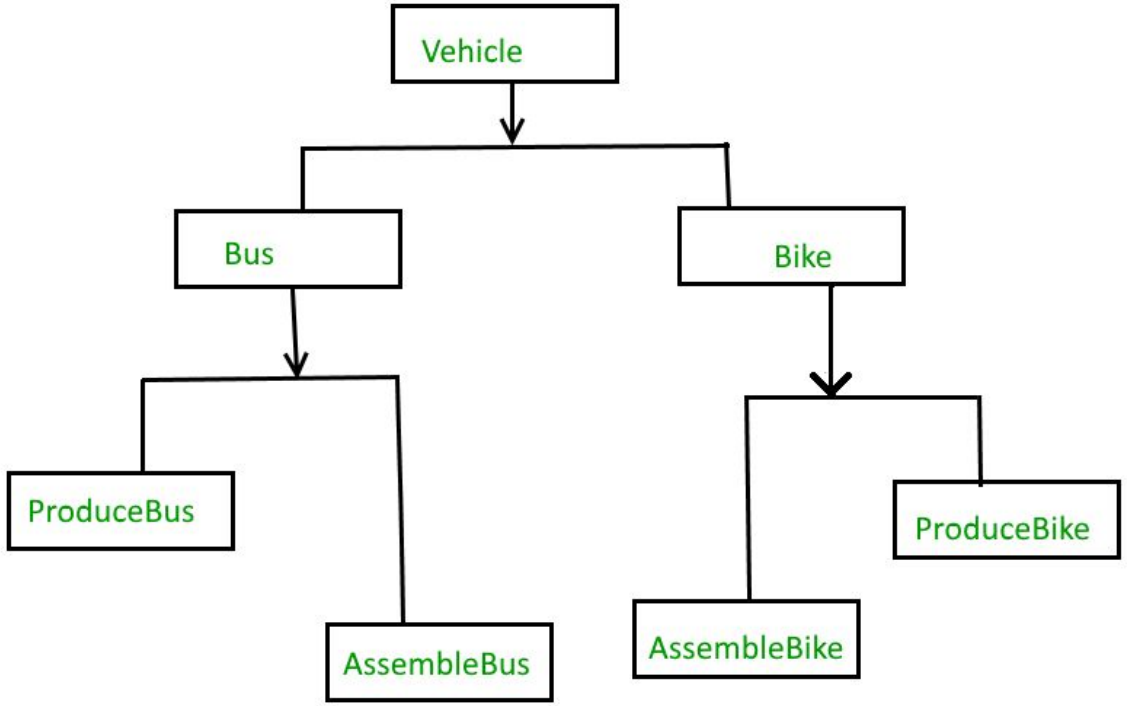
Bridge, Facade, Flyweight

Structural patterns

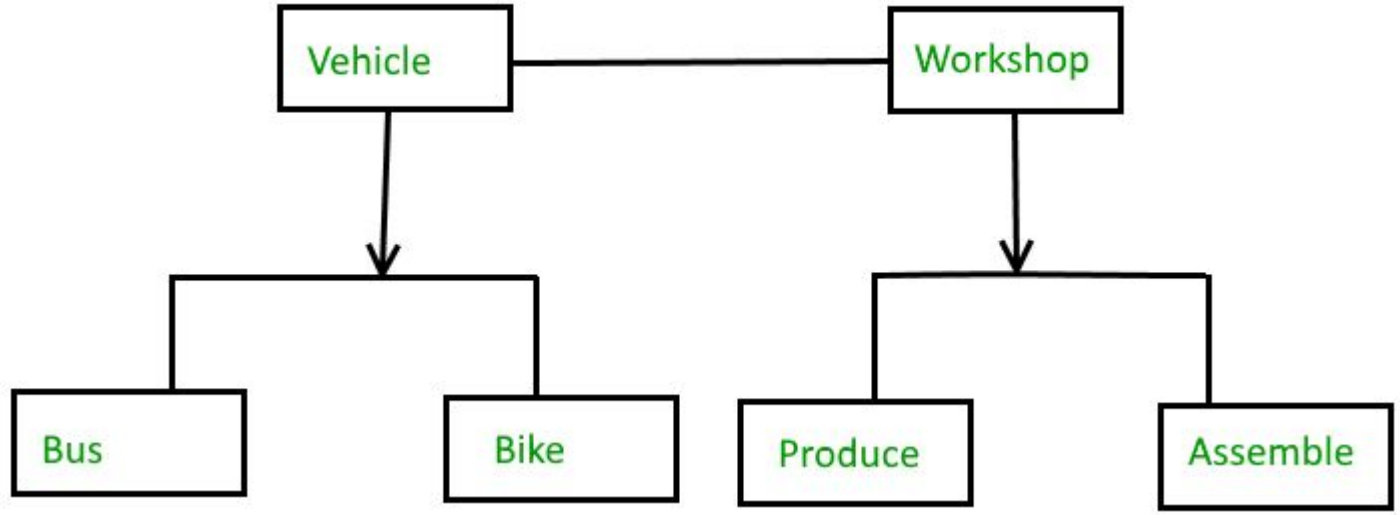
Bridge Pattern

- **Bridge** is a structural design pattern that lets you split a large class or a set of closely related classes into two separate hierarchies—**abstraction** and **implementation**—which can be developed independently of each other.
- This pattern involves an interface which acts as a bridge which makes the functionality of concrete classes independent from interface implementer classes.

