## 第一章

### 练习二：Object类的toString方法

1. 不运行代码，直接说出打印结果，并解释原因。

**public class** ToStringTest{  
 **static int** i = 1;  
 **public static void** main(String args[]){  
 System.**out**.println(**"love "** + **new** ToStringTest());//love java  
 ToStringTest a = **new** ToStringTest();  
 a.i++;  
 System.**out**.println(**"me "** + a.i);//me 2  
 }  
 **public** String toString(){  
 System.**out**.print(**"I "**);//I  
 **return "java "**;  
 }  
}

运行结果：I love java me 2

原因：当执行代码的时候，首先加载静态变量，然后执行main方法，由于main方法内部第一行代码为输出语句，里面new了此类对象，当执行此行代码时会先创建了本类的对象，由于此类重写了toString方法，会先执行toString方法的打印输出，然后返回“java ”，再执行main方法第一行打印输出。在Java中“System.out.println(类对象名);”实际输出的是该对象的toString()方法返回的字符串，即括号中的内容等价于类对象名.toString(),toString方法的好处是在碰到println方法的时候会被自动调用，不用显示的写出来。

### 练习三：Object类equals方法

1. 看下列程序，不运行说结果，写出答案后，并在IntelliJ IDEA中运行看看自己给的答案与运行结果是否正确，并分析原因。

(1)

String s1 = **new** String(**"abc"**);  
 String s2 = **"abc"**;  
 System.**out**.println(s1 == s2); //false  
 System.**out**.println(s1.equals(s2)); //true

(2)

String s1 = **"abc"**;  
 String s2 = **"abc"**;  
 System.**out**.println(s1 == s2); //true  
 System.**out**.println(s1.equals(s2)); //true

(3)

String s1 = **"a"** + **"b"** + **"c"**;  
 String s2 = **"abc"**;  
 System.**out**.println(s1 == s2); //true  
 System.**out**.println(s1.equals(s2)); //true

(4)

String s1 = **"ab"**;  
 String s2 = **"abc"**;  
 String s3 = s1 + **"c"**;  
 System.**out**.println(s3 == s2); //false  
 System.**out**.println(s3.equals(s2)); //true

## 第二章

### 练习二：Collection集合统计元素出现次数

1. 给定以下代码，请定义方法listTest()统计集合中指定元素出现的次数，如"a": 2,"b": 2,"c" :1, "xxx":0。

Collection<String> list = new ArrayList<>();

list.add("a");

list.add("a");

list.add("b");

list.add("b");

list.add("c");

System.out.println("a:"+listTest(list, "a"));

System.out.println("b:"+listTest(list, "b"));

System.out.println("c:"+listTest(list, "c"));

System.out.println("xxx:"+listTest(list, "xxx"));

**public class** CollectionTest01{  
 **public static void** main(String[] args) {  
 Collection<String> list = **new** ArrayList<>();  
 list.add(**"a"**);  
 list.add(**"a"**);  
 list.add(**"b"**);  
 list.add(**"b"**);  
 list.add(**"c"**);  
 System.**out**.println(**"a:"**+listTest(list, **"a"**));  
 System.**out**.println(**"b:"**+listTest(list, **"b"**));  
 System.**out**.println(**"c:"**+listTest(list, **"c"**));  
 System.**out**.println(**"xxx:"**+listTest(list, **"xxx"**));  
 }  
  
 //定义方法统计集合中指定元素出现的次数  
 **public static int** listTest(Collection<String> list,String s){  
 //定义计数器，初始化为0  
 **int** count = 0;  
 //增强for遍历集合  
 **for** (String string : list) {  
 //判断传入方法的字符与遍历集合的是否一致  
 **if** (s.equals(string)) {  
 //如果一致，加1  
 count++;  
 }  
 }  
 **return** count;  
 }  
}

### 练习三：Collection集合数组转集合

1. 定义一个方法，要求此方法把int数组转成存有相同元素的集合(集合里面的元素是Integer)，并返回。()

**public class** CollectionTest02 {  
 **public static void** main(String[] args) {  
 //定义int数组  
 **int**[] arr = {1,2,3,4,5};  
 ArrayList<Integer> list = listTest(arr);  
 System.**out**.println(list);  
 }  
  
