Alexandria University
Faculty of Engineering
Computer Systems & Engineering Department

جامعة الإسكندرية
كلية الهندسة
قسم هندسة الحاسب والنظم

Circus of plates

❑ Design description:

A frame and buttons to select your game level or load a previously saved one.

MenuBar to start a new game, pause, resume and save your game.

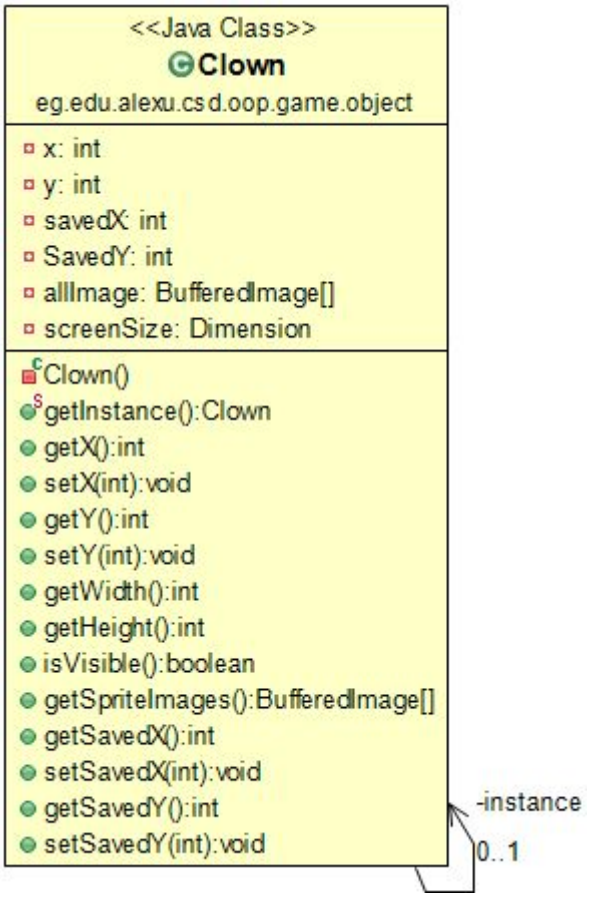
❑ Design Patterns:

- Singleton:

★*usage:*

we use it in many classes which need to have only one instance like clown as a single player game.

★*Simple diagram:*



- Factory:

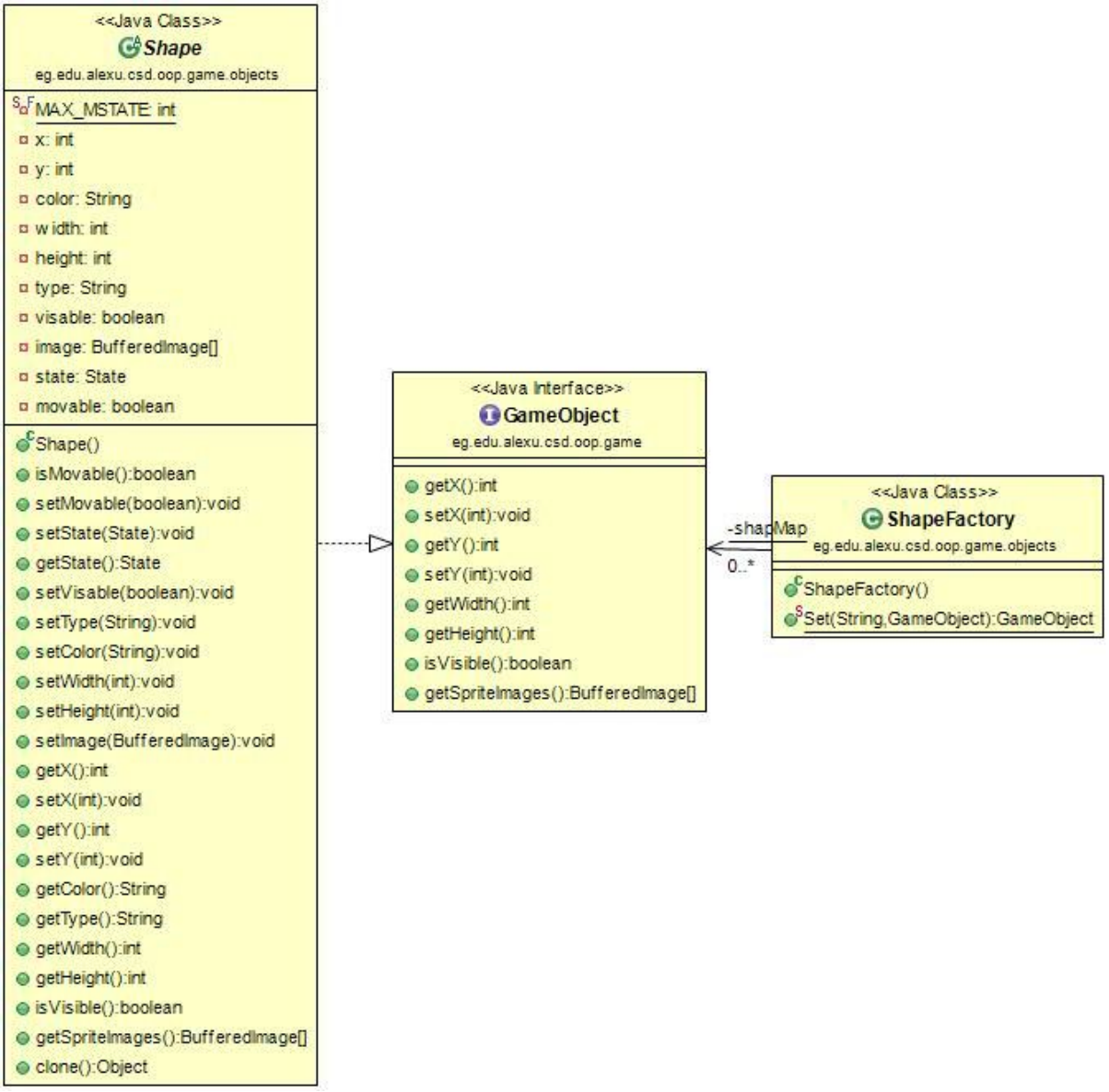
★*Usage:*

making new instances of shapes by taking parameter which is the name of the shape.

