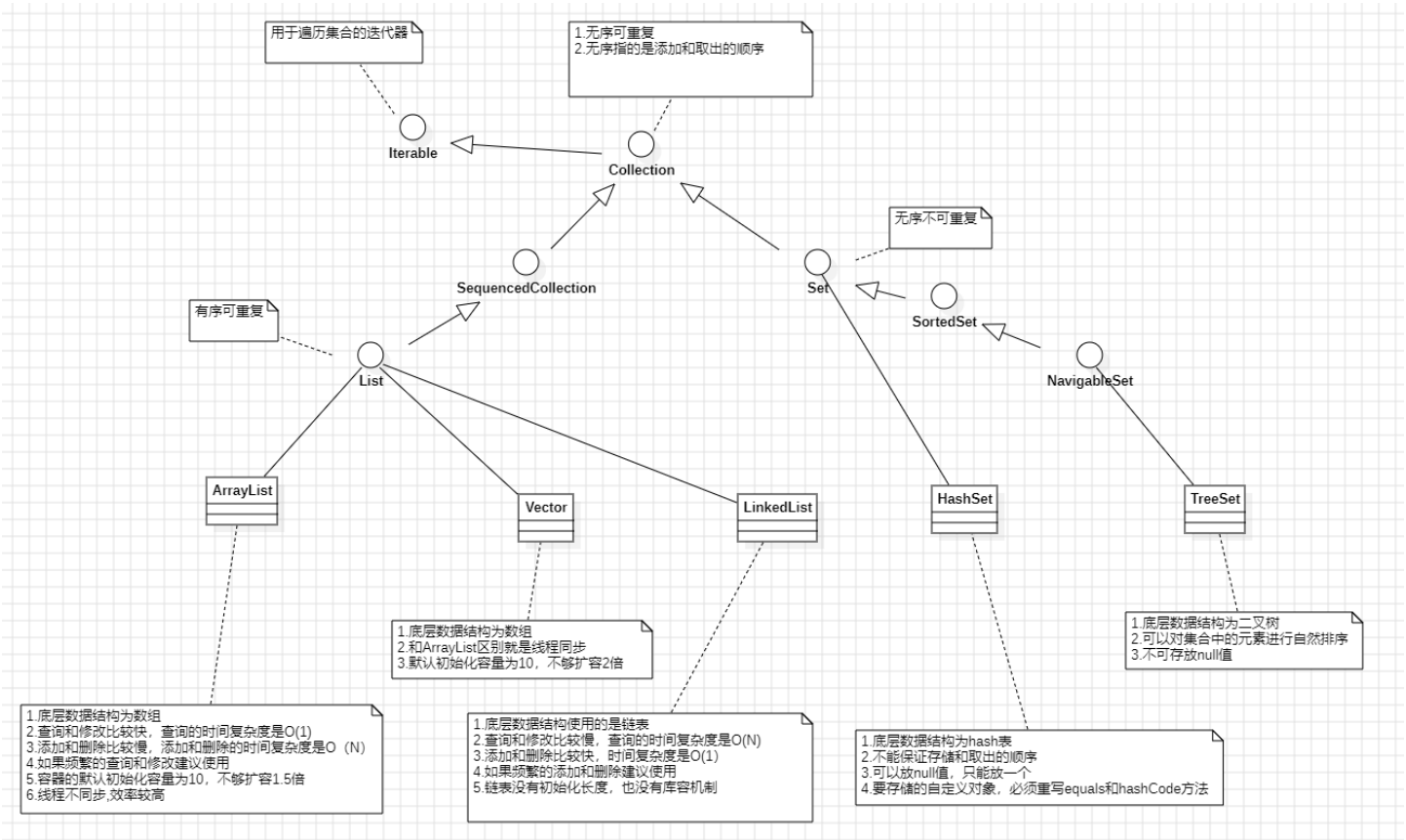
第十八章 集合与泛型

1. 集合

1.1 集合的概述

1. 集合和数组类似，都可以存储批量的数据
2. 集合是可变的，数组是不可变的

1.2 集合的家族体系



1.3 TreeSet自动排序

代码块

```
package com.powernode.collection02;

import java.util.ArrayList;
import java.util.Collection;
import java.util.TreeSet;

public class Test01 {
    public static void main(String[] args) {
        Collection collection = new TreeSet();
        collection.add(5);
        collection.add(1);
        collection.add(2);
        collection.add(3);
        collection.add(4);
        //java.lang.NullPointerException
        //TreeSet要排序，排序要比较，null存参与比较，所以报空指针异常
        collection.add(null);
        System.out.println(collection);
    }
}
```

1.4 List(有序可重复）和Set（无序不可重复）

代码块

```
package com.powernode.collection02;
```

```
2
3  import java.util.*;
4
5  public class Test02 {
6      public static void main(String[] args) {
7          List list = new ArrayList();//有序可重复
8          //把数据放入集合
9          list.add(11);
10         list.add(21);
11         list.add(31);
12         list.add(41);
13         list.add(11);//可重复
14         list.add(null);
15         list.add(null);
16         System.out.println(list);//[11, 21, 31, 41]
17
18         Set set = new HashSet();//无序不可重复
19         set.add(11);
20         set.add(21);
21         set.add(31);
22         set.add(41);
23         set.add(11);//不可重复
24         set.add(null);
25         set.add(null);
26         System.out.println(set);//[21, 41, 11, 31]
27
28     }
29 }
```

1.5 没有泛型存在的问题

- 数据类型比较混乱，什么都可以存储
- 为后期数据处理，带来麻烦（判断，强制转换）

代码块

```
1  package com.powernode.collection02;
2
3  import java.util.ArrayList;
4  import java.util.HashSet;
5  import java.util.List;
6  import java.util.Set;
7
8  public class Test03 {
9      public static void main(String[] args) {
10         List list = new ArrayList();
```

```
11          list.add(1);
12          list.add(2);
13          list.add(3);
14         /* list.add("abc");
15          list.add(true);*/
16          //list存储的数据类型比较混乱，为后期数据处理带来了麻烦
17          sum(list);
18
19      }
20      public static void sum(List list){
21          int sum = 0;
22          for (Object o : list) {
23              if (o instanceof Integer) {
24                  Integer i = (Integer) o;
25                  sum += i;
26              }
27          }
28          System.out.println(sum);
29      }
30  }
```

