

概论

2020年2月24日 8:06

- **重点**

第2章：语言基础（基本数据类型，运算符，表达式等）

第3章：基本控制语句（选择，循环等）

第4章：方法（含义，创建，调用方式，参数传递等）

第5章：数组和字符串（含义，创建，灵活应用等）

第6章：类和对象（面向对象的编程技术：类和对象的定义创建，应用等）

第7章：类的继承和多态机制（类的继承，方法的重载和覆盖等）

第8章：接口和包（含义，创建和应用）

- **考试方法**

- 理论考试

- 上机考试（必须通过实验考试后才能参加理论考试）

- 实验和理论都是系统自动批改

- **总评成绩构成**

- 总评=理论50+实验25+平时25（主要参考平时的上机作业上交情况）

- 理论成绩必须大于等于55，若理论成绩不满55，总评成绩按理论成绩登记

- **上交作业的网站**

目前：学在浙大（course.zju.edu.cn），课件，编程环境软件（jdk,eclipse）都可以在上面下载，第1，2周作业也在上面下载后按要求完成并上传

一般从第3周起用其它的自动批改的实验平台（与实验考试系统界面相同），网址到时公布

李峰email:lif@zju.edu.cn

第一章

2020年8月28日

15:38

• java简介

○ 程序设计语言经历

▪ 机器语言->汇编语言->高级语言

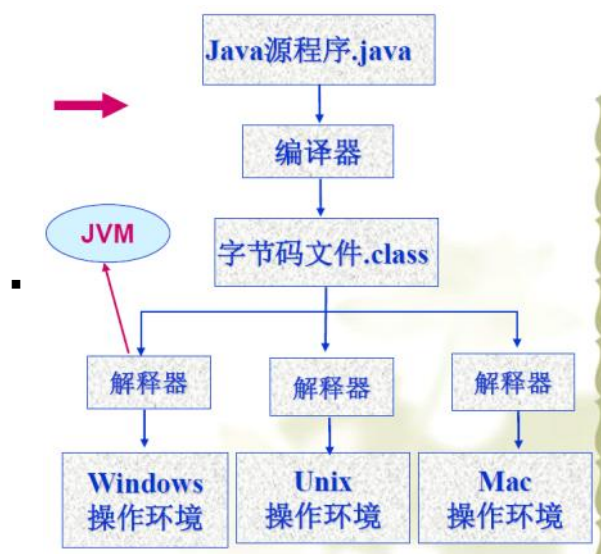
- 机器语言：由一条条指令（二进制代码）组成
- 汇编语言：用记忆符号来代替机器语言的二进制代码
- 高级语言：
 - ◆ 面向过程的高级语言：BASIC语言，C
 - ◆ 面向对象的高级语言：Visual Basic，C++，Java
 - ◇ 在程序设计中，总是先定义类，在创建属性类的对象

○ 高级语言

▪ 结构化程序设计->面向对象程序设计

○ Java的特点

- 去除了C++的头文件、指针变量、结构、运算符重载、多重继承
- Java的字节码需要经过Java虚拟机解释成机器码才能执行



- JDK是Sun公司推出的Java开发工具包，包括**Java类库**、**Java编译器**、**Java解释器**、**Java运行环境**和**Java命令行工具**，JDK主要有3种版本

- J2SE：Java标准版/标准平台，可开发Java桌面应用程序和低端服务器应用程序，也可开发Java Applet程序
- J2EE：企业版/企业平台，包含J2SE平台，增加了附加类库，支持目录管理，交易管理等功能
- J2ME：微型版或小型平台，用于嵌入式的消费产品中，如移动电话、掌上电脑等

○ 过程

- Java源程序 .java
- 编译器 javac.exe

- 字节码文件.class 由java.exe解释
- JVM
- Java没有指针类型
- Java仅允许单继承
- Java语言编译为字节码，通过JVM编译为机器码
- Java
 - Application
 - Java applet
- 源文件的命名规则：如果源文件中有多类，那么**只有一个类是public类**，源文件的名称必须与这个类的名字完全相同
- 一个java应用程序必须有一个类含有main方法，称为应用程序的主类，且main方法必须被说明为public static void，程序从main方法开始执行
- Java中 **一个程序必须有至少一个输出**
- **System**是Java类库中的一个**类**，**out**是System类中的一个**对象**，**println或print**是out对象的一个**方法**
- 是

第二章 Java基础

2020年3月1日 21:59

标识符和关键字

Java的源程序文件

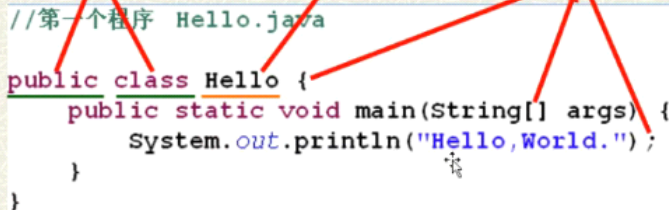
■ 文本文件

- 以 `.java` 为扩展名

Java的语言成分

- 关键字（保留字）、标识符、运算符和分隔符

```
//第一个程序 Hello.java  
  
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello,World.");  
    }  
}
```



