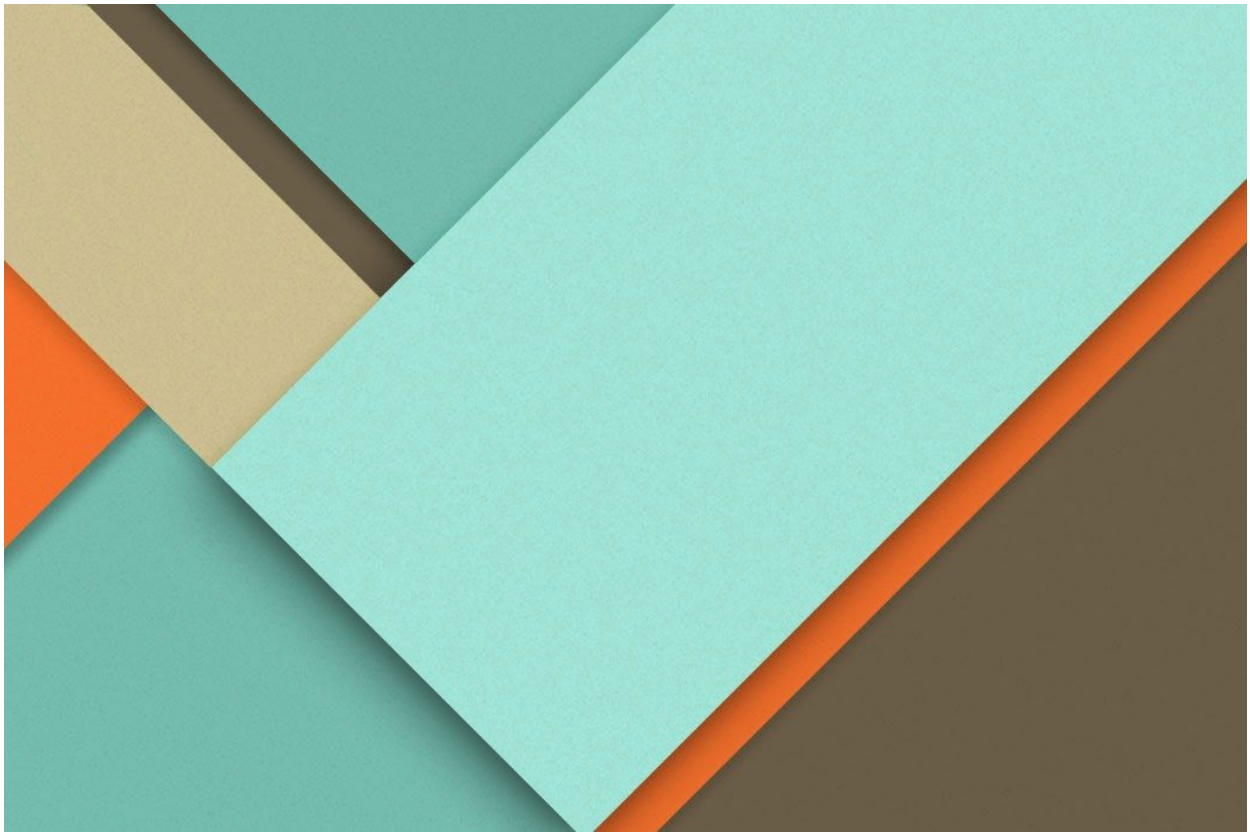


Circus Plates

Ahmed Yasser **10**

Hossam Eldeen Abdelghany **24**

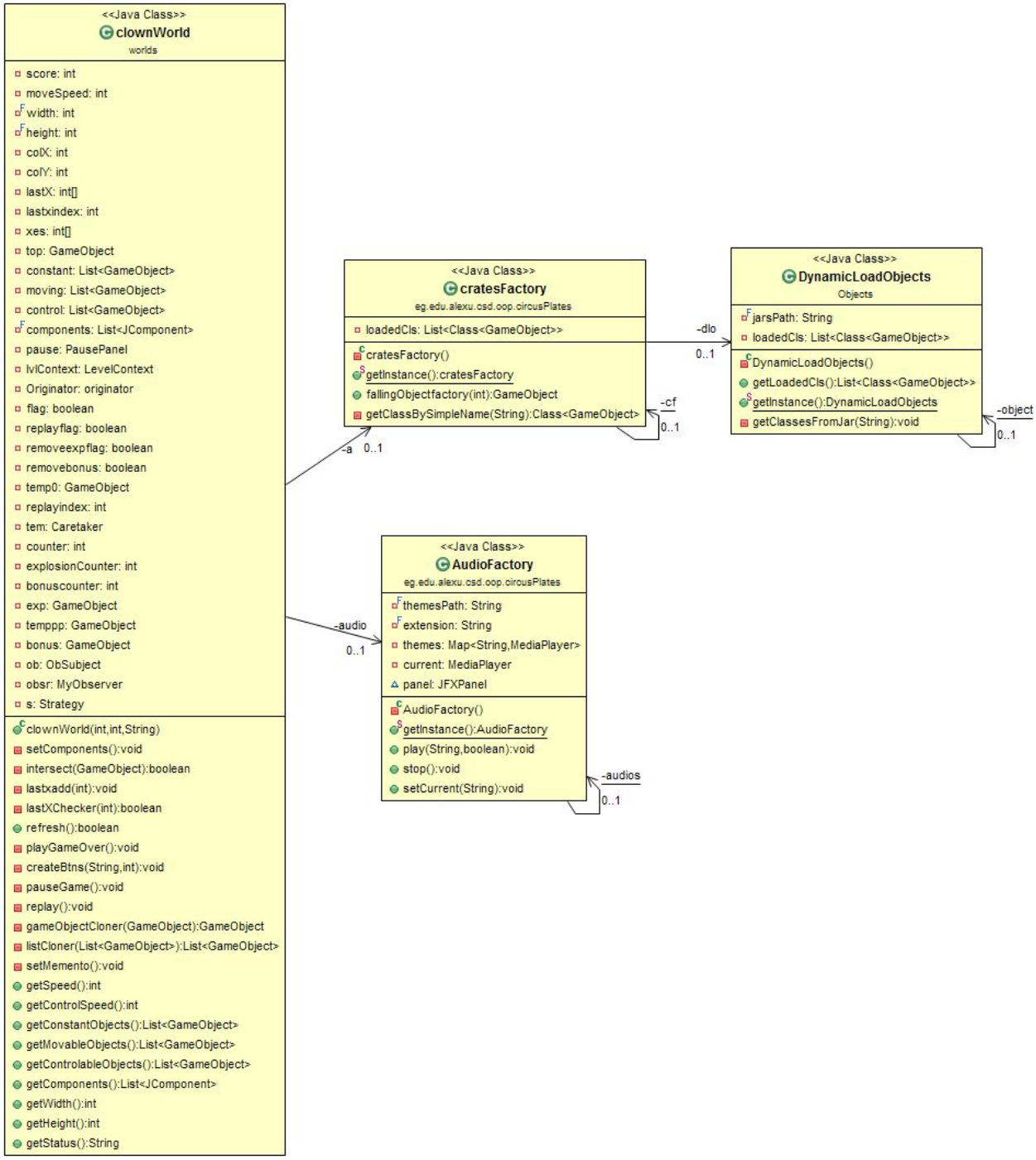
Hassan Abdelhameed Hassan **25**

