

## 二、第一个程序 HELLO WORLD

### (一)、代码

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

### (二)、面向对象语言

Java 是一种面向对象的语言:

每个 Java 文件都必须包含一个类声明

所有的代码都写在一个类里面

运行一个 Java 程序，需要定义一个 main 方法

Java 的语言特点：

大括号/花括号用于表示代码块/代码段的开始和结束

语句以分号结束

违反上面的一条会导致语法错误

```
sum = sum + i  
i = i + 1
```

```
    end  
output sum
```

## 三、变量

### (一)、静态类型

Java 是一种静态类型的语言:

1.所有变量、参数和方法都必须有一个声明的类型(也称为静态类型)。

2.变量必须在使用前声明。

3.表达式也有一个类型，例如  $5+10$  的类型为 int

Java 编译器确保类型一致性：

1.如果类型不一致，程序将无法编译

2.使用 IDE，编译器会在程序运行之前检查程序中的所有类型是否兼容

### (二)、声明、初始化、分配

变量可以被声明和初始化一次，并可以被分配多次。

```
public class Hello {  
    public static void main(String[] args) {  
        int num1 = 5;  
        int num2;  
        num2 = 10;  
        int num = num1 + num2;  
        System.out.println(num);  
    }  
}
```

### (二)、转义字符

通过添加反斜杠 \ 符号来转义特殊字符。

## 一、数据类型

### (一)、整数类型

byte 大小 8 bit 取值范围 -128 to 127

short 大小 16 bit 取值范围 -32,768 to 32,767

int 大小 32 bit 取值范围 -2^31 to 2^31 - 1

long 大小 64 bit 取值范围 -2^63 to 2^63 - 1

### (二)、小数类型

float 大小 32 bit 取值范围 +-10^38

double 大小 64 bit 取值范围 +-10^308

这些都可能有点不准确

可能会出现舍入错误

0.8 可能变成 0.7999999999999999

### (三)、布尔类型

boolean 类型只能存放 true 或者 false

### (四)、字符类型

char 类型只能存放单个字符, 比如

char ch = 'a';

只能使用单引号

### (五)、字符串类型

字符串类型需要使用双引号, 比如

String s = "aa";

字符串其实是由字符加起来的。

## 二、运算符号

### (一)、算数运算符

+ 加法运算符(也用于连接字符串)

- 减法运算符

\* 乘法运算符

/ 除法运算符

% 取模运算符 (取余数)

Java 中的算数运算需要看运算符两边的数据类型, 如果都是整数类型, 那么结果必定是整数类型, 这个其实是根据数据类型占据内存的大小来决定的, 计算结果是看占用内存大的数据类型。

### (二)、一元运算符

++ (+1)

-- (-1)

当自加、自减出现的时候, 需要给它们所在的那个变量空间中加 1, 或者减 1.

自加、自减运算:

自加自减运算符, 称为一元运算符。

表现形式:

1) ++ 或者 -- 在变量的右侧, 举例: i++, j--;

2) ++ 或者 -- 在变量的左侧, 举例: ++i, --j;

自加自减运算规律:

1) 当自加自减在变量的右侧时, 需要先把变量空间中的值临时存储, 也就是说在内存中需要开辟一个临时空间, 把这个值保存到临时空间中。然后给原来变量空间中 +1 或 -1, 最后把临时存储的那个数据和其他的运算符号进行运算。

2) 当自加自减在变量的左侧时, 这时直接给变量空间 +1 或 -1, 然后把空间中的值与其他的运算符号进行运算。

## 三、MATH

### (一)、MATH 类中的方法

1. 平方根: Math.sqrt(num);

2. 两个数的最大值 Math.max(num1,num2);

### 3. 随机数 Math.random()

常用的如下图：

Method	Description	Return Type
acos(x)	Returns the arccosine of x, in radians	double
asin(x)	Returns the arcsine of x, in radians	double
atan(x)	Returns the arctangent of x between -PI/2 and PI/2 radians	double
exp(x)	Returns the value of E <sup>x</sup>	double
expm1(x)	Returns e <sup>x</sup> -1	double
floor(x)	Returns the value of x rounded down to its nearest integer	double
hypot(x, y)	Returns sqrt(x <sup>2</sup> +y <sup>2</sup> ) without intermediate overflow or underflow	double
log(x)	Returns the natural logarithm (base E) of x	double
max(x, y)	Returns the number with the highest value	double   float   int   long
min(x, y)	Returns the number with the lowest value	double   float   int   long
pow(x, y)	Returns the value of x to the power of y	double
random()	Returns a random number between 0 and 1	double
round(x)	Returns the value of x rounded to its nearest integer	int
signum(x)	Returns the sign of x	double
sin(x)	Returns the sine of x (x is in radians)	double
sqr(x)	Returns the square root of x	double
tan(x)	Returns the tangent of an angle	double
toDegrees(x)	Converts an angle measured in radians to an approx. equivalent angle measured in degrees	double
toRadians(x)	Converts an angle measured in degrees to an approx. angle measured in radians	double

## 四、SCANNER 类

### (一)、SCANNER 类中的方法

创建 Scanner 类对象：

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

常用方法：

next(); 返回单个字符

nextLine(); 返回一行字符串

nextInt(); 返回一个 int 类型数据

nextDouble(); 返回一个 double 类型的数据

## 五、基本数据类型、包装类、字符串类型直接的相互转换

基本数据类型都有自己对应的包装类

- 3. != 表示非
- 4. == 表示相等
- 5. != 表示不等于
- 6. > 表示大于
- 7. >= 表示大于等于
- 8. < 表示小于
- 9. <= 表示小于等于

## 二、IF

### (一)、第一种定义方式

```
if( 判断的条件 ){  
    判断成立后应该执行的动作;  
}
```

注意：这里的判断条件最后必须是一个 boolean 值,如果判断条件是 true，则执行 if 后面的{}中的内容；如果判断条件是 false，则跳过大括号中的内容，向下继续执行其他内容。

### (二)、第二种定义方式

```
if( 判断的条件 )注意：这里的判断条件最后必须是一个 boolean 值,如果判断条件是  
true，则执行 if 后面的{}中的内容；如果判断条件是 false，则跳过大括号中的内容，执行  
else 后面的大括号中的内容。
```

```
{  
    判断条件成立后执行的语句;  
}  
else //注意：只有当 if 的条件不成立时，才会执行 else  
{  
    判断不成立后执行的语句;  
}
```

执行顺序：当 JVM 遇到 if else 结构的判断时，会先执行 if 的条件。如果 if 条件成立就执行 if 后面的语句，如果 if 的条件不成立，就自动的去执行 else 的语句。

注意：如果 if 成立了，执行完 if 后面的语句之后，直接跳过 else，不会执行 else 后面大括号中的内容，而是往下继续执行。

### (三)、第三种定义方式

当在程序中有多个条件的时候，每个 if 只能列出其中的一种可能条件，这时在 else 中就会隐含其他的所有条件，对剩余的条件，还要进行更加具体的细化，这时可以在 else 后面继续使用 if，区分出其他的各种情况

```
if( 判断的条件 ){
    逻辑代码;
} else if( 判断的条件 ){
    逻辑代码;
} else if( 判断的条件 ){
    逻辑代码;
}
...
else{
    上述的所有判断都不成立之后执行的语句;
}
```

**if (条件表达式1)**  
{  
    //语句1  
} **else if(条件表达式2) {**  
    //语句2  
} **else if(条件表达式3) {**  
    //语句3  
} **else{**  
    //最后的语句  
}

**多支if语句的执行过程：**  
首先，判断条件表达式1，  
条件成立(true)，则执行语句1，其它后续的else if判断和else都不会执行；  
条件不成立(false)，则向下继续判断条件表达式2。条件表达式2也存在两种情况：条件成立，则执行语句2，后续判断不执行；  
条件不成立  
如果条件表达2也为false，则向下继续判断条件表达式3（也有两种情况）  
**注意：else语句必须书写在多支if语句的最后**

### (一)、格式

```
while( 循环条件 )
{
    循环条件成立后要执行的语句; // 循环体
}
```

if 和 while 的区别：

```
if( 判断条件 )
{
    判断成立后执行的语句;
}
```

while 循环的执行顺序：

当 JVM 在执行程序的过程中，遇到 while 关键字，这时也会先执行 while 身后小括号中的表达式，然后根据表达式的真假确定是否执行 while 后面大括号中的代码。

