Alexandria University

Second Year Computer Engineering
Object Oriented Programming

Game Report.

Mohamed Ahmed Ramadan
Mohamed Ahmed Al-Rakad
Ziad
Mohamed Youssef

December 24, 2015

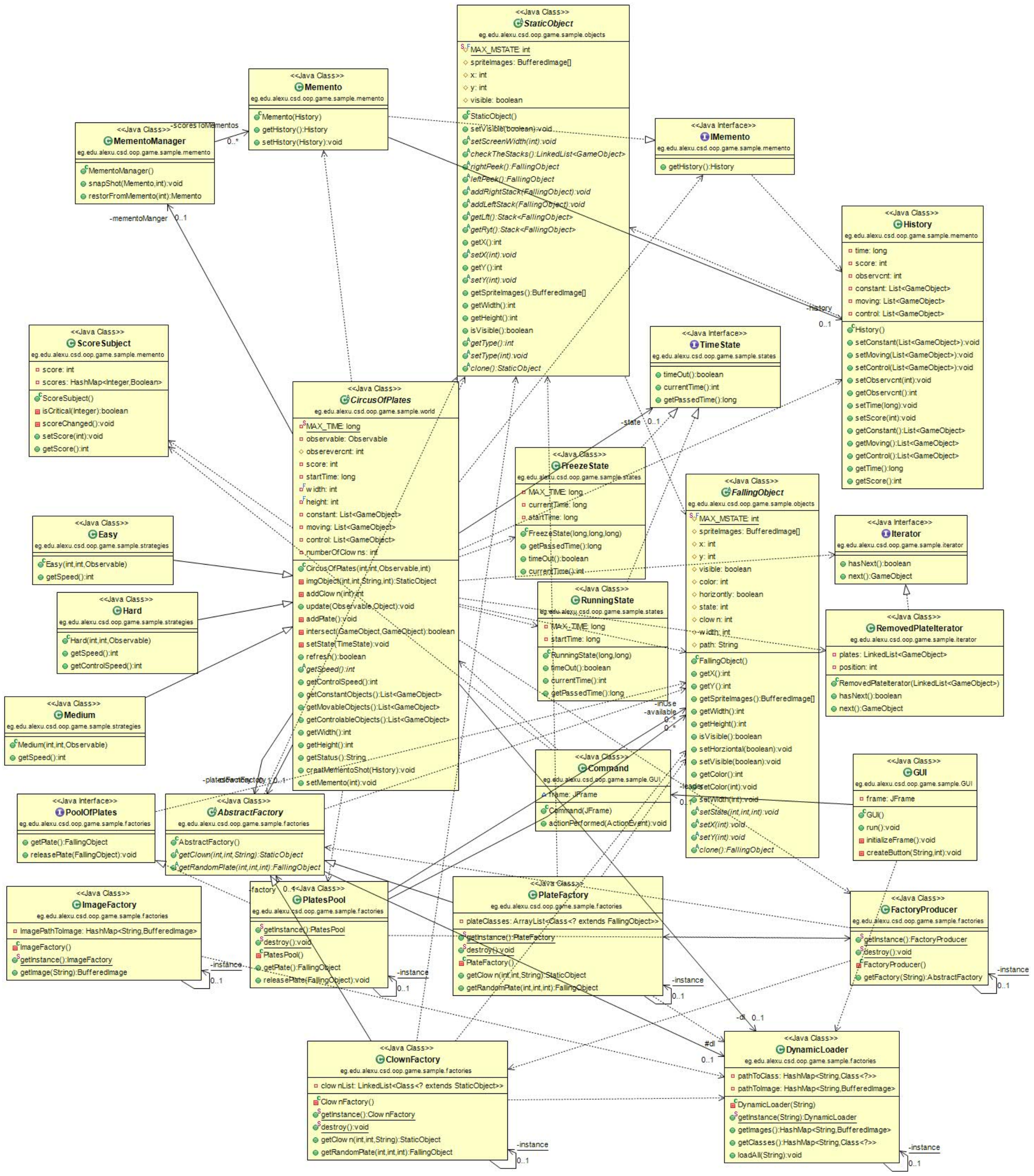
1 Problem Statement

Circus of Plates - Game It is single player-game in which each clown carry two stacks of plates, and there are a set of colored plates queues that fall down and he tries to catch them, if he manage to collect three consecutive plates of the same color, then they are vanished and his score increases. You are free to put rules ending the game.

2 Decisions and Assumptions

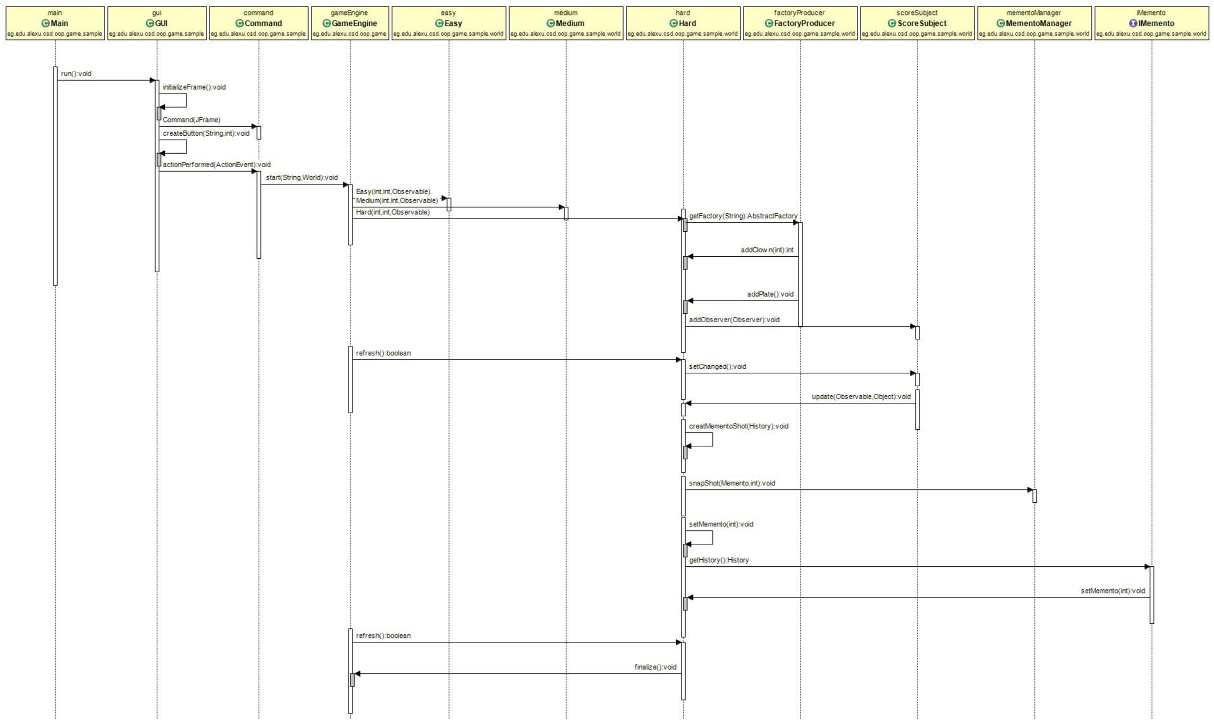
- There are three levels Easy is two clowns and low speed , medium two clowns high speed and hard one clown and high speed.
- The speed of the game will be increase inside the game if the user get scores 5,10,15,20
- If plates in the hand of clown full the screen the user have another chance from score the same state in score 5 again but exactly one chance
- Game over when time is finished or Plates full the screen again

