

## 一、数组

### (一)什么是数组

- 1、数组是一组数据，这组数据在内存的堆空间中连续存放，并可通过下标访问。
  - 2、数组由数组变量来引用，数组变量在栈空间，命名规则遵循变量名的命名规则。
  - 3、数组变量是引用变量，存放堆空间中数组的地址，堆空间的数组称为数组对象。
- 以下是数组变量、数组对象在内存中的示意图：

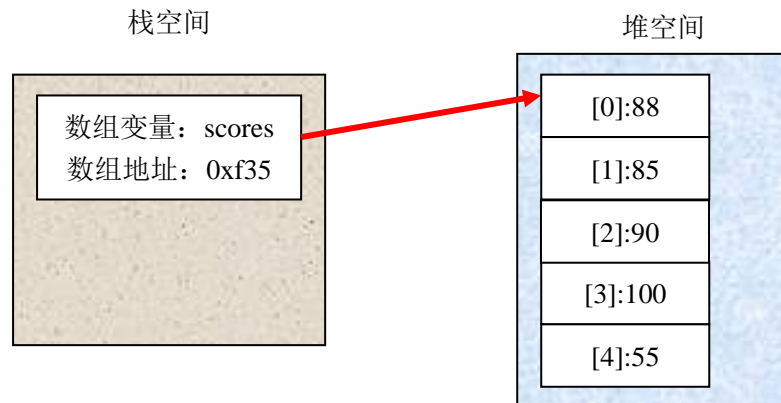


图-2

4、数组元素：数组中的个体称为数组元素，由数组变量名[下标]引用，数组元素的下标从 0 开始计算，最大值是数组长度-1。下标必须是 int 类型。

5、数组元素的类型可以是基本数据类型，String 类型，也可以是后面要介绍的对象类型。

提示：数组中所有元素的类型相同。

### (二)定义数组

#### 定义格式(1)

类型[] 数组变量名;

示例: `int[] scores;`

说明：以上代码在栈空间声明了一个数组的引用变量 scores，注意：该变量中还没有存放堆空间中任何数组的地址。

#### 定义格式(2)

类型 数组变量名[];

这种定义格式是 C 语言的风格，目的是兼容 C 语言的语法格式。不建议使用这种定义方式。

#### 定义格式(3)

类型[] 数组变量名=new 类型[数组容量];

示例: `int[] scores=new int[5];`

说明：在堆空间创建了如图-3 所示的数组。

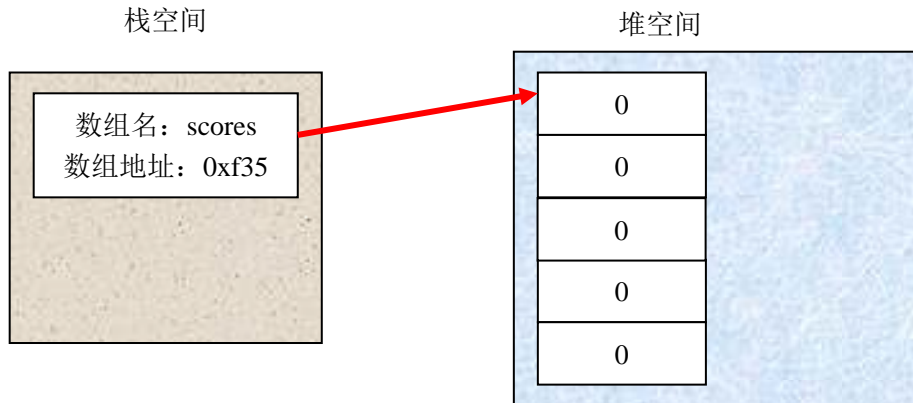


图-3

提示:

- 1、将数组的地址赋值给栈空间的引用变量 `scores`
- 2、此时数组中每个元素的值为 0。JVM 在堆空间中创建数组时，根据数组元素类型会为数组元素初始化数据：
  - 整数类型初始化为 0;
  - 浮点类型初始化为 0.0;
  - 字符类型初始化为 unicode 码为 1 的字符;
  - boolean 类型初始化为 false;
  - String 类型初始化为 null。

#### 定义格式(4)动态初始化数组

类型[] 数组变量名=new 类型[] {数组元素 1 的数据, …… , 数组元素 n 的数据};

说明: 在堆中创建数组并给每个数组元素赋值, 最后将数组的地址赋值给栈中的数组变量。

示例: `int[] scores=new int[] {88,85,100,90,55};`

说明: 以上定义的数组效果如图-2 所示。

提示:

定义数组并初始化数组元素值时, 不能再设置数组元素的容量, 因此, 以下是错误的定义:

```
int[] scores=new int[5]{88,85,100,90,55};
```

#### 定义格式(5)静态初始化数组

类型[] 数组变量名={数组元素 1 的数据, …… , 数组元素 n 的数据};

示例: `int[] scores={88,85,100,90,55};`

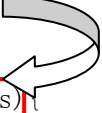
说明:

(1) 静态初始化只限定于声明数组时使用, 而动态初始化可以在代码中随时使用, 例如定义以下方法:

```
static void print(int[] scores){  
    //打印数组 scores 中各元素的值  
}
```

在调用该方法时, 可以用动态数组初始化的方法在堆中创建一个数组, 并将该数组地址赋值给 `scores` 数组:

```
public static void main(String args[]) {  
    print(new int[] {55, 88, 72});  
}  
  
static void print(int[] scores) {  
    //打印数组 scores 中各元素的值  
}
```



### 创建空数组

- 1、空数组是指堆中的数组对象没有数组元素，但数组有地址。
- 2、当一个数组的容量不能预先确定时，需要根据程序运行的具体情况再确定数组容量，此时应定义空数组。

定义格式(1):

类型[] 数组变量名=new 类型[0];

示例:

```
int[] b=new int[0];  
System.out.println("数组地址:"+b+" 数组长度:"+b.length);
```

运行结果:

数组地址:[I@de6ced 数组长度:0

定义格式(2)静态初始化定义空数组:

类型[] 数组变量名={ }

示例:

```
int[] a={};  
System.out.println("数组地址:"+a+" 数组长度:"+a.length);
```

运行结果:

数组地址:[I@c17164 数组长度:0

### (三)数组的赋值

方式(1)动态初始化

```
int[] scores=new int[] {88, 85, 100, 90, 55};
```

方式(2)静态初始化

```
int[] scores={88, 85, 100, 90, 55};
```

方式(3)先创建数组，再用赋值语句给数组元素赋值

```
int[] scores=new int[5];  
scores[0]=75;  
scores[1]=83;  
scores[2]=77;  
scores[3]=90;  
scores[4]=78;
```

### (四)length属性

length 属性存放了数组元素总数。

示例:

```
String[] names=new int[5];  
System.out.println("数组长度="+names.length);
```

提示: length 不是方法，因此没有(), 这与计算字符串长度的函数 length() 不同。

## (五) Arrays.toString()

1、Arrays 是 Java 提供的一个工具类，提供了排序等操作数组的方法。

2、toString(数组变量名)：该方法返回指定数组的所有元素的值。

示例：打印数组 a 的所有元素值：

```
int[] a={33,44,55};  
System.out.println(Arrays.toString(a));
```

打印结果：

```
[33, 44, 55]
```

## (六) 数组应用初阶

### 【示例-1】不用判断语句给 5 分制的分数评定成绩等级。

分析：

定义数组 grade，代码如下：

```
String[] grades={  
    "不及格","不及格","不及格","及格","良好","优秀"  
};
```

数组元素的值是分数的等级，数组元素的下标作为分数。

```
package com.ityw.basic.day05;  
import java.util.Scanner;  
public class Test01 {  
    public static void main(String[] args) {  
        Scanner scanner=new Scanner(System.in);  
        System.out.println("输入5分制的分数:");  
        int score;  
        do{  
            score=scanner.nextInt();  
        }while(score<1 || score>5);  
        String[] grades={  
            "不及格","不及格","不及格","及格","良好","优秀"  
        };  
        System.out.println(grades[score]);  
    }  
}
```

### 【课堂练习】

不用判断语句，输入百分制的分数，根据以下标准评定分数等级：

90~100:优秀

89-80:良好

79-70:中等

69-60:及格

0-59:不及格

### 【示例-2】通过数组元素查找下标。

已知以下学生的姓名，键盘输入姓名，查找学生姓名。

张飞,王菲,刘亦菲.咖啡,吗啡

分析：

步骤 1、将以上五个学生的姓名存放在一个字符串数组中；

步骤 2、键盘输入姓名。

步骤 3、用循环查找该姓名的学号。

```
package com.ityw.basic.day05;  
import java.util.Scanner;  
public class Test02 {
```

```

/**
 * 已知以下学生的姓名，键盘输入姓名，查找学生姓名。
 * 张飞，王菲，刘亦菲。咖啡，吗啡
 */
public static void main(String[] args) {
    String[] names={
        "张飞","王菲","刘亦菲","咖啡","吗啡"
    };
    Scanner scanner=new Scanner(System.in);
    System.out.println("name=");
    String name=scanner.next();
    for(int i=0;i<names.length;i++){
        if(names[i].equals(name)){
            System.out.println(name+"的学号: "+i);
        }
    }
}

```

【示例-3】随机产生 10 个[60, 100]的分数，显示所有超过平均分的分数和下标。

分析：

步骤 1、创建数组用于保存 10 个分数；

步骤 2、用一个循环随机产生 10 个分数，分别保存在数组中，并累加总分。

步骤 3、通过总分计算出平均分。

步骤 4、用一个循环依次判断数组中哪些分数超过了平均分并打印这些分数。

```

package com.ityw.basic.day05;
import java.util.Random;
public class Test03 {
    /**
     * 随机产生10个60~100之间的分数，显示所有超过平均分的分数和下标。
     */
    public static void main(String[] args) {
        int[] score=new int[10];
        Random random=new Random();
        int total=0;//总分
        //步骤1-随机产生10个分数，保存至数组中，并计算总分
        for (int i = 0; i < score.length; i++) {
            score[i]=random.nextInt(41)+60;
            total+=score[i];//总分累加
            //打印每个分数
            System.out.print(score[i]+" ");
        }
        System.out.println();//输出空行
        //步骤2-计算平均分
        double avg=total/10.0;
        System.out.println("avg="+avg);
        //步骤3-查找超过平均分的分数和下标
        for (int i = 0; i < score.length; i++) {
            if(score[i]>avg){ //查找并打印超过平均分的分数和下标
                System.out.println(score[i]+":"+i);
            }
        }
    }
}

```

【示例-4】随机产生 10 个[60, 100]的分数，找出最高分。

分析：

步骤 1、创建数组用于保存 10 个分数；

步骤 2、循环随机产生 10 个分数，分别保存在数组中。

步骤 3、选择排序法：

通过循环将最高分交换到数组下标为 0 的元素中。

```
package com.ityw.basic.day05;
import java.util.Random;
public class Test04 {
    /**
     * 随机产生10个60~100之间的分数，找出最高分。
     */
    public static void main(String[] args) {
        Random random=new Random();
        //步骤1-随机产生 10分数，保存至数组中
        int[] score=new int[10];
        for (int i = 0; i < score.length; i++) {
            score[i]=random.nextInt(41)+60;
            System.out.print(score[i]+" ");
        }
        /*步骤2-找最高分
         * 选择排序的方式：将最高分交换到score[0]
         */
        for(int j=1;j<score.length;j++){
            if(score[0]<score[j]){
                //交换score[0]和score[j]
                int t=score[0];
                score[0]=score[j];
                score[j]=t;
            }
        }
        System.out.println("最高分："+score[0]);
    }
}
```

## 二、访问修饰符

### 【示例-5】访问修饰符引入案例

编写一个查作业的程序，显示一个菜单，菜单中各项是第二章和第三章的部分例题。例如：

```
1-第二章-Test01
2-第二章-Test02
3-第三章-Test01
4-第三章-Test02
5-第三章-Test03
```

```
public class Test05 {
    /**
     * 编写一个查作业的程序，显示一个菜单，菜单中各项是第二章和第三章
     * 的部分例题。例如：
     * 1-第二章-Test01
     * 2-第二章-Test02
     * 3-第三章-Test01
     * 4-第三章-Test02
     * 5-第三章-Test03
     */
    public static void main(String[] args) {
```

```

Scanner scanner=new Scanner(System.in);
System.out.println("1-第二章-Test01");
System.out.println("2-第二章-Test02");
System.out.println("3-第三章-Test01");
System.out.println("4-第三章-Test02");
System.out.println("5-第三章-Test03");
int select=scanner.nextInt(); //接收输入的菜单数字
switch (select) {
case 1:
    com.ityw.basic.day02.Test01.main(null);
    break;
case 2:
    com.ityw.basic.day02.Test02.main(null);
    break;
case 3:
    Test01.main(null);
    break;
case 4:
    Test02.main(null);
    break;
case 5:
    Test03.main(null);
    break;
default:
    System.out.println("选择错误");
    break;
}
}
}

```

标注(1)

标注(2)

说明:

1、其它程序调用静态方法

Java 规定：在其它程序中调用类的静态方法的格式：

类名.方法(实参)

例如以上代码中的 Test03.main(null);

2、null 值

main 方法的形参是字符串数组 args，如前所述：数组变量在栈空间，数组中的数据在堆空间，若栈空间的数组变量不存放堆空间的数据的地址，则 Java 规定：给引用变量赋值为 null，表示该变量存放的地址为空。

3、导包问题

【示例-5】的代码中出现了类名相同的现象，即 case 1 和 case 3 调用的类名都是 Test01，这两个类分别存放在 com.ityw.basic.day02 和 com.ityw.basic.day03 包下，Java 规定：在同一包中的类可以互相访问，不同包下需要指明类所在的包，称为导包。

例如：

调当前包 (com.ityw.basic.day03) 下的 Test01.main 方法：

Test01.main(null);

调用 com.ityw.basic.day02 下的 Test01.main 方法：

com.ityw.basic.day02.Test01.main(null);

## (一) 访问修饰符概述

访问修饰符用来设置类、方法和类的成员变量被其它程序访问的范围。Java 的访问修饰符有以下四种：

- 1、public
- 2、默认
- 3、private
- 4、protected

## (二) 访问修饰符的作用

关键字	含义	本类的代码	当前包的其它类	其它包的类	子类
public	公有的	✓	✓	✓	✓
不写	默认的	✓	✓	×	×
private	私有的	✓	×	×	×
protected	保护的	✓	✓	×	✓

说明：

(1) public 修饰的类、方法和类的成员变量允许所有的代码访问，包括其它包中的程序。

(2) 不写访问修饰符的类、方法和类的成员变量，允许本类和当前包下的其它程序访问。

(3) private 修饰的类、方法和类的成员变量，只有本类中的代码能访问。

(4) protected 修饰的类、方法和成员变量，在本类和子类中能访问。

提示：

(1) 子类的概念在后面介绍面向对象时详述。

(2) 访问修饰符不能用于方法中的局部变量。

例 1：将【示例-20】中“标注(1)”所指向的 Test01 类前的 public 去掉，并保存程序（注意：程序只有保存，修改才能见效）如图-9 所示：

```
1 package com.ityw.basic.day02;
2
3 public class Test01 {
```

图-9

即：将该类的访问范围设置为 com.ityw.basic.day02 包下的程序可以访问，但其它包的程序不能访问。此时，【示例-21】的 Test20\_2 类会出现图-6 所示的编译错误：

```
switch (select) {
case 1:
    com.ityw.basic.day02.Test01.main(null);
    break;
case 2:
```

图-6

图-6 的编译错误是指 com.ityw.basic.day02.Test01 是不可见的。

在 Test01 类前重新加上 public 并保存程序，则图-6 中的编译错误消失。

### 【示例-6】访问修饰符和导包案例

步骤 1、创建 com.ityw.basic.day03.exercise 包；

步骤 2、将 com.ityw.basic.day03 包下的 Test11.java 复制到

com.ityw.basic.day03.exercise 包下；

步骤 3、修改 com.ityw.basic.day03.exercise 包下的 Test11 类名为 Test11\_2。

打开 Test20\_2 类，代码如下所示：

```
package com.ityw.basic.day05;
import java.util.Scanner;
/*
    以导包的形式指明Test01、Test02和Test03这三个类从工作空间的
    com/ityw/basic/day03文件夹下载入。
*/
import com.ityw.basic.day03.Test01;
import com.ityw.basic.day03.Test02;
```



```

import com.ityw.basic.day03.Test03;
public class Test07 {
    /**
     * 本类是从com.ityw.basic.day03包下复制过来的Test11类,
     * 类名已修改为Test11_2
     */
    public static void main(String[] args) {
        Scanner scanner=new Scanner(System.in);
        System.out.println("1-第二章-Test01");
        System.out.println("2-第二章-Test02");
        System.out.println("3-第三章-Test01");
        System.out.println("4-第三章-Test02");
        System.out.println("5-第三章-Test03");
        int select=scanner.nextInt();
        switch (select) {
            case 1:
                com.ityw.basic.day02.Test01.main(null);
                break;
            case 2:
                com.ityw.basic.day02.Test02.main(null);
                break;
            case 3:
                Test01.main(null);
                break;
            case 4:
                Test02.main(null);
                break;
            case 5:
                Test03.main(null);
                break;
            default:
                System.out.println("选择错误");
                break;
        }
    }
}

```

标注(1)

标注(2)

说明:

1、import 语句指明【示例-7】从 d:\workspace\corejava\com\ityw\basic\day03 文件夹下载入 Test01.java、Test02.class 和 Test03.class 这三个文件。

其中 d:\workspace\corejava 是项目所在的磁盘位置，com\ityw\basic\day03 就是 import 命令后面的 com.ityw.basic.day03 包。

2、以上代码的标注(1)和标注(2)指明 com.ityw.basic.day02 包下调用的 Test01 和 Test02 类。

提示:

- 1、如果将标注(1)中的包名去掉，则调用的是 day03 包下的 Test01 和 Test02 这两个类。
- 2、若用 import 命令导入 day02 包下的两个类，如图-7 所示：

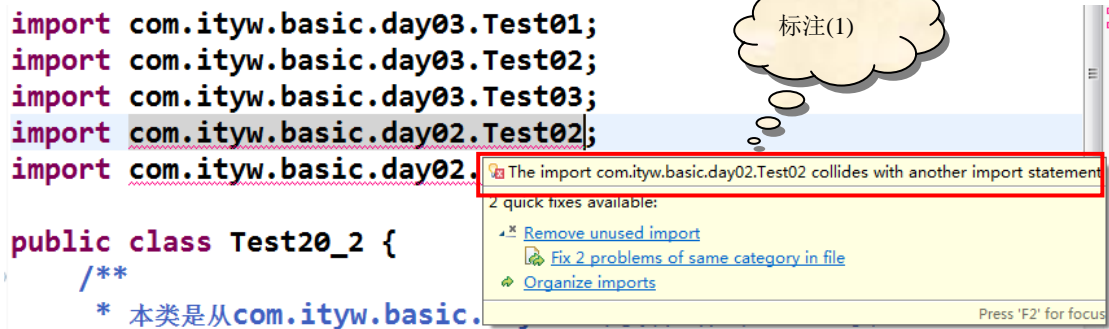


图-7

图-7 标注(1)的错误提示: import com.ityw.basic.day02.Test03 与另一个导包声明冲突。即: 在同一个类中不能用 import 命令设置从两个包下加载同名的两个类。

将 com.ityw.basic.day03 包下的 Test01 类前面的 public 删除, 改为默认访问修饰符, 即只能被当前包下的程序访问。那么将出现图-8 所示的编译错误:

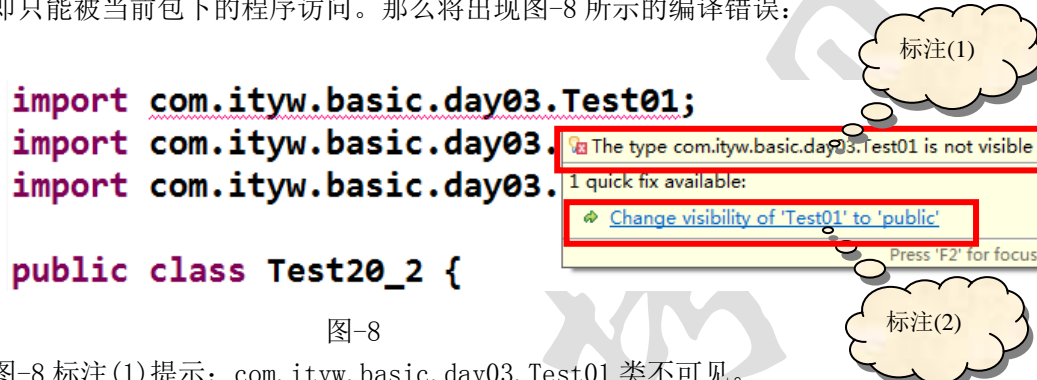


图-8

图-8 标注(1)提示: com.ityw.basic.day03.Test01 类不可见。

图-8 标注(2)建议将 Test01 的可见性设置为 public。

### (三)为什么要设置访问范围?

项目开发的原则: 类、方法和类的成员变量的范围在允许的情况下越小要好, 这样做的好处是:

- 1、减少使用者的学习成本。如果我们编写的类被其它人使用, 那么应该将不需要别人知道的类、方法和变量设置为对其不可见, 使其不需要去学习、记忆对其没有用的知识。
- 2、公有的类、方法和变量因可能被众多人使用, 是不能再做修改的, 否则会影响所有的使用者。而私有的类、方法和变量不会被这种因素所束缚。

#### 【示例-7】以方法为例说明设置访问范围的必要性

以下创建一个我的数学工具类, 该类具有与 Java 的 Math 类相似的功能, 用于常用的数学运算。该类可被创建为 .jar 文件, 作为工具包被其它 Java 程序员使用。

创建步骤如下:

步骤 1、在 CoreJava 项目的 src 文件夹下创建 myUtils(我的工具)包;

步骤 2、在 myUtils 包下创建 MyMath 类, 如图-9 所示:

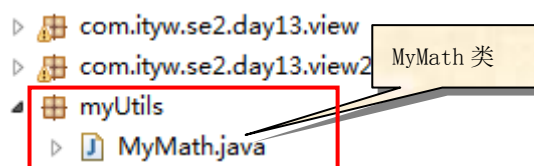


图-9

步骤 3、在 MyMath 类中输入如下代码：

```
package myUtils;

/**
 * MyMath: 我的数学工具类，用于数学计算
 */
public class MyMath {
    /**
     * pround方法：保留小数点后指定位数，对第n+1位四舍五入
     * pround方法对所有的程序都是可见的，可被所有的程序调用
     * @param value: 被保留的浮点数
     * @param n: 指定保留的位数
     * @return double
     */
    public static double pround(double value, int n) {
        //调用本类自定义的方法pow10
        return Math.round(value*pow10(n))*1.0/pow10(n);
    }
    //本方法只为本类的其它方法服务，对外部程序是不可见的
    private static double pow10(int n) {
        return Math.pow(10, n);
    }
}
```

说明：

MyMath 类中现在定义了两个方法：

- 1、 pround 方法，该方法就像 Math.round 方法一样，用于四舍五入，但功能要强于 Math.round，可保留小数点后任意位。该方法被定义为 public，目的就是能被所有程序调用。
- 2、 pow10 方法：计算  $10^n$ ，该方法被定义为 private，该方法对外不可见，只有本类的方法才能调用该方法，例如在 pround 方法中调用了 pow10()。

将 pow10 定义私有的，若以后因为 MyMath 版本的升级，需要将该方法名改为

```
private static pow(int m, int n) {
    return Math.pow(m, n);
}
```

即将该方法名由 pow10 改为 pow，增加参数，使得该方法能计算任意底数的任意次幂。那么，只需要在本类中修改 pround 方法中的代码，如下所示：

```
public static double pround(double value, int n) {
    //调用本类自定义的方法pow10
    return Math.round(value*pow(10, n))*1.0/pow10(10, n);
}
```

若 pow10 被设置为 public，则 pow10 方法可能被许多外部程序调用，这种情况下若要修改 pow10 的方法名和参数列表，则所有调用该方法的外部程序都会报错，都需要重新修改代码。增加了使用 MyMath 类的用户的维护成本。这种情况是灾难性的。

一般情况下，设置为 public 的方法，为了避免以上情况，以后是不会再做修改的。所

以，方法的访问范围应该设置得越小越好，这样项目的修改才不会束手束脚。

## (四) 方法的重载应用

【示例-8】在项目的myTools包中创建MyTools类(我的工具类)，在该类中编写重载的swap方法

要求：通过 swap 方法能交换 int、double、char 和 String 四种类型的数组中指定下标的两个数组元素值。

步骤 1、在 MyTools 包下创建 MyTools 类。

步骤 2、在该类中输入以下代码：

```
package myUtils;
public class MyTools {
    //交换int[]数组中下标为i和j的元素的值
    public static void swap(int[] b, int i, int j) {
        int c=b[i];
        b[i]=b[j];
        b[j]=c;
    }
    //交换double[]数组中下标为i和j的元素的值
    public static void swap(double[] b, int i, int j) {
        double c=b[i];
        b[i]=b[j];
        b[j]=c;
    }
    //交换char[]数组中下标为i和j的元素的值
    public static void swap(char[] b, int i, int j) {
        char c=b[i];
        b[i]=b[j];
        b[j]=c;
    }
    //交换String[]数组中下标为i和j的元素的值
    public static void swap(String[] b, int i, int j) {
        String c=b[i];
        b[i]=b[j];
        b[j]=c;
    }
}
```

步骤 3、在 com.ityw.basic.day05 包下创建 Test25 类，该类用于测试 MyTools 类中的 swap 方法，代码如下：

```
package com.ityw.basic.day04;
import java.util.Arrays;
import java.util.Scanner;
import myUtils.MyTools;
public class Test25 {
    /**
     * 测试myTools包下的MyTools类中的swap()
     */
    public static void main(String[] args) {
        Scanner scanner=new Scanner(System.in);
        System.out.println("1-int类型数组元素交换两个变量值");
        System.out.println("2-double类型数组元素交换两个变量值");
        System.out.println("3-char类型数组元素交换两个变量值");
        System.out.println("4-String类型数组元素交换两个变量值");
        int select=scanner.nextInt();
```

```

switch (select) {
case 1:
    int[] a1={35, 54, 65};
    System.out.println(Arrays.toString(a1));
    MyTools.swap(a1, 1, 2);
    System.out.println(Arrays.toString(a1));
    break;
default:
    break;
}
}
}

```

说明:

1、以上代码只测试了交换 int 类型数组下标为 1 和 2 的两个元素交换。测试交换另三个类型数组的代码请自行完成。

2、当输入到 MyTools.s 时, Eclipse 的智能导航自动提示 MyTools 中重载的四个 swap 方法, 显示这些方法的方法签名和返回类型等信息, 如图-9 所示:

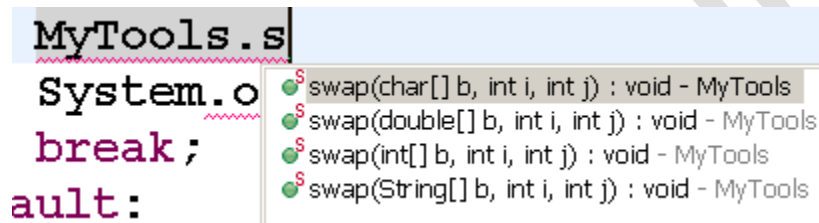


图-10

Java 在调用重载的方法时, 会根据实参的类型自动匹配 MyTools 类中的方法。

## 三、字符串操作函数

### (一)概述

String 类提供了许多实用的操作字符串的方法, 因这些方法都返回结果, 所以以下称为字符串函数。

### (二)常用函数(1)

#### 1、int length();

作用: 获取字符串的长度。

示例: "abc135".length 的值是6。

#### 2、public String substring(int beginIndex)

作用: 从字符串的指定位置开始提取子串。

参数- beginIndex: 提取子串的起始位置。

说明:

(1) 提取的子串从 beginIndex 至字符串结束。

(2) 字符串的位置从 0 开始计算。

示例: "abc135".substring(3) 的结果是 "135"

### 3、`public String substring(int beginIndex, int endIndex)`

作用：从字符串的指定的区域提取子串。

参数-beginIndex：提取子串的起始位置。

参数-endIndex：提取子串的结束位置+1。

说明：提取的子串区域：[beginIndex, endIndex)

示例："abc135".substring(0, 3)的结果是"abc"。

### 4、`public String trim();`

作用：将字符串开始和结束处可能出现的空格删除，返回没有空格的字符串。

示例：" Abc ".trim()的结果是"Abc"。

### 5、`public boolean equals(String target)`

作用：判断当前字符串与target代表的字符串是否相同。返回true或false。

参数-target：目标字符串。

示例："abc".equals("Abc")的结果是false

## (三)示例

### 【示例-9】打印字符图形(1)

输入1~9个字符串，例如12345，打印如下字符图形：

12345

2345

345

45

5

```
package com.ityw.basic.day04;
import java.util.Scanner;
public class Test09 {
    /**
     * 输入1~9个字符串，例如12345，打印如下字符图形：
     * 12345
     * 2345
     * 345
     * 45
     * 5
     */
    public static void main(String[] args) {
        Scanner scanner=new Scanner(System.in);
        System.out.println("输入1~9个字符串:");
        String s=scanner.next();
        s=s.trim();//将字符串s左右可能出现的空格删除
        for (int i = 0; i < s.length(); i++) {
            System.out.println(s.substring(i));
        }
    }
}
```

### 【示例-10】打印字符图形(2)

输入1~9个字符串，例如12345，打印如下字符图形：

12345

234

3

```
package com.ityw.basic.day04;
```

---

```
import java.util.Scanner;
/**
 * 输入1~9个字符串，例如12345, 打印如下字符图形:
 * 12345
 *  234
 *   3
 */
public class Test22 {
    public static void main(String[] args) {
        Scanner scanner=new Scanner(System.in);
        System.out.println("输入字符串1~9个:");
        String s;
        do{
            s=scanner.next().trim();
        }while((s.length()<=0 || s.length()>9)&& s.length()%2==0);
        String space=" ";
        for (int i = 0; i < s.length()/2+1; i++) {
            System.out.println(space+s.substring(i, s.length()-i));
            space+=" ";
        }
    }
}
```