Java基础：

·数据类型（变量的作用域）

·运算符

·分支和循环

·数组

1. 数据类型：

·基本数据类型（原始数据类型）：

数值：int short long double float byte

其他类型：boolean char

·引用数据类型：数组、接口、类

·基本数据类型的初始化值：

|  |
| --- |
| public class JavaDemo01{  public static void main(String args[]){  int j[] = new int[10];  System.out.println("HelloWorld" + j[0]);  }  } |

通过以上程序运行结果发现，int类型的初始值为0。

·变量的作用域：声明变量最近的一对{}就是变量的作用域范围

·成员（全局）变量：声明在方法的外部，类中的所有方法都可以访问的变量。

·局部变量：声明在方法中，只有特定的代码（声明变量的方法）才能够访问此变量。

数据类型转换：

·自动类型转换：向上转型就是自动类型转换

|  |
| --- |
| public class JavaDemo03{  public static void main(String args[]){    int i = 10;  float f = i;//自动类型转换  System.out.println(f);    }  } |

·强制类型转换：乡下转型为强制类型转换

|  |
| --- |
| public class JavaDemo04{  public static void main(String args[]){    double d = 10.0001;  int i = (int)d;//强制类型转换  System.out.println(i);  }  } |

1. 运算符

·数学运算符：+ - \* / %

%：表示取余数，比如10%3 =1，10%2=0

|  |
| --- |
| public class JavaDemo05{  public static void main(String args[]){    System.out.println(100%30);  }  } |

·关系运算符：> < >= <=

关系运算符的值都为boolean型，即值为true or false

|  |
| --- |
| public class JavaDemo06{  public static void main(String args[]){  int i = 10;  int j = 20;  System.out.println(i>j);  System.out.println(i<j);  }  } |

·逻辑运算符

&&：表示&&前后的表达式表达式都为true的时候，整个表达式为true，如果&&前后有一个表达式为false，则整个表达式都为false

&：表示&前后的表达式表达式都为true的时候，整个表达式为true，如果&&前后有一个表达式为false，则整个表达式都为false，和&&不同的是，&在任何情况下都会判断所有的表达式，而&&则是先判断第一个表达式，如果第一个表达式为false的话，则后面的表达式不会进行判断。

||：表示||前后的表达式如果有一个或者一个以上的表达式为真，则整个表达式为真， 如果||前后的所有表达式都为假，则整个表达式都为假。

|：表示|前后的表达式如果有一个或者一个以上的表达式为真，则整个表达式为真， 如果||前后的所有表达式都为假，则整个表达式都为假。和||的区别是在于|会判断所有的表达式，而||先判断第一个表达式，如果第一个表达式为真，则不会去判断第二个表达式。

！：表示非，即！false= true，！true = false。

* 三元运算符（三目运算符）：表达式1？值1：值2，

解释：表达式1的值如果为真，则整个表达式的值为值1，如果表达式1的值为假，则整个表达式的值为值2.

|  |
| --- |
| public class JavaDemo08{  public static void main(String args[]){  int i = 10;  int j = 20;  int x = i<j?i:j;  System.out.println(x);  }  } |

·赋值运算符：+= -= \*= /= ++ --

解释：  
 ·j += i ：表示j = j+i；

·j \*= i ：表示j = j\*i；

·j /= i ：表示j = j/i；

·j -= i ：表示j = j-i；

|  |
| --- |
| public class JavaDemo09{  public static void main(String args[]){  int i = 10;  int j = 20;  j \*= i;//j = j\*i  System.out.println(j);    }  } |

·j++:表示j = j+1

·j--:表示j = j-1

++:表示自增，++放在前面表示先自增，后操作，++放在后面表示先操作后自增

|  |
| --- |
| public class JavaDemo10{  public static void main(String args[]){  int i = 10;    System.out.println(i++);//10  System.out.println(i);//11    }  } |