3 Whole Class Diagram



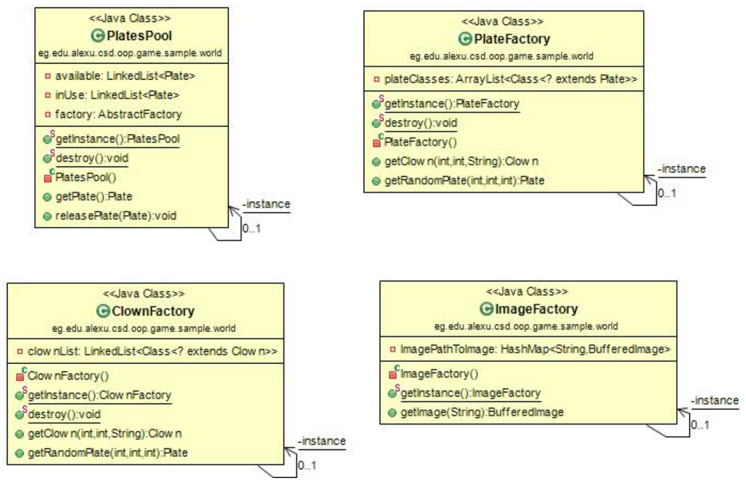
<<Java Class>>
StaticObject
eg.edu.alexu.csd.oop.game.sample.objects
MAX_MSTATE: int
spriteImages: BufferedImage[]
x: int
y: int
visible: boolean
StaticObject()
setVisible(boolean):void
setScreenWidth(int):void
checkTheStacks():LinkedList<GameObject>
rightPeek():FallingObject
leftPeek():FallingObject
addRightStack(FallingObject):void
addLeftStack(FallingObject):void
getLft():Stack<FallingObject>
getRyt():Stack<FallingObject>
getX():int
setX(int):void
getY():int
setY(int):void
getSpriteImages():BufferedImage[]
getWidth():int
getHeight():int
isVisible():boolean
getType():int
setType(int):void
clone():StaticObject
<<Java Class>>
Memento
eg.edu.alexu.csd.oop.game.sample.memento
Memento(History)
getHistory():History
setHistory(History):void
<<Java Class>>
MementoManager
eg.edu.alexu.csd.oop.game.sample.memento
MementoManager()
snapShot(Memento,int):void
restorFromMemento(int):Memento
-scoresToMementos
0..*
-mementoManger
0..1
<<Java Interface>>
IMemento
eg.edu.alexu.csd.oop.game.sample.memento
getHistory():History
<<Java Class>>
History
eg.edu.alexu.csd.oop.game.sample.memento
time: long
score: int
observcnt: int
constant: List<GameObject>
moving: List<GameObject>
control: List<GameObject>
History()
setConstant(List<GameObject>):void
setMoving(List<GameObject>):void
setControl(List<GameObject>):void
setObservcnt(int):void
getObservcnt():int
setTime(long):void
setScore(int):void
getConstant():List<GameObject>
getMoving():List<GameObject>
getControl():List<GameObject>
getTime():long
getScore():int
-history
0..1
<<Java Class>>
ScoreSubject
eg.edu.alexu.csd.oop.game.sample.memento
score: int
scores: HashMap<Integer,Boolean>
ScoreSubject()
isCritical(Integer):boolean
scoreChanged():void
setScore(int):void
getScore():int
<<Java Interface>>
TimeState
eg.edu.alexu.csd.oop.game.sample.states
timeOut():boolean
currentTime():int
getPassedTime():long
-state
0..1
<<Java Class>>
CircusOfPlates
eg.edu.alexu.csd.oop.game.sample.world
MAX_TIME: long
observable: Observable
obsereverent: int
score: int
startTime: long
width: int
height: int
constant: List<GameObject>
moving: List<GameObject>
control: List<GameObject>
numberOfClowns: int
CircusOfPlates(int,int,Observable,int)
imgObject(int,int,String,int):StaticObject
addClown(int,int)
update(Observable,Object):void
addPlate():void
intersect(GameObject,GameObject):boolean
setState(TimeState):void
refresh():boolean
getSpeed():int
getControlSpeed():int
getConstantObjects():List<GameObject>
getMovableObjects():List<GameObject>
getControlableObjects():List<GameObject>
getWidth():int
getHeight():int
getStatus():String
creatMementoShot(History):void
setMemento(int):void
<<Java Class>>
FreezeState
eg.edu.alexu.csd.oop.game.sample.states
MAX_TIME: long
currentTime: long
startTime: long
FreezeState(long,long,long)
getPassedTime():long
timeOut():boolean
currentTime():int
<<Java Class>>
FallingObject
eg.edu.alexu.csd.oop.game.sample.objects
MAX_MSTATE: int
spriteImages: BufferedImage[]
x: int
y: int
visible: boolean
color: int
horizontly: boolean
state: int
clown: int
width: int
path: String
FallingObject()
getX():int
getY():int
getSpriteImages():BufferedImage[]
getWidth():int
getHeight():int
isVisible():boolean
setHorziontal(boolean):void
setVisible(boolean):void
getColor():int
setColor(int):void
setWidth(int):void
setState(int,int,int):void
setX(int):void
setY(int):void
clone():FallingObject
<<Java Interface>>
Iterator
eg.edu.alexu.csd.oop.game.sample.iterator
hasNext():boolean
next():GameObject
<<Java Class>>
Easy
eg.edu.alexu.csd.oop.game.sample.strategies
Easy(int,int,Observable)
getSpeed():int
<<Java Class>>
Hard
eg.edu.alexu.csd.oop.game.sample.strategies
Hard(int,int,Observable)
getSpeed():int
getControlSpeed():int
<<Java Class>>
Medium
eg.edu.alexu.csd.oop.game.sample.strategies
Medium(int,int,Observable)
getSpeed():int
<<Java Class>>
RunningState
eg.edu.alexu.csd.oop.game.sample.states
MAX_TIME: long
startTime: long
RunningState(long,long)
timeOut():boolean
currentTime():int
getPassedTime():long
<<Java Class>>
RemovedPlateIterator
eg.edu.alexu.csd.oop.game.sample.iterator
plates: LinkedList<GameObject>
position: int
RemovedPlateIterator(LinkedList<GameObject>)
hasNext():boolean
next():GameObject
-inuse
-available
0..*
0..*
<<Java Class>>
Command
eg.edu.alexu.csd.oop.game.sample.GUI
frame: JFrame
Command(JFrame)
actionPerformed(ActionEvent):void
-leader
0..1
<<Java Class>>
GUI
eg.edu.alexu.csd.oop.game.sample.GUI
frame: JFrame
GUI()
run():void
initializeFrame():void
createButton(String,int):void
-platesFactory
-clownFactory
0..1
0..1
<<Java Interface>>
PoolOfPlates
eg.edu.alexu.csd.oop.game.sample.factories
getPlate():FallingObject
releasePlate(FallingObject):void
<<Java Class>>
AbstractFactory
eg.edu.alexu.csd.oop.game.sample.factories
AbstractFactory()
getClown(int,int,String):StaticObject
getRandomPlate(int,int,int):FallingObject
<<Java Class>>
ImageFactory
eg.edu.alexu.csd.oop.game.sample.factories
ImagePathToImage: HashMap<String,BufferedImage>
ImageFactory()
getInstance():ImageFactory
getImage(String):BufferedImage
-instance
0..1
-factory
0..1
<<Java Class>>
PlatesPool
eg.edu.alexu.csd.oop.game.sample.factories
getInstance():PlatesPool
destroy():void
PlatesPool()
getPlate():FallingObject
releasePlate(FallingObject):void
-instance
0..1
<<Java Class>>
PlateFactory
eg.edu.alexu.csd.oop.game.sample.factories
plateClasses: ArrayList<Class<? extends FallingObject>>
getInstance():PlateFactory
destroy():void
PlateFactory()
getClown(int,int,String):StaticObject
getRandomPlate(int,int,int):FallingObject
-instance
0..1
<<Java Class>>
FactoryProducer
eg.edu.alexu.csd.oop.game.sample.factories
getInstance():FactoryProducer
destroy():void
FactoryProducer()
getFactory(String):AbstractFactory
-instance
0..1
-dl
0..1
#dl
0..1
<<Java Class>>
ClownFactory
eg.edu.alexu.csd.oop.game.sample.factories
clownList: LinkedList<Class<? extends StaticObject>>
ClownFactory()
getInstance():ClownFactory
destroy():void
getClown(int,int,String):StaticObject
getRandomPlate(int,int,int):FallingObject
-instance
0..1
<<Java Class>>
DynamicLoader
eg.edu.alexu.csd.oop.game.sample.factories
pathToClass: HashMap<String,Class<?>>
pathToImage: HashMap<String,BufferedImage>
DynamicLoader(String)
getInstance(String):DynamicLoader
getImages():HashMap<String,BufferedImage>
getClasses():HashMap<String,Class<?>>
loadAll(String):void
-instance
0..1

