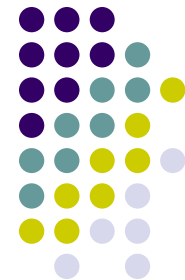


第四章 数组与字符串



4.1 数组

4.2 字符串

4.3 集合



4.1.1 数组的概念

- 数组由同一类型的一连串对象或者基本数据组成，并封装在同一个标识符（数组名称）下。
- 占用连续的内存地址
- 数组的静态性
 - 一旦创建就不能修改数组的长度

scores 数组引用

索引 = 3 的数组元素

scores[0]

scores[1]

scores[2]

scores[3]

scores[4]

scores[5]

scores[6]

scores[7]

scores[8]

scores[9]

80.5

69

73.5

60

91

84

78.5

66

87

45

数组元素的值

```
double[] scores = new double[10];
```

4.1.1 数组的概念



■ 数组是对象

- 动态初始化
- 可以赋值给Object类型的变量
- 在数组中可以调用类Object 的所有方法

■ 数组元素

- 数组中的变量被称作数组的元素
- 元素没有名字，通过数组名字和非负整数下标值引用数组元素。

4.1.2 数组的创建与引用



■ 数组声明（Declaration）

- 声明数组时无需指明数组元素的个数，也不为数组元素分配内存空间
- 不能直接使用，必须经过初始化分配内存后才能使用

■ 语法：

- `Type[] arrayName;`
 - `int[] intArray;`
 - `String[] stringArray;`
- `Type arrayName[];`
 - `int intArray[];`
 - `String stringArray[];`

4.1.2 数组的创建与引用



■ 数组的创建

- 用关键字new构成数组的创建表达式，可以指定数组的类型和数组元素的个数。
- 元素个数可以是常量也可以是变量
- 基本类型数组的每个元素都是一个基本类型的变量；引用类型数组的每个元素都是对象的引用。

4.1.2 数组的创建与引用



- `arrayName = new Type[components number];`
 - 例如：
`int[] ai; ai=new int[10];`
`String[] s; s=new String[3];`
 - 或者可以将数组的声明和创建一并执行
`int[] ai = new int[10];`
 - 可以在一条声明语句中创建多个数组
`String[] s1=new String[3], s2=new String[8];`
 - 数组下标从0开始。
`s2[0]`代表数组的第一个元素；
`s2[7]`代表数组的最后一个元素。



4.1.2 数组的创建与引用

■ 数组元素的初始化

- 声明数组名时，给出了数组的初始值，程序便会利用数组初始值创建数组并对它的各个元素进行初始化

```
int[] a={22, 33, 44, 55};
```

- 创建数组的时，如果没有指定初始值，数组便被赋予默认值初始值。
- 程序也可以在数组被构造之后改变数组元素值

类型	缺省值
byte	0x00
short	0
int	0
long	0
float	0
double	0
boolean	false
char	\u0000
Object	null



4.1.2 数组的创建与引用

■ 定义、创建、初始化数组

- 如果在创建数组时，就可以确定数组的值，则可以使用一条语句来完成数组的定义、创建、初始化

```
double[] scores = new double[] {63.0, 82.5, 47, 73};
```

或者：

```
double[] scores = {63.0, 82.5, 47, 73};
```

相当于：

```
double[] scores = new double[4];
```

```
scores[0] = 63.0;
```

```
scores[1] = 82.5;
```

```
scores[2] = 47;
```

```
scores[3] = 73;
```

- 注1: 如果用这种方法创建数组，则不能指定数组大小：

// 错误的写法：

```
double[] scores = new double[4]{63.0, 82.5, 47, 73};
```


4.1.2 数组的创建与引用



- 注2：简写的方式，必须在一行语句中完成。

// 错误的写法1:

```
double[] scores;  
scores = {63.0, 82.5, 47, 73};
```

// 错误的写法2:

```
double[] scores = new double[4];  
scores = {63.0, 82.5, 47, 73};
```

4.1.2 数组的创建与引用



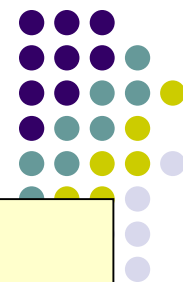
■ 数组的引用

- 通过下面的表达式引用数组的一个元素：

`arrayName[index]`

- 数组下标必须是 `int`, `short`, `byte`, 或者 `char`.
- 下标从零开始计数.
- 元素的个数即为数组的长度，可以通过 `arrayName.length` 引用
- 每个数组都有一个由 `public final` 修饰的成员变量：`length`，即数组含有元素的个数（`length` 可以是正数或零）
 - `scores.length`,
 - **注意：**`length` 不是方法，不能写为：`scores.length()`
- 元素下标最大值为 `length - 1`，如果超过最大值，将会产生数组越界异常(`ArrayIndexOutOfBoundsException`)

4.1.2 数组的创建与引用

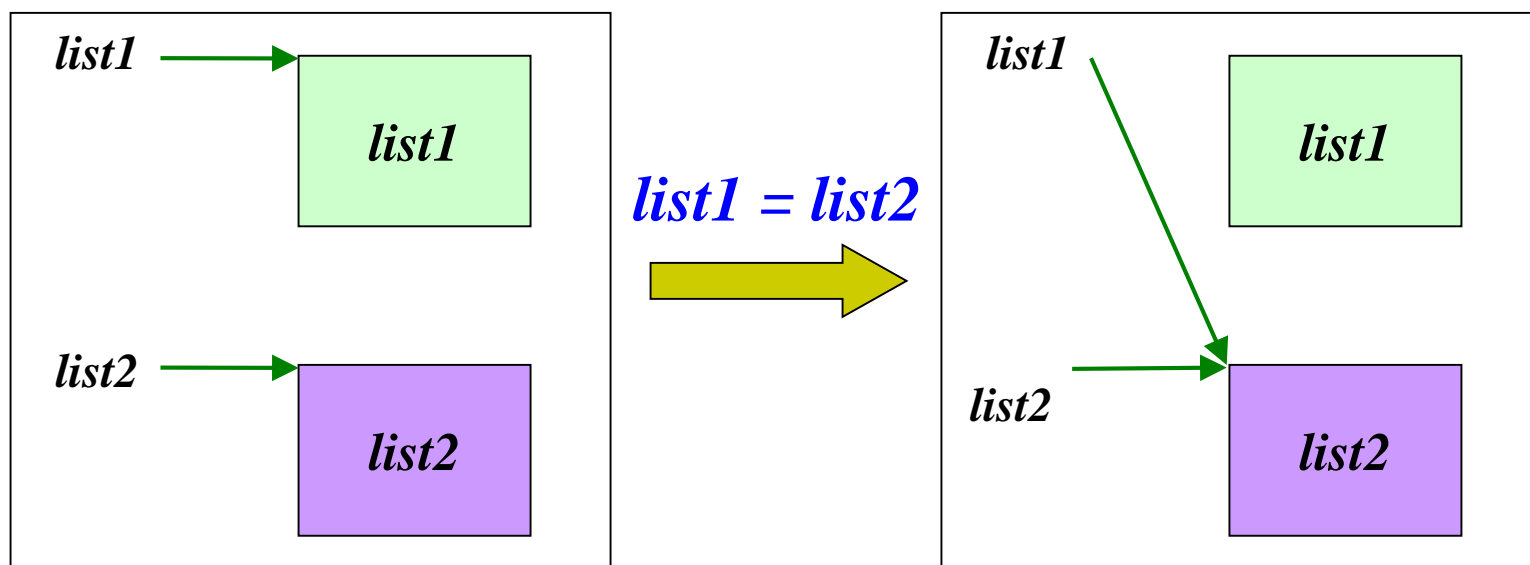


```
public class MyArray {  
    public static void main(String[] args){  
        int myArray[];           //声明数组  
        myArray=new int[10];      //创建数组  
        System.out.println("Index\t\tValue");  
        for(int i=0; i<myArray.length;i++)  
            System.out.println(i+"\t\t"+myArray[i]);  
        //证明数组元素默认初始化为0  
        //myArray[10]=100;    //将产生数组越界异常  
    }  
}
```

4.1.2 数组的创建与引用



- 拷贝数组
- 在程序中，经常需要复制数组的内容
 - 1、使用赋值语句。



在这种情况下，`list1`和`list2`指向相同的内存区域，修改`list1`的内容，会影响`list2`。**不是真正意义上的复制。**

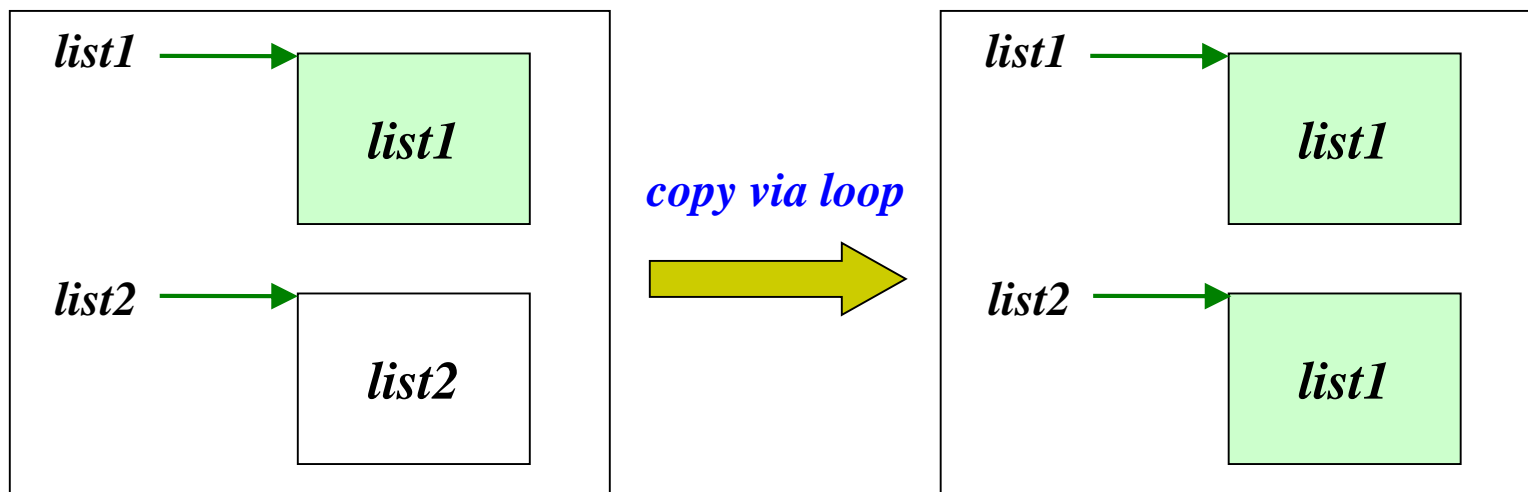
4.1.2 数组的创建与引用



- 2. 使用循环:

```
int[] list1 = {2, 3, 4, 5, 6};  
int[] list2 = new int[list1.length];  
  
for (int i=0; i<list1.length; i++)  
{  
    list2[i] = list1[i];  
}
```

- 此时，list1和list2指向不同的内存区域，但是内容相同。修改list1的内容，不会影响到list2.