--:表示自减，--放在前面表示先自减，后操作，--放在后面表示先操作后自减。

·运算符的优先级：和数学里面的运算符是一样的，（）拥有最高的运算级别。

1. 分支和循环（控制流语句）

分支语句：if　else ,if else if,switch case

·if else：例如成绩在60分以上的为几个，否则为不及格，现在小明的成绩为59分，判断是否及格。

|  |
| --- |
| public class JavaDemo11{  public static void main(String args[]){  int java\_score = 69;  if(java\_score >= 60){  System.out.println("小明的java成绩及格了");  }else{  System.out.println("小明的java成绩不及格");  }  }  } |

现在将成绩分等级：A：90-100，B：80-89，C：70-79，D:60-69，E:<60

现在小明的父母想知道小明java成绩的等级是什么。

|  |
| --- |
| public class JavaDemo12{  public static void main(String args[]){  int java\_score = 69;  if(java\_score >= 90){  System.out.println("小明的java成绩为A等");  }else{  if(java\_score >= 80){  System.out.println("小明的java成绩为B等");  }else{  if(java\_score >= 70){  System.out.println("小明的java成绩为C等");  }else{  if(java\_score >= 60){  System.out.println("小明的java成绩为D等");  }else{  System.out.println("小明的java成绩为E等");  }  }  }  }  }  } |

现在虽然功能已经实现了，但是发现if else嵌套过多，这样降低了程序的可读性和可维护性。

If else if来完成以上的功能。

|  |
| --- |
| public class JavaDemo13{  public static void main(String args[]){  int java\_score = 69;  if(java\_score >= 90){  System.out.println("小明的java成绩为A等");  }else if (java\_score >= 80){  System.out.println("小明的java成绩为B等");  }else if(java\_score >= 70){  System.out.println("小明的java成绩为C等");  }else if(java\_score >= 60){  System.out.println("小明的java成绩为D等");  }else{  System.out.println("小明的java成绩为E等");  }  }  } |

如果说是纯粹的分支语句，提供了另外一种写法：

switch case

假如有一个数字，如果为1的话打印这个数字为1.。。。

|  |
| --- |
| public class JavaDemo14{  public static void main(String args[]){  int x = 3;  switch(x){  case 1:  System.out.println("这个数字是1");  case 2:  System.out.println("这个数字是2");  case 3:  System.out.println("这个数字是3");  case 4:  System.out.println("这个数字是4");  case 5:  System.out.println("这个数字是5");  default: //这个表示之上的条件都不符合的情况下执行此处代码  System.out.println("这个数字是不在1-5之间");  }  }  } |

发现以上程序在执行过程中，满足一个条件之后，后面的都会执行，这个时候有人在想，如果能够在满足一个条件执行完之后能够立即退出这个分支语句，才是正确的。这个时候我们需要用到一个关键字break，表示退出分支。

|  |
| --- |
| public class JavaDemo14{  public static void main(String args[]){  int x = 3;  switch(x){  case 1:  System.out.println("这个数字是1");  break;  case 2:  System.out.println("这个数字是2");  break;  case 3:  System.out.println("这个数字是3");  break;  case 4:  System.out.println("这个数字是4");  break;  case 5:  System.out.println("这个数字是5");  break;  default: //这个表示之上的条件都不符合的情况下执行此处代码  System.out.println("这个数字是不在1-5之间");  break;  }  }  } |

·循环：在满足循环条件的情况下，反复的执行特定的代码。

循环有四个组成部分：初始化部分、循环条件部分、循环体部分、循环部分。

循环的分类：

·for循环

·while 循环

·do while 循环

for循环：

|  |
| --- |
| for(表达式1;表达式2;表达式3){  语句  } |

表达式1：循环条件的初始化部分

表达式2：循环条件的判断部分

表达式3：循环条件的改变部分（循环体部分）

语句：循环部分

范例：从1+2+3+++++100；

|  |
| --- |
| public class JavaDemo15{  public static void main(String args[]){  int sum = 0;  for(int i=1;i<=100;i++){  sum += i;//表示循环相加  }  System.out.println(sum);  }  } |

