Welcome to the Class

Department of Computing and Information System

Design pattern

Md. Selim Hossain
Senior Lecturer
Department of Computing and Information System
Daffodil International University (DIU),Dhaka, Bangladesh

Design Pattern

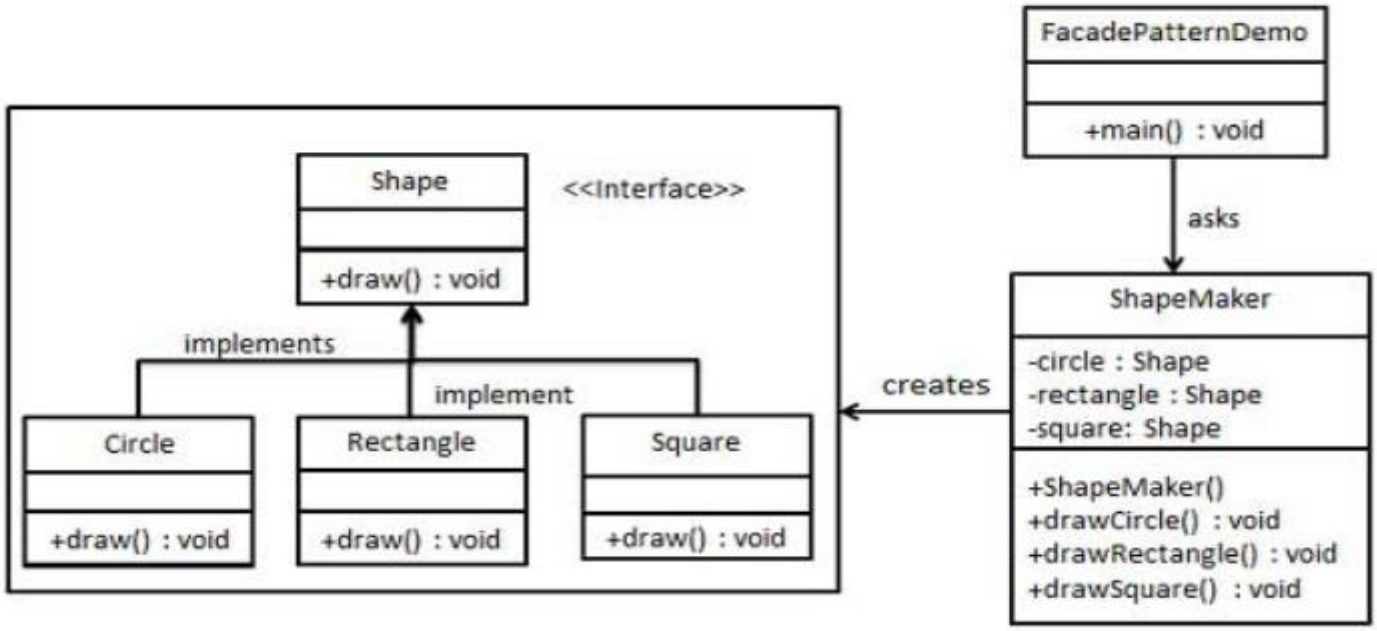
Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to existing system to hide its complexities.

This pattern involves a single class which provides simplified methods required by client and delegates calls to methods of existing system classes.

Implementation

We are going to create a *Shape* interface and concrete classes implementing the *Shape* interface. A facade class *ShapeMaker* is defined as a next step.

ShapeMaker class uses the concrete classes to delegate user calls to these classes. *FacadePatternDemo*, our demo class, will use *ShapeMaker* class to show the results.



