1、课程名称：JAVA SE 基础部分

2、知识点

2.1、课程说明

JAVA SE 本课程的主要目的还是为 JAVA EE 服务的，所以在讲解中主要重点的部分就是讲解与 JAVA EE 开发直接相关的内容，而且会给出许多的参考代码，在整个讲解中图形界面 100%不讲，重点部分就以下四块知识：

- 面向对象：充分的掌握接口的作用。
- 集合框架：数据结构的实现都在此部分完成。
- IO 编程：初学者最大难点，完全是应用面向对象的概念去理解的。
- 数据库编程（JDBC）：主要的作用是进行数据库的开发，主要与 Oracle 有管。

除了以上的核心部分之外，对于 JDK 1.5 之后的主要新特性也会讲解到，但是对于使用上不会有明确的要求。

在 JAVA SE 中的知识点主要分为以下几个部分的掌握程度：

- 重点：JAVA SE 的基础概念、面向对象、集合、IO、JDBC、类库

- 理解：泛型、枚举、Annotation、反射机制、类库的使用
- 了解：线程、网络编程

JAVA SE 的课程整体的代码会很多，而且依然是讲一部分概念之后会有一段练习代码，只要把练习代码掌握透了，所有的概念就通了。

2.2、本次预计讲解的知识点

1、 JAVA 的发展及环境的搭建配置；
2、 JAVA 中的主要数据类型及关键字；
3、 基本的运算操作符；
4、 程序的控制语句：循环、判断；
5、 方法及数组的使用。

3、具体内容

3.1、认识 Java（了解）

Java 是现在最流行的一种语言，而且在 Java 中完全的显示出了简单的特性，所以 java 语言足够简单。

Java 最早的时候是在 1991 年的 GREEN 项目诞生的，但是其原本的名字不叫 Java 而是称为 OAK（橡树），GREEN 的项目实际上就属于现在所提出的嵌入式的开发项目，通过 EMAIL 可以控制家电的工作。但是最早的时候 SUN 公司的设计人员原本是使用 C++进行开发，但是由于其开发过于复杂了，所以使用了 C++开发出了一套新的平台 —— OAK。

可是遗憾的是 OAK 项目并没有中标，被网景公司的 SGL 平台所打败，那么很明显就意味着死亡，但是后来 SUN 公司的人员开始向网景公司学习浏览器技术，后来产生了 HOTJava 的浏览器，之后并且在 1995 年的时候成功的将 OAK 更名为 JAVA，推出了 JDK 1.0 和 Applet 程序。

PS：斯坦伏大学，本身造就了很多的人才：YAHOO 、HP。

JAVA 语言的发展经历了以下的几个重大的版本：

- JDK 1.0：标志着整个 JAVA 体系的诞生；
- JDK 1.2：1998 年推出，Java 更名为 JAVA 2，加入了大量的图形界面的开发包，但是现在基本上已经不用了；
- JDK 1.5：2005 年推出，加入了很多的新特性，而且这些新特性几乎都是从.net 上学来的。

JAVA 从诞生之初发展到今天也分成了三个方面的发展：

- J2SE 的开发（JAVA SE）：主要提供了开发平台的底层支持，可以开发单机版程序；
- J2EE 的开发（JAVA EE）：主要是完成各种企业软件的开发，是在 JAVA SE 的基础上构建的；
- J2ME 的开发（JAVA ME）：使用 Java 完成嵌入式开发的平台。

以上的三个方面，核心的基础部分就是 J2SE，所以，在把 JAVA 学习透彻的话，就必须先把 JAVA SE 彻底的精通。

在讲解中，JAVA 主要使用的版本就是 JDK 1.6，但是需要说明的是，在实际的开发中使用最广泛的版本依然是 JDK 1.5，而且 JDK 1.6 之中还存在着 bug，最新的版本是 JDK 1.7。

以后在开发的时候一定要注意：只要是新的产品都要小心使用，因为有 80%的可能性会造成项目的失败。

在 JAVA 语言中有以下几个特点：

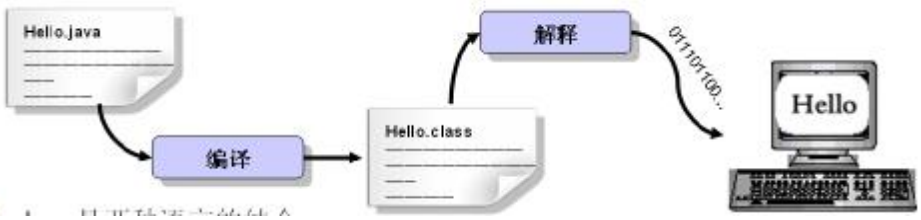
- 简单：Java 语言比任何一门都简单，但是这种简单只是针对于语法而言的，实际上对于 Java 来讲，其庞大和复杂程度确实太高了，支持公司太多了：IBM、BEA、Oracle、Apple、HP.
- 自动的垃圾收集：在程序的操作中会存在着许多的无用的内存空间，如果处理不当会使系统越来越庞大，在 java 中可以对无用的空间自动进行回收，而无需开发人员手工回收。
- 安全性高：Java 的所有程序都是通过字节码的方式保存的，所以其安全性相对较高。
- 多线程：多线程的处理可以使 JAVA 的处理能力提高的更多。
- 可移植性高：Java 发展到今天，一直提倡的口号就是可移植性高，可以在不同的操作系统平台上运行。

3.2、Java 的操作原理（理解）

对于编程语言来讲，主要有两种：

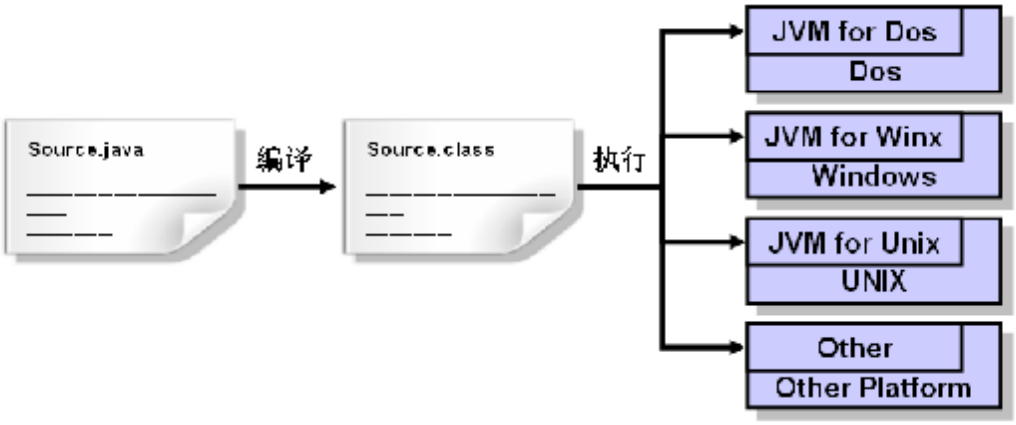
- 编译型：
- 解释型：将一段代码在一个指定的平台上进行解释执行。

正因为存在了解释器，所以 Java 语言开发的程序可以在各个平台上使用。



一个 Java 程序需要先经过编译（由 JDK 自动提供命令：javac 执行），编译之后将形成一个字节码的文件（*.class），之后再进行解释执行（由 JDK 自动提供的命令：java 执行），再在电脑上进行程序的运行显示。

