
# Report of assignment 5

By /

JamalEldin Ahmed

Ziad Taha

Mostafa Farrag

Mohamed Samy

Abdelrahman Alsayed Ahmed

# Table of contents

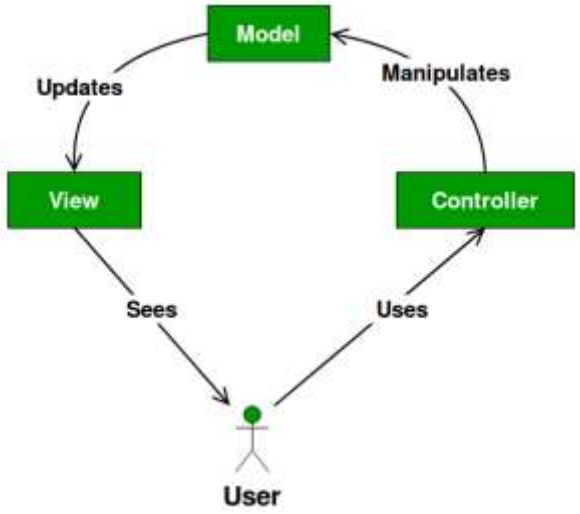
## Overview

It is single player-game in which each clown carries two stacks of plates, and there are a set of colored plates queues that fall, and he tries to catch them, if he manages to collect three consecutive plates of the same color, then they are vanished, and his score increases.

We have used another theme but still having the same rules. Instead of clown we have used “Mo Salah character”, he collects shapes that fall on his head (At two sides of his head) and our shapes are ball and the shape of goal. We will describe the rules of ending game below.

## Design

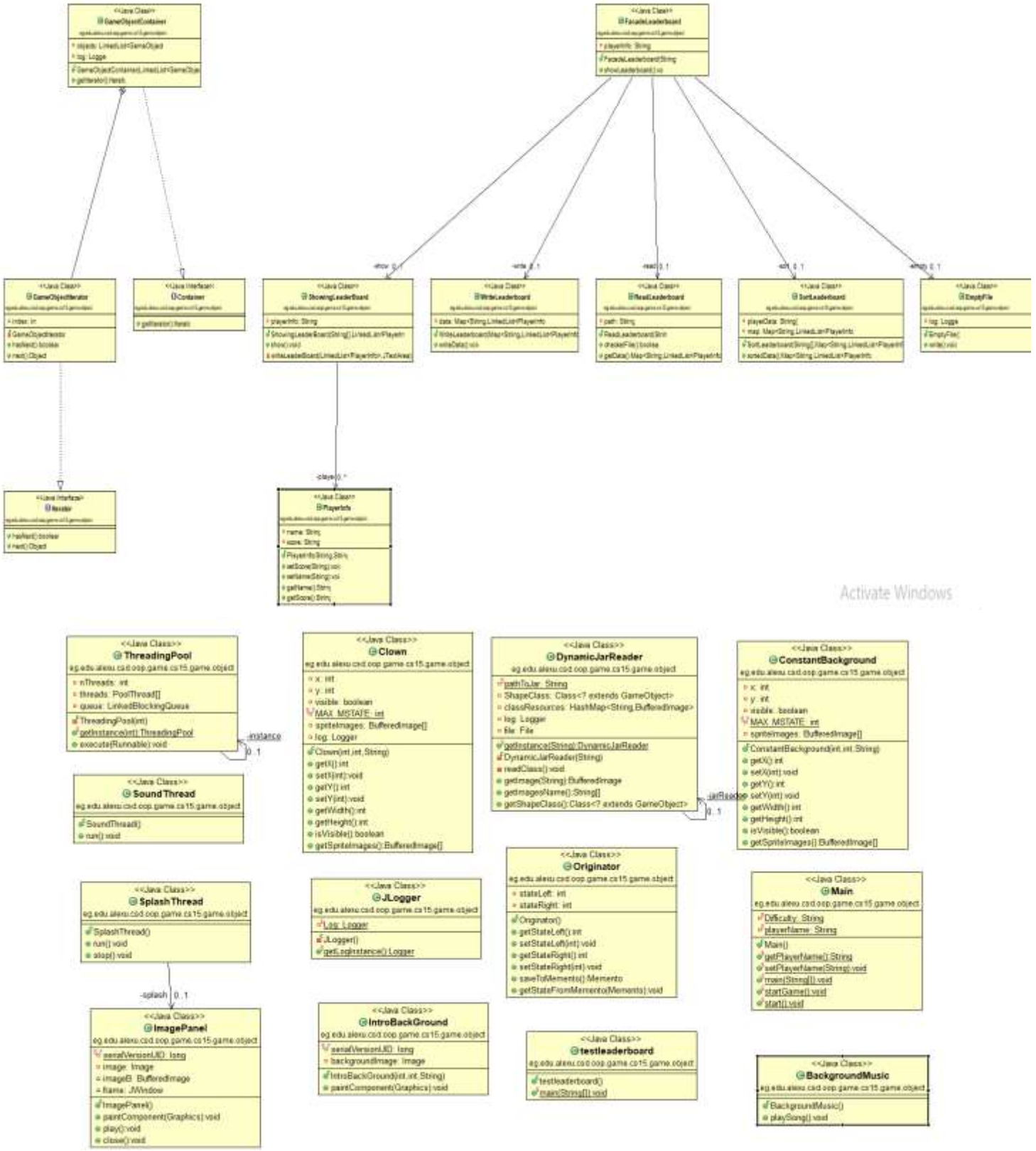
Due to game engine we must use MVC design pattern as game engine handles the view and controller and we have implemented model.