2.1 标识符和关键字

- 关键字：事先定义好的，比如 `public class`
 - Java已经使用的事先定义的有特殊含义的一些字符，用户不能使用
 - 关键字一律用**小写字母**表示
- 标识符：取得名字，比如 `Example`
 - Java中用来标记变量、常量、方法、类、对象等元素的符号，一些字符的组合
 - 取名规则：由字母、数字、`_`和`$`组成，长度不限，但实际命名不宜过长
 - 标识符的第一个字符必须为**字母**，`_`，`$`，标识符**区分大小写**，不包含运算符，不可以用数字开头
- 分隔符：`{}`，`()`，`;`，`]`，`[`，`,`等

◆ “见名知义”，规范大小写的使用方式

1. 大多数为小写字母开头

- ◆ 变量名、对象名、方法名、包名
- ◆ 标识符由多个单词构成，则首字母小写，其后单词的首字母大写，其余字母小写。如 `getAge`。

2. 类名首字母大写

3. 常量名全部字母大写

◆ 规则：非强制性

2.2 数据类型与常量、变量



Java整型

- 整型分为四种，差别在于内存空间和数据取值范围

| 数据类型（类型关键字） | 所占字节 | 取值范围 |
|-------------|------|--|
| long（长整型） | 8 | -9223372036854775808~9223372036854775807 |
| int（整型） | 4 | -2147483648~2147483647 |
| short（短整型） | 2 | -32768~32767 |
| byte（位） | 1 | -128~127 |

- Long-所占字节8-64位 表示为：345L/345l
- Int-4-32
- Short-2-16
- Byte-1-8

三种进制的整数表示

- 十进制：首位不可以为0
- 八进制：0为前缀
- 十六进制：0x/0X为前缀
- 定义之后输出的还是十进制

浮点数

- 标准计数法：12.31
- 科学记数法：2.5×10⁴表示为2.5E4
 - 由尾数，E/e，阶码组成
- 两种浮点数类型
 - Float 4个字节**
 - Double 8个字节**
 - 浮点数常量默认处理类型为double，一个浮点数的缺省类型为double
 - 若要表示浮点数为float型，可加后缀F/f，如34.5f

字符类型

- Char
 - Unicode字符，**1个字符站16位**，前256个字符与ASCII相同
 - 表示类型
 - 单引号括起来，如'C'
 - 用16进制Unicode表示，前缀码为'\u'，如'\u0008'

◇ 用8进制Unicode码表示

◆ 如果想要输出'\', 需要先转义下'\\'

▪ 字符串

□ 用英文状态下双引号括起来

□ 又如: "ab' 12\" 字符\\
结果: ab'12" 字符\

▪ 布尔类型

□ 逻辑运算

◆ 运算结果: true/false 1 布尔类型 boolean

◆ 表示逻辑运算, 运算结果叫做布尔值

◆ Java中的true关键字不等于1, false关键字也不等于0

◆ 布尔值

◇ true

◇ false

◇ 布尔值占1个字节

▪ 常量

□ 直接常量, 指在程序运行过程中其值始终保持不变的量。类似于数学中的“常数”

□ 另一种常量: 符号常量

◆ final【修饰符】类型标识符 常量名 = (直接) 常量

1. 增加了程序的可读性

◆ 从常量名可知常量的含义。

◆ 2. 增强了程序的可维护性

◆ 只要在声明处修改常量的值, 就自动修改了程序中所有地方所使用的常量值

▪ 变量

□ [修饰符] 类型标识符 变量名 [=常量]

□ 声明一个变量, 系统必须为变量分配内存单元。分配的内存单元大小由类型标识符决定

▪ 总结

| 关键字 | 数据类型 | 所占字节 | 表示范围 |
|---------|------|------|-------------------------|
| long | 长整型 | 8 | $2^{31}-1 \sim -2^{31}$ |
| int | 整型 | 4 | -2147483648~ 2147483647 |
| short | 短整型 | 2 | -32768~32767 |
| byte | 位 | 1 | -128~127 |
| char | 字符 | 2 | 0~256 |
| boolean | 布尔 | 1 | true 或 false |
| float | 单精度 | 4 | -3.4E38~ 3.4E38 |
| double | 双精度 | 8 | -1.7E308~ 1.7E308 |

