一：基础

1.public class类名称｛｝：文件名称必须和类名称保持一致，一个\*.java文件里面只能够有一个public class 定义。

2.class 类名称｛｝：文件名称可以与类名称不一致，但是生成的\*.class是更根据文件中定义的class名称一致的。

3.屏幕输出：

输出之后增加换行：System.out.println(内容)；

输出之后不增加换行: System.out .print(内容)；

4.

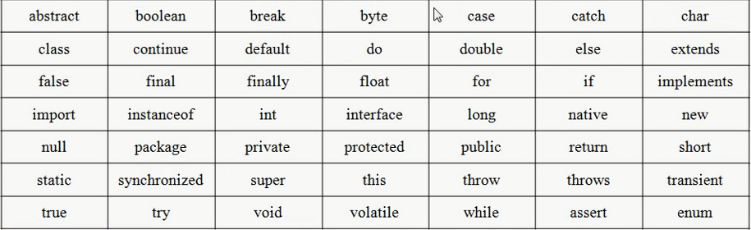
PATH：是属于操作系统属性，定义所有的可执行程序的路径；

CLASSPATH：是java程序解释类程序文件时所使用的加载路径。

默认路径设置：classpath=.

5.类名称由字母、数字、\_、$组成，$符号有特殊意义不要去使用。

6.Java关键字



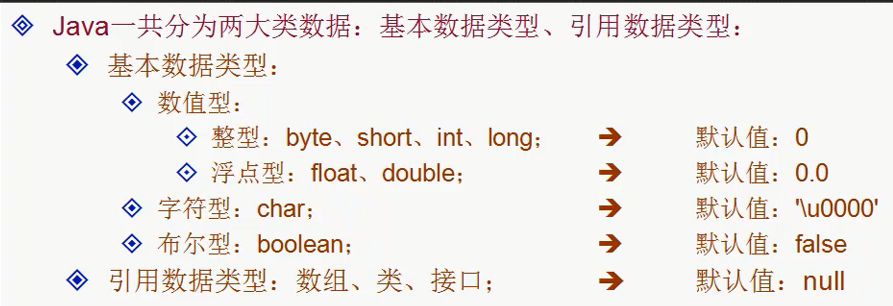
①java有两个未使用到的关键字：goto、const；

②Java有三个特殊的标记：true、false、null；

③JDK1.4之后增加了assert关键字；

④JDK1.5之后增加了enum关键字；

⑤可以是中文（但是不建议这样使用）；

7. 

字符型默认值为’\u0000’，引用数据类型：数组、类、接口。

8.基本类型不牵扯到内存分配，而引用数据类型需要开发者为其分配内存空间。

9. 