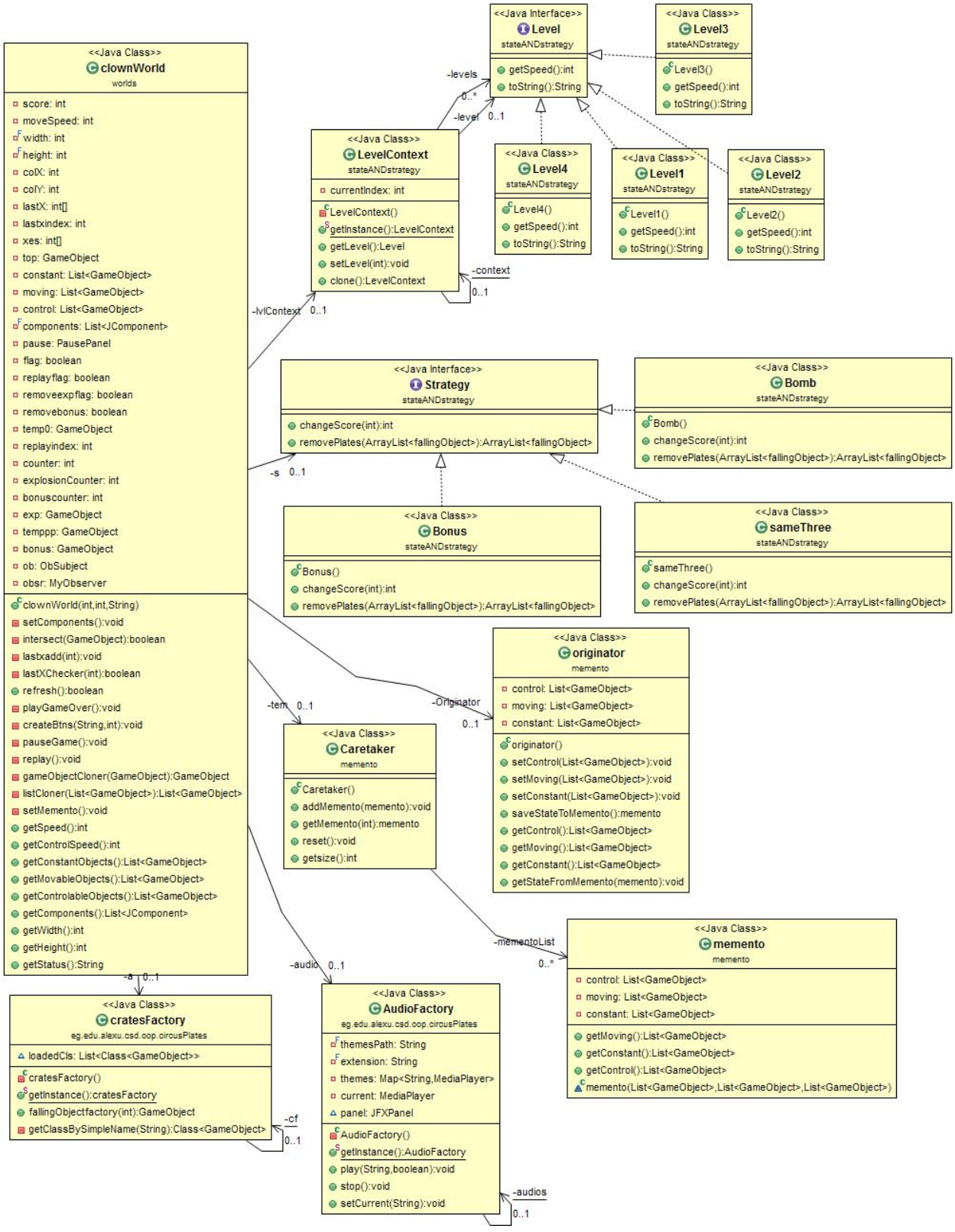
Ramez Maher **28**

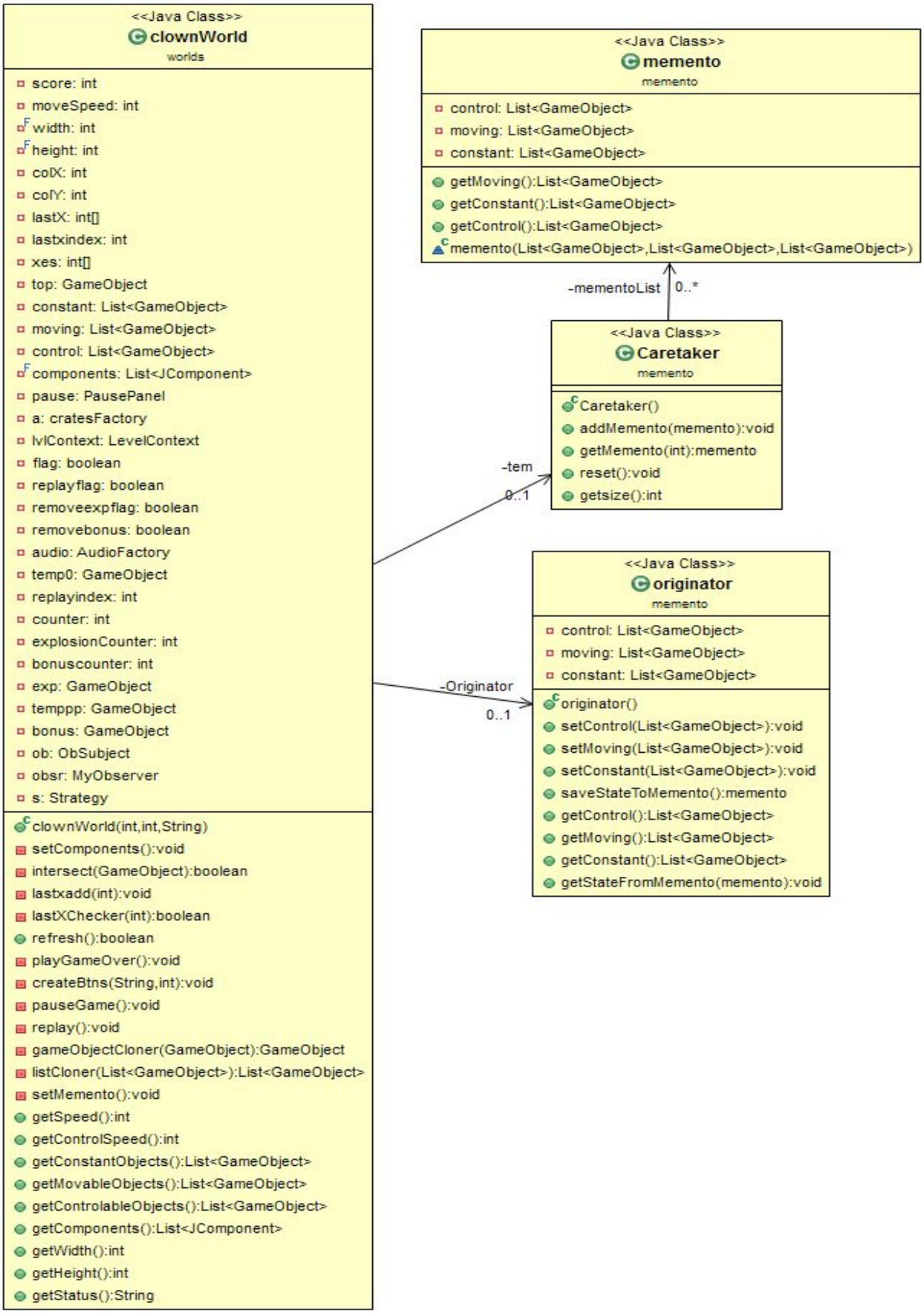
Overview

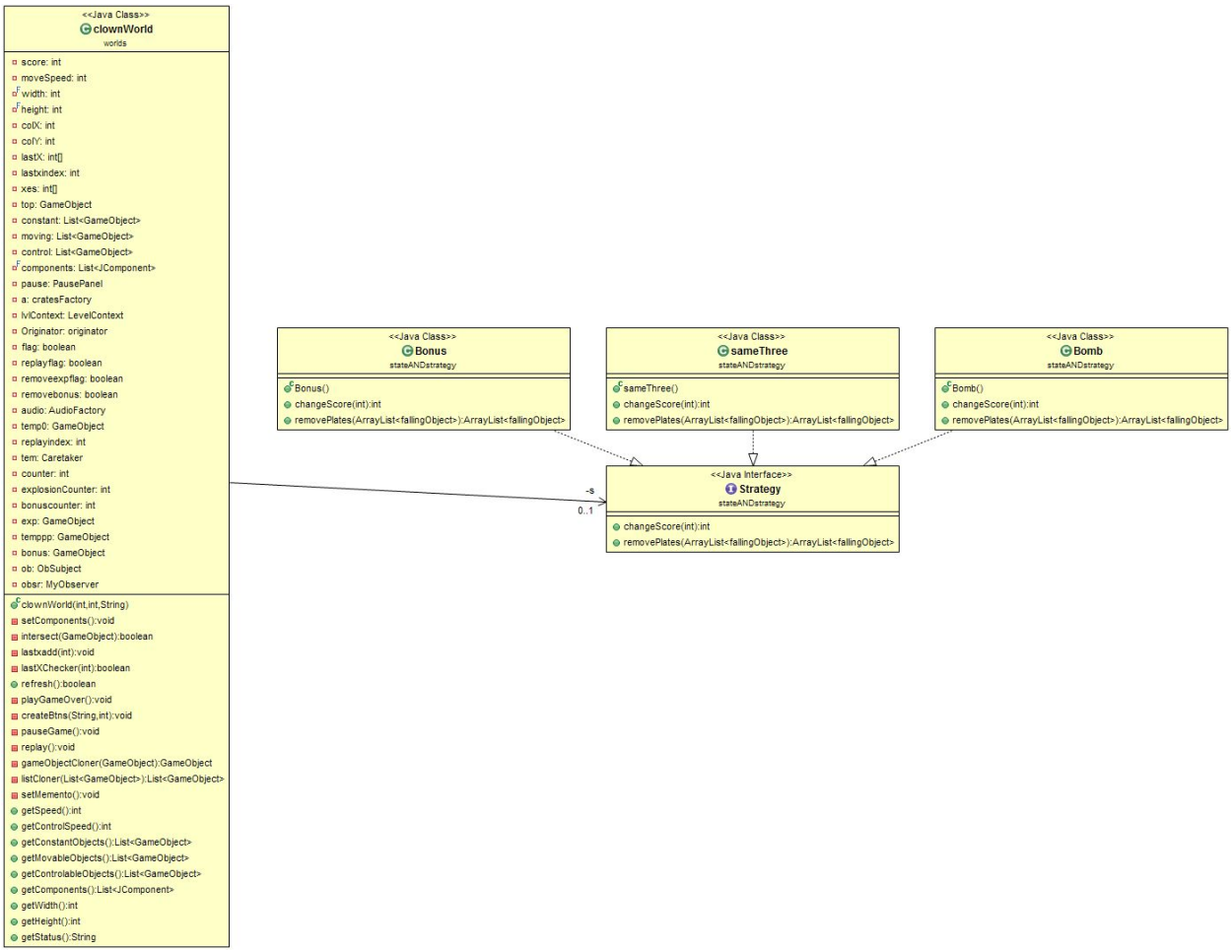
Circus Plates is a single player game where a clown holds a stack of plates and when three consecutive plates of the same color are collected they vanish and the score increases.

