**每日作业卷答案**

**JavaSE第3天**

传智播客.黑马程序员

# 基础题

### 练习一：List接口的特点

1. 请简述List接口的特点。

* 它是一个元素存取有序的集合。例如，存元素的顺序是11、22、33。那么集合中，元素的存储就是按照11、22、33的顺序完成的）。
* 它是一个带有索引的集合，通过索引就可以精确的操作集合中的元素（与数组的索引是一个道理）。
* 集合中可以有重复的元素，通过元素的equals方法，来比较是否为重复的元素。

### 练习二：hashCode和equals方法

1. 请简述HashSet去除重复元素的原理。

* 调用被添加元素的hashCode()，和HashSet中已有元素的hashCode比较是否相同
* 如果不相同，直接存储
* 如果相同，调用equals方法比较是否相同
* 不相同，直接存储元素
* 相同，认为是同一元素.不存储

### 练习三：数据结构

1. 简述常见的数据结构中元素的存取特点。

* 栈：stack，又称堆栈，对元素的存取特点是：先进后出。即，存进去的元素，要在后它后面的元素依次取出后，才能取出该元素。
* 队列：queue，简称队，对元素的存取特点是：先进先出。即，存进去的元素，要在后它前面的元素依次取出后，才能取出该元素。
* 数组:Array，是有序的元素序列，对元素的存取特点是：

1、查找元素快：通过索引，可以快速访问指定位置的元素

2、增删元素慢

（1）指定索引位置增加元素：需要创建一个新数组，将指定新元素存储在指定索 引位置，再把原数组元素根据索引，复制到新数组对应索引的位置。

（2）指定索引位置删除元素：需要创建一个新数组，把原数组元素根据索引，复 制到新数组对应索引的位置，原数组中指定索引位置元素不复制到新数组中。

* 链表:linked list，对元素的存取有如下的特点：

1、多个结点之间，通过地址进行连接。例如，多个人手拉手，每个人使用自己的 右手拉住下个人的左手，依次类推，这样多个人就连在一起了。

2、查找元素慢：想查找某个元素，需要通过连接的节点，依次向后查找指定元素。

3、增删元素快：

增加元素：只需要修改连接下个元素的地址即可。

删除元素：只需要修改连接下个元素的地址即可。

### 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);  
 }  
}

（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);  
 }  
}

### 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);  
 }  
 }  
}

### 练习七：List集合元素替换

1. 向list集合添加姓名{张三,李四,王五,二丫,钱六,孙七},将二丫替换为王小丫。

**public class** ListTest01 {  
 **public static void** main(String[] args) {  
 //1.创建List集合对象  
 List<String> list = **new** ArrayList<>();  
 //2.存入数据  
 list.add(**"张三"**);  
 list.add(**"李四"**);  
 list.add(**"王五"**);  
 list.add(**"二丫"**);  
 list.add(**"钱六"**);  
 list.add(**"孙七"**);  
 //3.遍历集合，找到"二丫",便将其替换为"王小丫"  
 //利用普通for循环遍历List集合  
 **for**(**int** i = 0;i<list.size();i++) {  
 //获取当前元素  
 String thisName = list.get(i);  
 //如果当前元素是"二丫"  
 **if**(**"二丫"**.equals(thisName)) {  
 //将其改为"王小丫"  
 list.set(i, **"王小丫"**);  
 }  
 }  
 System.**out**.println(list);  
 }  
}

### 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);  
 }  
 }  
}

### 练习九：Collections工具类使用

1. ArrayList集合中有如下内容： {33,11,77,55}，使用Collections.sort()对ArrayList集合中的数据进行排序，并打印出排序后的结果。

**public class** CollectionsTest01 {  
 **public static void** main(String[] args) {  
 // 1.创建ArrayList  
 ArrayList<Integer> arr = **new** ArrayList<Integer>();  
  
 // 2.使用add方法添加{33,11,77,55}四个元素  
 arr.add(33);  
 arr.add(11);  
 arr.add(77);  
 arr.add(55);  
  
 // 3.调用Collections的sort方法,对集合排序  
 Collections.sort(arr);  
  
