Circus of Plates

Aly Tarek Ibrahim # 39

Mohamed Ahmed Abo el-Hassan # 55

Description of usage of design pattern in the game :

MVC :

The ***project*** can be divided into ***three*** parts . ***the model*** , ***the view*** and ***the controller*** .
the model is represented through all the classes of the project except for two classes .

One of them is named view and it contains all the members related to the view of the game , the ***GUI*** and buttons and so on .

The other class is the controller class which is responsible for controlling changes that takes place through model regarding the ***data*** or the ***behavior*** and impacting this change on the view class .

So this design pattern is responsible for the separation of these ***three components*** .

Singleton :

Singleton is used ***many times*** in the project . the ***singleton*** design pattern allows the creation of an object of a certain class ***only once*** . it is used in the creation of the **object pool** class for the implementation of the object pool design pattern ***. moreover*** , it is used In the factory class to prevent the creation of more than one factory of objects object .

Also , it is used to create one instance of the ***number generator*** class which is responsible for randomizing numbers needed for randomizing the places of creation of the objects .

Factory design pattern :

A class named ***plateFactory*** is implementing this design pattern . as it ***encapsulates*** the creation of the objects (plates) outside any class causing ***decreasing the dependency*** of classes upon each other as much as possible .

The plate factory receives some information describing the situation and so it decides which object to be created .

It worth saying that all the objects created by the ***factory*** (the plates of the game) inherits from a common superclass named shape .

State design pattern :

The ***state design*** pattern is implemented by an interface showing the behavior required through the code . a java interface named plate state is used for that .

There are other classes ***implementing*** this interface to model the different situation the plate can exist in . for example : the plate can be freely moving in the canvas or can be attached to the stick with the clown or is currently un used and is in the pool object .

The classes that implements this behavior ***encapsulates*** the process of changing ***states*** with the least possible effort and without using too much and ***unnecessary code*** .

Object pool design pattern :

This design pattern is used to boost the performance of the ***game*** . as it ***compensate*** the time and computing power needed for the re-allocation of memory fragments and processing power for creation of possible ***heavy object*** .

It keeps a pool of objects named In this project as plate pool . it is important to mention that the

The object of this plate pool is a singleton object for the easiness of the i***mplementation*** .

The object pool maintains a number of object to be created whenever needed in order to support the ***game*** .

It has a method to specify the ***upper limit*** of the number of objects it can create .

Iterator design pattern :

The ***iterator design pattern*** provides a ***common way*** to iterate through the data structure that presents in the program . it is well supported by java to maintain an iterator for any built-in data structure .

The final design of how to implement the iterator design is not specified till now .
however , we have choices , either to implement an interface that define the behavior of how to access these data structure and make several classes to implement this interface .

Or we could create only one class and create an instance of it to ***handle this issue*** without need to define the behavior of ***the iteration process*** .

Dynamic linkage :

The project contains a class for that pattern . it simply ***encapsulates the run-time*** loading of external plate classes for creating ***new shapes and instance*** of the plate object .

