1. 操作符

1) 赋值操作符：

= ： int x=0,i=1,j=1;

a *= b ： 这里的”*=“由操作符”*”和”=“复合而成，它等价于 a=a*b; 这种复合操作符能使程序变得更加简洁。

a = a*b;

/= ： a/=b 等价于 a=a/b;

%= ： a%=b 等价于 a=a%b;

a +=b a=a+b

...

2) 比较操作符

> ： 大于

>= ： 大于等于

< ： 小于

<= ： 小

以上操作符只适用于整数类型和浮点数类型；

```
int a=1,b=1;
double d=1.0;
boolean result1 = a>b;        //result1的值为false;
boolean result2 = a<b;        //result2的值为false;
```

当小范围与大范围进行比较时,jvm会先将小范围类型转换成大范围类型在进行比较.

instanceof: 判断一个引用类型所引用的对象是否是一个类的实例。该操作符左边是一个引用类型，右边是一个类名或接口名。形式如下：

```
Student stu = new Student();
obj instanceof ClassName
    stu instanceof Student;
    stu instanceof Object;
    stu instanceof Teacher;
```

类型在转换之前,只有instanceof比较的对象返回为ture才能进行类型转换。否则报错类型转换异常

或者

obj instanceof InterfaceName

例如：

instanceof:判断引用属于那种类型
(默认都属于object类型)

```
String a = "zs";

System.out.println(a instanceof String);          //输出true;
```

3) 相等操作符

== ： 等于

基本类型比较 = = 比较内容

引用类型比较 = = 比较地址

!= ： 不等于

基本类型比较(= =比较内容),先将小范围数据类型转换成大范围数据类型再进行比较

既可以是基本类型，也可以是引用类型：

a. 基本类型：

```
int a=1,b=1;
float c=1.0f;
double d=1.0;
System.out.println(a==b);       //输出true;
System.out.println(a==c);       //输出true;
System.out.println(a==d);       //输出true;
System.out.println(c==d);       //输出true;
```

b. 引用类型：

这两个引用变量必须都引用同一个对象，结果才为true.

```
String s1 = new String("zs");
String s2 = new String("zs");
String s3 = s1;

System.out.println(s1 == s2);   //输出false;
System.out.println(s1 == s3);   //输出true;
System.out.println(s2 == s3);   //输出false;
```

4) 数学运算操作符

+ ： 数据类型值相加或**字符串连接**;

a. 数据类型值相加；

```
int    a=1+2;                    //a值为3;
先进行运算 后进行类型转换
        double b=1+2;                  //b值为3.0;
先进行转换 后进行运算
        double b=1+2.0;                //c值为3.0;
        先进行转换 后进行运算  再进行转换
        int b = (int)(1+2.0);                          //b值为3.0
```

b. 字符串连接；

```
所有与字符串相加的类型最后转换成字符串
System.out.println(“3”+”1”);          //输出31
System.out.println(1+2.0+"a");        //输出3.0a
System.out.println(1+2.0+"a"+true);   //输出3.0atrue
System.out.println(“a1”+2);           //输出a12
System.out.println(1+"a"+2);          //输出1a2
        System.out.println(1+”1”+2)             112
```

/ ： 整除, 如操作数均为整数，运算结果为商的整数部分

```
int a1=12/5;                          //a1变量的取值为2
int a2=13/5;                          //a2变量的取值为2
int a3=-12/5;                         //a3变量的取值为-2
int a4=-13/5;                         //a4变量的取值为-2
int a5=1/5;                          //a5变量的取值为0
double a6=-12/5;                        //a6变量的取值为-2.0
```

% ： 取模操作符, 如操作数均为整数，运算结果为商的整数部分

```
int a1=1%5;                           //a1变量的取值为1
int a2=13%5;                          //a2变量的取值为3
double a3=1%5;                          //a3变量的取值为1.0
```

5) 移位操作符

>> ： 算术右移位运算，也称做带符号右移位运算。

```
    移动位数n 如果n>32 使用n-32做为移动的位数
int a1 = 12 >> 1;                   //a1变量的取值为6;
int a2 = 128 >> 2;                  //a2变量的取值为32;
int a3 = 130 >> 2;                  //a3变量的取值为32;
int a3 = 12 >> 33 ;                 //a3变量的取值为6;
```

算数右移动的公式: 数字(正数)/2^n(n表示移动位数).