• 运算符和表达式

○ 算术运算符

10.3%3 //结果是1.3
 9%4 //结果是1
 -9%4 //结果是-1
 -9%-4 //结果是-1

实数求余，结果是实数，而当“%”用于2个整型操作数时，结果是整数。取余结果的符号和被除数的符号相同

- 单目运算符包括-（负号），++（自增），--（自减）
- ++、--
 - 仅用于整型变量，表示加1或减1
 - ++、--可在变量左边，也可在变量右边（如出现在变量左边，则表示先修改变量的值，在使用修改后的变量值，若出现在变量右边，则表示先使用变量的值，在再修改变量的值）
 - **单目运算符的优先级高于双目运算符**
- 关系运算符
 - 字符类型可以比较大小
 - **字符串不可以用>、<、>=、<=**
 - **==和!=可以用于字符和字符串**
 - **关系运算的运算结果是true/false**
 - **>、<、>=、<=的优先级高于==和!=**
 - ASCII: 数字 大写字母 小写字母
- 逻辑运算符
 - &&和||
 - 在组合条件中从左到右依次判断条件是否满足，一旦能确定结果就终止运算，不再进行右边剩余的操作
- 赋值运算符
 - 运算次序从右向左
- 位运算符

| a | b | ~a | a&b | a b | a^b |
|---|---|----|-----|-----|-----|
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |

位运算

- 操作数：整型
- 运算规则：按二进制的位
- 运算结果：整型值

位运算符有

- ~（位反）
- &（位与）、|（位或）
- ^（位异或）
- <<（左移）、>>（右移）
- >>>（无符号右移）

- >> (最高位不动，次高位用最高位的值来填)高位溢出，低位补0
- << 带符号的，低位溢出，高位用符号位的值补
- >>>, 低位溢出，高位补0
- 条件运算符——三目运算符
 - 格式: <111>?<222>:<333>
- 括号运算符
 - 先进性括号内的运算，再进行括号外的运算
- 运算符优先级
 - 算术运算，关系运算，逻辑运算，条件运算，赋值运算

| 运算符 | 描述 | 优先级 | 结合性 |
|-------------|---------|-----|------|
| . [] () | 域，数组，括号 | 1 | 从左至右 |
| ++ -- - ! ~ | 一元运算符 | 2 | 从右至左 |
| * / % | 乘，除，取余 | 3 | 从左至右 |
| + - | 加，减 | 4 | 从左至右 |
| << >> >>> | 位运算 | 5 | 从左至右 |
| < <= > >= | 关系运算 | 6 | 从左至右 |
| == != | 关系运算 | 7 | 从左至右 |
| & | 按位与 | 8 | 从左至右 |
| ^ | 按位异或 | 9 | 从左至右 |

| 运算符 | 描述 | 优先级 | 结合性 |
|---|---------|-----|------|
| | 按位或 | 10 | 从左至右 |
| && | 逻辑与 | 11 | 从左至右 |
| | 逻辑或 | 12 | 从左至右 |
| ? : | 条件运算 | 13 | 从右至左 |
| = *= /= %= += -= <<= >>= >>>= &= ^= = | 赋值和复合运算 | 14 | 从右至左 |

- 数据类型转换



- 从低级别往高级别转
- 如果是高级别的值赋给低级别变量，需强制类型转换
(int) 24.65 -> 24
- 双目运算中，若两数的类型不一致，系统将级别低的值转为级别高的再运算
- 字母大小写转换
 - 大+32=小

- char可以自动转为int, 值为字符的ASCII码值
- int转为char需要强制类型转换

```
public class exp00 {
    public static void main(String[] args)
    {
        char ch1='a',ch2;
        ch2=(char)(ch1-32);
        System.out.print(ch2);}
}
```

输出: A

例: 'A'+12-10.65

结果: 66.35

- 字符串运算符 +
 - 字符串+字符串
 - 字符串+非字符串
 - ◆ 自动将非字符串转换为字符串, 再合并
 - ◆ 用于输出语句

• Math类

- abs: int/float/...
- pow (double, double) 返回double
- sqrt返回double

round(x) //返回x 四舍五入取整的值, 结果为long型

random() //返回[0, 1)上的随机数

- round:
 - 对于正数: 四舍五入取整
 - 算法: 将操作数加0.5, 取最接近的最小整数
- Random
 - 返回double
 - 随机数范围 [0.0,1.0)
 - 返回a到b之间的随机整数(int类型 《两头包括a和b》)
 - (int)(a+Math.random()*(b-a+1))
- 对x四舍五入保留两位小数
 - (int)(x*100+0.5)/100.0
 - Math.round(x*100)/100

• 数据的输入和输出

- Scanner in = new Scanner (System.in)
- nextInt
- nextFloat
- nextDouble
- next 字符串, 输入的是一个单词, 空格结束
- nextLine 字符串 输入的是一个语句, 回车结束
- Scanner需要import java.util.*或者import java.util.Scanner
- System.in属于java标准输入流

第三章 Java流程控制

2020年3月16日 9:21

- 顺序结构
- 选择结构，也叫分支结构
 - else总与离他最近的if配对
 - **switch里面的表达式可以是byte、char、short、int，不能是浮点数和long**
- 循环结构
 - 计数控制
 - 事态控制
-

