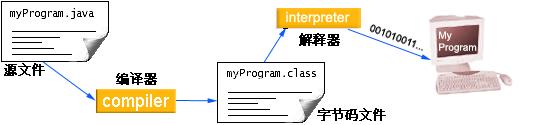
在基础班我们会学习Java的基础知识，即JavaSE（标准版）部分。学习完JavaSE并不能开发大型项目。

在就业班我们会学习Java的高级知识以及整个Java生态圈，即JavaEE（企业版）部分。学习完JavaEE部分就可以开发各种大型项目了。



在开始案例之前，我们需要了解一下Java应用程序的编写流程。

通过上图我们可以了解到编写的程序大致如下；

1. 源文件：编写Java源文件（我们也称之为源代码文件），它的扩展名为.java；
2. 编译：然后通过编译器把源文件编译成字节码文件，字节码文件扩展名为.class；
3. 运行：最后使用解释器来运行字节码文件。

Java中提供了三种注释方式，分别是：

* 单行注释 //注释内容
* 多行注释 /\*注释内容\*/
* 文档注释 /\*\*注释内容\*/

其中文档注释与多行注释作用基本相同，唯一的区别是文档注释可以使用javadoc.exe命令生成API文档。

在Java程序中，除了关键字以外基本上都是标识符了。

其实标识符就是名称的意思，所有的名称都统称为标识符。Java中经常要定义类、方法、变量（后面会学习到），在定义他们时总要给它们起名字，这些名字就是标识符。

这里我们要学习的是如何起名字，你可能会说起名字还要学吗？答案是当然要学了！我们要学习的就是标识符的规范。

* 组成元素
  + 英文字符: a-zA-Z
  + 数字: 0-9
  + 符号: \_与$
* 标识符规则
  + 数字不能开头
  + 不可以使用关键字
  + 严格区分大小写，不限制长度起名时，尽量达到见名知意

基本数据类型是Java语言中内置的类型，分别是整数类型、小数类型、字符类型、布尔类型。这四类基本类型是最简单、最基础的类型。

引用数据类型是强大的数据类型，它是基于基本数据类型创建的。JavaSE中提供了一个超级类库，类库中包含了近万种引用数据类型。

* 整数类型
  + 十进制表示方式：正常数字 如 13、25等
  + 二进制表示方式：以0b(0B)开头 如0b1011 、0B1001
  + 十六进制表示方式：以0x(0X)开头 数字以0-9及A-F组成 如0x23A2、0xa、0x10
  + 八进制表示方式：以0开头 如01、07、0721
* 小数类型
  + 如1.0、-3.15、3.168等
* 布尔类型
  + true、false
* 字符类型
  + 如'a'，'A', '0', '家'
  + 字符必须使用’’ 包裹，并且其中只能且仅能包含一个字符。
* 字符串类型
  + 字符串String类型是一种引用类型，我们先了解作为常量类型的使用方式
  + 如“我爱Java”，“0123”，“”，“null”
  + 字符串必须使用“”包裹，其中可以包含0~N个字符。

那么计算机内存是怎么存储数据的呢？无论是内存还是硬盘，计算机存储设备的**最小信息单元叫“位（bit）”**，我们又称之为“比特位”，通常用小写的字母b表示。而计算机最小的存储单元叫“字节（byte）”，通常用大写字母B表示，字节是由连续的8个位组成。

当程序需要使用存储空间时，操作系统最小会分派给程序1个字节，而不是1个位。你可能会说，如果程序只需要1个位的空间，系统分派不能只分派1个位吗？答案是不能！这就像你只需要1支烟，你到商店去买烟，商店分派的最小单元是1盒（20支），他不可能卖给你1支烟。

你可能会想，1个字节（8位）可以存储很大的数值了，1位最大是9那么8位最大值为99999999。你错了，因为计算机是采用二进行存储的，而不是我们生活中常用的十进制。所以1个字节存储的最大数据是11111111的二进制数。

除了字节外还有一些常用的存储单位，大家可能比较熟悉，我们一起来看看：

1B（字节） = 8bit

1KB = 1024B

1MB = 1024KB

1GB = 1024MB

1TB = 1024GB

1PB = 1024TB

Java中基本类型一共4类，把这4类展开后共8种基本类型。我们今后编写程序时使用的是这8种基本类型而不是4类，这8种基本类型指定了范围。

|  |  |  |  |
| --- | --- | --- | --- |
| **四类** | **八种** | **字节数** | **数据表示范围** |
| **整型** | **byte** | 1 | -128～127 |
| **short** | 2 | -32768～32767 |
| **int** | 4 | -2147483648～2147483648 |
| **long** | 8 | -263～263-1 |
| **浮点型** | **float** | 4 | -3.403E38～3.403E38 |
| **double** | 8 | -1.798E308～1.798E308 |
| **字符型** | **char** | 2 | 表示一个字符，如('a'，'A'，'0'，'家') |
| **布尔型** | **boolean** | 1 | 只有两个值true与false |

你可能会认为12345678901在-263~263-1之间，所以它是long类型。注意了，这是错误的！！！在Java中整数常量如果不在-2147483648~2147483648之间就必须添加“L”后缀（小写的也可以，但建议使用大写），在-2147483648~2147483648之间的也可以添加“L”后缀。也就是说12345678901不在-2147483648~2147483648之间，所以它在Java中是错误的常量，你必须这样写：12345678901L，这才是正确的常量。所以添加了“L”后缀的整数常量都是long类型的，例如：100L、12345678901L都是long类型的常量。

范围小的数据类型值（如byte），可以直接转换为范围大的数据类型值（如int）；

范围大的数据类型值（如int），不可以直接转换为范围小的数据类型值（如byte）

那么，大家还记得每种类型表示数据的范围吗？忘记了不要紧，我来告诉大家，将各种数据类型按照数据范围从小到大依次列出：

byte -> short -> int -> long -> float -> double

## 三元运算符

接下来我们要学习的三元运算符与之前的运算符不同。之前学习的均为一元或者二元运算符。元即参与运算的数据。

* 格式：

(条件表达式)？表达式1：表达式2；

* 表达式：通俗的说，即通过使用运算符将操作数联系起来的式子，例如：
  + 3+2，使用算数运算符将操作数联系起来，这种情况，我们称为算数表达式。
  + 3>2，使用比较运算符（也称为条件运算符）将操作数联系起来，这种情况，我们称为条件表达式。
  + 其他表达式，不再一一举例。
* 三元运算符运算规则：

先判断条件表达式的值，若为true，运算结果为表达式1；若为false，运算结果为表达式2。

一个比较复杂的代码：

int a = 5;

int b = 3;

int c = 1;

int n2 = (a>b && b>c) ? (c++) : (++c);

这段代码运算执行顺序我们也写一下：

* 1.小括号优先级高，我们先运算第一组小括号中的代码
* 1.1. 比较运算符”>” 优先级大于 逻辑运算符”&&”
  + 先执行 a>b,得到结果true；
  + 再执行 b>c,得到结果true；
  + 最后执行 a>b的结果 && b>c的结果，即 true && true, 结果为true
* 2.三元运算符中的条件判断结果为true，返回表达式1的结果 c++
  + 先将变量c的原有值赋值给变量n2，即n2值为1；
  + 再将变量c的值自增1,更新为2。

Scanner类，我们编写代码来使用下它：ScannerDemo01.java

**import** java.util.Scanner;

**public** **class** ScannerDemo01 {

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

//创建Scanner引用类型的变量

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

//获取数字

System.*out*.println("请输入一个数字");

**int** n = sc.nextInt();

System.*out*.println("n的值为" + n);

//获取字符串

System.*out*.println("请输入一个字符串");

String str = sc.next();

System.*out*.println("str的值为" + str);

}

}

* Random使用方式:
  + import导包：所属包java.util.Random
  + 创建实例格式：Random 变量名 = new Random();

接下来，通过一段代码，一起学习下Random类的使用，RandomDemo.java

**import** java.util.Random;

**public** **class** RandomDemo {

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

// 创建Random类的实例

Random r = **new** Random();

// 得到0-100范围内的随机整数，将产生的随机整数赋值给i变量

**int** i = r.nextInt(100);

//得到0.0-1.0范围内的随机小数，将产生的随机小数赋值给d变量

**double** d = r.nextDouble();

System.*out*.println(i);

System.*out*.println(d);

}

}

判断奇偶数的程序，IfDemo02.java

**public** **class** IfDemo02 {

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

**int** num = 19;

**if** (num % 2 == 0) {

// 判断条件成立，num被2整除

System.*out*.println("num是一个偶数");

} **else** {

System.*out*.println("num是一个奇数");

}

}

对学生考试成绩进行等级划分的程序，IfDemo03.java

**public** **class** IfDemo03 {

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

**int** grade = 75; // 定义学生成绩

**if** (grade > 80) {

// 满足条件 grade > 80

System.*out*.println("该成绩的等级为优");

} **else** **if** (grade > 70) {

// 不满足条件 grade > 80 ，但满足条件 grade > 70

System.*out*.println("该成绩的等级为良");

} **else** **if** (grade > 60) {

// 不满足条件 grade > 70 ，但满足条件 grade > 60

System.*out*.println("该成绩的等级为中");

} **else** {

// 不满足条件 grade > 60

System.*out*.println("该成绩的等级为差");