**一般在明确知道循环次数的情况下，使用for循环，在循环的时候要严防死循环的出现。**

死循环：程序条件一直满足，而且一直执行下去，直到程序崩溃。

现在从1输到100，

|  |
| --- |
| public class JavaDemo15{  public static void main(String args[]){  for(int i=1;i<=100;i++){  System.out.print(i+"\t");  }  }  } |

现在我想要输出到50就不再输出了，这个时候需要在输出50之后退出循环。这个时候需要使用break关键字，表示循环不再执行，即中断循环。

|  |
| --- |
| public class JavaDemo15{  public static void main(String args[]){  for(int i=1;i<=100;i++){  System.out.print(i+"\t");  if(i>=50){  break;//表示直接中断循环  }  }  }  } |

现在我只想输出1-100中的偶数，不输出奇数，现在怎么办呢？这个时候需要使用到continue关键字。continue表示终端本次循环，继续执行下一次循环。

|  |
| --- |
| public class JavaDemo15{  public static void main(String args[]){  for(int i=1;i<=100;i++){  if(i%2 != 0){  continue;  }  System.out.print(i+"\t");  }  }  } |

|  |  |
| --- | --- |
|  | 区别 |
| break | 中断循环，循环不再执行 |
| continue | 中断本次循环，继续执行下一次循环 |

·while循环

语法：while（表达式）{

语句

}

在不明确循环次数的情况下使用while循环。

|  |
| --- |
| public class JavaDemo16{  public static void main(String args[]){  int sum = 0;  int i = 1;  while (i<=100)//表达式  {  sum += i;  i++;  }  System.out.println(sum);  }  } |

·do while循环：先执行一次循环，再判断是否继续执行循环

|  |
| --- |
| public class JavaDemo17{  public static void main(String args[]){  int sum = 0;  int i = 1;  //表达式  do{  sum += i;  i++;  }while (i<=100);  System.out.println(sum);  }  } |

总结：在明确知道循环次数的情况下，一般使用for循环

|  |  |
| --- | --- |
|  | 区别 |
| while | 先判断再循环 |
| do while | 先循环一次，再判断是否需要继续循环 |

练习：使用循环打印以下图形

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*

\*\*\*

\*

分析：

上半部分：

第一行：空格4个 \*号1个 空格数：5-行数 \*号数：2\*行数-1

第二行：空格3个\*号3个

第三行：空格2个\*号5个

第四行：空格1个\*号7个

第五行：没有空格\*号9个

下半部分：

第一行：空格1个 \*号7个 空格数：行数 \*号数：9-2i

第二行：空格2个\*号5个

第三行：空格3个\*号3个

第四行：空格4个\*号1个

|  |
| --- |
| public class JavaDemo18{  public static void main(String args[]){  for(int i=1;i<=5;i++){  for(int j=1;j<=5-i;j++){  System.out.print(" ");  }  for(int j=1;j<=2\*i-1;j++){  System.out.print("\*");  }  System.out.println();  }  for(int i=1;i<=4;i++){  for(int j=1;j<=i;j++){  System.out.print(" ");  }  for(int j=1;j<=9-2\*i;j++){  System.out.print("\*");  }  System.out.println();  }  }  } |

通过大家练习发现：

·不知道怎么去定义类 ：使用public修饰的类名必须要和文件名一致，public class 类名

·类名的首字母要大写

·代码要错落有致

1. 数组：存储一组相同数据类型的存储单元

语法：数据类型 变量名[]