 **public static** ArrayList<Integer> listTest(**int**[] arr) {  
 //定义集合  
 ArrayList<Integer> list = **new** ArrayList<Integer>();  
 //遍历数组，把元素依次添加到集合当中  
 **for** (**int** a : arr) {  
 list.add(a);  
 }  
 **return** list;  
 }  
}

### 练习四：Collection集合集合转数组

1. 定义一个集合，并把集合(集合里面的元素是Integer)转成存有相同元素的数组，并将结果输出在控制台。（可以使用Object[]数组类型接收转换的数组）

**public class** CollectionTest03 {  
 **public static void** main(String[] args) {  
 //定义集合,添加数据  
 ArrayList<Integer> list = **new** ArrayList<Integer>();  
 list.add(100);  
 list.add(200);  
 list.add(300);  
 //Object[] toArray()转换成一个Object数组  
 Object[] obj = list.toArray();  
 // 遍历数组  
 **for** (**int** i = 0; i < obj.**length**; i++) {  
 System.**out**.println(obj[i]);  
 }  
 }  
}

### 练习五：Collection集合contains()方法使用

1. 定义一个方法listTest(ArrayList<String> al, String s),要求使用contains()方法判断al集合里面是否包含s。

**public class** CollectionTest04 {  
 **public static void** main(String[] args) {  
 //定义集合，添加数据  
 ArrayList<String> list = **new** ArrayList<String>();  
 list.add(**"itcast"**);  
 list.add(**"itheima"**);  
 list.add(**"java"**);  
 System.**out**.println(listTest(list,**"java"**));  
 }  
  
 **public static boolean** listTest(ArrayList<String> al, String s) {  
 //判断s是否在集合中存在,存在返回true，不存在返回false  
 **if** (al.contains(s)) {  
 **return true**;  
 }  
 **return false**;  
 }  
}

### 练习六：Collection集合isEmpty()方法的使用

1. 定义一个方法listTest(ArrayList<String> al), 要求使用isEmpty()判断al里面是否有元素。

**public class** CollectionTest05 {  
 **public static void** main(String[] args) {  
 //定义集合，添加数据  
 ArrayList<String> list = **new** ArrayList<String>();  
 list.add(**"1"**);  
 System.**out**.println(listTest(list));  
 }  
  
 **public static boolean** listTest(ArrayList<String> al) {  
 //判断al集合是否为空,为空返回true，不为空返回false  
 **if**(al.isEmpty()){  
 **return true**;  
 }  
 **return false**;  
 }  
}

### 练习八：Collection集合返回首次出现索引

1. 定义一个方法listTest(ArrayList<Integer> al, Integer s)，要求返回s在al里面第一次出现的索引，如果s没出现过返回-1。

**public class** CollectionTest06 {  
 **public static void** main(String[] args) {  
 //定义集合，添加数据  
 ArrayList<Integer> list = **new** ArrayList<Integer>();  
 list.add(1);  
 list.add(2);  
 list.add(3);  
 list.add(4);  
 list.add(5);  
 System.**out**.println(listTest(list, 5));  
 }  
  
 **public static int** listTest(ArrayList<Integer> al, Integer s) {  
 //遍历集合，获取元素，判断元素是否与s相等，相等返回索引  
 **for** (**int** i = 0; i < al.size(); i++) {  
 **if** (al.get(i).equals(s)) {  
 **return** i;  
 }  
 }  
 **return** -1;  
 }  
}

### 练习四：Comparable和Comparator比较器

1. 简述Comparable和Comparator两个接口的区别。

* Comparable：强行对实现它的每个类的对象进行整体排序。这种排序被称为类的自然排序，类的compareTo方法被称为它的自然比较方法。只能在类中实现compareTo()一次，不能经常修改类的代码实现自己想要的排序。实现此接口的对象列表（和数组）可以通过Collections.sort（和Arrays.sort）进行自动排序，对象可以用作有序映射中的键或有序集合中的元素，无需指定比较器。
* Comparator强行对某个对象进行整体排序。可以将Comparator 传递给sort方法（如Collections.sort或 Arrays.sort），从而允许在排序顺序上实现精确控制。还可以使用Comparator来控制某些数据结构（如有序set或有序映射）的顺序，或者为那些没有自然顺序的对象collection提供排序。

### 练习五：LinkedList方法的使用

1. 根据要求练习LinkedList方法：

（1）基本方法：add, set, get, remove, clear, size等方法；

（2）特有方法：addFirst, addLast, getFirst, getLast, removeFirst, removeLast, push, pop, clear等方法。

