福建師範大學
FUJIAN NORMAL UNIVERSITY

Java语言

学　　期：2025-2026-1

教　　师：赵珊珊

第8章集合类

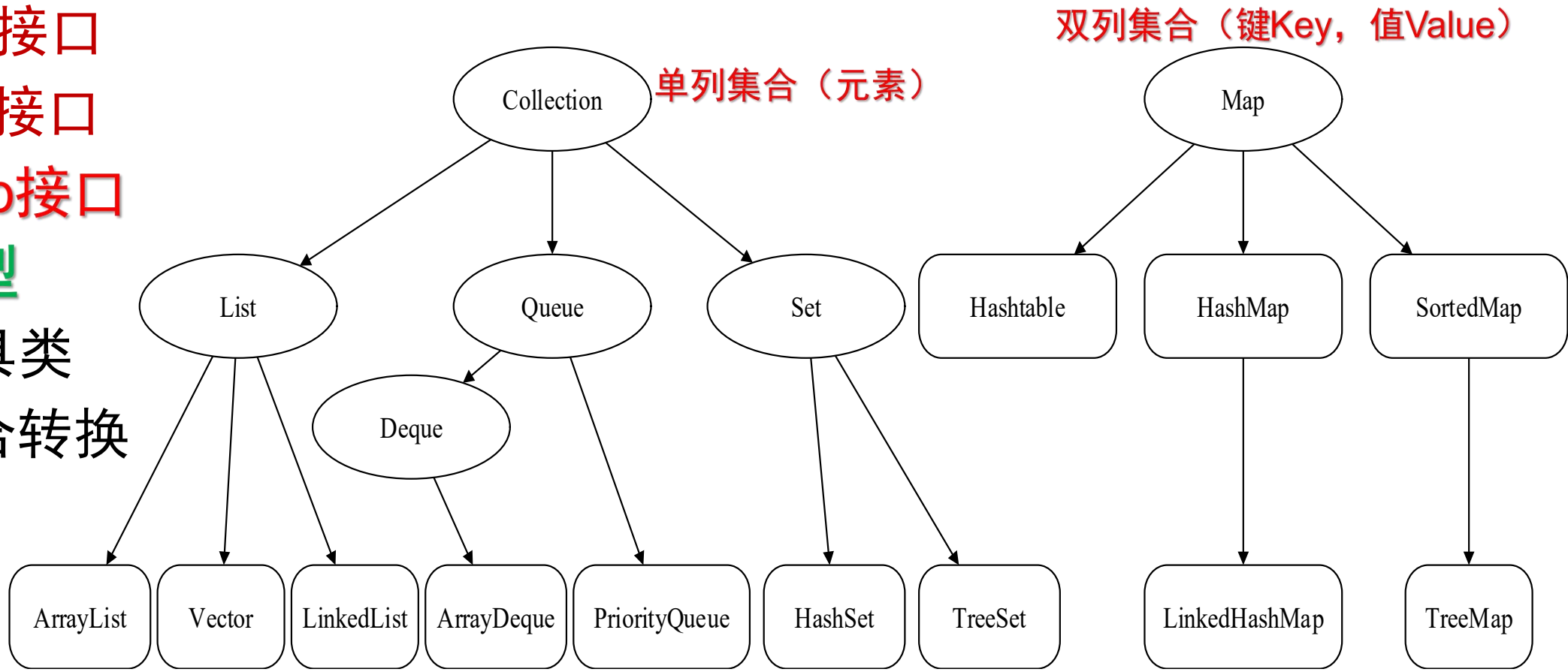
第8章集合类

可以存储任意类型的对象，并且长度可变的类，统称为集合。

集合类的功能：添加对象、删除对象、清空容器、判断容器是否为空等。

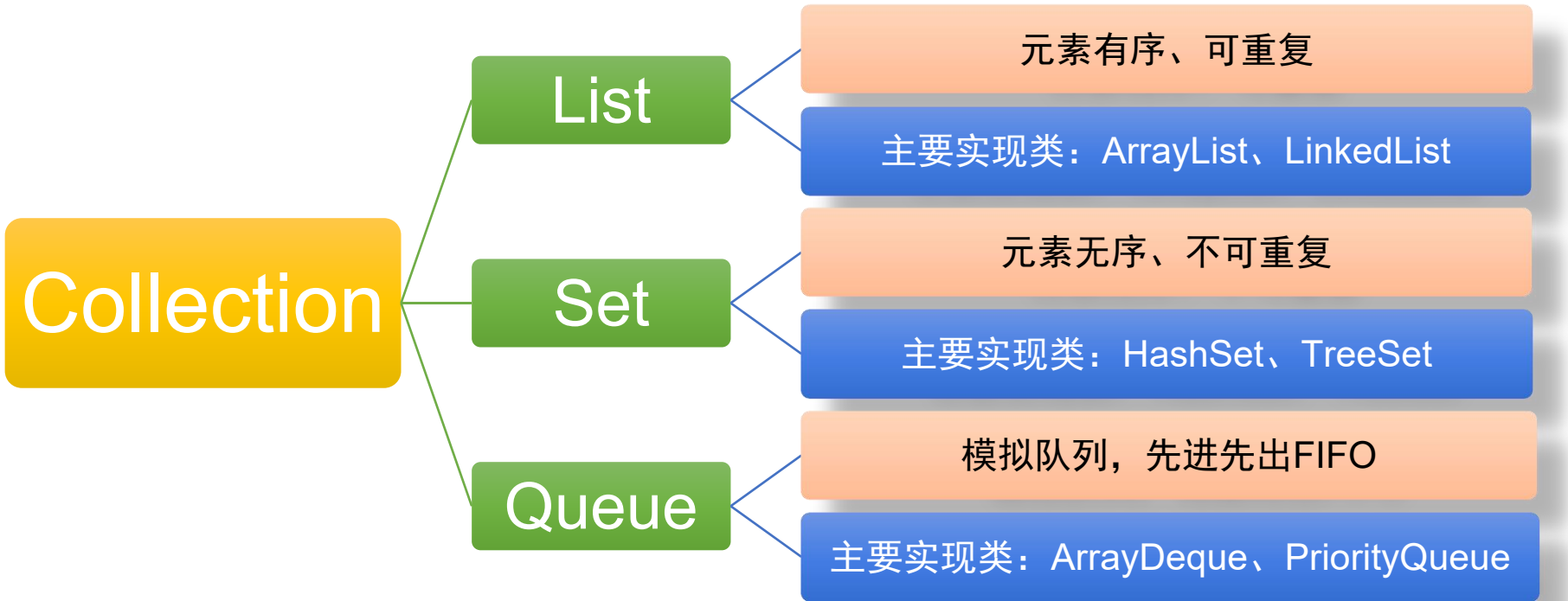
- 8.1Collection接口
- 8.2List接口
- 8.3Set接口
- 8.4Map接口
- 8.5泛型
- 8.6工具类
- 8.7集合转换

java.util包提供了一系列可使用的集合类，称为集合框架。如下图所示：



8.1Collection接口

- Collction 接口是单列集合的根接口，用于存储一系列符合某种规则的元素。
- Collction是List、Set和Queue等接口的父接口，该接口里定义的方法既可用于操作List集合，也可用于操作Set和Queue集合。



8.1Collection接口

方法	说明
boolean add(Object obj)	添加指定的元素。添加成功返回 true。
boolean addAll(Collection c)	将指定集合中的所有元素添加到此集合中。
void clear()	移除此集合中的所有元素。
int hashCode()	返回此集合的哈希码值。
boolean isEmpty()	如果此集合不包含任何元素，则返回 true。
boolean contains(Object o)	如果此集合包含指定的元素，则返回 true。
boolean containsAll(Collection c)	如果此集合包含指定集合中的所有元素，则返回 true。
boolean equals(Object o)	比较此集合与指定对象是否相等。
boolean remove(Object o)	从此集合中移除指定元素的单个实例（如果存在）。
boolean removeAll(Collection c)	移除此集合的所有包含在指定集合中的元素。
boolean retainAll(Collectio c)	仅保留此集合中包含在指定集合中的元素。
Iterator iterator()	返回在此集合的元素上进行迭代的迭代器。
