Lab 5

Given is the following simple Game application:

```
public class Game {
	private int totalPoints = 0;
	private int level = 1;

	public void play() {
		Random random = new Random();
		addPoints(random.nextInt(7));
		System.out.println("points="+totalPoints+" level="+level);
	}

	public int addPoints(int newPoints) {
		if (level == 1) {
			totalPoints = totalPoints + newPoints;
			if (totalPoints > 10) { // move to level 2
				level = 2;
				totalPoints = totalPoints + 1;//add 1 bonus point
			}
		} else if (level == 2) {
			totalPoints = totalPoints + 2 * newPoints;
			if (totalPoints > 20) { // move to level 3
				level = 3;
				totalPoints = totalPoints + 2; //add 2 bonus points
			}
		} else if (level == 3) {
			totalPoints = totalPoints + 3 * newPoints;
		}

		return totalPoints;
	}

}

public class Application {
	public static void main(String[] args) {
		Game game = new Game();
		game.play();
		game.play();
		game.play();
		game.play();
		game.play();
	}
}
```

Whenever we play the game, we get points. We can move up to a higher level of the game when we have a certain amount of points. In our current implementation we start with level 1, and if we have more than 10 points, we go to level 2. If we have more than 20 points in level 2, we move to level 3.

In level 1, our totalpoints is increased with the points we receive in each play. In level 2, our totalpoints is increased with 2 times the points we receive in each play. And in level 3, our totalpoints is increased with 3 times the points we receive in each play.
If you are upgraded to level 2, then you get 1 bonus point.
If you are upgraded to level 3, then you get 2 bonus points.

The problem with the given code is the addPoints() method. If we want to add another level, we have to change this method. If we want to change anything regarding point calculation rules, we always have to change this method. Just imagine how complex this method will be if we have 10 levels and very complex point calculations.
Redesign this application in such a way that

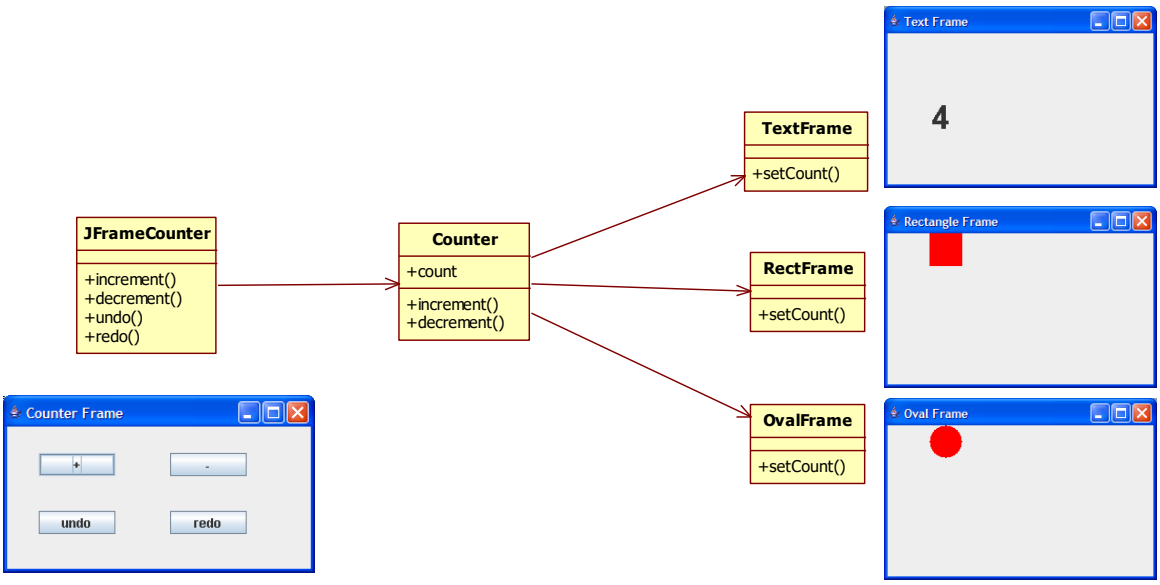
- It will be easier to add new levels
- It will be easier to change the point calculations
- The Game class is independent of the different levels. This means I should be able to add a new level without any change in the Game class.

If you are done, add another level 2_5 with the following business rules:

- If you are in level 2, and you have more or equal to 15 points, then you go to level 2_5
- In level 2_5 you get 1 bonus point.
- If you are in level 2_5, and you have more or equal to 20 points, then you go to level 3

a. Draw the class diagram of your design.
b. Draw the sequence diagram that shows how your design works. Show in the sequence diagram how you move from level 1 to level 2.
c. Implement your design in Java.

d. In lab 4 we applied the observer and command pattern to the following application:



Now we want to add the following functionality to this application:

- When the Counter value is a single digit number, then every button action (increment and decrement) will add or subtract 1 point from the current teller Counter.
- When the Counter value is a double digit number, then every button action (increment and decrement) will add or subtract 2 points from the current Counter value.
- When the Counter value is a triple digit number, then every button action (increment and decrement) will add or subtract 3 points from the current Counter value.

Draw the class diagram. Your class diagram needs to show both the observer pattern of lab 4, the command pattern of lab 3 and the state pattern in one diagram.

e. Draw the sequence diagram that shows the following scenario:
   a. The user clicks the increment button
   b. The user clicks undo

f. Implement your new design in Java. Your starting code should be the solution of lab 5. So the code should also implement the observer and command pattern.

g. Consider the gate application of lab4.

Now you receive a new requirement from marketing. When the gate is busy with opening or closing, and you press the button on the remote, then the gate should stop moving and stand completely still. The buzzer should also go idle if the gate stands completely still. In this state, when you press the button again, it should continue with the action it was doing, either opening or closing the gate. You notice now that the initial implementation of the GateController becomes ugly with a lot of conditional code.

Improve the application so that it is easy to add new states to the gate controller.

What to hand in?

1. A jpeg picture of part a, b, d and e
2. A zip file containing the project of part c, f and g