Circus of Plates – Game

Participants: -

1 - Omar Ahmed Fouad Hassan Wasfy.

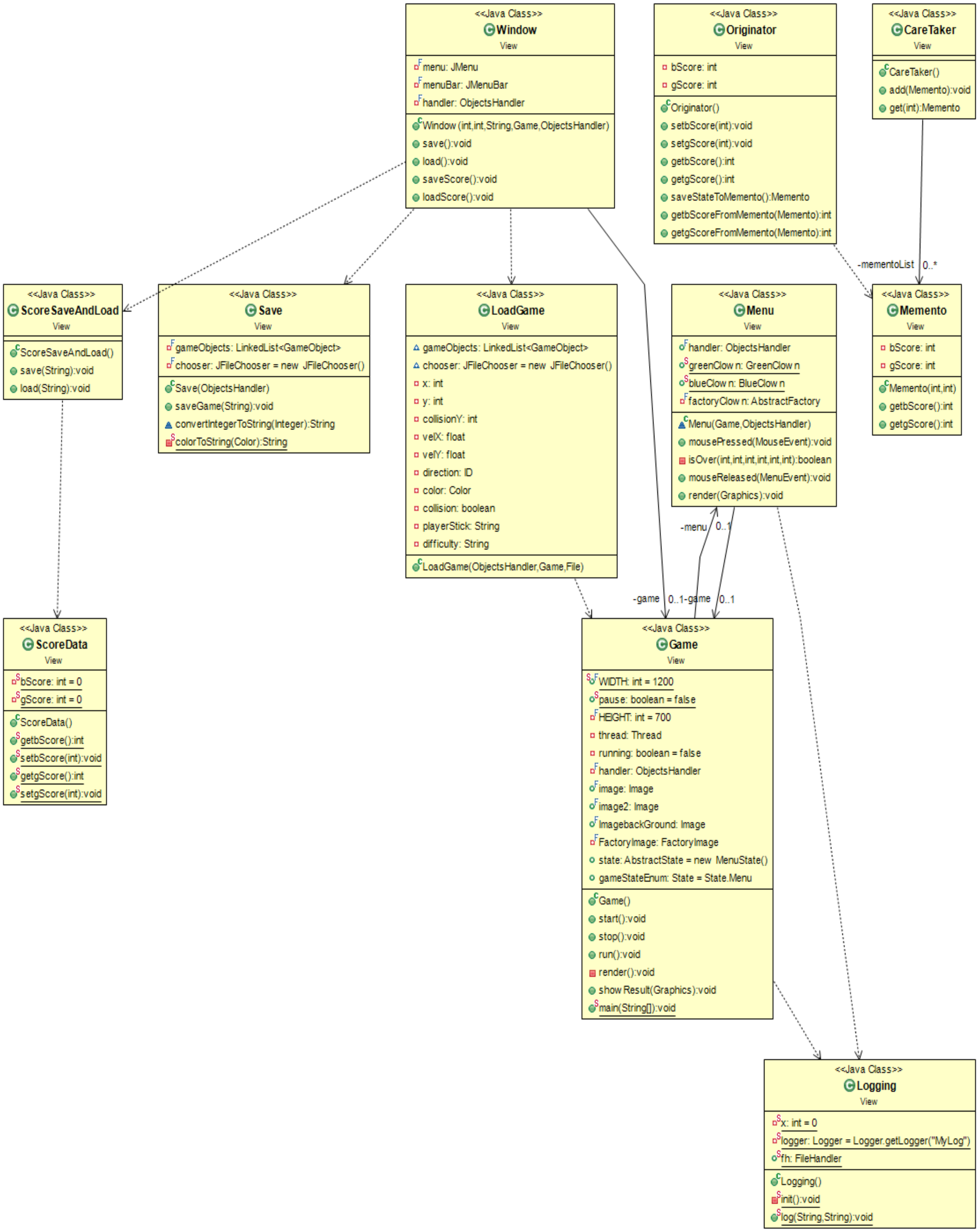
2 - Sherif Mohamed Abd El-Rahman.

