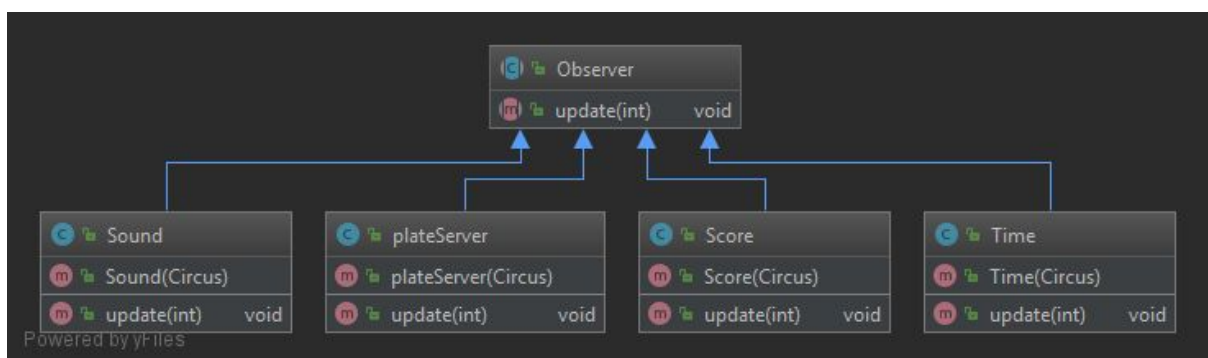


9. Flyweight

The flyweight pattern is used to reduce the number of loaded falling objects in the start of the game, it also randomizes the falling Shapes colors.



