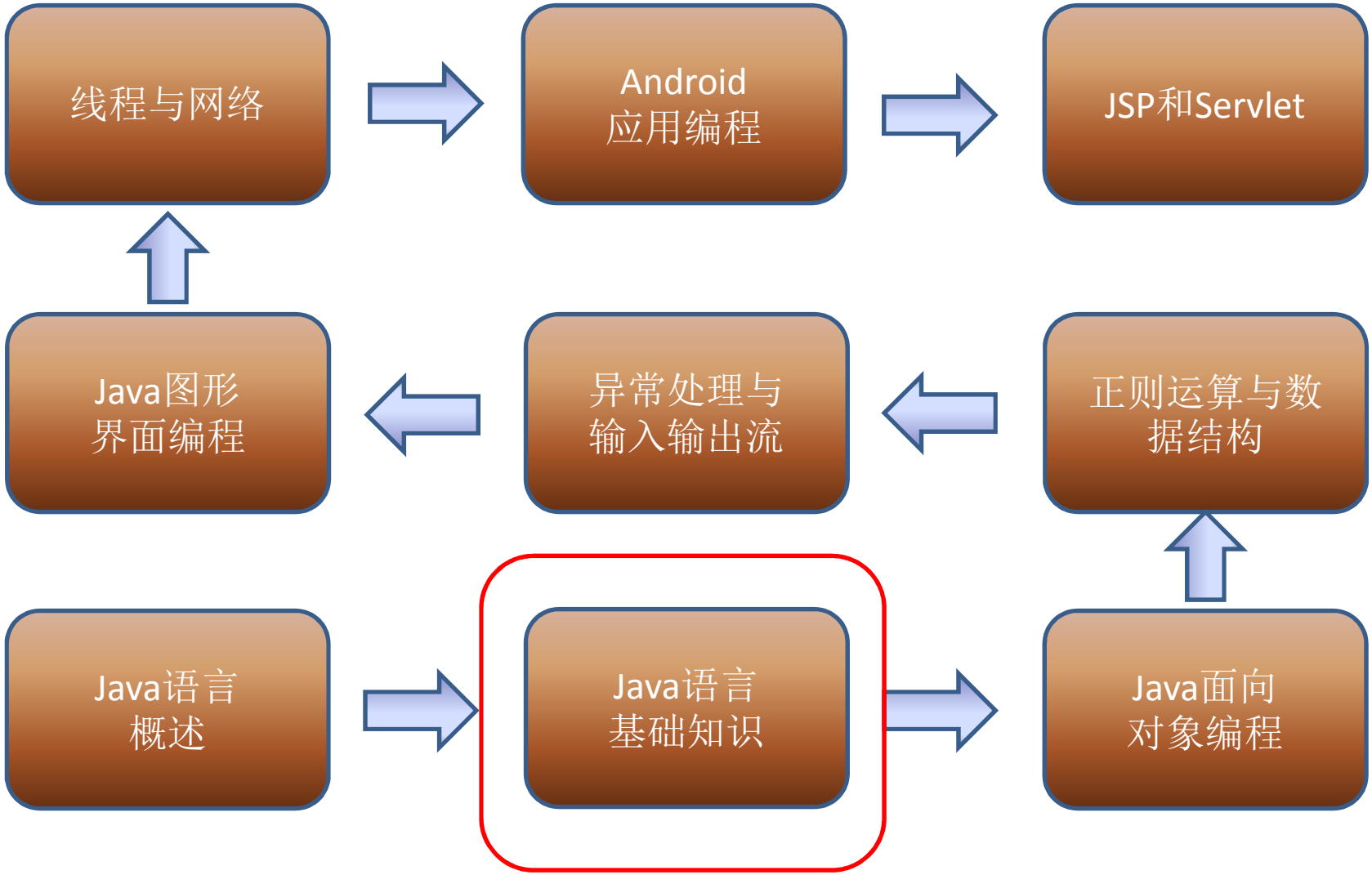
第二讲 Java语言基础知识

课程内容安排


线程与网络
Android
应用编程
JSP和Servlet
Java图形
界面编程
异常处理与
输入输出流
正则运算与数
据结构
Java语言
概述
Java语言
基础知识
Java面向
对象编程

课前思考

① 你认识boolean, int, throw, catch吗？

② 你定义过变量名 $1000D, 2mail吗？

③ 你认识这些数吗？ 0123，0x123，0.123，123e4，123.4f

④ 你的程序中出现过的最大整数是多少？

⑤ 1*2*3……21=？

⑥ 你知道如何打印双引号吗？

课前思考

⑦ 5除以2的结果还是2.5吗？

⑧ 3+2=?

