PDF-XChange
Click to buy NOW!
www.docu-track.com

四． 面向对象技术

1． 对象： everything is object.

对象：

有什么？ 指对象的属性

能做什么？ 指对象的方法

对象的属性是自身所特有的，与其它对象没什么关系。

所以，对象的属性是应该是私有的， 对于方法，该公开的公开， 该私有的私有。

对象的方法是： 一个对象对外的接口， 所以 方法非常重要 。

2． ？思考： 为什么要有面向对象的思想？

面向对象的思想，它符合人类对处理问题的逻辑思维方式。

面向对象是一套全新的思想方法，每个对象要各司其职，各尽所长。对象与对象之间互相交互，

面向对象也符合人类对客观世界的描述方式。

对于对象： 我们在设计的时候，一定要让对象简单，功能专一。 要求：可复用性， 弱耦合性，高内聚性， 可插入性。

3． 面向对象三大特性： 封装（Encapsulation）， 继承（Inheritance）， 多态。(Polymorphism)

1. 封装（Encapsulation）：

A． 屏蔽实现在细节。

B． 提供统一的用户接口。

C． 提高代码的重用性，维护性。

现在让我们来讨论一下： 面向对象和面向过程的区别？

面向对象： 先有对象， 也就是数据结构，后才有算法。

面向过程： 先有算法， 而后才有数据结构。

类：（CLASS）

什么是类？

是一种复杂的数据类型。（ 语法上解释）

PDF-XChange Click to buy NOW! www.docu-track.com

类是对对象的抽象。　　（语义上解释）

如何来定义一个类：

例：

```
class Animal {
    int age;
    String name;
}
```

这样就定义的一个动物类， 它有两个实例变量（成员属性）， age 和 name 。

Animal a = new Animal(); 注：对象变量 a 不包含对象，它只是指向一个对象，在 JAVA 中，任何对象变量的值都是指向存储在别处的对象的一个引用。

这样就定义了一个对象的引用 a, 它指向一个 Animal 对象。

注： 生成的对象是存在堆空间中的一块连续空间， 而变量 a 存的是此对象的首地址， 它存在 stack 空间中。

现在，让我们来看一下，如何申明一个方法：

方法申明： 修饰符 返回值类型 方法名（参数列表） 抛出的异常

EX: public String findNameById(int id) throws Exception

PDF-XChange Click to buy NOW! www.docu-track.com

类中最特殊的方法： 构造方法： （Constructor）

- 构造方法的作用：
  - 生成一个对象的同时调用仅也调用一次相应的构造方法
  - 在调用构造方法之前对象必需已经存在。
  - 构造方法不是用来生成对象的，而是对象一旦生成，就会自动地调用构造方法，注意：构造方法是自动被构造成的。
- 构造方法的特点
  - 构造方法没有返回值
  - 方法名必须与类名一致
  - 构造方法可以重载。
- 生成一个对象的步骤
  - 首先分配空间
  - 初始化属性（给实例变量赋默认值）
  - 调用构造方法

  注意： 如果有父类，则第一步应该是：递归地构造父类对象！ 实际上，在 JAVA 中，任何的类都有一基类，为 Object， 它是所有类的基类。

  所以，生成一个对象应为 4 步：

  首先递归地构造父类对象，（也就是说在构造父类对象时，也是按照上面三步进行的）

- 重载
  - 含义： 只要方法名相同，参数表不同，就可以构成方法重载。

PDF-XChange
Click to buy NOW!
www.docu-track.com

■ 方法重载，在 JAVA 中，也叫： 编译时多态。（由编译的时候来确定调用哪个方法）

■ 重载的原则： 向上就近匹配原则

注意： 如果只有 print(double d) 方法,而要调用 print(int) ，则编译器会自动地就近向上匹配 print(double) 方法。但是反过来就不行，
因为从大到小会丢失精度， 所以，没有就近向下匹配原则.
关键字： this
This: 指当前对象的。 。 。
有两种用法：

1. 表示当前对象： this.age = age; this.name = name;
2. 在调用本类的其它构造方法时。 This(参数表), 注：this 的这种用法只能放在第一行。