}

}

}

实现打印1~4之间的自然数，WhileDemo.java

**public** **class** WhileDemo {

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

**int** x = 1; // 定义变量x，初始值为1

**while** (x <= 4) { // 循环条件

System.*out*.println("x = " + x); // 条件成立，打印x的值

x++; // x进行自增

}

}

}

代码x++用于在每次循环时改变变量x的值，从而达到最终改变循环条件的目的。如果没有这行代码，整个循环会进入无限循环的状态，永远不会结束。

①表示初始化表达式、②表示循环条件、③表示操作表达式、④表示循环体，通过序号来具体分析for循环的执行流程。具体如下：

for（① ; ② ; ③）{

④

}

第一步，执行①

第二步，执行②，如果判断结果为true，执行第三步，如果判断结果为false，执行第五步

第三步，执行④

第四步，执行③，然后重复执行第二步

第五步，退出循环

接下来通过一个案例对自然数1~4进行求和，如下所示。ForDemo01.java

**public** **class** ForDemo01 {

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

**int** sum = 0; // 定义变量sum，用于记住累加的和

**for** (**int** i = 1; i <= 4; i++) { // i的值会在1~4之间变化

sum += i; // 实现sum与i的累加

}

System.*out*.println("sum = " + sum); // 打印累加的和

}

}

do…while循环语句来实现打印1~4之间的自然数DoWhileDemo.java。

**public** **class** DoWhileDemo {

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

**int** x = 1; // 定义变量x，初始值为1

**do** {

System.*out*.println("x = " + x); // 打印x的值

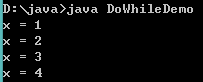
x++; // 将x的值自增

} **while** (x <= 4); // 循环条件

}

}

运行结果如下图所示。



1. 运行结果

我们发现WhileDemo.java与DoWhileDemo.java运行的结果一致，这说明什么呢？这说明do …while循环和while循环能实现同样的功能。然而在程序运行过程中，这两种语句还是有差别的。如果循环条件在循环语句开始时就不成立，那么while循环的循环体一次都不会执行，而do…while循环的循环体还是会执行一次。若将DoWhileDemo.java中的循环条件x<=4改为x < 1，DoWhileDemo.java运行结果会打印x=1，而WhileDemo.java运行结果什么也不会打印。

## 无限循环

最简单无限循环格式：

while(true){}

或

for(;;){}

无限循环存在的原因是并不知道循环多少次，而是根据某些条件，来控制循环。

实现使用“\*”打印直角三角形，如下所示。ForForDemo.java

1. **public** **class** ForForDemo {
2. **public** **static** **void** main(String[] args) {
3. **int** i, j; // 定义两个循环变量
4. **for** (i = 1; i <= 9; i++) { // 外层循环
5. **for** (j = 1; j <= i; j++) { // 内层循环
6. System.*out*.print("\*"); // 打印\*
7. }
8. System.*out*.print("\n"); // 换行
9. }
10. }
11. }



**continue语句**

continue语句用在循环语句中，它的作用是终止本次循环，执行下一次循环。接下来通过一个练习对1~100之内的奇数求和，ContinueDemo.java

**public** **class** ContinueDemo {

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

**int** sum = 0; // 定义变量sum，用于记住和

**for** (**int** i = 1; i <= 100; i++) {

**if** (i % 2 == 0) { // i是一个偶数，不累加

**continue**; // 结束本次循环

}

sum += i; // 实现sum和i的累加

}

System.*out*.println("sum = " + sum);

}

}

switch语句中使用switch关键字来描述一个表达式，使用case关键字来描述和表达式结果比较的目标值，当表达式的值和某个目标值匹配时，会执行对应case下的语句。具体实现代码如下：

switch(用于表示星期的数字) {

case 1 :

输出星期一;

break;

case 2 :

输出星期二;

break;

case 3 :

输出星期三

break;

case 4 :

输出星期四;

break;

case 5 :

输出星期五;

break;

case 6:

输出星期六;

break;

case 7:

输出星期天;

break;

}

定义数组时只指定数组的长度，由系统自动为元素赋初值的方式称作动态初始化。

在初始化数组时还有一种方式叫做静态初始化，就是在定义数组的同时就为数组的每个元素赋值。数组的静态初始化有两种方式，具体格式如下：

1、类型[] 数组名 = new 类型[]{元素，元素，……};

2、类型[] 数组名 = {元素，元素，元素，……};

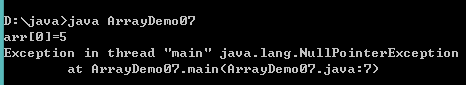
上面的两种方式都可以实现数组的静态初始化，但是为了简便，建议采用第二种方式。

每个数组的索引都有一个范围，即0~length-1。在访问数组的元素时，索引不能超出这个范围，否则程序会报错，数组越界异常ArrayIndexOutOfBoundsException。

在使用变量引用一个数组时，变量必须指向一个有效的数组对象，如果该变量的值为null，则意味着没有指向任何数组，此时通过该变量访问数组的元素会出现空指针异常，接下来通过一个案例来演示这种异常，如下所示。ArrayDemo07.java

1. **public** **class** ArrayDemo07 {
2. **public** **static** **void** main(String[] args) {
3. **int**[] arr = **new** **int**[3]; // 定义一个长度为3的数组
4. arr[0] = 5; // 为数组的第一个元素赋值
5. System.*out*.println("arr[0]=" + arr[0]); // 访问数组的元素
6. arr = **null**; // 将变量arr置为null
7. System.*out*.println("arr[0]=" + arr[0]); // 访问数组的元素
8. }
9. }

运行结果如下图所示。



1. 运行结果

通过上图所示的运行结果可以看出，上述代码中第4、5行代码都能通过变量arr正常地操作数组。第6行代码将变量置为null，当第7行代码再次访问数组时就出现了空指针异常NullPointerException。

一个案例来熟悉二维数组的使用。

例如要统计一个公司三个销售小组中每个小组的总销售额以及整个公司的销售额。如下所示

* 第一小组销售额为{11, 12}万元
* 第二小组销售额为{21, 22, 23}万元
* 第三小组销售额为{31, 32, 33, 34}万元。

ArrayDemo08.java

1. **public** **class** ArrayDemo10 {
2. **public** **static** **void** main(String[] args) {
3. **int**[][] arr = **new** **int**[3][]; // 定义一个长度为3的二维数组
4. arr[0] = **new** **int**[] { 11, 12 }; // 为数组的元素赋值
5. arr[1] = **new** **int**[] { 21, 22, 23 };
6. arr[2] = **new** **int**[] { 31, 32, 33, 34 };
8. **int** sum = 0; // 定义变量记录总销售额
9. **for** (**int** i = 0; i < arr.length; i++) { // 遍历数组元素
10. **int** groupSum = 0; // 定义变量记录小组销售总额
11. **for** (**int** j = 0; j < arr[i].length; j++) { // 遍历小组内每个人的销售额
12. groupSum = groupSum + arr[i][j];
13. }
14. sum = sum + groupSum; // 累加小组销售额
15. System.*out*.println("第" + (i + 1) + "小组销售额为：" + groupSum + " 万元");
16. }
17. System.*out*.println("总销售额为: " + sum + " 万元");
18. }
19. }

在全班同学中随机的打印出一名同学名字。

我们对本案例进行分析，得出如下分析结果：

1.存储全班同学名字

2.打印全班同学每一个人的名字

3.在班级总人数范围内，随机产生一个随机数，查找该随机数所对应的同学名字

在存储同学姓名时，如果对每一个同学都定义一个变量进行姓名存储，则会出现过多孤立的变量，很难一次性将全部数据持有。此时，我们可以使用数组解决多个数据的存储问题。

编写CallName.java文件，完成程序的编写。

* main方法中进行步骤1、2、3的代码实现

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

System.*out*.println("--------随机点名器--------");

// 创建一个存储多个同学名字的容器（数组）

String[] students = **new** String[3];

//1.存储全班同学名字

//2.打印全班同学每一个人的名字

//3.获取随机点名到的学生姓名，并打印

}

* 1.存储所有同学姓名

/\*\*

\* 1.存储全班同学名字

\* 创建一个存储多个同学名字的容器（数组）

\* 键盘输入每个同学的名字，存储到容器中（数组）

\*/

//键盘输入多个同学名字存储到容器中

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

**for** (**int** i = 0; i < students.length; i++) {

System.*out*.println("存储第"+i+"个名称：");

students[i] = sc.next();

}

* 2.打印全班同学每一个人的名字

/\*\*

\* 2打印全班同学每一个人的名字

\*/

//遍历数组，得到每个同学名字

**for** (**int** i = 0; i < students.length; i++) {

String name = students[i];

//打印同学名字

System.*out*.println("第"+i+"个学生名称：" + name);