10.**个人选择数据类型原则**

10.1如果想表示整数就使用int，表示小数就使用double

10.2如果要描述日期时间数字或表示文件（或内存）大小使用long；

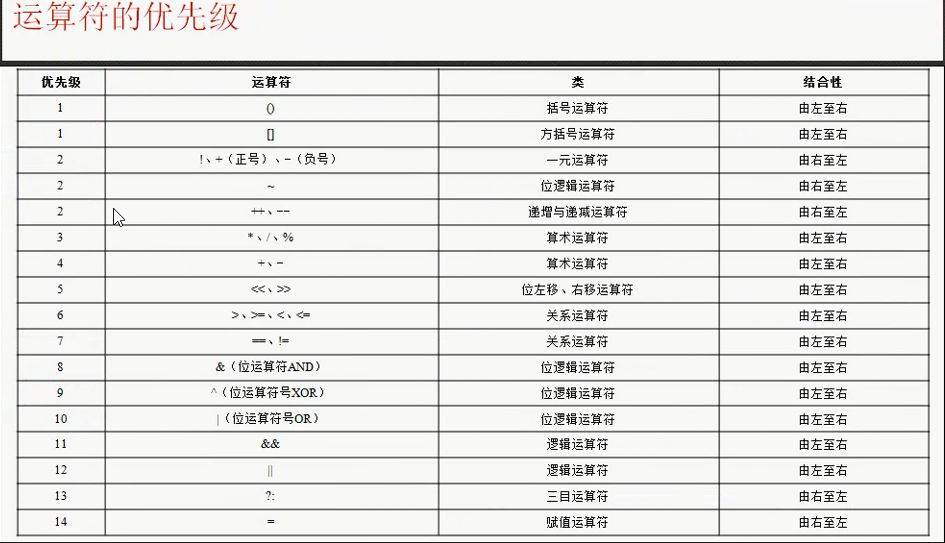
10.3如果要实现内容传递或者是编码转换使用byte；

10.4如果要想实现逻辑的控制，就使用boolean描述；

10.5如果想使用中文，使用char可以避免乱码问题；

11.java不允许使用0和1来填充boolean型数据，只能使用true和false来为boolean型数据赋值。

12.string属于引用数据类型，string属于类。  
13.在Java里面只要是类名称，每一个单词的首字母都是大写的，

14. 

15.普通与（&）和短路与（&&）的区别：

普通与第一个条件为false仍对第二个条件判断，短路与当第一个条件为false后不对第二个条件判断。

普通或（|）和短路或（||）的区别：

普通与第一个条件为true仍对第二个条件判断，短路与当第一个条件为true后不对第二个条件判断。

使用高效的短路与和短路或。

16.位与运算（&）位或运算（|），&&和||不能应用在位运算上。

17.while分为两种循环语句：

while｛

循环语句;

修改循环结束条件;

｝

do{

循环语句；

修改循环结束条件；

}while();

18.continue：退出本次循环。

Break:退出整个循环。

19.方法名称第一个单词首字母小写，其余单词首字母大写。

二：面向对象：

1. OOA（面向对象分析）、OOD（面向对象设计）、OOP（面向对象编程）
2. NULLPointerException(空指向异常)只要是引用数据类型都有可能出现。
3. 堆内存保存的是对象中的所有属性的内容，而栈内存保存的是堆内存的地址数值。
4. 构造方法只在实例化新对象（new）的时候调用一次。
5. 数组是引用型的数据。
6. 数组的静态初始化：两种格式

⑴简化格式：

数据类型 数组名称[] = {值,值,值,......};

⑵

数据类型 数组名称[] = new 数据类型[] {值,值,值,......};

1. 数组有一个最大的缺点：长度不能被改变。
2. 数组拷贝：System.arraycopy(原数组名称,原数组拷贝开始索引,目标数组名称,目标数组拷贝索引,长度);
3. 数组排序：java.util.Arrays.sort(数组名称);
4. Suoyou 引用数据类型都可以称为对象数组
5. 比较内容（与原始定义有一些差别）：public boolean equals(String str);例：stra.equals(strb);
6. 请解释在字符串相等的判断中“==”与“equal()”的区别？

·“==”是Java提供的关系运算如果用在符，主要的功能是进行数值相等进行判断，如果用在String对象上表示内存数值的比较；

·“equaals()”:是String提供的一个方法，此方法专门负责进行字符串 内容得比较。

1. 如果要判断输入的内容是否是某一字符串，请一定将字符串写在前面。（利用字符创常量为String类对象的匿名对象来避免空指向异常）
2. 请解释String对象两种实例化方式的区别？

* 直接赋值（Stirng str = “字符串”；）：只会开辟一块堆内存空间并且会自动保存在对象池之中以供下次重复使用。
* 构造方法（String str = new String(“字符串”)）：会开辟两块内存空间，不会自动入池，并且其中一块内存空间会成为垃圾，但用户可以采用intern()方法手工入池。

1. String的内容一旦声明则不能改变

疑似改变String内容的例子：

Public class StringDemo{

Public static void main(){

String str = “Hello” ;

str = str + “World” ;

str += “!!!” ;

