**Java SE**

**目录**

[一．预备知识 2](#_Toc61191014)

[JAVA SE的简单介绍 2](#_Toc61191015)

[Java 中的注释 2](#_Toc61191016)

[public class 和 class的区别 2](#_Toc61191017)

[二．JAVA语言基础 2](#_Toc61191018)

[标识符 2](#_Toc61191019)

[变量 2](#_Toc61191020)

[数据类型 3](#_Toc61191021)

[字节 5](#_Toc61191022)

[转义字符 5](#_Toc61191023)

[关于原码，反码，补码 5](#_Toc61191024)

[for语句(拓展) 6](#_Toc61191025)

[switch 语句 6](#_Toc61191026)

[三. 方法 6](#_Toc61191027)

[方法分三类 6](#_Toc61191028)

[方法的优点 7](#_Toc61191029)

[方法的本质 7](#_Toc61191030)

[方法定义的格式 7](#_Toc61191031)

[Java虚拟机中的三块主要内存空间 8](#_Toc61191032)

[方法重载机制(overload) 9](#_Toc61191033)

[四. 面向对象(Java语言的特点，核心) 10](#_Toc61191034)

[面向对象和面向过程的区别(盖浇饭和蛋炒饭) 10](#_Toc61191035)

[面向对象的三大特性 10](#_Toc61191036)

[类和对象的概念 10](#_Toc61191037)

[多态 13](#_Toc61191038)

[instanceof 运算符(双目运算符) 14](#_Toc61191039)

[final 关键字 15](#_Toc61191040)

[访问权限修饰符 15](#_Toc61191041)

[抽象类 15](#_Toc61191042)

[接口 17](#_Toc61191043)

****一．预备知识****

JAVA SE的简单介绍

JAVA SE （标准版）

JDK ：java 开发工具包

JRE ：java开发环境

JVM ：java虚拟机 (针对不同操作系统安装不同的JVM版本)

java 虚拟机（JVM）包含javac.exe 和 java.exe 命令

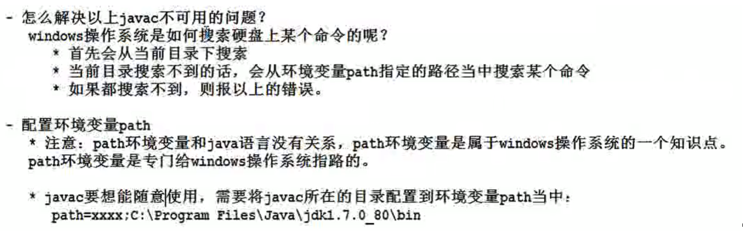
javac.exe 是编译器， java.exe是解释器

java源文件是运行在JVM中，不能直接与底层操作系统进行交互，javac.exe 将java源代码编译成字节码文件(.class)，java.exe 再将字节文件翻译成JVM所在平台的机器码，所以可以实现跨平台(一次编译到处执行)

JDK , JRE , JVM 之间的关系

JDK 包含 JRE ， JRE 包含 JVM

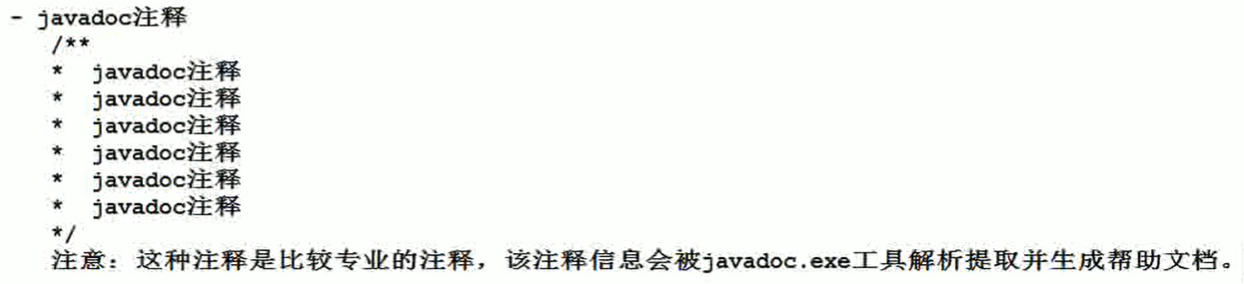
拓展:





Java 中的注释

// (单行注释) /\* \*/ (多行注释) 同C 语言一样



public class 和 class的区别

* + 1. 一个java源程序当中可以定义多个类(class)
    2. 一个java源程序可以不定义public 类
    3. 在编译时，java源程序中有几个类(class), 就会生成几个(.class)字节码文件
    4. 如果一个java源程序当中定义了一个public 类(有且只能定义一个public 类)，那源文件名必须与该public类名相同
    5. 每个class中都可以编写main方法，作为程序的入口

eg: 有一个源程序中有A,B,C 3个类，3个类中都编写了主方法( public static void main(String[] args) )， 经编译后会生成A.class，B.class，C.class，若要执行A.class 中的main方法，作为程序入口，就在命令行中执行 java A，同理执行B.class 中的main方法，作为程序入口，就在命令行中执行 java B。反过来要执行 java xxx，则xxx.class中就必须要有main方法，否则程序会报错(找不到程序的入口)

****二．JAVA语言基础****

标识符

1. 什么是标识符?

程序员有权力自己命名的单词就是标识符

1. 标识符的命名规则[必须符合规则，否则会报错]
2. 只能有数字，字母，下划线，美元符号($)组成
3. 不能由数字开头
4. 严格区分大小写
5. 关键字不能作为标识符
6. 理论上没有长度限制，但最好不要太长

3. 标识符的命名规范[只是种规范，不遵守也不会报错]

(1) 最好见名知意

(2) 遵守驼峰命名方式

eg:

SystemService, UserService

(3) 类名、接口名：首字母大写，后面每个单词首字母必须大写

(4) 变量名、方法名: 首字母小写，后面每个单词首字母必须大写

(5) 常量名：全部大写

注: 方法可以理解为C语言里的函数

变量

* 1. 什么是变量
     1. 变量实际上是内存中的一块存储空间，位置是确定的，存放在里面的数据不确定（简单的理解就是给内存中的一块存储空间命名）
     2. 变量包括三部分：数据类型、名称、字面值
     3. 变量是内存中存储数据的最基本单元
     4. 变量中存储的数据必须与数据类型一致

注: 字面值就是数据，并且存在各种不同类型的数据，简单的理解就是各种类型的常量

* 1. 变量的分类

根据变量声明的位置来分类:

1. 局部变量

在方法体当中声明的变量叫做局部变量

1. 成员变量

在方法体外[类体之内]声明的变量叫做成员变量

成员变量定义时没有赋初值，系统会默认赋初值

(类似于C语言中的全局变量)

注:

同一个作用域内变量名不能相同

成员变量又分静态变量(类变量)和实例变量

静态变量和实例变量的区分在于有没有用static关键字修饰

实例变量: 只能通过先创建对象，再使用 “ . ” 来引用

类变量: 可以通过先创建对象，再使用 “ . ” 来引用，也可以通过类名引用

原因:

类的字节码被加载到内存中时，如果该类没有创建对象，就不会为该类中的实例变量分配存储空间；而类变量不同，类的字节码被加载到内存中时，就会为该类中的类变量分配存储空间

数据类型

数据类型的作用

1. 不同的数据有不同的类型，不同的数据类型底层会分配不同大小的存储空间
2. 数据类型是指导程序在运行阶段应该分配多大的存储空间

注：java中数据类型分基本数据类型和引用数据类型

引用数据类型变量存放引用(地址号)

八种基本数据类型各自占用的空间大小

byte 1

short 2

int 4

long 8

float 4

double 8

boolean 1

char 2 (在c语言里占1个字节)

注：将级别低的变量赋值给级别高的变量时，系统会自动完成数据类型的转换，将级别高的变量赋值给级别低的变量时，需要强制类型转换，强制类型转换会损失精度，不建议用

eg:

(1) byte a = 40;

//理论上字面值40是被当做int类型数据来处理的，而这里的数据类型是byte，按道理要强制类型转换，但实际编译确不会报错,并且可以正确运行

(2) short b = 32767；同理

(这里的字面值32767也是被当做int类型来处理的)

注: 在java语言中当一个整数型数据没有超出byte，short或char类型取值范围时，该数据可以直接赋值给byte，short或char类型的变量，如果超出，就要强制类型转换后再赋值

