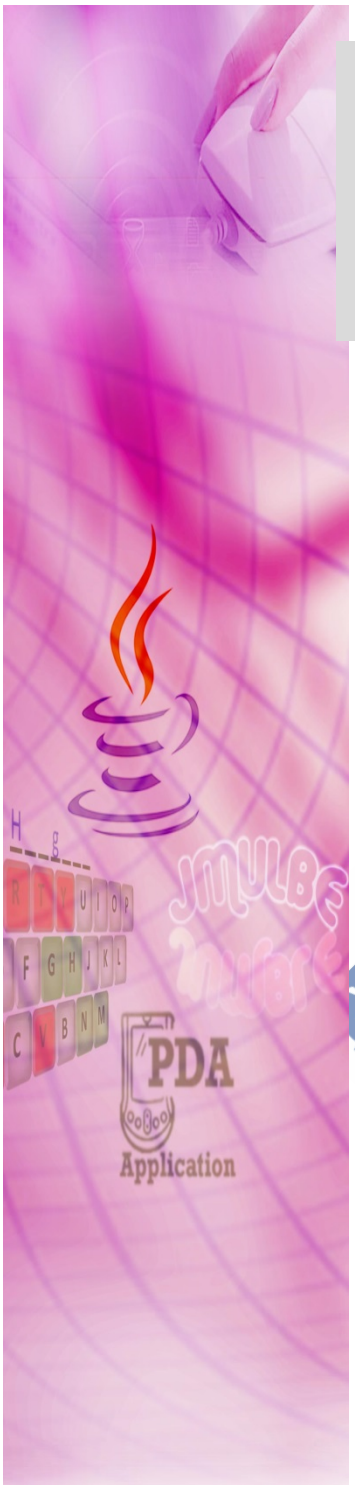


第 4 章

使用集合

有时，您需要处理对象集合。该任务可以借助于数组来执行。但是，数组仅对一种类型的数据有效。因此，要使用不同类型的对象集合，Java 提供了集合框架。该框架提供了用于实现数据结构的接口、类和算法。集合框架用于多种目的，例如存储、检索和操作对象。

本章解释如何使用各种接口（例如 `List`、`Set`、`Map` 和 `Deque`）来创建对象集合。此外，还讨论存储功能。

目标

在本章中，您将学习：

- 使用 Set 接口
- 使用 List 接口
- 描述 Map 接口
- 使用 Deque 接口
- 实现排序

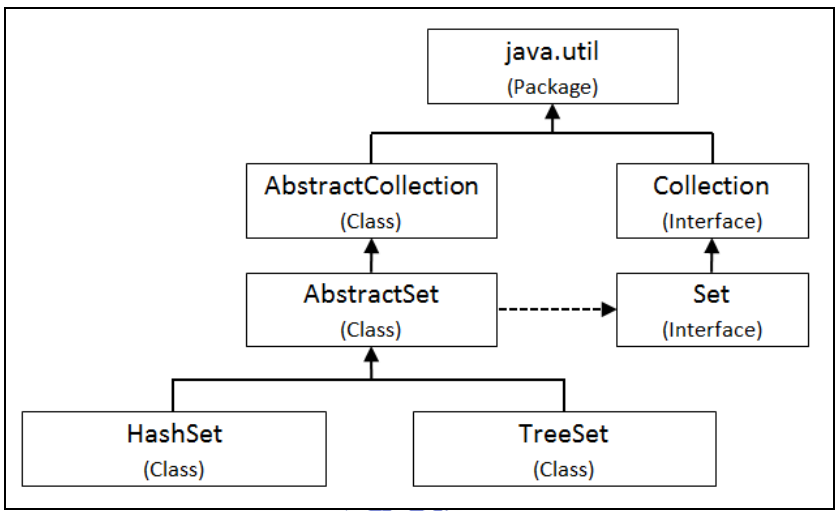

NIIT
training
china
.com


NIIT
training
china
.com

使用 Set 接口

思考以下场景：您需要开发一个聊天应用程序。在该应用程序中，您可以实现一组功能根据可用联系人列表动态创建一个聊天组。此外，还可以根据用户首选项在聊天组中添加和删除成员。要实现该功能，您可以创建联系人集合。但是，为确保联系人在聊天组中的唯一性，您需要使用 Set 接口创建该集合。

Set 接口用于创建唯一对象集合。下图显示用于实现 Set 接口的 java.util 包的类层次接口。



java.util 包的类层次接口

上图显示 java.util 包包含 AbstractCollection、AbstractSet、HashSet 和 TreeSet 类以及 Collection 和 Set 接口。AbstractSet 类实现 Set 接口并扩展 AbstractCollection 类，后者又通过 HashSet 和 TreeSet 类扩展。此外，Set 接口扩展 Collection 接口。

包 java.util 提供 Iterator 接口以遍历 set 集合。Iterator 接口的引用可以通过使用 Set 接口的 iterator() 方法获取。Iterator 接口包含多种方法，例如 hasNext() 和 next()，用于帮助遍历 set 集合。

使用 HashSet 类

HashSet 类提供 Set 接口的实现以使您能够创建一个可在其中快速插入的集合，因为它不对元素进行排序。HashSet 类提供可用于创建实例的多种构造函数。

下表列出 HashSet 类的常用构造函数。

构造函数	*描述*
HashSet()	*创建 HashSet 的空实例。*
HashSet(Collection<? extends E> c)	*创建具有指定集合的实例。*
HashSet(int initialCapacity)	*创建具有指定初始容量的 HashSet 空实例。初始容量是 HashSet 的大小，在 HashSet 中添加或删除对象时该大小会自动增加或缩减。*

HashSet 类的构造函数

下表列出 HashSet 类的常用方法。

方法	*描述*
boolean add(E e)	*用于向 HashSet 添加指定对象（如果不存在）。如果存在，它将保持 HashSet 不变并返回 false。*
void clear()	*用于从 HashSet 删除所有对象。*
boolean remove(Object o)	*用于从 HashSet 删除指定对象。*
int size()	*用于获取 HashSet 中的对象数量。*