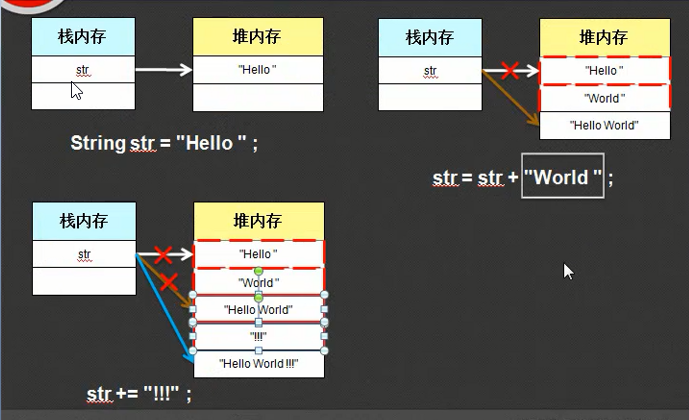
System.out.println(str) ;

}

}

释疑：

以上操作改变是利用引用的关系改变来改变str；每一次改变都会产生垃圾空间。



1. Stirng类的特点：

* String类对象的相等判断使用equals()方法完成，“==”实现的是地址数值的比较。
* 字符串内容一旦声明则不可改变，String类对象内容的改变是依靠引用关系的变更来实现的。
* String类有两种实例化方式，使用直接赋值可以不产生垃圾空间，并且可以自动入池，不要使用构造方法。

17. String方法必须全部背会。

18. Java apidocx:

·类中的成员组成

·类中所提供的构造方法；

·类中所提供的普通方法；

·成员、构造方法、普通方法的详细说明。

|  |  |  |  |
| --- | --- | --- | --- |
| No | 方法名称 | 类型 | 描述 |
| 1 | public String(char[] value) | 构造 | 将一个字符数组变为Sring类对象 |
| 2 | public String(char[] value,int offset, int count) | 构造 | 将部分数组变为String |
| 3 | public char charAt(int index) | 普通 | 返回字符串指定索引对应的字符 |
| 4 | public char[] toCharArray() | 普通 | 将字符串转换为字符数组 |

19.字节一般用于数据的传输和编码转换

|  |  |  |  |
| --- | --- | --- | --- |
| No | 方法名称 | 类型 | 描述 |
| 1 | public String(byte[] bytes) | 构造 | 将全部的字节数组变为字符串 |
| 2 | public String(byte[] bytes, int offset, int length) | 构造 | 将部分字节数组变为字符串 |
| 3 | public byte[] getBytes() | 普通 | 将字符串变为字节数组 |
|  | public byte[] getBytes(String charsetName) throws UnsupportedEncodingException | 普通 | 进行编码转换 |

20. 字符串相等判断

|  |  |  |  |
| --- | --- | --- | --- |
| No | 方法名称 | 类型 | 描述 |
| 1 | public boolean equals(String anString) | 普通 | 相等判断，区分大小写 |
| 2 | public boolean equalsIgnoreCase(String anotherString) | 普通 | 相等判断，不区分大小写 |
| 3 | public **int** compareTo(String anotherString) | 普通 | 判断两个字符串的大小（按照字符编码比较），此方法返回值有如下三种结果  • =0:表示要比较的字符串 内容相等;  • >0:表示大于的结果;  • <0:表示小于的结果; |

21.字符串查找

|  |  |  |  |
| --- | --- | --- | --- |
| No | 方法名称 | 类型 | 描述 |
| 1 | public **boolean** contains(String s) | 普通 | 判断指定的字符串是否存在 |
| 2 | public int indexOf(Stirng str) | 普通 | 由前向后查找指定字符串的位置，如果找到了返回（第一个字母）位置索引，如果找不到返回-1。 |
| 3 | public int indexOf(String str,int fromIndex) | 普通 | 由指定位置开始由前向后查找指定字符串的位置，如果找到了返回（第一个字母）位置索引，如果找不到返回-1。 |
| 4 | public int lastIndexOf(String str) | 普通 | 由后向前查找指定字符串的位置，如果找到了返回（最后一个字母）位置索引，如果找不到返回-1。 |
| 5 | public int lastIndexOf(String str,  int fromIndex) | 普通 | 由指定位置开始由后向前查找指定字符串的位置，如果找到了返回（最后一个字母）位置索引，如果找不到返回-1。 |
| 6 | public boolean startsWith(String prefix) | 普通 | 判断是否以指定的字符串开头如果是返回true 否返回false |
| 7 | public boolean startsWith(String prefix,  int toffset) | 普通 | 从指定位置判断是否以指定的字符串开头如果是返回true 否返回false |
| 8 | public boolean endsWith(String suffix) | 普通 | 判断是否以指定的字符串结尾如果是返回true 否返回false |

