1. **变量**

整型：byte short int long

长度： 8 16 32 64

byte：-128 127

字符型：

char a=‘样’;

浮点型：

float 32 double 64

默认小数值为double类型

foat f=5.32 错；foat f=5.32 对；

布尔型

ture false 不能用0 1 赋值

String类型 Immutable 不可变类

字面值：0x 0b 0

long i=500;仍为整型

long i=500L;

浮点数

float f=3.14f

double d=3.14

double e=314e-2

字符串、字符

制表符:\t 换行

双引号‘\”’‘”’

单引号‘\’’

反斜杠‘\\’

数据类型转换

从小到大自动转，从大到小强制转

byte-short-int-long-float-double

byte-char-int

强转

byte-short-int-long-float-double

8 16 32 64 32 64

Short char 16位

高精度向低精度转换有风险，如：

byte b；int a=300;b=(byte)a; 输出b为44

1B=8bit

300：int 00000000 00000000 00000001 00101100

byte： 00101100

精度小于int的数值运算的时候都回被自动转换为int后进行计算

整数运算：

      如果两个操作数有一个为long，则结果也为long；

      没有long时，结果为int。即使操作数全为short、byte，结果也是int。

浮点运算：

       如果两个操作数有一个为double，则结果为double；

       只有两个操作数都是float，则结果才为float。

变量命名：第一个是字母或者$或者\_

变量命名只能使用字母 数字 $ \_

关键字列表：

异常处理：try catch finally throw throws

对象相关：new extends implements class instanceof this super

字面值常量：false true null

方法相关：return void

包相关：package import

未使用的：const goto

基本数据类型、循环关键字、分支关键字

方法、变量和类修饰符：strictfp synchronized abstract transient public private protected final static

字段、属性 、成员变量、Field：一个变量在类中声明

参数：在方法上声明(函数)

局部变量：在方法内声明 声明开始位置到块结束位置

Final修饰变量 该变量只能赋值一次 还可以修饰类、修饰方法

如：final int a=1; a=2;报错

public static void main(String[] args) { //主方法对应的块}

1. **运算符和操作符**

Scanner类的使用

Import java.util.Scanner;

Scanner s=new Scanner(System.in);

int a=s.nextln();读取整数

float b=s.nextFloat();读取浮点数

String c=s.nextLine();读取字符串

注：通过nextInt()读取了整数后，再接着读取字符串，读出来的是回车换行:"\r\n",因为nextInt仅仅读取数字信息，而不会读取回车换行"\r\n".

故：

 Scanner s = **new** Scanner(System.in);

**int** i = s.nextInt();

        System.out.println("读取的整数是"+ i);

        String rn = s.nextLine();

        String a = s.nextLine();

        System.out.println("读取的字符串是："+a);

int+long=long

int+long=(int)

注意：float a,b,c;

c=a/b/b 不等于c=a/(b\*b)

关系操作符：

> >= < <= == !=

System.out.println(a==b) 结果 false true

逻辑操作符：

长路与& 短路与&&：均为真才为真 不同在于 短路与只要第一个是false 第二个就不再进行运算

 长路与  无论第一个表达式的值是true或者false,第二个的值，都会被运算

        int i = 2;

        System.out.println( i== 1 & i++ ==2  ); //无论如何i++都会被执行，所以i的值变成了3

短路与 只要第一个表达式的值是false的，第二个表达式的值，就不需要进行运算了

        int j = 2;

        System.out.println( j== 1 && j++ ==2  );  //因为j==1返回false,所以右边的j++就没有执行了，所以j的值，还是2

长路或| 短路或|| 同理 短路或只要第一个是true，第二个式子就不运算了

int i = 2;

        System.out.println( i== 1 | i++ ==2  ); //无论如何i++都会被执行，所以i的值变成了3

  int j = 2;

        System.out.println( j== 2 || j++ ==2  );  //因为j==2返回true,所以右边的j++就没有执行了，所以j的值，还是2