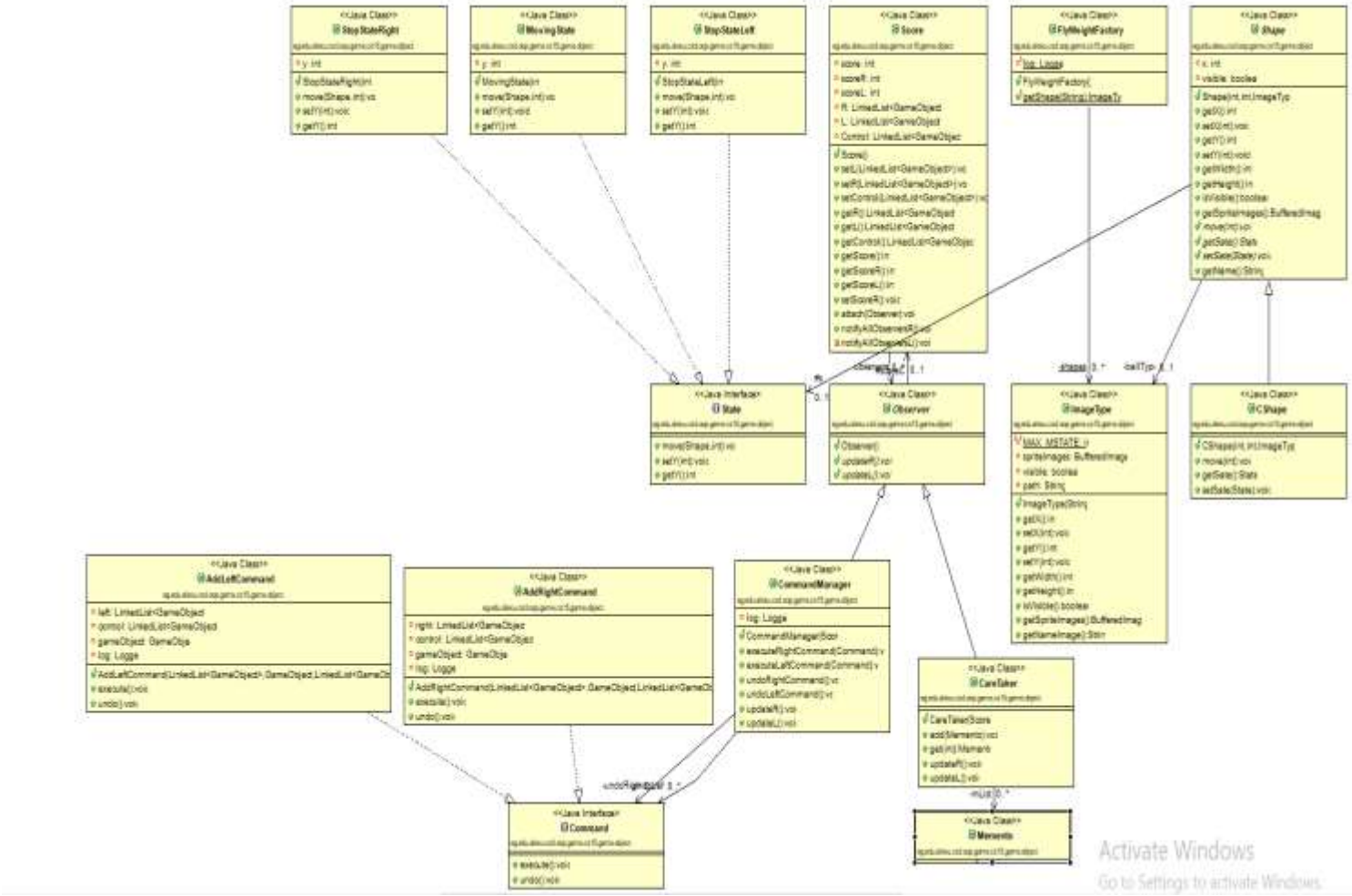



We have used 12 design patterns (Singleton, Factory, Iterator, Dynamic linkage, Snapshot (Memento), State, Strategy, Flyweight, Observer, Command, Façade, Thread pool).

We will break down each design pattern in the section of design patterns.

