جامعة الإسكندرية

ALEXANDRIA UNIVERSITY

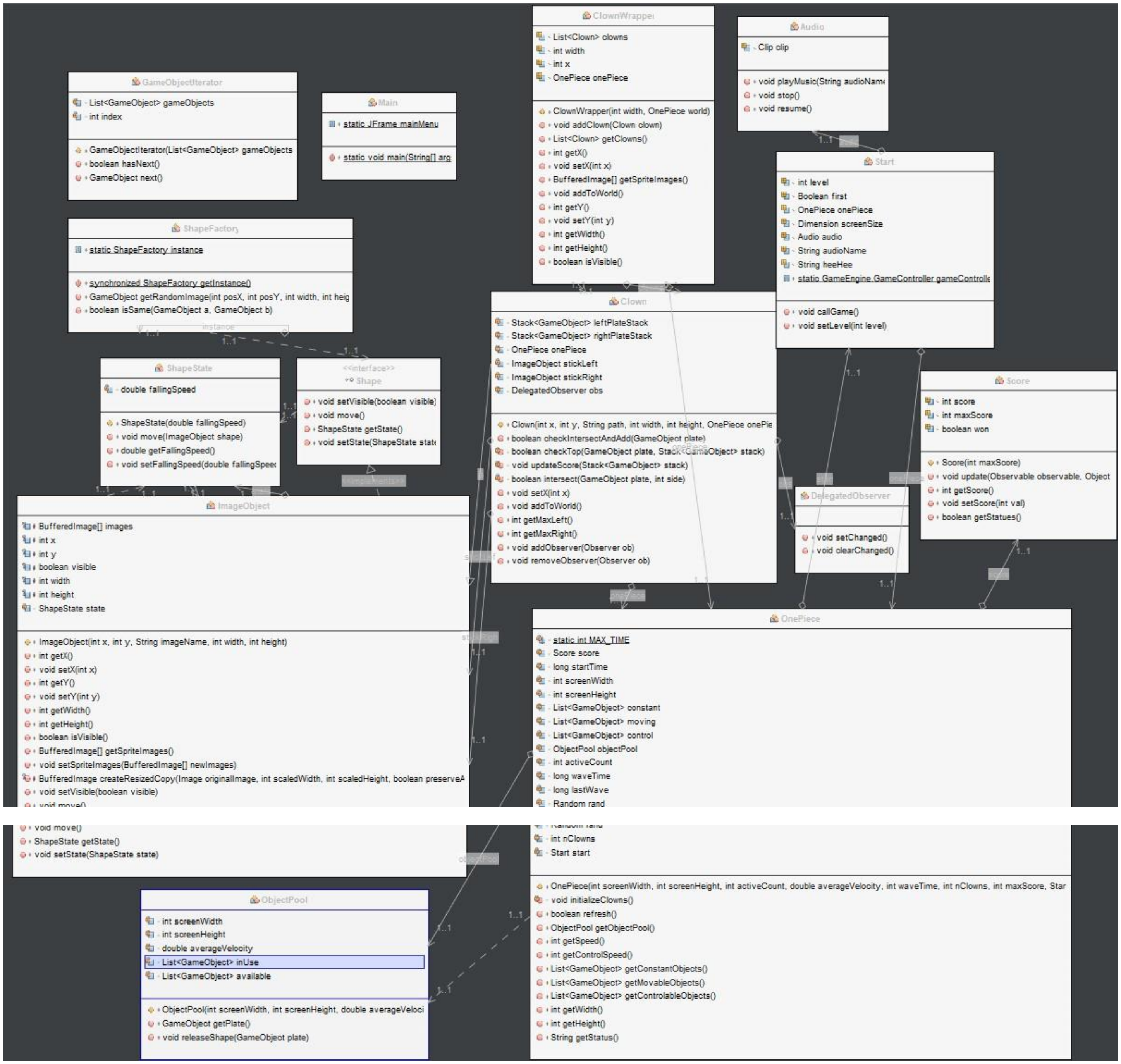
Programming 2

Circus of Plates: ONE PIECE Edition

Final Project

Ossama Ashraf Tammam	7352
Karim Ashraf ElKazaz	7476
Seif Eldin Mohamed Elsayaad	7401
Youssef Essam Youssef	7612

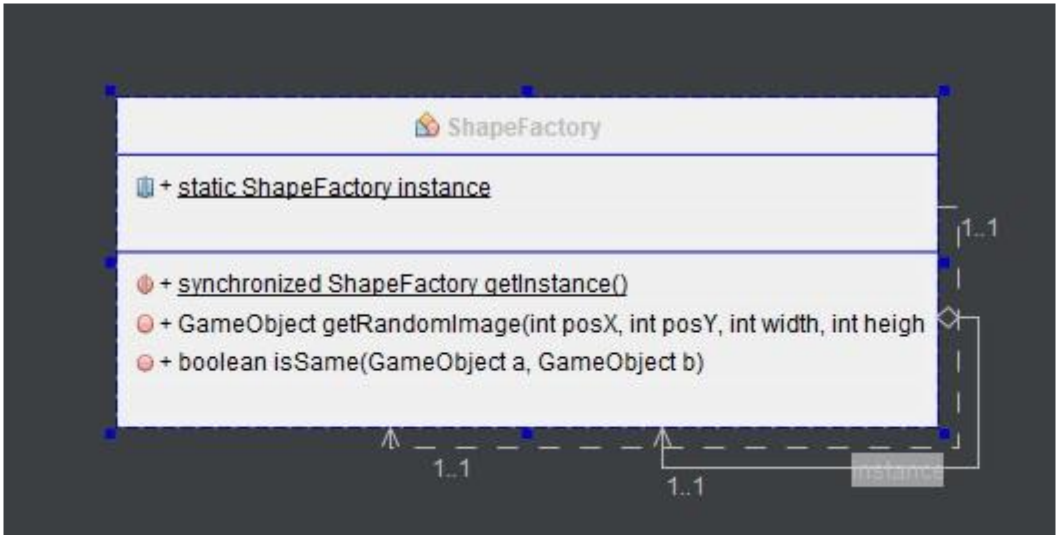
UML


ClownWrapper
List<Clown> clowns
int width
int x
OnePiece onePiece
ClownWrapper(int width, OnePiece world)
void addClown(Clown clown)
List<Clown> getClowns()
int getX()
void setX(int x)
BufferedImage[] getSpriteImages()
void addToWorld()
int getY()
void setY(int y)
int getWidth()
int getHeight()
boolean isVisible()
Audio
Clip clip
void playMusic(String audioName
void stop()
void resume()
GameObjectIterator
List<GameObject> gameObjects
int index
GameObjectIterator(List<GameObject> gameObjects
boolean hasNext()
GameObject next()
Main
static JFrame mainMenu
static void main(String[] arg
Start
int level
Boolean first
OnePiece onePiece
Dimension screenSize
Audio audio
String audioName
String heeHee
static GameEngine.GameController gameControlle
void callGame()
void setLevel(int level)
ShapeFactory
static ShapeFactory instance
synchronized ShapeFactory getInstance()
GameObject getRandomImage(int posX, int posY, int width, int heig
boolean isSame(GameObject a, GameObject b)
instance
1..1
Clown
Stack<GameObject> leftPlateStack
Stack<GameObject> rightPlateStack
OnePiece onePiece
ImageObject stickLeft
ImageObject stickRight
DelegatedObserver obs
Clown(int x, int y, String path, int width, int height, OnePiece onePie
boolean checkIntersectAndAdd(GameObject plate)
boolean checkTop(GameObject plate, Stack<GameObject> stack)
void updateScore(Stack<GameObject> stack)
boolean intersect(GameObject plate, int side)
void setX(int x)
void addToWorld()
int getMaxLeft()
int getMaxRight()
void addObserver(Observer ob)
void removeObserver(Observer ob)
ShapeState
double fallingSpeed
ShapeState(double fallingSpeed)
void move(ImageObject shape)
double getFallingSpeed()
void setFallingSpeed(double fallingSpee
<<interface>>
Shape
void setVisible(boolean visible)
void move()
ShapeState getState()
void setState(ShapeState stat
<<implements>>
Score
int score
int maxScore
boolean won
Score(int maxScore)
void update(Observable observable, Object
int getScore()
void setScore(int val)
boolean getStatues()
DelegatedObserver
void setChanged()
void clearChanged()
ImageObject
BufferedImage[] images
int x
int y
boolean visible
int width
int height
ShapeState state
ImageObject(int x, int y, String imageName, int width, int height)
int getX()
void setX(int x)
int getY()
void setY(int y)
int getWidth()
int getHeight()
boolean isVisible()
BufferedImage[] getSpriteImages()
void setSpriteImages(BufferedImage[] newImages)
BufferedImage createResizedCopy(Image originalImage, int scaledWidth, int scaledHeight, boolean preserveA
void setVisible(boolean visible)
void move()
ShapeState getState()
void setState(ShapeState state)
OnePiece
static int MAX_TIME
Score score
long startTime
int screenWidth
int screenHeight
List<GameObject> constant
List<GameObject> moving
List<GameObject> control
ObjectPool objectPool
int activeCount
long waveTime
long lastWave
Random rand
int nClowns
Start start
OnePiece(int screenWidth, int screenHeight, int activeCount, double averageVelocity, int waveTime, int nClowns, int maxScore, Star
void initializeClowns()
boolean refresh()
ObjectPool getObjectPool()
int getSpeed()
int getControlSpeed()
List<GameObject> getConstantObjects()
List<GameObject> getMovableObjects()
List<GameObject> getControlableObjects()
int getWidth()
int getHeight()
String getStatus()
ObjectPool
int screenWidth
int screenHeight
double averageVelocity
List<GameObject> inUse
List<GameObject> available
ObjectPool(int screenWidth, int screenHeight, double averageVeloci
GameObject getPlate()
void releaseShape(GameObject plate)

Design Patterns

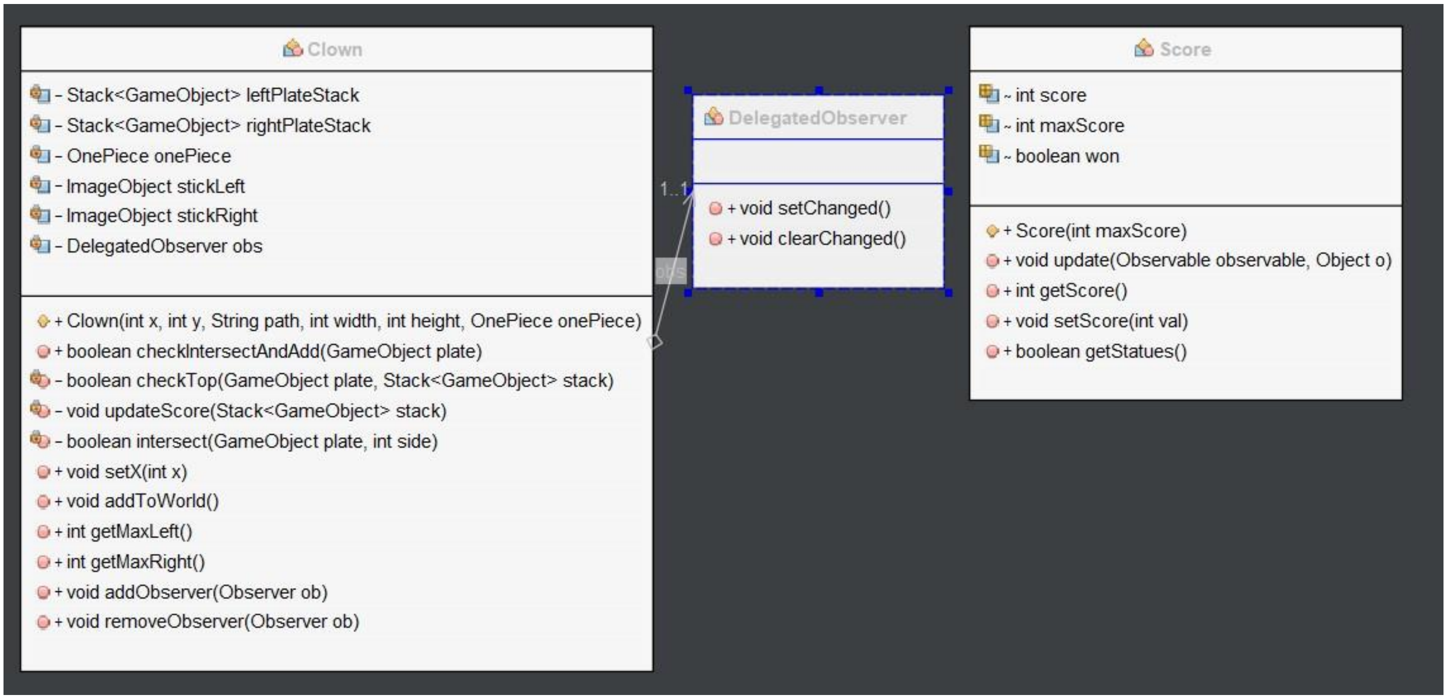
1. Singleton Design Pattern

It is used in “ShapeFactory” class which is responsible for loading shapes classes and getting instances from them.


ShapeFactory
+ static ShapeFactory instance
+ synchronized ShapeFactory getInstance()
+ GameObject getRandomImage(int posX, int posY, int width, int heigh
+ boolean isSame(GameObject a, GameObject b)
1..1
1..1
1..1
instance

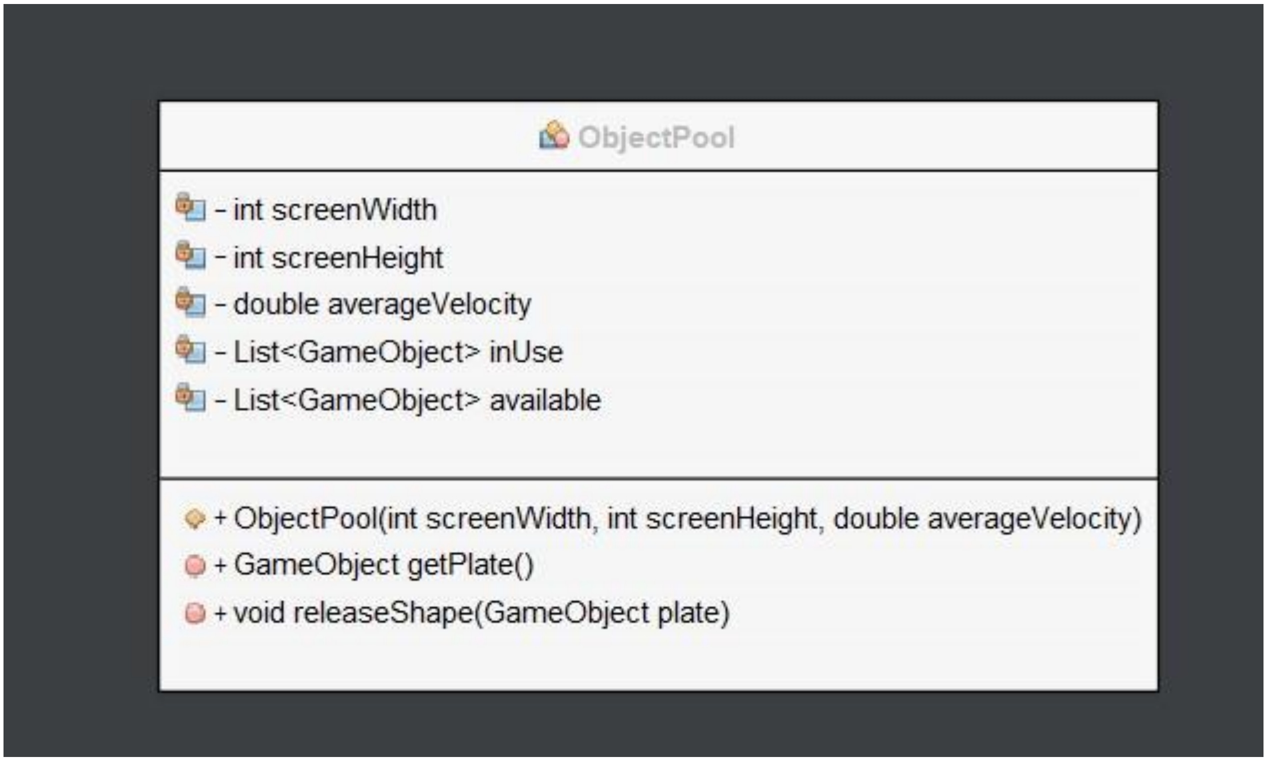
2. Observer Design Pattern

It is used in clown class when it collects three plates of the same type sequentially on any stick the observer set changed and notify that change thus player’s score increases by one.


Clown
- Stack<GameObject> leftPlateStack
- Stack<GameObject> rightPlateStack
- OnePiece onePiece
- ImageObject stickLeft
- ImageObject stickRight
- DelegatedObserver obs
+ Clown(int x, int y, String path, int width, int height, OnePiece onePiece)
+ boolean checkIntersectAndAdd(GameObject plate)
- boolean checkTop(GameObject plate, Stack<GameObject> stack)
- void updateScore(Stack<GameObject> stack)
- boolean intersect(GameObject plate, int side)
+ void setX(int x)
+ void addToWorld()
+ int getMaxLeft()
+ int getMaxRight()
+ void addObserver(Observer ob)
+ void removeObserver(Observer ob)
DelegatedObserver
+ void setChanged()
+ void clearChanged()
1..1
obs
Score
~ int score
~ int maxScore
~ boolean won
+ Score(int maxScore)
+ void update(Observable observable, Object o)
+ int getScore()
+ void setScore(int val)
+ boolean getStatues()

3. Object Pool Design Pattern

It is used to limit objects creation through reusing fallen plates when they disappear on the screen or be removed through using two Linked Lists, one to hold available objects and the other holds used objects.



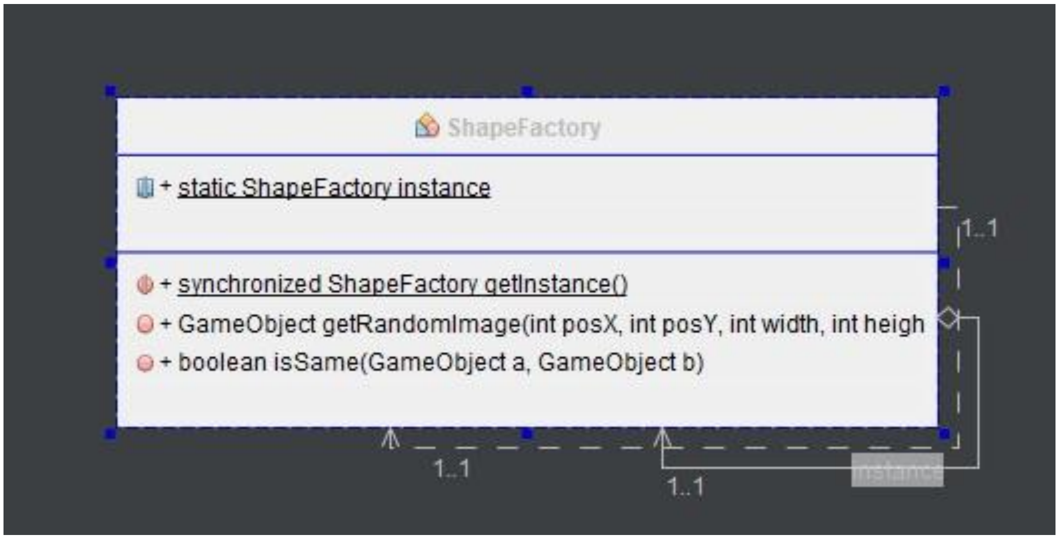
ObjectPool

- int screenWidth
- int screenHeight
- double averageVelocity
- List<GameObject> inUse
- List<GameObject> available

+ ObjectPool(int screenWidth, int screenHeight, double averageVelocity)
+ GameObject getPlate()
+ void releaseShape(GameObject plate)

4. Shape Factory Design Pattern

This class is responsible for creating the shapes that will be used in the game using Math.Random to pick a plate from between the three plates available in the game.



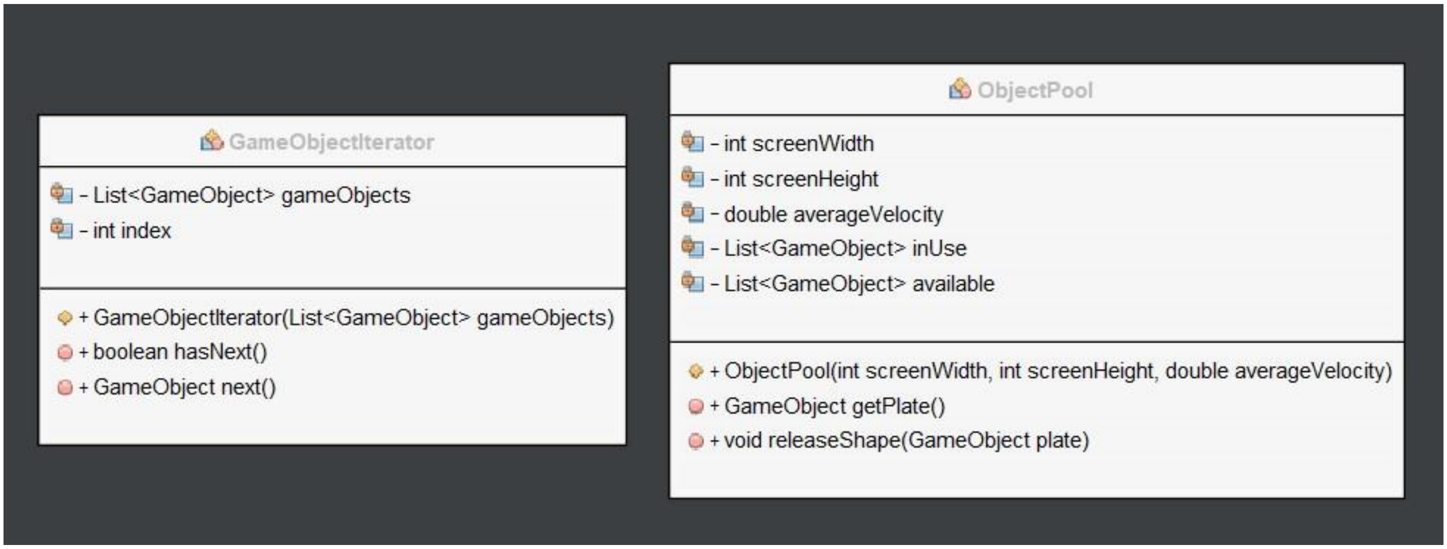
ShapeFactory

+ static ShapeFactory instance

+ synchronized ShapeFactory getInstance()
+ GameObject getRandomImage(int posX, int posY, int width, int heigh
+ boolean isSame(GameObject a, GameObject b)

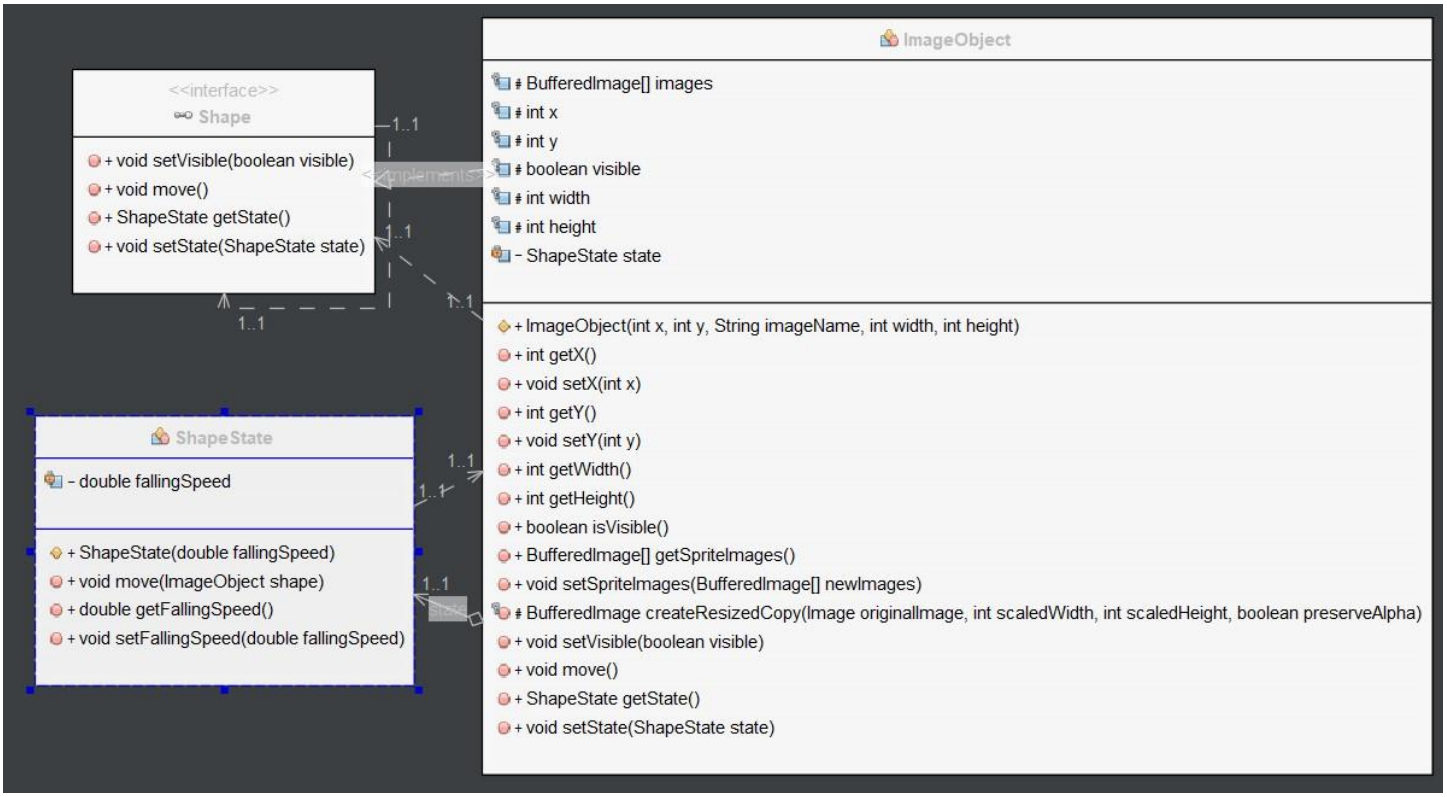
5. Iterator Design Pattern

Created a GameObjectIterator class to iterate over available Linked List in the ObjectPool class.


GameObjectIterator
- List<GameObject> gameObjects
- int index
+ GameObjectIterator(List<GameObject> gameObjects)
+ boolean hasNext()
+ GameObject next()
ObjectPool
- int screenWidth
- int screenHeight
- double averageVelocity
- List<GameObject> inUse
- List<GameObject> available
+ ObjectPool(int screenWidth, int screenHeight, double averageVelocity)
+ GameObject getPlate()
+ void releaseShape(GameObject plate)

6. State Design Pattern

It is used to give the falling plates their falling physics. Every falling plate has a GameState state initialized in it.


<<interface>>
Shape
+ void setVisible(boolean visible)
+ void move()
+ ShapeState getState()
+ void setState(ShapeState state)
1..1
<<implements>>
1..1
1..1
1..1
ImageObject
# BufferedImage[] images
# int x
# int y
# boolean visible
# int width
# int height
- ShapeState state
+ ImageObject(int x, int y, String imageName, int width, int height)
+ int getX()
+ void setX(int x)
+ int getY()
+ void setY(int y)
+ int getWidth()
+ int getHeight()
+ boolean isVisible()
+ BufferedImage[] getSpriteImages()
+ void setSpriteImages(BufferedImage[] newImages)
# BufferedImage createResizedCopy(Image originalImage, int scaledWidth, int scaledHeight, boolean preserveAlpha)
+ void setVisible(boolean visible)
+ void move()
+ ShapeState getState()
+ void setState(ShapeState state)
ShapeState
- double fallingSpeed
+ ShapeState(double fallingSpeed)
+ void move(ImageObject shape)
+ double getFallingSpeed()
+ void setFallingSpeed(double fallingSpeed)
1..1
1..1
1..1
state

Game Behavior

- ShapesFactory chooses random plates and prepares them to be used during the game.
- An ObjectPool was created to reduce the number of created plates by having two Linked Lists “isUse” which holds the current elements being on the screen and “available” which holds unused created elements to take from.
- “OnePiece” class is the basic environment; depending on its constructor it initializes instances –clowns and moving plates- that determine current level.
- “Clown” class is the class where we check if there is an intersection between the fallen plate and the clown stick depending on the width of its previous plate or stick, if there was one it adds the plate to the stick’s stack otherwise the plate falls down, gets out of screen border, and then be released by taking it back to “available” LinkedList.
- Every time happens an intersection it checks if the last 3 plates have the same image, then the observer notifies that, the score of player increases by one and plates are removed from stack and get imported into “available” LinkedList.