题目

1. 以下 switch 执行的结果是什么？

```
int num = 1;
switch (num) {
    case 0:
        System.out.print("0");
    case 1:
        System.out.print("1");
    case 2:
        System.out.print("2");
    case 3:
        System.out.print("3");
    default:
        System.out.print("default");
}
```

2. 以下代码可以正常运行吗？为什么？

```
int i = 0;
while (i < 3) {
    if (i == 2) {
        return;
    }
    System.out.println(++i);
}
```

题目

3. 以下输出的结果是什么？

```
String s = new String("laowang");
String s2 = new String("laowang");
System.out.println(s == s2);
switch (s) {
    case "laowang":
        System.out.println("laowang");
        break;
    default:
        System.out.println("default");
        break;
}
```

4. 以下的程序执行结果什么？

```
int i = 0;
do {
        System.out.println(++i);
} while (i < 3)
```

Java 方法

- println() 是一个方法。
- System 是系统类。
- out 是标准输出对象。

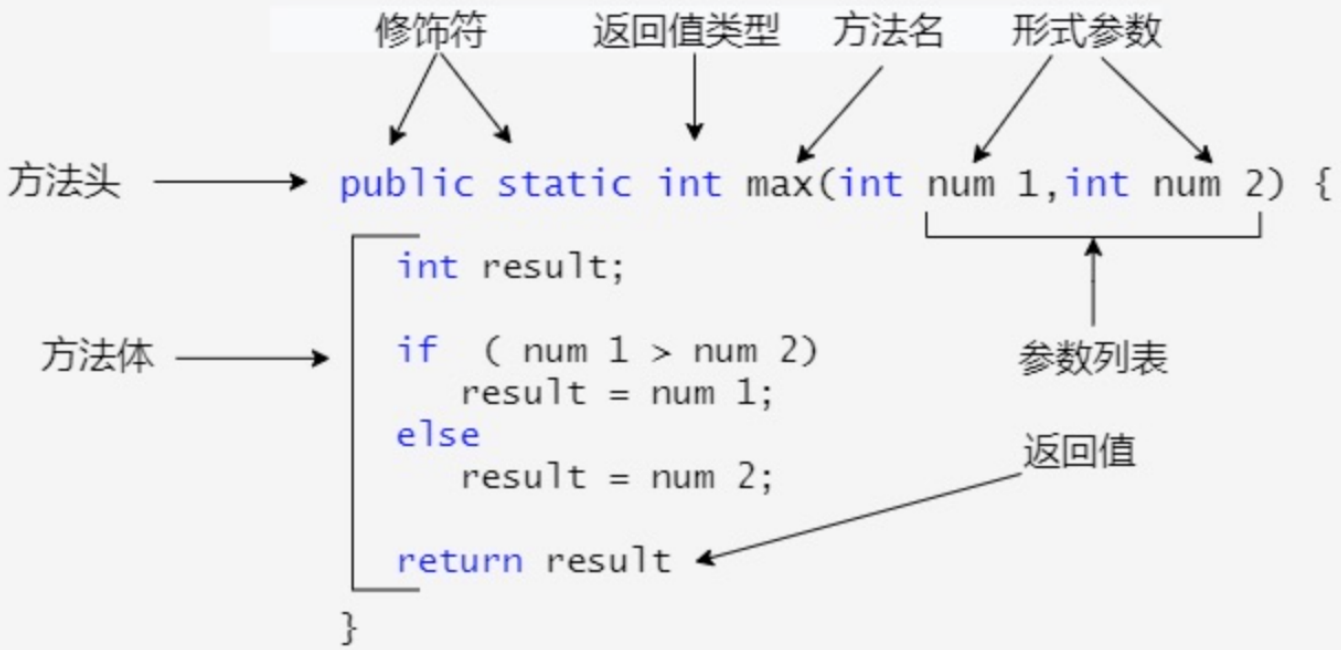
- Java方法是语句的集合，它们在一起执行一个功能。
  - 方法是解决一类问题的步骤的有序组合
  - 方法包含于类或对象中
  - 方法在程序中被创建，在其他地方被引用

Java方法的定义

```
修饰符 返回值类型 方法名(参数类型 参数名){
    ...
    方法体
    ...
    return 返回值;
}
```

- **修饰符**：修饰符，这是可选的，告诉编译器如何调用该方法。定义了该方法的访问类型。
- **返回值类型** ：方法可能会返回值。returnValueType 是方法返回值的数据类型。有些方法执行所需的操作，但没有返回值。在这种情况下，returnValueType 是关键字**void**。
- **方法名**：是方法的实际名称。方法名和参数表共同构成方法签名。
- **参数类型**：参数像是一个占位符。当方法被调用时，传递值给参数。这个值被称为实参或变量。参数列表是指方法的参数类型、顺序和参数的个数。参数是可选的，方法可以不包含任何参数。
- **方法体**：方法体包含具体的语句，定义该方法的功能。

Java方法的定义


修饰符
返回值类型
方法名
形式参数
方法头
public static int max(int num 1,int num 2) {
int result;
方法体
if (num 1 > num 2)
result = num 1;
else
result = num 2;
参数列表
返回值
return result
}

数组

本章知识点

- 数组的声明、创建、引用
  - 一维数组
  - 二维数组
    - 规则二维数组
    - 不规则二维数组
- 不定长参数与数组
- for each循环
- Arrays类

数组

- 数组：一组相同数据类型的元素按一定顺序线性排列。

- 数组的特点

  (1) 数组是相同数据类型的元素的集合。

  (2) 数组中的各元素是有先后顺序的。它们在内存中按照这个顺序连续存放在一起。

  (3) 每个数组元素用整个数组的名字和它自己在数组中的位置表达（此位置被叫做下标）。

4.1 声明数组

- Java中的数组是对象，因此属于引用类型，数组对象需要使用new关键字来创建。
- 数组
  - 一维数组
  - 多维数组

4.1 声明数组

1. 声明数组格式

数组元素类型[] 数组名;