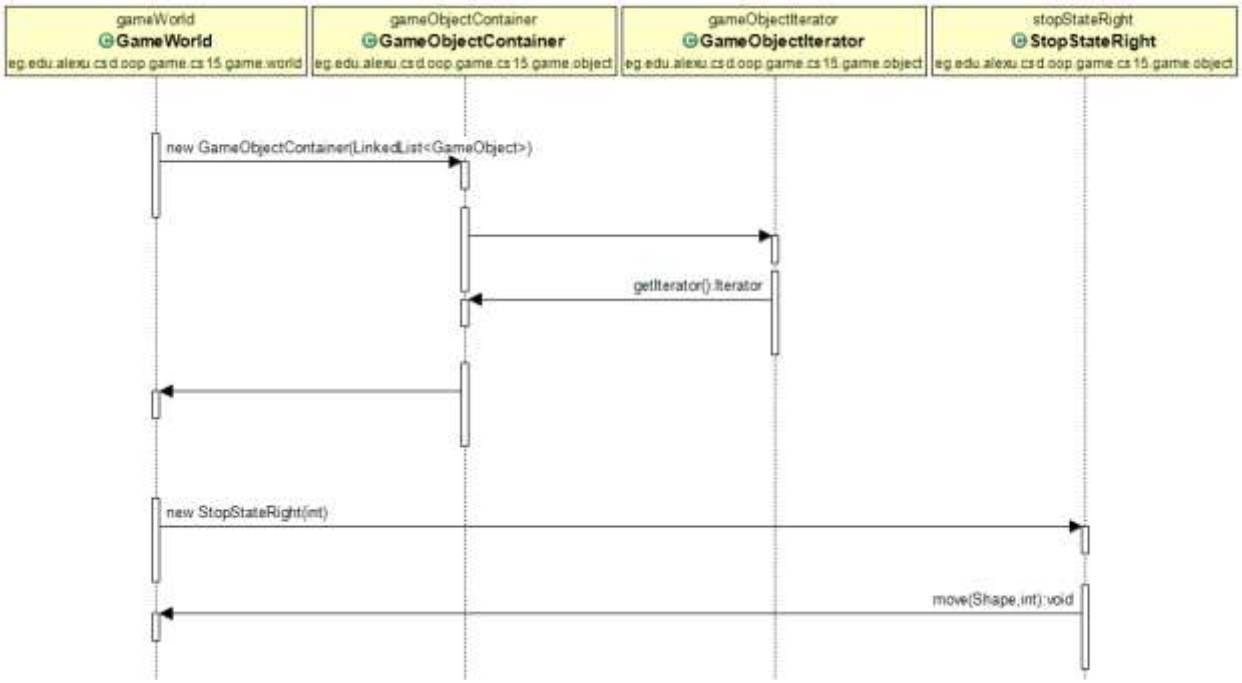
# Class diagram



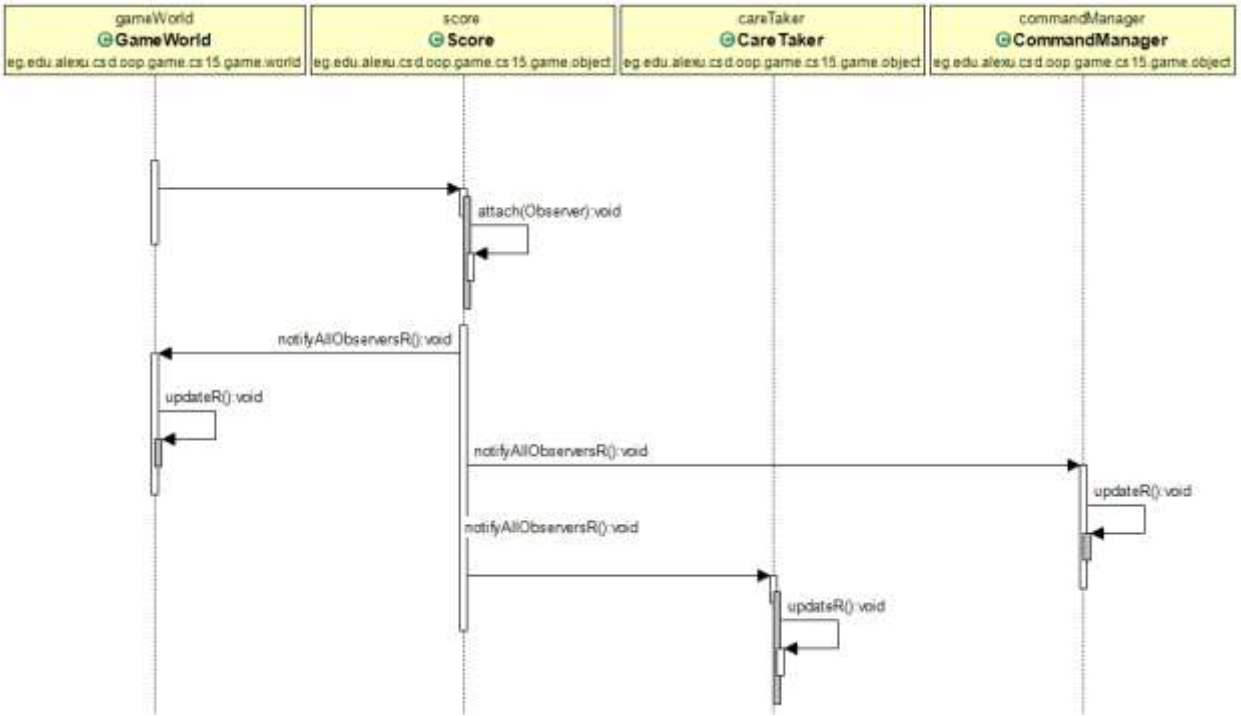

Activate Windows
Go to Settings to activate Windows.

# Sequence diagram

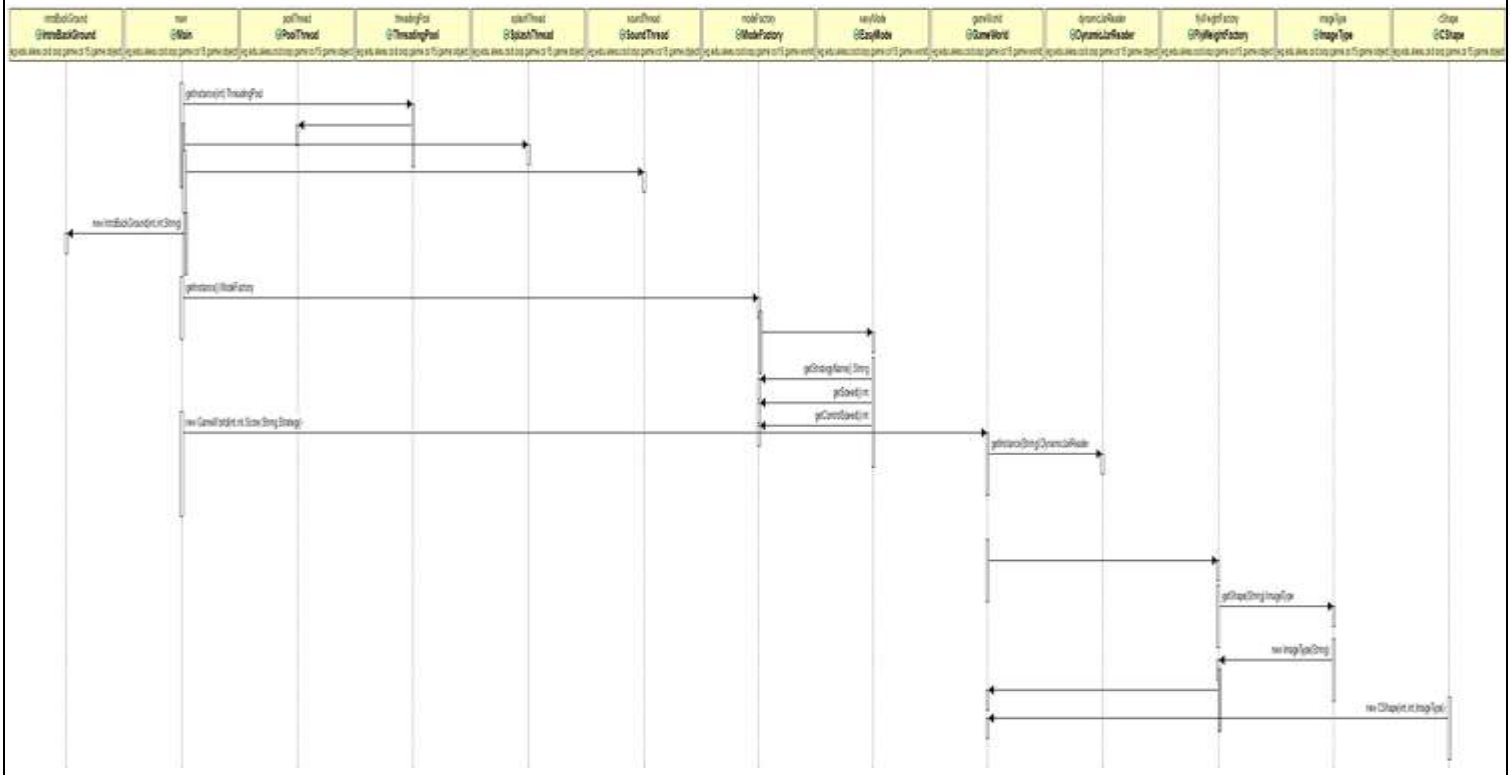
Scenario (1)