 ! 取反

^异或

位操作符：Integer.toBinaryString() 方法，将一个十进制整数转换为一个二进制字符串

|位或 &位与 ^异或 ~取非 <<左移 >>右移

任何数和自己进行异或 都等于 0  
任何数和0 进行异或 都等于自己

I|j

左移n位，即乘以2的n次方

I&j

I^j

5|6==7

带符号右移>> 无符号右移>>>

>>:正数：所有的位右移，并在最前面补0；负数：所有的位右移，并在最前面补1

>>>:负数，对应的二进制的第一位1也右移，第一位补0，使负数在无符号右移后得到一个正数

无符号右移移动后变正

如：int i=-10; int j=i>>1; int k=i>>>1;

-10带符号右移1位，移动后前面补齐1,第一位是1，所以依然是一个负数，对应的十进制是-5

无符号向右移1位，符号位也会向右移，第一位就变成了0,对应的十进制是2147483643

int i = 3; // 二进制是11  
int j = 2; // 二进制是10  
int c = ((i | j) ^ (i & j)) << 2 >>> 1;

其中 c=2

三元操作符 ?:

表达式?值1:值2  
如果表达式为真 返回值1  
如果表达式为假 返回值2

1. **类与对象**

变量类型：类，则该变量又叫做引用

多引用指向一个对象，如:

类hero中：hero h1=new hero();

hero h2=h1;hero h3=h1;hero h4=h1;即四个引用指向同一个对象

一个引用 多个对象 如：

hero h1=new hero();//对象1

h1=new hero();//对象2

一个引用在某时刻只能指向一个对象

继承：public class Weapon extends Item

重载：方法名一样 参数类型不一样

可变数量的参数

可变的参数类型 varargus ...

如:int... intArray

构造方法：隐式、有参

同时出现时无参的构造方法失效

this类似我 指代当前对象

System.out.println("打印this看到的虚拟地址："+**this**)

通过this关键字访问对象的属性

this.属性=形参

this(属性);this(属性,属性,属性);

=用于给类类型变量复制，=为指向，如 Book m=new Book()中，m为引用。引用m指向一个Book对象

import 包.类

访问修饰符：private public protected

package/friendly/default 不写

没有修饰符即代表package/friendly/default

类和类的关系：自身、同包子类、不同包子类、同包类、其他类【在不同包也无继承关系】

子类有继承关系

无修饰符同包子类可以继承 不同包子类不可以继承

Protected可以自身访问、同包或不同包子类均可继承、同包类可以访问、不同包类不可以访问

Public均可以被访问和继承

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 自身 | 同包子类 | 不同包子类 | 同包类 | 其他类 |
| private | 访问 | 不 | 不 | 不 | 不 |
| Package/default/friendly | 访问 | 继承 | 不 | 访问 | 不 |
| protected | 访问 | 继承 | 继承 | 访问 | 不 |
| public | 访问 | 继承 | 继承 | 访问 | 访问 |