(3) long z = 2147483648；

System.out.println(z)；

**编译出错**: 过大的整数: 2147483648

**原因**: 2147483648被当做4个字节的int类型来处理，但这个数据超过int类型的取值范围，long 同byte 和 short 不同的是虽然2147483648没有超出long类型的取值范围，但不能直接赋值

**解决方法:**

在2147483648后加个L，2147483648L是8个字节的long类型，z是long类型变量，**以下程序不涉及类型转换**

long z = 2147483648L；

System.out.println(z)；

(4) 同C语言一样，在java中浮点类型数据默认被当作double类型来处理；若想该数据当做float类型来处理，需要在该字面值后面加f or F

(5) 同C语言一样，float和double类型都不能精确的存储每一个浮点类型的字面值；如无线循环的数据(3.333333...)存储在有限的计算机内存中时，只能存储该数据的近似值。

(6) 在java中整型数据默认被当作int类型来处理

(7) 在java语言中boolean类型只有两个值：true，false，没有其它值，不像C语言中，除了true，false，还可以用1，0表示真和假

eg:

在C语言中

bool flag = 1；或 bool flag = true； 都可以将flag赋予真

而在java中

boolean flag = 1; 这条语句会报错，在java中1是一个整型的数据，并不能同c语言一样可以用1来表示真

(8) 八种基本数据类型除了布尔类型外，其他数据类型之间都可以相互转换

byte 、short、char混合运算时，各自先转换成int类型再进行运算

多种数据类型混合运算，先转换成容量最大的那种类型再进行运算

(9) 八种基本类型中char类型表示的是现实世界的文字（字符），文字（字符）和计算机二进制之间默认情况下是不存在任何转换关系，所以需要人为处理，将每一个文字(字符)都用一个对应的二进制表示，形成对照关系，即一个二进制代表一个文字(字符)，这种人为的转换过程称为字符编码。ACSII码和unicode码就是字符编码的产物，实际上就是一本计算机字典，再这本字典里可以查到某个文字(字符)用哪个二进制来表示

eg：

char a = ‘中’, b = ‘\u4e2d’；

System.out.println(a)；

System.out.println(b)；//输出结果都是 ‘ 中 ’

因为文字’中’的unicode码是4e2d，即文字’中’ 在unicode编码下 以4e2d表示；4e2d前面的\u必须加上，\u表示后面的4e2d是一个字符的编码，而不是字符串；若再将4e2d(十六进制形式)转化成对应的二进制形式，文字’中’就是以该二进制存储在计算机内存中的

注：java语言是遵循unicode编码的，所以标识符也可以为中文， 或其他国家文字

eg：

int 中文 = 100；

System.out.println(中文)；

//输出结果是100

字节

1 Byte = 8 bit (一个字节等于8个比特位，一个比特位表示一个二进制位 1 或 0)

1 KB = 1024 Byte

1 MB = 1024 KB

1GB = 1024 MB

1 TB = 1024 GB

1 TB = 1024 \* 1024 \* 1024 \* 1024 \* 8 = 8,796,093,022,208 bit

转义字符

转义字符出现在特殊字符之前，会将特殊字符转化成普通字符

eg: \’ (输出 ‘ )

\” (输出 “ )

\\ (输出 \ )

……

char c = ‘ ” ’；

System.out.print(c)；

//输出结果是 ” , 左右两边的单引号会自动匹配

关于原码，反码，补码

正数的原码，反码，补码相同

负数的补码为该负数绝对值的二进制形式，取反，再加一

如：求-5的补码

5 00000011 绝对值的二进制形式

11111100 取反

11111101 加一(这就是-5的补码)

byte a = 128;

System.out.println(a)；//输出结果是-128

推导过程: （已知补码求原码）

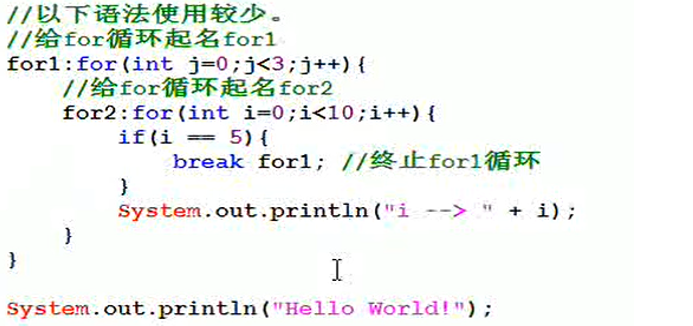
128 10000000 （补码）

01111111 （减一）

10000000 （取反）

因为补码最高位是1，所以为-128

for语句(拓展)



continue同样可使用上诉方法

switch 语句

格式: switch ()

{

case 1:

case 2:

case 3: java语句1；break;