Scenario (2)



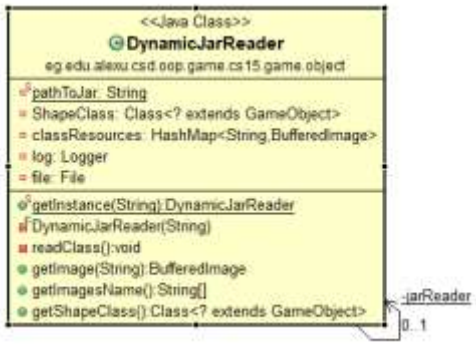
## Scenario (3)



## Design patterns

- Singleton

**Class diagram**



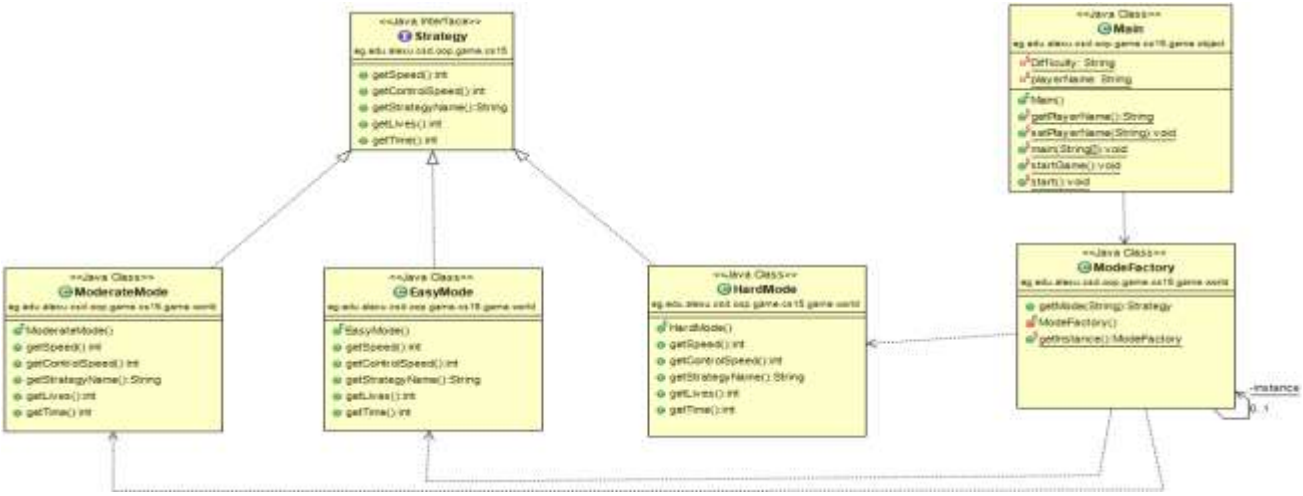

**Usage**

- this pattern involves a single class which is responsible to create an object while making sure that only single object gets created

- We use it in logger, Dynamic reader and factory

- Factory

**Class diagram**



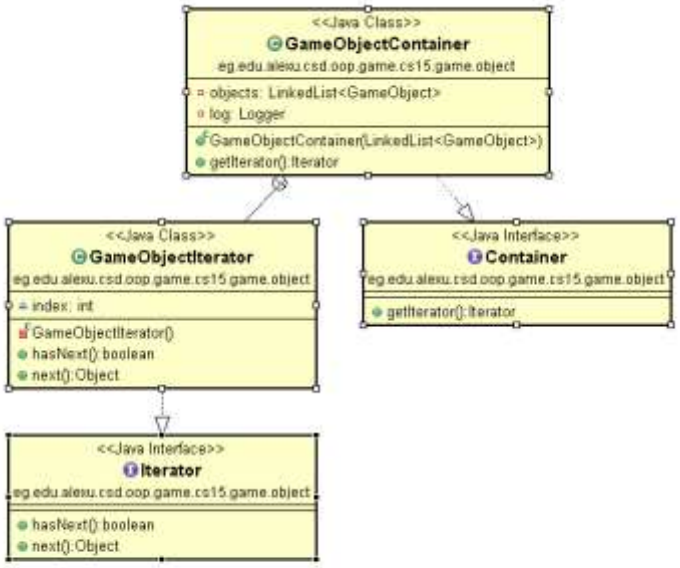

**Usage**

- In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface

- We use it to create the strategy (levels in the game) based on the client choose