Bridge Pattern

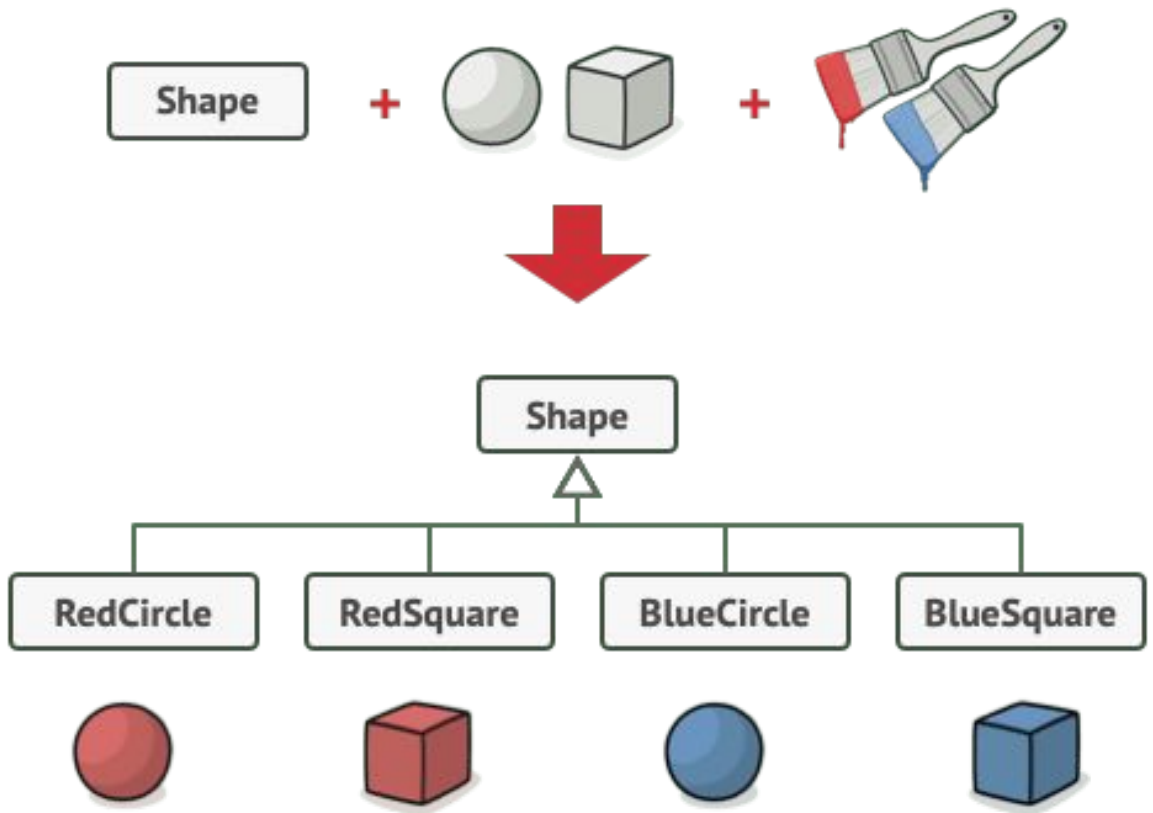


With out Bridge Pattern

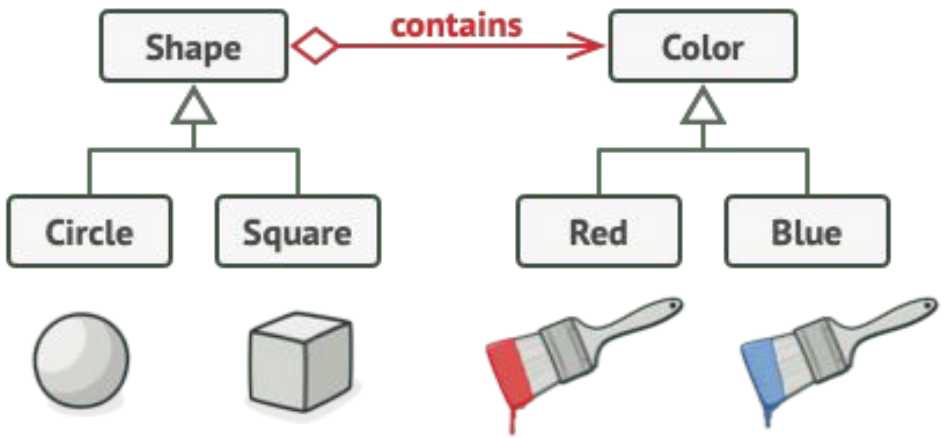


With Bridge Pattern

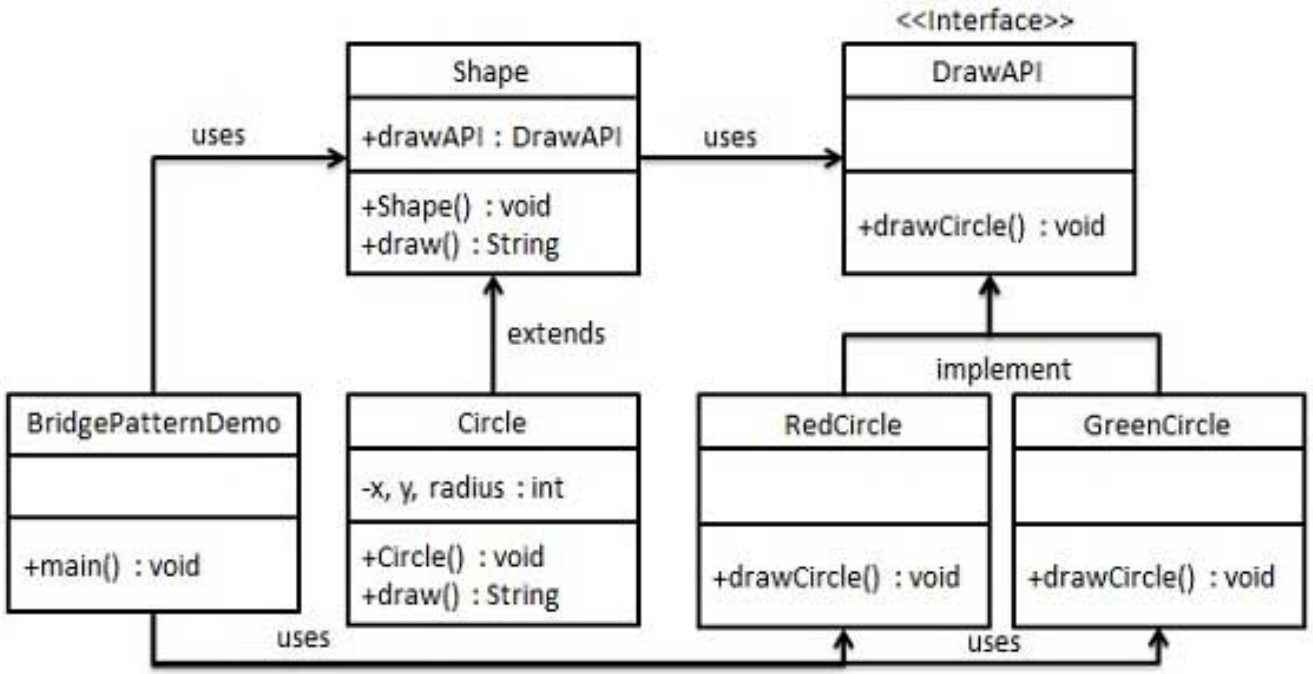
Bridge Pattern



Bridge Pattern



UML Diagram of Bridge Pattern



Implementation

```java
// abstraction in bridge pattern
abstract class Vehicle {
    protected Workshop workShop1;
    protected Workshop workShop2;

    protected Vehicle(Workshop workShop1, Workshop workShop2)
    {
        this.workShop1 = workShop1;
        this.workShop2 = workShop2;
    }

    abstract public void manufacture();
}
```

```java
}

// Refine abstraction 1 in bridge pattern
class Car extends Vehicle {
    public Car(Workshop workShop1, Workshop workShop2)
    {
        super(workShop1, workShop2);
    }

    @Override
    public void manufacture()
    {
        System.out.print("Car ");
