# 单例模式

# 工厂模式：

定义一个用于创建对象的接口，让子类决定实例化哪一个类。工厂方法使一个类的实例化延迟到其子类；

优点：

首先，良好的封装性，代码结构清晰。一个对象创建是有条件约束的，如一个调用者需

要一个具体的产品对象，只要知道这个产品的类名（或约束字符串）就可以了，不用知道创

建对象的艰辛过程，降低模块间的耦合。

其次，工厂方法模式的扩展性非常优秀。在增加产品类的情况下，只要适当地修改具体

的工厂类或扩展一个工厂类，就可以完成“拥抱变化”。例如在我们的例子中，需要增加一个

棕色人种，则只需要增加一个BrownHuman类，工厂类不用任何修改就可完成系统扩展。

再次，屏蔽产品类。这一特点非常重要，产品类的实现如何变化，调用者都不需要关

心，它只需要关心产品的接口，只要接口保持不变，系统中的上层模块就不要发生变化。因

为产品类的实例化工作是由工厂类负责的，一个产品对象具体由哪一个产品生成是由工厂类

决定的。在数据库开发中，大家应该能够深刻体会到工厂方法模式的好处：如果使用JDBC

连接数据库，数据库从MySQL切换到Oracle，需要改动的地方就是切换一下驱动名称（前提

条件是SQL语句是标准语句），其他的都不需要修改，这是工厂方法模式灵活性的一个直接

案例。

最后，工厂方法模式是典型的解耦框架。高层模块值需要知道产品的抽象类，其他的实

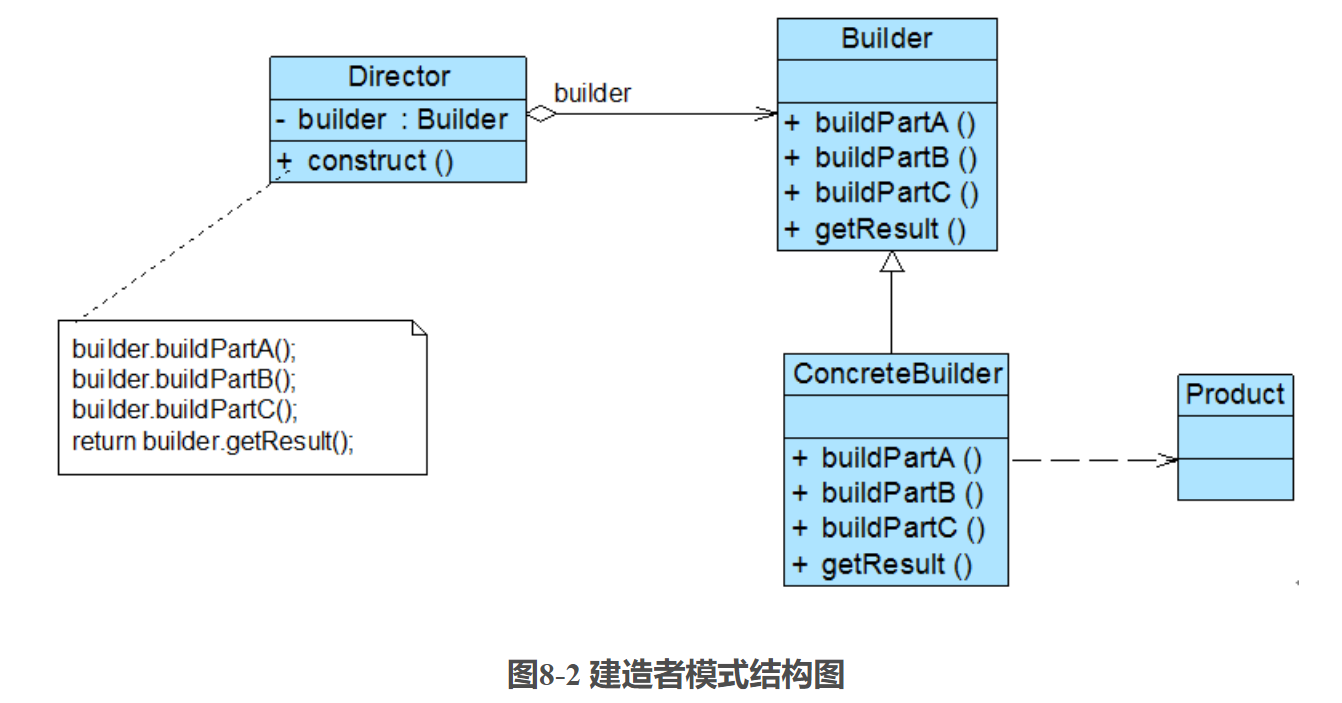
现类都不用关心，符合迪米特法则，我不需要的就不要去交流；也符合依赖倒置原则，只依

赖产品类的抽象；当然也符合里氏替换原则，使用产品子类替换产品父类，没问题！、

# 抽象工厂模式

# 建造者模式

如何一步步创建一个包含多个组成部分的复杂对象，建造者模式为解决此类问题而诞生。  
  
建造者模式(Builder Pattern)：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。建造者模式是一种对象创建型模式。



在建造者模式结构图中包含如下几个角色：

● Builder（抽象建造者）：它为创建一个产品Product对象的各个部件指定抽象接口，在该接口中一般声明两类方法，一类方法是buildPartX()，它们用于创建复杂对象的各个部件；另一类方法是getResult()，它们用于返回复杂对象。Builder既可以是抽象类，也可以是接口。

●ConcreteBuilder（具体建造者）：它实现了Builder接口，实现各个部件的具体构造和装配方法，定义并明确它所创建的复杂对象，也可以提供一个方法返回创建好的复杂产品对象。

●Product（产品角色）：它是被构建的复杂对象，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。

● Director（指挥者）：指挥者又称为导演类，它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。客户端一般只需要与指挥者进行交互，在客户端确定具体建造者的类型，并实例化具体建造者对象（也可以通过配置文件和反射机制），然后通过指挥者类的构造函数或者Setter方法将该对象传入指挥者类中

1.主要优点

建造者模式的主要优点如下：

(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，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，而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，用户使用不同的具体建造者即可得到不同的产品对象。由于指挥者类针对抽象建造者编程，增加新的具体建造者无须修改原有类库的代码，系统扩展方便，符合“开闭原则”

(3) 可以更加精细地控制产品的创建过程。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。

2.主要缺点

      建造者模式的主要缺点如下：

(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，例如很多组成部分都不相同，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，增加系统的理解难度和运行成本。

3.适用场景

      在以下情况下可以考虑使用建造者模式：

(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属性。

(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。

# 组合模式

处理树形结构；

对于树形结构，当容器对象（如文件夹）的某一个方法被调用时，将遍历整个树形结构，寻找也包含这个方法的成员对象（可以是容器对象，也可以是叶子对象）并调用执行，牵一而动百，其中使用了递归调用的机制来对整个结构进行处理。由于容器对象和叶子对象在功能上的区别，在使用这些对象的代码中必须有区别地对待容器对象和叶子对象，而实际上大多数情况下我们希望一致地处理它们，因为对于这些对象的区别对待将会使得程序非常复杂。组合模式为解决此类问题而诞生，它可以让叶子对象和容器对象的使用具有一致性。

**组合模式(Composite Pattern)：组合多个对象形成树形结构以表示具有“整体—部分”关系的层次结构。组合模式对单个对象（即叶子对象）和组合对象（即容器对象）的使用具有一致性，组合模式又可以称为“整体—部分”(Part-Whole)模式，它是一种对象结构型模式。**

