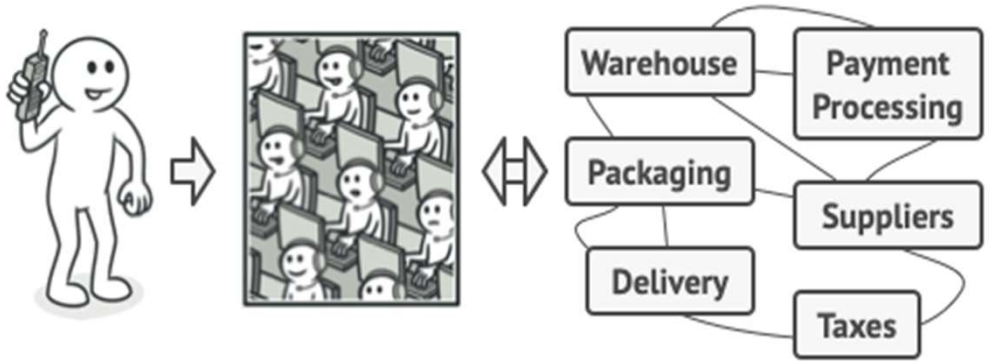
Facade Design Pattern

Real World Analogy



Placing orders by phone.

Working

When you call a shop to place a phone order, an operator is your facade to all services and departments of the shop.

The operator provides you with a simple voice interface to the ordering system, payment gateways, and various delivery services.

Facade 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.

Applicability

Use the Facade pattern when you need to have a limited but straightforward interface to a complex subsystem.

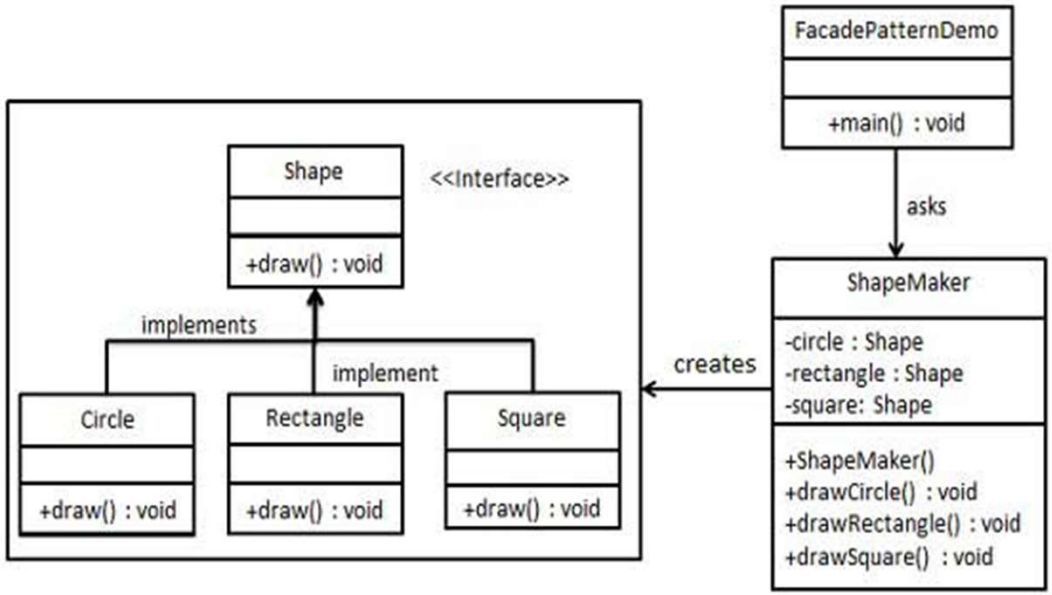
Use the Facade when you want to structure a subsystem into layers.

Pros and Cons

✓ You can isolate your code from the complexity of a subsystem.

✗ A facade can become **a god object** coupled to all classes of an app.

Example


FacadePatternDemo
+main() : void
asks
ShapeMaker
-circle : Shape
-rectangle : Shape
-square: Shape
+ShapeMaker()
+drawCircle() : void
+drawRectangle() : void
+drawSquare() : void
creates
Shape
<<Interface>>
+draw() : void
implements
implement
Circle
+draw() : void
Rectangle
+draw() : void
Square
+draw() : void

Create an interface.

Shape.java

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

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()");
   }
}
```

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();
   }
}
```

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();
   }
}
```

Verify the output.

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