1.6 使用泛型来约束集合

代码块

```
1   package com.powernode.collection02;
2
3   import java.util.ArrayList;
4   import java.util.List;
5
6   public class Test04 {
7       public static void main(String[] args) {
8           /**
9            * 1. <Integer>约束了list集合只能存储Integer对象
10           * 2. 存储其他数据类型的对象，报错
11           */
12          List<Integer> list = new ArrayList();
13          list.add(1);
14          list.add(2);
15          list.add(3);
16         /* list.add("abc");
17          list.add(true);*/
18          //list存储的数据类型比较混乱，为后期数据处理带来了麻烦
19          sum(list);
20
21      }
```

```
22        //使用泛型，可以解决不必要的数据类型强制转换
23        public static void sum(List<Integer> list){
24            int sum = 0;
25            for (Integer integer : list) {
26                sum += integer;
27            }
28            /*for (Object o : list) {
29                if (o instanceof Integer) {
30                    Integer i = (Integer) o;
31                    sum += i;
32                }
33            }*/
34            System.out.println(sum);
35        }
36    }
```

2. 泛型

2.1 泛型的概述

1. 泛型是什么
    a. 泛型是对集合中数据的一种约束
    b. 可以限定集合存储的数据类型
    c. 泛型支持多态存储
        i. 限定了父类
            1. 子类
            2. 父类
2. 泛型的作用
    a. 对集合约束，避免不必要的数据类型转换
    b. 把运行期可能出现的异常，提前到编译器
3. 泛型使用时的注意事项
    a. 泛型只支持引用类型，不能使用基本类型
    b. 泛型书写，前后必须保持一致，后面可以省略

代码块

```
1    List<Integer> list = new ArrayList<Integer>();//前后保持一致
2    List<Integer> list = new ArrayList<>();//后面可以省略
```

代码块

```
package com.powernode.generics03;

import java.util.ArrayList;
import java.util.List;

public class Test01 {
    public static void main(String[] args) {
        //Integer,Double,Long...extends Number
        List<Number> list = new ArrayList<>();
        list.add(1);
        list.add(1.2);
    }
}
```

2.2 泛型不支持协变

代码块

```
package com.powernode.generics03;

import java.util.ArrayList;
import java.util.List;

class Person{}
class Teacher extends Person{}
public class Test02 {
    public static void main(String[] args) {
        List<Person> lp = new ArrayList<>();
        printList(lp);
        List<Teacher> lt = new ArrayList<>();
        //printList(lt);
    }
    //List<Person> lp = new ArrayList<Teacher>();
    private static void printList(List<Person> lp) {
    }
}
```

2.3 方法重载忽略泛型

代码块

```
package com.powernode.generics04;

```

```java
import java.util.ArrayList;
import java.util.List;

class Person{}
class Teacher extends Person{}
public class Test {
    public static void main(String[] args) {
        List<Person> lp = new ArrayList<>();
        printList(lp);
        List<Teacher> lt = new ArrayList<>();
        printList(lt);
    }

    /**
     * - 方法重载
     *     1.同一个类中
     *     2.方法名称相同
     *     3.参数列表不同
     *         1.个数不同
     *         2.类型不同
     *         3.顺序不同
     *  注意：方法重载，参数列表，忽略泛型，所报错
     */
    //List<Person> lp = new ArrayList<Teacher>();
    private static void printList(List<Person> lp) {
    }
    private static void printList(List<Teacher> lp) {
    }
}
```

2.4 泛型的向上和向下限定

代码块

```java
package com.powernode.generics05;

import java.util.ArrayList;
import java.util.List;

class Person{}
class Teacher extends Person{}
class Student extends Person{}
public class Test {
    public static void main(String[] args) {
        List<Person> lp = new ArrayList<>();
        printList(lp);
```

```
13          List<Teacher> lt = new ArrayList<>();
14          printList(lt);
15          List<Student> ls = new ArrayList<>();
16          printList(ls);
17          System.out.println("--------------");
18          printList01(lp);
19          printList01(lt);
20          //printList01(ls);
21      }
22
23      /**
24       * 1.向上限定(确定了父类) :List<? extends Person>
25       *     1. ? 是占位符，表示不确定
26       *     2. ? extends Person，说明子类不确定
27       *     3. 可以传递:
28       *         1.父类
29       *         2.子类
30       */
31      private static void printList(List<? extends Person> lp) {
32      }
33
34      /**
35       * 2.向下限定（确定子类）：List<? super Teacher>
36       *     1. ? 是占位符，表示不确定
37       *     2.? super Teacher ：  说明父类不确定
38       *     3. 可以传递:
39       *        1.父类
40       *        2.子类（确定的子类Teacher）
41       */
42      private static void printList01(List<? super Teacher> lp) {
43      }
44
45
46
47  }
```

2.5 泛型通配符

1. T : Type
2. E: Element
3. ? : 不确定
4. K,V： Key,Value

2.5.1 T，E

代码块

```
package com.powernode.generics06;

import com.powernode.generics04.Test;

class Teacher{}
class TestObject<AAA>{//可以理解为形参，T就是一个引用类型的变量
    //实例变量
    int id;
    //实例变量不确定
    AAA t;

    public void setT(AAA t) {
        this.t = t;
    }
}

/**
 * T和E 不具有实际意义，只是定义一个形参，因为泛型传递的是数据类型，所以叫T(Type)更具有意义
 */
/*class TestObject<T>{//可以理解为形参，T就是一个引用类型的变量
    //实例变量
```

```
    int id;
    //实例变量不确定
    T t;

    public void setT(T t) {
        this.t = t;
    }
}*/
public class Test01 {
    public static void main(String[] args) {
        TestObject<Teacher> testObject = new TestObject();
        testObject.setT(new Teacher());
    }
}
```

2.6 ?