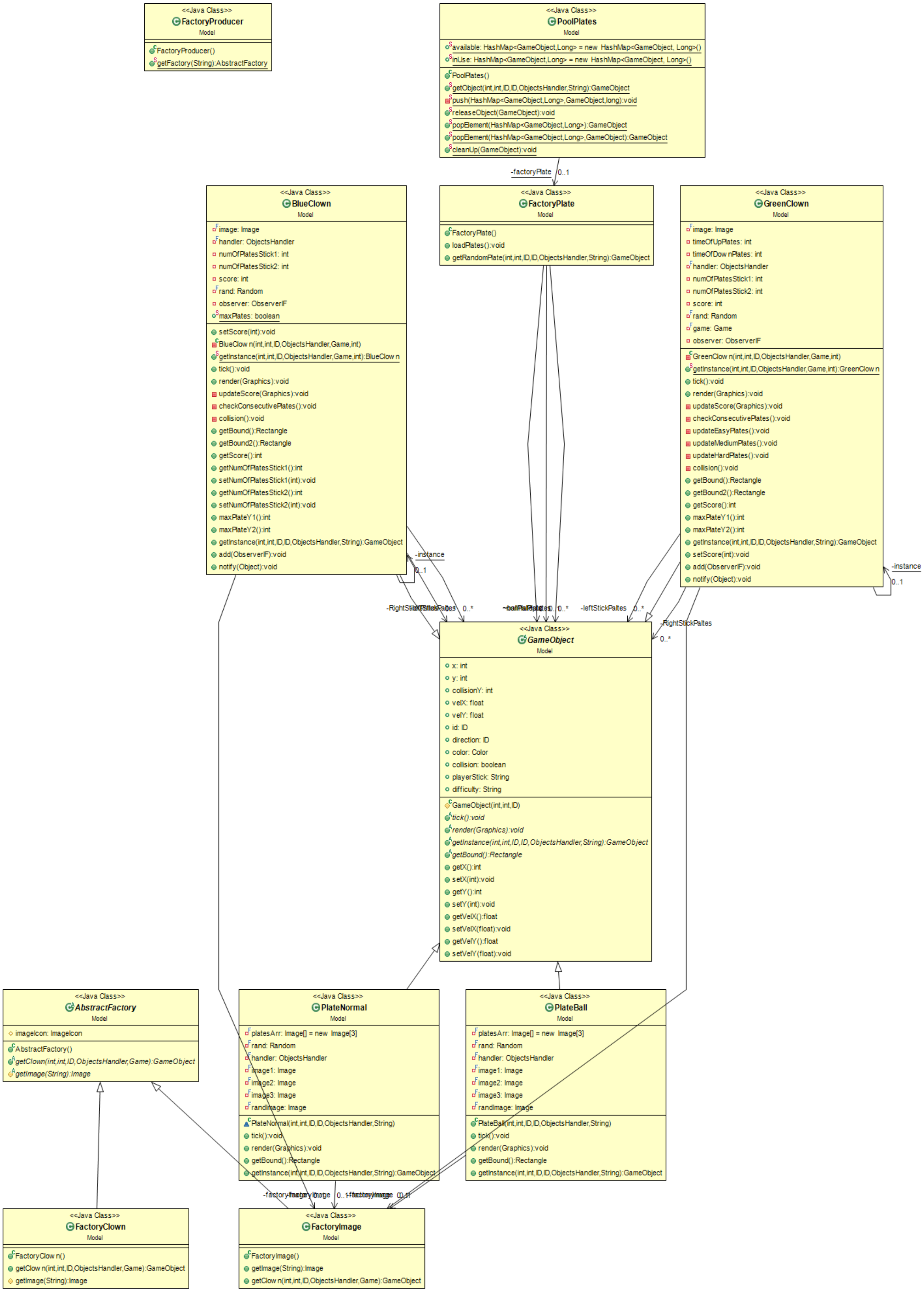
3 – Mostafa Tarek Ibrahim Mahmoud.

