# 一、静态导入

1. 静态导入的作用：简化书写

2. 静态导入的格式：

* import static 包名.类名.静态成员
* import static 包名.类名.\*

1. 静态导入可以作用于一个类的所有静态成员。

4. 静态导入要注意的事项：

* 如果静态导入的成员与本类的成员存在同名的情况下，那么默认使用本类的静态成员；如果需要指定使用静态导入的成员，那么需要在静态成员前边加上类名。

5.静态导入演示

|  |
| --- |
| **import** java.util.ArrayList; **import static** java.util.Collections.*sort*; **public class** Dome1 {  **public static void** main(String[] args) {  *sort*(**new** ArrayList<>());  } } |

# 二、增强for循环（foreach）

1、增强for循环的作用：

* 简化迭代器的书写格式（注意：增强for循环的底层还是使用了迭代器遍历的）

2、增强for循环的格式：

**for( 数据类型 变量名：遍历的目标 ){**

**迭代代码……**

**}**

3、增强for循环的作用范围：

* 所有实现了Iterable接口的对象 或者 所有的数组对象 都可以使用增强for循环。

4、 增强for循环需要注意的事项

* 增强for循环的底层也是使用迭代器获取的，只是获取迭代器由jvm完成，不需要我们获取而已，所以使用增强for循遍历元素的过程中，不能使用集合对象对元素个数进行修改。
* 迭代器遍历元素与增强for循环遍历元素的区别：
  + - 使用迭代器遍历元素时，可以删除集合的元素
    - 使用增强for循环遍历元素时，不能调用迭代器的remove方法删除元素
* 普通for循环与增强for循环的区别：
  + - 普通for循环可以没有遍历的目标
    - 增强for循环必须有遍历的目标

注意：Map集合没有实现Iterable接口，所以map集合不能直接使用增强for循环，如果需要使用增强for循环，需要借助与Collections的集合。

# 三、可变参数

## 1、可变参数的格式

**数据类型 … 变量名**

## 2、可变参数要注意的细节

1．如果一个函数在形参上使用了可变参数，那么调用该方法的时候，可以传递参数，也可以不传递参数

2.可变参数 实际上是一个数组

3.可变参数 必须放在形参的最后一个位置

4.可变参数 在一个函数中只能存在一位，因为需要放在参数的最后一个位置上。

## 3.案例

需求：写一个加法函数，进行加法运算的变量是不确定的

|  |
| --- |
| **public static void** add(**double**... element) {  **double** sum = 0;  **for** (**double** d : element) {  sum += d;  }  System.***out***.println(**"和是："** + sum); } |

# 四、自动装箱与拆箱

## 1. 包装类：

* java是面向对象的语言，任何事物都可以使用类进行描述，sun就使用了一些类描述java中八种基本数据类型数据

## 2. 八大基本数据类型对应的包装类

基本数据类型 包装类

byte Byte

short Short

int Integer

long Long

double Double

float Float

char Character

boolean Boolean

## 3. 基本数据类型有了包装类的好处：

3.1在jdk1.5之前，并没有包装类，当需要把引用数据类型当成基本数据类型使用时，需要手动进行转换;jdk1.5之后，有了包装了，基本数据类型和引用数据类型可以进行自动转换。

3.2可以把 字符串转换成数字类型的数据使用，也可以把数字类型数据转换成字符串使用

3.3可以把字符串当成对应的进制数据自动进行转换。

## 4. 自动拆装箱

4.1 自动装箱：*将基本数据类型的值直接赋值给引用数据类型（包装类）的过程，称为自动装箱*

* + *int ——>> Integer*

4.2 自动拆箱：*将引用数据类型（包装类）数据直接赋值给基本数据类型的过程，称为自动拆箱*

* + *Integer ——>> int*

## 5. Integer底层维护的是缓冲数组

Integer 内部维护了缓冲数组，该缓冲数组存储的是-128~127，这些数据在一个数组中。 如果你获取的数据是这个范围内，那么就直接从该缓冲区中获取对应的数据。