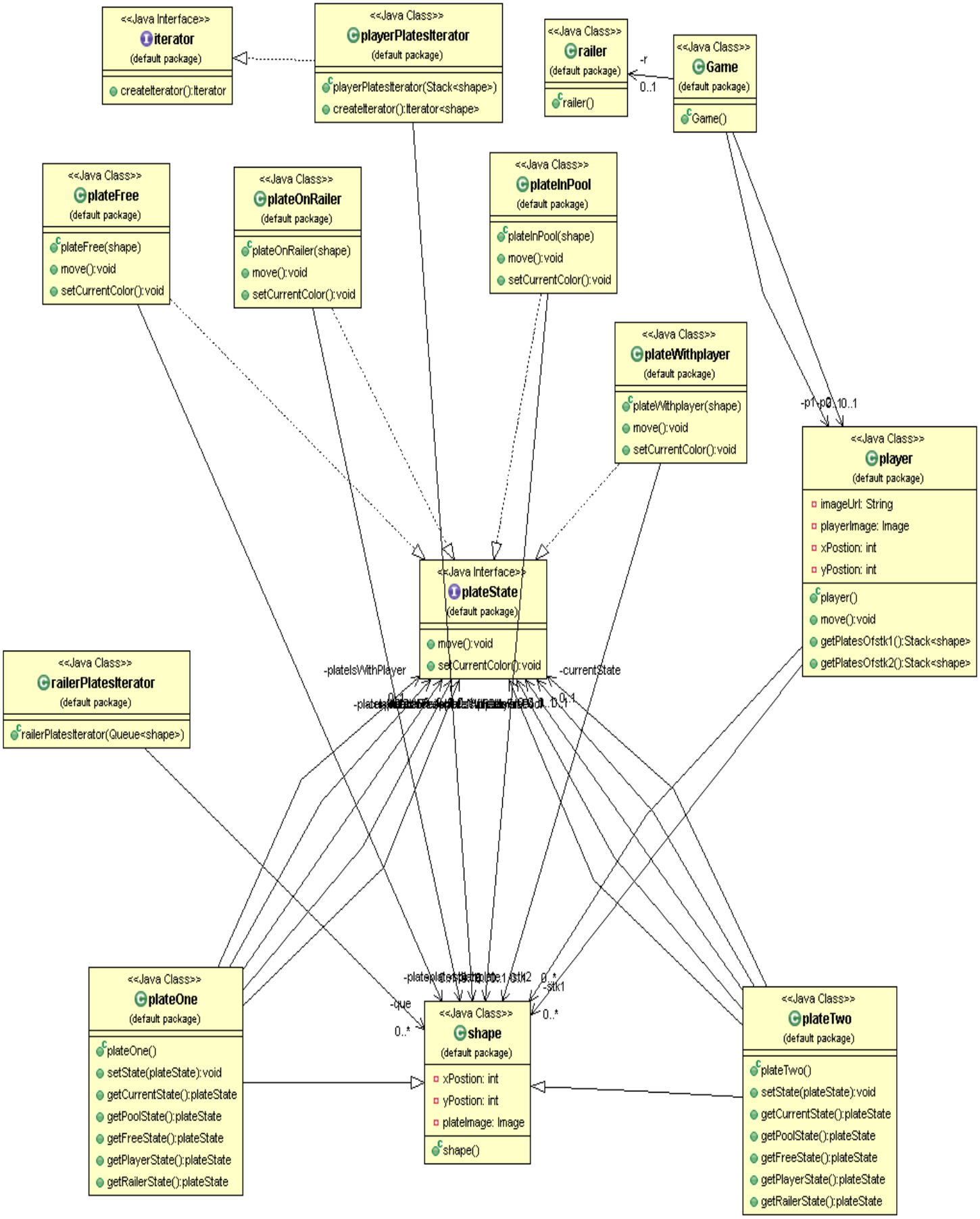
The dynamic linkage class has all the methods sufficient to refactor the loaded class , understands it and creating the ***desired run-time decidable*** objects .

