**Table of Content**

Contents

[1.1 Java输出 2](#_Toc527040250)

[1.2 Java标识符 3](#_Toc527040251)

[1.2.1 标识符 3](#_Toc527040252)

[1.3 Java打印 4](#_Toc527040253)

[1.3.1 打印 4](#_Toc527040254)

[1.4 Java数据类型和转换 5](#_Toc527040255)

[1.4.1 8种基本数据类型 5](#_Toc527040256)

[1.4.2 强制类型转换 5](#_Toc527040257)

[1.4.3 隐含强制类型转换 6](#_Toc527040258)

[1.4.4 算术运算符 6](#_Toc527040259)

[1.5 随机数Random和键盘输入Scanner 8](#_Toc527040260)

[1.5.1 Random 8](#_Toc527040261)

[1.5.2 Scanner 8](#_Toc527040262)

[1.6 Break和Continue 9](#_Toc527040263)

[1.6.1 Break 9](#_Toc527040264)

[1.6.2 Continue 10](#_Toc527040265)

[1.6.3 死循环 10](#_Toc527040266)

[1.6.4 while 10](#_Toc527040267)

[1.6.5 Do while 11](#_Toc527040268)

[1.6.6 For 11](#_Toc527040269)

[1.6.7 TestIfElseIf 12](#_Toc527040270)

[1.7 面向对象 14](#_Toc527040271)

[1.7.1 Break 14](#_Toc527040272)

[1.8 数组 15](#_Toc527040273)

[1.8.1 Break 15](#_Toc527040274)

## Java输出

### 经典的HelloWorld

实现了一个Java的HelloWorld程序  
实现步骤：  
 1. 定义类  
 2. 定义主方法  
 3. 一条命令，控制台输出了HelloWorld

**public class** TestDemo {  
 *//main主方法，固定格式，程序的入口点* **public static void** main(String[] args) {  
 *//系统 输出 打印 打印的内容* System.***out***.println(**"HelloWorld"**);  
 }  
}

## Java标识符和方法名

### 标识符

*标识符  
 Java中，自己定义的内容  
 自定义类的名字，上一个案例 HelloWorld  
 标识符的规则：  
 组成： 字母52个A-Z a-z 数字0-9 \_ 下划线 $ 美元符  
 注意： 不能数字开头，不能是关键字  
  
 定义名字：  
\_abc 0a a0 a#a a$a void  
YES NO YES NO YES NO  
  
 类的名字： 首字母大写，第二个单词首字母大写  
BeiJingShiHaiDianQuYiYuan  
MeiGuoJiaLiFuNiYa  
  
 方法的名字：首字母小写，每个单词首字母大写  
 addStudent*

### 方法名

*方法的定义格式  
 修饰符 返回值类型 方法的名字 (参数列表...){  
 方法的功能主体  
 循环,判断,变量,比较,运算  
 return ;  
 }  
   
修饰符: 固定写法 public static  
返回值类型: 方法在运算后,结果的数据类型  
方法名: 自定义名字,满足标识符规范, 方法名字首字母小写,后面每个单词首字母大写  
参数列表: 方法的运算过程中,是否有未知的数据, 如果有未知的数据,定义在参数列表上 (定义变量)  
return: 方法的返回, 将计算的结果返回. 结束方法*

## Java打印

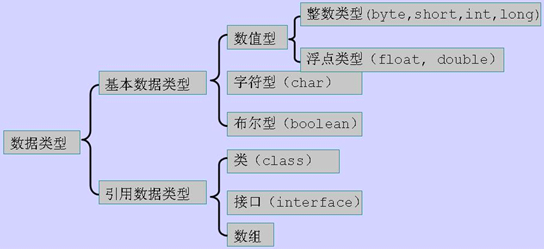
### 打印

System.***out***.println(15); *//15*System.***out***.println(0B1111); *//15---TestNG下不可以运行--Binary literals are not supported at language level 6* System.***out***.println(0xF); *//15* System.***out***.println(017); *//15  
 //输出布尔数据，只有2个值，true，false 关键字* System.***out***.println(**true**); *//true* System.***out***.println(**false**); *//false  
  
  
 //输出整数 十进制* System.***out***.println(50);*//50  
 //输出整数，二进制, 数字开头0B* System.***out***.println(0B11);*//3  
 //输出整数，八进制，数字开头0* System.***out***.println(051);*//41  
 //输出整数，十六进制，数组开头0X 0-9 A-F* System.***out***.println(0XE);*//14  
 //输出浮点数据* System.***out***.println(5.0);*//5.0  
  
 //输出字符常量，单引号包裹，只能写1个字符* System.***out***.println(**'a'**);  
 *//输出字符串常量，双引号包裹，可以写0-n个字符* System.***out***.println(**"HelloWorld"**);  
  
