CS221: OOP Fall 2016

Circus of Plates

January 26, 2017

Overview

Circus of Plates is a two player game . Each Player is a clown in the *circus of plates* at which his goal to collect a big number of plates as possible as he can on his special stick. The two players are competing with each other to entertain themselves and to make the watchers inside the circus be happy. Each player has two shelves over him and the plates fall like a waterfall on the player.

The game is implemented in *Java* programming language. Multiple design patterns are used in the game design beside using *Java* Multithreading techniques to enhance the parallel performance of players and the falling plates.

The Game Design

The main actors playing the great role in the game are the falling ***plates*** and the ***player*** and the game designers make their design assumptions to manage the ***plates*** and the ***player*** performance during the game.

I. **The Window of the Game Design:**

   The window is divided into two divisions (i.e Left for Player 1 - Right for Player 2) (as shown in **figure 1**). Each division consists of a Large and a small shelves from where the plates are falling on the player at the bottom end of the division.

The player's possible moves are right or left to collect the one of the falling plates.

Figure 1

II . The Software Design:

Packages:

The packages are divided into **3** packages the **1st** group for the view, the **2nd** group is for the Model and the **3rd** is for controlling the game flow and the synchronization between the plates and the players.

Classes :

Plates Classes:

Plate:

Is the parent class of all plates of all shapes used in the game and all the plates extend this class.

PlateDynamicLoader:

Is the class which makes the players to load dynamically the classes of various plates shapes before the start of the game.

PlateFactory:

Is the factory of all types of plates dynamically loaded before the game and the criterion of choosing the plates is random.

PlatePool:

Is the second creator of plates after the factory. Its main role is saving the plates which fell over the players and the player did not catch them by his stick so they are stored inside a data structure inside the pool as reusable objects to be used again by the **PlateFactory.**

ProjectingPlate:

Is a class extending **Thread** class**.** It is responsible for managing the process of falling the plate from the shelf over the clown. The game designers designed the falling path of plates to follow the equation of *Free Projectiles* in mechanics.

PlateProducer:

Is a class extending **Thread** class. It is responsible for the Producer part of (Producer-Consumer) technique for managing the synchronization between the producing and consuming of plates.

PlateConsumer:

Is a class extending **Thread** class. It is responsible for the Consumer part of (Producer-Consumer) technique for managing the synchronization between the producing and consuming of plates.

Processor:

It is considered the most important class in the (Producer-Consumer) technique because it manages both **PlateProducer** and **PlateConsumer** classes by implementing the two main method in this technique (produce or project and consume).

Players Classes:

Player:

A class implementing runnable interface. Is responsible for the player thread and his movement and the movement of the plates with the player on the stick.

PlayerControl:

Is a class extending **Thread** class. It is mainly responsible for processing the collisions of the plates with the player stick. To calculate the score.

Difficulty Classes:

DifficultStrategy - MediumStrategy - EasyStrategy:

Implement Strategy Interface which is part of strategy design pattern managing the difficulty strategy.

Some Other Classes:

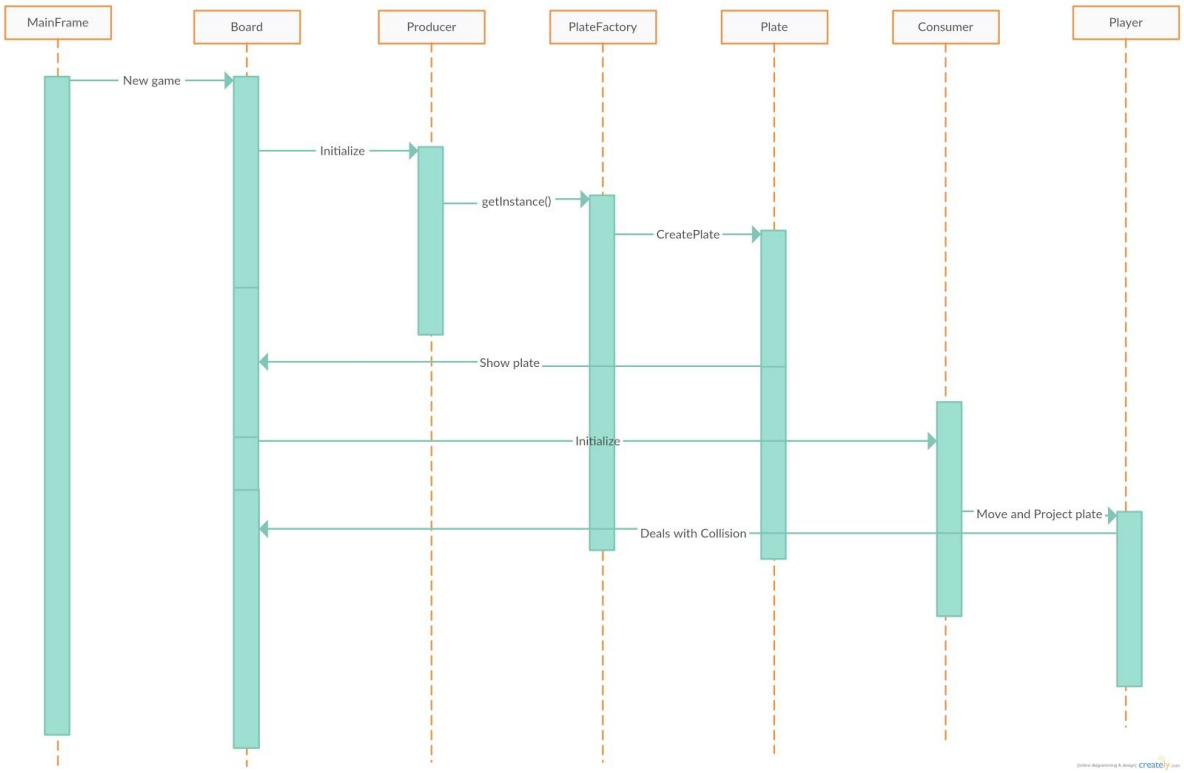
KeyHandler:

Responsible for key listening to make a right move when right arrow key is pressed and similarly with the left arrow key.

JSON:

Responsible for saving current game in JSON file format to be reloaded.

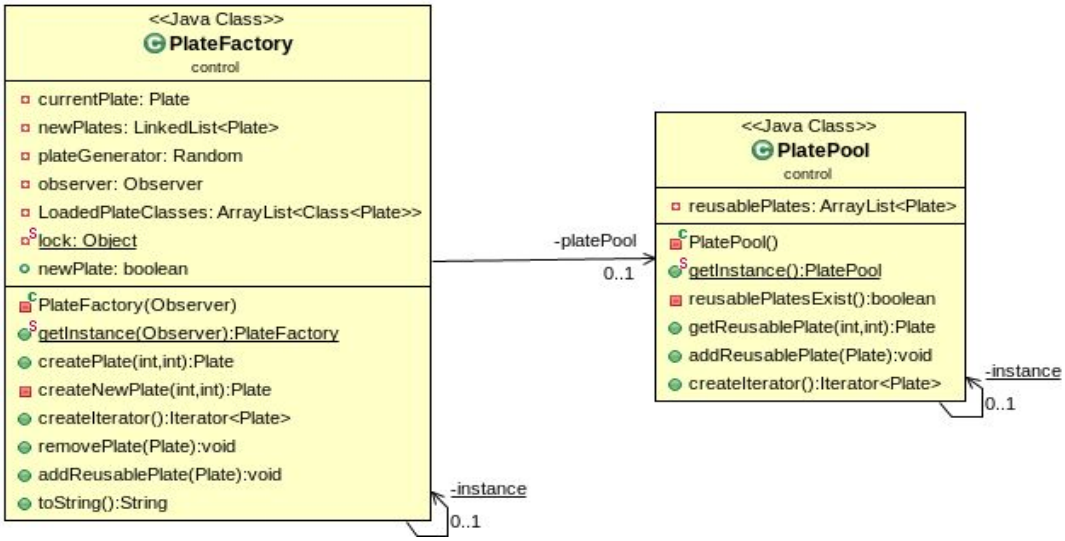
Sequence Diagram:



Design Patterns:

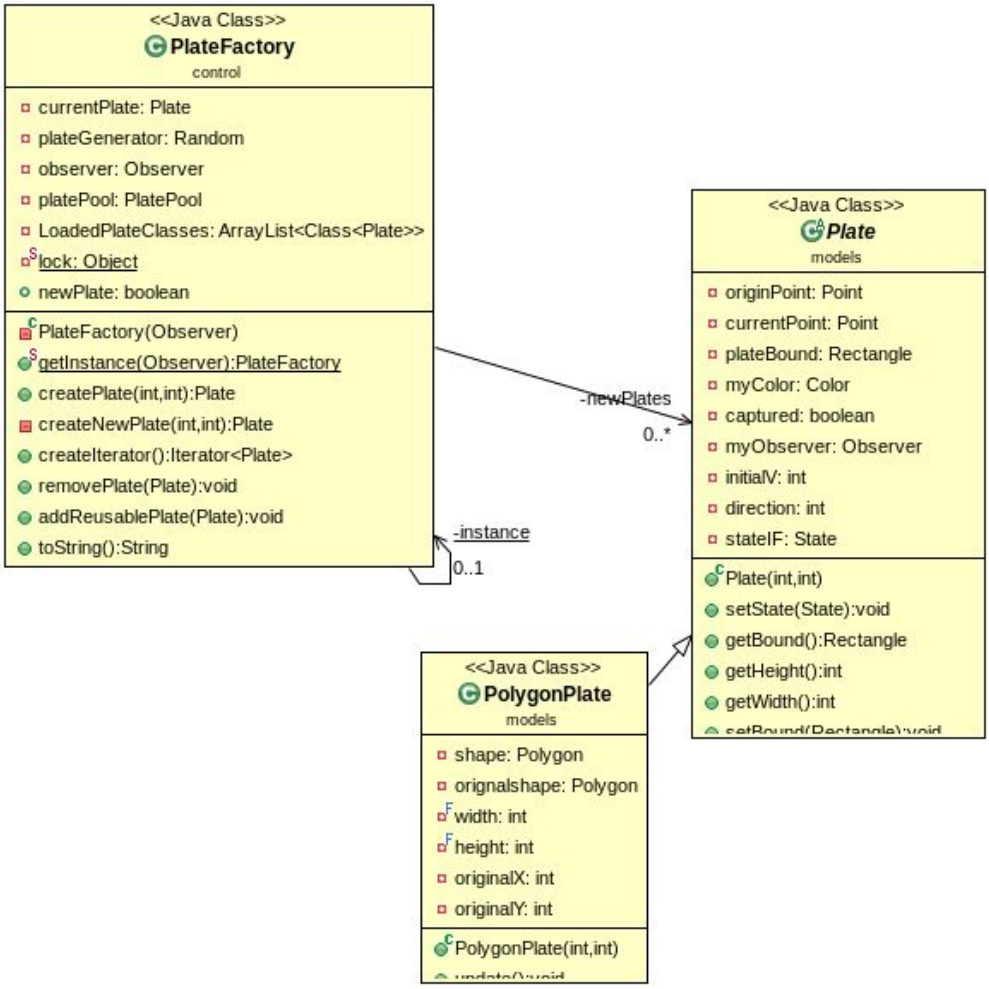
1. **Singleton:**

   Is used in several classes of the project which the designers found that they want only one instance from it. The UML class diagram shows this properly. The class names : PLateFactory - PlatePool - PlateDynamicLoader.



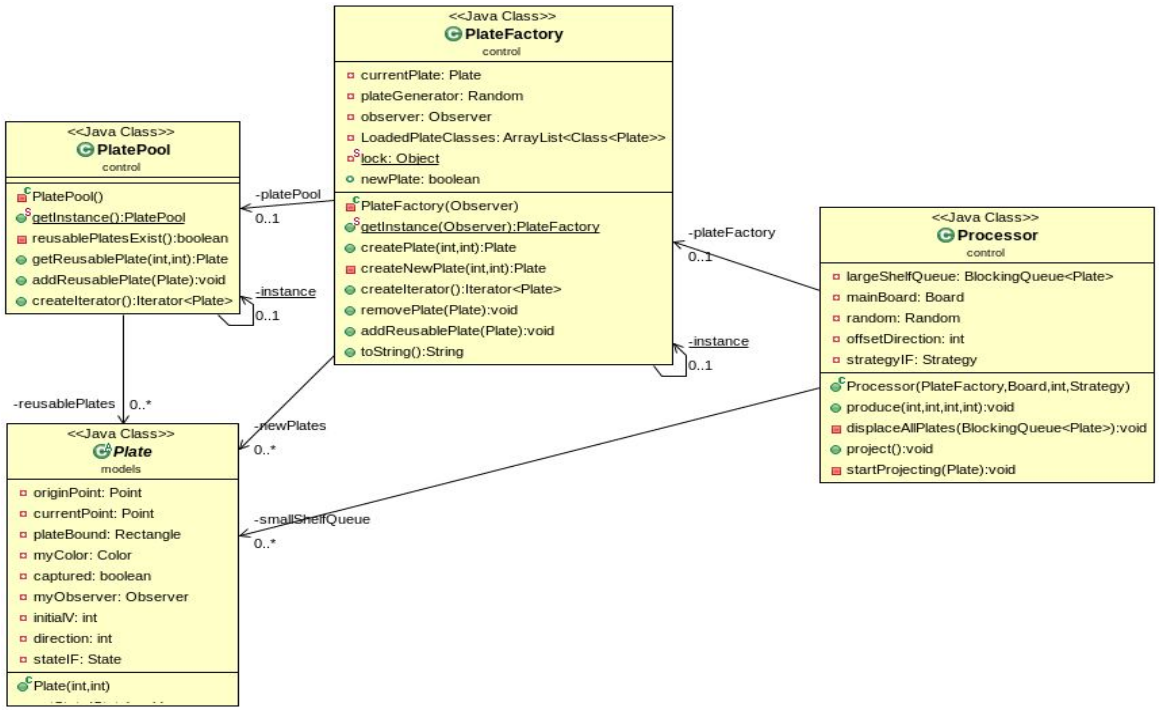
2. Factory:

The designers made a PlateFactory class which creates the plates that fall over the players continuously. The reason for making Factory is hiding the creation logic from the client. And the UML diagram shows that.



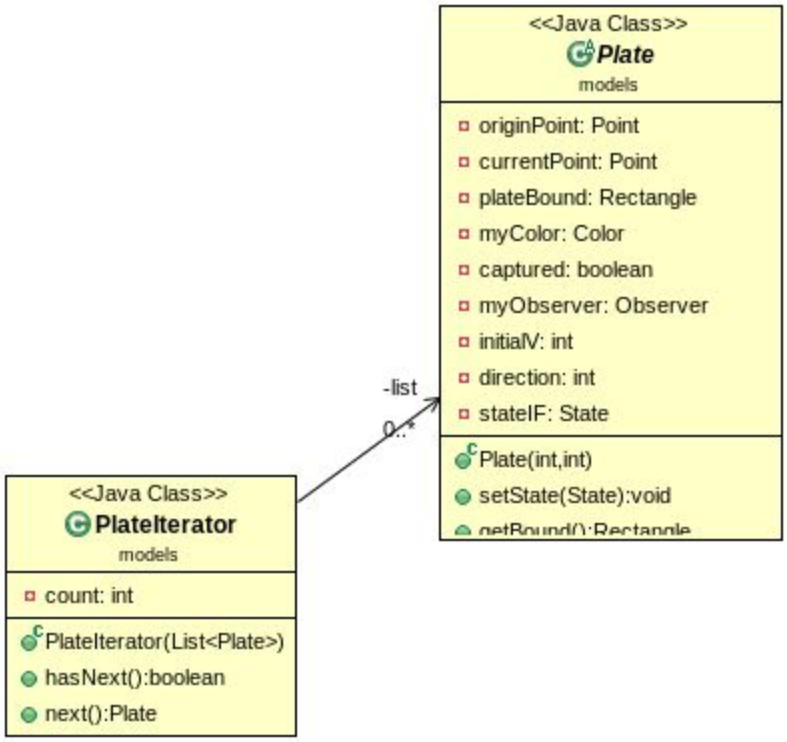
3. **Object Pool:**

The designers made PlatePool class which is responsible for storing the plates which did not take part in collision with the player's stick to be reused again and fall again from the shelves. So the factory always asks the Pool if it has reusable Plates or not to be used instead of created new ones.



4. Iterator:

The designers made their own PlateIterator interface to be implemented by any Collection containing Plate objects. The UML shown PlateIterator interface.



