Circus of plates
Lab 5

Names:	IDs:
Enas Morsy	20
Hazem Ahmed	23
Kareem Ahmed	48
Sara Muhammad	31

UML:

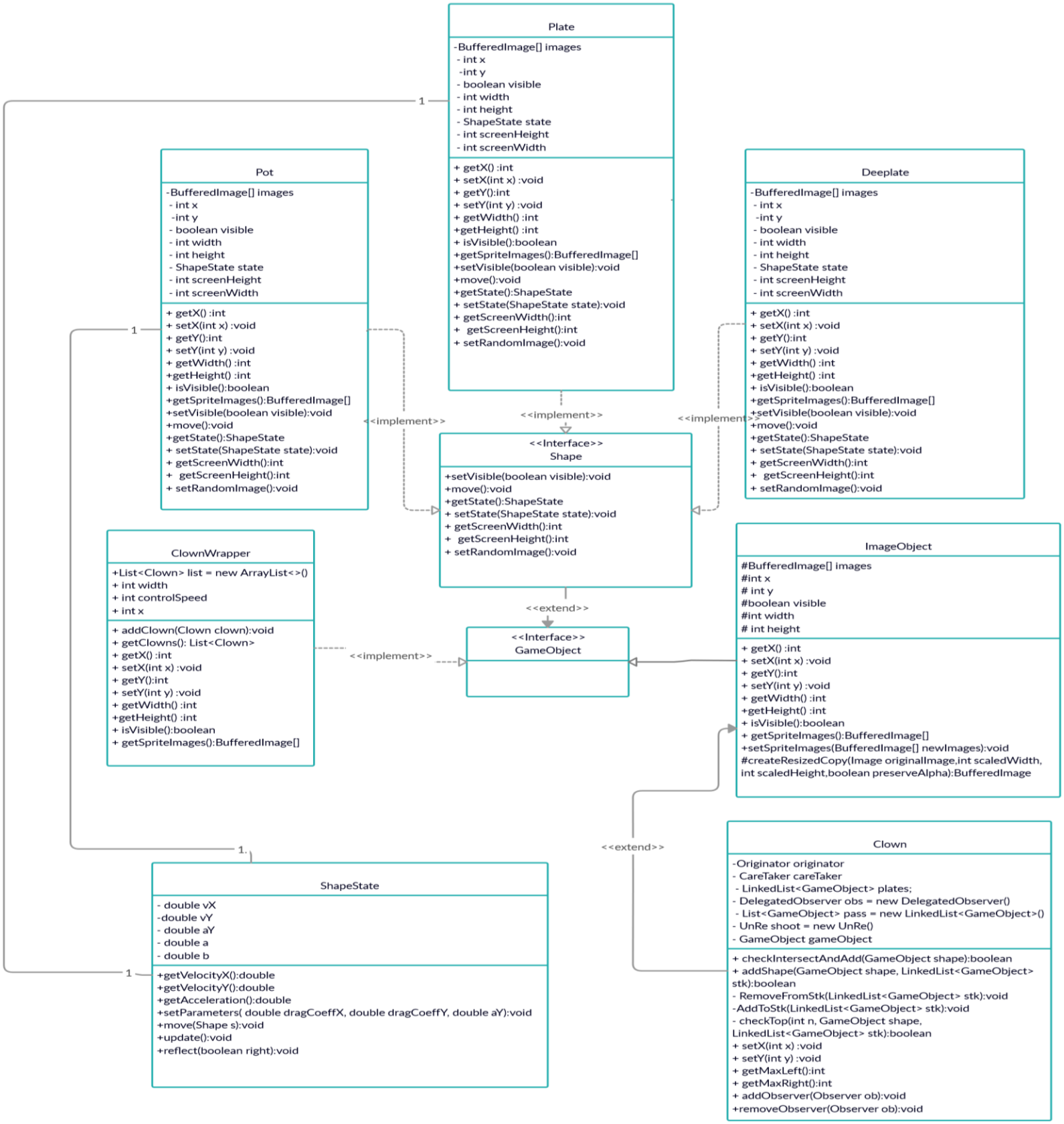
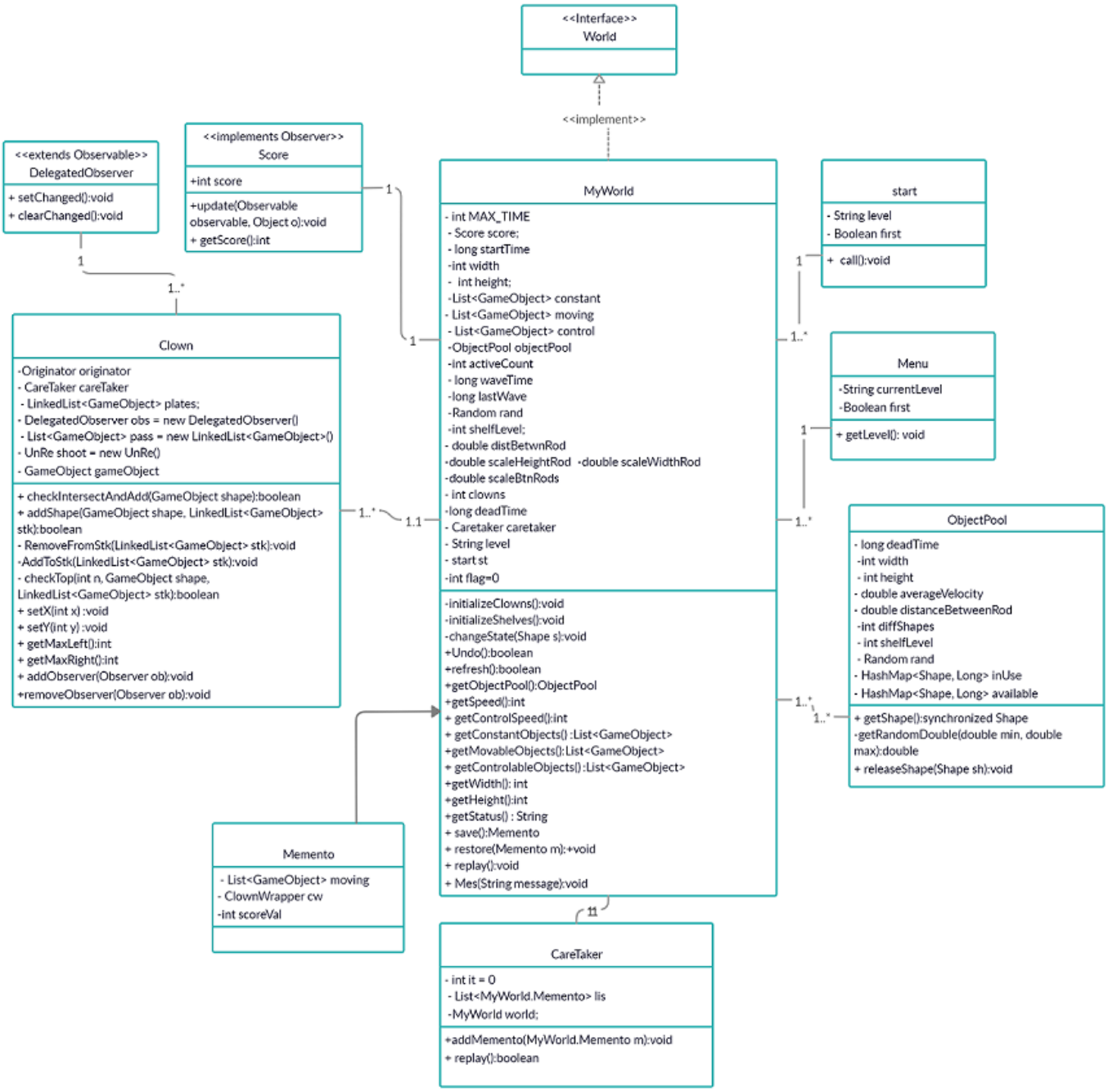
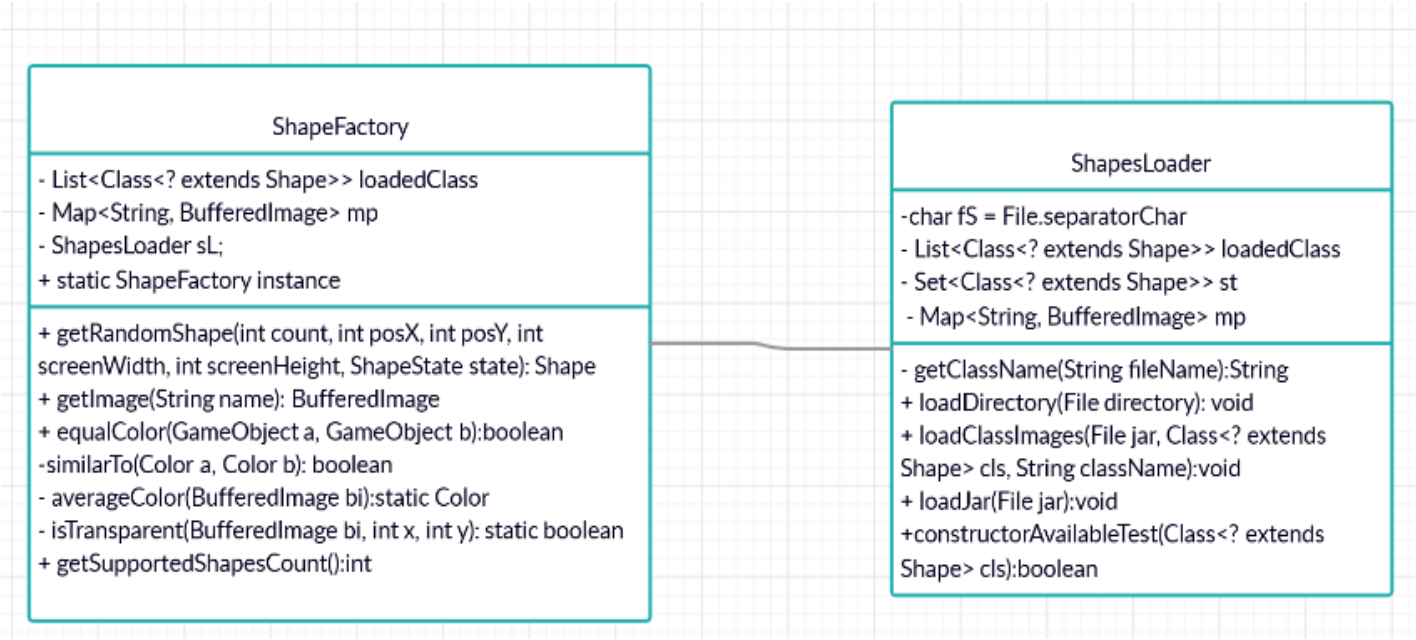


