# 《 Java 教程 》

　　　　　　　　　　　　　　　　　　　　　　　Sharon Zakhour Scott Hommel 著

# 第一章 入门

参考：<https://docs.oracle.com/javase/tutorial/getStarted/index.html>

　　在Java编程语言中,首先将所有源代码都写入纯文本文件中.以.java扩展名结束。 然后使用javac编译器把这些源文件编译成.class文件。

　　.class文件不包含对应于处理器的本机代码; 而是包含**字节码**——Java虚拟机的机器语言。然后java启动工具通过Java虚拟机的一个实例

运行应用程序。

# Java技术能做什么呢?

通用,高级Java编程语言是一个功能强大的软件平台。Java平台的每一个完整都能提供了以下功能:

* **开发工具**:开发工具提供进行编译、运行、监控、调试应用程序和编写应用程序文档所需要的所有功能。 作为一个新的开发人员,您将使用的主要工具javac编译器,java启动器,javadoc文档工具。
* **应用程序编程接口(API)**:这个API提供了Java编程语言的核心功能。 它提供了大量已经编写号的类，可以用于您自己的应用程序。 它跨越了从基本的对象,网络和安全,XML生成和数据库访问,等等。 核心API非常大; 它包含的概述,请参考[Java平台标准版本8文档](https://docs.oracle.com/javase/8/docs/index.html" \t "_blank)。
* **部署技术**: JDK软件提供了标准机制，比如Java Web Start软件和Java Plug-In软件，用于把应用程序部署给最终用户。
* **用户界面工具包**JavaFX,Swing和Java 2D工具包可以创建复杂的图形用户界面(GUI)。
* **集成库**:集成库如Java IDL API、JDBC API,Java命名和目录接口(JNDI)API、Java RMI和Java远程方法调用网络orb间协议技术(Java RMI - IIOP技术)实现数据库访问和操作远程对象。

## Netbean IDE的进一步介绍

使用IDE的Compile File（F9）和Run file（Shift-F6）命令，可以编译和运行单个文件（而不是整个工程）。如果使用Run Main Project命令，IDE就会运行被IDE关联为主工程的主类的文件。因此，如果在HelloWorldApp工程中创建了额外的类，然后试图使用Run Main Project命令运行这个文件，那么IDE就会运行HelloWorldApp，而不是这个文件。

## 源代码注释

1../\* text \*/

2./\*\* documentation \*/ 这表示文档注释。当Javadoc工具准备自动生成的文档时，他会使用文档注释。

3.//

## HelloWorldApp类定义

关键字class开始名为name的类的类定义。

**main方法的定义**： public static void main（String[] args）

main方法接受单一参数：String类型的元素的数组

# 第二章 面向对象的编程概念

## 对象是什么

软件对象在概念上类似于现实世界对象：它们也由状态和相关的行为构成。**对象把它的状态保存在字段（field）中，并通过方法（methods）/函数 实现它的行为。**

## 类是什么（[What Is a Class?](https://docs.oracle.com/javase/tutorial/java/concepts/class.html)）

类是创建独立对象的蓝图 ：Bicycle类没有包含main方法。这是应为他不是完整的应用程序，它可能只用于其他应用程序中的自行车的蓝图。创建和使用新的Bicycle对象的责任属于你的应用程序中的其他一些类。

共同行为可以定义在超类（superclass）中，并且使用extend关键字继承到子类（subclass）中。

字符串由**java.lang.String**.类表示。

## 继承是什么（What Is Inheritance?）

创建子类的语法很简单，在类定义的开头使用关键字**extends**，后面是被继承的类的名称。

## 接口是什么（What Is an Interface?）

接口常常是一组具有空方法体的相关方法。如果指定为接口，那么自行车的行为可能像下面这样：

interface Bicycle {

// wheel revolutions per minute

void changeCadence( int newValue );

void changeGear( int newValue );

void speedUp( int increment );

void applyBrakes( Int decrement )；

}

为了实现这个接口，应该改变类的名称（比如改为ACMEBicycle），并且需要在类定义中使用关键字**implements**：

class ACMEBicycle **implements** Bicycle {

int cadence = 0;

int speed = 0;

int gear = 1;

// The compiler will now require that methods

// changeCadence, changeGear, speedUp, and applyBrakes

// all be implemented. Compilation will fail if those

// methods are missing from this class.

void changeCadence(int newValue) {

cadence = newValue;

}

void changeGear(int newValue) {

gear = newValue;

}

void speedUp(int increment) {

speed = speed + increment;

}

void applyBrakes(int decrement) {

speed = speed - decrement;

}

void printStates() {

System.out.println("cadence:" +

cadence + " speed:" +

speed + " gear:" + gear);

}

}

通过实现接口，类可以更加正式的说明它承诺提供的行为。接口构成类和外部世界之间的契约，并且这个契约在编译时由编译器强制实施。如果你的类声明实现一个接口，那么这个接口的所有方法都必须出现在其源代码中，否则类就不能成功编译。

## 包是什么（What Is a Package?）

包是组织一组相关类和接口的名称空间。

# 第三章 语言基础

## 变量

实例变量（非静态字段）、类变量（静态字段）、局部变量、参数。

### 命名

如果你的变量用于储存常量值，比如static final int NUM\_GEARS = 6 ,那么惯例就稍有变化，每个字母都大写，并且使用下划线分割后续的单词。

## 特殊的char和String字面量的转义序列

\b（退格)、\t（制表符）、\n(换行)、\f（换页）、\r（回车）、\’’（双引号）、\’（单引号）、[\\（反斜线](file:///\\（反斜线)）。

特殊类型的字面量——类字面量（class literal），它的构成方法是类名称加后缀“.class”,比如String.class。这引用便是类型本身的对象（类型为class）。

## 数组（array）

声明： type[ ] arrayname 或 type arrayname[ ] 后者不推荐：括号标识数组类型，应和类型签名一起出现。

创建数组的一个途径是使用new操作符：anArray = new int[10];

**复制数组：**System类有一个arraycopy方法，可以使用它把数据从一个数组复制到另一个数组。

public static void arraycopy(Object src, int srcPos,

Object dest, int destPos, int length)

两个Objec参数分别指定被复制的数组和复制到的数组。三个int参数分别指定源数组中的开始位置.、目的数组中的开始位置、要复制的数组的数组元素的数量。T

## 操作符

**三元操作符**

？：可以把它看作if-then-else语句的简写方式。

Result=someCondition ？ value1 ： value2 ; //如果someCondtion为true,就把value1的值赋给result。否则，把value1的值赋给result。

**类型比较操作符instanceof**

Instanceof操作符把对象和指定的类型进行比较。

**条件操作符**

&& 条件与 || 条件或

**位操作符**

& 位与 | 位或 ^ 异或

## 控制流语句

**Break语句：**无标签break语句终止最内层switch、for、while或do-while语句，而带标签的break语句终止外层语句。

**Continue语句**： continue语句跳过for、while或者do-while循环的当代迭代。无标签形式跳到最内层循环体的结尾，并且计算控制循环的布尔表达式。带标签的continue语句跳过标有给定标签的外层循环的当前迭代。

# 第四章 类和对象

#### 类

类声明可以包含下面组件：修饰符、类的名称（首字母大写）、类的父类、类实现的接口清单、类体。

class *MyClass extends MySuperClass implements YourInterface* {

// field, constructor, and

// method declarations

}

##### 访问修饰符

Private修饰符——可以从它自己的类中访问这个字段

Pulibc修饰符——可以从所有类访问这个字段

#### 方法（C中的函数）

public double calculateAnswer(double wingSpan, int numberOfEngines,

double length, double grossTons) {

//do the calculation here

}

方法声明必需的元素是方法的返回值、名称、一对圆括号( )以及花括号{ }之间的方法体。

更加常见的方法声明有6个组成部分，按照顺序是：

修饰符（public、private）、返回类型、方法名称、圆括号中的参数清单、异常清单、扩在方括号中的方法体。

方法签名有方法声明的两个部分构成-方法的名称和参数类型。

#### 为类提供构造器

理解构造器之前，首先我们需要知道Java中为什么要引入构造器，以及构造器的作用。假设我们每一次编写一个类都要执行一个initialize()方法，该方法是提醒你，在使用对象之前，应首先调用initialize()方法进行初始化，这就意味着每一用户都能去执行这个方法。Java中引入构造器，确保每一个对象都得到初始化，Java在有能力操作对象之前，系统会自动调用相应的构造器，保证初始化的进行。

构造器最大的用处就是在创建对象时执行初始化，当创建一个对象时，系统会为这个对象的实例进行默认的初始化。如果想改变这种默认的初始化，就可以通过自定义构造器来实现。

构造器的声明和方法的声明类似——只是他们**使用类的名称**，而且没有返回值。

public Bicycle(int startCadence, int startSpeed, int startGear) {

gear = startGear;

cadence = startCadence;

speed = startSpeed;

}

为了创建新的名为myBiked的Bicycle对象，使用new操作符调用构造器：

Bicycle myBike = new Bicycle(30, 0, 8);

##### 任意数量的实参

当你不知道有多少个特定类型的实参要传递给方法时，可以使用可变参数。为了使用可变参数，在最后一个参数的类型之后加上省略号（3个点号…），然后时空格和参数名称，然后就可以使用任意数量的参数调用这个方法。

最常使用可变参数的地方是输出方法——例如下面这个printf方法：

public PrintStream printf(String format, Object... args)

允许你输出任意数量的对象，可以这样调用它：

System.out.printf(“%s: %d, %s%n ”， name, idnum, address );

##### 参数名称

同一个方法或者构造器的任意两个参数名称不能相同，并且参数名称不能和方法或者构造器内的局部变量名称相同。

参数名称可以和一个类的字段名称相同。如果这样使用，那么参数被称为使这个字段遮蔽(shadow)。遮蔽的字段使代码难以阅读，并且按照惯例只在设置的特定字段的构造器或者方法中使用。

##### 传递基本数据类型实参

原始实参，比如int或者double，是按值(by value)传递给方法的。这就是说，对参数值的任何改动都只限于这个方法的作用域内。当方法返回时，参数消失并且对参数的所有改动都会消失。

public class PassPrimitiveByValue {

public static void main(String[] args) {

int x = 3;

// invoke passMethod() with

// x as argument

passMethod(x);

// print x to see if its

// value has changed

System.out.println("After invoking passMethod, x = " + x);

}

// change parameter in passMethod()

public static void passMethod(int p) {

p = 10;

}

}

引用这个程序的输出结果：

After invoking passMethod, x = 3

### 创建对象

通过使用new操作符和构造器从类创建对象。New操作符返回创建好的对象的引用。可以把引用赋给变量，或者直接使用它们。

New关键字是用于创建对象的Java操作符。

New操作符后面调用构造器，构造器初始化新对象。

New操作符为新的对象分配内存，并且返回对这段内存的引用，从而实例化类。New操作符还调用对象的构造器。

### 返回类或接口

当方法使用一个类作为返回类型时，那么返回对象的类型类必须时返回类型指定的类或者它的子类。

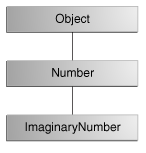
假设如图，**ImaginaryNumber** 是 java.lang.**Number**的子类, 而java.lang.Number是**Objecrt**的子类。假设你的一个方法被声明为返回Number:

public Number returnANumber() {

...

}

returnANumber方法可以返回ImaginaryNumber，但不能返回Object。因为ImaginaryNumber是Number的子类，所以它是一个Number对象。但Object对象不一定是Number对象——它可以是String或其他类型。



可以覆盖方法并且将他定义为返回原始方法的子类，就像下面这样：

public ImaginaryNumber returnANumber() {

...

}

这种技术称为协变返回类型，就是说允许返回类型和子类相同的方向变化。

也可以使用接口名称作为返回类型。在这种情况下，返回的对象必须实现指定的接口。

## 使用关键字this

在**实例方法**或**构造器**中，this引用当前对象，即被调用的方法或构造器所属的对象。通过使用this构造器，可以在实例方法或构造器中引用当前对象的任何成员。

1. **对字段使用this关键字**

使用this关键字最常见的原因是，方法或构造器的参数遮蔽了对象的成员字段。

例如，Point类写成这样：

public class Point {

public int x = 0;

public int y = 0;

**//constructor**

**public Point(int a, int b) {**

**x = a;**

**y = b;**

**}**

}

但也可以写成这样：

public class Point {

public int x = 0;

public int y = 0;

**//constructor**

**public Point(int x, int y) {**

**this.x = x;**

**this.y = y;**

**}**

}

第二个构造器的每个实参都遮蔽对象的一个字段——构造器内的x是构造器的第一个实参的局部副本。为了引用point字段x，构造器必须使用this.x 。

1. **对构造器使用this**

在构造器中，还可以使用this关键字调用同一个类中的另一个构造器。

public class Rectangle {

private int x, y;

private int width, height;

public Rectangle() {

**this(0, 0, 1, 1);**

}

public Rectangle(int width, int height) {

**this(0, 0, width, height);**

}

public Rectangle(int x, int y, int width, int height) {

this.x = x;

this.y = y;

this.width = width;

this.height = height;

}

...

}

## 控制对类成员的访问

访问控制有两个级别：

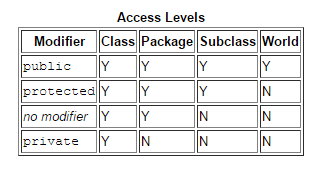
##### 最高级别——public，或者包私有（ *package-private*）（无显示修饰符）。

##### 成员级别——public、private、protected或者包私有。

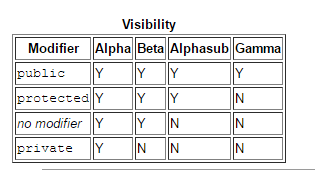
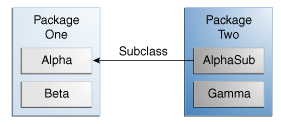
可以使用**public**声明类，在这种情况下，这个类对任何位置的所有类都是可见的。如果类没有修饰符，那么它只在直接的包内是可见的。

修饰符**private**指定成员只在它自己的类中是可以访问的。

修饰符**protected**指定成员只在它自己的包中是可以访问的，并且可以被另一个包中其类的子类访问。



上图列出每个修饰符允许的成员访问方式。



##### 了解实例成员和类成员

**类变量**

有时候，你会希望变量对所有对象是公共的。这是通过修饰符static实现的。在声明中使用**static修饰符的字段被称为静态字段或者类变量。它们和类相关，而不和任何对象相关。**

public class Bicycle {

private int cadence;

private int gear;

private int speed;

// **add an instance variable for the object ID**

private int id;

// **add a class variable for the**

// **number of Bicycle objects instantiated**

private **static** int numberOfBicycles = 0;

...

