Bridge is used when we need to decouple an abstraction from its implementation so that the two can vary independently. This type of design pattern comes under structural pattern as this pattern decouples implementation class and abstract class by providing a bridge structure between them.

This pattern involves an interface which acts as a bridge which makes the functionality of concrete classes independent from interface implementer classes. Both types of classes can be altered structurally without affecting each other.

We are demonstrating use of Bridge pattern via following example in which a circle can be drawn in different colors using same abstract class method but different bridge implementer classes.

interface DrawAPI{

public void drawCircle(int radius, int x ,int y);

}

class RedCircle implements DrawAPI{

@Override

public void drawCircle(int radius, int x, int y) {

System.out.println("drawing Circle red , radius ="+radius);

System.out.println("x ="+x);

System.out.println("y ="+y);

}

}

class GreenCircle implements DrawAPI{

@Override

public void drawCircle(int radius, int x, int y) {

System.out.println("drawing Circle green , radius ="+radius);

System.out.println("x ="+x);

System.out.println("y ="+y);

}

}

abstract class Shape{

protected DrawAPI drawAPI;

protected Shape(DrawAPI drawAPI) {

this.drawAPI = drawAPI;

}

public abstract void draw();

}

class Circle extends Shape{

private int x,y,radius;

public Circle(int x,int y,int radius,DrawAPI drawAPI) {

super(drawAPI);

this.x = x;

this.y = y;

this.radius = radius;

}

public void draw() {

drawAPI.drawCircle(radius, x, y);

}

}

public class BridgePatternDemo {

public static void main(String[] args) {

Shape redCircle = new Circle(100,100,10, new RedCircle());

Shape greenCircle = new Circle(100, 100, 10, new GreenCircle());

redCircle.draw();

greenCircle.draw();

}

}