当 while 的条件是 true 的时候，JVM 就会去执行 while 身后大括号中的语句，当把大括号中的所有语句执行完之后，这时 JVM 会返回到 while 的循环条件地方继续判断。直到 while 的条件为 false 的时候，这时这个循环才能结束，继续往下执行和循环并列的语句，如果这里的循环永远不能结束，程序就在这里一直循环。

## LEC4: FOR、数组、STATIC METHOD 1

### 一、FOR

#### (一)、格式

```
for(循环初始值 ; 循环条件 ; 修改循环条件){
    //重复执行的代码----》循环体。
}
```

在 for 循环的小括号中需要三个表达式，三个表达式之间需要使用英文分号隔开。

循环初始值：一般是一个定义赋值的表达式。可以省略

循环条件：for 循环的条件表达式，循环条件最后计算的结果必须是一个 boolean 值。可以省略，默认是 true

修改循环条件：一般是循环变量的更新表达式。可以省略

for 循环的执行过程：

```
for(循环初始值；循环条件；循环条件修改)
{
    //循环体代码
}
```

注意：循环初始值只会执行一次。  
后续for循环的执行都是  
在编号2、编号3、编号4之间循环执行

for循环的执行流程：  
首先，执行循环初始值；  
接着，判断循环条件  
    条件为true：执行循环体代码。  
    条件为false：for循环不会执行。执行for循环后面的代码  
当循环体中的代码执行完后，会执行循环条件的修改  
循环条件修改完成之后，会再次判断循环条件

## 二、嵌套循环

### (一)、定义格式

嵌套 for 循环的格式：

```
for(循环初始值,循环条件;循环条件修改) { //外层循环
```

```
    for(循环初始值;循环条件;循环条件修改) { //内层循环
```

        //重复执行的代码

}

注意：在开发中，通常嵌套 for 循环就是两层。（不建议书写超过两层以上的嵌套循环）

### (二)、嵌套演示

```
/*
    循环嵌套
*/
class ForForDemo {
    public static void main(String[] args) {
        //外层循环
        for (int i=1;i<=2;i++){
            //内层循环
            for (int j=1;j<=2;j++){

```

## 三、数组

### (一)、数组的概念

数组：数组表示的一串连续的存储空间。每个空间中都可以保存一个数据。当我们需要操作的时候，不要去面对单个空间，而直接面对这个整体的连续的存储区域。

### (二)、格式

格式：

数组中保存的数据类型[] 数组的名=new 数组中保存的数据类型[开辟的连续空间个数]；

把上述定义的格式可以翻译成下述格式：

数据类型[] 数组名 = new 数据类型[长度]；

```
int[] arr = new int[4];
JVM在执行上述代码的时候，会先执行 赋值号右侧的表达式
```

由于赋值号右侧是new 关键字，这时JVM会自动的去创建新的空间，并且创建的新空间的类型为int 类型，而具体创建的空间个数由中括号中指定的个数



开辟好空间之后，给这个整体起名为arr

格式：在定义数组的时候，如果已经知道数组中的具体数据，这时可以在定义数组的时候直接书写具体的数据。

数据类型[] 数组名 = {值，值，值.....}；

好处：

- 1、可以一次性开辟出更多的空间。
- 2、可以对多个空间进行统一的管理。
- 3、数组定义好之后，每个数组的空间中都有一个唯一的编号（索引 index，下标，角标）。我们在操作数组的时候，需要通过该数组的统一的名称和对应的下标来操作数组中每个空间。
- 4、数组的下标是从 0 开始，到长度-1 结束。如果操作的下标不再这个范围内，程序报错。
- 5、在数组中有个属性 length，可以获取到当前数组的长度。

注意：

数组也可以按照如下方式定义：

数据类型 数组名[] = new 数据类型[长度];

例如： double arr[] = new double[6];

#### (四)、数组的操作

我们需要通过数组的总名称 引用变量空间，然后通过这个引用空间中的地址，再去找到堆内存中的连续的空间，然后才能准确的去操作数组。

获得数组中每个空间中的值：

数组名[下标]:

给数组的每个空间赋值：

数组名[ 下标 ] = 值;

我们定义的数组名空间：

它是一个特殊的内存空间，它中只能保存某个数字在堆中的内存地址。而不能像前面学习普通变量一样来保存普通的常量。

数组名空间保存某个数组的地址，而数组的每个真实空间保存具体的普通常量数据。

#### (五)、遍历数组

数组的遍历：

遍历：把数组的所有空间都访问一遍。

访问数组的空间需要使用：数组名+数组的下标，才能访问到数组的每个空间。

同一个数组，名称是统一的，只有下标。而下标是从零开始，到长度-1 结束，下标是在有序的增加。

```
例:public class TestMain {  
    public static void main(String[] args){  
        int[] arr={1,2,12,4,5,6,-1,0};  
        for (int i = 0; i < arr.length; i++) {  
            System.out.println(arr[i]);  
        }  
    }  
}
```

#### 四、METHOD

##### (一)、函数的定义

函数：也可以称为方法。它表示的是一段可以独立运行的代码，具有独立功能。当在程序中需要使用的时候可以通过函数（方法）的名字去调用。  
把这段代码称为一个函数（方法）。

它的书写位置：它必须在类的大括号中，不能写在 main 方法里面，要和 main 方法在关系上属于并列关系，我们自己定义的函数（方法）和 main 方法没有先后次序。

格式：

```
函数的修饰符 函数的返回值类型 函数的名( 接收的参数类型 参数名 , 接收的参数类型  
参数名 ....){  
    函数体代码；//具体完成相应功能的代码。  
    return 返回值；//返回值的类型必须是之前定义函数时所声明的函数的返回值类型  
}
```

##### (二)、定义函数的细节

1) 函数的修饰符：用来修饰函数的一些符号(关键字)。目前针对函数的修饰符，我们先照抄 main 方法。

2) 函数的返回值类型：当我们程序封装了一段独立的代码之后，如果这段代码运行完，需要把一个具体的结果返回给调用者，这时当前这个函数的返回值类型就需要和返回的那个数据保持一致。如果这个函数调用完之后，不需要给调用者返回任何数据，这时函数的返回值类型必须写 void。

注意：如果需要返回，则用 return 关键字。

3) 函数的名：它就是前面学习的标识符。我们给独立代码命名需要遵守标识符的规则。就是封装的功能名字。(开发人员自己起名称)。

4) 参数列表：当定义一个功能的时候，需要接受调用者传递进来的数据时，就需要在定义函数的时候，书写对应的参数。

参数：参加运算的数据。

参数的类型：就是前面学习的变量的类型。

参数名：其实就是变量名。

如果不需要接受参数，这时小括号中的参数可以省略，但是小括号不能省略。

5) 函数体：就是在函数的大括号中书写代码，这里的代码和以前学习时在 main 方法中写的代码没有区别，以前在 main 方法中可以写的任何代码，都可以在自己定义的函数体中书写。

原则：在书写代码的时候，每段代码如果是一个独立的功能，这时需要把这些代码单独的抽离到一个函数中，不要把所有代码全部书写在 main 方法中。

### (三)、例题

数组求和

```
class ArrayTest2{  
    public static void main(String[] args){  
        //定义数组  
        int[] arr={1,3,4,5,6};  
        //调用自定义的函数  
        int sum=getArraySum(arr);  
        //在屏幕上打印数组中所有数据的和  
        System.out.println("sum="+sum);  
    }  
    //定义函数来计算数组中所有数据的和  
    /*  
     * 1.有没有返回值?  
     *     有，返回的是数组中所有数据的和  
     * 2.有没有参数?  
     *     有，具体的数组  
    */  
    public static int getArraySum(int[] arr){  
        //定义一个变量接收数组中所有数据的和  
        int sum=0;  
        //遍历取出数组中每个空间中的数据  
  
        for (int i=0;i<arr.length ;i++ ){  
            //将数组中的每个数据和变量 sum 相加  
        }  
    }  
}
```

16

CPT111 Java Programming

三

```
        sum=sum+arr[i];  
    }  
    //将数组的数据和 sum 返回给调用者  
    return sum;  
}  
}
```