★*Components:*

ShapeFactory: A class that contains ***getShape*** function which takes an integer from 0 to 5 to determine the desired shape.

★*Simple diagram:*



- Pool pattern:

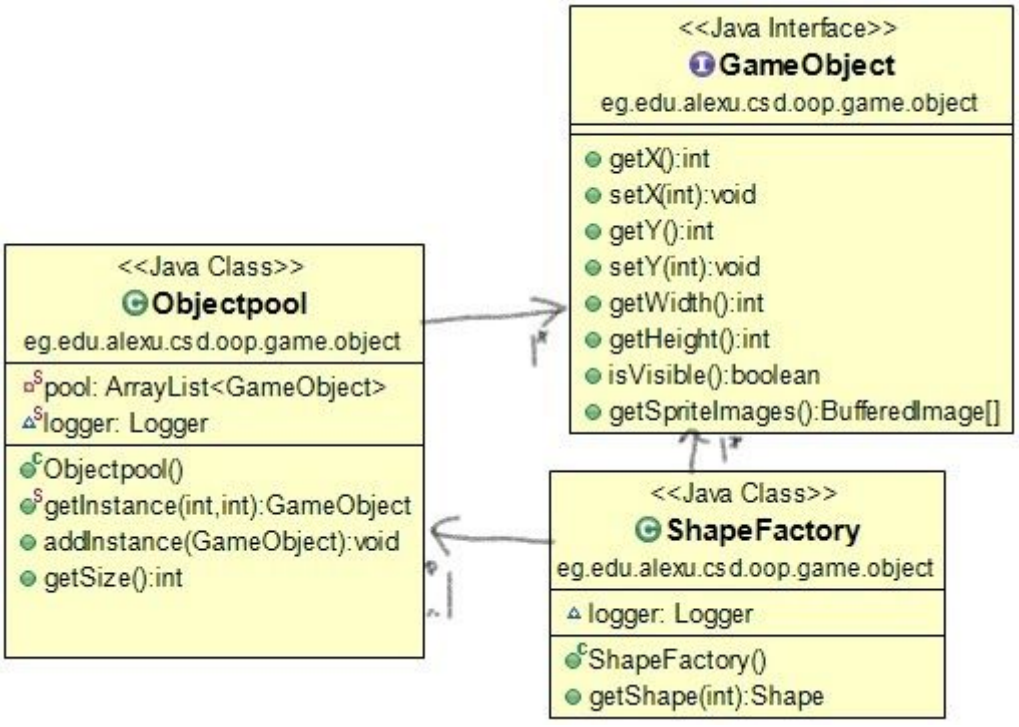
★*Usage:*

reusing fallen shape objects instead of making new instances.

★*Components:*

Objectpool: it is used to make shapes for game class.This is done by checking if there are useless objects in the pool to reuse them or not.

★*Simple diagram:*

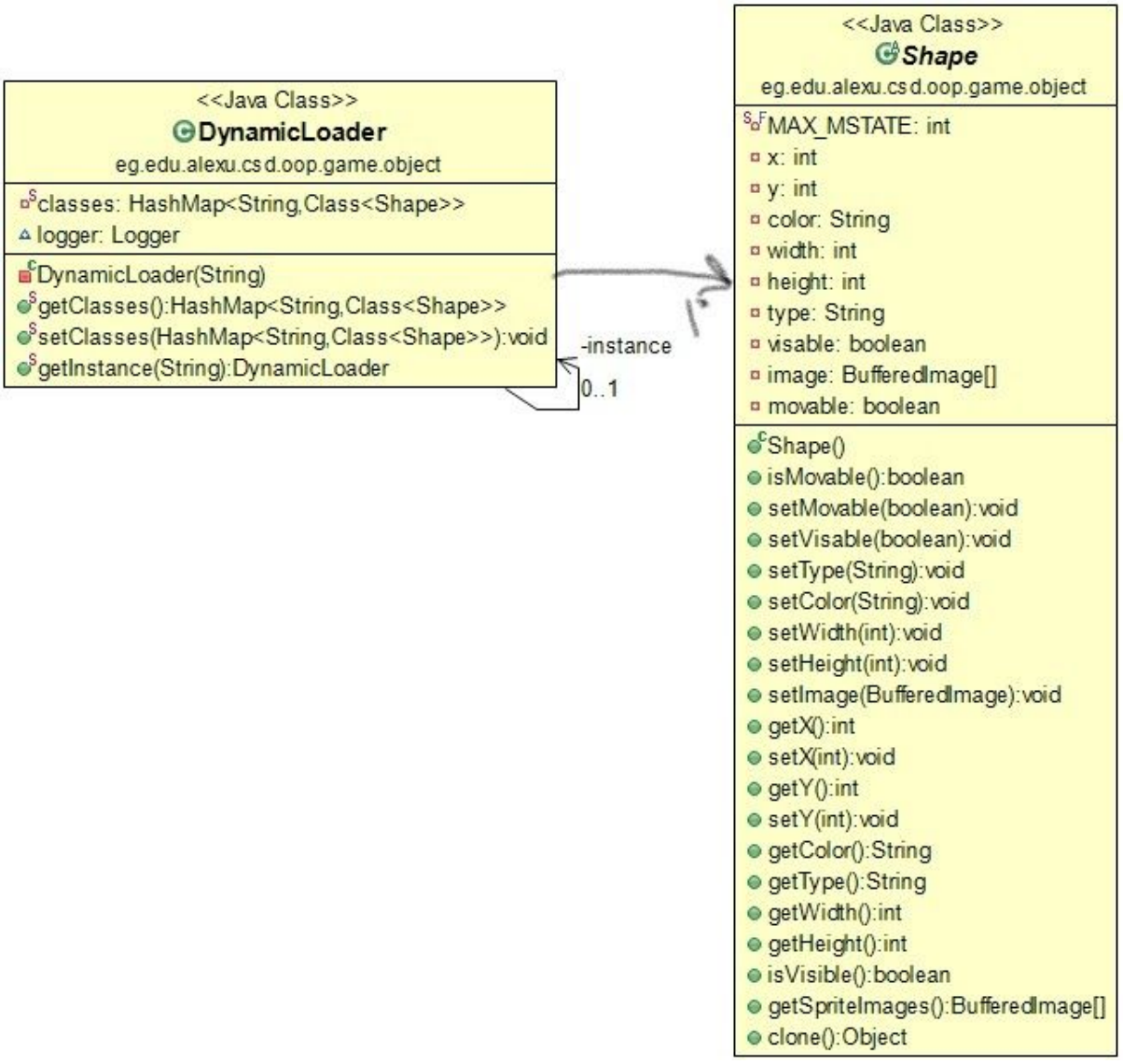


- Dynamic linkage:

★*Usage:*

it is used to load shapes from the jar.
Shape factory class is using it to make instances of shapes.