或

数组元素类型 数组名[];

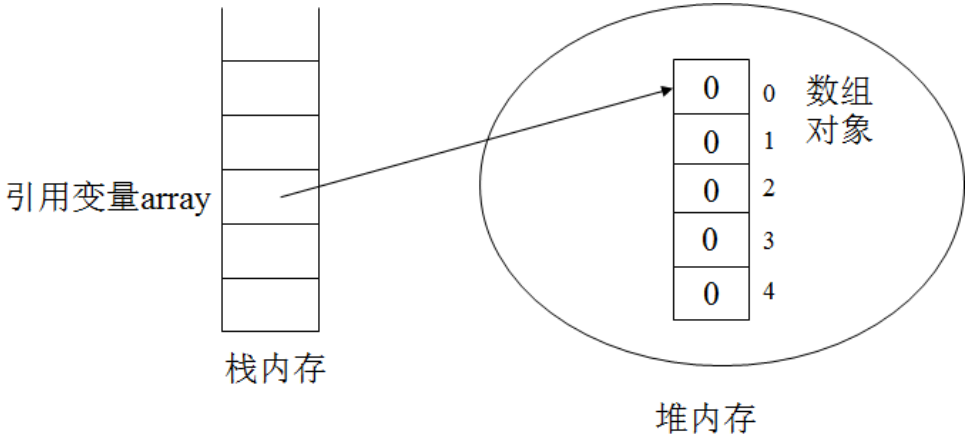
例如：int[] intArray;

4.2 创建数组对象

1、new关键字

数组引用名 = new 数组元素类型[数组元素个数];

```
int[]  Array;
Array = new  int [5] ;
```



4.2 创建数组对象

```
int[]  Array;
Array = new  int [5] ;
```

合二为一

```
int[]  Array = new  int [5];
```

4.2 创建数组对象

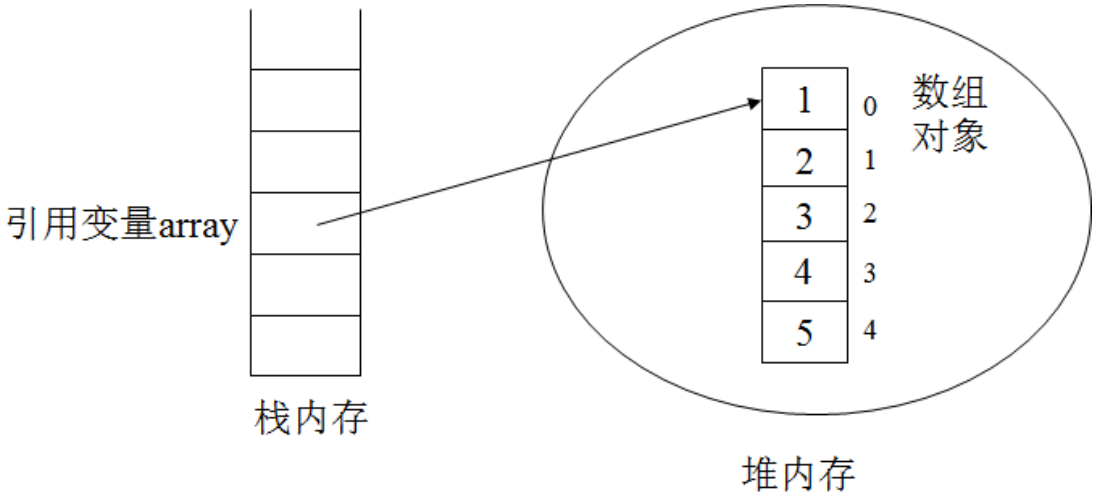
```
int[]  Array = new  int [5];   // 初值为0
```

- 用new关键字为一个数组分配内存空间后，系统将为每个数组元素都赋予一个初值，这个初值取决于数组的类型。
  - 所有数值型数组元素的初值为0
  - 字符型数组元素的初值为一个Unicode编码为0的不可见的控制符（'\u0000'）
  - 布尔型数组元素的初值为false

❖注意：Java中的数组对象一旦创建之后，在程序整个执行期间，就不能再改变数组元素的个数。

- 如果不希望数组的初值为默认值，可以在创建数组的同时对数组元素赋初值，其格式为：

```
int[]  Array = new  int []{1,2,3,4,5};
```



4.2 创建数组对象

2、匿名数组：

不需要引用名，可作为方法调用时的参数

```
new int[]{1,2,3,4,5}
```

```
public static void main(String[] args) {
            //匿名数组做参数
            int sum =getSum(new int[]{1,2,3,4,5});
            ……
}
public static int getSum(int[] a){   //形参接收实参数组
            ……
}
```

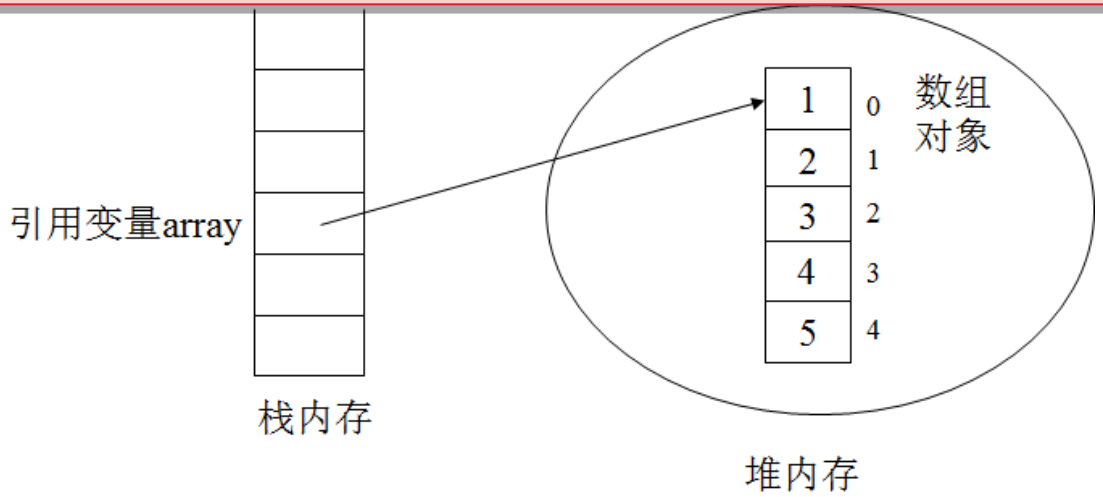
4.2 创建数组对象

3. 数组的显式初始化(用初始化的方式创建数组对象)