………

default: java语句2；

}

//( )里面必须是int型或String类型，byte,short,char也行，因为byte,short,char类型实际上在( )中也是自动被转化成int类型来处理的

简单来说，( )里只能是整型，字符型(实际也是被转化成整型来处理)，字符串类型；case标号后的表达式也必须是上诉三种类型

****三. 方法****

在java中，方法被称为类的功能，变量被称为类的属性

方法分三类

构造方法: 供类创建对象时使用，给类创建的对象一个初始状态

无参/有参的构造方法

有参构造方法可以用来初始化成员变量

类方法(用static 关键字修饰的方法)

实例方法(供对象调用的方法，即只能通过对象来调用这类方法)

方法的优点

1. 提高了代码的复用性(某一段代码可以重复使用)
2. 减少了代码的冗余性，即避免了重复性操作(可以减少代码量)

方法的本质

方法就是一段代码块，这段代码块可以完成某个特定的功能，并且可以重复使用；它就是为了解决大量且类似的问题而设计的

注：

方法在类当中定义，一个类中可以定义多个方法，并且在类中，方法定义的位置，没有先后顺序，可以随意，但方法体中的代码，遵守自上而下的执行顺序。方法也就是C语言中的函数，可以嵌套调用，但不能嵌套定义。

方法定义的格式

修饰符列表 方法类型 方法名(形参列表){ }

注:

* + - 1. 主方法是程序的入口，是由JVM负责调用的
      2. 如果在一个类中定义了一个类方法(也称静态方法，用static关键字修饰的方法，没有用static关键字修饰的方法称实例方法)，在其它类中的方法中调用这个类方法的格式为: 类名.方法名();
      3. 如果两个类方法是在同一个类中定义的，其中一个类方法调用另一个类方法的格式为：类名.方法名(); 或 方法名(); 在这种情况下类名可以省略，因为类名省略后，默认在当前类中找要调用的方法；简单来说同一个类中的类方法之间相互调用，类名可以省略

注：类方法不仅可以通过了类名来调用，还可以通过创建一个该类的对象来调用；

eg:

public class A {

public static void m1() {

System.out.println(“我是m1”)；

B.m4()；//编译错误，无法从静态的方法调用非静态方法

}

void m3() {

System.out.println(“我是m3”)；

}

public static void main(String[] args)

{

m1()； //A.m1()；也行

B.m2()；//m2()；会编译报错

m3；//编译错误，main是类方法，m3是实例方法，在同一个类中，类方法只能调用类方法，不能调用实例方法

}

}

class B {

public static void m2() {

System.out.println(“我是m2”)；

}

void m4() {

System.out.println(“我是m4”)；

}

}

Java虚拟机中的三块主要内存空间

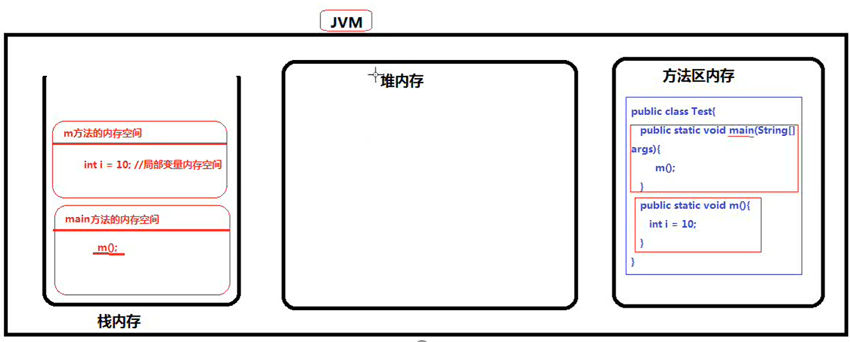
JVM内存划分上有三块主要的内存空间(除这三块外还有别的内存空间)

方法区内存 堆内存 栈内存

问题: 编写的方法代码片段(块)存放在哪里，方法的执行过程中吗，在哪里分配内存?

