Facade Pattern:

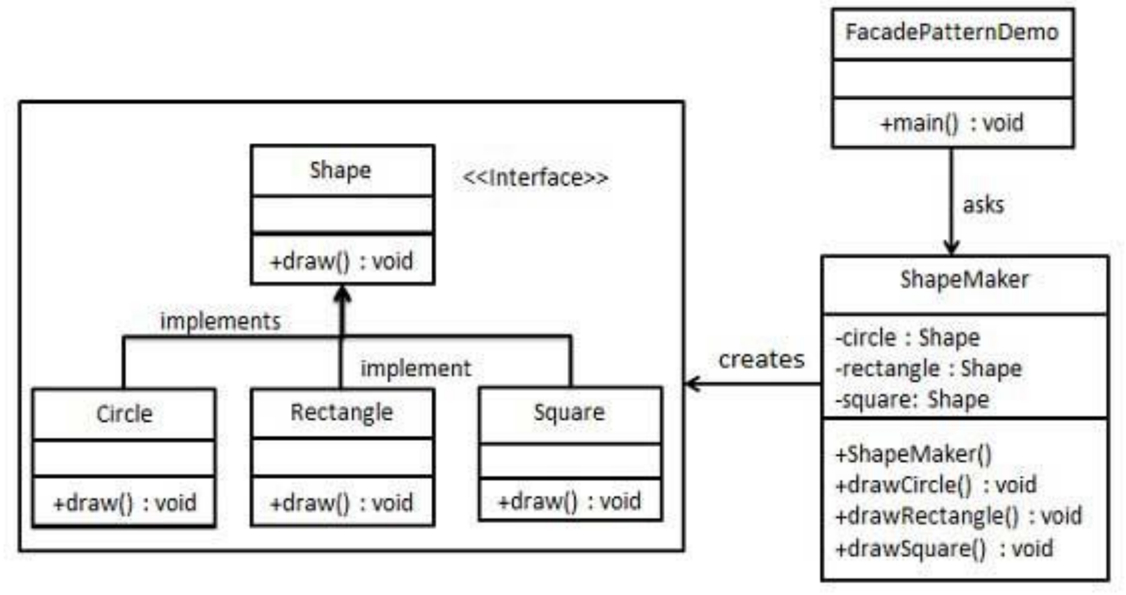
Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to existing system to hide its complexities.

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.



Step 1

Create an interface.

Shape.java

```
public interface Shape {
  void draw();
}
```

Step 2

Create concrete classes implementing the same interface.

Rectangle.java

```
public class Rectangle implements Shape {

  @Override
  public void draw() {
    System.out.println("Rectangle::draw()");
  }
}
```

Square.java

```
public class Square implements Shape {

  @Override
  public void draw() {
    System.out.println("Square::draw()");
  }
}
```

Circle.java

```
public class Circle implements Shape {

  @Override
  public void draw() {
    System.out.println("Circle::draw()");
  }
}
```

Step 3

Create a facade class.

ShapeMaker.java

```
public class ShapeMaker {
  private Shape circle;
  private Shape rectangle;
  private Shape square;
public ShapeMaker() {
      circle = new Circle();
      rectangle = new Rectangle();
      square = new Square();
  }

  public void drawCircle(){
      circle.draw();
  }
  public void drawRectangle(){
      rectangle.draw();
  }
  public void drawSquare(){
      square.draw();
  }
}
```

Step 4

Use the facade to draw various types of shapes.

FacadePatternDemo.java

```
public class FacadePatternDemo {
   public static void main(String[] args) {
      ShapeMaker shapeMaker = new ShapeMaker();

      shapeMaker.drawCircle();
      shapeMaker.drawRectangle();
      shapeMaker.drawSquare();
   }
}
```

Step 5

Verify the output.

```
Circle::draw()
Rectangle::draw()
Square::draw()
```

Proxy Pattern:

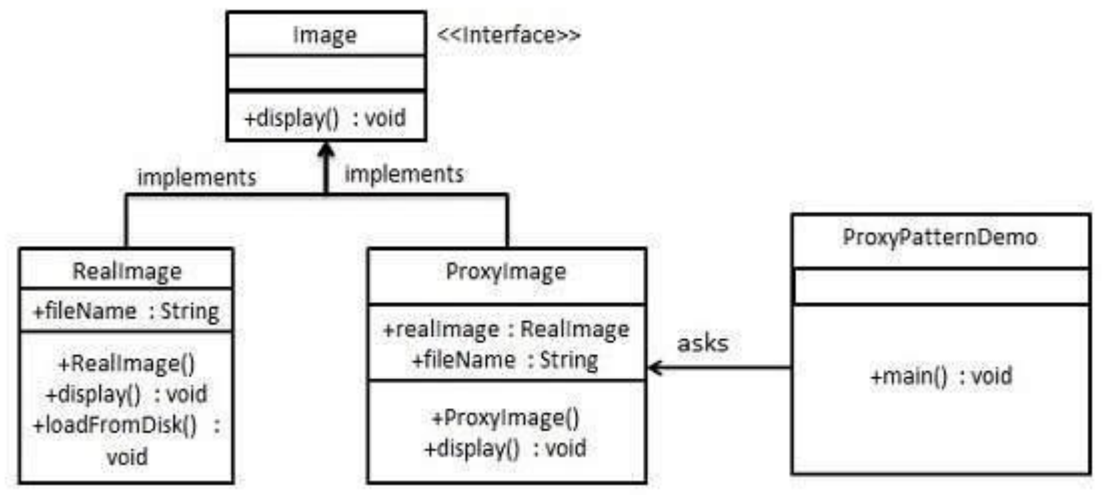
In proxy pattern, a class represents functionality of another class. This type of design pattern comes under structural pattern.

In proxy pattern, we create object having original object to interface its functionality to outer world.

Implementation

We are going to create an *Image* interface and concrete classes implementing the *Image* interface. *ProxyImage* is a proxy class to reduce memory footprint of *RealImage* object loading.

ProxyPatternDemo, our demo class, will use *ProxyImage* to get an *Image* object to load and display as it needs.



Step 1

Create an interface.

Image.java

```
public interface Image {
  void display();
}
```

Step 2

Create concrete classes implementing the same interface.

RealImage.java

```
public class RealImage implements Image {

   private String fileName;

   public RealImage(String fileName){
      this.fileName = fileName;
      loadFromDisk(fileName);
   }

   @Override
   public void display() {
      System.out.println("Displaying " + fileName);
   }

   private void loadFromDisk(String fileName){
      System.out.println("Loading " + fileName);
   }
}
```

ProxyImage.java

```
public class ProxyImage implements Image{

   private RealImage realImage;
   private String fileName;

   public ProxyImage(String fileName){
      this.fileName = fileName;
   }

   @Override
   public void display() {
      if(realImage == null){
         realImage = new RealImage(fileName);
      }
      realImage.display();
   }
}
```

Step 3

Use the *ProxyImage* to get object of *RealImage* class when required.

ProxyPatternDemo.java

```
public class ProxyPatternDemo {

   public static void main(String[] args) {
      Image image = new ProxyImage("test_10mb.jpg");

      //image will be loaded from disk
      image.display();
      System.out.println("");

      //image will not be loaded from disk
      image.display();
   }
}
```

Step 4

Verify the output.

```
Loading test_10mb.jpg
Displaying test_10mb.jpg

Displaying test_10mb.jpg
```