HashSet 类的方法

思考使用 HashSet 类创建集合的以下代码段：

```
import java.util.*;
class Contacts
{
/* code to create contacts */
}
public class ChatGroup
{
public static void main(String args[])
{
Contacts a=new Contacts();
Contacts b=new Contacts();
Contacts c=new Contacts();
HashSet<Contacts> hs=new HashSet<Contacts>();
hs.add(a);
hs.add(b);
hs.add(c);
```

```
}
}
```

在上述代码段中，创建了包含 Contacts 对象集合的 HashSet 类的泛型类型对象。HashSet 类的 add() 方法用于将 Contacts 对象添加到 HashSet 对象。

您可以使用以下代码段从 HashSet 对象删除 Contacts 对象：

```
hs.remove(c);
```

您可以使用以下代码段查找 HashSet 对象中可用的对象数：

```
hs.size();
```

您可以使用以下代码段遍历 HashSet 对象：

```
Iterator i = hs.iterator();
while(i.hasNext())
{
System.out.println(i.next());
}
```

在上述代码段中，iterator() 方法用于检索 Iterator 引用。此外，hasNext() 方法用于检查 HashSet 中是否存在更多元素，next() 方法用于检索元素。

您已学习泛型如何解决 Java 应用程序中的类型安全性问题。此外，您还使用了泛型 HashSet 集合，其中需要指定集合将存储的对象类型。但是，在引入泛型之前已开发了若干现有应用程序，此类应用程序应继续运行。要确保此类应用程序的互操作性，Java 还提供了对非泛型集合的支持。在上述示例中，通过指定泛型类型 <Contact> 创建了对象集合。在该上下文中，集合仅是 Contact 对象的集合。但是，您还可以使用可存储任何类型对象的非泛型 HashSet 集合。为此，可使用以下代码段：

```
Contacts a=new Contacts();
HashSet hs=new HashSet();
hs.add(a);
hs.add("String object");
hs.add(new Integer(3));
```

在上述代码段中，创建了包含不同对象集合的非泛型 HashSet 集合。

注释

原始类型是不带任何类型自变量的泛型类或接口的名称。

使用 TreeSet 类

TreeSet 类提供使您能够创建有序对象集合的 Set 接口的实现。插入或添加对象过程缓慢，因为每次插入时都要对对象排序。然而，TreeSet 自动按有序顺序排列对象，这样就改善了从大量信息中进行数据搜索。TreeSet 类提供可用于创建 TreeSet 实例的多种构造函数。下表列出 TreeSet 类的常用构造函数。

构造函数	***描述***
TreeSet()	*创建 TreeSet 的空实例。*
TreeSet(Collection<? extends E> c)	*创建具有指定集合的实例。*

TreeSet 类的构造函数

下表列出 TreeSet 类的常用方法。

方法	***描述***
boolean add(E e)	*用于向 TreeSet 添加指定对象（如果不存在）。*
void clear()	*用于从 TreeSet 删除所有对象。*
boolean remove(Object o)	*用于从 TreeSet 删除指定对象。*
int size()	*用于获取 TreeSet 中的对象数量。*

TreeSet 类的方法

思考在 TreeSet 集合中添加和删除对象的以下代码：

```
import java.util.Iterator;
import java.util.TreeSet;

public class TreeSetDemo
{
public static void main(String[] args)
{
TreeSet<Integer> obj = new TreeSet<Integer>();
Integer iobj1 = new Integer(114);
Integer iobj2 = new Integer(111);
Integer iobj3 = new Integer(113);
Integer iobj4 = new Integer(112);

System.out.println("Size of TreeSet is:" + obj.size());
```

```
obj.add(iobj1);
obj.add(iobj2);
obj.add(iobj3);
obj.add(iobj4);
obj.add(iobj2);

System.out.println("\nTreeSet after adding the objects:" + obj);
System.out.println("Size of TreeSet after adding objects:" + obj.size());

obj.remove(iobj3);
obj.remove(iobj1);

System.out.println("\nTreeSet after removing the objects:" + obj);
System.out.println("Size of TreeSet after removing objects:" + obj.size());

System.out.println("\nThe final TreeSet:");
Iterator i = obj.iterator();
while(i.hasNext())
{
System.out.println(i.next());
}
}
}
```

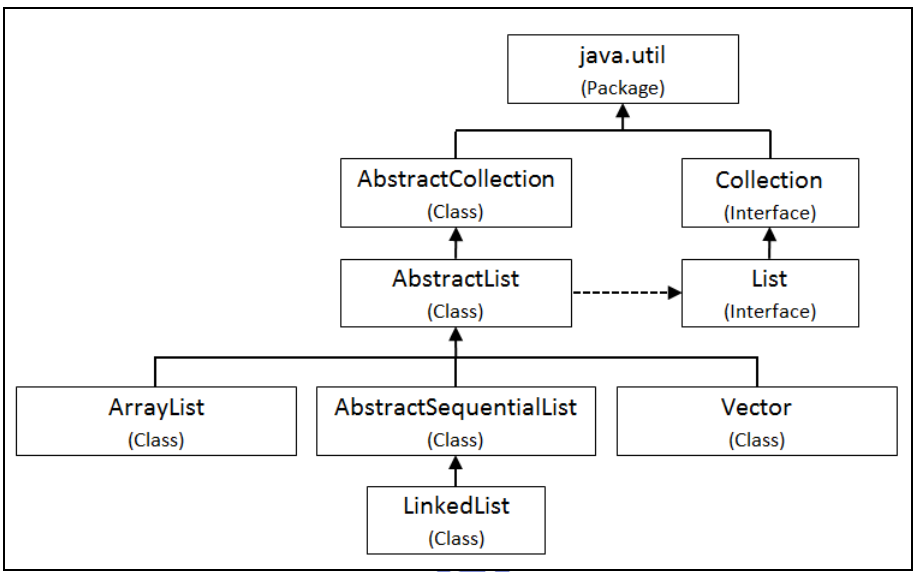
执行上述代码时，会显示以下输出：

```
Size of TreeSet is:0

TreeSet after adding the objects:[111, 112, 113, 114]
Size of TreeSet after adding objects:4

TreeSet after removing the objects:[111, 112]
Size of TreeSet after removing objects:2

The final TreeSet:
111
112
```