1. 方法代码片段(块)属于.class字节码的一部分，类加载器在硬盘中找到要加载的字节码文件，找到后把它加载到JVM中的方法区内存中，JVM的三块主要的内存空间中方法区内存最先拥有数据，用来存放了方法代码片段
2. 方法代码片段(块)虽然在方法区内存中只有一份，但是可以被重复调用，每调用一次这个方法，都需要为该方法分配存储空间(在栈内存中分配)

如图:



代码存放在方法区内存中，当执行到main方法时在栈内存中为main方法分配存储空间；执行到main方法中的 m(); 时，在栈内存为m方法分配存储空间(实际上是为方法中定义的局部变量分配存储空间)

方法重载机制(overload)

定义: 在同一个类中，功能相似的一些方法，可以起相同方法名；由于这些功能相似的方法都拥有相同的方法名，所有具体调用哪一个方法，由实参决定

优点: 方便编程人员对方法进行调用，使代码更加美观

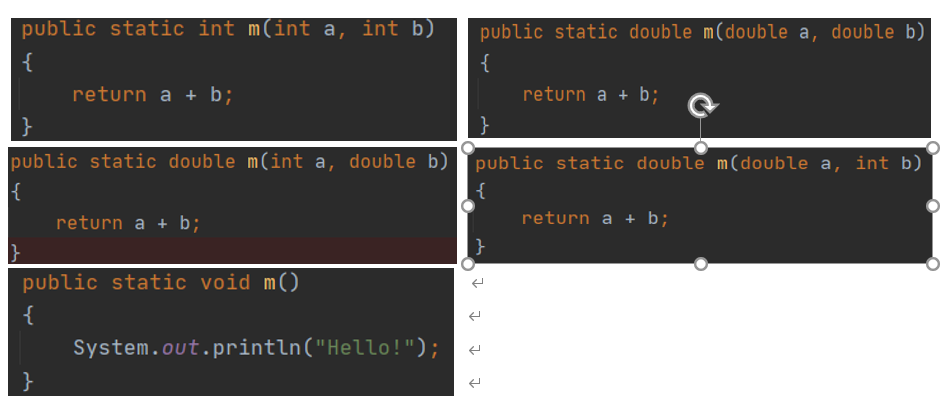
什么条件满足之后构成了方法重载?

在同一个类中

方法名相同

参数列表不同

参数个数不同，参数位置不同，参数类型不同

eg: 以下5个方法处于同一个类，可以构成方法重载

注: 方法重载与方法名及参数类型有关，和返回值类型，修饰符列表无关

方法重载也体现对态

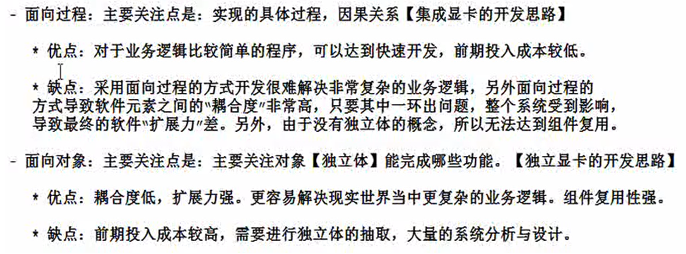
****四. 面向对象(Java语言的特点，核心)****

面向对象和面向过程的区别(盖浇饭和蛋炒饭)

面向过程是通过控制语句来营造整个事件的发生前后因果关系（耦合性高，扩展性差）

面向对象是通过每个个体之间组合去做将要发生的事（耦合性低，扩展性强）

C是纯面向过程的编程语言，C++是半面向对象的编程语言，Java是纯面 向对象的编程语言



面向对象的三大特性

封装

继承

多态

任何面向对象的编程语言都具备这三个特性

采用面向对象的方式开发一个软件，软件生命周期包含三个阶段：

面向对象的分析: OOA

面向对象的设计: OOD

面向对象的编程: OOP

类和对象的概念

什么是类?

- 类在现实世界中是不存在的，是一个模板，一个概念，是人类大脑思考抽象的结果

- 类代表了一类事物，用来描述这类事物的共同特征

- 在现实世界中，对象A和对象B具有相同的特征，进行抽象总结出一个模板，这个模板称为类

拓展：

类体内没有自上而下的语句执行顺序，方法体的语句自上而下顺序执行

除了声明变量，类体内不能直接编写java语句

eg:

public class Test

{

public static void main(String[] args)

{

int i = 10; (1)

//下面这行语句输出的结果是10

//若将(1)处的语句注释掉，下面这行语句输出结果为100

//由此可以看出java遵循就近原则

System.out.println(i);

}

int i = 100; //类体内可以声明变量

//System.out.println(i); 此行有误，类体内不能直接编写java语句

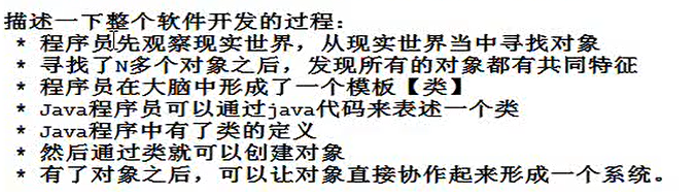
}

什么是对象?