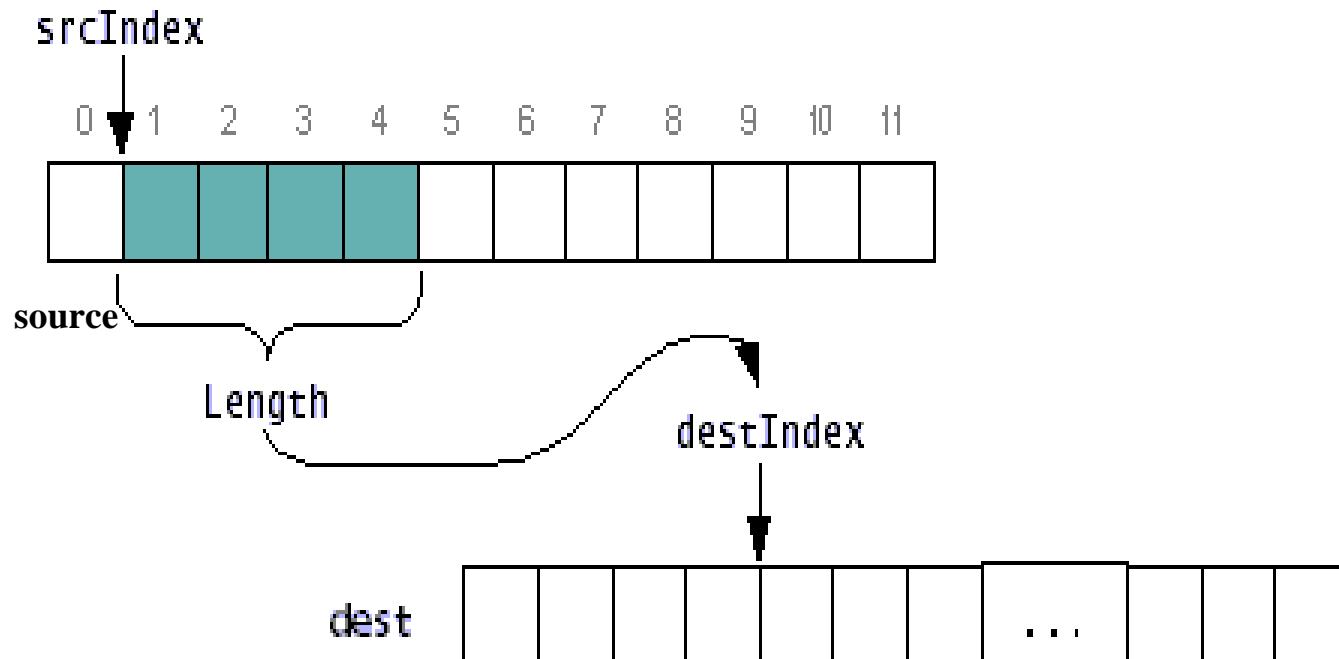
4.1.2 数组的创建与引用



- 3. 使用 `System` 类的 `arrayCopy()` 方法:

`public static void arraycopy(Object source ,
int srcIndex ,`

`Object dest int destIndex int length)`

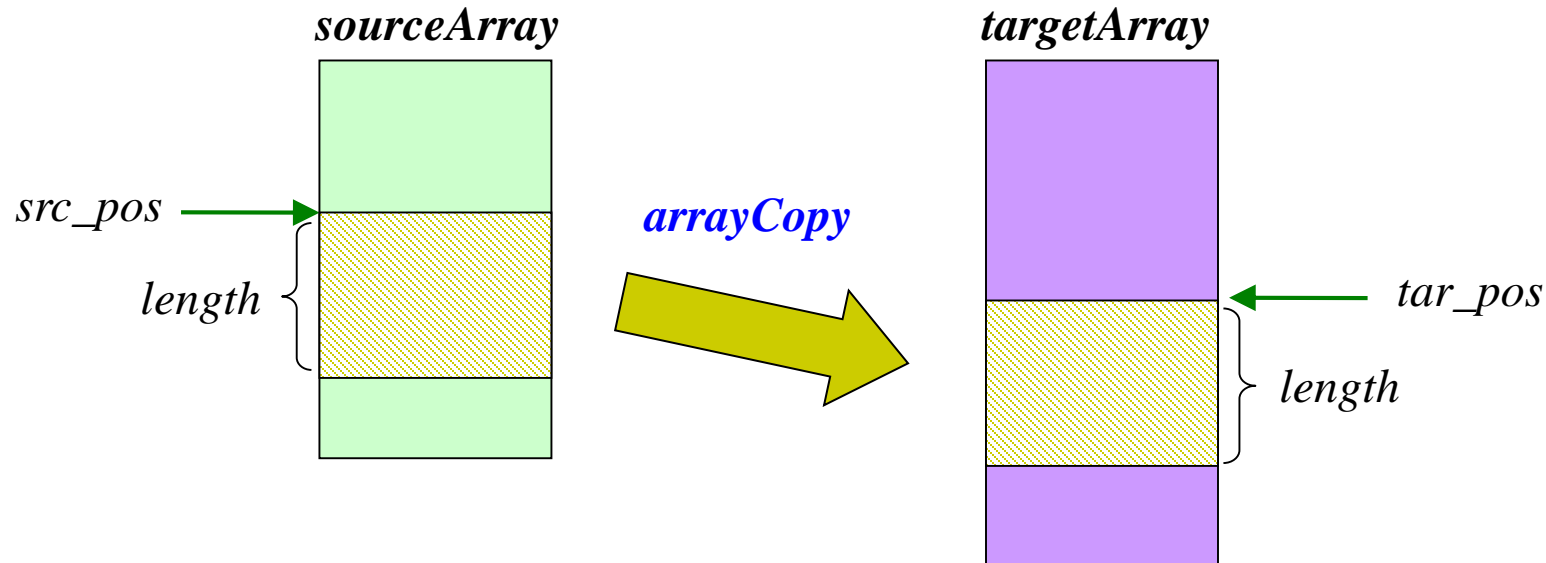


4.1.2 数组的创建与引用



● 3. 使用 System 类的 arrayCopy() 方法:

```
// sourceArray: 源数组  
// src_pos: 源数组中的起始位置  
// targetArray: 目标数组  
// tar_pos: 目标数组的起始位置  
// length: 要复制的数组元素的数量  
System.arrayCopy(sourceArray, src_pos,  
                  targetArray, tar_pos, length);
```



完全复制: `System.arrayCopy(sourceArray, 0, targetArray, 0, sourceArray.length);` ¹⁶

4.1.2 数组的创建与引用



■ 例子：将Array传递给方法

```
public static void printArray(int[] array)
{
    for (int i=0; i<array.length; i++)
    {
        System.out.print(array[i] + " ");
    }
}

public static void main(String[] args)
{
    int[] list = {1, 2, 3, 4, 5};
    printArray(list);

    printArray(new int[]{1, 2, 3, 4, 5});
}
```

Anonymous array (匿名数组)

这种 `new dataType[]{literal-0, literal-1, ..., literal-n};`
的写法，没有将数组赋值给一个引用变量，所以称为匿名数组

4.1.2 数组的创建与引用



- 在Java中，给方法(method)传递参数时，使用“按值传递”规则。
- 但是，在传递原始类型（int、double、.....）与传递数组的时候，不同
 - 当传递原始类型时，在方法内部修改本地变量，不会影响到外部变量
 - 当传递数组时，参数是一个数组的引用，该引用被传递给了方法。因此，在方法内部对数组元素的修改，会影响到外部的数组。

4.1.2 数组的创建与引用



```
public class PassByValueTest
{
    public static void main(String[] args)
    {
        int x = 1;
        int[] y = {10, 6, 23};
        int[] z = {27, 18, 36};
        test(x, y, z);

        System.out.println("x=" + x);
        System.out.println("y[0]=" + y[0]);
        System.out.println("z[0]=" + z[0]);
    }

    public static void test(int value, int[] array1, int[] array2)
    {
        value = 3;
        array1[0] = 11;
        array2 = new int[]{100, 200};
    }
}
```

