CSED 21

Alexandria University

Faculty of Engineering

Object Oriented Programming – Assignment 5

Circus of Plates

The Interview Game

By the Developers:

Muhammad Salah Mahmoud Osman (41)

Mustafa Nabil Muhammad Saied (51)

Muhammad Mustafa Ahmed Hassan AL-Nashar (44)

Tarek Muhammad Kamal EL-Deen (21)

Reham Muhammad Naguib Metwally (17)

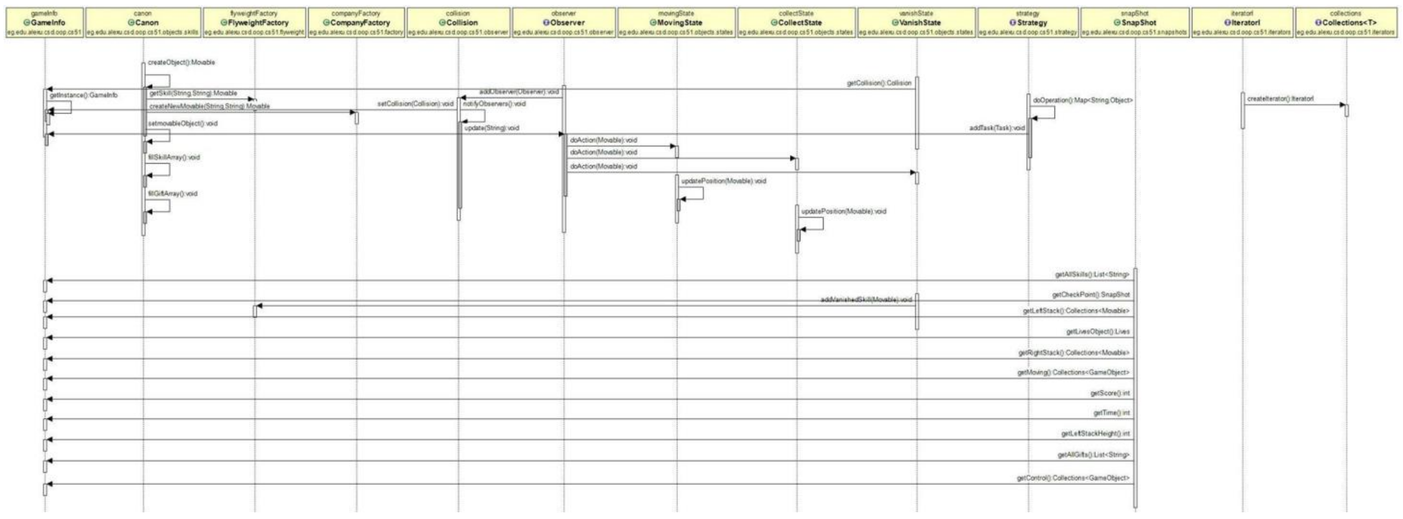
Description:

The Interview is a single player 2D computer game that describes the misery of a computer science student in his journey to get accepted in a company for an internship or a job. The game starts with the interviewee in front of three interviewers where he's trying to collect the skills required to get accepted in the companies required by the level. In order for the interviewee to complete the task required by the company he must collect the three required skills in sequence. While the interviewee is collected the required skills he may collect undesired skills in this case he must empty his stack by collect any three skills from the same company which will vanish to empty the stack. The interviewee has two stacks to help him complete his tasks. The player wins when he complete all his tasks.