5. Dynamic Linkage:

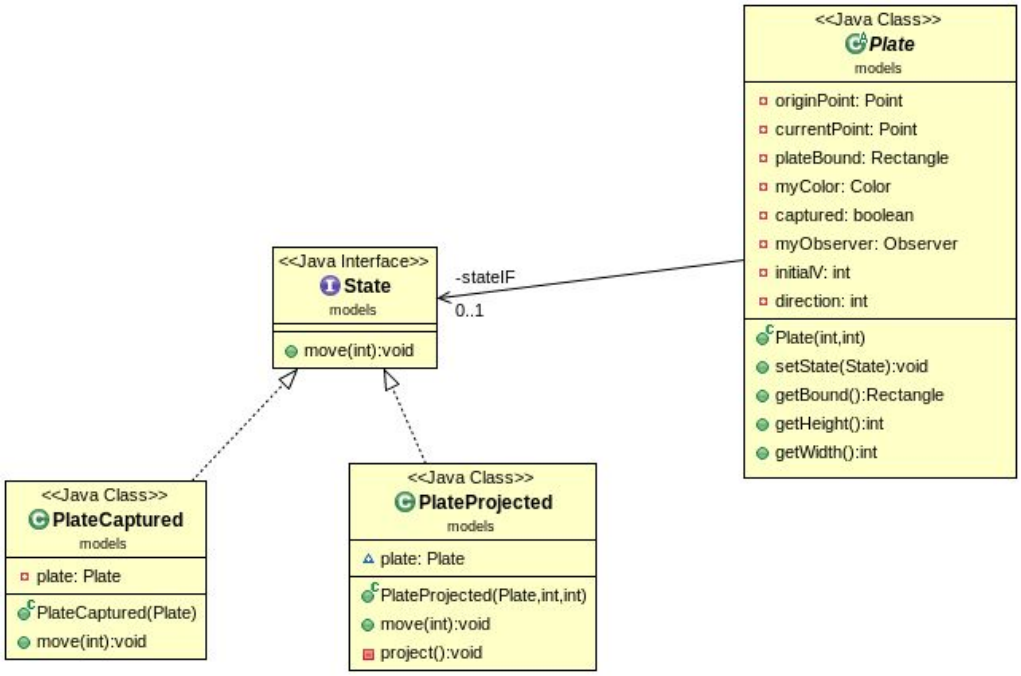
The designers used this pattern to fulfil the dynamic loading of Plates Classes before the start of the game. The loaded classes must extend the Plate class to have all the attributes of the Abstract Plate to be used inside the project without any problems.

6. Snapshot:

The designers used snapshot design pattern to save the states of the players during JSON saving operation.

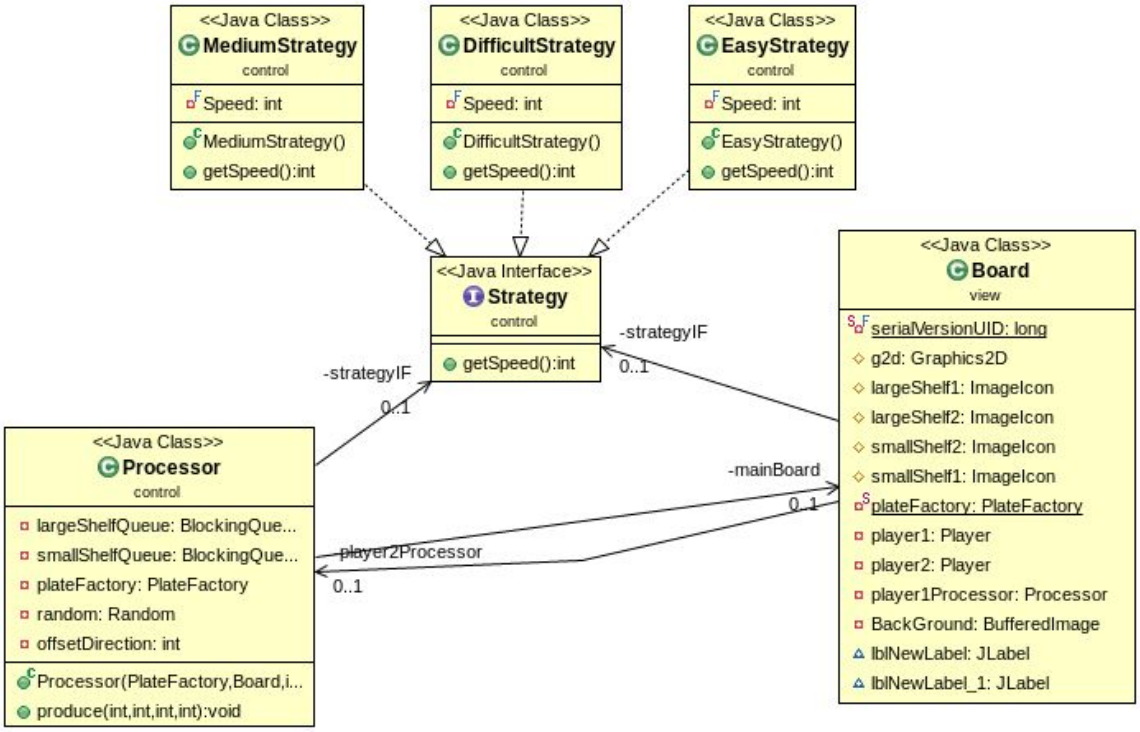
7.State:

This patterns is used for the plates state. The plates can be either projected or captured by the player so it preserve its state during the game until it is changed.