代码块

```
package com.powernode.generics06;

import java.util.ArrayList;
import java.util.List;

public class Test02 {
    public static void main(String[] args) {

    }

    /**
     * 1.T和E 只能限定一种数据类型
     * 2.? 数据类型不确定，可以限定多种数据类型（Integer,Double）
     * 3.如果方法返回一个集合，但是这个集合中的数据不确定，可以使用?
     */
    public static List<?> method(boolean flag){
        if (flag) {
            return new ArrayList<Integer>();
        }
        return new ArrayList<Double>();
    }
}
```

2.7 K,V

```java
package com.powernode.generics06;

class IMath<K,V>{
    public void add(K k,V v){}
}
public class Test03 {
    public static void main(String[] args) {
        IMath<Integer, Double> iMath = new IMath<>();
        iMath.add(2,1.1);
    }
}
```

3. 集合中的常用方法

3.1 Collection常用方法（一）

代码块

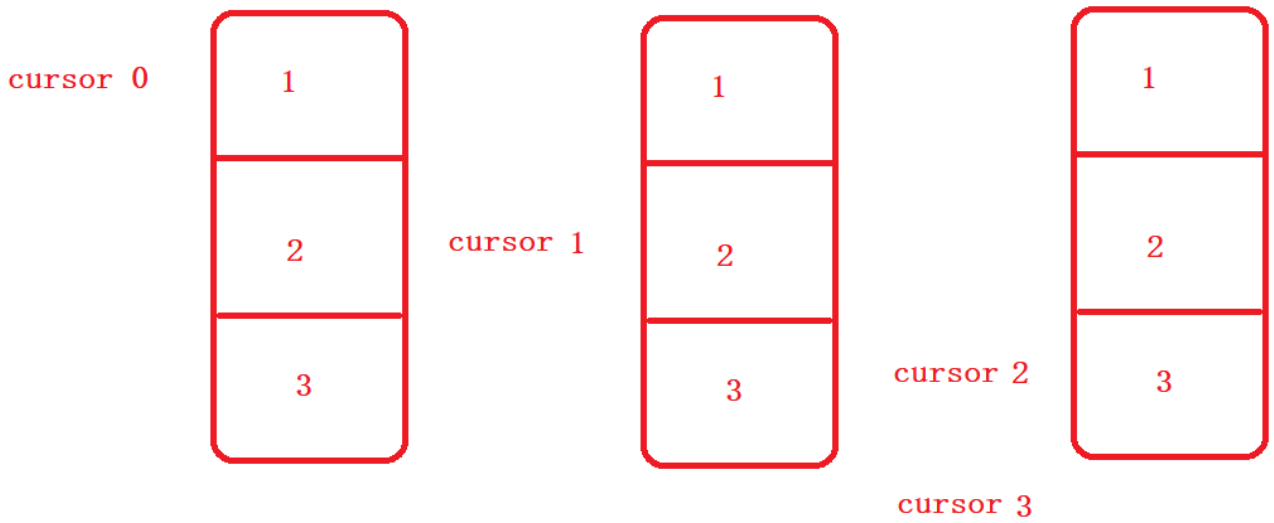
```java
package com.powernode.collection07;

import java.util.ArrayList;
import java.util.Collection;

public class Test01 {
    public static void main(String[] args) {
        /**
         * 1.添加
         *    boolean add(E e)  添加e元素对象到当前集合中,添加成功返回true
         *    boolean addAll(Collection<? extends E> c) 添加c集合中的所有元素对象到当前集合中，添加成功返回true
         * 2.判断
         *    boolean contains(Object o)    判断当前集合中是否包含0对象，包含返回true
         *    boolean containsAll(Collection<?> c)  判断当前集合是否包含c集合中所有元素，包含返回true
         *    boolean equals(Object o)  判断当前集合的元素值是否等于o集合
         *    boolean isEmpty() 判断当前集合是否为空集合
         * 3.查询
         *    int hashCode()    获取集合对象的哈希值
         *    Iterator<E> iterator()    返回此集合中的元素的迭代器。
         *    int size()    获取当前集合中实际存储的元素个数
         * 4.删除
         *    void clear()  清空集合元素
         *    E remove(int index )  删除集合中索引为index的对象，返回删除的对象
         *    boolean removeAll(Collection<?> c)    从集合中删除 c集合
```

```
25            *    boolean retainAll(Collection<?> c)    仅保留c中的集合元素
26            * 5.数组与集合
27            *    Object[] toArray()    返回包含当前集合中所有元素的数组
28            *    <T> T[] toArray(T[] a)    把集合转换为指定类型的数组
29            * 6.关于Stream流
30            *    default Stream<E> parallelStream()    返回一个可能并行 Stream与集合
    的来源。
31            *    default boolean removeIf(Predicate<? super E> filter) 当前集合中与
    filter的交集
32            *    default Spliterator<E> spliterator()  创建此集合中的元素的
    Spliterator。
33            *    default Stream<E> stream()    返回一个序列 Stream与集合的来源。
34            */
35           /**
36            * 1.添加
37            *    boolean add(E e)  添加e元素对象到当前集合中,添加成功返回true
38            *    boolean addAll(Collection<? extends E> c) 添加c集合中的所有元素对象
    到当前集合中，添加成功返回true
39            */
40           Collection<Integer> integers1 = new ArrayList<>();
41           integers1.add(1);
42           integers1.add(2);
43           integers1.add(3);
44           System.out.println("integers1 = " + integers1);//[1, 2, 3]
45
46           Collection<Integer> integers2 = new ArrayList<>();
47           integers2.add(4);
48           integers2.add(5);
49           integers2.add(6);
50           integers1.addAll(integers2);
51           System.out.println("integers1 = " + integers1);//[1, 2, 3, 4, 5, 6]
52           /**
53            * 2.判断
54            *    boolean contains(Object o)    判断当前集合中是否包含0对象，包含返回
    true
55            *    boolean containsAll(Collection<?> c)  判断当前集合是否包含c集合中所有
    元素，包含返回true
56            *    boolean equals(Object o)  判断当前集合的元素值是否等于o集合
57            *    boolean isEmpty() 判断当前集合是否为空集合
58            */
59           //integers1 = [1, 2, 3, 4, 5, 6]
60           System.out.println("integers1.contains(6) = " +
    integers1.contains(6));//true
61           System.out.println("integers1.containsAll(integers2) = " +
    integers1.containsAll(integers2));//true
62           System.out.println("integers1.equals(integers2) = " +
    integers1.equals(integers2));//false
```

```
63          System.out.println("integers1.isEmpty() = " +
    integers1.isEmpty());//false
64      }
65  }
```