输出结果:

x=1
y[0]=11
z[0]=27

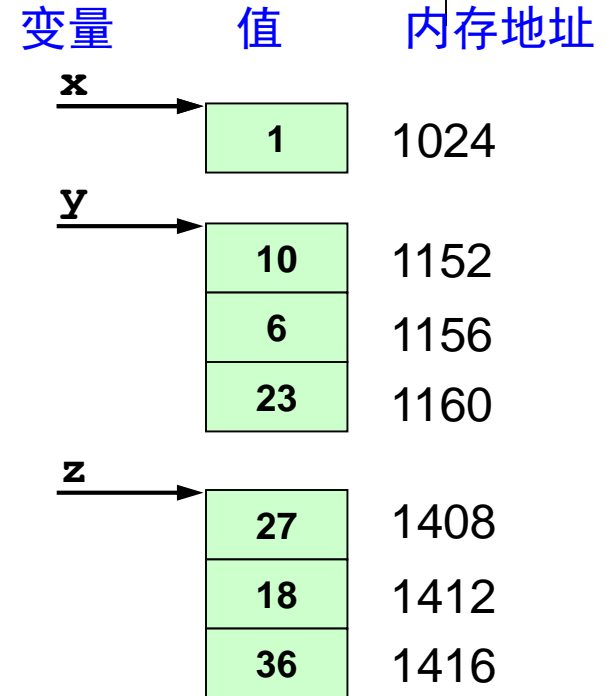
```

public class PassByValueTest
{
    public static void main(String[] args)
    {
        int x = 1;
        int[] y = {10, 6, 23};
        int[] z = {27, 18, 36};
        test(x, y, z);

        System.out.println("x=" + x);
        System.out.println("y[0]=" + y[0]);
        System.out.println("z[0]=" + z[0]);
    }

    public static void test(int value,
        int[] array1, int[] array2)
    {
        value = 3;
        array1[0] = 11;
        array2 = new int[]{100, 200};
    }
}

```



```

public class PassByValueTest
{
    public static void main(String[] args)
    {
        int x = 1;
        int[] y = {10, 6, 23};
        int[] z = {27, 18, 36};
        test(x, y, z);

        System.out.println("x=" + x);
        System.out.println("y[0]=" + y[0]);
        System.out.println("z[0]=" + z[0]);
    }

    public static void test(int value,
        int[] array1, int[] array2)
    {
        value = 3;
        array1[0] = 11;
        array2 = new int[]{100, 200};
    }
}

```



变量	值	内存地址
array1 →	1	1024
	10	1152
	6	1156
	23	1160
array2 →	27	1408
	18	1412
	36	1416
value →	3	2432

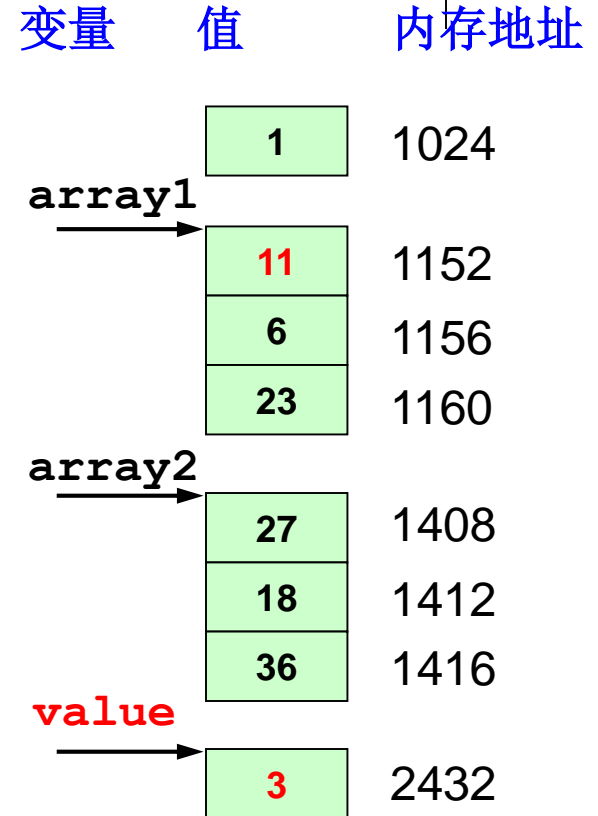
```

public class PassByValueTest
{
    public static void main(String[] args)
    {
        int x = 1;
        int[] y = {10, 6, 23};
        int[] z = {27, 18, 36};
        test(x, y, z);

        System.out.println("x=" + x);
        System.out.println("y[0]=" + y[0]);
        System.out.println("z[0]=" + z[0]);
    }

    public static void test(int value,
        int[] array1, int[] array2)
    {
        value = 3;
        array1[0] = 11;
        array2 = new int[]{100, 200};
    }
}

```



```

public class PassByValueTest
{
    public static void main(String[] args)
    {
        int x = 1;
        int[] y = {10, 6, 23};
        int[] z = {27, 18, 36};
        test(x, y, z);

        System.out.println("x=" + x);
        System.out.println("y[0]=" + y[0]);
        System.out.println("z[0]=" + z[0]);
    }

    public static void test(int value,
        int[] array1, int[] array2)
    {
        value = 3;
        array1[0] = 11;
        array2 = new int[]{100, 200};
    }
}

```



变量 值 内存地址

array1 →	1	1024
	11	1152
	6	1156
	23	1160
value →	27	1408
	18	1412
	36	1416
	3	2432
array2 →	100	2688
	200	2692

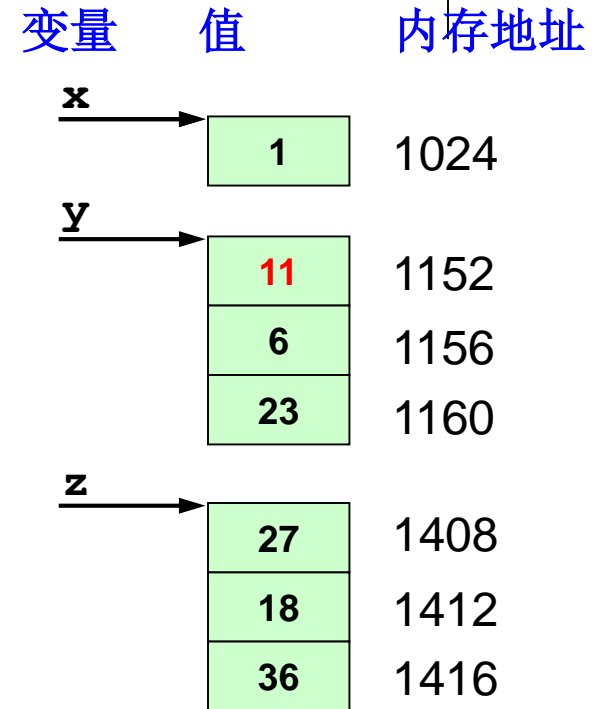
```

public class PassByValueTest
{
    public static void main(String[] args)
    {
        int x = 1;
        int[] y = {10, 6, 23};
        int[] z = {27, 18, 36};
        test(x, y, z);

        System.out.println("x=" + x);
        System.out.println("y[0]=" + y[0]);
        System.out.println("z[0]=" + z[0]);
    }

    public static void test(int value,
        int[] array1, int[] array2)
    {
        value = 3;
        array1[0] = 11;
        array2 = new int[]{100, 200};
    }
}

```



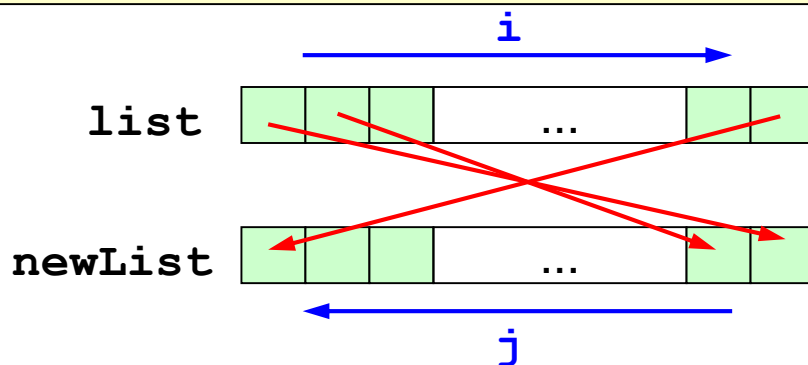
在方法中返回Array



```
public class ReverseListTest
{
    public static void main(String[] args)
    {
        int[] list1 = {1, 2, 3, 4};
        int[] reverseList = reverse(list1);
    }

    public static int[] reverse(int[] list)
    {
        int[] newList = new int[list.length];

        for (int i = 0, j = newList.length - 1; i < list.length; i++, j--)
        {
            newList[j] = list[i];
        }
        return newList;
    }
}
```



4.1.2 多维数组



- 最常用的多维数组是二维数组

```
int[ ][ ] a = new int[3][4];
```

- 二维数组可以理解成如下图所示的表格

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

行的下标值

列的下标值

4.1.2 多维数组



■ 二维数组的声明和构造

- `int[][] myArray ;`
 - `myArray` 可以存储一个指向2维整数数组的引用。其初始值为`null`。
- `int[][] myArray = new int[3][5] ;`
 - 建立一个数组对象，把引用存储到`myArray`。这个数组所有元素的初始值为零。
- `int[][] myArray = { {8,1,2,2,9}, {1,9,4,0,3}, {0,3,0,0,7} };`
 - 建立一个数组并为每一个元素赋值。

4.1.2 多维数组



■ 二维数组的长度

```
class UnevenExample2
{
    public static void main( String[ ] arg )
    {
        int[ ][ ] uneven = { { 1, 9, 4 }, { 0, 2}, { 0, 1, 2, 3, 4 } };
        System.out.println("Length is: " + uneven.length );
    }
}
```

■ 运行结果

- Length is: 3

4.1.2 多维数组



■ 每行的长度：

```
class UnevenExample3{
    public static void main( String[] arg ) {
        // 声明并构造一个2维数组
        int[ ][ ] uneven = { { 1, 9, 4 }, { 0, 2 }, { 0, 1, 2, 3, 4 } };
        // 数组的长度 (行数)
        System.out.println("Length of array is: " + uneven.length );
        // 数组每一行的长度 (列数)
        System.out.println("Length of row[0] is: " + uneven[0].length );
        System.out.println("Length of row[1] is: " + uneven[1].length );
        System.out.println("Length of row[2] is: " + uneven[2].length );
    }
}
```