  
 System.***out***.println(3 < 5 ? 99 : 88); *//99  
 //s三元运算符，true则是冒号之前的，flase则是冒号之后的* String s = 0 == 1 ? **"哈哈"** : **"呵呵"**; *//呵呵* String s2 = 1 == 1 ? **"哈哈"** : **"呵呵"**; *//h哈哈* System.***out***.println(s);  
 System.***out***.println(s2);  
  
  
 **int** a = 5;  
 **int** b = 3;  
 **int** c = 1;  
 *// T && T* **int** n2 = (a > b && b > c) ? (c++) : (++c);  
 **int** n1 = (a > b && b < c) ? (c++) : (++c);  
 System.***out***.println(n2);  
 System.***out***.println(c);  
 System.***out***.println(c++);  
 System.***out***.println(n1);  
 System.***out***.println(n2);  
  
}

## Java数据类型和转换

### 8种基本数据类型

*数据类型的转换  
自动: 取值范围小的类型,自动转成取值范围大的类型  
一个类型 boolean 不参与类型转换*







*变量定义和使用的注意事项  
 1. 变量定义后,不赋值,不能使用  
 2. 变量是有自己的作用范围,变量的有效范围,定义的一对大括号内  
 3. 变量不允许重复定义*

### 强制类型转换

*数据类型的强制转换  
 强制: 数据类型取值范围大的,转成取值范围小的  
  
 数据类型的强制转换,公式  
 被转后的数据类型 变量名 = (被转换后数据类型)要被转的数据  
  
强制类型换,没有要求的时候,不做*

### 隐含强制类型转换

### 算术运算符

*/\*  
 算数运算符  
 + 加法,连接  
 - 减法  
 \* 乘法  
 / 除法  
 % 取模, 获取余数  
 ++ 自增1  
 -- 自减1  
\*/*

*// + 连接作用,将一个字符串和其他数据连接起来  
// + 遇到字符串,变成连接符号"5+5=55"  
// + 任何类型只要和字符串+, 所有的数据都变成了字符串*

*++ 写在变量前面,和写在变量后面,区别  
a = 1  
a++  
++a  
符号写在前面,还是后面,在变量自己独立计算的时候,没有区别  
  
但是变量,要是参与了其他运算,有区别了  
  
j=i++ ++后算,先将i的值,赋值j, i自己在++  
n=++m ++先算,变量m++ 6 变量赋值给n*

*赋值运算符  
=  
+= -= \*= /= %=*

*比较运算符,计算结果只有2个可能,true,false  
结算结果的数据类型, boolean类型  
== 比较符号两边的数据  
!= 比较符号两边的数据  
> 比较符号两边的数据 1<2 true 2<2 false  
< 比较符号两边的数据  
>= 比较符号两边的数据 1>=2 false 2>=2 true  
<= 比较符号两边的数据*

*逻辑运算符,对两个boolean类型数据之间进行计算  
结果也是boolean类型  
  
& : 一边是false,运算结果就是false, 见了false,就是false  
| : 一边是true,运算结果就是true,见了true,就是true  
^ : 两边相同为false,不同为true  
! : 取反 !true = false !false=true  
&& : 短路与 , 一边是false,另一边不运行  
|| : 短路或 , 一边是true,另一边不运行*

*三元运算符  
公式:  
 布尔表达式 ? 结果1:结果2;  
 布尔表达式结果是true,三元运算符的结果,就是 结果1  
 布尔表达式结果是false,三元运算符的结果,就是 结果2*

## 随机数Random和键盘输入Scanner

### Random

*//创建Random类变量*Random ran = **new** Random();  
*//变量.使用功能nextInt()获取1-100随机数***int** ranNumber = ran.nextInt(100)+1;

### Scanner

*//创建Scanner类变量*

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

*//获取键盘输入***int** number = sc.nextInt();

## Break和Continue

### Break

*单循环体中，break是结束整个循环，而continue是结束本次循环（跳过下一步）  
嵌套循环体中，break是结束嵌套的那一层的整个循环，而continue是结束本次循环（跳过下一步）*

**public class** TestBreak {  
  
 **public static void** main(String[] args) {  
 **for** (**int** i = 1; i < 5; i++) {  
 **if** (i == 3) **break**;  
 System.***out***.println(i);  
 }  
 *// 只输出 1 ， 2 ， 到3就跳出循环了* **for** (**int** i = 0; i < 2; i++) {  
 **for** (**int** j = 0; j < 5; j++) {  
 System.***out***.print(**"j="** + j);  
 **break**;  
 }  
 System.***out***.println(**"i="** + i);  
 }  
 }  
}