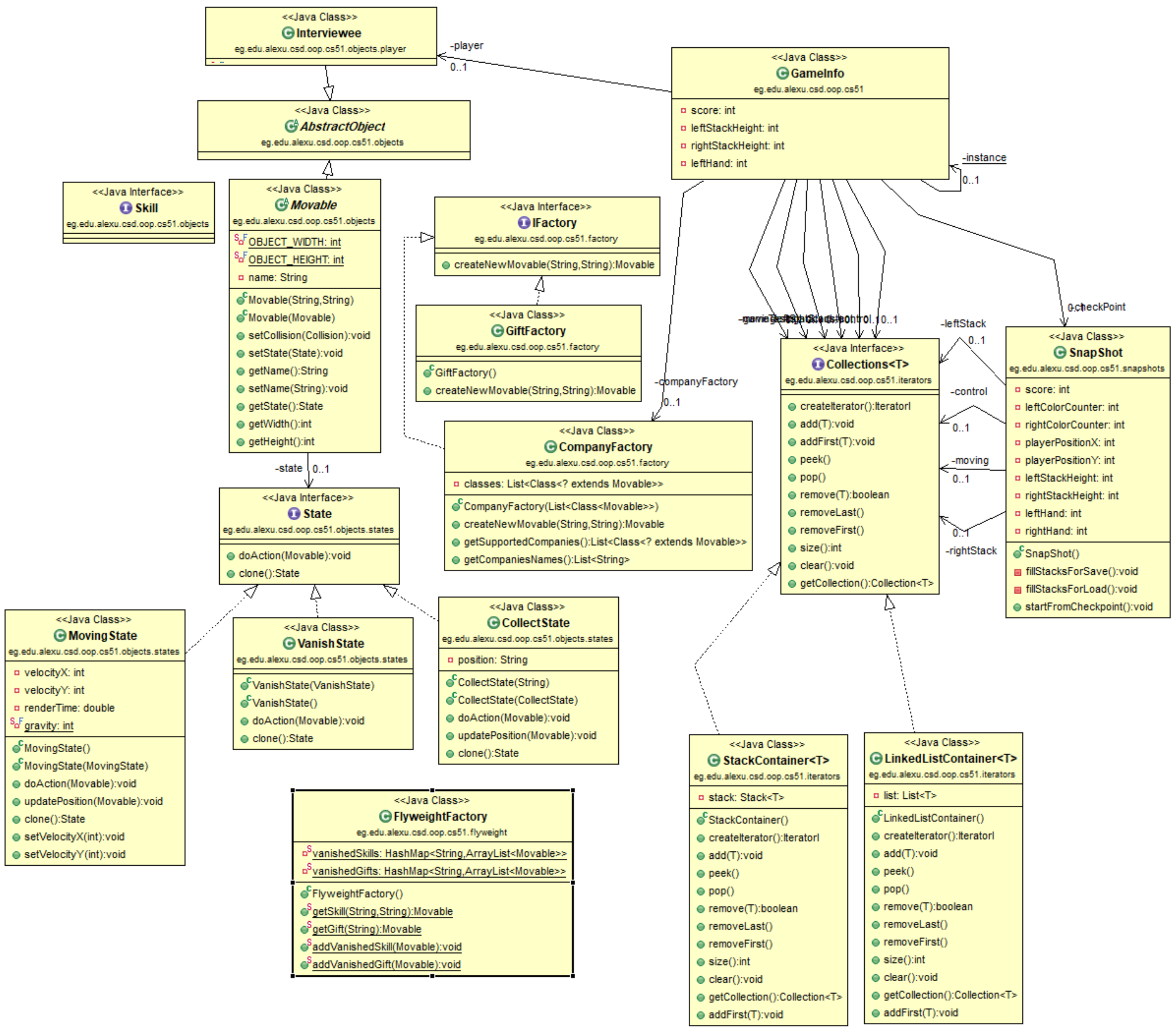
Design Description:

In order to build a robust design for the game. We used MVC structure to organize our project with the skills used in the object are dynamically loaded to allow the future adding of more different tasks. We also used some math algorithms to determine the position of the falling skills to form a second order function.

Sequence Diagram:

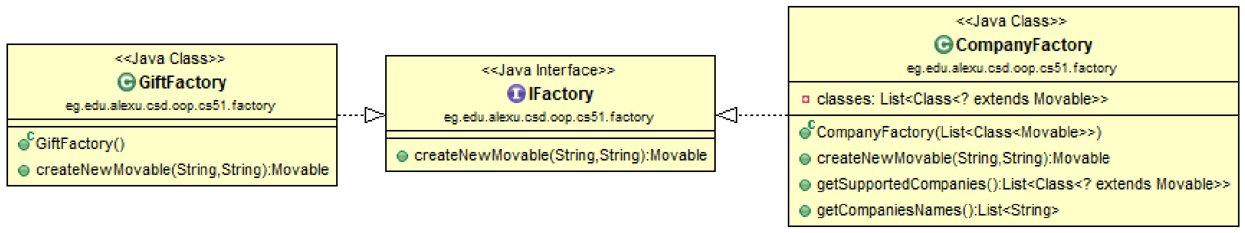

gameInfo
GameInfo
eg.edu.alexu.csd.oop.cs51
canon
Canon
eg.edu.alexu.csd.oop.cs51.objects.skills
flyweightFactory
FlyweightFactory
eg.edu.alexu.csd.oop.cs51.flyweight
companyFactory
CompanyFactory
eg.edu.alexu.csd.oop.cs51.factory
collision
Collision
eg.edu.alexu.csd.oop.cs51.observer
observer
Observer
eg.edu.alexu.csd.oop.cs51.observer
movingState
MovingState
eg.edu.alexu.csd.oop.cs51.objects.states
collectState
CollectState
eg.edu.alexu.csd.oop.cs51.objects.states
vanishState
VanishState
eg.edu.alexu.csd.oop.cs51.objects.states
strategy
Strategy
eg.edu.alexu.csd.oop.cs51.strategy
snapShot
SnapShot
eg.edu.alexu.csd.oop.cs51.snapshots
iteratorI
IteratorI
eg.edu.alexu.csd.oop.cs51.iterators
collections
Collections<T>
eg.edu.alexu.csd.oop.cs51.iterators
createObject():Movable
getInstance():GameInfo
getSkill(String,String):Movable
createNewMovable(String,String):Movable
setmovableObject():void
fillSkillArray():void
fillGiftArray():void
addObserver(Observer):void
setCollision(Collision):void
notifyObservers():void
update(String):void
doAction(Movable):void
doAction(Movable):void
doAction(Movable):void
updatePosition(Movable):void
updatePosition(Movable):void
getCollision():Collision
doOperation():Map<String,Object>
addTask(Task):void
createIterator():IteratorI
getAllSkills():List<String>
getCheckPoint():SnapShot
addVanishedSkill(Movable):void
getLeftStack():Collections<Movable>
getLivesObject():Lives
getRightStack():Collections<Movable>
getMoving():Collections<GameObject>
getScore():int
getTime():int
getLeftStackHeight():int
getAllGifts():List<String>
getControl():Collections<GameObject>

Class Diagram:

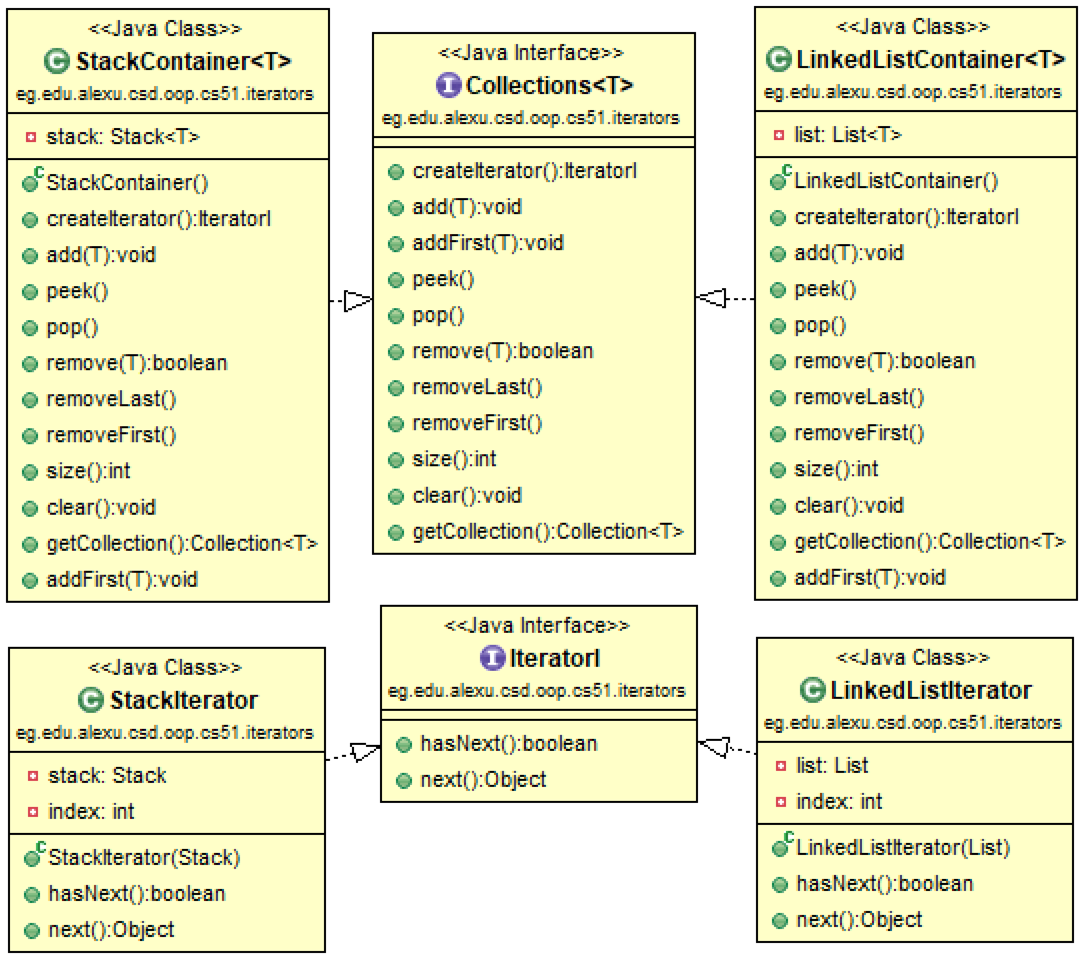


Used Design Patterns:

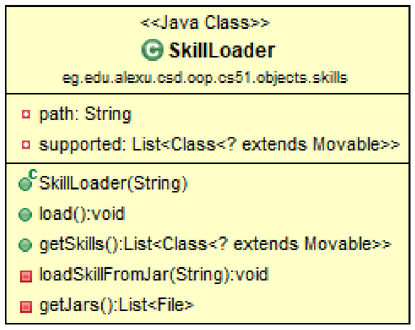
- **Singleton:** We used the singleton DP to Hold all the game information and the data that can't exist more than once, like the player data and the score. The GameInfo Class which can be found in the above class diagram is our singleton class.
- **Factory:** the factory DP is used in our design to instantiate new Skills to the canon which fires the skills, the factory has a list of the dynamically loaded classes, we also used the factory design pattern for the gifts that will be thrown during the game.


