【济南中心】JAVA 编程阶梯：基础篇之第十六章

- **集合框架练习题**

1.去除 ArrayList 中重复字符串元素方式

```
/**
 * 去除 ArrayList 中重复元素
 *
 * @author Somnus
 *
 */
public class Demo {

	public static void main(String[] args) {

		List<String> list = new ArrayList<String>();
		list.add("a");
		list.add("b");
		list.add("ab");
		list.add("ba");
		list.add("ab");

		getSingleElement(list);// 去除重复元素。

		System.out.println(list);

	}

	public static void getSingleElement(List<String> list) {

		// 1,创建一个临时容器。
		List<String> temp = new ArrayList<>();

		// 2,遍历原容器。
		for (Iterator<String> it = list.iterator(); it.hasNext();) {
			String obj = it.next();

			// 对遍历到的每一个元素都到临时容器中去判断
是否包含。

			if (!temp.contains(obj)) {// 如果不存在，
				temp.add(obj);// 添加到临时
容器。
```

```
                }
            }
            // 唯一性的元素已经被记录到临时容器中。
            // 清空原容器中的元素。
            list.clear();

            // 把临时容器中的元素添加到原容器中。
            list.addAll(temp);

        }
}
```

2.去除 ArrayList 中重复自定义对象元素（代码见附件）

- **LinkedList**

构造方法摘要：

LinkedList(): 构造一个空列表。

LinkedList(Collection<? extends E> c): 构造一个包含指定 collection 中的元素的列表，这些元素按其 collection 的迭代器返回的顺序排列。

方法摘要:(特有的)

|--->添加

public void addFirst(E e): 将指定元素插入此列表的开头。

public void addLast(E e): 将指定元素添加到此列表的结尾。

|--->获取元素，但不删除元素

public E get(int index): 返回此列表中指定位置处的元素。

public E getFirst(): 返回此列表的第一个元素。

public E getLast(): 返回此列表的最后一个元素。

|--->获取元素且删除元素

public E remove()：获取并移除此列表的头(第一个元素)。

public E remove(int index)： 移除此列表中指定位置处的元素。

public boolean remove(Object o)： 从此列表中移除首次出现的指定元素（如果存在）。

public E removeFirst()： 移除并返回此列表的第一个元素。

public E removeLast(): 移除并返回此列表的最后一个元素。

|--->修改

public E set(int index, E element) 将此列表中指定位置的元素替换为指定的元素。

- **栈和队列数据结构**

栈特点：先进后出 队列特点：先进先出

```
/**
 * 定义一个队列数据结构。
 *
 * @author Somnus
 */

class Queue {
		// 封装了一个链表数据结构。
		private LinkedList link;

		/*
		 * 队列初始化时，对链表对象初始化。
		 */
		Queue() {
				link = new LinkedList();
		}

		/**
		 * 队列的添加元素功能。
		 */
		public void myAdd(Object obj) {
				// 内部使用的就是链表的方法。
```

```
                link.addFirst(obj);
        }

        /**
          * 队列的获取方法。
          */
        public Object myGet() {
                return link.removeLast();
        }

        /**
          * 判断队列中元素是否空，没有元素就为 true。
          */
        public boolean isNull() {
                return link.isEmpty();
        }
}
```

- **泛型**

为什么会出现泛型？

因为集合存放的数据类型不固定，故往集合里面存放元素时，存在安全隐患，如果在定义集合时，可以想定义数组一样指定数据类型，那么就可以解决该类安全问题。JDK1.5 后出现了泛型，用于解决集合框架的安全问题。泛型是一个类型安全机制，只存在于编译期间。

泛型好处

将运行时期出现的 ClassCastException(类型转换异常)问题转移到编译时期；避免了强制转换的麻烦

泛型基本使用

通过<>来定义要操作的引用数据类型

```
ArrayList<String> al = new ArrayList<String>();
ArrayList<String> al = new ArrayList<>();
ArrayList<String> al = new ArrayList();
```

泛型使用注意事项

前后的泛型必须一致,或者后面的泛型可以省略不写(1.7 的新特性菱形泛型)

- **泛型的形式**

泛型类：即自定义泛型类

A：当类中要操作的引用数据类型不确定时，早起定义 Object 来完成扩展，现在定义泛型来完成

B：局限性：泛型类定义的泛型，在整个类中有效，如果该泛型类的方法被调用，当泛型类的对象明确要操作的类型后，所有要操作的类型就被固定。

泛型方法：泛型放在返回值前面，修饰符的后面

A:为了避免泛型类的局限性，让不同方法可以操作不同的类型，而且类型还不确定，则可以将泛型定义在方法上

B:特殊之处：静态方法不可以访问类上定义的泛型如果静态方法操作的应用数据类型不确定，可以将泛型定义在静态方法上

泛型接口：

当泛型定义在接口上时，则子类中要指定实现接口类型，同时还可以子类也可以定义为泛型类

- **泛型的高级应用**

？通配符

当指定两种泛型的集合，则迭代时也要定义两种泛型的迭代器，麻烦，此时可通过将迭代器的泛型改为<?>，如 Iterator<?> it=al.iterator();