一般 属性private 方法public 会被子类继承的方法为protected 作用范围最小原则 private升

类属性：静态属性

对象属性：实例属性、非静态属性

类属性则所有对象共享这个值 static

访问类属性两种方式：类名.类属性(荐)、对象名.类属性

对象属性和类属性的设置，所有对象均共享则类属性

对象方法：必须建立在有对象的基础上即实例方法、非静态方法

类方法：静态方法

类方法调用方式：类名.类方法(荐)、对象名.类方法

对象属性初始化：声明该属性时初始化、构造方法中初始化、初始化块{}

如：

**public** **class** Hero {

**public** String name = "some hero"; //声明该属性的时候初始化

    {

        maxHP = 200; //初始化块

    }

**public** Hero(){

        hp = 100; //构造方法中初始化

    }

}

类属性初始化：声明该属性时初始化、静态初始化块{}

如：

public static int itemCapacity=8; //声明的时候 初始化

static{

itemCapacity = 6;//静态初始化块 初始化

}

 Singleton模式，指的是一个类，在一个JVM里，只有一个实例存在

饿汉式单例模式

private构造方法 该类在外部无法通过new进行实例化 如 private 类名(){}

**public** **static** 类名 getInstance(){//public static方法 提供给调用者获取对象

**return** instance;

    }

enum 类 用于预先定义

Public enum Season{ SPRING,SUMMER,AUTUMN,WINTER}

1. **选择与循环**

if else

if else if

Switch(){case 1: 语句1;break; case 2: 语句2;break;default: 语句}

闰年：y%400==0||y%4==0&&y%100!=0

Java从1.7开始支持switch用String的，编译后是把String转化为hash值，其实还是整数

while do...while

Continue break 增加boolean变量结束外部循环

Debug

1. **面向对象**

类和对象：类的第一个字母大写 易于观看

属性：属性一般小写 多单词则后面单词的第一个字母大写

方法

1. **接口与继承/多态**

interface 使用：implements

@override注解 可写可不写

写了即：基类必须存在方法，（控制类型public,protected，返回值，参数列表类型）与子类方法完成一致的方法，否则会报错

不写即：当基类存在与子类各种条件都符合的方法是即实现覆盖；

想覆盖基类方法时，最好还是写上@Override注解，这样有利于编译器帮助检查错误

引用类型、对象类型

若不一致需要转换 子类转换成父类是可以的

类型转换指的是把一个引用所指向的对象的类型，转换为另一个引用的类型

父类转子类 可行可不行 必须强制转换 转换有风险

父类 a;

子类 b;

a=（子类）b;

没有继承关系的子类和子类转换一定会失败

某类实现接口的方法 则可以将这个类的对象转换成接口类，向上转型，能成功

类 a接口 b=a; 此时b指向的是a类

instanceof可以判断一个引用所指的对象是否为某类型

如 对象 instanceof 类 System.out.println(h1 **instanceof** Hero);

继承重写 继承后重复提供该方法即方法的重写 又称覆盖 override

调用重写的方法会执行重写的方法而不是父类的方法

多态

操作符的多态 如+算数运算、字符串连接

类的多态：父类引用指向子类对象

整型+整型 + 数字相加

整型+字符串 字符串+整型 字符串+字符串 + 字符串连接

多态：同一个类型、调用同一个方法、呈现不同的状态

实现类的多态：父类（接口）引用指向子类对象、调用的方法有重写

方法的重写：子类覆盖父类的对象方法

隐藏：子类覆盖父类的类方法

隐藏父类的静态方法即父类的静态方法和子类的静态方法同时存在 具体调用哪个方法看盗用方法的引用指向父类还是子类

关键字 super

子类实例化时定会调用父类的构造方法 默认调用父类的无参构造方法

父类有参构造方法 需要显示调用 使用super 如 super(name);

父类仅有 有参构造方法时，子类必须用super去调用该方法 否则尝试调用无参构造方法 会失败

Ojbect类：所有类的父类 声明类时默认extends Object

所有类都有toString() 返回当前对象的字符串表达 即该对象的String值

finalize() 当一个对象没有指向任何引用指向时，满足垃圾回收的条件，该方法被调用，JVM调用

equals()用于判断两个对象的内容是否相同 如h1.equals(h2) false或true

equals()重写 类型是boolean 常与instanceof一起搭配

如：

**public** **boolean** equals(Object o) {

**if** (o **instanceof** Item) {

            Item h = (Item) o;

**return** **this**.price == h.price;

        }

**return** **false**;

    }

== 用于判断两个引用是否指向同一个对象 就算值相同但不是同一个对象 也会返回false

hashCode()