4 Sequence diagram


main
Main
eg.edu.alexu.csd.oop.game.sample
gui
GUI
eg.edu.alexu.csd.oop.game.sample
command
Command
eg.edu.alexu.csd.oop.game.sample
gameEngine
GameEngine
eg.edu.alexu.csd.oop.game
easy
Easy
eg.edu.alexu.csd.oop.game.sample.world
medium
Medium
eg.edu.alexu.csd.oop.game.sample.world
hard
Hard
eg.edu.alexu.csd.oop.game.sample.world
factoryProducer
FactoryProducer
eg.edu.alexu.csd.oop.game.sample.world
scoreSubject
ScoreSubject
eg.edu.alexu.csd.oop.game.sample.world
mementoManager
MementoManager
eg.edu.alexu.csd.oop.game.sample.world
iMemento
IMemento
eg.edu.alexu.csd.oop.game.sample.world
run():void
initializeFrame():void
Command(JFrame)
createButton(String,int):void
actionPerformed(ActionEvent):void
start(String,World):void
Easy(int,int,Observable)
Medium(int,int,Observable)
Hard(int,int,Observable)
getFactory(String):AbstractFactory
addClown(int):int
addPlate():void
addObserver(Observer):void
refresh():boolean
setChanged():void
update(Observable,Object):void
creatMementoShot(History):void
snapShot(Memento,int):void
setMemento(int):void
getHistory():History
setMemento(int):void
refresh():boolean
finalize():void