<<Java Class>>
GiftFactory
eg.edu.alexu.csd.oop.cs51.factory
GiftFactory()
createNewMovable(String,String):Movable
<<Java Interface>>
IFactory
eg.edu.alexu.csd.oop.cs51.factory
createNewMovable(String,String):Movable
<<Java Class>>
CompanyFactory
eg.edu.alexu.csd.oop.cs51.factory
classes: List<Class<? extends Movable>>
CompanyFactory(List<Class<Movable>>)
createNewMovable(String,String):Movable
getSupportedCompanies():List<Class<? extends Movable>>
getCompaniesNames():List<String>

- **Iterator:** we used the iterator design patterns for most of our collections (Stacks and Lists) so that we can easily iterate throw any type of collection.

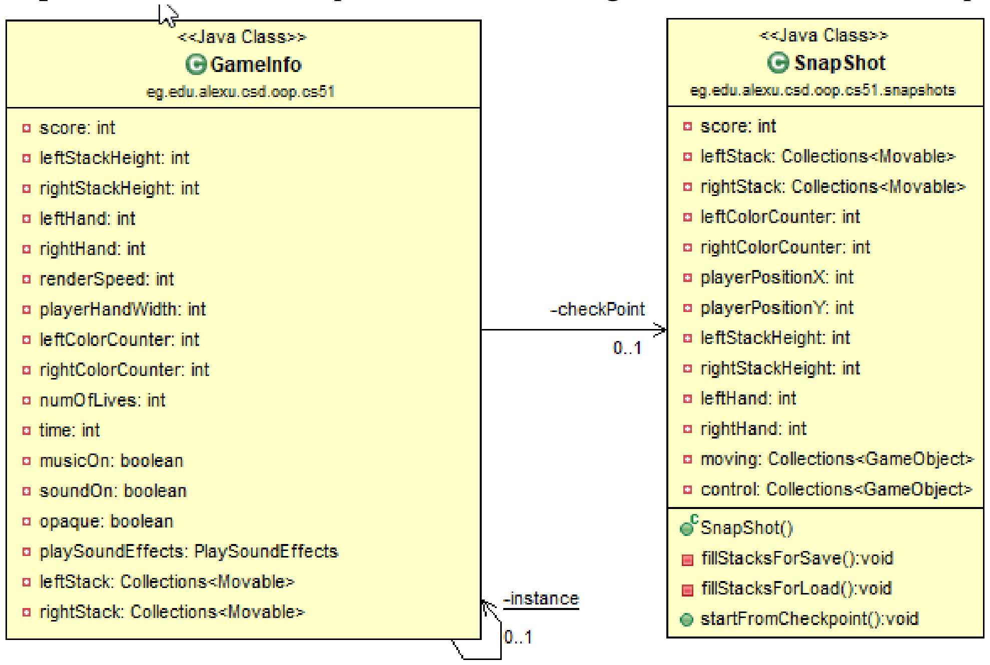


- **Dynamic Linkage:** it's the DP responsible for loading the skills' classes from the jar dynamically. It's a single class that loads the skills as classes and the images within the jars.