10. Observer,

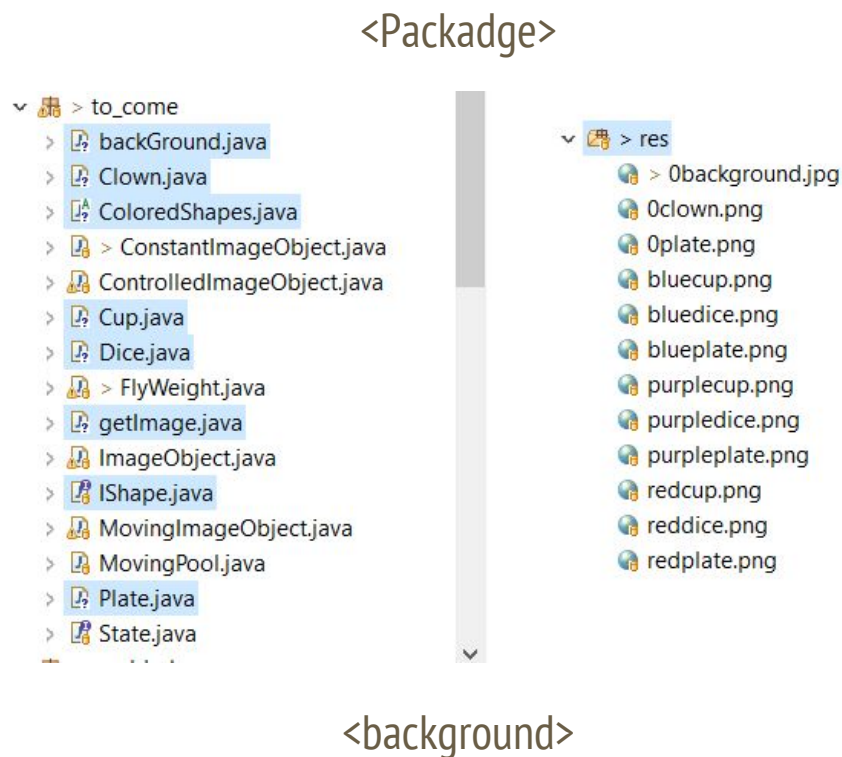


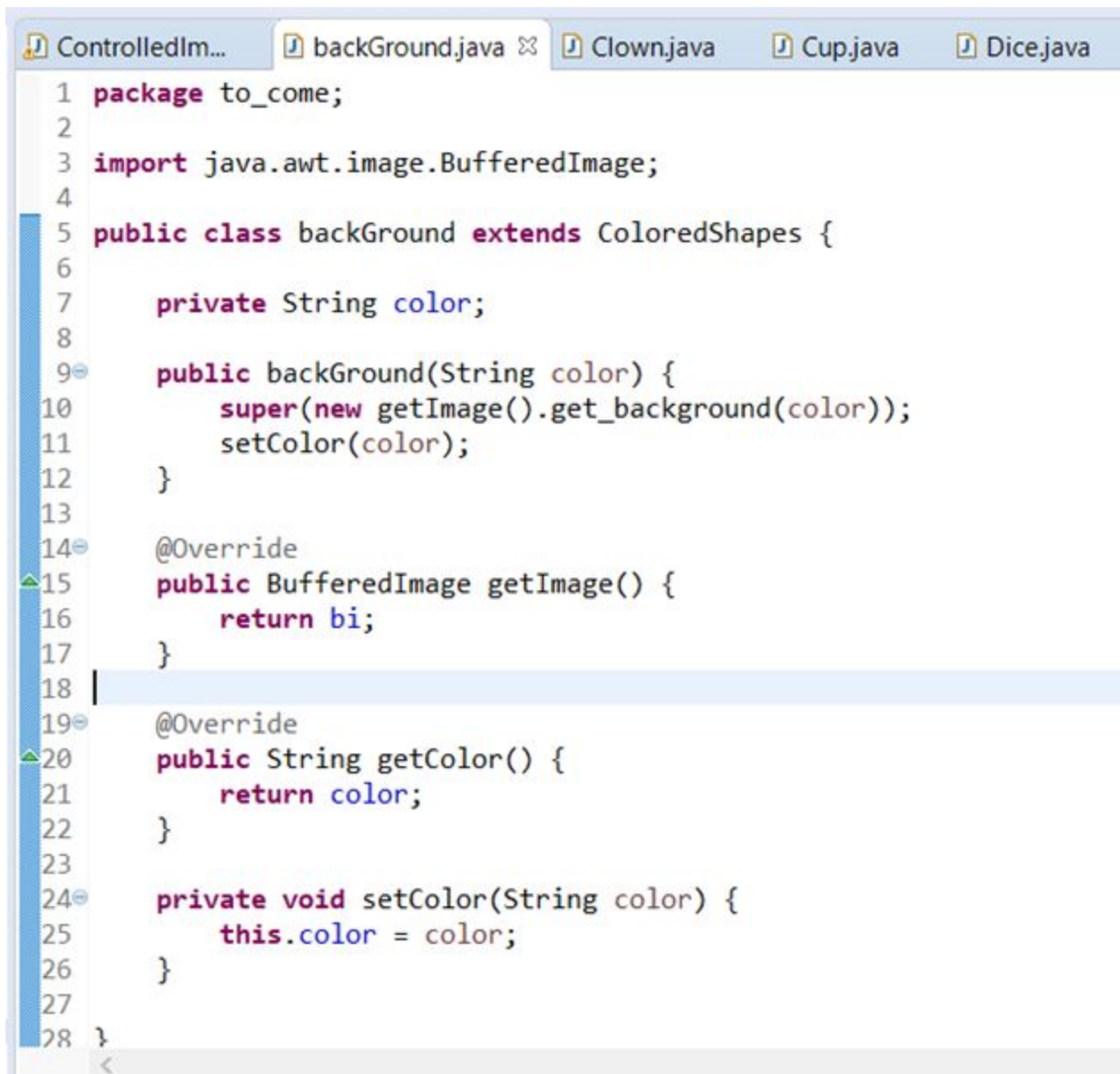
- 1) Use this design pattern to notify time, score, sound and removing plates from the stack.

- 2) This notifies occurrence when the gamer collects three shapes in the same color consecutive.

11. Decorator

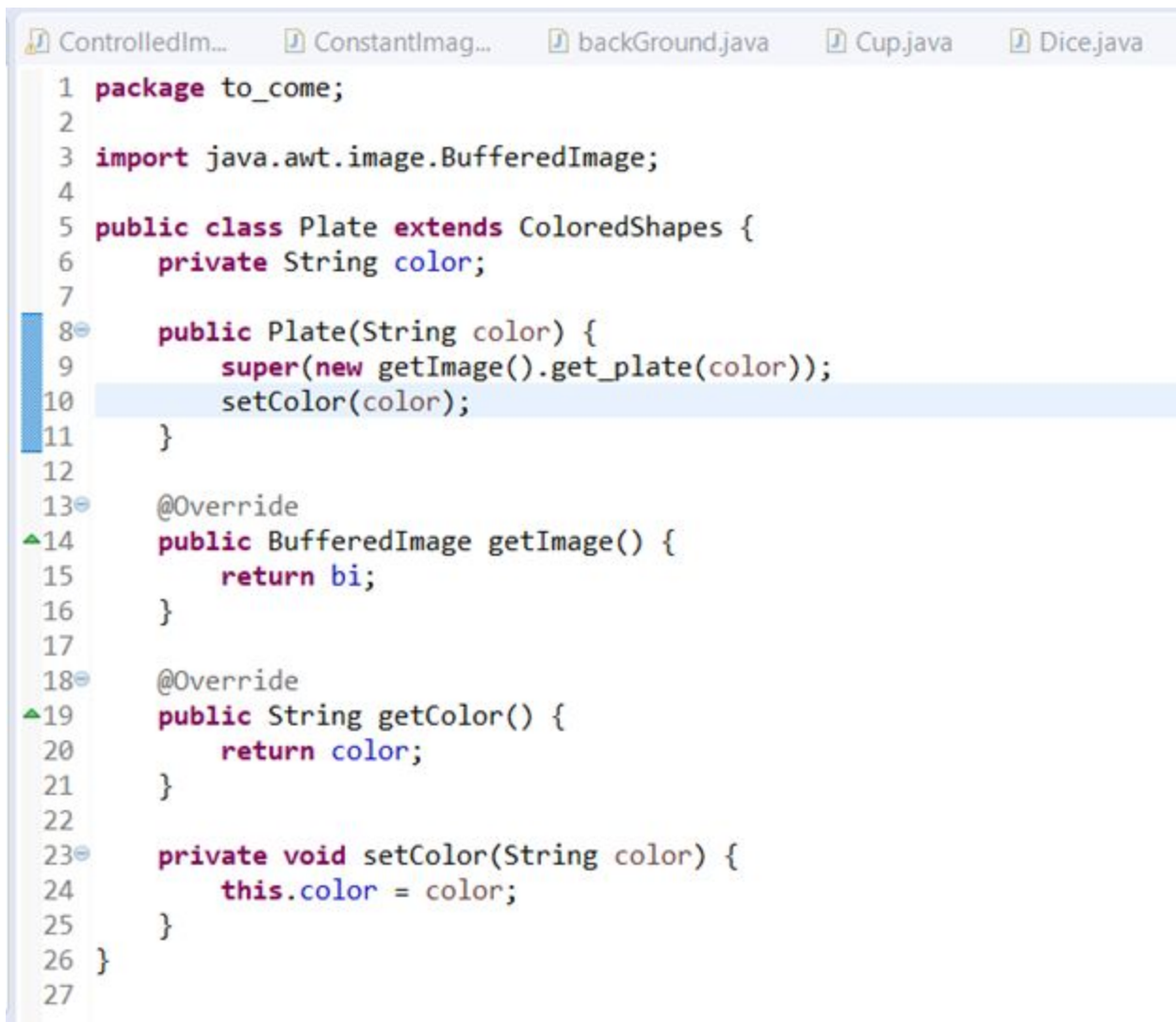
- 1) To make colored shapes in the game.





```
1 package to_come;
2
3 import java.awt.image.BufferedImage;
4
5 public class backGround extends ColoredShapes {
6
7     private String color;
8
9     public backGround(String color) {
10         super(new getImage().get_background(color));
11         setColor(color);
12     }
13
14     @Override
15     public BufferedImage getImage() {
16         return bi;
17     }
18
19     @Override
20     public String getColor() {
21         return color;
22     }
23
24     private void setColor(String color) {
25         this.color = color;
26     }
27
28 }
```