*循环的嵌套: 循环里面还有循环, for形式多  
 for(){  
  
 for(){  
  
}  
  
 }  
 总的循环次数 = 内循环次数 \* 外循环的次数  
 内循环,是外循环的循环体  
  
 外循环,控制的是行数  
 内循环,控制的是每行的个数*

*if语句:  
编写格式  
  
if(条件){  
 if语句的执行体  
}  
条件: 结果必须是布尔类型 1==1 true||false  
执行体: 当if中的条件为true的时候,执行大括号中的执行体  
 if条件是false的时候,什么也不做*

### Continue

**public class** TestContinue {  
  
 **public static void** main(String[] args) {  
 **for** (**int** i=1;i<=5;i++)  
 {  
 **if** (i==3) **continue**;  
 System.***out***.println(i);  
 }  
 *// 不输出3，因为continue跳过了，直接进入下一个迭代  
 // 只输出 1 ， 2 ， 4 ， 5* }  
}

### 死循环

**public class** TestDeadLoop {  
  
 **public static void** main(String[] args){  
 *//while形式死循环  
 /\*while(true){  
  
 }\*/  
 //for形式死循环* **for**( ; **true** ; ){  
  
 }  
 }  
}

### while

*循环语句: 可以让一部分代码,反复执行  
  
 while循环:  
 编写格式:  
  
 while(条件){  
 循环体  
 }  
 条件: 当条件是true,就执行循环体,执行完循环体后  
 程序再次执行while中的条件,如果条件还是true,继续执行循环体  
 直到条件是false的时候,循环就结束***public class** TestWhile{  
 **public static void** main(String[] args){  
 *//输出 1-4之间的整数  
 //定义变量,整数类型, 循环的条件* **int** i = 1;  
 **while**( i < 5 ){  
 System.***out***.println(i);  
 i++;  
 }  
 }  
}

### Do while

*/\*  
 do...while循环  
 编写格式:  
  
 do{  
 循环体  
 }while(条件);  
 特点: 无条件先执行一次  
\*/***public class** TestDoWhile{  
 **public static void** main(String[] args){  
 **int** i = 6;  
 **do**{  
 System.***out***.println(i);  
 i++;  
 }**while**( i < 5);  
 }  
}

### For

*for循环  
 编写格式  
 for(初始化变量 ; 条件 ; 增量){  
 循环体  
 }  
 初始化变量: 定义变量,作用,控制循环的次数  
 条件: 当条件是true,执行循环体,条件是false,结束循环  
 增量: 变量自增情况***public class** TestFor {  
 **public static void** main(String[] args) {  
 *//for循环,输出0-10* **for** (**int** i = 0; i < 11; i++) {  
 System.***out***.println(i);  
 }  
  
  
 **int** sum = 0;  
 *// 利用循环,将变量从1变化到4* **for** (**int** i = 1; i <= 4; i++) {  
 *//对变量进行求和* sum = sum + i;  
 }  
 System.***out***.println(sum);  
 }  
  
}

### TestIfElseIf

*if else if else 语句  
 适合在程序中,实现多条件的判断  
 编写格式:  
 if(条件){  
 if 执行体  
 }else if(条件){  
 if 执行体  
 }else if(条件){  
 if 执行体  
 }else{  
 else的执行体  
 }  
 当if中的条件是true,执行if执行体  
 if中的条件是false,执行else执行体  
 一个语句中,包含多个 if , 只要有1个if的条件是true,其他的代码,不在执行***public class** TestIfElseIf {  
 **public static void** main(String[] args) {  
 *//成绩判断要求 ,成绩>80 成绩>70 成绩>60 不及格  
 //定义变量,保存成绩* **int** grade = 75;  
 *//使用if else if 语句对成绩判断* **if** (grade > 80) {  
 System.***out***.println(grade + **" 成绩是优"**);  
 } **else if** (grade > 70) {  
 System.***out***.println(grade + **" 成绩是良"**);  
 } **else if** (grade > 60) {  
 System.***out***.println(grade + **" 成绩是中"**);  
 } **else** {  
 System.***out***.println(grade + **" 成绩是差"**);  
 }  
  
 */\*  
 执行效率低  
 if( grade > 80){  
 //优  
 }  
  
 if( grade > 70 && grade < 80){  
 //良  
 }  
  
 if( grade > 60){  
 //中  
 }\*/* }  
}

## 面向对象

### 达到

Java 是面向对象的高级编程语言，类和对象是 Java 程序的构成核心。围绕着 Java 类和 Java 对象，有三大基本特性：封装是 Java 类的编写规范、继承是类与类之间联系的一种形式、而多态为系统组件或模块之间解耦提供了解决方案。

