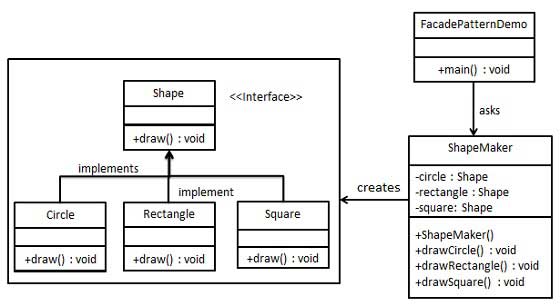
Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to existing system to hide its complexities.

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

\*/

interface Shape{

void draw();

}

/\* step 2

\* create concrete classes implementing the same interface

\* Rectangle.java

\*/

class Rectangle implements Shape{

@Override

public void draw() {

System.out.println("rectangle draw");

}

}

//Square

class Square implements Shape{

@Override

public void draw() {

System.out.println("square draw");

}

}

//circle

class Circle implements Shape{

@Override

public void draw() {

System.out.println("circle draw");

}

}

//create a facade class

//shpaeMaker.class

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();

}

}

public class TestFacade {

public static void main(String[] args) {

ShapeMaker shapeMaker = new ShapeMaker();

shapeMaker.drawCircle();

shapeMaker.drawRectangle();

shapeMaker.drawSquare();

}

}