所有对象都有一个对应的hashcode散列值

查找数据首先计算hashcode在数组中定位 若该位置有多个值 用equals()进行比较

空间换时间的做法

HashSet里封装了HashMap 本质 判断HashMap的值是否重复

线程同步相关方法：wait() notify() notifyAll()

getClass会返回一个对象的类对象 参见反射机制：

<https://how2j.cn/k/reflection/reflection-reflection/107.html>

final

final修饰类、方法、基本类型变量、引用的时候分别有不同的意思

final修饰类 则该类不可以被继承 该类成为最终类；修饰方法，则该方法不可以被重写；修饰基本类型变量，表示该变量只能一次性赋值，赋值两次仍为第一次的值，且会提示错误；修饰引用，则该引用只有1次指向对象的机会，如final hero h=new hero(),但是可以h.属性来修改属性值；修饰常量，即直接访问、可以公开、不会变化 常量需要初始化

abstract 类中声明一个方法 且该方法没有实现体 是一个空方法 即抽象方法

有抽象方法时 该类必须被声明为抽象类

如

public abstract class Hero {  
 public abstract void attack();  
}

抽象类可以没有抽象方法、一个类被声明为抽象类则不能够直接实例化

抽象类和接口

区别：子类可以继承多个接口 只能继承一个抽象类

接口声明的属性只能是public、static、final 即便没有显示的声明

抽象类属性可以是public、protected、package、private；final和非final；static和非静态属性

同：接口和抽象类均有实体方法 接口中的实体方法称为默认方法

内部类：非静态内部类、静态内部类、匿名类、本地类

非静态内部类可以在一个类里面直接声明 仅当外部类对象存在时才有意义

如：

   Hero garen = **new** Hero();

        garen.name = "盖伦";

        // 实例化内部类

        // BattleScore对象只有在一个英雄对象存在的时候才有意义

        // 所以其实例化必须建立在一个外部类对象的基础之上

        BattleScore score = garen.**new** BattleScore();

静态内部类可以直接实例化 不需要外部类的类的实例为基础

语法：new 外部类.静态内部类()

静态内部类不可以访问外部类的实例属性和方法 可以访问外部类的私有静态成员 其余和普通类无大差别

static class 类名

匿名类 声明一个类的同时实例化、直接实例化一个抽象类

在匿名类中以用外部的局部变量 该局部变量必须修饰为final jdk8可以不写 编译器自动添加

如：

**public** **abstract** **class** Hero {

**public** **abstract** **void** attack();

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

        //在匿名类中使用外部的局部变量，外部的局部变量必须修饰为final

**final** **int** damage = 5;

        Hero h = **new** Hero(){

**public** **void** attack() {

                System.out.printf("新的进攻手段，造成%d点伤害",damage );

            }

        };

    }

}

本地类 可以理解为有名字的匿名类 必须声明在成员的位置 但是本地类有自定义的类名

默认方法：JDK8特性 接口可以提供具体方法 用default关键字实现

可以扩展新的类、并做到不影响原来的类

同时继承两个default方法 必须重写 override

UML图 表示类之间的关系

统一建模语言 用于描述类的属性、方法以及类和类之间的关系

类图：

|  |  |
| --- | --- |
| *Animal* | 类名 斜体表示抽象类 |
| #legs : int | 属性名legs 类型int #表示protected |
| #Animal(legs:int)  +walk()  +*eat()* | 有下划线表示构造方法Animal 普通方法walk() eat斜体表示抽象方法  + 访问修饰符表示public |

UML接口图

|  |  |
| --- | --- |
| <<interface>>  Pet | 接口名 |
| +geyname():String  +setName(name:String)  +play() | 方法 返回String  + 访问修饰符 表示public |

继承关系：带箭头的实现 模糊也是表示表示不需要关注此部分内容

虚线指向接口

abstract Animal类

默认构造器：构造函数

如 public spider（）{super(8);}

构造器使用this关键字、并传递一个空字符串作为参数

Public cat(){ this(“”); }