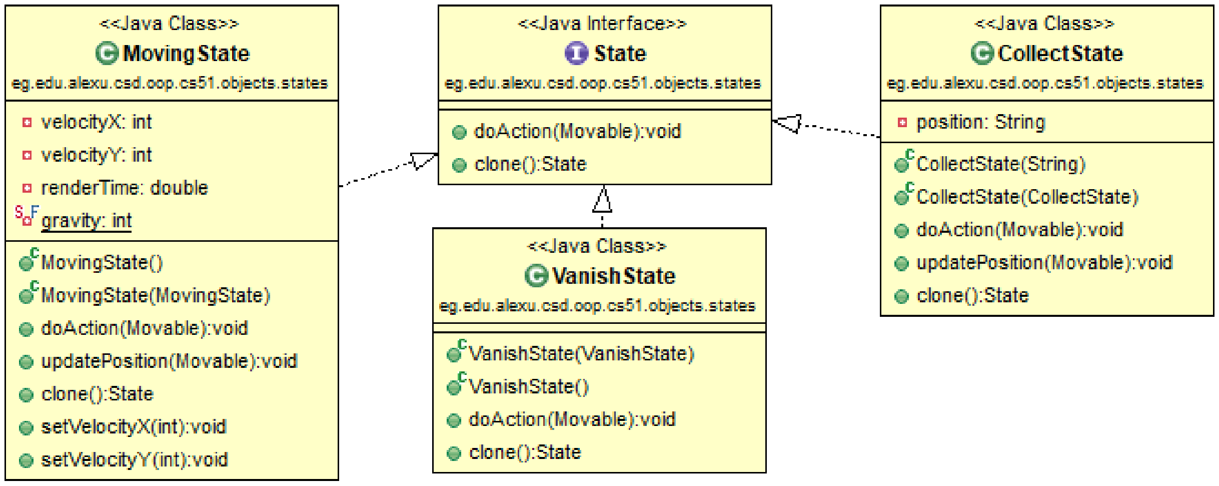


<<Java Class>>
SkillLoader
eg.edu.alexu.csd.oop.cs51.objects.skills
path: String
supported: List<Class<? extends Movable>>
SkillLoader(String)
load():void
getSkills():List<Class<? extends Movable>>
loadSkillFromJar(String):void
getJars():List<File>

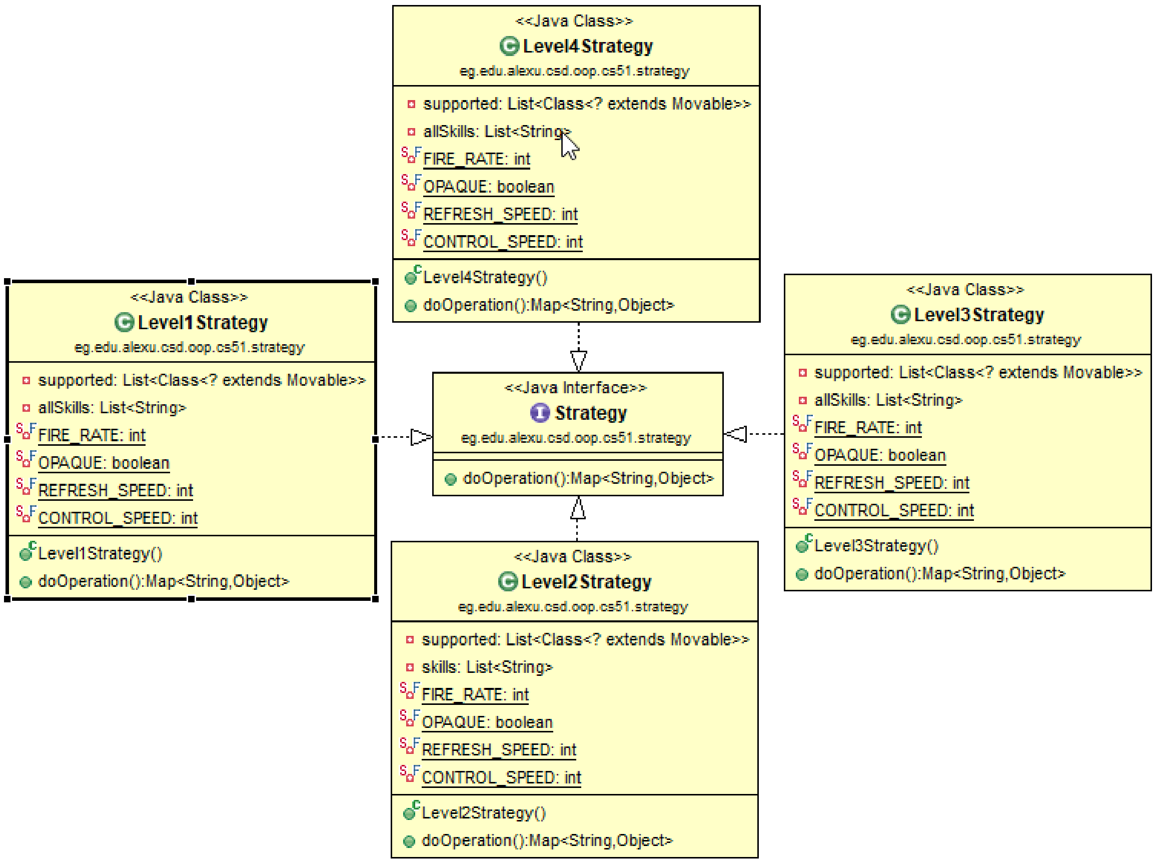
- **Snapshot:** we used the snapshot DP to save the game state to create a checkpoint.