两种泛型限定

向上限定： <? extends E> ;E 可以接收 E 类型或者 E 的子类

向下限定： <? super E> ;E 可以接收 E 类型或者 E 的父类

1.<?>：

1).变量可以指向什么类型的对象：具有任何泛型的集合对象；

2).可以存储什么东西：由于？不确定具体类型，所以不能 add()任何类型

3).取出时用什么接收：只能用 Object 接收；

作用：不能存入，只能获取，所以一般用作方法的返回值声明；

2.<? extends E> :接收 E 类型或者 E 的子类型

1).变量可以指向什么类型的对象：具有 E 泛型或者 E 子类泛型的集合；

2).可以存储什么东西：由于不确定是哪个 E 的子类，所以不能存任何东西；

3).取出时用什么接收：只能用 E 及 E 的父类类型接收；

应用：往集合里存的时候定义一个大的类型，可以存储 E 类型或者 E 的子类。

3.<? super E>： 接收 E 类型或者 E 的父类型

1).变量可以指向什么类型的对象：具有 E 泛型或者 E 父类泛型的集合；

2).可以存储什么东西：只能存储 E 或者 E 的子类对象；

3).取出时用什么接收：由于存储的可能是任何的 E 的子类对象，所以只能用 Object 接收；

应用：从集合里取出的时候进行操作的时候，可以定义 E 类型或者 E 的父类型接收

- **增强 for**

增强 for 概述

简化数组和 Collection 集合的遍历

格式：

for(元素数据类型 变量 ： 数组或者 Collection 集合) {

使用变量即可，该变量就是元素

}

注意事项：

迭代变量必须在( )中定义！

集合变量可以是数组或实现了 Iterable 接口的集合类

```
/**
 * 增强 for 循环。
 *
 * @author Somnus
 */

class Demo {
		public static void main(String[] args) {

			List<String> list = new ArrayList<String>();
			list.add("a");
			list.add("b");
			list.add("c");
			list.add("d");

			for (Object obj : list) {// 简化。
				System.out.println(obj);
			}

		}
}
```

- **静态导入**

import 语句可以导入一个类或某个包中的所有类

import static 语句导入一个类中的某个静态方法或所有静态方法

静态导入后，静态方法前面就不用写类名.方法的方式类调用

语法举例：

```
import static java.lang.Math.sin;//导入一个静态方法
import static java.lang.Math.*; //导入一个类中的所有静态方法
```

静态导入使用注意：

当类名重复时，需要制定具体的包名；

当方法重名时，需要制定具体所属的对象或者类

- **可变参数**

可变参数的特点：

可变参数只能出现在参数列表的最后；

...位于变量类型和变量名之间，前后有无空格都可以;

调用可变参数的方法时，编译器为该可变参数隐含创建一个数组，

在方法体中以数组的形式访问可变参数。

可变参数举例：

```
修饰符 返回值类型 方法名(数据类型…  变量名){} 如 int... arr 表示可变参数数组
public static void show(String str , int... arr){}
```

注意事项：

这里的变量其实是一个数组

如果一个方法有可变参数，并且有多个参数，那么，可变参数肯定是最后一个

Arrays 工具类

public static String toString(int[] a)： 把一个 int[]数组转换为一个 String；

public static void sort(int[] a)： 把数组 a 进行排序；对指定的 int 型数组按数字升序进行排序。该排序算法是一个经过调优的快速排序法

public static int binarySearch(int[] a,int key)：使用二分查找法，在数组 a 中查找值：key。如果没找到，返回"负值"；

静态方法摘要：

static <T> List<T> asList(T... a) 返回一个受指定数组支持的固定大小的列表。

注意：

该方法将一个数组变成集合后，不可以使用集合的增删方法，因为数组的长度是固定的 !如果增删，则发生 UnsupportedOprationException(不支持操作异常)

如果数组中的元素都是基本数据类型，则该数组变成集合时，会将该数组作为集合的一个元素出入集合

如果数组中的元素都是对象，如 String，那么数组变成集合后，数组中的元素就直接转成集合中的元素

识别二维码 关注黑马程序员视频库
免费获得更多 IT 资源