在上述代码中，`TreeSet` 对象包含 `Integer` 对象的集合。`TreeSet` 类的 `add()` 方法用于将 `Integer` 对象添加到 `TreeSet` 对象。 并且使用 `remove()` 方法除去两个对象。`Iterator` 接口用于在集合中逐个迭代对象。

使用 List 接口

List 接口用于创建有序对象集合，其中可以包含重复对象。在该集合中，可以逐个插入对象，也可以在列表的指定位置处插入对象。类似地，可以逐个访问对象，也可以从列表的指定位置访问对象。下图显示实现 List 接口的 java.util 包的类层次结构。



java.util 包的类层次接口

上图显示 java.util 包包含 AbstractCollection、AbstractList、ArrayList、AbstractSequentialList、Vector 和 LinkedList 类以及 Collection 和 List 接口。AbstractCollection 类通过 AbstractList 类扩展，后者由通过 ArrayList、AbstractSequentialList 和 Vector 类扩展。AbstractList 类实现 List 接口，后者扩展 Collection 接口。LinkedList 类扩展 AbstractSequentialList 类。

包 java.util 提供遍历 list 集合的 ListIterator 接口。ListIterator 接口的引用可通过使用 List 接口的 listIterator() 方法获得。ListIterator 接口包含有助于沿两个方向遍历列表的多种方法，例如 hasNext()、hasPrevious()、next() 和 previous()。

使用 ArrayList 类

ArrayList 类提供 List 接口的实现。ArrayList 类使您能够创建可调整大小的数组。其大小随着添加更多元素而动态增加。ArrayList 经证明在列表中搜索对象以及在列表末尾插入对象很有效。

ArrayList 类提供可用于创建实例的多种构造函数。下表列出 ArrayList 类的常用构造函数。

构造函数	*描述*
ArrayList()	*创建 ArrayList 的空实例。*
ArrayList(Collection<? extends E> c)	*创建具有指定集合的实例。*
ArrayList(int initialCapacity)	*创建具有指定初始容量的 ArrayList 空实例。初始容量是 ArrayList 的大小，在 ArrayList 中添加或删除对象时该大小会自动增加或缩减。*

ArrayList 类的构造函数

下表列出 ArrayList 类的常用方法。

方法	*描述*
boolean add(E e)	*用于在 ArrayList 末尾添加指定对象。*
void clear()	*用于从 ArrayList 删除所有对象。*
E get(int index)	*用于从 ArrayList 中检索指定索引处的对象。*
E remove(int index)	*用于从 ArrayList 中删除指定索引处的对象。*
boolean remove(Object o)	*用于从 ArrayList 删除指定对象的首次出现。*
int size()	*用于获取 ArrayList 中的对象数量。*

ArrayList 类的方法

思考在 ArrayList 集合中添加和删除对象的以下代码：

```
import java.util.ArrayList;
import java.util.ListIterator;

public class ArrayListDemo
{
public static void main(String[] args)
{
ArrayList<String> obj = new ArrayList<String>();
```

```
String sobj1 = new String("Element 1");
String sobj2 = new String("Element 2");
String sobj3 = new String("Element 3");
String sobj4 = new String("Element 4");

System.out.println("Size of ArrayList is:" + obj.size());

obj.add(sobj1);
obj.add(sobj2);
obj.add(sobj3);
obj.add(sobj4);
obj.add(sobj1);

System.out.println("\nArrayList after adding the objects:" + obj);
System.out.println("Size of ArrayList after adding objects:" + obj.size());

obj.remove(2);
obj.remove(sobj4);

System.out.println("\nArrayList after removing the objects:" + obj);
System.out.println("Size of ArrayList after removing objects:" + obj.size());

System.out.println("\nThe final ArrayList:");
ListIterator i = obj.listIterator();
while(i.hasNext())
{
System.out.println(i.next());
}
}
}
```

在上述代码中，`ArrayList` 对象包含 `String` 对象的集合。`ArrayList` 类的 `add()` 方法用于添加 `String` 对象。 并且使用 `remove()` 方法除去两个对象。`ListIterator` 接口用于在列表中逐个迭代对象。

执行上述代码时，会显示以下输出：

```
Size of ArrayList is:0

ArrayList after adding the objects:[Element 1, Element 2, Element 3, Element
4, Element 1]
Size of ArrayList after adding objects:5

ArrayList after removing the objects:[Element 1, Element 2, Element 1]
Size of ArrayList after removing objects:3



The final ArrayList:
Element 1
Element 2
Element 1
```

使用 LinkedList 类

LinkedList 类提供 List 接口的实现。LinkedList 类使您能够创建*双向链表*。双向链表是一组有序链接的记录，称为节点，其中每个节点包含两个链接，分别引用序列中的上一节点和下一节点。LinkedList 可以向前或向后遍历。LinkedList 经证明在列表中插入和删除对象时很有效。

LinkedList 类提供可用于创建实例的多种构造函数。下表列出 LinkedList 类的常用构造函数。

构造函数	*描述*
LinkedList()	*创建 LinkedList 的空实例。*
LinkedList(Collection<? extends E> c)	*创建具有指定集合的实例。*

LinkedList 类的构造函数

下表列出 LinkedList 类的常用方法。

方法	*描述*
boolean add(E e)	*用于在 LinkedList 末尾添加指定对象。*
void clear()	*用于从 LinkedList 删除所有对象。*
E get(int index)	*用于从 LinkedList 中检索指定索引处的对象。*
E remove(int index)	*用于从 LinkedList 中删除指定索引处的对象。*
boolean remove(Object o)	*用于从 LinkedList 删除指定对象的首次出现。*
int size()	*用于获取 LinkedList 中的对象数量。*

LinkedList 类的方法

思考在 LinkedList 集合中添加和删除对象的以下代码：

```
import java.util.LinkedList;
import java.util.ListIterator;

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

```
LinkedList<Integer> obj = new LinkedList<Integer>();
Integer iobj1 = new Integer(101);
Integer iobj2 = new Integer(102);
Integer iobj3 = new Integer(103);
Integer iobj4 = new Integer(104);