第四章 方法-Method

2020年8月30日 10:57

- **方法声明**

- **【修饰符】** 类型标识符 方法名 参数表
- 方法声明不能嵌套：不能在方法中再声明其它的方法

第五章 数组

2020年4月13日 8:32

- 一维数组
 - 声明：方括号可以在变量名后面，也可以在类型名后面
 - 先声明再初始化的话，初始化作用
 - 分配存储空间，确定元素个数
 - 用new初始化数组
 - **各元素的存储空间是连续的**
 - 获取元素个数：数组.length
 - 方法一
 - `int x[]`
 - `x = new int [10]`
 - 方法二
 - `int x[] = new int [10]`
 - 方法三
 - `int x[] = {}`
 - 所赋初值的个数决定数组元素的数目
- 二维数组
 - 方法一
 - `int x[][]`
 - `x = new int [10][10]`
 - 方法二
 - `int x[][] = new int [10][10]`
 - 方法三
 - `int x[][] = {{},{},{}}`
 - 数组名.length **数组的行数**
 - 数组名[行标].length **数组某行的列数**
 - **所以可以定义不规则的二维数组**，如：
 - `int b[][];`
 - `b=new int[2][];`
 - ◆ 必须先定义行才能定义列
 - `b[0]=new int[5];`
 - `b[1]=new int[3];`
- 数组的基本操作
 - 数组的引用
 - 数组的复制
 - `a=b`
 - `a`和`b`必须位数相同才能直接赋值
- 数组参数
 - 在**形式参数表中**，**数组名后的括号不能省略**，括号个数和数组的维数相等。不需给出数组元素的个数。
 - 在**实际参数表中**，**数组名后不需括号**。
 - **数组名做实际参数时，传递的是地址**，而不是值 → 即形式参数和实际参数具有相同的存储单元。形式参数值的改变将导致实际参数的值随之改变。但是**数组元素做参数时传递的是值**
- 字符串
 - **String 字符串变量 = new String();**
 - String类用很多成员方法来对字符串进行操作。如：字符串访问、比较和转换等。如`s= "abcdef"`；
 - **length()**：返回字符串的长度。则`s.length()`结果是6

- **charAt(int index):** 返回字符串中第index个字符。则s.charAt(0)结果是 'a'
- **indexOf(int ch):**
 - 返回字符串中字符ch第1次出现的位置。indexOf('a') 或者 indexOf(97), 找不到返回值-1
- **indexOf(String str, int index):**
 - 返回值是在该字符串中, 从第index个位置开始, 子字符串str第1次出现的位置, 找不到返回值-1
- **substring (int index1, int index2):**
 - 返回值是在该字符串中, 从第index1个位置开始, 到第index2-1个位置结束的子字符串。
 - 若省略index2, 则从index1位置开始, 到结束的子字符串
- **equals(String str)** 区分大小写, 返回类型true/false
- **equalsIgnoreCase(String str)**, 不区分大小写
- **==**也能进行字符串是否相等的比较, 但是==比较的是地址指针, 而equals比较的是实际内容, 所以要比较两个字符串是否相等时, 应该用equals方法

```
String s1=new String("Hello");
String s2=new String("Hello");
□ System.out.println(s1==s2);
System.out.println(s1.equals(s2));
```

//输出false

//输出true

- **int compareTo(String str)** 比谁大谁小。该字符串比str大则返回正值, 小则返回负值, 相等则返回0.返回值为两个字符串中第一对不相等字符的ASCII码的差值

将int, long, float, double, boolean等基本数据类型数据转换为String类型的方法是:

String.valueOf(基本类型数据)

将字符串型数据转换为其他基本类型的方法及实例

| 方法 | 返回值类型 | 返回值 |
|--------------------------------|---------|----------|
| Boolean.getBoolean("false")) | boolean | false |
| Integer.parseInt("123") | int | 123 |
| Long.parseLong("375") | long | 375 |
| Float.parseFloat("345.23") | float | 345.23 |
| Double.parseDouble("67892.34") | double | 67892.34 |

第六章 类和对象

2020年5月28日 17:03

- 类和对象

- 对象是类的实例
- 类是对象的抽象描述
- **面向对象的程序设计认为：程序由类（对象）组成**
- 只关心对象能够完成的操作，不关心实现功能的过程
- 类分为两种
 - java类库
 - 用户编写的类
- 类的声明
 - 【修饰符】 class 类名【extends 父类名】【implement 接口名列表】
 - 一般为：【修饰符】 class 类名

[修饰符] 类名 对象名= new 类名([实参列表]);

或声明和创建分开：

- [修饰符] 类名 对象名;
对象名 = new 类名([实参列表]);

- **注意：类属于引用数据类型，因此，在声明对象时，系统并没有为对象分配空间，用户需要应用new完成分配空间的任務**

- 构造方法和对象的初始化
 - 对象的初始化-构造方法
 - 构造方法名与类名相同
 - 构造方法没有返回值
 - 构造方法不能显式直接调用
 - 当使用new运算符实例化一个对象时，Java做两件事：
 - 系统为对象创建内存区域
 - 自动调用构造方法初始化成员变量
 - 一个类中可以定义多个构造方法
 - 缺省构造方法的使用
 - 如果没有定义构造方法，Java将调用无参的构造方法，使用默认值初始化对象的成员变量。那么数字变量则为0，char为 '\0'，bool为 false，string为null
 - **在Java中变量必须初始化，即赋初值才能使用变量**
- **编程与运行**