但是，此时的电脑并不是一台物理上存在的电脑，而是一台由软件和硬件模拟的一台虚拟电脑（Java 虚拟机）。



可以发现，所有的*.class 文件实际上最终认的不是操作系统，而是在操作系统上绑定的 JVM，依靠 JVM 执行，而 JVM 去适应不同的操作系统。

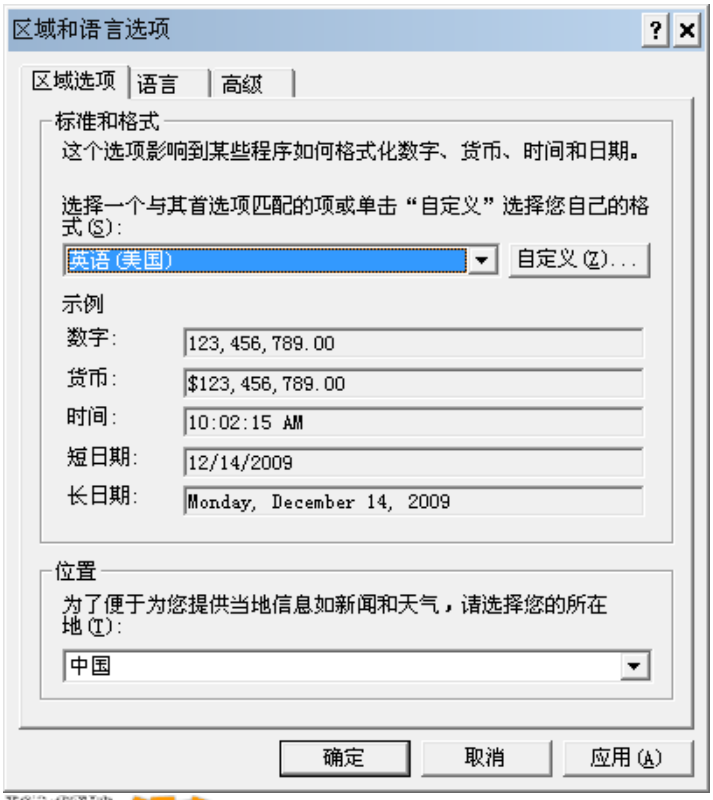
Windows XP 之后之所以不支持 Java，主要的原因就是从操作系统中将 JVM 移除掉了。

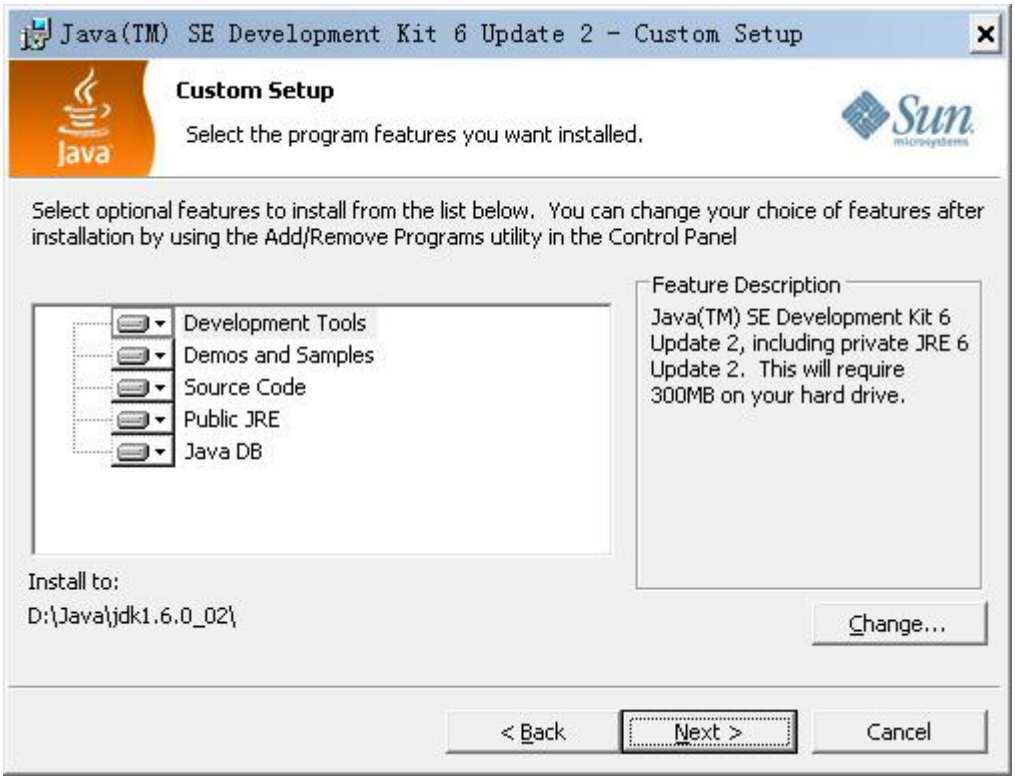
3.3、搭建 Java 的开发平台（重点）

如果要开发 JAVA 程序，则肯定要使用 JDK，现在的 JDK 的最新版本是 JDK 1.7，所以本次使用的是 JDK 1.6 版本。可以直接从 www.sun.com 上下载最新的 JDK 版本。

现在的 JDK 属于多国语言版，一个 JDK 可以同时支持多个国家，但是需要注意一点，由于现在是在 Windows 的中文环境下，那么 JDK 显示的时候肯定是以中文的样式显示出来的，这样的话许多的错误信息表示的并不明确。

【我的电脑】 à 【控制面板】 à 【区域和语言环境】 à 【英语（美国）】





安装的时候选择在 D 盘上安装 JAVA 开发环境。

安装完成之后，此时本机就具备了开发 Java 程序的能力，但是需要注意的是，如果要开发 Java 程序则肯定需要 javac 和 java 两个命令，但是这两个命令是在 D:\Java\jdk1.6.0_02\bin 目录下才存在的，本身的 windows 中是不支持的。

如果要想让 windows 可以正常的使用这些命令进行开发的话，则必须配置一个环境：path 路径。

【我的电脑】 à 【属性】 à 【高级】 à 【环境变量】 à 【编辑 path】 à 加入之前的目录即可，每一个 path 的配置之间使用“;”分隔。

配置完成之后，以后就可以在命令行方式下使用 JDK 所提供的各种命令了。

新的 path 路径配置完成之后，需要重新启动命令行窗口，因为每次在启动的时候才会将新的配置加载进来。

现在 Java 的开发环境搭建完成之后，下面开始编写第一个 java 程序，所有的程序依然以打印不完的“Hello World”为主。

范例：第一个程序 —— Hello.java

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

第一个程序完成之后，下面就要进入到命令行方式下进行执行，按照如下的步骤完成：

1、 编译程序：javac Hello.java

2、 解释程序：java Hello

但是，很遗憾出现了以下的问题：

```
Exception in thread "main" java.lang.UnsupportedClassVersionError: Hello (Unsupported major.minor version 50.0)
```

现在安装的 JDK 版本是 JDK 1.6，而且既然已经安装上了，则肯定在操作系统中使用的都是 JDK 1.6，下面通过如下的语句来验证一下当前的使用版本：java -version，此时显示：

```
java version "1.4.2_03"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM (build 1.4.2_03-b02, mixed mode)
```

现在的版本是 JDK 1.4，编译的版本是 JDK 1.6，但是执行的时候使用的是 JDK 1.4，之所以造成这种原因主要是由于 oracle 本身自己提供了一个自己的 JDK 开发工具，可以通过 path 路径的配置发现问题：

```
Path=D:\oracle\product\10.1.0\db_1\bin;D:\oracle\product\10.1.0\db_1\jre\1.4.2\bin\client;
D:\oracle\product\10.1.0\db_1\jre\1.4.2\bin;C:\WINDOWS\system32;C:\WINDOWS;
C:\WINDOWS\System32\Wbem;C:\Program Files\ATI Technologies\ATI.ACE\Core-Static;D:\Java\jdk1.6.0_02\bin
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
```