- Iterator

**Class diagram**

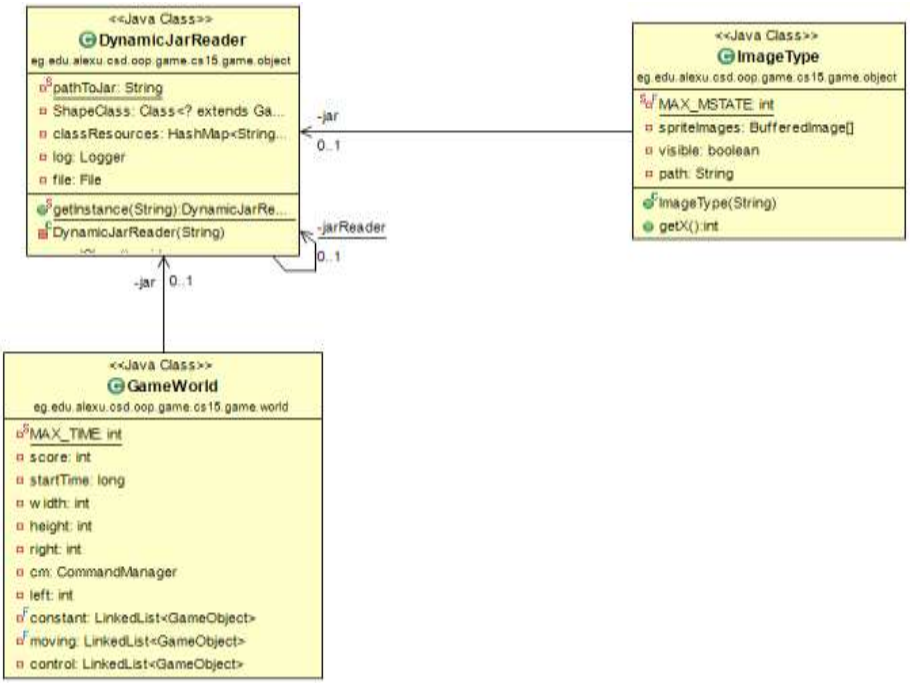



**Usage**

- We used iterator design pattern to make the looping process on the moving shapes LinkedList for checking intersection with the head, and the controls shapes linked list for checking reaching the edge of the screen

- Dynamic linkage

**Class diagram**

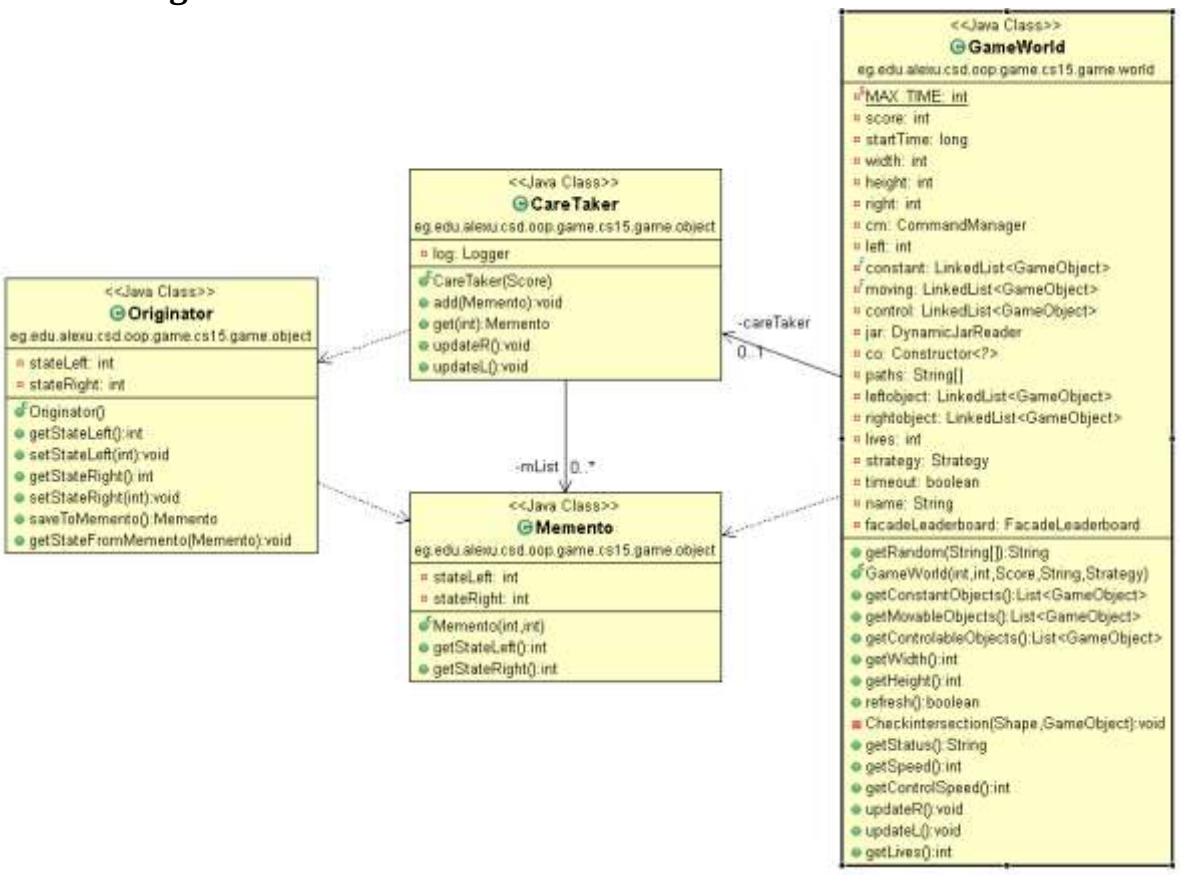



**Usage**

- The jar file has 1 class called CShape and a folder called "res" which has the images in it
- The jar file is being loaded to the class path at runtime until the end of the program
- The jarReader class is made by singleton so the whole program loading just one jar in the entire program
- Fixed path is being used to load the jar file
- The shapes in the jar(images) are put inside an instance of CShape.class then pick a random shape to make it fall
- This helped us to change the shapes dynamically and very easy without changing anything in the cod all you must do is generate a jar file and put the resources folder with the class called CShape

- Snapshot (Memento)

**Class diagram**



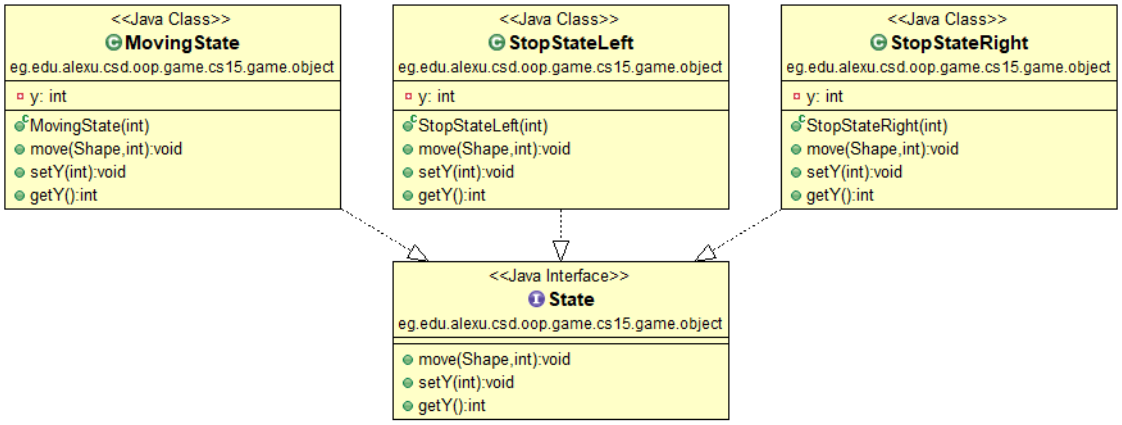

**Usage**

- We used memento design pattern to make checkpoints by saving the last state for controlled shapes and make the player go back to it when he loses and still have more lives.