（1）基本方法：

**public class** LinkedListTest01 {  
 **public static void** main(String[] args) {  
 // 1.创建LinkedList  
 LinkedList<String> arr = **new** LinkedList<String>();  
  
 // 2.使用add方法添加元素  
 arr.add(**"西门吹雪"**);  
 arr.add(**"西门吹雪"**);  
 arr.add(**"西门吹雪"**);  
 arr.add(**"西门吹风"**);  
 arr.add(**"西门吹水"**);  
  
 // 3.使用add方法在指定索引添加元素  
 arr.add(2, **"西门吹雨"**);  
  
 // 4.使用set方法修改指定位置索引  
 arr.set(0, **"东门"**);  
  
 **for** (String str : arr) {  
 System.**out**.println(str);  
 }  
 System.**out**.println(**"--------------"**);  
 // 5.使用get方法获取指定索引的元素  
 System.**out**.println(arr.get(1));  
  
 // 6.使用size方法获取集合大小  
 System.**out**.println(arr.size());  
  
 // 7.使用remove方法删除指定索引的元素  
 arr.remove(3);  
  
 // 8.使用clear清空集合中的元素  
 arr.clear();  
 System.**out**.println(arr);  
 }  
}

东门

西门吹雪

西门吹雨

西门吹雪

西门吹风

西门吹水

--------------

西门吹雪

6

[]

（2）特有方法

**public class** LinkedListTest02 {  
 **public static void** main(String[] args) {  
 // 1.创建LinkedList  
 LinkedList<String> linked = **new** LinkedList<String>();  
  
 // 2.使用add方法添加元素  
 linked.add(**"周杰伦"**);  
 linked.add(**"周星驰"**);  
 linked.add(**"周华健"**);  
 linked.add(**"周润发"**);  
  
 // 3.使用addFirst添加元素到集合最前面  
 linked.addFirst(**"周传雄"**);  
  
 // 4.使用addLast添加元素到集合最后面  
 linked.addLast(**"周渝民"**);  
  
 System.**out**.println(linked);  
  
 // 5.使用getFirst获取集合第一个元素  
 System.**out**.println(linked.getFirst());  
  
 // 6.使用getLast获取集合最后一个元素  
 System.**out**.println(linked.getLast());  
  
 // 7.使用removeLast删除集合第一个元素  
 String first = linked.removeFirst();  
 System.**out**.println(first);  
  
 // 8.使用removeLast删除集合最后一个元素  
 String last = linked.removeLast();  
 System.**out**.println(last);  
 System.**out**.println(linked);  
  
  
 // 9.使用pop弹出第一个元素  
 String p = linked.pop();  
 System.**out**.println(p);  
  
 // 10.使用push在集合开头插入元素  
 linked.push(**"周立波"**);  
 System.**out**.println(linked);  
  
 // 11.使用clear清空集合  
 linked.clear();  
 System.**out**.println(linked);  
 }  
}

[周传雄, 周杰伦, 周星驰, 周华健, 周润发, 周渝民]

周传雄

周渝民

周传雄

周渝民

[周杰伦, 周星驰, 周华健, 周润发]

周杰伦

[周立波, 周星驰, 周华健, 周润发]

[]

### 练习六：HashSet存储自定义类型

1. 定义人类，包含姓名和年龄属性。创建4个人存储到HashSet中，姓名和年龄相同的人看做同一人不存储。

Person类:

// 1.定义Person类.包好姓名年龄属性,重写hashCode()和equals()方法  
**public class** Person {  
 **private** String **name**;  
 **private int age**;  
  
 **public** Person() {  
 }  
  
 **public** Person(String name, **int** age) {  
 **this**.**name** = name;  
 **this**.**age** = age;  
 }  
  
 @Override  
 **public boolean** equals(Object o) {  
 **if** (**this** == o) **return true**;  
 **if** (!(o **instanceof** Person)) **return false**;  
  
 Person person = (Person) o;  
  
 **if** (**age** != person.**age**) **return false**;  
 **return name** != **null** ? **name**.equals(person.**name**) : person.**name** == **null**;  
 }  
  
 @Override  
 **public int** hashCode() {  
 **int** result = **name** != **null** ? **name**.hashCode() : 0;  
 result = 31 \* result + **age**;  
 **return** result;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Person{"** +  
 **"name='"** + **name** + **'\''** +  
 **", age="** + **age** +  
 **'}'**;  
 }  
}