5 Design Patterns

- Singleton
  - Class Diagram

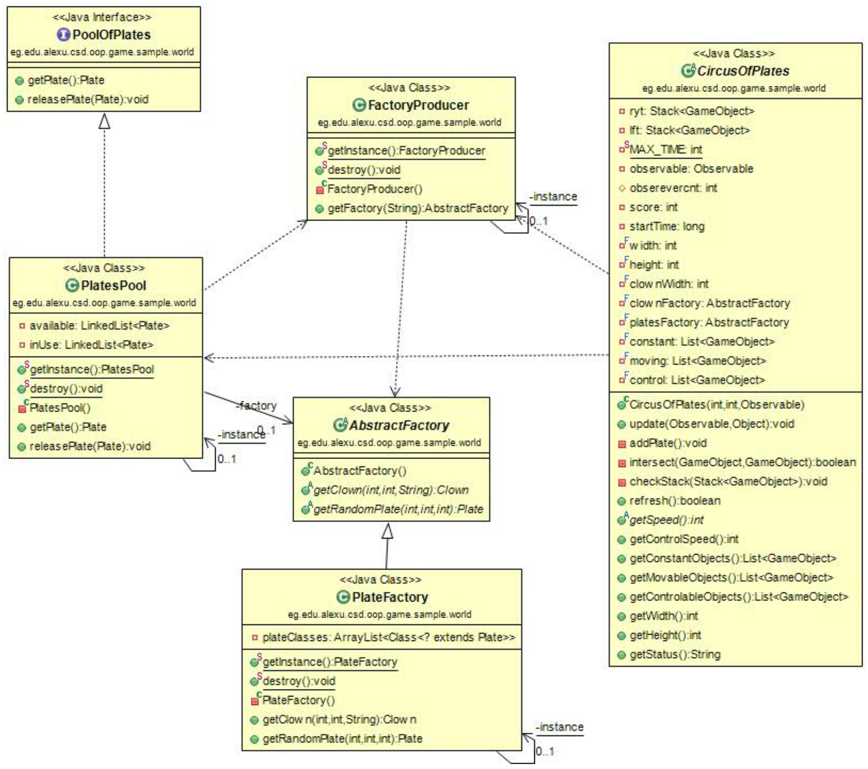

<<Java Class>>
PlatesPool
eg.edu.alexu.csd.oop.game.sample.world
available: LinkedList<Plate>
inUse: LinkedList<Plate>
factory: AbstractFactory
getInstance():PlatesPool
destroy():void
PlatesPool()
getPlate():Plate
releasePlate(Plate):void
-instance
0..1
<<Java Class>>
PlateFactory
eg.edu.alexu.csd.oop.game.sample.world
plateClasses: ArrayList<Class<? extends Plate>>
getInstance():PlateFactory
destroy():void
PlateFactory()
getClow n(int,int,String):Clow n
getRandomPlate(int,int,int):Plate
-instance
0..1
<<Java Class>>
ClownFactory
eg.edu.alexu.csd.oop.game.sample.world
clow nList: LinkedList<Class<? extends Clow n>>
Clow nFactory()
getInstance():Clow nFactory
destroy():void
getClow n(int,int,String):Clow n
getRandomPlate(int,int,int):Plate
-instance
0..1
<<Java Class>>
ImageFactory
eg.edu.alexu.csd.oop.game.sample.world
ImagePathToImage: HashMap<String,BufferedImage>
ImageFactory()
getInstance():ImageFactory
getImage(String):BufferedImage
-instance
0..1

- Usage

  To have a single Instance from those classes in the project there is no meaning of be more than one instance from them

- Pool

  - Class Diagram



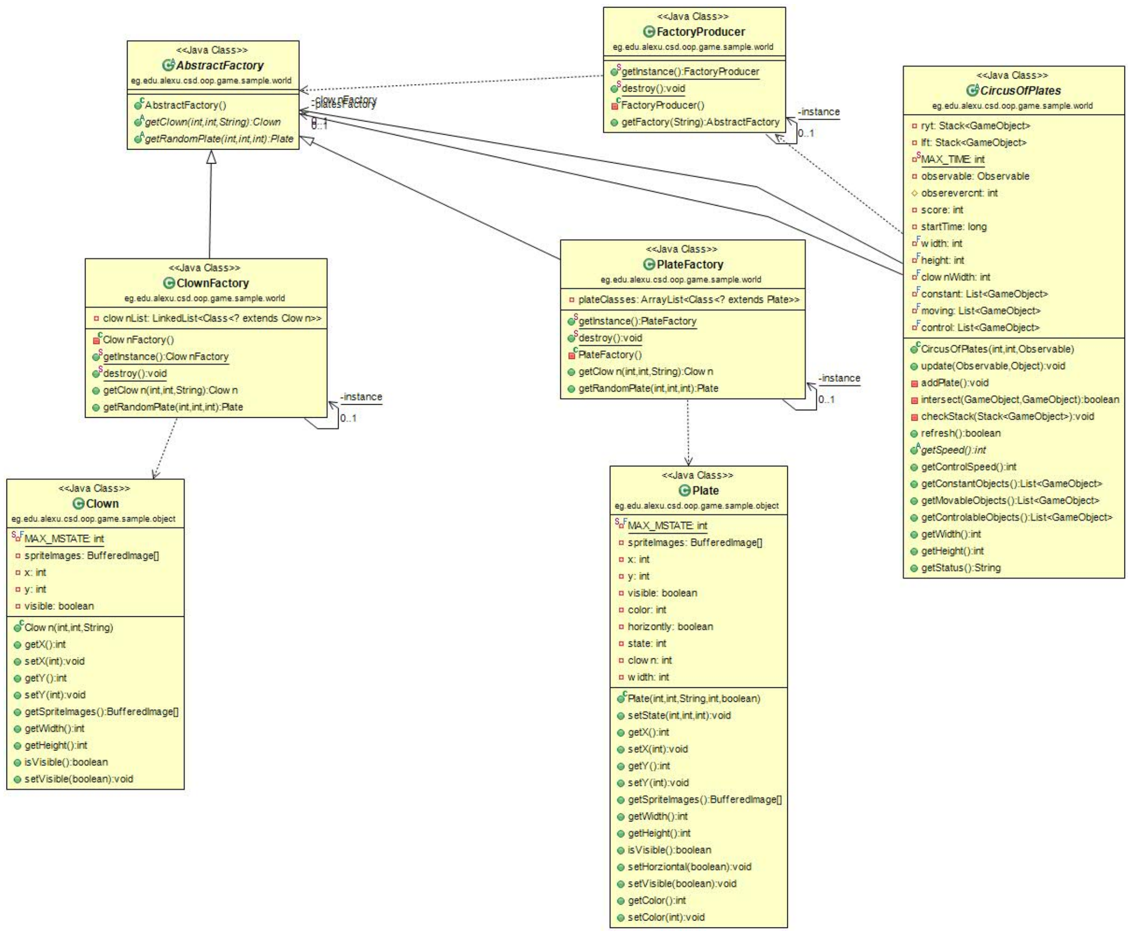
<<Java Interface>>
PoolOfPlates
eg.edu.alexu.csd.oop.game.sample.world
getPlate():Plate
releasePlate(Plate):void
<<Java Class>>
FactoryProducer
eg.edu.alexu.csd.oop.game.sample.world
getInstance():FactoryProducer
destroy():void
FactoryProducer()
getFactory(String):AbstractFactory
-instance
0..1
<<Java Class>>
CircusOfPlates
eg.edu.alexu.csd.oop.game.sample.world
ryt: Stack<GameObject>
lft: Stack<GameObject>
MAX_TIME: int
observable: Observable
obsereverent: int
score: int
startTime: long
width: int
height: int
clownWidth: int
clownFactory: AbstractFactory
platesFactory: AbstractFactory
constant: List<GameObject>
moving: List<GameObject>
control: List<GameObject>
CircusOfPlates(int,int,Observable)
update(Observable,Object):void
addPlate():void
intersect(GameObject,GameObject):boolean
checkStack(Stack<GameObject>):void
refresh():boolean
getSpeed():int
getControlSpeed():int
getConstantObjects():List<GameObject>
getMovableObjects():List<GameObject>
getControlableObjects():List<GameObject>
getWidth():int
getHeight():int
getStatus():String
<<Java Class>>
PlatesPool
eg.edu.alexu.csd.oop.game.sample.world
available: LinkedList<Plate>
inUse: LinkedList<Plate>
getInstance():PlatesPool
destroy():void
PlatesPool()
getPlate():Plate
releasePlate(Plate):void
-factory
0..1
-instance
0..1
<<Java Class>>
AbstractFactory
eg.edu.alexu.csd.oop.game.sample.world
AbstractFactory()
getClown(int,int,String):Clown
getRandomPlate(int,int,int):Plate
<<Java Class>>
PlateFactory
eg.edu.alexu.csd.oop.game.sample.world
plateClasses: ArrayList<Class<? extends Plate>>
getInstance():PlateFactory
destroy():void
PlateFactory()
getClown(int,int,String):Clown
getRandomPlate(int,int,int):Plate
-instance
0..1