### 面向对象思想

面向对象编程是当今主流的程序设计思想，已经取代了过程化程序开发技术，Java 是完全面向对象编程语言，所以必须熟悉面向对象才能够编写 Java 程序。

　　面向对象的程序核心是由对象组成的，每个对象包含着对用户公开的特定功能和隐藏的实现部分。程序中的很多对象来自 JDK 标准库，而更多的类需要我们程序员自定义。

　　从理论上讲，只要对象能够实现业务功能，其具体的实现细节不必特别关心。

　　面向对象有以下特点：

　　（1）面向对象是一种常见的思想，比较符合人们的思考习惯；

　　（2）面向对象可以将复杂的业务逻辑简单化，增强代码复用性；

　　（3）面向对象具有抽象、封装、继承、多态等特性。

　　面向对象的编程语言主要有：C++、Java、C#等。

### 类和对象的关系

**类：**对某类事物的普遍一致性特征、功能的抽象、描述和封装，是构造对象的模版或蓝图，用 Java 编写的代码都会在某些类的内部。类之间主要有：依赖、聚合、继承等关系。

**对象：**使用 new 关键字或反射技术创建的某个类的实例。同一个类的所有对象，都具有相似的数据（比如人的年龄、性别）和行为（比如人的吃饭、睡觉），但是每个对象都保存着自己独特的状态，对象状态会随着程序的运行而发生改变，需要注意状态的变化必须通过调用方法来改变，这就是封装的基本原则。

### 封装思想

核心思想就是“隐藏细节”、“数据安全”：将对象不需要让外界访问的成员变量和方法私有化，只提供符合开发者意愿的公有方法来访问这些数据和逻辑，保证了数据的安全和程序的稳定。

　　具体的实现方式就是：

　　使用 private 修饰符把成员变量设置为私有，防止外部程序直接随意调用或修改成员变量，然后对外提供 public 的 set 和 get 方法按照开发者的意愿（可以编写一些业务逻辑代码，虽然很少这样做）设置和获取成员变量的值。

　　也可以把只在本类内部使用的方法使用 private，这就是封装的思想，是面向对象最基本的开发规范之一。

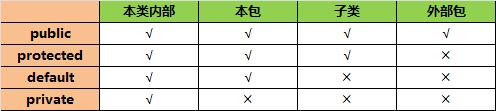
　　在此，我们有必要说一下 Java 的访问权限修饰关键字。Java 中主要有 private、protected、public 和 默认访问权限 四种：

　　public 修饰符，具有最大的访问权限，可以访问任何一个在 CLASSPATH 下的类、接口、异常等。

　　protected 修饰符，主要作用就是用来保护子类，子类可以访问这些成员变量和方法，其余类不可以。

　　default 修饰符，主要是本包的类可以访问。

　　private 修饰符，访问权限仅限于本类内部，在实际开发过程中，大多数的成员变量和方法都是使用 private 修饰的。



### Java反射

Java 的访问控制是停留在编译层的，只在编译时进行访问权限检查，不会在类文件中留下痕迹。通过反射机制，还是可以访问类的私有成员的。在实际的开发过程中，这样的封装方式已经成了 Java Bean 代码编写的规范。现在主流的框架在使用反射技术为对象赋值、取值时使用的都是 set 和 get 方法，而不是直接操作字段的值。

### 继承和类实例化过程

（1）在多个不同的类中抽取出共性的数据和逻辑，对这些共性的内容进行封装一个新的类即父类（也叫做超类或基类），让之前的类来继承这个类，那些共性的内容在子类中就不必重复定义，比如 BaseDAO、BaseAction 等。

　　\* （2）Java 的继承机制是单继承，即一个类只能有一个直接父类。

　　\* （3）如果子类和父类有同名成员变量和方法，子类可以使用 super 关键字调用父类的成员变量和方法，上述使用方式前提是成员在子类可见。

　　\* （4）在调用子类构造方法时，会隐式的调用父类的构造方法 super()。如果父类没有无参构造方法，为了避免编译错误，需要在子类构造方法中显式的调用父类的含参构造方法。

　　（5）子类创建时调用父类构造方法：子类需要使用父类的成员变量和方法，所以就要调用父类构造方法来初始化，之后再进行子类成员变量和方法的初始化。因此，构造方法是无法覆盖的。