}

* 3.获取随机点名到的学生姓名，并打印

/\*\*

\* 3.在班级总人数范围内，随机产生一个随机数

\*/

//根据数组长度，获取随机索引

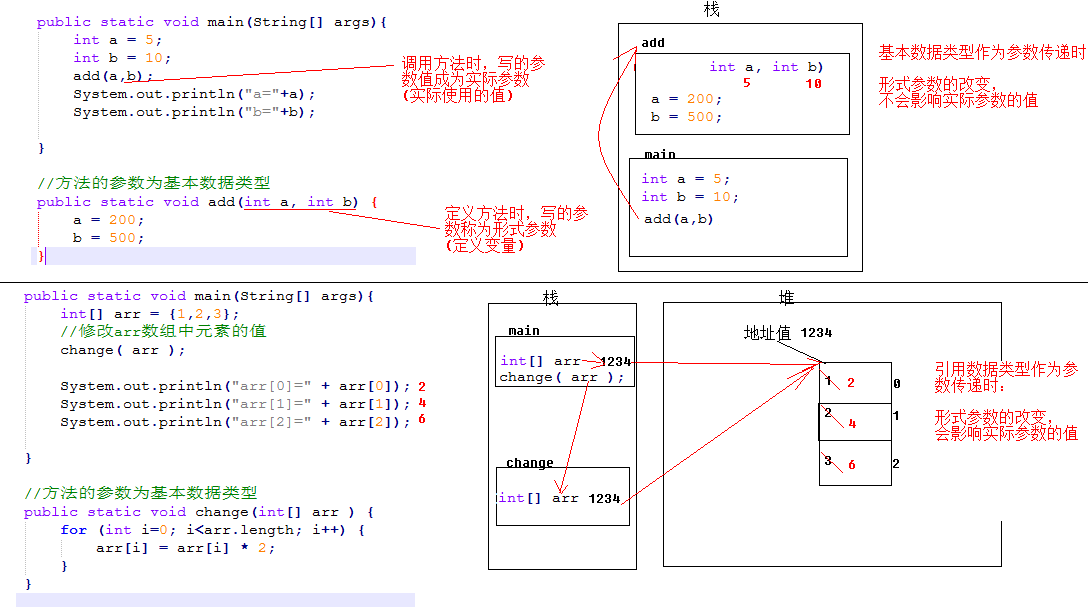
**int** index = **new** Random().nextInt(students.length);

//通过随机索引从数组中获取名称

String name = students[index];

//返回随机点到的名称

上述代码中，通过随机数类Random产生一个从0到数组长度的随机索引。使用该索引获取students数组中对应的值，便得到了全班同学的随机姓名。



通过上面的两段程序可以得出如下结论：

* 当调用方法时，如果传入的数值为基本数据类型（包含String类型），形式参数的改变对实际参数不影响
* 当调用方法时，如果传入的数值为引用数据类型（String类型除外），形式参数的改变对实际参数有影响

随机点名器案例2 用定义方法的方法实现

* main方法中调用三个独立方法

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

System.*out*.println("--------随机点名器--------");

// 创建一个存储多个同学名字的容器（数组）

String[] students = **new** String[3];

/\*

\* 1.存储全班同学名字

\*/

addStudentName(students);

/\*

\* 2.打印全班同学每一个人的名字

\*/

printStudentName(students);

/\*

\* 3.获取随机点名到的学生姓名，并打印

\*/

String randomName = *randomStudentName*(students);

System.*out*.println("被点到名的同学是 :" + randomName);

}

* 1.存储所有同学姓名

/\*\*

\* 1.存储全班同学名字

\* 创建一个存储多个同学名字的容器（数组）

\* 键盘输入每个同学的名字，存储到容器中（数组）

\*/

**public** **static** **void** addStudentName(String[] students) {

//键盘输入多个同学名字存储到容器中

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

**for** (**int** i = 0; i < students.length; i++) {

System.*out*.println("存储第"+i+"个名称：");

students[i] = sc.next();

}

}

上述方法中，通过键盘录入，完成为指定数组元素赋值。方法定义时，将参数定义为字符串数组，用于接收存放的同学姓名。

* 打印全班同学每一个人的名字

/\*\*

\* 2打印全班同学每一个人的名字

\*/

**public** **static** **void** printStudentName(String[] students) {

//遍历数组，得到每个同学名字

**for** (**int** i = 0; i < students.length; i++) {

String name = students[i];

//打印同学名字

System.*out*.println("第"+i+"个学生名称：" + name);

}

}

上述方法中，方法参数students数组中存储了所有学生的姓名。通过遍历将数组中的每一个元素访问一遍，得到每一个同学名称。

* 3.获取随机点到的学生姓名

/\*\*

\* 3.在班级总人数范围内，随机产生一个随机数,返回随机数位置上的学生姓名

\*/

**public** **static** String randomStudentName(String[] students) {

//根据数组长度，获取随机索引

**int** index = **new** Random().nextInt(students.length);

//通过随机索引从数组中获取名称

String name = students[index];

//返回随机点到的名称

**return** name;

}

上述方法中，通过随机数类Random产生一个从0到数组长度的随机索引。使用该索引获取students数组中对应的值，便得到了全班同学的随机姓名。

库存管理管理案例

* 功能菜单

/\*\*

\* 库存管理功能菜单

\* **@return** 管理员键盘输入的功能操作序号

\*/

**public** **static** **int** chooseFunction() {

System.*out*.println("-------------库存管理------------");

System.*out*.println("1.查看库存清单");

System.*out*.println("2.修改商品库存数量");

System.*out*.println("3.退出");

System.*out*.println("请输入要执行的操作序号：");

//接收键盘输入的功能选项序号

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

**int** choose = sc.nextInt();

**return** choose;

}

上述方法用来完成库存管理功能菜单的显示、接收管理员选择的功能操作序号。这是完成了案例的第一步。接下来完成“查看、修改、退出”这三项功能。

* 编写main主方法，调用库存管理功能菜单方法，与“查看、修改、退出”这三个方法。

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

//记录库存商品信息

//品牌型号

String[] brands = **new** String[]{"MacBookAir", "ThinkpadT450"};

//尺寸大小

**double**[] sizes = **new** **double**[]{13.3, 14.0};

//价格

**double**[] prices = **new** **double**[]{6988.88, 5999.99};

//库存个数

**int**[] counts = **new** **int**[]{0, 0};

//通过while循环模拟管理员进行功能重复选择操作

while (true) {

//打印功能菜单操作,接收键盘输入的功能选项序号

**int** choose = *chooseFunction*();

//执行序号对应的功能

**switch** (choose) {

**case** 1://查看库存清单

*printStore*(brands, sizes, prices, counts);

**break**;

**case** 2://修改商品库存数量

*update*(brands, counts);

**break**;

**case** 3://退出

*exit*();

**return**;

default:

System.*out*.println("----------------------------------");

System.*out*.println("功能选择有误，请输入正确的功能序号!");

**break**;

}

}

}

在主方法中，创建了5个数组，分别用来存储商品的品牌型号、尺寸大小、价格、配置、库存个数，通过接收到的功能选项序号，进行switch语句判断后，调用对应的功能方法。

* 查看库存清单功能

/\*\*

\* 查看库存清单

\* **@param** brands 商品品牌型号

\* **@param** sizes 商品尺寸大小

\* **@param** prices 商品价格

\* **@param** counts 商品库存个数

\*/

**public** **static** **void** printStore(String[] brands, **double**[] sizes, **double**[] prices, **int**[] counts) {

//统计总库存个数、统计库存总金额

**int** totalCount = 0;

**double** totalMoney = 0.0;

**for** (**int** i = 0; i < brands.length; i++) {

totalCount += counts[i];

totalMoney += counts[i] \* prices[i];

}

//列表顶部

System.*out*.println("---------------------------查看库存清单--------------------------");

System.*out*.println("品牌型号 尺寸 价格 库存数");

//列表中部

**for** (**int** i = 0; i < brands.length; i++) {

System.*out*.println(brands[i]+" "+sizes[i]+" "+prices[i]+" "++counts[i]);

}

//列表底部

System.*out*.println("-------------------------------------------------------------");

System.*out*.println("总库存数："+totalCount);

System.*out*.println("库存商品总金额："+totalMoney);

}

上述方法用来完成打印库存清单功能，5个方法参数用来打印的库存商品相关信息

* 修改商品库存数量功能

/\*\*

\* 修改商品库存数量

\* **@param** brands 商品品牌型号

\* **@param** counts 商品库存个数

\*/

**public** **static** **void** update(String[] brands, **int**[] counts){

System.*out*.println("------------修改商品库存数量-----------");

**for** (**int** i = 0; i < brands.length; i++) {

System.*out*.println("请输入"+ brands[i] +"商品库存数");

counts[i] = **new** Scanner(System.*in*).nextInt();

}

}

上述方法用来完成修改商品库存数量功能，2个方法参数用来指定所要修改的商品与库存数

* 退出功能

/\*\*

\* 退出

\*/

public static void exit(){

System.*out*.println("----------------退出---------------");

System.*out*.println("您已退出系统");

}

上述方法用来完成退出程序的功能

数组可以保存多个元素，但在某些情况下无法确定到底要保存多少个元素，此时数组将不再适用，因为数组的长度不可变。例如，要保存一个学校的学生，由于不停有新生来报道，同时也有学生毕业离开学校，这时学生的数目很难确定。为了保存这些数目不确定的元素，JDK中提供了一系列特殊的类，这些类可以存储任意类型的元素，并且长度可变，统称为集合。在这里，我们先介绍ArrayList集合，其他集合在后续课程中学习。

ArrayList集合是程序中最常见的一种集合，它属于引用数据类型（类）。在ArrayList内部封装了一个长度可变的数组，当存入的元素超过数组长度时，ArrayList会在内存中分配一个更大的数组来存储这些元素，因此可以将ArrayList集合看作一个长度可变的数组。