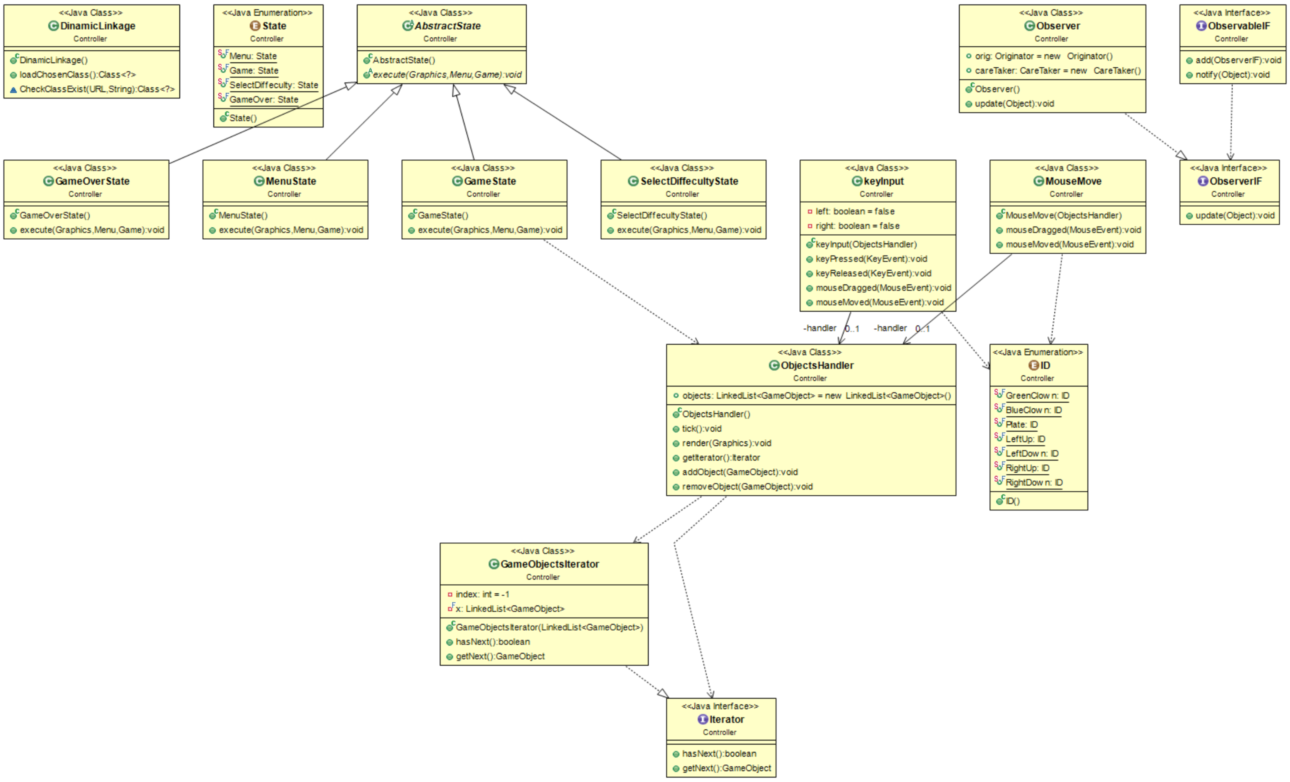
The description of design:-

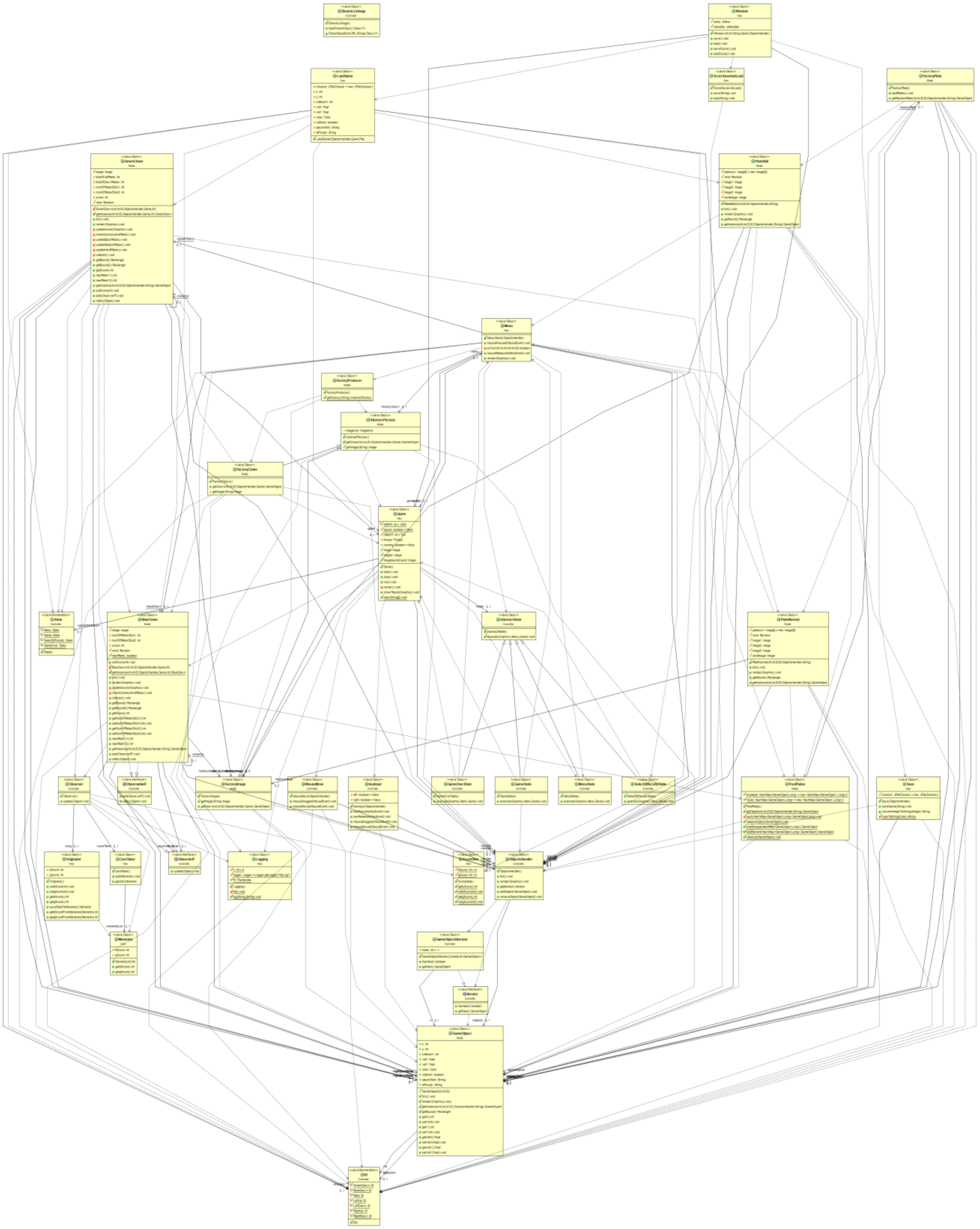
We used most of the design patterns that are recommended in the assignment , at all our design is based on how to make more than one object make an action at the same time so that we have to use threads in our design and we used in our project an interfaces, classes, abstract classes and enums .