对象是实际存在的个体，现实当中实际存在。

比如说: 人(一个不存在的，抽象化的概念)就相当于一个类，我们就是一个个对象(一个个存在的，独立的个体)；因为我们具有相同的特征(一个头，两只手，两天腿…)，所以都可以归纳为人；总而言之，类是对象的集合，对象是类的实例化，对象也称实例

eg:



类由属性(成员变量)和方法组成

类的属性描述的是对象的状态信息

类的方法描述的是对象的动作信息

创建对象

先声明一个Student类

class Student{

String name, sno;

char sex;

int score;

……

}

Public class Test{

Public static void main(String[] args) {

new Student(); //在JVM中的堆内存中开辟一个存储空间，这个存储空间包括了Student类中的所有成员变量的存储空间

简单的来说，这条语句就是为Student类中的成员变量分配一块存储空间；但这里缺少一个对象指向这个存储空间。

Student stu = new Student();

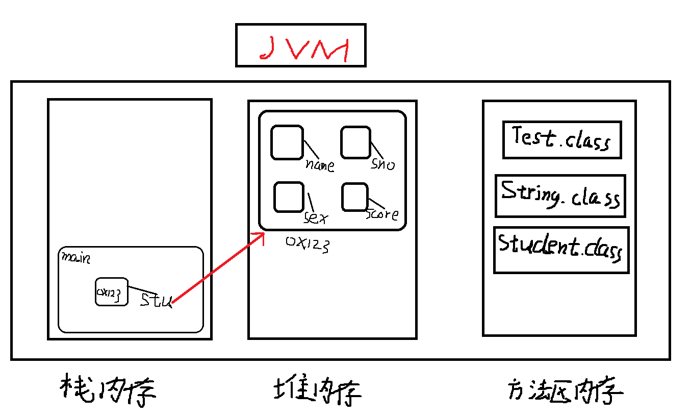
//为Student类中的成员变量分配了一块存储空间，再调用Student();

构造造方法将该存储空间的首地址赋予前面定义的引用型变量stu(对象stu)

}

}

如图:

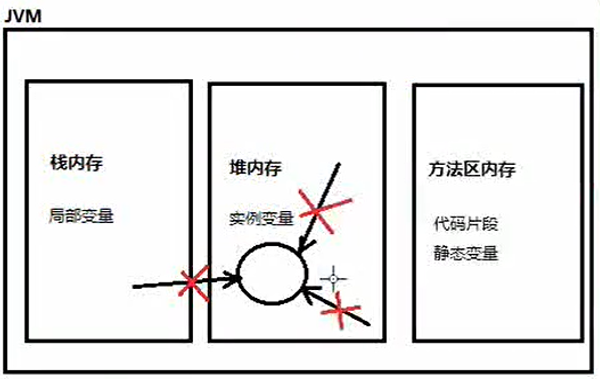


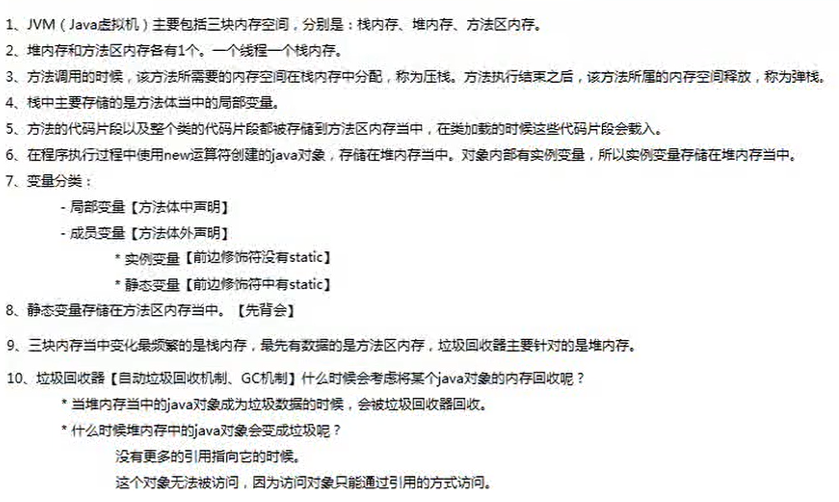
注:

对象stu是一个类类型变量，即引用型变量

stu中存放的是“引用”（实际就是地址号），将new Student(); 开辟的存储空间的首地址赋予对象stu, stu就能操作这块存储空间；类似于C语言中的结构体变量

拓展:

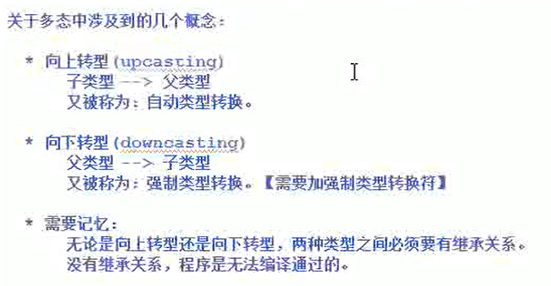




拓展：

* + - 1. 在一个类中，如果没有定义任何构造方法，系统默认给该类提供一个无参的构造方法，供类创建对象使用，这个构造方法又被称为缺省构造器；若定义了构造方法，则系统不再提供
      2. 一个类中可以定义多个构造方法，构造方法也支持重载机制
      3. 类中的构造方法实际也有返回值和返回值类型(数据类型)，返回值是一块内存单元的首地址 (这个地址会被赋予前面定义的对象)，返回值类型就是类本身(类名)；因为“返回值”的类型就是类本身，所以java中规定在定义构造方法时，无需声明构造方法的返回值类型，在构造方法中也不需要再编写return语句返回一个值(地址)，系统会默认返回。

多态



向上转型只要编译通过，运行一定不会出问题

例如：

Animal a = new Cat(); //Cat 是 Animal 的子类

向下转型需要注意一个常见的错误，因为向下转型要用到强制类型转换，可能存在隐患(编译通过了，但是运行错了)

例如:

Animal a = new Bird()； //Bird也是Animal的子类

Cat c = (Cat)a；

编译器检测到a是Animal类型，c是Cat类型，Cat又是Animal的子

可以向下转型，编译阶段通过，但运行阶段会报错

可以采用以下方法解决这个隐患

Animal a = new Bird()； //Bird也是Animal的子类

if (a instanceof Cat) //当a引用的对象是Cat类型时(当a是Cat类的实例时)，在执行下面的语句进行向下转型

Cat c = (Cat)a；

注:

父类对象变量引用子类对象可以体现多态(上转型对象)

接口回调也可以体现多态

接口回调: 将实现某个接口的类的实例的引用赋予用该接口声明的变量(接口变量)，就可以通过接口变量调用接口中被实现的抽象方法。(实际上还是通知相应的对象去调用重写的抽象方法)

方法重载也体现多态，即功能的多态

instanceof 运算符(双目运算符)

格式: 对象变量 instanceof 类(引用型数据类型)

该运算符的运行结果是布尔类型，即要么运行结果为true，要么为false

例如：

a instanceof Animal；

如果运行结果为true

表示a引用的对象是Animal类型

如果运行结果为true

表示a引用的对象不是Animal类型

final 关键字

- final 修饰的变量就变成了常量

定义时必须人为初始化，不能采用系统默认值，并且值不能再更改

- final 修饰的实例变量是不可变的，这种变量一般和static联合使用，被称为常量

常量定义语法格式:

public static final 数据类型 常量名 = 值；//java规范中常量名要大写

- final 修饰的引用型变量，如果该变量已引用了一个对象，就不能再引用另一个对象，但对已引用对象中的成员变量的值，还是可以修改的；本质上还是final修饰的引用型变量的值不能再更改

例如:

final User u1 = new User();