<Plate>



The screenshot shows a Java IDE with a tab bar at the top containing five files: ControlledIm..., ConstantImag..., backGround.java, Cup.java, and Dice.java. The main editor displays the code for the Plate class. The code is as follows:

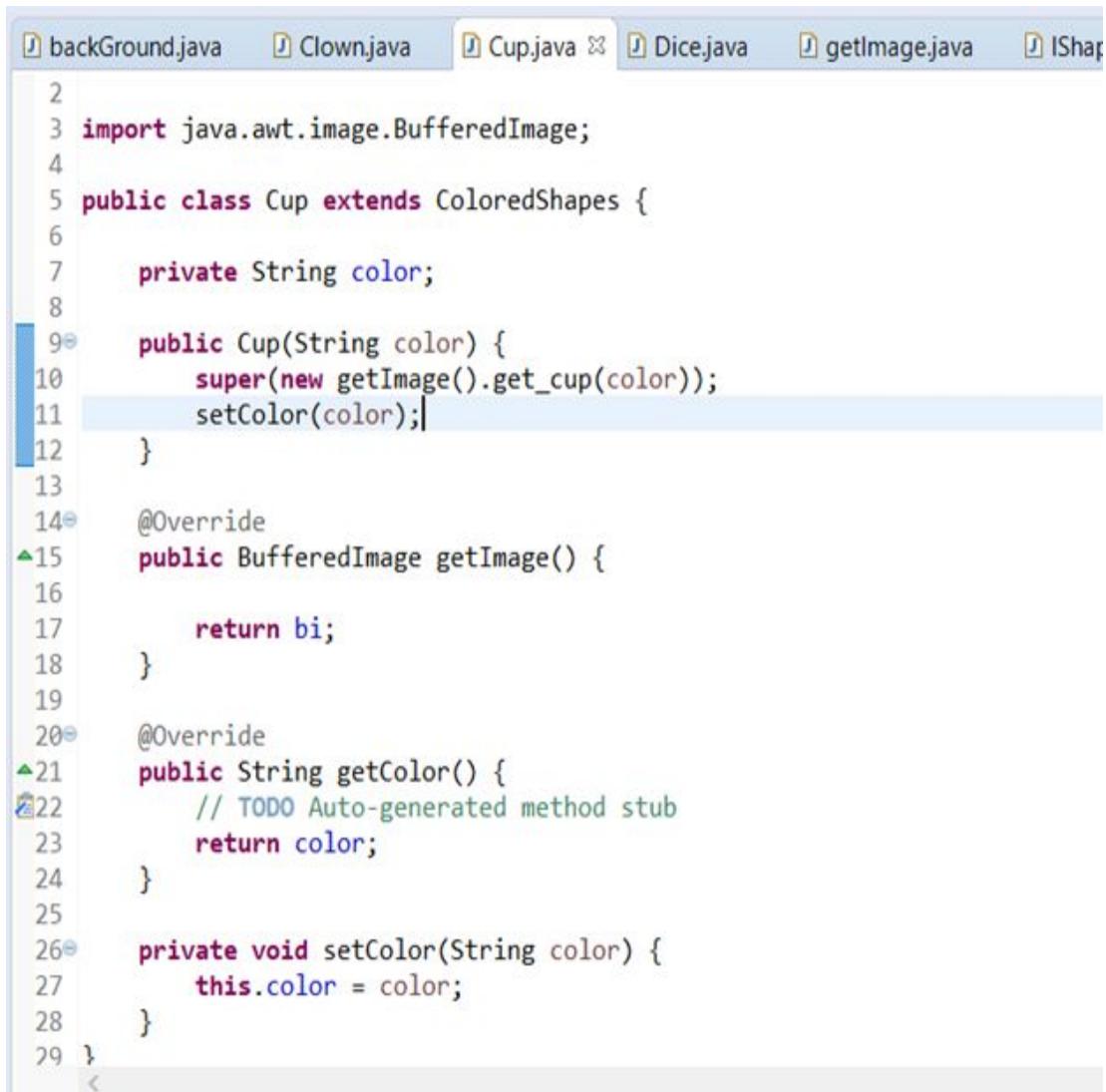
```
1 package to_come;
2
3 import java.awt.image.BufferedImage;
4
5 public class Plate extends ColoredShapes {
6     private String color;
7
8     public Plate(String color) {
9         super(new getImage().get_plate(color));
10        setColor(color);
11    }
12
13    @Override
14    public BufferedImage getImage() {
15        return bi;
16    }
17
18    @Override
19    public String getColor() {
20        return color;
21    }
22
23    private void setColor(String color) {
24        this.color = color;
25    }
26 }
27
```