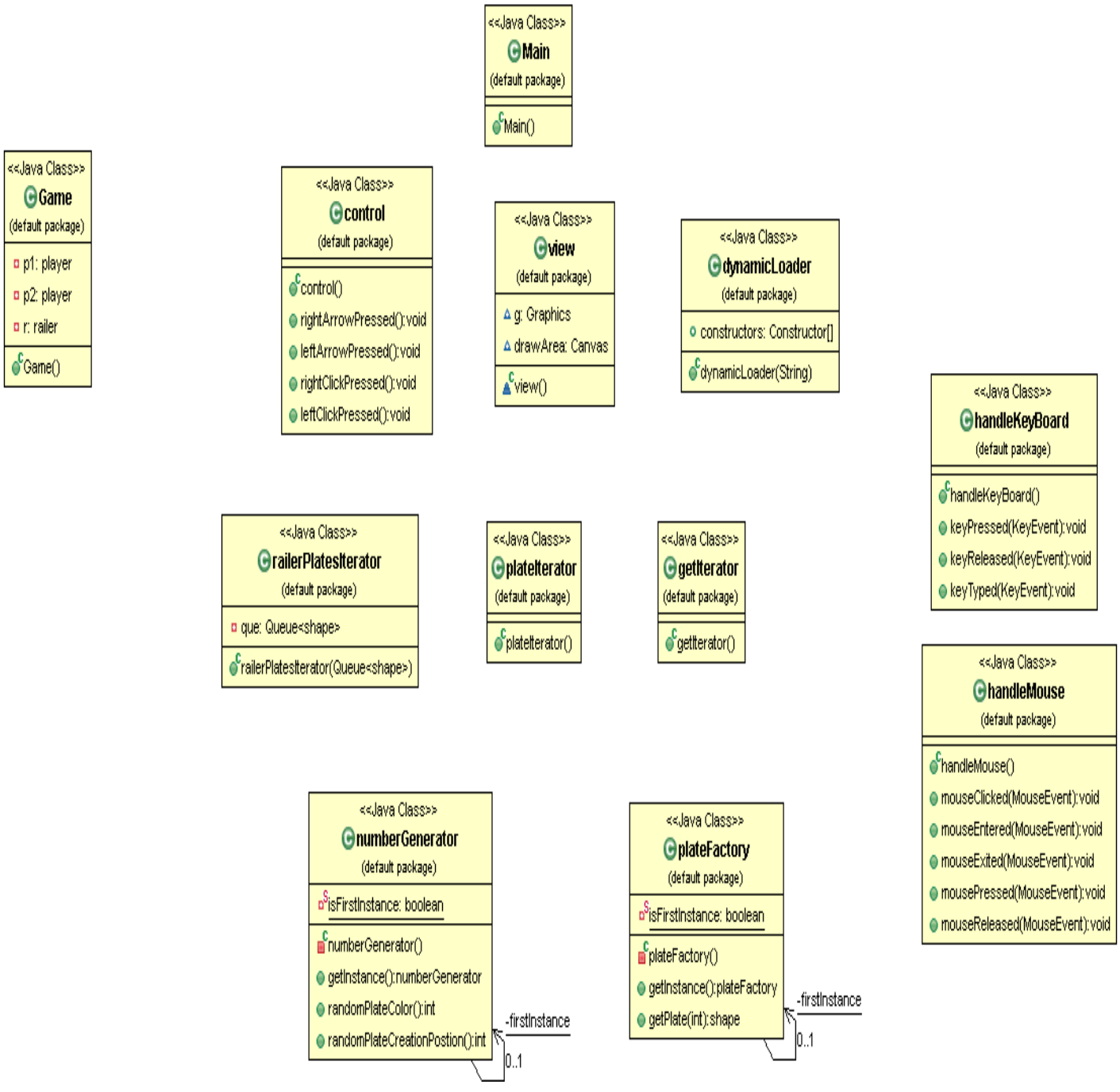
Observer design pattern :

Well , this design pattern is not supported yet in our project . it the final report we will introduce our idea of implementation of this design pattern , however it is not decided till now how to cope with the assignment tasks regarding this part .

The snapshot design pattern :

This design pattern is for ***the saving*** the game at a ***certain moment*** using java ***serializable*** without the need of ***defining*** how to save the attributes of any thing .


<<Java Interface>>
iterator
(default package)
createIterator():Iterator
<<Java Class>>
playerPlatesIterator
(default package)
playerPlatesIterator(Stack<shape>)
createIterator():Iterator<shape>
<<Java Class>>
railer
(default package)
railer()
-r
0..1
<<Java Class>>
Game
(default package)
Game()
<<Java Class>>
plateFree
(default package)
plateFree(shape)
move():void
setCurrentColor():void
<<Java Class>>
plateOnRailer
(default package)
plateOnRailer(shape)
move():void
setCurrentColor():void
<<Java Class>>
plateInPool
(default package)
plateInPool(shape)
move():void
setCurrentColor():void
<<Java Class>>
plateWithplayer
(default package)
plateWithplayer(shape)
move():void
setCurrentColor():void
-p1
-p0
0..1
<<Java Class>>
player
(default package)
imageUrl: String
playerImage: Image
xPostion: int
yPostion: int
player()
move():void
getPlatesOfstk1():Stack<shape>
getPlatesOfstk2():Stack<shape>
<<Java Interface>>
plateState
(default package)
move():void
setCurrentColor():void
-plateIsWithPlayer
-currentState
0..1
<<Java Class>>
railerPlatesIterator
(default package)
railerPlatesIterator(Queue<shape>)
<<Java Class>>
plateOne
(default package)
plateOne()
setState(plateState):void
getCurrentState():plateState
getPoolState():plateState
getFreeState():plateState
getPlayerState():plateState
getRailerState():plateState
-que
0..*
-stk2
-stk1
<<Java Class>>
shape
(default package)
xPostion: int
yPostion: int
plateImage: Image
shape()
<<Java Class>>
plateTwo
(default package)
plateTwo()
setState(plateState):void
getCurrentState():plateState
getPoolState():plateState
getFreeState():plateState
getPlayerState():plateState
getRailerState():plateState