现在发现在 oracle 中配置了 JDK 1.4，所以导致以上的程序无法使用，那么现在有两种解决方式：

1、 将所有 oracle 中有关 JDK 的配置删除掉；

2、 由于 path 在读取的时候采用的是顺序读取方式，那么此时，可以将新的配置放在前面。

那么以后在使用命令的时候肯定是先找到自己安装的 JDK 的环境命令，此时再次执行，可以正常运行。

3.4、第一个 Java 程序解释（重点）

第一个 Java 程序完成之后，下面对于 Java 程序来做进一步的了解。

3.4.1、文件名称

在编写 java 程序的时候一定要注意，只要是 java 程序则一定要放在一个类之中，使用如下的语法定义类：

```
[public] class 类名称{}
```

在编写类名称的时候单词的首字母是采用大写的方式进行的，例如：TestHelloJava。

但是，如果一个类使用 public class 和 class 声明的话是有区别的。

- public class：文件名称必须与类名称保持一致。
- class：文件名称可以与类名称不一致，执行的时候执行的是生成的*.class 文件。

所以在一个*.java 的文件中，只能有一个 public class 声明，但是允许有多个 class 声明，在编译之后会生成不同的*.class 文件。

3.4.2、程序理解

在一个 java 程序之中，所有的程序都是从主方法中开始执行，在 java 中主方法定义如下：

```
public static void main(String args[]){}
```

在主方法后面的“{}”之中编写具体的语句，例如：系统输出：

```
System.out.println() ;
```

此语句表示的是在屏幕上进行打印，如果后面有“ln”的话表示输出之后会加一个换行出来，如果没有“ln”表示只是输出而没有换行。

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

3.4.3、classpath 属性

例如：在 d:\testjava\easy 文件夹之中保存着所有的*.class 文件。如果现在在其他路径上，则无法执行，但是现在就非想执行的话，则就需要 classpath 的操作属性。

在正常情况下，一个*.class 文件只能从本目录中被访问，因为在默认的要求下 classpath 就是默认的是从当前所在的文件夹中查找所要的文件，也就是说现在的 java 的运行机制：java 命令 à classpath à *.class。所以按照这种理解，如果将 classpath 指定好了一个位置的话，则以后不管在何种目录下都可以访问了。

```
SET classpath=d:\testjava\easy
```

以上的设置表示的是，以后不管在那里，都从 d:\testjava\easy 文件夹之中找到所需要的*.class 文件，那么即使不在当前类所在文件夹之中，依然可以访问，但是从一个正常的开发习惯来讲，这种配置并不常见，因为最好的执行还是从当前

所在的文件夹中查找类执行最方便，那么一般情况下 classpath 都会存在以下的设置：

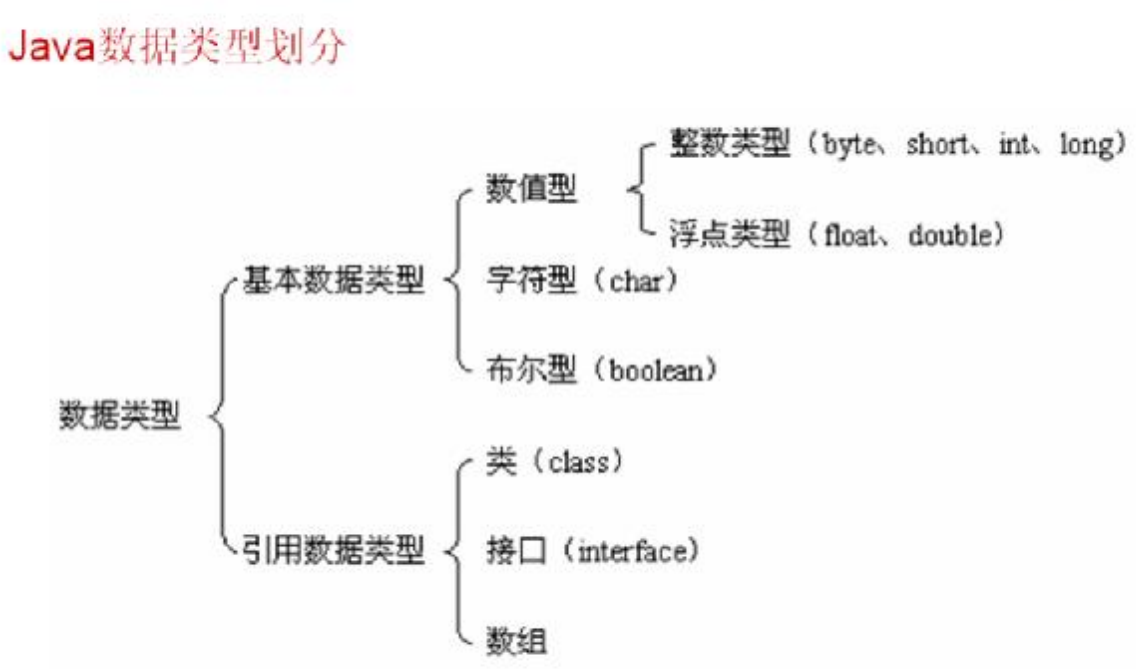
```
SET classpath=.
```

“.”表示的是从当前所在的文件夹之中查找所需要的*.class 文件，而且以上通过命令设置的 classpath 也只是在当前命令行窗口下起作用，如果要想对全体其作用，则需要通过环境属性进行配置，增加一个 classpath 的选项：

需要注意的是：在定义新的变量的时候，所有的单词都必须采用大写的形式出现。

3.5、数据类型（重点）

既然 Java 是一门语言，则在这门语言肯定存在各种数据类型，Java 数据类型分为两种：



其中布尔型只有两种取值范围：true 或 false

在数值型中使用最多的就是：byte、int

在浮点型中：float 和 double 都经常使用，double 数据类型可以装下全宇宙最大的数字。

每种数据类型都会有其相应的取值范围：

Java基本数据类型

No.	数据类型	大小/位	可表示的数据范围
1	long（长整数）	64	-9223372036854775808～9223372036854775807
2	int（整数）	32	-2147483648～2147483647
3	short（短整数）	16	-32768~32767
4	byte（位）	8	-128～127
5	char（字符）	2	0～255
6	float（单精度）	32	-3.4E38（-3.4×10^{38}）～3.4E38（3.4×10^{38}）
7	double（双精度）	64	-1.7E308（-1.7×10^{308}）～1.7E308（1.7×10^{308}）

一般在程序中出现了一个整数都属于 int 型数据，如果出现的是小数则都属于 double 型数据。

范例：定义 int 型数据

```
public class Demo01{
    public static void main(String args[]){
        int x = 10 ;        // 定义整型
        int y = 20 ;        // 定义整型
        System.out.println(x + y) ;
    }
};
```

范例：使用 double 型数据

```
public class Demo02{
    public static void main(String args[]){
        double x = 10.3 ;// 定义小数
        double y = 20.9 ;// 定义小数
        System.out.println(x + y) ;
    }
};
```

如果现在要是想使用 float 的话，则就比较麻烦，需要表示出一个数字应该按照 float 进行接收。

```
public class Demo03{
    public static void main(String args[]){
        float x = 10.3f ;  // 定义小数
```

```
        float y = (float)20.9 ;  // 定义小数
        System.out.println(x + y) ;
    }
};
```

