Exercise 2 - Patterns

Part 1

My program from Exercise 1 implements a simple Pong game using the LibGDX framework. I did not finish the game as I wanted in the last exercise, it lacked the score board. The very first step was therefore to implement this.

The game consists of three java classes: “MyPongGame”, “Ball”, and “Paddle”. MyPongGame serves as the core of the game, handling the rendering, input, and game logic. It initializes the game objects such as the paddles and ball, sets up the input processing for paddle movement, and manages the scoring system. The game loop, implemented in the render() method, updates the game state, checks for collisions, and renders the game objects on the screen. The Ball class represents the ball object in the game. It encapsulates its position, radius, speed, and logic for movement and collision detection. The ball moves continuously within the game window, bouncing off the paddles and walls based on collision detection. The Paddle class defines the paddles used by the players to hit the ball. It handles paddle movement and rendering. Paddle movement is being controlled by dragging the mouse or touch input.

Part 2

In task 2, the next version of the Pong game includes a Singleton class named “PongManager”. It manages the game parameters and difficulty settings as well as encapsulating the difficulty level, ball speed, and paddle height. The major difference between this program and the previous one lies in how the game parameters are handled. Instead of hardcoding values for ball speed and paddle height directly in the MyPongGame class, these parameters are now accessed through the PongManager singleton. In a game like this, it makes sense to have one and only one object which acts as the central manager, and therefore I used the Singleton pattern for this purpose.

The singleton provides methods to set and retrieve the current game level, which dynamically adjusts the game parameters accordingly. For instance, based on the selected level (easy, medium, or hard), the adjustGameParameters() method in the PongManager class modifies the ball speed and paddle height, making the game easier or harder. This program offers more flexibility and scalability to the game, as adjustments to game parameters can be easily made within the manager class without modifying the core game logic. By utilizing the Singleton design pattern, there is only one instance of PongManager throughout the game, ensuring consistent access across different components and classes.

Part 3

In the final version of the Pong game, I used the Model-View-Controller pattern (MVC) to separate logic and improve the structure of the code. The model components (Ball, Paddle and PongManager) represent the data and logic of the game, encapsulating properties and behaviors related to game objects and parameters. The PongManager class still works as a singleton and it provides methods to get and update parameters in the other classes, ensuring a centralized control over the game. The Ball and Paddle classes represent game objects and encapsulate their properties and behaviors, such as position, speed, and collision detection.

In the view-folder, I now have the Renderer and Button classes responsible for the visual presentation of the game. The Renderer renders the main game objects and user interface elements, while the Button class represent the clickable buttons for selecting the game levels. In the controller-folder, I have a GameController class that works in between the model and view classes, handling the user input and updating the state of the game accordingly. The controller communicates with the model classes to retrieve parameters and update objects' positions and states, while also updating the view-classes to render the game correctly. This separation of logic in the classes enhances modularity, maintainability and scalability of the code. It makes it easier to understand the whole structure of the program and for different developers to work on different parts of the game.

Part 4

a) The Model-View-Controller pattern is an architectural pattern. The other ones are design patterns. Design patterns is used to solve smaller and more specific design problems in software systems, and it provides solutions to individual classes or components. On the other hand, the architectural patterns act as a more general or structural guide for designing software systems. It is used for managing the interaction and organization between major components (like the model, view and controller).

b) As mentioned in Part 3, the model components are the Ball, Paddle and PongManager classes. They represent the main objects in the game and the singleton game manager. The Renderer and Button classes are responsible for the view of the system as they render the objects on the screen. The GameController works between the Model and View classes, as it updates the game based on user input.

c) Using the Model View Controller (MVC) pattern in this program offers several benefits. Firstly, it makes a clear separation of the code and dividing the application into distinct components/parts. This enhances the modularity and maintainability, and it makes the code easier to understand. However, there may be a downside to the increased complexity of this pattern, for instance in very large applications where managing interactions between components will be more difficult. A strict separation in the system may also lead to more overhead when organizing and communicating between the components.

Part 5

I did the exercise individually.