u1 = new User() //试图再引用另一个对象，错误

u1 = null; //错误

u1.xxx = xxx; //正确

- final修饰的类不能被继承，修饰的方法不能被覆盖

访问权限修饰符

作用: 访问权限修饰符用来控制元素的访问范围

访问权限修饰符包括

public 表示公开的，在任何位置都可以访问

protected 同包中访问，不同包中可以通过子类继承父类来访问

缺省 同包中访问

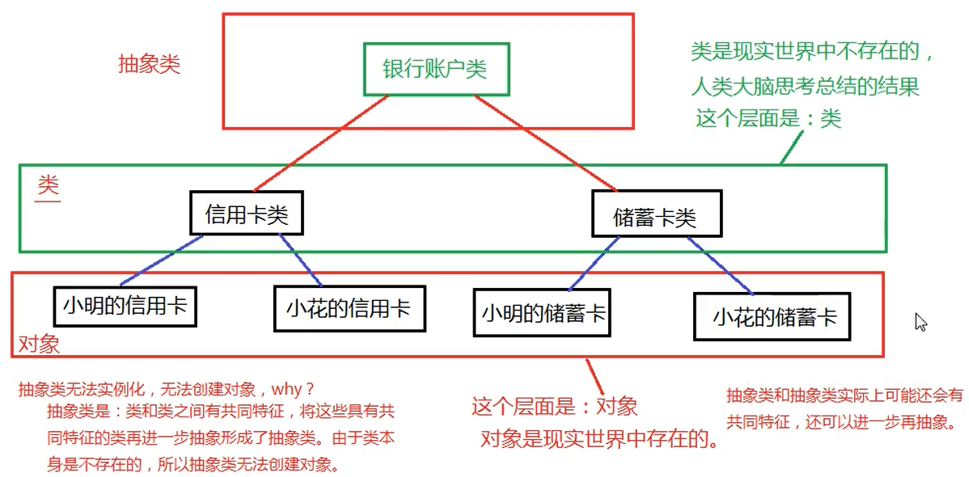
Private 表示私有的，只能在本类中访问

修饰符的范围

Private < 缺省 < protected < public

抽象类

图解:



什么是抽象类?

类和类之间具有共同特征，讲这些共同特征提取出来，形成的类就是抽象类

类本身是虚拟的，所以无法用抽象类创建对象(抽象类无法实例化)

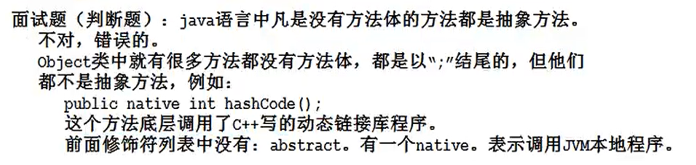
抽象类用关键字abstract修饰

注意:

* + 1. 抽象类不能用fianl修饰，会报错；因为定义抽象类(用abstract修饰)就是想让其它类继承，final修饰的类不能被继承，此处存在矛盾冲突。在java中规定abstract和final不能联合使用，联合使用会提示错误信息:非法的修饰符组合
    2. 抽象类中可以没有抽象方法，但抽象方法要么在抽象类中，要么在接口中。
    3. 抽象类中的抽象方法，只被声明(没有类体)，没有被实现；
    4. 如果一个非抽象类继承一个抽象类，必须实现/重写/覆盖抽象类中的抽象方法(在子类中重写抽象类中的抽象方法)；这是java规定的，如果没重写，就会报错，并且在子类中重写时，必须去掉abstract修饰符
    5. 抽象类可以继承抽象类，如果一个抽象类继承另一个抽象类，可以继承父类的抽象方法(只继承，不重写)，也可以重写父类的抽象方法，还可以定义自己的抽象方法。

问: 为什么非抽象类继承抽象类，在非抽象类中必须实现(重写)抽象类中的抽象方法?

答: 前面提到过除了抽象类的其它类都不能有抽象方法，若一个非抽象类继承一个抽象类，就顺理成章的继承了该抽象类中的抽象方法，然而非抽象类中不能有抽象方法，此处产生矛盾；所以在非抽象类中必须通过方法重写将这个继承过来的抽象方法覆盖(隐藏)



接口

- 接口也是一种引用型数据类型，编译后也是同类一样，会生成一个class字节码文件

- 接口是完全抽象的(因为接口只能声明抽象方法，和常量，接口中的抽象方法默认使用public abstract修饰的)，抽象类是半抽象的(因为抽象类中除了可以声明抽象方法，还可以声明非抽象方法)

- 接口怎么定义

[修饰符列表] interface 接口名{ }

- 接口支持多继承，类不可以，即一个接口可以继承多个接口

- 因为java中类只能单继承，不支持多继承，为解决不支持多继承的短板，java提供了接口

- 接口中只包含常量和抽象方法，常量默认用public static fianl修饰(可省略)，抽象方法默认用public abstract 修饰(可省略)

- 当一个非抽象的类实现接口，必须将该接口中的抽象方法全部实现

接口回调体现多态

接口回调: 将实现某个接口的类的实例的引用赋予用该接口声明的变量(接口变量)，就可以通过接口变量调用接口中被实现的抽象方法。(实际上还是通知相应的对象去调用重写的抽象方法)