Plate
-BufferedImage[] images
- int x
-int y
- boolean visible
- int width
- int height
- ShapeState state
- int screenHeight
- int screenWidth
+ getX() :int
+ setX(int x) :void
+ getY():int
+ setY(int y) :void
+ getWidth() :int
+getHeight() :int
+ isVisible():boolean
+getSpriteImages():BufferedImage[]
+setVisible(boolean visible):void
+move():void
+getState():ShapeState
+ setState(ShapeState state):void
+ getScreenWidth():int
+ getScreenHeight():int
+ setRandomImage():void
Pot
-BufferedImage[] images
- int x
-int y
- boolean visible
- int width
- int height
- ShapeState state
- int screenHeight
- int screenWidth
+ getX() :int
+ setX(int x) :void
+ getY():int
+ setY(int y) :void
+ getWidth() :int
+getHeight() :int
+ isVisible():boolean
+getSpriteImages():BufferedImage[]
+setVisible(boolean visible):void
+move():void
+getState():ShapeState
+ setState(ShapeState state):void
+ getScreenWidth():int
+ getScreenHeight():int
+ setRandomImage():void
Deeplate
-BufferedImage[] images
- int x
-int y
- boolean visible
- int width
- int height
- ShapeState state
- int screenHeight
- int screenWidth
+ getX() :int
+ setX(int x) :void
+ getY():int
+ setY(int y) :void
+ getWidth() :int
+getHeight() :int
+ isVisible():boolean
+getSpriteImages():BufferedImage[]
+setVisible(boolean visible):void
+move():void
+getState():ShapeState
+ setState(ShapeState state):void
+ getScreenWidth():int
+ getScreenHeight():int
+ setRandomImage():void
<<Interface>>
Shape
+setVisible(boolean visible):void
+move():void
+getState():ShapeState
+ setState(ShapeState state):void
+ getScreenWidth():int
+ getScreenHeight():int
+ setRandomImage():void
ClownWrapper
+List<Clown> list = new ArrayList<>()
+ int width
+ int controlSpeed
+ int x
+ addClown(Clown clown):void
+ getClowns(): List<Clown>
+ getX() :int
+ setX(int x) :void
+ getY():int
+ setY(int y) :void
+ getWidth() :int
+getHeight() :int
+ isVisible():boolean
+ getSpriteImages():BufferedImage[]
<<Interface>>
GameObject
ImageObject
#BufferedImage[] images
#int x
int y
#boolean visible
#int width
int height
+ getX() :int
+ setX(int x) :void
+ getY():int
+ setY(int y) :void
+ getWidth() :int
+getHeight() :int
+ isVisible():boolean
+ getSpriteImages():BufferedImage[]
+setSpriteImages(BufferedImage[] newImages):void
#createResizedCopy(Image originalImage,int scaledWidth, int scaledHeight,boolean preserveAlpha):BufferedImage
ShapeState
- double vX
-double vY
- double aY
- double a
- double b
+getVelocityX():double
+getVelocityY():double
+getAcceleration():double
+setParameters(double dragCoeffX, double dragCoeffY, double aY):void
+move(Shape s):void
+update():void
+reflect(boolean right):void
Clown
-Originator originator
- CareTaker careTaker
- LinkedList<GameObject> plates;
- DelegatedObserver obs = new DelegatedObserver()
- List<GameObject> pass = new LinkedList<GameObject>()
- UnRe shoot = new UnRe()
- GameObject gameObject
+ checkIntersectAndAdd(GameObject shape):boolean
+ addShape(GameObject shape, LinkedList<GameObject> stk):boolean
- RemoveFromStk(LinkedList<GameObject> stk):void
-AddToStk(LinkedList<GameObject> stk):void
- checkTop(int n, GameObject shape, LinkedList<GameObject> stk):boolean
+ setX(int x) :void
+ setY(int y) :void
+ getMaxLeft():int
+ getMaxRight():int
+ addObserver(Observer ob):void
+removeObserver(Observer ob):void
<<implement>>
<<implement>>
<<implement>>
<<extend>>
<<implement>>
<<extend>>
1
1
1
1
1.