求数组中的最大值

```
class ArrayTest3 {  
    public static void main(String[] args) {  
        //定义数组  
        int[] arr={1,3,4,5,6};  
        //调用自定义函数  
        int max=getArrayMax(arr);  
        System.out.println("max="+max);  
    }  
    /*  
     * 定义一个函数  
     * 1.有没有返回值  
     *     有，将数组中最大的值返回给调用者  
     * 2.有没有参数  
    */  
}
```

有，固定数组

```
/*
public static int getArrayMax(int[] arr){
    //定义一个变量，存放最值
    int max=arr[0];
    //使用循环取出数组中空间的数据和假设的最大值进行比较
    for (int i=0;i<arr.length;i++){
        //使用判断语句进行判断，如果取出的数据大于 max 中保存的数据
        //则将取出的数据赋值给 max
        if (max<arr[i]){
            max=arr[i];
        }
    }
    //比较结束后将最大值 max 返回给调用者
    return max;
}
```

### (一)、定义

成员变量：直接定义在类中的变量。

局部变量：定义在类中局部位置（函数中）的变量。

```
class Demo1
{
    int a = 100;//成员变量

    void show(){
        int b = 1000;//局部变量

        System.out.println(a + " -- " + b);
    }
}
```

### (二)、区别

1、从定义的位置上：

- a) 局部变量：函数里面
- b) 成员变量：直接定义在类中

2、从内存上看：

- a) 局部变量：栈内存中
- b) 成员变量：堆内存中

3、从生命周期（指的是从分配空间到回收空间）上看：

- a) 局部变量：随着函数进栈执行，开始分配空间；函数执行结束出栈，空间被回收
- b) 成员变量：随着创建对象开始分配空间；随着对象的空间变成垃圾空间被回收而被回收；

4、从使用范围上：

- a) 局部变量：仅限于定义它的局部范围里面（比如函数中）
- b) 成员变量：整个类中

5、从初始值上看：

- a) 局部变量：没有默认值，使用前必须先要初始化
- b) 成员变量：都有默认值；使用前可以先赋值，也可以不用赋值

## 二、函数重载

## (一)、重载概念

重载：函数的重载，要求必须在同一个类（程序）中，有多个同名的函数，它们的参数列表不同，这时我们称为函数的重载。

注意：

- 1、要求必须在同一个类中（程序）。
- 2、要求函数的名称必须相同
- 3、要求函数的参数列表必须不同。

参数列表不同，主要针对参数的个数，类型，顺序不同。

由于函数可以在同一个类出现重载的现象，因此在调用的时候，具体应该执行哪个函数，需要根据调用者传递的实际参数决定。

结论：重载就是在一个类中具有相同名称的函数，但是这些相同函数的参数列表不能相同。

## 三、STRING

### (一)、创建 STRING

String 类代表字符串。Java 程序中的所有字符串字面值（如 "abc"）都作为此类的实例实现。

字符串是常量；它们的值在创建之后不能更改。字符串缓冲区支持可变的字符串。因为 String 对象是不可变的，所以可以共享。例如：

String str = "abc";

在 Java 中，所有用英文的双引号括起来的内容，都是字符串，都是 String 类的对象

就表示创建一个对象，对象的字面值，就是 abc；

等效于：

```
char data[] = {'a', 'b', 'c'};  
String str = new String(data);
```

下面给出了一些如何使用字符串的更多示例：

```
System.out.println("abc");  
String cde = "cde";  
System.out.println("abc" + cde);  
String c = "abc".substring(2, 3);  
String d = cde.substring(1, 2);
```

构造函数

String(byte[] bytes)

通过使用平台的默认字符集解码指定的 byte 数组，构造一个新的 String。

String(byte[] bytes, Charset charset)

通过使用指定的 charset 解码指定的 byte 数组，构造一个新的 String。

String(byte[] bytes, int offset, int length)

通过使用平台的默认字符集解码指定的 byte 子数组，构造一个新的 String。

String(byte[] bytes, int offset, int length, Charset charset)

通过使用指定的 charset 解码指定的 byte 子数组，构造一个新的 String。

1、根据字符数组创建字符串对象

String(char[] value)  
分配一个新的 String，使其表示字符数组参数中当前包含的字符序列。

String(char[] value, int offset, int count)  
分配一个新的 String，它包含取自字符数组参数一个子数组的字符。

```
10  private static void test3() {  
11      char[] chs = {'a', 's', 'd', 'f', 'g', 'h'};  
12      String s1 = new String(chs);  
13      String s2 = new String(chs, 1, 4);  
14      System.out.println(s1); // "asdfgh"  
15      System.out.println(s2); // "sdfg"  
16  }
```

Problems Javadoc Declaration Console  
<terminated> Demo1 (9) [Java Application] E:\java\bin\javaw.exe (2016年3月25日 下午3:18:23)  
asdfgh  
sdfg

2、根据 int 数组创建字符串对象

String(int[] codePoints, int offset, int count)  
分配一个新的 String，它包含 Unicode 代码点数组参数一个子数组的字符。

```
11  private static void test4() {  
12      String str = new String(new int[]{99, 100, 101, 24386}, 0, 4);  
13      System.out.println(str);  
14  }
```

```
10 private static void test2() {  
11     String str = "天目将军真帅！";  
12     char ch = str.charAt(5);  
13     System.out.println(ch);  
14 }
```

Problems Javadoc Declaration Console  
<terminated> Demo2 (5) [Java Application] F:\java\bin\javaw.exe (20)  
帅

#### 6、获取指定字符（或字符串）在字符串中出现的位置

int <a href="#">indexOf(int ch)</a>	返回指定字符在此字符串中第一次出现处的索引。
int <a href="#">indexOf(int ch, int fromIndex)</a>	返回在此字符串中第一次出现指定字符处的索引，从指定的索引开始搜索。
int <a href="#">indexOf(String str)</a>	返回指定子字符串在此字符串中第一次出现处的索引。
int <a href="#">indexOf(String str, int fromIndex)</a>	返回指定子字符串在此字符串中第一次出现处的索引，从指定的索引开始。

```
12  
13 private static void test5() {  
14     String str = "天目将军真帅真帅真帅真帅真帅真帅！";  
15     int index = str.indexOf("非常帅"); //如果不存在，就返回-1  
16     System.out.println(index);  
17 }
```

Problems Javadoc Declaration Console  
<terminated> Demo2 (5) [Java Application] F:\java\bin\javaw.exe (2016年3月25日 下午3:59:05)  
-1

int <a href="#">lastIndexOf(int ch)</a>	返回指定字符在此字符串中最后一次出现处的索引。
int <a href="#">lastIndexOf(int ch, int fromIndex)</a>	返回指定字符在此字符串中最后一次出现处的索引，从指定的索引处开始进行反向搜索。
int <a href="#">lastIndexOf(String str)</a>	返回指定子字符串在此字符串中最右边出现处的索引。
int <a href="#">lastIndexOf(String str, int fromIndex)</a>	返回指定子字符串在此字符串中最后一次出现处的索引，从指定的索引处开始反向搜索。

```
14  private static void test6() {  
15      String str = "天目将军真帅真帅真帅真帅真帅！";  
16      int index = str.lastIndexOf('帅');//如果不存在，就返回-1  
17      System.out.println(index);  
18  }
```

15

Problems Javadoc Declaration Console <terminated> Demo2 (5) [Java Application] F:\java\bin\javaw.exe (2016年3月25日 下午4:01:18)

#### 7、获取字符串中的子串（截取字符串）

String **substring**(int beginIndex)  
返回一个新的字符串，它是此字符串的一个子字符串。  
String **substring**(int beginIndex, int endIndex)  
返回一个新字符串，它是此字符串的一个子字符串。

```
14  private static void test7() {  
15      String str = "天目将军真帅真帅真帅真帅真帅！";  
16      String s = str.substring(4);  
17      System.out.println(s);  
18      s = str.substring(4,8);//从第一个参数表示的索引开始，到第二个参数表示的索引-1处为止  
19      System.out.println(s);  
20  }
```

真帅真帅真帅真帅真帅！  
真帅真帅

Problems Javadoc Declaration Console <terminated> Demo2 (5) [Java Application] F:\java\bin\javaw.exe (2016年3月25日 下午4:07:04)