Design Patterns - Strategy Pattern

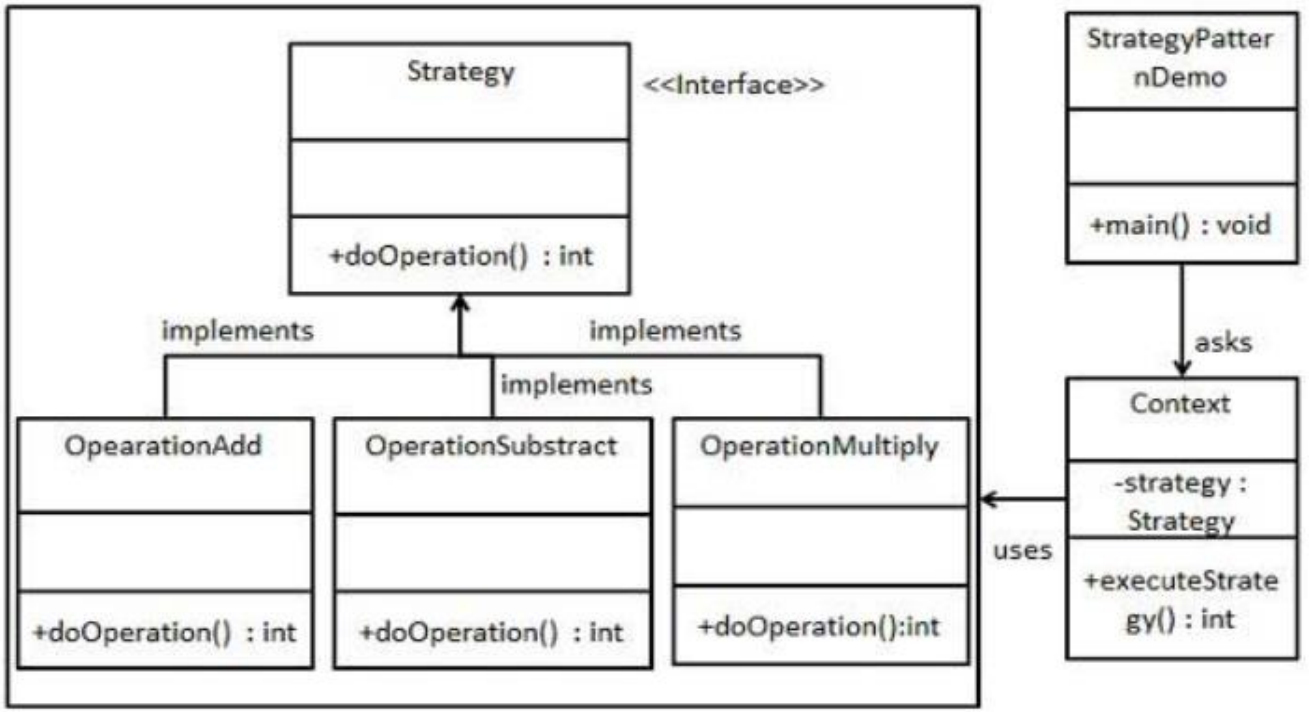
In Strategy pattern, a class behavior or its algorithm can be changed at run time. This type of design pattern comes under behavior pattern.

In Strategy pattern, we create objects which represent various strategies and a context object whose behavior varies as per its strategy object. The strategy object changes the executing algorithm of the context object.

Implementation

We are going to create a *Strategy* interface defining an action and concrete strategy classes implementing the *Strategy* interface. *Context* is a class which uses a Strategy.

StrategyPatternDemo, our demo class, will use *Context* and strategy objects to demonstrate change in Context behaviour based on strategy it deploys or uses.



Verify the output.

```
10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
```

Flyweight pattern is primarily used to reduce the number of objects created and to decrease memory footprint and increase performance. This type of design pattern comes under structural pattern as this pattern provides ways to decrease object count thus improving the object structure of application.

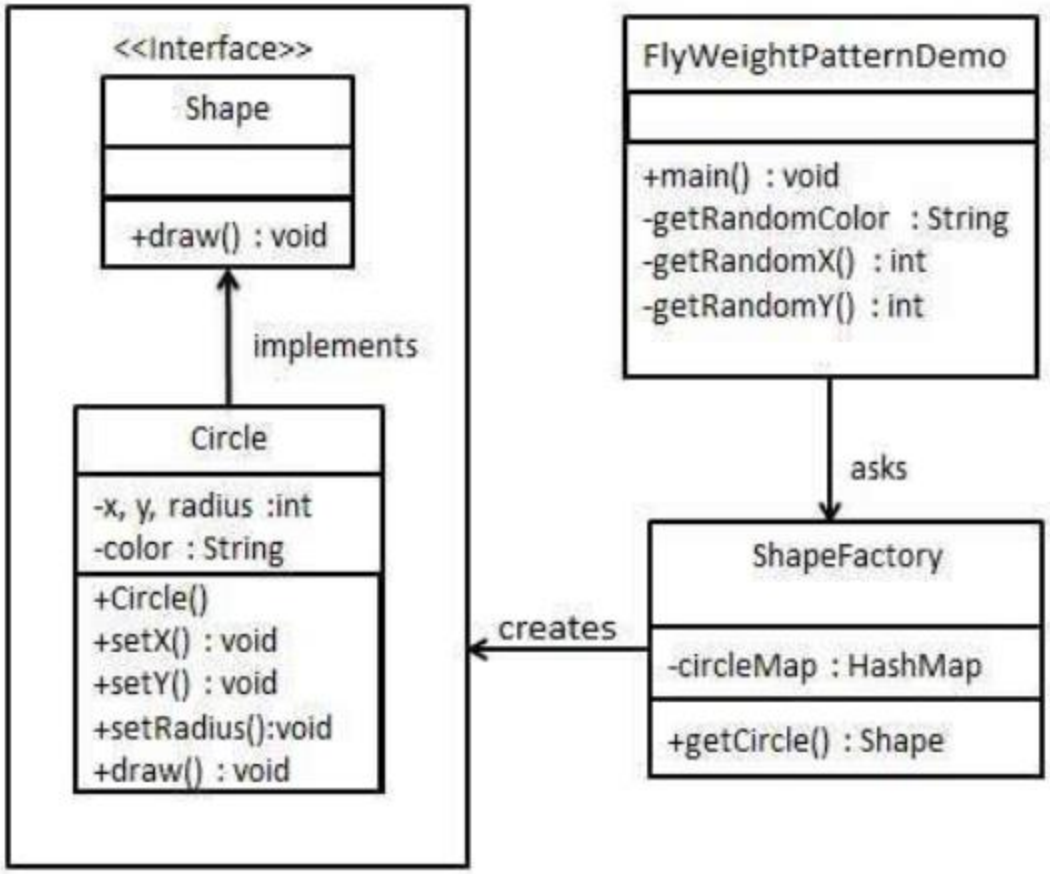
Flyweight pattern tries to reuse already existing similar kind objects by storing them and creates new object when no matching object is found. We will demonstrate this pattern by drawing 20 circles of different locations but we will create only 5 objects. Only 5 colors are available so color property is used to check already existing *Circle* objects.

Implementation

We are going to create a *Shape* interface and concrete class *Circle* implementing the *Shape* interface. A factory class *ShapeFactory* is defined as a next step.

ShapeFactory has a *HashMap* of *Circle* having key as color of the *Circle* object. Whenever a request comes to create a circle of particular color to *ShapeFactory*, it checks the circle object in its *HashMap*, if object of *Circle* found, that object is returned otherwise a new object is created, stored in hashmap for future use, and returned to client.

FlyWeightPatternDemo, our demo class, will use *ShapeFactory* to get a *Shape* object. It will pass information (*red / green / blue/ black / white*) to *ShapeFactory* to get the circle of desired color it needs.


<<Interface>>
Shape
+draw() : void
implements
Circle
-x, y, radius :int
-color : String
+Circle()
+setX() : void
+setY() : void
+setRadius():void
+draw() : void
FlyWeightPatternDemo
+main() : void
-getRandomColor : String
-getRandomX() : int
-getRandomY() : int
asks
ShapeFactory
-circleMap : HashMap
+getCircle() : Shape
creates

Thanks to All