⑨ 3+”abc”=?

⑩ 3+2+”abc”=?

11 3+”abc”+2=?

12 3+”abc”+”2”=?

13 Java语言中有指针吗？

14 浮点数的最大值是多少？

15 整数的最小值是多少？

学习目标

1. 掌握Java编程语言的基本语法知识
2. 掌握简单数据类型，运算符和表达式，控制语句，数组及字符串的处理

简单数据类型

标 识 符

- 标识符：程序员对程序中的各个元素加以命名时使用的命名记号称为标识符（identifier）。

  Java语言中，标识符是以字母、下划线（_）、美元符($)开始的一个字符序列，后面可以跟字母、下划线、美元符、数字。

标 识 符

- 合法的标识符

  identifier　　userName　User_Name

  _sys_val　$change

- 非法的标识符

  2mail　　room#　　class

保留字

- 保留字：具有专门的意义和用途，不能当作一般的标识符使用，这些标识符称为保留字(reserved word)，也称为关键字：

  abstract break byte boolean catch case class char continue default double do else extends false final float for finally if import implements int interface instanceof long native new null package private protected public return switch synchronized short static super try true this throw throws transient void while goto const

数据类型

- 简单类型

  逻辑型　：　boolean

  文本型　：　char

  整　型　：　byte, short, int, long

  浮点型　：　double, float

- 复合类型

  class

  interface

  数组

常量和变量

- 常量定义：

  final typeSpecifier varName=value[,varName[=value]…];

  如： final int NUM=100;

- 变量定义：

  typeSpecifier varName[=value[,varName[=value]…];

  如： int count; char c=‘a’;

布尔类型——boolean

- 布尔型数据只有两个值true和false，且它们不对应于任何整数值。

  布尔型变量的定义如： boolean b=true;

字符类型——char

- 字符常量：用单引号括起来的一个字符，如‘a’，‘A’;
- 字符型变量：类型为char，它在机器中占16位，其范围为0～65535。字符型变量的定义如：

```
char c='a';/*定义变量c为char型，且赋初值为'a'*/
```

转义字符(以反斜杠开头)

\ddd 表示1～3位的八进制数据(ddd)所代表的字符

\uxxxx 表示1~4位的十六进制数据(xxxx)所代表的字符

\’ 表示单引号字符

\\ 表示反斜杠字符

\r 表示回车

\n 表示换行

\b 表示退格

\f 表示走纸换页

\t 表示横向跳格

```
class CharA{
    public static void main(String args[]){
        char a1,a2,a3;
        a1='a';
        a2='\u0061';
        a3='\141';
        System.out.println(a1);
        System.out.println(a2);
        System.out.println(a3);
        //打印双引号
        //System.out.println("""");
        System.out.println("\"");
        //回车换行
        System.out.println("\r\n");
    }
}
```

打印所有可打印字符

```
class Chars{
    public static void main(String args[]){
        for(char ch='!';ch<='~';ch++)
                        System.out.print(ch);
        System.out.println();
    }
}
```

整型数据

- 整型常量包括：

1. 十进制整数

   如123，－456，0

2. 八进制整数

   以0开头，如0123表示十进制数83，－011表示十进制数－9。

3. 十六进制整数

   以0x或0X开头，如0x123表示十进制数291，－0x12表示十进制数－18。

整型数据

❑整型变量

数据类型	所占位数	数的范围
byte	8	$-2^7 \sim 2^7-1$
short	16	$-2^{15} \sim 2^{15}-1$
int	32	$-2^{31} \sim 2^{31}-1$
long	64	$-2^{63} \sim 2^{63}-1$