System.out.println("Size of LinkedList is:" + obj.size());

obj.add(iobj1);
obj.add(iobj2);
obj.add(iobj3);
obj.add(iobj4);
obj.add(iobj1);

System.out.println("\nLinkedList after adding the objects:" + obj);
System.out.println("Size of LinkedList after adding objects:" + obj.size());

obj.remove(iobj2);
obj.remove(iobj3);

System.out.println("\nLinkedList after removing the objects:" + obj);
System.out.println("Size of LinkedList after removing objects:" + obj.size());

System.out.println("\nThe final LinkedList:");
ListIterator i = obj.listIterator();
while(i.hasNext())
{
System.out.println(i.next());
}
}
}
```

执行上述代码时，会显示以下输出：

```
Size of LinkedList is:0

LinkedList after adding the objects:[101, 102, 103, 104, 101]
Size of LinkedList after adding objects:5

LinkedList after removing the objects:[101, 104, 101]
Size of LinkedList after removing objects:3


The final LinkedList:
101
104
101
```

在上述代码中，`LinkedList` 对象包含 `Integer` 对象的集合。`LinkedList` 类的 `add()` 方法用于添加 `Integer` 对象。 并且使用 `remove()` 方法除去两个对象。`ListIterator` 接口用于在列表中逐个迭代对象。

使用 Vector 类

Vector 类类似于 ArrayList 和 LinkedList 类。 然而，由于 Vector 类的方法是同步的，因此在多线程环境中给定时间点只能有一个线程访问该类。而另一方面，ArrayList 和 LinkedList 类的方法不同步。

Vector 类提供可用于创建实例的多种构造函数。下表列出 Vector 类的常用构造函数。

构造函数	***描述***
Vector()	*创建 Vector 类的空实例。*
Vector(Collection<? extends E> c>	*创建具有指定集合的实例。*
Vector(int initialCapacity)	*创建具有指定初始容量的 Vector 空实例。初始容量是 Vector 的大小，在 Vector 中添加或删除对象时该大小会自动增加或缩减。*

Vector 类的构造函数

下表列出 Vector 类的常用方法。

方法	***描述***
boolean add(E e)	*用于在 Vector 末尾添加指定对象。*
void clear()	*用于从 Vector 删除所有对象。*
E get(int index)	*用于从 Vector 中检索指定索引处的对象。*
E remove(int index)	*用于从 Vector 中删除指定索引处的对象。*
boolean remove(Object o)	*用于从 Vector 删除指定对象的首次出现。*
int size()	*用于获取 Vector 中的对象数量。*

Vector 类的方法

思考在 Vector 集合中添加和删除对象的以下代码：

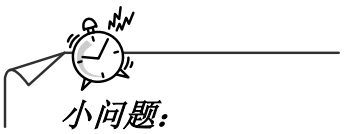
```
import java.util.ListIterator;
import java.util.Vector;
public class VectorDemo
```

```
{
public static void main(String[] args)
{
Vector<Double> obj = new Vector<Double>();
Double dobj1 = new Double(77.5);
Double dobj2 = new Double(68.1);
Double dobj3 = new Double(52.8);
Double dobj4 = new Double(40.2);

System.out.println("Size of Vector is:" + obj.size());

obj.add(dobj1);
obj.add(dobj2);
obj.add(dobj3);
obj.add(dobj4);
obj.add(dobj1);

System.out.println("\nVector after adding the objects:" + obj);
System.out.println("Size of Vector after adding objects:" + obj.size());

obj.remove(dobj1);
obj.remove(dobj3);

System.out.println("\nVector after removing the objects:" + obj);
System.out.println("Size of Vector after removing objects:" + obj.size());

System.out.println("\nThe final Vector:");
ListIterator i = obj.listIterator();
while(i.hasNext())
{
System.out.println(i.next());
}
}
}
```

执行上述代码时，会显示以下输出：

```
Size of Vector is:0

Vector after adding the objects:[77.5, 68.1, 52.8, 40.2, 77.5]
Size of Vector after adding objects:5

Vector after removing the objects:[68.1, 40.2, 77.5]
Size of Vector after removing objects:3

The final Vector:
68.1
40.2
```

在上述代码中，`Vector` 对象包含 `Double` 对象的集合。`Vector` 类的 `add()` 方法用于添加 `Double` 对象。并且使用 `remove()` 方法除去两个对象。`ListIterator` 接口用于逐个迭代对象。



小问题：

以下哪个类使您能够创建唯一对象的集合？

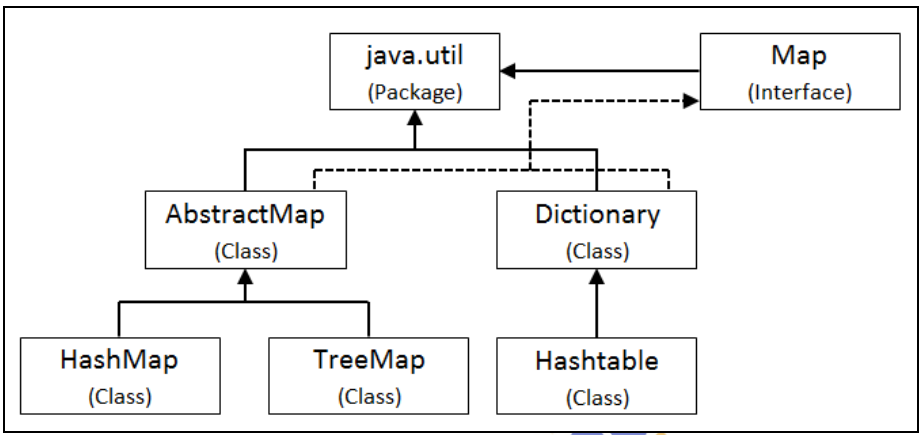
1. `List`
2. `Vector`
3. `TreeSet`
4. `ArrayList`

答案：

3.`TreeSet`

使用 Map 接口

Map 接口使您能够创建具有键 - 值对对象的集合。在该集合中，键和值都是对象。您需要使用键对象访问对应的值对象。Map 接口允许存在重复值对象，但键对象必须唯一。下图显示用于实现 Map 接口的 java.util 包的类层次结构。



java.util 包的类层次接口

上图显示 java.util 包包含 bstractMap、Dictionary、HashMap、TreeMap 和 Hashtable 类以及 Map 接口。HashMap 和 TreeMap 类扩展用于实现 Map 接口的 AbstractMap 类。此外，Hashtable 类扩展也用于实现 Map 接口的 Dictionary 类。

使用 HashMap 类

HashMap 类使您能够创建可使用键访问值的集合。HashMap 按无序形式存储对象。HashMap 类允许一个键具有一个 null 值，值具有任意数量的 null 值。

HashMap 类提供可用于创建实例的多种构造函数。下表列出 HashMap 类的常用构造函数。

构造函数	***描述***
HashMap()	*创建 HashMap 类的空实例。*
HashMap(int initialCapacity)	*创建具有指定初始容量的 HashMap 空实例。初始容量是 HashMap 的大小，在 HashMap 中添加或删除对象时该大小会自动增加或缩减。*
构造函数	***描述***
HashMap(Map<? extends K, ? extends V> m)	*创建具有指定 Map 的相同映射的实例。*

HashMap 类的构造函数

下表列出 HashMap 类的常用方法。

方法	*描述*
void clear()	*用于从 HashMap 删除所有映射。*
V get(Object key)	*用于从 HashMap 获取指定键对象的值对象。*
V put(K key, V value)	*用于在 HashMap 中添加具有指定值对象的指定键对象。*
V remove(Object o)	*用于从 HashMap 中删除指定键对象的映射。*
int size()	*用于获取 HashMap 中的键 - 值映射的数量。*

HashMap 类的方法

注释

K 和 V 分别表示键和值。K 和 V 均用于各种类型的类。

思考在 HashMap 集合中添加和删除对象的以下代码：