- Usage

  It is costly to always generate new plate in the game to be used so we store some of them only in pool and if we need to use it get it from pool and if we finish from using it return it back to pool to reuse it.

- Abstract Factory

  - Class Diagram

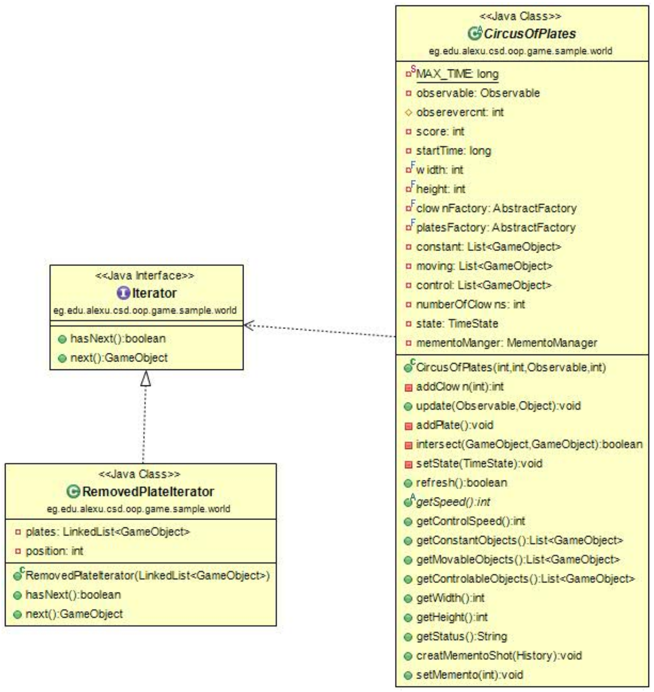

<<Java Class>>
AbstractFactory
eg.edu.alexu.csd.oop.game.sample.world
AbstractFactory()
getClown(int,int,String):Clown
getRandomPlate(int,int,int):Plate
<<Java Class>>
FactoryProducer
eg.edu.alexu.csd.oop.game.sample.world
getInstance():FactoryProducer
destroy():void
FactoryProducer()
getFactory(String):AbstractFactory
-clownFactory
-platesFactory
0..1
-instance
0..1
<<Java Class>>
CircusOfPlates
eg.edu.alexu.csd.oop.game.sample.world
ryt: Stack<GameObject>
lft: Stack<GameObject>
MAX_TIME: int
observable: Observable
obsereverCnt: int
score: int
startTime: long
width: int
height: int
clownWidth: int
constant: List<GameObject>
moving: List<GameObject>
control: List<GameObject>
CircusOfPlates(int,int,Observable)
update(Observable,Object):void
addPlate():void
intersect(GameObject,GameObject):boolean
checkStack(Stack<GameObject>):void
refresh():boolean
getSpeed():int
getControlSpeed():int
getConstantObjects():List<GameObject>
getMovableObjects():List<GameObject>
getControlableObjects():List<GameObject>
getWidth():int
getHeight():int
getStatus():String
<<Java Class>>
ClownFactory
eg.edu.alexu.csd.oop.game.sample.world
clownList: LinkedList<Class<? extends Clown>>
ClownFactory()
getInstance():ClownFactory
destroy():void
getClown(int,int,String):Clown
getRandomPlate(int,int,int):Plate
-instance
0..1
<<Java Class>>
PlateFactory
eg.edu.alexu.csd.oop.game.sample.world
plateClasses: ArrayList<Class<? extends Plate>>
getInstance():PlateFactory
destroy():void
PlateFactory()
getClown(int,int,String):Clown
getRandomPlate(int,int,int):Plate
-instance
0..1
<<Java Class>>
Clown
eg.edu.alexu.csd.oop.game.sample.object
MAX_MSTATE: int
spriteImages: BufferedImage[]
x: int
y: int
visible: boolean
Clown(int,int,String)
getX():int
setX(int):void
getY():int
setY(int):void
getSpriteImages():BufferedImage[]
getWidth():int
getHeight():int
isVisible():boolean
setVisible(boolean):void
<<Java Class>>
Plate
eg.edu.alexu.csd.oop.game.sample.object
MAX_MSTATE: int
spriteImages: BufferedImage[]
x: int
y: int
visible: boolean
color: int
horizontly: boolean
state: int
clown: int
width: int
Plate(int,int,String,int,boolean)
setState(int,int,int):void
getX():int
setX(int):void
getY():int
setY(int):void
getSpriteImages():BufferedImage[]
getWidth():int
getHeight():int
isVisible():boolean
setHorziontal(boolean):void
setVisible(boolean):void
getColor():int
setColor(int):void

- Usage

  There is some related objects that we may always change them at the running time so we must encapsulate it's creation in Class to make the other Classes independent from it's creation so we need factories create them dynamically at the running time