浮点型（实型）数据

- 浮点型数据的常量

1. 十进制数形式

   由数字和小数点组成，且必须有小数点，如0.123, .123, 123.,123.0

2. 科学计数法形式

   如：123e3或123E3，其中e或E之前必须有数字，且e或E后面的指数必须为整数。

3. float型的值,必须在数字后加f或F,如1.23f

浮点型（实型）数据

❑浮点型数据的变量

数据类型	所占位数	数的范围
float	32	$3.4e^{-38} \sim 3.4e^{+38}$
double	64	$1.7e^{-308} \sim 1.7e^{+308}$

使用举例

```
public class Assign {
    public  static  void  main (String args [ ] )  {
        int  x ,  y ;
        float  z = 1.234f ;
        double  w = 1.234 ;
        boolean  flag = true ;
```

使用举例


```
    char  c ;
     String  str ;
     String  str1 = " Hi ";
    c = ' A ' ;
    str = " bye ";
    x = 12 ;
    y = 300;
   ......
  }
}
```

数据类型转换

- 自动类型转换

  整型,浮点型,字符型数据可以混合运算。运算中，不同类型的数据先转化为同一类型，然后进行运算，转换从低级到高级；

  低----------->高

  byte,short,char—> int —> long —> float —> double

自动类型转换规则

操作数1类型	操作数2类型	转换后的类型
byte、short、char	int	int
byte、short、char、int	long	long
byte、short、char、int、long	float	float
byte、short、char、int、long、float	double	double

强制类型转换

- 高级数据要转换成低级数据，需用到强制类型转换，如：

```
int i;
byte  b=(byte)i;    /*把int型变量i强制转换
  为byte型*/
```

简单数据类型的封装类

- Java的简单数据类型，在Java中没有作为类来实现。但是有些时候，了解有关简单数据类型的信息，要比了解其值更重要。通过简单类型数据的类包装，可以实现这个目标。

简单数据类型	封装类
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double


```
class Wrapper{
    public static void main(String args[]){
        Integer intObj1=new Integer(18);
        Integer intObj2=new Integer("90");

        int i1=intObj1.intValue();
        int i2=intObj2.intValue();

        System.out.println(i1);
        System.out.println(i2);
    }
}
```

浮点数的最大最小值

最大值：Float.MAX_VALUE、Double. MAX_VALUE

最小值：Float.MIN_VALUE、Double. MIN_VALUE

正无穷大：　Float.POSITIVE_INFINITY、
Double. POSITIVE_INFINITY

负无穷大；　Float.NEGATIVE_INFINITY、
Double. NEGATIVE_INFINITY

0/0：　Float.NaN、Double.NaN


```
public class ConstantTest{
  /** main()方法 */
  public static void main(String args[ ]){
    System.out.println("Integer.MAX_VALUE ="+
    Integer.MAX_VALUE); //输出最大的int型值
    System.out.println("Integer.MIN_VALUE ="+
    Integer.MIN_VALUE); //输出最小的int型值
    System.out.println("Long.MAX_VALUE ="+
    Long.MAX_VALUE);      //输出最大的long型值
    System.out.println("Long.MIN_VALUE
    ="+Long.MIN_VALUE); //输出最小的long型值
```


```
System.out.println("Float.MAX_VALUE ="+ Float.MAX_VALUE);
//输出最大的float型值
System.out.println("Float.MIN_VALUE ="+ Float.MIN_VALUE);
//输出最小的float型值
System.out.println("Double.MAX_VALUE ="+
Double.MAX_VALUE); //输出最大的double型值
System.out.println("Double.MIN_VALUE ="+
Double.MIN_VALUE); //输出最小的double型值
  System.out.println(“Float.POSITIVE_INFINITY =”+
Float.POSITIVE_INFINITY);  /*输出float型正无穷大*/
  System.out.println(“Float.NEGATIVE_INFINITY =”+
Float.NEGATIVE_INFINITY); /*输出float型负无穷大*/
```

```
System.out.println(“Double.POSITIVE_INFINITY =”+
  Double.POSITIVE_INFINITY); //输出double型正无穷大
System.out.println("Double.NEGATIVE_INFINITY ="+
  Double.NEGATIVE_INFINITY); //输出double型负无穷大
System.out.println("Float.NaN ="+ Float.NaN);
      //输出float型0/0
System.out.println("Double.NaN ="+ Double.NaN);
   //输出double型0/0
  }
}
```

运算符和表达式

运算符(按照操作数区分)