8种基本数据类型所对应的引用数据类型表示形式:

|  |  |
| --- | --- |
| **基本数据类型** | **对应的引用数据类型表示形式** |
| **byte** | Byte |
| **short** | Short |
| **Int** | Integer |
| **long** | Long |
| **float** | Float |
| **double** | Double |
| **char** | Character |
| **boolean** | Boolean |

## 点名器arraylist方法实现代码步骤

每名学生都拥有多项个人信息，为了方便管理每个人的信息，我们对学生信息进行封装，编写Student.java文件

/\*\*

\* 学生信息类

\*/

**public** **class** Student {

String name; //姓名

**int** age; //年龄

}

上述代码中，对学生信息（姓名、年龄）进行了封装。这样做的好处在于，以后只要找到这名学生，就能够知道他的每项个人信息了。

接下来我们编写CallName.java文件，完成程序的编写。

* main方法中调用三个独立方法

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

ArrayList<Student> list = **new** ArrayList<Student>(); //1.1创建一个可以存储多个同学名字的容器

/\*

\* 1.存储全班同学信息

\*/

addStudent(list);

/\*

\* 2.打印全班同学每一个人的信息（姓名、年龄）

\*/

printStudent(list);

/\*

\* 3.随机对学生点名，打印学生信息

\*/

randomStudent(list);

}

* 存储所有学生的个人信息

/\*\*

\* 1.存储全班同学名字

\*/

**public** **static** **void** addStudent(ArrayList<Student> list) {

//键盘输入多个同学名字存储到容器中

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

**for** (**int** i = 0; i < 3; i++) {

//创建学生

Student s = **new** Student();

System.*out*.println("存储第"+i+"个学生姓名：");

s.name = sc.next();

System.*out*.println("存储第"+i+"个学生年龄：");

s.age = sc.nextInt();

//添加学生到集合

list.add(s);

}

}

上述方法中，方法参数list中用来表示已存储所有学生。通过Scanner，完成新学生信息（姓名，年龄）的录入，并将学生添加到集合中。

* 打印全班同学每一个人的信息

/\*\*

\* 2.打印全班同学每一个人的信息（姓名、年龄）

\*/

**public** **static** **void** printStudent (ArrayList<Student> list) {

**for** (**int** i = 0; i < list.size(); i++) {

Student s = list.get(i);

System.*out*.println("姓名："+s.name +",年龄："+s.age);

}

}

上述方法中，方法参数list中用来表示已存储所有学生。通过遍历集合中的每个元素，得到每个同学信息，并输出打印。

* 随机对学生点名，打印学生信息

/\*\*

\* 3.随机对学生点名，打印学生信息

\*/

**public** **static** **void** randomStudent (ArrayList<Student> list) {

//在班级总人数范围内，随机产生一个随机数

**int** index = **new** Random().nextInt(list.size());

//在容器（ArrayList集合）中，查找该随机数所对应的同学信息（姓名、年龄）

Student s = list.get(index);

System.*out*.println("被随机点名的同学："+s.name + "，年龄:" + s.age);

}

上述方法中，通过随机数类Random产生一个从0到集合长度的随机索引。使用该索引获取ArrayList集合中对应的值，便得到了全班同学的随机学生信息并打印。

库存管理 arraylist方法实现

每种库存商品都拥有多项商品信息，为了方便管理每种商品的信息，我们对商品信息进行封装，编写Goods.java文件

/\*

\* 库存商品类

\*/

**public** **class** Goods {

String brand; // 商品品牌型号

**double** size; // 商品尺寸大小

**double** price; // 商品价格

**int** count; // 商品库存个数

}

上述代码中，对商品信息（品牌、尺寸、价格、库存数）进行了封装。这样做的好处在于，以后只要找到这个商品，就能够知道该商品的每项信息了。

编写Demo库存管理.java，完成如下功能：

* 功能菜单

/\*\*

\* 库存管理功能菜单

\* **@return** 管理员键盘输入的功能操作序号

\*/

**public** **static** **int** chooseFunction() {

System.*out*.println("-------------库存管理------------");

System.*out*.println("1.查看库存清单");

System.*out*.println("2.修改商品库存数量");

System.*out*.println("3.退出");

System.*out*.println("请输入要执行的操作序号：");

//接收键盘输入的功能选项序号

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

**int** choose = sc.nextInt();

**return** choose;

}

上述方法用来完成库存管理功能菜单的显示、接收管理员选择的功能操作序号。这是完成了案例的第一步。接下来完成“查看、修改、退出”这三项功能。

* 编写main主方法，调用库存管理功能菜单方法，与“查看、修改、退出”这三个方法。

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

//记录库存商品信息

ArrayList<Goods> list = **new** ArrayList<Goods>();

//添加商品到库存

*addStore*(list);

//通过while循环模拟管理员进行功能重复选择操作

while (true) {

//打印功能菜单操作,接收键盘输入的功能选项序号

**int** choose = *chooseFunction*();

//执行序号对应的功能

**switch** (choose) {

**case** 1://查看库存清单

*printStore*(list);

**break**;

**case** 2://修改商品库存数量

*update*(list);

**break**;

**case** 3://退出

*exit*();

**return**;

default:

System.*out*.println("----------------------------------");

System.*out*.println("功能选择有误，请输入正确的功能序号!");

**break**;

}

}

}

在主方法中，创建了ArrayList集合，用来存储库存商品信息，通过接收到的功能选项序号，进行switch语句判断后，调用对应的功能方法。

* 查看库存清单功能

/\*\*

\* 查看库存清单

\*/

**public** **static** **void** printStore(ArrayList<Goods> list) {

//统计总库存个数、统计库存总金额

**int** totalCount = 0;

**double** totalMoney = 0.0;

//列表顶部

System.*out*.println("---------------------------查看库存清单--------------------------");

System.*out*.println("品牌型号 尺寸 价格 库存数");

//列表中部

**for** (**int** i = 0; i < list.size(); i++) {

Goods item = list.get(i);

System.*out*.println(item.brand+" "+item.size+" "+item.price+" "+ item.count);

//统计总库存个数、统计库存总金额

totalCount += item.count;

totalMoney += item.count \* item.price;

}

//列表底部

System.*out*.println("-------------------------------------------------------------");

System.*out*.println("总库存数："+totalCount);

System.*out*.println("库存商品总金额："+totalMoney);

}

上述方法用来完成打印库存清单功能，参数list表示库存商品相关信息集合。

* 修改商品库存数量功能

/\*\*

\* 修改商品库存数量

\*/

**public** **static** **void** update(ArrayList<Goods> list){

System.*out*.println("------------修改商品库存数量-----------");

**for** (**int** i = 0; i < list.size(); i++) {

Goods item = list.get(i);

System.*out*.println("请输入"+ item.brand +"商品库存数量");

item.count = **new** Scanner(System.*in*).nextInt();

list.set(i, item);

}

}

上述方法用来完成修改商品库存数量功能，参数list表示库存商品相关信息集合。

* 退出功能

/\*\*

\* 退出

\*/

public static void exit(){

System.*out*.println("----------------退出---------------");

System.*out*.println("您已退出系统");

}

上述方法用来完成退出程序的功能

求 1+3+5+7+……+99 的和值

public class Test01 {

public static void main(String[] args) {

int sum = 0;

for (int i = 0; i < 100; i++) {

if (i%2==1) {

sum += i;

}

}

System.*out*.println("累加和的值 " + sum);

}

}

输出所有的水仙花数

水仙花数是指一个3位数（100-999之间），其每位数字立方之和等于该3位数本身。如153 = 1\*1\*1 + 3\*3\*3 + 5\*5\*5，

即 3位数本身 = 百位数立方 + 十位数立方 + 个位数立方;

public class Test02 {

public static void main(String[] args) {

for (int i = 100; i < 1000; i++) {

int bai = i/100%10;

int shi = i/10%10;

int ge = i%10;

if (i == bai\*bai\*bai + shi\*shi\*shi + ge\*ge\*ge) {

System.*out*.println(i);

}

}

}

}

## ASCII编码表

American Standard Code for Information Interchange，美国标准信息交换代码.

