
CIRCUS
OF
PLATES

Team Members[1]

Ahmed Ezzat #11
Ahmed Reda #9
Fady Nabil #49
Mona Alaa #76

[1] Names are arranged in English alphabetical order.

A. Design Patterns

Behavioral Design Patterns

1. State Design Pattern

"Allow an object to alter its behavior when its internal state changes. The object will appear to change its class. "[2]

Circus of Plates extends the challenge through allowing a variety of 5 levels of difficulties through:

1) Decreasing game time.
2) Increasing plates speeds.
3) Increasing clowns speeds.
4) Reducing distance between plates.

State design pattern appears clearly and obviously through changing the game state -level of difficulty- from one state to another according to the player's choice on playing the Circus of Plates.

2. Snapshot Design Pattern (Memento)

"Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later. "

Circus of Plates harnesses the java code to follow the intent of the memento design pattern, through stoping the game timeline at a specific point, which allowed saving feature to be implemented and handled easily, memento design pattern allowed saving:

1) Clowns' plates.
2) Columns' coordinates.
3) Level of difficulty.
4) Points Scored.
5) Time.

3. Observer Design Pattern

"Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. "

Since observer design pattern resembles simple publisher-reader relationship through a reader subscribes to the publisher's feed and updates -or new magazine version… etc-.

Circus of Plates icons the relation between the clown's hand and the falling plates, where the clown's hand -acting as the publisher (observable)- which informs the falling plates -which acts as the reader (observer)- by there position and coordinates through the game flow.

---

[2] Design Patterns Elements of Reusable Object Oriented Software (GoF)

4. Model-View-Controller (MVC)

Following the main principles of *Object Oriented Design* of encapsulation and separation, MVC design pattern is used to isolate the main algorithm that controls the game from the parts that form the game's body.

This is clearly shown through dividing the whole project to 3 packages of the MVC.

5. Iterator Design Pattern

"Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation."

Iterator design pattern is implemented through our code to count the plates of similar color, where when 3 plates of same color are grouped consecutively they are erased from the game pane and the player's score is incremented.

With the iterator design pattern we eliminated the direct use of nested loops and that sort of methods to count an aggregate.

Creation Design Patterns

1. Factory Design Pattern

"Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses."

Circus of plates does contain an infinite number of plates, this infinite number should be created continuously during game live runtime to ensure game continuity and enhance our difficulties levels. So, factory design pattern was used to create multiple instances of multiple types of colorful plates which implements their general interface *shapes,* where the factory produces different type of plates according to their color. We have created our factory!

2. Singleton Design Pattern

"Ensure a class only has one instance, and provide a global point of access to it."

A unique factory of its type does complete and reinforce the abstract design idea of the game where there is no need for more than a factory of a global access to its products!
That is, a multi-spreader factory that deliver its products (plates) everywhere with no need to have more than a factory to:

1) Release any expected issues or bugs.
2) Ensure global access of an instance.

3. Objects' Pool Design Pattern

"Uses a set of initialized objects kept ready to use – a "pool" – rather than allocating and destroying them on demand"[3]

---

[3] https://en.wikipedia.org/wiki/Object_pool_pattern

Although they are infinite, they are few! This paradox is easily solved by the objects' pool design pattern, where a set of plates is pre-created by our factory and added to the pool and when a plate appears on the game pane it is acquired from the pool, if a plate is not catch by the clown it is re-added to the pool to be reviewed on the one again, but, if it is catch by the clown it remains in use -does not return to the pool- this may have caused our pool to dry if we have not used our factory for plates supply.

4. Prototype Design Pattern

"Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype"

Combined with memento design pattern, prototype design pattern copies the clown/s object/s to be included in that snapshot helping to make a confident snapshot for the game at a certain moment.

5. Builder Design Pattern

"Separate the construction of a complex object from its representation so that the same construction process can create different representations."

Clearly appears on building the whole from smaller parts that forms it which is present in our game in the object of the clown which is built of smaller objects (hands and body) that have a significant and major role in our simple game.

6. Dynamic Linkage Design Pattern

Allowing any shapes implementing our *shapes* interface to be added as a plugin to the Circus of Plates, this option clarify the game extendability beyond plates only to other worlds of imagination and creativity of a designer combined with a developer's implementation and reinforced design.

B. User's Guide

Circuit of Plates game is about a clown that moves on the horizontal axis catching plates, on catching 3 consecutive plates of same color they disappear and a point is added to your score.

Controllers

- RIGHT and LEFT arrows in the QWERTY keyboard to move the clown in both directions to catch plates.
- Letters: A - D in the QWERTY keyboard to increase or decrease the clown speed.

Levels

Starts from level 1 of lowest plates speed, time between plates is large and lowest clown speed where difficulty increases till reaching level 5 of maximum speeds and least plates distance.

