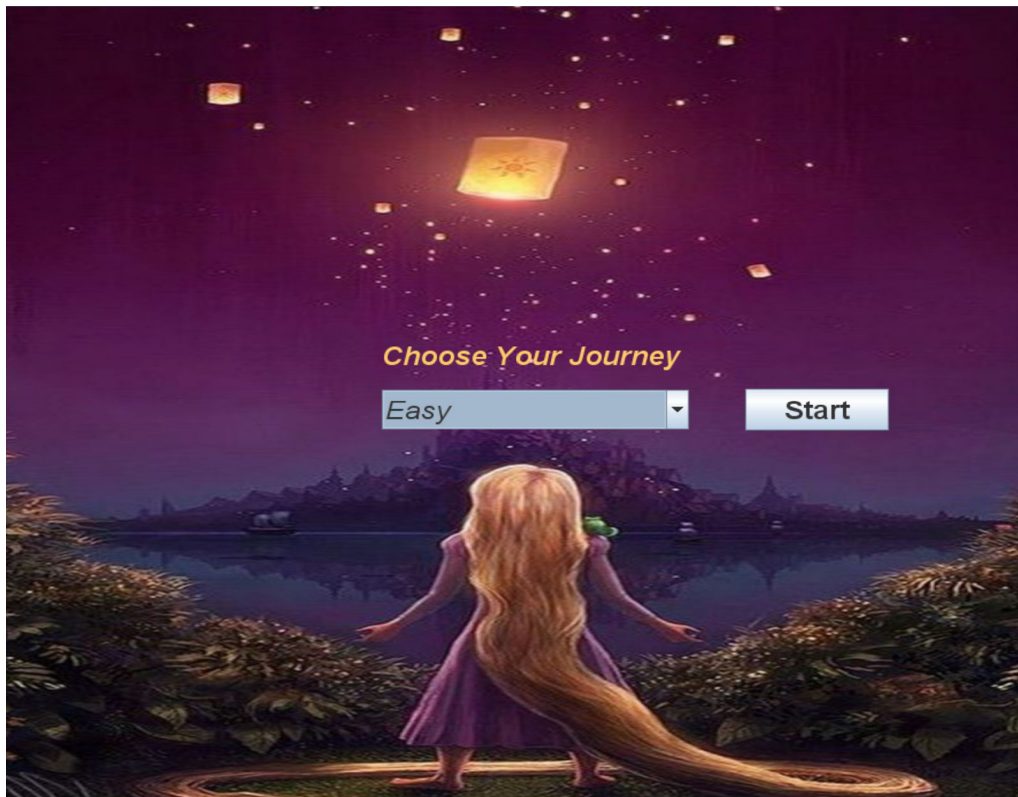**Alexandria University**
Faculty of Engineering
Computer and Systems Engineering Dept. CS:
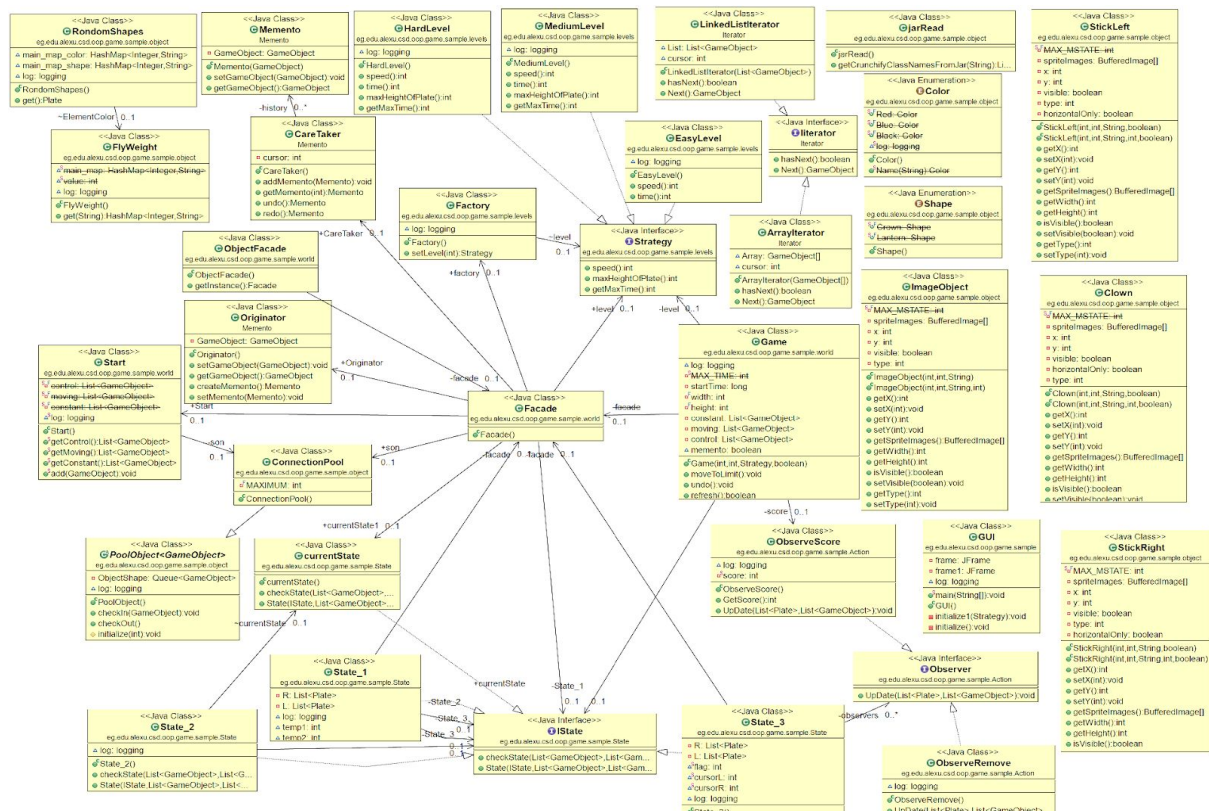**PROGRAMMING 2**