3.2 Collection常用方法（二）

- ArrayList 中有一个内部类，private class Itr implements Iterator<E> {



代码块

```
1   package com.powernode.collection07;
2
3   import java.util.ArrayList;
4   import java.util.Collection;
5   import java.util.Iterator;
6
7   public class Test02 {
8       public static void main(String[] args) {
9           /**
10           * 1.添加
11           *    boolean add(E e)  添加e元素对象到当前集合中,添加成功返回true
12           *    boolean addAll(Collection<? extends E> c) 添加c集合中的所有元素对象
    到当前集合中，添加成功返回true
13           * 2.判断
14           *    boolean contains(Object o)    判断当前集合中是否包含o对象，包含返回
    true
15           *    boolean containsAll(Collection<?> c)  判断当前集合是否包含c集合中所有
    元素，包含返回true
16           *    boolean equals(Object o)  判断当前集合的元素值是否等于o集合
```

```
17            *    boolean isEmpty() 判断当前集合是否为空集合
18            * 3.查询
19            *    int hashCode()    获取集合对象的哈希值
20            *    Iterator<E> iterator()    返回此集合中的元素的迭代器。
21            *    int size()    获取当前集合中实际存储的元素个数
22            * 4.删除
23            *    void clear()  清空集合元素
24            *    E remove(int index )  删除集合中索引为index的对象，返回删除的对象
25            *    boolean removeAll(Collection<?> c)    从集合中删除 c集合
26            *    boolean retainAll(Collection<?> c)    仅保留c中的集合元素
27            * 5.数组与集合
28            *    Object[] toArray()    返回包含当前集合中所有元素的数组
29            *    <T> T[] toArray(T[] a)    把集合转换为指定类型的数组
30            * 6.关于Stream流
31            *    default Stream<E> parallelStream()    返回一个可能并行 Stream与集合的来源。
32            *    default boolean removeIf(Predicate<? super E> filter) 当前集合中与filter的交集
33            *    default Spliterator<E> spliterator()  创建此集合中的元素的Spliterator。
34            *    default Stream<E> stream()    返回一个序列 Stream与集合的来源。
35            */
36           /**
37            * 3.查询
38            *    int hashCode()    获取集合对象的哈希值
39            *    Iterator<E> iterator()    返回此集合中的元素的迭代器。
40            *    int size()    获取当前集合中实际存储的元素个数
41            */
42           Collection<Integer> integers1 = new ArrayList<>();
43           integers1.add(1);
44           integers1.add(2);
45           integers1.add(3);
46           System.out.println("integers1.hashCode() = " + integers1.hashCode());
47           //通过集合拿到一个迭代器，用于遍历集合中元素
48           Iterator<Integer> iterator = integers1.iterator();
49           while(iterator.hasNext()){
50               //指针+1，取出元素
51               Integer next = iterator.next();
52               System.out.println("next = " + next);
53           }
54           System.out.println(integers1.size());
55           System.out.println(iterator.next());//java.util.NoSuchElementException, 迭代器到底了，没有元素可取
56       }
57   }
```

3.3 Collection常用方法（三）

代码块

```
package com.powernode.collection07;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Iterator;

public class Test03 {
    public static void main(String[] args) {
        /**
         * 1.添加
         *    boolean add(E e)  添加e元素对象到当前集合中,添加成功返回true
         *    boolean addAll(Collection<? extends E> c) 添加c集合中的所有元素对象到当前集合中，添加成功返回true
         * 2.判断
         *    boolean contains(Object o)    判断当前集合中是否包含o对象，包含返回true
         *    boolean containsAll(Collection<?> c)  判断当前集合是否包含c集合中所有元素，包含返回true
         *    boolean equals(Object o)  判断当前集合的元素值是否等于o集合
         *    boolean isEmpty() 判断当前集合是否为空集合
         * 3.查询
         *    int hashCode()    获取集合对象的哈希值
         *    Iterator<E> iterator()    返回此集合中的元素的迭代器。
         *    int size()    获取当前集合中实际存储的元素个数
         * 4.删除
         *    void clear()  清空集合元素
         *    E remove(int index )  删除集合中索引为index的对象，返回删除的对象
         *    boolean removeAll(Collection<?> c)    从集合中删除 c集合
         *    boolean retainAll(Collection<?> c)    仅保留c中的集合元素
         * 5.数组与集合
         *    Object[] toArray()    返回包含当前集合中所有元素的数组
         *    <T> T[] toArray(T[] a)    把集合转换为指定类型的数组
         * 6.关于Stream流
         *    default Stream<E> parallelStream()    返回一个可能并行 Stream与集合的来源。
         *    default boolean removeIf(Predicate<? super E> filter) 当前集合中与filter的交集
         *    default Spliterator<E> spliterator()  创建此集合中的元素的Spliterator。
         *    default Stream<E> stream()    返回一个序列 Stream与集合的来源。
         */
        /**
```

```
38            * 4.删除
39            *    void clear()  清空集合元素
40            *    E remove(Object o )   删除集合中指定的数据
41            *    boolean removeAll(Collection<?> c)    从集合中删除 c集合
42            *    boolean retainAll(Collection<?> c)    仅保留c中的集合元素
43            */
44           Collection<Integer> integers1 = new ArrayList<>();
45           integers1.add(1);
46           integers1.add(2);
47           integers1.add(3);
48           System.out.println("integers1.remove(2) = " + integers1.remove(2));
49           System.out.println(integers1);//[1, 3]
50
51           Collection<Integer> integers2 = new ArrayList<>();
52           integers2.add(4);
53           integers2.add(5);
54           integers2.add(6);
55           integers1.addAll(integers2);
56           System.out.println("integers1 = " + integers1);//[1, 3, 4, 5, 6]
57           integers1.removeAll(integers2);
58           System.out.println("integers1 = " + integers1);//[1, 3]
59           integers1.addAll(integers2);
60           System.out.println("integers1 = " + integers1);//[1, 3, 4, 5, 6]
61           integers1.retainAll(integers2);
62           System.out.println("integers1 = " + integers1);//[4, 5, 6]
63           integers1.clear();
64           System.out.println("integers1 = " + integers1);//[]
65           /**
66            * 5.数组与集合
67            *    Object[] toArray()    返回包含当前集合中所有元素的数组
68            *    <T> T[] toArray(T[] a)    把集合转换为指定类型的数组
69            */
70           Collection<Integer> integers = new ArrayList<>();
71           integers.add(11);
72           integers.add(22);
73           integers.add(33);
74           //Object 类型处理不方便
75           Object[] array = integers.toArray();
76           //把集合转换为指定类型的数组
77           Integer[] array1 = integers.toArray(new Integer[integers.size()]);
78           System.out.println(Arrays.toString(array1));//[11, 22, 33]
79
80       }
81   }
```

