Final project

Programming-II

Mohamed Ramadan Mohsen 7505

Nour Mohamed Mahmoud 7591

Omar Ahmed Mohamed 7461

Mostafa Islam Mostafa 7644

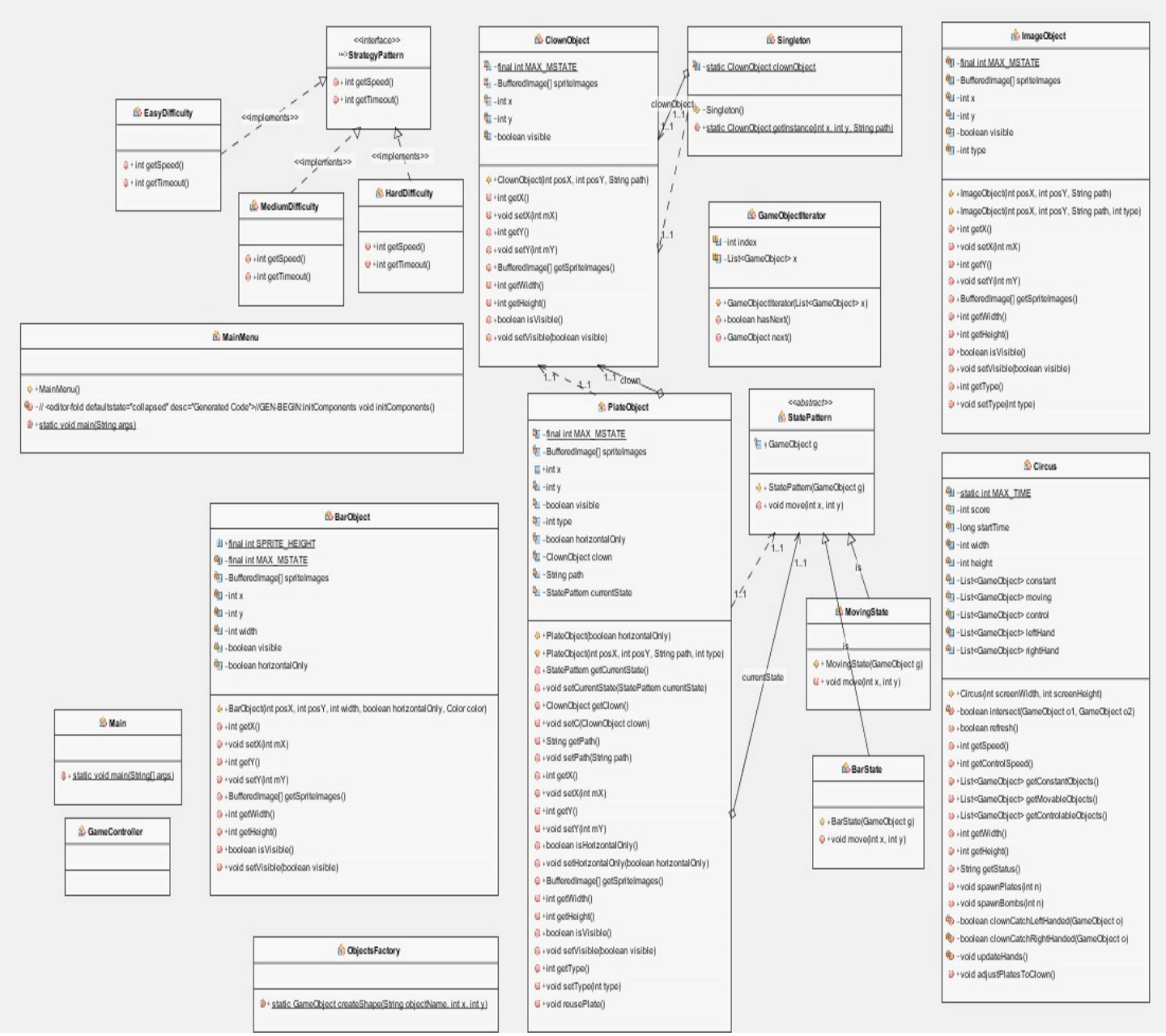
Submitted To:

ENG. Tarek

Due to:

Jan ,4th, 2023

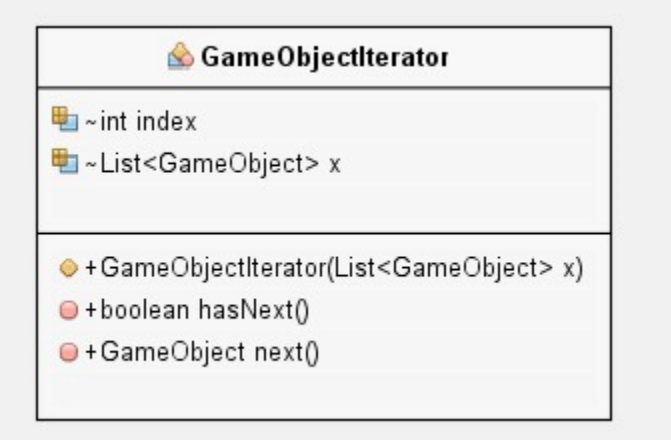
Class diagram:



Pattern Designs:

1- Iterator

an iterator is an object that allows you to traverse through a collection, such as a list or a set, and access the elements of the collection one by one. It is called an iterator because it "iterates" over the elements of the collection. An iterator is created by implementing the Iterator interface, which defines the following methods: boolean hasNext(): Returns true if the iterator has more elements to visit, false otherwise. E next(): Returns the next element in the iteration and advances the iterator. void remove(): Removes the last element returned by the iterator from the collection.Used in class named **GameObjectIterator.**

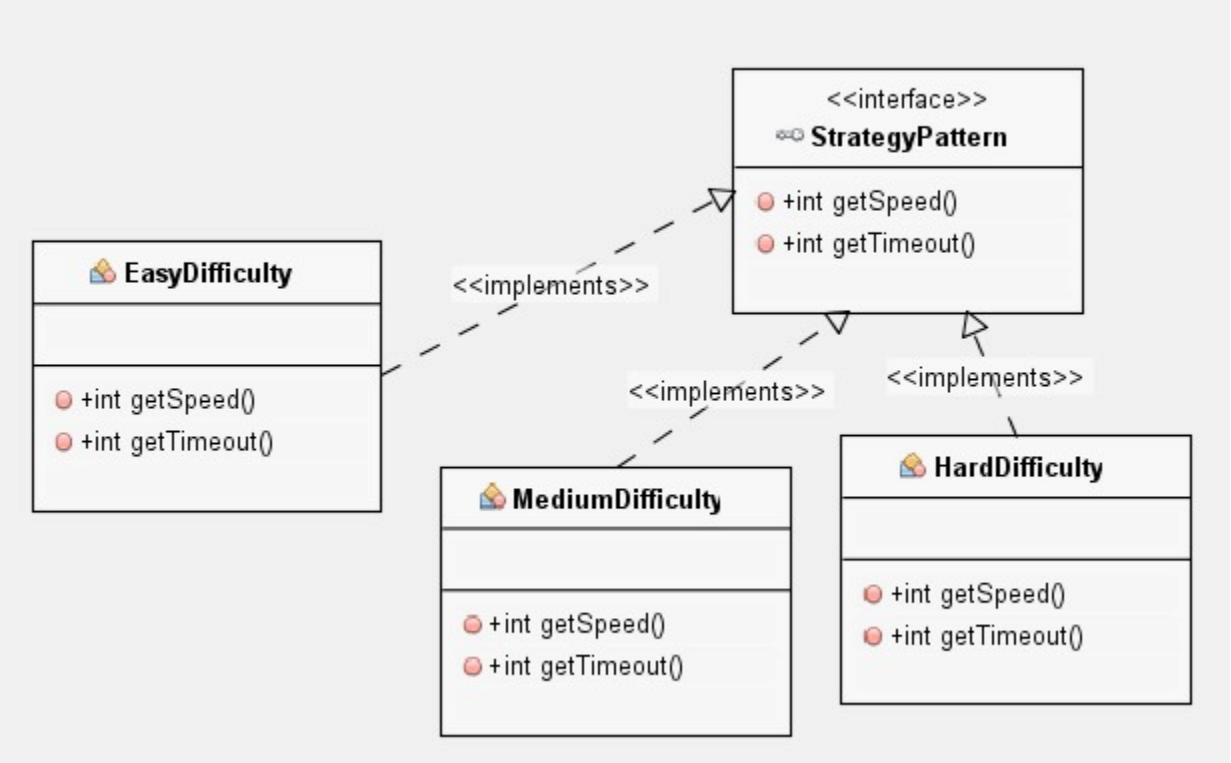


2- Strategy

In the context of software design, a strategy is a behavioral design pattern that enables an algorithm's behavior to be selected at runtime. The strategy pattern defines a family of algorithms, encapsulates each algorithm, and makes the algorithms interchangeable

The idea behind the strategy pattern is to allow the client code to choose from a range of strategies, or algorithms, at runtime, rather than having the algorithm hard-coded into the client. This allows the client to be more flexible and adaptable, because it can choose the most appropriate algorithm for a given situation. The strategy pattern was used to implement

three different levels in the Circus of Plates game: Easy, Medium, and Hard. For each “strategy”, getSpeed() method is implemented



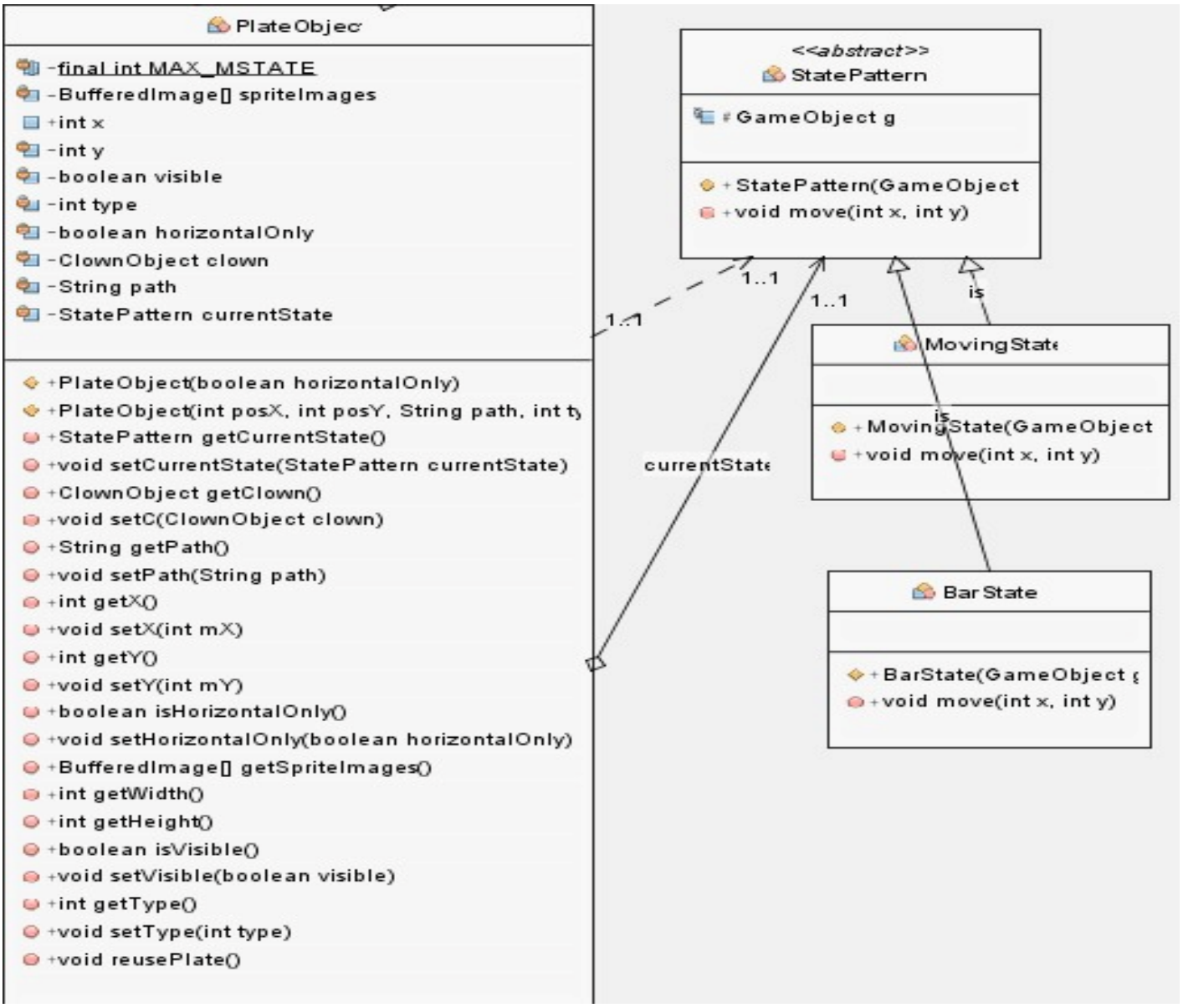
3- State

the term "state" usually refers to the values of variables that an object holds at a given time. For example, if you have a class representing a person, that class might have instance variables for the person's name, age, and height. The state of an object of that class would be the current values of those variables

In object-oriented programming, objects often have behavior as well as state. The behavior of an object is implemented as methods, which are functions associated with the object that can operate on its state. For example, a person object might have a method called "setAge" that allows you to change the person's age. When you call this method, you are changing the state of the object.

It's important to understand that the state of an object can change over time, as the object's behavior is invoked and modifies its variables. The state of an object at any given time is determined by the values of its variables at that time.