#### 8、String 类保存数据的原理

private final char value[]; 在字符串里面，就是使用一个char数组保存字符数据

```
15  private static void test8() {  
16      String str = "天目将军真帅真帅真帅真帅真帅！";  
17      for (int i = 0; i < str.length(); i++) {  
18          System.out.print(str.charAt(i)+" ");  
19      }  
20  }
```

### (二)、构造方法

概念：构造函数又叫做构造器，就是在创建对象的同时由 JVM 调用的函数；

作用：在创建对象的同时给对象的成员变量赋值；

写法：

修饰符：可以使用访问权限修饰符（public; private;），不能使用 static 修饰；  
返回值类型：没有返回值类型，连 void 都没有；  
函数名：必须和类名完全一致；  
参数列表：构造函数可以重载，参数列表根据具体的需求而定；  
return 语句：构造函数也是通过 return 语句出栈，所以构造函数中也有 return 语句，一般的都不写；

调用：在创建对象时写在 new 关键字后面，由 JVM 自动调用；  
执行：在创建对象时，JVM 会先在堆内存中开辟空间，然后为成员变量赋默认值，然后根据 new 关键字后面书写的内容调用相应的构造函数进栈执行；  
默认构造函数：所有类中都至少有一个构造函数；如果没有写，编译器会帮我们添加一个没有方法体的无参构造函数，这个构造函数就是默认构造函数；如果在类中写了构造函数，编译器就不会再帮我们添加；  
构造函数和一般函数之间的调用：构造函数可以调用一般函数，一般函数不能调用构造函数；

构造函数之间的相互调用：构造函数可以调用其他构造函数，通过 this 关键字实现；

通过 this 调用其他构造函数时必须写在一个构造函数的第一行；

构造函数调用其他构造函数，不能形成交叉调用；

### (三)、THIS

this: 表示指向自身所在对象的引用;

哪个对象调用了 this, this 就指向那个对象;

this 的应用:

- 1.通过 this 可以调用本类其他的构造函数;
- 2.可以区分成员变量和局部变量;

```
class ThisDemo2{  
    int a = 100;  
    void show(){  
        int a = 10;  
        System.out.println(a);  
        System.out.println(this.a);  
    }  
}  
class Test  
{  
    public static void main(String[] args)  
    {  
        new ThisDemo2().show();  
    }  
}
```

## 二、封装

### (一)、PUBLIC、PRIVATE

java 中的访问权限有四类:

private: 私有的, 表示只能在本类中使用

默认的 (包级访问), 不使用任何修饰符, 表示只能在同一个包中访问;

protected: 受保护的, 主要给有继承关系的类使用; 除了同一个包中可以访问, 不同的包中, 有继承关系也可以访问;

public: 公共的, 表示在所有地方都可以使用;

java 访问权限表: (yes: 可以访问; no: 不可访问)

	private (私有的)	默认的 (什么都 不写)	protected (受保 护的)	public (公共的 )
同一个类中	yes	yes	yes	yes
同一个包中不同 类之间	no	yes	yes	yes
不同包中有继承 关系的类	no	no	yes	yes
不同包中没有继 承关系的类	no	no	no	yes

### (二)、封装

面向对象的三个基本特征: 封装、继承、多态。

概念: 就是包装的意思; 可以隐藏事物的细节;

封装的好处: 使用封装, 可以提高程序的安全性, 易用性, 代码的复用性;

在 Java 中, 可以使用关键字: private 修饰成员变量和函数, 被修饰的成员变量和函数就是私有的, 只能在定义的类中使用;

私有化的成员, 在本类之外都不能直接使用; 要使用需要对外提供公开的方法;

一般开发中, 对这些方法的方法名 有规定:

修改方法: set 成员变量 (要修改的值); 举例: int age; setAge(int a)

获取方法: get 成员变量 (); 返回值类型是成员变量的类型; 举例: int getAge()

```
private int age;//年龄  
  
public void setAge(int a){  
    if(a > 0 && a < 150){  
        age = a;  
    }else{  
        //正式开发中, 如果出现错误数据, 是不能继续执行的, 需要使用异常的知识:  
        System.out.println("年龄输入错误, 正确范围是: 0~150");  
    }  
}  
  
public int getAge(){  
    return age;  
}
```

总结: 封装, 就是包装, 核心思想, 就是隐藏事物的实现细节, 同时如果外界要访问, 就提供公开的方法;

### 静态成员变量

概念： 使用 static 关键字修饰的成员变量叫做静态成员变量

格式： 直接将 static 关键字写在成员变量的数据类型前面

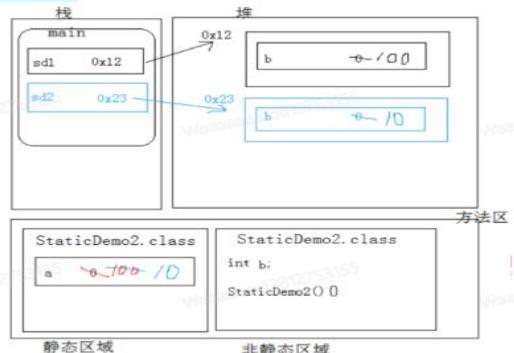
```
class StaticDemo2
{
    static int a = 100;
}
```

特点： 静态成员变量和类的对象无关，可以直接通过类名使用；一个类中的静态成员变量，在这个类的所有对象中共享；

### 静态成员变量内存原理：

```
class StaticDemo2
{
    //静态成员变量
    static int a;
    //非静态成员变量
    int b;
}
class Test
{
    public static void main(String[] args)
    {
        //创建一个StaticDemo2的对象
        StaticDemo2 sd1 = new StaticDemo2();
        sd1.b = 10;
        sd1.a = 100; //静态成员时直接通过类名来使用的
        System.out.println(sd1.a + " -- " + sd1.b);

        //创建一个StaticDemo2的对象
        StaticDemo2 sd2 = new StaticDemo2();
        sd2.b = 10;
        sd2.a = 10;
        System.out.println(sd1.a + " -- " + sd1.b);
    }
}
```



JVM 加载一个类的时候，会将类中的静态成员加载到方法区的静态区域，非静态成员加载到非静态区域

加载静态成员的时候，会先加载所有静态成员，然后给所有静态成员赋默认值，然后给所有静态成员赋显示值

JVM 创建对象时，在堆内存中开辟空间，然后要为类的成员变量在对象空间中分配内存，只是为非静态的成员变量分配

非静态的成员和类的对象有关，静态成员和类的对象无关，只和类本身有关。

### (四)、==、EQUALS ()

== 是比较两个对象的地址值，equals 方法是比较两个对象具体的值；

比如：

```
String s1 = new String("abc");
String s2 = new String("abc");
```

`s1==s2` 就会返回 false, `s1.equals(s2)` 却会返回 true, 原因就是 `s1` 与 `s2` 在堆内存中是两个不同内存空间，但是这两个空间都指向了常量池中的“abc”，所以 == 会返回 false, equals 会返回 true; （一般我们在写封装类的时候，会重写 hashCode 和 equals 方法来判断是否都是同一个对象）

## 一、继承

### (一)、定义

**概念：** java 中的继承，指的就是使用关键字 extends 在两个类之间建立的一种关系；  
**写法：**

```
class F{//表示继承中的父类（超类）}  
class Z extends F{//表示继承中的子类（派生类）}
```

**作用：** 在继承关系中，子类可以直接拥有父类的非私有成员；

### (二)、如何使用

#### 使用注意：

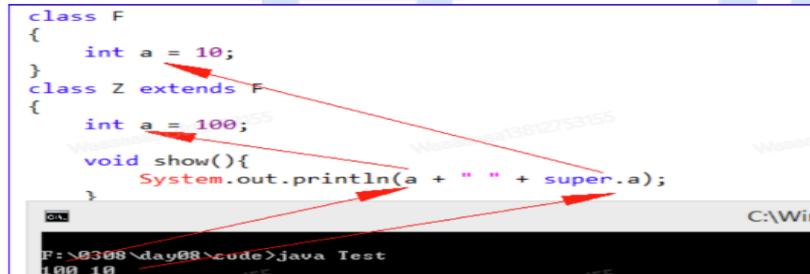
- 1、继承中，子类可以直接拥有父类的成员，不包括私有成员；构造函数也不会被继承；
- 2、在使用中，可以使用 extends 关键字在任意两个类之间建立继承的关系，语法上不报错；但是一般开发中，不会随便使用继承；  
必须这两个类之间有关系；就是子类描述的事物 是 父类描述事物的特例；  
例如：学生，是人，可以使用继承关系；  
比如一个类描述的是猫，一个类描述的是狗，它们两个类就不能相互继承；  
如果两个类不能相互继承，有存在共同的信息，就要找他们共同的父类，然后将相同的信息写在父类中，再让这二者分别继承父类；