3.4 SequencedCollection方法

代码块

```
package com.powernode.collection07;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.SequencedCollection;

public class Test04 {
    public static void main(String[] args) {
        SequencedCollection<Integer> sc = new ArrayList<>();
        sc.add(6);
        sc.add(7);
        sc.add(8);
        System.out.println(sc);//[6, 7, 8]
        sc.addFirst(5);
        System.out.println(sc);//[5, 6, 7, 8]
        sc.addLast(9);
        System.out.println(sc);//[5, 6, 7, 8,9]
        System.out.println("sc.getFirst() = " + sc.getFirst());//5
        System.out.println("sc.getLast() = " + sc.getLast());//9
        System.out.println("sc.removeFirst() = " + sc.removeFirst());//5 ,返回移除的第一个元素
        System.out.println("sc = " + sc);//[6, 7, 8, 9]
        System.out.println("sc.removeLast() = " + sc.removeLast());//9, 返回移除的最后一个元素
        System.out.println("sc = " + sc);//[6, 7, 8]
        System.out.println("sc.reversed() = " + sc.reversed());//[8, 7, 6]

    }
}
```

3.5 List常用方法

代码块

```
package com.powernode.collection08;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.ListIterator;

public class Test01 {
    public static void main(String[] args) {
```

```java
        /**
         * void add(int index, E element)   在当前集合中index处插入element元素
         * boolean addAll(int index, Collection<? extends E> c)     将c集合中的所有元素插入到指定位置
         * E get(int index) 返回index位置的元素。
         * int indexOf(Object o)    返回o在当前集合中的第一次出现的索引，-如果不包含o元素，返回- 1
         * int lastIndexOf(Object o)    返回o在当前集合中的索引，-如果包含o元素，返回- 1。
         * default E getFirst() 返回第一个元素
         * ListIterator<E> listIterator()   返回列表元素的列表迭代器。
         * ListIterator<E> listIterator(int index)   从当前集合中指定[index,最后)返回一个迭代器。
         * E set(int index, E element)  指定index位置，替换为element元素
         * sort(Comparator<? super E> c)
         * 分类列表使用提供的 Comparator比较元素。
         * List<E> subList(int fromIndex, int toIndex)  截取当前List中从[fromIndex,toIndex)的子集，返回一个List对象
         * -- static <E> Set<E> copyOf(Collection<? extends E> coll)    java10新增特性，静态方法，复制一个集合
         * -- static <E> Set<E> of(E e1....)    返回一个不可变的集合，如果添加删除等会报UnsupportedOperationException
         */
        List<Integer> list1 = new ArrayList<>();
        list1.add(11);
        list1.add(1, 22);//[11, 22]
        System.out.println("list1 = " + list1);
        List<Integer> list2 = new ArrayList<>();
        list2.add(33);
        list2.add(44);
        list2.add(55);
        list1.addAll(2,list2);
        System.out.println("list1 = " + list1);//[11, 22, 33, 44, 55]

        System.out.println("list1.get(0) = " + list1.get(0));//11
        System.out.println("list1.indexOf(33) = " + list1.indexOf(33));//2
        System.out.println("list1.lastIndexOf(33) = " + list1.lastIndexOf(33));//2
        //通过list1拿到list专用迭代器，可以从前往后迭代，也可以从后往前迭代
        ListIterator<Integer> integerListIterator = list1.listIterator();
        System.out.println("------------------从前往后迭代----------------------");
        while(integerListIterator.hasNext()){
            System.out.println("返回元素的索引: " + integerListIterator.nextIndex());
            System.out.println("返回元素: " + integerListIterator.next());
        }
```

```
        System.out.println("------------------从后往前迭代----------------------
");
        while(integerListIterator.hasPrevious()){
            System.out.println("返回元素的索引: " +
integerListIterator.previousIndex());
            System.out.println("返回元素: " + integerListIterator.previous());
        }
        System.out.println(list1);//[11, 22, 33, 44, 55]
        list1.set(3, 88);
        System.out.println(list1);//[11, 22, 33, 88, 55]
        list1.sort(new Comparator<Integer>() {
            @Override
            public int compare(Integer o1, Integer o2) {
                return o2 - o1;
            }
        });
        System.out.println(list1);//[88, 55, 33, 22, 11]
        System.out.println("list1.subList(2,4) = " + list1.subList(2,
4));//[33, 22]

    }
}
```

3.6 Set常用方法

代码块

```
package com.powernode.collection08;

import java.util.*;

public class Test02 {
    public static void main(String[] args) {
        /**
         * -- static <E> Set<E> copyOf(Collection<? extends E> coll)    java10
新增特性，静态方法，复制一个集合
         * -- static <E> Set<E> of(E e1....)    返回一个不可变的集合，如果添加删除等
会报UnsupportedOperationException
         */
        Set<Integer> set = new HashSet<>();
        set.add(11);
        set.add(22);
        set.add(33);

        Set<Integer> set1 = Set.copyOf(set);//可以理解为只读集合
        System.out.println(set1);//[33, 22, 11]
```

```
        System.out.println(set == set1);//false 两块内存
        //set1.add(55);java.lang.UnsupportedOperationException 不支持的方法异常
        //set1.remove(22);java.lang.UnsupportedOperationException 不支持的方法异常


        Set<Integer> set2 = Set.of(1, 2, 3);
        //set2.add(4);java.lang.UnsupportedOperationException 不支持的方法异常
        //set2.remove(1);java.lang.UnsupportedOperationException 不支持的方法异常


    }
}
```

3.7 HashSet存储自定义对象属性问题

代码块

```
package com.powernode.collection09;

import java.util.HashSet;
import java.util.Set;

class Student{
    private String name;
    private int age;

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return "Student{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
public class Test {
    public static void main(String[] args) {
        Set<Student> set = new HashSet<>();
        set.add(new Student("zs", 23));
        set.add(new Student("zs", 23));
```