int size()	返回此集合中的元素数。
Object[] toArray()	返回一个包含此集合中所有元素的数组。
Stream stream()	返回以此集合作为源的顺序流（JDK 8 新增）。
Stream parallelStream()	返回以此集合作为源的可能并行的流（JDK 8 新增）。
void forEach(Consumer action)	对此集合的每个元素执行给定的操作（继承自 Iterable，常结合 Lambda 使用）。

List 添加元素总返回 true， Set 中若元素已存在则返回 false

元素相等性判断：
contains(), remove(), containsAll(), removeAll(), retainAll()等方法依赖于equals()方法来判断对象是否相等。

8.1Collection接口

```
目标1的大小为：6   [地科院楼, 地科院楼, 1000.0, 117.5, 27.22, 12]
目标2的大小为：5   [地科院楼, 27.22, 12, 117.5, 1000.0]
```

```
import java.util.*;
public class CollectionExam {
    public static void main(String[] args) {
        Collection c1 = new ArrayList();
        Collection c2 = new HashSet();
        c1.add("地科院楼");
        c1.add("地科院楼");
        c1.add(1000.0);c1.add(117.5);c1.add(27.22);c1.add(12);
        System.out.println("目标1的大小为："+c1.size()+"   "+c1);
        c2.add("地科院楼");
        c2.add("地科院楼");//HashSet元素不可重复，添加失败
        c2.add(1000.0);c2.add(117.5);c2.add(27.22);c2.add(12);
        System.out.println("目标2的大小为："+c2.size()+"   "+c2);//HashSet元素无序
    }
}
```