在计算机中，所有的数据在存储和运算时都要使用二进制数表示, a、b、c、d这样的52个字母（包括大写）、以及0、1等数字还有一些常用的符号, 在计算机中存储时也要使用二进制数来表示，而具体用哪些二进制数字表示哪个符号，当然每个人都可以约定自己的一套（这就叫编码），而大家如果要想互相通信而不造成混乱，那么大家就必须使用相同的编码规则，于是美国有关的标准化组织就出台了ASCII编码，统一规定了上述常用符号用哪些二进制数来表示。



数字0-9对应ASCII编码十进制为48-57, 字母a-z对应ASCII编码十进制为97-122，字母A-Z对应ASCII编码十进制为65-90

for循环打印 9\*9 表

public class Test05 {

public static void main(String[] args) {

for (int j = 1; j < 10; j++) {

for (int k = 1; k <= j; k++) {

System.*out*.print(k +"\*"+ j +"="+ j\*k +"\t");

}

System.*out*.println();

}

}

}

按照给定的格式打印[11, 33, 44, 22, 55]

//打印数组

**public** **static** **void** printArray(**int**[] arr) {

System.***out***.print("[");

**for** (**int** i = 0; i < arr.length; i++) {

**if** (i == arr.length - 1) {

System.***out***.println(arr[i]+"]");

} **else** {

System.***out***.print(arr[i]+", ");

}

}

}

//数组元素逆序

**public** **static** **void** receive(**int**[] arr){

**for** (**int** start = 0, end = arr.length-1; start < end; start++,end--) {

**int** temp = arr[start];

arr[start] = arr[end];

arr[end] = temp;

}

}

//选择排序

**public** **static** **void** selectSort(**int**[] arr) {

//功能

//外层循环用来控制数组循环的圈数

**for** (**int** i = 0; i < arr.length-1; i++) {

//内层循环用来完成元素值比较，把小的元素值互换到要比较的第一个元素中

**for** (**int** j = i+1; j < arr.length; j++) {

**if** (arr[i] > arr[j]) {

**int** temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

//冒泡排序

**public** **static** **void** bubbleSort(**int**[] arr) {

//功能

//外层循环用来控制数组循环的圈数

**for** (**int** i = 0; i < arr.length-1; i++) {

//j < arr.length-1 为了避免角标越界

//j < arr.length-1-i 为了比较效率,避免重复比较

//内层循环用来完成元素值比较，把大的元素值互换到后面

**for** (**int** j = 0; j < arr.length-1-i; j++) {

**if** (arr[j] > arr[j+1]) {

**int** temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

}

}

//普通查找

**public** **static** **int** getArrayIndex(**int**[] arr, **int** number) {

//把数组中的元素依次与指定的数值 进行比较

**for** (**int** i = 0; i < arr.length; i++) {

**if** (arr[i] == number) {

//找到了

return i;

}

}

**return** -1;

}

//二分查找法(折半查找法)

**public** **static** **int** halfSearch(**int**[] arr, **int** number) {

//定义3个变量，用来记录min, min, mid的位置

**int** min = 0;

**int** max = arr.length-1;

**int** mid = 0;

**while** (min <= max) {

mid = (min+max)/2;

//没找了， 更新范围，继续比较

//更新范围

**if** (arr[mid] > number) {

//在左边

max = mid-1;

} **else** **if**(arr[i] < number){

//在右边

min = mid+1;

}

**else**{

**return** mid ;

}

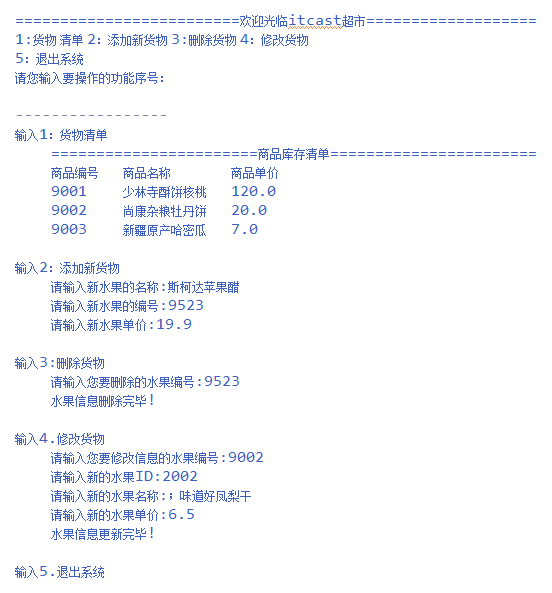
**return** -1;

}

# 超市库存管理系统

## 案例介绍

模拟真实的库存管理逻辑，完成超市管理系统的日常功能实现，见下图



## 案例需求分析

根据案例介绍，我们进行分析，首先需要一个功能菜单，然后输入功能序号后，调用序号对应的功能方法，实现想要的操作。分析步骤如下

* 1.完成超市商品初始化。创建商品，将商品添加到集合
* 2.显示来到超市能做的操作，也就是显示主菜单
* 3.根据接收到的功能选项，执行对应的功能
* 3.1.库存货物查询
* 3.2.添加新货物
* 3.3.删除货物
* 3.4.修改货物
* 3.5.退出系统,结束main方法的运行
* 4.循环，回到 2.显示主菜单

## 实现代码步骤

每种库存商品都拥有多项商品信息，为了方便管理每种商品的信息，我们对商品信息进行封装，编写FruitItem.java文件

**public** **class** FruitItem {

**int** ID; //商品编号

String name; //商品名称

**double** price; //商品单价

**double** number; //商品数量

**double** money; //商品金额

}

上述代码中，对商品信息（编号、名称、单价、数量、金额）进行了封装。这样做的好处在于以后只要找到这个商品，就能够知道该商品的每项信息了。

编写FruitStore.java，完成如下功能：

* 编写main主方法

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

//定义数组,记录每个商品信息

ArrayList<FruitItem> list = **new** ArrayList<FruitItem>();

/\*

\* 1.超市商品初始化

\*/

*init*(list);

while(true){

/\*

\* 2.显示主菜单

\*/

mainMenu();

/\*

\* 3.根据接收到的功能选项，执行对应的功能

\*/

chooseFunction(list);

}

}

上述代码中，通过while(true)循环，完成用户调用不同功能。实现重复功能选择操作。

* 1.超市商品初始化。创建商品，将商品添加到集合

**public** **static** **void** init(ArrayList<FruitItem> list) {

FruitItem item = **new** FruitItem();

item.name = "少林寺酥饼核桃";

item.ID = 9001;

item.price = 120;

FruitItem item2 = **new** FruitItem();

item2.name = "尚康杂粮牡丹饼";

item2.ID = 9002;

item2.price = 20;

FruitItem item3 = **new** FruitItem();

item3.name = "新疆原产哈密瓜";

item3.ID = 3;

item3.price = 9007;

list.add(item);

list.add(item2);

list.add(item3);

}

上述代码中，完成商品集合的初始化操作，添加了3件商品。

* 2.显示来到超市能做的操作，也就是显示主菜单

**public** **static** **void** mainMenu() {

System.*out*.println("=========================欢迎光临itcast超市=========================");

System.*out*.println("1:查询货物 2：添加新货物 3:删除货物 4：修改货物");

System.*out*.println("5：退出系统");

}

上述代码，用来完成用户操作界面的显示。

* 3.根据接收到的功能选项，执行对应的功能

**public** **static** **void** chooseFunction(ArrayList<FruitItem> list) {

System.*out*.println("请您输入要操作的功能序号：");

Scanner sc = **new** Scanner(System.*in*);//键盘输入对象

String choose = sc.next();

//根据接收到的选择，执行对应的功能

**switch** (choose) {

**case** "1": //3.1.库存货物查询

showFruitList(list);

break;

**case** "2": //3.2.添加新货物

addFruitItem(list);

break;

**case** "3": //3.3.删除货物

delFruitItem(list);

break;

**case** "4": //3.4.修改货物

updateFruitItem(list);

break;

**case** "5": //3.5.退出系统,退出JVM

System.*out*.println("退出系统");

return;

default:

System.*out*.println("对不起，没有您输入的功能，请重新选择");

break;

}

}

上述代码中，参数list为库存商品集合。当键盘输入功能序号后，通过switch语句匹配后，调用对应的方法完成对应功能的实现。

* 3.1.库存货物查询

**public** **static** **void** showFruitList(ArrayList<FruitItem> list) {

System.*out*.println("=======================商品库存清单=======================");

System.*out*.println("商品编号\t商品名称\t\t商品单价");

//查询每种库存商品信息

**for** (**int** i = 0; i < list.size(); i++) {

FruitItem item = list.get(i);

System.*out*.println(item.ID + "\t" + item.name + "\t" + item.price);

}

}

上述代码中，用来查询所有库存商品信息。

* 3.2.添加新货物

**public** **static** **void** addFruitItem(ArrayList<FruitItem> list) {

//创建新获取对象

FruitItem newItem = **new** FruitItem();

Scanner sc = **new** Scanner(System.*in*);//键盘输入对象

//提示输入信息

System.*out*.print("请输入新水果的名称:");

newItem.name = sc.next();

System.*out*.print("请输入新水果的编号:");

newItem.ID = sc.nextInt();

System.*out*.print("请输入新水果单价:");

newItem.price = sc.nextDouble();

//向货物集合中添加新的物品项

list.add(newItem);

}

上述代码中，通过键盘录入，将录入的商品信息存储到库存商品集合中。

* 3.3.删除货物

**public** **static** **void** delFruitItem(ArrayList<FruitItem> list) {

System.*out*.print("请输入您要删除的水果编号:");

Scanner sc = **new** Scanner(System.*in*);//键盘输入对象

**int** fruitID = sc.nextInt();

//删除集合元素

**for** (**int** i = 0; i < list.size(); i++) {

FruitItem thisItem = list.get(i);

**if**(thisItem.ID == fruitID) {

list.remove(thisItem);

System.*out*.println("水果信息删除完毕!");

return;

}

}

System.*out*.println("对不起，没有这个编号的水果!");

}

上述代码中，通过键盘录入，在库存集合中查找对应的商品编号，如果找到该商品，将该商品删除，否则，提示没有这个编号的商品。

* 3.4.修改货物

**public** **static** **void** updateFruitItem(ArrayList<FruitItem> list) {

System.*out*.println();

System.*out*.print("请输入您要修改信息的水果编号:");