Made By :

1. Aya Gamal (1)
2. Afnan Mousa (15)
3. Shimaa kamal(34)
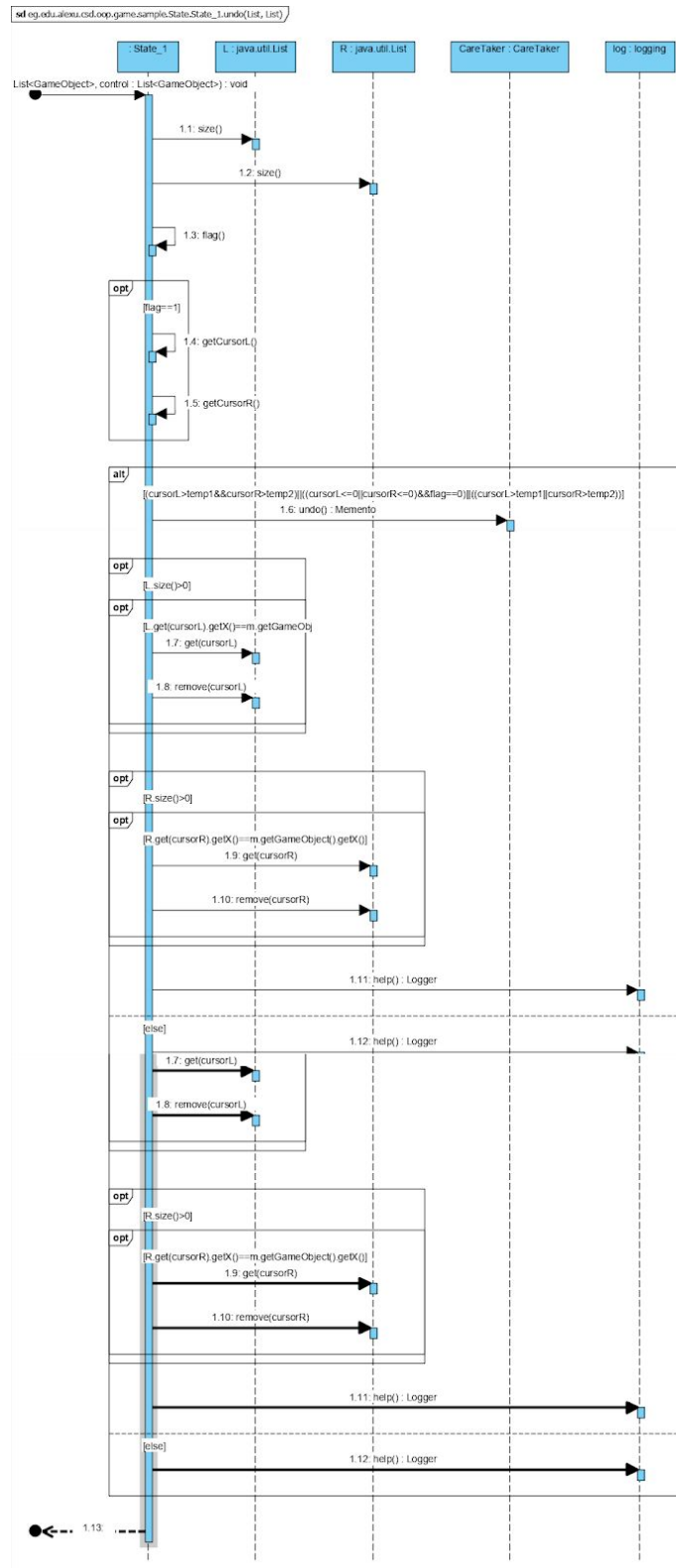4. Linh Ahmed (50)

Description:

A single player-game in which a clown carries two stacks of crowns and lanterns and there are a set of colored shapes queues that fall and he tries to catch them, if he manages to collect three consecutive shapes of the same color, then they vanish and his score increases. The game has Three levels/difficulties which varies the speed of the falling shapes, the score you get when you successfully catch 3 shapes of the same color, the given time and max height for each stack.