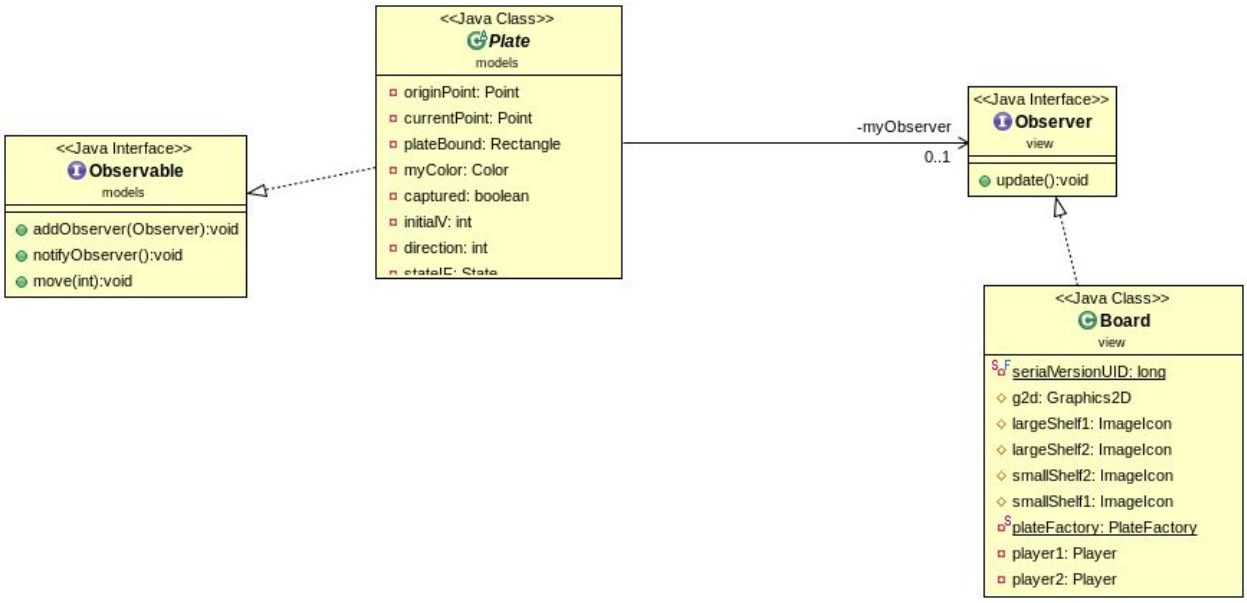
8.Strategy:

This pattern is used to choose the difficulty strategy at which the game will be played. The game has 3 difficulty strategy (Easy - Medium - Hard).