- 在声明数组时，进行初始化。数组元素的个数由初始化列表中数据个数决定。

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

```
int[]  Array = new  int []{1,2,3,4,5};
```



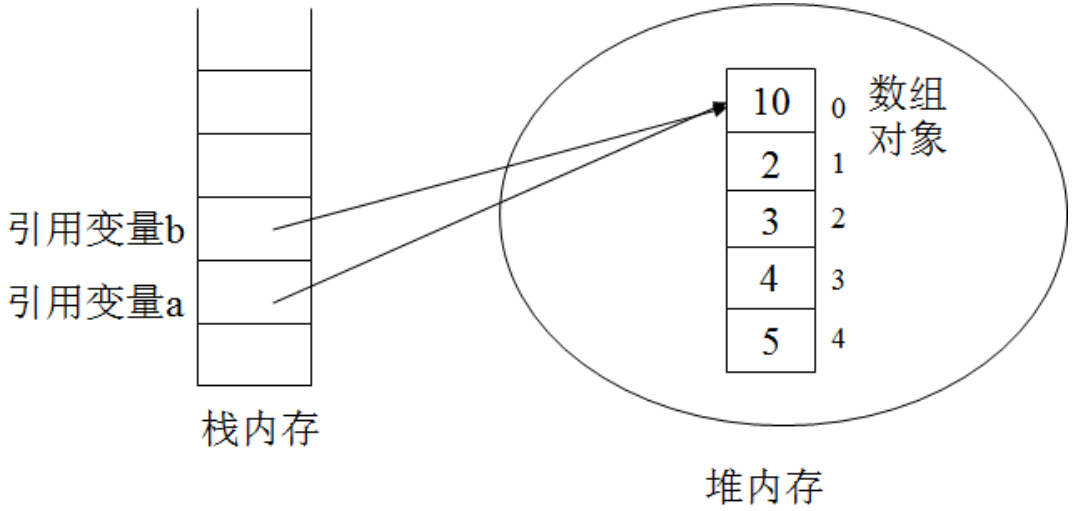
4.2 创建数组对象

【例3-1】写出下面代码的运行结果。

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

        b=a;
        b[0]=10;
        System.out.println("a[0]="+a[0]);
}
```

4.2 创建数组对象

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

        b=a;
        b[0]=10;
        System.out.println("a[0]="+a[0]);
}
```



4.2 数组元素的引用

1. 数组元素的使用
   - 一维数组元素的使用方式

     数组名[下标]
   - 下标必须是整型或者可以转化成整型的量。下标的取值范围：从0开始到数组的长度减1。

     例如, int intArray=new int[5];

     intArray[0], intArray[1],…., intArray[4]

     intArray[length-1]
   - 所有的数组都有一个属性length，这个属性存储了数组元素的个数。

     intArray.length 的值为5。

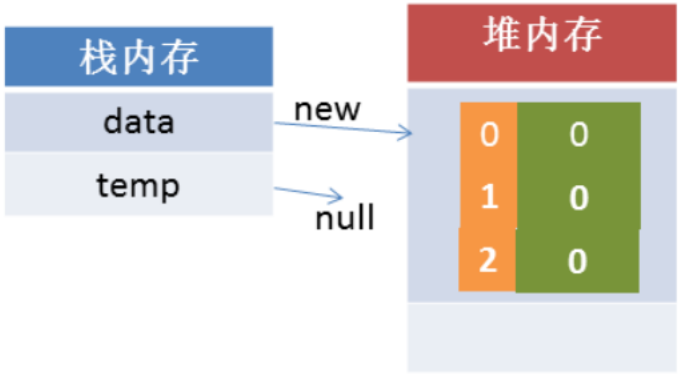
4.3 数组元素的引用

- Java系统能自动检查是否有数组下标越界的情况
  - 如果在程序中使用intArray[10]，就会发生数组下标越界，此时Java系统会自动终止当前的流程，并产生一个名为ArrayIndexOutOfBoundsException的异常，通知使用者出现了数组下标越界。
  - 避免越界发生的有效方法是利用length属性作为数组下标的上界。

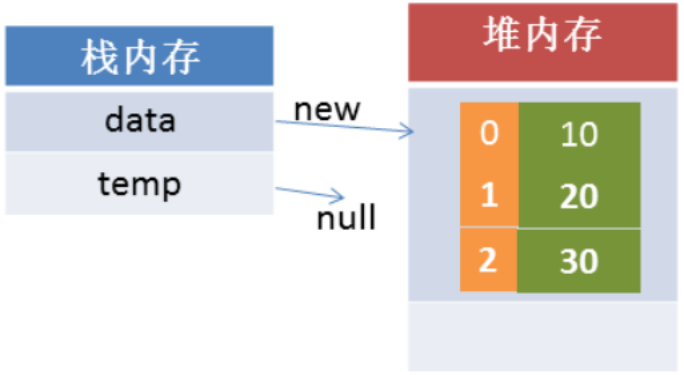
4.3 数组元素的引用

```
public class ArrayDemo {
        public static void main(String args[]) {
                int data[] = null;
                data = new int[3]; //开辟一个长度为3的数组
                int temp[] = null; //声明对象
                data[0] = 10;
                data[1] = 20;
                data[2] = 30;
                temp = data;  //int temp[] = data;
                temp[0] = 99;
                for(int i = 0; i < temp.length; i++) {
                        System.out.println(data[i]);
                }
        }
}
```