在Java中，每个类是一个独立的程序段，编译后产生一个字节码文件(扩展名为class)。

- 本例中，有两个类，因此有两个.class文件，分别建立这两个文件构成一个整体程序

能够被运行的程序是内有main()方法的类（主类）

- 析构方法

- public void finalize () {} 主动释放对象，在销毁之前需要做的事情
- 内存空间有限，不能存放无限多的对象

- 类的封装

- 封装性是面向对象的核心特征之一

- 类是一个不可分割的独立单位

- 类中既要提供与外部联系的接口，同时又要隐藏类的实现细节

- 类的访问权限

- 声明一个类：可使用的权限修饰符：public对所有包的类可见、缺省（只能对本包的类可见）
- 不能使用：private protected
- 包是一组相关类和接口的集合，存储观念看，包相当于一个文件夹
- 来康康：

- public（公有）

- 成员变量和成员方法可以Java在所有类中访问

- protected（保护）

- 成员变量和成员方法可在声明他们的类中访问

- 可以在该类的子类中访问

- 也可以在与该类位于同一包的类中访问——不能被其它包的非子类中访问

- 缺省

- 不使用权限修饰符

- 成员变量和方法可在自己的类及该类位于同一包的类中访问

- 不能在位于其它包的类中访问

-

- private（私有）

- 成员变量和成员方法只能在声明他们的类中访问

- 不能在其他类（包括其子类）中访问

- Private指定的访问权限范围最小

| 访问控制 | 本类 | 同一包中的类 | 其他包中子类 | 其他包中的类(子类除外) |
|-----------|----|--------|--------|--------------|
| public | √ | √ | √ | √ |
| private | √ | × | × | × |
| protected | √ | √ | √ | × |
| 缺省 | √ | √ | × | × |

- 声明一个类

- 可使用：public/缺省，不能用protected，private
- 一个Java源程序文件中可以包含多个类，一个java文件中只能有一个类使用public修饰符，该类的名字与源程序文件的名字相同，一般而言，main方法包含在Java源程序文件的public类中
- 有多个类，必须有一个类含有main方法，称为应用程序的主类，且main方法必须被说明为public static void。程序从main方法开始执行

- 类成员（静态）-变量和方法

- Java的类中包含两种成员

- 实例成员和类成员
- 实例成员
 - 实例成员属于对象
- 类成员
 - 包括类成员变量和类成员方法，申明时用static修饰
 - 没有用static修饰的变量为实例成员变量，用static修饰的变量为类成员变量
 - 可以通过类名/对象来引用
 - 没有创建对象也可以引用类成员
 - 类成员变量在该类被加载到内存时，就被分配存储空间，类的所有对象共享这个类成员变量
- static是静态，静态的含义是定义了类，则其静态成员也存在了

使用static修饰的变量为类成员变量
没有使用static修饰的变量为实例成员变量

- 类的所有对象共享类成员变量
- 类成员变量可以通过类名或者对象名访问，实例成员变量名只能通过对象引用

- 类成员方法

- 没有用static修饰的变量为实例成员方法，用static修饰的变量为类成员方法

类成员方法中使用——受限制

- ◆ 本方法中声明的局部变量
- ◆ 可以访问类成员变量，不能访问实例成员变量

实例成员方法中使用--不受限制

- 本方法中声明的局部变量
- 可以访问类成员变量
- 也可以访问实例成员变量

◆ 类成员方法中只能调用类成员方法，不能调用实例成员方法

◆ 实例成员方法既可以调用类成员方法，也可以调用实例成员方法。

实例方法只能通过对象访问

类成员方法既可以通过类名访问，也可以通过对象名访问

类成员方法中只能直接调用类成员方法，不能直接调用实例成员方法,必须通过创建对象访问实例成员方法

第七章 类的继承和多态

2020年6月11日 15:23

• 类的继承

- 定义了student类，此时需要处理大学生，增加部分信息
 - 此时定义子类collegestudent
 - Class collegestudent extends Student
- 关键字 extends
 - 已经存在的类（student）成为超类super class/基类 base class/父类 parent class
 - 新的类称为派生类derived class/子类 subclass/孩子类 child class
- Java类是一个层次结构
 - 一个超类可以派生出多个子类
 - 每个子类作为超类又可以派生出多个子类
 - 最顶端有一个叫做Object的超类，是所有类的祖先，所有类都直接或间接地继承自Object类
 - 如果在定义类的时候没有指出其超类，则默认继承Object
- 一旦定义了一个类，就自动继承了Object类的许多成员和成员方法，比如 hashCode()和equals()检测是否与另外一个对象相等
- 子类继承父类的
 - 属性和方法
 - 可以修改父类的属性或重载父类的方法
 - 在父类的基础上添加新的属性和方法
- Java语言只支持单继承
 - 每个子类只允许有一个父类
 - 如果要定义多继承，可以使用接口
- 子类不能继承父类的构造方法

Rules:

1. 子类继承父类的成员变量，包括：
 - ◆ 实例成员变量
 - ◆ 类成员变量
2. 子类继承父类成员方法
 - ◆ 实例成员方法
 - ◆ 类成员方法
 - ◆ 但不包括构造方法
 - 父类的构造方法 → 创建父类对象
 - 子类需定义自己的构造方法 → 创建子类的对象
3. 子类可以重新定义父类成员
 - ◆ 如方法的覆盖

本质上说，子类和父类都是“类”，因此存在访问权限问题

子类继承了父类的成员变量和成员方法

- 但并非对父类所有成员变量和成员方法都有访问权限，换句话说：
- 子类声明的方法中并不能访问父类所有成员变量或成员方法！

子类访问父类成员的权限如下

- ◆ 对父类的private成员没有访问权限。
- ◆ 对父类的public和protected成员具有访问权限。
- ◆ 对父类的缺省权限成员访问权限分2种情况
 1. 对同一包中父类的缺省权限成员具有访问权限
 2. 对其它包中父类的缺省权限成员没有访问权限

○ super

- 在子类中可以声明与父类中同名的成员变量和成员方法，super用来区分父类和子类存在同名的成员变量/方法时
- 在子类的构造方法调用父类构造方法

□ super (参数表) 必须是子类的构造方法体的第一条

○ this的典型应用

- 当类中的成员变量和成员方法中的形式参数，局部变量存在同名的情况
- 调用当前对象的其它不同参数的构造方法

```
public Point(int x, int y) {           //带参数的构造方法
    this.x=x;
    this.y=y;
    System.out.print( "["+this.x+","+this.y+"]" );
}
public Point() {                       //不带参数的构造方法
    this(5,5);                         //调用带参数的构造方法
}
```

• 类的多态性 重载和覆盖

○ 多态性

- 多态性指同一名字的方法可以有多种实现，即不同的方法体
- 在面向对象的程序中多态表现为
 - 可以利用**重载**在同一类中定义多个同名的不同方法实现多态
 - 也可以通过子类对父类方法的**覆盖**实现多态
- 重载
 - 在同一类中，多个方法具有相同的方法名，但采用不同的形式参数列表
 - **形参个数/类型/顺序，返回类型和参数名称不同不算**
 - 比如area面积，圆、三角形、长方形
 - 比如distance，二维/三维、
- 覆盖
 - 覆盖表现为父类与子类之间方法的多态性
 - 子类定义了父类的同名方法，方法头不变，但方法体内容不同

• final类和final成员

- final pi=3.14 定义符号常量
- 修饰类及类中的成员变量和方法成员
- 特点
 - 用final修饰的类不能被继承
 - 用final修饰的成员方法不能被覆盖
 - 用final修饰的成员变量不能被修改
- final类
 - final class 类名{类体}
 - 不能被继承
- final成员方法
 - 方法不允许被覆盖
- final成员变量
 - 其值不能改变

第八章 接口和包

2020年6月13日 21:20

- 抽象类和方法

- 为实现多继承-接口
- 接口是一种特殊的类
 - 一组常量
 - 抽象方法-只有方法头，没有方法体
- 包
 - java程序的组织形式，类和接口的集合
 - 包括一组类、接口
- 抽象类是**供子类继承、却不能创建实例的类**，用abstract修饰
- 抽象类中
 - 可以声明**只有方法头、没有方法体的抽象方法**，也可以没有抽象方法
 - 由抽象方法的类必须声明成抽象类
 - 抽象类用于描述抽象的概念
 - 抽象方法约定了多个子类共用的方法头
- 抽象类的子类

◆ **Java中，任何一个类都可以定义子类**

◆ **抽象类的子类**

- - ◆ 必须完成父类定义的每一个抽象方法，除非该子类也是抽象类。
 - ◆ 它的主要用途是用来描述概念性的内容，这样可以提高开发效率，更好地统一用户“接口”

- 抽象方法和抽象类
 - 构造方法不能声明为抽象方法

- **[权限修饰符] abstract 返回类型 方法名(参数表);**

```
[权限修饰符] abstract class 类名{  
    成员变量;  
    abstract 方法名(); //定义抽象方法  
}
```

- 抽象类中可以定义非抽象方法
- 抽象类

- 抽象类体中可以定义非抽象方法
- 抽象类体中，可以包含抽象方法，也可以不包含抽象方法
 - 类体中包含抽象方法的类必须声明为抽象类
- 抽象类只能被继承，不能实例化，即便抽象类不包含抽象方法
- 抽象类的子类必须给出每个抽象方法的具体实现
 - 若有个抽象方法在子类中没有被实现，该子类也必须被声明为抽象类

• 接口 interface

- 接口可以用来实现类间多继承结构
 - **一个类只能继承于一个父类，但是可以实现多个接口**
- 接口内部只能定义 **public** 的抽象方法和 **静态的、公有常量**，所有的方法需要在实现接口的类中实现

```
[访问权限] interface 接口名 [extends 父接口名列表] {
    成员变量表(常量)
    成员方法列表 (抽象)
}
```