Class Diagram:

Sequence Diagram:

sd controller.state.State_1.checkState(List, List, List, GameObject, List, List, Strategy, boolean)
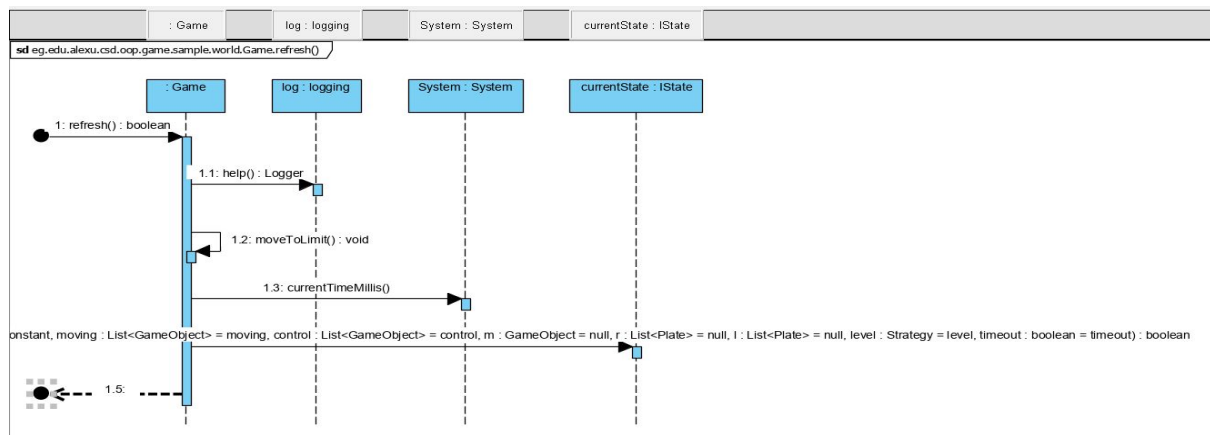
Participants (lifelines):
- State_1 : State_1
- m : GameObject
- log : logging
- L : java.util.List
- R : java.util.List
- m : Plate
- Originator : Originator
- CareTaker : CareTaker
- Start : Start

control : List<GameObject>, m : GameObject, r : List<Plate>, l : List<Plate>, level : Strategy, timeout : boolean) : boolean