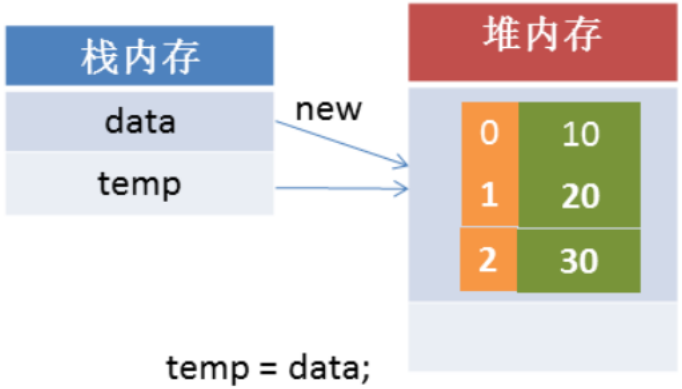
4.3 数组元素的引用



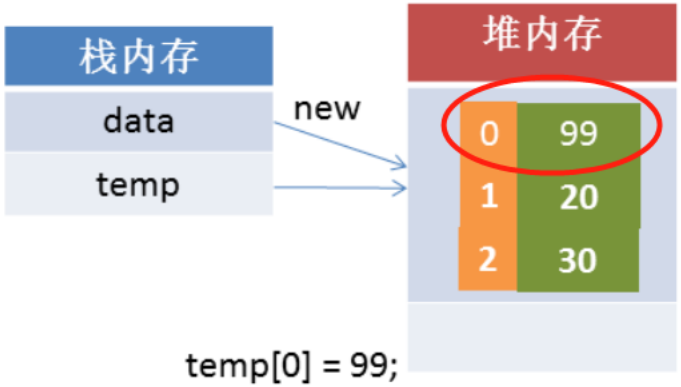
int data[] = new int[3];
int temp[] = null;



data[0] = 10;
data[1] = 20;
data[2] = 30;



temp = data;



temp[0] = 99;

4.3 数组元素的引用

```
1  class ChangeIt
2  {
3      static void doIt( int[] z )
4      {
5          z = null ;
6      }
7  }
8
9  class TestIt
10 {
11     public static void main ( String[] args )
12     {
13         int[] myArray = {1, 2, 3, 4, 5};
14         ChangeIt.doIt( myArray );
15         for(int j=0; j<myArray.length; j++)
16         System.out.print( myArray[j] + " " );
17     }
18 }
```

A. 1 2 3 4 5

B. 什么都不会打印出来

C. 程序将因运行时错误而停止。

D. 0 0 0 0 0

4.3 数组和不定长参数

- 在调用某个方法时，有时会出现方法的参数个数事先无法确定的情况，比如printf方法：

```
System.out.printf("%d",a);
System.out.printf("%d  %d",a, b);
System.out.printf("%d  %d %d",a, b, c);
```

定义时无法事先决定参数的个数。

4.3 数组和不定长参数

- Java SE 5.0之后开始支持不定长参数用以解决这个问题。
- 不定长度的形参实为一个数组参数，不定长参数定义的语法格式为：

  数据类型…　参数名

不定长的形参只能处于形参列表的最后。一个方法中最多只能包含一个不定长参数。调用包含不定长参数的方法时，既可以向其传入多个参数，也可以传入一个数组。

4.3 数组和不定长参数

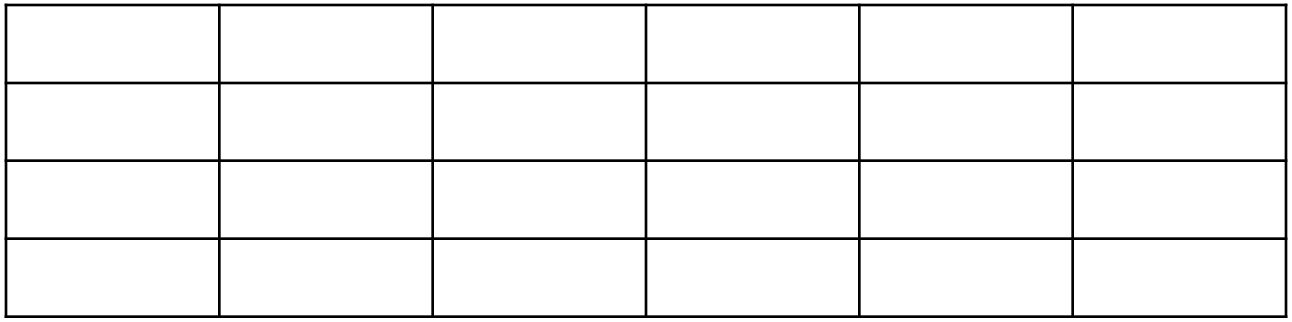
【例3-5】定义一个对不定个数的一组数进行求和的方法。

- 分析：因为不确定被求和的数字的个数，所以使用不定长形参。

4.4 多维数组

带有两个以上下标的数组称为多维数组。

在Java语言中，多维数组被看做是数组的数组。



4.4 多维数组

二维数组声明

数据类型 数组引用名[][];

int[][] array;