即使除不尽也不要紧,商就是最后需要保留数(计算的数)

注：a. 对12右移一位的过程为：舍弃二进制数的最后一位，在二进制数的开头增加一位符号位，由于12是正整数，因此增加的符号位为0;

b. 对-12右移二位的过程为：舍弃二进制数的最后二位，在二进制数的开头增加二位符号位，由于-12是负整数,因此增加的符号位为1;

1100 (12) ->1 0110 (6)
1000 0000 (128) ->2 0010 0000 (32)
1000 0010 (130) ->2 0010 0000 32

如果能除尽可以使用公式:数字/2^n(n表示移动位数) 前面加个-

如果不能除尽使用取反原始方法求值(e.g －13 先获取－13在内存中补吗,再进行>>2运算得到新的补吗,再反退回源码获取最后的值)

-130>>3 (源码)10000000000.....24位1000 0010(8位)

11111111111111.....24位1110 1111(补码->源码)

10000000000000000024位0001 0001

-17

-12>>1 ==1000 1100 取反 1111 0011＋1==>1111 0100(补吗)==>1111 1010

取反+1==>1000 0110==-6

-14>>1=-7

c. 表达式" a>>b " 等价于： a/(2^b)

右移33位相当于右移1位。

右移n位相当于除以2的n次方，大于32则相当于n-32位

12 /2 =6

<< ：左移位运算，也称为不带符号左移位运算。

左移:由于没有符号位，对于负数而言,也不需要求负数的源码.可以直接使用负数对应的正数的补吗来进行移位运行

不需要分情况,可以直接使用公式:数字(不区分正负数)*2^n(n表示移动位数)

```
int a1 = 12 << 1;                 //a1变量的取值为24;
int a2 = -12 << 2;                //a2变量的取值为-48;
int a3 = 128 << 2;                //a3变量的取值为512;
int a4 = -1 << 2;                 //a4变量的取值为-4;
```

注：a. 对12左移一位的过程为：舍弃二进制数的开头一位，在二进制数的尾部增加一个0;

b. 对-12左移二位的过程为：舍弃二进制数的开头二位，在二进制数的尾部增加二个0;

6) 位运算操作符

1010&0001 = 00000 1010 & 0001 0000 0
1010 | 0001 1011 11
1010 ^ 0001 1011 11

10 ~10 0000 1010(源码 补吗)

1111 0101(补吗—>源码) 取反+1 1000 1010+1 1000 1011(-11)

& ： 与运算，对两个操作元的每个二进制位进行与运算，运算规则为：1&1->1, 1&0->0, 0&1->0, 0&0->0;

全1为1,有0为0

| ： 或运算，对两个操作元的每个二进制位进行或运算，运算规则为：1|1->1, 1|0->1, 0|1->1, 0|0->0;

全0为0,有1为1

^ ： 异或运算，对两个操作元的每个二进制位进行或运算，运算规则为：1^1->0, 1^0->1, 0^1->1, 0^0->0;

两个值相同，为0， 不同为1；

~ ： 取反运算, ~1->0, ~0->1;

7) 逻辑操作符

短路操作符，如果能根据操作左边的布尔表达式就能推算出整个表达式的布尔值，将不执行操作符右边

的布尔表达式；

&&:左右两边都为true,整个表达式为true

||:左右两边一边为true,整个表达式为true

8) 条件操作符(三目运算符)

布尔表达式 ? 表达式1 : 表达式2

如果布尔表达式的值为true, 就返回表达式1的值， 否则返回表达式2的值。

```
int score = 61;
String result = score>60?”及格":"不及格";
```

注意:

String类型控制部分来自于?后面给定的对象类型

如果此时不是”及格”和”不及格” 而是 true和false

只能用boolean

```
boolean result = score>=90?true:false
```

2. 类型转换

1) 使用在基本数据类型和实例对象之间的转换。

2) 隐式转换和显式转换

3) **隐式转换是在运行期间转换，从子类转换到父类**。第五章会详细讲解。

4) **显式转换，缩小变化。强制类型转换**

自动类型转换，也称隐式类型转换，是指不需要书写代码，由系统自动完成的类型转换。由于实际开发中这样的类型转换很多，所以Java语言在设计时，没有为该操作设计语法，而是由JVM自动完成。

转换规则

从存储范围小的类型到存储范围大的类型。

具体规则为：

byte→short(char)→int→long→float→double

也就是说byte类型的变量可以自动转换为short类型，示例代码：

```
byte b = 10; int b = 128;
int i = b;   byte i = (byte)128; -128
```

这里在赋值时，JVM首先将b的值转换为short类型，然后再赋值给sh。

在类型转换时可以跳跃。示例代码：

```
byte b1 = 100;
int n = b1;
```

注意问题

在整数之间进行类型转换时，数值不发生改变，而将整数类型，特别是比较大的整数类型转换成小数类型时，由于存储方式不同，有可能存在数据精度的损失。

强制类型转换，也称显式类型转换，是指必须书写代码才能完成的类型转换。该类类型转换很可能存在精度的损失，所以必须书写相应的代码，并且能够忍受该种损失时才进行该类型的转换。

转换规则

从存储范围大的类型到存储范围小的类型。

具体规则为：

double→float→long→int→short(char)→byte

语法格式为：

(转换到的类型)需要转换的值

示例代码：

```
double d = 3.10;
int n = (int)d;
```

这里将double类型的变量d强制转换成int类型，然后赋值给变量n。需要说明的是小数强制转换为整数，采用的是“去1法”，**也就是无条件的舍弃小数点的所有数字**，则以上转换出的结果是3。

整数强制转换为整数时取数字的低位，例如int类型的变量转换为byte类型时，则只取int类型的低8位(也就是最后一个字节)的值。

示例代码：

```
int n = 123;
byte b = (byte)n;
int m = 128;
byte b1 = (byte)m;-128
 10101110
```

则b的值还是123，而b1的值为-128。b1的计算方法如下：

l 注意问题

强制类型转换通常都会存储精度的损失，所以使用时需要谨慎

3. 条件语句

有些程序代码只有满足特定条件的情况下才会被执行，Java语言支持两种条件处理语句：

i 1) if ... else

照书上讲的内容先讲if...else的语法。

a. if后面的表达式必须是布尔表达式，而不能为数字类型，例如下面的if(x)是非法的。

if和else if是并列关系,一次只能有一种情况被执行到.即使有多条语句都满足条件,也不可能执行多种情况.

```
int x=0;

if(x) {              //编译出错
   System.out.println("x不等于0");
} else {
   System.out.println("x等于0");
}
```

b.

e.g

if(true)

Test test = new Test();

这个程序虽然只有一句话,但是产生两个执行步骤,

1.Test test;

2.new Test();

对于内存理解也是2句话.所以必须添加{}才能编译通过.

if(true)

Test test = new Test();

此时代码会报错

1.Test test = new Test();实际是两句话

Test test; test = new Test();

2.if后面如果不跟大括号只能省略一句

3.省略的第一句中声明的变量test

再第二句中不存在,test的生命周期只在

if的语句中,不在if的外围

Integer.parseInt(args[0]):args[0]接受终端传入的值,该值时字符串类型

如果需要将字符串转换成整形.args[0] 表示接受第一个值,可以有多个值

ArrayIndexOutOfBoundsException:数组越界 0 表示第一个值

课堂练习：1) 写一个方法实现分时问侯, 如是8点至12点，返回"上午好", 12点至14点，返回"中午好",
14点至18点，返回"下午好", 其它时间返回"晚上好"

```
public String sayHello(int hour) {
    String msg;
    if(hour >=8 && hour < 12)
        msg = "Good morning";
    else if(hour>=12 && hour <14)
        msg = "Good noon";
    else if(hour>=14 && hour <18)
        msg = "Good afternoon";
    else
        msg = "Good evening";

    return msg;
}
```