 // 4.使用增强for遍历ArrayList集合  
 **for** (Integer integer : arr) {  
 System.**out**.println(integer);  
 }  
 }  
}

## 扩展题

### 练习十：LinkedList使用

1. 已知数组存放一批QQ号码，QQ号码最长为11位，最短为5位String[] strs = {"12345","67891","12347809933","98765432102","67891","12347809933"}。

将该数组里面的所有qq号都存放在LinkedList中，将list中重复元素删除，将list中所有元素分别用迭代器和增强for循环打印出来。

**public class** LinkedListTest03 {  
 **public static void** main(String[] args) {  
 // 1.定义QQ号码数组String[] strs = {"12345","67891",1"2347809933","98765432102","67891","12347809933"}  
 String[] strs = { **"12345"**, **"67891"**, **"2347809933"**, **"98765432102"**, **"67891"**, **"12347809933"** };  
  
 // 2.创建LinkedList  
 LinkedList<String> qqList = **new** LinkedList<>();  
  
 // 3.遍历strs获取每个qq号码  
 **for** (String qq : strs) {  
 // 4.判断LinkedList是否已经存在这个qq号码  
 **if** (!qqList.contains(qq)) {  
 // 5.不存在这个qq号码则添加到LinkedList中  
 qqList.add(qq);  
 }  
 }  
  
 // 6.增强for遍历  
 **for** (String qq : qqList) {  
 System.**out**.println(qq);  
 }  
 System.**out**.println(**"------------------"**);  
  
 // 7.迭代器遍历  
 Iterator<String> iterator = qqList.iterator();  
 **while** (iterator.hasNext()) {  
 String string = iterator.next();  
 System.**out**.println(string);  
 }  
 }  
}

### 练习十一：LinkedHashSet的使用

1. 键盘录入一个字符串，去掉其中重复字符，打印出不同的那些字符，必须保证顺序。例如输入：aaaabbbcccddd，打印结果为：abcd。

**public class** LinkedHashSetTest02 {  
 **public static void** main(String[] args) {  
 // 1.创建Scanner对象,用于键盘录入  
 Scanner sc = **new** Scanner(System.**in**);  
  
 System.**out**.println(**"请输入一个字符串"**);  
  
 // 2.调用Scanner的nextLine()方法,让用户输入一个字符串  
 String line = sc.nextLine();  
  
 // 3.创建LinkedHashSet.用于去除重复的字符串,并保证迭代顺序  
 LinkedHashSet<Character> chs = **new** LinkedHashSet<>();  
  
 // 4.将字符串串转成char[]  
 **char**[] charArray = line.toCharArray();  
  
 // 5.使用增强for循环遍历每个字符  
 **for** (**char** c : charArray) {  
 // 6.将每个字符添加到LinkedHashSet中  
 chs.add(c);  
 }  
  
 // 7.使用增强for打印LinkedHashSet中的内容  
 **for** (**char** c : chs) {  
 System.**out**.print(c);  
 }  
 }  
}

### 练习十二：HashSet的使用

1. 双色球规则：双色球每注投注号码由6个红色球号码和1个蓝色球号码组成。红色球号码从1—33中选择；蓝色球号码从1—16中选择；请随机生成一注双色球号码。（要求同色号码不重复）

**public class** HashSetTest02 {  
 **public static void** main(String[] args) {  
 // 1.创建Random随机数对象  
 Random ran = **new** Random();  
  
 // 2.创建HashSet用于保存不重复的红球  
 HashSet<Integer> hs = **new** HashSet<>();  
  
 // 3.循环判断红球数量是否小于6个  
 **while** (hs.size() < 6) {  
 // 4.球数量小于6个就产生一个红球.添加到HashSet中

// 5.如果产生重复号码，往HashSet里添加不进去，所以会再次生成号码  
 **int** num = ran.nextInt(33) + 1;  
 hs.add(num);  
 }  
  
 // 6.再生成一个篮球  
 **int** blueBall = ran.nextInt(16) + 1;  
  
 // 7.打印中奖号码  
 System.**out**.println(**"双色球中奖号码:"**);  
 System.**out**.print(**"红球是: "**);