```
import java.util.HashMap;

public class HashMapDemo
{
public static void main(String[] args)
{
HashMap<String, Integer> obj = new HashMap<String, Integer>();
Integer iobj1 = new Integer(11);
Integer iobj2 = new Integer(22);
Integer iobj3 = new Integer(33);
Integer iobj4 = new Integer(44);

System.out.println("Size of HashMap is:" + obj.size());

obj.put("L1", iobj1);
obj.put("L2", iobj2);
obj.put("L3", iobj3);
obj.put("L4", iobj4);
obj.put("L5", iobj1);

System.out.println("\nHashMap after adding the objects:" + obj);
System.out.println("Size of HashMap after adding objects:" + obj.size());

obj.remove("L3");
```

```
obj.remove("L5");

System.out.println("\nHashMap after removing the objects:" + obj);
System.out.println("Size of the HashMap after removing objects:" +
obj.size());
}
}
```

执行上述代码时，会显示以下输出：

```
Size of HashMap is:0

HashMap after adding the objects:{L1=11, L2=22, L3=33, L4=44, L5=11}
Size of HashMap after adding objects:5

HashMap after removing the objects:{L1=11, L2=22, L4=44}
Size of the HashMap after removing objects:3
```

在上述代码中，HashMap 对象包含键 - 值对对象的集合。HashMap 类的 put() 方法用于添加键 - 值对对象。并且使用 remove() 方法除去两个对象的映射。

使用 TreeMap 类

TreeMap 类使您能够使用唯一键按有序顺序创建对象集合。有序集合可改善对象的检索过程。

TreeMap 类提供可用于创建实例的多种构造函数。下表列出 TreeMap 类的常用构造函数。

构造函数	***描述***
TreeMap()	*创建具有空 TreeMap 的实例。*
TreeMap(Map<? extends K, ? extends V> m)	*创建具有指定 Map 的相同映射的实例。*

TreeMap 类的构造函数

下表列出 TreeMap 类的常用方法。

方法	*描述*
void clear()	*用于从 TreeMap 删除所有映射。*
V get(Object key)	*用于从 TreeMap 获取指定键对象的值对象。*
V put(K key, V value)	*用于在 TreeMap 中添加具有指定值对象的指定键对象。*
V remove(Object o)	*用于从 TreeMap 中删除指定键对象的映射。*
int size()	*用于获取 TreeMap 中的键 - 值映射的数量。*

TreeMap 类的方法

思考在 TreeMap 集合中添加和删除对象的以下代码：

```
import java.util.TreeMap;

public class TreeMapDemo
{
public static void main(String[] args)
{
TreeMap<Integer, String> obj = new TreeMap<Integer, String>();
String sobj1 = new String("Value 1");
String sobj2 = new String("Value 2");
String sobj3 = new String("Value 3");
String sobj4 = new String("Value 4");

System.out.println("Size of TreeMap is:" + obj.size());
obj.put(101, sobj1);
obj.put(102, sobj2);
obj.put(103, sobj3);
obj.put(104, sobj4);
obj.put(105, sobj1);
System.out.println("\nTreeMap after adding the objects:" + obj);
System.out.println("Size of TreeMap after adding objects:" + obj.size());
obj.remove(103);
obj.remove(105);
System.out.println("\nTreeMap after removing the objects:" + obj);
System.out.println("Size of the TreeMap after removing objects:" +
obj.size());
}
}
```

执行上述代码时，会显示以下输出：

```
Size of TreeMap is:0

TreeMap after adding the objects:{101=Value 1, 102=Value 2, 103=Value 3,
104=Value 4, 105=Value 1}
Size of TreeMap after adding objects:5

TreeMap after removing the objects:{101=Value 1, 102=Value 2, 104=Value 4}
Size of the TreeMap after removing objects:3
```

在上述代码中，TreeMap 对象包含键 - 值对对象的集合。TreeMap 类的 put() 方法用于添加键 - 值对对象。并且使用 remove() 方法除去两个对象的映射。

使用 Hashtable 类

Hashtable 类使您能够创建无序对象集合，其中不能包含 null 对象。由于 Hashtable 类的方法是同步的，因此在多线程环境中给定时间点只能有一个线程访问该类。而另一方面，HashMap 和 TreeMap 类的方法不同步。

Hashtable 类提供可用于创建实例的多种构造函数。下表列出 Hashtable 类的常用构造函数。

构造函数	***描述***
Hashtable()	*创建具有空 Hashtable 的实例。*
Hashtable(int initialCapacity)	*创建具有指定初始容量的空 Hashtable 的实例。初始容量是 Hashtable 的大小，在 Hashtable 中添加或删除对象时该大小会自动增加或缩减。*
Hashtable(Map<? extends K, ? extends V> t)	*创建具有指定 Map 的相同映射的实例。*

Hashtable 类的构造函数

下表列出 Hashtable 类的常用方法。

方法	*描述*
void clear()	*用于从 Hashtable 删除所有对象。*
V get(Object key)	*用于从 Hashtable 获取指定键对象的值对象，如果它不包含指定键对象的任何值，获取 null。*
V put(K key, V value)	*用于在 Hashtable 中添加具有指定值对象的指定键对象。*
V remove(Object o)	*用于从 Hashtable 中删除指定键对象的映射。*
int size()	*用于获取 Hashtable 中的键的数量。*

Hashtable 类的方法

思考在 Hashtable 集合中添加和删除对象的以下代码：

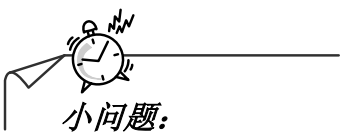
```
import java.util.Hashtable;

public class HashtableDemo
{
public static void main(String[] args)
{
Hashtable<Integer, Double> obj = new Hashtable<Integer, Double>();
Double dobj1 = new Double(55.6);
Double dobj2 = new Double(34.6);
Double dobj3 = new Double(98.6);
Double dobj4 = new Double(12.5);

System.out.println("Size of Hashtable is:" + obj.size());

obj.put(55, dobj1);
obj.put(60, dobj2);
obj.put(65, dobj3);
obj.put(70, dobj4);
obj.put(75, dobj3);

System.out.println("\nHashtable after adding the objects:" + obj);
System.out.println("Size of Hashtable after adding objects:" + obj.size());

obj.remove(65);
obj.remove(75);

System.out.println("\nHashtable after removing the objects:" + obj);
System.out.println("Size of Hashtable after removing objects:" + obj.size());
```

```
    }
    }
```

执行上述代码时，会显示以下输出：

```
Size of Hashtable is:0

Hashtable after adding the objects:{65=98.6, 75=98.6, 60=34.6, 70=12.5,
55=55.6}
Size of Hashtable after adding objects:5

Hashtable after removing the objects:{60=34.6, 70=12.5, 55=55.6}
Size of Hashtable after removing objects:3
```

在上述代码中，`Hashtable` 对象包含键 - 值对对象的集合。`Hashtable` 类的 `put()` 方法用于添加键 - 值对对象。并且使用 `remove()` 方法除去两个对象的映射。



小问题：

以下哪个类包含同步方法？

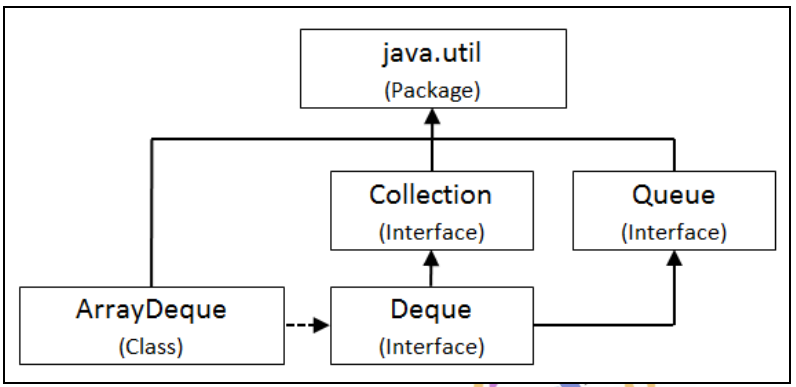
1. *Hashtable*
2. *HashSet*
3. *TreeMap*
4. *ArrayList*

答案：

1. *Hashtable*

使用 Deque 接口

Deque 接口允许创建可在两端插入或删除对象的对象集合。Deque 是双端队列的缩写。Deque 接口扩展 Queue 接口，后者提供在末端插入对象，在开头删除对象的集合。Queue 接口主要用于“先进先出”(FIFO) 算法。下图显示实现 Deque 接口的 java.util 包的类层次结构。



java.util 包的类层次接口

上图显示 java.util 包包含 ArrayDeque 类和 Collection、Queue 和 Deque 接口。ArrayDeque 类实现从 Collection 和 Queue 接口扩展的 Deque 接口。

使用 ArrayDeque 类

ArrayDeque 类提供 Deque 接口的实现以使您能够创建可调整大小的数组，允许在两端执行插入和删除操作。然而，由于 ArrayDeque 类的方法不是同步的，因此在多线程环境中给定时间点可以有多个线程访问该类。

ArrayDeque 类提供可用于创建实例的多种构造函数。下表列出 ArrayDeque 类的常用构造函数。

构造函数	***描述***
ArrayDeque()	*创建 ArrayDeque 类的空实例。*
ArrayDeque(Collection<? extends E> c)	*创建具有指定集合的实例。*
ArrayDeque(int numElement)	*创建具有可容纳指定数量元素的初始容量的 ArrayDeque 空实例。*

ArrayDeque 类的构造函数

下表列出 ArrayDeque 类的常用方法。

方法	*描述*
boolean add(E e)	*用于在 ArrayDeque 末尾添加指定对象。*
void addFirst(E e)	*用于将指定对象作为第一个元素添加到 ArrayDeque 中。*
void addLast(E e)	*用于将指定对象作为最后一个元素添加到 ArrayDeque 中。*
void clear()	*用于从 ArrayDeque 删除所有对象。*
E getFirst()	*用于检索 ArrayDeque 的第一个对象。*
E getLast()	*用于检索 ArrayDeque 的最后一个元素。*
E removeFirst()	*用于检索并删除 ArrayDeque 的第一个对象。*
E removeLast()	*用于检索并删除 ArrayDeque 的最后一个对象。*
int size()	*用于检索 ArrayDeque 中的对象数量。*

ArrayDeque 类的方法

思考在 ArrayDeque 集合中添加和删除对象的以下代码：

```
import java.util.ArrayDeque;

public class ArrayDequeDemo
{
    public static void main(String[] args)
    {
        ArrayDeque<Double> obj = new ArrayDeque<Double>();
        Double dobj1 = new Double(7.5);
        Double dobj2 = new Double(8.5);
        Double dobj3 = new Double(9.5);
        Double dobj4 = new Double(10.5);

        System.out.println("Size of ArrayDeque is:" + obj.size());

        obj.add(dobj1);
        obj.add(dobj2);

        System.out.println("\nArrayDeque after adding the objects:" + obj);
```

```
        System.out.println("Size of ArrayDeque after adding objects:" +
    obj.size());

        obj.addFirst(dobj3);
        obj.addLast(dobj4);

        System.out.println("\nArrayDeque after adding the objects at first
    and last:" + obj);
        System.out.println("Size of ArrayDeque after adding objects at first
    and last:" + obj.size());

        obj.removeFirst();

        System.out.println("\nArrayDeque after removing the first object:" +
    obj);
        System.out.println("Size of ArrayDeque after removing the first
    objects:" + obj.size());
    }
}
```

执行上述代码时，会显示以下输出：

```
Size of ArrayDeque is:0

ArrayDeque after adding the objects:[7.5, 8.5]
Size of ArrayDeque after adding objects:2

ArrayDeque after adding the objects at first and last:[9.5, 7.5, 8.5, 10.5]
Size of ArrayDeque after adding objects at first and last:4

ArrayDeque after removing the first object:[7.5, 8.5, 10.5]
Size of ArrayDeque after removing the first objects:3
```

在上述代码中，`ArrayDeque` 对象包含 `Double` 对象的集合。`ArrayDeque` 类的 `add()` 方法用于将 `Double` 对象添加到 `Deque` 对象。添加两个对象，一个添加在队列开头，一个添加在末尾。此外还使用 `remove()` 方法删除对象，使用 `removeFirst()` 方法删除第一个位置处存储的元素。

实现排序

思考一个地址簿的场景，其中包含联系人详细信息，例如姓名、联系号码和地址。现在需要创建一个居住在亚洲的联系人列表。此外，还需要确保该列表可以灵活地添加或删除任意数量的联系人。为此，您决定创建联系人集合。之后，需要根据联系人姓名对列表排序。为此，Java 提供了以下接口对集合中的对象排序：

- `Comparable` 接口
- `Comparator` 接口

`Comparable` 和 `Comparator` 接口提供自己的方法在集合上实现排序。然而，`Comparable` 和 `Comparator` 接口与 `Collections` 和 `Arrays` 类的 `sort()` 方法结合使用以对对象集合排序。

使用 Comparable 接口

`Comparable` 接口在 `java.lang` 包中定义，用于排序和比较对象集合。该接口提供用于比较对象引用的 `compareTo()` 方法。`compareTo()` 方法返回具有以下特征的 `int` 类型的值：

- **负值**：如果当前对象小于要比较的对象
- **零**：如果当前对象等于要比较的对象
- **正数**：如果当前对象大于要比较的对象

思考要存储学生姓名和分数的示例。之后，要根据分数对每个学生的详细信息排序。为此，您决定创建 `Student` 类，如以下代码所示：

```
public class Student implements Comparable
{
private String name;
private int marks;

public Student(String nm, int mk)
{
this.name = nm;
this.marks = mk;
}

public int getMarks()
{
return marks;
}
public String getName() {
return name;
}

public void setName(String name) {
this.name = name;
}
public int compareTo(Object obj)
{
Student s = (Student) obj;
```

```
if (this.marks > s.getMarks())
return 1;
else if (this.marks < s.getMarks())
return -1;
else
return 0;

}

public String toString()
{
StringBuffer buffer = new StringBuffer();
buffer.append("Name:"+ name +"\n");
buffer.append("Marks:" + marks + "\n");
return buffer.toString();
}

}
```

在上述代码中，创建 Student 类以重写 Comparable 接口的 compareTo() 方法。compareTo() 方法根据分数比较 Student 对象。此外，Object 类的 toString() 方法被重写，这样在 println() 方法内部使用 Student 类的对象时显示学生的姓名和分数。

之后，您可以使用以下代码创建 ComparableDemo 类：

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.ListIterator;

public class ComparableDemo
{
public static void main(String[] args)
{
Student s1 = new Student("Alex", 88);
Student s2 = new Student("Bob", 90);
Student s3 = new Student("Joe", 78);

ArrayList<Student> obj = new ArrayList<>();
obj.add(s1);
obj.add(s2);
obj.add(s3);

System.out.println("Student details are:");

ListIterator li = obj.listIterator();
while(li.hasNext())
{
System.out.println(li.next());
}

Collections.sort(obj);

System.out.println("Student details after sorting are:");
```

```
ListIterator li2 = obj.listIterator();
while(li2.hasNext())
{
System.out.println(li2.next());
}
}
}
```

执行上述代码时，会显示以下输出：

```
Student details after sorting are:
Name:Joe
Marks:78

Name:Alex
Marks:88

Name:Bob
Marks:90
```

在上述代码中，创建 ComparableDemo 类，以便通过 add() 方法添加对象来创建包含三个 Student 对象的数组列表。此外，使用 Collections 类的 sort() 方法对该数组列表排序。

使用 Comparator 接口

您已学习用于排序和比较对象集合的 Comparable 接口。但是，通过使用该接口，不能定义多重排序的逻辑。例如，在上述场景中，您想要根据学生分数和姓名创建排序逻辑。为此，您可以使用 java.util 包中定义的 Comparator 接口。该接口提供用于比较集合中两个对象的 compare() 方法。compare() 方法返回具有以下特征的 int 类型的值：

- **负值**：如果当前对象小于要比较的对象
- **零**：如果当前对象等于要比较的对象
- **正数**：如果当前对象大于要比较的对象

思考定义 Student 类的以下代码：

```
public class Student
{
private String name;
private int marks;

public Student(String nm, int mk)
{
this.name = nm;
this.marks = mk;
}
public int getMarks()
{
return marks;
}
public String getName() {
return name;
```

```
}

public void setName(String name) {
this.name = name;
}

public String toString()
{
StringBuffer buffer = new StringBuffer();
buffer.append("Name:"+ name +"\n");
buffer.append("Marks:" + marks + "\n");
return buffer.toString();
}

}
```