Java 中，一个类，只能直接继承一个父类；

Java 中的类，直接继承只能单一继承，但是一个父类可以有多个子类；

### (三)、继承中成员变量特点

- 1、子类直接拥有父类非私有成员变量；
- 2、子类中存在和父类中同名的成员变量；如果子类中存在和父类中同名的成员变量，在使用子类对象时，优先使用子类中的成员；如果要使用父类中的成员，就要通过关键字：super；



```
class F  
{  
    int a = 10;  
}  
class Z extends F  
{  
    int a = 100;  
    void show(){  
        System.out.println(a + " " + super.a);  
    }  
}
```

F:\>0308\day08\code>java Test  
100 10

#### (四)、继承中成员方法

1、子类直接拥有父类非私有成员方法

2、子类中可以定义和父类中同样的成员方法；如果在子类中定义了和父类中一样的函数，然后需要在子类中使用父类的函数，也需要使用 super 关键字；

#### (五)、重写

**概念：**在子类中定义和父类中相同的函数，直接调用函数时，实际使用的是子类中的函数，这种情况叫做方法的重写（覆盖）；

**重写注意事项：**

- 1、子类重写的函数要和父类中的函数名要相同；
- 2、子类重写的函数要和父类中的函数参数列表要相同；
- 3、子类重写的函数要和父类中的函数返回值类型相同；
- 4、子类重写的函数要和父类中的函数的访问方式相同  
(也就是说，父类的方法是静态的，重写的方法也必须是静态的；父类的方法不是静态的，重写的方法也必须不是静态的)；
- 5、子类重写的函数，访问权限不能比父类的低；

总结起来，就是四同一不低于：

函数名、参数列表、返回值类型和是否静态都相同；

子类重写的方法的访问权限不低于父类的函数；

#### (六)、FINAL

final 可以修饰三种东西，分别是，类，方法，变量；

修饰类：表示这个类不可以被继承；

修饰方法：表示这个方法不可以被重写；

修饰变量：表示这是一个常量，值不可以被更改；

## 二、多态

#### (一)、定义

1. 定义：

Java 中的多态：子类型对象赋值给父类型引用：  
(接口的实现类的对象赋值给接口类型引用)。

2. 使用前提：

必须有继承关系存在；

#### (二)、多态的好处

1、减少重复代码，提高代码的复用性；

2、提高程序的扩展性；降低程序的耦合性（联系的紧密程度）；

降低耦合性，又叫做解耦合；

在程序开发中，提高程序的扩展性，降低程序的耦合性，是必要的；

高内聚，低耦合；

一个程序，一般都可以分为多个模块，相同功能的代码，尽量封装在一个模块中，模块与模块之间的联系尽量低；

#### (三)、多态的弊端

多态使用的弊端：不能使用子类中独有的成员；（不能通过多态调用子类特有的方法）

想要使用子类独有的成员可以通过类型转换：

自动向上转型：

```
Animal dog = new Dog();
```

子类型的对象，赋值给父类型的引用变量，就相当于把子类对象的类型给提升为父类型引用；  
这个过程是可以自动进行的，因为子类中可以直接拥有父类中的所有非私有成员，所以直接通过父类型引用操作它的成员变量和函数，都没有问题；

强制向下转型：

```
Animal dog = new Dog();
```

dog 变量的类型是父类型；

如果要使用对象的独有的功能，需要强制向下转型，

```
Dog dd = (Dog)dog;
```

将父类型的引用赋值给子类型的对象，就相当与将引用的类型向下转型了；

因为父类型中不一定有子类型中的某些功能和属性，如果通过子类型的引用操作父类型的数据，

有可能会出现问题，所以这种转型需要强制进行；

在多态中，要使用子类中独有的功能和属性，需要对父类型引用强制向下转型为子类型引用；

封装在面向对象中使用 `private` 关键字保护数据的安全性。

## (二)、继承

继承在 Java 中可以起到扩展新功能的作用，比如新的手机类继承了老的手机类，在继承过程中新的手机继承了旧手机的打电话、发短信功能，而在新的手机中，有玩游戏，发视频等新的功能。

## (三)、多态

定义：事物的多重形态。

多态使用的前提是必须要有继承关系，因为有了继承关系，所以在代码中子类型可以定义为父类型，但是这有个弊端，就是父类型的对象不能直接调用子类型定义的新功能，当然我们可以通过类型转换来解决他。

## 二、异常

### (一)、定义

异常：程序出现期望之外的情况；

异常发生的过程：

当 JVM 执行函数，遇到问题时，就会创建一个对象，将问题的具体信息（问题的类型、导致问题的原因、发生问题的位置等）保存在对象中，然后通过关键字 `throw` 将这个对象抛出来  
如果程序中书写了针对该问题的解决的代码，就执行这些代码；  
如果没有书写，就会将问题返回给函数的调用者同时结束该函数；  
函数的调用者，如果书写了问题的处理代码，就会处理该问题，如果没有书写，就会将发生问题的位置也添加到保存问题的信息的对象中，然后将这个对象继续向上一级抛出；  
如果一直都没有处理问题的代码，最终问题就会抛给 JVM，JVM 就会将对象中保存的问题的详细信息输出在控制台，然后结束程序。

异常的作用：

- 1、异常可以记录发生问题的详细信息，如果程序中出现问题，开发者可以很方便的排查错误；
- 2、通过在程序中合理的设置异常和异常处理代码，可以提高程序的安全性和健壮性；

### 2、异常捕获

在程序中，可以将抛出的异常的对象通过 `try {} ——catch () {}` 代码块，将抛出的异常对象捕获住，然后直接在 `catch` 块里面书写处理的代码；

这个就叫做异常的捕获；如果异常被捕获了，函数对外就不会有问题；

一般在开发中，有些逻辑，如果不管是否抛出异常，一定都要执行到，就应该与在 finally 代码块中；

允许的组合：必须有 try 块，但是不能单独存在；catch 块可以有多个或 0 个，finally 可有可无；三个都不能单独存在；

try-catch:

(try-catch……-catch)

try-catch-finally:

(try-catch……-catch-finally)

try-finally:

## LEC9: FILE I/O AND GUI WITH JAVAFX

### 一、FILE I/O

#### (一)、FILE

File(File parent, String child)

File(String parent, String child)

上面的两个构造方法都可以将父目录和文件或文件夹合并在一起封装成一个新的 File 对象，但是第一个参数永远都是文件夹。

```
/*
 * 演示 File 类中的获取方法
 */
public class FileGetMethodDemo {
    public static void main(String[] args) throws IOException {
        // 创建一个 File 对象
        File file = new File("d:/test");
        //getAbsolutePath 获取当前 File 对象表示的文件或文件夹的全路径（绝对路径）返回的 String
        System.out.println("getAbsolutePath=" + file.getAbsolutePath());
        //getAbsoluteFile 获取当前 File 对象表示的文件或文件夹的全路径（绝对路径）返回的 File 对象
        System.out.println("getAbsoluteFile=" + file.getAbsoluteFile());
        // 获取到的是具体的真正的全路径
        System.out.println("getCanonicalPath=" + file.getCanonicalPath());
        System.out.println("getCanonicalFile=" + file.getCanonicalFile());
        // 获取到的 File 对象中封装的最后一级的名称
        System.out.println("getName=" + file.getName());
```

// getParent 获取到的除去最后一级剩余的父目录名称

System.out.println("getParent=" + file.getParent());

// getPath 获取到 File 中书写的需要被封装的信息

System.out.println("getPath=" + file.getPath());