2) 写一个方法判断某一年是否为闰年。

标准：1) 能被4整除，但不能被100整除;
或
2) 能被400整除；

```
public String isLeapYear(int year) {
    if((year%4==0 && year%100!=0) || (year%400==0))
        return “LeapYear”;
    else
        return “Year”;
}
```

2) switch

语法：

```
switch(expr) {
    case value1:{
        statements;
        break;
    …}
    case valueN
        statments;
        break;

    default:
        statements;
        break;
}
```

a. **expr的类型必须是byte, short, char或者int;**

b. valuesN类型必须是byte, short, char或者int, 该值必须是常量。各个case子句的valueN值不同；

c. 当switch表达式的值不与任何case子句匹配时，程序执行default子句，假如没有default子句，则程序直接退出switch语句。default子句可以位于switch语句中的任何位置。**永远都是最后被匹配**

d. 如果switch表达式与某个case表达式匹配，或者与default情况匹配，就从这个case子句或default子句开始执行。假如遇到break，就退出整个switch语句，否则依次执行switch语句中后续的case子句，不再检查case表达式的值。

e. switch语句的功能也可以用if...else语句来实现。但switch语句会使程序更简洁，可读性更强。而if...else功能更为强大。

课堂练习：1) 写一个方法，能实现数值星期和英文星期的转换，如0会转换为Sunday, 1会转换为Monday。

```
public String switchWeekLabel(int week) {
    String result;

    switch(week) {
      case 0:
          result = “Sunday”;
          break;
      case 1:
          result = “Monday”;
          break;
      case 2:
          result = “Tuesday”;
          break;
      case 3:
          result = “Wednesday”;
          break;
      case 4:
          result = "Thursday";
          break;
      case 5:
          result = "Friday";
          break;
      case 6:
          result = "Saturday";
          break;
      default:
          result = “error”;
     }
                    return result;
}
```

4. 循环语句

循环语句的作用是反复执行一段代码，直到不满足循环条件为止。循环语句一般应包括如下四部分内容：

. 初始化部分：用来设置循环的一些初始条件，比如循环控制变量的初始值；
. 循环条件： 这是一个布尔表达式，每一次循环都要对该表达式求值，以判断到底继续循环还是终止循环。
. 循环体： 这是循环操作的主体内容，可以是一条语句，也可以是多条语句；
. 迭代部分： 用来改变循环控制变量的值，从而改变循环条件表达式的值；

Java语言提供三种循环语句：for语句、while语句和do...while语句。for语句、while语句在执行循环体之前
测试循环条件，而do...while语句在执行循环体之后测试循环条件。因此for语句、while语句有可能连一次循
环都未执行，而do...while至少执行一次循环体。

while和do.....while的循环体中,如果先写i++
最后求出的count值偏大,先写count+=i,后写i++
最后求出的count值偏小.

1) for循环
初始化部分—>循环条件—>循环体—>迭代部分—>循环条件—>循环体.....
语法：for(初始化部分；循环条件；迭代部分) {
循环体
}

```
int i=10,count=0;
do {
  count+=i;
  i++;
}while(i<10);
for(int i=1;i<10;i++) {
  System.out.println(i);
}
int i=10,count=0;
while(i<10) {
    i++;
    count += i;
}
System.out.println(count);
```

在执行for语句时，先执行初始化部分，这部分只会被执行一次；
接下来计算作为循环条件的布尔表达式，如果为true，就执行循环体；
接着执行迭代部分，然后再计算作为循环条件的布尔表达式，如此反复；

课堂练习：1) 写一方法，完成计算从1加到100的和；

```
public int sum() {
```

```
        int result = 0;
        for(int i=1;i<=100;i++) {
            result = result + i;
        }

        return result;
    }
```

