JAVA是一门面向对象的语言，那么其面向对象主要有以下几个特性和原则：

[**面向对象三大特性六大原则**](http://www.cnblogs.com/corvoh/p/5747856.html)

面向对象的三大特性是"封装、"多态"、"继承"，六大原则是"单一职责原则"、"开放封闭原则"、"里氏替换原则"、"依赖倒置原则"、"接口分离原则"、"迪米特原则（高内聚低耦合）"。

**什么是面向对象**

面向对象(Object Oriented,OO)是软件开发方法。面向对象的概念和应用已超越了程序设计和软件开发，扩展到如数据库系统、交互式界面、应用结构、应用平台、分布式系统、网络管理结构、CAD技术、人工智能等领域。

面向对象是一种对现实世界理解和抽象的方法，是计算机编程技术[1] 发展到一定阶段后的产物。

这里拿 PHP 的 OOP 举个编程实例。



**三大基本特性：封装，继承，多态**

**封装**

封装，**就是把客观事物封装成抽象的类，**并且类可以把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏。**一个类就是一个封装了数据以及操作这些数据的代码的逻辑实体**。在一个对象内部，**某些代码或某些数据可以是私有的，不能被外界访问。**通过这种方式，对象对内部数据提供了不同级别的保护，以防止程序中无关的部分意外的改变或错误的使用了对象的私有部分。

**继承**

继承，指可以让**某个类型的对象获得另一个类型的对象的属性和方法**。它支持按级分类的概念。继承是指这样一种能力：**它可以使用现有类的所有功能**，并在无需重新编写原来的类的情况下对这些功能进行扩展。 通过继承创建的新类称为“子类”或“派生类”，被继承的类称为“基类”、“父类”或“超类”。继承的过程，就是从一般到特殊的过程。要**实现继承，可以通过 “继承”（Inheritance）和“组合”（Composition）来实现**。继承概念的实现方式有二类：**实现继承与接口继承**。实现继承是指直接使用 基类的属性和方法而无需额外编码的能力；接口继承是指仅使用属性和方法的名称、但是子类必须提供实现的能力。

**多态**

多态，是指**一个类实例的相同方法在不同情形有不同表现形式**。多态机制使具有不同内部结构的对象可以共享相同的外部接口。这意味着，虽然针对不同对象的具体操作不同，但通过一个公共的类，它们（那些操作）可以通过相同的方式予以调用。（比如输入的形参可以不同等去实现同一个方法从而得到不同的表现形式,）重写与重载是多态的体现。

**六大基本原则：SPR, OCP, LSP, DIP, ISP,LoD**

**单一职责原则SRP(Single Responsibility Principle)**

是**指一个类的功能要单一，不能包罗万象**。如同一个人一样，分配的工作不能太多，否则一天到晚虽然忙忙碌碌的，但效率却高不起来。

**开放封闭原则OCP(Open－Close Principle)**

**一个模块在扩展性方面应该是开放的而在更改性方面应该是封闭的**。比如：一个网络模块，原来只服务端功能，而现在要加入客户端功能，那么应当在不用修改服务端功能代码的前提下，就能够增加客户端功能的实现代码，这要求在设计之初，**就应当将服务端和客户端分开，公共部分抽象出来。**

**里式替换原则LSP(the Liskov Substitution Principle LSP)**

**子类应当可以替换父类并出现在父类能够出现的任何地方。**（比如父类public，子类一定是public）比如：公司搞年度晚会，所有员工可以参加抽奖，那么不管是老员工还是新员工，也不管是总部员工还是外派员工，都应当可以参加抽奖，否则这公司就不和谐了。

**依赖倒置原则DIP(the Dependency Inversion Principle DIP)**

A.高层次的模块不应该依赖于低层次的模块，**他们都应该依赖于抽象。**

B.抽象不应该依赖于具体实现，具体实现应该依赖于抽象。

**具体依赖抽象，上层依赖下层。高层模块就是调用端，底层模块就是具体实现类。（应该让底层模块定义抽象接口并且实现，让高层模块调用抽象接口，而不是直接调用实现类。）**

通俗来讲：依赖倒置原则的本质就是通过抽象（接口或抽象类）使个各类或模块的实现彼此独立，互不影响，实现模块间的松耦合。

问题描述：类A直接依赖类B，假如要将类A改为依赖类C，则必须通过修改类A的代码来达成。这种场景下，类A一般是高层模块，负责复杂的业务逻辑；类B和类C是低层模块，负责基本的原子操作；假如修改类A，会给程序带来不必要的风险。