　　\* （6）当子类需要扩展父类的某个方法时，可以覆盖父类方法，但是子类方法访问权限必须大于或等于父类权限。

　　（7）继承提高了程序的复用性、扩展性，也是 Java 语言多态特征的前提。

（8）在实际开发、程序设计过程中，并非先有的父类，而是先有了子类中通用的数据和逻辑，然后再抽取封装出来的父类。

我们简单了解下类的实例化过程

　　（1）JVM 读取指定 classpath 路径下的 class 文件，加载到内存，如果有直接父类，也会加载父类；

　　（2）堆内存分配空间；

　　（3）执行父类、子类静态代码块；

　　（4）对象属性进行默认初始化；

　　（5）调用构造方法；

　　（6）在构造方法中，先调用父类构造方法初始化父类数据；

　　（7）初始化父类数据后，显示初始化，执行子类的构造代码块；

　　（8）再进行子类构造方法的特定初始化；

　　（9）初始化完毕后，将地址赋值给引用

### 多态、反射和组件解耦

多态指允许不同类的对象对同一“消息”做出响应。即同一消息可以根据发送对象的不同而采用多种不同的行为方式。可以用于消除类型之间的耦合关系，Spring 的核心就是多态和面向接口编程。

　　（1）Java 中可以使用父类、接口变量引用子类、实现类对象；

　　（2）在这个过程中，会对子类、实现类对象做自动类型提升，其特有功能就无法访问了，如果需要使用，可以做强制类型转换。

　　Java 的反射技术和多态特性是框架开发、组件解耦的核心，在这方面，Spring 的 IOC 和 DI 为我们提供了一个极好的学习范例，Spring 的 IOC 使用反射技术创建、管理对象，DI 使用多态技术为组件注入依赖对象。

　　在没有学习 Spring 之前，简单的解决方案是使用一个 .properties 文件保存程序中使用的接口、实现类类型键值信息，然后在程序中使用一个全局 Properties 对象保存这些信息，并且使用反射技术把这些实现类初始化、提供一个静态的方法获取指定接口的实现类对象，在组件中就可以使用依赖对象的键获取需要的对象。

　　这样的方案带来的好处就是：当我们需要修改某个组件的实现方式时，比如把之前 JDBC 的 DAO 实现改为 Hibernate 实现，只要把这些新的实现类放到 classpath 下，把 .properties 文件对应接口的实现类类型改成新的 Hibernate 实现类，而不需要修改依赖组件的代码。

## 数组

### 一维数组

*数组的两个定义方式  
 数据类型[] 变量名 = new 数据类型[]{元素1,元素2,元素3};  
注意事项: new 后面的中括号中,不允许写任何内容,写了就编译失败  
  
数据类型[] 变量名 = {元素1,元素2,元素3};*

*//定义数组,定义的同时,对元素进行赋值***int**[] arr = **new int**[]{1,2,4,5,7,9};  
System.***out***.println(arr.**length**);  
System.***out***.println(arr[5]);  
  
  
System.***out***.println(**"=================="**);  
*//定义数组,直接大括号赋值***int**[] arr2 = {0,2,4,6,8};  
System.***out***.println(arr2.**length**);  
System.***out***.println(arr2[1]);

### 一维数组的遍历

*定义方法,实现数组的遍历*

**public class** ArrayMethodTest{  
 **public static void** main(String[] args){  
 **int**[] arr = {11,44,55,33,66};  
 *printArray*(arr);  
   
 **int**[] arr2 = {22,88,99,33,66};  
 *printArray*(arr2);  
   
 }  
 */\*  
 定义方法,实现功能  
 返回值: void  
 方法参数: 数组  
 \*/* **public static void** printArray(**int**[] arr){  
 *//输出一半中括号,不要换行打印* System.***out***.print(**"["**);  
 *//数组进行遍历* **for**(**int** i = 0 ; i < arr.**length** ; i++){  
 *//判断遍历到的元素,是不是数组的最后一个元素  
 //如何判断 循环变量 到达 length-1* **if**( i == arr.**length**-1 ){  
 *//输出数组的元素和]* System.***out***.print(arr[i]+**"]"**);  
 }**else**{  
 *//不是数组的最后一个元素,输出数组元素和逗号* System.***out***.print(arr[i]+**","**);  
 }  
 }  
 System.***out***.println();  
 }  
}

### 一维数组的逆序

*数组的逆序:  
 数组中的元素,进行位置上的交换  
逆序 不等于 反向遍历  
就是数组中最远的两个索引,进行位置交换,实现数组的逆序  
使用的是数组的指针思想,就是变量,思想,可以随时变换索引  
反转 reverse  
实现步骤:  
 1. 定义方法,实现数组的逆序  
 2. 遍历数组  
 实现数组的最远索引换位置  
 使用临时的第三方变量*

**public class** ArrayMethodTest\_1{  
 **public static void** main(String[] args){  
 **int**[] arr = {3,5,7,1,0,9,-2};  
 *//调用数组的逆序方法  
 reverse*(arr);  
 *//看到数组的元素,遍历  
 printArray*(arr);  
 }  
   
 */\*  
 定义方法,实现数组的逆序  
 返回值: 没有返回值  
 参数: 数组就是参数  
 \*/* **public static void** reverse(**int**[] arr){  
 *//利用循环,实现数组遍历,遍历过程中,最远端换位  
 //for的第一项,定义2个变量, 最后,两个变量++ --* **for**( **int** min = 0 , max = arr.**length**-1 ; min < max ; min++,max--){  
 *//对数组中的元素,进行位置交换  
 //min索引和max索引的元素交换  
 //定义变量,保存min索引* **int** temp = arr[min];  
 *//max索引上的元素,赋值给min索引* arr[min] = arr[max];  
 *//临时变量,保存的数据,赋值到max索引上* arr[max] = temp;  
 }  
 }  
   
 */\*  
 定义方法,实现功能  
 返回值: void  
 方法参数: 数组  
 \*/* **public static void** printArray(**int**[] arr){  
 *//输出一半中括号,不要换行打印* System.***out***.print(**"["**);  
 *//数组进行遍历* **for**(**int** i = 0 ; i < arr.**length** ; i++){  
 *//判断遍历到的元素,是不是数组的最后一个元素  
 //如何判断 循环变量 到达 length-1* **if**( i == arr.**length**-1 ){  
 *//输出数组的元素和]* System.***out***.print(arr[i]+**"]"**);  
 }**else**{  
 *//不是数组的最后一个元素,输出数组元素和逗号* System.***out***.print(arr[i]+**","**);  
 }  
 }  
 System.***out***.println();  
 }  
}

### 二维数组

**int**[][] arr = { {1,2,3},{4,5},{6,7,8,9},{0} };

**public class** ArrayArrayDemo\_3{  
 **public static void** main(String[] args){  
 **int**[][] arr = { {11,12} , {21,22,23} ,{31,32,33,34} };  
 *//定义变量,保存2个求和的值* **int** sum = 0 ; *//所有元素的求和* **int** groupSum = 0 ; *//每个一维数组的求和* **for**(**int** i = 0 ; i < arr.**length** ; i++){  
 **for**(**int** j = 0 ; j < arr[i].**length** ; j++){  
 *//将一维数组的元素求和* groupSum += arr[i][j];  
 }  
 System.***out***.println(**"每个小组总金额 "**+groupSum);  
   
 *//将每个一维数组的总和在相加* sum += groupSum;  
 *//每个唯一数组求和,清空* groupSum = 0;  
 }  
 System.***out***.println(**"本公司总金额 "**+sum);  
 }  
}

*定义数组容器  
定义数组容器,要素,强制数据类型的语言  
必须有数据类型, 大小, 就是存储数据的个数  
定义数组公式:  
 数据类型[] 变量名 = new 数据类型[存储元素的个数];  
   
 数据类型: 数组中存储元素的数据类型  
 [] 表示数组的意思  
 变量名 自定义标识符  
   
 new 创建容器关键字  
 数据类型: 数组中存储元素的数据类型  
 [] 表示数组的意思  
 元素个数,就是数组中,可以存储多少个数据 (恒定, 定长)  
   
数组是一个容器: 存储到数组中的每个元素,都有自己的自动编号  
自动编号,最小值是0, 最大值,长度-1  
自动编号专业名次, 索引(index), 下标, 角标  
访问数组存储的元素,必须依赖于索引, 公式 数组名[索引]  
  
Java提供一个属性,操作索引的  
数组的一个属性,就是数组的长度, 属性的名字 length  
使用属性: 数组名.length 数据类型 int  
  
数组的最小索引是0, 最大索引数组.length-1*

## Swith Case

### 选择语句

选择语句 switch语句  
 编写格式  
 swtich(表达式){  
 case 常量1 :  
 要执行的语句;  
 break;  
   
 case 常量2 :  
 要执行的语句;  
 break;  
   
 case 常量3 :  
 要执行的语句;  
 break;  
   
 default:  
 要执行的语句;  
 break;  
 }  
 执行流程: 表达式,和case后面的常量进行比较  
 和哪个case后的常量相同,就执行哪个case后面的程序,遇到break,就全结束  
   
 关键字: switch case default break  
如果等于1，则输出星期一  
如果等于2，则输出星期二  
如果等于3，则输出星期三  
如果等于4，则输出星期四  
如果等于5，则输出星期五  
如果等于6，则输出星期六  
如果等于7，则输出星期天  
  
switch语句中的表达式的数据类型,是有要求的  
JDK1.0 - 1.4 数据类型接受 byte short int char  
JDK1.5 数据类型接受 byte short int char enum(枚举)  
JDK1.7 数据类型接受 byte short int char enum(枚举), String

**public class** SwitchDemo{  
 **public static void** main(String[] args){  
 *//定义变量,保存星期几的整数* **int** week = 13;  
 *//switch语句* **switch**(week){  
 **case** 1:  
 System.***out***.println(**"星期一"**);  
 **break**;  
   
