**接口**

接口：interface

接口中的属性都是公开静态常量，所有方法都是公开抽象方法

示例：

interface MyInterface{

public static int VALUE1=100;

public static int VALUE2=200;

public abstract void m1();

public abstract void m2(int n);

}

一个.java文件中可以有多个接口，但是最多只能有一个公开接口，并且公开的接口的接口名与文件名相同

接口也可以被子类继承，关键字implements

需要注意的是：

1，一个类如果实现接口，如果这个类不想成为抽象类，那么久必须实现接口中的所有方法

2，接口中的所有方法都是公开的，因此在实现接口的方法时，实现类的方法也必需是公开的，由于类的方法中，默认的修饰符是default，所以在实现接口的方法中，public修饰符不能少

接口和接口之间也能继承，关键字也为extends

示例：

interface IA{

void ma();

}

interface IB extends IA{

void mb();

}

接口之间可以多继承

如：interface IC extends IA，IB{

void mc();

}

如果有一个类实现IC的话，就必需实现ma，mb，mc三个方法

一个类可以继承一个类并且实现多个接口

接口作用：

1，用来实现多继承

2，用来解耦合（作用很大）

**内部类**

内部类定义：类中定义类

内部类类型：

1，成员内部类

可以访问外部类的私有成员

创建成员内部类对象，必须先创建外部类对象，在通过成员外部类对象.new构造成员内部类对象

成员内部类中不能有静态成员

外部类类名.this指向外部类的当前对象.

2，静态内部类

静态内部类只能访问外部类的静态成员

创建静态内部类对象，不需要先创建外部类对象

3，局部内部类

适用范围等同于局部变量

不仅可以访问外部类的私有成员，还可以访问外部类的局部变量，但是必需声明为final

4，匿名内部类

特殊的局部内部类

继承一个类或者实现一个接口

只会创建该类的一个对象

匿名内部类不能定义构造方法

内部类可以访问外部类的私有成员而不破坏封装

接口公开，接口的实现类作为内部类隐藏起来，强制弱耦合

**Object**

Object是所有类的父类 类继承关系树的树根

Object o； o中可以装任何对象

Object中的方法是所有对象都具有的方法

1. finalize():由垃圾收集器在回收垃圾对象时，最后调用的方法
2. 垃圾回收器如何认定垃圾对象，零引用算法
3. 垃圾回收器何时回收垃圾对象，必须释放的时候
4. getClass（）：返回对象的实际类型
5. toString（）：返回对象的String类型，打印一个对象就是打印这个对象的toString方法的返回值

toString返回 类名@地址

1. equals（）：boolean equals（Object o）：判断this和o内容是否相同

**包装类**

为8种简单类型提供对象属性，让Object统一所有数据

对于数值类型的属性，区分0和null

包装类 简单类型 String类型之间的转换

JDK5.0 自动分箱

由编译器自动完成简单类型和包装类之间的转换

**集合框架**

集合：一个对象，用来储存其他多个对象

Collection

特点：元素是Object（对象）

方法：

add(Object o):把对象o放入集合

addAll（Collection c）:把c中的所有对象放入集合

clear（）：清空集合

contains（Object o）：判断对象o在集合中是否存在

isEmpty（）：判断集合是否为空

remove（Object o）：把对象o从集合中删去

size（）：返回集合中对象的个数

遍历：

实现类：无

List

特点：元素有顺序（下标），可重复

方法：add(int pos,Object o):把元素o插入到pos位置

get(int pos):返回pos位置的元素

indexOf（Object o）：返回对象o在数组中的首下标

indexOf（Object o）：返回对象o在数组中的末下标

remove（int pos）：删除pos位置的元素

set（int pos，Object o）：把pos位置的元素替换为o

subList(int start,int end) 获得start到end的集合（[a,b)//a能取到，b取不到）

遍历：1，for 下标遍历

2，迭代遍历

实现类：

ArrayList：用数组实现list 查询快 增删慢

1.0 轻量级 线程不安全

Vector：用数组实现

1.2 重量级 线程安全

LinkedList：用链表实现 查询慢 增删快

set

特点：元素无顺序，不可重复

方法：全部继承自Collection接口

遍历：迭代遍历

Iterator it=new Iterator(){

if(it.hasNext)

Object o=it.Next;

System.out.println(o);

}

实现类：

HashSet 把自定义的对象放入HashSet，为保证元素不重复，必须：

1. 覆盖HashCode（），保证相同对象返回相同的Hash码（int）

为了提高效率，尽量保持不同对象返回不同的Hash码（int）

1. 覆盖equals（），保证相同的对象返回true

TreeSet

实现了SortedSet（Set的子接口）对元素自动排序

要求元素对象必须实现Comparable接口

Map

特点：元素由键值对组成 其中键 唯一 值 可以重复

方法：

put（Object k，Object v）把k-v这个键值对放入Map

如果k已经存在，那么新的v会替换旧的v

get（Object k）：返回Map中k对应的v

remove（Object k）：删除k所在的键值对

size（）：返回键值对的数量

containsValue（Object v）：判断v这个值是否存在

遍历：

1. 值遍历：values（）：返回Map中所有值的集合：Collection
2. 键遍历：keySet（）：返回Map中所有键的集合：Set
3. 键值对遍历：entrySet（）：返回Map中所有键值对的集合

键值对：Map.Entry对象 封装了一个key和一个value对象

实现类：

1. HashMap 1.2 允许用null作为key或value
2. HashTable 1.0 不允许用null作为key或value
3. Properties ：Hashtable的子类 key和value都是String，一般用于配置文件的读取
4. TreeMap：实现了SortedMap（Map的子接口） 自动对key排序

package chp10;

import java.util.\*;

public class TestMap {

public static void main(String[] args){

//创建map对象

Map map=new HashMap();

//添加键值对

map.put("Italy", "Germany");

map.put("America", "Brazil");

map.put("France","France");

map.put("Korea", "Brazil");

map.put("Germany", "Italy");

String s=(String) map.get("America");//返回对应值（value）

System.out.println(s);//打印对应值

System.out.println("美国是否举行过世界杯"+map.containsKey("America"));//判断是否存在键

System.out.println("德国是否获得过世界杯冠军"+map.containsValue("Germany"));//判断是否存在值

printValues(map);

printKeys(map);

printKeysValues(map);

}

//值遍历

static void printValues(Map map){

Collection vs=map.values();//返回所有值的集合

Iterator it=vs.iterator();

while(it.hasNext()){

Object v=it.next();

System.out.println(v);

}

}

//键（值）遍历

static void printKeys(Map map){

Set ks=map.keySet();//键为Set类型

Iterator it=ks.iterator();

while(it.hasNext()){

Object k=it.next();

Object v=map.get(k);//调用get方法得到值

System.out.println(k+"-----"+v);

}

}

static void printKeysValues(Map map){

Set es=map.entrySet();

Iterator it=es.iterator();

while(it.hasNext()){

Map.Entry me=(Map.Entry)it.next();

Object k=me.getKey();

Object v=me.getValue();

System.out.println(k+"-----"+v);

}

}

}