运行结果：

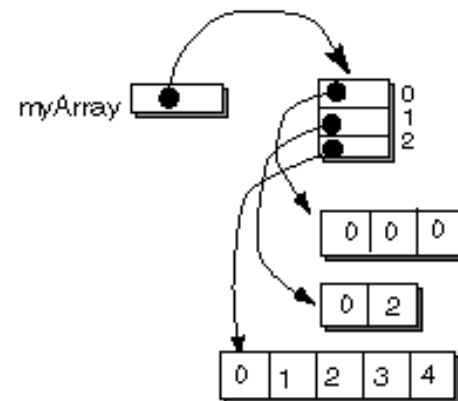
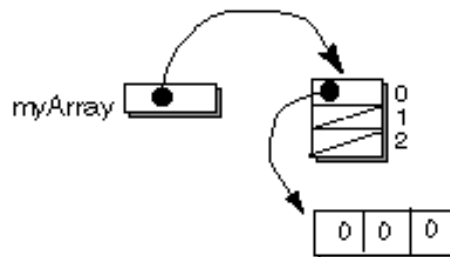
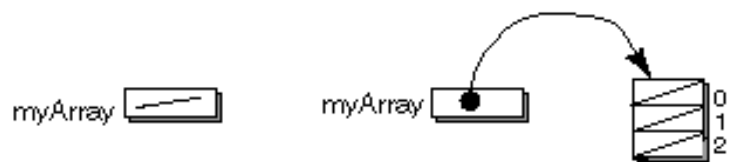
```
Length of array is: 3
Length of row[0] is: 3
Length of row[1] is: 2
Length of row[2] is: 5
```

4.1.2 多维数组



■ 多维数组的创建和引用

```
int[ ][ ] myArray;  
myArray = new int[3][ ];  
myArray[0] = new int[3];
```



4.1.3 对象数组



■ 数组

- 在Java提供的存储及随机访问对象序列的各种方法中，数组是效率最高的一种
 - 类型检查
 - 边界检查
- **优点：**数组知道其元素的类型、编译时类型检查、大小已知
- **代价：**数组对象的大小是固定的，在生存期内大小不可变

■ 对象数组——用数组存储对象

- 数组元素是类的对象
- 所有元素具有相同的类型
- 每个元素都是一个对象的引用

4.1.3 对象数组



■ 对象数组的初始化

- 静态初始化：在声明和定义数组的同时对数组元素进行初始化，例如：

```
BankAccount[] accounts = {  
    new BankAccount("Zhang", 100.00),  
    new BankAccount("Li", 2380.00),  
    new BankAccount("Wang", 500.00),  
    new BankAccount("Liu", 175.56),  
    new BankAccount("Ma", 924.02)};
```

- 动态初始化：使用运算符new，需要经过两步：

- 首先给数组分配空间

```
type arrayName[ ]=new type[arraySize];
```

- 然后给每一个数组元素分配空间

```
arrayName[0]=new type(paramList);
```

```
...
```

```
arrayName[arraySize-1]=new type(paramList);
```

4.1.3 对象数组



- 例子
- 使用数组存储一个班的学生信息及考试成绩。学生信息包括学号、姓名、三门课（英语、数学、计算机）的成绩及总成绩。
- 首先声明**学生类Student**
 - 属性包括
 - 学号（id），姓名（name），英语成绩（eng），数学成绩（math），计算机成绩（comp），总成绩（sum）
 - 方法包括
 - 构造方法，get方法，set方法，toString方法，equals方法，compare方法（比较两个学生的总成绩, 结果分大于，小于，等于），sum方法（计算总成绩）


```
import java.io.*;
public class Student implements Serializable {
    private String id;           //学号
    private String name;         //姓名
    private int eng;             //英语成绩
    private int math;            //数学成绩
    private int comp;            //计算机成绩
    private int sum;             //总成绩
    public Student(String id,String name,int eng,int math,int comp){
        this.id=id;
        this.name=name;
        this.eng=eng;
        this.math=math;
        this.comp=comp;
        sum();                   //计算总成绩
    }
    public Student(Student s){
        this.id=s.id;
        this.name=new String(s.name);
        this.eng=s.eng;
        this.math=s.math;
        this.comp=s.comp;
        sum();                   //计算总成绩
    }
}
```

```
public void setId(String id) { this.id=id;}
public void setName(String name) { this.name=name;}
public void setEng(int eng) {
    this.eng=eng;
    sum();          //计算总成绩
}
public void setMath(int math){
    this.math=math;
    sum();          //计算总成绩
}
public void setComp(int comp){
    this.comp=comp;
    sum();          //计算总成绩
}
public String getId() { return id; }
public String getName() { return name; }
public int getEng() { return eng; }
public int getMath() { return math; }
public int getComp() { return comp; }
public int getSum() { return sum; }
void sum() { this.sum=eng+math+comp; }
```

```
public String toString(){
    return getId() + "\t"+getName() + "\t"+getEng() + "\t"+getMath()
        +"\t"+getComp() + "\t"+getSum();
}
public boolean equals(Object x) {
    if (this.getClass() != x.getClass())
        return false;
    Student b = (Student) x;
    return (this.getId().equals(b.getId()));
}
```

//比较成绩大小,当前对象成绩比参数对象成绩大时返回1,相等时返回0,其它返回-1.

```
public int compare(Student A){
    if(this.getSum()>A.getSum())
        return 1;
    else if(this.getSum()==A.getSum())
        return 0;
    else return -1;
}
}
```

4.1.3 对象数组



■ 下面声明**班级类StudentClass**：

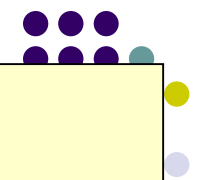
- 属性包括
 - 班级名称（name），容量（capacity），学生（students），实际人数（size）。
- 方法包括
 - 构造方法，get方法，set方法，toString方法。

```
public class StudentClass{
    private String name;           //班级名称
    static int capacity = 40;      //最大容量
    private Student [] students;   //学生
    private int size;              //实际人数
    public StudentClass(String name, int size){
        this.name = name;
        this.size = size;
        students = new Student[capacity];
    }
    public String getName(){
        return name;
    }
    public int getCapacity(){
        return capacity;
    }
    public Student[] getStudents(){
        return students;
    }
    public int getSize(){
        return size;
    }
}
```

```
public void setName(String name){
    this.name = name;
}
public void setCapacity(int capacity){
    this.capacity = capacity;
}
public void setSize(int size){
    this.size = size;
}
public void setStudents(Student[] students){
    for (int i = 0; i<size; i++)
        this.students[i] = new Student(students[i]);
}
public String toString(){
    String s;
    s = " 班级:" + name + "\t" + "容量:" + capacity + "\t" +
        "实际人数:" + size + "\n\n";
    s = s + "学号" + "\t" + "姓名" + "\t" + "英语" + "\t" +
        "数学" + "\t" + "计算机" + "\t" + "总成绩\n";
    for (int i=0; i<size; i++)
        s = s + students[i].getId()+"\t"+students[i].getName()+"\t"
            +students[i].getEng()+"\t"+students[i].getMath()+"\t"
            +students[i].getComp()+"\t"+students[i].getSum()+"\n";
    return s;
}
}
```



- 下面声明测试类Tester1，为测试简单，仅生成具有5名学生的班级，5名学生的信息从键盘输入，为了避免以后再重复输入，可将输入的学生信息保存到文件中。



```
import java.io.*;

public class Tester1{
    public static void main(String args[]){
        Student[] students;
        StudentClass aClass = new StudentClass("软件0201",5);
        students=new Student[5];
        for (int i=0; i<5; i++)
            students[i] = new Student(getAStudent(i+1));
        aClass.setStudents(students);
        System.out.println(aClass);
        try {
            FileOutputStream fo = new FileOutputStream("stu.ser");
            ObjectOutputStream so = new ObjectOutputStream(fo);
            for (int i=0; i<5; i++)
                so.writeObject(students[i]);
            so.close();
        }
        catch(Exception e)
        {
            System.out.println(e) ;
        }
    }
}
```



```
public static Student getAStudent(int i){
    Student studenti;
    System.out.println("输入第" + i + "个学生的信息:");
    System.out.print("学号:");
    String id = Keyboard.getString();
    System.out.print("姓名:");
    String name = Keyboard.getString();
    System.out.print("英语成绩:");
    int eng = Keyboard.getInteger();
    System.out.print("数学成绩:");
    int math = Keyboard.getInteger();
    System.out.print("计算机成绩:");
    int comp = Keyboard.getInteger();
    studenti = new Student(id,name,eng,math,comp);
    return studenti;
}
}
```

■ 运行结果如下（其中学生信息的输入只显示一部分）：



输入第1个学生的信息:

学号:250201

姓名:李红

英语成绩:88

数学成绩:76

计算机成绩:60

输入第2个学生的信息:

.....