2) 在练习一基础上，完成计算从1加到指定数值的和；

```
    public int sum(int n) {
        int result = 0;
        for(int i=1;i<=n;i++) {
            result = result + i;
        }

        return result;
    }
```

2) while循环

语法：[初始化部分]

```
while(循环条件) {
    循环体,包括迭代部分
}
```

当循环条件为true时，就重复执行循环，否则终止循环；

课堂练习：1) 用while循环完成计算从1加到指定数值的和；

```
    public int sum(int n) {
        int result = 0,i=1;

        while(i<=n) {
            result = result + i;
            i=i+1;
        }

        return result;
    }
```

3) do ... while循环

和while非常类似，只不过先执行循环体，然后再判断循环条件。

语法：[初始化部分]

```
do {
    循环体,包括迭代部分
} while(循环条件);
```

do。。。while都有可能会多产生一个数据(大数据),for循环不会多产生数据

当迭代部分位于循环之上,换言之.先执行迭代部分在执行循环部分会产生多一条的数据.do...while先执行再判断 while 先判断再执行

课堂练习： 1) 用do...while循环完成计算从1加到指定数值的和；

```
public int sum(int n) {
    int result = 0,i=1;

    do {
        result = result + i;
        i=i+1;
    } while(i<=n)

    return result;
}
```

5. 循环语句中流程跳转

1) break: 终止当前或指定循环;

如果break是在if语句,会直接跳出if和前一个循环.

不能跳出多个循环(终止多个循环return).如果跳出制定的循环,可以在循环方法前+ loop:

break后面不允许添加语句,添加语句会报错

break跳出制定的for循环后不会再进入到该for内部,即使满足比较条件.

e.g

```
loop:for(…) {
   for(…){
          break loop:跳出第一个for循环
          }
 }
public int sum(int n) {
    int result = 0,i=1;

    while(i<=n) {
        result = result + i;
        i=i+1;
        if(i>10)
           break;
    }

    return result;
}
```

实现1加到10;

2) continue: 跳过本次循环，执行下一次循环和break不相同.break跳出
循环后不会再进入循环.而continue当前循环的跳出不影响下一次
循环的进入,或执行标号标识的循环体。
如果跳出制定的循环,可以在循环方法前+ loop:

```
public int sum(int n) {
    int result = 0;

    for(int i=1;i<=100;i++) {
        if(i%2==0)
                continue;
        result = result + i;
    }

    return result;
}
```

实现指定范围内奇数的和；

3) label: 标号用来标识程序中的语句，标号的名字可以是任意的合法标识符。
continue语句中的标识必须定义在while、do...while和for循环语句前面；
break语句中的标识必须定义在while、do...while和for循环语句或switch语句前面；

6.Array

数组是指一组数据的集合，数组中的每个数据称为元素。在Java中，数组也是Java对象。数组中的元素可以是任意类型(包括基本类型和引用类),但同一个数组里只能存放类型相同的元素。创建数组大致包括如下步骤：

. 声明一个数组类型的引用变量，简称为数组变量；
. 用new语句构造数组的实例。new语句为数组分配内存，并且为数组中的每个元素赋予默认值；
. 初始化，即为数组的每个元素设置合适的初始值。

主要围绕以下内容展开；

. 数组的创建，包括基本类型数组的创建和类类型数组的创建；
. 访问数组的元素和长度属性；
. 创建一个数组的数组(多维数组)；

一. 数组变量的声明；

声明数组：
1) 一个存放同一类型数据的集合
a. 即可以是基本类型，也可以是对象类型；
b. 数组中的每个数据为元素；

2) 数组是一个对象，成员是数组长度和数组中的元素;
3) 申明了一个数组变量并不是创建了一个对象;
4) 申明数组的方式;