8.2List接口

- List接口继承了Collection接口，是单列集合的一个分支，实现了List接口的对象称为List集合。

在List集合中允许出现重复的元素，所有的元素是以*一种线性方式进行存储的*，在程序中*可以通过索引来访问*集合中的指定元素。另外，List集合还有一个特点就是元素有序，即元素的存入顺序和取出顺序一致。

方法	说明
boolean add(Object e)	将元素 e**追加**到列表末尾
void add(int index, Object element)	在**指定索引** index 处插入元素 element
boolean addAll(Collection c)	将集合 c中的所有元素**追加**到此列表末尾
boolean addAll(int index, Collection c)	从**指定索引** index 开始，将集合 c中的所有元素插入此列表
Object get(int index)	返回列表中**指定索引** index 处的元素
Object set(int index, Object element)	将**指定索引** index 处的元素**替换**为 element
Object remove(int index)	**移除**指定索引 index处的元素，并返回被移除的元素
boolean remove(Object o)	移除列表中**第一次出现**的指定元素 o（如果存在）
int indexOf(Object o)	返回元素 o在列表中**第一次出现**的索引，未找到则返回 -1
int lastIndexOf(Object o)	返回元素 o在列表中**最后一次出现**的索引，未找到则返回 -1
int size()	返回列表中元素的**数量**
boolean isEmpty()	判断列表是否**为空**
boolean contains(Object o)	判断列表是否**包含**指定元素 o
void clear()	**移除**列表中的所有元素
Object[] toArray()	将列表转换为**数组**

8.2List接口

- 8.2.1ArrayList集合　　存储结构为数组，查询快，增删慢。
- ArrayList是List接口的一个实现类，它是程序中最常见的一种集合。在ArrayList内部封装了一个长度可变的数组对象，当存入的元素超过数组长度时，ArrayList会在内存中分配一个更大的数组来存储这些元素，因此可以将ArrayList集合看作一个长度可变的数组。

```
import java.util.*;
public class ArrayListExam {
    public static void main(String[] args) {
        ArrayList list = new ArrayList();
        list.add("Hello");
        list.add("Hello Echo");
        list.add(123); // 编译器不会报错，但这是一个Integer
        System.out.println(list.size()+" "+list.get(1));
        for (Object obj : list) {
            // 当遇到123这个Integer对象时，强制转换就会失败，抛出ClassCastException
            String str = (String) obj; // Runtime Error: ClassCastException
            System.out.println(str.length());
        }
    }
}
```

```
3 Hello Echo
5
10
Exception in thread "main" java.lang.ClassCastException: class java.lang.Integer cannot be cast to class java.lang.String
```

8.2List接口

- 8.2.2LinkedList集合
- ArrayList集合在查询元素时速度很快，但在删除元素时效率很低，为了克服这种局限性，可以使用List接口的另外一个实现类LinkedList。
- 该集合内部包含了两个Node类型的first和last属性维护一个*双向循环链表*，链表中的每一个元素都使用引用的方式来记住它的前一个和后一个元素，从而可以将所有的元素连接起来。
- 常用方法：add(int index,Object o)
- void addFirst(Object o) void addLast(Object o)
- (Object) getFirst() getLast() removeFirst() removeLast()