班级:软件0201 容量:40 实际人数:5

学号	姓名	英语	数学	计算机	总成绩
250201	李红	88	76	60	224
250202	张林	78	67	80	225
250203	董玉梅	86	80	75	241
250204	张力	70	68	75	213
250205	何为	80	90	78	248

第四章 数组与字符串



4.1 数组

4.2 字符串

4.3 集合

4.2.1 String类



■ String类

- 该类字符串对象的值和长度都不变化
- 称为常量字符串

■ 生成String类对象的方法

- 可以这样生成一个常量字符串

```
String aString;
```

```
aString = "This is a string"
```

- 调用构造方法生成字符串对象

```
new String();
```

```
new String(String value);
```

```
new String(char[] value);
```

```
new String(char[] value, int offset, int count);
```

```
new String(StringBuffer buffer);
```

4.2.1 String类



■ 构造一个字符串

```
String stringRefVal = new String(stringLiteral);
```

■ 由于字符串使用频繁，所以Java提供了创建字符串的快速初始化方法：

```
String stringRefVal = "some string";
```

■ 例：

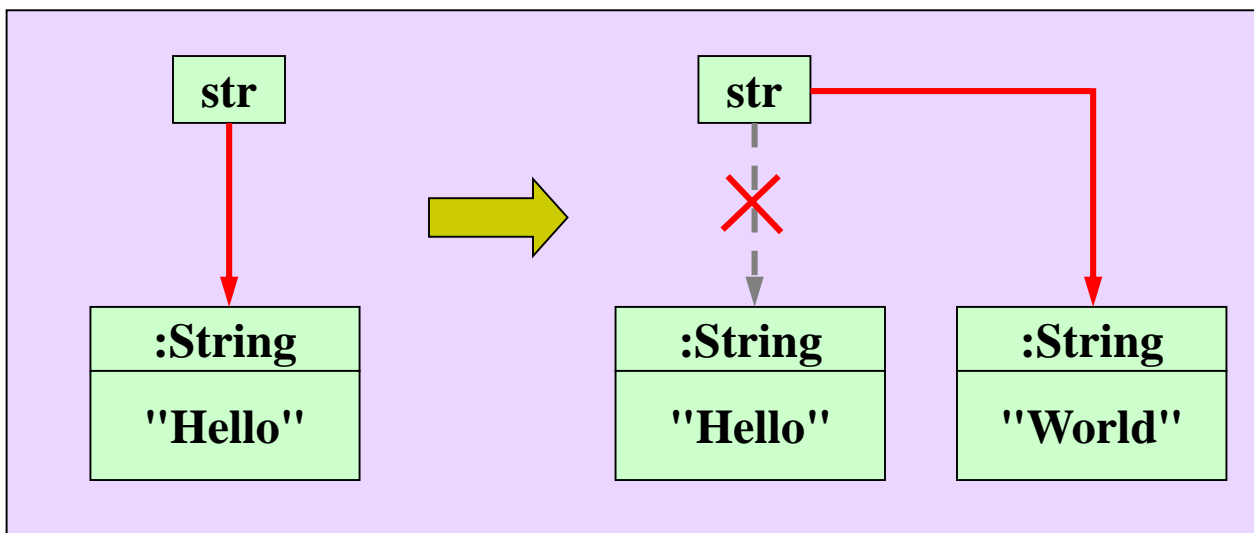
```
String message1 = "Hello World";  
String message2 = new String("Hello World");  
String message3 = "";  
String message4 = new String("");
```

4.2.1 String类



- String对象是永久的 (Immutable)
 - String对象是永久的，它的内容不能改变
 - 下面的代码能够改变字符串的内容吗？

```
String str = "Hello";  
str = "World";
```



可以看到，在执行 `str = "World"` 时，会产生一个新的字符串对象。

4.2.2 String类的常用方法



- String类有
 - 15 个构造方法
 - 65 个方法
- 详细说明见 JDK 文档



4.2.2 String类的常用方法

名称	解释
int length ()	返回字符串中字符的个数
char charAt (int index)	返回序号index处的字符
int indexOf (String s)	在接收者字符串中进行查找，如果包含子字符串s，则返回匹配的第一个字符的位置序号，否则返回-1
String substring (int begin, int end)	返回接收者对象中序号从begin开始到end-1的子字符串
public String[] split (String regex) public String[] split (String regex, int limit)	以指定字符为分隔符，分解字符串
String concat (String s)	返回接收者字符串与参数字符串s进行连接后的字符串

4.2.2 String类的常用方法



名称	解释
String replace (char oldChar, char newChar);	将接收者字符串的oldChar替换为newChar
int compareTo (String s);	将接收者对象与参数对象进行比较
boolean equals (String s);	接收者对象与参数对象的值进行比较
String trim ();	将接收者字符串两端的空字符串都去掉
String toLowerCase ()	将接收者字符串中的字符都转为小写
String toUpperCase ()	将接收者字符串中的字符都转为大写

4.2.2 String类的常用方法--字符串长度



- 使用 length() 方法得到字符串的长度：
 - 由于Java中字符串是用 Unicode 存储的，因此 length() 得到的是 Unicode 字符的个数
 - 不管是中文、英文字符，每个字符长度都是 1

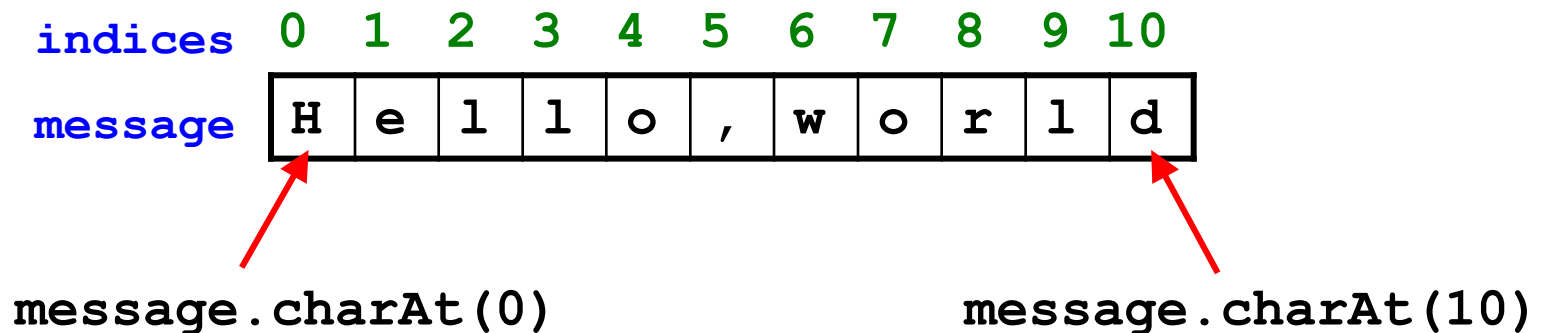
```
String str1 = "Long time no see";  
System.out.println(str1.length()); // 输出 16  
  
String str2 = "河海大学";  
System.out.println(str2.length()); // 输出 4  
  
String str3 = "你happy了";  
System.out.println(str3.length()); // 输出 7
```

- 注意：
 - String 的 length() 是方法
 - Array 的 length 不是方法

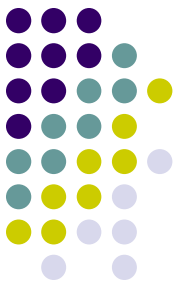
4.2.2 String类的常用方法



- 获得字符串中的单个字符
- 使用 `charAt(index)` 方法
 - 如 `message.charAt(5)`;
 - 索引从 0 开始, 到 `message.length()-1` (如果`message`长度不为0)
 - 注意, 不能用 `message[5]`



`message.length()` is 11



4.2.2 String类的常用方法

■ 字符串连接

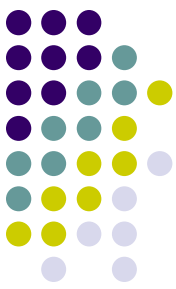
■ 两种方法：

- 使用 String 的 concat 方法
- 字符串直接相加（用"+"）

```
String str1 = "Hello";  
String str2 = "World";  
  
String str3 = str1.concat(str2);    // 方法一  
  
String str4 = str1 + str2;          // 方法二
```

- 多个字符串相加，支持连写：

```
String str = str1 + str2 + str3 + str4;  
String str = str1.concat(str2).concat(str3).concat(str4);
```



4.2.2 String类的常用方法--提取子串

- 可以使用charAt 方法从字符串中提取单个字符。也可以使用String 类中的substring方法从字符串中提取子串。

```
String message = "Hello,world";  
String str = message.substring(0, 6) + "Java";
```

indices	0	1	2	3	4	5	6	7	8	9	10
message	H	e	l	l	o	,	w	o	r	l	d

`message.substring(1, 5)` `message.substring(6)`

substring 有两个方法:

1) `public String substring(int beginIndex, int endIndex)`

beginIndex: 开始点的索引, 包含本字符;

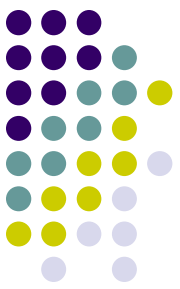
endIndex: 结束点的索引, 不包含本字符

得到从beginIndex到endIndex之间的子串

2) `public String substring(int beginIndex)`

beginIndex: 开始点的索引, 包含本字符

得到从beginIndex开始直到字符串结束的字符串



4.2.2 String类的常用方法

- **字符串比较**：字符串值是否相等
- 使用 equals 方法，而不是 ==
 - ==：比较的是字符串**对象引用的值**，即判断是否指向相同的对象
 - equals() 方法：比较的是字符串的**内容**

```
public boolean equals(Object anObject)
```

说明：比较两个对象是否相等

参数：anObject: 待比较的对象

返回值：= true：说明相等； = false：说明不相等

```
String str1 = new String("Hello,world");  
String str2 = "Hello,world";
```

```
System.out.println(str1 == str2);           // 输出 false  
System.out.println(str1.equals(str2));       // 输出 true
```

4.2.2 String类的常用方法



■ 字符串比较：compareTo() 方法

```
public int compareTo(String anotherString)
```

说明：按照字典序来进行比较两个字符串，如 `abcd < abd`

字符之间的比较，按照其 Unicode 进行

参数：anotherString: 待比较的字符串

返回值：= 0：说明两个字符串相等

> 0：说明 `this > anotherString`

< 0：说明 `this < anotherString`

```
String str1 = new String("Hello,world");  
String str2 = "Hello,world";  
String str3 = "hello,world";  
System.out.println(str1.compareTo(str2)); // 输出 0  
System.out.println(str1.compareTo(str3)); // 输出一个负数
```

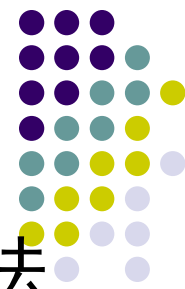
注：String 有一个 `compareToIgnoreCase()` 方法，执行的是不区分大小写的比较。

4.2.2 String类的常用方法



- 字符串转换
- 一旦字符串被创建之后它的内容就不能改变。但是可以使用以下方法将字符串转换为新字符串
- 常用的转换函数有：
 - `toLowerCase()`：转换为全小写
 - `toUpperCase()`：转换为全大写
 - `trim()`：去掉头尾的空格
 - `replace(oldChar, newChar)`：将字符串中的 `oldChar` 用 `newChar` 来代替
- 详细说明请参见 JDK 文档