// getTotalSpace 获取到当前某个盘符的总容量（字节）

System.out.println("getTotalSpace=" + file.getTotalSpace());

// 获取到 当前某个盘符剩余可以被使用的容量

System.out.println("getFreeSpace=" + file.getFreeSpace());

System.out.println("getUsableSpace=" + file.getUsableSpace());

// 获取系统的所有根目录

File[] roots = File.listRoots();

for (File root : roots) {

System.out.println(root);

}

}

## 一、LIST

### (一)、LIST 接口介绍

List 接口它是 Collection 接口的一个更加具体的子接口。List 接口下定义的所有集合都拥有下标 (index)，并可以保存重复的元素。

List 接口继承了 Collection 接口，那么就可以直接使用 Collection 接口中的所有方法，但是由于 List 接口下的集合拥有下标。

因此在 List 接口中又定义了比 Collection 接口中更多的方法，这些方法都是围绕集合的下标而设计的。

### (二)、LIST 方法

#### 1.添加

```
public static void method1() {  
    //创建集合对象  
    List list = new ArrayList();  
    //演示添加元素  
    list.add("aaa");  
    /*  
     * 使用 List 接口中的 add (int index , Object e) 方法添加元素  
     * 指定的 index 下标必须保证它的前面有数据  
     */  
    list.add(10, "bbb");  
    System.out.println(list);  
}
```

#### //获取方法

```
public static void method20 {  
    //创建集合对象  
    List list = new ArrayList();  
    //演示添加元素  
    list.add("aaa");  
    list.add("bbb");  
    list.add("ccc");  
    //获取元素  
    System.out.println(list.get(0));  
    //使用 List 接口中的 get(int index)方法可以直接遍历 List 集合  
    for( int i = 0 ; i < list.size() ; i++ ){  
        System.out.println(list.get(i));  
    }  
}
```

#### 3.修改方法

```
public static void method30 {  
    //创建集合对象  
    List list = new ArrayList();  
    //演示添加元素  
    list.add("aaa");  
    list.add("bbb");  
    list.add("ccc");  
    /*  
     * 修改指定位置上的元素  
     * List 接口中的 set(int index , Object e)  
     * 这里指定的 index 一定是集合中有元素的下标  
     */  
    list.set(3, "ddd");  
    //使用迭代器遍历集合  
    for( Iterator it = list.iterator(); it.hasNext(); ) {
```

#### 4.删除方法

```
//删除
public static void method4() {
    //创建集合对象
    List list = new ArrayList();
    //演示添加元素
    list.add("aaa");
    list.add(2);
    list.add("ccc");
    /*
     * 删除集合中的元素 2
     * 由于 Collection 接口中已经存在了 remove 的方法,
     * 格式: remove(Object obj) 根据指定的对象删除
     * 在 List 接口中也有一个特殊的 remove 方法
     * 格式: remove(int index) 根据指定的下标删除
     * 在使用 List 集合调用 remove 方法的时候, 如果指定的参数是 int 值,
     * 这时这个值一定是下标, 不会自动装箱成 Integer 对象。
     */
    list.remove(2);
    System.out.println(list);
}
```

List 接口，它是 Collection 接口的子接口，它肯定会继承到 iterator 方法，可以使用普通的迭代器直接遍历 list 集合。

List 接口还拥有了自己特有的迭代器：ListIterator。

Iterator 在遍历集合的时候，只要从前往后遍历结束，这个迭代器就无法再次遍历集合。

ListIterator：当直接获取到 ListIterator 迭代器之后，隐式的光标默认也会在集合的第一个元素前面，可以从前往后遍历，

但是遍历结束之后，还可以从后往前遍历。如果在获取 ListIterator 迭代器的时候，指定了从集合某个位置开始遍历的话，

这个 ListIterator 可以从集合的任意位置开始遍历集合。

ListIterator 同时也具备了对 List 集合进行增 删 改 查的方法

ArrayList 集合，它是 List 接口的实现类。也是开发中使用频率较高的集合。ArrayList 类将 List 接口中的所有方法全部实现。

ArrayList 集合它的底层采用的数据存储结构是可变数组。

数据存储结构：数据在容器具体的存储方式。

可变数组

1	2	3	4
---	---	---	---

ArrayList 集合的底层：  
默认会创建拥有10个空间的数组，然后存储数据，如果存储的时候，数据超过的10个，这时会自动的创建一个新的数组，长度是原来的1.5倍。将原来数组中的数据全部复制到新数组中，销毁原来的数组，接着将新的数据存储在后面的空间。

1	2	3	4	5	6
---	---	---	---	---	---

#### (四)、LINKEDLIST 方法

List 接口的链接列表实现。实现所有可选的列表操作，并且允许所有元素（包括 null）。除了实现 List 接口外，LinkedList 类还为在列表的开头及结尾 get、remove 和 insert 元素提供了统一的命名方法。这些操作允许将链接列表用作堆栈、队列或双端队列。

此类实现 Deque 接口，为 add、poll 提供先进先出队列操作，以及其他堆栈和双端队列操作。

LinkedList 集合：它也是 List 接口的实现类。但是它的底层使用的数据结构是 链接列表。可以 LinkedList 模拟队列或堆栈数据结构。

## (一)、SET

Set:一个不包含重复元素的 collection。更确切地讲，set 不包含满足 `e1.equals(e2)` 的元素对 `e1` 和 `e2`，并且最多包含一个 null 元素。正如其名称所暗示的，此接口模仿了数学上的 set 抽象。

Set 接口下的所有的实现类不能保存重复元素，并且 Set 接口中没有定义任何的特有方法，全部继承于 Collection 接口。

Set 接口下有 1 个重要的实现类：HashSet

## (二)、HASHSET

### 1. 定义

55

CPT111 Java Programming

再一课

此类实现 Set 接口，由哈希表（实际上是一个 HashMap 实例）支持。它不保证 set 的迭代顺序；特别是它不保证该顺序恒久不变。此类允许使用 null 元素。

```
/*
 * 演示 HashSet 集合
 */
```

任何对象都可以给集合中保存，任何对象如果给 HashSet 中保存的时候，都需要计算位置。这时计算元素在 HashSet 中的存储位置的算法对应的方法是 Object 类中的 hashCode 方法。如果两个对象调用 hashCode 之后返回的值相同，这时还要调用这两个对象的 equals 方法。  
总结：只要给 HashSet 集合中保存的任何对象，都会调用它们自身的 hashCode 和 equals 方法。

Collection 下的所有集合容器，可以存储任意的对象，它们只能存储单一的对象，如果对象之间有一定的对应关系，这时存储的 Collection 下的集合中，这种关系是没有办法直接维护的。

例如：

姓名           住址

“张三” ----- “北京”

“李四” ----- “上海”

Map 集合：它恰好是用来存放具有一定对应关系的对象数据。

Collection：也被称为单列集合。

Map：也被称为双列集合。

将键映射到值的对象。一个映射不能包含重复的键；每个键最多只能映射到一个值。

Map 接口下的所有集合容器：存放的数据是 key 和 value 这一组数据。要求 key 不能重复。每个 key 都会有一个 value 值和其一一对应。

Map 集合不能直接使用 Iterator 迭代器进行遍历。

Map 集合中保存的数据由 key 和 value 组成的一组对应数据。取出数据，只能通过 key 来获取对应的 value 值。

取出数据的时候应该按照组的方式取出。

Map 中给出了三种遍历的方式：

1、Map 集合中的所有 key 是不会重复的。我们可以直接将 Map 中的所有 key 值拿到，然后组成一个 Set 集合。

可以去遍历 Set 集合，获取到 Map 中的每个 key 值，然后通过 key 获取 value 数据

2、Map 中保存的是由 key 和 value 组成的一组数据，可以将一组数据再次封装成一个对象，然后操作。

3、获取到 Map 中的所有 value 值，组成的集合遍历。没有太大的用途。

### (一)、阶乘

阶乘 fact(n) n >= 0 定义如下

- fact(0) = 1

- fact(n) = n \* n-1 \* n-2 \* ... \* 3 \* 2 \* 1。

### (二)、递归之神 VS 递归调用

递归

```
public static long fact(int n) {  
    if (n == 0)  
        return 1;  
    else  
        return fact(n-1) * n;  
}
```