<<Interface>>
World
<<implement>>
<<extends Observable>>
DelegatedObserver
+ setChanged():void
+ clearChanged():void
<<implements Observer>>
Score
+int score
+update(Observable observable, Object o):void
+ getScore():int
MyWorld
- int MAX_TIME
- Score score;
- long startTime
-int width
- int height;
-List<GameObject> constant
- List<GameObject> moving
- List<GameObject> control
-ObjectPool objectPool
-int activeCount
- long waveTime
-long lastWave
-Random rand
-int shelfLevel;
- double distBetwnRod
-double scaleHeightRod -double scaleWidthRod
-double scaleBtnRods
- int clowns
-long deadTime
- Caretaker caretaker
- String level
- start st
-int flag=0
-initializeClowns():void
-initializeShelves():void
-changeState(Shape s):void
+Undo():boolean
+refresh():boolean
+getObjectPool():ObjectPool
+getSpeed():int
+ getControlSpeed():int
+ getConstantObjects() :List<GameObject>
+getMovableObjects():List<GameObject>
+ getControlableObjects() :List<GameObject>
+getWidth(): int
+getHeight():int
+getStatus() : String
+ save():Memento
+ restore(Memento m):+void
+ replay():void
+ Mes(String message):void
start
- String level
- Boolean first
+ call():void
Menu
-String currentLevel
-Boolean first
+ getLevel(): void
ObjectPool
- long deadTime
-int width
- int height
- double averageVelocity
- double distanceBetweenRod
-int diffShapes
- int shelfLevel
- Random rand
- HashMap<Shape, Long> inUse
- HashMap<Shape, Long> available
+ getShape():synchronized Shape
-getRandomDouble(double min, double max):double
+ releaseShape(Shape sh):void
Clown
-Originator originator
- CareTaker careTaker
- LinkedList<GameObject> plates;
- DelegatedObserver obs = new DelegatedObserver()
- List<GameObject> pass = new LinkedList<GameObject>()
- UnRe shoot = new UnRe()
- GameObject gameObject
+ checkIntersectAndAdd(GameObject shape):boolean
+ addShape(GameObject shape, LinkedList<GameObject> stk):boolean
- RemoveFromStk(LinkedList<GameObject> stk):void
-AddToStk(LinkedList<GameObject> stk):void
- checkTop(int n, GameObject shape, LinkedList<GameObject> stk):boolean
+ setX(int x) :void
+ setY(int y) :void
+ getMaxLeft():int
+ getMaxRight():int
+ addObserver(Observer ob):void
+removeObserver(Observer ob):void
Memento
- List<GameObject> moving
- ClownWrapper cw
-int scoreVal
CareTaker
- int it = 0
- List<MyWorld.Memento> lis
-MyWorld world;
+addMemento(MyWorld.Memento m):void
+ replay():boolean
1
1..*
1..*
1.1
11