- 一元运算符：＋＋，－－，＋，－
- 二元运算符：＋，－，>
- 三元运算符：？：

运算符(按照功能来分)

1）算术运算符： +，—，*，/，%，++，——

3+2;　a-b;　i++;　--i;

2）关系运算符： >，<，>=，<=，==，!=

count>3;　i==0;　n!=-1;

3）布尔逻辑运算符： !，&&，||

flag=true;　!(flag);　flag&&false;

运算符(按照功能来分)

4）位运算符： >>，<<，>>>，&，|，^，~

a=10011101; b=00111001;

a<<3 =11101000;

a>>3 =11110011;

a>>>3=00010011;

a&b=00011001; a|b=10111101;

~a=01100010; a^b=10100100;

运算符(按照功能来分)

5）赋值运算符 =，及其扩展赋值运算符如 +=， —=， *=， /=等。

```
i=3;          i+=3 等效于i=i+3
```

6）条件运算符 ？：

```
result=(sum= =0 ? 1 : num/sum);
```

运算符(按照功能来分)

7）其它：包括分量运算符· ，下标运算符[]，实例运算符instanceof，内存分配运算符new，强制类型转换运算符 (类型)，方法调用运算符 ( ) 等。

```
System.out.println(“Hello world!”);
int array1[]=new  int[4];
```


```
public class OperatorsAndExpressions {
  void singleArithmaticOperator(){
    float i=2.0f,j=10.0f;
    int m=20,n=10;
    System.out.println((++i)*(j--));
    System.out.println("i="+i+",j="+j);
    System.out.println((i++)*(j--));
    System.out.println("i="+i+",j="+j);
    System.out.println((--m)*(n++));
```

```
 System.out.println("m="+m+",n="+n);
    System.out.println((m--)*(n++));
    System.out.println("m="+m+",n="+n);
  }
public static void main(String args[]){
  OperatorsAndExpressions OperAndExp=new
    OperatorsAndExpressions();
  //一元算术运算符的应用
  OperAndExp.singleArithmaticOperator();
}
```

表达式

- 表达式是由操作数和运算符按一定的语法形式组成的符号序列。
  - ✓一个常量或一个变量名字是最简单的表达式，其值即该常量或变量的值；
  - ✓表达式的值还可以用作其他运算的操作数，形成更复杂的表达式。

表达式的类型

- 表达式的类型由运算以及参与运算的操作数的类型决定，可以是简单类型，也可以是复合类型：

  布尔型表达式：　x&&y||z;

  整型表达式：　num1+num2;


```
public class OperatorsAndExpressions {
  void doubleArithmaticOperator(){
    System.out.println(9/2);
    System.out.println(5/2.0);
    byte x=3,y=4;
    long r=80L;
    System.out.println(r/y);
    System.out.println(x*y);
     float z=12.5f,w=5.5f;
```

```
 System.out.println(z+w);
 System.out.println(z-x);
}
public static void main(String args[]){
  OperatorsAndExpressions OperAndExp=new
   OperatorsAndExpressions();
  //二元算术运算符的应用
  OperAndExp.doubleArithmaticOperator();
 }
}
```

短路(short-circuit)逻辑运算符

- &&和||被称为短路逻辑运算符

```
int a=5,b=10;
if ((a!=5)&&(b<30)) {
    System.out.println("ok");
}
```

- &与 |虽然是位运算符，但是可以用来进行逻辑运算，充当非短路逻辑运算符

运算符的优先次序

- 表达式的运算按照运算符的优先顺序从高到低进行，同级运算符从左到右进行：

```
.          []          ()
++    --   !   ~     instanceof
new (type)
*     /     %
+     -
>>    >>>         <<
>     <     >=      <=
```

运算符的优先次序(接上)

==　　!=

&

^

|

&&

||

?:

=　　+=　-=　　*=　　/=　　%=　　^=

&=　　|=　　<<=　　>>=　　>>>=

```
public class OperatorsAndExpressions {
  void priorityOfArithmaticOperator(){
    int a=10,b=4,c=20,d=6;
    System.out.println(a+b*c+d);
    System.out.println(a+c%b);
    System.out.println(a++*b+c*--d);
  }
public static void main(String args[]){
  OperatorsAndExpressions OperAndExp=new
    OperatorsAndExpressions();
  //'算术运算符的优先级
  OperAndExp. priorityOfArithmaticOperator();
}
}
```