递归之神

```
public static long fact(int n) {  
    if (n == 0)  
        return 1;  
    else  
        return n * fact(n-1);  
}
```

## (二)、递归步骤

1. 尝试一些案例来查看实例之间的递归关系：

- 在解决问题时，首先尝试一些具体的情况，以便观察问题如何随着输入的变化而变化。这有助于识别递归关系。

2. 为 n 制定一般公式：

- 通过观察和试验，形成适用于问题的一般性公式，其中 n 代表问题的规模或参数。这个公式描述了问题如何递归地分解成更小的子问题。

3. 信任“递归之神”解决更小的实例：

- 这里的“递归之神”指的是信任递归过程，相信递归调用能够解决较小规模的问题。在递归中，问题被分解为更小的实例，通过递归调用来解决。

4. 确保达到基本情况：

- 递归算法通常包含一个基本情况，当问题的规模足够小时，可以直接解决而不再进行递归。确保算法最终达到这个基本情况，以避免无限递归。

总体而言，这是描述递归问题解决过程的一般步骤。通过理解递归关系、制定一般公式、信任递归调用解决小问题，并确保基本情况的到达，可以构建出递归算法。。

古典问题：有一对兔子，从出生后第 3 个月起每个月都生一对兔子，小兔子长到第三个月后每隔 3 个月又生一对兔子，假如兔子都不死，问每个月的兔子对数是多少？

```
public class Learn {  
    public static void main(String[] args) {  
        System.out.println("请输入一个月份 n: ");  
        Scanner scanner = new Scanner(System.in);  
  
        while (scanner.hasNextLine()) {  
            String str = scanner.nextLine();  
            Integer n = Integer.valueOf(str).intValue();  
            int [] arr=new int[n];  
            arr[0]=1;  
            arr[1]=1;  
            //第一个月和第二个月都是只有 1 对兔子，在这里单独处理即可。  
            if(n==1||n==2){  
                System.out.println("第"+n+"个月的兔子对数是： 1 对");  
            }else{  
                int count=0;  
                for(int i=2;i<arr.length;i++){  
                    arr[i]=arr[i-2]+arr[i-1];  
                    count=arr[i];  
                }  
            }  
        }  
    }  
}
```

```
        System.out.println("第"+n+"个月的兔子对数是： "+count+"对");  
    }  
}  
scanner.close();  
}
```

- 给定一个仅包含大写字母 A-Z 或小写字母 a-z 的单词
  - 返回单词的所有子序列，用逗号分隔。
- 其中子序列是在单词中以它们出现的顺序找到的字母字符串。

```

public class SubsequenceGenerator {

    public static void main(String[] args) {
        String word = "abc";
        String result = allSubsequences(word);
        System.out.println(result);
    }

    public static String allSubsequences(String word) {
        StringBuilder result = new StringBuilder();
        generateSubsequences("", word, result);
        return result.toString().trim();
    }

    private static void generateSubsequences(String current, String remaining, StringBuilder
result) {
        if (remaining.isEmpty()) {
            result.append(current).append(", ");
            return;
        }

        generateSubsequences(current + remaining.charAt(0), remaining.substring(1), result);
        generateSubsequences(current, remaining.substring(1), result);
    }
}

```

这个 Java 程序定义了一个 SubsequenceGenerator 类，包含一个 allSubsequences 方法来生成给定单词的所有子序列。通过递归调用 generateSubsequences 方法，可以获取所有可能的子序列。

#### 一、公共权威的信息

这是一项法案，目的是为公共机构或为其提供服务的个人的信息披露提供规定。这项法案在实践中有效。

你有权知晓的数据包括：请求“由公共机构持有的任何记录信息，如政府部门、地方议会或国家学校”。

问题：私人组织如亚马逊等受此法案约束吗？

• 目的

“开放对于现代国家的政治健康至关重要。”

“政府中不必要的保密导致治理的傲慢和有缺陷的决策制定。”

#### 二、信息

- 公共机构发布有关其活动的某些信息；

- 公众可以向公共机构请求信息。

- 信息的种类：存储在计算机上、电子邮件中以及印刷或手写文件中的信息，以及图像、视频和音频记录。

不适用于此法案的情况：

人们获取自己的个人数据，例如公共机构持有的有关他们的健康记录或信用参考的信息。

接下来是数据保护法。

#### 三、如何实施

实施者：数据保护官（DPO）

● 处理时间：1 个月；对于复杂情况，可能延长至 2 个月

● 费用：免费，但可能产生行政成本

● 不适用于：

您的请求不清楚

您的请求意味着创建新信息

您的请求会不公平地泄露有关他人的个人详细信息。

● 可以保留信息

“防止、检测或调查犯罪”

“国家安全或军队”

“税收的评估或征收”

“司法或部长任命”

## (一)、英国数据保护法

### 一、你的信息

- 《通用数据保护条例》(GDPR)，当前版本为 2018 年
- "规定了组织、企业或政府如何使用您的个人信息"
- 您的权利
  - "了解政府和其他组织存储关于您的信息"，包括
  - 访问/更新/删除您的个人数据
  - 控制对您个人数据的处理

接下来

- 处理您的数据时政府和其他组织需要遵循的原则
- 您数据的法律保护。

### 二、什么是个人数据

法律保护涉及您的数据，包括以下方面：

- 种族
- 民族背景
- 政治观点
- 宗教信仰
- 工会会员资格
- 遗传信息
- 生物识别信息（用于身份验证的情况）
- 健康信息
- 性生活或取向

### 三、原则

政府和其他组织需要遵循以下原则：

- "公平、合法和透明地使用"
- "仅用于特定、明确的目的"
- "以足够、相关且仅限于必要的方式使用"
- "准确，必要时保持更新"
- "保留时间不超过必要的时间"
- "以确保适当安全性的方法处理，包括防止非法或未经授权的处理、访问、丢失、销毁或损害"

- "负有责任的控制者必须对合规性负责，并能够证明合规。"

#### ● 1. 合法性、公平性和透明性

- 合法性意味着用户已经同意这样做；您必须为履行合同而这样做，并且这是履行法定义务所必需的。
  - 公平性意味着您不会滥用或误用您收集的数据。
  - 透明性：在向数据主体清楚、公开和诚实地说明您是谁，以及为什么以及如何处理他们的个人数据。

#### ● 2. 目的限制

- 目的限制意味着数据仅"为特定、明确和合法的目的而收集"。
  - 您对处理数据的目的必须得到明确的确认。并且必须通过隐私通知清楚地告知个人。

最后，您必须密切遵循这些目的，仅限制数据处理为您已经声明的目的。

- 如果您想要将已收集的数据用于与最初目的不兼容的新目的，除非您在法律中明确规定了明确的义务或功能，否则请明确征得再次同意。

#### ● 3. 数据最小化

- 只收集完成您目的所需的最少量数据。例如，如果您想要为电子时事通讯收集订阅者，您应该只请求发送时事通讯所需的信息。避免收集与您的目的无直接关系的个人数据，如电话号码或家庭地址。

#### ● 4. 准确性

- 由您负责确保您收集和存储的数据的准确性。建立检查和平衡机制，以更正、更新或删除进来的不正确或不完整的数据。还应定期进行审计，以确保存储的数据的净化。

#### ● 5. 存储限制

- 根据 GDPR 规定，您必须证明您保存的每个数据片段的时间长度是合理的。制定数据保留期限是符合存储限制政策的好方法。创建一个标准时间段，之后您将对任何您不再主动使用的数据进行匿名化。一旦您使用个人数据完毕，请立即删除。

#### ● 6. 完整性和保密性

- GDPR 要求您维护您收集的数据的完整性和保密性，基本上是将其保护免受内部或外部威胁。这需要计划和主动的勤勉。您必须保护数据免受未经授权或非法处理以及意外的丢失、销毁或损坏。

## ● 7. 责任

- GDPR 监管机构知道组织可能会声称他们遵循所有规则，而实际上并非如此。这就是为什么他们要求一定程度的责任：您必须采取适当的措施和记录作为遵守数据处理原则的证明。监管机构可以随时要求提供这些证据。文档记录在这里至关重要。它为您和监管机构提供了一条审计路径，如果需要证明责任，您和监管机构都可以遵循这条路径。