- The game loads jar files and extracts plates images through “shape loader” class which enables the user to add his jars to have his own shapes only if it implements the shape interface.
- shapes Factory chooses random shapes and prepares them to be used during the game.
- A pool was created to reduce the number of created shapes by having two HashMaps; “isUse” which holds the current elements being on the screen and “available” which holds unused created elements to take from.
- “MyWorld” class is the basic environment; depending on its constructor it initializes instances –clowns, shelves and moving plates- that determine current level.
- “Clown” class is the class where we check if there is an intersection between the fallen plate and the clown stick depending on the width of its previous plate or stick, if there

was one it adds the plate to the stick's stack otherwise the plate falls down, gets out of screen border and then be released by taking it back to "available" hashmap.

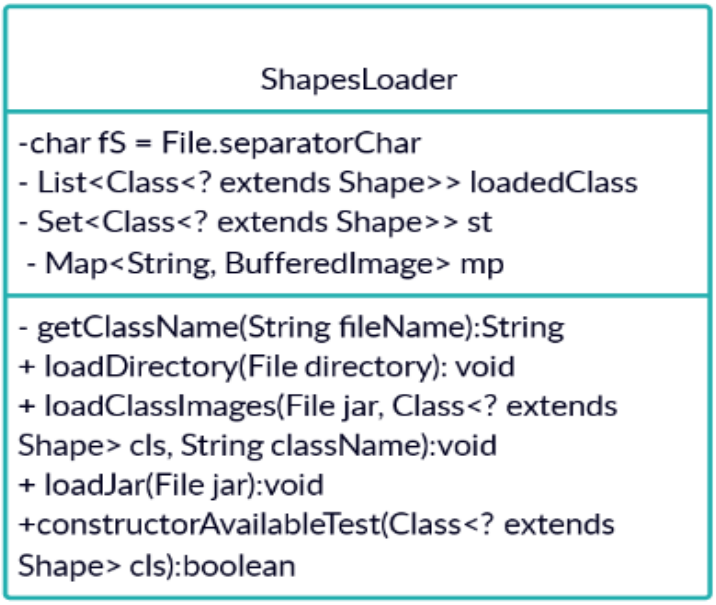
- Every time happens an intersection it checks if the last 3 plates have the same color then the observer notify that, score of player increases by one and plates are removed from stick and backed to "available" hashmap.
- Memento design pattern is used to save the states of sticks for each clown in order to reusing them in case of "Remove" button which needs to go back through them.

Design patterns:

- **Dynamic Linkage:**

  It is used to make the player able to add his own shapes during runtime by loading their jar files.

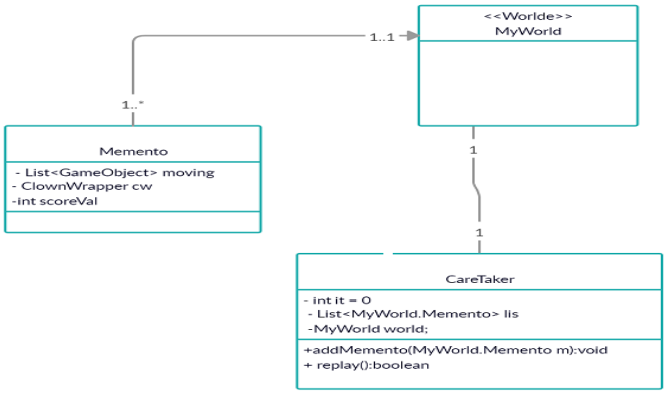
  It works through taking the path, iterating through the jar searching for classes which implement the shape interface and loads all its resources.