- We used three class memento to save the state, originator to handle saving to a memento object, care taker to handle putting the memento object in the LinkedList of mementos caretaker class extends observer and update every the player scores a point, we don't we decided to save only size of each array as we don't actually need more than one memento in the linked list as it's the checkpoint that we need

- Sate

**Class diagram**

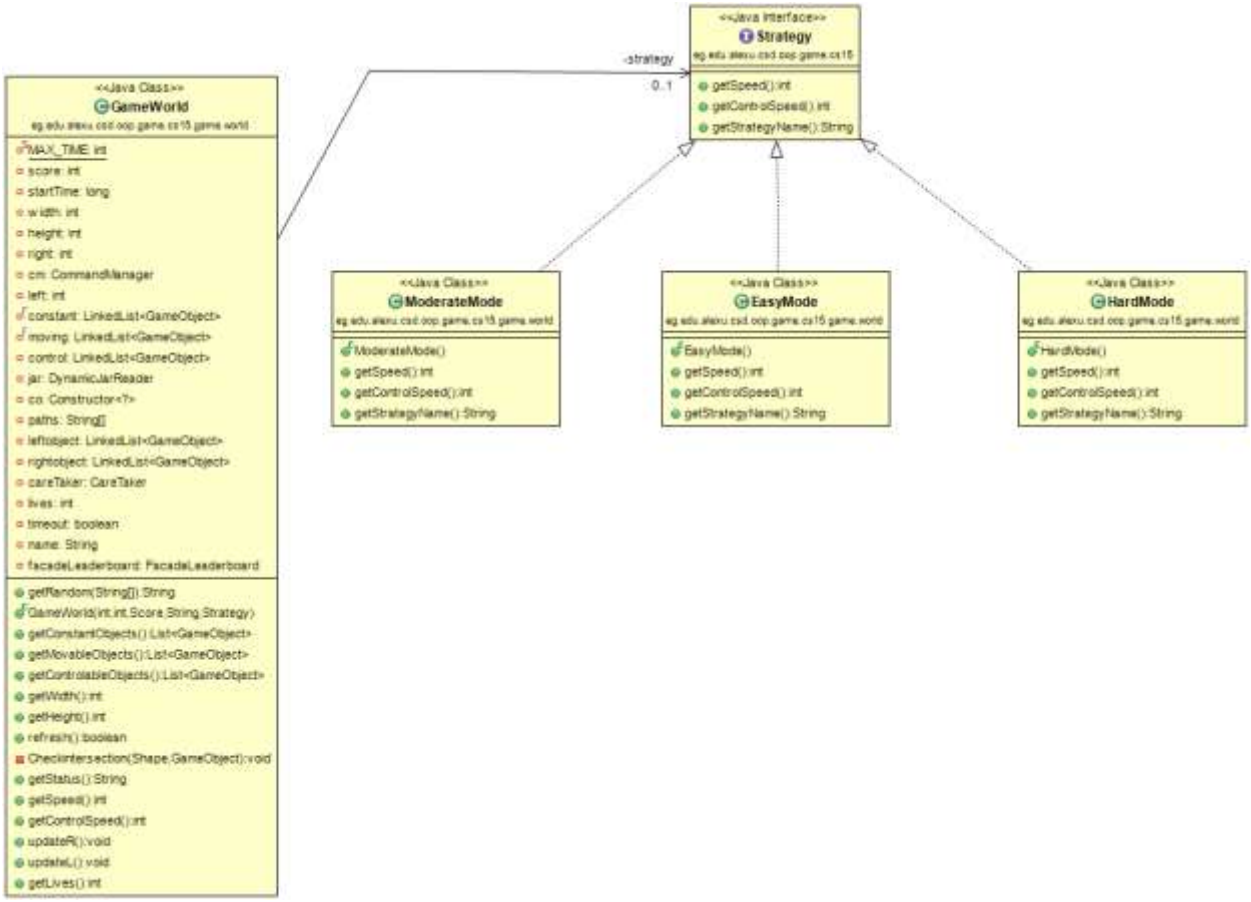



**Usage**

- This design pattern allows us to change the ball behavior based on its state
- We use three state one for moving ball and tow for stopped balls