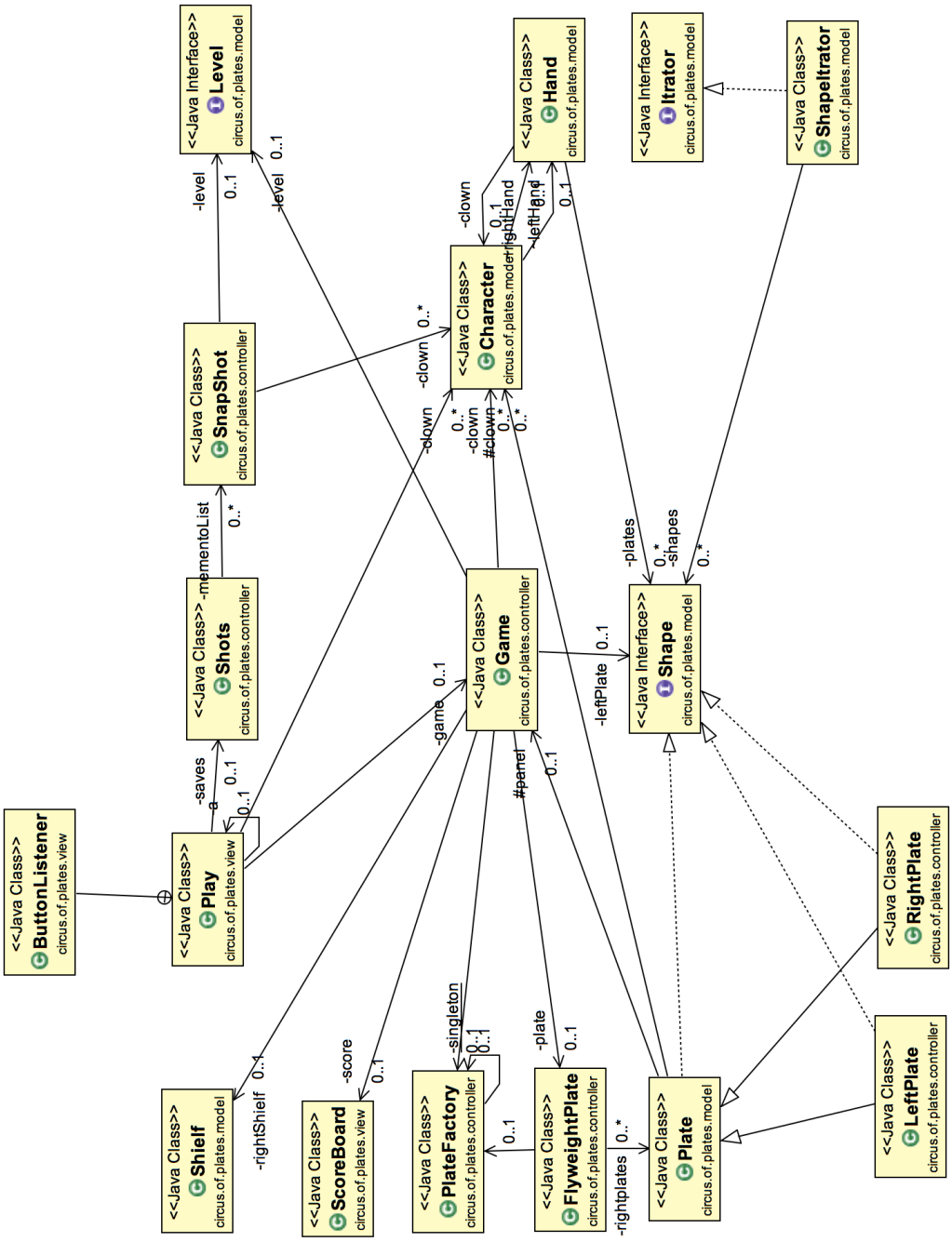
Levels are incremented automatically by time passing where for example level 2 is reached after passing 5 minutes of starting a new game.

