Facade

Lia Cardoso - 107548

April 24th, 2024

When should we use this pattern?

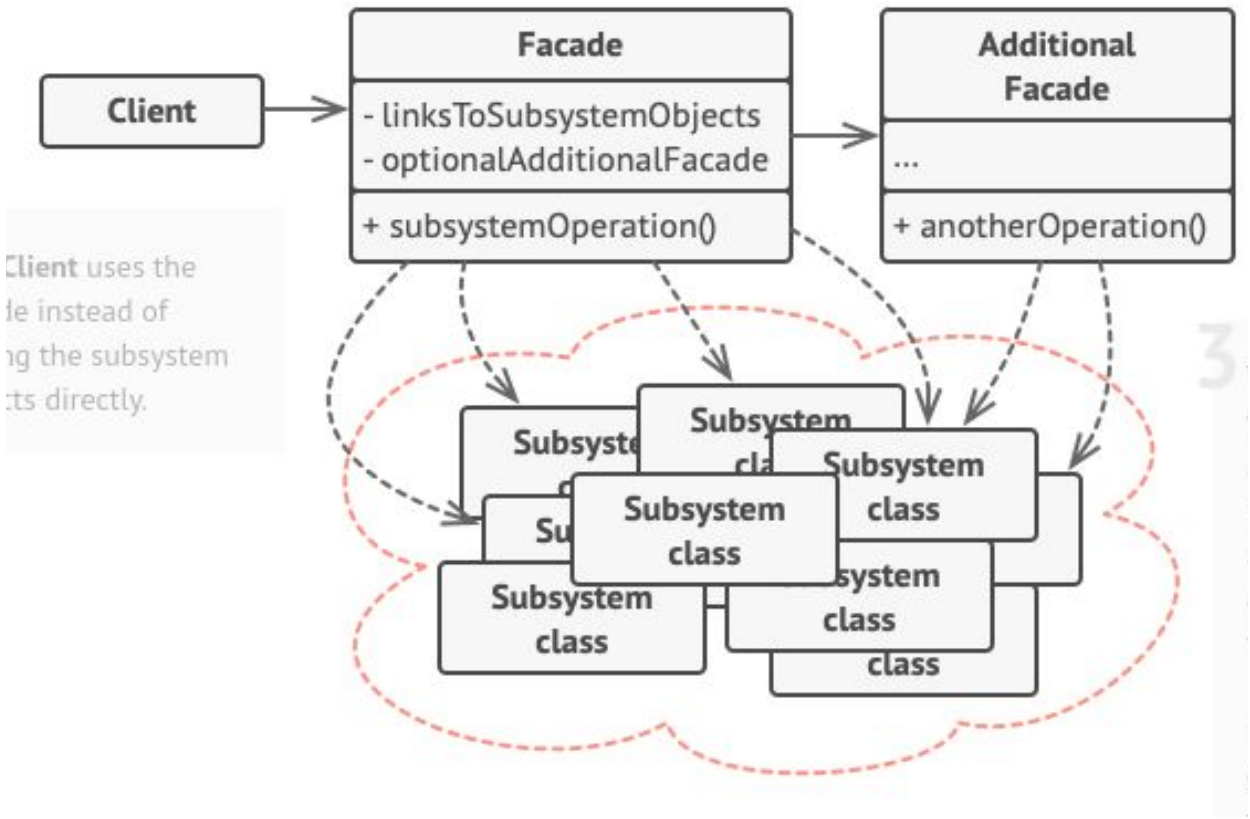
- When we need you need to have a limited but straightforward interface to a complex to a complex system;
  -> In simpler terms:
  - It creates a shortcut to the most used features

- When you want to structure a subsystem into layers;
  -> In simpler terms:
  - We define entry points for each level of the subsystem

How to implement this pattern?

1. Simplify the interface for an existing subsystem, making client code less dependent on the subsystem's classes.
2. Implement this simpler interface in a new facade class that manages subsystem initialization and lifecycle, redirecting client calls to the subsystem.
3. Ensure all client interactions with the subsystem are through the facade to protect against changes in the subsystem, such as upgrades.
4. Split the facade into smaller facades if it becomes overly complex.

(Base) Class Structure



- Client: uses the facade;
- Facade: provides convenient access to a particular part of the subsystem's functionality;
- Additional Facade: can be created to prevent polluting a single facade with unrelated features that might make it yet another complex structure;
- Complex Subsystem: consists of dozens of various objects;

Code Example(s)

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()");
   }
}
```

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();
   }
}
```