本程序也就相当于属于数据类型的转换问题，一般在进行数据类型转换的时候都会采用由小到大的方式进行，例如：double 的范围大于 float，所以可以自动转型，double 的范围大于 int 也可以自动转型，但是如果由大到小的方式进行的话，则必须采用强制的手段，上面的代码就是采用了强制的方式。

```
public class Demo04 {
    public static void main(String args[]){
        int x = 10 ;
        double y = x ;          // 由小到大
        System.out.println(y * y) ;
    }
};
```

范例：由大到小

```
public class Demo05 {
    public static void main(String args[]){
        double d = 100.9 ;
        int x = (int)d ;
        System.out.println(x * x) ;
    }
};
```

在整型的操作中如果出现了小数位，则肯定会自动抹掉，不会存在。

```
public class Demo06 {
    public static void main(String args[]){
        System.out.println(10 / 3) ;
        // 整型在计算前会先向 float 转型
        System.out.println(10 / (float)3) ;
    }
};
```

在整型的计算中也存在着长整型的数据。

```
public class Demo07 {
    public static void main(String args[]){
        int x = 999999 ;
        int y = 999999 ;
        System.out.println(x * y) ;
    }
};
```

当一个数字的最小值 –1 就是数字的最大值，当一个数字的最大值再加上 1 则变成了最小值。

```
public class Demo08 {
    public static void main(String args[]){
        int max = Integer.MAX_VALUE ; // 整型最大值
        int min = Integer.MIN_VALUE ;  // 整型最小值
        System.out.println(max + 1) ;
```

```
        System.out.println(min - 1) ;
    }
};
```

这种代码就称为数据的溢出，如果要想避免溢出的话，则需要将数据类型扩大。

```
public class Demo09 {
    public static void main(String args[]){
        long x = 999999L ;
        long y = 999999L ;
        System.out.println(x * y) ;
    }
};
```

这个时候实际上就出现了以下的一种题目：

```
public class Demo10 {
    public static void main(String args[]){
        System.out.println(11 + 1l) ;
    }
};
```

字符（char）也是一种常见的操作类型，所有的字符都是使用“''”括起来的一个内容，例如：字母 A 就是一个字符。

```
public class Demo11 {
    public static void main(String args[]){
        char c = 'A' ;      // 字符
        System.out.println(c) ;
    }
};
```

但是，在程序中字符和数字往往是可以相互转换的，转换成 UNICODE 码。

```
public class Demo12 {
    public static void main(String args[]){
        char c = 'A' ;      // 字符
        int x = c ;         // 将字符变为 int 型
        System.out.println(x) ;
        x = x + 1;          // 求出 x 的内容并且加一
        c = (char) x ;      // 将数字变回字符
        System.out.println(c) ;
    }
};
```

在字符的操作中还存在着转义字符，例如：有如下的转移字符：

- '\n'：表示换行
- '\t'：表示 TAB
- '\\'：表示一个“\”
- '\"'：表示一个"
- '\'：表示一个'

范例：验证转义字符

```
public class Demo13 {
    public static void main(String args[]){
```

```
        System.out.println("Hello \nWorld\t!!!") ;
        System.out.println("\"Hello World\"\\\\") ;
    }
};
```

问题：一个字符能否定义一个中文

```
public class Test {
    public static void main(String args[]){
        char c = '国' ;
        System.out.println(c) ;
        System.out.println((int)c) ;
    }
};
```

可以包含中文，但是只能在中文的环境显示中文。

对于 boolean 类型，本身只能存在 true 或 false 两种内容，不能像其他语言那样通过 0 表示 false，非 0 表示 true。一般 boolean 型的数据都会用在流程控制上。

```
public class Demo14 {
    public static void main(String args[]){
        boolean flag = false ;
        if(flag){
            System.out.println("满足条件！") ;
        } else {
            System.out.println("不满足条件！") ;
        }
    }
};
```

以上都属于一些常用的基本数据类型，但是在这些基本数据类型之上有一种 String 类型，表示的是字符串，一般字符串都是使用“""”括起来的数据，所以 String 本身属于一个类，但是此类的使用之后再解释。

```
public class Demo15 {
    public static void main(String args[]){
        String info = "你好啊，美丽的祖国！" ;       // 一串信息
        System.out.println(info + " !!!!!!!!") ;
    }
};
```

在字符串中，“+”表示的是字符串连接，可以定义一串信息。

```
public class Demo16 {
    public static void main(String args[]){
        int x = 10 ;
        int y = 20 ;
        System.out.println(x + " + " + y + " = " + ( x + y ) ) ;
    }
};
```

在使用“+”操作的时候，如果出现了字符串的话，则所有的数据类型都会自动向字符串转换，如果对于某些计算操作，需要正确执行的话，要使用“()”完成。

3.6、标识符及关键字

Java 中一共存在了 7 * 7 = 49 个关键字。

在 Java 的关键字之中，对于 goto 和 const 是两个没有使用到的关键字。

标识符：

可以用来定义方法、类名称、变量名称的一种标记，在 Java 中所有的标识符的定义风格如下：

- 由字母、数字、下划线、$符号组成，其中不能以数字开头，不能是 Java 中的关键字。

例如：demo、x、$Hello

但是从一般的开发来看，一般的标识符用的最多的就是字母、数字、下划线。

3.7、运算符（重点）

程序的主要功能就是计算，所以在 Java 中提供了以下的运算符：

- 赋值运算：=
- 三目运算：布尔表达式 ？ 条件满足 ： 条不满足
- 数学运算符：+、-、*、/、%
- 关系运算符：>、>=、<、<=、==、!=
- 位运算：>>、<<、>>>、^、~
- 逻辑运算：&&、&、||、|、！
- 简便运算符：++、--

以上的各个运算符的使用基本上就是程序构成的主要的语法，但是对于位操作，本身并不严格要求。

范例：三目运算

```
public class Demo17 {
    public static void main(String args[]){
        int x = 10 ;
        int y = 20 ;
        // 如果 x 的内容大于 y，则将 x 的值给 max，否则将 y 的值给 max
        int max = x>y ? x : y ;
        System.out.println(max) ;
    }
};
```

范例：数学运算

```
public class Demo18 {
    public static void main(String args[]){
        int x = 10 ;
        int y = 20 ;
        System.out.println(x + y) ;
        System.out.println(x - y) ;
        System.out.println(x * y) ;
        System.out.println(x / y) ;
        System.out.println(x % y) ;
    }
};
```

范例：关系运算

```
public class Demo19 {
    public static void main(String args[]){
        int x = 10 ;
        int y = 20 ;
        System.out.println(x > y) ;
        System.out.println(x < y) ;
        System.out.println(x >= y) ;
        System.out.println(x <= y) ;
        System.out.println(x != y) ;
        System.out.println(x == y) ;
    }
};
```

以上的各个关系运算符中返回的都是 boolean 型数据，那么如果现在有多个条件要同时判断的话，则需要对条件进行连接：与、或、非

与和或的说明：

在 Java 的运算符操作中，与和或分别有两种使用方法：

- 与：所有条件同时满足
  - |- &：所有的条件都要判断
  - |- &&：短路与，如果前面的条件不满足则后面的不再判断
- 或：所有的条件有一个满足即可

|- |：所有的条件都要判断

|- ||：当前面的条件满足时，后面的条件不再判断

在验证之前先来看以下的一段代码：

```
public class Demo20 {
    public static void main(String args[]){
        System.out.println(10 / 0) ;
    }
};
```

在数学中被除数是不能为 0 的，那么下面就利用以上的关系观察代码。

使用非短路与	使用短路与

public class Demo21 {
public static void main(String args[]){
if(1==2 & 10/0==0){
System.out.println("条件满足") ;
}
}
};
	public class Demo21 {
	public static void main(String args[]){
	if(1==2 && 10/0==0){
	System.out.println("条件满足") ;
	}
	}
	};