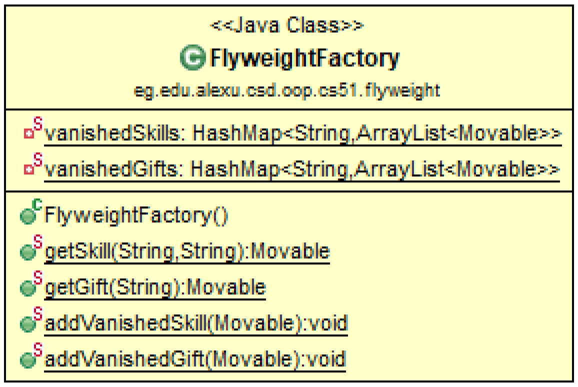
- **State:** the state of the Skills and gifts, each has three states, the moving state which holds the speed and direction of the moving object, the collect state which informs that the skill is collected I the stack and finally the vanish state which the moving object will eventually reach.



- **Strategy:** used to determine the difficulty of each level.

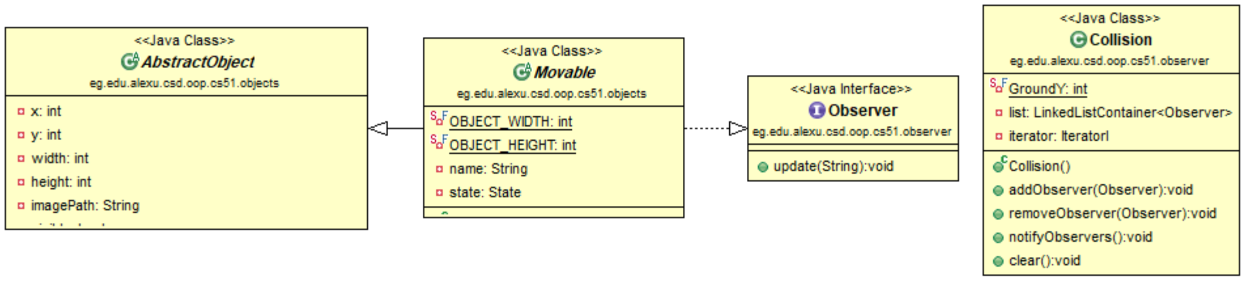


- **Flyweight:** it's used to handle and control the creation of the skills so we don't create more non used objects it collects the vanished skills and reuse them.



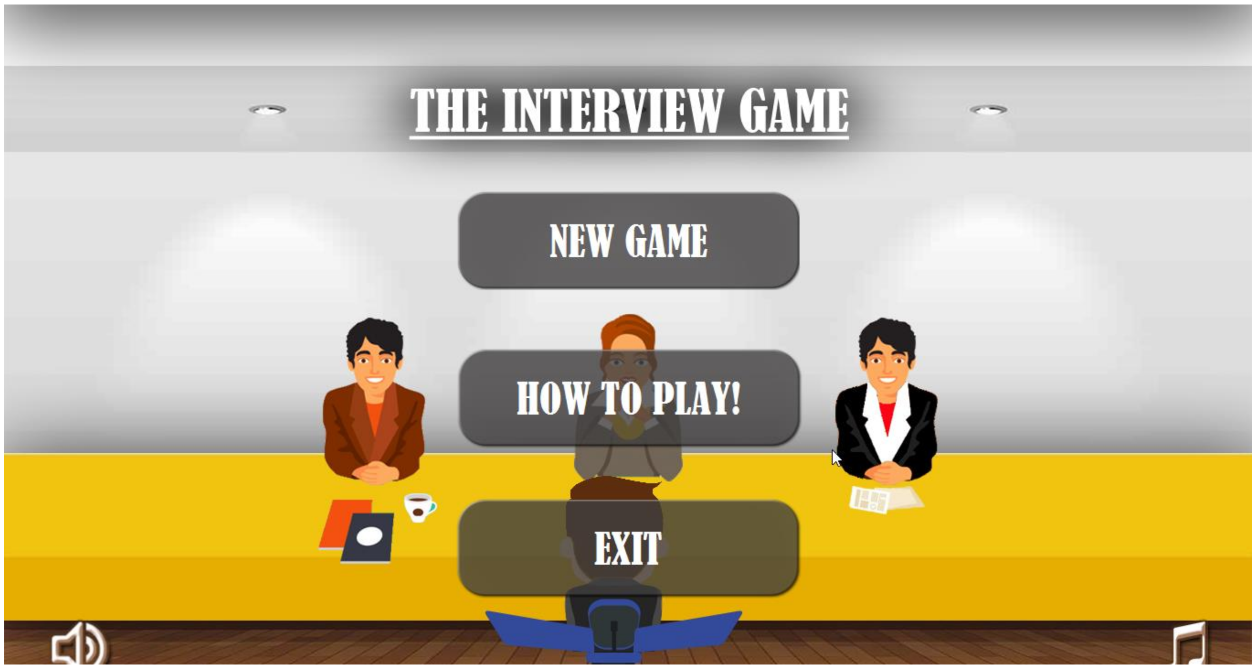
<<Java Class>>
FlyweightFactory
eg.edu.alexu.csd.oop.cs51.flyweight
vanishedSkills: HashMap<String,ArrayList<Movable>>
vanishedGifts: HashMap<String,ArrayList<Movable>>
FlyweightFactory()
getSkill(String,String):Movable
getGift(String):Movable
addVanishedSkill(Movable):void
addVanishedGift(Movable):void

