**泛型**

JDk5.0

泛型 模板编程

不同泛型之间不能相互赋值

泛型的通配符：

？ 任何类型

？ extends Number Number类或者Number的任何子类

？ super Number Number类或者Number的任何父类

？ extends Comparable 任何实现Comparable接口的类

泛型方法：

<T extends Number&Comparable> T既是Number的子类，又实现Comparable接口

泛型类：

泛型技术应用在集合框架中：实现类型安全的集合

泛型：约定集合中的元素类型

静态引入：用到一个类中静态成员时，可以省略类名

package chp10;

import java.util.\*;

import static java.lang.System.\*;

import static java.lang.Math.\*;

public class TestStaticImport {

public static void main(String[] args) {

int[] a={1,4,3,7,2};

java.util.Arrays.sort(a);

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

System.out.println(a[i]);

}

System.out.println(Math.random());

//可以写成 out.println(random());

}

}

可变长参数：由编译器负责将多个实参封装成数组

一个方法中，最多只能有一个边长参数，且必须是参数表中最后一个参数

package chp10;

public class TestVarArgs {

public static void main(String[] args) {

method();

method("Zhang3","Li4");//相当于数组

method("Zhang3","Li4","Wang2");

}

static void method(String... str){

System.out.println(str.length);

}

}

格式化输入输出：

枚举：枚举是类，这个类的对象时构造好的，不允许用户构造该类的新对象

特殊类：默认是Enum的子类 final类

所有构造方法必须是私有的

枚举值：枚举类的对象，也是枚举类中的公开静态常量

枚举中可以定义抽象方法，由枚举值分别实现

package Chp10;

public class TestEnum {

public static void main(String[] args) {

Season s1=Season.Autumn;

//Season1 s2=Season1.AUTUMN;

//System.out.println(s2.name());

//System.out.println(s2.ordinal());

Season1[] s2=Season1.values();

for(Season1 s:s2){

System.out.println(s.ordinal()+" "+s.name());

}

Operator[] os=Operator.values();

for(Operator s:os){

System.out.println(s.operator(10,2));

}

}

}

class Season{

public static final Season Spring=new Season();

public static final Season Summer=new Season();

public static final Season Autumn=new Season();

public static final Season Winter=new Season();

private Season(){}

}

enum Season1{

SPRING,

SUMMER,

AUTUMN,

WINTER;

}//和类Season等同

enum Operator{

ADD{

public double operator(double a,double b){

return a+b;

}

},

SUBSTRACT{

public double operator(double a,double b){

return a-b;

}

},

MULTIPLY{

public double operator(double a,double b){

return a\*b;

}

},

DIVIDE{

public double operator(double a,double b){

return a/b;

}

};

public abstract double operator(double a,double b);

}