扩展运算符“＋”

Java对‘＋’运算符进行了扩展，使它能够进行字符串的连接:“abc”+“de”➔“abcde”。

通过‘＋’运算符还能够将字符串和其他类型的数据进行连接，其结果是字符串:

“abc”＋3➔“abc3”

3.0＋“abc”➔”3.0abc”；但是一般说来，如果‘＋’运算符的第一个操作数是字符串，则Java系统会自动将后续的操作数类型转换成为字符串类型，然后再进行连接；如果‘＋’运算符的第一个操作数不是字符串，则运算结果依据后续的操作数决定

3＋4＋5＋“abc”➔“12abc”还是“345abc”?


```
public class OperatorsAndExpressions {
  void stringPlus(){
    int x=3,y=4,z=5;
    String s="xyz=";
    System.out.println(x+y+z);
    System.out.println(s+x+y+z);
    System.out.println(x+y+z+s);
    System.out.println("abc"+3);
    System.out.println(3.0+"abc");
  }
```

```
public static void main(String args[]){
  OperatorsAndExpressions
    OperAndExp=new
    OperatorsAndExpressions();
  //'+'运算符在字符串中的应用
  OperAndExp.stringPlus();
}
}
```


```
public class OperatorsAndExpressions {
  void equalsMethod1(){
    //基本类型的数据之间的比较：
    int i=10,j=15;
    System.out.println(i==j);
    //复合数据类型的数据之间的比较:
    String s1=new String("how are you");
    String s2=new String("how are you");
    System.out.println(s1==s2);
    String s3="how are you";
    String s4="how are you";
    System.out.println(s3==s4);
}
```


```
public static void main(String args[]){
  OperatorsAndExpressions operAndExp=new
    OperatorsAndExpressions();
  operAndExp.equalsMethod1();
 }
}
```

程序运行结果为：

```
false
false
true
```

控制语句

控制语句

1. 分支语句： if-else, switch
2. 循环语句： while, do-while, for
3. 与程序转移有关的其它语句： break, continue, return
4. 异常处理语句： try-catch-finally

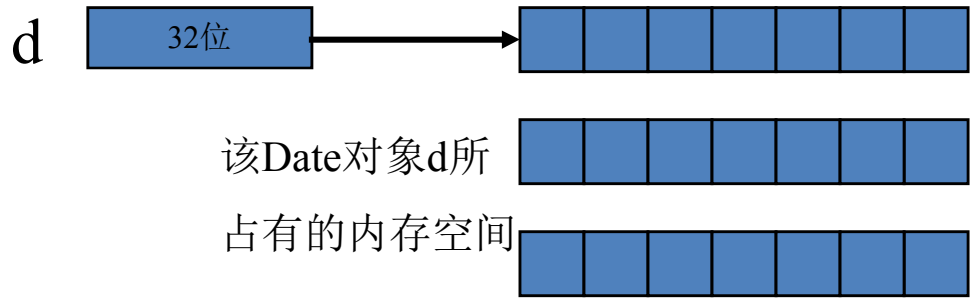
数 组

引用（Reference）的概念

- 每个引用占据32位的内存空间，其值指向对象实际所在的内存中的位置，例如：

  Date d=new Date( );

  通常我们称d是Date型的对象，实际上d就是引用，它是一个32位的数据，它的值指向该Date对象实际所在的内存空间。



数 组

- 数组是一种最简单的复合数据类型，数组是有序数据的集合。

一维数组

- 一维数组的定义： type arrayName[ ]；

  类型(type)可以为Java中任意的数据类型，包括简单类型和复合类型，例如：

  int intArray[ ]；

  Date dateArray[];

  int []intArray;

  Date []dateArray;