        workShop1.work();
        workShop2.work();
    }
}

// Refine abstraction 2 in bridge pattern
class Bike extends Vehicle {
    public Bike(Workshop workShop1, Workshop workShop2)
    {
        super(workShop1, workShop2);
    }

    @Override
    public void manufacture()
    {
        System.out.print("Bike ");
        workShop1.work();
        workShop2.work();
    }
}
```

Implementation

```
// Implementer for bridge pattern
interface Workshop
{
    abstract public void work();
}

// Concrete implementation 1 for bridge pattern
class Produce implements Workshop {
    @Override
    public void work()
    {
        System.out.print("Produced");
    }
}

// Concrete implementation 2 for bridge pattern
class Assemble implements Workshop {
    @Override
    public void work()
    {
        System.out.print(" And");
        System.out.println(" Assembled.");
    }
}

// Demonstration of bridge design pattern
class BridgePattern {
    public static void main(String[] args)
    {
        Vehicle vehicle1 = new Car(new Produce(), new Assemble());
        vehicle1.manufacture();
        Vehicle vehicle2 = new Bike(new Produce(), new Assemble());
        vehicle2.manufacture();
    }
}
```

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

Facade Pattern

Shape.java

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

Rectangle.java

```
public class Rectangle implements Shape {

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

Circle.java

```
public class Circle implements Shape {

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

Square.java

```
public class Square implements Shape {

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

Facade Pattern

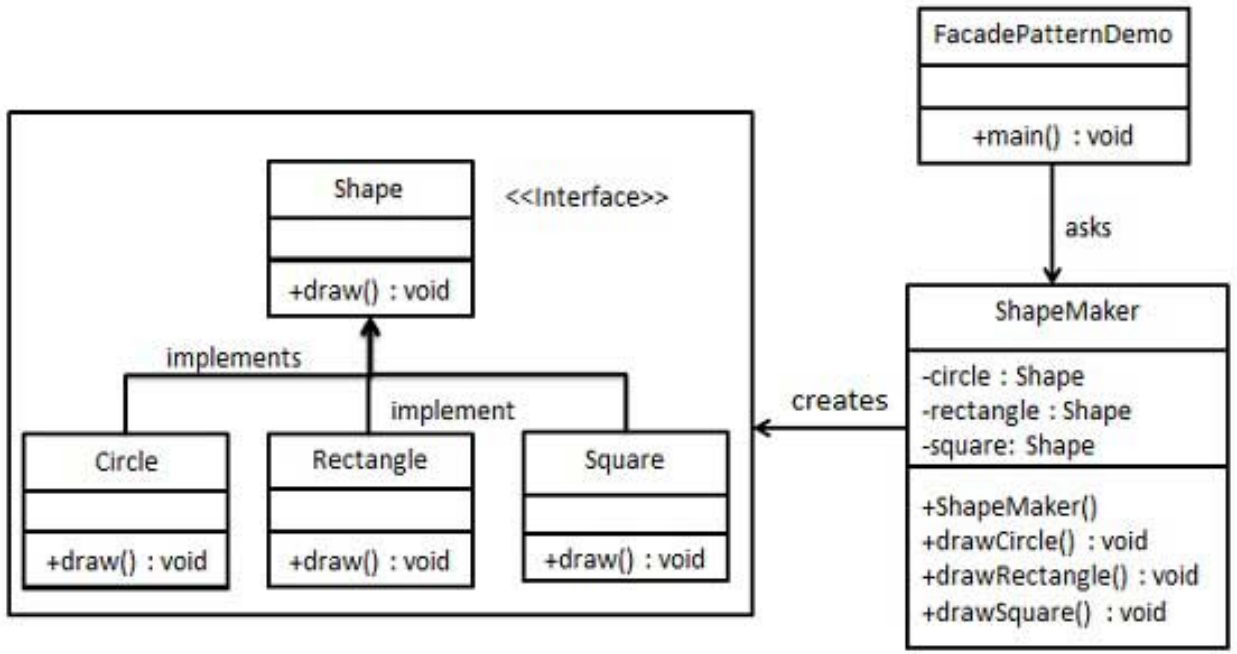
ShapeMaker.java