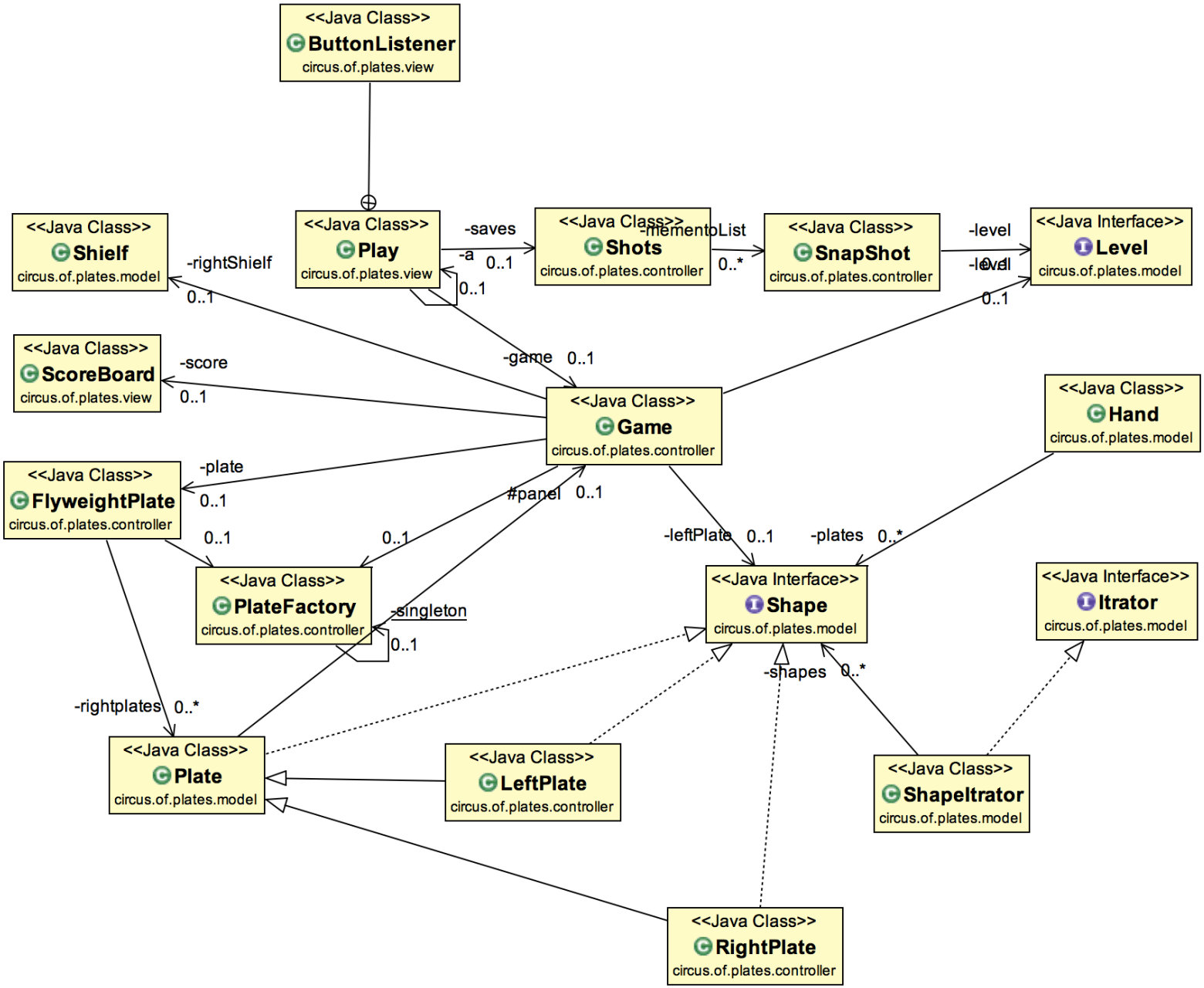
Features

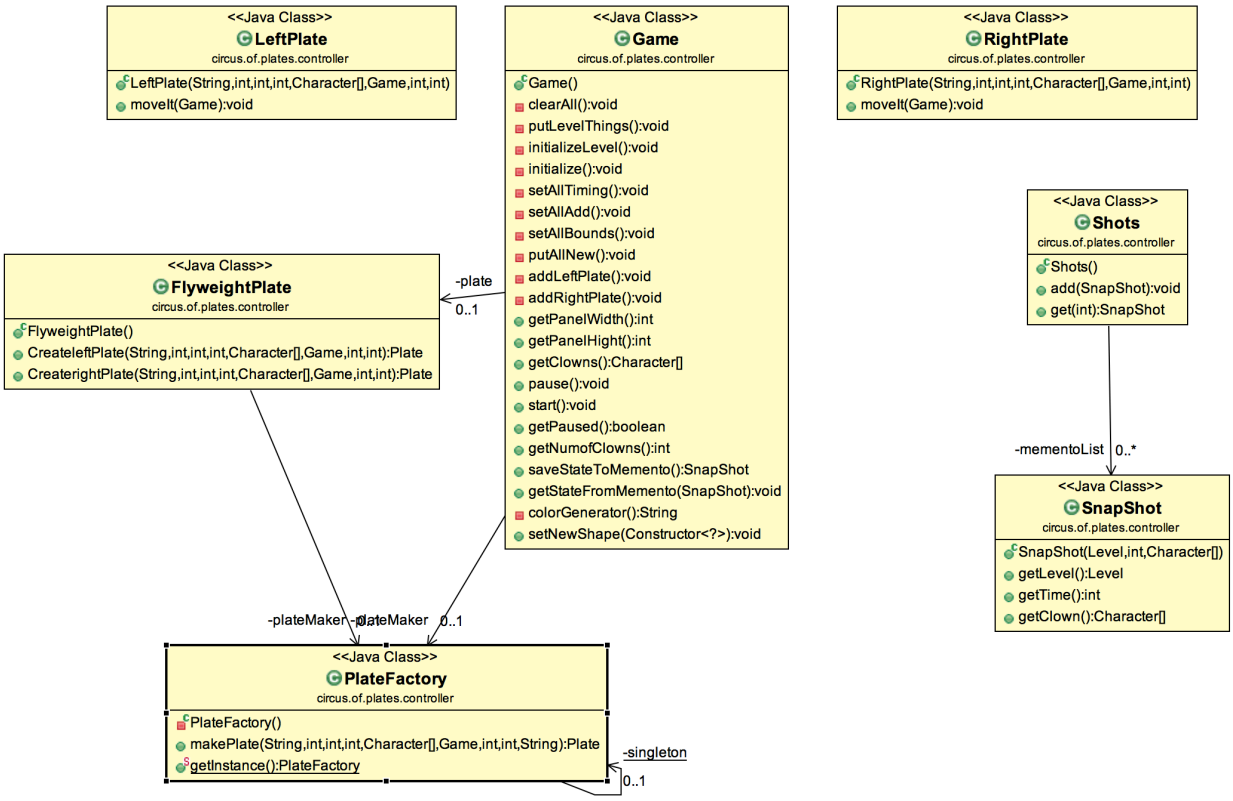
1) Saving a game at a specific point of time.
2) Loading a saved game.
3) Using plugins to extend the variety of shapes.

