[

Java千百问_05面向对象（001）_类、对象到底有什么秘密

,

点击进入_更多_Java千百问

1、类、对象的概念是什么

Java是目前应用最为广泛的面向对象特的语言，它具有以下基本概念：

- 类
- 对象
- 方法
- 抽象化
- 多态性
- 继承
- 封装

我们首先看看类和对象的概念。

类

类是一个模版。是一个可以定义一类具有相同属性、行为的模版。
例如：狗是一个类，它具有四肢、尾巴、头、脊椎等属性，具有吠叫、吃、繁殖等行为。

对象

对象是一个具体实例。根据是一个类的具体实例。
例如：我家对门养的一只狗，具体到了某一只。

2、Java如何定义类

类的定义如下：

```
public class Dog{
   String breed;
   int age;
   String color;

   void barking(){
   }

   void hungry(){
   }

   void sleeping(){
   }
}
```

类有以下关键点：

- 1、类可以包含以下任意类型的变量。
  局部变量、实例变量、类变量。了解三者详情看这里：局部变量、类变量、实例变量有什么区别
- 2、类可以有任意数量的方法。
  在上面的例子中，该类拥有barking(), hungry() 和 sleeping()三个方法。
- 3、类的构造函数

有关于类的讨论，其中最重要的部分之一是构造函数。每个类都有一个构造函数，如果我们不明确地写一个构造函数的类，Java编译器生成一个默认的构造函数（无参数的）。
在每次创建一个类的新对象时，至少有一个构造函数被调用，在这里会按照不同需求初始化一些对象内部的属性。
构造函数的主要规则是，他们应该具有相同的名称作为类。一个类可以有多个构造函数（当然每一个的参数都应该不同）。

构造函数的例子如下：

```
public class Puppy{
   public Puppy(){
   }

   public Puppy(String name){
      // This constructor has one parameter, name.
   }
}
```

Java还支持单实例类，在这里能够创建的类只有一个实例。更多单例模式看这里：[单例模式（Singleton）]

3、java中如何创建对象

如前面提到的，类提供的是模版，所以基本上一个对象是根据一个类创建的。
在Java中，使用关键字new创建新的对象。

根据类创建对象有三个步骤：
- 声明: 变量声明，一个变量名的对象类型。
- 实例化: 使用new关键字创建对象。
- 初始化: 关键字new后跟调用一个构造函数。初始化新的对象。

创建对象的实例：

```
public class Puppy{

   public Puppy(String name){
      // This constructor has one parameter, name.
      System.out.println("Passed Name is :" + name );
   }
   public static void main(String []args){
      // Following statement would create an object myPuppy
      Puppy myPuppy = new Puppy( "tommy" );
   }
}
```

将产生以下结果：

Passed Name is :tommy

4、如何访问对象的实例变量和方法

实例变量和方法是通过刚才创建的对象来访问的。
要访问一个实例变量和方法如下：

```
/* First create an object */
ObjectReference = new Constructor();

/* Now call a variable as follows */
ObjectReference.variableName;

/* Now you can call a class method as follows */
```

```
ObjectReference.MethodName();
```

例子：

```
public class Puppy{

   int puppyAge;

   public Puppy(String name){
      // This constructor has one parameter, name.
      System.out.println("Passed Name is :" + name );
   }
   public void setAge( int age ){
       puppyAge = age;
   }

   public int getAge( ){
       System.out.println("Puppy's age is :" + puppyAge );
       return puppyAge;
   }
   public static void main(String []args){
      /* Object creation */
      Puppy myPuppy = new Puppy( "tommy" );

      /* Call class method to set puppy's age */
      myPuppy.setAge( 2 );

      /* Call another class method to get puppy's age */
      myPuppy.getAge( );

      /* You can access instance variable as follows as well */
      System.out.println("Variable Value :" + myPuppy.puppyAge );
   }
}
```

将产生以下结果：

```
Passed Name is :tommy
Puppy's age is :2
Variable Value :2
```

]

[

Java千百问_05面向对象（002）_package和import作用是什么

,

点击进入_更多_Java千百问

1、java中package是什么

即包。简单地说，它是分类class(类)与interface(接口)的方式。
在Java开发中，将会写数以百计的类和接口，因此，对它们分类是必须的。
了解更多类看这里：类、对象到底有什么秘密
了解更多接口看这里：接口和抽象类有什么区别

2、java中import有什么用

在java中，编译器是根据包名+类名找到类的，而import语句是告诉编译器找到指定的类。
例如，要求编译器来加载所有目录中java安装/java/io可用的类：

```
import java.io.*;
```

3、源文件中如何声明的规则

在源文件中，声明类、声明包和写import语句时，有些规则是必不可少的。

1. 在每一个源文件中只能有一个public类。
2. 源文件可以有多个非public类。
3. public类名应该是源文件，以及应当以.java扩展名结尾。例如：类名是- public class Employee{}将源文件应为Employee.java。
4. 如果类在包中定义，那么package语句应该是源文件中的第一条语句。
5. 如果import语句都存在，那么它们必须写package语句和类声明之间。如果没有包（package）语句，那么import语句应该是源文件中的第一行。
6. import和package语句是针对源文件中的所有类。不同的import或package语句不能在同一个源文件中。
7. 类有四个访问级别，且有不同的类型，如抽象类，final类等（使用修饰符表示）。java修饰符看这里：public、privite、protected有什么区别
8. 除了上述类型的类，java中还有内部类、匿名类等。

