# Bridge design pattern

Muresan Alex

E30432

# Introduction

- It is a structural  design pattern
- Used when we need to decouple abstraction from implementation
- Decouples implementation class and abstract class by providing a bridge between them

# Problems solved

- An abstraction and its implementation should be defined and extended independently from each other.
- A compile-time binding between an abstraction and its implementation should be avoided so that an implementation can be selected at run-time.

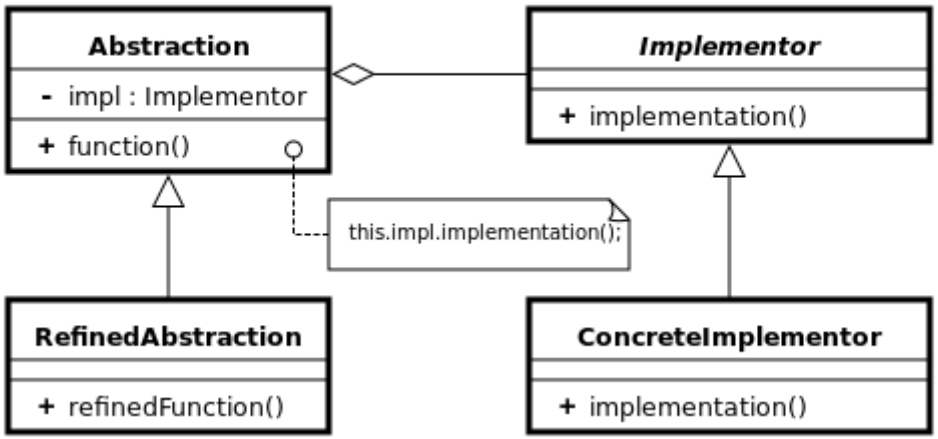
# Advantages

- Reduction in the number of sub classes – Sometimes, using pure inheritance will increase the number of sub-classes.
- Cleaner code and Reduction in executable size
- Interface and implementation can be varied independently – Maintaining two different class hierarchies for interface and implementation entitles to vary one independent of the other.

# Disadvantages

- Increases complexity.
- Double indirection – This will have a slight impact on performance. The abstraction needs to pass messages along to the implementor for the operation to get executed.

# Class diagram




**Abstraction (abstract class)**

defines the abstract interface

maintains the Implementor reference.

**RefinedAbstraction (normal class)**

extends the interface defined by Abstraction

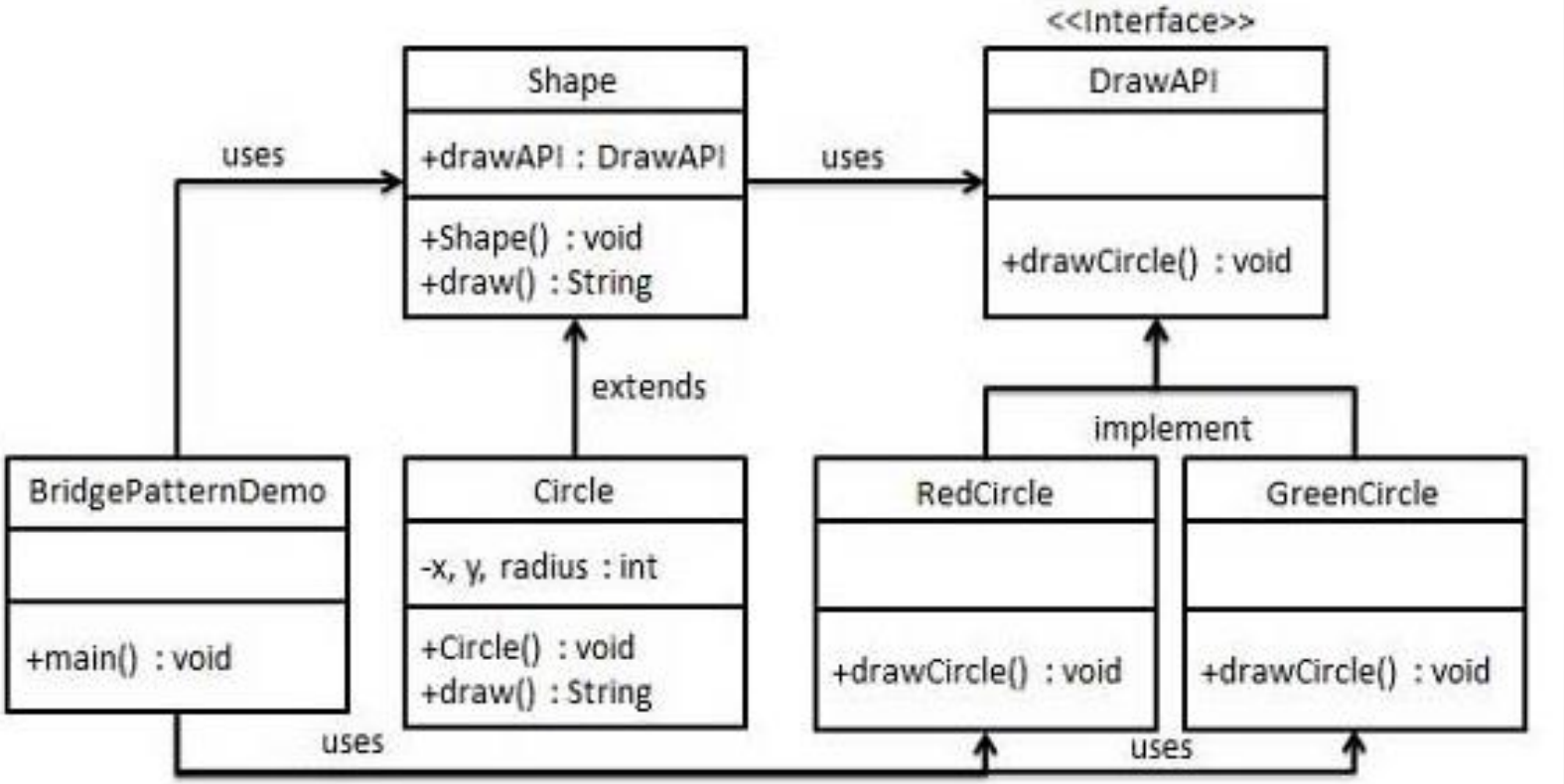
**Implementor (interface)**

defines the interface for implementation classes

**ConcreteImplementor (normal class)**

implements the Implementor interface

# Example


<<Interface>>
Shape
+drawAPI : DrawAPI
+Shape() : void
+draw() : String
DrawAPI
+drawCircle() : void
uses
uses
extends
implement
BridgePatternDemo
+main() : void
Circle
-x, y, radius : int
+Circle() : void
+draw() : String
RedCircle
+drawCircle() : void
GreenCircle
+drawCircle() : void
uses
uses

# Code

```
public interface DrawAPI {
   public void drawCircle(int radius, int x, int y);
}
```

```
public class RedCircle implements DrawAPI {
   @Override
   public void drawCircle(int radius, int x, int y) {
      System.out.println("Drawing Circle[ color: red, radius: " + radius + ", x: " +
   }
}
```

```
public abstract class Shape {
   protected DrawAPI drawAPI;

   protected Shape(DrawAPI drawAPI){
      this.drawAPI = drawAPI;
   }
   public abstract void draw();
}
```

```
public 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);
   }
}
```

# References


- https://en.wikipedia.org/wiki/Bridge_pattern
- https://www.tutorialspoint.com/design_pattern/bridge_pattern.htm
- https://www.geeksforgeeks.org/bridge-design-pattern/