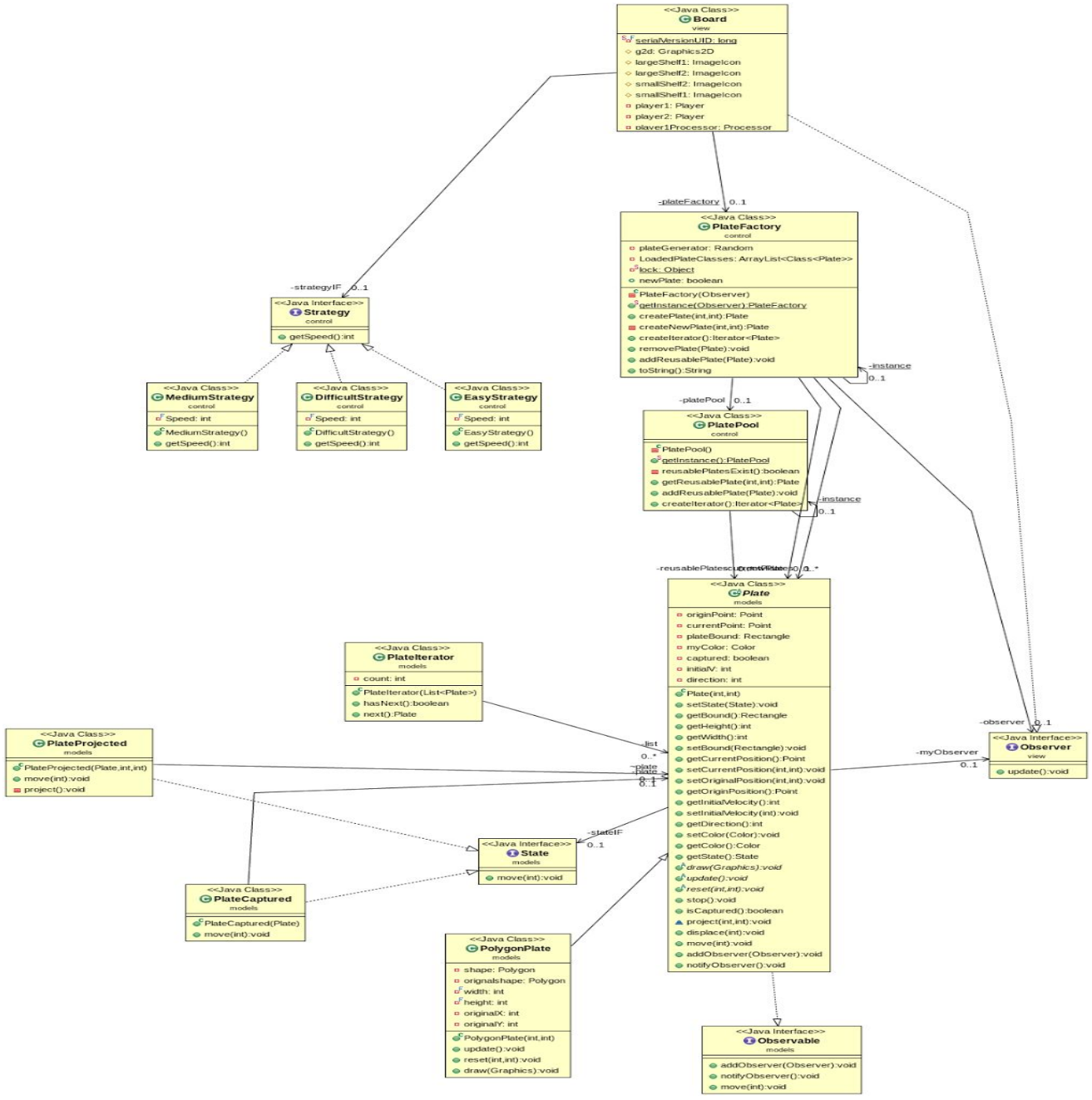
9.Observer:

The MainBoard is considered an observer to the falling plates i.e: is updated on every changes happens to the plates during movement.

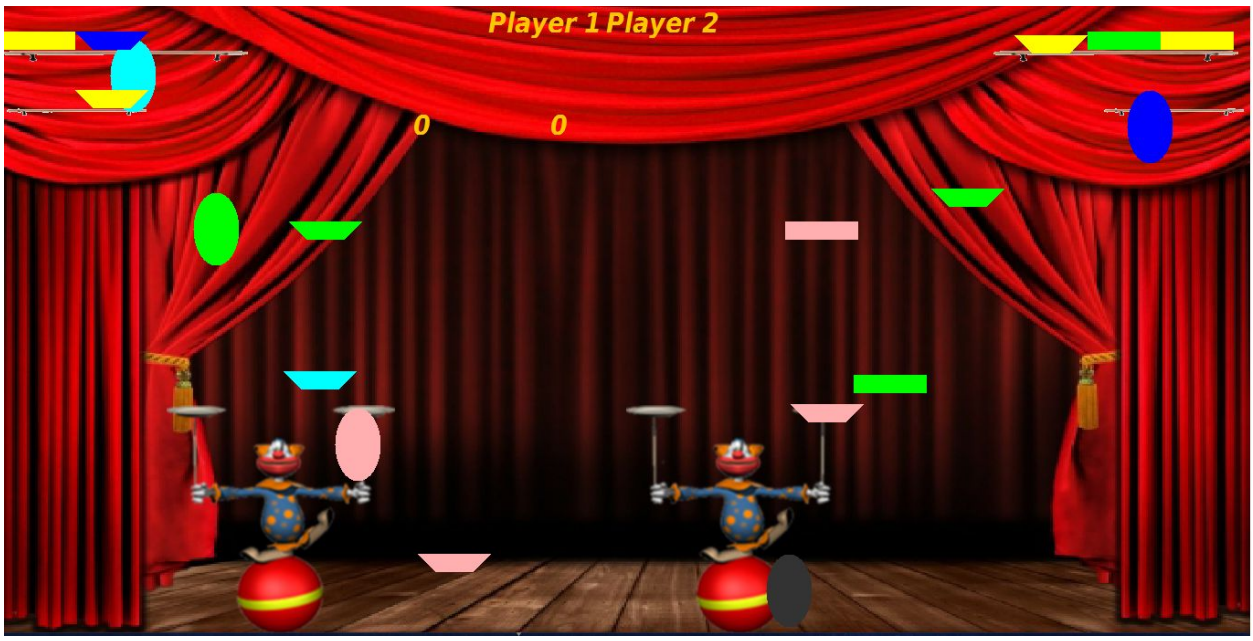


10.MVC:

The Model-View-Controller dominates all the project and divides its classes into 3 main packages for each one of its 3 parts. The **Model**: Contains the logic classes which is responsible for the plates methods of movement and players methods of collision . The **View**: Contains the classes of UI which is responsible for viewing the game e.g: MainFrame Class and Board Class. The **Controller**: contains the plates Factory, Pool ,processor ,producer and consumer classes and difficulty strategies.


<<Java Class>>
Board
view
serialVersionUID: long
g2d: Graphics2D
largeShelf1: ImageIcon
largeShelf2: ImageIcon
smallShelf2: ImageIcon
smallShelf1: ImageIcon
player1: Player
player2: Player
player1Processor: Processor
-plateFactory
0..1
<<Java Class>>
PlateFactory
control
plateGenerator: Random
LoadedPlateClasses: ArrayList<Class<Plate>>
lock: Object
newPlate: boolean
PlateFactory(Observer)
getInstance(Observer):PlateFactory
createPlate(int,int):Plate
createNewPlate(int,int):Plate
createIterator():Iterator<Plate>
removePlate(Plate):void
addReusablePlate(Plate):void
toString():String
-instance
0..1
-strategyIF
0..1
<<Java Interface>>
Strategy
control
getSpeed():int
<<Java Class>>
MediumStrategy
control
Speed: int
MediumStrategy()
getSpeed():int
<<Java Class>>
DifficultStrategy
control
Speed: int
DifficultStrategy()
getSpeed():int
<<Java Class>>
EasyStrategy
control
Speed: int
EasyStrategy()
getSpeed():int
-platePool
0..1
<<Java Class>>
PlatePool
control
PlatePool()
getInstance():PlatePool
reusablePlatesExist():boolean
getReusablePlate(int,int):Plate
addReusablePlate(Plate):void
createIterator():Iterator<Plate>
-instance
0..1
-reusablePlates
0..*
<<Java Class>>
Plate
models
originPoint: Point
currentPoint: Point
plateBound: Rectangle
myColor: Color
captured: boolean
initialV: int
direction: int
Plate(int,int)
setState(State):void
getBound():Rectangle
getHeight():int
getWidth():int
setBound(Rectangle):void
getCurrentPosition():Point
setCurrentPosition(int,int):void
setOriginalPosition(int,int):void
getOriginPosition():Point
getInitialVelocity():int
setInitialVelocity(int):void
getDirection():int
setColor(Color):void
getColor():Color
getState():State
draw(Graphics):void
update():void
reset(int,int):void
stop():void
isCaptured():boolean
project(int,int):void
displace(int):void
move(int):void
addObserver(Observer):void
notifyObserver():void
<<Java Class>>
PlateIterator
models
count: int
PlateIterator(List<Plate>)
hasNext():boolean
next():Plate
list
0..*
<<Java Class>>
PlateProjected
models
PlateProjected(Plate,int,int)
move(int):void
project():void
-plate
-plate
0..1
0..1
-observer
0..1
<<Java Interface>>
Observer
view
update():void
-myObserver
0..1
-stateIF
0..1
<<Java Interface>>
State
models
move(int):void
<<Java Class>>
PlateCaptured
models
PlateCaptured(Plate)
move(int):void
<<Java Class>>
PolygonPlate
models
shape: Polygon
orignalshape: Polygon
width: int
height: int
originalX: int
originalY: int
PolygonPlate(int,int)
update():void
reset(int,int):void
draw(Graphics):void
<<Java Interface>>
Observable
models
addObserver(Observer):void
notifyObserver():void
move(int):void

SnapShots of GUI:



User Guide:

1. The user must load at least one shape of plates before clicking new game. The Plates files are of JAR extension.
2. Before clicking **new game** button the user has three levels of difficulties **Easy, Medium** and **Hard**. or the game will use the default which is **Medium**.There are different criteria for difficulties e.g: Speed of Plates Falling and their falling rate per unit time.
3. When the game begins each one of the two player must control his clown through Arrow keys in the keyboard for player 1 and the mouse for player 2.
4. Each clown of some player can move right or left only. Players try to catch the falling plates, if they manage to collect three consecutive plates of the same color, then they are vanished and their score increases.
5. The player who has higher score before the end of the game wins the game.