<<Java Class>>
Main
(default package)
Main()
<<Java Class>>
Game
(default package)
p1: player
p2: player
r: railer
Game()
<<Java Class>>
control
(default package)
control()
rightArrowPressed():void
leftArrowPressed():void
rightClickPressed():void
leftClickPressed():void
<<Java Class>>
view
(default package)
g: Graphics
drawArea: Canvas
view()
<<Java Class>>
dynamicLoader
(default package)
constructors: Constructor[]
dynamicLoader(String)
<<Java Class>>
handleKeyBoard
(default package)
handleKeyBoard()
keyPressed(KeyEvent):void
keyReleased(KeyEvent):void
keyTyped(KeyEvent):void
<<Java Class>>
railerPlatesIterator
(default package)
que: Queue<shape>
railerPlatesIterator(Queue<shape>)
<<Java Class>>
plateIterator
(default package)
plateIterator()
<<Java Class>>
getIterator
(default package)
getIterator()
<<Java Class>>
handleMouse
(default package)
handleMouse()
mouseClicked(MouseEvent):void
mouseEntered(MouseEvent):void
mouseExited(MouseEvent):void
mousePressed(MouseEvent):void
mouseReleased(MouseEvent):void
<<Java Class>>
numberGenerator
(default package)
isFirstInstance: boolean
numberGenerator()
getInstance():numberGenerator
randomPlateColor():int
randomPlateCreationPostion():int
-firstInstance
0..1
<<Java Class>>
plateFactory
(default package)
isFirstInstance: boolean
plateFactory()
getInstance():plateFactory
getPlate(int):shape
-firstInstance
0..1

User Guide

1. First run the code from the Game Class.
2. Player one should use the arrows of the keyboard to move his/her Clown.
3. Player two should use the right and left mouse buttons to move his/her Clown.
4. A continues flow of plates will fall randomly in the game plane.
5. If a player is in suitable range from the plate when it lands then the plate will be held by the Clown in the appropriate hand.
6. If a player gathers Three consecutive Shapes of the same color (regardless of its type) these three shapes will be popped and returned to the pool.
7. Press “S” on the keyboard to save and Press “L”on the keyboard.
8. The Game ends if any player reaches the limit of number of accumulative shapes with deletion
9. A player wins when the game ends and he/she has maximum score.

Design Decisions

- Run the application from class Game.
- Closing from the close window will cause an error.
- We load the classes PlateOne and PlateTwo once in the plate factory.
- We did not use a JFileChooser to make it game-like.

Design Description

- We put our model in Class Game
- Our View and Drawing occurs in class paint
- All the Control is in classes Control and PlateControl.
- We Dynamically load Classes Plate and Cup in the start of the game.
- The Model is accessed by the view and the Controllers
- The Controllers can access all classes
- The Model Only contacts the Controllers
- We Strongly designed the STATE DESIGN PATTERN , we analyzed all possible states in which the Shapes(Cups/Plates) could exist in and made transitions between them efficiently and Intelligently
- Class PlatePool makes LAZY INSTATNTIATION to the Shapes when needed
- Class ControlPlate Updates all transitions of all the Shapes and Observes the rest of the plates and the plates notifies the class with the updates…. In this class exists an array of shapes inService which contains all shapes in motion .
- Once Three shapes of the same color are put on one stack they disappear to the pool which reuses them again and generates them on the railler
- The NumberGenerator Class is the CONTROL CENTER OF RANDOMIZATION needed in any class in our humble project. It generates the endpoint trajectory of the shapes on leaving the railler and randomizes the types and colors of shapes and which railler the next shape out of the pool will move on.