测试类

**public class** HashSetTest01 {  
 **public static void** main(String[] args) {  
 // 2.创建HashSet用于存储Person类型  
 HashSet<Person> hashSet = **new** HashSet<Person>();  
  
 // 3.添加多个Person到HashSet中  
 hashSet.add(**new** Person(**"王昭君"**, 21));  
 hashSet.add(**new** Person(**"西施"**, 21));  
 hashSet.add(**new** Person(**"杨玉环"**, 20));  
 hashSet.add(**new** Person(**"貂蝉"**, 19));  
 hashSet.add(**new** Person(**"杨玉环"**, 20));  
 hashSet.add(**new** Person(**"貂蝉"**, 19));  
  
 // 4.遍历获取HashSet中的内容  
 **for** (Person p : hashSet) {  
 System.**out**.println(p);  
 }  
 }  
}

### 练习八：LinkedHashSet基本使用

1. 使用LinkedHashSet存储以下元素："王昭君","王昭君","西施","杨玉环","貂蝉"。使用迭代器和增强for循环遍历LinkedHashSet。

**public class** LinkedHashSetTest01 {  
 **public static void** main(String[] args) {  
 // 1.创建LinkedHashSet  
 LinkedHashSet<String> lhSet = **new** LinkedHashSet<String>();  
  
 // 2.使用add方法添加元素到LinkedHashSet  
 lhSet.add(**"王昭君"**);  
 lhSet.add(**"王昭君"**);  
 lhSet.add(**"王昭君"**);  
 lhSet.add(**"西施"**);  
 lhSet.add(**"杨玉环"**);  
 lhSet.add(**"貂蝉"**);  
  
 // 3.使用迭代器获取LinkedHashSet中的元素  
 Iterator<String> iterator = lhSet.iterator();  
 **while** (iterator.hasNext()) {  
 System.**out**.println(iterator.next());  
 }  
  
 // 4.使用增强for获取LinkedHashSet中的元素  
 System.**out**.println(**"---------------------"**);  
 **for** (String string : lhSet) {  
 System.**out**.println(string);  
 }  
 }  
}

王昭君

西施

杨玉环

貂蝉

---------------------

王昭君

西施

杨玉环

貂蝉

### 练习三：Map接口中的常用方法

1. 请使用Map集合的方法完成添加元素，根据键删除，以及根据键获取值操作。

**public class MapTest01**{  
**public static void** main(String[] args) {  
 // 1.创建HashMap  
 HashMap<String, String> hm = **new** HashMap<String, String>();  
  
 // 2.使用put添加元素  
 hm.put(**"黄晓明"**, **"Baby"**);  
 hm.put(**"邓超"**, **"孙俪"**);  
 hm.put(**"李晨"**, **"范冰冰"**);  
 hm.put(**"大黑牛"**, **"范冰冰"**);  
  
 // 3.使用put修改元素  
 String v1 = hm.put(**"李晨"**, **"白百合"**);  
  
 // 4.使用get获取元素  
 String string = hm.get(**"大黑牛"**);  
  
 // 5.使用remove删除元素  
 String v2 = hm.remove(**"大黑牛"**);  
 System.**out**.println(v2);  
  
 // 6.打印集合中的元素  
 System.**out**.println(hm);  
 }  
}

V1:范冰冰

String:范冰冰

V2:范冰冰

Hm:{邓超=孙俪, 李晨=白百合, 黄晓明=Baby}

### 练习四：Map接口中的方法

1. 往一个Map集合中添加若干元素。获取Map中的所有value，并使用增强for和迭代器遍历输出每个value。

**public class** MapTest02 {  
**public static void** main(String[] args) {  
// 1.创建HashMap  
HashMap<String, String> hm = **new** HashMap<String, String>();  
  
// 2.使用put添加元素  
hm.put(**"黄晓明"**, **"Baby"**);  
 hm.put(**"邓超"**, **"孙俪"**);  
 hm.put(**"李晨"**, **"范冰冰"**);  
 hm.put(**"大黑牛"**, **"范冰冰"**);  
  
// 3.使用Map的values方法获取到所有的value  
Collection<String> values = hm.values();  
  
// 4.使用增强for获取每个value  
**for** (String value : values) {  
 System.**out**.println(value);  
 }  
  