- Strategy

**Class diagram**



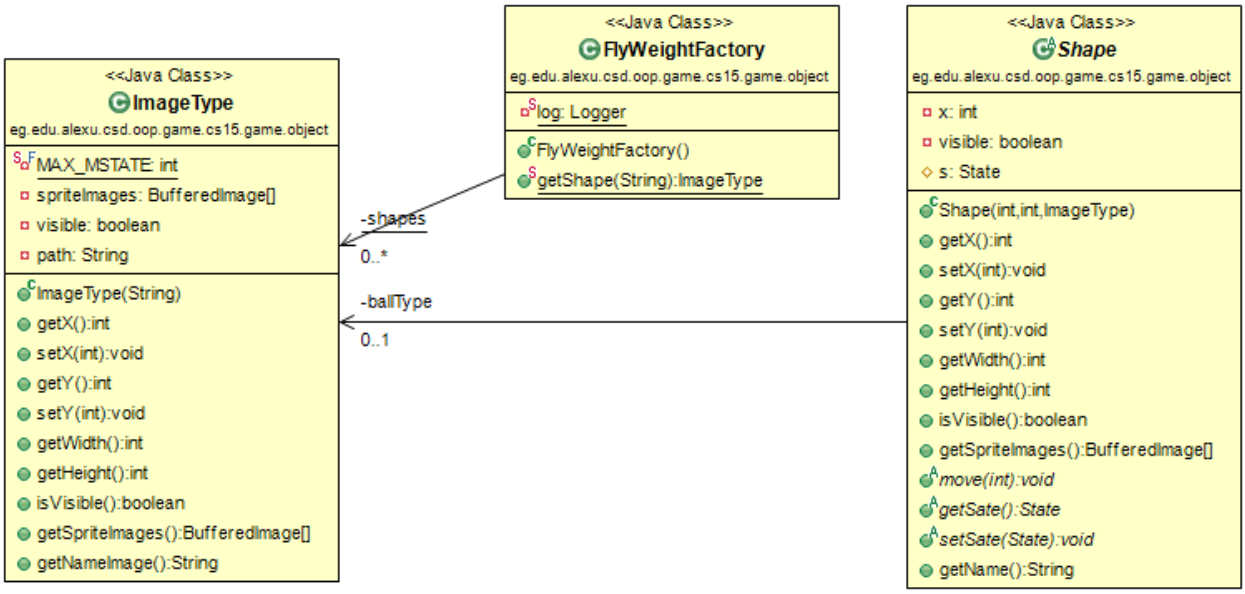

**Usage**

- For the Difficulty modes in the game we made three levels (Easy, Moderate and Hard) each level has its own clown speed but with the same method in the GameWorld (getControllerSpeed)

- So, we used Strategy Design Pattern where there is an interface contains the method which control the clown speed and we implement each level in different class that implement the same interface each class process the methods depending on the difficulty level the we implements this interface in the GameWorld class in order to accesses the needed level(class) through the interface

- Flyweight

**Class diagram**



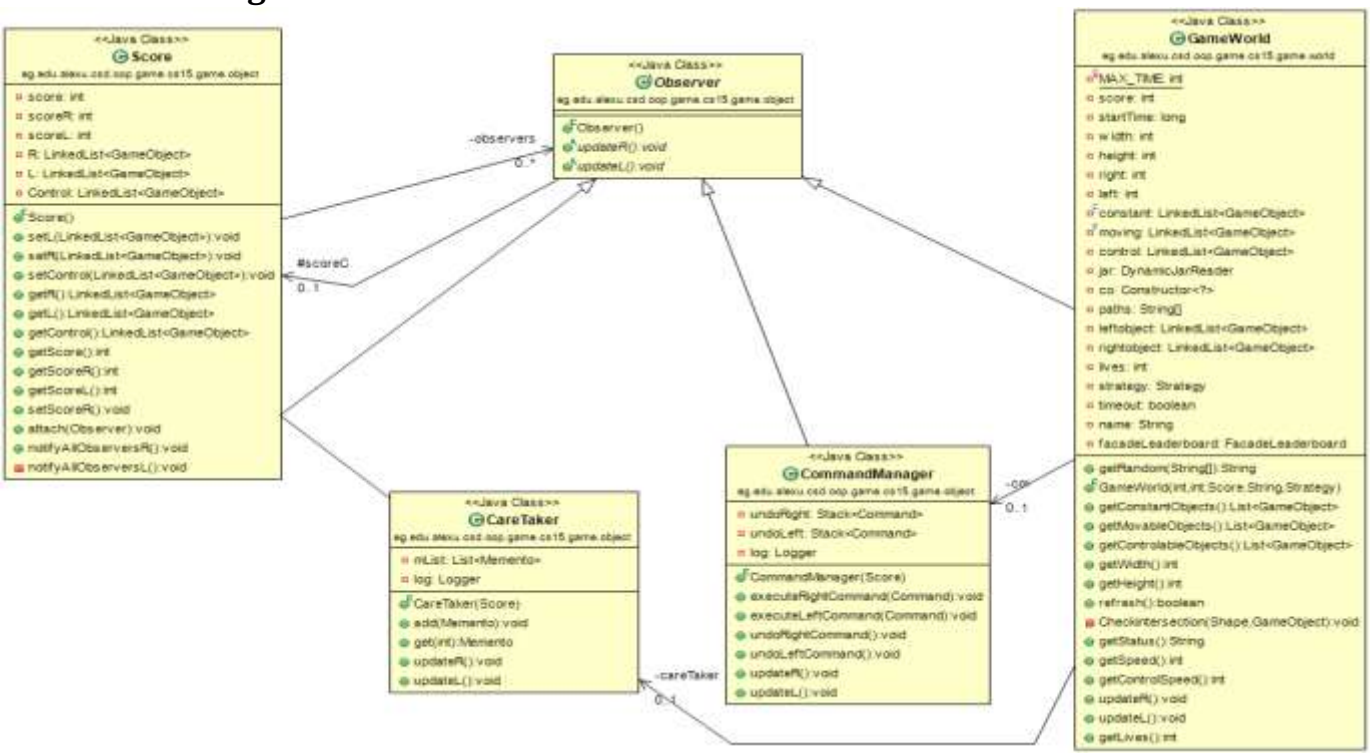

**Usage**

- In order to generate multiple objects each one with its unique properties we used the Flyweight Design Pattern

- So, that the ImageType class has the properties of each shape then Shape class set each shape its properties and the FlyWeightFactory decides whether to use existing shape or generate new one in order to save memory

- Observer

**Class diagram**



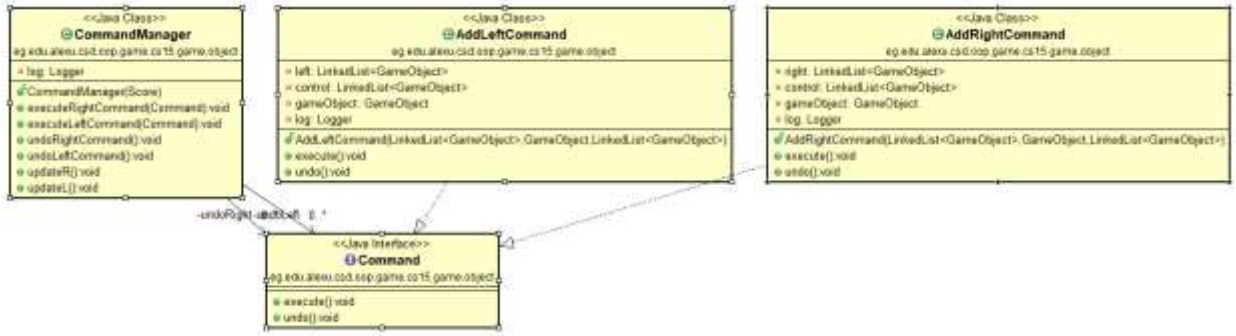

**Usage**

- We used observer design pattern to notify other classes with the score State Update