所有的条件现在都要判断	之前的条件不满足，所以后面的不再判断

再来观察或的操作：

使用非短路或	使用短路或

public class Demo22 {
public static void main(String args[]){
if(1!=2 | 10/0==0){
System.out.println("条件满足") ;
}
}
};
	public class Demo22 {
	public static void main(String args[]){
	if(1!=2 || 10/0==0){
	System.out.println("条件满足") ;
	}
	}
	};

所有的条件必须判断	第一个满足之后所有的条件不再判断

所以，根据以上的概念可以发现，肯定开发中使用“&&”和“||”操作最为方便。

简便运算符：++、--，这些符号如果习惯于使用，则使用，不习惯就别使。

旧的写法	新的写法

public class Demo23 {
public static void main(String args[]){
int x = 10 ;
x = x + 1 ;
System.out.println(x) ;
}
};
	public class Demo23 {
	public static void main(String args[]){
	int x = 10 ;
	System.out.println(++x) ;
	}
	};

++或--的操作基本上都是在最早的时候，由于内存很昂贵的时候所采用的写法。但是++和--的位置不同，操作的结果也有所不同。

```
public class Demo24 {
    public static void main(String args[]){
        int x = 10 ;
        System.out.println(x++) ;    // 先输出（计算）再自增
```

```
        System.out.println(++x) ;
    }
};
```

但是对于以上的操作代码千万别写成：++x –x – x + x++。

位运算

在进行 Java 的应用开发中，位运算出现的几率并不高，一般而言在一些加密的程序中会使用的到。

但是，在进行位操作之前，先来研究一下十进制变成二进制，除 2 取余：45 二进制：101101

在位操作中，可以使用如下的运算符：&、|、^

范例：验证&操作

```
public class Demo25 {
    public static void main(String args[]){
        int x = 45 ;
        int y = 10 ;
        System.out.println(x & y) ;
        System.out.println(x) ;
    }
};
```

在经过位运算操作之后，原本的内容根本就不改变。

```
public class Demo25 {
    public static void main(String args[]){
        int x = 45 ;
        int y = 10 ;
        System.out.println(x & y) ;
        System.out.println(x) ;
    }
};
```

```
        00000000 00000000 00000000 00101101    45
&       00000000 00000000 00000000 00001010    10

        00000000 00000000 00000000 00001000    8
```

范例：验证或的操作

```
public class Demo26 {
    public static void main(String args[]){
        int x = 45 ;
        int y = 10 ;
        System.out.println(x | y) ;
    }
};
```

将以上的要求变为图形显示：

```
00000000 00000000 00000000 00101101    45
|   00000000 00000000 00000000 00001010    10
-----------------------------------------------
00000000 00000000 00000000 00101111    47
```

还存在着移位操作：

```
public class Demo27 {
    public static void main(String args[]){
        int x = 45 ;
        System.out.println(x>>2) ;
    }
};
```

```
00000000 00000000 00000000 00101101    45
00000000 00000000 00000000 00001011 --> 01    11
```

范例：观察左移位操作

```
public class Demo28 {
    public static void main(String args[]){
        int x = 2 ;
        System.out.println(x<<3) ;
    }
};
```

```
00000000 00000000 00000000 00000010    2
00000000 00000000 00000000 00010000    16
```

3.8、程序控制（重点）

程序的控制部分主要就是判断、循环操作。

判断：if、if…else、switch

```
public class Demo29 {
    public static void main(String args[]){
        int x = 2 ;
        // boolean flag = x > 2 ;
        if(x>=2){          // if(flag)
            System.out.println("条件满足") ;
        }
    }
};
```

也可以多条件进行判断：

```
public class Demo30 {
    public static void main(String args[]){
```

```
        int age = 300 ;
        if(age>=0 && age<=12){
            System.out.println("现在是少年期！") ;
        } else if (age>12 && age<=18){
            System.out.println("现在是青少年期！") ;
        } else if(age>18 && age<=40){
            System.out.println("青年期") ;
        } else if(age>40 && age<=60){
            System.out.println("中老年期") ;
        } else if(age>200) {
            System.out.println("老妖精|！")  ;
        }
    }
};
```

switch 虽然也可以同时判断多个条件，但是一般而言只能判断数字、字母、枚举（JDK 1.5 之后增加的）。

```
public class Demo31 {
    public static void main(String args[]){
        char c = 'A' ;
        switch(c){
            case 'A' :{
                System.out.println("成绩优秀！") ;
                break ;
            }
            case 'B' :{
                System.out.println("成绩中上！") ;
                break ;
            }
            case 'C' :{
                System.out.println("成绩中等！") ;
                break ;
            }
            default:{
                System.out.println("没有满足的条件！") ;
                break ;
            }
        }
    }
};
```

循环语句是整个程序中的重点部分：当某一段代码需要被反复执行的时候就要使用循环的操作，循环使用 for、while、do…while 语句完成。

范例：观察循环

```
public class Demo32 {
    public static void main(String args[]){
        int x = 0 ;          // 循环的初始值
```

```
        while(x<10){    // 写出判断的条件
            System.out.println("x = " + x) ;    // 执行循环语句
            x ++ ;              // 修改循环的条件
        }
    }
};
```

while 循环属于先判断后执行，如果条件满足了，则一直执行下去。

```
public class Demo33 {
    public static void main(String args[]){
        int x = 0 ;         // 循环的初始值
        do{  // 写出判断的条件
            System.out.println("x = " + x) ;    // 执行循环语句
            x ++ ;              // 修改循环的条件
        } while(x<10) ;
    }
};
```

for 循环是使用最多的操作，因为其会将初始内容和循环条件一起编写。

```
public class Demo34 {
    public static void main(String args[]){
        for(int x=0;x<10;x++){
            System.out.println("x = " + x) ;
        }
    }
};
```

在循环中还存在着两个控制循环的语句：break、continue。

```
public class Demo35 {
    public static void main(String args[]){
        for(int x=0;x<10;x++){
            if(x==3){
                break ;
            }
            System.out.println("x = " + x) ;
        }
    }
};
```

```
public class Demo35 {
    public static void main(String args[]){
        for(int x=0;x<10;x++){
            if(x==3){
                continue ;
            }
            System.out.println("x = " + x) ;
        }
    }
};
```

对于循环操作本身也是允许嵌套的。

范例：打印三角形

- 需要两层循环，一层是控制显示的行，另外一层是控制输出“*”或者是“ ”的。

```
public class Demo37 {
    public static void main(String args[]){
        int line = 10 ;    // 定义出要打印的行数
        for(int x=0;x<line;x++){    // 控制行
            for(int y=0;y<line-x;y++){
                System.out.print(" ") ;
```

```
                }
                for(int y=0;y<=x;y++){
                        System.out.print("* ") ;
                }
                System.out.println() ;  // 换行
        }
    }
};
```

范例：打印九九乘法表

```
public class Demo38 {
    public static void main(String args[]){
        for(int x=1;x<10;x++){
            for(int y=1;y<=x;y++){
                System.out.print(x + "*" + y + "=" + x * y + "\t") ;
            }
            System.out.println() ;
        }
    }
};
```

3.9、方法（绝对重点）

在各个语言中实际上都有函数的定义，那么函数在 Java 中称为方法。

3.9.1、方法的基本概念

在程序中，如果某一段代码需要被重复调用的时候，就可以通过方法来完成，但是由于现在的方法是采用主方法直接调用的形式完成的，所以方法的定义格式如下：

```
public static 返回值类型 方法名称(参数类型 参数名称,.....){
    [return 返回值] ;
}
```