- 接口权限只有两种：**public**和**缺省**
- 接口体中定义的方法都是抽象、公有的
 - **public**和**abstract**可以省略
- 成员变量只能是静态的、公有的常量
 - **public**、**static**、**final**可以省略
- 一个接口可以继承其他接口，称子接口
- 接口允许没有父接口，即接口不存在最高层
 - 类有最高层，即Object类
- 接口实现

```
class 类名 [extends 父类名] implements 接口名列表 {
    类体
}
```

说明

- 一个类可以实现多个接口，各个接口之间用逗号分开
- 该类必须要实现接口中所有的抽象方法，即使本类中不使用的抽象方法也要实现，即方法体空写
- 对不使用的抽象方法
 - 空方法：方法中没有语句，实现不需要返回值的方法
 - 返回默认值：在方法中使用默认值返回，如0
- 实现抽象方法，需指定访问权限为**public**（即使接口中定义时没写）
- 实现接口抽象方法，需指定访问权限为**public**

- ◆ 接口不存在最高层
 - ◆ 类有最高层: **Object**类
- ◆ 接口中的方法只能被声明为**public**和**abstract**, 如果不声明, 则默认为**public abstract**;
- ◆ 接口中的成员变量只能用**public**、**static**和**final**来声明, 如果不声明, 则默认为**public static final**。
- ◆ 接口中只给出方法名、返回值和参数表, 而不能定义方法体

• 接口和抽象类的异同

- 相同: 都不能被实例化
- 不同
 - 抽象类
 - 一个类只能继承一个抽象类, 是单继承
 - 抽象类是类, 类中可以包含成员变量、构造方法、抽象方法和普通方法
 - 抽象类中的成员具有和普通类中的成员一样的访问权限
 - 接口
 - 一个类可以实现多个接口, 具有多重继承功能
 - 接口是一组抽象方法和常量的组合
 - 接口中成员访问权限只能是public

• 包

- 包是一组相关的类和接口的集合
- 将类和接口分装在不同的包中, 可以避免重名类的冲突
- 使用包机制
 - 首先要建立与包名相同的文件夹
 - 再声明类或接口所在的包
 - 包中所包含的所有类或接口的字节码文件存放于与包同名的文件夹中
- 包的声明
 - package 包名
 - **package语句必须位于程序的第一行**
 - 一个源程序文件中**只能有一条package语句**
 - 在该源程序文件中所定义的所有类和接口, 都属于package语句所声明的包
 - 子包和其父包及祖先包名之间用 "." 隔开
 - 包名与文件夹名大小写要完全一致
- 引用包中的类或接口
 - 一个源程序导入包中的类或接口后, 引用该类和接口时, 包名可省略
 - 一个类如果需要引用其他包中的类或接口, 格式为
 - 包名.类名 或 包名.接口名

- 导入包中的类和接口
 - import 包名.类名 或 import 包名.接口名 或 import 包名.* (表示导入包中所有的类或接口, 但是**不包括该包子包中的类或接口**)
 - import语句可以不止一条
 - **import语句在源程序中必须位于其它类或接口申明之前**
- 一个源程序文件中 (.java) 可以定义多个类或接口, **但只能定义一个public类或public接口, 并且该类或接口名与文件名相同**

第九章 异常处理

2020年9月1日 14:14

- **Java异常处理机制**

- 引进很多用来描述和处理异常的类——异常类
- 异常情况分为
 - **Exception异常**
 - 解决程序本身及应用环境产生多产生的一场，可以在应用程序中捕获并进行相应处理
 - **Error错误**
 - 处理较少发生的内部系统错误，程序员无能为力



- **异常处理方式**

```
public Exception(); //构造方法
```

```
public Exception(String s); //构造方法
```

```
public String toString(); //将异常情况转成字符串
```

```
public String getMessage(); //得到异常情况描述
```

- 两种方法
 - 使用**try...catch...finally**结构对一场进行捕获和处理
 - 若有异常，终止try代码块运行，跳转到对应的catch块中
 - 无论异常是否发生，finally中的代码最终必定执行
 - 通过**throw (语句)** 和**throws (选项)** 抛出异常
 - 通常情况下异常由系统自动捕获
 - 如果不想捕获可以通过throw语句抛出异常
 - ◆ 格式为: **throw new 异常类名 (信息)**
 - ◇ 异常类名为系统异常类名或用户自定义的异常类名
 - ◇ “信息” 是可选信息
 - ▶ 若有该信息，toString () 方法中将增加该信息内容
 - ▶ toString () 方法用于返回系统给出的异常信息