Class diagram:-



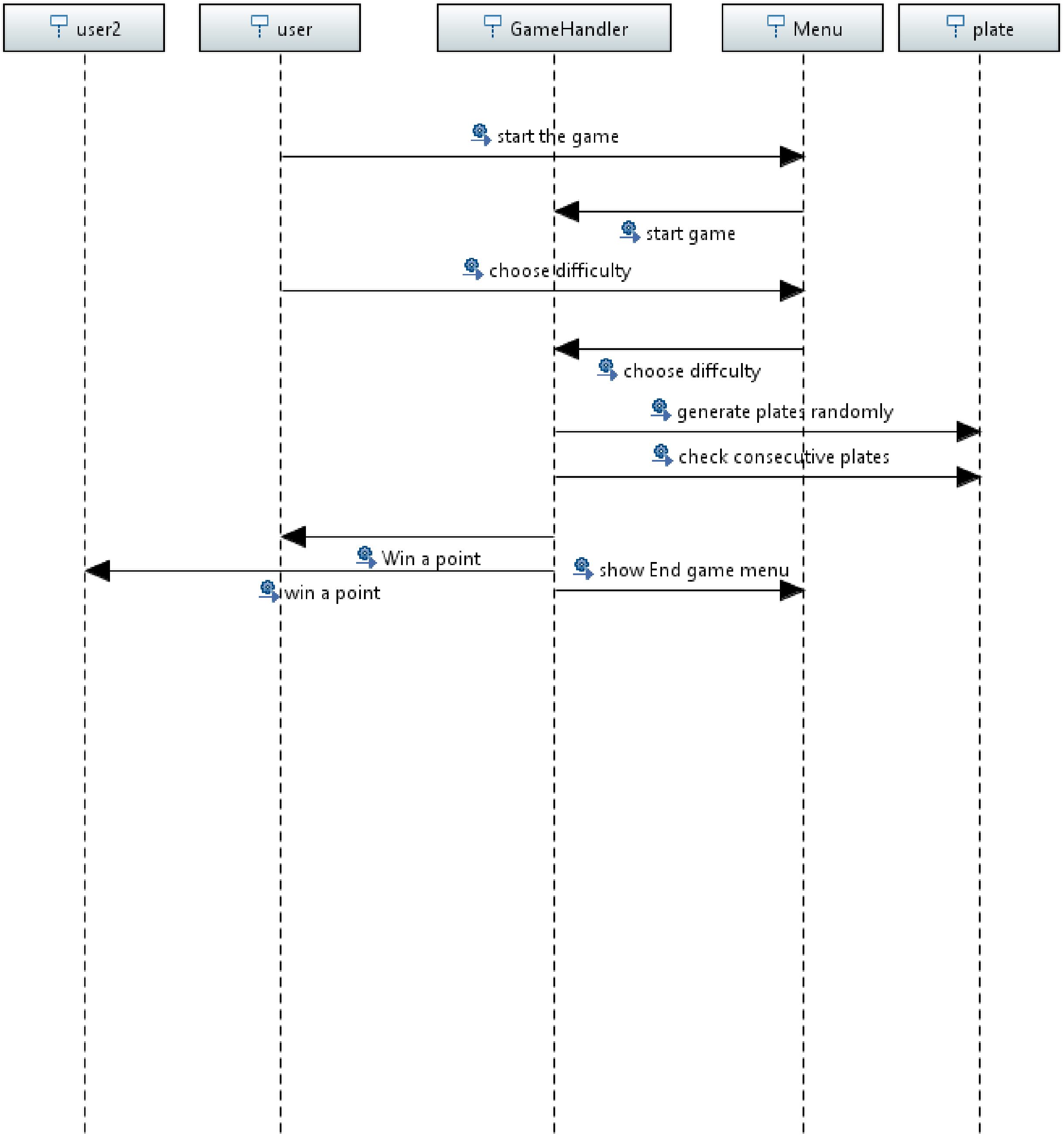

<<Java Class>>
FactoryProducer
Model
FactoryProducer()
getFactory(String):AbstractFactory
<<Java Class>>
PoolPlates
Model
available: HashMap<GameObject,Long> = new HashMap<GameObject, Long>()
inUse: HashMap<GameObject,Long> = new HashMap<GameObject, Long>()
PoolPlates()
getObject(int,int,ID,ID,ObjectsHandler,String):GameObject
push(HashMap<GameObject,Long>,GameObject,long):void
releaseObject(GameObject):void
popElement(HashMap<GameObject,Long>):GameObject
popElement(HashMap<GameObject,Long>,GameObject):GameObject
cleanUp(GameObject):void
-factoryPlate
0..1
<<Java Class>>
BlueClown
Model
image: Image
handler: ObjectsHandler
numOfPlatesStick1: int
numOfPlatesStick2: int
score: int
rand: Random
observer: ObserverIF
maxPlates: boolean
setScore(int):void
BlueClown(int,int,ID,ObjectsHandler,Game,int)
getInstance(int,int,ID,ObjectsHandler,Game,int):BlueClown
tick():void
render(Graphics):void
updateScore(Graphics):void
checkConsecutivePlates():void
collision():void
getBound():Rectangle
getBound2():Rectangle
getScore():int
getNumOfPlatesStick1():int
setNumOfPlatesStick1(int):void
getNumOfPlatesStick2():int
setNumOfPlatesStick2(int):void
maxPlateY1():int
maxPlateY2():int
getInstance(int,int,ID,ID,ObjectsHandler,String):GameObject
add(ObserverIF):void
notify(Object):void
<<Java Class>>
FactoryPlate
Model
FactoryPlate()
loadPlates():void
getRandomPlate(int,int,ID,ID,ObjectsHandler,String):GameObject
<<Java Class>>
GreenClown
Model
image: Image
timeOfUpPlates: int
timeOfDownPlates: int
handler: ObjectsHandler
numOfPlatesStick1: int
numOfPlatesStick2: int
score: int
rand: Random
game: Game
observer: ObserverIF
GreenClown(int,int,ID,ObjectsHandler,Game,int)
getInstance(int,int,ID,ObjectsHandler,Game,int):GreenClown
tick():void
render(Graphics):void
updateScore(Graphics):void
checkConsecutivePlates():void
updateEasyPlates():void