数组初始化：

·静态初始化：在定义数组的同时给数组开辟空间并且赋值

·动态初始化：先给数组开辟空间，再赋值

|  |
| --- |
| **package** com.wanczy.arrayDemo;  **public** **class** ArrayDemo01 {  **public** **static** **void** main(String[] args) {  **int** i[] = {1,2,3,4,5,6};//静态初始化，在数组初始化的时候给数组开辟了6个空间，并且给每个空间赋值  **int** j[] = **new** **int**[10];//动态初始化，在数组初始化的时候给数组开辟了10个空间，但是没有给开辟的空间赋值    }  } |

数组中提供了一个属性length，用于取得数组的空间大小，就是取得数组中有多少个元素。

数组的元素下标从0开始，最后一个元素的下标是length-1

给动态初始化的数组赋值：

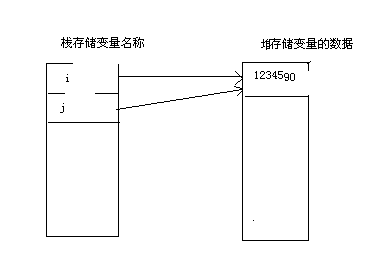
|  |
| --- |
| **package** com.wanczy.arrayDemo;  **public** **class** ArrayDemo01 {  **public** **static** **void** main(String[] args) {  **int** i[] = {1,2,3,4,5,6};//静态初始化，在数组初始化的时候给数组开辟了6个空间，并且给每个空间赋值  **int** j[] = **new** **int**[10];//动态初始化，在数组初始化的时候给数组开辟了10个空间，但是没有给开辟的空间赋值    **for**(**int** x = 0;x<j.length;x++){//给动态初始化数组赋值  j[x] = x\*100;  }    System.*out*.println(j[j.length-1]);  }  } |

数组输出：

|  |
| --- |
| **package** com.wanczy.arrayDemo;  **public** **class** ArrayDemo01 {  **public** **static** **void** main(String[] args) {  **int** i[] = {1,2,3,4,5,6};//静态初始化，在数组初始化的时候给数组开辟了6个空间，并且给每个空间赋值  **int** j[] = **new** **int**[10];//动态初始化，在数组初始化的时候给数组开辟了10个空间，但是没有给开辟的空间赋值    **for**(**int** x = 0;x<j.length;x++){//给动态初始化数组赋值  j[x] = x\*100;  }  System.*out*.println("i数组的输出");  **for** (**int** j2 = 0; j2 < i.length; j2++) {  System.*out*.print(i[j2]+"\t");  }  System.*out*.println();  System.*out*.println("j数组的输出");  **for** (**int** j2 = 0; j2 < j.length; j2++) {  System.*out*.print(j[j2]+"\t");  }  }  } |

数组的引用传递：

|  |
| --- |
| **package** com.wanczy.arrayDemo;  **public** **class** ArrayDemo02 {  **public** **static** **void** main(String[] args) {  **int** i[] = {1,2,3,4,5,6,7};    **int** j[] = i;  System.*out*.println("j数组赋值之前输出i数组");  **for** (**int** j2 = 0; j2 < i.length; j2++) {  System.*out*.print(i[j2] + "\t");  }  j[5] = 90;//重新给j的下标为5的元素赋值  System.*out*.println();  System.*out*.println("j数组赋值之后输出i数组");  **for** (**int** j2 = 0; j2 < i.length; j2++) {  System.*out*.print(i[j2] + "\t");  }      }  } |



数组的引用传递，就是指传递内存地址的使用权。只要是引用数据类型都存在引用传递的问题。

数组的排序：

·冒泡

·选择

·插入

·快速

|  |
| --- |
| **package** com.wanczy.arrayDemo;  **import** java.util.Arrays;  **public** **class** ArrayDemo03 {  **public** **static** **void** main(String[] args) {  **int** x[] = {90,70,50,88,23,32};  Arrays.*sort*(x);//java中提供的排序方法  **for** (**int** i = 0; i < x.length; i++) {  System.*out*.print(x[i] + "\t");  }  }  } |