21.字符串的替换

|  |  |  |  |
| --- | --- | --- | --- |
| No | 方法名称 | 类型 | 描述 |
| 1 | public String replaceAll(String regex,  String replacement) | 普通 | 用新的内容替换掉全部旧的内容 |
| 2 | public String replaceFirst(String regex,  String replacement) | 普通 | 替换首个满足条件的内容 |

22.字符串的截取

|  |  |  |  |
| --- | --- | --- | --- |
| No | 方法名称 | 类型 | 描述 |
| 1 | public String substring(int beginIndex) | 普通 | 从指定索引截取到结尾 |
| 2 | public String substring(int beginIndex,int endIndex) | 普通 | 截取部分子字符串的数据 |

23.字符串拆分

|  |  |  |  |
| --- | --- | --- | --- |
| No | 方法名称 | 类型 | 描述 |
| 1 | public String[] split(String regex) | 普通 | 按照指定的字符串进行拆分 |
| 2 | public String[] split(String regex, int limit) | 普通 | 按照指定的字符串进行拆分，最后的数组长度就是由limit决定的 |

注：如果拆分时只写了一个空字符串（””不是null），则按照每一个字符拆分。

如果是一些敏感字符（正则标记）严格讲是拆分不了的，如果遇到拆分不了的情况使用\\（即\）进行转义后才可以拆分。

24.String的其他方法

|  |  |  |  |
| --- | --- | --- | --- |
| No | 方法名称 | 类型 | 描述 |
| 1 | public String concat(String str) | 普通 | 字符串的连接与“+”类似 |
| 2 | public String toLowerCase() | 普通 | 转小写 |
| 3 | public String toUpperCase() | 普通 | 转大写 |
| 4 | public String trim() | 普通 | 去掉字符串左右两边的空格中间的空格保留 |
| 5 | public int length() | 普通 | 取得字符串的长度 |
| 6 | public String intern() | 普通 | 数据入池 |
| 7 | public boolean isEmpty() | 普通 | 判断是否为空字符串（不是NULL是””）即长度为零的字符串 |

注：isEmpty相当于””.equals(str);

25.只要访问类中的属性必须使用this。

26.简单Java类的编写原则：

• 类名称 = 表名称

• 属性名称（类型） = 表字段（类型）

• 一个实例化对象 = 一行记录

• 多个实例化对象（对象数组） = 多行记录 （外键）

• 引用关系 = 外键约束

27. 合成设计模式

28.数据表与简单Java类的映射

第一步：设置基本字段

第二步：设置关联字段（引用）

第三步：设置关系以及数据的取出

29、static方法不能直接访问和调用非static属性和方法，只能狗调用static方法。

非static方法可以不受限制的访问和调用static属性和方法

30、

属性用对象保存

类没有属性使用static 方法

31、如果将代码块写在类里面，那么这个代码码块 被称为构造块

32、实例化类对象时，虽然执行的是类的构造方法，但是最终构造块的调用优先构造方法执行。

33、静态块优先于构造块执行，且不论有多少实例化对象，静态块只执行一次。

34、静态块优先于主方法执行。

35、代码块在开发中能别用就别用，唯一好用的是静态块。（了解）

**内部类：**

36、a．内部类的基本结构；b.使用static 定义内部类；c.在方法中定义内部类；

37、内部类可以方便的访问外部类中定义的一个私有属性。同样外部类也可以方便的访问内部类的私有属性

38、内部类可以使用private声明，声明之后无法在外部实例化内部类对象；

Outer$Inner.class所有$是在文件中的命名，如果换回到了程序里面就变了“.”，也就是说内部类的名称就是“外部类.内部类”。此时可以给出内部类对象的实例化语法：