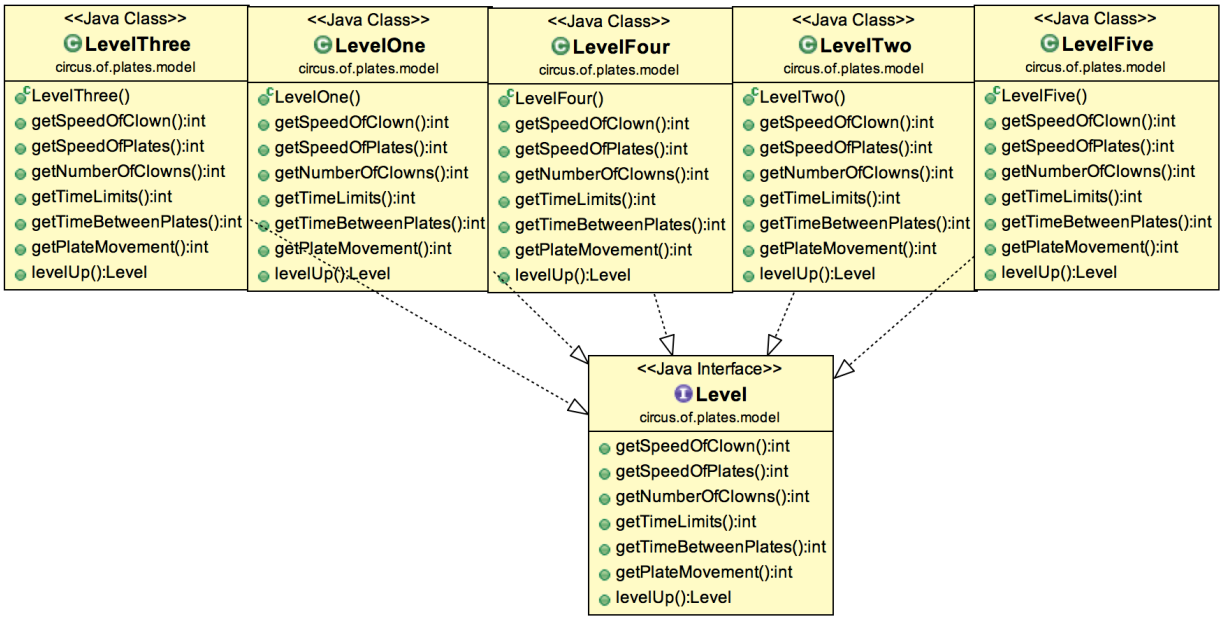
C. Unified Modeling Language

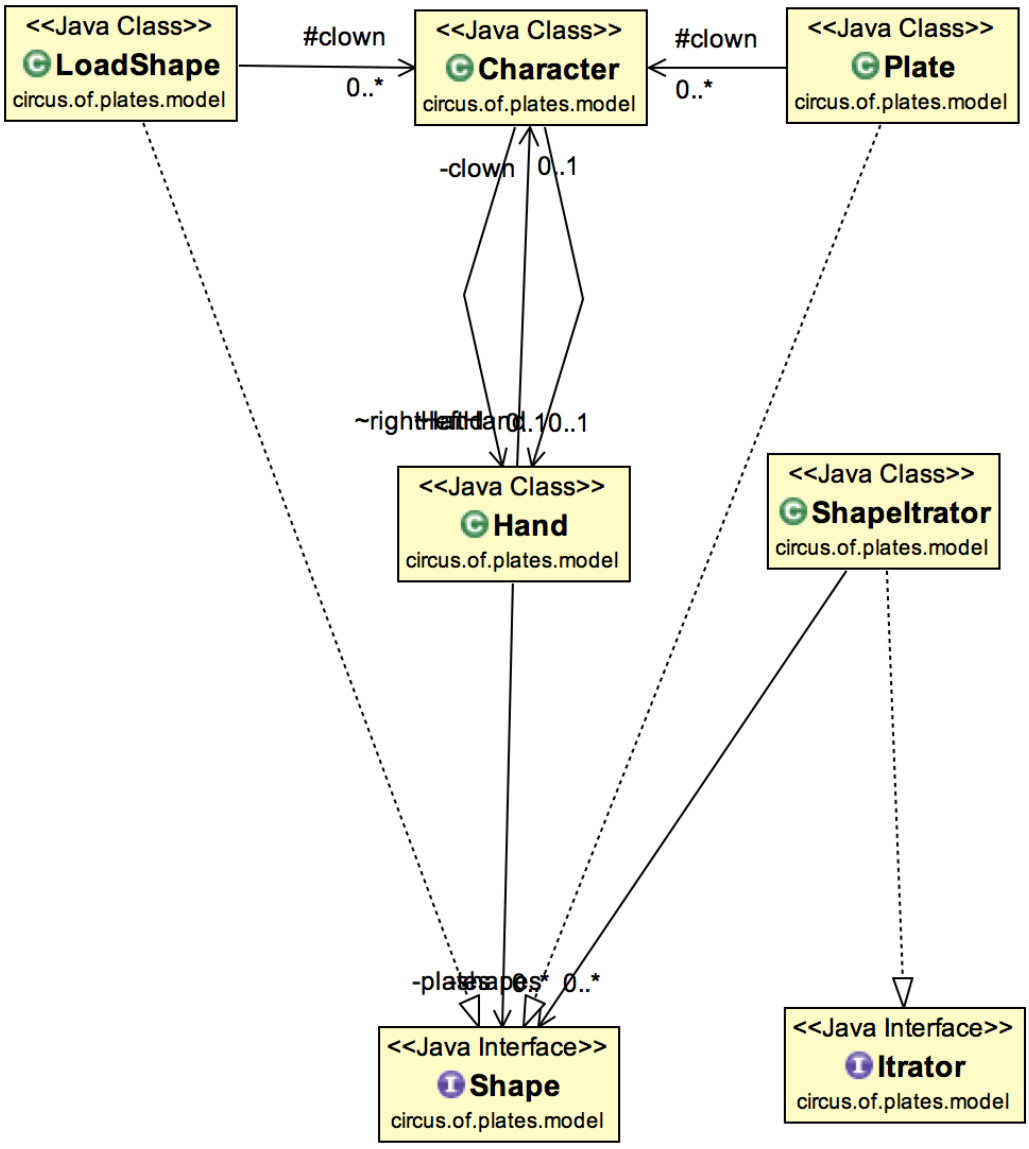
I. Class Diagram



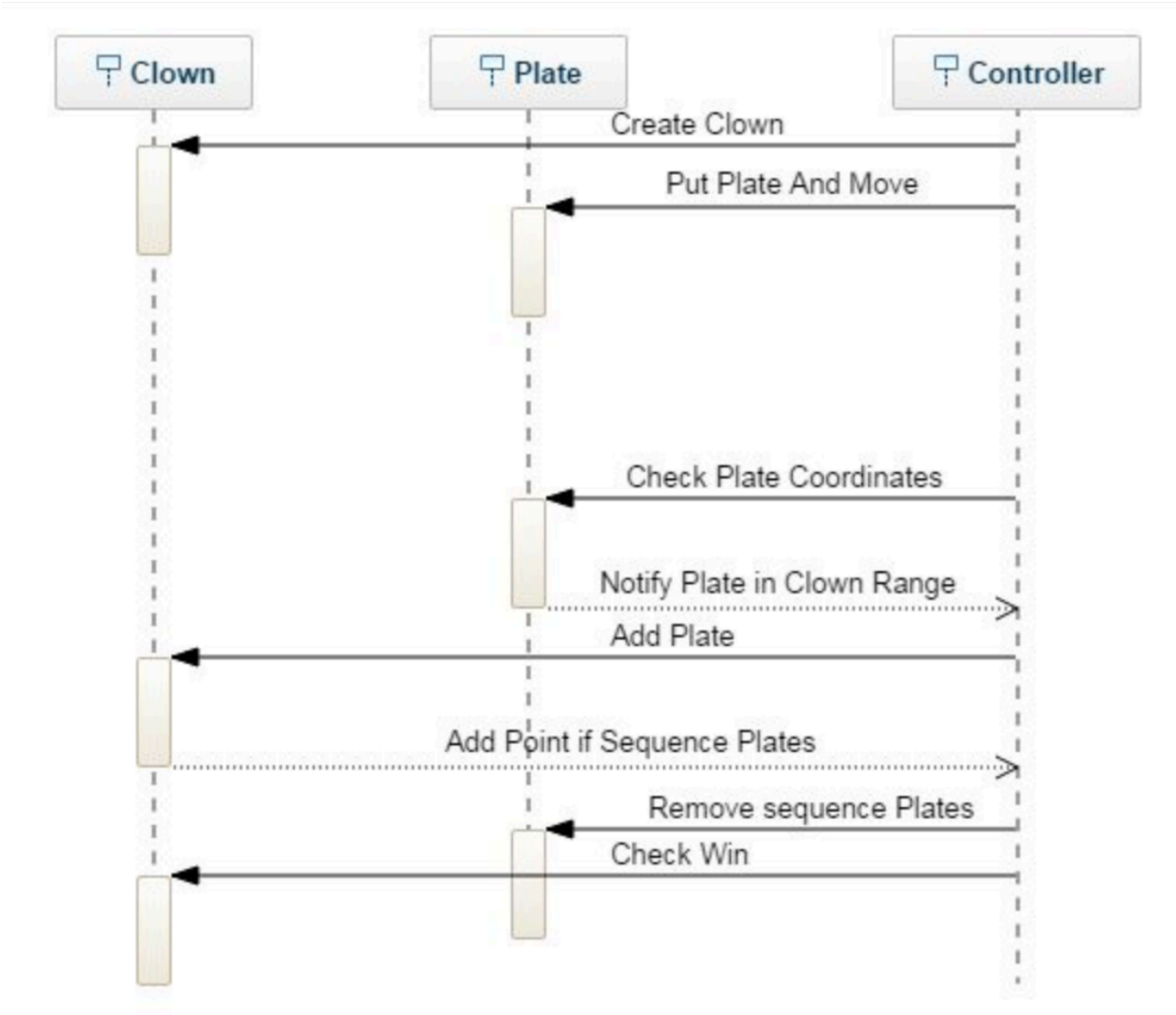

<<Java Class>>
ButtonListener
circus.of.plates.view
<<Java Class>>
Play
circus.of.plates.view
-saves
-a
0..1
0..1
<<Java Class>>
Shots
circus.of.plates.controller
-mementoList
0..*
<<Java Class>>
SnapShot
circus.of.plates.controller
-level
-level
0..1
<<Java Interface>>
Level
circus.of.plates.model
<<Java Class>>
Shielf
circus.of.plates.model
-rightShielf
0..1
<<Java Class>>
ScoreBoard
circus.of.plates.view
-score
0..1
-game 0..1
<<Java Class>>
Game
circus.of.plates.controller
<<Java Class>>
Hand
circus.of.plates.model
<<Java Class>>
FlyweightPlate
circus.of.plates.controller
-plate
0..1
0..1
0..1
#panel 0..1
-leftPlate 0..1
-plates 0..*
<<Java Class>>
PlateFactory
circus.of.plates.controller
-singleton
0..1
<<Java Interface>>
Shape
circus.of.plates.model
<<Java Interface>>
Itrator
circus.of.plates.model
-shapes 0..*
-rightplates 0..*
<<Java Class>>
Plate
circus.of.plates.model
<<Java Class>>
LeftPlate
circus.of.plates.controller
<<Java Class>>
ShapeItrator
circus.of.plates.model
<<Java Class>>
RightPlate
circus.of.plates.controller


<<Java Class>>
LeftPlate
circus.of.plates.controller
LeftPlate(String,int,int,int,Character[],Game,int,int)