```
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();
   }
}
```

FacadePatternDemo.java

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

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

UML Diagram of Façade Pattern



Façade Pattern

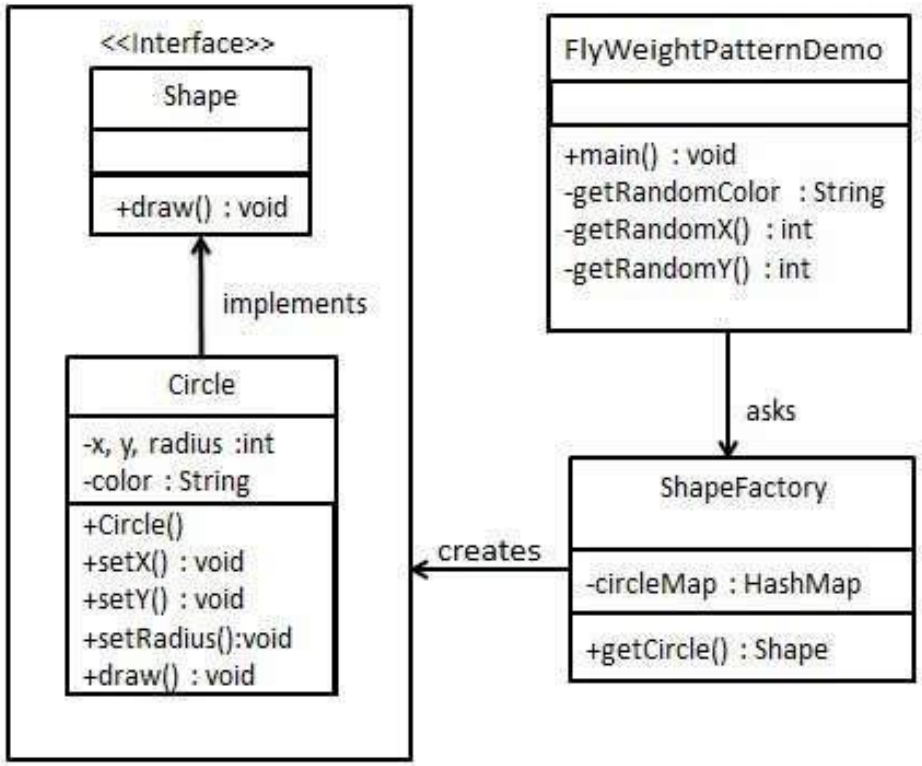
Similar pattern od façade is abstract factory.

Why abstract factory is creational and facade is structural?

Flyweight Pattern

- 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.

UML Diagram of Flyweight Pattern



Flyweight Pattern

- Using 5 colors, can you create 20 objects of circles only creating 5 objects?