4、java源文件如何编写

首先打开记事本，创建两个类：Employee和EmployeeTest，保存为名称为Employee.java、EmployeeTest.java，这就是java源文件。
这里Employee类和EmployeeTest类是公共类。
其中，Employee类有一个明确的构造函数，它接受一个参数。且有四个实例变量的名字，年龄，名称和工资。

```
import java.io.*;
public class Employee{
   String name;
   int age;
   String designation;
   double salary;

   // This is the constructor of the class Employee
   public Employee(String name){
```

```
      this.name = name;
   }
   // Assign the age of the Employee  to the variable age.
   public void empAge(int empAge){
      age =  empAge;
   }
   /* Assign the designation to the variable designation.*/
   public void empDesignation(String empDesig){
      designation = empDesig;
   }
   /* Assign the salary to the variable        salary.*/
   public void empSalary(double empSalary){
      salary = empSalary;
   }
   /* Print the Employee details */
   public void printEmployee(){
      System.out.println("Name:"+ name );
      System.out.println("Age:" + age );
      System.out.println("Designation:" + designation );
      System.out.println("Salary:" + salary);
   }
}
```

下面给出的是EmployeeTest类，它创建了Employee类的两个实例对象，并调用方法为每个对象中的属性赋值。
其中有一个特殊方法：main方法。运行类实际上是执行该类的main方法（没有main方法的类不能运行）。

```
import java.io.*;
public class EmployeeTest{

   public static void main(String args[]){
      /* Create two objects using constructor */
      Employee empOne = new Employee("James Smith");
      Employee empTwo = new Employee("Mary Anne");

      // Invoking methods for each object created
      empOne.empAge(26);
      empOne.empDesignation("Senior Software Engineer");
      empOne.empSalary(1000);
      empOne.printEmployee();

      empTwo.empAge(21);
      empTwo.empDesignation("Software Engineer");
      empTwo.empSalary(500);
      empTwo.printEmployee();
   }
}
```

现在，编译这两个类，然后运行EmployeeTest（如何编译看这里：如何用记事本编写Java程序）：

```
C > javac Employee.java
C > vi EmployeeTest.java
C > javac EmployeeTest.java
C > java EmployeeTest
```

执行结果如下：

```
Name:James Smith
Age:26
Designation:Senior Software Engineer
Salary:1000.0
Name:Mary Anne
Age:21
Designation:Software Engineer
Salary:500.0
```

]

[

Java千百问_05面向对象（003）_java中抽象概念如何体现的

,

点击进入_更多_Java千百问

1、抽象是什么

抽象，和具体对立，定义了事物的性质，事物的性质会随着抽象概念的改变而改变。

2、java中的抽象类是什么

java中最直接抽象概念的应用就是抽象类和接口，这里我们看一下抽象类。

抽象类和普通类一样，是一个模版。相比普通类，抽象类不具备实例化对象的能力。抽象类也可以定义属性和方法，比之普通类，它还可以定义没有实现的方法，即抽象方法。
通常会用一个具体类（子类）继承抽象类（父类），实现抽象类中的抽象方法。父类包含子类的集合的通用功能，但父类本身过于抽象而无法被单独使用。

例如：将脊椎动物定义为一个抽象类，它具有头、躯干、尾等属性，具有吃、繁殖等具体行为，还有一个抽象行为：吠叫，吠叫这个行为并不具体，因为不同种类的脊椎动物吠叫行为并不相同。狗类继承脊椎动物类，实现狗的吠叫行为。

了解更多类看这里：类、对象到底有什么秘密
接口和抽象类的区别看这里：接口和抽象类有什么区别

3、抽象类如何编写

java中使用abstract关键字来声明一个类的抽象。abstract需要写在class关键字前面。

实例：

```
/* File name : Employee.java */
public abstract class Employee
{
   private String name;
   private String address;
   private int number;
   public Employee(String name, String address, int number)
   {
      System.out.println("Constructing an Employee");
      this.name = name;
      this.address = address;
      this.number = number;
   }
   public double computePay()
   {
     System.out.println("Inside Employee computePay");
     return 0.0;
   }
   public void mailCheck()
   {
      System.out.println("Mailing a check to " + this.name
       + " " + this.address);
   }
   public String toString()
   {
      return name + " " + address + " " + number;
```

```
   }
   public String getName()
   {
      return name;
   }
   public String getAddress()
   {
      return address;
   }
   public void setAddress(String newAddress)
   {
      address = newAddress;
   }
   public int getNumber()
   {
     return number;
   }
}
```

请注意，这个Employee类与具体类没有什么不同。现在这个类是抽象的，但它仍然有三个字段，七种方法，和一个构造函数。

但现在如果想实例化这个类，如下：

```
/* File name : AbstractDemo.java */
public class AbstractDemo
{
   public static void main(String [] args)
   {
      /* Following is not allowed and would raise error */
      Employee e = new Employee("George W.", "Houston, TX", 43);

      System.out.println("
Call mailCheck using Employee reference--");
      e.mailCheck();
    }
}
```

编译后，会得到以下错误：
Employee.java:46: Employee is abstract; cannot be instantiated
Employee e = new Employee(“George W.”, “Houston, TX”, 43);
^
1 error

我们可以继承Employee类，如下所示：

```
/* File name : Salary.java */
public class Salary extends Employee
{
   private double salary; //Annual salary
   public Salary(String name, String address, int number, double
      salary)
   {
       super(name, address, number);
       setSalary(salary);
   }
   public void mailCheck()
   {
       System.out.println("Within mailCheck of Salary class ");
       System.out.println("Mailing check to " + getName()
       + " with salary " + salary);
   }
   public double getSalary()
   {
       return salary;
```

```
   }
   public void setSalary(double newSalary)
   {
       if(newSalary >= 0.0)
       {
          salary = newSalary;
       }
   }
   public double computePay()
   {
      System.out.println("Computing salary pay for " + getName());
      return salary/52;
   }
}
```