 **case** 2:  
 System.***out***.println(**"星期二"**);  
 **break**;  
   
 **case** 3:  
 System.***out***.println(**"星期三"**);  
 **break**;  
   
 **case** 4:  
 System.***out***.println(**"星期四"**);  
 **break**;  
   
 **case** 5:  
 System.***out***.println(**"星期五"**);  
 **break**;  
   
 **case** 6:  
 System.***out***.println(**"星期六"**);  
 **break**;  
   
 **case** 7:  
 System.***out***.println(**"星期日"**);  
 **break**;  
   
 **default**:  
 System.***out***.println(**"星期不存在"**);  
 **break**;  
 }  
 }  
}

### switch特性: case 穿透性 案例: 星期1-5输出 工作日 , 星期6-7 输出休息日 case 1 case 2... case5 输出结果相同 case 后面的常量,和switch中数据相同 ,没有break,一直向下穿透 case 后面没有break, 程序就会一直向下穿透public class SwitchDemo\_1{ public static void main(String[] args){ int week = 30; switch(week){ case 1: case 2: case 3: case 4: case 5: System.*out*.println("工作日"); break; case 6: case 7: System.*out*.println("休息日"); break; } } }

## 方法的定义

### 方法的定义

方法定义和使用的注意事项  
 1. 方法不能定义在另一个方法的里面  
2. 写错方法名字  
3. 写错了参数列表  
4. 方法返回值是void,方法中可以省略return 不写  
 return 下面不能有代码  
5. 方法返回值类型,和return 后面数据类型必须匹配  
6. 方法重复定义问题  
7. 调用方法的时候,返回值是void, 不能写在输出语句中

方法,调用中的参数传递问题  
1. 方法参数是基本数据类型  
2. 方法参数是引用类型  
 传递的是内存地址!!!

## 方法的OVERLOAD

### 方法overload

方法的重载特性 (overload)  
在同一个类中,允许出现同名的方法,只要方法的参数列表不同即可,这样方法就是重载  
参数列表不同: 参数的个数,数据类型,顺序

方法重载的注意事项  
1. 参数列表必须不同  
2. 重载和参数变量名无关  
3. 重载和返回值类型无关  
4. 重载和修饰符无关  
技巧: 重载看方法名和参数列表

## ArrayList

### ArrayList集合的使用

ArrayList集合的使用  
 也是引用数据类型  
 步骤:  
 1. 导入包 java.util包中  
2. 创建引用类型的变量  
 数据类型< 集合存储的数据类型> 变量名 = new 数据类型 <集合存储的数据类型> ();  
 集合存储的数据类型: 要将数据存储到集合的容器中  
 创建集合引用变量的时候,必须要指定好,存储的类型是什么  
   
 ArrayList<String> array = new ArrayList<String>();  
   
3. 变量名.方法   
  
注意: 集合存储的数据,8个基本类型对应8个引用类型  
存储引用类型,不存储基本类型

*//创建集合容器,指定存储的数据类型  
//存储字符串*ArrayList<String> array = **new** ArrayList<String>();  
  
*//创建集合容器,存储整数*ArrayList<Integer> array2 = **new** ArrayList<Integer>();  
  
*//创建集合容器,存储手机类型*ArrayList<Phone> array3 = **new** ArrayList<Phone>();

### ArrayList集合中的方法

*add(参数) 向集合中添加元素,数据存储进去  
方法中的参数类型,定义集合对象时候的类型是一致  
  
ArrayList<Integer> array = new ArrayList<Integer>();  
array.add(3);*

*get(int index) 取出集合中的元素,get方法的参数,写入索引  
  
size() 返回集合的长度, 集合存储元素的个数*

### ArrayList集合的遍历

*集合的遍历  
实现思想也是索引思想  
集合的索引从0开始,到 size()-1  
方法get(int index)*

### ArrayList集合的补充

*集合ArrayList方法的补充  
   
add(int 索引,存储的元素) 将元素添加到指定的索引上  
set(int 索引,修改后的元素) 将指定索引的元素,进行修改  
remove(int 索引) 删除指定索引上的元素  
clear() 清空集合中的所有元素*