★*Simple diagram:*



- Snapshot:

★*Usage:*

it is used to save the game at a certain moment and load it whenever user wants.

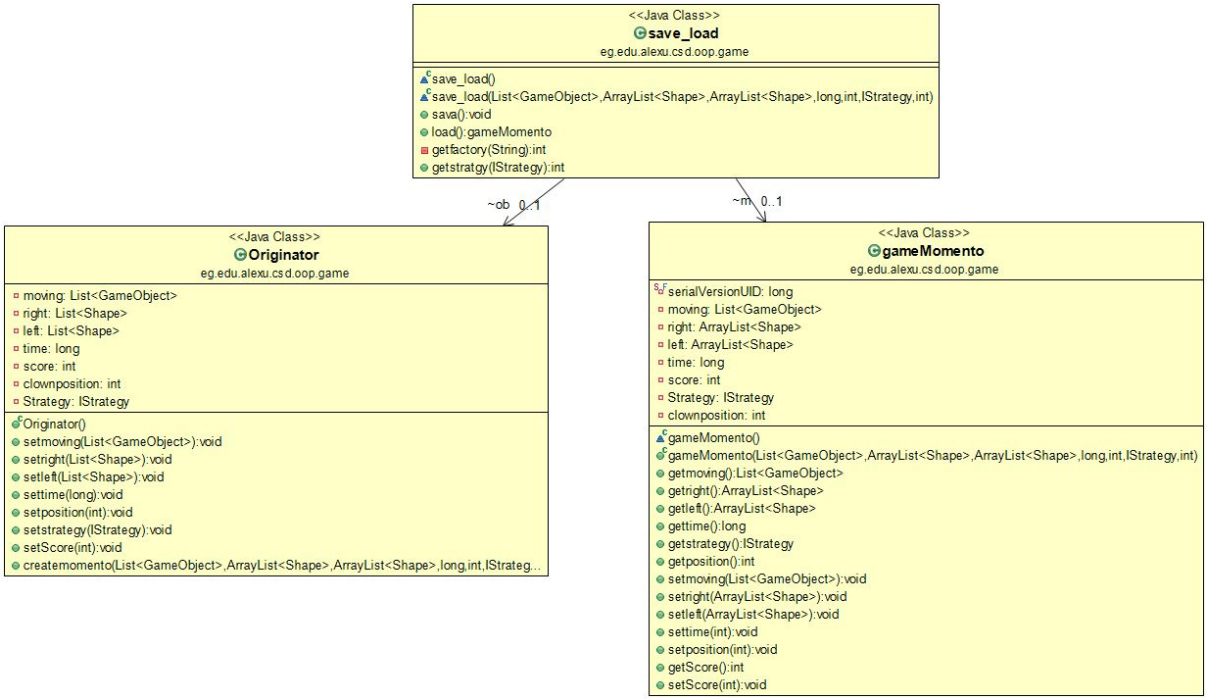
★*Components:*

gameMomento: used to save everything in the game.

originator: is the class which creates momento object,and this object has all parameters passed to originator class.

Save_load: controls the process of saving and loading game momento

★*Simple diagram:*



- State:

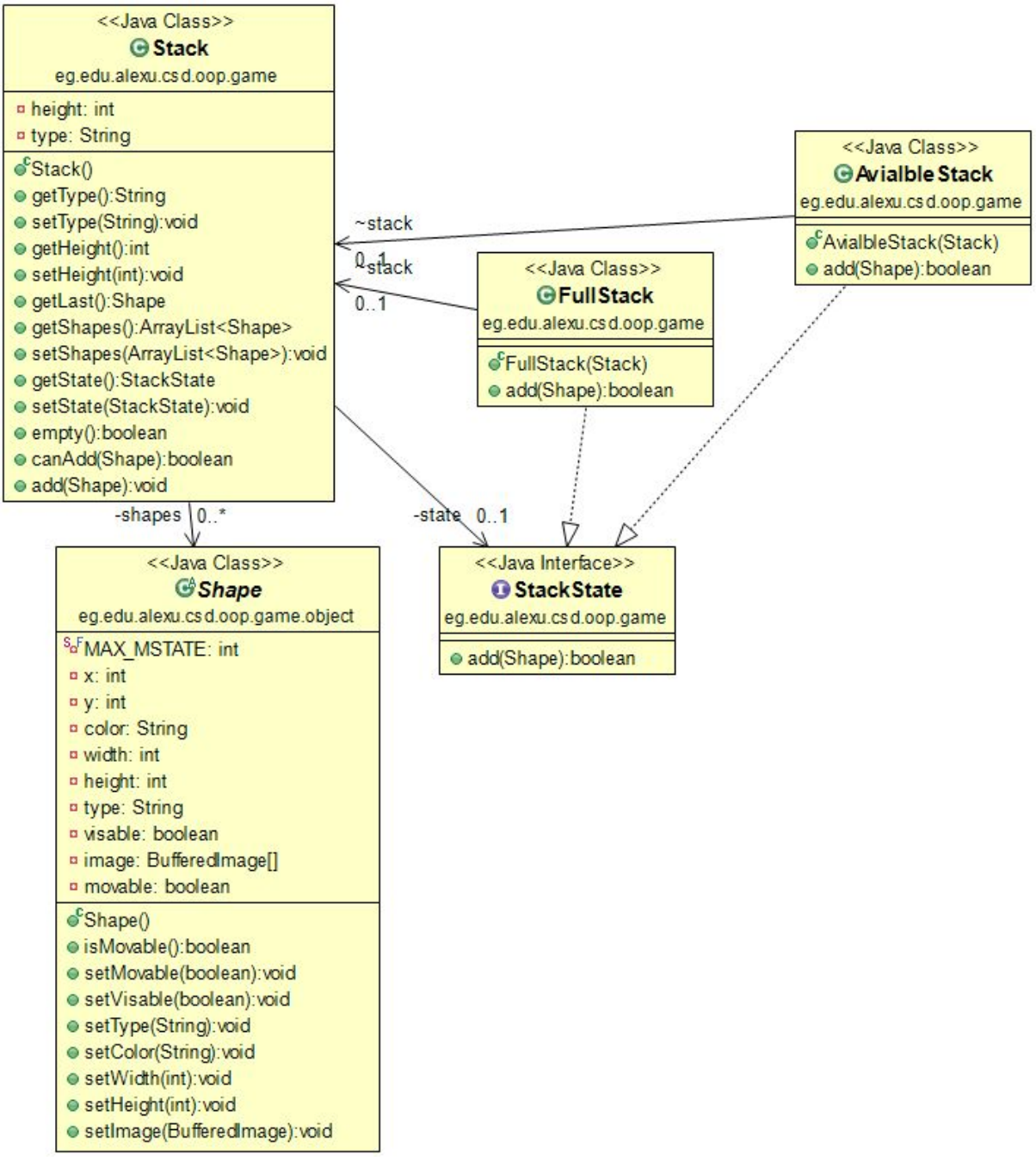
★*Usage:*

Determining whether the stack of shapes is full or not.

★*Components:*

Stack : Class that determines the state of shapes stack.It has ***canAdd*** function which returns true if the state of the stack is available and false otherwise.

Full Stack and Available stack: represent different states of stack class.

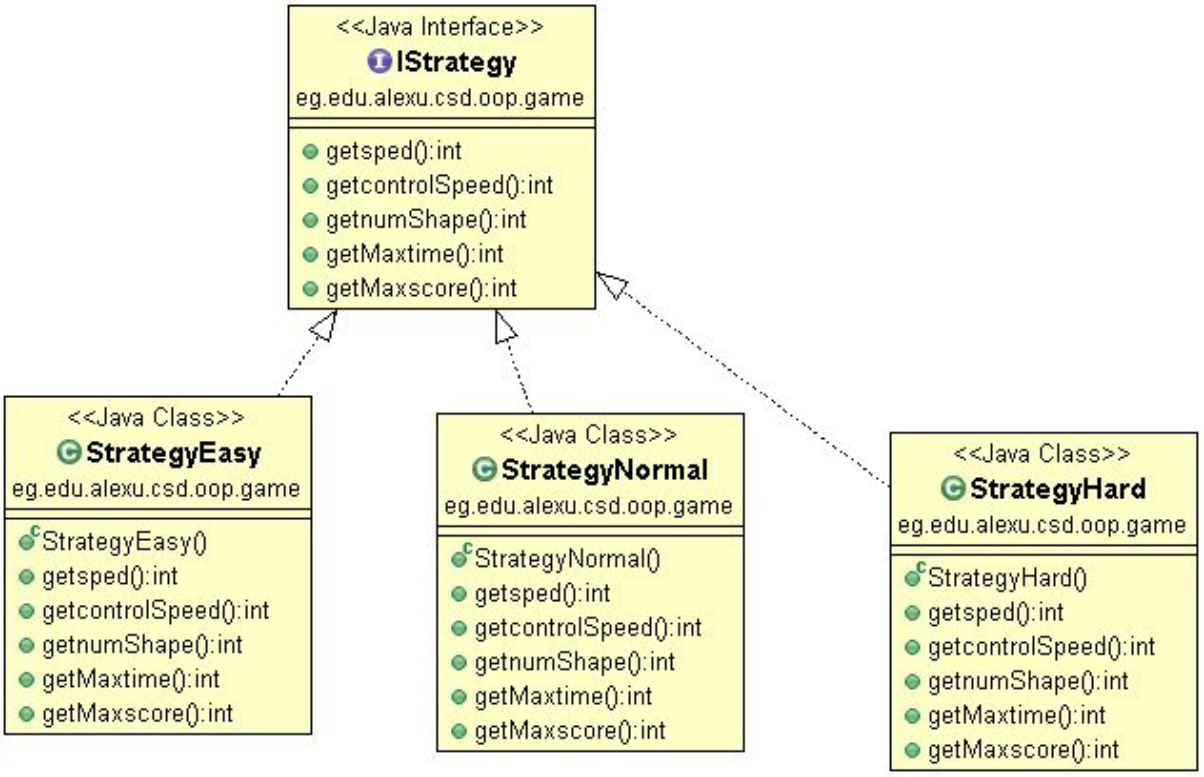


- Strategy:

★ *Usage:*

As we need to encapsulate the family of algorithms which differing the game difficulty by **speed of objects** ,**controlspeed** and **endtime.**