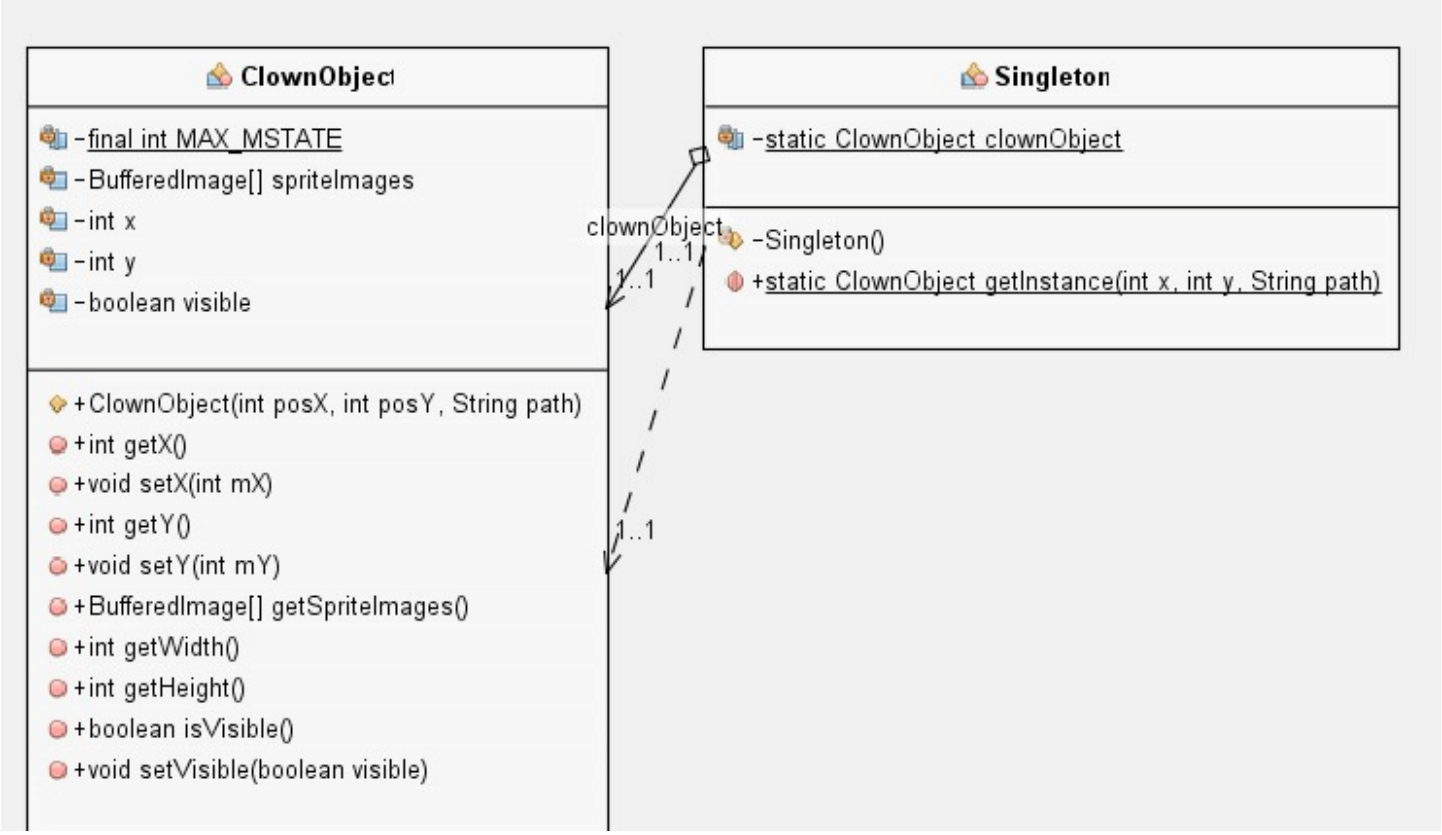
Each time a new ShapeObject is created, its initial state is set to BarState, and whenever the ShapeObject is no longer on the shelf, its state is set to MovingState. For an object with BarState, move() only moves the shape horizontally, while for a MovingState object, the shape moves both horizontally and vertically.


PlateObjec
-final int MAX_MSTATE
-BufferedImage[] spriteImages
+int x
-int y
-boolean visible
-int type
-boolean horizontalOnly
-ClownObject clown
-String path
-StatePattern currentState
+PlateObject(boolean horizontalOnly)
+PlateObject(int posX, int posY, String path, int ty
+StatePattern getCurrentState()
+void setCurrentState(StatePattern currentState)
+ClownObject getClown()
+void setC(ClownObject clown)
+String getPath()
+void setPath(String path)
+int getX()
+void setX(int mX)
+int getY()
+void setY(int mY)
+boolean isHorizontalOnly()
+void setHorizontalOnly(boolean horizontalOnly)
+BufferedImage[] getSpriteImages()
+int getWidth()
+int getHeight()
+boolean isVisible()
+void setVisible(boolean visible)
+int getType()
+void setType(int type)
+void reusePlate()
<<abstract>>
StatePattern
#GameObject g
+StatePattern(GameObject
+void move(int x, int y)
1..1
1..1
1..1
is
is
currentState
MovingState
+MovingState(GameObject
+void move(int x, int y)
BarState
+BarState(GameObject g
+void move(int x, int y)

4- Singleton

A singleton is a design pattern that allows you to ensure that a class has only one instance, while providing a global access point to this instance.

Since there is only one clown in Circus of Plates, it was appropriate to use the singleton design pattern. To get the only instance of the clown, ClownObject class has a public static method, getInstance().



5- Factory

a factory is a design pattern that involves creating an object in a super class, but letting subclasses alter the type of objects that will be created whose getShape() method returns a different object depending on the type of its given arguments. The GameObject class used to generate all the falling shapes