二维数组的创建

- new

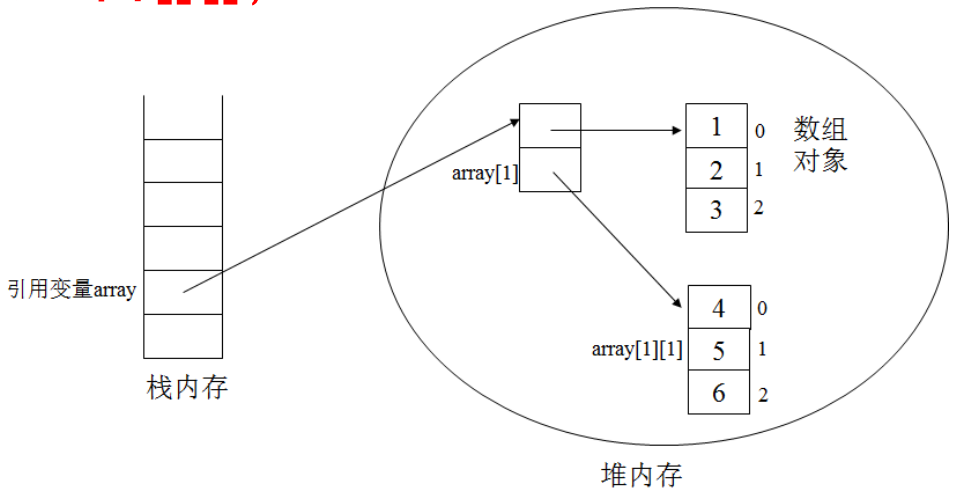
array = new int[2][3];

或者：

array = new int[][]{{1,2,3},

- 通过赋初值的形式创建

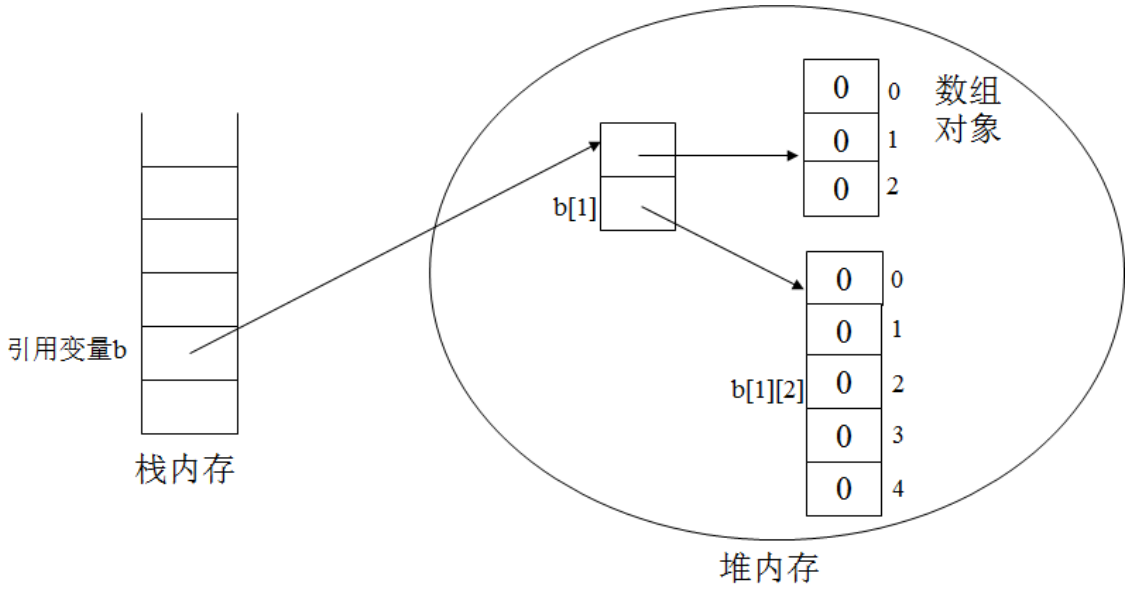
array ={{1,2,3},{4,5,6}};



4.4 多维数组

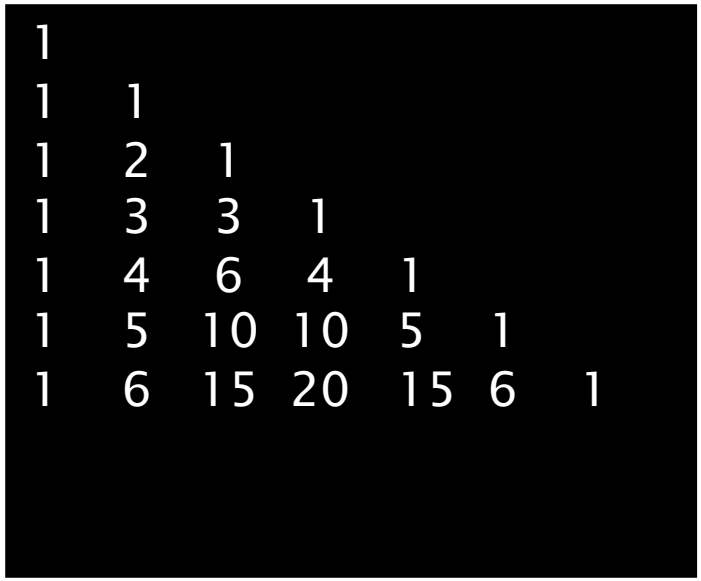
创建二维数组对象可以进行动态分配

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



4.4 多维数组

【例4-6】存储并打印杨辉三角形的前n行。



Java中的数组对象可以在程序运行的过程中根据需求动态创建，且可以是不规则的。

length属性的使用！

4.5 for each循环

- Java SE 5.0中增加了一种循环结构，可以用来更便捷地遍历数组中的每个元素，程序设计者不必为指定下标值而分心，称为**for each循环**。

- 语句格式

```
for(数据类型  迭代变量: 数组|集合){
        //迭代变量即为依次访问的数组中的元素
}
```

4.5 for each循环

- for each+一维数组

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

```
int[] array = new int[5];
for(int element:array){
    System.out.print(element+" ");
}
```

- for each+二维数组

```
for(int[] rows: tri){
        //二维数组的每行是一维数组，迭代变量是一维数组元素类型int[]
        for(int element :rows){
                //对每行进行迭代，迭代的对象为一维数组元素rows
                System.out.printf("%4s",element);
        }
        System.out.println();
}
```