4.2.2 String类的常用方法



■ 在字符串中找到一个字符或子串：indexOf() 方法

```
public int indexOf(int ch);  
public int indexOf(int ch, int fromIndex);  
public int indexOf(String str);  
public int indexOf(String str, int fromIndex);  
  
public int lastIndexOf(int ch);  
public int lastIndexOf(int ch, int fromIndex);  
public int lastIndexOf(String str);  
public int lastIndexOf(String str, int fromIndex);
```

说明：在本字符串中，查找一个字符/字符串。可以指定开始查找的位置。

indexOf：从前向后查找；lastIndexOf：从后向前查找

返回值：indexOf：得到第一次出现的位置。

lastIndexOf：得到最后一次出现的位置。

如果找不到，则返回 -1

4.2.2 String类的常用方法



```
"Welcome to Java".indexOf('W');           // returns 0.
"Welcome to Java".indexOf('x');           // returns -1.
"Welcome to Java".indexOf('o', 5);        // returns 9.
"Welcome to Java".indexOf("come");        // returns 3.
"Welcome to Java".indexOf("Java", 5);     // returns 11.
"Welcome to Java".indexOf("java", 5);     // returns -1.
"Welcome to Java".lastIndexOf('a');       // returns 14.
"Welcome to Java".lastIndexOf('a', 13);   // returns 12.
"Welcome to Java".lastIndexOf("Java", 5); // returns -1.
```

indices 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

message	W	e	l	c	o	m	e		t	o		J	a	v	a
---------	---	---	---	---	---	---	---	--	---	---	--	---	---	---	---

4.2.2 String类的常用方法



■ 将字符和数字转换成字符串：valueOf() 方法

```
public static String valueOf(boolean b);  
public static String valueOf(char c);  
public static String valueOf(char[] data);  
public static String valueOf(char[] data, int offset, int count);  
public static String valueOf(double d);  
public static String valueOf(float f);  
public static String valueOf(int i);  
public static String valueOf(long l);  
public static String valueOf(Object obj);
```

说明：将字符或数字转换为字符串。

返回值：转换后的字符串

```
String str1 = String.valueOf(5.44); // str1="5.44"  
String str2 = String.valueOf(true); // str2="true"
```

4.2.2 String类的常用方法



- 举例：检测回文串
- 提示用户输入一个字符串，然后报告该串是否为回文串：如果从前向后读它和从后向前读它都一样，则称为回文串。

```
import javax.swing.JOptionPane;

public class CheckPalindrome {
    public static void main(String[] args) {
        String s = JOptionPane.showInputDialog("Enter a string:");
        String output = "";
        if (isPalindrome(s))
            output = s + " is a palindrome";
        else
            output = s + " is not a palindrome";
        JOptionPane.showMessageDialog(null, output);
    }
    public static boolean isPalindrome(String s) {
        int low = 0;
        int high = s.length() - 1;
        while (low < high) {
            if (s.charAt(low) != s.charAt(high))
                return false; // Not a palindrome
            low++;
            high--;
        }
        return true; // The string is a palindrome
    }
}
```



4.2.2 String类的常用方法

■ 字符串分解：使用 split() 方法

```
public String[] split(String regex);
```

说明：在本字符串中，根据正则表达式（regex），拆分此字符串。

regex：正则表达式。暂时可以简单理解为“拆分时使用的分隔符”

返回值：拆分后，得到的字符数组

```
String str = "Hello, world";  
String[] arr1 = str.split(",");  
// arr1: "Hello"      " world"  
  
String[] arr2 = str.split(" ");  
// arr2: "Hello,"     "world"  
  
String[] arr3 = str.split("o");  
// arr3: "Hell"       ", w"      "rld"
```

4.2.2 String类的常用方法



■ 字符串替换：使用 replace() 方法

```
public String replace(char oldChar, char newChar);  
public String replace(CharSequence target, CharSequence  
                        replacement);  
public String replaceAll(String regex, String  
                        replacement);  
public String replaceFirst(String regex, String  
                        replacement);
```

说明：

一个四个方法，用于字符串替换。

返回值：

替换后，得到的新字符串

```
String old = "Hello, world";  
// 1. 按字符替换, 全部替换  
String new1 = old.replace('o', '@');  
           // new1="Hell@, w@rld"  
// 2. 按字符串替换, 全部替换  
String new2 = old.replace("world", "china");  
           // new2="Hello, china"  
String new3 = old.replace("o", "@");  
           // new3="Hell@, w@rld"  
String new4 = old.replaceAll("o", "xyz");  
           // new4="Hellxyz, wxyzrld"  
  
// 3. 按字符串替换, 只替换第一次 (从左向右扫描)  
String new5 = old.replaceFirst("o", "xyz");  
           // new5="Hellxyz, world"  
  
// 注意: 替换时, 会从左向右扫描字符串  
String all_is_a = "aaa";  
String new6 = all_is_a.replace("aa", "b");  
           // new6="ba"
```


4.2.3 字符类 Character



- Java为每个原始类型提供了一个包装类（Wrapper Class）

原始类型	对应的包装类
char	Character
boolean	Boolean
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

- 这里先介绍 Character

4.2.3 字符类 Character



■ Character UML 类图（部分）：

java.lang.Character

+Character(value:char)

+charValue():char

+compareTo(anotherCharacter:Character):int

+equals(anotherCharacter:Character):boolean

+isDigit(ch:char):boolean

+isLetter(ch:char):boolean

+isLetterOrDigit(ch:char):boolean

+isLowerCase(ch:char):boolean

+isUpperCase(ch:char):boolean

+toLowerCase(ch:char):char

+toUpperCase(ch:char):char

构造函数

得到原始类型 char

比较两个 Character 的大小

比较两个 Character 是否相等

是否为数字

是否为字母

是否为字母或数字

是否为小写

是否为大写

转换为小写

转换为大写

4.2.3 字符类 Character



■ 举例

```
Character c = new Character('d');  
c.compareTo(new Character('a')); // 返回一个 >0 的值  
c.compareTo(new Character('f')); // 返回一个 <0 的值  
c.equals(new Character('b')); // 返回 false  
c.equals(new Character('d')); // 返回 true
```

举例：统计字符串中的每个字母



```
import javax.swing.JOptionPane;
public class CountEachLetter {
    public static void main(String[] args){
        String s = JOptionPane.showInputDialog("Enter a string:");
        int[] counts = countLetters(s.toLowerCase());
        String output = "";
        for (int i = 0; i < counts.length; i++){
            if (counts[i] != 0)
                output += (char) ('a' + i) + " appears " +
                    counts[i] + ((counts[i] == 1) ? " time\n" : " times\n");
        }
        JOptionPane.showMessageDialog(null, output);
    }
    public static int[] countLetters(String s){
        int[] counts = new int[26];
        for (int i = 0; i < s.length(); i++){
            if (Character.isLetter(s.charAt(i)))
                counts[s.charAt(i) - 'a']++;
        }
        return counts;
    }
}
```

4.2.4 StringBuffer类



- String是Immutable的
- 由于String的Immutable特性，因此每次对String的内容的修改，都会导致创建新的String对象
 - 如： `String str = "aa" + "bb" + "cc" + "dd"`
一共创建了多少个String对象？
 - 答：7个： "aa", "bb", "cc", "dd", "aabb", "aabbcc", "aabbccdd"
- 为了高效地进行字符串的“添加”、“插入”、“修改”操作，引入StringBuffer/StringBuilder类

4.2.4 StringBuffer类



■ StringBuffer类

- 其对象是可以修改的字符串
 - 字符的个数称为对象的长度(length)
 - 分配的存储空间称为对象的容量(capacity)
- 与String类的对象相比，执行效率要低一些
- 该类的方法不能被用于String类的对象

4.2.4 StringBuffer类



生成StringBuffer类的对象

- 顾名思义，StringBuffer 封装了一个字符串缓冲区，可以给字符串缓冲器中添加、插入或追加内容。使用方法和String类似。
- 构造方法：

```
// 创建一个空的 StringBuffer，长度为 16  
public StringBuffer();
```

```
// 创建一个空的 StringBuffer，长度为 capacity  
public StringBuffer(int capacity);
```

```
//和string参数一样的字符串。初始容量为 16 + str.length()  
public StringBuffer(String str);
```

4.2.5 StringBuffer类的常用方法

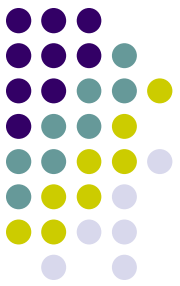


名称	解释
<code>int length ()</code>	返回字符串对象的长度
<code>int capacity()</code>	返回字符串对象的容量
<code>void ensureCapacity(int size)</code>	设置字符串对象的容量
<code>void setLength(int len)</code>	设置字符串对象的长度。如果len的值小于当前字符串的长度，则尾部被截掉
<code>char charAt(int index)</code>	返回index处的字符



4.2.5 StringBuffer类的常用方法

名称	解释
<code>void setCharAt(int index, char c)</code>	将index处的字符设置为c
<code>void getChars(int start, int end, char [] charArray, int newStart)</code>	将接收者对象中从start位置到end-1位置的字符拷贝到字符数组charArray中，从位置newStart开始存放
<code>StringBuffer reverse()</code>	返回将接收者字符串逆转后的字符串
<code>StringBuffer insert(int index, Object ob)</code>	将ob插入到index位置
<code>StringBuffer append(Object ob)</code>	将ob连接到接收者字符串的末尾