- So, we made the GameWorld class, the CareTaker class (Memento) and the CommandManger class (Command) observers when they are notified with the change in the score state each one executes the methods in the abstract class (updateR, updateL) independently

- Command

**Class diagram**



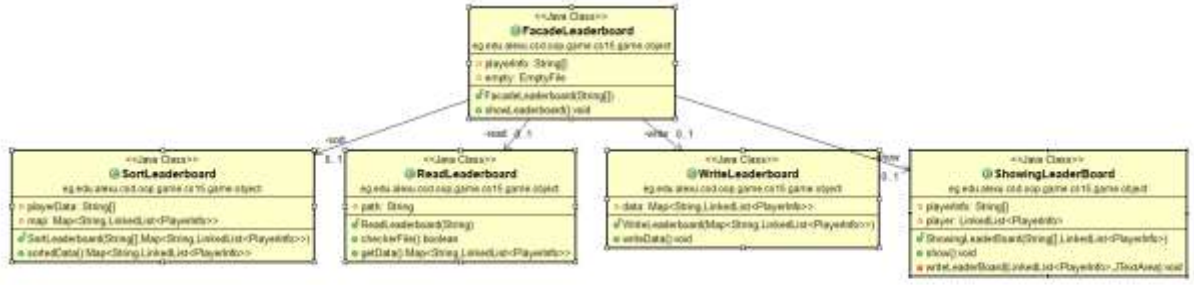

**Usage**

- We used this pattern to control the collecting of the fallen shapes. We used AddRightCommand and AddLeftCommand

- This pattern manages us to have an undo option to back to the correct state after having three shapes in the same color. This is controlled by CommandManager

- Façade

**Class diagram**




**Usage**

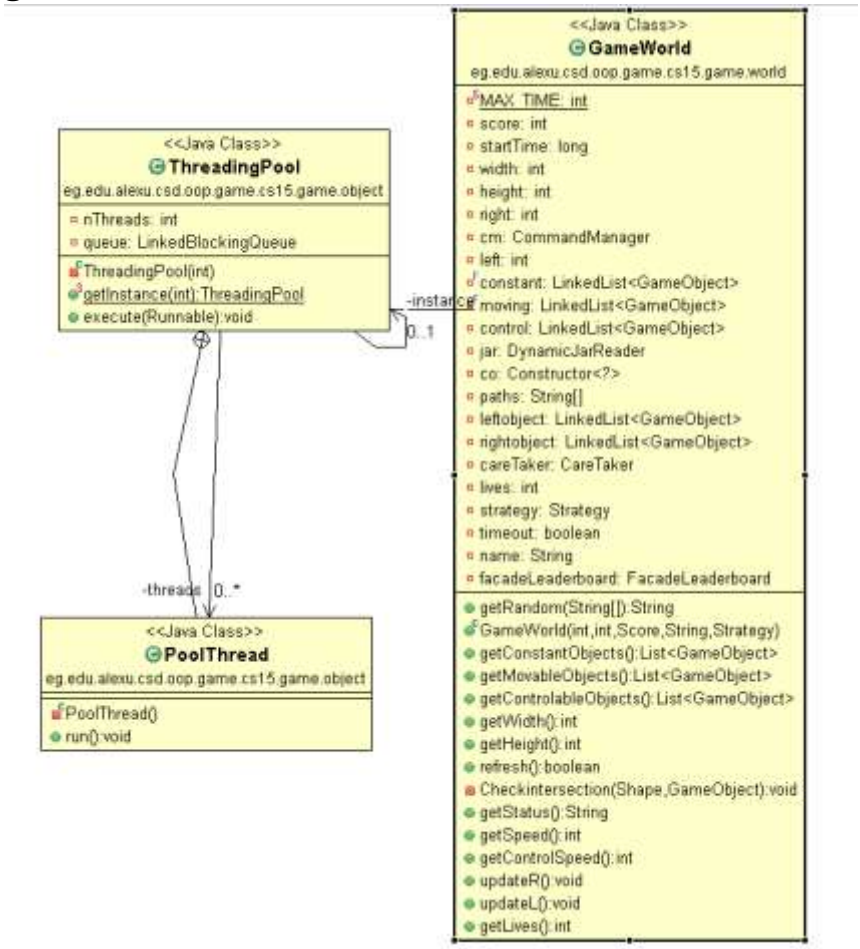
- We have used this pattern to make our leaderboard

- Our leaderboard system consists of four parts (read, sort, write and show)

- So, we have to use façade design pattern to handle all these classes

- Thread pool

**Class diagram**



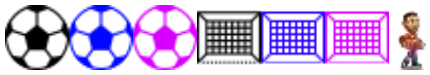

**Usage**

- We used Threading pool in the game to run multiple threads with multi tasks
- We used thread pool of two threads to execute three tasks, the first thread runs the splash gif screen then runs the background music, the second thread had one task which is running the game, threading pool design pattern made us use multi-threading with ease.
- we used thread pool class which contains an array of the needed number of pool thread objects which extends thread
- We made tasks (runnable object queue) and execute the tasks by the threads in the array

## Design decision

- Our main character

- Our supported fallen shapes



- How to collect fallen shapes

- When we start the game there will be a splash screen then we will require to enter our name (If we did not enter our name or close the dialog the game will not continue) after entering the name we will choose one of our modes (It is required that enter name consists of 5 letters if we enter more than 5 letters we will trunk it in the leaderboard)

- We have three modes in our game (Easy, Moderate, Hard). Each mode controls three things in the game the number of lives, time and speed of fallen shapes

- The rules of ending game; the game will end if we lose all lives. We decrease lives if we collect “Ramos character” or the time came to zero

- If we lose any life, we will return to last check point that we have win in it

- After losing all lives a new window will appear to show our leaderboard of all players who played this game before

- We have an individual leaderboard for every mode and we will show the suitable leaderboard according to your selected mode

- During the game you can pause, resume the game and start new game

- After having game over if we want to play another game we must restart the game