## 定义类

### 类的定义

*使用类的形式,对现实中的事物进行描述  
 事物: 方法,属性*

*方法: 这个事物具备的功能  
属性: 变量  
  
 格式:  
 public class 类名{  
 属性定义  
 修饰符 数据类型 变量名 = 值  
   
 方法定义  
 修饰符 返回值类型 方法名(参数列表){  
   
 }  
 }  
   
 要求: 使用Java中的类,描述生活中的手机这个事物  
 手机事物,具备哪些属性, 属于他自己的特性  
 颜色,品牌, 大小*

*实现引用类型的步骤  
 1: 导入包 , 类都是在同一个文件夹,不需要导入包  
 2: 创建引用类型的变量  
 3: 变量.类型中的功能*

*测试手机类  
创建2个手机类的变量*

## 排序

### bubbleSort

定义方法,实现数组的冒泡排序  
 返回值: 没有  
 参数: 数组**public static void** bubbleSort(**int**[] arr){  
 **for**(**int** i = 0 ; i < arr.**length** - 1; i++){  
 *//每次内循环的比较,从0索引开始, 每次都在递减* **for**(**int** j = 0 ; j < arr.**length**-i-1; j++){  
 *//比较的索引,是j和j+1* **if**(arr[j] > arr[j+1]){  
 **int** temp = arr[j];  
 arr[j] = arr[j+1];  
 arr[j+1] = temp;  
 }  
 }  
 }  
}

### selectSort

定义方法,实现数组的选择排序  
 返回值: 没有  
 参数: 数组  
 实现步骤:  
 1.嵌套循环实现排序  
 外循环,控制的是一共比较了多少次  
 内循环,控制的是每次比较了多少个元素  
 2. 判断元素的大小值  
 小值,存储到小的索引**public static void** selectSort(**int**[] arr){  
 **for**(**int** i = 0 ; i < arr.**length** - 1; i++){  
 *//内循环,是每次都在减少,修改变量的定义* **for**(**int** j = i+1 ; j < arr.**length** ; j++){  
 *//数组的元素进行判断* **if**(arr[i] > arr[j]){  
 *//数组的换位* **int** temp = arr[i];  
 arr[i] = arr[j];  
 arr[j] = temp;   
 }  
 }  
 }  
}

### selectSort

两种排序的方式  
选择排序: 数组的每个元素都进行比较  
冒泡排序: 数组中相邻元素进行比较  
规则: 比较大小,位置交换

## 查找

### 顺序查找

定义方法,实现数组的普通查询  
 返回值: 索引  
 参数: 数组, 被找的元素  
   
 实现步骤:  
 1. 遍历数组  
2. 遍历过程中,使用元素和数组中的元素进行比较  
 如果相同,返回元素在数组中的索引  
如果不同,返回负数

**public static int** search(**int**[] arr, **int** key){  
 *//遍历数组* **for**(**int** i = 0 ; i < arr.**length** ; i++){  
 *//数组元素,被查找的元素比较* **if**(arr[i] == key){  
 *//返回索引* **return** i;  
 }  
 }  
 **return** -1;  
}

### 二分查找

对于已经排序的数组进行二分查找

*定义方法,实现,折半查找  
返回值: 索引  
参数: 数组,被找的元素   
实现步骤:  
 1. 需要的变量定义  
 三个,三个指针  
   
 2. 进行循环折半  
 可以折半的条件 min <= max  
   
 3. 让被找元素,和中间索引元素进行比较  
 元素 > 中间索引 小指针= 中间+1  
 元素 < 中间索引 大指针= 中间-1  
 元素 == 中间索引 找到了,结束了,返回中间索引  
   
 4. 循环结束,无法折半  
 元素没有找到 ,返回-1*

**public static int** binarySearch(**int**[] arr, **int** key){  
 *//定义三个指针变量* **int** min = 0 ;  
 **int** max = arr.**length** -1 ;  
 **int** mid = 0;  
 *//循环折半,条件 min<=max* **while**( min <= max){  
 *//公式,计算中间索引* mid = (min+max)/2;  
 *//让被找元素,和中间索引元素进行比较* **if**(key > arr[mid]){  
 min = mid + 1;  
 }**else if** (key < arr[mid]){  
 max = mid - 1;  
 }**else**{  
 *//找到元素,返回元素索引* **return** mid;  
 }  
 }  
 **return** -1;  
}

## ASCII

### ASCII

*ASCII编码表演示  
字符Java 数据类型,char  
整数Java 数据类型,int  
  
int 类型和 char 数据类型转换  
char 两个字节, int 四个字节  
  
char转成int类型的时候,类型自动提示,char数据类型,会查询编码表,得到整数  
int转成char类型的时候,强制转换,会查询编码表  
  
char存储汉字,查询Unicode编码表  
  
char可以和int计算,提示为int类型, 内存中两个字节  
char取值范围是0-65535, 无符号的数据类型*

**public class** ASCIIDemo{  
 **public static void** main(String[] args){  
 **char** c = **'a'**;  
 **int** i = c + 1;  
 System.***out***.println(i);  
   
 **int** j = 90;  
 **char** h = (**char**)j;  
 System.***out***.println(h);  
   
 System.***out***.println( (**char**)6 );  
   
 *//char k = '你';  
 //System.out.println(k);  
   
   
 //char m = -1;* }  
}