int[] IArray 或者 int IArray[]　基本数据类型数组，数组中存放的是基本数据类型。

Teacher[] tArray 或者 Teacher tArray[]　类数组，数组中存放的是Teacher类创建的若干个的对象。

注意：1) 声明数组变量的时侯，不能指定数组的长度，以下声明方式是非法的。

```
int x[1];
int[2] x;
```

二. 初始化

初始化：自变量创建后首次赋值的过程；

静态初始化：初始化时由程序员显式指定每个数组元素的初始值，由系统决定数组长度，如：

```
//只是指定初始值，并没有指定数组的长度，但是系统为自动决定该数组的长度为4
String[] computers = {"Dell", "Lenovo", "Apple", "Acer"};　　//缩写
//只是指定初始值，并没有指定数组的长度，但是系统为自动决定该数组的长度为3
String[] names = new String[]{"多啦A梦", "大雄", "静香"};　　//全写
```

动态初始化：初始化时由程序员显示的指定数组的长度，由系统为数据每个元素分配初始值，如：

```
//只是指定了数组的长度，并没有显示的为数组指定初始值，但是系统会默认给数组数组元素分配初始值为null
String[] cars = new String[4];
```

1. 创建数组对象；

数组对象和其他Java对象一样，也用new语句创建；

```
int[] iArray = new int[2];
```

new语句执行以下步骤：

a. 在堆区中为数组分配内存空间，以上代码创建了一个包含2个元素的int数组；每个元素都是int类型，占4个字节，因此
整个数组对象在内存中占用8个字节。

b. 为数组中的每个元素赋予其数据类型的默认值。

byte/short/int/long	0
float	0.0f
double	0.0d
String	null
char	空或者'\u0000'
boolean	false

c. 返回数组对象的引用

在用new语句创建数组对象时，需要指定数组长度。数组长度表示数组中包含的元素数目。数组长度可以用具体的数值表示，也可以用变量表示。如：

```
int[] x = new int[10];
或
int size=10;
int[] x = new int[size];
```

数组的长度可以为0，此时数组中一个元素也没有。例如：

```
int[] x = new int[0];
```

对于Java类的程序入口方法main(String args[]),如果运行时这个类没有输入参数，那么main()方法的参数args并不是null, 而是一个
长度为0的数组。例如：

```
public class Sample {
public static void main(String[] args) {
        System.out.println(args.length);          //打印0
}
}
```

数组对象创建后，它的长度是固定的。数组对象的长度是无法改变的，但是数组变量可以改变所引用的数组对象。

```
int[] x = new int[3];
int[] y = x;
x = new int[4];
```

ArrayIndexOutOfBoundsException 是运行时异常(编译时不报错)
数组越界:表示使用的数组下标超过了数组的最大下标,数组的
最大下标是数组长度－1 (length-1),这个错出现在运行阶段,编译阶段不报错.不是checkexception

2. 初始化数组对象；

数组中的每个元素都有一个索引，或者称为下标。数组中的第一个元素的索引为0，第二个元素的索引为1, 依次类推。
通过索引可以访问数组中的元素或者给数组中元素内容赋值。

1) 声明、创建、初始化分开：

```
int[] iArray;
iArray = new int[2];
iArray[0] = 0;
iArray[1] = 1;
```

2) 声明、创建的同时并初始化数组；

```
int[] iArray = {0, 1};
Student sArray[] = new Student[] { new Student(“George”, “Male”, 20), new Student()};
```

合法:Student[] stArray = { new Student(), new Student()} ;

注意：a. 非法的数组初始化方式：

```
int[] x = new int[5]{5,4,3,2,1};          //编译出错，不能在[]中指定数组的长度；
```

```
int[] x;
x = {5,4,3,2,1};                          //{5,4,3,2,1}必须在声明数组变量的语句中使用，不能单独使用
```

3. 多维数组；

Java支持多维数组。假定某个宾馆有三层楼，第一层有4个房间，第二层有3个房间，第三层有5个房间。某一天客户人住宿情况如下图所示:

```
第三层:          |    |Tom |Jerry|    |Rose|
第二层：          |Mary|    |Kevin|
第一层：          |Mike|Jane|Duke |    |
```