 System.**out**.println(**"----------------"**);  
// 5.使用迭代器获取每个value  
Iterator<String> itr = values.iterator();  
**while** (itr.hasNext()) {  
 System.**out**.println(itr.next());  
 }  
 }  
}

孙俪

范冰冰

范冰冰

Baby

----------------

孙俪

范冰冰

范冰冰

Baby

### 练习五：HashMap存储键是自定义对象值是String

1. 请使用Map集合存储自定义数据类型Car做键，对应的价格做值。并使用keySet和entrySet两种方式遍历Map集合。

**public class** MapTest03 {  
**public static void** main(String[] args) {  
// 2.创建HashMapkey保存汽车对象,value是汽车价格  
HashMap<Car, Integer> hm = **new** HashMap<>();  
  
// 3.添加汽车到HashMap中  
Car c1 = **new** Car(**"长安奔奔"**, **"黄色"**);  
 Car c3 = **new** Car(**"奇瑞QQ"**, **"黑色"**);  
 Car c2 = **new** Car(**"铃木奥拓"**, **"白色"**);  
  
 hm.put(c1, 10000);  
 hm.put(c2, 20000);  
 hm.put(c3, 30000);  
  
// 4.使用keySet方式遍历Map  
Set<Car> keySet = hm.keySet();  
**for** (Car c : keySet) {  
// 根据key获取value  
Integer value = hm.get(c);  
 System.**out**.println(c.getName() + **","**+ c.getPrice() + **" - "**+ value);  
 }  
  
 System.**out**.println(**"-------------"**);  
  
// 5.使用entrySet方式遍历Map  
Set<Map.Entry<Car, Integer>> entrySet = hm.entrySet();  
**for** (Map.Entry<Car, Integer> entry : entrySet) {  
 Car key = entry.getKey();  
 Integer value = entry.getValue();  
 System.**out**.println(key.getName() + **","**+ key.getPrice() + **" - "**+ value);  
 }  
 }  
}

## 第三章

# 基础题

## 练习一:相对路径和绝对路径的使用

描述:创建两个文件对象，分别使用相对路径和绝对路径创建。

答案

操作步骤:

绝对路径创建文件对象：使用File类一个参数的构造方法。

相对路径创建文件对象：使用File类两个参数的构造方法。

代码:

**public class** Test01\_01 {

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

// 创建文件对象：绝对路径

File f1 = **new** File("d:/aaa/a.txt");

// 创建文件对象：相对路径

File f2 = **new** File("a.txt");

}

}

## 练习二:检查文件是否存在,文件的创建

描述:检查D盘下是否存在文件a.txt,如果不存在则创建该文件。

答案

操作步骤:

1. 使用绝对路径创建对象关联到D盘的a.txt。
2. 通过文件对象方法判断文件是否存在。
3. 不存在则调用创建文件的方法创建文件。

代码:

**public class** Test01\_02 {

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

// 创建文件对象：绝对路径

File f = **new** File("d:/a.txt");

// 如果文件不存在，则创建文件

**if**(!f.exists()) {

f.createNewFile();

}

}

}

## 练习三:单级文件夹的创建

描述:在D盘下创建一个名为bbb的文件夹。

答案

操作步骤:

1. 创建文件对象指定路径为d:/bbb
2. 调用文件对象创建文件夹的方法

代码:

**public class** Test01\_03 {

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

// 创建文件对象

File f = **new** File("d:/bbb");

// 创建单级文件夹

f.mkdir();

}

}

## 练习四:多级文件夹的创建

描述:在D盘下创建一个名为ccc的文件夹，要求如下：

1.ccc文件夹中要求包含bbb子文件夹

2.bbb子文件夹要求包含aaa文件夹

答案:

操作步骤:

1. 创建文件对象关联路径：d:/ccc/bbb/aaa
2. 调用文件对象创建多级文件夹的方法

代码:

**public class** Test01\_04 {

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

// 创建文件对象

File f = **new** File("d:/ccc/bbb/aaa");

// 创建多级文件夹

f.mkdirs();

}

}

## 练习五:删除文件和文件夹

描述:

将D盘下a.txt文件删除

将D盘下aaa文件夹删除,要求文件夹aaa是一个空文件夹。

答案:

操作步骤:

1. 创建文件对象关联路径：d:/a.txt
2. 调用文件对象删除文件的方法
3. 创建文件对象关联路径：d:/aaa
4. 调用文件对象删除文件夹的方法

代码:

**public class** Test01\_05 {

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

// 创建文件对象

File f = **new** File("d:/a.txt");

// 删除文件

f.delete();

// 创建文件夹对象

File dir = **new** File("d:/aaa");

// 删除文件夹

dir.delete();

}

}

## 练习六:获取文件信息:文件名,文件大小,文件的绝对路径,文件的父路径

描述:

获取D盘aaa文件夹中b.txt文件的文件名，文件大小，文件的绝对路径和父路径等信息，并将信息输出在控制台。

答案:

操作步骤:

1. 在D盘aaa文件夹中创建一个b.txt文件并输入数据
2. 创建文件对象关联路径：d:/aaa/b.txt
3. 调用文件对象的相关方法获得信息并输出。可以通过API帮助文档查询方法。

代码:

**public class** Test01\_06 {

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

// 创建文件对象

File f = **new** File("d:/aaa/b.txt");

// 获得文件名

String filename = f.getName();

// 获得文件大小

**long**filesize = f.length();

// 获得文件的绝对路径

String path = f.getAbsolutePath();

// 获得父文件夹路径，返回字符串

String parentPath = f.getParent();

// 获得父文件夹路径，返回文件对象

File parentFile = f.getParentFile();

// 输出信息

System.***out***.println("文件名：" + filename);

System.***out***.println("文件大小：" + filesize);

System.***out***.println("文件路径：" + path);

System.***out***.println("文件父路径：" + parentPath);

System.***out***.println("文件父路径：" + parentFile);

}

}

## 练习七:文件夹或文件的判断

描述:

1.判断File对象是否是文件,是文件则输出：xxx是一个文件，否则输出：xxx不是一个文件。

2.判断File对象是否是文件夹,是文件夹则输出：xxx是一个文件夹，否则输出：xxx不是一个文件夹。(xxx是文件名或文件夹名)

答案:

操作步骤:

1. 创建两个文件对象分别关联到不同的文件，比如：d:/a.txt，d:/aaa
2. 调用文件对象的判断是否是文件或是否是文件夹的方法
3. 获得文件名，根据判断结果输出信息。

代码:

**public class** Test01\_07 {

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

// 创建文件对象

File f1 = **new** File("d:/b.txt");

// 判断是否是一个文件

**if**(f1.isFile()) {

System.***out***.println(f1.getName()+"是一个文件");

} **else** {

System.***out***.println(f1.getName()+"不是一个文件");

}

// 创建文件对象

File f2 = **new** File("d:/aaaa");

// 判断是否是一个文件夹

**if**(f2.isDirectory()) {

System.***out***.println(f2.getName()+"是一个文件夹");

} **else** {

System.***out***.println(f2.getName()+"不是一个文件夹");

}

}

}

## 练习八:文件夹的获取方法

描述:

获取指定文件夹下所有的文件，并将所有文件的名字输出到控制台。

注意：不包含子文件夹下的文件

答案

操作步骤:

1. 创建文件对象关联到指定文件夹，比如：c:/aaa
2. 调用文件对象的listFiles方法获得文件数组
3. 遍历文件数组将每一个文件的名字输出到控制台

代码:

**public class** Test01\_08 {

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

// 创建文件对象

File f = **new** File("d:/aaa");

// 获得文件夹下所有文件

File[] files = f.listFiles();

// 遍历文件数组

**for** (File file :files) {

// 将文件的名字打印到控制台

System.***out***.println(file.getName());

}

}

## 练习六:字节流复制文件

描述:利用字节流将E盘下的a.png图片复制到D盘下(文件名保存一致)

要求：

一次读写一个字节的方式

答案

操作步骤:

1. 创建字节输入流对象关联文件路径：E盘下的a.png
2. 创建字节输出流对象关联文件路径：D盘下的a.png
3. 使用循环不断从字节输入流读取一个字节，每读取一个字节就利用输出流写出一个字节。
4. 关闭流，释放资源

代码:

**public class** Test01\_06 {

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

// 创建字节输入流对象并关联文件

FileInputStream fis = **new** FileInputStream("e:/a.png");

// 创建字节输出流对象并关联文件

FileOutputStream fos = **new** FileOutputStream("d:/a.png");

// 定义变量接收读取的字节数

**int** len = -1;

// 循环读取图片数据

**while**((len = fis.read()) != -1) {

// 每读取一个字节的数据就写出到目标文件中

fos.write(len);