外部类.内部类 对象 = new 外部类().new 内部类();

39、如果一个内部类使用了static定义的话，那么这个内部类就相当于定义了一个外部类。

此时如果想要取得内部类的实例化对象，使用的语法如下：

外部类.内部类 对象 = new 外部类.内部类();

此时不再需要先产生外部类对象，再产生内部类对象，仿佛就变成了一个独立的类。

40、方法中的内部类可以直接访问方法的参数以及定义的变量，但是这种操作只适合JDK1.8之后的版本。如果在JDK1.7以及之前的版本有个严格要求：方法中定义的内部类如果要想访问方的参数或者是方法定义的变量，那么参数或变量前一定要加上final标记。

**继承：**

41、继承关键字： extends

class 子类 extends 父类｛｝

42、Java不允许多继承。

43、所有私有属性属于隐式继承，所有非私有属性属于显式继承。

44、在实例化子类对象前，先实例化了父类对象以及调用了父类的构造方法 。

那么此时对于子类构造而言，就相当于隐含了一个”super()”

如果说此时父类中没有无参构造方法，那就必须使用super()明确调用父类有参构造方法。

45、面试题：请解释重载与覆写的区别？（请解释Overloading 和 Overrid 的区别？）

|  |  |  |
| --- | --- | --- |
| 区别 | 重载 | 覆写 |
| 英文单词 | Overloading | Overrid |
| 发生范围 | 发生在一个类里面 | 发生在继承关系中 |
| 定义 | 方法名称相同 | 方法名称相同、参数的类型、个数相同、方法返回值相同 |
| 权限 | 没有权限的限制 | 被覆写的方法不能拥有比父类更为严格的访问控制权限 |

46、

|  |  |  |
| --- | --- | --- |
| 区别 | This | Super |
| 功能 | 调用本类构造、本类方法、本类属性 | 子类调用父类方法、父类属性 |
| 形式 | 先查找本类中是否存在有指定的调用结构、如果有则直接调用，如果没有则调用父类定义 | 不查找子类，直接调用父类操作 |
| 特殊 | 表示本类的当前对象 | — |

47、开发中，对于本类或者父类的中的操作，强烈建议加上“this.”或者“super.”方便区分子类或父类的属性和操作。

48、final定义的方法和类不能继承或覆写 final详解： <http://www.importnew.com/7553.html>

49、所有常量名称都使用大写字母表示

50、public final static 声明全局常量都用大写字母表示

51、对象转型：

|- 向上转型：父类 父类对象 = 子类实例

|- 想下转型：子类 子类对象= （子类）父类实例

• 80%的情况都会使用向上转型，因为可以得到参数类型的统一方便于我们使用。

• 5%的情况会使用向下转型，目的是调用子类的特殊方法：

•15%的情况下是不转型，例如：String

52、对象 instanceof类 返回boolean型

如果某个对象是某个类的实例，那么就返回true，否则返回false。

53、抽象类的继承子类里面会有明确的方法覆写要求，而普通类并没有；

54、普通类对象可以直接实例化，但是抽象类必须经过向上转型后才可以实例化；

55、实际开发中一个子类可以去继承任意的一个普通类，但是从开发的实际要求来讲，普通类只能继承抽象类。

56、抽象类必须有子类，即：每一个抽象类一定要被子类所继承；

57、抽象类的子类（子类不是抽象类）必须要覆写抽象类之中的全部抽象方法（强制子类覆写）；

58、一定要先进行向上转型后才可以进行向下转型

59、子类尽量不要过多的扩充与父类无关的操作

60、抽象类不能使用final定义：因为抽象类必须有子类，而final定义的类不能够有子类

61、外部的抽象类不允许用static声明，而内部的抽象类允许使用static声明，使用static的内部抽象类就相当于一个外部抽象类，继承的时候使用“外部类.内部类”的形式表示类名称。