<Clown>



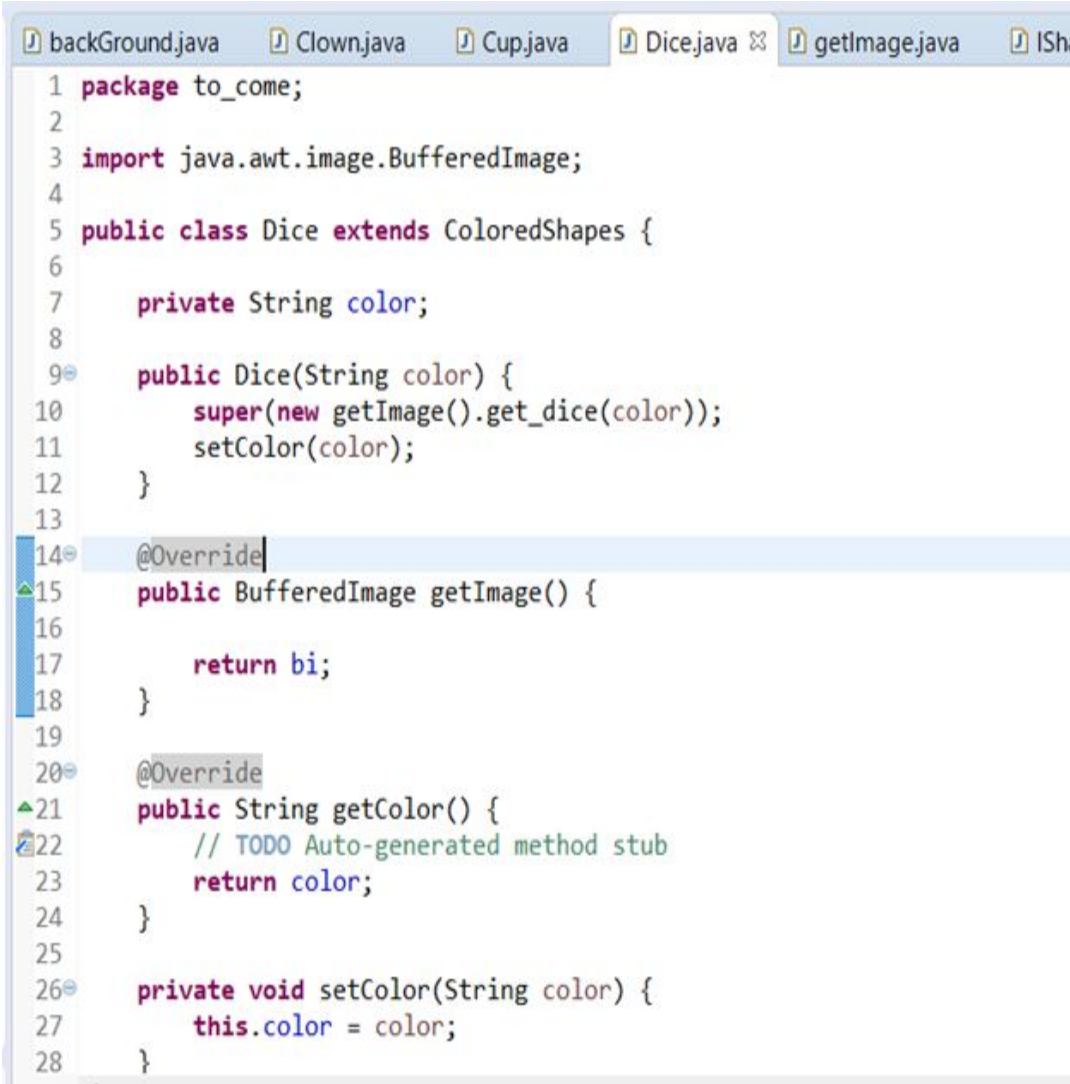
```
1 package to_come;
2
3 import java.awt.image.BufferedImage;
4
5 public class Clown extends ColoredShapes {
6
7     private String color;
8
9     public Clown(String color) {
10         super(new getImage().get_clown(color));
11         setColor(color);
12     }
13
14     @Override
15     public BufferedImage getImage() {
16         return bi;
17     }
18
19     @Override
20     public String getColor() {
21         // TODO Auto-generated method stub
22         return color;
23     }
24
25     private void setColor(String color) {
26         this.color = color;
27     }
28 }
```

<Cup>



```
2
3 import java.awt.image.BufferedImage;
4
5 public class Cup extends ColoredShapes {
6
7     private String color;
8
9     public Cup(String color) {
10         super(new getImage().get_cup(color));
11         setColor(color);
12     }
13
14     @Override
15     public BufferedImage getImage() {
16
17         return bi;
18     }
19
20     @Override
21     public String getColor() {
22         // TODO Auto-generated method stub
23         return color;
24     }
25
26     private void setColor(String color) {
27         this.color = color;
28     }
29 }
```

<Dice>



```
1 package to_come;
2
3 import java.awt.image.BufferedImage;
4
5 public class Dice extends ColoredShapes {
6
7     private String color;
8
9     public Dice(String color) {
10         super(new getImage().get_dice(color));
11         setColor(color);
12     }
13
14     @Override
15     public BufferedImage getImage() {
16
17         return bi;
18     }
19
20     @Override
21     public String getColor() {
22         // TODO Auto-generated method stub
23         return color;
24     }
25
26     private void setColor(String color) {
27         this.color = color;
28     }
```