一维数组的初始化

- 静态初始化

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

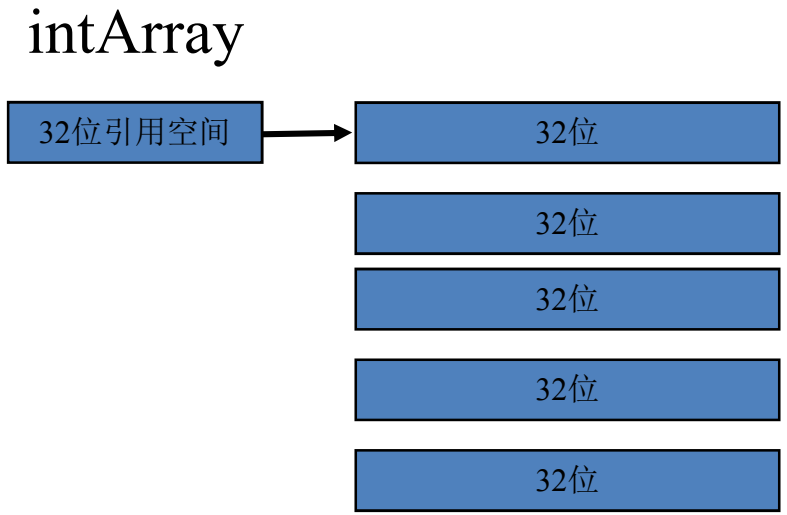
  String stringArray[]= {“abc”, “How ”, “you”};

- 动态初始化

  简单类型的数组:

  int intArray[];

  intArray = new int[5];



一维数组的初始化

一复合类型的数组

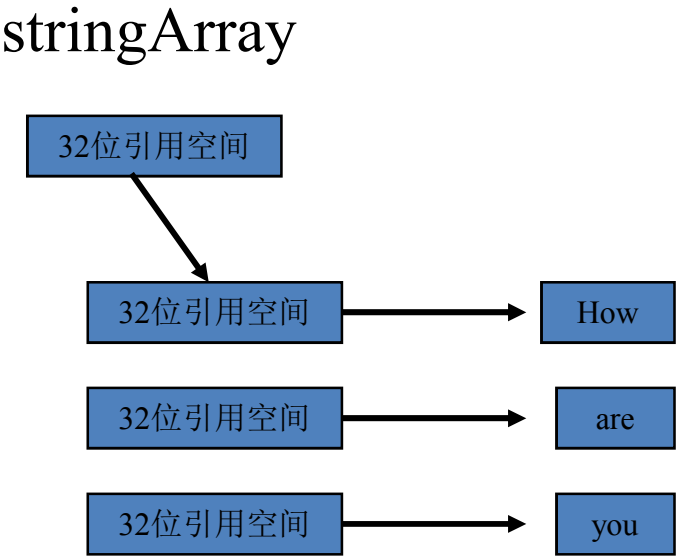
String stringArray[ ];

stringArray = new String[3]; /*为数组中每个元素开辟引用空间(32位) */

stringArray[0]= new String(“How”);

stringArray[1]= new String(“are”);

stringArray[2]= new String(“you”);



一维数组元素的引用

- 数组元素的引用方式为：

  arrayName[index]

  index为数组下标，它可以是整型常数或表达式，下标从0开始。每个数组都有一个属性length指明它的长度，例如：intArray.length指明数组intArray的长度。

多维数组

- Java语言中，多维数组被看作数组的数组。
- 二维数组的定义

  type arrayName[ ][ ];

  type [ ][ ]arrayName;

二维数组的初始化

- 静态初始化

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

  Java语言中，由于把二维数组看作是数组的数组，数组空间不是连续分配的，所以不要求二维数组每一维的大小相同。

动态初始化

- 直接为每一维分配空间，格式如下：

  arrayName = new type[arrayLength1][arrayLength2];

例如： int a[ ][ ] = new int[2][3];

动态初始化

- 从最高维开始，分别为每一维分配空间：

```
arrayName = new type[arrayLength1][ ];
arrayName[0] = new type[arrayLength20];
arrayName[1] = new type[arrayLength21];
…
arrayName[arrayLength1-1] =
                        new  type[arrayLength2n];
```

动态初始化

- 例：

```
int a[ ][ ] = new int[2][ ];
a[0] = new int[3];
a[1] = new int[5];
```

必须注意：

✓在Java语言中，必须首先为最高维分配引用空间，然后再顺次为低维分配空间。

✓与一维数组相同，对于复合类型的数组，必须为每个数组元素单独分配空间。

动态初始化


```
String s[ ][ ] = new String[2][ ];
s[0]= new String[2];
s[1]= new String[2];
s[0][0]= new String(“Good”);
s[0][1]= new String(“Luck”);
s[1][0]= new String(“to”);
s[1][1]= new String(“You”);
```