## 五、个人权利

### ● 知情权

- 个体有权获知其个人数据的使用情况，包括数据收集的目的、处理方式以及与之相关的其他信息。

### ● 访问权

- 个体有权获得对其个人数据的访问，以了解组织是否在处理其个人数据。

### ● 纠正权

- 个体有权纠正关于其个人数据的不准确信息，并可以在数据被更正之后，要求通知相关的数据接收者。

### ● 删除权

- 个体有权要求组织删除其个人数据，前提是没有任何合法的理由继续处理这些数据。

### ● 限制处理权

- 个体有权要求在某些情况下限制其个人数据的处理，例如在纠正数据准确性期间或在确定处理是否合法时。

### ● 数据可携带性权

- 个体有权获得其个人数据的副本，并在需要时将其传输给其他数据控制者。

### ● 反对权

- 个体有权反对其个人数据的处理，特别是在基于合法权益或执行任务的公共权力时。

### ● 与自动决策和档案制作有关的权利

- 个体有权不受到仅基于自动处理和产生的档案制作对其产生法律效果的影响。

### ● 知情权

- 根据 GDPR，个体有权了解公司如何收集和使用他们的个人数据，计划保留数据的时间以及与谁分享这些数据。收集数据的公司必须向数据主体提供某些信息，包括数据控制者和数据保护官（如果已任命 DPO）的身份和联系方式。

### ● 访问权

- 个体有权知道和审查公司究竟收集了哪些信息，他们如何存储和处理这些数据，以及他们打算如何使用它。

### ● 纠正权

- 数据主体有权使不完整的数据得到补充，纠正不正确的数据。

### ● 删除权

- 数据主体有权永久删除个人数据。这也被称为“被遗忘的权利”。在这种情况下，公司不能主张他们处理用户数据的合法权益超过个体有权要求删除它的权利。然而，如果受到删除请求的数据的处理对于履行公司的法定义务是必要的，那么这个权利就不适用。

### ● 限制处理权

- 如果数据主体不能要求数据控制者删除他们的个人信息，他们可以根据信息专员办公室 (ICO) 规定的某些情况限制数据控制者处理这些数据的能力。

### ● 数据可携带性权

- 个体有权获取并重复使用他们的个人数据，以满足他们自己的目的，跨不同的服务。数据主体可以要求数据控制者将其个人数据文件以电子形式发送给第三方。如果在技术上可行，公司必须以通用的、可机器读取的格式提供数据。

## 六、个人数据泄露

在 GDPR 下，所有组织都有法定义务报告：

- 任何个人数据泄漏

- 到信息专员办公室 (ICO)

- 在察觉到泄漏后的 72 小时内。

## ● 法律与伦理

- 单独的法律无法限制人类行为
  - 描述/执行所有可接受行为是不切实际/不可能的
- 伦理/道德对于大多数人来说是足够的自我控制
- 伦理并非宗教（但宗教包含伦理原则）
- 伦理原则并非普世的
  - 在不同文化中有所不同
  - 在同一文化中甚至在不同个体中也有所不同
- 伦理具有多元性质

70

- 与科学和技术常常只有一个正确答案形成鲜明对比。

- 功利主义原则：“采取能够为所有相关方创造最大价值的行动”
- 风险规避原则：“采取对所有相关方造成最小伤害或产生最小成本的行动”
- 道德无免费午餐规则：“假设所有有形和无形的物体都属于其他人，除非相反证明。”“如果有人为你创造了某种价值的东西，那么那个人可能希望得到你对其使用的补偿”

## (二)、保护程序和数据

- 版权 — 旨在保护思想的表达（思维的创意作品）

- 思想本身是自由的
  - 不同的人可以有相同的想法
  - 表达思想的方式受版权保护
  - 版权是对表达形式制作复制的独占权
- 版权保护知识产权（IP）
- IP 必须是：
  - 原创作品
  - 在某种有形的表达媒介中

## 一、IP

### ● 法律保护：

- 版权、商标、商业机密和专利

### ● 类别：

- 发明、商标、标志和工业设计
- 版权材料（文学和艺术作品，在线材料）
- 公平使用原则：用于特定目的

### ● 信息技术对知识产权的影响：

- 容易下载/分发视频，不需要征得许可
- 容易修改他人的作品
- 容易复制和粘贴

### ● 专利 — 旨在保护有形物体或制造它们的方法（而非思想的作品）

- 被保护的实体必须是新颖且非显而易见的
- 获得专利的第一位发明家将其发明受到专利侵权的保护
- 仅自 1981 年起对算法申请专利

### ● 商业机密 — 提供竞争优势的信息

- 只有在保持秘密的情况下才有价值的信息
- 无法或非常困难地撤销一个秘密的泄露

- 用于揭示商业机密的逆向工程是合法的!
- 商业机密的保护非常适用于计算机软件
  - 例如，使用他人不知道的算法的程序

## 二、计算机道德

● 道德准则：在信息和通信技术（ICT）中，是确定特定计算机行为是道德还是不道德的指南。

- 计算机犯罪：涉及计算机的任何违法行为。
- 网络法律：旨在保护互联网和其他在线系统的法律框架。
- 案例研究 - 入侵他人银行账户：常见的方法和预防措施包括：
  - 技术安全性：使用强密码、多因素身份验证等技术手段，确保系统和账户的安全。
  - 网络监测：实施实时监测系统，以便及时发现任何可疑活动。
  - 教育和培训：提高用户的网络安全意识，教育人们如何辨别和避免潜在的网络威胁。
  - 法规遵守：遵守适用的法律和法规，确保组织在网络空间的活动合法合规。

这些方法的目标是防范计算机犯罪，维护网络安全，并促进在数字环境中的合法和道德行为。

## 三、黑客与白客

- 黑帽黑客：他们利用自己的知识进行恶意目的，比如敲诈勒索等。
- 白帽黑客：他们具备与黑帽黑客相同的知识，但利用这些知识来帮助公司增强对抗黑帽黑客的防御能力。

这两种类型的黑客代表了两种截然不同的动机和行为。黑帽黑客通常从事非法活动，而白帽黑客致力于以正面方式应对潜在的网络威胁。在网络安全领域，白帽黑客的工作通常包括漏洞测试、安全审计和协助组织加强其网络和系统的安全性。

## 四、IT 专业问题

专业主义是什么？

根据韦氏词典的定义，专业主义指的是表征或标志一个职业或专业人士的行为、目标或品质。

IT 领域中的专业问题，包括但不限于：

- 专业沟通：研究、写作和演示等方面的能力。
- 专业伦理：遵循道德规范和原则，确保在信息技术领域的行为是正当的和负责任的。
- 法律、社会和文化问题：处理与法律规定、社会期望和文化差异相关的问题。
- IT 治理：包括制定和实施原则、变革和风险管理，以确保信息技术的有效和负责任的使用。
- 团队合作概念和社会问题：开发和维护团队协作技能，同时考虑到社会和文化背景的差异。

## 五、针对 IT 专业人员的道德规范

这是 IEEE 计算机协会（IEEE CS）的职业道德准则：

- 公共利益：以与公众利益一致的方式行事。
- 客户和雇主：在与公共利益一致的前提下，以符合客户和雇主最佳利益的方式行事。
- 产品：确保其产品和相关修改符合可能的最高专业标准。
- 判断力：在专业判断中保持诚信和独立性。
- 管理：订阅并促进在软件开发和维护管理方面的道德方法。
- 职业：促进与公共利益一致的专业完整性和声誉。
- 同事：对同事公平支持。
- 个人：参与终身学习，不断提高对专业实践的认识，并促进道德方法在专业实践中的应用。

## 六、关于 JavaDoc 的内容

- JavaDoc 注释的结构：

- JavaDoc 注释由标准的多行注释标签`/\*`和`\*/`包围。开头的标签（称为开始注释定界符）包含一个额外的星号，即`\*/`。
- 注释中的第一个段落通常是对所记录方法的描述。
- 在描述之后，可能有多个描述性标签，包括：
  - `@param`：描述方法的参数。
  - `@return`：指定方法返回的内容。
  - `@throws`：指示方法可能引发的任何异常。
  - 其他不太常见的标签，如`@see`（“参见”标签）。

这个结构有助于直接从源代码生成全面而清晰的 Java 文档。