数组查找：

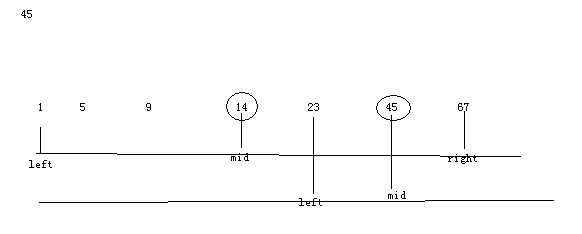
·傻瓜式查找：从数组的第一个元素开始一个一个的判断查找

·二分查找法（折半查找法）：

范例：傻瓜式查找

|  |
| --- |
| **package** com.wanczy.arrayDemo;  **public** **class** ArrayDemo04 {  **public** **static** **void** main(String[] args) {  String x[] = {"张三","李四","王五","赵六","田七"};    String name = "王五";    **for** (**int** i = 0; i < x.length; i++) {  **if**(**null** != x[i] && x[i].equals(name)){//equals方法用于判断字符串内容是否相同  System.*out*.println("已经查找到，对应数组的下标为"+i);  **return**;//结束词方法的运行  }  }  System.*out*.println("查无此人");  }  } |
| **package** com.wanczy.arrayDemo;  **public** **class** ArrayDemo05 {  **public** **static** **void** main(String[] args) {  **int** x[] = {1,2,3,4,5};    **int** name = 6;    **for** (**int** i = 0; i < x.length; i++) {  **if**(x[i]==name){//equals方法用于判断字符串内容是否相同  System.*out*.println("已经查找到，对应数组的下标为"+i);  **return**;//结束词方法的运行  }  }  System.*out*.println("查无此人");  }  } |

二分查找：二分查找首先要进行数组的排序：



|  |
| --- |
| **package** com.wanczy.arrayDemo;  **import** java.util.Arrays;  **public** **class** ArrayDemo06 {  **public** **static** **void** main(String[] args) {  **int** i[] = {1,23,45,14,67,5,9};  //排序  Arrays.*sort*(i);  **for** (**int** j = 0; j < i.length; j++) {  System.*out*.println(i[j] + "\t" );  }  **int** find = 2;  **int** left = 0;  **int** right = i.length - 1;  **do**{  **int** mid = (left+right)/2;  **if**(find > i[mid]){  left = mid + 1;  }**else** **if**(find < i[mid]){  right = mid -1 ;  }**else** **if**(find == i[mid]){  System.*out*.println("查找成功，对应数组的下标为"+mid);  **return**;  }      }**while**(left<=right);  System.*out*.println("查无数据");  }  } |

多维数组：就是指数组的元素还是一个数组，正常情况下，最多只会使用到二维数组。以二维数组为例，定义多维数组：

int i[][] = {{1,2,3},{2,3,4},{3,4,5}};

|  |
| --- |
| **package** com.wanczy.arrayDemo;  **public** **class** ArrayDemo07 {  **public** **static** **void** main(String[] args) {  **int** i[][] = {{1,2,3},{2,3,4},{3,4,5}};  **for** (**int** j = 0; j < i.length; j++) {  **for**(**int** x = 0;x<i[j].length;x++){  System.*out*.print(i[j][x]+"\t");  }  System.*out*.println();  }  }  } |

数组的复制：比如说现在有一个数组a，值为{1,2,3,4,5}，另外有一个数组为b数组，值为{9,8,7,6,5,4,3,2,1}，现在我要将a数组中的第1到第3个元素复制给b数组的第0到2的元素，复制之后，b数组的内容变为：{2,3，4,6,5,4,3,2,1}

在System类中提供了一个方法，arraycopy（源数组,源数组的下标，目标数组，目标数组的下标，复制的长度）方法用于数组的复制

|  |
| --- |
| **package** com.wanczy.arrayDemo;  **public** **class** ArrayDemo08 {  **public** **static** **void** main(String[] args) {  **int** a[] = {1,2,3,4,5};  **int** b[] = {9,8,7,6,5,4,3,2,1};  System.*arraycopy*(a, 1, b, 0, 3);//数组进行复制  **for** (**int** i = 0; i < b.length; i++) {  System.*out*.println(b[i] + "\t");  }  }  } |