updateMediumPlates():void
updateHardPlates():void
collision():void
getBound():Rectangle
getBound2():Rectangle
getScore():int
maxPlateY1():int
maxPlateY2():int
getInstance(int,int,ID,ID,ObjectsHandler,String):GameObject
setScore(int):void
add(ObserverIF):void
notify(Object):void
-instance
0..1
-instance
0..1
-RightStickPaltes
0..*
0..*
0..*
-leftStickPaltes
0..*
-RightStickPaltes
0..*
<<Java Class>>
GameObject
Model
x: int
y: int
collisionY: int
velX: float
velY: float
id: ID
direction: ID
color: Color
collision: boolean
playerStick: String
difficulty: String
GameObject(int,int,ID)
tick():void
render(Graphics):void
getInstance(int,int,ID,ID,ObjectsHandler,String):GameObject
getBound():Rectangle
getX():int
setX(int):void
getY():int
setY(int):void
getVelX():float
setVelX(float):void
getVelY():float
setVelY(float):void
<<Java Class>>
AbstractFactory
Model
imageIcon: ImageIcon
AbstractFactory()
getClown(int,int,ID,ObjectsHandler,Game):GameObject
getImage(String):Image
<<Java Class>>
PlateNormal
Model
platesArr: Image[] = new Image[3]
rand: Random
handler: ObjectsHandler
image1: Image
image2: Image
image3: Image
randImage: Image
PlateNormal(int,int,ID,ID,ObjectsHandler,String)
tick():void
render(Graphics):void
getBound():Rectangle
getInstance(int,int,ID,ID,ObjectsHandler,String):GameObject
<<Java Class>>
PlateBall
Model
platesArr: Image[] = new Image[3]
rand: Random
handler: ObjectsHandler
image1: Image
image2: Image
image3: Image
randImage: Image
PlateBall(int,int,ID,ID,ObjectsHandler,String)
tick():void
render(Graphics):void
getBound():Rectangle
getInstance(int,int,ID,ID,ObjectsHandler,String):GameObject
-factoryImage
0..1
-factoryImage
0..1
<<Java Class>>
FactoryClown
Model
FactoryClown()
getClown(int,int,ID,ObjectsHandler,Game):GameObject
getImage(String):Image
<<Java Class>>
FactoryImage
Model
FactoryImage()
getImage(String):Image
getClown(int,int,ID,ObjectsHandler,Game):GameObject