内存分配过程

String s[ ][ ] = new String[2][ ];

String s[0]= new String[2];

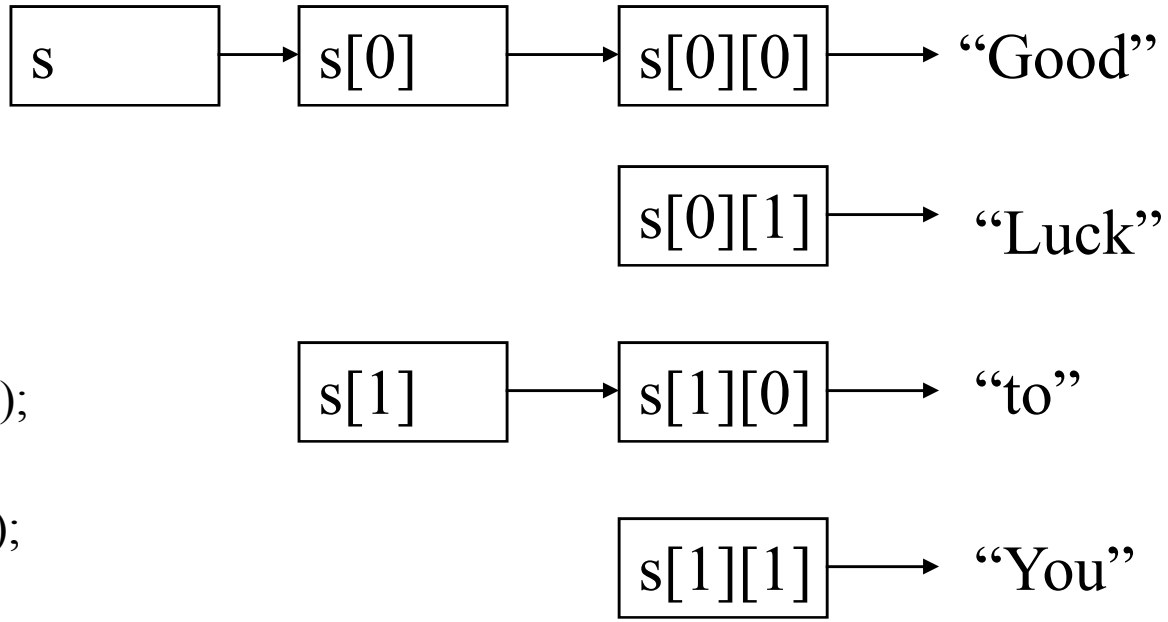
String s[1]= new String[2];

String s[0][0]= new String(“Good”);

String s[0][1]= new String(“Luck”);

String s[1][0]= new String(“to”);

String s[1][1]= new String(“You”);



二维数组元素的引用

- 引用方式为：arrayName[index1][index2]

- 例如： num[1][0];

矩阵计算

$$\begin{bmatrix} 2 & 3 & 4 \\ 4 & 6 & 5 \end{bmatrix} \times \begin{bmatrix} 1 & 5 & 2 & 8 \\ 5 & 9 & 10 & -3 \\ 2 & 7 & -5 & -18 \end{bmatrix} = ?$$

$A_{mn} \times B_{nl} = C_{ml}$

其中，$c_{ij}=a_{ik} \times b_{kj}$ (i=1,2……,m; k=1,2,……,n;
j=1.2,……, l)

例7.2.4 二维数组

```
int a[][]={{2,3,4},{4,6,8}};
int b[][]={{1,5,2,8},{5,9,10,-3},{2,7,-5,-18}};
for( i=0; i<2; i++ ){
    for( j=0; j<4; j++ ){
        c[i][j]=0;
        for( k=0; k<3; k++ )
        c[i][j]+=a[i][k]*b[k][j];
    }
}
```