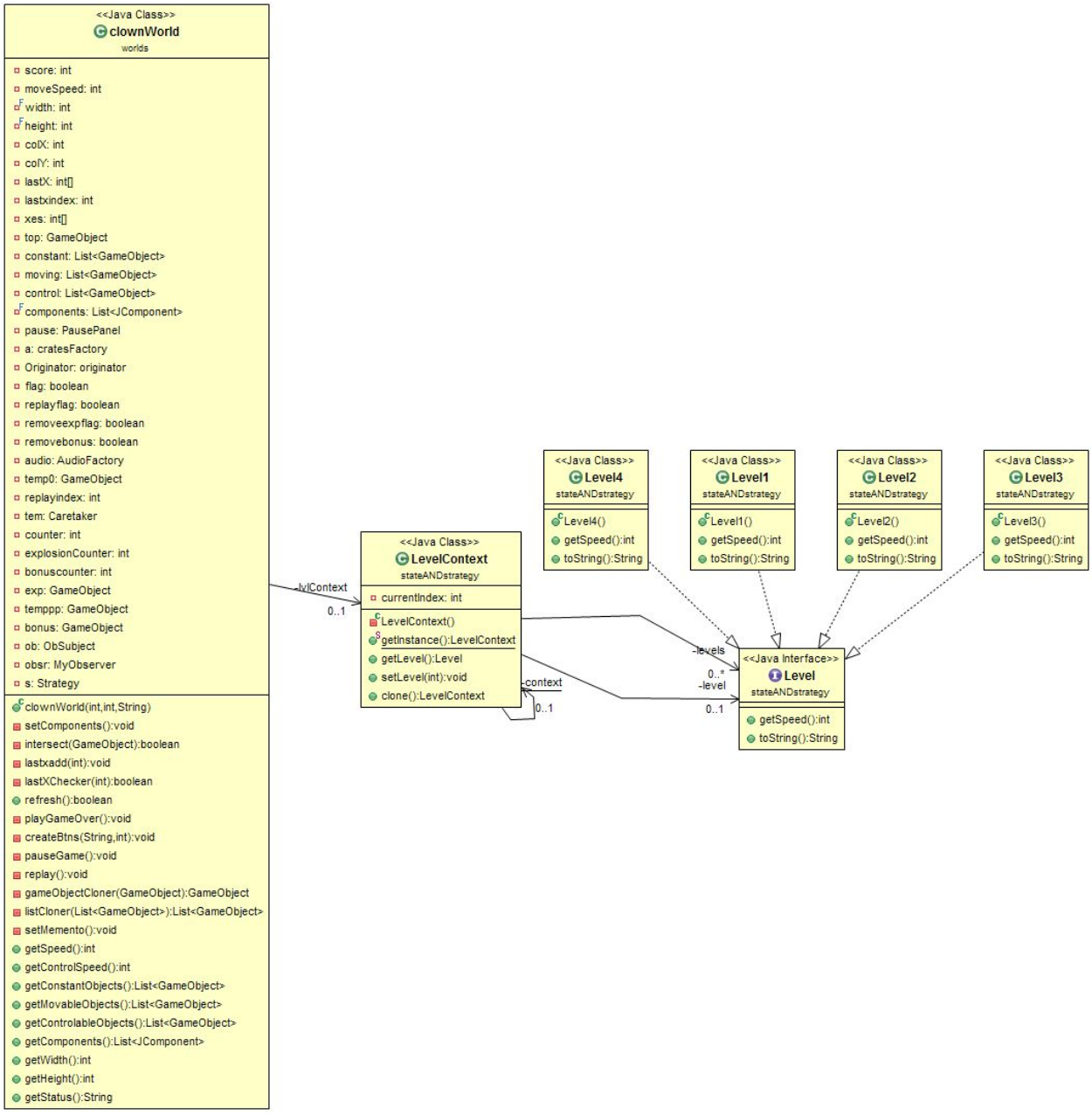
Class Diagram

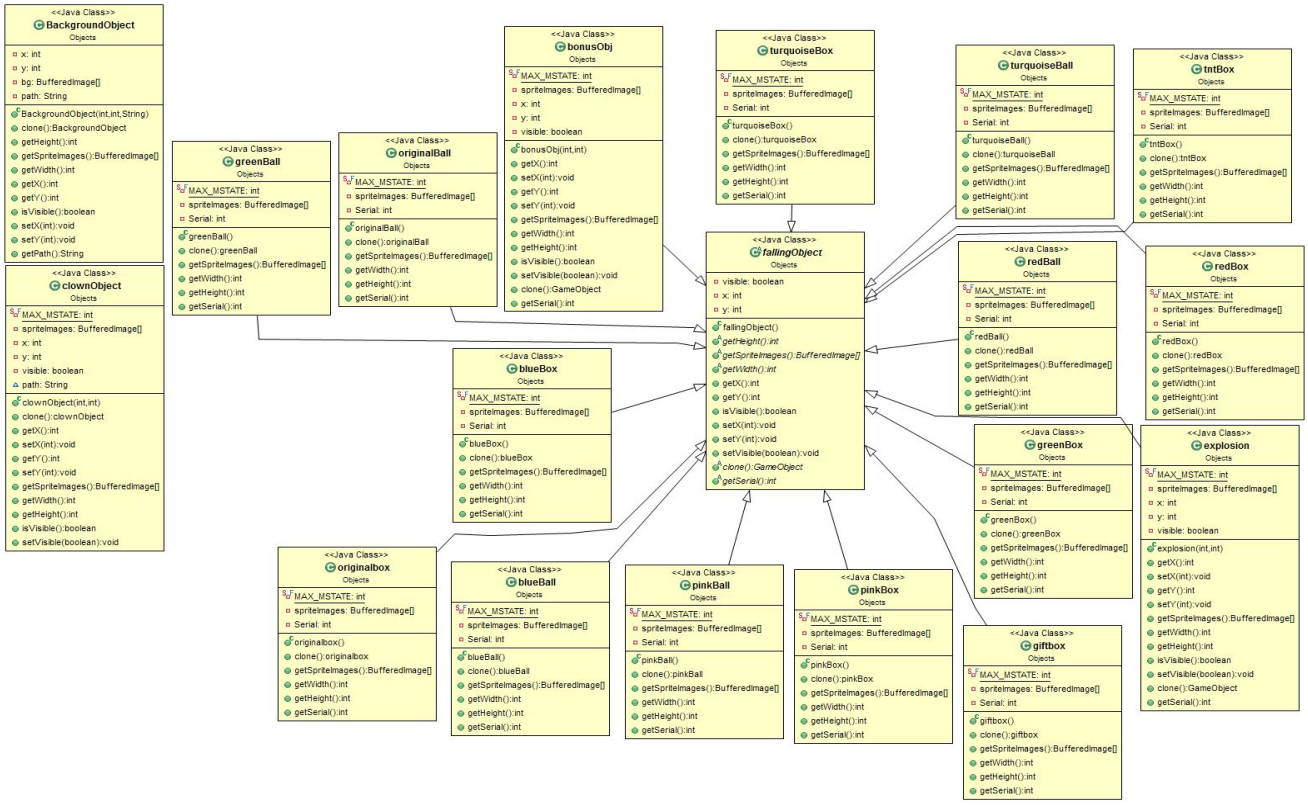

<<Java Class>>
clownWorld
worlds
score: int
moveSpeed: int
width: int
height: int
colX: int
colY: int
lastX: int[]
lastxindex: int
xes: int[]
top: GameObject
constant: List<GameObject>
moving: List<GameObject>
control: List<GameObject>
components: List<JComponent>
pause: PausePanel
lvlContext: LevelContext
Originator: originator
flag: boolean
replayflag: boolean
removeexpflag: boolean
removebonus: boolean
temp0: GameObject
replayindex: int
tem: Caretaker
counter: int
explosionCounter: int
bonuscounter: int
exp: GameObject
temppp: GameObject
bonus: GameObject
ob: ObSubject
obsr: MyObserver
s: Strategy
clownWorld(int,int,String)
setComponents():void
intersect(GameObject):boolean
lastxadd(int):void
lastXChecker(int):boolean
refresh():boolean
playGameOver():void
createBtns(String,int):void
pauseGame():void
replay():void
gameObjectCloner(GameObject):GameObject
listCloner(List<GameObject>):List<GameObject>
setMemento():void
getSpeed():int
getControlSpeed():int
getConstantObjects():List<GameObject>
getMovableObjects():List<GameObject>
getControlableObjects():List<GameObject>
getComponents():List<JComponent>
getWidth():int
getHeight():int
getStatus():String
<<Java Class>>
cratesFactory
eg.edu.alexu.csd.oop.circusPlates
loadedCls: List<Class<GameObject>>
cratesFactory()
getInstance():cratesFactory
fallingObjectfactory(int):GameObject
getClassBySimpleName(String):Class<GameObject>
-a
0..1
-cf
0..1
-dlo
0..1
<<Java Class>>
DynamicLoadObjects
Objects
jarsPath: String
loadedCls: List<Class<GameObject>>
DynamicLoadObjects()
getLoadedCls():List<Class<GameObject>>
getInstance():DynamicLoadObjects
getClassesFromJar(String):void
-object
0..1
-audio
0..1
<<Java Class>>
AudioFactory
eg.edu.alexu.csd.oop.circusPlates
themesPath: String
extension: String
themes: Map<String,MediaPlayer>
current: MediaPlayer
panel: JFXPanel
AudioFactory()
getInstance():AudioFactory
play(String,boolean):void
stop():void
setCurrent(String):void
-audios
0..1