现在所编写的方法必须严格的按照以上的要求编写。

范例：定义方法

```
public class MethodDemo01{
    public static void main(String args[]){
        fun() ;
        fun() ;
        fun() ;
    }
    public static void fun(){
        System.out.println("**************************") ;
        System.out.println("*       JAVA 一起学！       *") ;
        System.out.println("**************************") ;
```

```
    }
};
```

以上定义了一个 fun()方法，但是在 fun()方法之中并没有返回值，所以在返回值类型上明确的表示出现在是无返回值是，使用 void 进行声明。

如果需要一个方法存在返回值的话，则必须在返回值类型上写清楚具体的数据类型。

```
public class MethodDemo02 {
    public static void main(String args[]){
        System.out.println(add(10,20)) ;
    }
    public static int add(int x,int y){    // 定义两个数字相加
        return x + y ;
    }
};
```

一般在某些代码需要重复编写的时候就会将其定义方法以供调用。

3.9.2、方法的重载

OverLoading（重载）是语言的一个重要特性，例如：有如下一个要求，建立一个方法，此方法可以计算 int 型、float 型数据的相加操作，那么如果现在按照最早的做法：

```
public class MethodDemo03 {
    public static void main(String args[]){
        System.out.println(add1(10,20)) ;
        System.out.println(add2(10.0f,20.3f)) ;
        System.out.println(add3(10,20,30)) ;
    }
    public static int add1(int x,int y){  // 定义两个数字相加
        return x + y ;
    }
    public static float add2(float x,float y){  // 定义两个数字相加
        return x + y ;
    }
    public static int add3(int x,int y,int z){    // 定义两个数字相加
        return x + y + z ;
    }
};
```

以上的功能实现了，但是有问题：现在的操作实际上都属于加法操作，但是将方法名称拆分了，如果这种定义的话，假设有 100 个类似方法的时候就已经晕了，所以这个时候就可以使用方法重载的概念。

方法重载：方法名称相同，参数的类型或个数不同

```
public class MethodDemo04 {
    public static void main(String args[]){
        System.out.println(add(10,20)) ;
        System.out.println(add(10.0f,20.3f)) ;
        System.out.println(add(10,20,30)) ;
```

```
    }
    public static int add(int x,int y){    // 定义两个数字相加
        return x + y ;
    }
    public static float add(float x,float y){    // 定义两个数字相加
        return x + y ;
    }
    public static int add(int x,int y,int z){    // 定义两个数字相加
        return x + y + z ;
    }
};
```

同样的方法，会根据传递的参数类型或个数不同以完成不同的功能，但是，来观察以下的一个操作：

```
public class MethodDemo05 {
    public static void main(String args[]){
        System.out.println(10) ;
        System.out.println(10.3) ;
        System.out.println('A') ;
        System.out.println("Hello") ;
        System.out.println(true) ;
    }
};
```

可以发现，System.out.println()方法就属于被重载过的方法，所以同一个方法名称，根据传入的参数类型或个数不同可以完成不同的功能。

但是，有一点需要注意的是，在方法的重载中只是根据参数类型或个数来区分的，与返回值是否相同无关。

```
public class MethodDemo06 {
    public static void main(String args[]){
        System.out.println(add(10,20)) ;
        System.out.println(add(10,20)) ;
    }
    public static int add(int x,int y){    // 定义两个数字相加
        return x + y ;
    }
    public static float add(int x,int y){ // 定义两个数字相加
        return x + y ;
    }
};
```

3.9.3、方法的递归调用（理解）

递归调用：就是自己调用自己，但是如果在实际的开发中使用了递归操作的话，那么有可能造成内存溢出问题，所以开发中基本上是不建议使用递归完成操作的，例如：现在有如下一段代码：

```
public class MethodDemo07 {
    public static void main(String args[]){
```

```
        int sum = 0 ;
        for(int x=1;x<=100;x++){
            sum += x ;
        }
        System.out.println(sum) ;
    }
};
```

以上的代码中可以发现：在循环中有循环的开始和结束条件，以上的代码也可以使用递归的方式完成，但是一旦使用递归之后，一定要注意，要设置好递归的结束条件。

```
public class MethodDemo08 {
    public static void main(String args[]){
        System.out.println(add(100)) ;
    }
    public static int add(int temp){
        if(temp==1){    // 递归的结束控制
            return 1 ;
        } else {
            return temp + add(temp-1) ; // 递归调用，自己调用自己
            // 相当于：add(100) + add(99) + add(98) + ... + add(1)
        }
    }
};
```

3.9.4、题目

要求计算以下程序的结果：1! + 2! + 3! + …. + 35!，求出结果。

方法一：通过普通的循环完成

```
public class MethodDemo09 {
    public static void main(String args[]){
        double sum = 0.0 ;
        for(int x=1;x<35;x++){
            double temp = 1 ;
            for(int y=1;y<=x;y++){
                temp *= y ;
            }
            sum += temp ;
        }
        System.out.println(sum) ;
    }
};
```

方法二：通过递归完成

```
public class MethodDemo10 {
    public static void main(String args[]){
```

```
        double sum = 0.0 ;
        for(int x=1;x<35;x++){
            sum += fun(x) ;
        }
        System.out.println(sum) ;
    }
    public static double fun(int temp){
        if(temp==1){
            return 1 ;
        }else{
            return temp * fun(temp-1) ;
        }
    }
};
```

3.10、数组（重点）

数组本身属于引用数据类型，所以在使用的时候要过多的考虑内存的关系。

3.10.1、数组的基本概念

数组：就是一组相关变量的集合，例如：现在如果按照最早的概念要定义 100 个整型变量：i1、i2、…i100，很明显这样管理起来会很麻烦，那么可以通过数组的方式来进行管理，在 Java 中数组有两种定义的方式。

格式一：声明并开辟数组

数据类型 数组名称[] = new 数据类型[长度] ;
数据类型 [] 数组名称 = new 数据类型[长度] ;

格式二：分两步完成

声明数组：	数据类型 数组名称[] = null ;
开辟数组空间：	数组名称 = new 数据类型[数组长度] ;

一定要注意，数组本身是引用数据类型，所以引用数据类型的默认值是 null。

在定义各个变量的时候一定要给出变量的初始内容。

上面的两种格式都属于数组的动态初始化的方式，即：里面的所有内容都是默认值。

```
public class ArrayDemo01 {
    public static void main(String argbs[]){
        int data[] = new int[10] ;      // 定义 10 个大小的数组
        System.out.println("数组长度是：" + data.length) ;
        for(int x=0;x<data.length;x++){
            data[x] = x ;
        }
        for(int x=0;x<data.length;x++){
            System.out.print(data[x] + "、") ;
        }
```

```
    }
};
```