4.6 Array类

- 查询API
- 具有以下功能：
  - 给数组赋值：通过 fill 方法。
  - 对数组排序：通过 sort 方法,按升序。
  - 比较数组：通过 equals 方法比较数组中元素值是否相等。
  - 查找数组元素：通过 binarySearch 方法能对排序好的数组进行二分查找法操作。

4.6 Array类—sort()类

- Arrays类中的排序方法使用的是优化的快速排序算法。

- sort方法是Arrays类中的静态方法，可以通过类直接进行调用
- sort方法有各种参数类型的重载版本，可以实现对各种数据类型数组的排序

4.6 Array类—sort()类

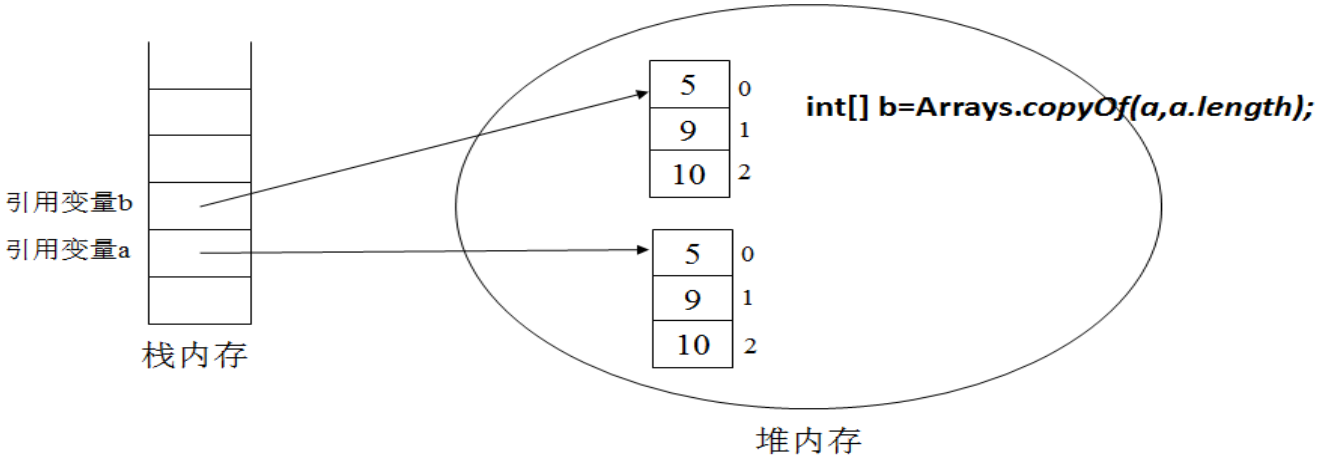
【例3-7】利用Arrays类对int型数组进行排序。

```
public static void main(String[] args) {
        int[] a = new int[]{32, 32, 96, 10, 29, 55};

        System.out.println(Arrays.toString(a));
        //以"[32, 32, 96, 10, 29, 55]"形式打印输出

        Arrays.sort(a);   //对数组a进行快速排序

        System.out.println(Arrays.toString(a));
        //以"[10, 29, 32, 32, 55, 96]"形式打印输出
}
```

4.6 Array类—copyof

- Arrays类中的copyOf方法能够实现数组的复制，两种格式：
  - *type copyOf(type[] a, int length)*
  - *type copyOf(type[] a, int start, int end)*

  - copyOf实现的复制，已经使目标数组脱离了源数组，即复制得到一个新的数组对象



4.7 String类

- String类也是一种引用类型，在某些语言课程中String类型当作一种特殊的数组类型，而在Java语言中，String类是一种类类型，关于类，在下面的章节中将进行介绍。在本章中，读者可以把String类型当成一种特殊的数据类型来进行理解。
- 1．String对象的初始化
- String类的构造函数

构造函数	作用描述
String(byte [ ] bytes)	通过byte数组构造字符串对象 。
String(char[ ] value)	通过char数组构造字符串对象
String(Sting str)	通过字符串常量构造字符串对象
String(StringBuffer buffer)	通过StringBuffer数组构造字符串对象。

4.7 String类

- 2.String类的常用方法
- ⑴char charAt (int index)：取字符串中的某一个字符，其中的参数index指的是字符串中序数。字符串的序数从0开始到length()-1。
  例如：String s = new String("abcdefghijklmnopqrstuvwxyz");
  System.out.println("s.charAt(5): " + s.charAt(5));
  结果为： s.charAt(5): f
- ⑵int compareTo(String anotherString)：当前String对象与anotherString比较。相等时返回 0；不相等时，从两个字符串第0个字符开始比较，返回第一个不相等的字符差，另一种情况，较长字符串的前面部分恰巧是较短的字符串，返回它们的长度差。
- 即：如果当前字符串与s相同，该方法返回值0；如果当前字符串对象大于s，该方法返回正值；如果小于s，该方法返回负值。

4.7 String类

- ⑶int compareTo(Object o) ：如果o是String对象，和(2)的功能一样；否则抛出ClassCastException异常。
- 例如:String s1 = new String("abcdefghijklmn");
- String s2 = new String("abcdefghij");
- String s3 = new String("abcdefghijalmn");
- System.out.println("s1.compareTo(s2): " + s1.compareTo(s2)); //返回长度差
  System.out.println("s1.compareTo(s3): " + s1.compareTo(s3)); //返回'k'-'a'的差
  结果为：s1.compareTo(s2): 4
- s1.compareTo(s3): 10

4.7 String类

- ⑷String concat(String str)：将该String对象与str连接在一起。
- 例如： String cc="134"+ h1.concat("def");
- System.out.println(cc);
- 结果：134abcdef
- ⑸public boolean startsWith(String prefix)
- public boolean endsWith (String suffix)
- 判断当前字符串对象的前缀/后缀是否是参数指定的字符串s。
- 例如：String s1 = new String("abcdefghij");
- String s2 = new String("ghij");
- System.out.println("s1.endsWith(s2): " + s1.endsWith(s2));
- 结果为：s1.endsWith(s2): true