★*Simple diagram:*



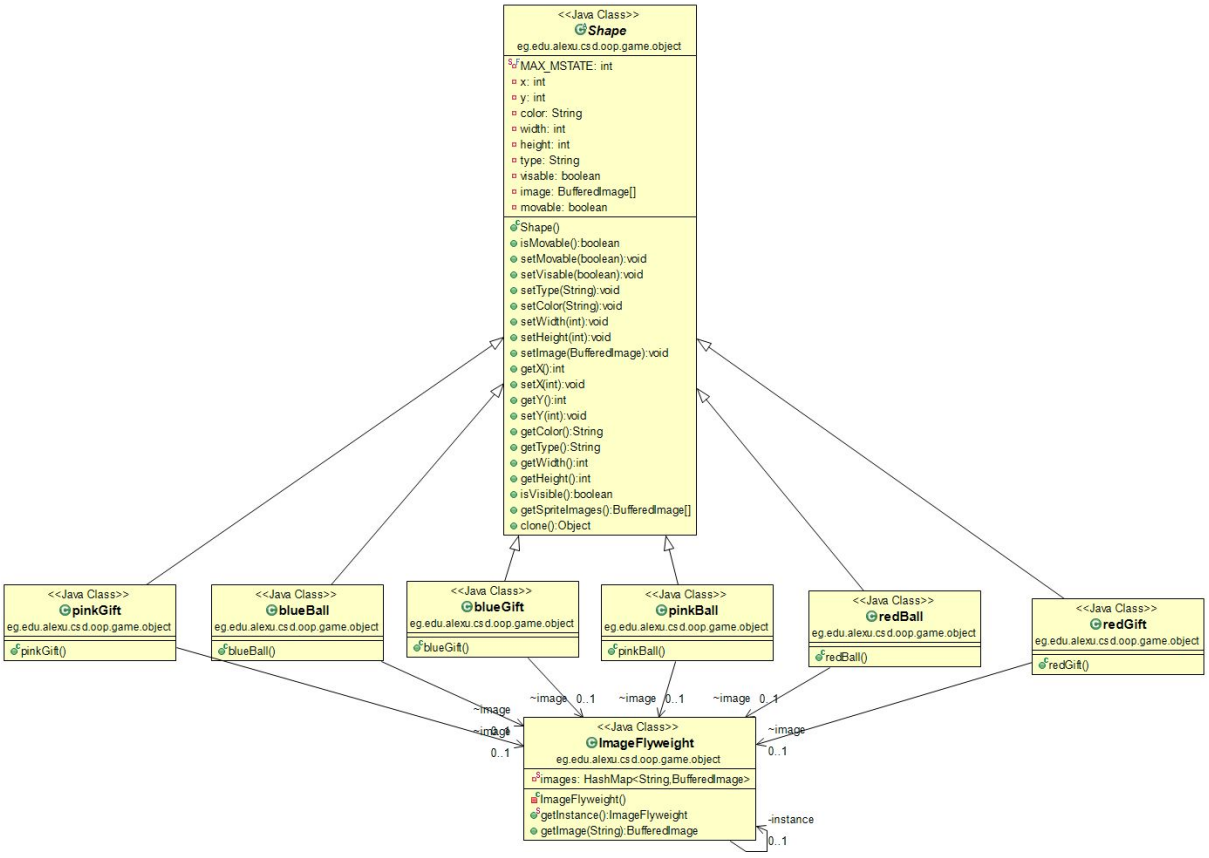
- Flyweight:

★*Usage:*

setting only one image for each shape instance.

★*Components:*

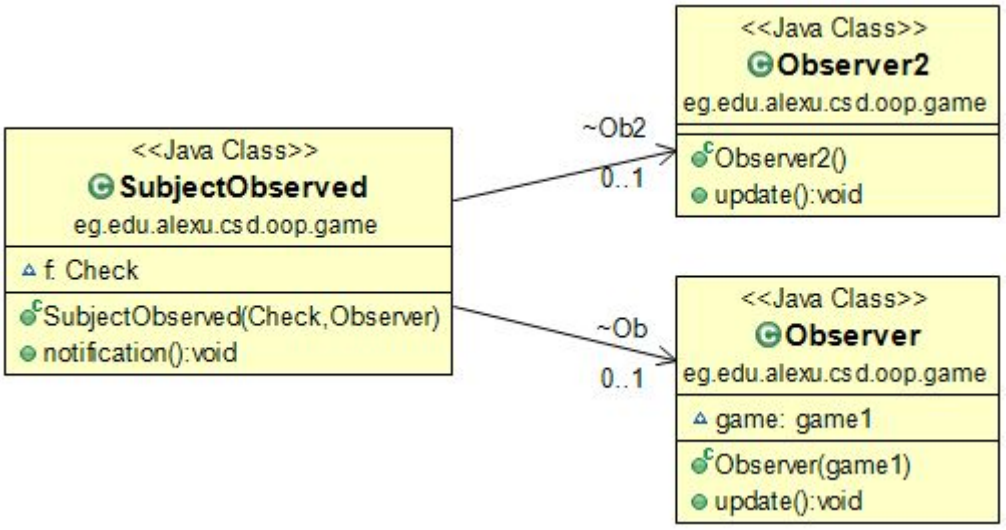
ImageFlyweight class:it has hashMap for all shapes.getImage function takes the shape name as a parameter ,then it checks whether the image of that shape exists in the hashMap or not.



- Observer:

★*usage:*

As we need to notify **score** and **audioplayer** when game have update when player gets 3 same color

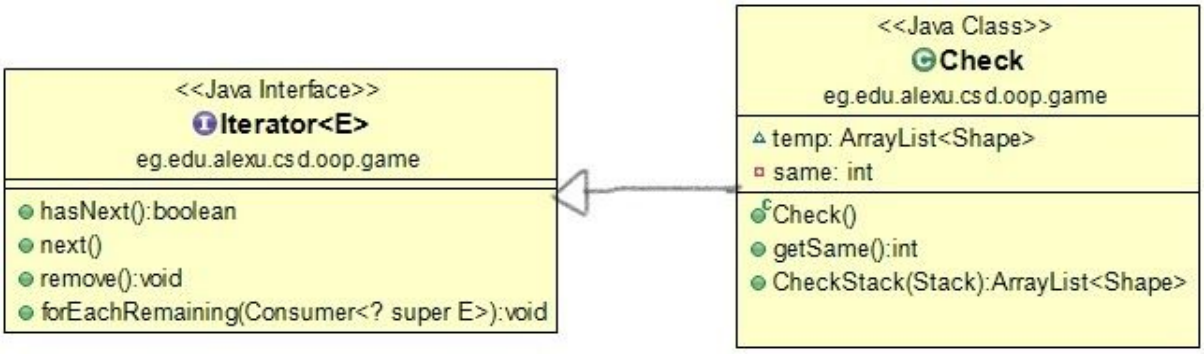


- Iterator:

★*Usage:*

It is used to access the contents of arraylist of shapes.

★*Simple diagram:*



- facade:

★*Usage:*

User interface class is using it to determine the level wanted.