方法调用：传值， 传引用
记住： 在 JAVA 中，简单类型变量一定是传值。 对象变量一这是传引用（也就是指向对象的首地址）

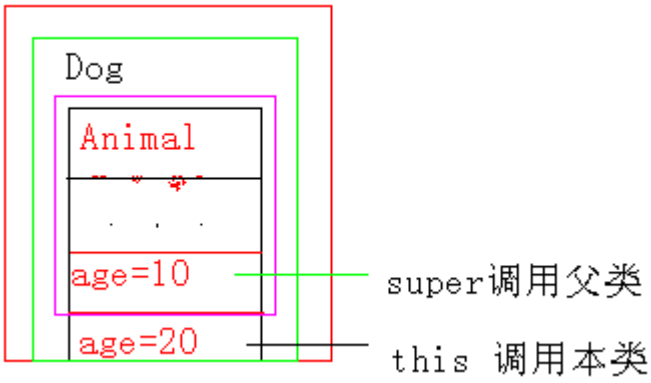
二． 继承

1．构造方法不能被继承。
2．方法和属性可以被继承。
3．子类的构造方法会默认自动调用父类构造方法
4．如果父类中没有默认的构造方法，则子类中必须用 super(参数) 来明确地调用父类相应的带参构造方法。
注： super 语句也必须放在第一条写出。

如：

```
class Animal {
    int age = 10;
}
class Dog extends Animal {
    int age = 20;
}
```

PDF-XChange Click to buy NOW! www.docu-track.com



注：子类中的 age 变量会遮盖(shadow)父类中的变量 age，所以，如果想在子类中访问父类的 age
变量，必须使用 super.age 来访问才行。

- 同名的子类变量会遮盖(shadow)父类实例变量;
- 有了继承，现在来看一下方法的覆盖。Overwritting
  - 覆盖条件：
    - 返回值类型相同
    - 方法名相同
    - 参数表相同
    - 修饰符相同或者权限更宽，并不能抛出比父类更多的异常

注：有一种特殊情况，在 JDK5.0 中，返回值类型可以是父子关系。

关键字总结： this, super

在一个子类构造方法中，会有一个隐藏的 super()语句，它会自动地完成调用父类构造方法，所以，此句可以不写。
但是，如果在子类中要调用父类带参的构造方法时，则必须明确地指定 super(参数) 语句，且只能写在第一条。。

```
Class Animal{
    Private int age;
    Public Animal() {}
    Public Animal(int age) { this.age = age; }
}
Class Dog extends Animal {
    Private int legs;
    Private String name;
    Public Dog() {}
    Public Dog(int age, int legs,String name) {
        Super(age);   //显示地调用父类的带参构造方法。
        This.legs = legs;
        This.names = names;
```

PDF-XChange Click to buy NOW! www.docu-track.com

```
        }
    }
```

好了，现在，到了该谈论一下各种修饰符的访问权限了。
在 JAVA 中，一共有四种访问修饰符，比 C++多一种， default
Private: 私有，只限在本类成员才能访问。
Default: 缺省，JAVA 的默认权限，除了本类，同包的可以访问
Protected: 保护， 本类，同包， 还有子类都可以访问
Public: 公开， 都能访问， 任何其它类。
由上述可以看出，JAVA 的访问权限是越来越宽， 从上至下而言。
Private--->default---->protected----->public
继承原则： Li- Substitution Principle: 里氏代换原则：
LSP： 任何使用父类的场合，都可以替换成子类，才能满足继承关系。
例如： 长方形（Rect）和正方形(Square)之间，谁也不能替换谁吧， 所以说它们并不满足 LSP 原则，所以不能应该使用继承。
而矩形则可以成为它们的父类。因为不论是长方形或是正方形都可以说是矩形。
慎用覆盖： 意思是子类不该太多地重写父类的方法，如果一个子类要过多地重写类的方法，则应考虑他们是否适合继承关系。

三．代码复用的两种实现：

1. 通过继承来实现，这是一种‘白盒复用’
2. 通过组合来实现， 这是一种‘黑盒复用’