4.7 String类

- ⑹boolean equals(Object anObject)：比较当前字符串对象的内容是否与参数指定的字符串s的内容是否相同。
- ⑺int indexOf(int ch)：只找第一个匹配字符位置。
- int indexOf(int ch, int fromIndex)：从fromIndex开始找第一个匹配字符位置。
- int indexOf(String str)：只找第一个匹配字符串位置。
- int indexOf(String str, int fromIndex) ：从fromIndex开始找第一个匹配字符串位置 。

4.7 String类

- ⑻int length()：返回当前字符串长度。
- ⑼String replace(char oldChar , char newChar)：将字符号串中第一个oldChar替换成newChar。
- public String replaceAll(String old ,String new)
- 调用该方法可以获得一个串对象，这个串对象是通过用参数new指定的字符串替换s中由old指定的所有字符串而得到的字符串。
- “shout:miao miao”.replaceAll(“miao”,”wang”));
- ⑽boolean startsWith(String prefix)：该String对象是否以prefix开始。boolean startsWith(String prefix, int toffset)：该String对象从toffset位置算起，是否以prefix开始。

4.7 String类

- ⑾String substring(int beginIndex)：取从beginIndex位置开始到结束的子字符串。
- String substring(int beginIndex, int endIndex)：取从beginIndex位置开始到endIndex位置的子字符串。
- 例如："hamburger".substring(4, 8) returns "urge"
- "smiles".substring(1, 5) returns "mile"
- ⑿char[] toCharArray()：将该String对象转换成char数组。
- 关于String类的使用就介绍这么多，其它的方法以及这里到的方法的详细声明可以参看对应的API文档。

4.7 StringBuffer类

- StringBuffer类和String一样，也用来代表字符串，在StringBuffer类中存在很多和String类一样的方法，这些方法在功能上和String类中的功能是完全一样的。但是有一个最显著的区别在于，对于StringBuffer对象的每次修改都会改变对象自身，这点是和String类最大的区别，所以通常我们称String类为字符串常量，StringBuffer为字符串变量。并且由于StringBuffer的内部实现方式和String不同，所以StringBuffer在进行字符串处理时，不生成新的对象，在内存使用上要优于String类。所以在实际使用时，如果经常需要对一个字符串进行修改，例如插入、删除等操作，使用StringBuffer要更加适合一些。

4.7 StringBuffer类

- 1. StringBuffer对象的初始化
- StringBuffer对象的初始化不像String类的初始化一样，Java提供的有特殊的语法，而通常情况下一般使用构造方法进行初始化。

构造方法	作用描述
StringBuffer()	构造一个没有任何字符的StringBuffer类
StringBuffer(int length)	构造一个没有任何字符的StringBuffer类，并且，其长度为length。
StringBuffer(String str)	以str为初始值构造一个StringBuffer类

4.7 StringBuffer类

- 2. StringBuffer对象的常用函数
- StringBuffer类中的方法主要偏重于对于字符串的变化，例如追加、插入和删除等，这个也是StringBuffer和String类的主要区别。
- ⑴append方法
- public StringBuffer append(boolean b)
- 该方法的作用是追加内容到当前StringBuffer对象的末尾，类似于字符串的连接。调用该方法以后，StringBuffer对象的内容也发生改变，例如：
- StringBuffer sb = new StringBuffer(“abc”);
- sb.append(true);
- 则对象sb的值将变成” abctrue”。

4.7 StringBuffer类

- ⑵insert方法
- public StringBuffer insert(int offset, boolean b)
- 该方法的作用是在StringBuffer对象中插入内容，然后形成新的字符串。例如：
- StringBuffer sb = new StringBuffer(“TestString”);
- sb.insert(4,false);
- 该示例代码的作用是在对象sb的索引值4的位置插入false值，形成新的字符串，则执行以后对象sb的值是”TestfalseString”。

4.7 StringBuffer类

- ⑶reverse方法
- public StringBuffer reverse()
- 该方法的作用是将StringBuffer对象中的内容反转，然后形成新的字符串。例如：
- StringBuffer sb = new StringBuffer(“abc”);
- sb.reverse();
- 经过反转以后，对象sb中的内容将变为”cba”。
- ⑷setCharAt方法
- public void setCharAt(int index, char ch)
- 该方法的作用是修改对象中索引值为index位置的字符为新的字符ch。例如：
- StringBuffer sb = new StringBuffer(“abc”);
- sb.setCharAt(1,’D’);
- 则对象sb的值将变成”aDc”。

4.7 StringBuffer类

- ⑸trimToSize方法
- public void trimToSize()
- 该方法的作用是将StringBuffer对象的中存储空间缩小到和字符串长度一样的长度，减少空间的浪费。
- 总之，在实际使用时，String和StringBuffer各有优势和不足，可以根据具体的使用环境，选择对应的类型进行使用。

Java中的字符串比较相等与大小

在C++中，两个字符串比较的代码可以为：
(string1==string2)
但在java中，这个代码即使在两个字符串完全相同的情况下也会返回false
Java中必须使用string1.equals(string2)来进行判断

