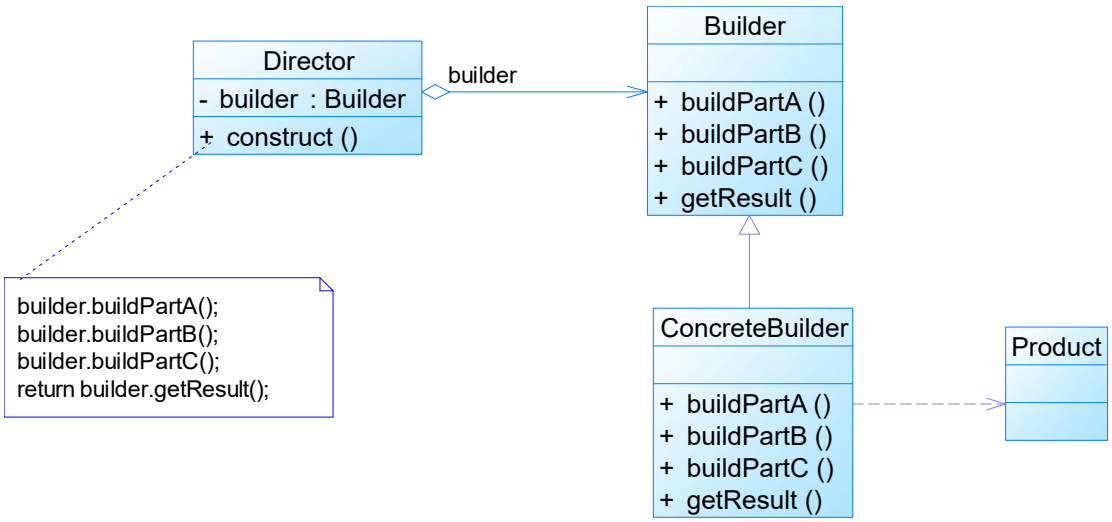
# 建造者模式

在软件开发过程中，存在一些复杂对象，它们是由多个组件（即一系列的成员属性）以一定顺序组合而成，按不同的顺序会生产出不同的产品，而对于用户来说，往往只是需要一个成品即可，对于其所含属性及组装方式并不关心，此时就可以采用**建造者模式：将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示**。

## 建造者模式类图：




在该模式中，有一个创建者类（Builder）用于生产产品的各个组件，还有一个导演类（Director）用于给建造者指定组件的组装顺序，而对于用户来说，只需要知道具体产品的类型并调用导演类中相应的方法即可得到一个成品。

## 通用代码如下：

```
//产品类
class Product
{
    private String partA; //可以是任意类型
    private String partB;
    private String partC;
    //省略 getter、setter 方法
}
//抽象建造者类
abstract class Builder
{
    protected Product product=new Product();
```

```
    //组装顺序
    protected List<String> sequence = new ArrayList<String>();

    public abstract void buildPartA();
    public abstract void buildPartB();
    public abstract void buildPartC();

    //设置组装顺序
    public void setSequence(List<String> sequence)
    {
        this.sequence = sequence;
    }
    //安组装顺序组装
    public Product getProduct()
    {
        for (int i = 0; i < sequence.size(); i++)
        {
            String action = sequence.get(i);
            if (action.equals("buildPartA"))
                buildPartA();
            else if (action.equals("buildPartB"))
                buildPartB();
            else if (action.equals("buildPartC"))
                buildPartC();
        }
        return product;
    }
}
//具体建造者
class ConcreteBuilder extends Builder
{
    public void buildPartA(){}
    public void buildPartB(){}
    public void buildPartC(){}
}
//导演类
class Director
{
    private Builder builder;
    public Director(Builder builder)
    {
        this.builder = builder;
    }
    public void setBuilder(Builder builder)
```

```
    {
        this.builder = builder;
    }
    public Product constructA()
    {
        List<String> sequence = new ArrayList<>();
        sequence.add("buildPartA");
        sequence.add("buildPartB");
        sequence.add("buildPartC");
        builder.setSequence(sequence);
        return builder.getProduct();
    }
    public Product constructB()
    {
        List<String> sequence = new ArrayList<>();
        sequence.add("buildPartB");
        sequence.add("buildPartA");
        sequence.add("buildPartC");
        builder.setSequence(sequence);
        return builder.getProduct();
    }
}
```

# 模式优点：

1、在建造者模式中，客户端不必知道产品内部组成的细节，将产品本身与产品的创建过程解耦。
2、每一个具体建造者都相对独立，因此可以很方便地替换具体建造者或增加新的具体建造者，用户使用不同的具体建造者即可得到不同的产品对象，符合“开闭原则”，利于系统的扩展。
3、可以更加精细地控制产品的创建过程。将复杂产品的创建步骤分解在不同的方法中，使得创建过程更加清晰，也更方便使用程序来控制创建过程。

# 模式缺点：

1、建造者模式所创建的产品一般具有较多的共同点，其组成部分相似，如果产品之间的差异性很大，则不适合使用建造者模式，因此其使用范围受到一定的限制。
2、如果产品的内部变化复杂，可能会导致需要定义很多具体建造者类来实现这种变化，导致系统变得很庞大。

# 建造者模式与抽象工厂模式的比较：

与抽象工厂模式相比，建造者模式返回一个组装好的完整产品，而抽象工厂模式返回一系列相关的产品，这些产品位于不同的产品等级结构，构成了一个产品族。
在抽象工厂模式中，客户端实例化工厂类，然后调用工厂方法获取所需产品对象，而在建造者模式中，客户端可以不直接调用建造者的相关方法，而是通过指挥者类来指导如何生成对象，包括对象的组装和建造过程，它侧重于一步步构造一个复杂对象，返回一个完整的对象。
如果将抽象工厂模式看成汽车配件生产工厂，生产一个产品族的产品，那么建造者模式就是一个汽车组装工厂，通过对部件的组装可以返回一辆完整的汽车。