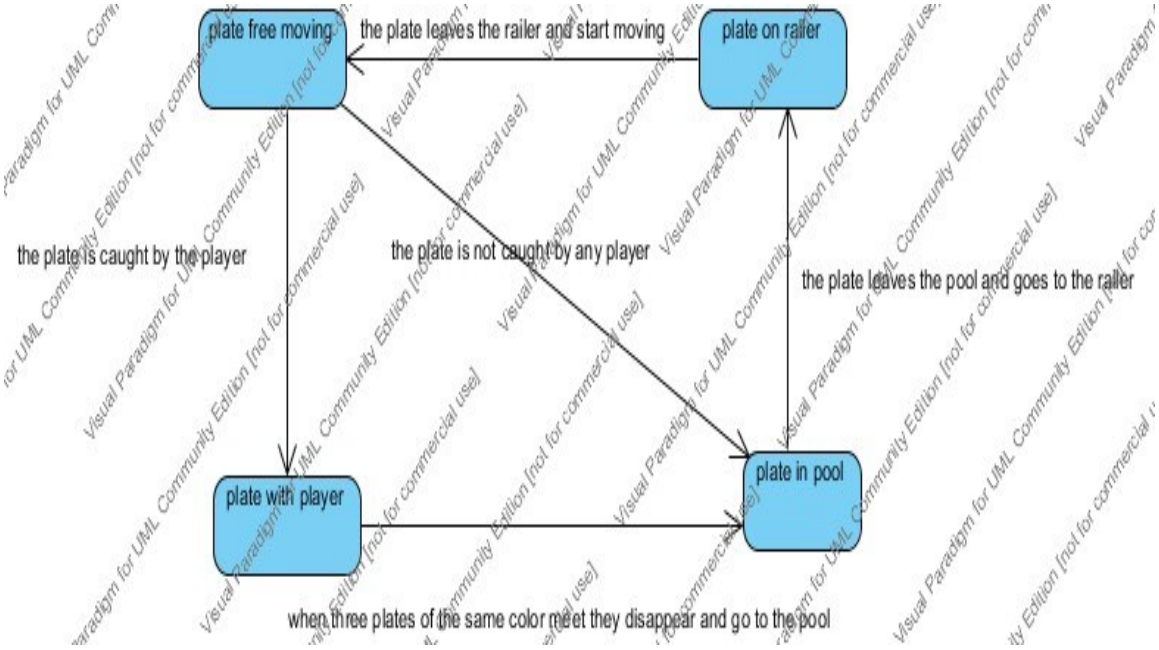
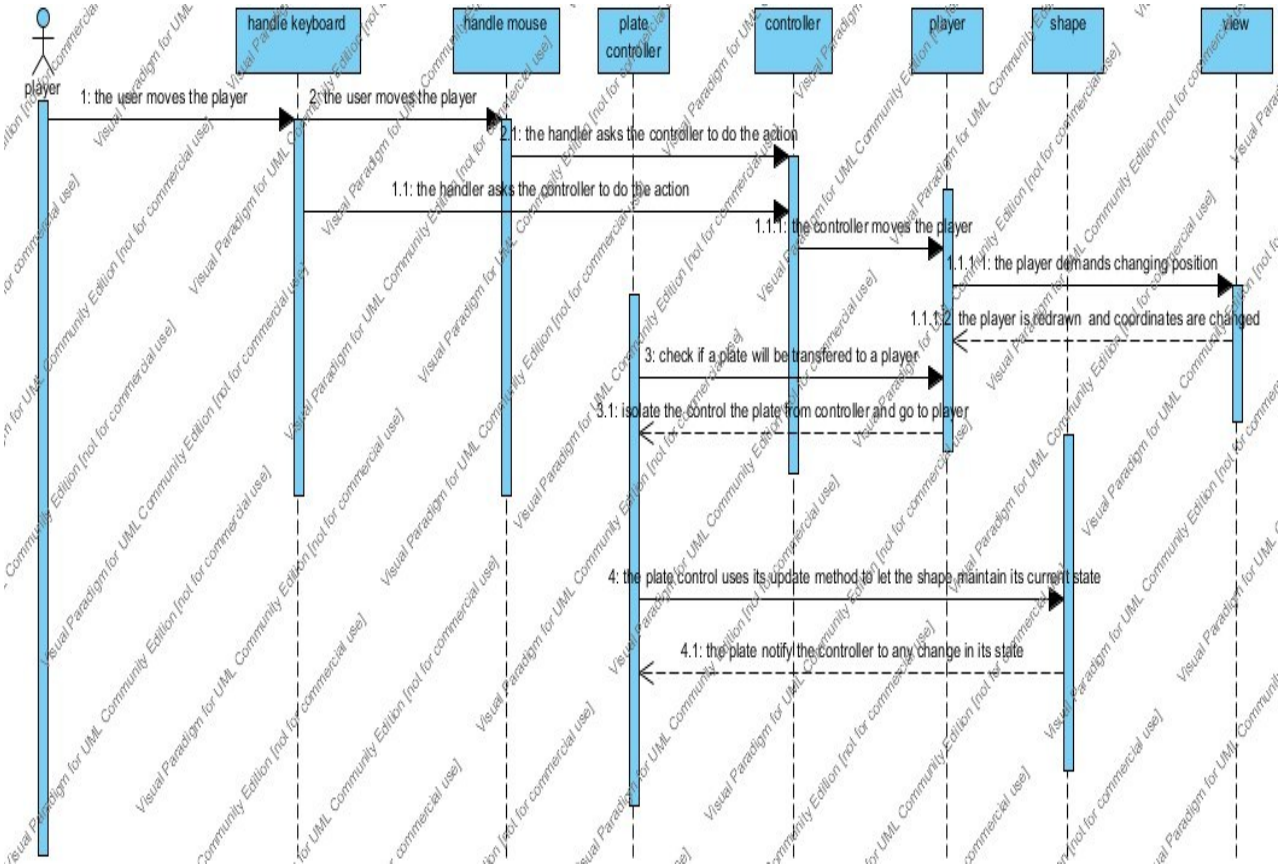