如果：
string s1="Hello";
string s2="Hello";
则(s1==s2)=true;
因为他们指向的同一个对象。
如果：
String s1=new String("Hello");
String s2=new String("Hello");
则(s1==s2)=false
如果把其他变量的值赋给s1和s2，即使内容相同，由于不是指向同一个对象，也会返回false。所以建议使用equals()，因为equals比较的才是真正的内容

equalsIgnoreCase()方法与equals()的区别

String.equals()对大小写敏感，而String.equalsIgnoreCase()忽略大小写

例如："ABC".equals("abc")是false
"ABC".equalsIgnoreCase("abc")为ture

总结：

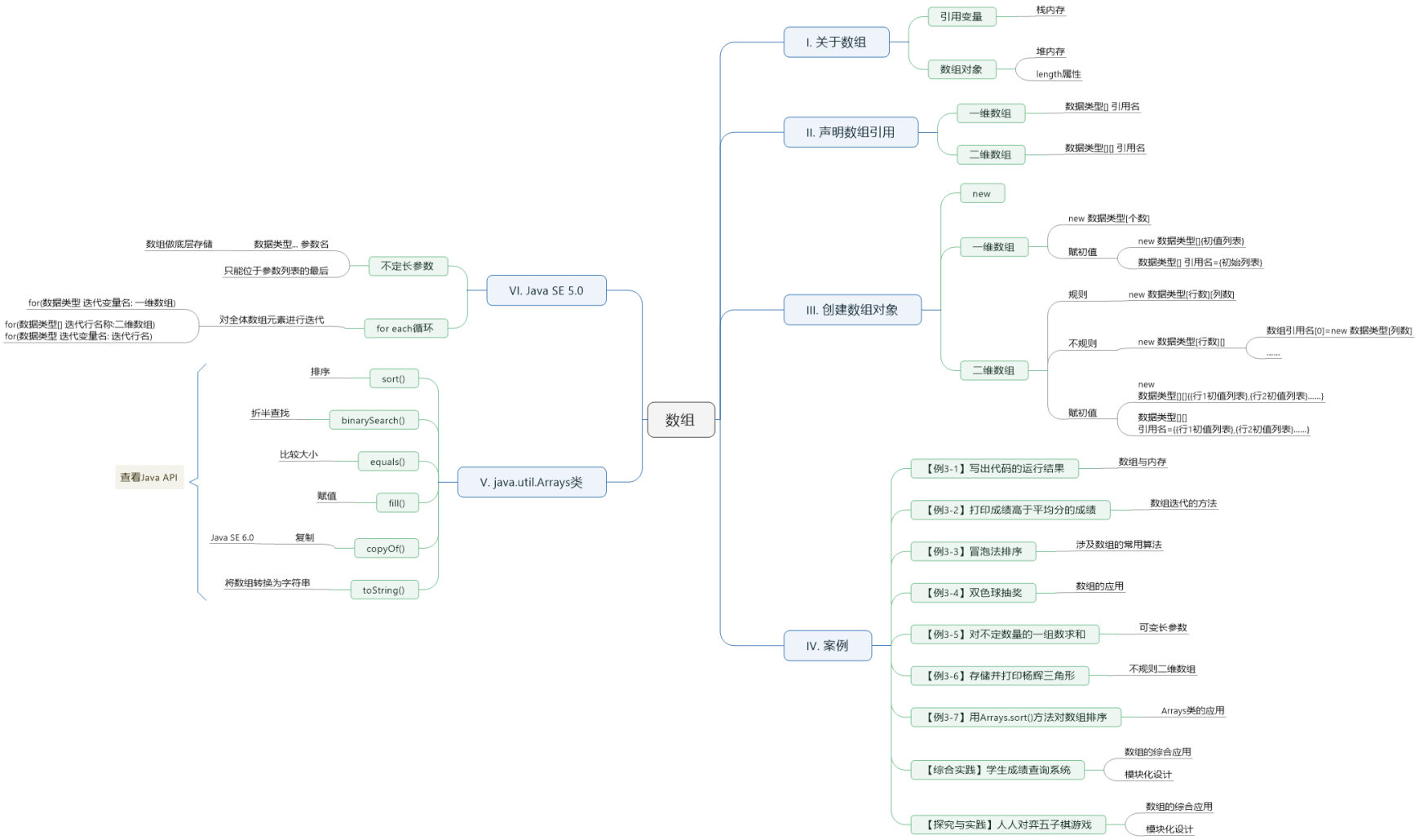
如果比较字符串的大小使用：
str1.compareTo(String str2)

如果比较字符串是否相等用：

String.equals()对大小写敏感；

String.equalsIgnoreCase()忽略大小写。

本章思维导图


数组
I. 关于数组
引用变量
栈内存
数组对象
堆内存
length属性
II. 声明数组引用
一维数组
数据类型[] 引用名
二维数组
数据类型[][] 引用名
III. 创建数组对象
new
一维数组
new 数据类型[个数]
赋初值
new 数据类型[]{初值列表}
数据类型[] 引用名={初始列表}
二维数组
规则
new 数据类型[行数][列数]
不规则
new 数据类型[行数][]
数组引用名[0]=new 数据类型[列数]
.....
赋初值
new
数据类型[][]{{行1初值列表},{行2初值列表}.....}
数据类型[][]
引用名={{行1初值列表},{行2初值列表}.....}
IV. 案例
【例3-1】写出代码的运行结果
数组与内存
【例3-2】打印成绩高于平均分的成绩
数组迭代的方法
【例3-3】冒泡法排序
涉及数组的常用算法
【例3-4】双色球抽奖
数组的应用
【例3-5】对不定数量的一组数求和
可变长参数
【例3-6】存储并打印杨辉三角形
不规则二维数组
【例3-7】用Arrays.sort()方法对数组排序
Arrays类的应用
【综合实践】学生成绩查询系统
数组的综合应用
模块化设计
【探究与实践】人人对弈五子棋游戏
数组的综合应用
模块化设计
V. java.util.Arrays类
查看Java API
排序
sort()
折半查找
binarySearch()
比较大小
equals()
赋值
fill()
Java SE 6.0
复制
copyOf()
将数组转换为字符串
toString()
VI. Java SE 5.0
不定长参数
数组做底层存储
数据类型... 参数名
只能位于参数列表的最后
for each循环
对全体数组元素进行迭代
for(数据类型 迭代变量名: 一维数组)
for(数据类型[] 迭代行名称:二维数组)
for(数据类型 迭代变量名: 迭代行名)