可以用两维数组来存储各个房间的客人信息。

```
String[][] room = new String[3][];
room[0] = new Stirng[]{"Mike","Jane","Duke",null};
room[1] = new String[]{"Mary",null,"kevin"};
room[2] = new String[]{null,"Tom","Jerry",null,"Rose"}
```

以上代码等价于：

```
String[][] room = {{"Mike","Jane","Duke",null},
        {"Mary",null,"kevin"},
        {null,"Tom","Jerry",null,"Rose"}};
```

几乎所有的程序设计语言都支持数组。Java也不例外。当我们需要多个类型相同的变量的时候，就考虑定义一个数组。在Java中，数组变量是引用类型的变量，同时因为Java是典型的静态语言，

因此它的数组也是静态的，所以想要使用就必须先初始化（为数组对象的元素分配空间）。

二维数组需要注意:
如果只是确定了二维数组中的第一个一维数组,第二个一维数组没有确定长度.此时不能直接使用二维数组一个一个赋值.只能在第一个一维
数组中来制定另一个一维数组值
e.g
String[][] str = new String[3][];
str[0][0] = “hello”;报错 空指针
str[0] = new String[]{“hello”};正确

4. 数组的边界：

1) 一个数组的下标从0开始，数组通过数组的对象引用变量的下标访问数组。

数组中第一个元素的索引为0, 第二元素的索引为1，依次类推。如果一个数组长度是5，要访问最后一个数组元素可以通过
下标4来访问，如果通过下标5访问，超出了数组的边界，在运行时会抛出ArrayIndexOutOfBoundsException。

2) 通过调用数组的length方法可以获得一个数组的元素个数（数组长度）。

所有Java数组都有一个length属性，表示数组的长度. 该属性只能读取，但是不能修改。

以下修改数组的length属性，这是非法的。

```
int[] x = new int[4];
x.length = 10;          //编译出错，length属性不能被修改。
```

注：a. 数组变量必须在引用一个数组对象之后，才能访问其元素。

```
public class Sample {
private int[] x;
public static void main(String[] args) {
        Sample s = new Sample();
         System.out.println(s.x);          //打印null
        System.out.println(s.x[0]);       //运行时抛出
```

NullPointerException

```
                                        System.out.println(s.x.length);     //运行时抛出
NullPointerException
                                }
                                }
```

课堂练习：1) 带着一起练习: 求一组值的平均值：

```
public class ArrayAvgTest {
    public double avg(int[] n) {
        double result = 0.0;
        for(int i=0;i<n.length;i++) {
            result += n[i];
        }
        result /= n.length;

        return result;
    }

    public static void main(String[] args) {
        ArrayAvgTest a = new ArrayAvgTest();
        int[] n = {100,60,80,90,75,38};
        System.out.println("Avg score: "
                                + a.avg(n));
    }
}
```

2) 自行练习: 求一组值的最大值：

```
public class ArrayMaxTest {
    public int max(int[] n) {
        int result = n[0];
        for(int i=1;i<n.length;i++) {
            if(result<n[i])
                result = n[i];
        }

        return result;
    }

    public static void main(String[] args) {
        ArrayAvgTest a = new ArrayAvgTest();
        int[] n = {100,60,80,90,75,38};
        System.out.println("Max score: " + a.max(n));
    }
}
```

3) 带着练习：数组内容排序

冒泡排序：值较小的数逐渐向数组的顶部(即朝第一个元素)冒上来，就像水中的气泡上升一样，同时，值较大的数据逐渐

向数组的底部(即朝最后一个元素)沉下去。这种算法用嵌套的循环对整个数组进行数次遍历，每次遍历都要比

较数组中相邻的一对元素，如果这对元素以升序(或者值相等)的顺序排列，那么保持它们的位置不变；如果这
对元素以降序的顺序排列，那么交换它们的值。

数组原内容：100,60,80,90,75,38
38 60 75 80 90 100
冒泡排序
第一个for循环:排序n个值,需要取几个值进行操作
至少n-1个,可以是n个

```
for(int i=0;i<iarray.length-1;i++) {
        for(int j=0;j<iarray.length—1-i;j++) {

        }
}
```

第二for循环:取中间的某一个数,至少需要比较多少次
换位操作需要第三方变量 int temp
temp = 100
100 = 60
60 = temp

	100	60	80	75	38	90	
第一次循环：	60	80	90	75	38	100	5-0
第二次循环：	60	80	75	38	90	100	5-1
第三次循环：	60	75	38	80	90	100	5-2 length-1-i
第四次循环：	60	38	75	80	90	100	5-3
第五次循环：	38	60	75	80	90	100	5-4

```
public class ArraySortTest {

        public void sort(int[] n) {
                for(int i=0;i<n.length-1;i++) {
                        for(int j=0;j<n.length-i-1;j++) {
                                if(n[j]>n[j+1]) {
                                        int temp = n[j];
                                        n[j] = n[j+1];
                                        n[j+1] = temp;
                                }
                        }
                        print(n);
                }
        }

        public void print(int[] n) {
                for(int i=0;i<n.length;i++)
                        System.out.print(n[i] + "\t");
                System.out.println();
        }
```

```
		public static void main(String[] args) {
			ArraySortTest s = new ArraySortTest();
			int[] n = {100,60,80,90,75,38};
			s.sort(n);
			s.print(n);
		}
	}
```

4) 在练习三的基础上，引入java.util.Arrays辅助类的介绍。介绍其sort(int[] n)以及binarySearch(int[] n,int key)
方法的使用。
java.lang.*
java.util.*
java.io.*
注意：a. 这二个方法均为static方法，可直接通过类名使用；
b. binarySearch方法必须保证数组中的元素已经按照升序排列，这样才能得到正确的结果；

```
	import java.util.Arrays;

	public class ArraySortTest2 {

		public void print(int[] n) {
			for(int i=0;i<n.length;i++)
				System.out.print(n[i] + "\t");
			System.out.println();
		}

		public static void main(String[] args) {
			ArraySortTest2 s = new ArraySortTest2();
			int[] n = {100,60,80,90,75,38};
			Arrays.sort(n);
			System.out.println(Arrays.binarySearch(n, 80));
			s.print(n);
			System.out.println(Arrays.binarySearch(n, 80));
		}
	}
```

如果找到了目标，（对象在数组可以被查询到）Arrays.binarySearch()
产生的返回值就大于或等于0。表示传入的数位于已经拍好序(升序)的数组中的索引号
否则，它产生负返回值，表示若要保持数组的排序状态此元素所应该插入的位置。这个负值的计算方式是：
-（插入点）-1： 插入点表示该字段插入到排好序的数组的中的位置的索引号

---

5) 带着练习 从众多手机号码中抽取一个获奖手机号码

```
	public class ArrayRandomTest {
		public String getTel(String[] n) {
			int index = (int)(Math.random()*n.length);
			//Math.random() [0,1)*4  [0,4) 0 1 2 3
```

```
                //获取随机索引号,可以使用Math.random()*该数组的长度
                //得到的结果取整数,就是需要的随机索引号
                return n[index];
        }

        public static void main(String[] args) {
                ArrayRandomTest a = new ArrayRandomTest();
                String[] n = { "1318259016",
"13564560540","13858687810","13999999999"};
                System.out.println(a.getTel(n));
        }
}
```

5. 数组的拷贝：

数组的长度一旦确定之后便不能调整，我们可以通过复制数组的内容变通实现改变数组长度。在System类中提供一个辅助的arraycopy方法提供复制数组内容的功能：

```
public static void arraycopy(Object src,
                int srcPos,
                Object dest,
                int destPos,
                int length)
```

srcPos和destPos表示的是对应的索引号，
srcPos表示源数组中要复制的元素的起始位置
descPos表示目标数组中需要复制到的目标地点的起始位置
src - the source array.
srcPos - starting position in the source array.
dest - the destination array.
destPos - starting position in the destination data.
length - the number of array elements to be copied.