<Getting image>

```

1  package circus/src/to_come/ControlledImageObject.java
2
3  import java.awt.Dimension;
4
14 public class getImage {
15
16     public BufferedImage get_clown(String color) {
17         try {
18             jarread x = new jarread();
19             List<Class<?>> listofClasses = x.getCrunchifyClassNamesFromJar("JAR_F.jar");
20             return ImageIO.read(listofClasses.get(4).getResourceAsStream("/" + color + "clown.png"));
21         } catch (IOException e) {
22             e.printStackTrace();
23         }
24         return null;
25     }
26
27     public BufferedImage get_plate(String color) {
28         try {
29             jarread x = new jarread();
30             List<Class<?>> listofClasses = x.getCrunchifyClassNamesFromJar("JAR_F.jar");
31             return ImageIO.read(listofClasses.get(1).getResourceAsStream("/" + color + "plate.png"));
32         } catch (IOException e) {
33             e.printStackTrace();
34         }
35         return null;
36     }
37
38     public BufferedImage get_cup(String color) {
39         try {
40             jarread x = new jarread();
41             List<Class<?>> listofClasses = x.getCrunchifyClassNamesFromJar("JAR_F.jar");
42             return ImageIO.read(listofClasses.get(0).getResourceAsStream("/" + color + "cup.png"));
43         } catch (IOException e) {
44             e.printStackTrace();
45         }
46         return null;
47     }
48
49     public BufferedImage get_dice(String color) {
50         try {
51             jarread x = new jarread();
52             List<Class<?>> listofClasses = x.getCrunchifyClassNamesFromJar("JAR_F.jar");
53             return ImageIO.read(listofClasses.get(5).getResourceAsStream("/" + color + "dice.png"));
54         } catch (IOException e) {
55             e.printStackTrace();
56         }
57         return null;
58     }
59
60     public BufferedImage get_background(String color) {
61         try {
62             jarread x = new jarread();
63             List<Class<?>> listofClasses = x.getCrunchifyClassNamesFromJar("JAR_F.jar");
64
65             Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
66             int height = (int) (screenSize.getHeight() - 120);
67             int width = (int) screenSize.getWidth();
68
69             BufferedImage img = ImageIO.read(listofClasses.get(3).getResourceAsStream("/" + color + "background.jpg"));
70             Image tmp = img.getScaledInstance(width, height, Image.SCALE_SMOOTH);
71             BufferedImage dimg = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
72
73             Graphics2D g2d = dimg.createGraphics();
74             g2d.drawImage(tmp, 0, 0, null);
75             g2d.dispose();
76
77             return dimg;
78         } catch (IOException e) {
79             e.printStackTrace();
80         }
81         return null;
82     }
83
84 }
85
86
87
88
89
90
91

```

<Colored>



```

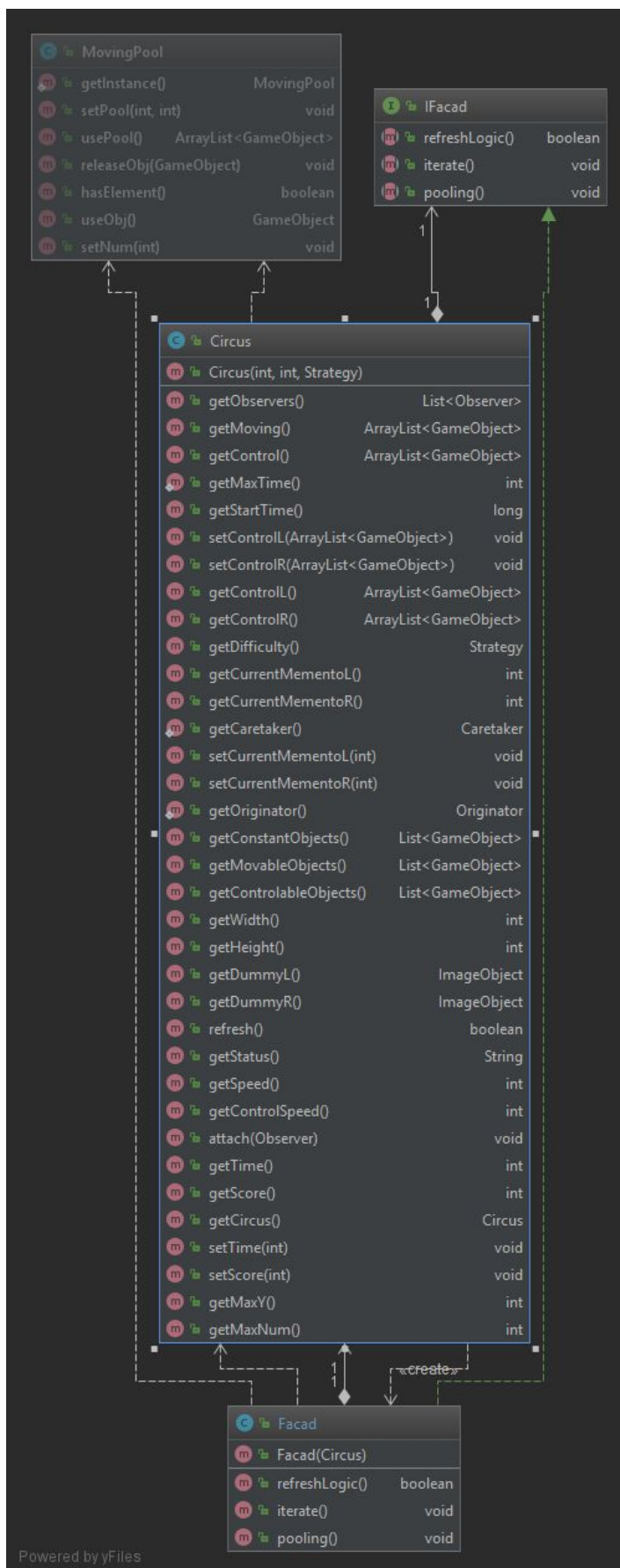
1 package to_come;
2
3 import java.awt.image.BufferedImage;
4
5 public class Clown extends ColoredShapes {
6
7     private String color;
8
9     public Clown(String color) {
10         super(new getImage().get_clown(color));
11         setColor(color);
12     }
13
14     @Override
15     public BufferedImage getImage() {
16         return bi;
17     }
18
19     @Override
20     public String getColor() {
21         // TODO Auto-generated method stub
22         return color;
23     }
24
25     private void setColor(String color) {
26         this.color = color;
27     }
28 }