## 6.案例

|  |
| --- |
| **public static void** main(String[] args) {  String str1 = **"10"**;  *// 字符串转数字* **int** data1 = Integer.*parseInt*(str1) + 1;  System.***out***.println(**"字符串转数字："** + data1); *// 11   // 把数字转换成字符串* String string1 = Integer.*toString*(data1);  System.***out***.println(**"数字转字符串："** + string1); *// "11"   // 把字符串当成对应进制的数据使用* String str2 = **"10"**;  **int** data2 = Integer.*parseInt*(str2, 2); *// 二进制* System.***out***.println(**"十进制的10转换成二进制："** + data2);*//2  // 自动装箱：把基本数据类型包装成引用数据类型* **int** data3 = 2;  Integer str3 = data3;*//将基本数据类型的值直接赋值给引用数据类型（包装类）的过程，称为自动装箱   // 自动拆箱:把引用数据类型转换成基本数据类型* Integer integer = **new** Integer(2);  **int** data4 = integer;*// 将引用数据类型（包装类）数据直接赋值给基本数据类型的过程，称为自动拆箱* System.***out***.println(**"自动装箱："** + str3 + **" 自动拆箱："** + data4);*//2 2    // Integer 内部维护了缓冲数组，该缓冲数组存储的是-128~127，这些数据在一个数组中。 如果你获取的数据是这个范围内，那么就直接从该缓冲区中获取对应的数据。* Integer data5 = 127;  Integer data6 = 127;  System.***out***.println(**"是同一个对象吗？"** + (data5 == data6));*//true* Integer data7 = 128;  Integer data8 = 128;  System.***out***.println(**"是同一个对象吗？"** + (data7 == data8));*//false* Integer data9 = -128;  Integer data10 = -128;  System.***out***.println(**"是同一个对象吗？"** + (data9 == data10));*//true* Integer data11 = -129;  Integer data12 = -129;  System.***out***.println(**"是同一个对象吗？"** + (data11 == data12));*//false* } |

# 五、枚举

## 1、问题：

* 某些方法所接受的数据必须是在固定范围之内的，怎么解决？
  + 方案一：jdk1.5之前，自定义一个类，私有化构造函数，在自定义类中创建本类的对象对外使用。
  + 方案二：jdk1.5之后，使用枚举类(enum)来解决这种问题。

1.1 解决方案一：自定义类

|  |
| --- |
| ***// 方案一：jdk1.5之前，自定义一个类，私有化构造方法，在自定义类中创建类的对象，对外使用* class** Gender {  String **value**;***//成员变量* public static final** Gender ***man*** = **new** Gender(**"男"**);  **public static final** Gender ***woman*** = **new** Gender(**"女"**);    ***//私有化构造方法，使外部无法构建类的实例* private** Gender(String value) {  **this**.**value** = value;  } } |

1.2 方案二：枚举（enum）

|  |
| --- |
| **enum** Sex {  ***//7.枚举值必须位于枚举类的第一个语句，多个枚举值之间使用逗号(,)分开  //1.枚举值默认的修饰符是private static final  woman***(**"女"**) {  @Override  **public void** run() {  System.***out***.println(**"女人在跑..."**);  }  },   ***//2.枚举值的数据类型就是枚举值所属类的类型，它指向了本类的对象。  //2.枚举值 man("男") 与 private static final Sex man = new Sex("男")；相等  man***(**"男"**) {  @Override  **public void** run() {  System.***out***.println(**"男人在跑..."**);  }  };   String **value**;***// 3.枚举类可以有成员变量* private void** sun() {***//3.枚举类可以有成员函数*** System.***out***.println(**"枚举类的成员函数..."**);  }   ***//4.枚举类可以有构造函数，但是修饰符必须是private，默认也是private* private** Sex(String value) {  **this**.**value** = value;  }   ***// 5.枚举类可以存在抽象方法，但是枚举值必须实现抽象方法* public abstract void** run();  } |

## 2.枚举类

* 一些方法在运行时，他需要的数据不能是任意的，必须是固定范围之内的值，jdk1.5后可以直接使用枚举(enum)解决

2.1 枚举类的格式

**enum 类名{**

**// 枚举值**

**}**

2.2 枚举类需要注意的细节

1.枚举类是一个特殊的类  
2.枚举值默认的修饰符是：public static final  
3.枚举值就是枚举值所属的类的数据类型，然后枚举值是指向了本类的对象的。  
4.枚举类的构造方法的修饰符，默认是private  
5.枚举类可以定义自己的成员变量和成员函数。  
6.枚举类可以自定义构造函数，但是修饰符必须是private  
7.枚举类可以存在抽象的方法，但是枚举值必须实现抽象方法  
8.枚举值必须要位于枚举类的第一个语句。

## 3.在switch中使用枚举类

switch适用的数据类型：byte、short、int、char、String、枚举  
 注意：case语句后边跟的枚举值，只需要单写枚举值即可，不需要再声明该枚举值是属于哪个枚举类的。

3.1 案例

|  |
| --- |
| **public class** Dome4 {  **public static void** main(String[] args) {  Season season = Season.***spring***;***//枚举类的构造方法默认是private，不能在外部创建实例* switch** (season) {  **case *spring***:***//当option是枚举类的对象时，case选项不需要使用类名.属性，直接使用属性即可。*** System.***out***.println(**"春天来了...."**);  **break**;  **case *summer***:  System.***out***.println(**"夏天来了...."**);  **break**;  **case *autumn***:  System.***out***.println(**"秋天来了...."**);  **break**;  **case *winter***:  System.***out***.println(**"冬天来了...."**);  **break**;  }  } } ***// 季节枚举类* enum** Season {  ***spring***, ***summer***, ***autumn***, ***winter // 枚举值*** } |