第8章

工具类库与数据结构接口

面向对象程序设计

8.1 Java语言的工具类库概述

在**Java**语言提供的工具类库中包含了大量的标准类，有效地利用这些标准类可以使程序设计开发人员从繁杂的基础开发中解脱出来，这是面向对象程序设计开发方法倡导的软件重用的具体体现，是缩短软件开发周期的主要途径，是提高软件产品质量的关键所在。

java.applet 包含了有关**Applet**应用程序的所有类。

java.awt 包含了所有与图形用户界面及事件处理有关的类。

java.Swing 包含了所有与图形用户界面及事件处理有关的类。

java.beans 包含了所有与**JavaBeans**组件模型有关的类。

java.io 包含了与输入输出有关的类和接口。

java.lang 包含了许多**Java**语言的核心类。

java.math 包含了支持任意精度的整数和浮点数运算的类。

java.net 包含了支持与其他系统进行网络连接的类和接口。

java.security 包含了支持访问控制和认证的类和接口，

java.util 包含了大量实用工具类和接口。

8.2 几种常用的工具类库

在**java.util**包中包含了很多工具类，例如，产生随机数的**Random**、表示日期的**Date**、与数组作用相同，但操作方式更加灵活的向量**Vector**等，充分地利用这些工具类可以提高程序设计的效率，降低程序运行的出错概率，改善最终程序的可维护性。

8.2.1 随机数类

- 生成随机数是许多程序设计语言提供的一种功能。**Java**语言提供了一个**Random**类。使用这个类可以创建各种各样、相互独立的随机数发生器，从而满足应用程序的各种需求。
- **Random**类的每个对象都是一个随机数发生器，它们可以产生**int**、**long**、**float**或**double**类型的随机数。在产生随机数时，根据由成员方法的参数带入的“种子（**seed**）”值选定相应的算法生成不同的数值序列。

Random类的基本构成

- 在**Random**类中，有一个**private**的**long**类型的成员变量**seed**，它记录了每个对象对应的随机数发生器的“种子”，“种子”决定了随机数发生器生成随机数时采用的具体方式。
- **Random**类提供了两个构造方法。一个是不带参数的默认型构造方法，该方法将获取计算机时钟的当前时间作为“种子”值创建随机数发生器对象；另一个则带有一个**long**类型的参数作为“种子”值，使用这个构造方法可以显式地为随机数发生器指定一个“种子”值。

面向对象程序设计

Random类提供的成员方法

- **nextInt()** 返回一个**int**类型的随机数。
- **nextInt(int limit)** 返回一个大于或等于**0**且小于**limit**的**int**类型的随机数。
- **nextLong()** 返回一个**long**类型的随机数。
- **nextFloat()** 返回一个**float**类型的随机数。
- **nextDouble()** 返回一个**double**类型的随机数。
- **nextGaussion()** 返回一个**Gaussion**分布的**double**类型的随机数。
- **nextBoolean()** 返回**true**或**false**作为随机数值。
- **nextByte(byte[] bytes)** 用产生的随机数为数组**bytes**中的每个元素赋值。
- **setSeed(long seed)** 将“种子”设置为**seed**。

例 8.2.1 掷骰子游戏。

该游戏的玩法是：掷两个骰子，如果投掷的结果都为6，则输出“You win!!”，并结束应用程序的执行；如果投掷6次还没有取得上述结果，就输出“Sorry,you lost..”。

面向对象程序设计

```
public class Simulator
{
    public static void main(String[] args)
    {
    Random diceValues=new Random();
        String[] theThrow=
              {"First","Second","Third","Fourth","Fifth","Sixth"};
        int die1=0;     int die2=0;
        System.out.println("You have six throws of a pair of dice.\n“
         +"The objective is to get a double six.Here goes...\n");
    for (int i=0;i<6;i++)  {
            die1=1+diceValues.nextInt()%6;
            die2=1+diceValues.nextInt()%6;
            System.out.println(theThrow[i]+"throw: "+die1+", "+die2);
            if (die1+die2==12) { System.out.println("    You win!!");    return; }
        }
        System.out.println("Sorry,you lost...");    return;
    }
}
```

在这个程序中，利用默认的构造方法创建一个**Random**对象，因此在每次运行这个程序时，随机数发生器都将会以当前的系统时间作为“种子”值，产生随机数序列，由于每次运行程序时的系统时间不会相同，所以产生的随机数序列也不会相同。