Scanner sc = **new** Scanner(System.*in*);//键盘输入对象

**int** fruitID = sc.nextInt();

//更新集合元素

**for** (**int** i = 0; i < list.size(); i++) {

FruitItem thisItem = list.get(i);

**if**(thisItem.ID == fruitID) {

System.*out*.print("请输入新的水果ID:");

thisItem.ID = sc.nextInt();

System.*out*.print("请输入新的水果名称:");

thisItem.name = sc.next();

System.*out*.print("请输入新的水果单价:");

thisItem.price = sc.nextDouble();

System.*out*.println("水果信息更新完毕!");

return;

}

}

System.*out*.println("对不起，没有这个编号的水果!");

}

上述代码中，通过键盘录入，在库存集合中查找对应的商品编号，如果找到该商品，则键盘录入修改商品信息，否则，提示没有这个编号的商品。

## 什么是面向过程、面向对象

面向过程与面向对象都是我们编程中，编写程序的一种思维方式。

* 面向过程的程序设计方式，是遇到一件事时，思考“我该怎么做”，然后一步步实现的过程。

例如：公司打扫卫生（擦玻璃、扫地、拖地、倒垃圾等），按照面向过程的程序设计方式会思考“打扫卫生我该怎么做，然后一件件的完成”，最后把公司卫生打扫干净了。

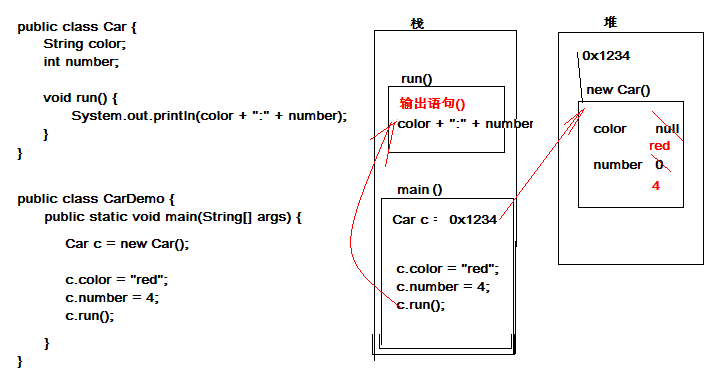
* 面向对象的程序设计方式，是遇到一件事时，思考“我该让谁来做”，然后那个“谁”就是对象，他要怎么做这件事是他自己的事，反正最后一群对象合力能把事就好就行了。

例如，公司打扫卫生（擦玻璃、扫地、拖地、倒垃圾等），按照面向对象的程序设计方式会思考“我该让谁来做，如小明擦玻璃、让小丽扫地、让小郭拖地、让小强倒垃圾等”,这里的“小明、小丽、小郭、小强”就是对象，他们要打扫卫生，怎么打扫是他们自己的事，反正最后一群对象合力把公司卫生打扫干净了。

## 对象的内存图解

经过上面对小汽车的描述，和Java代码测试，我们虽然可以将生活中的事物使用Java代码描述出来，但是这些代码在内存中是如何执行的，接下来我们需要研究下对象在内存的图解。

接下来就是分析对象在内存中的分配情况。这里需要画图一步一步演示，严格按照画图流程讲解内存对象创建过程。



## 局部变量和成员变量区别

理解清楚了类和对象之后，结合前5天的学习知识，发现在描述类的属性和前面学习定义变量差别不大，唯一区别就是位置发生了改变，那么类中定义的变量，和在方法定义的变量有啥差别呢？

回忆以前学习时变量的定义方式，和位置，以及现在定义类中属性的特点。总结下面几点异同

区别一：定义的位置不同

定义在类中的变量是成员变量

定义在方法中或者{}语句里面的变量是局部变量

区别二：在内存中的位置不同

成员变量存储在堆内存的对象中

局部变量存储在栈内存的方法中

区别三：声明周期不同

成员变量随着对象的出现而出现在堆中，随着对象的消失而从堆中消失

局部变量随着方法的运行而出现在栈中，随着方法的弹栈而消失

区别四：初始化不同

成员变量因为在堆内存中，所有默认的初始化值

局部变量没有默认的初始化值，必须手动的给其赋值才可以使用。

当在方法中出现了局部变量和成员变量同名的时候，那么在方法中怎么区别局部变量成员变量呢？可以在成员变量名前面加上this.来区别成员变量和局部变量

* + this到底代表什么呢？this代表的是对象，具体代表哪个对象呢？哪个对象调用了this所在的方法，this就代表哪个对象。
  + 上述代码中的 p.setAge(30)语句中，setAge(int age)方法中的this代表的就是p对象。

封装，它也是面向对象思想的特征之一。面向对象共有三个特征：封装，继承，多态。

点名器案例 get set重构

每名学生都拥有多项个人信息，为了方便管理每个人的信息，我们对学生信息进行封装，编写Student.java文件

/\*\*

\* 学生信息类

\*/

**public** **class** Student {

private String name; // 姓名

private **int** age; // 年龄

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

}

上述代码中，对学生信息（姓名、年龄）进行了封装。这样做的好处在于，以后只要找到这名学生，就能够知道他的每项个人信息了。

接下来我们编写CallName.java文件，完成程序的编写。

* main方法中调用三个独立方法

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

ArrayList<Student> list = **new** ArrayList<Student>(); //1.1创建一个可以存储多个同学名字的容器

/\*

\* 1.存储全班同学信息

\*/

addStudent(list);

/\*

\* 2.打印全班同学每一个人的信息（姓名、年龄）

\*/

printStudent(list);

/\*

\* 3.随机对学生点名，打印学生信息

\*/

randomStudent(list);

}

* 存储所有学生的个人信息

/\*\*

\* 1.存储全班同学名字

\*/

**public** **static** **void** addStudent(ArrayList<Student> list) {

//键盘输入多个同学名字存储到容器中

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

**for** (**int** i = 0; i < 3; i++) {

//创建学生

Student s = **new** Student();

System.***out***.println("存储第"+i+"个学生姓名：");

String name = sc.next();

s.setName(name);

System.***out***.println("存储第"+i+"个学生年龄：");

**int** age = sc.nextInt();

s.setAge(age);

//添加学生到集合

list.add(s);

}

}

上述方法中，方法参数list中用来表示已存储所有学生。通过Scanner，完成新学生信息（姓名，年龄）的录入，并将学生添加到集合中。

* 打印全班同学每一个人的信息

/\*\*

\* 2.打印全班同学每一个人的信息（姓名、年龄）

\*/

**public** **static** **void** printStudent (ArrayList<Student> list) {

**for** (**int** i = 0; i < list.size(); i++) {

Student s = list.get(i);

System.***out***.println("姓名："+s.getName() +",年龄："+s.getAge());

}

}

上述方法中，方法参数list中用来表示已存储所有学生。通过遍历集合中的每个元素，得到每个同学信息，并输出打印。

* 随机对学生点名，打印学生信息

/\*\*

\* 3.随机对学生点名，打印学生信息

\*/

**public** **static** **void** randomStudent (ArrayList<Student> list) {

//在班级总人数范围内，随机产生一个随机数

**int** index = **new** Random().nextInt(list.size());

//在容器（ArrayList集合）中，查找该随机数所对应的同学信息（姓名、年龄）

Student s = list.get(index);

System.***out***.println("被随机点名的同学："+s.getName() + "，年龄:" + s.getAge());

}

上述方法中，通过随机数类Random产生一个从0到集合长度的随机索引。使用该索引获取ArrayList集合中对应的值，便得到了全班同学的随机学生信息并打印。

* 类只支持单继承，不允许多继承，也就是说一个类只能有一个直接父类，例如下面这种情况是不合法的。

class A{}

class B{}

class C extends A,B{} // C类不可以同时继承A类和B类

* 多个类可以继承一个父类，例如下面这种情况是允许的。

class A{}

class B extends A{}

class C extends A{} // 类B和类C都可以继承类A

* 多层继承是可以的，即一个类的父类可以再去继承另外的父类，例如C类继承自B类，而B类又可以去继承A类，这时，C类也可称作A类的子类。下面这种情况是允许的。

class A{}

class B extends A{} // 类B继承类A，类B是类A的子类

class C extends B{} // 类C继承类B，类C是类B的子类，同时也是类A的子类

在子类中，访问父类中的成员变量格式：

super.父类中的成员变量

看如下代码：

class Fu

{

//Fu中的成员变量。

int num = 5;

}

class Zi extends Fu

{

//Zi中的成员变量

int num = 6;

void show()

{

//子父类中出现了同名的成员变量时

//在子类中需要访问父类中非私有成员变量时，需要使用super关键字

//访问父类中的num

System.out.println("Fu num="+super.num);

//访问子类中的num2

System.out.println("Zi num2="+this.num);

}

}

class Demo5

{

public static void main(String[] args)

{

Zi z = new Zi(); //创建子类对象

z.show(); //调用子类中的show方法

}

}

* 方法重写（覆盖）的应用：

当子类需要父类的功能，而功能主体子类有自己特有内容时，可以重写父类中的方法，这样，即沿袭了父类的功能，又定义了子类特有的内容。

举例：比如手机，当描述一个手机时，它具有发短信，打电话，显示来电号码功能，后期由于手机需要在来电显示功能中增加显示姓名和头像，这时可以重新定义一个类描述智能手机，并继承原有描述手机的类。并在新定义的类中覆盖来电显示功能，在其中增加显示姓名和头像功能。