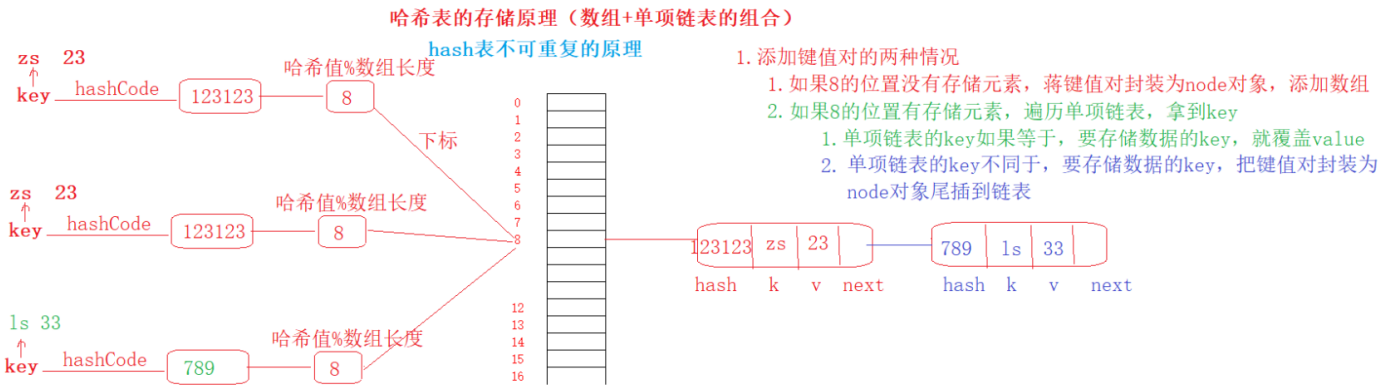
```
        System.out.println(set);
        /**
         * 1.Set 无序不可重复，为什么两个对象都存储进去了呢?
         * 2.HashSet存储的是两个对象，对象的属性是zs,23,所以都存储进去了
         * 3.Set存储自定义对象，保证属性唯一
         *     1.通过hashCode和equals进行保证
         *     2.首先判断hashCode
         *         1.相等，调用equals判断内容
         *             1.true:相同不添加
         *             2.false：不相同添加
         *         2.不相等
         *             不会调用equals直接添加
         */
    }
}
```

3.8 解决HashSet存储自定义对象属性唯一问题

代码块

```
package com.powernode.collection10;

import java.util.HashSet;
import java.util.Objects;
import java.util.Set;

class Student{
    private String name;
    private int age;

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public boolean equals(Object object) {
        if (this == object) return true;
        if (object == null || getClass() != object.getClass()) return false;
        Student student = (Student) object;
        return age == student.age && Objects.equals(name, student.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
```

```
27      }
28
29      @Override
30      public String toString() {
31          return "Student{" +
32                  "name='" + name + '\'' +
33                  ", age=" + age +
34                  '}';
35      }
36  }
37  public class Test {
38      public static void main(String[] args) {
39          Set<Student> set = new HashSet<>();
40          set.add(new Student("zs", 23));
41          set.add(new Student("zs", 233));
42          System.out.println(set);
43          /**
44           * 1.Set 无序不可重复，为什么两个对象都存储进去了呢?
45           * 2.HashSet存储的是两个对象，对象的属性是zs,23,所以都存储进去了
46           * 3.Set存储自定义对象，保证属性唯一
47           *      1.通过hashCode和equals进行保证
48           *      2.首先判断hashCode
49           *          1.相等，调用equals判断内容
50           *              1.true:相同不添加
51           *              2.false：不相同添加
52           *          2.不相等
53           *              不会调用equals直接添加
54           *
55           * 4.HashSet 存储自定义对象，首先判断hashCode,hashCode都相等了，为什么还要判
     断equals
56           *      4.1.哈希表的工作原理
57           *          HashSet是依靠哈希表来实现的，采用的“数组 + 链表|红黑树”这种结构，
     当要存储一个对象时，会按照如下步骤：
58           *              1.计算存储位置：借助hashCode算出hash值，进行确定该对象在数组中
     的存储位置
59           *              2.处理hash冲突：有多个对象算出的hash值相同，这就是hash冲突，使
     用尾插法
60           *      4.2.为什么要equals判断
61           *          哈希冲突的影响，不同的对象可能会算出相同的hashCode,hashCode算出的
     位置仅仅表明对象存储在同一个位置，不能表明它们就是同一个对象
62           *          确保元素的唯一性：HashSet不允许存储重复元素，就必须使用equals进行判
     断
63           *
64           *
65           */
66      }
67  }
```