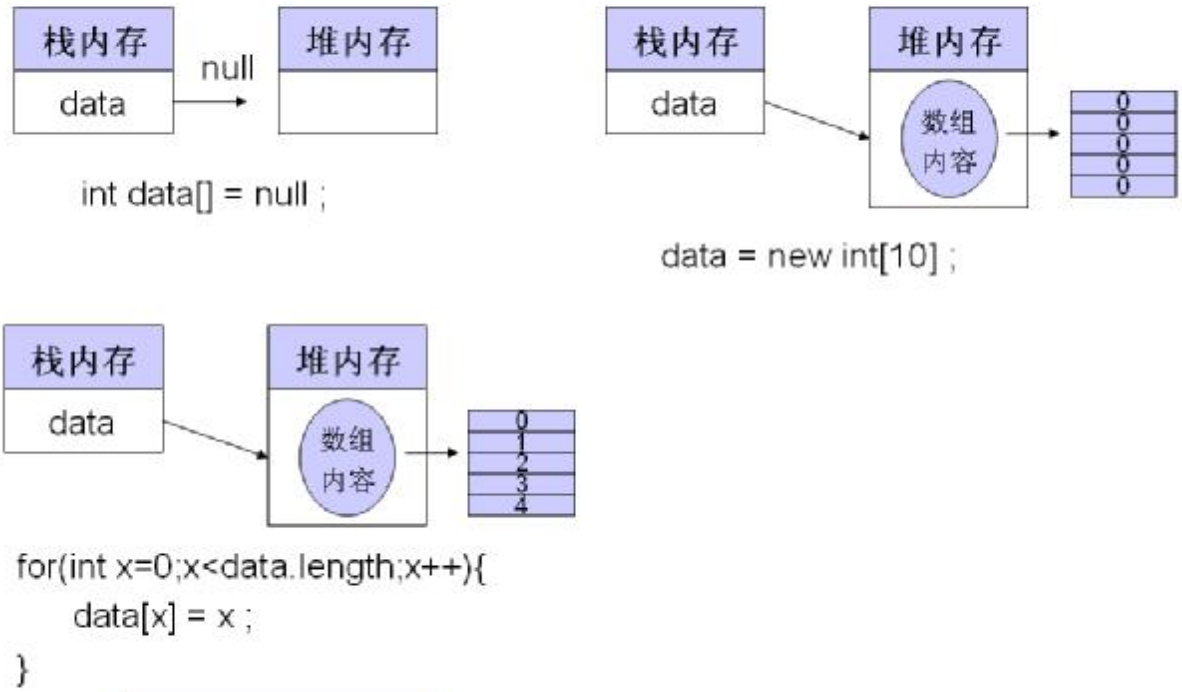
那么，此时，数组的一切都已经可以正常使用了，但是下面就要分析了，以上的程序到底经过了那些步骤进行。

```
public class ArrayDemo02 {
    public static void main(String argbs[]){
        int data[] = null ;        // 声明数组
        data = new int[10] ;    // 开辟空间
        System.out.println("数组长度是：" + data.length) ;
        for(int x=0;x<data.length;x++){
            data[x] = x ;
        }
        for(int x=0;x<data.length;x++){
            System.out.print(data[x] + "、") ;
        }
    }
};
```

下面来分析数组的内存关系，观察以上代码的每一步到底做了那些事情。

在分析之前必须先强调两块内存的概念：

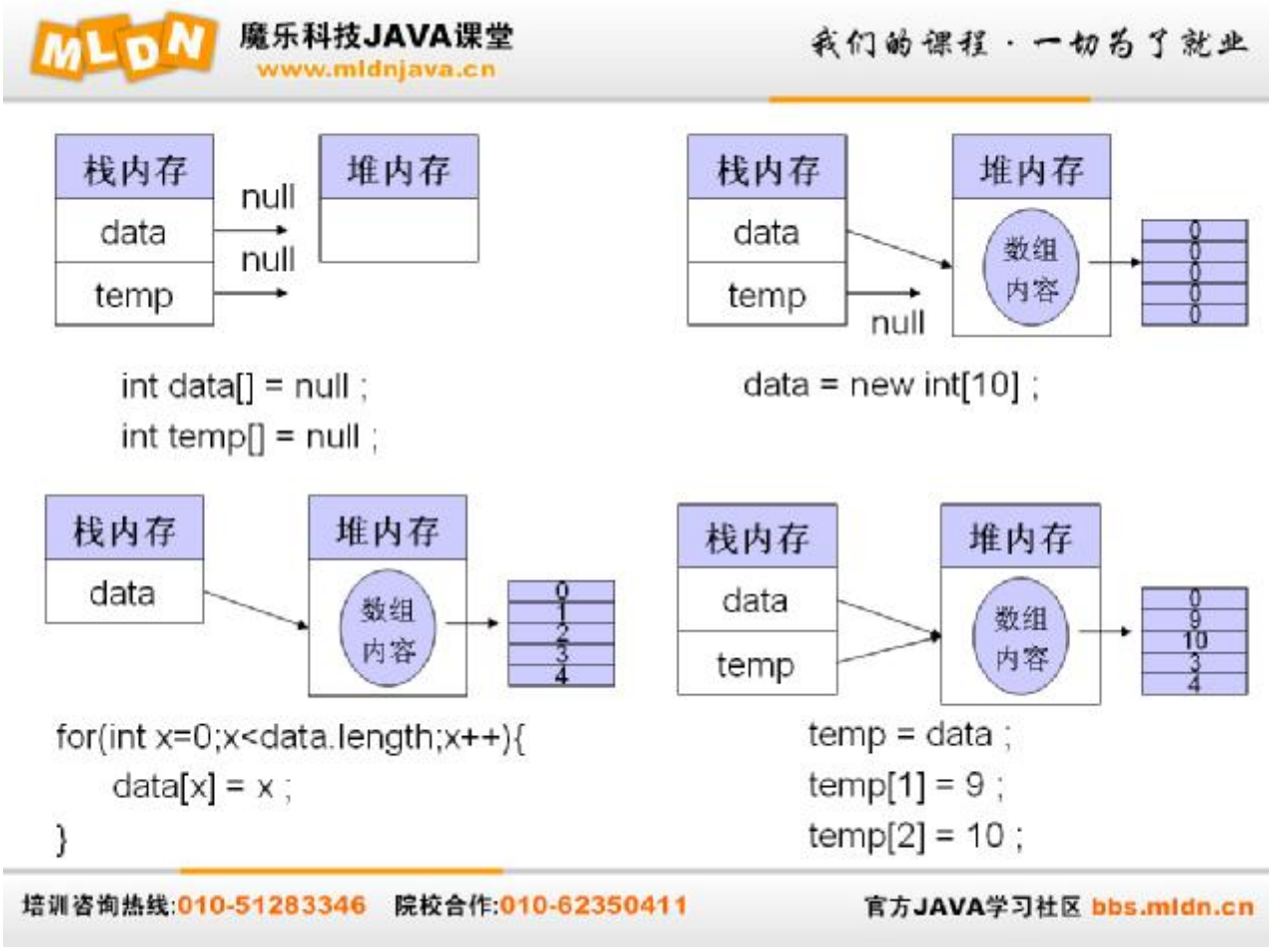
- 所有的数组的具体内容实际上都保存在了堆内存之中
- 但是堆内存的内容一般都要通过一个地址访问，为了方便的地址的操作，数组名称就表示了一个地址。所有的堆内存的访问地址实际上都是在栈内存中保存的，以后为了方便，所以说表示成：数组名称保存在栈内存中。



从生活的方式来看，一个人可能还会有小名，证明一个人会有多个名字。所以，一个堆内存空间，可以同时被多个栈内存所指向，并且每一个名字的修改都会直接影响到堆内存空间的内容。

```
public class ArrayDemo03 {
    public static void main(String argbs[]){
        int data[] = null ;        // 声明数组
        int temp[] = null ;        // 声明数组
        data = new int[10] ;    // 开辟空间
        System.out.println("数组长度是：" + data.length) ;
        for(int x=0;x<data.length;x++){
            data[x] = x ;
        }
        temp = data ;              // 将 data 数组的内容给了 temp
        temp[1] = 9 ;              // data 也要变化
        temp[2] = 10 ;             // data 也要变化
        for(int x=0;x<data.length;x++){
            System.out.print(data[x] + "、") ;
        }
    }
};
```