本章总结：

·自动类型转换和强制类型转换

·理解数组的引用传递

·数组的排序和查找

·在明确知道循环次数的情况下，一般使用for循环

·会使用多维数组

·数组的复制

练习：写几个方法，对数组进行增删改查，先定义一个数组，长度比较大100个元素，然后写五个方法，增加、修改、删除、查找、打印的方法。main方法去调用这五个方法。

|  |
| --- |
| **package** com.wanczy.arrayDemo;  **public** **class** ArrayDemo09 {  **int** a[] = **new** **int**[4];    **public** **void** ins(**int** x){  **for** (**int** i = 0; i < a.length; i++) {  **if**(a[i] == 0){  a[i] = x;  System.*out*.println("新增成功");  **return**;  }  }  System.*out*.println("新增失败");  }  **public** **void** del(**int** x){  **int** y = **this**.find(x);  **if**(y == -1){  System.*out*.println("删除失败，没有查找到需要修改的元素");  }**else**{  **for** (**int** i = y; i < a.length; i++) {  **if**(y == a.length-1){  a[y] = 0;  **break**;  }**else**{  **if**(i<a.length-1){  a[i] = a[i+1];  a[a.length-1] = 0;  }  }    }  System.*out*.println("删除成功");  }  }  **public** **void** upd(**int** old\_x,**int** new\_x){  **int** x = **this**.find(old\_x);  **if**(x == -1){  System.*out*.println("修改失败，没有查找到需要修改的元素");  }**else**{  a[x] = new\_x;  System.*out*.println("修改成功");  }  }  **public** **int** find(**int** x){  **for** (**int** i = 0; i < a.length; i++) {  **if**(a[i]==x){  **return** i;//查找到，返回查找后的下标  }  }  **return** -1;//表示没有查找到元素  }  **public** **void** print(){  **for** (**int** i = 0; i < a.length; i++) {  System.*out*.print(a[i]+"\t");  }  System.*out*.println();  }  } |
| **package** com.wanczy.arrayDemo;  **import** java.util.Scanner;  **public** **class** TestDemo {  /\*\*  \* **@param** args  \*/  **public** **static** **void** main(String[] args) {  ArrayDemo09 arr = **new** ArrayDemo09();  Scanner sc = **new** Scanner(System.*in*);//表示键盘输入  **int** i = 0;  **do**{  System.*out*.println("输入1为新增， 输入2为删除，输入3为修改，输入4为查找，输入5为打印，输入0为退出");  i = sc.nextInt();//获得从键盘输入的int类型的值  **int** x;  **switch**(i){  **case** 1:  System.*out*.println("您选择了新增操作，请输入要新增的数字:");  x = sc.nextInt();  arr.ins(x);  **break**;  **case** 2:  System.*out*.println("您选择了删除操作，请输入要删除的数字:");  x = sc.nextInt();  arr.del(x);  **break**;  **case** 3:  System.*out*.println("您选择了修改操作，请输入要修改的数字:");  **int** old\_x = sc.nextInt();  System.*out*.println("请输入修改后的数字");  **int** new\_x = sc.nextInt();  arr.upd(old\_x,new\_x);  **break**;  **case** 4:  System.*out*.println("您选择了查找操作，请输入要查找的数字:");  x = sc.nextInt();  **int** ren = arr.find(x);  **if**(ren == -1){  System.*out*.println("查找失败，未找到需要查找的数字");  }**else**{  System.*out*.println("查找成功，对应元素在数组中下标为：" + ren);  }  **break**;  **case** 5:  arr.print();  **break**;  }  }**while**(i!=0);    }  } |