<<Java Class>>
DinamicLinkage
Controller
DinamicLinkage()
loadChosenClass():Class<?>
CheckClassExist(URL,String):Class<?>
<<Java Enumeration>>
State
Controller
Menu: State
Game: State
SelectDiffeculty: State
GameOver: State
State()
<<Java Class>>
AbstractState
Controller
AbstractState()
execute(Graphics,Menu,Game):void
<<Java Class>>
Observer
Controller
orig: Originator = new Originator()
careTaker: CareTaker = new CareTaker()
Observer()
update(Object):void
<<Java Interface>>
ObservableIF
Controller
add(ObserverIF):void
notify(Object):void
<<Java Class>>
GameOverState
Controller
GameOverState()
execute(Graphics,Menu,Game):void
<<Java Class>>
MenuState
Controller
MenuState()
execute(Graphics,Menu,Game):void
<<Java Class>>
GameState
Controller
GameState()
execute(Graphics,Menu,Game):void
<<Java Class>>
SelectDiffecultyState
Controller
SelectDiffecultyState()
execute(Graphics,Menu,Game):void
<<Java Class>>
keyInput
Controller
left: boolean = false
right: boolean = false
keyInput(ObjectsHandler)
keyPressed(KeyEvent):void
keyReleased(KeyEvent):void
mouseDragged(MouseEvent):void
mouseMoved(MouseEvent):void
<<Java Class>>
MouseMove
Controller
MouseMove(ObjectsHandler)
mouseDragged(MouseEvent):void
mouseMoved(MouseEvent):void
<<Java Interface>>
ObserverIF
Controller
update(Object):void
-handler
0..1
-handler
0..1
<<Java Class>>
ObjectsHandler
Controller
objects: LinkedList<GameObject> = new LinkedList<GameObject>()
ObjectsHandler()
tick():void
render(Graphics):void
getIterator():Iterator
addObject(GameObject):void
removeObject(GameObject):void
<<Java Enumeration>>
ID
Controller
GreenClown: ID
BlueClown: ID
Plate: ID
LeftUp: ID
LeftDown: ID
RightUp: ID
RightDown: ID
ID()
<<Java Class>>
GameObjectsIterator
Controller
index: int = -1
x: LinkedList<GameObject>
GameObjectsIterator(LinkedList<GameObject>)
hasNext():boolean
getNext():GameObject
<<Java Interface>>
Iterator
Controller
hasNext():boolean
getNext():GameObject



<<Java Class>> DinamicLinkage Controller
<<Java Class>> Window View
<<Java Class>> LoadGame View
<<Java Class>> ScoreSaveAndLoad View
<<Java Class>> FactoryPlate Model
<<Java Class>> GreenClown Model
<<Java Class>> PlateBall Model
<<Java Class>> Menu View
<<Java Class>> FactoryProducer Model
<<Java Class>> AbstractFactory Model
<<Java Class>> FactoryClown Model
<<Java Class>> Game View
<<Java Enumeration>> State Controller
<<Java Class>> BlueClown Model
<<Java Class>> AbstractState Controller
<<Java Class>> PlateNormal Model
<<Java Class>> Observer Controller
<<Java Interface>> ObservableIF Controller
<<Java Class>> FactoryImage Model
<<Java Class>> MouseMove Controller
<<Java Class>> keyInput Controller
<<Java Class>> GameOverState Controller
<<Java Class>> GameState Controller
<<Java Class>> MenuState Controller
<<Java Class>> SelectDifficultyState Controller
<<Java Class>> PoolPlates Model
<<Java Class>> Save View
<<Java Class>> Originator View
<<Java Class>> CareTaker View
<<Java Interface>> ObserverIF Controller
<<Java Class>> Logging View
<<Java Class>> ScoreData View
<<Java Class>> ObjectsHandler Controller
<<Java Class>> Memento View
<<Java Class>> GameObjectIterator Controller
<<Java Interface>> Iterator Controller
<<Java Class>> GameObject Model
<<Java Enumeration>> ID Controller