本程序的内存关系图如下：



这种内存的操作就称为引用传递。

在数组的操作中有许多的操作的代码形式，下面来看一个：

范例：求出数组的最大值

```
public class ArrayDemo04 {
    public static void main(String argbs[]){
        int data[] = null ;        // 声明数组
        int temp[] = null ;        // 声明数组
        data = new int[10] ;    // 开辟空间
        System.out.println("数组长度是：" + data.length) ;
        for(int x=0;x<data.length;x++){
            data[x] = x ;
        }
        temp = data ;              // 将 data 数组的内容给了 temp
        temp[1] = 9 ;              // data 也要变化
        temp[2] = 10 ;             // data 也要变化
        int max = data[0] ;        // 取出第一个元素
        for(int x=0;x<data.length;x++){
            if(data[x]>max){
                max = data[x] ;  // 修改 max 的内容
            }
        }
        System.out.println("最大值是：" + max) ;
    }
};
```

3.10.2、静态初始化

以上的代码都是在数组生命之后为数组里面的内容进行赋值，属于动态初始化的操作，数组本身也存在着静态初始化，可以在定义数组的时候直接指定出其内容。

```
public class ArrayDemo05 {
    public static void main(String argbs[]){
        int data[] = {1,3,4,1,3,54,5,43,3,2,1,5,7,87} ;
        for(int x=0;x<data.length;x++){
            System.out.print(data[x] + "、") ;
        }
    }
};
```

但是以上的内容本身是属于无序的操作，如果需要排序呢？可以使用冒泡排序。

```
public class ArrayDemo06 {
    public static void main(String argbs[]){
        int data[] = {1,3,4,1,3,54,5,43,3,2,1,5,7,87} ;
        for(int x=0;x<data.length;x++){
            for(int y=0;y<data.length-1;y++){
                if(data[y] < data[y+1]){
```

```
                    int temp = data[y] ;
                    data[y] = data[y+1] ;
                    data[y+1] = temp ;
                }
            }
        }
        for(int x=0;x<data.length;x++){
            System.out.print(data[x] + "、") ;
        }
    }
};
```

3.10.3、二维数组

二维数组就是相当于一张表的结构。

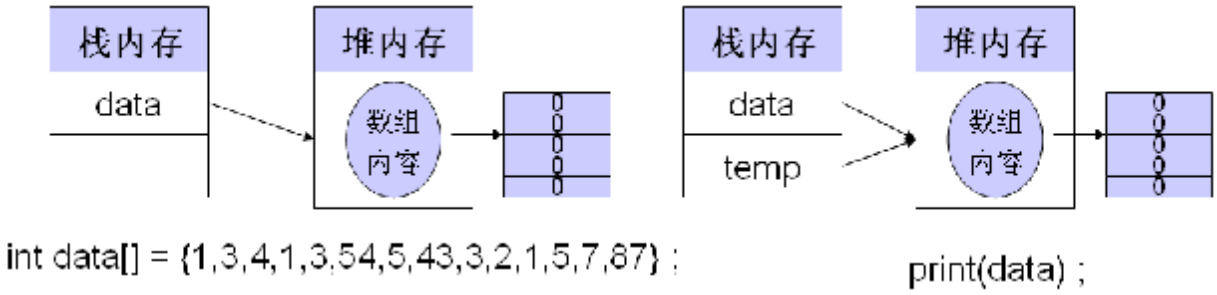
```
public class ArrayDemo07 {
    public static void main(String argbs[]){
        int data[][] = {{1,3,5} , {2,5} , {7,8,9,10}} ;
        for(int x=0;x<data.length;x++){
            for(int y=0;y<data[x].length;y++){
                System.out.print(data[x][y] + "、") ;
            }
            System.out.println() ;
        }
    }
};
```

一般在 java 中最多只用到二维数组，而且情况出现的较少。

3.10.4、数组与方法（重点）

一个数组实际上是在堆内存中保存的一快数据代码，而且一个堆内存中本身可以被多个栈内存所指向，那么实际上这一点在方法的使用上也非常的类似。

```
public class ArrayDemo08 {
    public static void main(String argbs[]){
        int data[] = {1,3,4,1,3,54,5,43,3,2,1,5,7,87} ;
        print(data) ;            // int temp[] = data ;
    }
    public static void print(int temp[]){
        for(int x=0;x<temp.length;x++){
            System.out.print(temp[x] + "、") ;
        }
    }
};
```



以上的代码是将内容交给了方法进行输出，同样可以将内容交给方法处理，例如：定义一个可以排序的方法。

```
public class ArrayDemo09 {
    public static void main(String argbs[]){
        int data[] = {1,3,4,1,3,54,5,43,3,2,1,5,7,87} ;
        sort(data) ;
        print(data) ;               // int temp[] = data ;
    }
    public static void sort(int temp[]){
        for(int x=0;x<temp.length;x++){
            for(int y=0;y<temp.length-1;y++){
                if(temp[y] < temp[y+1]){
                    int t = temp[y] ;
                    temp[y] = temp[y+1] ;
                    temp[y+1] = t ;
                }
            }
        }
    }
    public static void print(int temp[]){
        for(int x=0;x<temp.length;x++){
            System.out.print(temp[x] + "、") ;
        }
    }
};
```

以上表示的是一个方法接收数组，但是一个方法依然可以返回一个数组，只需要在返回值类型上明确的指出，返回的是数组类型即可。

```
public class ArrayDemo10 {
    public static void main(String argbs[]){
        int data[] = init() ;
        print(data) ;               // int temp[] = data ;
    }
    public static int[] init(){     // 返回初始化的数组内容
        int x[] = {1,2,34,3,67,45,3} ;
        return x ;
    }
    public static void print(int temp[]){
```

```
        for(int x=0;x<temp.length;x++){
            System.out.print(temp[x] + "、") ;
        }
    }
};
```

3.10.5、Java 对数组的支持

在 Java 的开发平台之中，为了方便开发，已经提供了很多的支持的操作方法，也有许多专门为数组操作的方法。

范例：数组排序

```
public class ArrayDemo11 {
    public static void main(String argbs[]){
        int data[] = {1,3,4,1,3,54,5,43,3,2,1,5,7,87} ;
        java.util.Arrays.sort(data) ;       // 排序
        print(data) ;               // int temp[] = data ;
    }
    public static void print(int temp[]){
        for(int x=0;x<temp.length;x++){
            System.out.print(temp[x] + "、") ;
        }
    }
};
```

也可以完成数组的拷贝操作，可以将一个数组的部分内容拷贝到另外一个数组之中。

```
public class ArrayDemo12 {
    public static void main(String argbs[]){
        // 最终结果，data1 是：{1,2,3,55,66,77,7,8,9}
        int data1[] = {1,2,3,4,5,6,7,8,9} ;
        int data2[] = {11,22,33,44,55,66,77,88,99} ;
        // 系统方法：System.arraycopy(源数组，源数组开始点，目标数组，目标数组开始点，长度) ;
        System.arraycopy(data2,4,data1,3,3) ;
        print(data1) ;
    }
    public static void print(int temp[]){
        for(int x=0;x<temp.length;x++){
            System.out.print(temp[x] + "、") ;
        }
    }
};
```

以上的两个操作属于 JDK 内部的支持，只需要记下来如何使用即可。

4、总结

1、 Java 实现可移植性的操作原理：*.java à *.class à JVM à OS
2、 public class 和 class 声明类的区别
3、 classpath 及 path 属性的作用
4、 数据类型的划分
5、 各个操作运算符、流程控制
6、 方法的作用及重载的使用
7、 数组及内存的分配问题

5、预习任务

面向对象的主要特征、类与对象的关系、构造方法、private 关键字、String 类、引用传递。