plate free moving
the plate leaves the railer and start moving
plate on railer
the plate is caught by the player
the plate is not caught by any player
the plate leaves the pool and goes to the railer
plate with player
plate in pool
when three plates of the same color meet they disappear and go to the pool


player
handle keyboard
handle mouse
plate controller
controller
player
shape
view
1: the user moves the player
2: the user moves the player
2.1: the handler asks the controller to do the action
1.1: the handler asks the controller to do the action
1.1.1: the controller moves the player
1.1.1.1: the player demands changing position
1.1.1.2: the player is redrawn and coordinates are changed
3: check if a plate will be transfered to a player
3.1: isolate the control the plate from controller and go to player
4: the plate control uses its update method to let the shape maintain its current state
4.1: the plate notify the controller to any change in its state

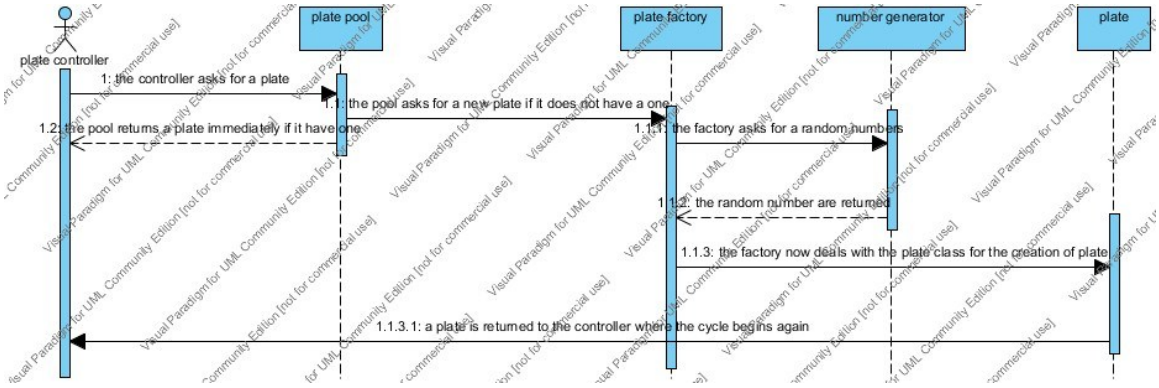
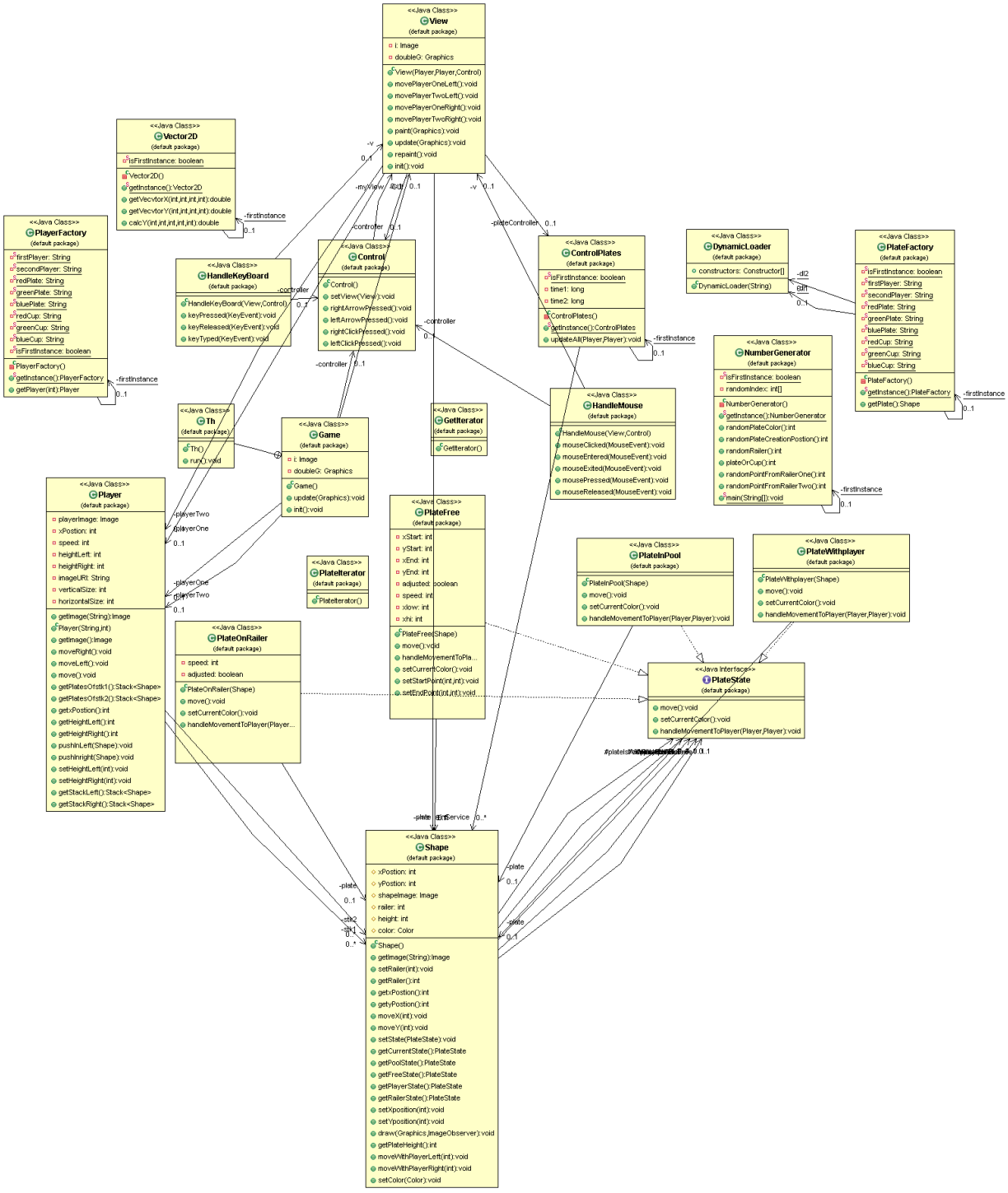


plate controller
plate pool
plate factory
number generator
plate
1: the controller asks for a plate
1.1: the pool asks for a new plate if it does not have a one
1.2: the pool returns a plate immediately if it have one
1.1.1: the factory asks for a random numbers
1.1.2: the random number are returned
1.1.3: the factory now deals with the plate class for the creation of plate
1.1.3.1: a plate is returned to the controller where the cycle begins again