解决方案：**将类A修改为依赖接口interface，类B和类C各自实现接口interface**，**类A通过接口interface间接与类B或者类C发生联系，则会大大降低修改类A的几率。（比如A依赖于车的轮胎，速度，牌子等接口，然后让B，C直接实现这些接口的方法，A间接通过接口与BC发生联系。）**

好处：依赖倒置的好处在小型项目中很难体现出来。但在大中型项目中可以减少需求变化引起的工作量。使并行开发更友好。

**接口分离原则ISP(the Interface Segregation Principle ISP)**

**模块间要通过抽象接口隔离开，而不是通过具体的类强耦合起来，即面向接口编程。**

核心思想：类间的依赖关系应该建立在最小的接口上通俗来讲：建立单一接口，不要建立庞大臃肿的接口，尽量细化接口，接口中的方法尽量少。

也就是说，我们要为各个类建立专用的接口，而不要试图去建立一个很庞大的接口供所有依赖它的类去调用。

问题描述：类A通过接口interface依赖类B，类C通过接口interface依赖类D，如果接口interface对于类A和类C来说不是最小接口，则类B和类D必须去实现他们不需要的方法。

需注意：接口尽量小，但是要有限度。对接口进行细化可以提高程序设计灵活性，但是如果过小，则会造成接口数量过多，使设计复杂化。所以一定要适度提高内聚，减少对外交互。使接口用最少的方法去完成最多的事情为依赖接口的类定制服务。只暴露给调用的类它需要的方法，它不需要的方法则隐藏起来。只有专注地为一个模块提供定制服务，才能建立最小的依赖关系。

**迪米特法则（Law of Demeter,简称LoD）**

核心思想：类间解耦。

通俗来讲：一个类对自己依赖的类知道的越少越好。自从我们接触编程开始，就知道了软件编程的总的原则：**低耦合，高内聚**。无论是面向过程编程还是面向对象编程，只有使各个模块之间的耦合尽量的低，才能提高代码的复用率。

**耦合是：**

简单地说，软件工程中对象之间的耦合度就是对象之间的依赖性。指导使用和维护对象的主要问题是对象之间的多重依赖性。对象之间的耦合越高，维护成本越高。因此对象的设计应使类和构件之间的耦合最小。

有软硬件之间的耦合，还有软件各模块之间的耦合。  
耦合性是程序结构中各个模块之间相互关联的度量。它取决于各个模块之间的接口的复杂程度、调用模块的方式以及哪些信息通过接口。

耦合可以分为以下几种，它们之间的耦合度由高到低排列如下：

* 内容耦合。当一个模块直接修改或操作另一个模块的数据时，或一个模块不通过正常入口而转入另一个模块时，这样的耦合被称为内容耦合。内容耦合是最高程度的耦合，应该避免使用之。
* 公共耦合。两个或两个以上的模块共同引用一个全局数据项，这种耦合被称为公共耦合。在具有大量公共耦合的结构中，确定究竟是哪个模块给全局变量赋了一个特定的值是十分困难的。
* 外部耦合 。一组模块都访问同一全局简单变量而不是同一全局数据结构，而且不是通过参数表传递该全局变量的信息，则称之为外部耦合。
* 控制耦合 。一个模块通过接口向另一个模块传递一个控制信号，接受信号的模块根据信号值而进行适当的动作，这种耦合被称为控制耦合。
* 标记耦合 。若一个模块A通过接口向两个模块B和C传递一个公共参数，那么称模块B和C之间存在一个标记耦合。
* 数据耦合。模块之间通过参数来传递数据，那么被称为数据耦合。数据耦合是最低的一种耦合形式，系统中一般都存在这种类型的耦合，因为为了完成一些有意义的功能，**往往需要将某些模块的输出数据作为另一些模块的输入数据。**
* 非直接耦合 。两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用来实现的。

***总结***  
耦合是影响软件复杂程度和设计质量的一个重要因素，在设计上我们应采用以下原则：如果模块间必须存在耦合，就尽量**使用数据耦合**，少用控制耦合，限制公共耦合的范围，尽量避免使用内容耦合。

**同一个模块内的各个元素之间要高度紧密，但是各个模块之间的相互依存度却要不那么紧密。**