3.9 TreeSet存储自定义对象

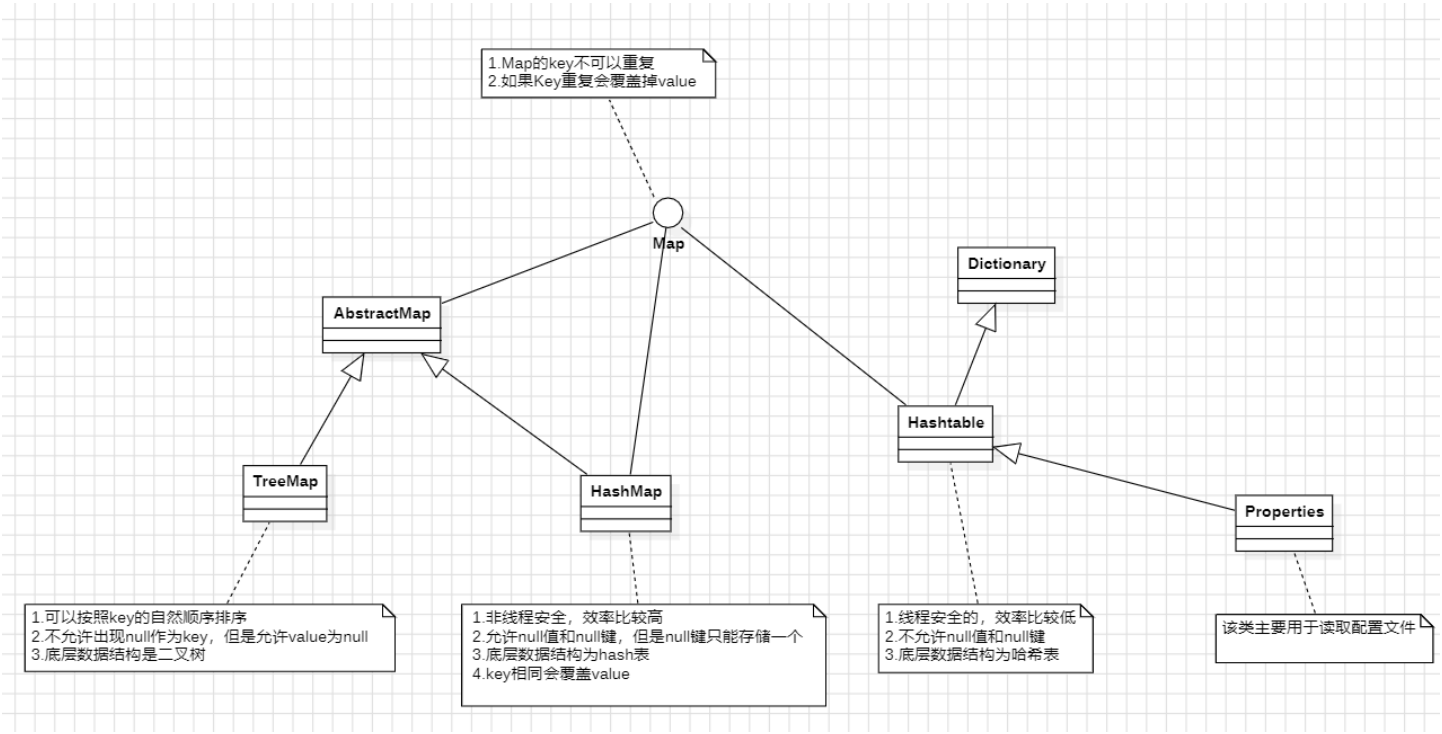
代码块

```
package com.powernode.collection11;

import java.sql.Struct;
import java.util.Set;
import java.util.TreeSet;

class Student implements Comparable<Student>{
    private String name;
    private int age;

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return "Student{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }


    @Override
```

```
26      public int compareTo(Student o) {
27          if (age > o.age) {
28              return 1;
29          } else if (age < o.age) {
30              return -1;
31          }
32          return name.compareTo(o.name);
33      }
34  }
35  public class Test {
36      public static void main(String[] args) {
37          Set<Student> set = new TreeSet<>();
38          set.add(new Student("zs", 23));
39          set.add(new Student("zs", 23));
40          System.out.println(set);
41          /**
42           *  1. ClassCastException: Student cannot be cast to class Comparable
43           *  2. TreeSet存储自定义对象，该类必须实现Comparable接口
44           *  3. 因为TreeSet按照自然顺序排序，所以存储的对象必须具有比较性
45           *  4. 自定义对象具有比较性，必须实现Comparable接口
46           *      1.compareTo 对象两个对象的内容是否相同，如果相同不添加到TreeSet
47           *      2.compareTo 尽量把所有的属性都写上
48           */
49      }
50  }
```

4. Map

4.1 Map的家族体系



4.2 HashMap，Hashtable和TreeMap的区别

代码块

```
    package com.powernode.collection12;

import java.util.HashMap;
import java.util.Hashtable;
import java.util.Map;
import java.util.TreeMap;

public class Test01 {
    public static void main(String[] args) {
        /**
         *  HashMap，Hashtable和TreeMap的区别
         *    1.HashMap
         *        1.非线程安全，效率比较高
         *        2.允许null值和null键，但是null键只能存储一个
         *        3.底层数据结构为hash表
         *        4.key相同会覆盖value
         *    2.Hashtable
         *        1.线程安全的，效率比较低
         *        2.不允许null值和null键
         *        3.底层数据结构为哈希表
         *    3.TreeMap
         *       1.可以按照key的自然顺序排序
         *       2.不允许出现null作为key，但是允许value为null
         *       3.底层数据结构是二叉树
         */
        Map<Integer, String> map = new HashMap<>();
```

```
27          map.put(1001, "zs");
28          map.put(1001, "ls");//key相同会覆盖value
29          System.out.println(map);//{1001=ls}
30          map.put(null, null);
31          System.out.println(map);
32          map.put(null, "ww");
33          System.out.println(map);
34          System.out.println("-------Hashtable--------");
35          Map<Integer, String> map1 = new Hashtable<>();
36          //map1.put(null, "zs");NullPointerException
37          //map1.put(1001, null);NullPointerException
38          System.out.println("-------TreeMap--------");
39          Map<Integer, String> map2 = new TreeMap<>();
40          //map2.put(null, "zs");NullPointerException
41          map2.put(1001, null);
42          map2.put(1001, "zs");
43          map2.put(1000, "zs");
44          map2.put(1002, "zs");
45
46          System.out.println(map2);
47
48      }
49  }
```

4.3 Map的常用方法

代码块

```
1   package com.powernode.collection12;
2
3   import java.util.*;
4
5   public class Test02 {
6       public static void main(String[] args) {
7           /**
8            * V put(K key, V value);   指定key和value添加到集合中
9            * V get(Object key)    通过key获得value
10           * Set<K> keySet()  返回一个包含key值的set集合
11           * Collection<V> values()   获得所有的value
12           * V remove(Object key)     通过指定的key删除元素，返回删除的value
13           * boolean isEmpty()    是否为空
14           * void putAll(Map<? extends K, ? extends V> m)     把m 添加到当前map中
15           * boolean containsKey(Object key)  是否包含指定的key
16           * boolean containsValue(Object value)  是否包含指定的值     是否包含指定的
    值
17           * Set<Map.Entry<K, V>> entrySet()   返回一个set集合
```

```
18            * void clear()      清空map
19            * default V replace(K key, V value)      替换指定key上的value值
20            */
21           Map<Integer, String> map = new HashMap<>();
22           map.put(11, "aa");
23           map.put(22, "bb");
24           map.put(33, "cc");
25           System.out.println("map = " + map);//{33=cc, 22=bb, 11=aa}
26           System.out.println("map.get(22) = " + map.get(22));//bb
27
28           Set<Integer> keys = map.keySet();
29           for (Integer key : keys) {
30               System.out.println(key);
31           }
32
33           Collection<String> values = map.values();
34           Iterator<String> iterator = values.iterator();
35           while (iterator.hasNext()) {
36               String next = iterator.next();
37               System.out.println(next);
38           }
39           System.out.println(map);//{33=cc, 22=bb, 11=aa}
40           System.out.println(map.remove(22));//bb 返回删除的value
41           System.out.println(map);//{33=cc, 11=aa}
42           System.out.println(map.isEmpty());//false
43
44           Map<Integer, String> map1 = new HashMap<>();
45           map1.put(22, "bb");
46           map1.put(44, "cc");
47           map.putAll(map1);
48           System.out.println(map);//{33=cc, 22=bb, 11=aa, 44=cc}
49           System.out.println(map.containsKey(22));//true
50           System.out.println(map.containsValue("bb"));//true
51           map.replace(44, "FF");
52           System.out.println(map);//{33=cc, 22=bb, 11=aa, 44=FF}
53           map.clear();
54           System.out.println(map);//{}
55
56       }
57   }
```

