# 第三章 面向对象

## 3.1 什么是封装

例如一辆小车看作是一个Java对象，这个对象由许多部分组成，车轮、车门、发动机等。一扇车门怎么设计合理是自己内部的事情，规则是由内部拟定的，跟外界似乎没什么关系，所以就引出了封装的概念，那么什么是封装呢或者封装可以实现哪些目的。

1. 封装可以隐藏实现的一些细节。
2. 让使用者通过对象内部对外暴露的接口来访问某些方法，该方法的实现细节由我们内部自己把控，包括增加逻辑的控制，限制对数据的不合理访问。
3. 便于修改，提高代码的可维护性。

为了实现良好的封装，需要从两个方面考虑：

1. 把字段（成员变量）和实现细节隐藏起来，不允许外部直接访问。
2. 把方法对外暴露出来，让方法去控制这些成员变量进行安全的访问和操作。

因此，简言之，把该隐藏的隐藏起来，把该暴露的暴露出去。

## 3.1.1 访问控制符

Java提供了三种访问权限，准确的说还有一种是默认的访问权限，加上它一共四种。

1. private 当前类可访问
2. default 当前包内可访问
3. protected 当前类和它的派生类可访问
4. public 全局访问

## 3.2 继承

Java的类可以分为三类。

1. 类：使用class定义，没有抽象方法
2. 抽象类：使用abstract class定义，可以有也可以没有抽象方法
3. 接口: 使用interface定义，只能有抽象方法

* 类可以继承类和抽象类（必须实现所有抽象方法），只能单继承，接口多实现。
* 抽象类可以继承类和抽象类（可全部、部分、或者完全不实现父类抽象方法），接口多实现（可全部、部分、或者完全不实现父类抽象方法）。
* 接口可以继承多个接口。

## 3.2.1 继承以后子类可以得到什么

* 1. 子类拥有父类非私有属性和方法
  2. 子类对父类进行扩展
  3. 子类可以重新定义父类的方法，即多态里面的覆盖

关于构造函数：

1. 构造函数不能被继承，子类可以通过super()显示调用父类的构造函数。
2. 创建子类时候，编译器会自动调用父类的无参构造函数。
3. 如果父类没有定义无参构造函数，子类必须在构造函数的第一行代码使用super()显示调用。
4. 类默认拥有无参构造函数，如果定义了其他有参构造函数，则无参构造函数失效，所以父类没有定义无参数构造函数，不是指没有写无参构造。

## 3.2.2 为什么需要继承

首先是代码的复用，最重要的是转型，即父类的引用变量可以指向子类对象。

## 3.3 多态的前奏

覆盖/重写的规则：

1. 子类不能覆盖父类私有方法，private对子类不可见，如果子类定义了一个和父类private 方法相同的方法，实为新增方法。
2. 重写方法的修饰符一定要大于或等于被重写方法的修饰符（public > protected > default > private）。
3. 重写抛出的异常需与父类相同或是父类异常的子类，或者重写方法干脆不写throws
4. 重写方法的返回值必须与被重写方法一致，否则编译报错。
5. 静态方法不能被重写为非静态方法，否则编译报错。
6. 子类重写父类方法时，不能使用比父类中被重写的方法更严格的权限。

## 3.4 多态的定义

指允许不同类的对象对同一消息做出相应。简单来说，就是用基类的引用指向子类的对象。

多态的前提：

1. 要有继承关系
2. 要有方法重写
3. 要有父类引用指向子类对象

口诀：

成员变量：编译看左边（父类），运行看左边（父类）

成员方法：编译看左边（父类），运行看右边（子类）

静态方法：编译看左边（父类），运行看左边（父类）

## 3.5 多态中向上转型和向下转型

**对象的向上转型**：父类 父类对象 = 子类实例

1. 父类有的方法，都可以调用，如果被子类重写了，则会调用子类的方法。
2. 父类没有的方法，而子类存在，则不能调用。
3. 向上转型只对方法有影响，对属性没有影响，属性不存在重写。

**对象的向下转型**：子类 子类对象 = （子类）父类实例

1. 为什么需要向下转型？当父类需要调用子类的扩充方法时，才需要向下转型。（多态的弊端就是不能调用子类的特有功能）。

## 多态的好处和弊端

* 多态的好处：

1.提高了代码的维护性（继承保证）

2.提高了代码的扩展性（由多态保证）

* 多态的弊端

1.不能使用子类的特有属性和方法。

## 3.7 静态和动态绑定

1. 静态绑定（前期绑定）：即在程序执行前，即编译的时候已经实现了该方法与所在类的绑定，像C就是静态绑定，针对Java简单的可以理解为程序编译期的绑定具体过程就是执行这个方法，只要到这个类的方法表里拿出这个方法在内存里的地址，然后就可以执行了。
2. Java中只有static，final，private和构造方法，以及成员变量是静态绑定，其他的都属于动态绑定，而private的方法其实也是final方法（隐式），而构造方法其实是一个static方法（隐式），所以可以看出把方法声明为final，第一可以让他不被重写，第二也可以关闭它的动态绑定。

* private方法的特点是不能被继承，也就是不存在调用其子类的对象，只能调用对象自身，因此private方法和定义该方法的类绑定在一起。
* static方法又称类方法，类方法属于类文件。它不依赖对象而存在，在调用的时候就已经知道是哪个类的，所以是类方法是属于静态绑定。
* final方法：final方法可以被继承，但是不能被重写，所以也就是说final方法是属于静态绑定的，因为调用的方法是一样的。
* 总结：如果一个方法不可被继承或者继承后不可被覆盖，那么这个方法就采用的静态绑定。

1. 动态绑定（后期绑定）：运行时根据对象的类型进行绑定，Java中的大多数方法都是属于动态绑定，也就是实现多态的基础。
2. Java实现了后期绑定，则必须提供一些机制，可在运行期间判断对象的类型，并分别调用适当的方法。也就是说，编译的时候该方法不与所在类绑定，编译器此时依然不知道对象的类型，但方法调用机制能自己去调查，找到正确的方法主体。Java里实现动态绑定的是JVM.