<<Java Class>>
clownWorld
worlds
score: int
moveSpeed: int
width: int
height: int
colX: int
colY: int
lastX: int[]
lastxindex: int
xes: int[]
top: GameObject
constant: List<GameObject>
moving: List<GameObject>
control: List<GameObject>
components: List<JComponent>
pause: PausePanel
flag: boolean
replayflag: boolean
removeexpflag: boolean
removebonus: boolean
temp0: GameObject
replayindex: int
counter: int
explosionCounter: int
bonuscounter: int
exp: GameObject
temppp: GameObject
bonus: GameObject
ob: ObSubject
obsr: MyObserver
clownWorld(int,int,String)
setComponents():void
intersect(GameObject):boolean
lastxadd(int):void
lastXChecker(int):boolean
refresh():boolean
playGameOver():void
createBtns(String,int):void
pauseGame():void
replay():void
gameObjectCloner(GameObject):GameObject
listCloner(List<GameObject>):List<GameObject>
setMemento():void
getSpeed():int
getControlSpeed():int
getConstantObjects():List<GameObject>
getMovableObjects():List<GameObject>
getControlableObjects():List<GameObject>
getComponents():List<JComponent>
getWidth():int
getHeight():int
getStatus():String
<<Java Interface>>
Level
stateANDstrategy
getSpeed():int
toString():String
<<Java Class>>
Level3
stateANDstrategy
Level3()
getSpeed():int
toString():String
-levels
0..*
-level
0..1
<<Java Class>>
LevelContext
stateANDstrategy
currentIndex: int
LevelContext()
getInstance():LevelContext
getLevel():Level
setLevel(int):void
clone():LevelContext
-context
0..1
<<Java Class>>
Level4
stateANDstrategy
Level4()
getSpeed():int
toString():String
<<Java Class>>
Level1
stateANDstrategy
Level1()
getSpeed():int
toString():String
<<Java Class>>
Level2
stateANDstrategy
Level2()
getSpeed():int
toString():String
-lvlContext
0..1
<<Java Interface>>
Strategy
stateANDstrategy
changeScore(int):int
removePlates(ArrayList<fallingObject>):ArrayList<fallingObject>
<<Java Class>>
Bomb
stateANDstrategy
Bomb()
changeScore(int):int
removePlates(ArrayList<fallingObject>):ArrayList<fallingObject>
-s
0..1
<<Java Class>>
Bonus
stateANDstrategy
Bonus()
changeScore(int):int
removePlates(ArrayList<fallingObject>):ArrayList<fallingObject>
<<Java Class>>
sameThree
stateANDstrategy
sameThree()
changeScore(int):int
removePlates(ArrayList<fallingObject>):ArrayList<fallingObject>
<<Java Class>>
originator
memento
control: List<GameObject>
moving: List<GameObject>
constant: List<GameObject>
originator()
setControl(List<GameObject>):void
setMoving(List<GameObject>):void
setConstant(List<GameObject>):void
saveStateToMemento():memento
getControl():List<GameObject>
getMoving():List<GameObject>
getConstant():List<GameObject>
getStateFromMemento(memento):void
-Originator
0..1
-tem
0..1
<<Java Class>>
Caretaker
memento
Caretaker()
addMemento(memento):void
getMemento(int):memento
reset():void
getsize():int
-mementoList
0..*
<<Java Class>>
memento
memento
control: List<GameObject>
moving: List<GameObject>
constant: List<GameObject>
getMoving():List<GameObject>
getConstant():List<GameObject>
getControl():List<GameObject>
memento(List<GameObject>,List<GameObject>,List<GameObject>)
-a
0..1
-audio
0..1
<<Java Class>>
cratesFactory
eg.edu.alexu.csd.oop.circusPlates
loadedCls: List<Class<GameObject>>
cratesFactory()
getInstance():cratesFactory
fallingObjectfactory(int):GameObject
getClassBySimpleName(String):Class<GameObject>
-cf
0..1
<<Java Class>>
AudioFactory
eg.edu.alexu.csd.oop.circusPlates
themesPath: String
extension: String
themes: Map<String,MediaPlayer>
current: MediaPlayer
panel: JFXPanel
AudioFactory()
getInstance():AudioFactory
play(String,boolean):void
stop():void
setCurrent(String):void
-audios
0..1

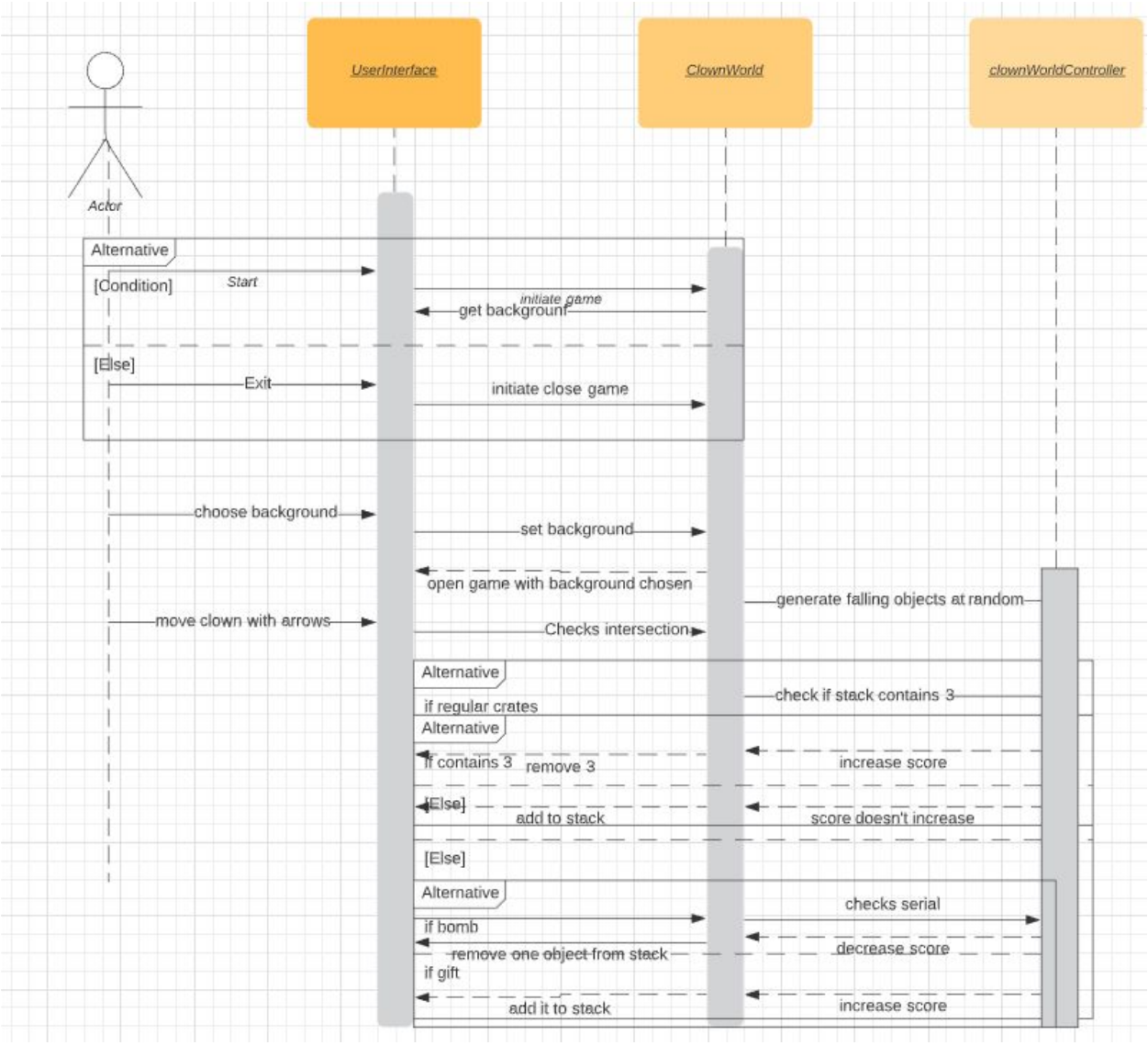

<<Java Class>>
clownWorld
worlds
score: int
moveSpeed: int
width: int
height: int
colX: int
colY: int
lastX: int[]
lastxindex: int
xes: int[]
top: GameObject
constant: List<GameObject>
moving: List<GameObject>
control: List<GameObject>
components: List<JComponent>
pause: PausePanel
a: cratesFactory
lvlContext: LevelContext
flag: boolean
replayflag: boolean
removeexpflag: boolean
removebonus: boolean
audio: AudioFactory
temp0: GameObject
replayindex: int
counter: int
explosionCounter: int
bonuscounter: int
exp: GameObject
temppp: GameObject
bonus: GameObject
ob: ObSubject
obsr: MyObserver
s: Strategy
clownWorld(int,int,String)
setComponents():void
intersect(GameObject):boolean
lastxadd(int):void
lastXChecker(int):boolean
refresh():boolean
playGameOver():void
createBtns(String,int):void
pauseGame():void
replay():void
gameObjectCloner(GameObject):GameObject
listCloner(List<GameObject>):List<GameObject>
setMemento():void
getSpeed():int
getControlSpeed():int
getConstantObjects():List<GameObject>
getMovableObjects():List<GameObject>
getControlableObjects():List<GameObject>
getComponents():List<JComponent>
getWidth():int
getHeight():int
getStatus():String
<<Java Class>>
memento
memento
control: List<GameObject>
moving: List<GameObject>
constant: List<GameObject>
getMoving():List<GameObject>
getConstant():List<GameObject>
getControl():List<GameObject>
memento(List<GameObject>,List<GameObject>,List<GameObject>)
-mementoList
0..*
<<Java Class>>
Caretaker
memento
Caretaker()
addMemento(memento):void
getMemento(int):memento
reset():void
getsize():int
-tem
0..1
<<Java Class>>
originator
memento
control: List<GameObject>
moving: List<GameObject>
constant: List<GameObject>
originator()
setControl(List<GameObject>):void
setMoving(List<GameObject>):void
setConstant(List<GameObject>):void
saveStateToMemento():memento
getControl():List<GameObject>
getMoving():List<GameObject>
getConstant():List<GameObject>
getStateFromMemento(memento):void
-Originator
0..1


<<Java Class>>
clownWorld
worlds
score: int
moveSpeed: int
width: int
height: int
colX: int
colY: int
lastX: int[]
lastxindex: int
xes: int[]
top: GameObject
constant: List<GameObject>
moving: List<GameObject>
control: List<GameObject>
components: List<JComponent>
pause: PausePanel
a: cratesFactory
lvlContext: LevelContext
Originator: originator
flag: boolean
replayflag: boolean
removeexpflag: boolean
removebonus: boolean
audio: AudioFactory
temp0: GameObject
replayindex: int
tem: Caretaker
counter: int
explosionCounter: int
bonuscounter: int
exp: GameObject
temppp: GameObject
bonus: GameObject
ob: ObSubject
obsr: MyObserver
clownWorld(int,int,String)
setComponents():void
intersect(GameObject):boolean
lastxadd(int):void
lastXChecker(int):boolean
refresh():boolean
playGameOver():void
createBtns(String,int):void
pauseGame():void
replay():void
gameObjectCloner(GameObject):GameObject
listCloner(List<GameObject>):List<GameObject>
setMemento():void
getSpeed():int
getControlSpeed():int
getConstantObjects():List<GameObject>
getMovableObjects():List<GameObject>
getControlableObjects():List<GameObject>
getComponents():List<JComponent>
getWidth():int
getHeight():int
getStatus():String
-s
0..1
<<Java Class>>
Bonus
stateANDstrategy
Bonus()
changeScore(int):int
removePlates(ArrayList<fallingObject>):ArrayList<fallingObject>
<<Java Class>>
sameThree
stateANDstrategy
sameThree()
changeScore(int):int
removePlates(ArrayList<fallingObject>):ArrayList<fallingObject>
<<Java Class>>
Bomb
stateANDstrategy
Bomb()
changeScore(int):int
removePlates(ArrayList<fallingObject>):ArrayList<fallingObject>
<<Java Interface>>
Strategy
stateANDstrategy
changeScore(int):int
removePlates(ArrayList<fallingObject>):ArrayList<fallingObject>


<<Java Class>>
clownWorld
worlds
score: int
moveSpeed: int
width: int
height: int
colX: int
colY: int
lastX: int[]
lastxindex: int
xes: int[]
top: GameObject
constant: List<GameObject>
moving: List<GameObject>
control: List<GameObject>
components: List<JComponent>
pause: PausePanel
a: cratesFactory
Originator: originator
flag: boolean
replayflag: boolean
removeexpflag: boolean
removebonus: boolean
audio: AudioFactory
temp0: GameObject
replayindex: int
tem: Caretaker
counter: int
explosionCounter: int
bonuscounter: int
exp: GameObject
temppp: GameObject
bonus: GameObject
ob: ObSubject
obsr: MyObserver
s: Strategy
clownWorld(int,int,String)
setComponents():void
intersect(GameObject):boolean
lastxadd(int):void
lastXChecker(int):boolean
refresh():boolean
playGameOver():void
createBtns(String,int):void
pauseGame():void
replay():void
gameObjectCloner(GameObject):GameObject
listCloner(List<GameObject>):List<GameObject>
setMemento():void
getSpeed():int
getControlSpeed():int
getConstantObjects():List<GameObject>
getMovableObjects():List<GameObject>
getControlableObjects():List<GameObject>
getComponents():List<JComponent>
getWidth():int
getHeight():int
getStatus():String
-lvlContext
0..1
<<Java Class>>
LevelContext
stateANDstrategy
currentIndex: int
LevelContext()
getInstance():LevelContext
getLevel():Level
setLevel(int):void
clone():LevelContext
-context
0..1
<<Java Class>>
Level4
stateANDstrategy
Level4()
getSpeed():int
toString():String
<<Java Class>>
Level1
stateANDstrategy
Level1()
getSpeed():int
toString():String
<<Java Class>>
Level2
stateANDstrategy
Level2()
getSpeed():int
toString():String
<<Java Class>>
Level3
stateANDstrategy
Level3()
getSpeed():int
toString():String
-levels
0..*
-level
0..1
<<Java Interface>>
Level
stateANDstrategy
getSpeed():int
toString():String



Sequence Diagram



User manual

On running the game ,Main menu appears having some buttons .

The buttons and what they do are :

Start : starts the game and takes the user to three choices of backgrounds whether night , ice or fire mood.

Exit : ends the game.

Info: tells the user how to play the game

After choosing a theme to the game ,the game starts and Another buttons appear.

Pause:pauses the game and a menu having three buttons appear which are :

Resume:continues where the user paused

Restart:restarts the game

Main menu: stops the game and returns to the main menu.

Replay:The last movements of the user are replayed until the frame at which the user pressed replay button

On Playing the game the user will find below a ***status bar*** that show the status of the player and is updated every time a change happens .

The status bar contains:

Location of the clown

Score

Current level

Now,the most important question is :

HOW TO PLAY THE GAME ????

Which takes us to the next point which is

Important design decision and Game Features:

1.**Score**:On collecting three plates **of the same color** the score increases by five and three plates vanish

2.**Losing condition**: the user loses when the stack reaches maximum height

3.**Bomb and Bonus** : if a bomb is collected the user loses a point but removes a plate while if a gift box is collected the user gains three points but three gifts cannot be collected so it increases the size of stack ,So the user has to choose what is suitable

4.**Shapes**: there are two shapes balls and boxes with their colors .

5.**HighScore**: If the user breaks a new high score then the highest score that he reaches will be saved as a new high score.BUT, it is only saved when the user completes the game until he loses.

6.**Multiple levels**:As the score increases,levels increase until reaching the most difficult level which is level 4.But,once a level is reached , it does not decrease .It only gets higher.

7.**Replay**:The last moves made by the user can be replayed on clicking replay button.

8. The user can pause then choose whether restart , resume or go to main menu.

9.Status Bar: shows all information needed by the user like location , score , level and NEW HIGH SCORE!!! If a new high score is reached.

10.**Animation and sounds**:there are sounds and animation of a bomb and bonus and a sound for high score.

11.**The most important part about the game which is**

Design Patterns

Singleton

Involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without the need to instantiate the object of the class.

Singleton is used in classes:**cratesFactory,DynamicLoadObjects,AudioFactory**

Iterator :**we used the iterator in java**

Used it in clownWorld in refresh method to iterate on the list of moving objects

```
public boolean refresh() {
    Iterator<GameObject> itr = moving.iterator();
    while(itr.hasNext()){
        if(((GameObject)itr.next()).getY()>height){itr.remove();}
    }
```

Used it in obSubject class to iterate on the list of observers to notify them with updates

```
public void notifyObs (ArrayList<fallingObject> arr) {
    Iterator iterator = listObs.iterator();
    while(iterator.hasNext()) {
        ((MyObserver)iterator.next()).update(arr);
    }
}

}
```

Dynamic Linkage

Used it in the class of DynamicLoadObjects to load classes of different shapes from jar

Flyweight

Used in class audioFactory where audios are saved in a hashmap

Observer

The interface MyObserver is implemented by circusObserver these observers are notified and updated by class obSubject.Observer design pattern observes the stack of shapes collected by the clown and there is an integer,updates, where according to this integer it is decided whether the user loses or score or increases.

Facade

Involves a single class which provides simplified methods required by client where it aims to hide the complexities of the system and provide an interface to the client using which the client can access the system.

Used in **circusfacade** class where this class is instantiated in the interface and when the user chooses a specific background all the ui does is call choosecircus method in circusfacade which is very simple but inside the circusfacade it is responsible for starting the game according to the chosen background which is a parameter in method choosecircus .

Strategy

`This patterns aims to change` a class behavior or its algorithm at run time where there is an interface this interface has a method and then there are classes implementing this interface in each class there is a different implementation of this method .
What does this have to do with changing the behaviour at run time ???

There is a Strategy interface . This interface is implemented by three classes which are sameThree , Bomb and Bonus . we declare an object of strategy interface and according to a certain number , updates is an integer obtained from the observer which tells us what should happen whether increase score and remove plates in case of three similar plates or remove plate and decrease score in case of bomb or give bonus in case of gift box or lose when the stack reaches maximum height .Now we know what we should do so in case of the first case we initialize the strategy object as an object of sameThree class **OR** in second case we initialize the strategy object as an object of Bomb **OR** in third case we initialize the strategy object as an object of Bonus.Then we call method change score and change in stack .**And here comes the answer ,** *the implementation of this method in each class is different and in each time refresh happens the Strategy object is initialized in a different way and the method is implemented in a different way and all this happens at run time*

State

State design pattern is used when an Object changes its behavior based on its internal state.In this game state design pattern is used to change the levels of the game depending on the score . **HOW???**

First of all, State design pattern is applied to the game where there are level context class , level interface and level1,level2,level3 and level4 classes implementing level interface.Level context class maintains reference of a concrete class from those implementing level interface and also there is logic inside that decides at which level the game is .This logic is that it chooses the level depending on the score . Level interface encapsulates the methods that should be implemented according to at which state or level the game is.Level interface has method called getmovespeed .This method decides the movespeed which differs according to each level

An object of level context is initialized in clown world and then we get the speed from the level reference inside it.

Factory

This pattern is that when a method returns one of several possible classes that share the same superclass.This is used in *cratesFactory* class.The method *fallingObjectFactory* returns one of the classes of the shapes.BUT,all these shapes share one superclass which is fallingObject.This method takes a number and according to this number it returns a specific shape.

N.B

The number which is given to the method is chosen by random.

Memento

It is a way to save previous states of an object.Where it is used in this game in Memento,Originator and careTaker.In this game it is used to replay the last frames of the game.Originator is used to set the current state with all its attributes to a memento where this memento is saved in a list inside the

caretaker and when the user wants to replay we loop on the list of memento and make it the current state until reaching the frame at which the user was before clicking replay button.

Mvc

The classes of the code are constructed and divided following the mvc model the classes that hold the data are separate from the classes that control the flow of the code “worldController” and are separate from the gui classes ”clown world” which gives the code a better dynamic and easy readability