Java基础素质培训第一期 心得体会

商用产品研发部 李语君

设计模式是在特定环境下为解决某一通用软件设计问题提供的一套定制的解决方案，该方案描述了对象和类之间的互相作用。这是一套被反复使用、多数人知晓的代码设计经验的总结，实用设计模式是为了可重用代码、让代码更容易地被他人理解并且保证代码质量。大部分设计模式都兼顾了系统的可重用性和可扩展性，这使得我们可以更好地重用一些已有的设计方案、功能模块甚至一个完整的软件系统，避免我们经常做一些重复的设计或编写一些重复的代码，以提高开发效率和质量。

面向对象设计原则为支持可维护性复用而诞生，这些原则蕴含在很多设计模式中，它们是从许多设计方案中总结出的指导性原则。

(1) 单一职责原则:一个对象应该只包含单一的职责，并且该职责被完整的封装在一个类中。或者可以定义为：就一个类而言，应该只有一个引起它变化的原因。单一职责原则是实现高内聚、低耦合的指导方针。

(2) 开闭原则：一个软件实体应当对扩展开放，对修改关闭。即软件实体应尽量在不修改原有代码的情况下进行扩展。为了满足开闭原则，需要对系统进行抽象化设计，抽象化是开闭原则的关键。

(3) 里氏替换原则: 所有引用基类的地方必须能透明地使用其子类的对象。在软件中将一个基类对象替换成它的子类对象，程序将不会产生任何错误和异常。里氏代换原则是实现开闭原则的重要方式之一，由于使用基类对象的地方都可以使用子类对象，因此在程序中尽量使用基类类型来对对象进行定义，而在运行时再确定其子类类型，方便拓展系统功能。里氏替换伴随着继承，继承能够提高代码重用性、可拓展性，但也会增加耦合性、侵入性。

(4) 依赖倒转原则: 抽象不应该依赖于细节，细节应当依赖于抽象。换言之，针对接口编程，而不是针对实现编程。在程序代码中传递参数或在关联关系中，尽量引用层次高的抽象层类，即使用接口和抽象类进行变量类型声明、参数类型声明、方法返回类型声明等。如果说开闭原则是目标，里氏替换原则是基础，依赖倒转原则是手段。

(5) 接口隔离原则： 客户端不应该依赖那些它不需要的接口。需要注意控制接口的粒度，接口不能太小，如果太小会导致系统中接口泛滥，不利于维护；接口也不能太大，太大的接口将违背接口隔离原则，灵活性较差。

(6) 迪米特法则又称为最少知识原则，一个软件实体应当尽可能少地与其他实体发生相互作用。应当尽量创建松耦合的类，类之间的耦合度越低，就越有利于复用，一个处在松耦合中的类一旦被修改，不会对关联的类造成太大波及；在类的结构设计上，每一个类都应当尽量降低其成员变量和成员函数的访问权限；

(7) 合成复用原则: 尽量使用对象组合，而不是继承来达到复用的目的。在面向对象设计中，可以通过两种方法在复用已有的设计和实现，即通过组合/聚合关系或通过继承。组合/聚合可以使系统更加灵活，降低类与类之间的耦合度，类的变化对其他类造成的影响相对较少。

对于设计模式，主要有三种类型：创建性模式主要用于描述如何创建对象，将对象创建的职责独立；结构型模式主要用于描述如何实现类或对象的组合；行为型模式主要用于描述类或对象怎样交互以及怎样分配职责。而创建型设计模式有简单工厂模式、工厂模式、抽象工厂模式、单例模式、原型模式、建造者模式，其中使用较频繁的有工厂模式、抽象工厂以及单例。

(1) 简单工厂模式： 定义一个工厂类，它可以根据参数的不同返回不同类的实例，被创建的实例通常都具有共同的父类。工厂类负责创建的对象比较少，客户端只知道传入工厂类的参数，对于如何创建对象并不关心

(2) 工厂模式： 定义一个用于创建对象的接口，让子类工厂决定实例化哪一类产品。工厂方法模式让一个类的实例化延迟到其子类。具有良好的封装性和扩展性，隐藏了创建细节，并且当需要增加新产品时无需修改已有代码，添加一个具体工厂和对应的具体产品类即可，但是缺点是在添加新产品时，需要编写新的具体产品类，而且还要提供与之对应的具体工厂类。抽象层的引入增加了系统的抽象性和理解难度。

(3) 抽象工厂模式：提供一个创建一系列相关或相互依赖对象的接口，而无须指定它们具体的类。简单的说，在工厂模式的基础上，每个工厂能够创建一个产品族。抽象工厂模式是对工厂方法模式的改进，用于处理产品不只一类的情况。抽象工厂模式除了具有工厂方法模式的优点外，最主要的优点就是可以在类内部对产品族进行约束。增加新的产品族很方便，无须修改已有系统，符合“开闭原则”。增加新的产品类别麻烦，需要对原有系统进行较大的修改，甚至需要修改抽象层代码，这显然会带来较大的不便，违背了“开闭原则”。

(4) 单例模式： 确保某一个类只有一个实例，而且自行实例化并向整个系统提供这个实例，这个类称为单例类，它提供全局访问的方法。单例模式提供了对唯一实例的受控访问。进行扩展后允许可变数目的实例。由于在系统内存中只存在一个对象，因此可以节约系统资源。由于单例模式中没有抽象层，因此单例类的扩展有很大的困难。单例类的职责过重，在一定程度上违背了“单一职责原则”。因为单例类既充当了工厂角色，提供了工厂方法，同时又充当了产品角色，包含一些业务方法。

(5) 原型模式：使用原型实例指定创建对象的种类，并且通过拷贝这些原型创建新的对象。当创建新的对象实例较为复杂时，使用原型模式可以简化对象的创建过程。

扩展性较好，由于在原型模式中提供了抽象原型类，在客户端可以针对抽象原型类进行编程，而将具体原型类写在配置文件中，增加或减少产品类对原有系统都没有任何影响。

使用原型模式创建对象比直接new一个对象在性能上要好的多，因为Object类的clone方法是一个本地方法，直接操作内存中的二进制流，复制大对象时性能的差别非常明显。需要为每一个类配备一个克隆方法，当对已有的类进行改造时，需要修改源代码，违背了“开闭原则”。当对象之间存在多重的嵌套引用时实现起来可能会比较麻烦。

(6) 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。在建造者模式中，客户端不必知道产品内部组成的细节，将产品本身与其创建过程解耦。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，使得我们能够更加精确的控制复杂对象的产生过程。建造者模式很容易进行扩展，每一个具体建造者都相对独立，而与其他具体建造者无关要求构建产品的组成成分相似，构建步骤（算法）是不能剧烈变化的，最好是不变的，影响了灵活度。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内部复杂的变化。