Sequence diagram



Design pattern:-

1 - At first we design an abstract factory which include two different object which represent the stone corner of this project, two different objects are clown and plates, At all we have two clown which use the same object clown and we have a lot of plates which have the same object but different in some its attributes like color,… etc.

2 – After that we have to make singleton design pattern on both clown that we create it only one but it is not available for the plates that we created it more than ones at the same time.

3 – We have to use state design pattern such that at the beginning of running game we will be at the menu then we will move to another state until reach to play.

4 – We have to use an iterator design pattern at looping for the game's object by rapid velocity in order to achieve our goal that for playing with high quality and high performance and you will find this at two method (Tick():void) and (Render():void).

5 – We have to use an observer design pattern at giving each clown its score because it will change dynamically that when three object of the same color falls one over the other so that the score will increase by one so we need to notify observers

that the score changes and at once all observers will be notified and update the right value of score for both of them.

6 – (Model – View – Controller) design pattern is considered as the big image of our work we have three package one for model , the other for view and the last for controller ,one main usage for that design pattern is to separate the logic of the game from the view

7 – We have to use Dynamic Linkage design pattern in case of loading an image to be an object of falling objects.

8 – We have to use factory design pattern when creating the clown.

9 – We have to use object pool design pattern to reduce the creation of object that you initialize the minimum and maximum size of their creations.

10 – We used strategy design pattern to differentiate between plates and clowns behavior (tick()method and render())

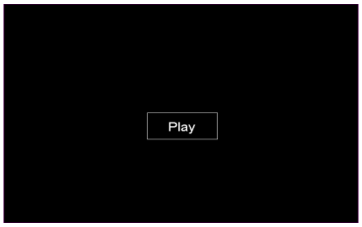
11 – We used snap shot design pattern to save two states menu state and game state and traverse between them

When needed.

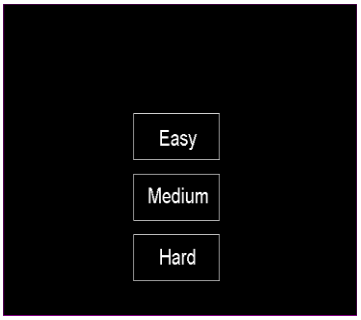
<u>User guide:-</u>

When you run the play you should follow the instruction:

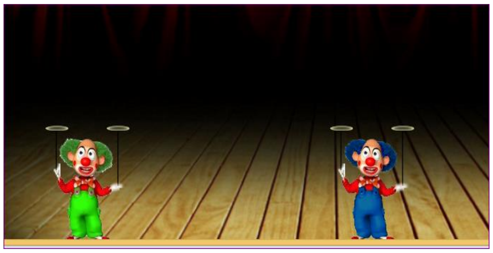
1 – Click on play button.



2 – Select the difficulty of the game.



3 – Now you have two players on the game.

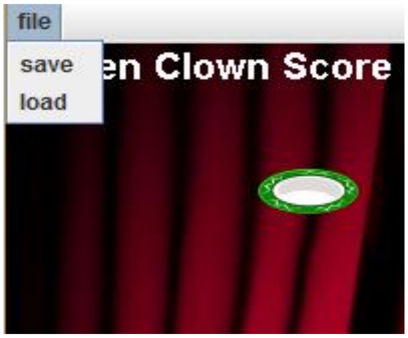


4 – First player uses mouse to move the clown right and left.

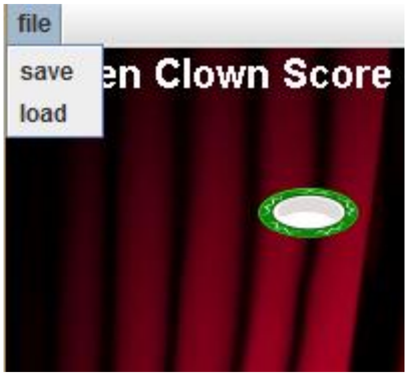
5 – Second player uses keyboard to move the clown right and left.

6 – When a player can carry three shapes with the color but sequential then they will removed from that hand and the score increase by one and so on.