View
i: Image
doubleG: Graphics
View(Player,Player,Control)
movePlayerOneLeft():void
movePlayerTwoLeft():void
movePlayerOneRight():void
movePlayerTwoRight():void
paint(Graphics):void
update(Graphics):void
repaint():void
init():void
Vector2D
isFirstInstance: boolean
Vector2D()
getInstance():Vector2D
getVecvtorX(int,int,int,int):double
getVecvtorY(int,int,int,int):double
calcY(int,int,int,int,int):double
PlayerFactory
firstPlayer: String
secondPlayer: String
redPlate: String
greenPlate: String
bluePlate: String
redCup: String
greenCup: String
blueCup: String
isFirstInstance: boolean
PlayerFactory()
getInstance():PlayerFactory
getPlayer(int):Player
HandleKeyBoard
HandleKeyBoard(View,Control)
keyPressed(KeyEvent):void
keyReleased(KeyEvent):void
keyTyped(KeyEvent):void
Control
Control()
setView(View):void
rightArrowPressed():void
leftArrowPressed():void
rightClickPressed():void
leftClickPressed():void
ControlPlates
isFirstInstance: boolean
time1: long
time2: long
ControlPlates()
getInstance():ControlPlates
updateAll(Player,Player):void
DynamicLoader
constructors: Constructor[]
DynamicLoader(String)
PlateFactory
isFirstInstance: boolean
firstPlayer: String
secondPlayer: String
redPlate: String
greenPlate: String
bluePlate: String
redCup: String
greenCup: String
blueCup: String
PlateFactory()
getInstance():PlateFactory
getPlate():Shape
NumberGenerator
isFirstInstance: boolean
randomIndex: int[]
NumberGenerator()
getInstance():NumberGenerator
randomPlateColor():int
randomPlateCreationPostion():int
randomRailer():int
plateOrCup():int
randomPointFromRailerOne():int
randomPointFromRailerTwo():int
main(String[]):void
Th
Th()
run():void
Game
i: Image
doubleG: Graphics
Game()
update(Graphics):void
init():void
GetIterator
GetIterator()
HandleMouse
HandleMouse(View,Control)
mouseClicked(MouseEvent):void
mouseEntered(MouseEvent):void
mouseExited(MouseEvent):void
mousePressed(MouseEvent):void
mouseReleased(MouseEvent):void
Player
playerImage: Image
xPostion: int
speed: int
heightLeft: int
heightRight: int
imageURl: String
verticalSize: int
horizontalSize: int
getImage(String):Image
Player(String,int)
getImage():Image
moveRight():void
moveLeft():void
move():void
getPlatesOfstlk1():Stack<Shape>
getPlatesOfstlk2():Stack<Shape>
getxPostion():int
getHeightLeft():int
getHeightRight():int
pushInLeft(Shape):void
pushInright(Shape):void
setHeightLeft(int):void
setHeightRight(int):void
getStackLeft():Stack<Shape>
getStackRight():Stack<Shape>
PlateIterator
PlateIterator()
PlateFree
xStart: int
yStart: int
xEnd: int
yEnd: int
adjusted: boolean
speed: int
xdow: int
xhi: int
PlateFree(Shape)
move():void
handleMovementToPla...
setCurrentColor():void
setStartPoint(int,int):void
setEndPoint(int,int):void
PlateInPool
PlateInPool(Shape)
move():void
setCurrentColor():void
handleMovementToPlayer(Player,Player):void
PlateWithplayer
PlateWithplayer(Shape)
move():void
setCurrentColor():void
handleMovementToPlayer(Player,Player):void
PlateOnRailer
speed: int
adjusted: boolean
PlateOnRailer(Shape)
move():void
setCurrentColor():void
handleMovementToPlayer(Player...
PlateState
move():void
setCurrentColor():void
handleMovementToPlayer(Player,Player):void
Shape
xPostion: int
yPostion: int
shapeImage: Image
railer: int
height: int
color: Color
Shape()
getImage(String):Image
setRailer(int):void
getRailer():int
getxPostion():int
getyPostion():int
moveX(int):void
moveY(int):void
setState(PlateState):void
getCurrentState():PlateState
getPoolState():PlateState
getFreeState():PlateState
getPlayerState():PlateState
getRailerState():PlateState
setXposition(int):void
setYposition(int):void
draw(Graphics,ImageObserver):void
getPlateHeight():int
moveWithPlayerLeft(int):void
moveWithPlayerRight(int):void
setColor(Color):void