这里，我们可以实例化一个Salary对象，这个对象将继承Employee类的三个字段，七种方法。

```
/* File name : AbstractDemo.java */
public class AbstractDemo
{
   public static void main(String [] args)
   {
      Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 3600.00);
      Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.00);

      System.out.println("Call mailCheck using Salary reference --");
      s.mailCheck();

      System.out.println("
Call mailCheck using Employee reference--");
      e.mailCheck();
    }
}
```

运行后产生以下结果：
Constructing an Employee
Constructing an Employee
Call mailCheck using Salary reference –
Within mailCheck of Salary class
Mailing check to Mohd Mohtashim with salary 3600.0

Call mailCheck using Employee reference–
Within mailCheck of Salary class
Mailing check to John Adams with salary 2400.

4、什么是抽象方法

如果你想在抽象类中定义一个方法，但是希望该方法由子类来决定实际的执行情况，可以在父类中定义抽象方法。
声明一个抽象方法也需要abstract关键字。抽象方法只是一个方法签名，没有方法实现，故不需要使用花括号。
如下所示：

```
public abstract class Employee
{
   private String name;
   private String address;
   private int number;

   public abstract double computePay();
```

```
   //Remainder of class definition
}
```

声明一个抽象方法有两个规则：

1. 这个类也必须是抽象类。
2. 所有子类要么重写抽象方法，要么声明本身也为抽象。

如果Salary类是Employee类的子类，那么它必须实现computePay() 方法，如下：

```
/* File name : Salary.java */
public class Salary extends Employee
{
   private double salary; // Annual salary

   public double computePay()
   {
      System.out.println("Computing salary pay for " + getName());
      return salary/52;
   }

   //Remainder of class definition
}
```

]

[

Java千百问_05面向对象（004）_java接口到底是什么

,

点击进入_更多_Java千百问

1、什么是接口

接口（interface）不是一个类，它是抽象方法的集合。一个类实现一个接口，从而继承和实现接口的抽象方法。
抽象方法看这里：java中抽象概念如何体现的

接口的特点如下，先看与类类似的特点：

1. 接口的写法和写一个类类似，但它们是两个不同的概念。类描述对象的属性和行为。接口仅仅定义了事物的行为，且不会具体化这个行为。
2. 除非实现接口的是抽象类，不然接口中的所有方法必须在类（实现这个接口的类）中定义且实现。
3. 接口同类一样，可以包含任何数量的方法。
4. 接口同类一样，被写在同一个.java扩展名的源文件中，文件名与接口名称一致。
5. 接口同类一样，会被编译为一个.class文件。
6. 同类一样，需要指定包（package），来表明接口所在的目录结构。了解更多package看这里：package和import作用是什么

与类不同的几个方面：

1. 接口不能被实例化。
2. 接口不包含任何构造函数。
3. 接口中的所有方法都是抽象的。
4. 接口不包含实例字段。但可以定义常量，使用static和final关键字。
   了解常量看这里：局部变量、类变量、实例变量有什么区别
5. 类通过实现接口重写接口的方法，而不是继承。
6. 接口可以继承另一个接口。

2、如何编写一个接口

使用interface关键字声明一个接口。
例子：

```
/* File name : NameOfInterface.java */
import java.lang.*;
//Any number of import statements

public interface NameOfInterface
{
   //Any number of final, static fields
   //Any number of abstract method declarations
}
```

编写接口需要注意：

1. 接口是隐式抽象的。声明一个接口，不需要使用abstract关键字。
2. 接口中的每个方法也隐式抽象的。方法也不需要abstract关键字。
3. 接口中的方法是隐式公开的，即public。

例子：

```
/* File name : Animal.java */
interface Animal {
```

```
   public void eat();
   public void travel();
}
```

3、如何实现一个接口

当一个类实现一个接口，可以认为该类同意接受接口定义的行为。如果一个类不实现该接口的所有行为，该类必须自己声明为abstract。
类使用implements关键字来实现一个接口。如下：

```
/* File name : MammalInt.java */
public class MammalInt implements Animal{

   public void eat(){
      System.out.println("Mammal eats");
   }

   public void travel(){
      System.out.println("Mammal travels");
   }

   public int noOfLegs(){
      return 0;
   }

   public static void main(String args[]){
      MammalInt m = new MammalInt();
      m.eat();
      m.travel();
   }
}
```

这将产生以下结果:
Mammal eats
Mammal travels

重写接口中的方法有几个规则：

1. 接口方法抛出的异常在被实现时也需要抛出。
2. 重写方法时，应保持接口方法的参数和返回类型一致。
3. 一个实现类本身可以是抽象的，如果抽象类，接口方法可以不实现。
4. 一个类只能扩展一个类，但能实现多个接口。
5. 一个接口可以扩展另一个接口，类似于一个类可以扩展另一个类。

]

[

Java千百问_05面向对象（005）_接口和抽象类有什么区别

,

点击进入_更多_Java千百问

1、接口和抽象类有什么区别

在Java语言中，抽象类abstract class和接口interface是抽象定义的两种机制。

正是由于这两种机制的存在，才赋予了Java强大的面向对象能力。抽象类abstract class和接口interface在对于抽象定义方面具有很大的相似性，甚至可以相互替换。因此很多开发者在进行抽象定义时对二者的选择显得比较随意。其实，两者之间还是有很大的区别，对于它们的选择能反映出对问题本质的理解、对设计意图的理解。