1.1: j : LinkedListIterator

loop [j.hasNext()]
- 1.2: Next() : GameObject
- 1.3: setY((m.getY() + 1))

opt [m.getY() == 800]
- 1.4: setY(-1 * (int) (Math.random() * 800))
- 1.5: setX((int) (Math.random() * 900))
- 1.6: setX(m.getX())

1.7: help() : Logger

opt [!L.isEmpty()]
- 1.8: get(0)

loop [j.hasNext()]
- 1.13: Next() : GameObject

alt [R.isEmpty()]

opt [intersect(control.get(0), m) && control.get(0).getY() == m.getY() + m.getHeight()]
- 1.14: setX(x : int = control.get(0).getX()) : void
- 1.15: setY(y : int = m.getY()) : void
- 1.16: add(m)
- 1.17: setGameObject(GameObject : GameObject = m) : void
- 1.18: addMemento(m : Memento = facade.Originator.createMemento()) : void
- 1.19: help() : Logger

[(!R.isEmpty()) && (intersect(R.get(R.size() - 1), m))]
- 1.20: setX(x : int = R.get(R.size() - 1).getX()) : void
- 1.21: setY(y : int = R.get(R.size() - 1).getY() - m.getHeight()) : void
- 1.22: add(m)
- 1.2x: setGameObject(GameObject : GameObject = m) : void

1.32: setY(y : int = m.getY() - 5) : void

[!L.isEmpty() && intersect(L.get(L.size() - 1), m)]
- 1.33: setX(x : int = L.get(L.size() - 1).getX() - 15) : void
- 1.34: setY(y : int = L.get(L.size() - 1).getY() - m.getHeight()) : void
- 1.35: add(m)
- 1.36: setGameObject(GameObject : GameObject = m) : void
- 1.37: addMemento(m : Memento = facade.Originator.createMemento()) : void
- 1.38: help() : Logger

opt [flag == true]
- 1.39: help() : Logger

1.40: State(state : IState = State_2, constant : List<GameObject> = constant, moving : List<GameObject> = moving, control : List<GameObject> = control, m : GameObject = m, r : List<Plate> = R, l : List<Plate> = L, level : Strategy = level, timeout : boolean = timeout) : boolean

opt [moving.size() < 3]
- 1.36: setGameObject(GameObject : GameObject = m) : void
- 1.37: addMemento(m : Memento = facade.Originator.createMemento()) : void
- 1.38: help() : Logger

opt [flag == true]
- 1.39: help() : Logger

1.40: State(state : IState = State_2, constant : List<GameObject> = constant, moving : List<GameObject> = moving, control : List<GameObject> = control, m : GameObject = m, r : List<Plate> = R, l : List<Plate> = L, level : Strategy = level, timeout : boolean = timeout) : boolean

opt [moving.size() < 3]
- 1.41: getMoving() : List<GameObject>

opt [level.maxHeightOfPlate() == R.size() || level.maxHeightOfPlate() == L.size()]
- 1.42: help() : Logger

1.43:

Design Patterns:

Singleton:

Is used in several classes of the project when we need only one instance of a class. The class names ObserveScore , logging.
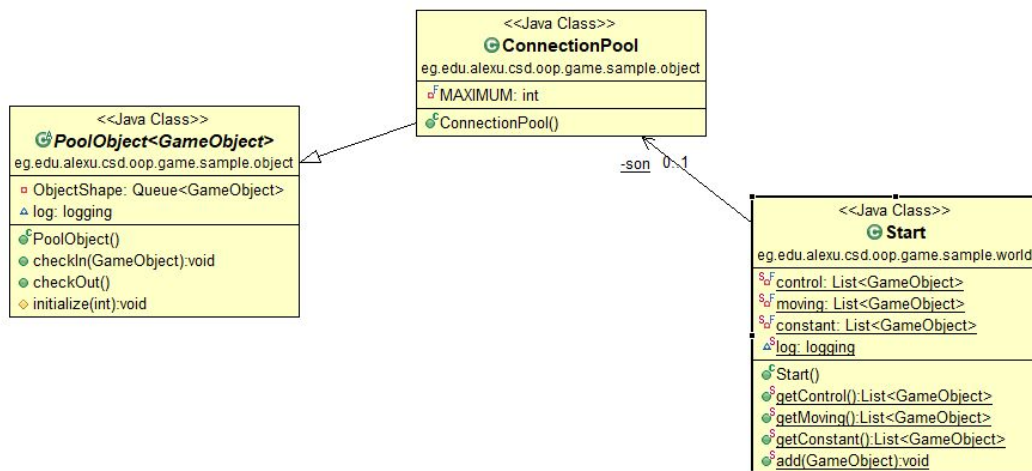
Factory:

when we deal with a reference from the interface and we have many options to instantiate this option to (many classes that implement the same interface) and we what to decide which class to create.
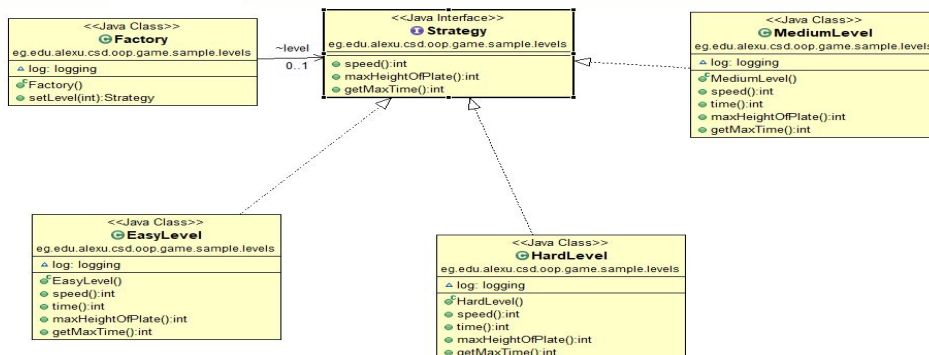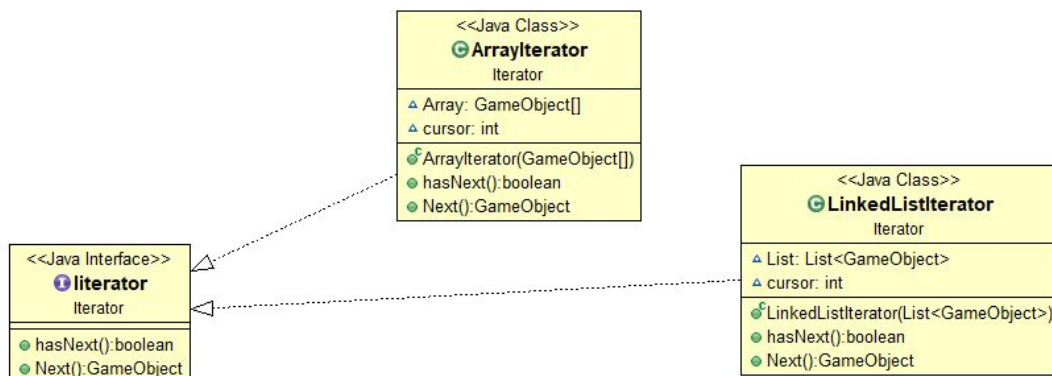
Pool:

We use the pool design pattern when there is a pool of resources (objects) that can be used under certain conditions (like time limit and max number and so on).