moveIt(Game):void
<<Java Class>>
Game
circus.of.plates.controller
Game()
clearAll():void
putLevelThings():void
initializeLevel():void
initialize():void
setAllTiming():void
setAllAdd():void
setAllBounds():void
putAllNew():void
addLeftPlate():void
addRightPlate():void
getPanelWidth():int
getPanelHight():int
getClowns():Character[]
pause():void
start():void
getPaused():boolean
getNumofClowns():int
saveStateToMemento():SnapShot
getStateFromMemento(SnapShot):void
colorGenerator():String
setNewShape(Constructor<?>):void
<<Java Class>>
RightPlate
circus.of.plates.controller
RightPlate(String,int,int,int,Character[],Game,int,int)
moveIt(Game):void
<<Java Class>>
FlyweightPlate
circus.of.plates.controller
FlyweightPlate()
CreateleftPlate(String,int,int,int,Character[],Game,int,int):Plate
CreaterightPlate(String,int,int,int,Character[],Game,int,int):Plate
-plate
0..1
<<Java Class>>
Shots
circus.of.plates.controller
Shots()
add(SnapShot):void
get(int):SnapShot
-mementoList
0..*
<<Java Class>>
SnapShot
circus.of.plates.controller
SnapShot(Level,int,Character[])
getLevel():Level
getTime():int
getClown():Character[]
-plateMaker
-plateMaker
0..1
<<Java Class>>
PlateFactory
circus.of.plates.controller
PlateFactory()
makePlate(String,int,int,int,Character[],Game,int,int,String):Plate
getInstance():PlateFactory
-singleton
0..1

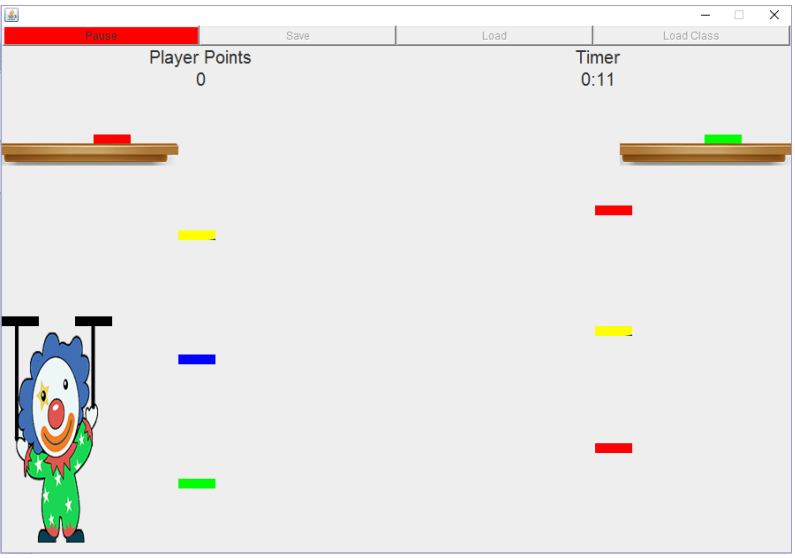

<<Java Class>>
LevelThree
circus.of.plates.model
LevelThree()
getSpeedOfClown():int
getSpeedOfPlates():int
getNumberOfClowns():int
getTimeLimits():int
getTimeBetweenPlates():int
getPlateMovement():int
levelUp():Level
<<Java Class>>
LevelOne
circus.of.plates.model
LevelOne()
getSpeedOfClown():int
getSpeedOfPlates():int
getNumberOfClowns():int
getTimeLimits():int
getTimeBetweenPlates():int
getPlateMovement():int
levelUp():Level
<<Java Class>>
LevelFour
circus.of.plates.model
LevelFour()
getSpeedOfClown():int
getSpeedOfPlates():int
getNumberOfClowns():int
getTimeLimits():int
getTimeBetweenPlates():int
getPlateMovement():int
levelUp():Level
<<Java Class>>
LevelTwo
circus.of.plates.model
LevelTwo()
getSpeedOfClown():int
getSpeedOfPlates():int
getNumberOfClowns():int
getTimeLimits():int
getTimeBetweenPlates():int
getPlateMovement():int
levelUp():Level
<<Java Class>>
LevelFive
circus.of.plates.model
LevelFive()
getSpeedOfClown():int
getSpeedOfPlates():int
getNumberOfClowns():int
getTimeLimits():int
getTimeBetweenPlates():int
getPlateMovement():int
levelUp():Level
<<Java Interface>>
Level
circus.of.plates.model
getSpeedOfClown():int
getSpeedOfPlates():int
getNumberOfClowns():int
getTimeLimits():int
getTimeBetweenPlates():int
getPlateMovement():int
levelUp():Level

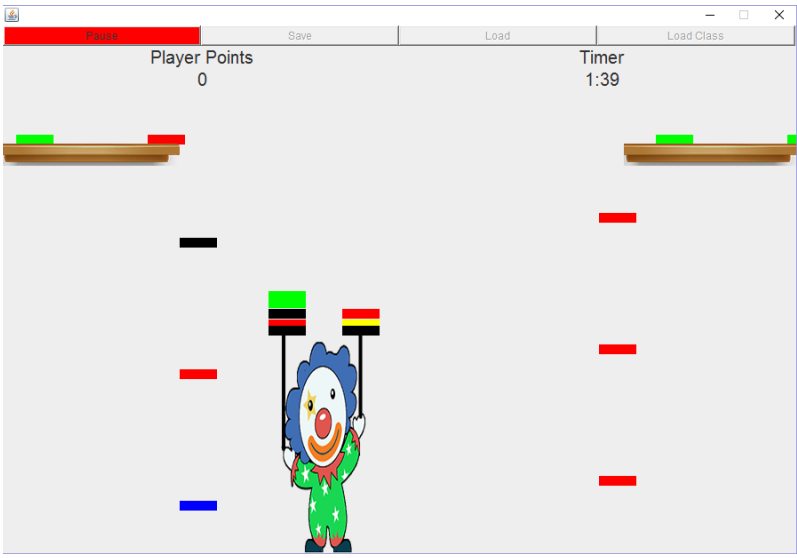

<<Java Class>>
LoadShape
circus.of.plates.model
#clown
0..*
<<Java Class>>
Character
circus.of.plates.model
#clown
0..*
<<Java Class>>
Plate
circus.of.plates.model
-clown
0..1
0..1
<<Java Class>>
Hand
circus.of.plates.model
<<Java Class>>
ShapeItrator
circus.of.plates.model
0..*
0..*
<<Java Interface>>
Shape
circus.of.plates.model
<<Java Interface>>
Itrator
circus.of.plates.model