```

12. Facade

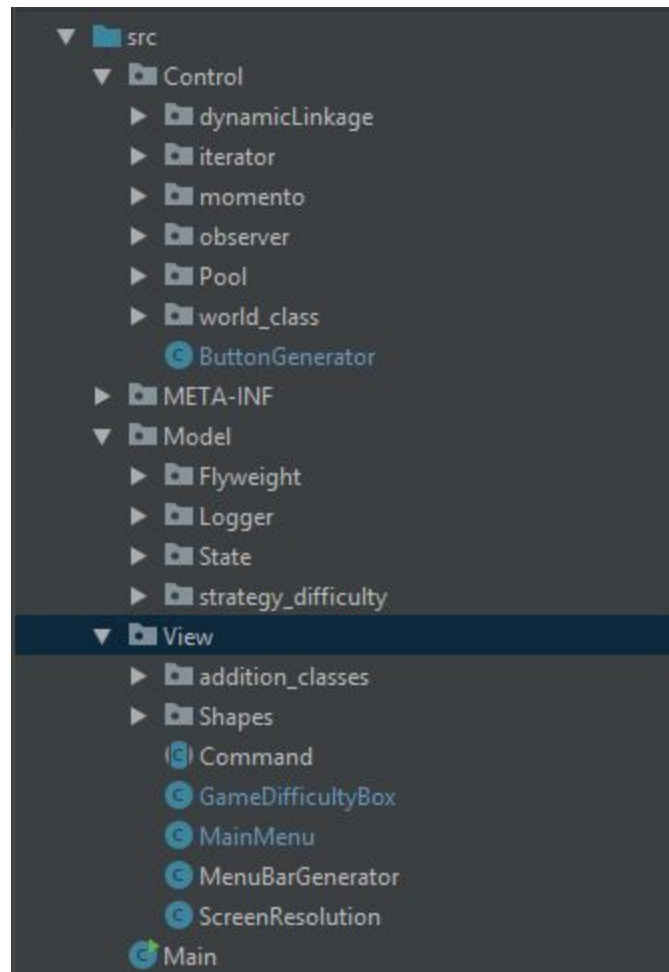
- 1) Connecting all game logic in a single class throw the refresh method of the world interface.
- 2) handling Creating plates when taking
- 3) Refreshing the scene of the game





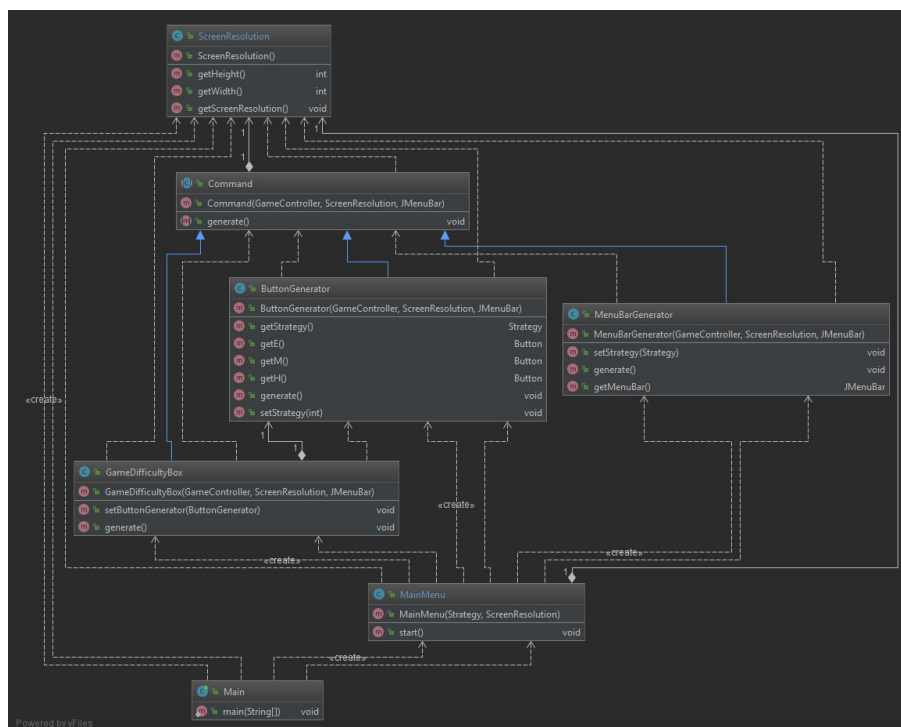
13. MVC

1) We applied MVC on our Design.