- Iterator

  - Class Diagram



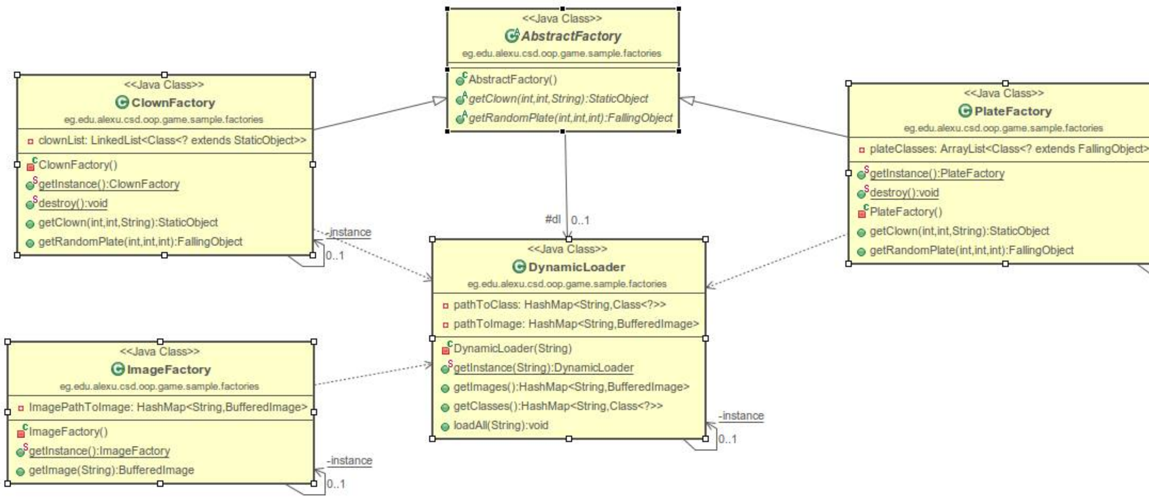
<<Java Class>>
CircusOfPlates
eg.edu.alexu.csd.oop.game.sample.world
MAX_TIME: long
observable: Observable
obserevercnt: int
score: int
startTime: long
width: int
height: int
clownFactory: AbstractFactory
platesFactory: AbstractFactory
constant: List<GameObject>
moving: List<GameObject>
control: List<GameObject>
numberOfClowns: int
state: TimeState
mementoManger: MementoManager
CircusOfPlates(int,int,Observable,int)
addClown(int):int
update(Observable,Object):void
addPlate():void
intersect(GameObject,GameObject):boolean
setState(TimeState):void
refresh():boolean
getSpeed():int
getControlSpeed():int
getConstantObjects():List<GameObject>
getMovableObjects():List<GameObject>
getControlableObjects():List<GameObject>
getWidth():int
getHeight():int
getStatus():String
creatMementoShot(History):void
setMemento(int):void
<<Java Interface>>
Iterator
eg.edu.alexu.csd.oop.game.sample.world
hasNext():boolean
next():GameObject
<<Java Class>>
RemovedPlateIterator
eg.edu.alexu.csd.oop.game.sample.world
plates: LinkedList<GameObject>
position: int
RemovedPlateIterator(LinkedList<GameObject>)
hasNext():boolean
next():GameObject

– Usage

We don't have faced the main problem of iterator design pattern that there is multiple classes implements an interface or extends an abstract class or share same property and we want an collection of them and it's implementation differ in every class (linked List,Array ,..) so we need to make one implementation for them all so we decide to implement it as a iterator on removed shapes from moving we collect them on Linked List and iterate over it.

- Dynamic Linkage

– Class Diagram



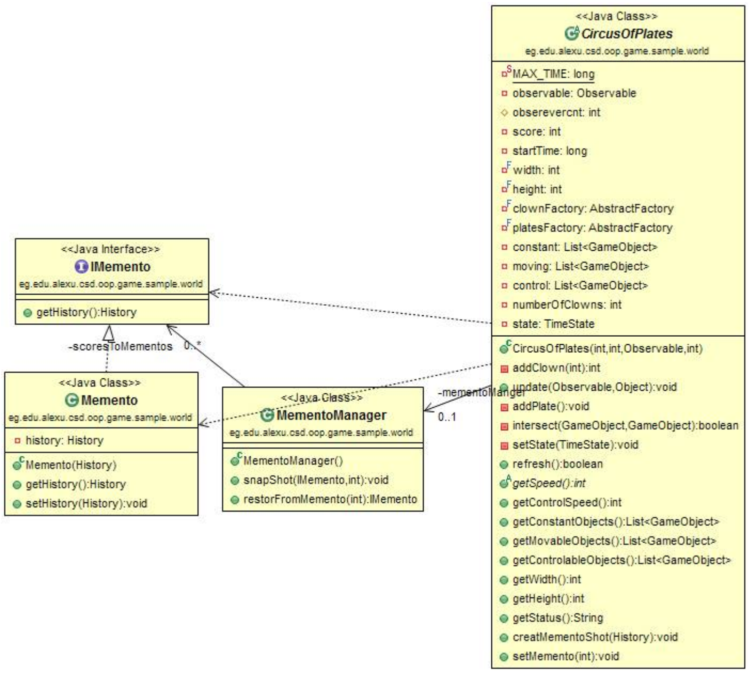
<<Java Class>>
AbstractFactory
eg.edu.alexu.csd.oop.game.sample.factories
AbstractFactory()
getClown(int,int,String):StaticObject
getRandomPlate(int,int,int):FallingObject
<<Java Class>>
ClownFactory
eg.edu.alexu.csd.oop.game.sample.factories
clownList: LinkedList<Class<? extends StaticObject>>
ClownFactory()
getInstance():ClownFactory
destroy():void
getClown(int,int,String):StaticObject
getRandomPlate(int,int,int):FallingObject
-instance
0..1
<<Java Class>>
PlateFactory
eg.edu.alexu.csd.oop.game.sample.factories
plateClasses: ArrayList<Class<? extends FallingObject>
getInstance():PlateFactory
destroy():void
PlateFactory()
getClown(int,int,String):StaticObject
getRandomPlate(int,int,int):FallingObject
#dl
0..1
<<Java Class>>
DynamicLoader
eg.edu.alexu.csd.oop.game.sample.factories
pathToClass: HashMap<String,Class<?>>
pathToImage: HashMap<String,BufferedImage>
DynamicLoader(String)
getInstance(String):DynamicLoader
getImages():HashMap<String,BufferedImage>
getClasses():HashMap<String,Class<?>>
loadAll(String):void
-instance
0..1
<<Java Class>>
ImageFactory
eg.edu.alexu.csd.oop.game.sample.factories
ImagePathToImage: HashMap<String,BufferedImage>
ImageFactory()
getInstance():ImageFactory
getImage(String):BufferedImage
-instance
0..1

– Usage

The Dynamic Linkage design patter is one of the most important design patterns, used to provide various edits and updates for an application without the need to recode it again .. And in this project we used it to load difference shapes implementation and the images used for each shape