了解抽象类看这里：java中抽象概念如何体现的
了解接口看这里：java中接口到底是什么
了解更多继承看这里：java类的继承有什么意义

具体如下：

运算	抽象类abstract class	接口interface
编写	使用abstract关键字	使用interface关键字
属性	可以有属性，public、protected、private及默认属性均可	只能有常量属性，即public static final
方法	有private方法，非abstract方法可以实现	均是public、隐式的abstract方法
实例化	不能实例化	不能实例化
实现	通过继承，使用extends关键词	使用implements关键词
类关系	只能继承一个abstract类，一个abstract可被多个类继承	一个类可以实现多个interface，一个interface可以被多个类实现
设计理念	Is-a关系（继承）	Like-a关系（组合）

了解is-a,has-a,like-a关系看这里：is-a，has-a，like-a是什么

2、interface应用在什么场合

1. 类与类之间需要特定的接口进行协调，而不在乎其如何实现。
2. 作为能够实现特定功能的标识存在，也可以是什么接口方法都没有的纯粹标识。如序列化接口：Serializable
3. 需要将一组类视为单一的类，而调用者只通过接口来与这组类发生联系。
4. 需要实现特定的多项功能，而这些功能之间可能完全没有任何联系。

3、abstract class应用在什么场合

1. 定义了一组接口，但又不想强迫每个实现类都必须实现所有的接口。可以用abstract class定义一组方法体，甚至可以是空方法体，然后由子类选择自己所感兴趣的方法来覆盖。
2. 某些场合下，只靠纯粹的接口不能满足类与类之间的协调，还必需类中表示状态的属性来区别不同的关系。
3. 规范了一组相互协调的方法，其中一些方法是共同的，与状态无关的，可以共享的，无需子类分别实现；而另一些方法却需要各个子类根据自己特定的状态来实现特定的功能。

]

[

Java千百问_05面向对象（006）_is-a，has-a，like-a是什么

,

点击进入_更多_Java千百问

1、is-a，has-a，like-a是什么

在面向对象设计的领域里，有若干种设计思路，主要有如下三种：
is-a、has-a、like-a
java中在类、接口、抽象类中有很多体现。
了解java看这里：什么是Java
了解类和对象看这里：类、对象到底有什么秘密
了解接口和抽象类看这里：接口和抽象类有什么区别

2、is-a是什么

is-a，顾名思义，是一个，代表继承关系。
如果A is-a B，那么B就是A的父类。
一个类完全包含另一个类的所有属性及行为。
例如PC机是计算机，工作站也是计算机，PC机和工作站是两种不同类型的计算机，但都继承了计算机的共同特性。因此在用Java语言实现时，应该将PC机和工作站定义成两种类，均继承计算机类。
了解更多继承看这里：java类的继承有什么意义

3、has-a是什么

has-a，顾名思义，有一个，代表从属关系。
如果A has a B，那么B就是A的组成部分。
同一种类的对象，通过它们的属性的不同值来区别。
例如一台PC机的操作系统是Windows，另一台PC机的操作系统是Linux。操作系统是PC机的一个成员变量，根据这一成员变量的不同值，可以区分不同的PC机对象。

4、 like-a是什么

like-a，顾名思义，像一个，代表组合关系。
如果A like a B，那么B就是A的接口。
新类型有老类型的接口，但还包含其他函数，所以不能说它们完全相同。
例如一台手机可以说是一个微型计算机，但是手机的通讯功能显然不是计算机具备的行为，所以手机继承了计算机的特性，同时需要实现通讯功能，而通讯功能需要作为单独接口，而不是计算机的行为。

5、is-a，has-a，like-a如何应用

如果你确定两件对象之间是is-a的关系，那么此时你应该使用继承；比如菱形、圆形和方形都是形状的一种，那么他们都应该从形状类继承。
如果你确定两件对象之间是has-a的关系，那么此时你应该使用聚合；比如电脑是由显示器、CPU、硬盘等组成的，那么你应该把显示器、CPU、硬盘这些类聚合成电脑类。
如果你确定两件对象之间是like-a的关系，那么此时你应该使用组合；比如空调继承于制冷机，但它同时有加热功能，那么你应该把让空调继承制冷机类，并实现加热接口。

]

[

Java千百问_05面向对象（007）_java类的继承有什么意义

,

点击进入_更多_Java千百问

1、继承是什么

继承，是面向对象语言的重要机制。
概念：一个类（子类）可以使用从另一个类（父类、超类）继承属性和方法。
java中的继承是单一继承，即一个子类只能有一个父类。
当然，接口(interface)也可以继承。
了解什么是类看这里:类、对象到底有什么秘密
了解接口的概念看这里:java接口到底是什么
接口和抽象类的区别看这:接口和抽象类有什么区别

2、如何使用继承

广义上讲，继承类、继承抽象类、实现接口都可以称为继承，但目前所说的java继承只是继承类或抽象类，即is-a模式最直接的体现。
了解更多is-a模式看这里：is-a，has-a，like-a是什么
继承类、抽象类使用extends关键字，实现接口使用implements关键字。通过使用这些关键字，我们可以使一个对象获得另一个对象的属性及方法。

例如：

```
public class Animal{
}

public class Mammal extends Animal{
}

public class Reptile extends Animal{
}

public class Dog extends Mammal{
}
```

根据上面的例子，面向对象的术语即：

- 动物是哺乳动物类的父类。
- 动物是爬虫类的父类。
- 哺乳动物和爬行动物是动物类的子类。
- 狗是哺乳动物双方和动物类的子类。
- 因此：狗也是动物的子类。

使用这个例子：

```
public class Dog extends Mammal{
```

```
   public static void main(String args[]){

      Animal a = new Animal();
      Mammal m = new Mammal();
      Dog d = new Dog();

      System.out.println(m instanceof Animal);
      System.out.println(d instanceof Mammal);
      System.out.println(d instanceof Animal);
   }
}
```

这将产生以下结果：

```
true
true
true
```

要注意的一个非常重要的事实是，Java只支持单一继承。这意味着，一个类不能扩展多个类。因此，以下是非法的：

```
public class extends Animal, Mammal{
}
```

3、如何继承接口

接口也可以扩展另一个接口，类似于一个类扩展另一个类中的方法。同样使用extends关键字来扩展接口，使子接口继承父接口的方法。
下面的Sports接口是由Hockey和Football接口扩展。

```
//Filename: Sports.java
public interface Sports
{
   public void setHomeTeam(String name);
   public void setVisitingTeam(String name);
}

//Filename: Football.java
public interface Football extends Sports
{
   public void homeTeamScored(int points);
   public void visitingTeamScored(int points);
   public void endOfQuarter(int quarter);
}

//Filename: Hockey.java
public interface Hockey extends Sports
{
   public void homeGoalScored();
   public void visitingGoalScored();
   public void endOfPeriod(int period);
   public void overtimePeriod(int ot);
}
```

Hockey接口有四个方法，但它继承了两个Sports接口的方法，因此，实现Hockey类需要实现六个方法。同样地，一个实现Football类需要实现Football的三个方法，以及Sports的两个方法。

4、java如何多重继承

我们知道，一个java类只能继承一个父类，所以，多重继承是不允许的。
但是，一个接口可以扩展多个父接口。
例如，Sports和Event都是接口：

```
public interface Hockey extends Sports, Event
```

]

[

Java千百问_05面向对象（008）_java中覆盖是什么

,

点击进入_更多_Java千百问

1、什么是覆盖

在java中，覆盖是针对继承才有的概念，某一个子类需要某些方法或属性，但又不想使用父类中的同名的方法或属性，就需要使用覆盖。
直白的来说，就是在子类中编写与父类同名、同参数、同返回值的方法，或同名、同类型的属性，子类对象调用该方法/属性时，运行的是子类的方法，而不会执行父类的方法（除非在方法第一行写super();会先执行父类方法，再继续执行子类代码。）

了解类的构造函数看这里：类、对象到底有什么秘密
了解更多继承看这里：java类的继承有什么意义

2、构造函数如何覆盖

了解类的构造函数看这里：类、对象到底有什么秘密
当子类继承一个父类时，构造子类时需要调用父类的构造函数，存在三种情况

1. 父类无构造函数或者一个无参数构造函数时。子类若无构造函数或者有无参数构造函数，子类构造函数中不需要显式调用父类的构造函数，系统会自动在调用子类构造函数前调用父类的构造函数。
2. 父类只有有参数构造函数时。子类在构造方法中必须要显示调用父类的构造函数，否则编译出错。
3. 父类既有无参数构造函数，也有有参构造函数。子类可以不在构造方法中调用父类的构造函数，这时使用的是父类的无参数构造函数。

3、方法如何覆盖

1. 子类覆盖父类的方法，必须有同样的参数和返回类型。
2. 子类覆盖父类的方法，在jdk1.5后，参数、返回类型可以是父类方法返回类的子类。
3. 子类覆盖父类的方法，可以修改方法的修饰符，但只能把方法的作用域放大，而不能把public修改为private。
   了解更多java修饰符看这里：public、privite、protected有什么区别
4. 子类方法能够访问父类的protected属性，但不能够访问默认的属性。
5. 子类的静态方法，与父类同名静态方法互不影响。由于静态方法使用类名调用，使用子类类名调用子类的方法，使用父类类名调用父类的方法。
6. 多态时，当子类覆盖了父类的方法，使用子类覆盖的方法。
   了解什么是多态：[java的多态性都有什么表现][6]
   [6]:

4、属性如何覆盖

1. 当子类覆盖父类的实例变量时，父类方法使用的是父类的实例变量，子类方法使用的是子类的实例变量。
2. 子类或父类使用实例变量时，都相当于在前面加了一个this指针（this.）。

了解更多java变量看这里：局部变量、类变量、实例变量有什么区别

5、实例

```
class SuperClass {
    private int number;


    public SuperClass() {
```

```
        this.number = 0;
    }


    public SuperClass(int number) {
        this.number = number;
    }


    public int getNumber() {
        number++;
        return number;
    }
}

class SubClass1 extends SuperClass {
    public SubClass1(int number) {
        super(number);
    }

}

class SubClass2 extends SuperClass {
    private int number;


    public SubClass2(int number) {
        super(number);
    }

}

public class SubClass extends SuperClass {

    private int number;


    public SubClass(int number) {
        super(number);
    }


    public int getNumber() {
        number++;
        return number;
    }


    public static void main(String[] args) {
        SuperClass s = new SubClass(20);
        SuperClass s1 = new SubClass1(20);
        SuperClass s2 = new SubClass2(20);
        System.out.println(s.getNumber());
        System.out.println(s1.getNumber());
        System.out.println(s2.getNumber());
        //结论一：多态时，当子类覆盖了父类的方法，使用子类覆盖的方法
        //结论二：当子类覆盖父类的实例变量时，父类方法使用的是父类的 实例变量，子类方法使用的是子类的实例变量
    }

}
```

输出结果：

1
21
21

]

[

Java千百问_05面向对象（009）_java的多态性都有什么表现

,

点击进入_更多_Java千百问

1、什么是多态

多态是对象具有多种表现形式的能力。
在面向对象语言中，接口的多种不同的实现方式即为多态。
多态性的科学解释：允许你将父对象设置成为一个或更多的他子对象的技术，赋值之后，父对象就可以根据当前赋值给它的子对象的特性以不同的方式运作。
通俗的解释，就是一句话：可以把一个子类的对象转换为父类的对象。
在Java中，所有的Java对象是多态的，因为任何对象都可以设置为自己本身的类和Object类（Object是所有类的父类）。

了解跟多继承看这里：java类的继承有什么意义

2、如果表现多态

让我们来看一个例子。
例子：

```
public interface Vegetarian{}
public class Animal{}
public class Deer extends Animal implements Vegetarian{}
```

现在，Deer类被认为是多态的，因为这有多重继承。于是：

- 鹿is-a动物（is-a等于继承，了解更多is-a看这里：is-a，has-a，like-a是什么）
- 鹿like-a素食（is-a等于实现）
- 鹿 is-a Object

由于多态性，下面的声明是合法的：

```
Deer d = new Deer();
Animal a = d;
Vegetarian v = d;
Object o = d;
```

所有d的参考变量d,a,v,o在堆中都指向相同的Deer对象。

了解更多引用传递看这里：
了解更多变量内存存储看这里：

3、方法多态是什么

我们知道方法覆盖，其中一个子类可以在其父覆盖的方法。即覆盖方法的本质是隐藏父类方法，除非子类使用的重载方法中的super关键字。无论子类以何种形态被声明，都会执行子类的方法，这就是方法的多态。

了解更多覆盖看这里：java中覆盖是什么
了解更多重载和覆盖的区别看这里：java中重载和覆盖有什么关系

例如：

```
/* File name : Employee.java */
public class Employee
{
   private String name;
   private String address;
   private int number;
   public Employee(String name, String address, int number)
   {
      System.out.println("Constructing an Employee");
      this.name = name;
      this.address = address;
      this.number = number;
   }
   public void mailCheck()
   {
      System.out.println("Mailing a check to " + this.name
       + " " + this.address);
   }
   public String toString()
   {
      return name + " " + address + " " + number;
   }
   public String getName()
   {
      return name;
   }
   public String getAddress()
   {
      return address;
   }
   public void setAddress(String newAddress)
   {
      address = newAddress;
   }
   public int getNumber()
   {
     return number;
   }
}
```

Salary类继承Employee类，如下所示：

```
/* File name : Salary.java */
public class Salary extends Employee
{
   private double salary; //Annual salary
   public Salary(String name, String address, int number, double
      salary)
   {
       super(name, address, number);
       setSalary(salary);
   }
   public void mailCheck()
   {
       System.out.println("Within mailCheck of Salary class ");
       System.out.println("Mailing check to " + getName()
       + " with salary " + salary);
   }
   public double getSalary()
   {
       return salary;
   }
   public void setSalary(double newSalary)
   {
       if(newSalary >= 0.0)
       {
          salary = newSalary;
```

```
        }
    }
    public double computePay()
    {
       System.out.println("Computing salary pay for " + getName());
       return salary/52;
    }
}
```

测试代码：

```
/* File name : VirtualDemo.java */
public class VirtualDemo
{
   public static void main(String [] args)
   {
      Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 3600.00);
      Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.00);
      System.out.println("Call mailCheck using Salary reference --");
      s.mailCheck();
      System.out.println("
Call mailCheck using Employee reference--");
      e.mailCheck();
    }
}
```

这将产生以下结果：

Constructing an Employee
Constructing an Employee
Call mailCheck using Salary reference –
Within mailCheck of Salary class
Mailing check to Mohd Mohtashim with salary 3600.0

Call mailCheck using Employee reference–
Within mailCheck of Salary class
Mailing check to John Adams with salary 2400.0

结论如下：

1. 在调用s.mailCheck()时，调用Salary类中的mailCheck()。
2. 在调用e.mailCheck()时，由于e是Salary对象的一个引用，所以实际调用的是Salary类中的mailCheck()。
3. 这种调用被称为虚拟方法调用，该方法被称为虚拟方法。

]

[

Java千百问_05面向对象（010）_java中重载和覆盖有什么关系

,

点击进入_更多_Java千百问

1、什么是重载

java的重载，简单说，就是方法有同样的名称，但是参数不相同。这样的同名不同参数的方法之间，互相称之为重载方法。
需要注意的是：
1. 方法名相同。
2. 参数不同，这里是说对应位置的参数类型至少有一个不同，当然List和List**都是List，算作相同。**
3. 返回值可以不同，可以相同。

例子：
“
public class Test{
public void a() {
System.out.println(“a()”);
};

```
public void a(int i) {
    System.out.println("a(int i)");
};

public void a(String j) {
    System.out.println("a(String j)");
};

public void a(int i, String j) {
    System.out.println("a(int i, String j)");
}

public String a(String j, int i) {
    System.out.println("a(String j, int i)");
    return j;
}

}
```

public class TestMain{

public static void main(String args[]){
Test test = new Test();
test.a();
test.a(1);
test.a(“1”);
test.a(1, “1”);

test.a(“1”, 1);
}
}
“
运行后产生以下结果：
a()
a(int i)
a(String j)
a(int i, String j)
a(String j, int i)

2、重载和覆盖（重写）的区别

了解覆盖看这里：java中覆盖是什么
了解继承看这里：java类的继承有什么意义
1. 覆盖必须继承，是针对父子类的；重载无需继承，是针对本类的。
2. 覆盖的方法名，参数完全一致；重载的方法名相同，参数列表不同。
3. 覆盖的方法修饰符大于等于父类的方法（例如不能把public修改为private），重载和修饰符无关。
了解修饰符看这里：public、privite、protected有什么区别
4. 覆盖不可以抛出父类没有抛出的一般异常，可以抛出运行时异常；重载方法可以抛出不同异常。

]

[

Java千百问_05面向对象（011）_引用传递和值传递有什么区别

,

点击进入_更多_Java千百问

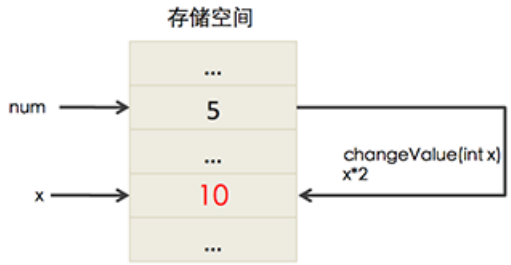
1、什么是值传递

值传递，是将内存空间中某个存储单元中存放的值，传送给另一个存储单元。（java中的存储单元并不是物理内存的地址，但具有相关性）
例如：

```
//定义了一个改变参数值的函数
public static void changeValue(int x) {
 x = x *2;
}
public class TestMain{
//调用该函数
int num = 5;
System.out.println(num);
changeValue(num);
System.out.println(num);
}
```

结果如下：
5
5

调用函数changeValue()前后num的值都没有改变。具体过程如图：



1. num作为参数传递给changeValue(int x)方法时，首先在内存空间中为x变量分配一个存储单元（我们说x指向这个存储单元）。
2. 将内存空间中num指向的存储单元中存放的值（即”5”），传递给了changeValue(int x)中的参数变量（即”x”），也就是把”5”传给了x变量指向的存储单元中。
3. changeValue(int x)方法中对x变量的一切操作，都是针对x指向的存储单元。与num指向的存储单元没有关系，当然也不会改变这个存储单元中的值。

所以，值传递，传递的是存储单元中的内容（8种基本类型：值，非基本类型：实际对象的地址）。
对于String来说JVM有他特殊的处理，了解更多看这里：String在内存中如何存放

2、什么是引用传递

java中只有值传递，没有引用传递。

所谓的引用传递，只是一个错误的概念。
例如：

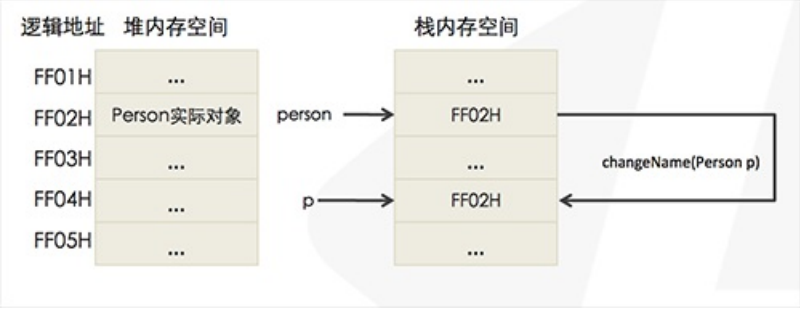
```
class person {
public static String name = "Jack";

//定义一个改变对象属性的方法
public static void changeName(Person p) {
 p.name = "Rose";
}
public static void main(String[] args) {
//定义一个Person对象，person是这个对象的引用
Person person = new Person();
//先显示这个对象的name属性
System.out.println(person.name);
//调用changeName(Person p)方法
changeName(person);
//再显示这个对象的name属性，看是否发生了变化
System.out.println(person.name);
}
}
```

执行后结果：
Jack
Rose

从结果看，方法用了一个对象参数，操作参数就可以改变传入对象。我们的对引用传递的错误观念这么认为：
该对象复制了一个引用副本，传给调用方法的参数，使得该方法可以对这个对象进行操作。
这种观念是初学者常犯的错误。

实际上过程如图：



1. main方法中new了一个对象Person，存储空间中实际分配了两个对象：新创建Person类的实体对象、指向该对象的引用变量person。
   其中，实体对象存放在堆内存中，引用变量存放在栈内存（Java存储特性）。
   了解更多java存储看这里：

2. 引用变量person指向的栈内存中，存放的是堆中实体对象的逻辑地址。
3. 调用changeName(Person p)方法，将person引用变量传入该方法参数p中（按照值传递，传递的是：实体对象的逻辑地址）。此时，changeName方法中对p的操作，与person没有关系。
4. changeName方法中，是对p指向的存储单元中的值（即实体对象的逻辑地址）所指向的实体对象进行操作。直接改变了该实体对象。
5. 由于person指向的存储单元中的值也是该实体对象的逻辑地址，这个实体对象已经在第4步中被改变了。所以有上面的结果。

3、引用传递和值传递有什么区别

引用传递是个伪概念，**java中只有值传递**。

]

[

Java千百问_05面向对象（012）_泛型是什么

,

点击进入_更多_Java千百问

1、什么是泛型

泛型是Java SE 1.5的新特性，泛型即参数化类型，也就是说所操作的数据类型被指定为一个参数。
这种参数类型可以用在类、接口和方法的创建中，分别称为泛型类、泛型接口、泛型方法。
如何使用泛型方法、泛型类/接口看这里：泛型如何使用
在没有泛型的情况的下，通过对类型Object的引用来实现参数的“任意化”，“任意化”带来的缺点是要做显式的强制类型转换，而这种转换是要求开发者对实际参数类型可以预知的情况下进行的。
对于强制类型转换错误的情况，编译器可能不提示错误，在运行的时候才出现异常，这是一个安全隐患。
所以，Java语言引入泛型的好处是安全简单，在编译的时候检查类型安全，并且所有的强制转换都是自动和隐式的，以提高代码的重用率。

2、泛型有那些特点

1. 泛型的类型参数只能是类类型（包括自定义类），不能是基础类型（如int，double和char）。
2. 同一种泛型可以对应多个版本（因为参数类型是不确定的），不同版本的泛型类实例是不兼容的。
3. 类的泛型类型可以有多个。
4. 泛型之间没有继承关系，即使String继承了Object。下面的代码是非法的
   List ls = new ArrayList();
   List lo = ls;

5. 泛型的类型参数可以使用extends语句，例如。习惯上称为有界类型。
6. 泛型的类型参数还可以是通配符类型。例如Class

]

[

Java千百问_05面向对象（013）_泛型如何使用

,

点击进入_更多_Java千百问

1、如何使用泛型方法

了解什么是泛型看这：泛型是什么
以下是定义泛型方法的规则：

1. 声明泛型方法时，在返回类型之前，需要有一个由尖括号（<>）分隔的泛型类型部分。
   2 . 一个泛型类型，也称为类型参数，是一个标识符，用于指定一个泛型类型的名称。
2. 类型参数可以用来声明返回类型和充当占位符传递给泛型方法。
3. 泛型方法的身体与其他方法一样。

例子：

```
public class GenericMethodTest
{
   // generic method printArray
   public static < E > void printArray( E[] inputArray )
   {
      // Display array elements
         for ( E element : inputArray ){
            System.out.printf( "%s ", element );
         }
         System.out.println();
    }

    public static void main( String args[] )
    {
        // Create arrays of Integer, Double and Character
        Integer[] intArray = { 1, 2, 3, 4, 5 };
        Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
        Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };

        System.out.println( "Array integerArray contains:" );
        printArray( intArray  ); // pass an Integer array

        System.out.println( "
Array doubleArray contains:" );
        printArray( doubleArray ); // pass a Double array

        System.out.println( "
Array characterArray contains:" );
        printArray( charArray ); // pass a Character array
    }
}
```

这将产生以下结果：
Array integerArray contains:
1 2 3 4 5 6
Array doubleArray contains:
1.1 2.2 3.3 4.4
Array characterArray contains:
H E L L O

泛型类型还可以被限制，使用extends关键字限制泛型的父类。
例子：

```
public class MaximumTest
{
   // determines the largest of three Comparable objects
   public static <T extends Comparable<T>> T maximum(T x, T y, T z)
   {
      T max = x; // assume x is initially the largest
      if ( y.compareTo( max ) > 0 ){
         max = y; // y is the largest so far
      }
      if ( z.compareTo( max ) > 0 ){
         max = z; // z is the largest now
      }
      return max; // returns the largest object
   }
   public static void main( String args[] )
   {
      System.out.printf( "Max of %d, %d and %d is %d

",
                  3, 4, 5, maximum( 3, 4, 5 ) );

      System.out.printf( "Maxm of %.1f,%.1f and %.1f is %.1f

",
                  6.6, 8.8, 7.7, maximum( 6.6, 8.8, 7.7 ) );

      System.out.printf( "Max of %s, %s and %s is %s
","pear",
         "apple", "orange", maximum( "pear", "apple", "orange" ) );
   }
}
```

这将产生以下结果：
Maximum of 3, 4 and 5 is 5
Maximum of 6.6, 8.8 and 7.7 is 8.8
Maximum of pear, apple and orange is pear

2、如何使用泛型类/接口

泛型类/接口的声明与非泛型类类似，除了类名后增加了一个泛型类型。
与泛型方法相比，泛型类的类型参数部分可以用逗号分隔的一个或多个泛型类型。

例子：

```
public class Box<T> {

  private T t;

  public void add(T t) {
    this.t = t;
  }

  public T get() {
    return t;
  }

  public static void main(String[] args) {
     Box<Integer> integerBox = new Box<Integer>();
     Box<String> stringBox = new Box<String>();
```

```
     integerBox.add(new Integer(10));
     stringBox.add(new String("Hello World"));

     System.out.printf("Integer Value :%d

", integerBox.get());
     System.out.printf("String Value :%s
", stringBox.get());
  }
}
```

这将产生以下结果：

Integer Value :10
String Value :Hello World

]

[

Java千百问_05面向对象（014）_如何获取范型的类Class

,

点击进入_更多_Java千百问

1、如何获取范型的类Class

java中，无法获取范型的类型，例如：

```
public class Box<T> {

  public static void main(String[] args) {
     System.out.printf(T);//编译错误
  }
}
```

其实，由于java是强类型语言，在编译时我们并不知道T是什么具体类型，只有在编译后，不同场景指定之后才会知道，所以在编译前是无法获取T的类型。如果想获取T的类型，可以在泛型类中声明一个对象，通过对象获取当前指定的类型。

例如：

```
public class TestGeneric<T> {

    private T t;

    public T getT() {
        return t;
    }

    public void setT(T t) {
        this.t = t;
    }

    public void test() {
        System.out.println(t.getClass() + ", " + t);
    }

    public static void main(String[] args) {
        TestGeneric<String> testGeneric = new TestGeneric<String>();
        testGeneric.setT("test");
        testGeneric.test();
    }
}
```

]