7 – When you want to save the game at any location in your on computer, you have a menu at left corner of the screen then you can select from it save.

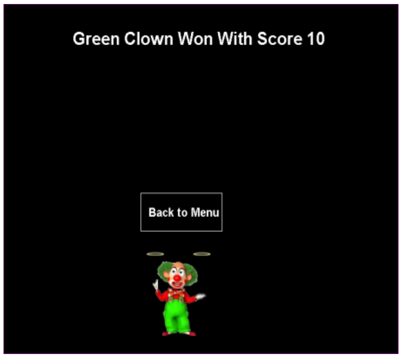


8 – When you want to load the game you saved before, you have a menu at left corner of the screen then you can select from it load.



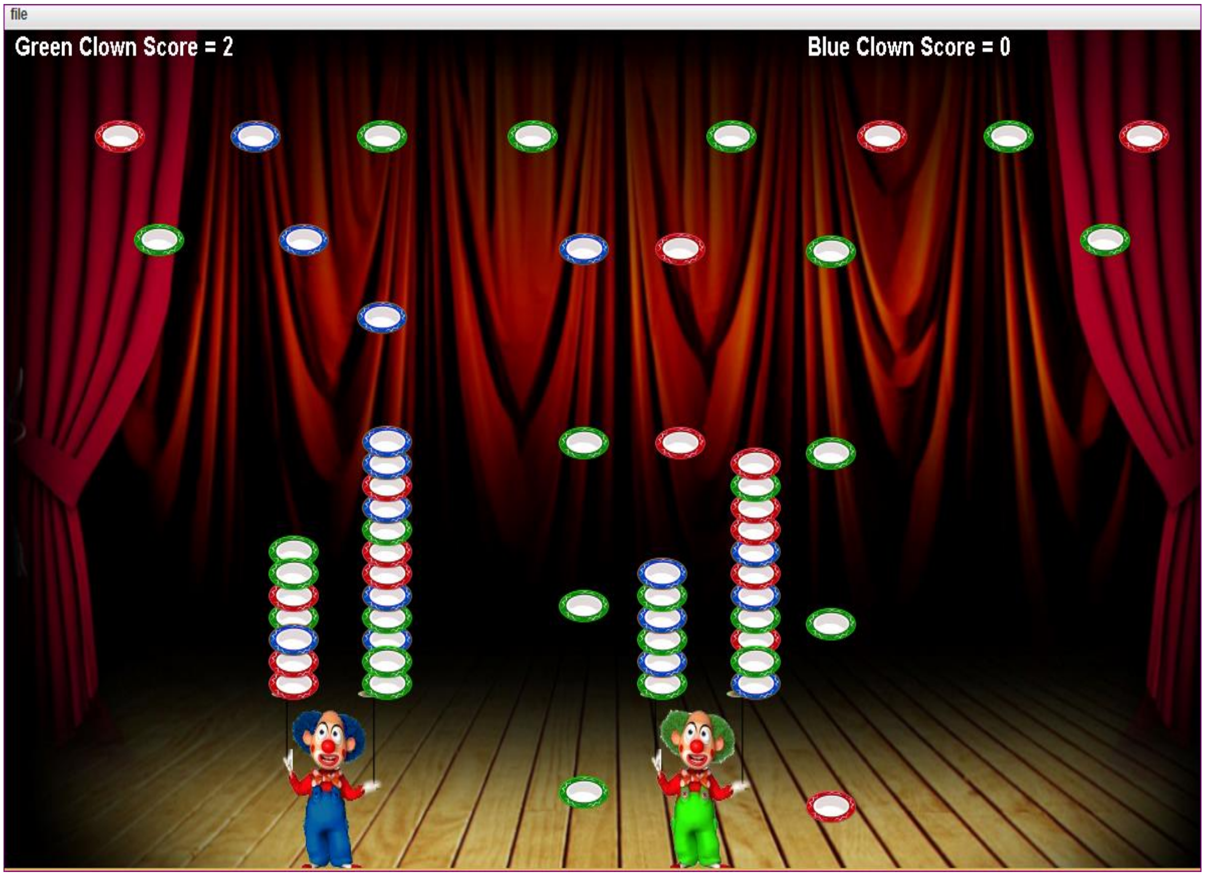
9 – When you want to pause the game click on space.

7 – When the number of the plates reaches its maximum then the maximum player's score will win.


Green Clown Won With Score 10

8 – The game will end at once.

Snapshots of GUI



Design decisions:-

1 - We decided to use singleton design pattern

In clowns classes only

2 - We decided to use snapshot design pattern to save two states menu state and select difficulty