}

// 关闭流

fis.close();

fos.close();

}

}

## 练习七:字符输出流写出字符数据

项目需求：请用户从控制台输入信息，程序将信息存储到文件Info.txt中。可以输入多条信息，每条信息存储一行。当用户输入：”886”时，程序结束。

答案

操作步骤:

1. 创建MainAPP类,并包含main()方法
2. 按照上述要求实现程序

代码:

p**ublic class** Test01\_07 {

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

//1. 指定输出流， 对应的文件Info.txt

FileWriter bw**= new** FileWriter("Info.txt");

//2.采用循环的方式，把每条信息存储一行到Info.txt中

Scanner sc= **new** Scanner(System.in);

while(true){

//获取键盘输入的一行内容

System.out.print("请输入内容：");

String str= sc.nextLine();

//当用户输入：”886”时，程序结束。

if ("886".equals(str)) {

break;//跳出循环

}

//把内容写入到Info.txt文件中

bw.write(str);

//换行

bw.write(System.lineSeparator());

}

//关闭流

bw.close();

}

}

## 练习一:高效字节输出流写出字节数据

描述:利用高效字节输出流往C盘下的d.txt文件输出一个字节数。

答案

操作步骤:

1. 创建字节输出流对象关联文件路径
2. 利用字节输出流对象创建高效字节输出流对象
3. 调用高效字节输出流对象的write方法写出一个字节
4. 关闭高效流，释放资源。

代码:

**public class** Test01\_01 {

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

// 创建字节输出流FileOutputStream对象并指定文件路径。

FileOutputStream fos = **new** FileOutputStream("c:\\d.txt");

// 利用字节输出流创建高效字节输出流对象

BufferedOutputStream bos = **new** BufferedOutputStream(fos);

// 调用高效字节输出流对象的write(int byte)方法写出一个字节数据

bos.write(97);

// 关闭流

bos.close();

}

}

## 练习二:高效字节输出流写出字节数组数据

描述:利用高效字节输出流往C盘下的e.txt文件写出一个字节数组数据，如写出：”i love java”

答案

操作步骤:

1. 创建字节输出流对象关联文件路径
2. 利用字节输出流对象创建高效字节输出流对象
3. 定义字符串存放要输出的数据，然后将字符串转换为字节数组。
4. 调用高效字节输出流对象的write方法将字节数组输出。
5. 关闭高效流。

代码:

**public class** Test01\_02 {

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

// 创建字节输出流FileOutputStream对象并指定文件路径。

FileOutputStream fos = **new** FileOutputStream("c:\\e.txt");

// 利用字节输出流创建高效字节输出流对象

BufferedOutputStream bos = **new** BufferedOutputStream(fos);

// 调用高效字节输出流对象的write(byte[] buff)方法写出一个字节数据

bos.write("i love java".getBytes());

// 关闭流

bos.close();

}

}

## 练习三:高效流文件复制

描述:利用高效字节输入流和高效字节输出流完成文件的复制。

要求：

1. 将C盘下的c.png文件复制到D盘下
2. 一次读写一个字节数组方式复制

答案

操作步骤:

1. 创建字节输入流对象并关联文件路径
2. 利用字节输入流对象创建高效字节输入流对象
3. 创建字节输出流对象并关联文件路径
4. 利用字节输出流对象创建高效字节输出流对象
5. 创建字节数组用来存放读取的字节数
6. 利用高效字节输入流循环读取文件数据，每读取一个字节数组，利用高效字节输出流对象将字节数组的内容输出到目标文件中。直到读取到文件末尾。
7. 关闭高效流对象

代码:

**public class** Test01\_03 {

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

// 创建字节输入流对象并关联文件路径

FileInputStream fis = **new** FileInputStream("c:\\c.png");

// 利用字节输出流对象创建高效字节输出流对象

BufferedInputStream bis = **new** BufferedInputStream(fis);

// 创建字节输出流对象并指定文件路径。

FileOutputStream fos = **new** FileOutputStream("d:\\c.png");

// 利用字节输出流创建高效字节输出流对象

BufferedOutputStream bos = **new** BufferedOutputStream(fos);

// 定义字节数组接收读取的字节

**byte**[] buffer = **new byte**[1024];

// 定义变量接收读取的字节数

**int** len = -1;

// 循环读取图片数据

**while**((len = bis.read(buffer)) != -1) {

// 每读取一个字节的数据就写出到目标文件中

bos.write(buffer,0,len);