4.4 entrySet方法

代码块

```
1   package com.powernode.collection12;
```

```java

import java.util.*;

public class Test03 {
    public static void main(String[] args) {
        Map<Integer, String> map = new HashMap<>();
        map.put(11, "aa");
        map.put(22, "bb");
        map.put(33, "cc");
        System.out.println(map);//{33=cc, 22=bb, 11=aa}
        //Entry 是 Map的内部接口
        Set<Map.Entry<Integer, String>> entries = map.entrySet();
        Iterator<Map.Entry<Integer, String>> iterator = entries.iterator();
        while (iterator.hasNext()) {
            Map.Entry<Integer, String> entry = iterator.next();
            Integer key = entry.getKey();
            String value = entry.getValue();
            System.out.println(key + ":" + value);
        }
    }
}
```

作业

1. 练习题
   a. 编写程序，在main方法中创建一个字符串数组
   b. 存储5个无序的字符串　　String [] sArray = {"12","11","13","16","15"};
   c. 创建TreeSet类型的集合，将数组中无序数字添加到集合中；
   d. 增强型for循环遍历该集合，打印所有元素。并求和
2. 练习题
   a. 改写上一节练习程序，使用迭代器遍历集合，打印所有元素，并将所有元素之和打印出来。
3. 练习题
   a. 编写程序，在main方法中创建Map集合（使用泛型），用来存放圆的半径（key）和面积（value）；
   b. 以半径为key，面积为value，将半径1-50的圆面积数据(按四舍五入取整)保存其中；
   c. 将Map中的半径数据取至Set集合中；
   d. 遍历Set集合的半径，逐一从Map中取出对应的面积值，并将半径和面积打印出来。

```
代码块
```

```java
package com.powernode.exercise01;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

public class Test03 {
    public static void main(String[] args) {
        /**
         * 练习题
         * 1. 编写程序，在main方法中创建Map集合（使用泛型），用来存放圆的半径（key）和面积（value）；
         * 2. 以半径为key，面积为value，将半径1-50的圆面积数据(按四舍五入取整)保存其中；
         * 3. 将Map中的半径数据取至Set集合中；
         * 4. 遍历Set集合的半径，逐一从Map中取出对应的面积值，并将半径和面积打印出来。
         */
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 1; i <= 50; i++) {
            /*System.out.println(i);
            System.out.println(Math.round(3.14 * i * i));*/
            map.put(i, (int) Math.round(3.14 * i * i));
        }
        //System.out.println(map);
        Set<Integer> integers = map.keySet();
        Iterator<Integer> iterator = integers.iterator();
        while (iterator.hasNext()) {
            Integer next = iterator.next();
            System.out.println("半径:" + next);
            System.out.println("面积: " + map.get(next));//通过key获得value
        }

    }
}
```

4. 练习题

   a. 编写程序，在类中声明一个calcSum方法，该方法可接收泛型类型参数为Integer或Float的集合，在方法中打印输出集合中所有元素，并计算和打印所有元素的整数和；

   b. 在main方法中：

      i. 创建一个List< Integer >类型的集合，保存10个100以内的随机整数，然后调用calcSum计算总和

ii. 创建一个List< Float>类型的集合，保存10个100以内的随机小数，然后调用calcSum计算总和

iii. 注：在java.lang包中，Number是Integer类和Float类的父类。

代码块

```
package com.powernode.exercise01;

import java.util.*;

public class Test04 {
    public static void main(String[] args) {
        /**
         * 1. 编写程序，在类中声明一个calcSum方法，该方法可接收泛型类型参数为Integer或Float的集合，
         * 在方法中打印输出集合中所有元素，并计算和打印所有元素的整数和；
         * 2. 在main方法中：
         *    1. 创建一个List< Integer >类型的集合，保存10个100以内的随机整数，然后调用calcSum计算总和
         *    2. 创建一个List< Float>类型的集合，保存10个100以内的随机小数，然后调用calcSum计算总和
         *       注：在java.lang包中，Number是Integer类和Float类的父类。
         */
        List<Integer> li = new ArrayList<>();
        for (int i = 0; i < 10; i++) {
            //System.out.println(new Random().nextInt(100));
            //System.out.println(Math.round(Math.random() * 100));
            li.add((int)Math.round(Math.random() * 100));
        }
        calcSum(li);


        List<Float> lf = new ArrayList<>();
        for (int i = 0; i < 10; i++) {
            //System.out.println(Math.random() * 100);
            lf.add((float)(Math.random() * 100));
        }
        calcSum(lf);
    }

    private static void calcSum(List<? extends Number> list) {

        int sum = 0;
        float fsum = 0;
        Iterator<? extends Number> iterator = list.iterator();
```

```
37          if(list.isEmpty()) return;
38          boolean flag = list.getFirst() instanceof Integer;
39          while (iterator.hasNext()) {
40              Number next = iterator.next();
41              if (flag) {
42                  sum += (int)next;
43              }else{
44                  fsum += (float)next;
45              }
46          }
47          if (flag) {
48              System.out.println(sum);
49          }else{
50              System.out.println(fsum);
51          }
52
53      }
54  }
```

4.5