❑ Design Decisions:

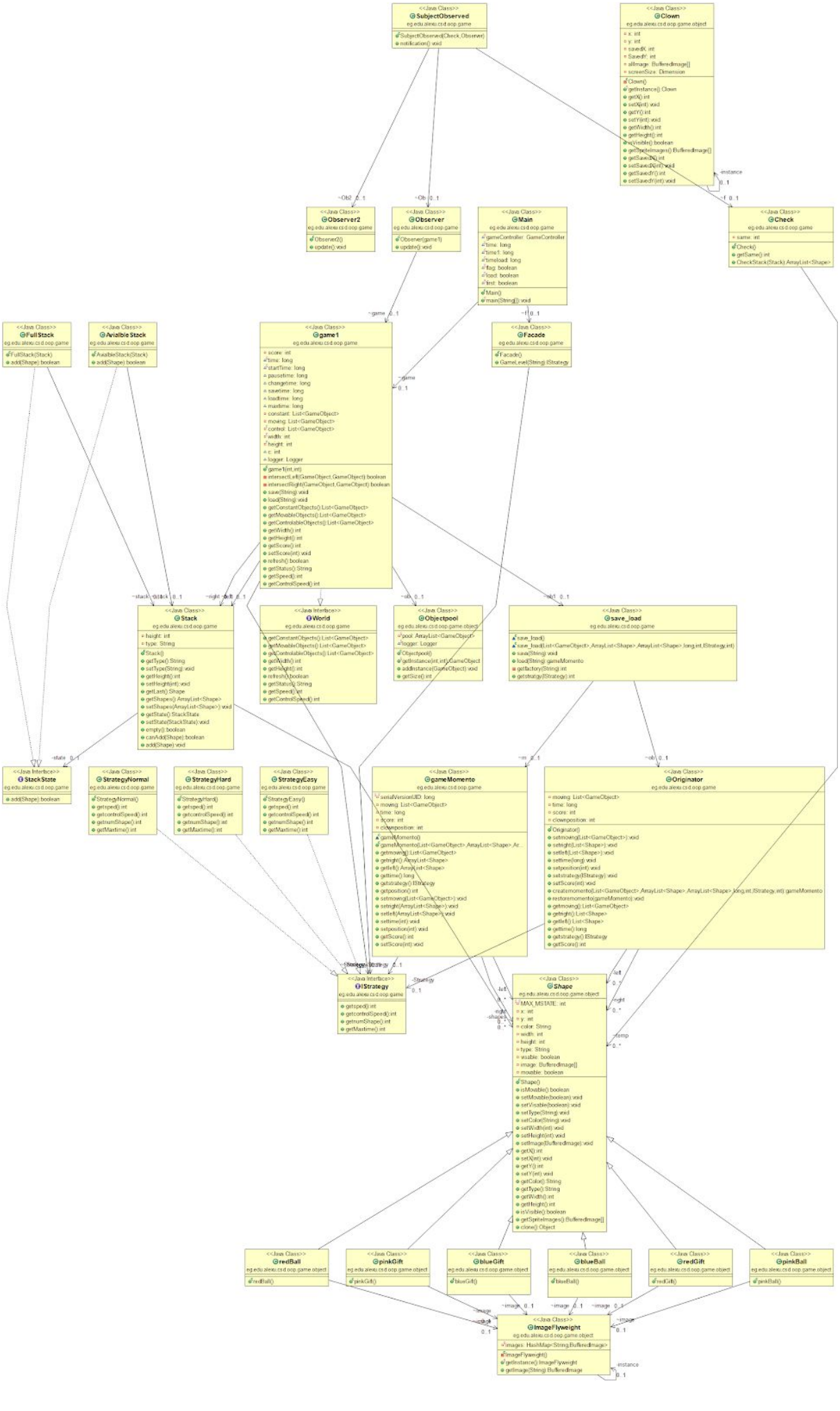
I. It is a 1player game.

II. Player can save the game and load it again to resume playing.

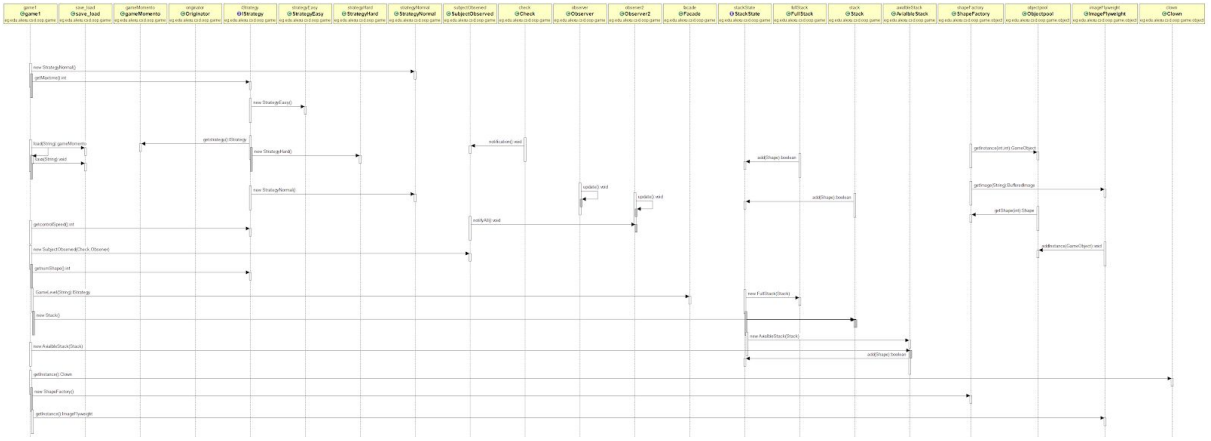
III. Player loses the game if the time is exceeded or number of shapes with the clown exceeded 11.