14. Command

- 1) We made the Main Menu to choose the desired difficulty. We used the Command design pattern to implement that part.
- 2) The Main Class creates an instance of ScreenResolution to get the screen resolution then creates an instance of MainMenu. Then calls method start which instantiates 3 classes that extend the Command Abstract Class. We made it Abstract Class just to fill the Constructor.
- 3) The 3 subclasses are MenuBarGenerator which generates the MenuBar at the Game, ButtonGenerator which creates the buttons holding the difficulties & GameDifficultyBox which is the Main Menu basically.



Code Design

The main class is the world-class, which is passed through the constructor to the game engine after the user chooses his game mode from the main menu. The game mode changes some features in the game using the strategy design pattern as time, speed & score calculation method.

Then after these classes that contain images, we load them using dynamic linkage.

After that music is loaded..and the pool is loaded using the pool design pattern

In refresh memento design pattern is used to record objects fall on left & right hand.state design pattern is used to switch between different states of objects as constant, movable & controllable Observer is used to updating time, score, sound & update plates after 3 plates of the same colour are above each other and return them back to the world

Design Decisions

1)The main game features

How to catch shapes?

The clown initially has right and left plates from which the user can decide the area of intersection so he can adjust the clown to catch the shapes,,, the falling shape center must be inside this interval ,, but as the clown catching the shapes things won't be straight some shapes will go little bit left or right but still inside this interval, so another condition is added for the catch process to

happen, besides being in the intersection of the first plate, the falling shape has to intersect with the last plate too .

This leads us to the last possibility ,, that the falling shape intersect with the last shape but at the same time it doesn't intersect with the intersection area of the first 2 plates at this point the falling shape doesn't intersect but it causes the last shape to fall down with it ,, the fallen plate will be returned to the moving shapes again but the user loses a point from this ..

This last situation happens when the plates go in the same direction right of a shape that was right or left of a shape that was left

So that the user has to try to center all the shapes.

The user starts with a limit of falling shapes decided by the mode, when the user catches a shape its reduced from the limit of shapes available when the user collects any 3 successive shapes of the same color he gains score **and the 3 shapes are added again to the pool of shapes available** ,,,

The user has another **feature** to drop the shapes from the clown


2)The game ends when

1)the time run out

2)the user catches all the shapes without any 3 successive shapes of the same color

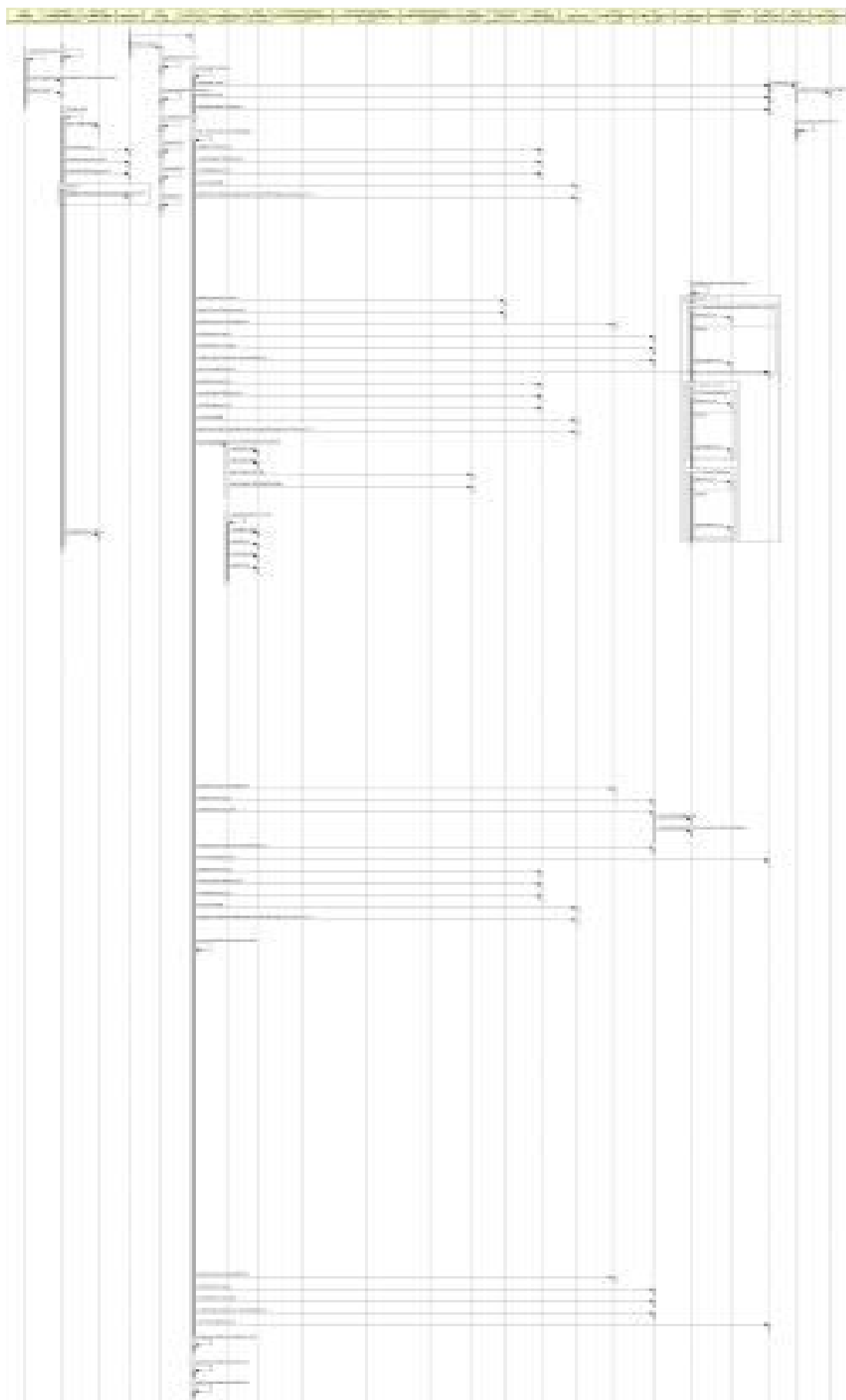
3)the shapes on either hand exceeds the max y for this shapes