Iterator:

It's used to iterate on a collection of objects.

Snapshot:
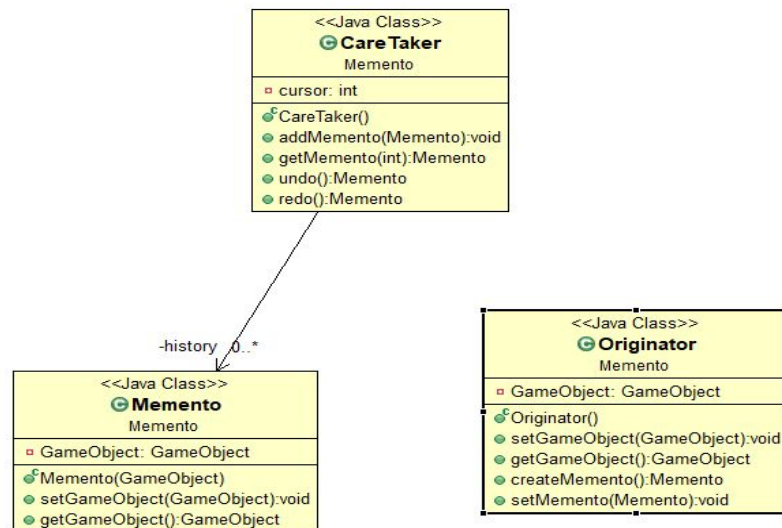
State:

We used a state design pattern to indicate the current state of shapes if the clown caught or not and if there are three consecutive shapes of the same color to increase the score.
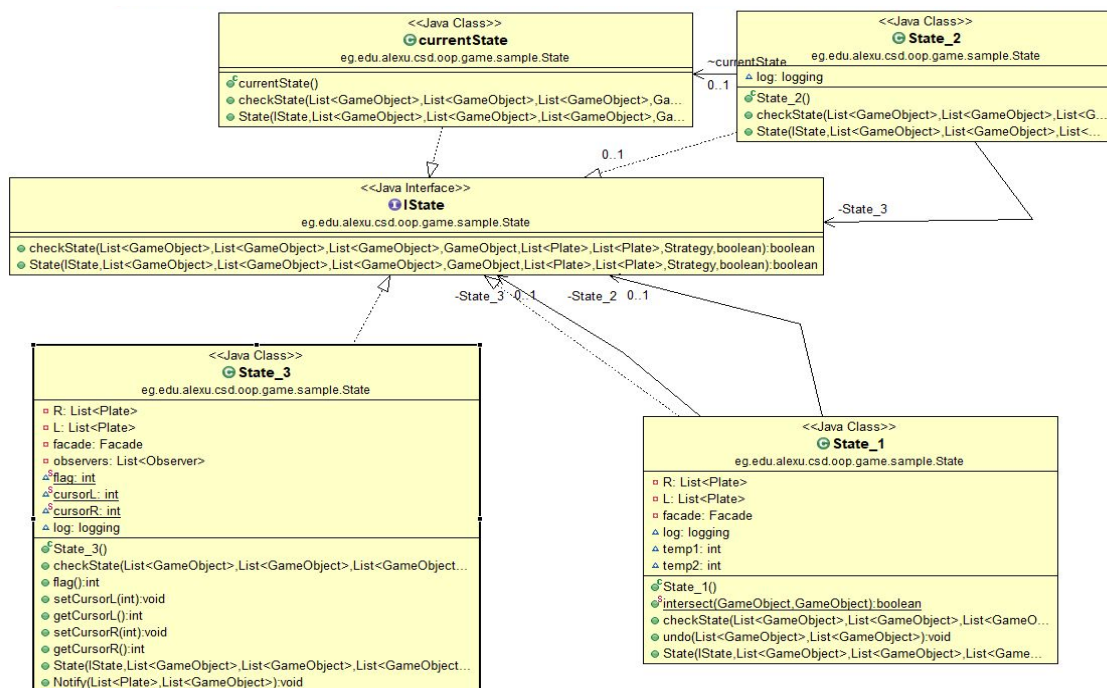
Strategy:

To determine all levels properties when levels become harder the game speed and the control speed are increased ,the max time is reduced , the number of shapes falling is increased, the time is reduced and the max height the plates can reach before he loses is reduced.
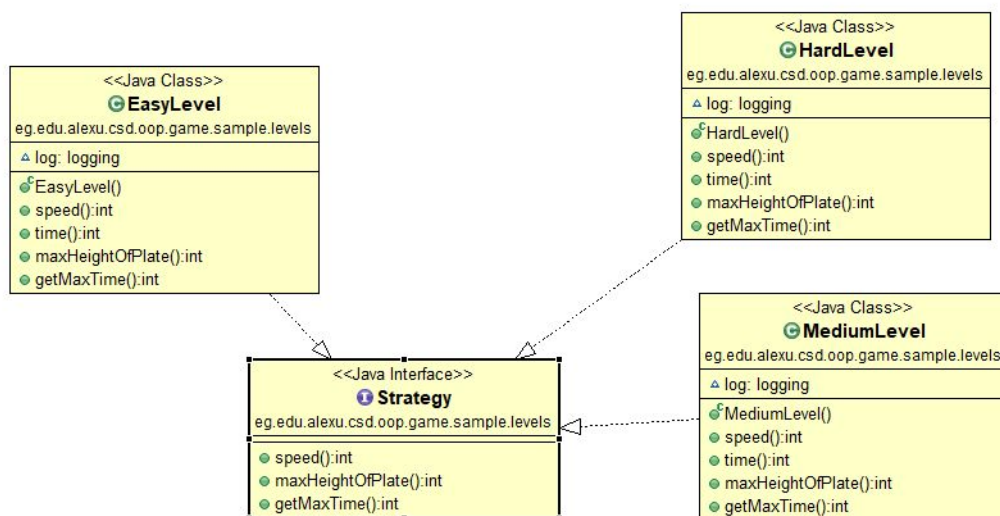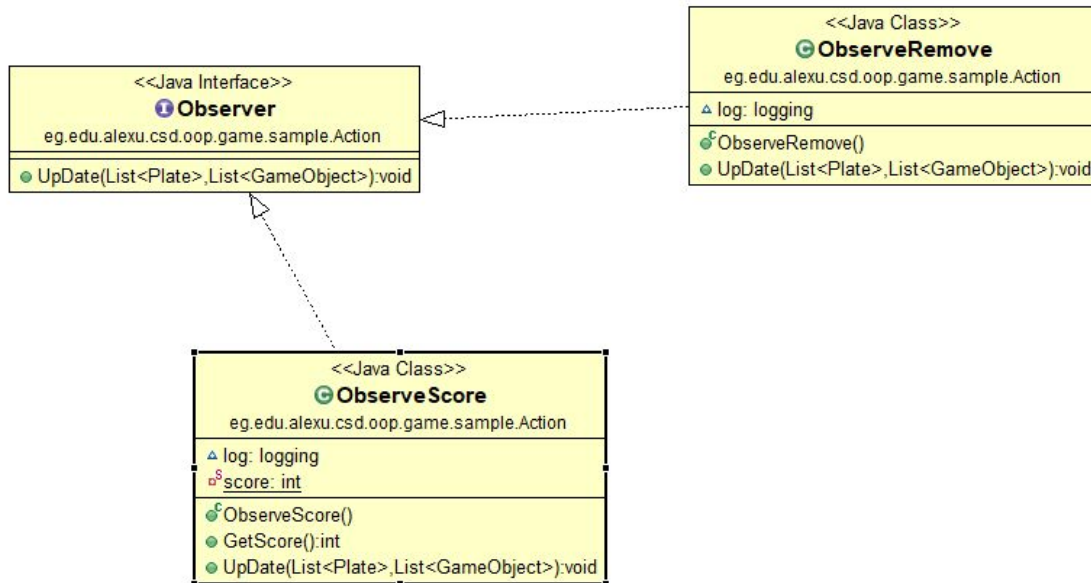

Flyweight:

Is used to add new color selected by the user to the pool.