- Snapshot

  – Class Diagram

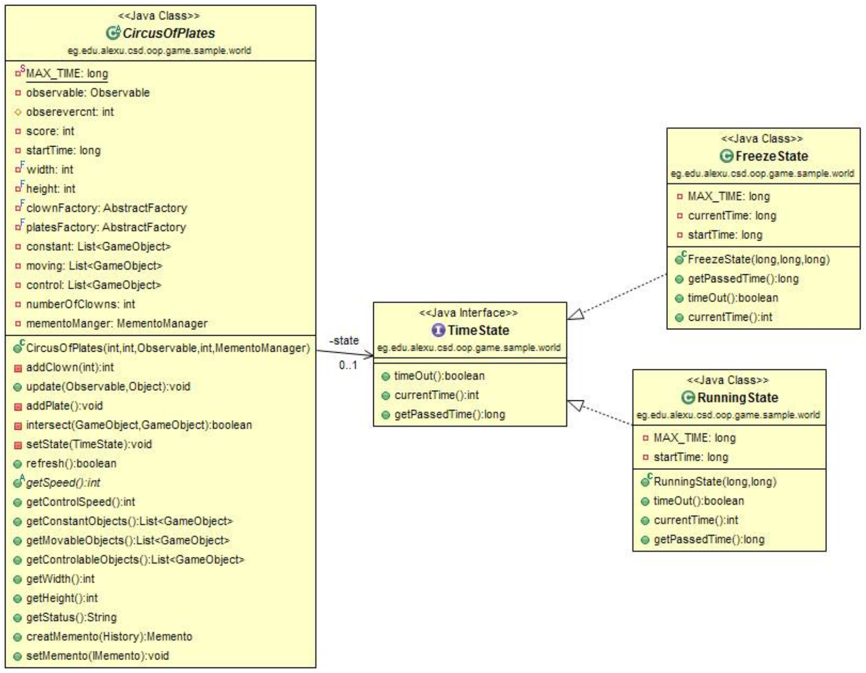


– Usage

1. SnapShot is a behavioral pattern
2. You need to create a snapshot of an objects state and also be able to restore the state of the object.
3. two methods in circuit of plates : creatMementoShot : takes history and creates a memento then send it to memento manager to be saved setMemento: which takes score as a parameter then get from memento manager a memento and set it to the game.
4. there is an interface for memento.
5. memento has only one field which is history .
6. memento manager has a hashmap to save mementos and hash them by their scores then can return therm by a score as a parameter it react with CircusOfPlates Object.
7. You want a mechanism that saves and restores an objects state to be independent of the objects internal structure,
8. the internal structure can change without having to modify the save/restore mechanism.

- State

– Class Diagram

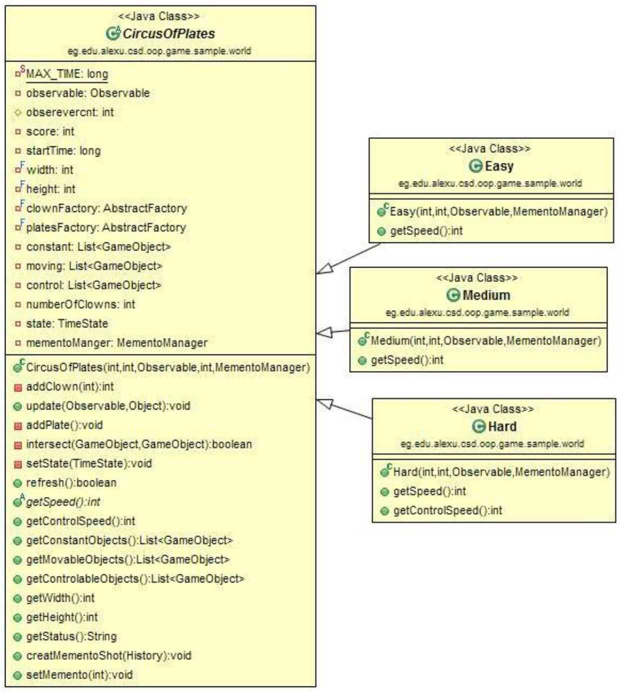


– Usage

We have Two state first state when game is running normally and if Clown manage to take star Object we convert to state of freezing state which freeze the Time for 5 seconds and the return to the Running state

- Strategy

– Class Diagram

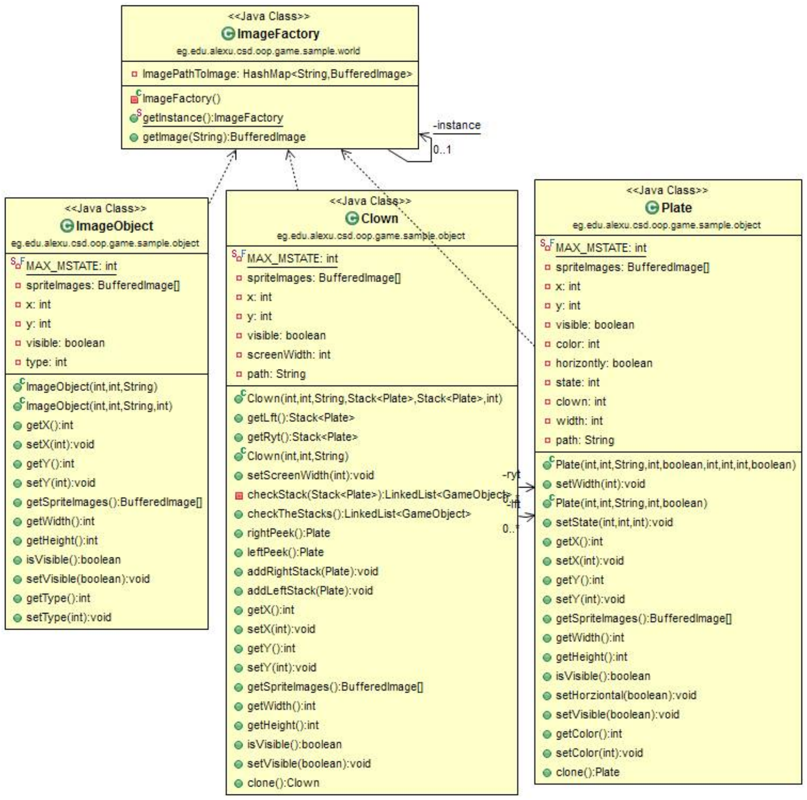

<<Java Class>>
CircusOfPlates
eg.edu.alexu.csd.oop.game.sample.world
MAX_TIME: long
observable: Observable
obserevercnt: int
score: int
startTime: long
width: int
height: int
clownFactory: AbstractFactory
platesFactory: AbstractFactory
constant: List<GameObject>
moving: List<GameObject>
control: List<GameObject>
numberOfClowns: int
state: TimeState
mementoManger: MementoManager
CircusOfPlates(int,int,Observable,int,MementoManager)
addClown(int):int
update(Observable,Object):void
addPlate():void
intersect(GameObject,GameObject):boolean
setState(TimeState):void
refresh():boolean
getSpeed():int
getControlSpeed():int
getConstantObjects():List<GameObject>
getMovableObjects():List<GameObject>
getControlableObjects():List<GameObject>
getWidth():int
getHeight():int
getStatus():String
creatMementoShot(History):void
setMemento(int):void
<<Java Class>>
Easy
eg.edu.alexu.csd.oop.game.sample.world
Easy(int,int,Observable,MementoManager)
getSpeed():int
<<Java Class>>
Medium
eg.edu.alexu.csd.oop.game.sample.world
Medium(int,int,Observable,MementoManager)
getSpeed():int
<<Java Class>>
Hard
eg.edu.alexu.csd.oop.game.sample.world
Hard(int,int,Observable,MementoManager)
getSpeed():int
getControlSpeed():int

- Usage

  There is three Classes do the same algorithm to run the game have the same purpose but each one have it's way to do that Easy , Medium and Hard are strategies for Circus Of Plates Class.