IV. Score increases by one whenever player collects three shapes of the same color.

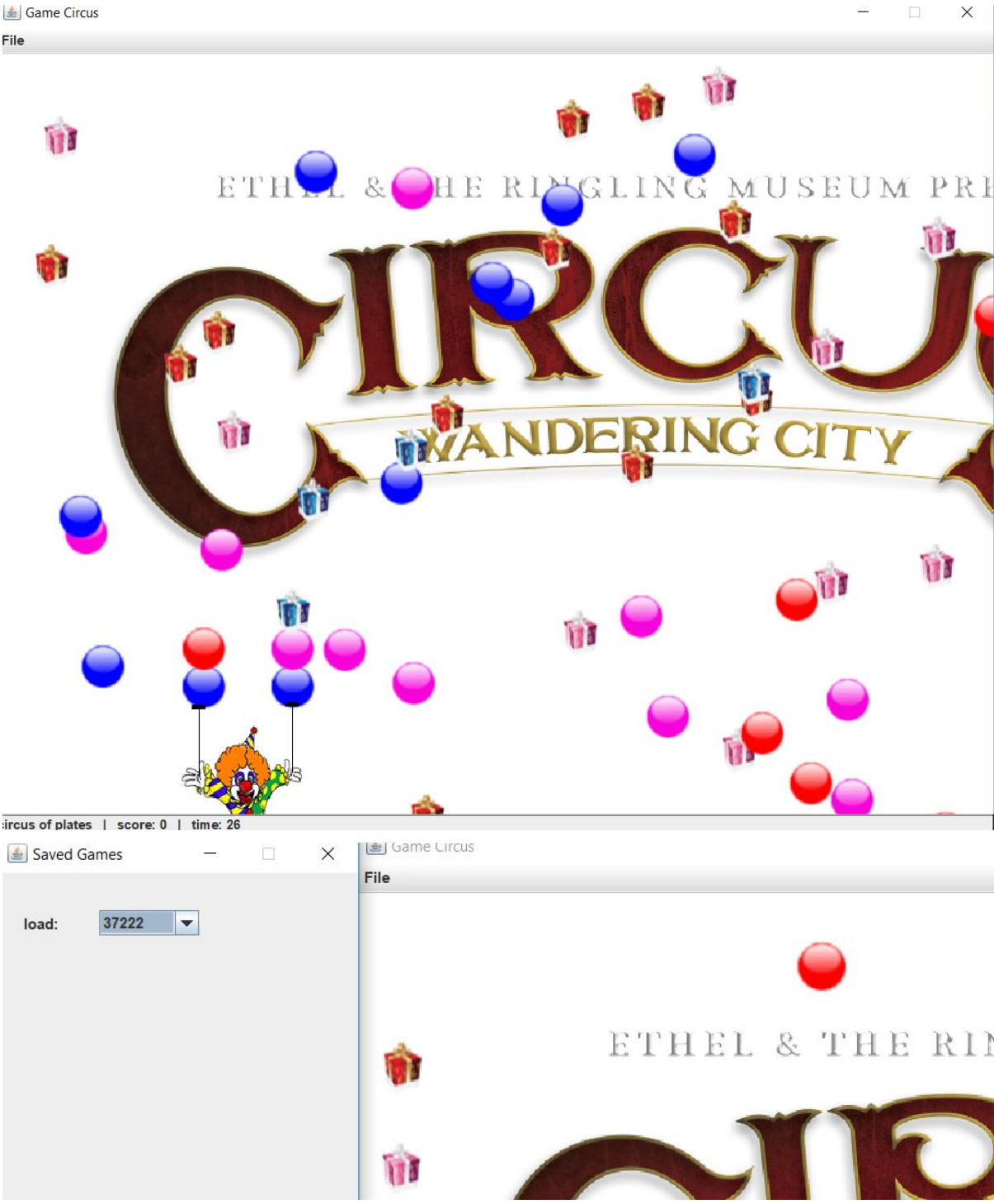
❑ Class diagram:


SubjectObserved
eg.edu.alexu.csd.oop.game
SubjectObserved(Check,Observer)
notification():void
Clown
eg.edu.alexu.csd.oop.game.object
x: int
y: int
savedX: int
SavedY: int
allImage: BufferedImage[]
screenSize: Dimension
Clown()
getInstance():Clown
getX():int
setX(int):void
getY():int
setY(int):void
getWidth():int
getHeight():int
isVisible():boolean
getSpriteImages():BufferedImage[]
getSavedX():int
setSavedX(int):void
getSavedY():int
setSavedY(int):void
-instance
0..1
-Ob2 0..1
-Ob 0..1
-f 0..1
Observer2
eg.edu.alexu.csd.oop.game
Observer2()
update():void
Observer
eg.edu.alexu.csd.oop.game
Observer(game1)
update():void
Main
eg.edu.alexu.csd.oop.game
gameController: GameController
time: long
time1: long
timeload: long
flag: boolean
load: boolean
first: boolean
Main()
main(String[]):void
Check
eg.edu.alexu.csd.oop.game
same: int
Check()
getSame():int
CheckStack(Stack),ArrayList<Shape>
-game 0..1
-f 0..1
FullStack
eg.edu.alexu.csd.oop.game
FullStack(Stack)
add(Shape):boolean
AvaiableStack
eg.edu.alexu.csd.oop.game
AvaiableStack(Stack)
add(Shape):boolean
game1
eg.edu.alexu.csd.oop.game
score: int
time: long
startTime: long
pausetime: long
changetime: long
savetime: long
loadtime: long
maxtime: long
constant: List<GameObject>
moving: List<GameObject>
control: List<GameObject>
width: int
height: int
c: int
logger: Logger
game1(int,int)
intersectLeft(GameObject,GameObject):boolean
intersectRight(GameObject,GameObject):boolean
save(String):void
load(String):void
getConstantObjects():List<GameObject>
getMovableObjects():List<GameObject>
getControlableObjects():List<GameObject>
getWidth():int
getHeight():int
getScore():int
setScore(int):void
refresh():boolean
getStatus():String
getSpeed():int
getControlSpeed():int
Facade
eg.edu.alexu.csd.oop.game
Facade()
GameLevel(String):IStrategy
-game 0..1
-stack 0..1
-tStack 0..1
-right 0..1
-left 0..1
-ob 0..1
-ob1 0..1
Stack
eg.edu.alexu.csd.oop.game
height: int
type: String
Stack()
getType():String
setType(String):void
getHeight():int
setHeight(int):void
getLast():Shape
getShape():ArrayList<Shape>
setShape(ArrayList<Shape>):void
getState():StackState
setState(StackState):void
empty():boolean
canAdd(Shape):boolean
add(Shape):void
World
eg.edu.alexu.csd.oop.game
getConstantObjects():List<GameObject>
getMovableObjects():List<GameObject>
getControlableObjects():List<GameObject>
getWidth():int
getHeight():int
refresh():boolean
getStatus():String
getSpeed():int
getControlSpeed():int
Objectpool
eg.edu.alexu.csd.oop.game.object
pool: ArrayList<GameObject>
logger: Logger
Objectpool()
getInstance(int,int):GameObject
addInstance(GameObject):void
getSize():int
save_load
eg.edu.alexu.csd.oop.game
save_load()
save_load(List<GameObject>,ArrayList<Shape>,ArrayList<Shape>,long,int,IStrategy,int)
save(String):void
load(String):gameMomento
getfactory(String):int
getstrategy(IStrategy):int
-state 0..1
-m 0..1
-ob 0..1
StackState
eg.edu.alexu.csd.oop.game
add(Shape):boolean
StrategyNormal
eg.edu.alexu.csd.oop.game
getspeed():int
getcontrolSpeed():int
getnumShape():int
getMaxtime():int
StrategyHard
eg.edu.alexu.csd.oop.game
getspeed():int
getcontrolSpeed():int
getnumShape():int
getMaxtime():int
StrategyEasy
eg.edu.alexu.csd.oop.game
StrategyEasy()
getspeed():int
getcontrolSpeed():int
getnumShape():int
getMaxtime():int
gameMomento
eg.edu.alexu.csd.oop.game
serialVersionUID: long
moving: List<GameObject>
time: long
score: int
clownposition: int
gameMomento()
gameMomento(List<GameObject>,ArrayList<Shape>,Ar...
getmoving():List<GameObject>
getright():ArrayList<Shape>
getleft():ArrayList<Shape>
gettime():long
getstrategy():IStrategy
getposition():int
setmoving(List<GameObject>):void
setright(ArrayList<Shape>):void
setleft(ArrayList<Shape>):void
settime(int):void
setposition(int):void
getScore():int
setScore(int):void
Originator
eg.edu.alexu.csd.oop.game
moving: List<GameObject>
time: long
score: int
clownposition: int
Originator()
setmoving(List<GameObject>):void
setright(List<Shape>):void
setleft(List<Shape>):void
settime(long):void
setposition(int):void
setstrategy(IStrategy):void
setScore(int):void
createmomento(List<GameObject>,ArrayList<Shape>,ArrayList<Shape>,long,int,IStrategy,int):gameMomento
restorememento(gameMomento):void
getmoving():List<GameObject>
getright():List<Shape>
getleft():List<Shape>
gettime():long
getstrategy():IStrategy
getScore():int
-Strategy 0..1
-Strategy
IStrategy
eg.edu.alexu.csd.oop.game
getspeed():int
getcontrolSpeed():int
getnumShape():int
getMaxtime():int
-left
-right
-shapes
-temp
Shape
eg.edu.alexu.csd.oop.game.object
MAX_MSTATE: int
x: int
y: int
color: String
width: int
height: int
type: String
visable: boolean
image: BufferedImage[]
movable: boolean
Shape()
isMovable():boolean
setMovable(boolean):void
setVisable(boolean):void
setType(String):void
setColor(String):void
setWidth(int):void
setHeight(int):void
setImage(BufferedImage):void
getX():int
setX(int):void
getY():int
setY(int):void
getColor():String
getType():String
getWidth():int
getHeight():int
isVisible():boolean
getSpriteImages():BufferedImage[]
clone():Object
redBall
eg.edu.alexu.csd.oop.game.object
redBall()
pinkGift
eg.edu.alexu.csd.oop.game.object
pinkGift()
blueGift
eg.edu.alexu.csd.oop.game.object
blueGift()
blueBall
eg.edu.alexu.csd.oop.game.object
blueBall()
redGift
eg.edu.alexu.csd.oop.game.object
redGift()
pinkBall
eg.edu.alexu.csd.oop.game.object
pinkBall()
-image 0..1
ImageFlyweight
eg.edu.alexu.csd.oop.game.object
images: HashMap<String,BufferedImage>
ImageFlyweight()
getInstance():ImageFlyweight
getImage(String):BufferedImage
-instance
0..1

❑ Sequence diagram:

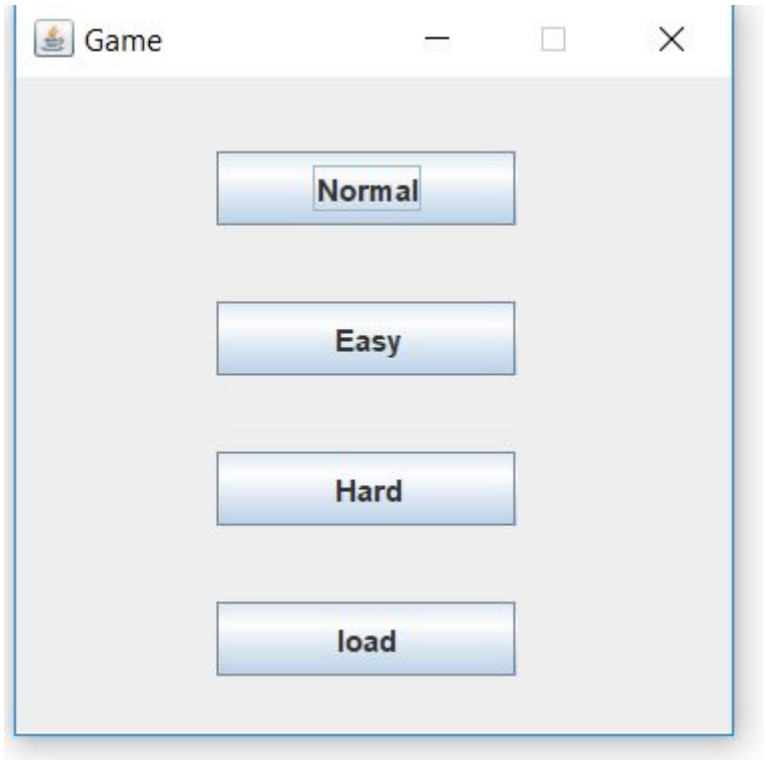


- Sample Runs


Game Circus
File
WANDERING CITY
ircus of plates | score: 0 | time: 26
Saved Games
load:
37222
Game Circus
File


Game Circus
File
ETHEL & THE RINGLING MUSEUM PR
CIRCU
WANDERING CITY
circus of plates | score: 3 | time: 50


Game Circus
File
ETHEL & THE RINGLING MUSEUM PR
CIRCU
WANDERING CITY
circus of plates | score: 2 | time: 47


Game
Normal
Easy
Hard
load