//手机类

class Phone{

public void sendMessage(){

System.out.println("发短信");

}

public void call(){

System.out.println("打电话");

}

public void showNum(){

System.out.println("来电显示号码");

}

}

//智能手机类

class NewPhone extends Phone{

//覆盖父类的来电显示号码功能，并增加自己的显示姓名和图片功能

public void showNum(){

//调用父类已经存在的功能使用super

super.showNum();

//增加自己特有显示姓名和图片功能

System.out.println("显示来电姓名");

System.out.println("显示头像");

}

接口是功能的集合，同样可看做是一种数据类型，是比抽象类更为抽象的”类”。

接口只描述所应该具备的方法，并没有具体实现，具体的实现由接口的实现类(相当于接口的子类)来完成。这样将功能的定义与实现分离，优化了程序设计。

请记住：一切事物均有功能，即一切事物均有接口。

与定义类的class不同，接口定义时需要使用interface关键字。

定义接口所在的仍为.java文件，虽然声明时使用的为interface关键字的编译后仍然会产生.class文件。这点可以让我们将接口看做是一种只包含了功能声明的特殊类。

定义格式：

public interface 接口名 {

抽象方法1;

抽象方法2;

抽象方法3;

}

使用interface代替了原来的class，其他步骤与定义类相同：

* 接口中的方法均为公共访问的抽象方法
* 接口中无法定义普通的成员变量

类与接口的关系为实现关系，即类实现接口。实现的动作类似继承，只是关键字不同，实现使用implements。实现类仍然需要重写方法以实现具体的功能。

格式：

class 类 implements 接口 {

重写接口中方法

}

* 接口中定义功能，当需要具有该功能时，可以让类实现该接口，只声明了应该具备该方法，是功能的声明。
* 在具体实现类中重写方法，实现功能，是方法的具体实现。

于是，通过以上两个动作将功能的声明与实现便分开了。(此时请重新思考：类是现实事物的描述，接口是功能的集合。)

接口中可以定义变量，但是变量必须有固定的修饰符修饰，public static final 所以接口中的变量也称之为常量，其值不能改变。

2、接口中可以定义方法，方法也有固定的修饰符，public abstract

3、接口不可以创建对象。

4、子类必须覆盖掉接口中所有的抽象方法后，子类才可以实例化。否则子类是一个抽象类。

interface Demo { ///定义一个名称为Demo的接口。

public static final int NUM = 3;// NUM的值不能改变

public abstract void show1();

public abstract void show2();

}

//定义子类去覆盖接口中的方法。类与接口之间的关系是 实现。通过 关键字 implements

class DemoImpl implements Demo { //子类实现Demo接口。

//重写接口中的方法。

public void show1(){}

public void show2(){}

}

为什么要定义接口，使用抽象类描述也没有问题，接口到底有啥用呢？

接口最重要的体现：解决多继承的弊端。将多继承这种机制在java中通过多实现完成了。

interface Fu1

{

void show1();

}

interface Fu2

{

void show2();

}

class Zi implements Fu1,Fu2// 多实现。同时实现多个接口。

{

public void show1(){}

public void show2(){}

}

多继承时，当多个父类中有相同功能时，子类调用会产生不确定性。

其实核心原因就是在于多继承父类中功能有主体，而导致调用运行时，不确定运行哪个主体内容。

为什么多实现能解决了呢？

因为接口中的功能都没有方法体，由子类来明确。

子类通过继承父类扩展功能，通过继承扩展的功能都是子类应该具备的基础功能。如果子类想要继续扩展其他类中的功能呢？这时通过实现接口来完成。

class Fu {

public void show(){}

}

interface Inter {

pulbic abstract void show1();

}

class Zi extends Fu implements Inter {

public void show1() {

}

}

接口的出现避免了单继承的局限性。父类中定义的事物的基本功能。接口中定义的事物的扩展功能。

***父类 接口 含义解释案例：***

犬分为很多种类，他们吼叫和吃饭的方式不一样，在描述的时候不能具体化，也就是吼叫和吃饭的行为不能明确。当描述行为时，行为的具体动作不能明确，这时，可以将这个行为写为抽象行为，那么这个类也就是抽象类。

可是当缉毒犬有其他额外功能时，而这个功能并不在这个事物的体系中。这时可以让缉毒犬具备犬科自身特点的同时也有其他额外功能，可以将这个额外功能定义接口中。

如下代码演示：

interface 缉毒{

public abstract void 缉毒();

}

//定义犬科的这个提醒的共性功能

abstract class 犬科{

public abstract void 吃饭();

public abstract void 吼叫();

}

// 缉毒犬属于犬科一种，让其继承犬科，获取的犬科的特性，

//由于缉毒犬具有缉毒功能，那么它只要实现缉毒接口即可，这样即保证缉毒犬具备犬科的特性，也拥有了缉毒的功能

class 缉毒犬 extends 犬科 implements 缉毒{

public void 缉毒() {

}

void 吃饭() {

}

void 吼叫() {

}

}

class 缉毒猪 implements 缉毒{

public void 缉毒() {

}

}

***接口和抽象类的区别：***

***相同点:***

* ***都位于继承的顶端,用于被其他类实现或继承;***
* ***都不能直接实例化对象;***
* ***都包含抽象方法,其子类都必须覆写这些抽象方法;***

***区别:***

* ***抽象类为部分方法提供实现,避免子类重复实现这些方法,提高代码重用性;接口只能包含抽象方法;***
* ***一个类只能继承一个直接父类(可能是抽象类),却可以实现多个接口;(接口弥补了Java的单继承)***
* ***抽象类是这个事物中应该具备的你内容, 继承体系是一种 is..a关系***
* ***接口是这个事物中的额外内容,继承体系是一种 like..a关系***

***二者的选用:***

* ***优先选用接口,尽量少用抽象类;***
* ***需要定义子类的行为,又要为子类提供共性功能时才选用抽象类;***

多态是继封装、继承之后，面向对象的第三大特性。

现实事物经常会体现出多种形态，如学生，学生是人的一种，则一个具体的同学张三既是学生也是人，即出现两种形态。

Java作为面向对象的语言，同样可以描述一个事物的多种形态。如Student类继承了Person类，一个Student的对象便既是Student，又是Person。

Java中多态的代码体现在一个子类对象(实现类对象)既可以给这个子类(实现类对象)引用变量赋值，又可以给这个子类(实现类对象)的父类(接口)变量赋值。

如Student类可以为Person类的子类。那么一个Student对象既可以赋值给一个Student类型的引用，也可以赋值给一个Person类型的引用。

最终多态体现为父类引用变量可以指向子类对象。

多态的前提是必须有子父类关系或者类实现接口关系，否则无法完成多态。

在使用多态后的父类引用变量调用方法时，会调用子类重写后的方法。

* 多态成员变量

当子父类中出现同名的成员变量时，多态调用该变量时：

简单记：编译和运行都参考等号的左边。编译运行看左边。

* 多态成员方法

简而言之：编译看左边，运行看右边。

## instanceof关键字

我们可以通过instanceof关键字来判断某个对象是否属于某种数据类型。如学生的对象属于学生类，学生的对象也属于人类。

使用格式：

boolean b = 对象 instanceof 数据类型;

如

Person p1 = new Student(); // 前提条件，学生类已经继承了人类

boolean flag = p1 instanceof Student; //flag结果为true

boolean flag2 = p2 instanceof Teacher; //flag结果为false

多态弊端：

//描述动物类，并抽取共性eat方法

**abstract** **class** Animal {

**abstract** **void** eat();

}

// 描述狗类，继承动物类，重写eat方法，增加lookHome方法

**class** Dog **extends** Animal {

**void** eat() {

System.***out***.println("啃骨头");

}

**void** lookHome() {

System.***out***.println("看家");

}

}

// 描述猫类，继承动物类，重写eat方法，增加catchMouse方法

**class** Cat **extends** Animal {

**void** eat() {

System.***out***.println("吃鱼");

}

**void** catchMouse() {

System.***out***.println("抓老鼠");

}

}

**public** **class** Test {

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

Animal a = **new** Dog(); //多态形式，创建一个狗对象

a.eat(); // 调用对象中的方法，会执行狗类中的eat方法

// a.lookHome();//使用Dog类特有的方法，需要向下转型，不能直接使用

// 为了使用狗类的lookHome方法，需要向下转型

// 向下转型过程中，可能会发生类型转换的错误，即ClassCastException异常

// 那么，在转之前需要做健壮性判断

if( !a instanceof Dog){ // 判断当前对象是否是Dog类型

System.out.println("类型不匹配，不能转换");

return;

}

Dog d = (Dog) a; //向下转型

d.lookHome();//调用狗类的lookHome方法

}

}

public class Test {

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

// 多态形式

毕姥爷 a = **new** 毕老师(); // 向上转型

a.讲课(); // 这里表象是毕姥爷，其实真正讲课的仍然是毕老师，因此调用的也是毕老师的讲课功能

a.钓鱼(); // 这里表象是毕姥爷，但对象其实是毕老师，而毕老师继承了毕姥爷，即毕老师也具有钓鱼功能

// 当要调用毕老师特有的看电影功能时，就必须进行类型转换

毕老师 b = (毕老师) a; // 向下转型

b.看电影();