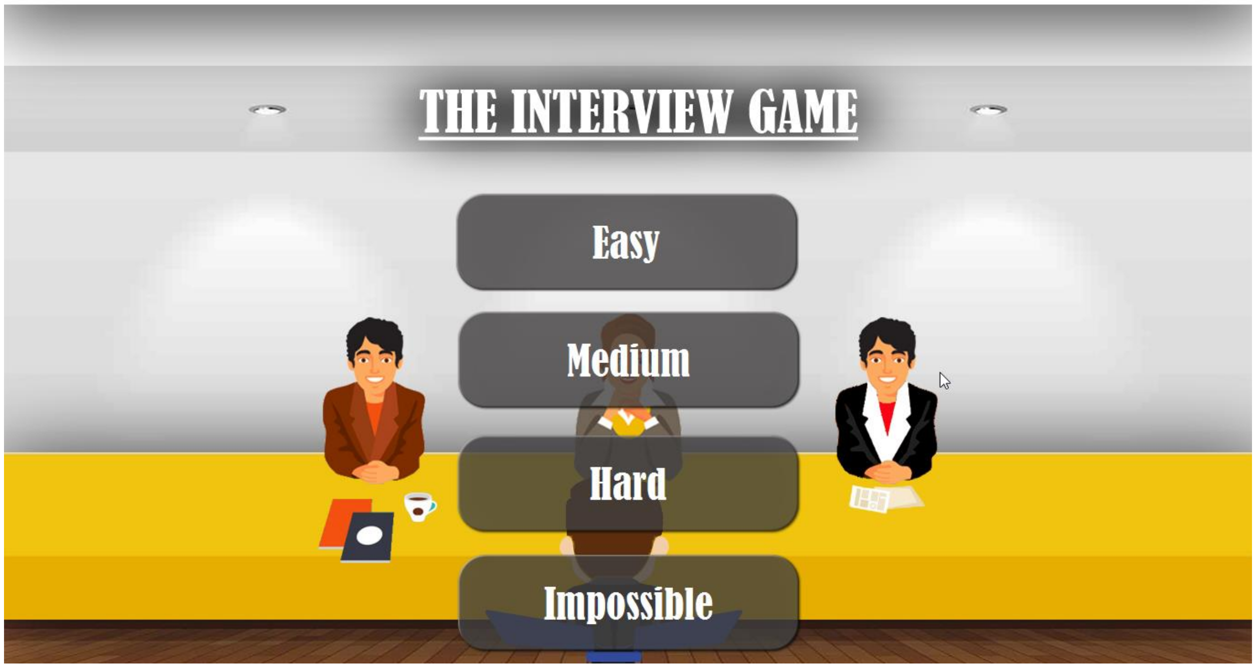
- **Observer:** we created a collision class that handles the collision of the skills and gifts so the movable object implements the observer interface and the collision object updates the observers in every render to detect collision and change the state of the skill or the gift.