8.2List接口

```
头部插入 100000 个元素耗时：
ArrayList: 388 毫秒
LinkedList: 5 毫秒
LinkedList 比 ArrayList 快约 77 倍

头部删除 100000 个元素耗时：
ArrayList: 400 毫秒
LinkedList: 3 毫秒
LinkedList 比 ArrayList 快约 133 倍
```

```java
import java.util.*;
public class ListEfficiencyTest {
    private static final int OPERATION_COUNT = 100000;     // 测试的数据量
    public static void main(String[] args) {
        // 分别创建ArrayList和LinkedList
        List arrayList = new ArrayList();  List linkedList = new LinkedList();
        // 测试头部插入效率
        System.out.println("头部插入 " + OPERATION_COUNT + " 个元素耗时：");
        long arrayListInsertTime = insertAtHead(arrayList, OPERATION_COUNT);
        long linkedListInsertTime = insertAtHead(linkedList, OPERATION_COUNT);
        System.out.println("ArrayList: " + arrayListInsertTime + " 毫秒");
        System.out.println("LinkedList: " + linkedListInsertTime + " 毫秒");
        System.out.println("LinkedList 比 ArrayList 快约 " + (arrayListInsertTime / linkedListInsertTime) + " 倍");
        // 清空列表，进行下一步测试
        arrayList.clear(); linkedList.clear();
        // 填充列表，准备测试头部删除效率
        fillList(arrayList, OPERATION_COUNT);   fillList(linkedList, OPERATION_COUNT);
        System.out.println("\n头部删除 " + OPERATION_COUNT + " 个元素耗时：");
        long arrayListRemoveTime = removeFromHead(arrayList); long linkedListRemoveTime = removeFromHead(linkedList);
        System.out.println("ArrayList: " + arrayListRemoveTime + " 毫秒");
        System.out.println("LinkedList: " + linkedListRemoveTime + " 毫秒");
        System.out.println("LinkedList 比 ArrayList 快约 " + (arrayListRemoveTime / linkedListRemoveTime) + " 倍");
    }
    // 测试头部插入耗时
    private static long insertAtHead(List<Integer> list, int count) {
        long startTime = System.currentTimeMillis();
        for (int i = 0; i < count; i++)
            list.add(0, i); // 在头部插入
        return System.currentTimeMillis() - startTime;
    }
    // 测试头部删除耗时
    private static long removeFromHead(List<Integer> list) {
        long startTime = System.currentTimeMillis();
        while (!list.isEmpty())
            list.remove(0); // 删除头部元素
        return System.currentTimeMillis() - startTime;
    }
    // 填充列表用于删除测试
    private static void fillList(List<Integer> list, int count) {
        for (int i = 0; i < count; i++)
            list.add(i);
    }
}
```

8.2List接口

- 8.2.3Iterator接口
- Iterator接口是Java集合框架中的一员，但它与Collection、Map接口有所不同，Collection接口和Map接口主要用于存储元素，而Iterator主要用于迭代访问（遍历）Collection中的元素，因此Iterator对象也称为**迭代器**。

```
import java.util.*;
public class IteratorExam {
    public static void main(String[] args) {

        ArrayList fruits = new ArrayList();
        fruits.add("Apple"); fruits.add("Banana"); fruits.add("Cherry");

        Iterator iterator = fruits.iterator();
        while (iterator.hasNext()) { // 检查是否还有下一个元素
            System.out.println(iterator.next());// 获取下一个元素
        }
    }
}
```

```
Apple
Banana
Cherry
```

8.2List接口

- 8.2.4ListIterator接口
- List接口额外提供了一个listIterator()方法，该方法返回一个ListIterator对象， ListIterator接口继承了Iterator接口，提供了一些用于操作List的方法

8.2List接口

```
向后遍历:
当前元素: Apple, 下一个索引: 1
当前元素: Banana, 下一个索引: 2
当前元素: Cherry, 下一个索引: 3

向前遍历:
当前元素: Cherry, 上一个索引: 1
当前元素: Banana, 上一个索引: 0
当前元素: Apple, 上一个索引: -1

在'Banana'后添加'Orange': [Apple, Banana, Orange, Cherry]
替换'Cherry'为'Cheery Fruit': [Apple, Banana, Orange, Cheery Fruit]
删除'Banana': [Apple, Orange, Cheery Fruit]
```

```java
import java.util.*;
public class ListIteratorExample {
    public static void main(String[] args) {
        ArrayList list = new ArrayList();
        list.add("Apple");     list.add("Banana");     list.add("Cherry");
        ListIterator iterator = list.listIterator();
        System.out.println("向后遍历:");
        while (iterator.hasNext()) {
            Object element = iterator.next();
            System.out.println("当前元素: " + element + ", 下一个索引: " + iterator.nextIndex());
        }

        System.out.println("\n向前遍历:");
        while (iterator.hasPrevious()) {
            Object element = iterator.previous();
            System.out.println("当前元素: " + element + ", 上一个索引: " + iterator.previousIndex());
        }

        iterator = list.listIterator();       // 重置迭代器到开头
        while (iterator.hasNext()) {
            Object element = iterator.next();
            if (element.equals("Banana"))
                iterator.add("Orange"); // 在 "Banana" 之后插入 "Orange"
        }
        System.out.println("\n在'Banana'后添加'Orange': " + list);

        iterator = list.listIterator();// 获取一个元素后才能替换
        while (iterator.hasNext()) {
            Object element = iterator.next();
            if (element.equals("Cherry")) {
                iterator.set("Cheery Fruit"); // 将 "Cherry" 替换为 "Cheery Fruit"
            }
        }
        System.out.println("替换'Cherry'为'Cheery Fruit': " + list);

        iterator = list.listIterator();
        while (iterator.hasNext()) {
            Object element = iterator.next();
            if (element.equals("Banana"))
                iterator.remove(); // 删除 "Banana"
        }
        System.out.println("删除'Banana': " + list);
    }
}
```

8.2List接口

- 8.2.5foreach循环
- foreach循环是一种更加简洁的for循环，也称增强for循环。
  foreach循环用于遍历数组或集合中的元素，其语法格式如下：

  for(容器内元素类型 临时变量:容器变量){
  　　语句
  }

```
import java.util.*;
public class ForeachExam {
    public static void main(String[] args) {
        ArrayList list = new ArrayList();
        list.add("地科院");
        list.add("教学楼");
        list.add("地下停车场");
        for(Object o:list)
            System.out.println(o);
    }
}
```

```
<terminated>
地科院
教学楼
地下停车场
```

访问代码简洁，但不能对元素进行修改。

8.3Set接口

- Set集合中元素是无序的、不可重复的。Set接口也是继承自Collection接口，但它没有对Collection接口的方法进行扩充。
- Set中元素有无序性的特点，这里要注意，**无序性不等于随机性**，无序性指的是元素在底层存储位置是无序的。Set接口的主要实现类是HashSet和TreeSet。

8.3Set接口

- 8.3.1HashSet集合
- HashSet类是Set接口的一个实现类，它所存储的元素是*不可重复*的，并且元素都是无序的。
- HashSet按哈希算法来存储集合中的元素，当向HashSet集合中添加一个元素时，首先会调用元素的hashCode()方法来确定元素的存储位置，然后再调用元素对象的equals()方法来确保该位置没有重复元素。
- 另外，集合中的元素可以为null。

哈希算法是一种将任意长度的数据映射为固定长度二进制串（通常是一串十六进制数字）的算法。这个生成的固定长度串就是**哈希值**（Hash Value），也可以叫**摘要**（Digest）或**指纹**（Fingerprint）。

8.3Set接口

- 8.3.1HashSet集合

```
import java.util.*;
public class HashSetExam {
    public static void main(String[] args) {
        HashSet ha=new HashSet();
        ha.add(new Construct("地科院",11));
        ha.add(new Construct("花香园",4));
        ha.add(new Construct("花香园",4));
        for(Object o:ha)
            System.out.println(o);
    }
}
class Construct{
    String ObjName;
    int ObjNO;
    public Construct(String ona,int ono) {
        this.ObjName=ona;
        this.ObjNO=ono;
    }
    public String toString() {
        return "地理目标："+this.ObjName+"  编号："+this.ObjNO;
    }
}
```

```
地理目标：地科院  编号：11
地理目标：花香园  编号：4
地理目标：花香园  编号：4
```

重写hashCode()和equals()方法

```
public int hashCode() {
    final int prime=31; int result = 1;
    result=prime*result+this.ObjNO;
    result=prime*result+((this.ObjName==null)?0:this.ObjName.hashCode());
    return result;
}
public boolean equals(Object obj) {
    if(this==obj)
        return true;
    if(obj==null)
        return false;
    if(getClass()!=obj.getClass())
        return false;
    Construct c=(Construct) obj;
    if(this.ObjNO!=c.ObjNO)
        return false;
    if(this.ObjName==null) {
        if(c.ObjName!=null)
            return false;
    }else if(!ObjName.equals(c.ObjName))
        return false;
    return true;
}
```

```
地理目标：地科院  编号：11
地理目标：花香园  编号：4
```

8.3Set接口

- 8.3.2TreeSet集合
- TreeSet类是Set接口的另一个实现类，TreeSet集合和HashSet集合都可以保证容器内元素的唯一性，但它们底层实现方式不同，TreeSet底层是用自平衡的排序二叉树实现的，所以它既能保证元素唯一性，又可以对元素进行排序。

方法	说明
Comparator comparator()	返回用于排序的比较器对象。如果 TreeSet使用自然排序，则返回 null
Object first()	返回集合中**当前排序下的第一个（最小）**元素
Object last()	返回集合中**当前排序下的最后一个（最大）**元素
Object lower(Object e)	返回集合中**小于** e的**最大元素**
Object higher(Object e)	返回集合中**大于**e的**最小元素**
SortedSet subSet(Object o1, Object o2)	返回此集合的子集，范围从 o1（**包含**）到 o2（**不包含**）
SortedSet headSet(Object toElement)	返回此集合中**格小于** toElement的所有元素的子集
SortedSet tailSet(Object fromElement)	返回此集合中**大于等于** fromElement的所有元素的子集

8.3Set接口

- 8.3.2TreeSet集合
- TreeSet有两种排序方法，自然排序和定制排序，默认情况下，TreeSet采用自然排序。

```
71 24 69 36 94 96 67 91 68 73
[24, 36, 67, 68, 69, 71, 73, 91, 94, 96]
```

```java
import java.util.*;
public class TreeSetExam {
    public static void main(String[] args) {
        TreeSet t=new TreeSet();
        for(int i=0;i<10;i++) {
            int n =(int)(Math.random() * 100);
            t.add(n);
            System.out.print(n+" ");
        }
        System.out.println("\n"+t);
    }
}
```

思考：100改为10，结果会是10个顺序数吗？

8.3Set接口

- 8.3.2TreeSet集合
- 1.自然排序
- TreeSet类会调用集合元素的compareTo(Object obj)方法来比较元素之间的大小关系，然后将集合内元素按升序排序，这就是自然排序。
- 添加进TreeSet集合的对象必须实现Comparable接口，且必须来自同一个类，否则会抛出ClassCastExeption。

```java
import java.util.*;
public class TreeSetExam {
    public static void main(String[] args) {
        TreeSet t=new TreeSet();
        t.add(new Const(10001,1)); 改成100?
        t.add(new Const(102,2));
        System.out.println(t);
    }
}
class Const implements Comparable{
    int no,NO;
    public Const(int no,int NO) {
        this.NO=NO;this.no=no;
    }
    public int compareTo(Object o) {
        return 1;
    }
}
```

```
[Const@cac736f, Const@5e265ba4]
```

```
[Const@cac736f, Const@5e265ba4]
```

8.3Set接口

- 8.3.2TreeSet集合
- 2.定制排序
- TreeSet的自然排序是根据集合元素大小，按升序排序，如果需要按特殊规则排序或者元素自身不具备比较性时，就需要用到定制排序，比如按降序排列。Comparable接口包含一个int compare(T t1,T t2)方法，该方法可以比较t1和t2大小，若返回正整数，则说明t1大于t2，若返回0，则说明t1等于t2，若返回负整数，则说明t1小于t2。

8.3Set接口

- 8.3.2TreeSet集合

```
[张三 202380 地科院 13, 王五 203945 花香园 16, 李四 203902 星广场 47]
```

```java
import java.util.*;
public class TreeSetExam {
    public static void main(String[] args) {
        TreeSet t=new TreeSet();
        t.add(new Const("张三","202380","地科院",13));
        t.add(new Const("李四","203902","星广场",47));
        t.add(new Const("王五","203945","花香园",16));
        System.out.println(t);
    }
}
class Const implements Comparable{
    String StuName,StuNO,ObjName;
    int ObjNO;
    public Const(String sna,String sno,String ona,int ono) {
        this.StuName=sna;
        this.StuNO=sno;
        this.ObjName=ona;
        this.ObjNO=ono;
    }
    public int compareTo(Object o) {
        Const c=(Const) o;
        if(this.ObjNO!=c.ObjNO)
            return this.ObjNO-c.ObjNO;
        else
            return this.ObjName.compareTo(c.ObjName);
    }
    public String toString() {
        return this.StuName+" "+this.StuNO+" "+this.ObjName+" "+this.ObjNO;
    }
}
```

8.4Map接口

- Map接口不是继承自Collection接口，它与Collection接口是并列存在的，用于存储键—值对(key-value)形式的元素，描述了由不重复的键到值的映射。
- Map接口是一种双列集合，它的每个元素都包含一个键对象Key和值对象Value，键和值对象之间存在一种对应关系，称为映射。Map中的key（键）和value（值）可以是任何引用类型的数据。Map 中的key用Set来存放，不允许重复，即同一个Map对象所对应的类，必须重写hashCode()方法和equals()方法。

注意表8-6：Map和Collection相同功能方法名的不同。

8.4Map接口

- 8.4.1HashMap集合
- HashMap集合是Map接口的一个实现类，它用于存储键值映射关系，该集合的键和值允许为空，但键不能重复，且集合中的元素是无序的。
- HashMap集合判断两个key相等的标准是：两个key通过equals()方法返回true，hashCode值也相等。HashMap集合判断两个value相等的标准是：两个value通过equals()方法返回true。

8.4Map接口

```
7
{0=李四, null=null, 1=张三, 2=孙七, 3=赵六, 4=周八, 5=王五}
周八 null
{0=李四, null=null, 1=孙二, 2=孙七, 3=赵六, 4=周八, 5=王五}
0:李四 null:null 1:孙二 2:孙七 3:赵六 4:周八 5:王五
0:李四.null:null.1:孙二.2:孙七.3:赵六.4:周八.5:王五.
```

```java
import java.util.*;
public class HashMapExam {
    static String name[]= {"张三","李四","王五","赵六","孙七","周八"};
    static String no[]= {"1","0","5","3","2","4"};
    public static void main(String[] args) {
        Map m=new HashMap();
        for(int i=0;i<name.length;i++)
            m.put(no[i], name[i]);
        m.put(null, null);//可以添加key和value都为null
        System.out.println(m.size()+"\n"+m);
        System.out.println(m.get("4")+" "+m.get(4));//"4"正确key,4无key
        m.put("1", "孙二");//key相同，value被覆盖
        System.out.println(m);
        Iterator it=m.keySet().iterator();//keySet实例
        while(it.hasNext()) {
            Object key=it.next();
            Object value=m.get(key);
            System.out.print(key+":"+value+" ");
        }
        System.out.println();
        Set s = m.entrySet();//entrySet实例
        it = s.iterator();
        while(it.hasNext()) {
            Map.Entry en = (Map.Entry) it.next();
            Object key = en.getKey();
            Object value = en.getValue();
            System.out.print(key+":"+value+".");
        }
    }
}
```