在上述代码中，name 和 marks 是 Student 类的属性。此外，还为这两个属性定义了 setter 和 getter 方法。并且重写 Object 类的 toString() 方法。

思考根据学生分数定义排序逻辑的以下代码：

```
import java.util.Comparator;
public class MarkCompare implements Comparator{
public int compare(Object a, Object b)
{
Student x = (Student) a;
Student y = (Student) b;

if (x.getMarks() > y.getMarks())
return 1;
else if (x.getMarks() < y.getMarks())
return -1;
else
return 0;
}
}
```

在上述代码中，Comparator 接口的 compare() 方法被重写，并定义根据学生分数对 student 对象排序的逻辑。

思考根据学生姓名定义排序逻辑的以下代码：

```
import java.util.Comparator;
public class NameCompare implements Comparator{
public int compare(Object a, Object b)
{
Student x = (Student) a;
Student y = (Student) b;
return x.getName().compareTo(y.getName());
}
}
```

在上述代码中，Comparator 接口的 compare() 方法被重写，并定义根据学生姓名对 student 对象排序的逻辑。由于学生姓名是 String 对象，因此使用 String 类的 compare() 方法实现该排序逻辑。

之后，您可以使用以下代码创建 ComparatorDemo 类：

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.ListIterator;

public class ComparatorDemo
{
public static void main(String[] args)
{
Student s1 = new Student("Alex", 88);
Student s2 = new Student("Bob", 90);
Student s3 = new Student("Joe", 78);

ArrayList<Student> obj = new ArrayList<>();
obj.add(s1);
obj.add(s2);
obj.add(s3);

System.out.println("Student details are:");

ListIterator li = obj.listIterator();
while(li.hasNext())
{
System.out.println(li.next());
}

Collections.sort(obj, new MarkCompare());

System.out.println("Mark wise sort:");

ListIterator li2 = obj.listIterator();
while(li2.hasNext())
{
System.out.println(li2.next());
}
Collections.sort(obj, new NameCompare());

System.out.println("Name wise sort:");

ListIterator li3 = obj.listIterator();
while(li3.hasNext())
{
System.out.println(li3.next());
}
}
}
```

执行上述代码时，会显示以下输出：

```
Student details are:
Name:Alex
Marks:88

Name:Bob
Marks:90

Name:Joe
Marks:78

Mark wise sort:
Name:Joe
Marks:78

Name:Alex
Marks:88

Name:Bob
Marks:90

Name wise sort:
Name:Alex
Marks:88

Name:Bob
Marks:90

Name:Joe
Marks:78
```

在上述代码中，创建 `ComparatorDemo` 类，以便通过 `add()` 方法添加对象来创建包含三个 `Student` 对象的数组列表。此外，使用 `Collections` 类的 `sort()` 方法对该数组列表排序。

活动 4.1：使用集合

练习问题

1. 以下哪个类实现 `List` 接口？
   a. `HashSet`
   b. `TreeSet`
   c. `Hashtable`
   d. `Vector`

2. 以下哪个类是从 `Dictionary` 类继承的？
   a. `HashSet`
   b. `TreeSet`
   c. `Hashtable`
   d. `HashMap`

3. 以下哪个接口使您能够遍历对象列表？
   a. `Iterator`
   b. `List`
   c. `Collection`
   d. `Comparable`

4. 以下哪个方法用于在 `TreeMap` 中添加对象？
   a. `put()`
   b. `add()`
   c. `addObject()`
   d. `putObject()`

5. 以下哪个接口用于创建唯一对象集合？
   a. `Set`
   b. `Map`
   c. `Deque`
   d. `List`

小结

在本章中，您学习了：

- Set 接口用于创建唯一对象集合。
- 包 java.util 提供 Iterator 接口以遍历 set 集合。
- HashSet 类提供 Set 接口的实现以使您能够创建一个可在其中快速插入的集合，因为它不对元素进行排序。
- TreeSet 类提供使您能够创建有序对象集合的 Set 接口的实现。
- List 接口用于创建有序对象集合，其中可以包含重复对象。
- 包 java.util 提供遍历 list 集合的 ListIterator 接口。
- ArrayList 类提供 List 接口的实现。ArrayList 类使您能够创建可调整大小的数组。
- LinkedList 类提供 List 接口的实现。LinkedList 类使您能够创建双向链表。
- LinkedList 可以向前或向后遍历。
- Vector 类类似于 ArrayList 和 LinkedList 类。
- 由于 Vector 类的方法是同步的，因此在多线程环境中给定时间点只能有一个线程访问该类。
- ArrayList 和 LinkedList 类的方法不同步。
- Map 接口使您能够创建具有键 - 值对对象的集合。
- Map 接口允许存在重复值对象，但键对象必须唯一。
- HashMap 类使您能够创建可使用键访问值的集合。HashMap 按无序形式存储对象。
- TreeMap 类使您能够使用唯一键按有序顺序创建对象集合。
- Hashtable 类使您能够创建无序对象集合，其中不能包含 null 对象。
- 由于 Hashtable 类的方法是同步的，因此在多线程环境中给定时间点只能有一个线程访问该类。
- HashMap 和 TreeMap 类的方法不同步。
- Deque 接口允许创建可在两端插入或删除对象的对象集合。
- ArrayDeque 类提供 Deque 接口的实现以使您能够创建可调整大小的数组，允许在两端执行插入和删除操作。
- 由于 ArrayDeque 类的方法不是同步的，因此在多线程环境中给定时间点可以有多个线程访问该类。
- Java 提供了以下接口对集合中的对象排序：
  - Comparable 接口
  - Comparator 接口
- Comparable 接口在 java.lang 包中定义，用于排序和比较对象集合。该接口提供用于比较对象引用的 compareTo() 方法。
- Comparator 接口在 java.util 包中定义。该接口提供用于比较集合中两个对象的 compare() 方法。