}

# 笔记本电脑经典案例

## 案例介绍

定义USB接口（具备开启功能、关闭功能），笔记本要使用USB设备，即笔记本在生产时需要预留可以插入USB设备的USB接口，即就是笔记本具备使用USB设备的功能，但具体是什么USB设备，笔记本并不关心，只要符合USB规格的设备都可以。鼠标和键盘要想能在电脑上使用，那么鼠标和键盘也必须遵守USB规范，不然鼠标和键盘的生产出来无法使用

进行描述笔记本类，实现笔记本使用USB鼠标、USB键盘

* USB接口，包含开启功能、关闭功能
* 笔记本类，包含运行功能、关机功能、使用USB设备功能
* 鼠标类，要符合USB接口
* 键盘类，要符合USB接口

## 案例需求分析

阶段一：

使用笔记本，笔记本有运行功能，需要笔记本对象来运行这个功能

阶段二：

想使用一个鼠标，又有一个功能使用鼠标，并多了一个鼠标对象。

阶段三：

还想使用一个键盘 ，又要多一个功能和一个对象

问题：每多一个功能就需要在笔记本对象中定义一个方法，不爽，程序扩展性极差。

降低鼠标、键盘等外围设备和笔记本电脑的耦合性。

## 实现代码步骤

* 定义鼠标、键盘，笔记本三者之间应该遵守的规则

interface USB {

void open();// 开启功能

void close();// 关闭功能

}

* 鼠标实现USB规则

class Mouse implements USB {

public void open() {

System.*out*.println("鼠标开启");

}

public void close() {

System.*out*.println("鼠标关闭");

}

}

* 键盘实现USB规则

class KeyBoard implements USB {

public void open() {

System.*out*.println("键盘开启");

}

public void close() {

System.*out*.println("键盘关闭");

}

}

* 定义笔记本

class NoteBook {

// 笔记本开启运行功能

public void run() {

System.*out*.println("笔记本运行");

}

// 笔记本使用usb设备，这时当笔记本对象调用这个功能时，必须给其传递一个符合USB规则的USB设备

public void useUSB(USB usb) {

// 判断是否有USB设备

if (usb != null) {

usb.open();

usb.close();

}

}

public void shutDown() {

System.*out*.println("笔记本关闭");

}

}

public class Test {

public static void main(String[] args) {

// 创建笔记本实体对象

NoteBook nb = new NoteBook();

// 笔记本开启

nb.run();

// 创建鼠标实体对象

Mouse m = new Mouse();

// 笔记本使用鼠标

nb.useUSB(m);

// 创建键盘实体对象

KeyBoard kb = new KeyBoard();

// 笔记本使用键盘

nb.useUSB(kb);

// 笔记本关闭

nb.shutDown();

}

}

## final的概念

继承的出现提高了代码的复用性，并方便开发。但随之也有问题，有些类在描述完之后，不想被继承，或者有些类中的部分方法功能是固定的，不想让子类重写。可是当子类继承了这些特殊类之后，就可以对其中的方法进行重写，那怎么解决呢？

要解决上述的这些问题，需要使用到一个关键字final，final的意思为最终，不可变。final是个修饰符，它可以用来修饰类，类的成员，以及局部变量。

 main方法为静态方法仅仅为程序执行入口，它不属于任何一个对象，可以定义在任意类中。

## 定义静态常量

开发中，我们想在类中定义一个静态常量，通常使用public static final修饰的变量来完成定义。此时变量名用全部大写，多个单词使用下划线连接。

定义格式：

public static final 数据类型 变量名 = 值;

如下演示：

**class** Company {

**public** **static** **final** String ***COMPANY\_NAME*** = "传智播客";

**public** **static** **void** method(){

System.***out***.println("一个静态方法");

}

}

接口中的每个成员变量都默认使用public static final修饰。

所有接口中的成员变量已是静态常量，由于接口没有构造方法，所以必须显示赋值。可以直接用接口名访问。

interface Inter {

public static final int *COUNT* = 100;

}

访问接口中的静态变量

Inter.COUNT

# 星级酒店案例

## 案例介绍

某五星级酒店，资金雄厚，要招聘多名员工（经理、厨师、服务员）。入职的员工需要记录个人信息（姓名、工号、经理特有奖金属性）。他们都有自己的工作要做。

本案例要完成如下需求：

* 获取酒店幸运员工；
* 酒店开设VIP服务，酒店的厨师与服务员可以提供VIP服务。（厨师做菜加量、服务员给顾客倒酒）。
* 编写测试类
  + 向酒店中，增加多名员工（其中包含1名经理，1名厨师、2名服务员）；
  + 调用酒店员工的工作功能
  + 调用酒店员工的VIP服务功能

## 案例需求分析

* 根据“某五星级酒店，资金雄厚……都有自己的工作要做。”分析出，该题目中包含酒店，可以把它封装成类，多名员工）。

class 员工 {

属性：姓名

属性：工号

方法：工作

}

class 厨师 extends 员工{}

class 服务员 extends 员工{}

class 经理 extends 员工 {

属性：奖金

}

员工的类型有经理、厨师、服务员，它们有共同的属性（姓名、工号、），经理额外属性（奖金）。

* 根据“向酒店中，增加多名员工（其中包含1名经理，1名厨师、2名服务员）”。分析出，要创建一个酒店对象，并添加4名员工到酒店对象的员工集合中。

酒店员工集合添加新员工： 经理对象

酒店员工集合添加新员工： 厨师对象

酒店员工集合添加新员工： 服务员对象

酒店员工集合添加新员工： 服务员对象

* 根据“获取酒店幸运员工”。分析出，从酒店员工集合随机得到一名员工对象。

1. 从酒店员工集合长度范围内，随机产生一个随机数

2. 使用该随机数作为集合的索引，返回该索引处对应的员工对象

* 根据“酒店开设VIP服务，酒店的厨师与服务员可以提供VIP服务。（厨师做菜加量、服务员给顾客倒酒）”。分析出，这是要增加一个VIP的接口，接口中提供个VIP服务的方法。让厨师与服务员实现该接口。

interface VIP服务{

抽象方法：服务

}

class 厨师 extends 员工 implements VIP服务{ 重写服务方法 }

class 服务员 extends 员工 implements VIP服务{ 重写服务方法 }

## 实现代码步骤

* VIP服务

public interface VIP {

public abstract void server(); //服务

}

* 员工

/\*

\* 员工：

姓名 String

工号 String

\*/

**public** **abstract** **class** YuanGong {

// 成员变量

**private** String xingMing;

**private** String gongHao;

// 构造方法

**public** YuanGong() {

**super**();

}

**public** YuanGong(String xingMing, String gongHao) {

**super**();

**this**.xingMing = xingMing;

**this**.gongHao = gongHao;

}

// 抽象方法

**public** **abstract** **void** work();

// getters与setters

**public** String getXingMing() {

**return** xingMing;

}

**public** **void** setXingMing(String xingMing) {

**this**.xingMing = xingMing;

}

**public** String getGongHao() {

**return** gongHao;

}

**public** **void** setGongHao(String gongHao) {

**this**.gongHao = gongHao;

}

}

* 服务员

/\*

\* 定义员工的子类 服务员类

\*/

**public** **class** FuWuYuan **extends** YuanGong **implements** VIP {

**public** FuWuYuan() {

**super**();

}

**public** FuWuYuan(String xingMing, String gongHao) {

**super**(xingMing, gongHao);

}

@Override

**public** **void** work() {

System.***out***.println("亲，全身心为您服务，记得给好评哦");

}

@Override

**public** **void** server() {

System.***out***.println("给顾客倒酒");

}

}

* 经理

/\*

\* 经理在员工的基础上，添加了奖金成员

\*/

**public** **class** JingLi **extends** YuanGong {

**private** **double** jiangJin;

**public** JingLi() {

**super**();

}

**public** JingLi(String xingMing, String gongHao, **double** jiangJin) {

**super**(xingMing, gongHao);

**this**.jiangJin = jiangJin;

}

**public** **double** getJiangJin() {

**return** jiangJin;

}

**public** **void** setJiangJin(**double** jiangJin) {

**this**.jiangJin = jiangJin;

}

@Override

**public** **void** work() {

System.***out***.println("哪个员工让顾客不满意，我扣谁钱");

};

}

* 厨师

/\*

\* 定义员工的子类 厨师类

\*/

**public** **class** ChuShi **extends** YuanGong **implements** VIP{

**public** ChuShi() {

**super**();

}

**public** ChuShi(String xingMing, String gongHao) {

**super**(xingMing, gongHao);

}

@Override

**public** **void** work() {

System.***out***.println("我做饭，放心吃吧，包您满意");

}

@Override

**public** **void** server() {

System.***out***.println("做菜加量加料");

}

}

* 测试类

**public** **class** Test {

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

}

}

Java 的API（API: **A**pplication(应用) **P**rogramming(程序) **I**nterface(接口)）

Java API就是JDK中提供给我们使用的类，这些类将底层的代码实现封装了起来，我们不需要关心这些类是如何实现的，只需要学习这些类如何使用即可。

在JDK安装目录下有个src.zip文件，这个文件解压缩后里面的内容是所有Java类的源文件。可以在其中查看相对应的类的源码。

我们在每次查看类中的方法时，都打开源代码进行查看，这种方式过于麻烦。其实，我们可以通过查帮助文档的方式，来了解Java提供的API如何使用。