62、在任何一个类构造执行之前，所有属性的内容都是其对应数据类型的默认值，而子类构造执行之前一定先执行父类构造，那么此时子类构造由于没有执行，所以num就是0。

63、接口只有一种访问权限-public,接口不能继承抽象类。

Java中，接口的主要功能是解决单继承局限问题。

64、接口三大作用：

• 定义不同层之间的操作标准

• 定义一种操作的能力

• 表示将服务器端的远程方法视图暴露给客户端

65、

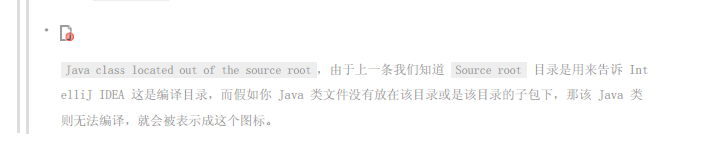
|  |  |  |  |
| --- | --- | --- | --- |
| NO | 区别 | 抽象类 | 接口 |
| 1 | 关键字 | Class | Interface |
| 2 | 组成 | 构造方法、普通方法、抽象方法、static方法、常量、变量 | 抽象方法、全局常量 |
| 3 | 子类使用 | Class子类 | Class子类implements接口，接口，... |
| 4 | 关系 | 抽象类可以实现多个接口 | 接口不能够继承抽象类，却可以继承多个父接口 |
| 5 | 权限 | 可以使用各种权限 | 只能够使用public权限 |
| 6 | 限制 | 单继承权限 | 没有单继承权限 |
| 7 | 子类 | 抽象类和接口都必须有子类，子类子类必须要覆写全部的抽象方法 | |
| 8 | 实例化对象 | 依靠子类对象的向上转型进行对象的实例化 | |

当抽象类和接口都可以使用的时候，优先考虑接口。

**IntelliJ IDEA问题解决**

**调试工具详解：**

[**http://www.cnblogs.com/Bowu/p/4026117.html**](http://www.cnblogs.com/Bowu/p/4026117.html)





**多线程**

实现多线程的方法：  
1. 继承Thread类

2.实现Runnable接口（Callable接口）；

直接调用run()方法，并不能启动多线程，多线程启动的唯一方法就是Thread类中的start()方法：public void start() (调用此方法执行的方法体是run()方法定义的)。

与继承Thread类相比，此时的MyThread类在结构上与之前是没有区别的，但是有一点是有严重区别的，如果继承了Thread类，那么可以直接继承start()方法，但是如果实现的是Runnable 接口并没有start方法可以被继承。

不管何种情况下，如果想启动多线程一定要依靠Thread类完成，在Thread类中定义有以下的构造方法：

● 构造方法：public Thread(Runnable target),接受的是Runnable接口对象；

范例：启动多线程

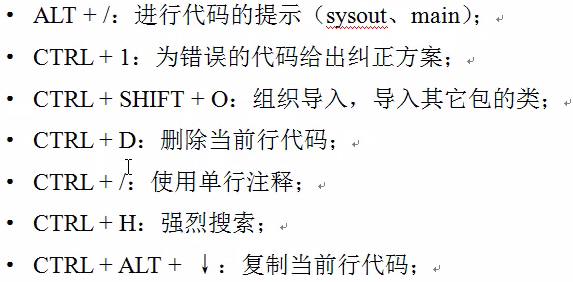
|  |
| --- |
| package xuanmiao.demo1;  class MyThread implements Runnable { //这是一个多线程的操作类  private String name ;  public MyThread(String name) {  this.name = name ;  }  @Override  public void run() { //覆写run方法，作为线程的主题方法  for (int x =0 ; x < 200 ; x ++) {  System.*out*.println(this.name + " --> " + x);  }  } } public class TestDemo {  public static void main(String[] args) {  MyThread mt1 = new MyThread("线程A") ;  MyThread mt2 = new MyThread("线程B") ;  MyThread mt3 = new MyThread("线程C") ;  new Thread(mt1).start();  new Thread(mt2).start();  new Thread(mt3).start();  } } |