8.4Map接口

- 8.4.2LinkedHashMap集合
- LinkedHashMap类是HashMap的子类，LinkedHashMap类可以维护Map的迭代顺序，迭代顺序与键值对的插入顺序一致，如果需要输出的顺序与输入时的顺序相同，那么就选用LinkedHashMap集合。

```
1:张三 0:李四 5:王五 3:赵六 2:孙七 4:周八
```

```java
import java.util.*;

public class LinkedHashMapExam {
    static String name[]= {"张三","李四","王五","赵六","孙七","周八"};
    static String no[]= {"1","0","5","3","2","4"};
    public static void main(String[] args) {
        Map m=new LinkedHashMap();
        for(int i=0;i<name.length;i++)
            m.put(no[i], name[i]);
        Iterator it = m.entrySet().iterator();
        while(it.hasNext()) {
            Map.Entry en = (Map.Entry) it.next();
            Object key = en.getKey();
            Object value = en.getValue();
            System.out.print(key+":"+value+" ");
        }
    }

}
```

8.4Map接口

- 8.4.3TreeMap集合
- Java中Map接口还有一个常用的实现类TreeMap类，它也是用来存储键值映射关系的，并且不允许出现重复的键。TreeMap集合存储键—值对时，需要根据键—值对进行排序。TreeMap集合可以保证所有的键值对处于有序状态。

```
import java.util.*;
public class TreeMapExam {
    static String name[]= {"张三","李四","王五","赵六","孙七","周八"};
    static String no[]= {"1","0","5","3","2","4"};
    public static void main(String[] args) {
        Map m=new TreeMap(); new MyComparator()
        for(int i=0;i<name.length;i++)
            m.put(no[i], name[i]);
        Iterator it = m.entrySet().iterator();
        while(it.hasNext()) {
            Map.Entry en = (Map.Entry) it.next();
            Object key = en.getKey();
            Object value = en.getValue();
            System.out.print(key+":"+value+" ");
        }
    }
}
```

```
<terminated> TreeMapExam [Java Application] D
0:李四 1:张三 2:孙七 3:赵六 4:周八 5:王五
```

```
class MyComparator implements Comparator{//逆序
    public int compare(Object o1,Object o2) {
        Integer i1=Integer.parseInt(o1.toString());
        Integer i2=Integer.parseInt(o2.toString());
        return i2.compareTo(i1);
    }
}
```

```
5:王五 4:周八 3:赵六 2:孙七 1:张三 0:李四
```

8.5泛型

- 程序中有时会无法确定一个集合的元素的类型，取出元素时容易发生强制类型转换错误。
- 泛型是JDK新加入的特性，解决了数据类型的安全性问题，其主要原理是在类声明时通过一个标识，表示类中某个属性的类型或者是某个方法的返回值及参数类型。这样在类声明或实例化时只要指定好需要的具体类型即可。

8.5泛型

Multiple markers at this line
- ArrayList is a raw type. References to generic type ArrayList<E> should be parameterized
- ArrayList is a raw type. References to generic type ArrayList<E> should be parameterized

- 1.泛型定义
- 泛型在定义集合类时，使用“<参数化类型>”的方式指定该集合中方法操作的数据类型，具体示例如下：

  ArrayList<参数化类型> list=new

  ArrayList<参数化类型>();

[教学楼，学院楼，食堂]

```java
import java.util.*;
public class ArrayListExam {
    public static void main(String[] args) {
        ArrayList list = new ArrayList();
        list.add("教学楼");
        list.add("学院楼");
        list.add("食堂");
        System.out.println(list);
    }
}
```

```java
import java.util.*;
public class GenericExam {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<String>();
        list.add("教学楼");
        list.add("学院楼");
        list.add("食堂");
        System.out.println(list);
    }
}
```

