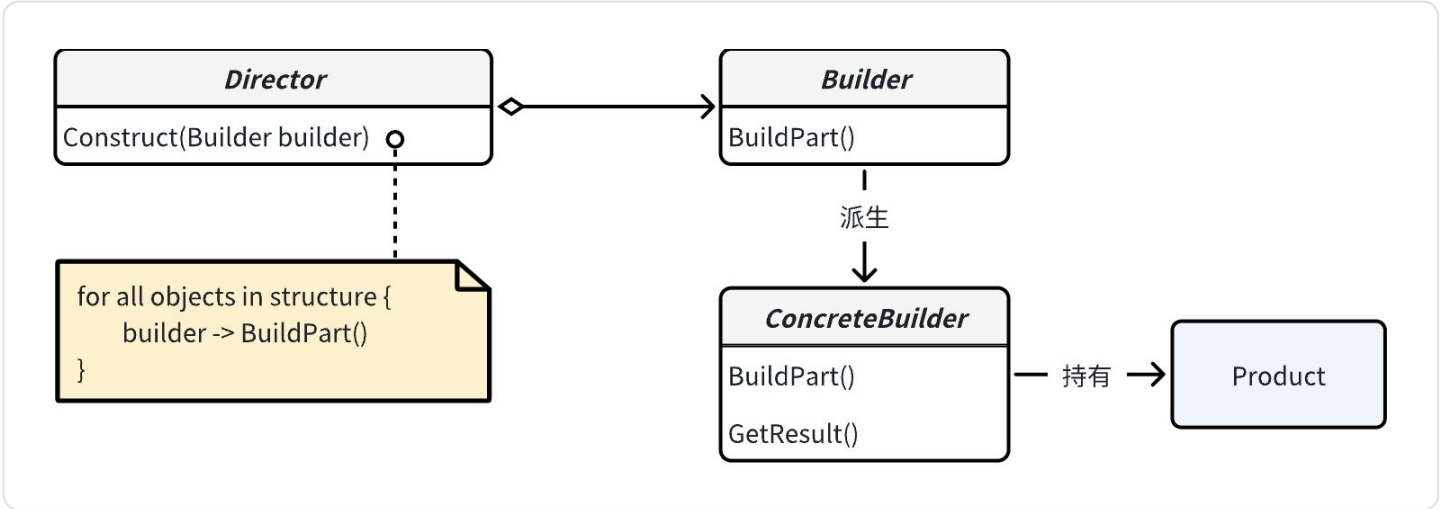
建造者模式

概念

- 建造者模式：将一个复杂对象的构建过程分解为多个简单的步骤，使得构建过程可以一步步进行，并且可以灵活地创建不同的表示。
- 设计意图：将对象的构建过程与其表示分离，使得同样的构建过程可以创建不同的表示。它主要用于创建一些复杂的对象，这些对象由多个部分组成。
- 使用场景
  - 需要生成的对象具有复杂的内部结构。
  - 需要生成的对象的内部属性相互依赖。
  - 希望通过不同的组合来创建不同的对象。
- 构成
  - Builder（建造者）：定义创建产品各个部分的接口。
  - ConcreteBuilder（具体建造者）：实现Builder接口，构建和装配各个部分。
  - Director（指挥者）：构建一个使用Builder接口的对象。
  - Product（产品）：表示被构建的复杂对象。
- 优点
  - 更好的控制对象的创建过程：可以一步步地创建对象，并且可以灵活地创建不同的表示。
  - 代码更具可读性和可维护性：将复杂对象的创建过程分解为多个简单的步骤。
  - 符合单一职责原则：将对象的创建过程与其表示分离。
- 缺点
  - 增加了代码的复杂性：需要定义多个Builder类和Director类。
  - 不适合变化多的产品：如果产品的内部结构经常变化，可能需要频繁修改Builder类。



实例

- 抽象建造者

```
1  public abstract class Builder
2  {
3      public abstract void BuildPartA();
4      public abstract void BuildPartB();
5      public abstract Product GetResult();
6  }
```

- 具体建造者

```
1  public class ConcreteBuilder : Builder
2  {
3      private Product product = new Product();
4
5      public override void BuildPartA()
6      {
7          product.Add("PartA");
8      }
9
10     public override void BuildPartB()
11     {
12         product.Add("PartB");
13     }
14
15     public override Product GetResult()
16     {
17         return product;
18     }
19 }
```

- 指挥者

```
public class Director
{
    public void Construct(Builder builder)
    {
        builder.BuildPartA();
        builder.BuildPartB();
    }
}
```

- 产品类

```
public class Product
{
    private List<string> parts = new List<string>();

    public void Add(string part)
    {
        parts.Add(part);
    }

    public void Show()
    {
        Console.WriteLine("\nProduct Parts -------");
        foreach (string part in parts)
        Console.WriteLine(part);
    }
}
```

- 客户端代码

```
static void Main(string[] args)
{
    Director director = new Director();
    Builder builder = new ConcreteBuilder();

    director.Construct(builder);
    Product product = builder.GetResult();
    product.Show();
```

```
9 }
```

- 打印结果

```
1 Product Parts -------
2 PartA
3 PartB
```