}

// 关闭流

bis.close();

bos.close();

}

}

## 练习四:高效字符流和集合的综合使用

描述:

分析以下需求，并用代码实现

实现一个验证码小程序，要求如下：

1. 在项目根目录下新建一个文件：data.txt,键盘录入3个字符串验证码，并存入data.txt中，要求一个验证码占一行；

2. 键盘录入一个需要被校验的验证码，如果输入的验证码在data.txt中存在：在控制台提示验证成功，如果不存在控制台提示验证失败

答案代码

public class Test01\_04 {

public static void main(String[] args) throws Exception {

//键盘录入3个字符串并写入项目根路径下的data.txt文件中

writeString2File();

//验证码验证

verifyCode();

}

/\*\*

\* 验证码验证

\* @throws Exception

\*/

private static void verifyCode() throws Exception {

//创建ArrayList集合，用于存储文件中的3个验证码

ArrayList<String>list = new ArrayList<>();

//创建高效字符缓冲输入流对象,并和data.txt文件关联

BufferedReader br = new BufferedReader(new FileReader(new File("data.txt")));

String line = null;

//循环读取每一行

while(null!=(line = br.readLine())) {

//将读到的每一行信息存入到list集合中

list.add(line);

}

//关闭流对象

br.close();

//创建键盘录入对象

Scanner sc = new Scanner(System.in);

//提示用户输入验证码

System.out.println("请输入一个验证码");

String code = sc.nextLine();

if(list.contains(code)) {

System.out.println("验证成功");

} else {

System.out.println("验证失败");

}

}

/\*\*

\* 键盘录入3个字符串并写入项目根路径下的data.txt文件中

\* @throws Exception

\*/

private static void writeString2File() throws Exception {

//创建高效字符缓冲输出流对象并和data.txt文件关联

BufferedWriter bw = new BufferedWriter(new FileWriter(new File("data.txt")));

String line = null;

//创建键盘录入对象

Scanner sc = new Scanner(System.in);

for(inti = 0;i<3;i++) {

System.out.println("请输入第"+(i+1)+"个字符串验证码");

//读取用户键盘录入的一行验证码信息

line = sc.nextLine();

//将读取到的一行验证码写入到文件中

bw.write(line);

//写入换行符

bw.newLine();

}

//关闭流对象

bw.close();

}

}

## 练习五:转换输出流的使用

描述:现有一字符串：”我爱Java”。将该字符串保存到当前项目根目录下的a.txt文件中。

要求：使用gbk编码保存。

注意：idea的默认编码是utf-8,所以可以通过file🡪settings🡪file encodings设置为gbk格式，否则打开a.txt文件看到的将会是乱码。

答案

操作步骤:

1. 创建文件字节输出流关联目标文件
2. 根据文件字节输出流创建转换输出流对象，并指定编码字符集为：gbk
3. 调用流对象的方法将字符串写出到文件中。
4. 关闭流并释放资源。

代码:

**public class** Test01\_05 {

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

// 要保存的字符串

String content = "我爱Java";

// 创建字节输出流对象

FileOutputStream fos = **new** FileOutputStream("a.txt");

// 创建转换输出流对象

OutputStreamWriter osw = **new** OutputStreamWriter(fos, "gbk");

// 调用方法写出数据

osw.write(content);

// 关闭流释放资源

osw.close();

}

}

## 练习六:转换输入流的使用

描述:利用转换输入流将当前项目根目录下使用gbk编码的a.txt文件的内容读取出来，并打印在控制台上。

要求：不能出现乱码的情况。

答案

操作步骤:

1. 创建字节输入流对象指定文件路径。
2. 根据字节输入流对象创建转换输入流对象并指定字符集编码为：gbk
3. 调用转换输入流对象的读取方法读取内容
4. 关闭流释放资源

代码:

**public class** Test01\_06 {

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

// 创建字节输入流对象并关联文件

FileInputStream fis = **new** FileInputStream("a.txt");

// 创建转换输入流对象

InputStreamReader isr = **new** InputStreamReader(fis,"gbk");

// 定义字符数组存放读取的内容

**char**[] buffer = **newchar**[1024];

// 定义变量接收读取的字符个数

**int**len = -1;

**while**((len = isr.read(buffer)) != -1) {

System.***out***.print(**new** String(buffer,0,len));

}

// 关闭流

isr.close();

}

}