DESIGN PRINCIPLES AND PATTERNS ASSIGNMENT

- Submitted by (Kavita Goodwani INT 438)

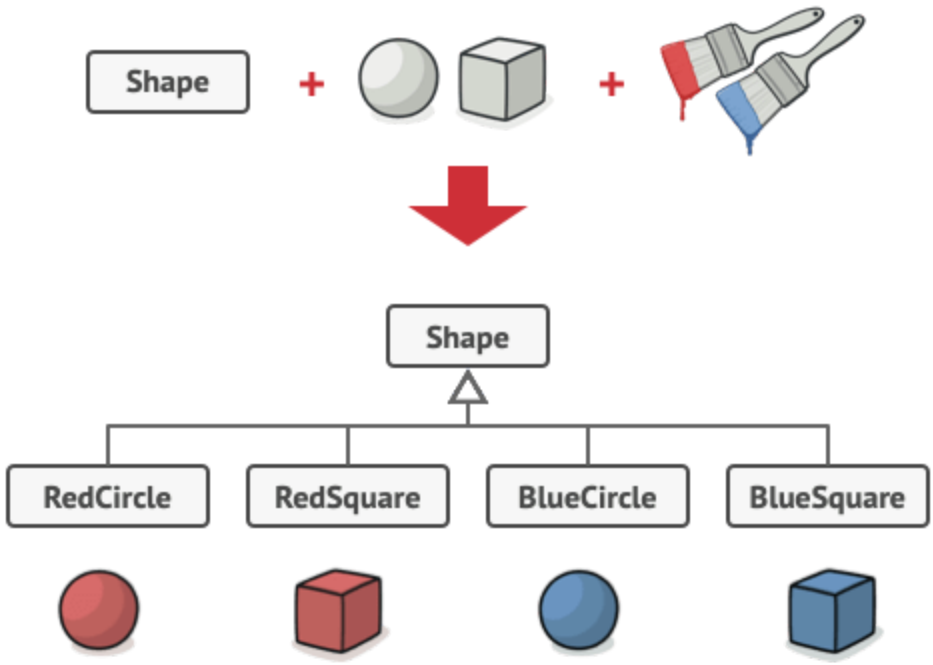
1. Explain a design pattern in detail

Bridge Design Pattern:

Bridge is a structural design pattern that is used to pave way for users to reach from the definition of one class to the other. It helps up split large classes i.e. a class containing say all the attributes into smaller isolated classes that can be created independent of each other.

The abstraction and implementation are defined to implement two separate hierarchies in bridge pattern.

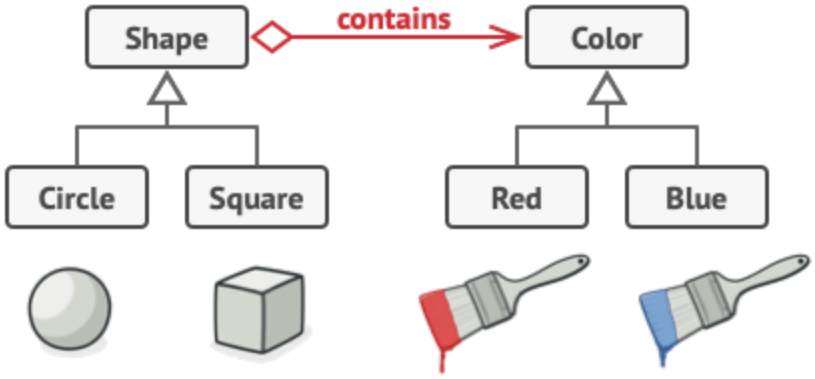
The given example explains the above definition of Bridge pattern:



Shapes can be of n types and n colors so the total number of subclasses increases exponentially as n*n times.

The reason behind such as scenario is the idea of extending shape class in two different dimensions.

In order to solve this problem, we move from Inheritance to Object Composition , i.e Shape extends Colors and has subclasses say Circle and Square and Color has subclasses Red and Blue.



In order to extract one of the dimensions into a separate class hierarchy, so that the original classes will reference an object of the new hierarchy, instead of having all of its state and behaviors within one class.

- The client can access the Abstraction part without any concern about the Implementation part.
- So in the above example, The abstraction is an interface or abstract class and the implementor is also an interface or abstract class.

Sample Code for above example:

```
abstract class Shape {
    protected color shape1;
    protected color shape2;

    protected Shape(color shape1, color shape2)
    {
        this.shape1 = shape1;
        this.shape2 = shape2;
    }

    abstract public void manufacture();
}



class Square extends Shape {
    public Square(color shape1, color shape2)
    {
        super(shape1, shape2);
    }

    @Override
    public void manufacture()
    {
        System.out.print("Square ");
        shape1.work();
        shape2.work();
    }
}
```

```
class Circle extends Shape {
    public Circle(color shape1, color shape2)
    {
        super(shape1, shape2);
    }

    @Override
    public void manufacture()
    {
        System.out.print("Circle ");
        shape1.work();
        shape2.work();
    }
}
```

```
interface color
{
    abstract public void work();
}
class Red implements color {
    @Override
    public void work()
    {
        System.out.print("Red Color");
    }
}

class Blue implements color {
    @Override
    public void work()
    {
        System.out.print(" And");
        System.out.println(" Blue.");
    }
}



class BridgePattern {
    public static void main(String[] args)
    {
        Shape vehicle1 = new Square(new Red(), new Blue());
        vehicle1.manufacture();
        Shape vehicle2 = new Circle(new Red(), new Blue());
        vehicle2.manufacture();
    }
}
```

OUTPUT

```
Square Red Color And Blue.
Circle Red Color And Blue.
```

Advantages of Bridge Design Patterns

1. Separates abstraction from implementation- Making codes isolated, easier for debugging and introducing new change
2. Useful in platform independent features