- Flyweight

  - Class Diagram

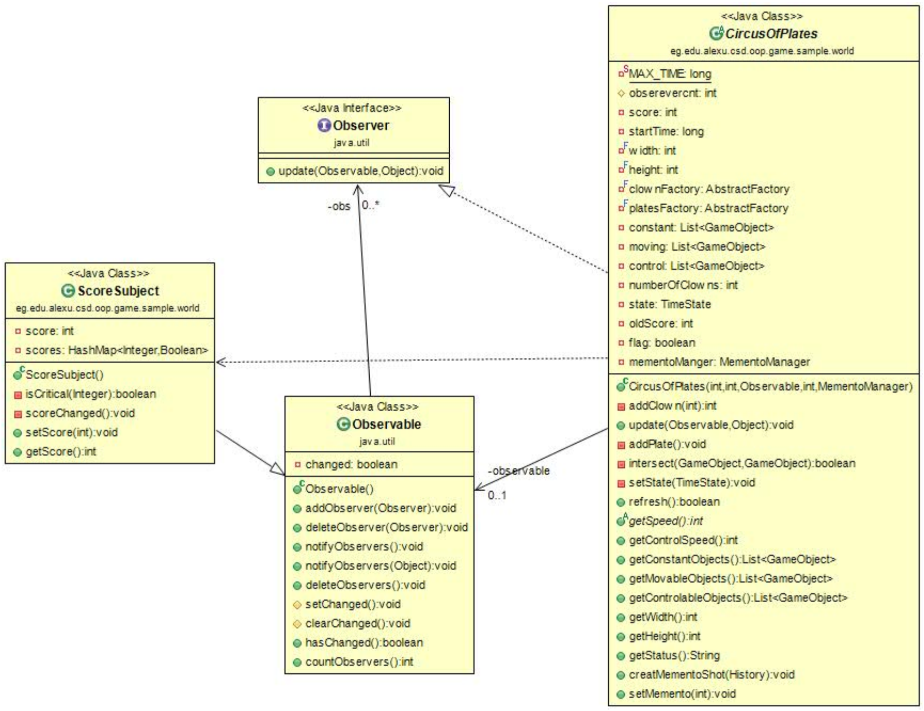


- Usage

  1. we have an application that uses a similar objects
  2. we want to reduce the memory overhead of having a large number of similar objects so we used flyweight
  3. fly weight is used in loading images which is created in clown ,imageObject and plate
  4. so images is loaded only one time as we shared images for this three objects : clown , imageObject and plate.

- Observer

  - Class Diagram

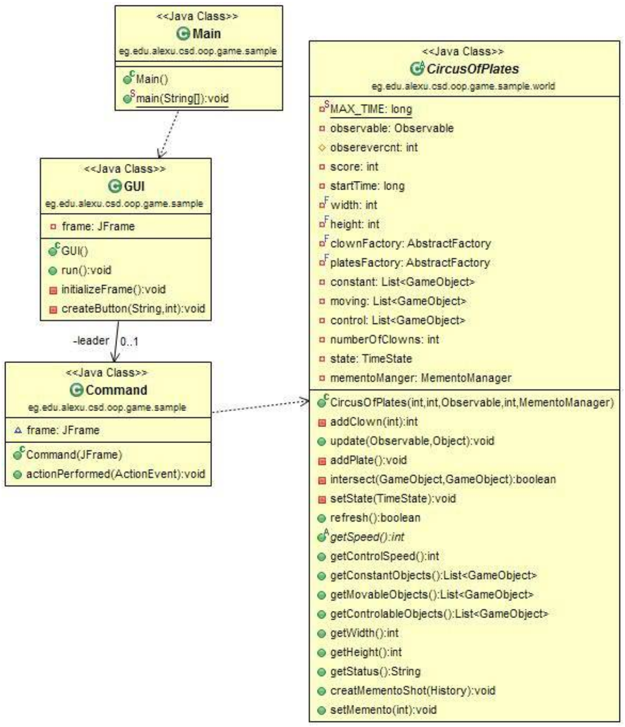


- Usage

  We make a Observable class on score which observe score if it is 5,10,15 or 20 it's notify there observers which here is Circus of plates to change the speed of the game .

- Command

  - Class Diagram


<<Java Class>>
Main
eg.edu.alexu.csd.oop.game.sample
Main()
main(String[]):void
<<Java Class>>
GUI
eg.edu.alexu.csd.oop.game.sample
frame: JFrame
GUI()
run():void
initializeFrame():void
createButton(String,int):void
-leader
0..1
<<Java Class>>
Command
eg.edu.alexu.csd.oop.game.sample
frame: JFrame
Command(JFrame)
actionPerformed(ActionEvent):void
<<Java Class>>
CircusOfPlates
eg.edu.alexu.csd.oop.game.sample.world
MAX_TIME: long
observable: Observable
obserevercnt: int
score: int
startTime: long
width: int
height: int
clownFactory: AbstractFactory
platesFactory: AbstractFactory
constant: List<GameObject>
moving: List<GameObject>
control: List<GameObject>
numberOfClowns: int
state: TimeState
mementoManger: MementoManager
CircusOfPlates(int,int,Observable,int,MementoManager)
addClown(int):int
update(Observable,Object):void
addPlate():void
intersect(GameObject,GameObject):boolean
setState(TimeState):void
refresh():boolean
getSpeed():int
getControlSpeed():int
getConstantObjects():List<GameObject>
getMovableObjects():List<GameObject>
getControlableObjects():List<GameObject>
getWidth():int
getHeight():int
getStatus():String
creatMementoShot(History):void
setMemento(int):void

– Usage

Instead of making an Action listener to every button in GUI to choose which level user want w make a command design pattern which implements an interface Action listener to be one implementation to any Button