**第六章 建造者模式**

**一、建造者模式概述**

建造者模式可以将部件本身和它们的组装过程分开，关注如何一步步创建一个包含多个组成部分的复杂对象，用户只需要指定复杂对象的类型即可得到该对象，而无须知道其内部的具体构造细节。考虑一下日常生活中的汽车，它包括车轮、方向盘、发动机等多种部件，但对于大多数用户而言，并不知道这些部件的装配细节，也几乎不会使用单独某个部件，而是使用一辆完整的汽车。

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。

**二、建造者模式结构与实现**

****

建造者模式结构图

（1）Builder（抽象建造者）：可以是抽象类，也可以是接口。在Builder中一般声明两类方法，一类是buildPartX（）（例如buildPartA（）、buildPartB（）等），它们用于**创建复杂对象的各个部件**；第二类是getResult（），用于**返回复杂对象**。

典型代码如下：

•protected 关键字

1.子类可以通过继承获得不同包父类的protected权限成员变量和成员方法，在子类中可以直接访问。

2. 在子类中可以通过子类的对象访问父类的protected成员变量和方法。

**public** **abstract** **class** Builder {

//创建产品对象

**protected** Product product=**new** Product();

**public** **abstract** **void** buildPartA();

**public** **abstract** **void** buildPartB();

**public** **abstract** **void** buildPartC();

•包 package

为了更好地组织类，Java 提供了包机制，可以简单地将包视为文件夹，让我们的类文件更整洁规范。

//返回产品对象

**public** Product getResult() {

**return** product;

}

}

（2）ConcreteBuilder（具体建造者）：继承于Builder类，用于**实现各个部件的具体构造方法和装配方法**，定义并明确所创建的复杂对象，还可以提供一个方法返回创建好的负责产品对象（该方法也可由抽象建造者实现）。

典型代码如下：

**public** **class** ConcreteBuilder1 **extends** Builder{

**public** **void** buildPartA() {

product.setPartA("A1");//partA的Setter方法

}

**public** **void** buildPartB() {

product.setPartB("B1");//partB的Setter方法

}

**public** **void** buildPartC() {

product.setPartC("C1");//partC的Setter方法

}

}

（3）Product（产品）：是被构造的复杂对象（就比如汽车），包含多个组成部件，具体建造者创建该产品的内部表示并定义它的装配过程。

典型代码如下：

**public** **class** Product {

//定义部件，部件可以是任意类型，包括值类型和引用类型

**private** String partA;

**private** String partB;

**private** String partC;

//partA的Getter方法和Setter方法省略

//partB的Getter方法和Setter方法省略

//partC的Getter方法和Setter方法省略

}

（4）Director（指挥者）：负责**安排复杂对象的建造次序**，指挥者与抽象建造者之间存在关联关系，可以在其construct（）建造方法中调用建造者对象的部件构造与装配方法。完成复杂对象的建造。

典型代码如下：

•this 关键字

大部分时候，普通方法访问其他方法、成员变量时无须使用this，但如果方法里有局部变量和成员变量同名，但程序又需要在该方法里访问这个被覆盖的成员变量，则必须使用this。

**this**.builder是当前对象具有的变量builder。

builder是参数传递的数值

**public** **class** Director {

**private** Builder builder;

//构造注入

**public** Director(Builder builder) {

**this**.builder=builder;

}

//设值注入

**public** **void** setBuilder(Builder builder) {

**this**.builder=builer;

}

//产品构建与组装方法

**public** Product construct() {

builder.buildPartA();

builder.buildPartB();

builder.buildPartC();

**return** builder.getResult();

}

}

（5）客户端典型代码：

**public** **class** Client{

**public** **static** **void** main(String arg[]) {

……

Builder builder = **new** ConcreteBuilder1(); //可通过配置文件实现

Director director = **new** Director(builder);

Product product = director.construct();

//首先创建一个builder对象，然后通过构造注入的方式创建一个director对象，最后通过调用director的方法construct（），按序完成PartA、PartB、PartC部件的构造，最后通过getResult（）返回一个产品对象。

……

}

}

**三、建造者模式优缺点与适用环境**

**优点：**

（1） 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，使得相同的创建过程可以创建不同的产品对象。

（2）每一个具体建造者都相对独立，与其他的具体建造者无关，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，扩展方便，符合开闭原则。

（3）可以更加精细地控制产品的创建过程。

**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，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，增加了系统的理解难度和运行成本。

**适用环境：**

（1）需要生成的产品对象有复杂的内部结构，这些产品对象通常包含多个成员变量。

（2）需要生成的产品对象的属性相互依赖，需要指定其生成顺序。

（3）对象的创建过程独立于创建该对象的类。在建造者模式中通过引入了指挥者类，将创建过程封装在指挥者类中，而不在建造者类和客户类中。

（4） 隔离复杂对象的创建和使用，并使得相同的创建过程可以创建不同的产品。

**四、课后习题答案**

1. D

2. C

3. D

4. 参考类图如下