**for** (Integer redBall : hs) {  
 System.**out**.print(redBall + **" "**);  
 }  
 System.**out**.println();  
 System.**out**.println(**"蓝球是: "** + blueBall);  
 }  
}

### 练习十三：Comparable和Comparator两个接口的使用

1. 分别用Comparable和Comparator两个接口对下列四位同学的成绩做降序排序，如果成绩一样，那在成绩排序的基础上按照年龄由小到大排序。

|  |  |  |
| --- | --- | --- |
| 姓名（String） | 年龄（int） | 分数（float） |
| liusan | 20 | 90.0F |
| lisi | 22 | 90.0F |
| wangwu | 20 | 99.0F |
| sunliu | 22 | 100.0F |

1. Comparable

Student 类

**public class** Student **implements** Comparable<Student>{  
 **private** String **name**;  
 **private int age**;  
 **private float score**;  
  
 **public** Student(String name, **int** age, **float** score) {  
 **this**.**name** = name;  
 **this**.**age** = age;  
 **this**.**score** = score;  
 }  
  
 **public** String toString() {  
 **return name**+**"\t\t"**+**age**+**"\t\t"**+**score**;  
 }  
  
 @Override  
 **public int** compareTo(Student o) {  
 **if**(**this**.**score**>o.**score**){  
 //由高到底排序  
 **return** -1;  
 }**else if**(**this**.**score**<o.**score**) {  
 **return** 1;  
 }**else**{  
 **if**(**this**.**age**>o.**age**) {  
 //由底到高排序  
 **return** 1;  
 }**else if**(**this**.**age**<o.**age**) {  
 **return** -1;  
 }**else** {  
 **return** 0;  
 }  
 }  
 }  
}

测试类

**public class** ComparableTest {  
 **public static void** main(String[] args) {  
 Student stu[]={**new** Student(**"liusan"**,20,90.0f),  
 **new** Student(**"lisi"**,22,90.0f),  
 **new** Student(**"wangwu"**,20,99.0f),  
 **new** Student(**"sunliu"**,22,100.0f)};  
 java.util.Arrays.*sort*(stu);  
 **for**(Student s:stu){  
 System.***out***.println(s);  
 }  
 }  
}

1. Comparator

Student 类

**public class** Student {  
 **private** String **name**;  
 **private int age**;  
 **private float score**;  
  
 **public** Student(String name, **int** age, **float** score) {  
 **this**.**name** = name;  
 **this**.**age** = age;  
 **this**.**score** = score;  
 }  
  
 **public** String getName() {  
 **return name**;  
 }  
 **public void** setName(String name) {  
 **this**.**name** = name;  
 }  
 **public int** getAge() {  
 **return age**;  
 }  
 **public void** setAge(**int** age) {  
 **this**.**age** = age;  
 }  
 **public float** getScore() {  
 **return score**;  
 }  
 **public void** setScore(**float** score) {  
 **this**.**score** = score;  
 }  
  
 **public** String toString() {  
 **return name**+**"\t\t"**+**age**+**"\t\t"**+**score**;  
 }  
}

Comparator比较器

**public class** StudentComparator **implements** Comparator<Student> {  
 @Override  
 **public int** compare(Student o1, Student o2) {  
 **if**(o1.getScore()>o2.getScore()) {  
 **return** -1;  
 }**else if**(o1.getScore()<o2.getScore()) {  
 **return** 1;  
 }**else**{  
 **if**(o1.getAge() > o2.getAge()) {  
 **return** 1;  
 }**else if**(o1.getAge() < o2.getAge()) {  
 **return** -1;  
 }**else**{  
 **return** 0;  
 }  
 }  
 }  
}

测试类

**public class** ComparatorTest {  
 **public static void** main(String[] args) {  
 Student stu[]={**new** Student(**"liusan"**,20,90.0f),  
 **new** Student(**"lisi"**,22,90.0f),  
 **new** Student(**"wangwu"**,20,99.0f),  
 **new** Student(**"sunliu"**,22,100.0f)};  
 java.util.Arrays.sort(stu,**new** StudentComparator());  
 **for**(Student s:stu) {  
 System.**out**.println(s);  
 }  
 }  
}