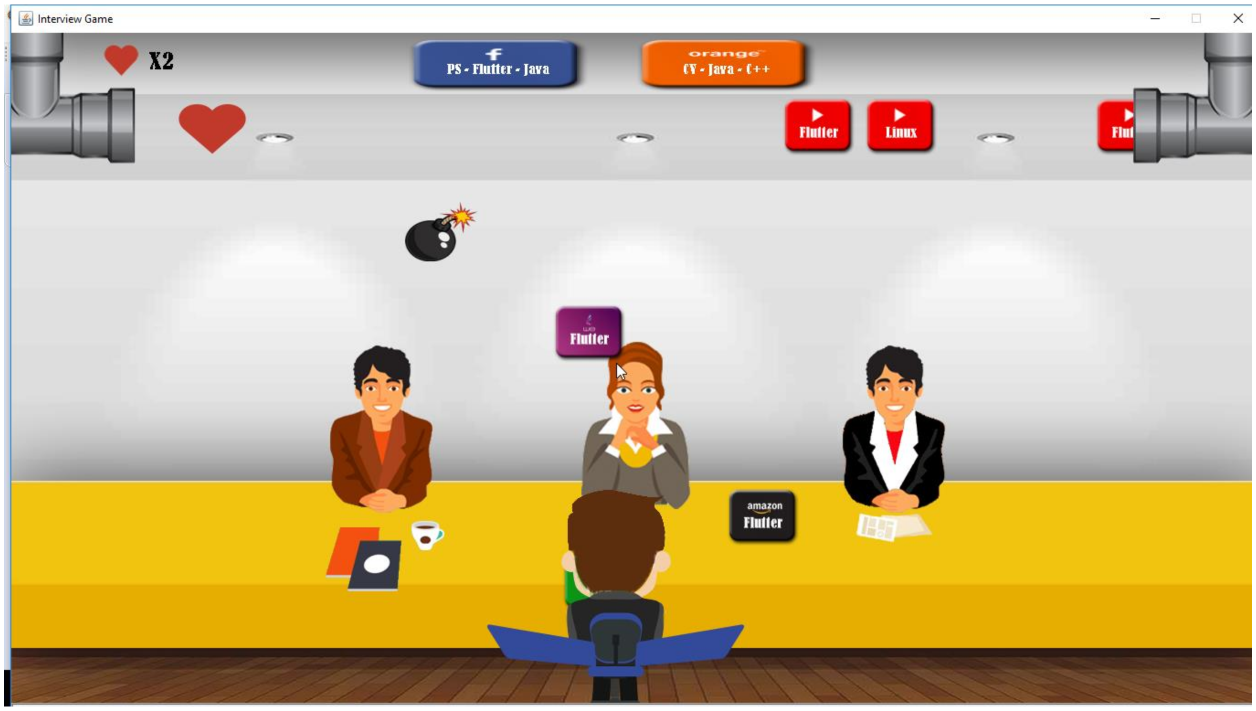


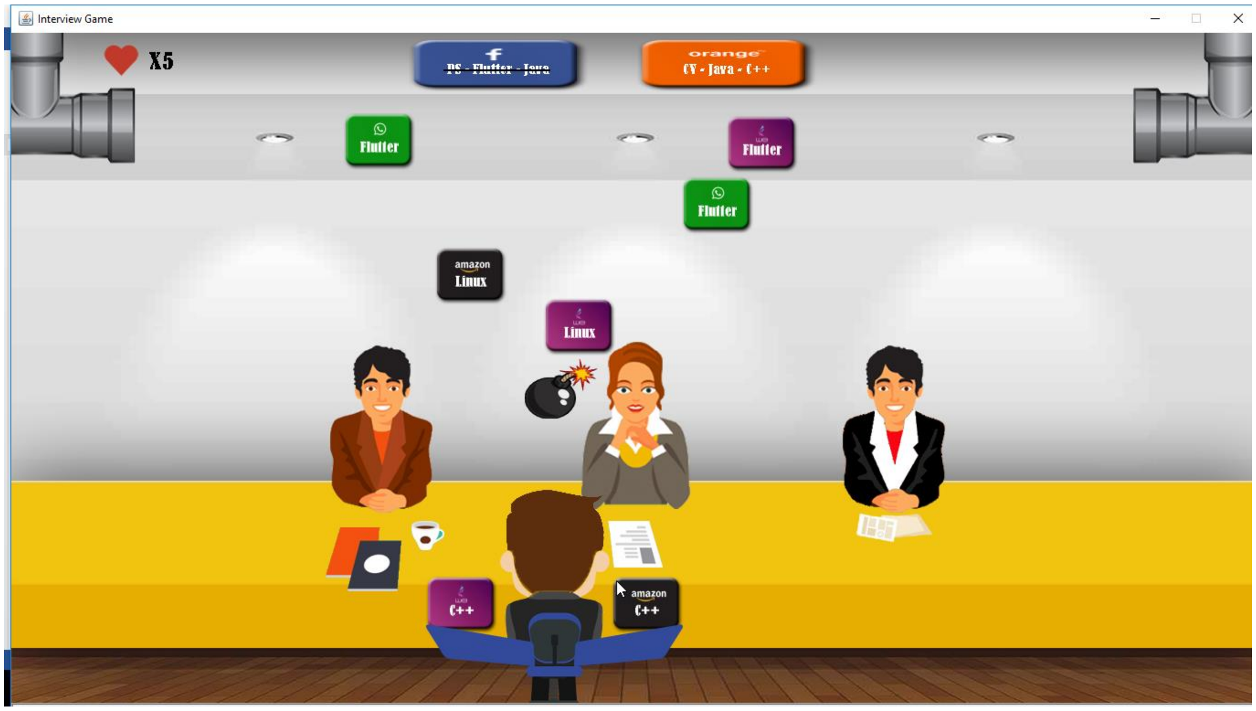
- **Marker:** We used the marker Interface for the dynamic loading and to differentiate between the gifts and the skills as both implements the Movable abstract class.

Screenshots:






Interview Game
X2
PS - Flutter - Java
CV - Java - C++
Flutter
Linux
Flutter
Flutter


Interview Game
X5
~~PS - Flutter - Java~~
CV - Java - C++
Flutter
Flutter
Flutter
Linux
Linux
C++
C++