□ throws短语

■ 定义一个方法时可以声明该方法代码被调用执行时可以抛出某个异常对象，让调用该方法的方法去捕获这个异常对象作相应处理。

[修饰符] 返回值类型 方法名 [(参数表)] [throws 异常类型名]

```
{  
    变量声明  
    语句  
}
```

○ 自定义异常类，通过继承Exception类来实现

- Class 自定义异常类名 extends Exception{ 异常类体; }

第十章 Java输入

2020年9月1日 15:28

- **Java的输入输入**

- 以流 (stream) 的方式来处理的。如字节流、字符流等
- 流式输入、输出的特点是数据的获取和发送均按数据序列顺序进行
- Java系统提供两个最基本的类对字节流进行处理
 - InputStream输入流类
 - OutputStream输出流类
 - 及他们的子类
- Java提供Reader类和Writer类及它们的子类对字符流进行处理
- 这些类都在java.io包 (InputStream, OutputStream, Reader, Writer, File) 中

- **标准输入/输出**

- 输入
 - 标准输入
 - 在一般应用程序中
 - ◆ 频繁地向标准输出设备即显示器输出信息
 - ◆ 频繁地向标准输入设备即键盘输入信息
 - Java系统预先定义2个流对象分别表示标准输入、标准输出
 - ◆ System.in 标准输入设备：键盘
 - ◆ System.out 标准输出设备：显示器
 - **System.in属于InputStream类的对象**，使用read()方法可以从键盘读入数据
 - ◆ c= (char) System.in.read()
 - ◆ 使用read () 时，应对IOException类异常进行捕获或抛出
 - 利用BudderedReader类 (Reader类的子类) 的readLine()方法可以从键盘输入字符串，
 - 类似Scanner类的nextLine()
 - 其他输入方式——Scanner类
 - 其他输入方式——命令行输入
 - String[] args里面可以赋值，在运行-运行配置的自变量中输入
- 标准输出
 - **System.out属于PrintStream类 (OutputStream类的子类) 对象**
 - PrintStream类的print () 方法和println () 方法输出各类数据
 - 输出格式的控制
 - “\t” 水平制表符，输出一个相当于按一个tab键，表示的间距是8列
 - 使用System.out.printf () 方法输出，在jdk1.5及之后的版本才支持的
 - System.out.printf (“格式说明” , 数据项)

- %: 格式说明的其实符号, %f十进制浮点数, %o八进制, %x十六进制, %u无符号十进制数, %%输出百分号
- -: 有-表示左对齐, 省略则右对齐输出
- m.n
 - ◆ m指域宽, 对应的输出项在输出设备上所占宽度 (列数)
 - ◆ n指精度, 用于说明输出的实型数的小数位数
- 举例子: "%-4d", "%6.2f"

第十一章 Java进阶——Java Applet

2020年9月1日 16:08

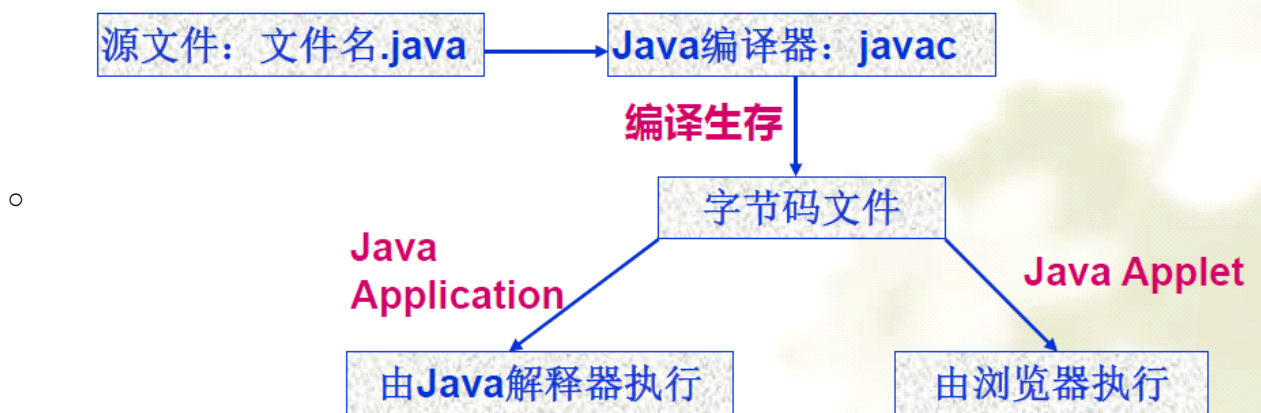
- **Java程序分为**

- **Java应用程序 Java Application**

- 必须通过Java解释器 (java.exe) 来解释执行其字节码文件，能够独立运行

- **Java小应用程序 Java Applet**

- 不能独立运行，必须借助浏览器才能运行
 - Applet程序继承java.applet.Applet类，嵌入HTML文档中，通常置于服务器端，客户端下载后通过浏览器运行



- **多线程机制**

- 使用多线程技术可以使系统同时运行多个执行体
 - 线程的生命周期
 - 每个Java程序都有一个主线程，即main () 方法对应的线程
 - 要实现多线程，必须在主线程中创建新的线程
 - 线程用Thread类及其子类的对象来表示
 - 线程从新生到死亡的状态变化称为生命周期
 - 新生->就绪->运行->阻塞->死亡
 - 在Java中，创建线程的方法有两种：
 - 创建Thread类的子类
 - 实现Runnable接口的类