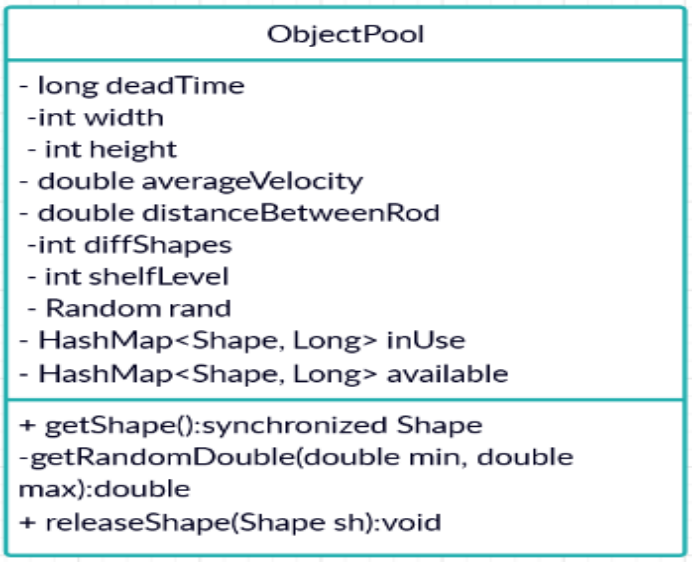
- **Snapshot**:

  It is used to save specific states of the game when a plate intersects with any clowns' stick.

  Depending on three classes: Originator, Memento and CareTacker states are Saved in a LinkedList and reused during using remove button through "Undo" function.