8.5泛型

- 2.通配符
- 在讲解了泛型的定义后，这里要引入一个通配符的概念，类型通配符用符号“?”表示，比如List<?>，它是List<String>、List<Object>等各种泛型List的父类。
- 通配符主要用于增加代码的灵活性，同时尽可能保证类型安全，元素类型只能是null。

```
import java.util.*;
public class GenericExam {
    public static void main(String[] args) {
        List<?> l = null;
        List<Integer> li=new ArrayList<Integer>();
        List<String> ls= new ArrayList<String>();
        li.add(1);
        ls.add("张三 2383948204");
        read(li);
        read(ls);
    }
    static void read(List<?> list) {
        for (Object o:list)
            System.out.print(o+" ");
    }
}
```

```
1 张三 2383948204
```

8.5泛型

- 3.有界类型
- 上一节中讲解了利用通配符“?”来声明泛型类型，Java还提供了有界类型，可以创建声明超类的上界和声明子类的下界。

PECS（**P**roducer **E**xtends, **C**onsumer **S**uper）是有效使用通配符的重要原则：

•**生产者（Producer）使用** extends：当你需要一个泛型对象来**提供数据**（读取）时，使用 <? extends T>。

•**消费者（Consumer）使用** super：当你需要一个泛型对象来**接受数据**（写入）时，使用 <? super T>。

```
import java.util.*;
public class GenericPECS {
    public static void main(String[] args) {
        List<?extends Construction>l1 = null;//l1的泛型类型为Construction及子类
        l1 = new ArrayList<Construction>();
        l1 = new ArrayList<Build>();
        List<?super Build> l2=null;//l2的泛型类型为Build及父类Construction
        l2 = new ArrayList<Construction>();
        l2 = new ArrayList<Build>();
    }
}
class Construction{

}
class Build extends Construction{

}
```

8.6工具类

- 8.6.1Collections工具类
- Collections类中提供大量的静态方法用于对集合中元素进行排序、查找和修改等操作。

8.6工具类

- 1.排序操作

```
static void
reverse(List li);
 shuffle(List li);
 sort(List li);
 swap(List li, int i, int j);
```

- 2.查找、替换操作

```
static int binarySearch(List li,Object o)
static Object max(Collection co)
static Object min(Collection co)
static Boolean replaceAll(List li, Object o1, Object o2)
Int frequency(Collection coll, Object o)
```

8.6工具类

```
元素为：[2, 8, null, 5, 10, 5]
null替换为0的结果为：[2, 8, 0, 5, 10, 5]
最大最小值为：10，  0
二分法查找元素10的下标为4
排序后的集合为：[0, 2, 5, 5, 8, 10]
反转后的集合为：[10, 8, 5, 5, 2, 0]
乱序后的集合为：[5, 10, 2, 0, 5, 8]
元素5出现了2次
```

```java
import java.util.*;
public class CollectionFunction {
    public static void main(String[] args) {
        ArrayList<Integer> li=new ArrayList<Integer>();
        Collections.addAll(li, 2, 8, null, 5, 10, 5);
        System.out.println("元素为："+li);
        Collections.replaceAll(li, null, 0);//集合li中所有null用0替换
        System.out.println("null替换为0的结果为："+li);
        System.out.println("最大、最小值为："+Collections.max(li)+",  "+Collections.min(li));
        System.out.println("二分法查找元素10的下标为"+Collections.binarySearch(li, 10));
        Collections.sort(li);
        System.out.println("排序后的集合为："+li);
        Collections.reverse(li);
        System.out.println("反转后的集合为："+li);
        Collections.shuffle(li);
        System.out.println("乱序后的集合为："+li);
        System.out.println("元素5出现了"+Collections.frequency(li, 5)+"次");
    }
}
```

8.6工具类

- 8.6.2Arrays工具类
- 在java.util包中，除了针对集合操作提供了一个集合工具类Collections，还针对数组操作提供了一个数组工具类—Arrays。
- 静态方法static：
  void sort(Object[] ob)
- int binaySearch(Object[] ob,Object o)
- void fill(Object[] ob, Object o)
- String toString(Object[] ob)
- Object[] copyOfRange(Object[] ob,int i,int j)

8.7集合转换

- 在开发中，我们可能需要将集合对象（List，Set）转换为数组对象，或者将数组对象转换为集合对象。Java提供了相互转换的方法。集合可以直接转换为数组，数组也可以直接转换为集合。
- 1.集合转换为数组：toArray()
- 2.数组转换为集合: asList(**)

```
import java.util.*;
public class ArrayAndArrayList {
    public static void main(String[] args) {
        List l=new ArrayList();
        l.add(1);l.add(2);l.add(3);
        //集合转换为数组
        Object[] ob = l.toArray();
        for(Object o:ob)
            System.out.print(o+"\t");
        //数组转换为集合
        l= Arrays.asList(ob);
        System.out.println(l);
    }
}
```

```
1       2       3       [1, 2, 3]
```