4.2.5 StringBuffer类的常用方法

■ 追加新内容：用 append() 方法

```
public StringBuffer append(boolean b);  
public StringBuffer append(char c);  
public StringBuffer append(char[] str);  
public StringBuffer append(double d);  
public StringBuffer append(float f);  
public StringBuffer append(int i);  
public StringBuffer append(long lng);  
public StringBuffer append(Object obj);  
public StringBuffer append(String str);  
public StringBuffer append(StringBuffer sb);
```

```
StringBuffer sb = new StringBuffer();  
sb.append("Hello");  
sb.append(' ');  
sb.append("World");  
String str = sb.toString();
```

// StringBuffer 支持连写:

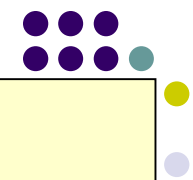
```
sb.append("Hello").append(' ').append("World");
```

4.2.5 StringBuffer类的常用方法



- 已知一个字符串，返回将字符串中的非字母字符都删除后的字符串

```
public class StringEditor {  
    public static String removeNonLetters(String original) {  
        StringBuffer aBuffer = new StringBuffer(original.length());  
        char aCharacter;  
        for (int i=0; i<original.length(); i++) {  
            aCharacter = original.charAt(i);  
            if (Character.isLetter(aCharacter))  
                aBuffer.append(new Character(aCharacter));  
        }  
        return new String(aBuffer);  
    }  
}
```

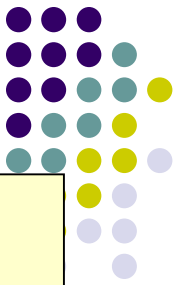


```
public class StringEditorTester {  
    public static void main(String args[]) {  
        String original = "Hello123, My Name is Mark,  
                           234I think you are my classmate?!!";  
        System.out.println(  
            StringEditor.removeNonLetters(original));  
    }  
}
```

■ 运行结果

HelloMyNameisMarkIthinkyouaremyclassmate

忽略既非字母又非数字的字符，判断是否为回文串



```
public class PalindromeIgnoreNonAlphanumeric {
    public static void main(String[] args) {
        String s = JOptionPane.showInputDialog("Enter a string:");
        String output = "Ignoring non-alphanumeric characters, \nis "
            + s + " a palindrome? " + isPalindrome(s);
        JOptionPane.showMessageDialog(null, output);
    }
    public static boolean isPalindrome(String s) {
        String s1 = filter(s);
        String s2 = reverse(s1);
        return s2.equals(s1);
    }
    public static String filter(String s) {
        StringBuffer strBuf = new StringBuffer();
        for (int i = 0; i < s.length(); i++) {
            if (Character.isLetterOrDigit(s.charAt(i))) {
                strBuf.append(s.charAt(i));
            }
        }
        return strBuf.toString();
    }
    public static String reverse(String s) {
        StringBuffer strBuf = new StringBuffer(s);
        strBuf.reverse();
        return strBuf.toString();
    }
}
```

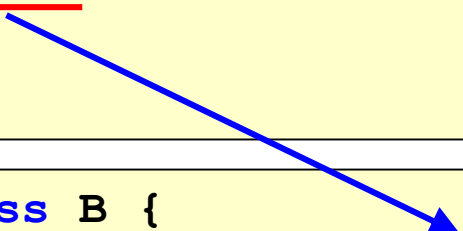
4.2.6 main 方法



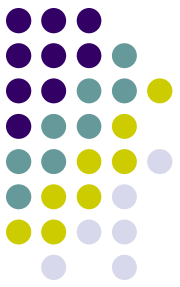
■ main 方法只是一个普通方法

- main方法与带有参数的普通方法一样。可以通过传递实参来调用main方法
- 例如，在B类中的main方法被A中的方法调用，如下所示：

```
public class A {  
    public static void main(String[] args) {  
        String[] strs = {"BeiJing", "ShangHai", "Xi'an"};  
        B.main(strs);  
    }  
}
```



```
public class B {  
    public static void main(String[] args) {  
        for (int i=0; i<args.length; i++) {  
            System.out.println(args[i]);  
        }  
    }  
}
```



4.2.6 main 方法

- main 方法又不是一个普通的方法
 - 它是Java执行程序的入口点
 - String[] args 参数，又称为“命令行参数”
- 命令行参数
 - 一般的格式：

```
java ApplicationName arg0 arg1 arg2 ... argn
```

Java
解释器

程序名称

命令行参数

- 在main 方法中，是从args[0], args[1], ..., args[n]中得到参数，它们分别对应命令行中的 arg0, arg1, ..., argn

4.2.6 main 方法



■ 举例：

```
public class ArgsTest {  
    public static void main(String[] args) {  
        for (int i=0; i<args.length; i++) {  
            System.out.println(args[i]);  
        }  
    }  
}
```

执行： java ArgsTest BeiJing ShangHai Xian

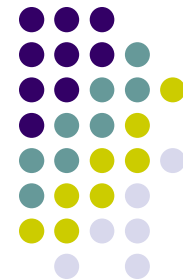
输出：

BeiJiong

ShangHai

Xian

第四章 数组与字符串



4.1 数组

4.2 字符串

4.3 集合

4.3 集合



■ 数组的优点

- 是Java提供的随机访问对象序列的最有效方法
- 是一个简单的线性序列，访问元素的速度较快

■ 数组的缺点

- 大小自创建以后就固定了，在其整个生存期内其大小不可改变
- 数组元素只能是同一类型

■ 集合

- 可动态改变其大小
- 可在序列中存储不同类型的数据

4.3 集合



■ 集合

- 把具有相同性质的一类东西，汇聚成一个整体
- 在Java2中有很多与集合有关的接口及类
- 它们被组织在以Collection及Map接口为根的层次结构中，称为集合框架
- 在Java2之前，在Java 1.0/1.1中，没有完整的集合框架。只有一些简单的可以自扩展的容器类
 - Vector —— 向量
 - Hashtable —— 哈希表

4.3.1 集合框架



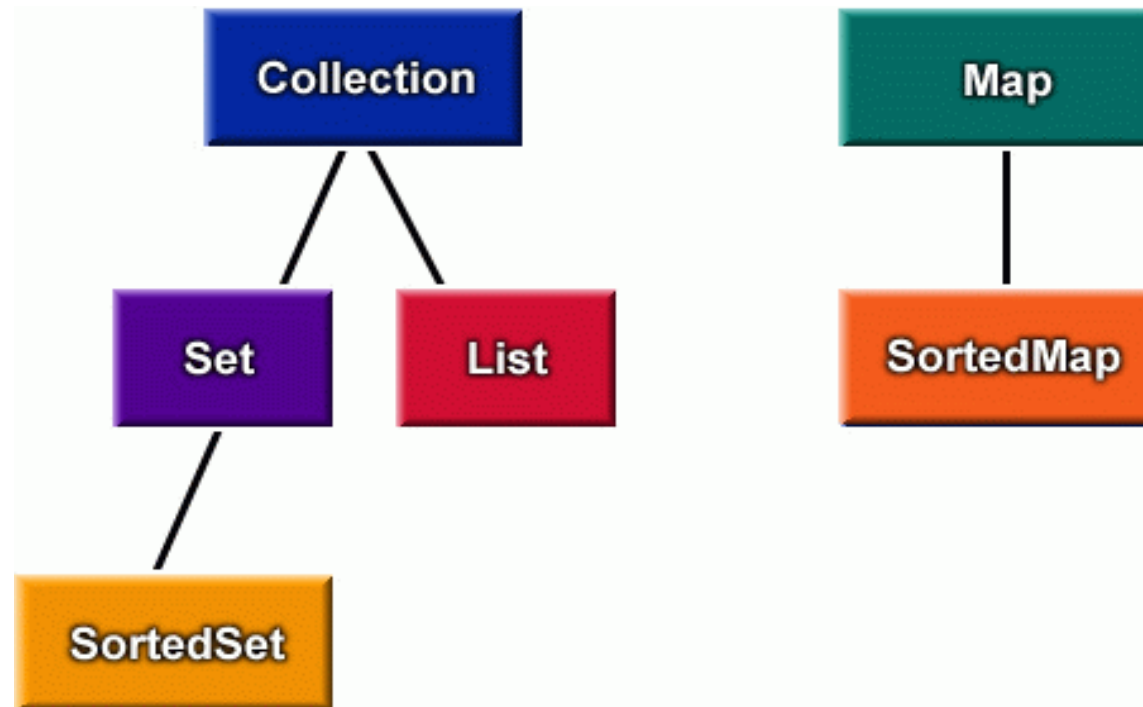
■ 集合框架(Java Collections Framework)

- 为表示和操作集合而规定的一种统一的标准体系结构
- 提供了一些现成的数据结构可供使用，程序员可以利用集合框架快速编写代码，并获得优良性能
- 包含三大块内容
 - 对外的接口：表示集合的抽象数据类型，使集合的操作与表示分开
 - 接口的实现：指实现集合接口的Java类，是可重用的数据结构
 - 对集合运算的算法：是指执行运算的方法，例如在集合上进行查找和排序



4.3.1 集合框架

- 对外的接口
- 集合框架接口
 - 声明了对各种集合类型执行的一般操作
 - 包括Collection、Set、List、SortedSet、Map、SortedMap
 - 基本结构如图