Dynamic linkage:

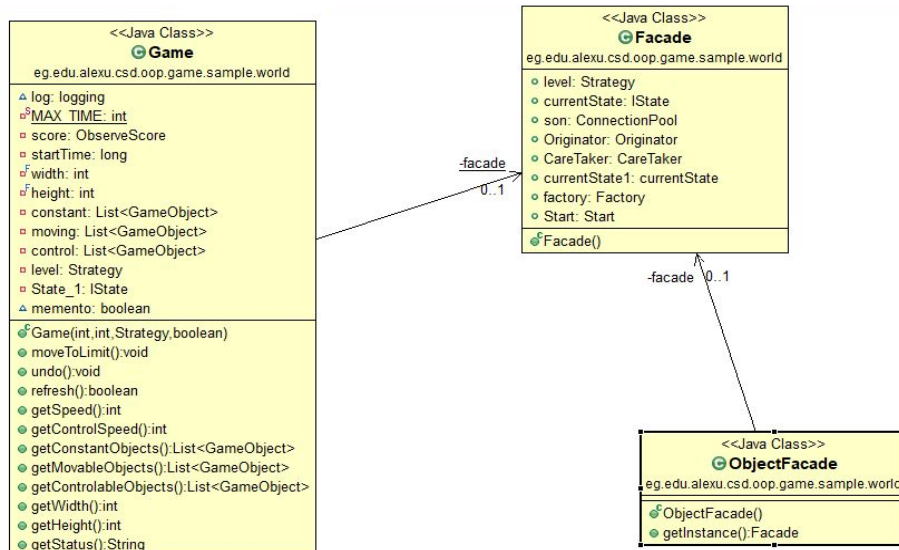It is used to load images from a jar which contains all game images.

Observer:

To update score and stacks of plates when needed.

Facade:

Connecting all game logic in a single class.

MVC:

The Model-View-Controller dominates all the project classes into 3 main packages.

Model: contains classes of shapes ,logging,snapshot and pool.

View: contains the class of GUI which is responsible for viewing the start frame of the game to choose the level then makes a call to the game engine to start the game.

Controller: contains classes of facade, iterator,observer,state and strategy.

Design decisions and Assumptions :

- There are some required design patterns have no specific requires such as
  - ➢ Snapshot :
    is used to save the shapes that intersected with the stick of the clown and enable the player to undo his latest moves.
  - ➢ Observer :
    is used to determine when shapes intersect with sticks or with each other. And when 3 shapes fall on one stick and having the same color the score will be increased.
- The main class is the Game class, which is passed through the constructor to the game engine after the user chooses his game level from the main menu.
- The game level changes some features in the game using the strategy design pattern as time & speed calculation method.
- Then after these classes that contain images, we load the images using dynamic linkage.

- The clown initially has right and left plates from which the user can decide the area of intersection so he can adjust the clown to catch the shapes
- The falling shape center must be inside this interval.
- The user starts with a limit of falling shapes decided by the level
    - ★ When the user catches a shape its reduced from the limit of shapes available.
    - ★ When the user collects any 3 successive shapes of the same color he gains score.
- The game ends when
    - ★ The time runs out.
    - ★ The shapes on either hand exceeds the max specified shape height for this level.

User Guide :

- Before clicking the Start button the user has three levels of difficulties Easy, Medium and Hard. or the game will use the default which is Easy.
- When the game starts the player should use the arrow keys on the keyboard to move the clown.
- Clown can move right and left only.
- At any time of the game If the number of plates reached the height of shapes for the chosen level then he loses.

SnapShots: