Project By:

Youssef ElKady 7651 **Farida Soffar 7720** **Salma ElMahy 7668**

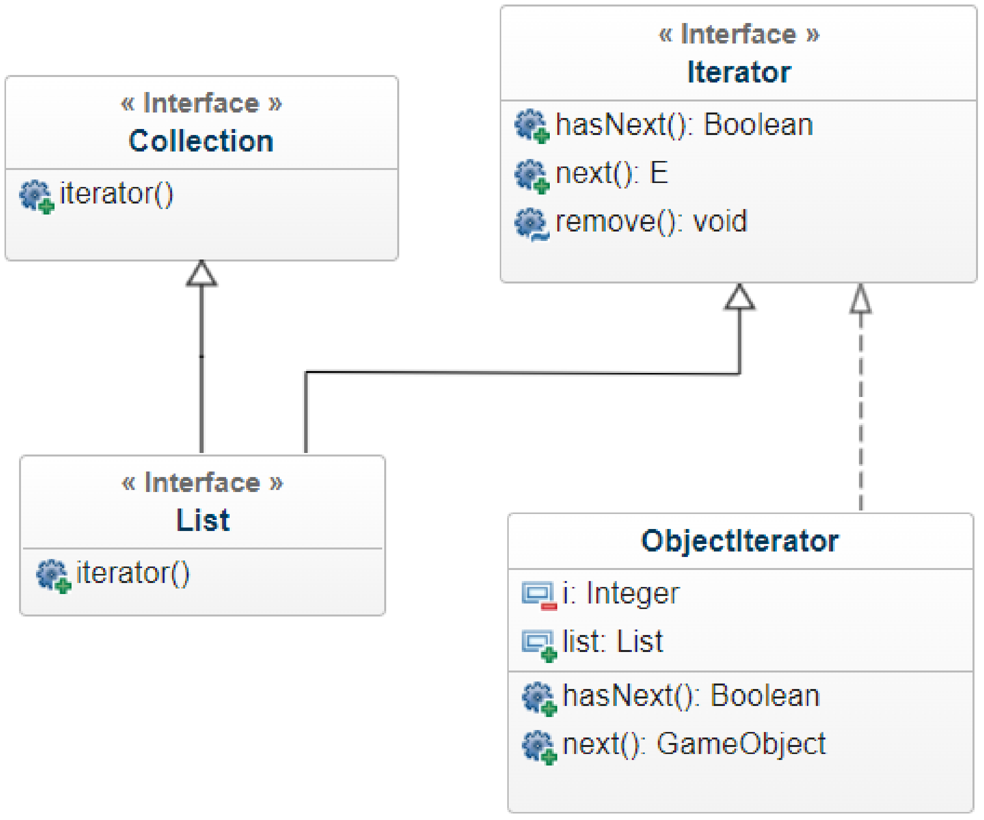
Circus Of Plates

Design Patterns used:

1. Iterator (Behavioral design pattern)

The aim of using the Iterator Pattern was to ease traversing through a collection without exposing its underlying representation. Collections are one of the most used data types in programming (list, stack, tree, etc..). The main idea of the Iterator pattern is to extract the traversal behavior of a collection into a separate object called an ***iterator***.

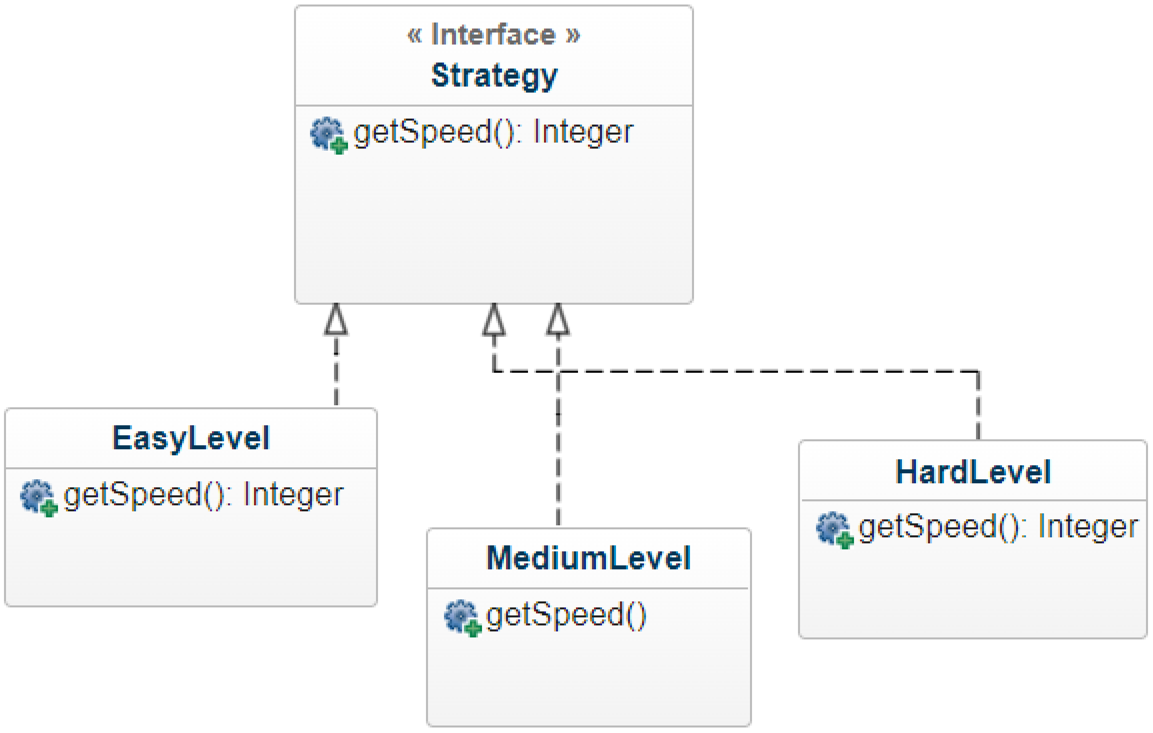
In our code, an ***ObjectIterator*** class was created and this class implements the Iterator interface in the java.util package. Only two methods were overridden and used, ***hasNext*** (which returns a Boolean of whether an element exists in the next position in the list or not) and ***next*** (which returns a GameObject with the index initialized inside the *ObjectIterator* class then increments the index). In *OurWorld*, two iterators were created. ***mIterator*** to traverse through the first half of the ***moving*** Arraylist- containing Plates and Bombs- and the ***nIterator*** to traverse through the second half of the ***moving*** Arraylist after reversing it.



2. Strategy (Behavioral design pattern)

The Strategy Pattern is used to define a family of algorithms (usually that perform the same task), encapsulates each one and makes their objects interchangeable. Switching between algorithms happens at runtime. Different algorithms will be appropriate at different times.

In our code, we used Strategy for different Levels of the game (Easy, Medium, and Hard). The interface ***Strategy*** is implemented by the 3 classes (levels). In each level, the ***getSpeed*** method is implemented differently. It returns a different value for the game speed (Easy having the lowest speed –so, the slowest- and Hard being the fastest). The user chooses from the options which level they want to play and there, the ***EasyLevel***, ***MediumLevel*** or ***HardLevel*** is used as an attribute when creating the circus world.

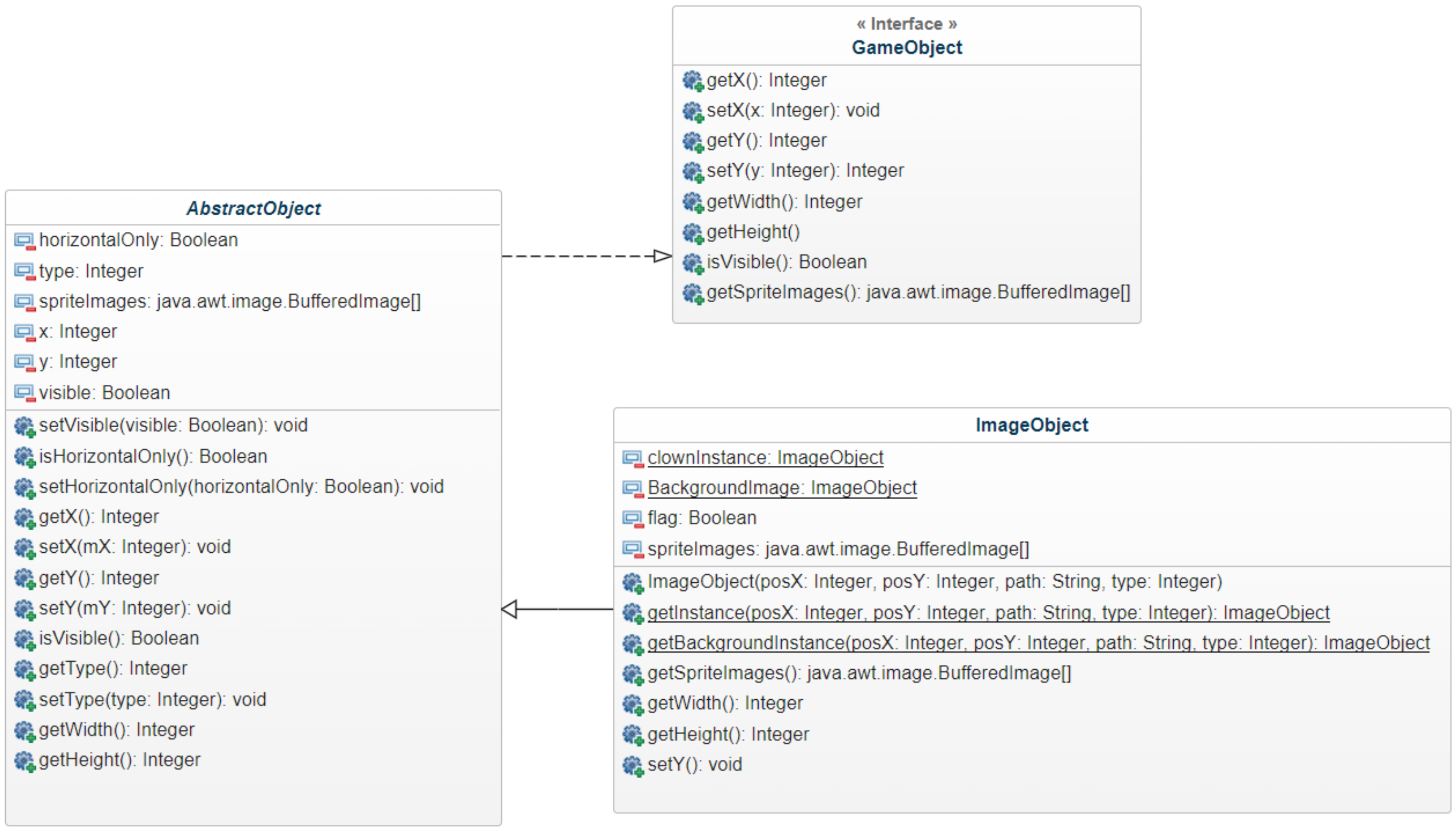


Project By:
Youssef ElKady 7651 Farida Soffar 7720 Salma ElMahy 7668

3. Singleton (Creational design pattern)

The Singleton pattern ensures that a class has just a single instance and provides a global point of access to that instance.

In our code, since we have only 1 clown image and only 1 background image, it is appropriate to use the Singleton. The class ***ImageObject*** extends ***AbstractObject*** which implements the ***GameObject*** interface. In ***Generator***, the method ***getInstance*** returns the ***clownInstance*** to be added as the first element in the ***control*** array list. In ***OurWorld*** constructor, the ***getBackgroundInstance*** that returns ***BackgroundImage*** to be set as the first element in the ***constant*** array list.


« Interface »
GameObject
getX(): Integer
setX(x: Integer): void
getY(): Integer
setY(y: Integer): Integer
getWidth(): Integer
getHeight()
isVisible(): Boolean
getSpriteImages(): java.awt.image.BufferedImage[]
AbstractObject
horizontalOnly: Boolean
type: Integer
spriteImages: java.awt.image.BufferedImage[]
x: Integer
y: Integer
visible: Boolean
setVisible(visible: Boolean): void
isHorizontalOnly(): Boolean
setHorizontalOnly(horizontalOnly: Boolean): void
getX(): Integer
setX(mX: Integer): void
getY(): Integer
setY(mY: Integer): void
isVisible(): Boolean
getType(): Integer
setType(type: Integer): void
getWidth(): Integer
getHeight(): Integer
ImageObject
clownInstance: ImageObject
BackgroundImage: ImageObject
flag: Boolean
spriteImages: java.awt.image.BufferedImage[]
ImageObject(posX: Integer, posY: Integer, path: String, type: Integer)
getInstance(posX: Integer, posY: Integer, path: String, type: Integer): ImageObject
getBackgroundInstance(posX: Integer, posY: Integer, path: String, type: Integer): ImageObject
getSpriteImages(): java.awt.image.BufferedImage[]
getWidth(): Integer
getHeight(): Integer
setY(): void

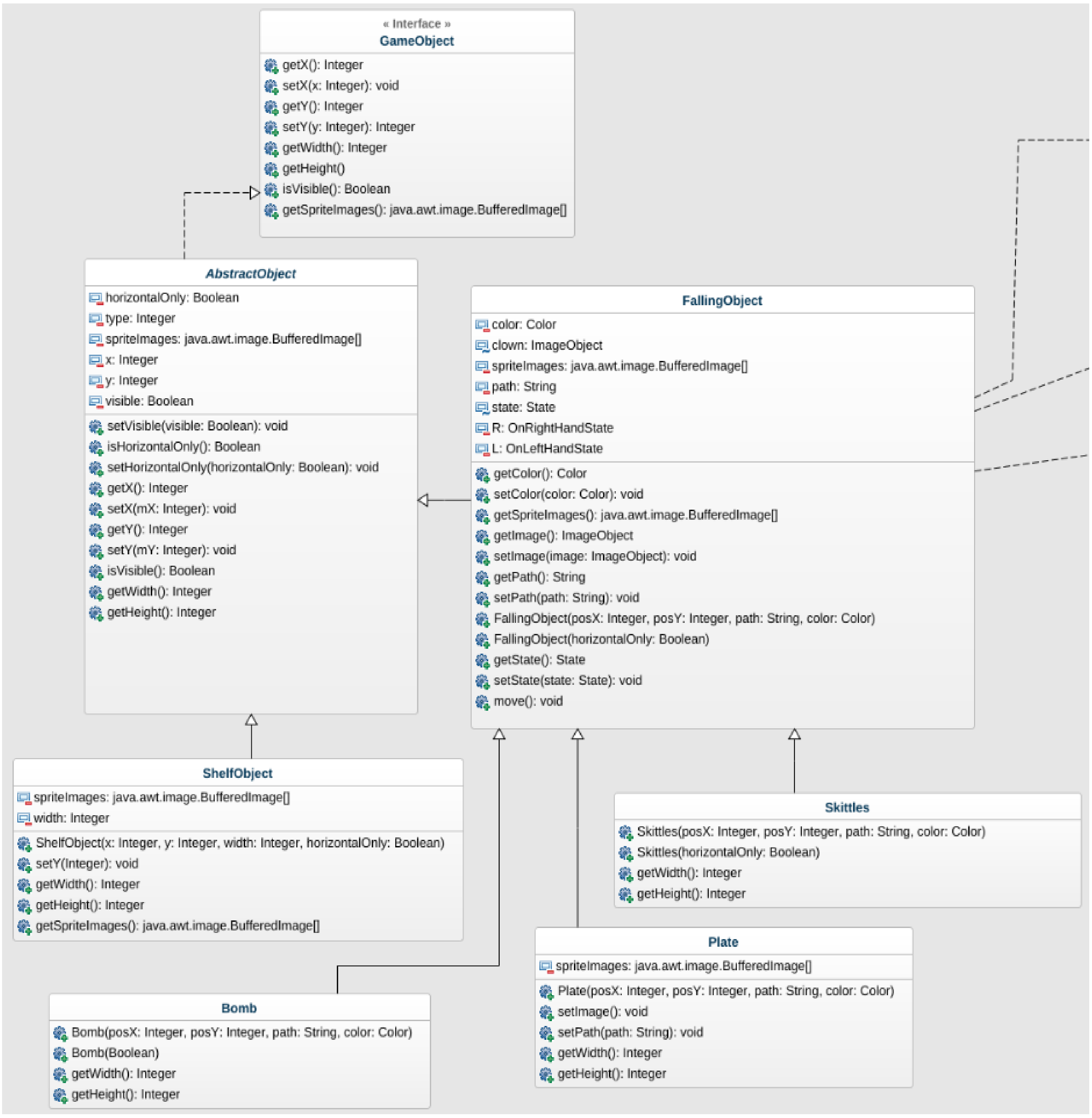
4. Factory (Creational design pattern)

The Factory pattern is considered one of the best ways to create an object. In Factory pattern, we create an object without exposing the creation logic to the user and refer to newly created object using a common interface.

In our code, we created an abstract class which implements the Interface ***GameObject***. This interface is already created in the Game Engine that we used. This class is called ***AbstractObject***. All the following objects then extend this ***AbstractObject*** class. A class for the falling objects (***FallingObjects***), which are the objects that move with time, was created to be then extended by the ***Plate***, ***Bomb*** and ***Skittles*** classes. Another class that extends ***AbstractObject*** is ***ShelfObject***. This class contains all the necessary variables and methods of the shelves that the plates fall from. In the end, a factory class was created named ***ObjectFactory***, which contains all the constructors used in the Game (such as Plate, Shelf, Clown, Bomb and Skittles). The ***ObjectFactory*** class is the class that is instantiated when creating any type of object in the game and all other objects aren't instantiated anywhere. This is seen in the ***Generator*** class. This class is used to generate all the game objects.

Project By:

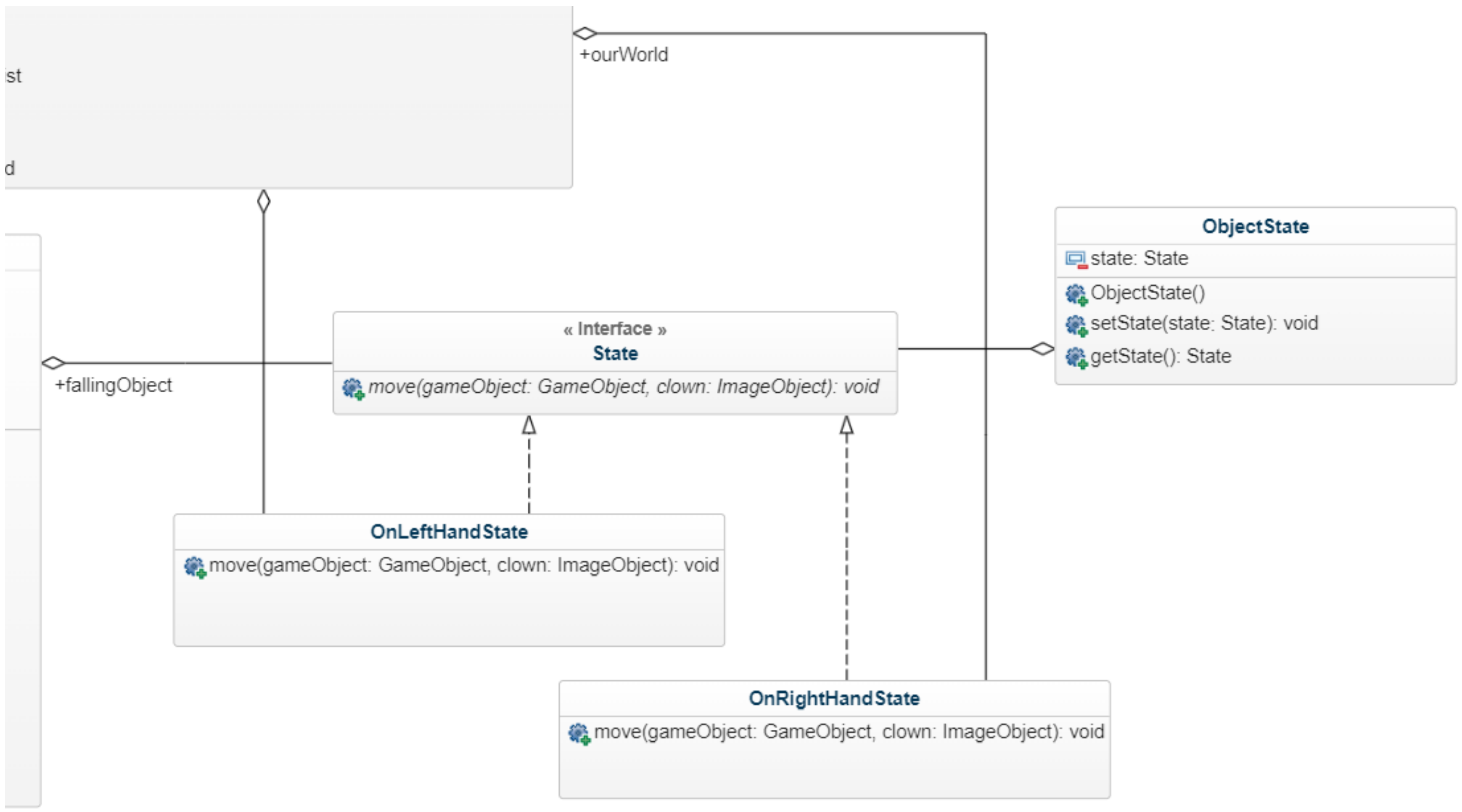
Youssef ElKady 7651 **Farida Soffar 7720** **Salma ElMahy 7668**


« Interface »
GameObject
getX(): Integer
setX(x: Integer): void
getY(): Integer
setY(y: Integer): Integer
getWidth(): Integer
getHeight()
isVisible(): Boolean
getSpriteImages(): java.awt.image.BufferedImage[]
AbstractObject
horizontalOnly: Boolean
type: Integer
spriteImages: java.awt.image.BufferedImage[]
x: Integer
y: Integer
visible: Boolean
setVisible(visible: Boolean): void
isHorizontalOnly(): Boolean
setHorizontalOnly(horizontalOnly: Boolean): void
getX(): Integer
setX(mX: Integer): void
getY(): Integer
setY(mY: Integer): void
isVisible(): Boolean
getWidth(): Integer
getHeight(): Integer
FallingObject
color: Color
clown: ImageObject
spriteImages: java.awt.image.BufferedImage[]
path: String
state: State
R: OnRightHandState
L: OnLeftHandState
getColor(): Color
setColor(color: Color): void
getSpriteImages(): java.awt.image.BufferedImage[]
getImage(): ImageObject
setImage(image: ImageObject): void
getPath(): String
setPath(path: String): void
FallingObject(posX: Integer, posY: Integer, path: String, color: Color)
FallingObject(horizontalOnly: Boolean)
getState(): State
setState(state: State): void
move(): void
ShelfObject
spriteImages: java.awt.image.BufferedImage[]
width: Integer
ShelfObject(x: Integer, y: Integer, width: Integer, horizontalOnly: Boolean)
setY(Integer): void
getWidth(): Integer
getHeight(): Integer
getSpriteImages(): java.awt.image.BufferedImage[]
Skittles
Skittles(posX: Integer, posY: Integer, path: String, color: Color)
Skittles(horizontalOnly: Boolean)
getWidth(): Integer
getHeight(): Integer
Plate
spriteImages: java.awt.image.BufferedImage[]
Plate(posX: Integer, posY: Integer, path: String, color: Color)
setImage(): void
setPath(path: String): void
getWidth(): Integer
getHeight(): Integer
Bomb
Bomb(posX: Integer, posY: Integer, path: String, color: Color)
Bomb(Boolean)
getWidth(): Integer
getHeight(): Integer

5. State (Behavioral design pattern)

In **State** pattern, a class behavior changes based on its state. In State pattern, we create objects which represent various states and a context object whose behavior varies as its state object changes.

In our code, we used the State design principle by creating a ***State*** interface that contains an abstract method called ***move***. Then we created 2 other classes named ***OnLeftHandState*** and ***OnRightHandState*** that implement this ***State*** interface and override its ***move*** method. Since each hand of the clown needs a separate move method, these 2 classes were created and instantiated in the ***OurWorld*** class where it is used and in the ***Plates*** Class. Another class created is called ***ObjectState***, this class contains getters and setters for the ***state*** attribute found in the ***FallingObject*** class and sets the initial state to null. In the ***Refresh*** method inside the ***OurWorld*** Class, the state of the falling object is updated accordingly, depending on whether the falling object touched the Left or Right hand of the clown.

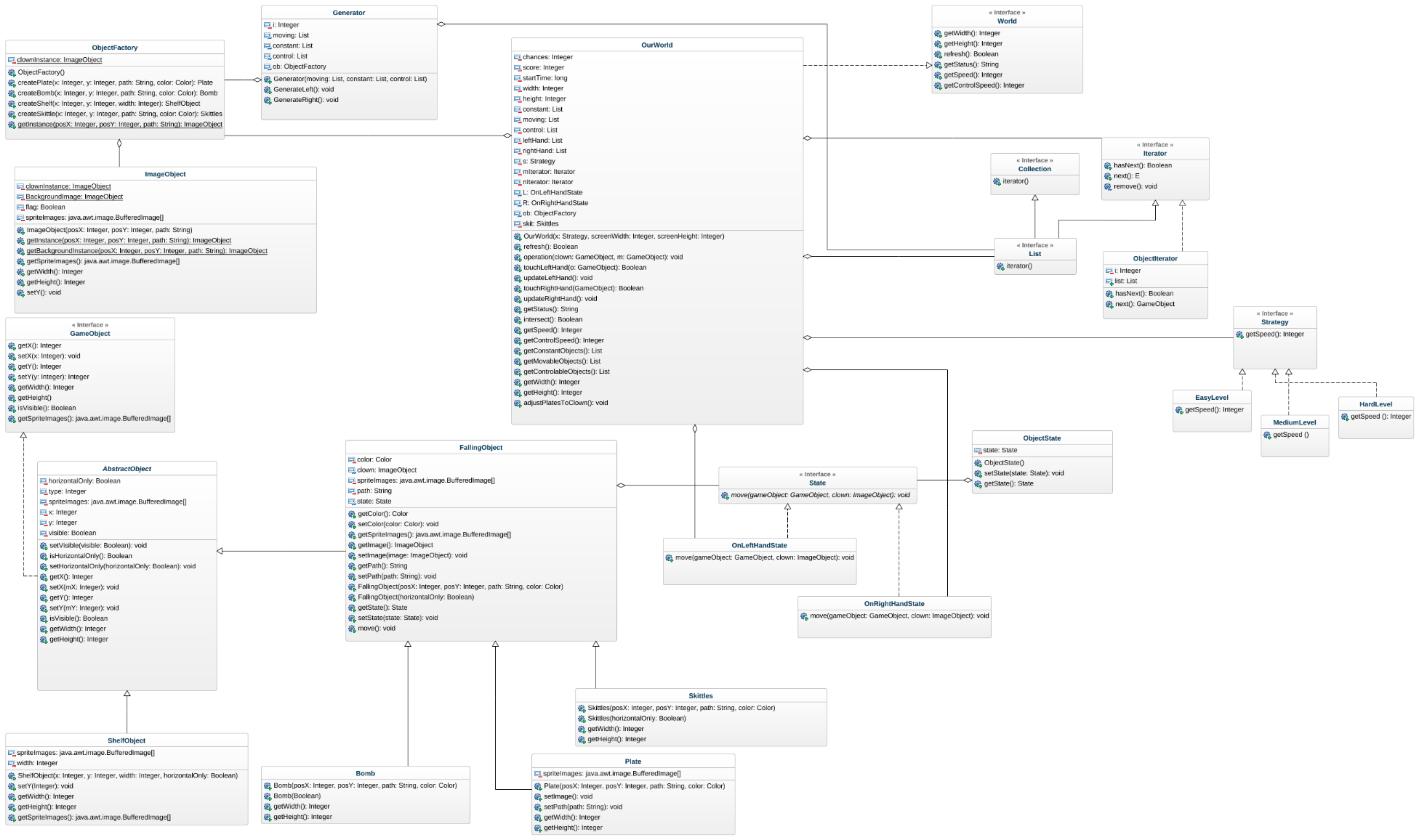

st
d
+ourWorld
+fallingObject
« Interface »
State
move(gameObject: GameObject, clown: ImageObject): void
ObjectState
state: State
ObjectState()
setState(state: State): void
getState(): State
OnLeftHandState
move(gameObject: GameObject, clown: ImageObject): void
OnRightHandState
move(gameObject: GameObject, clown: ImageObject): void

Project By:

Youssef ElKady 7651 **Farida Soffar 7720** **Salma ElMahy 7668**

<u>Game Class Diagram:</u>

Link: https://app.genmymodel.com/api/projects/_-ywz8In-Ee29ie0vpi-P5A/diagrams/_-yxbAon-Ee29ie0vpi-P5A/svg


