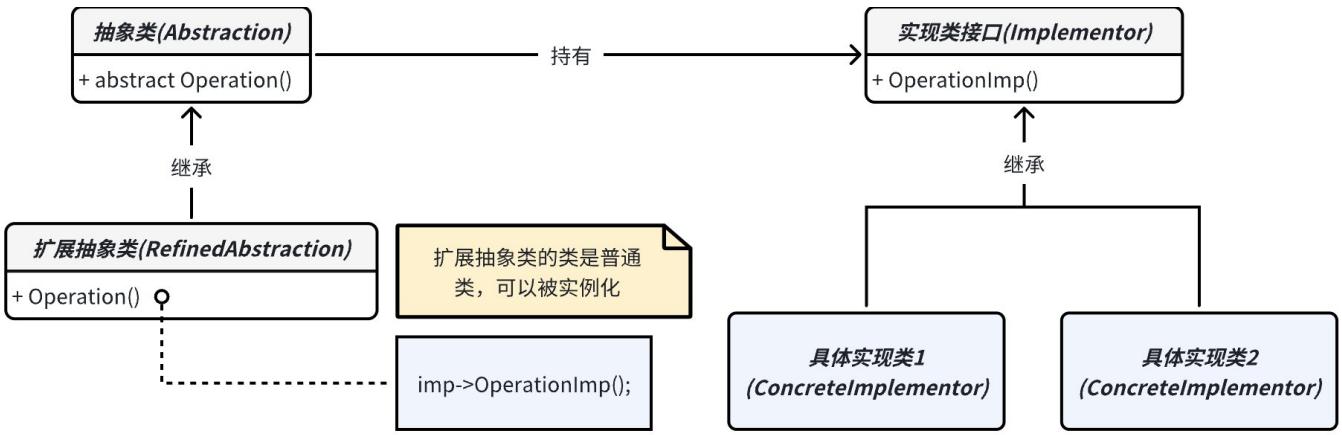
桥接模式

定义

- 桥接模式（Bridge Pattern）是一种结构型设计模式，旨在将抽象部分与其实现部分分离，使它们可以独立地变化；它通过组合而不是继承来实现这种分离。
- 桥接模式的设计意图是：
  - 解耦抽象和实现：使得抽象和实现可以独立变化，不会相互影响。
  - 提高系统的扩展性：通过分离抽象和实现，可以更容易地扩展系统的功能。
- 桥接模式适用于以下场景：
  - 当一个系统需要在多个维度上扩展时，例如形状和颜色的组合。
  - 当不希望在抽象类和实现类之间建立紧密耦合时。
  - 当系统希望避免由于增加实现类数量而导致的类的爆炸性增加时。
- 桥接模式主要涉及以下几个组成部分：
  - 实现类接口（Implementor）：定义实现类的接口，提供具体的实现。
  - 具体实现类（ConcreteImplementor）：实现实现类接口，提供具体的实现方式。
  - 抽象类（Abstraction）：定义了使用实现类接口的接口。
  - 扩展抽象类（RefinedAbstraction）：实现抽象类所定义的接口，可能增加一些额外的方法。
- 优点
  - 提高系统的扩展性：抽象和实现可以独立扩展。
  - 减少类的数量：通过组合而不是继承来减少类的数量。
- 缺点:
  - 增加系统的复杂性：由于引入了额外的抽象层，系统的复杂性可能会增加。



实例

- 实现类接口

```
1 public interface IColor
2 {
3     void ApplyColor();
4 }
```

- 具体实现类

```
1  public class Red : IColor
2  {
3      public void ApplyColor()
4      {
5          Console.WriteLine("Painting in Red.");
6      }
7  }
8
9  public class Green : IColor
10 {
11     public void ApplyColor()
12     {
13         Console.WriteLine("Painting in Green.");
14     }
15 }
```

- 抽象类

```
1 public abstract class Shape
```

```
{
    protected IColor color;

    protected Shape(IColor color)
    {
        this.color = color;
    }

    public abstract void Draw();
}
```

- 扩展抽象类

```
public class Circle : Shape
{
    public Circle(IColor color) : base(color) { }

    public override void Draw()
    {
        Console.Write("Drawing Circle. ");
        color.ApplyColor();
    }
}

public class Square : Shape
{
    public Square(IColor color) : base(color) { }

    public override void Draw()
    {
        Console.Write("Drawing Square. ");
        color.ApplyColor();
    }
}
```

- 客户端代码

```
static void Main(string[] args)
{
    Shape redCircle = new Circle(new Red());
    Shape greenSquare = new Square(new Green());

    redCircle.Draw();  // 输出: Drawing Circle. Painting in Red.
```

```
    greenSquare.Draw();  // 输出: Drawing Square. Painting in Green.
}
```