向量 Vector

- 向量被看做是一种长度可变的数组。向量中的元素必须是对象，不能是基本数据类型。向量的效率比数组低。
- 向量在包java.util.*中。

Vector 的主要方法


```
public Vector(int initialCapacity)
public Vector();
public void addElement(Object newElement)
public Object elementAt(int index);
public void setElementAt(Object newElement,int index);
public void removeElement(int index);
public void removeAllElements();
public int size();
public Object clone();
```

```
import java.util.*;
public class VectorCase {
    public static void main(String args[]){
        Vector v=new Vector();
        Integer i=new Integer(100);
        v.addElement(i);
        Float f=new Float(2.3);
        v.addElement(f);
        String s=new String("I am vector element");
        v.addElement(s);
        for (int n=0;n<v.size();n++){
            System.out.println(v.elementAt(n));
        }
    }
}
```

向量中每个元素的类型是不一样的，这也是与数组的区别之一。

字符串处理

字符串的表示

- Java语言中，把字符串作为对象来处理，类String和StringBuffer都可以用来表示一个字符串。(类名都是大写字母打头)
- 字符串常量

  字符串常量是用双引号括住的一串字符。

  "Hello World!”

  String表示字符串常量.

用String表示字符串

String(String);

String(StringBuffer);

String(char chars[ ]);

String(char chars[ ], int startIndex, int numChars);

String(byte ascii[ ], int hiByte);

String(byte ascii[ ], int hiByte, int startIndex, int numChars);

String使用示例

- *String s=new String() 生成一个空串*
- 下面用不同方法生成字符串“abc”

```
char chars1[]={‘a’,’b’,’c’};
char chars2[]={‘a’,’b’,’c’,’d’,’e’};
String s1=new String(chars1);
String s2=new String(chars2,0,3);
byte ascii1[]={97,98,99};
byte ascii2[]={97,98,99,100,101};
String s3=new String(ascii1,0);
String s4=new String(ascii2,0,0,3);
```

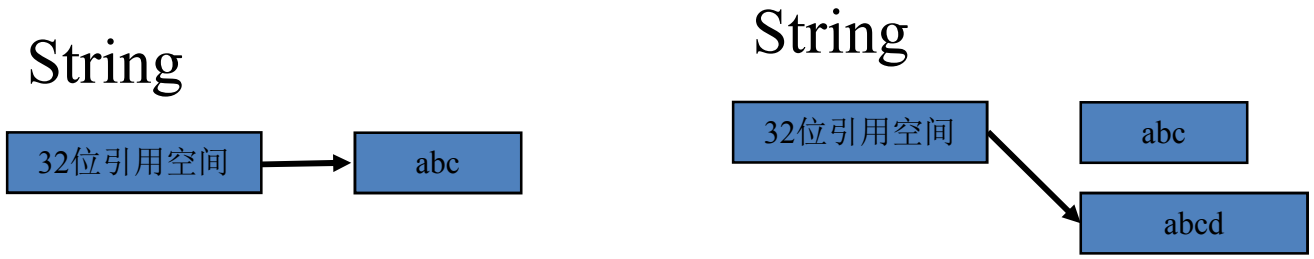
访问字符串

- 类String中提供了length( )、charAt( )、indexOf( )、lastIndexOf( )、getChars( )、getBytes( )、toCharArray( )等方法。

修改字符串

- String

  类String表示不变字符串，因此不能直接对它的内容进行修改，而是通过生成String类对象的一个拷贝，同时完成对字符串的修改。



修改字符串

- String类提供的方法：

  concat( )

  replace( )

  substring( )

  toLowerCase( )

  toUpperCase( )

字符串的比较

- String中提供的方法：

  equals( )和equalsIgnoreCase( )

  它们与运算符‘= =’实现的比较是不同的。运算符‘= =’比较两个对象是否引用同一个实例，而equals( )和equalsIgnoreCase( )则比较两个字符串中对应的每个字符值是否相同。

字符串的转化

- java.lang.Object中提供了方法toString( )把对象转化为字符串。

字符串“+”操作

- 运算符‘+’可用来实现字符串的连接：

  String s = “He is ”+age+“ years old.”;

- 其他类型的数据与字符串进行“+”运算时，将自动转换成字符串。

本讲小结

1. 简单数据类型
2. 复合数据类型
3. 运算符的优先级和表达式
4. 控制语句
5. 数组的初始化、内存空间分配和使用
6. 字符串处理

谢谢！