II. Sequence Diagram

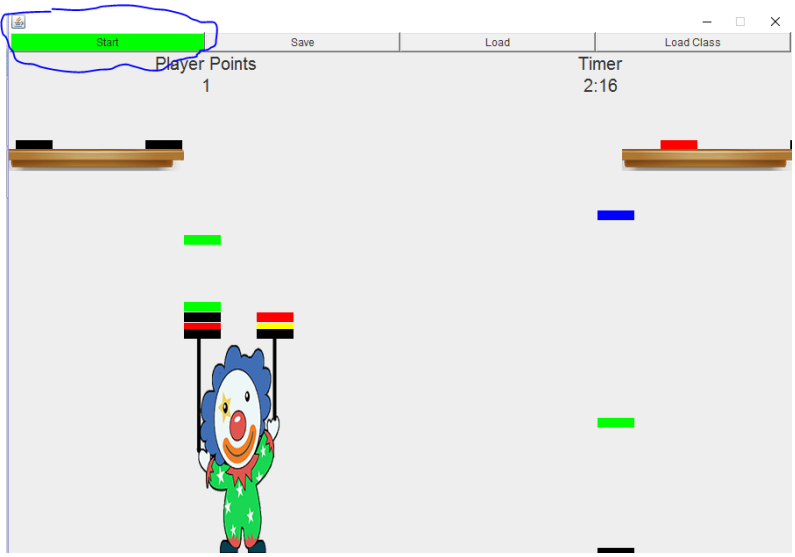


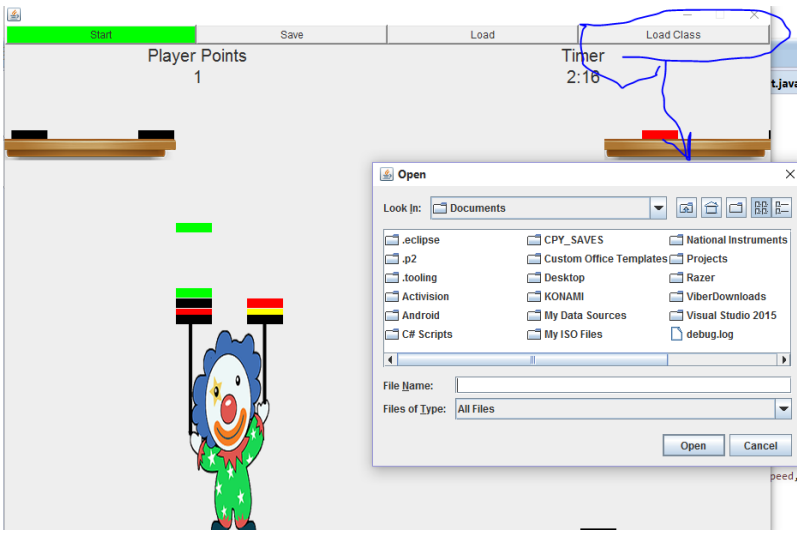
D. Game Screenshots



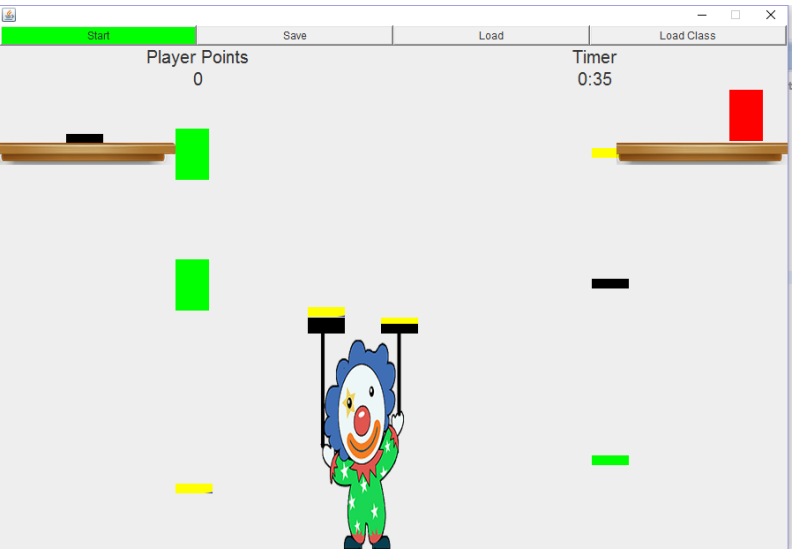


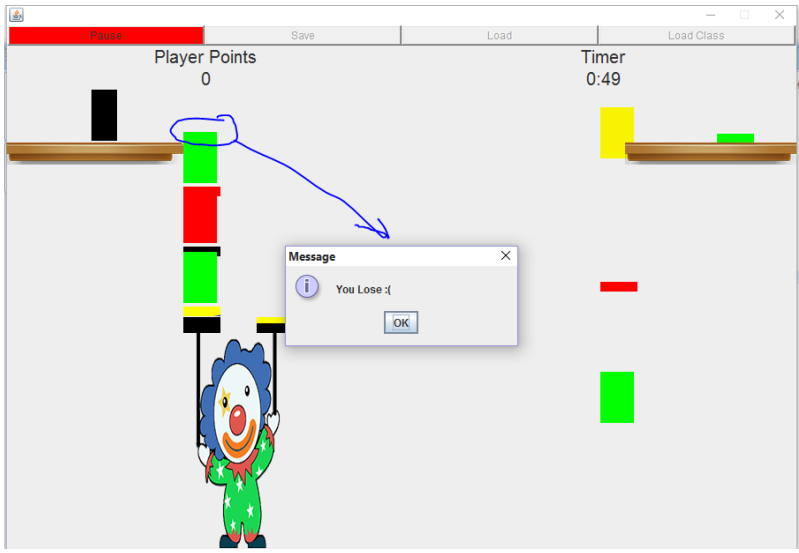
Starting the game and collecting plates



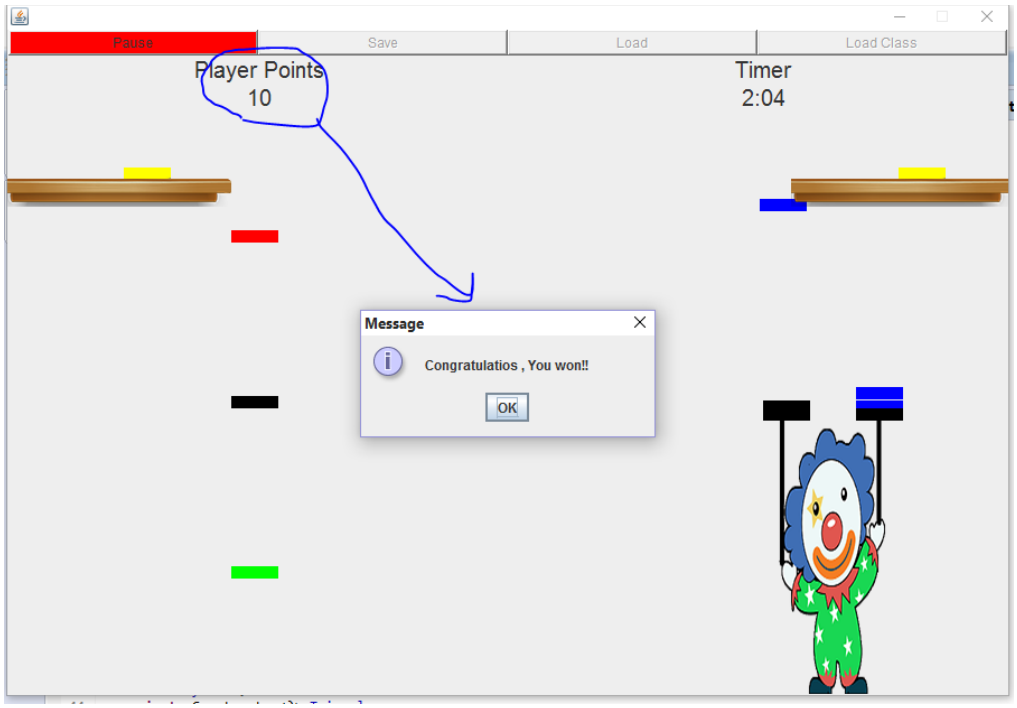


Loading external shapes through class loading





You lose when a stack is full!



You win on collecting 10 points