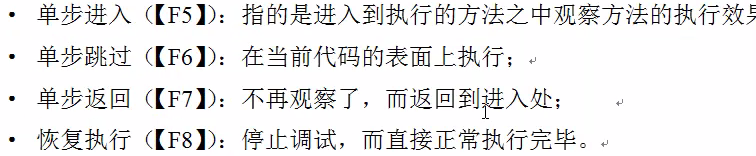
此时避免了单继承局限，那么也就是说

●Thread类是Runnable接口的子类，使用Runnble 实现多线程可以避免单继承局限。

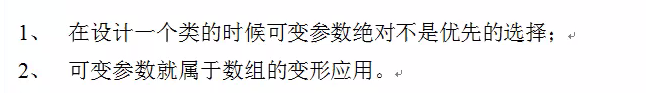
●Runnable 接口实现的多线程可以比Thread类实现的多线程更加清楚的描述数据共享的概念。

eclipse





### ****可变参数的应用：****



**package** xuanmiao.demo;

**public** **class** TestDemo {

**public** **static** **void** main(String[] args) {

System.***out***.println(*add*(1,2,3));

System.***out***.println(*add*(10,20));

}

/\*\*

\* 实现多个整型数据的相加操作

\* **@param** data 由于要接受多个整型数据，所以使用数组完成接受

\* **@return** 多个整型数据的累加结果

\*/

**public** **static** **int** add(**int** ... data){

**int** sum = 0 ;

**for** (**int** x = 0 ; x < data.length ; x ++){

sum += data[x] ;

}

**return** sum;

}

}

### foreach输出



**JDK1.5之前：**

**package** xuanmiao.demo;

**public** **class** ForEach {

**public** **static** **void** main(String[] args) {

**int** data[] = **new** **int**[]{1,2,3,4,5};

**for**(**int** i = 0 ; i < data.length ; i ++){

System.***out***.println(data[i]);

}

}

}

JDK1.5之后：

**package** xuanmiao.demo;

**public** **class** ForEach {

**public** **static** **void** main(String args[]){

**int** data[] = **new** **int**[]{1,2,3,4,5};

**for**(**int** x : data){

System.***out***.println(data[x]);

}

}

}

## 泛型

应用一：

**package** xuanmiao.demo;

**class** Point {

**private** Object x;

**private** Object y;

**public** **void** setX(Object x) {

**this**.x = x;

}

**public** **void** setY(Object y) {

**this**.y = y;

}

**public** Object getX() {

**return** x ;

}

**public** Object getY() {

**return** y;

}

}

**public** **class** generics {

**public** generics() {

// **TODO** Auto-generated constructor stub

}

**public** **static** **void** main(String[] args) {

Point p =**new** Point() ;

p.setX("东经20度");

p.setY("北纬30度");

//第二步：取出数据

String x = (String) p.getX();

String y = (String) p.getY();

System.***out***.println("X坐标：" + x + ",y坐标：" + y);

}

}

应用二：

**package** xuanmiao.demo;

**class** Point<T> {

**private** T x;

**private** T y;

**public** **void** setX(T x) {

**this**.x = x;

}

**public** **void** setY(T y) {

**this**.y = y;

}

**public** T getX() {

**return** x ;

}

**public** T getY() {

**return** y;

}

}

**public** **class** generics {

**public** generics() {

// **TODO** Auto-generated constructor stub

}

**public** **static** **void** main(String[] args) {

Point<Integer> p =**new** Point<Integer>() ;

p.setX(20);

p.setY(10);

//第二步：取出数据

**int** x = (Integer) p.getX();

**int** y = (Integer) p.getY();

System.***out***.println("X坐标：" + x + ",y坐标：" + y);

}

}