

# JAVA

Java是一门[面向对象](#)编程语言，不仅吸收了C++语言的各种优点，还摒弃了C++里难以理解的多继承、[指针](#)等概念，因此Java语言具有功能强大和简单易用两个特征。Java语言作为静态面向对象编程语言的代表，极好地实现了面向对象理论，允许程序员以优雅的思维方式编写复杂的编程。

Java具有简单性、面向对象、[分布式](#)、[健壮性](#)、[安全性](#)、平台独立与可移植性、[多线程](#)、动态性等特点。Java可以编写[桌面应用程序](#)、[Web应用程序](#)、[分布式系统](#)和[嵌入式系统](#)应用程序等。

## 发展历史

20世纪90年代，硬件领域出现了单片式计算机系统，这种价格低廉的系统一出现就立即引起了自动控制领域人员的注意，因为使用它可以大幅度提升消费类电子产品（如电视机顶盒、面包烤箱、移动电话等）的[智能化](#)程度。Sun公司为了抢占市场先机，在1991年成立了一个称为Green的项目小组，[帕特里克·詹姆斯·高斯林](#)、麦克·舍林丹和其他几个工程师一起组成的工作小组在加利福尼亚州[门洛帕克市](#)沙丘路的一个小工作室里面研究开发新技术，专攻计算机在家电产品上的嵌入式应用。

由于C++所具有的优势，该项目组的研究人员首先考虑采用C++来编写程序。但对于[硬件](#)资源极其匮乏的单片式系统来说，C++程序过于复杂和庞大。另外由于消费电子产品所采用的嵌入式处理器芯片的种类繁杂，如何让编写的程序[跨平台](#)运行也是个难题。为了解决困难，他们首先着眼于语言的开发，假设了一种结构简单、符合嵌入式应用需要的硬件平台体系结构并为其制定了相应的规范，其中就定义了这种硬件平台的[二进制](#)机器码指令系统（即后来成为“[字节码](#)”的指令系统），以待语言开发成功后，能有[半导体芯片](#)生产商开发和生产这种硬件平台。对于新语言的设计，Sun公司研发人员并没有开发一种全新的语言，而是根据嵌入式软件的要求，对C++进行了改造，去除了留在C++的一些不太实用及影响安全的成分，并结合嵌入式系统的实时性要求，开发了一种称为Oak的[面向对象语言](#)。

由于在开发Oak语言时，尚且不存在运行字节码的硬件平台，所以为了在开发时可以对这种语言进行实验研究，他们就在已有的硬件和软件平台基础上，按照自己所指定的规范，用软件建设了一个运行平台，整个系统除了比C++更加简单之外，没有什么大的区别。1992年的夏天，当Oak语言开发成功后，研究者们向硬件生产商进行演示了Green操作系统、Oak的程序设计语言、类库和其硬件，以说服他们使用Oak语言生产硬件芯片，但是，硬件生产商并未对此产生极大的热情。因为他们认为，在所有人对Oak语言还一无所知的情况下，就生产硬件产品的风险实在太大了，所以Oak语言也就因为缺乏硬件的支持而无法进入市场，从而被搁置了下来。

1994年6、7月间，在经历了一场历时三天的讨论之后，团队决定再一次改变了努力的目标，这次他们决定将该技术应用于[万维网](#)。他们认为随着Mosaic浏览器的到来，[因特网](#)正在向同样的高度互动的[远景](#)演变，而这一远景正是他们在有线电视网中看到的。作为原型，帕特里克·诺顿写了一个小型万维网浏览器WebRunner。

1995年，互联网的蓬勃发展给了Oak机会。业界为了使死板、单调的静态网页能够“灵活”起来，急需一种软件技术来开发一种程序，这种程序可以通过网络传播并且能够跨平台运行。于是，世界各大IT企业为此纷纷投入了大量的人力、物力和财力。这个时候，Sun公司想起了那个被搁置起来很久的Oak，并且重新审视了那个用软件编写的试验平台，由于它是按照嵌入式系统硬件平台体系结构进行编写的，所以非常小，特别适用于网络上的传输系统，而Oak也是一种精简的语言，程序非常小，适合在网络上传输。Sun公司首先推出了可以嵌入网页并且可以随同网页在网络上传输的Applet（Applet是一种将小程序嵌入到网页中进行执行的技术），并将Oak更名为Java（在申请注册商标时，发现Oak已经被人使用了，再想了一系列名字之后，最终，使用了提议者在喝一杯Java咖啡时无意提到的Java词语）。5月23日，Sun公司在Sun world会议上正式发布Java和HotJava浏览器。[IBM](#)、[Apple](#)、[DEC](#)、[Adobe](#)、[HP](#)、[Oracle](#)、Netscape和[微软](#)等各大公司都纷纷停止了自己的相关开发项目，竞相购买了Java使用许可证，并为自己的产品开发了相应的Java平台。

1996年1月，Sun公司发布了Java的第一个开发工具包（JDK 1.0），这是Java发展历程中的重要里程碑，标志着Java成为一种独立的开发工具。9月，约8.3万个网页应用了Java技术来制作。10月，Sun公司发布了Java平台的第一个即时（JIT）编译器。

1997年2月，JDK 1.1面世，在随后的3周时间里，达到了22万次的下载量。4月2日，Java One会议召开，参会者逾万人，创当时全球同类会议规模之纪录。9月，Java Developer Connection社区成员超过10万。

1998年12月8日，第二代Java平台的企业版J2EE发布。1999年6月，Sun公司发布了第二代Java平台（简称为Java2）的3个版本：[J2ME](#)（Java2 Micro Edition，Java2平台的微型版），应用于移动、无线及有限资源的环境；[J2SE](#)（Java 2 Standard Edition，Java 2平台的标准版），应用于桌面环境；[J2EE](#)（Java 2 Enterprise Edition，Java 2平台的企业版），应用于基于Java的应用服务器。Java 2平台的发布，是Java发展过程中最重要的一个里程碑，标志着Java的应用开始普及。

1999年4月27日，HotSpot虚拟机发布。[HotSpot](#)虚拟机发布时是作为JDK 1.2的附加程序提供的，后来它成为了JDK 1.3及之后所有版本的Sun JDK的默认虚拟机。



Java创始人之一：詹姆斯·高斯林

2000年5月，JDK1.3、JDK1.4和J2SE1.3相继发布，几周后其获得了[Apple](#)公司Mac OS X的工业标准的支持。2001年9月24日，J2EE1.3发布。2002年2月26日，J2SE1.4发布。自此Java的计算能力有了大幅提升，与J2SE1.3相比，其多了近62%的类和接口。在这些新特性当中，还提供了广泛的[XML](#)支持、安全套接字（Socket）支持（通过SSL与TLS协议）、全新的I/O API、正则表达式、日志与断言。2004年9月30日，J2SE1.5发布，成为Java语言发展史上的又一里程碑。为了表示该版本的重要性，J2SE 1.5更名为Java SE 5.0（内部版本号1.5.0），代号为“Tiger”，Tiger包含了从1996年发布1.0版本以来的最重大的更新，其中包括泛型支持、基本类型的自动装箱、改进的循环、枚举类型、格式化I/O及可变参数。

2005年6月，在Java One大会上，Sun公司发布了Java SE 6。此时，Java的各种版本已经更名，已取消其中的数字2，如J2EE更名为[JavaEE](#)，J2SE更名为JavaSE，J2ME更名为[JavaME](#)。

2006年11月13日，Java技术的发明者Sun公司宣布，将Java技术作为免费软件对外发布。Sun公司正式发布的有关Java平台标准版的第一批源代码，以及Java迷你版的可执行源代码。从2007年3月起，全世界所有的开发人员均可对Java源代码进行修改。

2009年，甲骨文公司宣布收购Sun。2010年，Java编程语言的共同创始人之一詹姆斯·高斯林从Oracle公司辞职。2011年，甲骨文公司举行了全球性的活动，以庆祝Java7的推出，随后Java7正式发布。2014年，甲骨文公司发布了Java8正式版。

## 核心优势

跨平台、可移植性。

## JDK

Java Development Kit开发环境

## JRE

Java Runtime Environment：java虚拟机、库函数、运行java应用所必须的文件。运行环境

## JVM

Java Virtual Machine java虚拟机，他定义了指令集、寄存器集、结构栈、垃圾收集堆、内存区域。JVM负责将java字节码解释运行，边解释边运行，这样速度就会受到一定的影响。

不同的操作系统你那个有不同的虚拟机，java虚拟机机制屏蔽了底层运行平台的差别，实现了“一次编译，随处运行”。

## 标识符

标识符是用来给变量、类、方法及包进行命名的

标识符必须以字母、下划线、美元\$开头

标识符其他部分可以是字母、下划线“\_”、美元\$和数字的任意组合

java标识符大小写敏感，且长度无限制

标识符不可以是java的关键字

### 标识符的驼峰命名规则

表示类名的标识符：每个单词的首字母大写：如Man，GoodMan，NiceTry

表示方法和变量的标识符：第一个单词小写，从第二个单词开始首字母大写

java不采用通常语言使用的ASCII字符集，而是采用Unicode这样标准的国际字符集，因此，这里字母的含义不仅仅是英文，还包括汉字等等，但是不建议使用汉字来定义标识符。

## 变量

### 局部变量

声明位置：方法或语句块内部，

从属于方法/语句块，

生命周期（作用域）：从生命位置开始，知道方法或语句块执行完毕，局部变量消失。

### 成员变量（实例变量）

声明位置：类内部，方法外部

从属于：对象

生命周期：对象创建，成员变量也跟着创建，对象消失，成员变量也跟着消失。

## 静态变量

声明位置：类内部，static修饰

从属于：类

生命周期：类被加载，静态变量就有效，类被卸载，静态变量消失。

## 常量 (Constant)

类似于专用车位，利用关键字 `final` 修饰。

常量只能初始化一次，只要定义了就不能再更改

如 `final double PI = 3.14`

## 数据类型

### 基本数据类型

**数值型：**byte(1字节)、short(2字节)、int(4字节)、long(8字节)、float(4字节)、double(8字节)

**字符型：**char

**布尔型：**boolean

一个字节(byte)=八位(bit)

### 类型转换

1):小类型的变量赋值给大类型，会自动转换

2):大类型的变量赋值给小类型，强制转换

语法：在变量前添加要转换的类型

```
int i = 11;
```

```
short s = (short)i;
```

```
long l = i;
```

### 引用数据类型

类，接口，数组

引用类型指向一个对象，不是原始值，指向对象的变量是引用变量

在java里面除去基本数据类型的其他类型都是引用类型，自己定义的class类都是引用类型，可以像基本类型一样使用。

引用类型常见的有：String、StringBuffer、ArrayList、HashSet、HashMap等。

String也属于引用数据类型

如果要对比两个对象是否相同，则需要使用equals()方法。

**注意！！：**equals()方法的默认行为是比较引用，如果是自己写的类，应该重写equals()方法来比较对象的内容，大多数java类库都实现了比较对象内容的equals()方法。

## 运算符优先级问题

运算符	结合性
[ ] . ( ) (方法调用)	从左向右
! ~ ++ -- +(一元运算) -(一元运算)	从右向左
* / %	从左向右
+ -	从左向右
> >>>	从左向右
< >= instanceof	从左向右
== !=	从左向右
&	从左向右
^	从左向右
	从左向右
&&	从左向右
	从左向右
?:	从右向左
=	从右向左

一个特殊的例子：

```
public class stlye
{
    public static void main(String[] args)
    {
        int a=10,b=6;
        System.out.println("改变之前的数: a="+a+",b="+b);
        a-=b++;
        System.out.println("改变之后的数: a="+a+",b="+b);
    }
}
```

运算结果为：

改变之前的数: a=10,b=6

改变之后的数: a=4,b=7

因为b++运算中先执行++, 再返回后置++运算表达式(b++)的返回值 (6) 给-=运算符。

在这个程序中a-=b++等于a=a-b++=10-6,所以a=4

## 条件运算符 (三目运算)

语法格式 `x?y:z` 其中x为boolean类型表达式, 先计算x的值, 若为true, 则整个运算的结果为表达式y的值, 否则整个运算结果为表达式z的值

```
int score = 80;
int x = -100;
String type = score < 60 ? "及格" : "不及格";
int flag = x > 0 ? 1 : (x == 0 ? 0 : -1);
System.out.println("type=" + type);
System.out.println("flag=" + flag);
```

输出: type=及格

flag=-1

## 使用Scanner获得键盘的输入

```
import java.util.Scanner;
Scanner scanner = new Scanner(System.in);
String str1 = scanner.nextLine();
System.out.println(str1);
```

## break语句和continue语句

break: 终止循环并跳出整个循环

continue: 终止本次循环并直接开始下一次循环

## 方法重载 (overload)

方法的重载是指一个类中可以定义多个方法名相同, 但参数不同的方法。调用时, 会根据不同的参数自动匹配对应的方法

构成方法重载的条件:

不同的含义: 形参类型、形参个数、形参顺序不同

只有返回值不同不构成方法的重载。如int a(int a)和void a(int a)不构成方法重载

只有形参的名称不同, 不构成方法的重载

```

public class TestOverload {
    public static int add(){
        return 0;
    }
    public static int add(int a,int b){
        return a+b;
    }
    public static void main(String[] args) {
        System.out.println(add());
        System.out.println(add(1,3));
    }
}

```

输出：0

4

## 递归

递归是一种常见的解决问题的方法，即把问题逐渐简单化，递归的基本思想就是“自己调用自己”，一个使用递归技术方法将会字节或者简洁的调用自己

定义递归头：什么时候不调用自身方法，如果没有头，将陷入死循环，也就是递归的约束条件

递归体：什么时候需要调用自身方法

求10的阶乘：

```

public class TestRecursion {
    static int sum = 1;
    public static void main(String[] args) {
        System.out.println(factorial(10));
    }
    public static int factorial(int a){
        if (a == 1){
            return 1;
        }
        return a * factorial(a-1);
    }
}

```

结果：3628800

## 面向对象

### 表格结构和类结构

我们在现实生活中，思考问题、发现问题、处理问题，往往都会用“表格”作为工具。实际上，“表格思维”就是一种典型的面向对象思维。

实际上，互联网上所有的数据本质上都是“表格”，总表格表示数据开始，引入对象和类，就会发现，原来“表格就是对象”。

类：对应表格结构，对象：表格内容



## 面向过程和面向对象的区别

面向过程和面向对象都是对软件分析、设计和开发的一种思想，它知道着人们以不同的方式去分析、设计和开发软件。早期先有面向过程思想，随着软件规模的扩大，问题复杂性的提高，面向过程的弊端越来越明显的显示出来，出现了面向对象思想并成为目前主流的方式。两者都贯穿于软件分析、设计和开发各个阶段，对应面向对象就分别称为面向对象分析（OOA）、面向对象设计（OOD）和面向对象编程（OOP）。C语言是一种典型的面向过程语言，java是一种典型的面向对象语言。

面向过程思想思考问题时，首相思考“怎么按步骤实现？”，面向过程适合简单、不需要协作的食物，重点关注如何执行。

面向对象（Oriented-Object）思想更契合人的思维模式，首先思考的是“怎么设计这个事物？”比如思考造车，首先思考“车怎么设计？”，而不是“怎么按照步骤造车的问题”。这就是思维方式的转变。

因此，面向对象可以帮助我们从宏观上把握、从整体上分析整个系统，但是，具体到实现部分的微观操作（就是一个个方法），仍然需要面向过程的思路去处理。

**面向过程是一种“执行者思维”**

**面向对象是一种“设计者思维”**

# 封装

## 什么是封装

封装（Encapsulation）是面向对象方法的重要原则，就是把对象的属性和操作（或服务）结合为一个独立的整体，并尽可能隐藏对象的内部实现细节。

- 将类的某些信息隐藏在类的内部，不允许外部程序进行直接的访问调用。
- 通过该类提供的方法来实现对隐藏信息的操作和访问。
- 隐藏对象的信息。
- 留出访问的对外接口。

举个比较通俗的例子，比如我们的USB接口。如果我们需要外设且只需要将设备接入USB接口中，而内部是如何工作的，对于使用者来说并不重要。而USB接口就是对外提供的访问接口。

说了这么多，那为什么使用封装？

## 封装的特点

- 对成员变量实行更准确的控制。
- 封装可以隐藏内部程序实现的细节。
- 良好的封装能够减少代码之间的耦合度。
- 外部成员无法修改已封装好的程序代码。
- 方便数据检查，有利于保护对象信息的完整性，同时也提高程序的安全性。
- 便于修改，提高代码的可维护性。

## 封装的使用

- 使用private修饰符，表示最小的访问权限。
- 对成员变量的访问，统一提供setXXX，getXXX方法。

一个Student实体对象类：

```
public class Student implements Serializable {
```



```
private Long id;
private String name;
private Integer sex;

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public Integer getSex() {
    return sex;
}

public void setSex(Integer sex) {
    this.sex = sex;
}
}
```

分析：对于上面的一个实体对象，我想大家都已经很熟悉了。将对象中的成员变量进行私有化，外部程序是无法访问的。但是我们对外提供了访问的方式，就是set和get方法。

而对于这样一个实体对象，外部程序只有赋值和获取值的权限，是无法对内部进行修改，因此我们还可以在内部进行一些逻辑上的判断等，来完成我们业务上的需要。

## 继承

### 什么是继承

继承就是子类继承父类的特征和行为，使得子类对象（实例）具有父类的实例域和方法，或子类从父类继承方法，使得子类具有父类相同的行为。当然，如果在父类中拥有私有属性(private修饰)，则子类是不能被继承的。

### 继承的特点

#### 1, 关于继承的注意事项：

只支持单继承，即一个子类只允许有一个父类，但是可以实现多级继承，及子类拥有唯一的父类，而父类还可以再继承。

子类可以拥有父类的属性和方法。

子类可以拥有自己的属性和方法。

子类可以重写覆盖父类的方法。

**继承的特点：**

提高代码复用性。

父类的属性方法可以用于子类。

可以轻松的定义子类。

使设计应用程序变得简单。

## 继承的使用

**1, 在父子类关系继承中, 如果成员变量重名, 则创建子类对象时, 访问有两种方式。**

a, 直接通过子类对象访问成员变量

等号左边是谁, 就优先使用谁, 如果没有就向上找。

b, 间接通过成员方法访问成员变量

该方法属于谁, 谁就优先使用, 如果没有就向上找。

```
public class FU {
    int numFU = 10;
    int num = 100;
    public void method(){
        System.out.println("父类成员变量: "+numFU);
    }
    public void methodFU(){
        System.out.println("父类成员方法!");
    }
}
```

```
public class Zi extends FU{
    int numZi = 20;
    int num = 200;
    public void method(){
        System.out.println("子类成员变量: "+numFU);
    }
    public void methodZi(){
        System.out.println("子类方法! ");
    }
}
```

```
public class ExtendDemo {
    public static void main(String[] args) {
        FU fu = new FU();
        // 父类的实体对象只能调用父类的成员变量
        System.out.println("父类: " + fu.numFU);    // 结果: 10

        Zi zi = new Zi();
        System.out.println("调用父类: " + zi.numFU); // 结果: 10
        System.out.println("子类: " + zi.numZi);    // 结果: 20

        /** 输出结果为200, 证明在重名情况下, 如果子类中存在则优先使用,
         *  如果不存在则去父类查找, 但如果父类也没有那么编译期就会报错。
         */
        System.out.println(zi.num); // 结果: 200
    }
}
```

```

        * 通过成员方法调用成员变量
        */
        zi.method();    // 结果: 10
    }
}

```

## 2, 同理:

成员方法也是一样的, 创建的对象是谁, 就优先使用谁, 如果没有则直接向上找。

### 注意事项:

无论是成员变量还是成员方法, 如果没有都是向上父类中查找, 绝对不会向下查找子类的。

## 3, 在继承关系中, 关于成员变量的使用:

局部成员变量: 直接使用

本类成员变量: `this.成员变量`

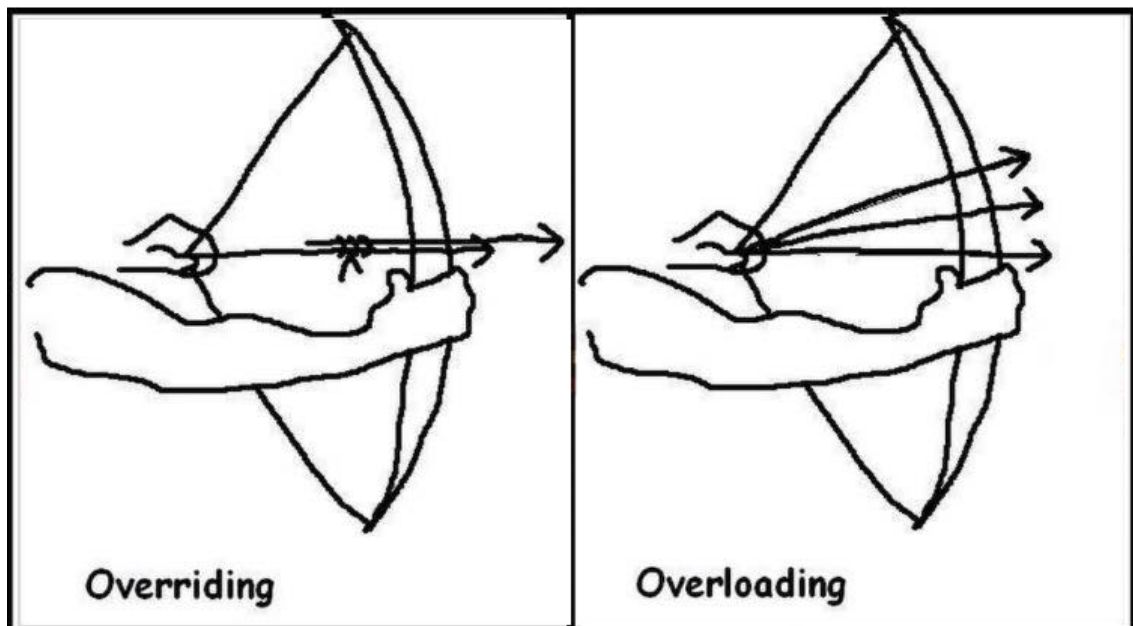
父类成员变量: `super.父类成员变量`

```

int numZi = 10;
public void method() {
    int numMethod = 20;
    System.out.println(numMethod);    // 访问局部变量
    System.out.println(this.numZi);   // 访问本类成员变量
    System.out.println(super.numFu);   // 访问本类成员变量
}

```

## 重写, 重载



### 重写(override)

是子类对父类的允许访问的方法的实现过程进行重新编写, 返回值和形参都不能改变。即外壳不变, 核心重写!

```

class Animal{
    public void move(){
        System.out.println("动物行走!");
    }
}

class Dog extends Animal{

```

```

    public void move(){
        System.out.println("狗可以跑和走");
    }
}

public class TestDog{
    public static void main(String args[]){
        Animal a = new Animal(); // Animal 对象
        Animal b = new Dog(); // Dog 对象
        a.move();// 执行 Animal 类的方法
        b.move();// 执行 Dog 类的方法
    }
}

```

重写的规则：

- 1, 参数列表必须与被重写方法相同。
- 2, 访问权限不能比父类中被重写的方法的访问权限更低 (public>protected>(default)>private) 。
- 3, 父类成员的方法只能被它的子类重写。
- 4, 被final修饰的方法不能被重写。
- 5, 构造方法不能

#### 重载(overload)

是在一个类里面，方法名字相同，而参数不同。返回类型可以相同也可以不同。每个重载的方法（或者构造函数）都必须有一个独一无二的参数类型列表。

最常用的地方就是构造器的重载。

```

public class Overloading {
    public int test(){
        System.out.println("test1");
        return 1;
    }
    public void test(int a){
        System.out.println("test2");
    }
    //以下两个参数类型顺序不同
    public String test(int a,String s){
        System.out.println("test3");
        return "returntest3";
    }
    public String test(String s,int a){
        System.out.println("test4");
        return "returntest4";
    }
    public static void main(String[] args){
        Overloading o = new Overloading();
        System.out.println(o.test());
        o.test(1);
        System.out.println(o.test(1,"test3"));
        System.out.println(o.test("test4",1));
    }
}

```

重载规则：

- 1, 被重载的方法必须改变参数列表 (参数个数或者类型不一样)。
- 2, 被重载的方法可以改变返回类型。
- 3, 被重载的方法可以改变访问修饰符。

this, super关键字

super()关键字的用法

- 1, 子类的成员方法中, 访问父类的成员变量。
- 2, 子类的成员方法中, 访问父类的成员方法。
- 3, 子类的构造方法中, 访问父类的构造方法。

this关键字用法:

- 1, 本类成员方法中, 访问本类的成员变量。
- 2, 本类成员方法中, 访问本类的另一个成员方法。
- 3, 本类的构造方法中, 访问本类的另一个构造方法。

注意:

- this关键字同super一样, 必须在构造方法的第一个语句, 且是唯一的。
- this与super不能同时存在。

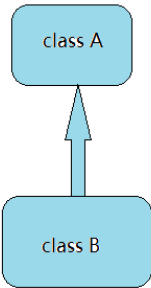
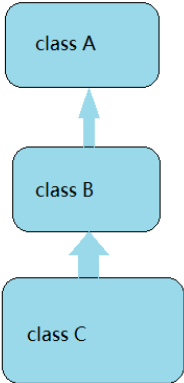
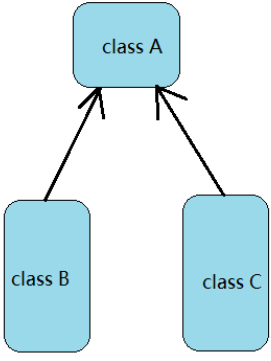
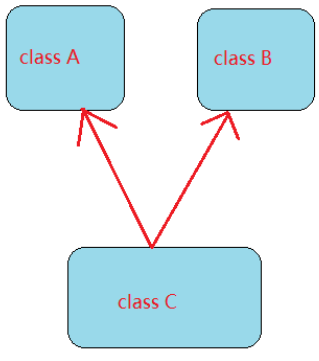
构造器

继承关系中, 父子类构造方法的访问特点:

- 1, 在子类构造方法中有一个默认隐含的super();调用, 因此一定是先调用父类构造方法, 再调用子类构造方法。
- 2, 子类构造可以通过super();调用父类的重载构造。(重载)
- 3, super();的父类调用构造方法, 必须在子类构造中的第一行, 就是第一个;号结束的元素, 并且只能调用一次。

关于继承的注意事项:

- 1, Java语言是单继承的, 一个子类只能有唯一——一个父类
- 2, Java语言可以是多级继承, 一个子类有一个父类, 一个父类还可以有一个父类。
- 3, 一个子类只有一个父类, 但是一个父类可以有多个子类。

单继承 public class A{} public class B extends B{} 	多级继承 public class A{} public class B extends A{} public class C extends B{} 	同一个父类多个子类 public class A{} public class B extends A{} public class C extends A{} 	<del>多继承 public class A{} public class B{} public class C extends A,B{} </del> <div>错误, Java不支持多继承</div>
--	---	---	--

# 多态

## 什么是多态

多态是同一个行为具有多个不同表现形式或形态的能力。

## 多态的特点

- 1, 消除类型之间的耦合关系, 实现低耦合。
- 2, 灵活性。
- 3, 可扩充性。
- 4, 可替换性。

## 多态的体现形式

- 继承
- 父类引用指向子类
- 重写

**注意：在多态中，编译看左边，运行看右边**

```
public class MultiDemo {
    public static void main(String[] args) {
        // 多态的引用，就是向上转型
        Animals dog = new Dog();
        //调用子类方法。输出狗在吃骨头
        dog.eat();

        Animals cat = new Cat();
        cat.eat();

        // 如果要调用父类中没有的方法，则要向下转型
        Dog dogDown = (Dog)dog;
        dogDown.watchDoor();
    }
}

class Animals {
    public void eat(){
        System.out.println("动物吃饭！");
    }
}

class Dog extends Animals{
    public void eat(){
        System.out.println("狗在吃骨头！");
    }
    public void watchDoor(){
        System.out.println("狗看门！");
    }
}

class Cat extends Animals{
    public void eat(){
        System.out.println("猫在吃鱼！");
    }
}
```

```
}
```

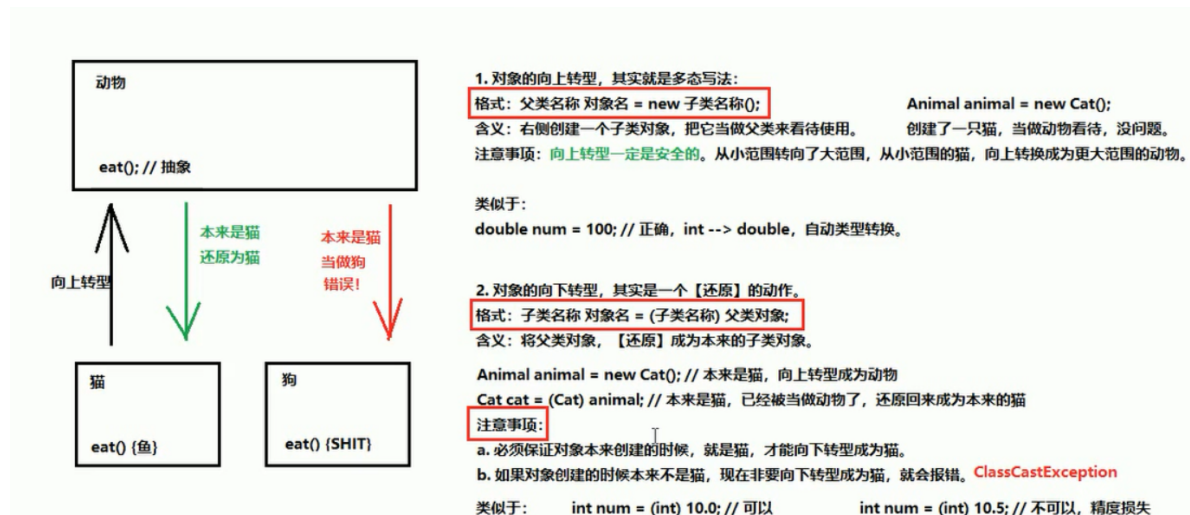
## 向上转型

1, 格式: 父类名称 对象名 = new 子类名称();

含义: 右侧创建一个子类对象, 把它当作父类来使用。

注意: 向上转型一定是安全的。

缺点: 一旦向上转型, 子类中原本特有的方法就不能再被调用了。



## 接口

### 问题描述:

现在接口中需要抽取一个公有的方法, 用来解决默认方法中代码重复的问题。  
但是这个共有的方法不能让实现类实现, 所以应该设置为私有化。

### 在JDK8之后:

- 1, default修饰, 接口里允许定义默认的方法, 但默认方法也可以覆盖重写。
- 2, 接口里允许定义静态方法。

### 在JDK9之后:

- 1, 普通私有方法, 解决多个默认方法之间代码重复的问题。
- 2, 静态私有化, 解决多个静态方法之间代码重复问题。

### 接口的注意事项:

- 1, ==不能通过接口的实现类对象去调用接口中的静态方法。==
- 正确语法: 接口名称调用静态方法。

### 接口当中的常量的使用:

- 1, 接口当中定义的常量: 可以省略public static final。
- 2, 接口当中定义的常量: 必须进行赋值。
- 3, 接口当中定义的常量: 常量的名称要全部大写, 多个名称之间使用下划线进行分割。

### 使用接口的注意事项:

- 1, 接口是没有静态代码块或者构造方法
- 2, 一个类的直接父类是唯一的, 但是一个类可以同时实现多个接口。
- 3, 如果实现类没有覆盖重写接口中所有的抽象方法, 那么实现类就必须是一个抽象类
- 4, 如果实现类中实现多个接口, 存在重复的抽象方法, 那么只需要覆盖重写一次即可。
- 5, 在Java中, 如果实现类的直接继承父类与实现接口发生冲突时, 父类优先级高于接口。



## 接口之间的关系：

- 1, 多个接口之间是继承关系。
- 2, 多个父接口当中默认方法如果重复, 那么子接口必须进行默认方法的覆盖重写。

# 抽象

在面向对象的概念中, 所有的对象都是通过类来描绘的, 但是反过来, 并不是所有的类都是用来描绘对象的, 如果一个类中没有包含足够的信息来描绘一个具体的对象, 这样的类就是抽象类。

抽象类除了不能实例化对象之外, 类的其它功能依然存在, 成员变量、成员方法和构造方法的访问方式和普通类一样。

由于==抽象类不能实例化对象==, 所以==抽象类必须被继承, 才能被使用==。也是因为这个原因, 通常在设计阶段决定要不要设计抽象类。

父类包含了子类集合的常见的方法, 但是由于父类本身是抽象的, 所以不能使用这些方法。

在 Java 中抽象类表示的是一种继承关系, 一个类只能继承一个抽象类, 而一个类却可以实现多个接口。

## 抽象类

在 Java 语言中使用 abstract class 来定义抽象类。如下实例：

```
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;
    }
}

```

## 继承抽象类

```

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;
    }
}

```

尽管我们不能实例化一个 Employee 类的对象，但是如果我们实例化一个 Salary 类对象，该对象将从 Employee 类继承 7 个成员方法，且通过该方法可以设置或获取三个成员变量。

```

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("\n Call mailCheck using Employee reference--");
        e.mailCheck();
    }
}

```

以上程序编译运行结果如下：

```

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.

```

## 抽象方法

如果你想设计这样一个类，该类包含一个特别的成员方法，该方法的具体实现由它的子类确定，那么你可以在父类中声明该方法为抽象方法。

Abstract 关键字同样可以用来声明抽象方法，抽象方法只包含一个方法名，而没有方法体。

抽象方法没有定义，方法名后面直接跟一个分号，而不是花括号。

```

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

    public abstract double computePay();

    //其余代码
}

```

声明抽象方法会造成以下两个结果：

- 如果一个类包含抽象方法，那么该类必须是抽象类。
- 任何子类必须重写父类的抽象方法，或者声明自身为抽象类。

==继承抽象方法的子类必须重写该方法。否则，该子类也必须声明为抽象类。==最终，必须有子类实现该抽象方法，否则，从最初的父类到最终子类都不能用来实例化对象。

如果Salary类继承了Employee类，那么它必须实现computePay()方法：

```
public class Salary extends Employee
{
    private double salary; // Annual salary

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

    //其余代码
}
```

## 抽象类总结规定

- 1. 抽象类不能被实例化(初学者很容易犯的错), 如果被实例化, 就会报错, 编译无法通过。只有抽象类的非抽象子类可以创建对象。
- 2. ==抽象类中不一定包含抽象方法, 但是有抽象方法的类必定是抽象类。==
- 3. 抽象类中的抽象方法只是声明, 不包含方法体, 就是不给出方法的具体实现也就是方法的具体功能。
- 4. 构造方法, 类方法(用 static 修饰的方法)不能声明为抽象方法。
- 5. 抽象类的子类必须给出抽象类中的抽象方法的具体实现, 除非该子类也是抽象类。

## 抽象类与接口区别

抽象类是用来捕捉子类的通用特性的, 而接口则是抽象方法的集合; 抽象类不能被实例化, 只能被用作子类的超类, 是被用来创建继承层级里子类的模板, 而接口只是一种形式, 接口自身不能做任何事。

其次, 抽象类可以有默认的方法实现, 子类使用extends关键字来继承抽象类, 如果子类不是抽象类的话, 它需要提供抽象类中所有声明方法的实现。而接口完全是抽象的, 它根本不存在方法的实现, 子类使用关键字implements来实现接口, 它需要提供接口中所有声明方法的实现。

抽象类可以有构造器, 除了不能实例化抽象类之外, 它和普通的java类没有任何区别, 抽象方法可以有public、protected和default这些修饰符。而接口不能有构造器, 是完全不同的类型, 接口方法默认修饰符是public, 不可以使用其他修饰符。