3)the grading



1)the user gains bonus score, time added to the max time when he collects 3 successive shapes of the same color and loses score when a plate fall off his hands

Sequence Diagram



The diagram illustrates a highly complex digital logic circuit, possibly a CPU or microcontroller architecture. It features a dense network of interconnected functional blocks and signal lines. Key components include:

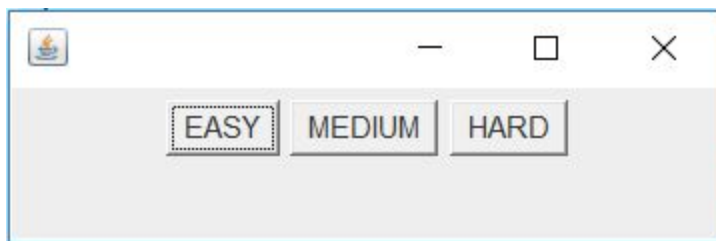
- Input/Output and Control:** At the top, there are several input/output blocks and control logic, including a "V. Control" block and a "V. Status" block.
- Arithmetic Logic Unit (ALU):** A central block labeled "V. ALU" is connected to multiple registers and control logic.
- Registers and Memory:** Numerous registers are shown, each with a "V. Register" label and a "V. Status" label. These are connected to a central bus system.
- Control Logic:** A large section on the right contains complex control logic, including a "V. Control" block and a "V. Status" block.
- Signal Flow:** The circuit is characterized by a dense network of signal lines, many of which are labeled with "V. Status" or "V. Control", indicating the flow of control signals and data.

The overall layout is highly organized, with a clear separation between control logic, data processing, and status monitoring components.

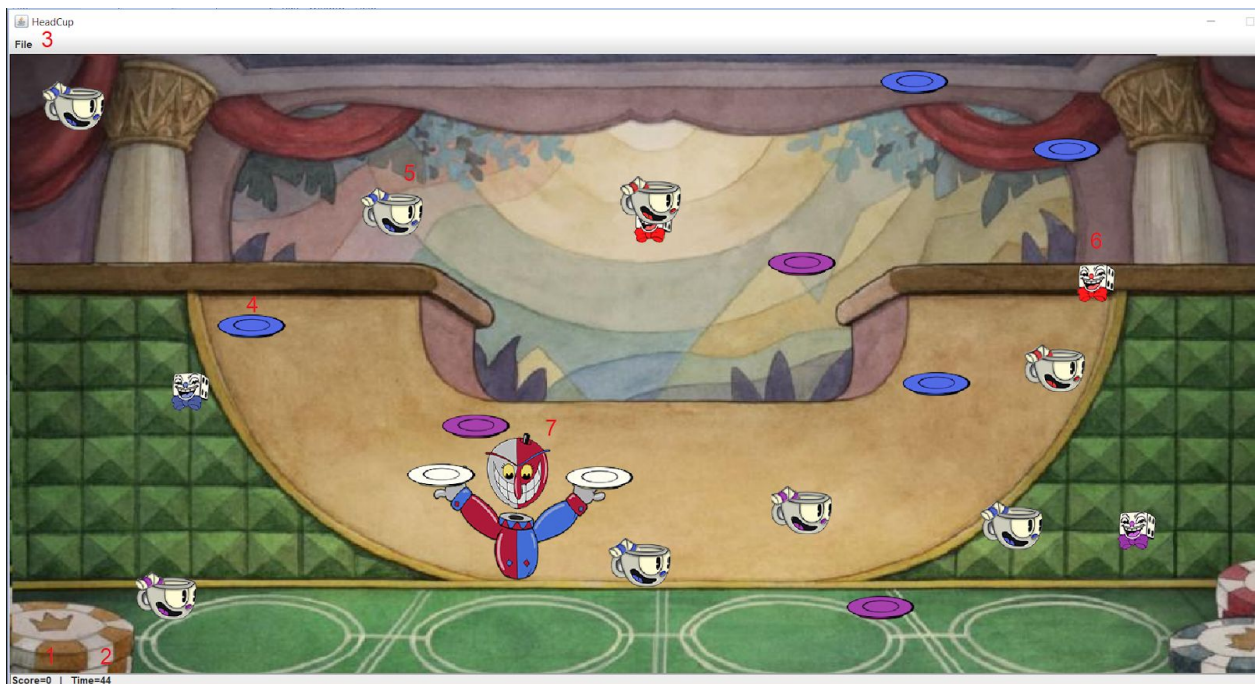
User Guide:

Game provides choosable mode eg. easy, normal and hard

These modes differ in speed of shapes, increases of a score, number of shapes, increases of time and max height of shapes.



When selecting a mode the game will begin.



- 1) Gamers score.
- 2) remainedtime.
- 3) Menu bar to use option.
- 4) Plate shape (blue).
- 5) Cup shape (blue).
- 6) Dice shape (red).
- 7) Clown.

When collecting three shapes have the same color Consecutive the score will increase, time bonus will be added and sound will play.



If the shape falls on the edge of the last shape the two shapes will fall
(in this example the dice will take blue plate and fall).



Finally, the game will over after 60sec.