}

通过类名称本身引用类变量，例如：

Bicycle.numberOfBicycles

这就清楚的表示了他是类变量。

**类方法**

* 实例方法可直接访问实例变量和实例方法。
* 实例方法可以直接访问类变量和类方法。
* 类方法可以直接访问类变量和类方法。
* 类方法***不能***直接访问实例变量或实例方法——必须使用一个对象引用。 同时,类方法不能使用this关键字，因为没有实例可供this引用。

**常量**

Final修饰符指定这个字段的值不能改动。

## 初始化字段

#### 1.静态初始化块

在程序中任何变量或者代码都是在编译时由系统自动分配内存来存储的，而所谓静态就是指在编译后所分配的内存会一直存在，直到程序退出内存才会释放这个空间，也就是只要程序在运行，那么这块内存就会一直存在。

静态初始化块被放在花括号中，前面是static关键字。

static {

// whatever code is needed for initialization goes here

}

#### 2.初始化实例成员

通常把初始化实例变量的代码放在构造器中。对于使用构造器初始化实例变量，有两个替换方案：初始化器块和final方法。

。用于实例变量的初始化器块和静态初始化块差不多，只是前者没有static关键字：

{

// whatever code is needed for initialization goes here

}

Java编译器把初始化器块复制到每个构造器中。因此，这种方法用于在多个构造器之间共享一个代码块。

不能在子类中覆盖final方法。下面是final方法初始化实例变量的一个例子：

class Whatever {

private varType myVar = initializeInstanceVariable();

protected final varType initializeInstanceVariable() {

// initialization code goes here

}

}

### 实例

在Java程序里面，所有的东西都是对象，而对象的抽象就是类，对于一个类而言，如果要使用他的成员，那么普通情况下必须先实例化对象后，通过对象的引用才能够访问这些成员，但是有种情况例外，就是该成员是用static声明的（在这里所讲排除了类的访问控制），例如：   
未声明为static   
class ClassA{  
int b;   
public void ex1(){   
…   
}   
}   
class ClassB{   
void ex2{   
int i;   
ClassA a = new ClassA();   
i = a.b; //这里通过对象引用访问成员变量b   
a.ex1; //这里通过对象引用访问成员函数ex1   
}   
}   
声明为static   
class ClassA{   
static int b;   
static void ex1(){   
…   
}   
}   
class ClassB{   
void ex2{   
int i;   
i = ClassA.b; //这里通过类名访问成员变量b   
ClassA.ex1; //这里通过类名访问成员函数ex1   
}   
}   
通过以上两种比较，就可以知道static用来修饰类成员的主要作用了，在java类库当中有很多类成员都声明为static，可以让用户不需要实例化对象就可以引用成员，最基本的有Integer.parseInt()，Float.parseFloat()等等用来把对象转换为所需要的基本数据类型。这样的变量和方法我们又叫做类变量和类方法。   
接下来讲一下被static修饰后的变量的值的问题，刚才在前面讲过，被static修饰后的成员，在编译时由内存分配一块内存空间，直到程序停止运行才会释放，那么就是说该类的所有对象都会共享这块内存空间，看一下下面这个例子：   
class TStatic{   
static int i;   
  
public TStatic(){   
i = 4;   
}   
  
public TStatic(int j){   
i = j;   
}   
  
public static void main(String args[]){   
TStatic t = new TStatic(5); //声明对象引用，并实例化   
TStatic tt = new TStatic(); //同上   
System.out.println(t.i);   
System.out.println(tt.i);   
System.out.println(t.i);   
}   
}   
这段代码里面Tstatic类有一个static的int变量I，有两个构造函数，第一个用于初始化I为4，第二个用于初始化i为传进函数的值，在main中所传的值是5，程序首先声明对象的引用t，然后调用带参数的构造函数实例化对象，此时对象t的成员变量I的值为5，接着声明对象tt，调用无参数的构造函数，那么就把对象tt的成员变量i的值初始化为4了，注意了，在这里i是static，那就是说该类的所有对象都共享该内存，那也就是说在实例化对象tt的时候改变了i的值，那么实际上对象t的i值也变了，因为实际上他们引用的是同一个成员变量。最后打印的结果是三个4