Generator
i: Integer
moving: List
constant: List
control: List
ob: ObjectFactory
Generator(moving: List, constant: List, control: List)
GenerateLeft(): void
GenerateRight(): void
ObjectFactory
clownInstance: ImageObject
ObjectFactory()
createPlate(x: Integer, y: Integer, path: String, color: Color): Plate
createBomb(x: Integer, y: Integer, path: String, color: Color): Bomb
createShelf(x: Integer, y: Integer, width: Integer): ShelfObject
createSkittle(x: Integer, y: Integer, path: String, color: Color): Skittles
getInstance(posX: Integer, posY: Integer, path: String): ImageObject
ImageObject
clownInstance: ImageObject
BackgroundImage: ImageObject
flag: Boolean
spriteImages: java.awt.image.BufferedImage[]
ImageObject(posX: Integer, posY: Integer, path: String)
getInstance(posX: Integer, posY: Integer, path: String): ImageObject
getBackgroundInstance(posX: Integer, posY: Integer, path: String): ImageObject
getSpriteImages(): java.awt.image.BufferedImage[]
getWidth(): Integer
getHeight(): Integer
setY(): void
« Interface »
GameObject
getX(): Integer
setX(x: Integer): void
getY(): Integer
setY(y: Integer): Integer
getWidth(): Integer
getHeight()
isVisible(): Boolean
getSpriteImages(): java.awt.image.BufferedImage[]
OurWorld
chances: Integer
score: Integer
startTime: long
width: Integer
height: Integer
constant: List
moving: List
control: List
leftHand: List
rightHand: List
s: Strategy
mIterator: Iterator
nIterator: Iterator
L: OnLeftHandState
R: OnRightHandState
ob: ObjectFactory
skit: Skittles
OurWorld(x: Strategy, screenWidth: Integer, screenHeight: Integer)
refresh(): Boolean
operation(clown: GameObject, m: GameObject): void
touchLeftHand(o: GameObject): Boolean
updateLeftHand(): void
touchRightHand(GameObject): Boolean
updateRightHand(): void
getStatus(): String
intersect(): Boolean
getSpeed(): Integer
getControlSpeed(): Integer
getConstantObjects(): List
getMovableObjects(): List
getControlableObjects(): List
getWidth(): Integer
getHeight(): Integer
adjustPlatesToClown(): void
« Interface »
World
getWidth(): Integer
getHeight(): Integer
refresh(): Boolean
getStatus(): String
getSpeed(): Integer
getControlSpeed(): Integer
« Interface »
Collection
iterator()
« Interface »
Iterator
hasNext(): Boolean
next(): E
remove(): void
« Interface »
List
iterator()
ObjectIterator
i: Integer
list: List
hasNext(): Boolean
next(): GameObject
« Interface »
Strategy
getSpeed(): Integer
EasyLevel
getSpeed(): Integer
MediumLevel
getSpeed ()
HardLevel
getSpeed (): Integer
ObjectState
state: State
ObjectState()
setState(state: State): void
getState(): State
« Interface »
State
move(gameObject: GameObject, clown: ImageObject): void
OnLeftHandState
move(gameObject: GameObject, clown: ImageObject): void
OnRightHandState
move(gameObject: GameObject, clown: ImageObject): void
AbstractObject
horizontalOnly: Boolean
type: Integer
spriteImages: java.awt.image.BufferedImage[]
x: Integer
y: Integer
visible: Boolean
setVisible(visible: Boolean): void
isHorizontalOnly(): Boolean
setHorizontalOnly(horizontalOnly: Boolean): void
getX(): Integer
setX(mX: Integer): void
getY(): Integer
setY(mY: Integer): void
isVisible(): Boolean
getWidth(): Integer
getHeight(): Integer
FallingObject
color: Color
clown: ImageObject
spriteImages: java.awt.image.BufferedImage[]
path: String
state: State
getColor(): Color
setColor(color: Color): void
getSpriteImages(): java.awt.image.BufferedImage[]
getImage(): ImageObject
setImage(image: ImageObject): void
getPath(): String
setPath(path: String): void
FallingObject(posX: Integer, posY: Integer, path: String, color: Color)
FallingObject(horizontalOnly: Boolean)
getState(): State
setState(state: State): void
move(): void
Skittles
Skittles(posX: Integer, posY: Integer, path: String, color: Color)
Skittles(horizontalOnly: Boolean)
getWidth(): Integer
getHeight(): Integer
ShelfObject
spriteImages: java.awt.image.BufferedImage[]
width: Integer
ShelfObject(x: Integer, y: Integer, width: Integer, horizontalOnly: Boolean)
setY(Integer): void
getWidth(): Integer
getHeight(): Integer
getSpriteImages(): java.awt.image.BufferedImage[]
Bomb
Bomb(posX: Integer, posY: Integer, path: String, color: Color)
Bomb(Boolean)
getWidth(): Integer
getHeight(): Integer
Plate
spriteImages: java.awt.image.BufferedImage[]
Plate(posX: Integer, posY: Integer, path: String, color: Color)
setImage(): void
setPath(path: String): void
getWidth(): Integer
getHeight(): Integer