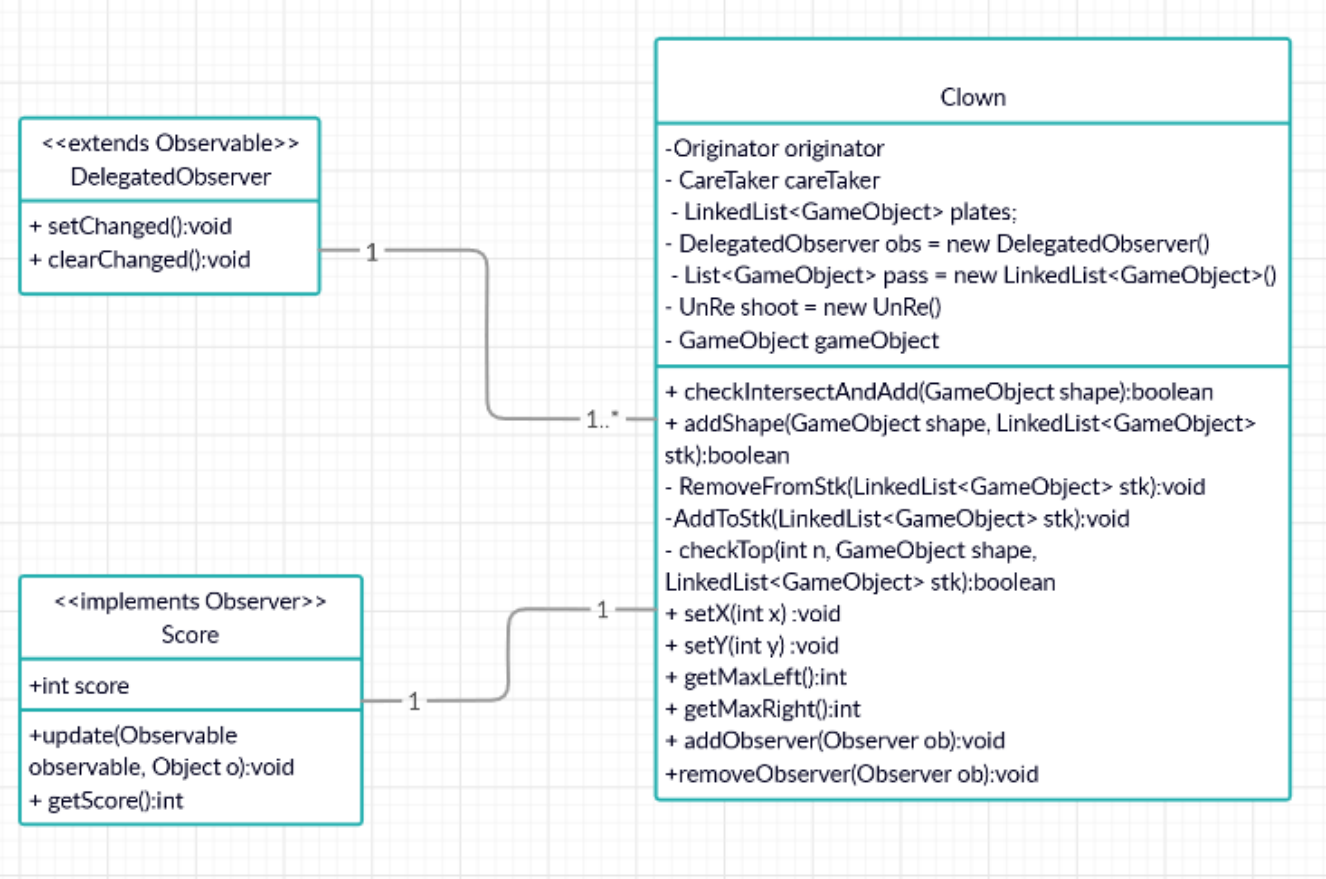


Pool: It is used to limit objects creation through reusing fallen plates when they disappear on the screen or be removed through using two HashMaps; one to hold available objects and the other holds used objects.



- **Observer:**

  It is used in clown class when it collects three plates of the same color sequentially on any stick the observer set changed and notify that change thus player's score increases by one.



- **States:**

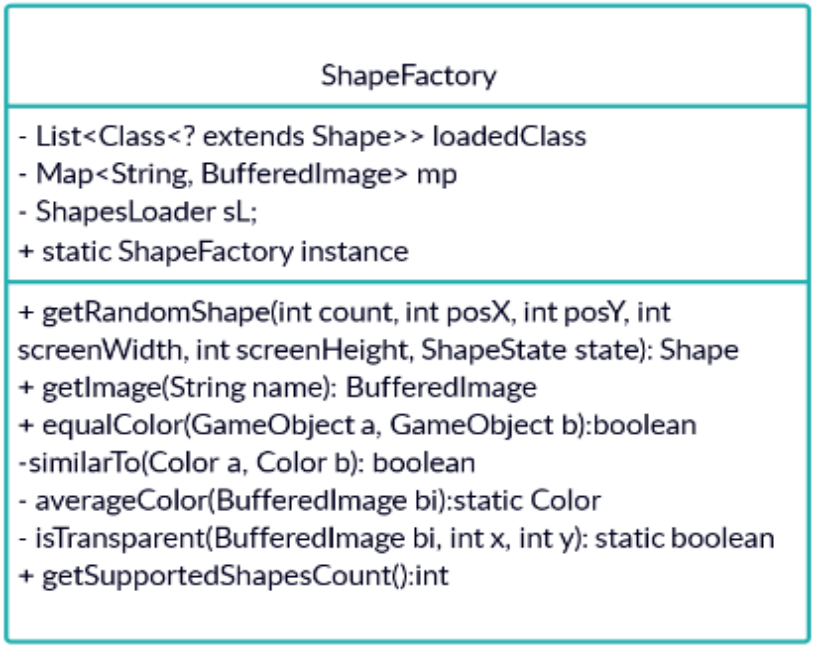
  It is used in changing the plates states during moving and falling off the shelves through changing their coordinates continuously.

- **Singleton:**

  It is used in “ShapeFactory” class which is responsible for loading shapes classes and getting instances from them.

- **Flyweight:**

  It is used in shape factory and shape loader during creating new random shapes to avoid creating the same shape two times through saving every created shape in a hash map that contains the shape name and its bufferedimage and before creating a new one it checks if it is in the hash map so it returns it and if not it creates a new one.



- **Strategy:**

  It is used with the classes of default three plates that implements shape interface where each one has its own “setRandomImage()” function depending on the plate type.

user guide:

- **One-player game:**
  Player controls clowns through right and lift arrows, his mission is to collect different plates of the same color on their sticks.
- **Consists of 3 levels:**
  Each level has one more clown than the previous one -all clowns have the same movement- and fallen plates with more speed.
- **rules:**
  1. When player collects three plates of the same color regardless their shape he gets a point and the three plates disappear.

2. To pass a level player should collect 3 points before time ends if you play the easy level, 5 points to pass the medium level and 7 points to pass the hard level.
3. Time for each level is 120 seconds.
4. When he passes a level, it automatically moves him to the next level.
5. If the time ends, the menu of levels will appear.

- **Aids:**
  1. Player has chance to replay the game.
  2. He can load any jar contains specific shape to be included in fallen plates.
- **File list:**

  Through it player can **pause, resume** or **restart** the game.