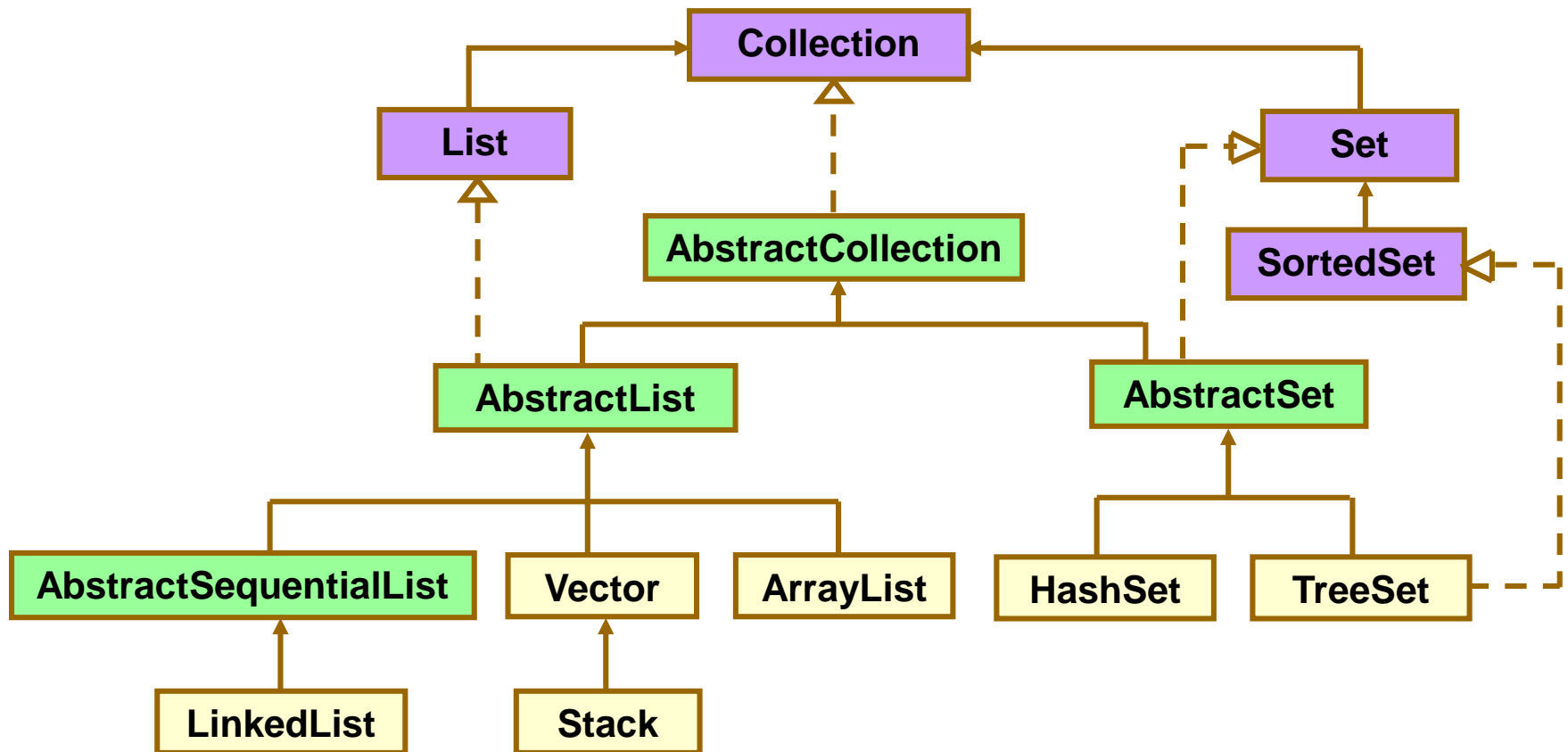


4.3.1 集合框架



■ Collection接口

- 类层次如图，包括4个接口、4个抽象类及6个具体类



4.3.1 集合框架



■ Collection接口

- 声明了一组操作成批对象的抽象方法：查询方法、修改方法
- 查询方法
 - `int size()` – 返回集合对象中包含的元素个数
 - `boolean isEmpty()` – 判断集合对象中是否还包含元素，如果没有任何元素，则返回true
 - `boolean contains(Object obj)` – 判断对象是否在集合中
 - `boolean containsAll(Collection c)` – 判断方法的接收者对象是否包含集合中的所有元素

4.3.1 集合框架



■ Collection接口

● 修改方法包括

- `boolean add(Object obj)` – 向集合中增加对象
- `boolean addAll(Collection c)` – 将参数集合中的所有元素增加到接收者集合中
- `boolean remove(Object obj)` – 从集合中删除对象
- `boolean removeAll(Collection c)` -将参数集合中的所有元素从接收者集合中删除
- `boolean retainAll(Collection c)` – 在接收者集合中保留参数集合中的所有元素，其它元素都删除
- `void clear()` – 删除集合中的所有元素

4.3.1 集合框架



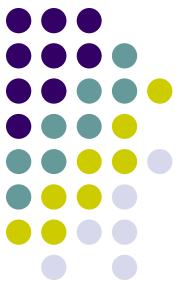
■ Set接口

- 扩展了Collection
- **禁止重复的元素**，是数学中“集合”的抽象
- 对equals和hashCode操作有了更强的约定，如果两个Set对象包含同样的元素，二者便是相等的
- 实现它的两个主要类是哈希集合(HashSet)及树集合(TreeSet)

■ SortedSet接口

- 一种特殊的Set
- 其中的元素是升序排列的，还增加了与次序相关的操作
- 通常用于存放词汇表这样的内容

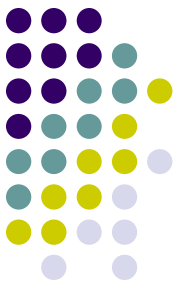
4.3.1 集合框架



■ List接口

- 扩展了Collection
- 可包含重复元素
- 元素是有顺序的，每个元素都有一个index值（从0开始）标明元素在列表中的位置
- 实现它的四个主要类是
 - Vector
 - ArrayList：一种类似数组的形式进行存储，因此它的随机访问速度极快
 - LinkedList：内部实现是链表，适合于在链表中间需要频繁进行插入和删除操作
 - 栈Stack

4.3.1 集合框架



■ Map接口

- 不是Collection接口的继承
- 用于维护键/值对(key/value pairs)
- 描述了从不重复的键到值的映射，是一个从关键字到值的映射对象
- 其中不能有重复的关键字，每个关键字最多能够映射到一个值

■ SortedMap接口

- 一种特殊的Map，其中的关键字是升序排列的
- 与SortedSet对等的Map，通常用于词典和电话目录等

4.3.1 集合框架



■ 接口的实现

- Collection没有直接的实现，只是作为其他集合接口的根
- 除Collection 以外，其余五个接口都有实现
- 主要的实现有
 - Set→HashSet
 - SortedSet→TreeSet
 - List→Vector / ArrayList / LinkedList
 - Map→HashMap
 - SortedMap→TreeMap

4.3.2 向量



■ Vector类的构造方法

- `Vector myVector = new Vector();` //初始容量为10
- `Vector myVector = new Vector(int cap);`
- `Vector myVector = new Vector(Collection col);`
 - 以参数col中的元素进行初始化
 - 也可用数组元素生成，但需先将数组转换成List对象，如
`String[] num = {"one", "two", "three", "four", "five"};`
`Vector aVector = new Vector(java.util.Arrays.asList(num));`

■ ArrayList的构造方法与Vector类似

- `ArrayList myList = new ArrayList();`
- `ArrayList myList = new ArrayList(int cap);`
- `ArrayList myList = new ArrayList(Collection col);`

4.3.2 向量



■ 常用方法

- **void add(Object obj)** —— 添加一个对象，如

```
Vector teamList = new Vector();  
teamList.add("Zhang Wei");  
teamList.add("Li Hong");
```

- **boolean addAll(Collection col)** —— 添加整个集合，如果接收者对象的结果有变化，则返回true，如

```
Vector teamList = new Vector();  
teamList.add("Zhang Wei");  
teamList.add("Li Hong");  
Vector yourList = new Vector();  
yourList.addAll(teamList);
```

4.3.2 向量



- `int size()` —— 返回元素的个数。
- `boolean isEmpty()` —— 如果不含元素，则返回true
- `Object get(int pos)` —— 返回指定位置的元素，如

```
Vector teamList = new Vector();
```

```
teamList.add("Zhang Wei");
```

```
teamList.add("Li Hong");
```

```
teamList.add("Yu Hongshu");
```

```
teamList.get(1);    // 返回 "Li Hong"
```

```
teamList.get(3);    // 产生例外 ArrayIndexOutOfBoundsException
```

4.3.2 向量



- `void set(int pos, Object obj)` —— 用参数对象替换指定位置的对象，如

```
Vector teamList = new Vector();  
teamList.add("Zhang Wei");  
teamList.add("Li Hong");  
teamList.add("Yu Hongshu");  
teamList.set(2, "Liu Na");  
System.out.println(teamList);  
    // 显示[Zhang Wei, Li Hong, Liu Na]  
teamList.set(3, "Ma Li");  
    // 产生例外ArrayIndexOutOfBoundsException
```


4.3.2 向量



- **boolean remove(Object obj)** ——去除给定对象的第一次出现，如果找到了对象，则返回true。去除一个对象后，其后面的所有对象都依次向前移动。如

```
Vector teamList = new Vector();  
teamList.add("Zhang Wei");  
teamList.add("Li Hong");  
teamList.add("Yu Hongshu");  
teamList.remove("Li Hong");  
teamList.remove("Wang Hong");//不做任何事，也不出现错误  
System.out.println(teamList); // 显示[Zhang Wei,Yu Hongshu]
```

4.3.2 向量



- **Object remove(int pos)** —— 去除给定位置的元素，并返回被去除的对象。如

```
Vector teamList = new Vector();  
teamList.add("Zhang Wei");  
teamList.add("Li Hong");  
teamList.add("Yu Hongshu");  
teamList.remove(0);      //去除Zhang Wei  
teamList.remove(0);      //去除 Li Hong  
System.out.println(teamList); // 显示[Yu Hongshu]  
teamList.remove(1); //产生例外 ArrayIndexOutOfBoundsException
```

4.3.2 向量



- `void clear()` —— 去除所有的元素
- `boolean contains(Object obj)` —— 返回是否包含指定的对象，如果包含则返回true；否则，返回false
- `boolean containsAll(Collection col)` —— 返回是否包含参数col中的所有对象
- `int indexOf(Object obj)` —— 返回给定对象在Vector /ArrayList中第一次出现的位置，如不存在，则返回-1。如

```
Vector teamList = new Vector();  
teamList.add("Zhang Wei");  
teamList.add("Li Hong");  
teamList.indexOf("Li Hong");    // 返回1。  
teamList.indexOf("Zhang Li");   // 返回-1。
```