如：

```
Class Oldclass {
    Public void a() { }
    Public void b() { }
    Public void c() { }
}
Class Newclass extends Oldclass {
    Public void a() { super.a(); }
}
```

如上所述，采用继承实现代码复用，子类会拥有父类所有的方法。而实际上我们只想 Newclass 中只要有 a() 方法，而不要其它方法。但继承的方式会暴露所有父类的方法。
组合合成实现： 在新类中，定义一个 Oldclass 的实例变量，然后在方法体中通过 Oldclass 的对象来调用其方法。
采用组合方式复用，可以对不需要的方法进行屏蔽。
实际上，在用组合方式时，方法的调用本质上是一种调用的委托，在新类中的 a() 方法，本质上是通过旧对象引用 a 的 a() 方法来完成任务的。
所以说它是一种方法调用的委托。

- 多态

PDF-XChange Click to buy NOW! www.docu-track.com

- LSP（Li- Substitution Principle）， 少覆盖，组合/继承复用原则，
- 特点：
  - 对象不变
  - 只能对对象调用编译时运行所定义的方法。
  - 运行时根据运行时类型进行自动判定。
    - 对属性： 看编译时类型
    - 对方法的重载： 看编译时类型

原则： 共性放在基类， 个性放在子类。

多态的表现：

1. 对象不变，运行时类型是不会随着编译时类型改变而改变的。
2. 编译时调用编译时类型的方法
3. 运行时要根据运行时类型找方法。

例：

```
class Animal {
    Public void eat() { . . .}
    Public void sleep() { . . . }
}
class Dog extends Animal {
    Public void spark() { . . .}
    Public void eat() { . . . }
```

PDF-XChange Click to buy NOW! www.docu-track.com

}

……

Animal a = new Dog(); //对象变量的引用 a，指向一个 Dog 对象

a.eat(); //调的是实际运行时类型的方法，也就是 Dog 中的方法，不是 Animal 的。

a.sleep(); // it's ok

a.spark(); // it's error. Why? 请自己想想

但是，我就是要调用 a.spark() 方法，而又要编译不出错，这就必须要强制类型转换了。

Dog d = (Dog)a; //此转换一定会成加，因为 a 的真正类型就是 Dog.

d.spark(); // 这样就可以调用了

原因： 在于 d 是 Dog 类型变量，它调用自己类的方法当然 OK，但是，如果变量 a 实质的类型不是 Dog， 则这种强制转换就会出错，但是它只会

在运行时表现出来，我们可以使用 instanceof 操作符来进行判断。

- instanceof

用法：（对象 instanceof 类名） 返回 true 或者 false

当 类名能够做为对象的编译时类型时，返回 true.

当 类名不能做为对象的编译时类型时，返回 false.

Ex:

PDF-XChange Click to buy NOW! www.docu-track.com

class Animal { }

class Dog extends Animal{ }

class Cat extends Animal{ }

Animal a = new Dog(); (a instanceof Animal) ---- true

Dog d = new Dog(); (d instanceof Dog) ------ true

Animal c = new Cat(); (c instanceof Dog) ------ false

大家可以这么来记：类名要么是对象的父类，要么是对象本身类型，否则都将返回 false.

此操用符往往用在类型转换前， 用来判定是否进行强制类型转换。

如： if(a instanceof Dog) Dog d = (Dog)a;

现在，让我们来写一个多态表现的例子：

```
public class TestPolymorphism {
   public static void main(String[] args) {
      Animal[] a = new Animal[3];
      a[0] = new Dog();
      a[1] = new Tiger();
      a[2] = new Cat();
      for(int i=0;i<a.length;i++) {
         a[i].eat();
      }
   }
}
class Animal {
   public void eat(){
   }
}
class Dog extends Animal{
   public void eat() {
      System.out.println("Dog eat bone. . .");
   }
}
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
class Cat extends Animal {
   public void eat() {
      System.out.println("Cat eat fish. . .");
   }
}
class Tiger extends Animal {
   public void eat() {
      System.out.println("Tiger eat person. . .");
   }
}
```

作者： 叶加飞 (Steven Ye)

mailto: leton.ye@gmail.com