8.2.2 日期类(Date)

用于对日期进行比较的三个成员方法:

在Date类中，提供了三个用于对日期进行比较的public成员方法。在应用程序中，可以利用它们很容易地判断两个日期:

after(Date earlier) **如果当前对象记录的日期晚于参数带入的日期，则返回true，否则返回false。**

before(Date later) **如果当前对象记录的日期早于参数带入的日期，则返回true，否则返回false。**

equals(Object aDate) **如果当前对象记录的日期与参数带入的日期一样，则返回true，否则返回false。**

日期显示格式

在Java的java.text包中提供了一个抽象类DateFormat，它描述了Date对象的String表示形式。在DateFormat类中，定义了4个代表不同日期表示法的常量。

SHORT 日期或时间用完整的数字格式表示。

MEDIUM 日期用相应语种月份的缩写形式与数字相结合的格式表示。

LONG 日期用相应语种月份的缩写形式与数字相结合的格式表示。

FULL 日期或时间的综合表示格式。

面向对象程序设计

由于DateFormat类是抽象的，所以不能够直接地创建该类的对象，但可以利用该类中定义的static成员方法获取DateFormat类对象。

getTimeInstance(int timeStyle,Locale aLocale)

用timeStyle指定的时间格式返回aLocale指定的地区的格式化时间。

getDateInstance(int dateStyle,Locale aLoacle)

用dateStyle指定的日期格式返回aLocale地区的格式化日期对象。

getDateTimeInstance(int dateStyle,
int timeStyle,Locale aLocale)

分别用dateStyle和timeStyle指定的格式返回aLocale地区的格式化日期时间对象。

```
public class TryDateFormats
{
    public static void main(String[] args)
    {
      Date today=new Date();
      Locale[] locales={Locale.US,Locale.UK,Locale.GERMANY,Locale.FRANCE};
      int[] styles={DateFormat.FULL,DateFormat.LONG,
                    DateFormat.MEDIUM,DateFormat.SHORT};
      DateFormat fmt;
      String[] styleText={"FULL","LONG","MEDIUM","SHORT"};
      for (int i=0;i<locales.length;i++)  {
        System.out.println("\nThe Date for "+locales[i].getDisplayCountry()+" : ");
        for (int j=0;j<styles.length;j++)    {
          fmt=DateFormat.getDateInstance(styles[j],locales[i]);
          System.out.println("\tIn "+styleText[j]+" is "+fmt.format(today));
        }
      }
    }
}
```


面向对象程序设计

8.2.3 向量类

几乎所有的程序设计语言都提供了数组类型。在大多数情况下，用数组存放一组数据类型相同的元素序列是一种明智的选择。在程序中，数组往往担负着举足轻重的作用，它所提供的操作手段往往决定着整个程序的复杂程度。另外，在现行的程序设计过程中，越来越渴望能够有相应的技术支持将不同类型的元素组合在一起的能力，由此，Java语言除了保留大家都很熟悉的数组外，还提供了Vector（向量）类。它不但具有数组类型的全部功能，还在元素的组织形式和操作手段方面有了很大改进，使得应用程序的操作方式变得更加方便、灵活。

一、Vector类

Vector是定义在java.util包中的一个public类，通常被称为向量类。在这个类中，定义了三个protected成员变量，它们分别为：

protected Object elementData[];　//存放向量元素的数组

protected int elementCount;　　//向量允许存放的最多元素个数

protected int capacityIncrement;　　//每次扩展向量单元的个数

Vector类提供的构造方法

```
public Vector()

public Vector(int initialCapacity)

public Vector(int initialCapacity,
                          int capacityIncrement)

public Vector(Collection c)
```

面向对象程序设计

二、Vector类提供的主要成员方法

public int capacity()
public int size()
public void copyInto(Object anArray[])
public int indexOf(Object elem)
public Object elementAt(int index)
public void setElementAt(Object obj, int index)
public void removeElementAt(int index)
public void insertElementAt(Object obj, int index)
public void addElement(Object obj)
public boolean removeElement(Object obj)
public Object clone()
public Object get(int index)
public boolean add(Object o)
public Object remove(int index)
public String toString()

应用举例

```
class Person        //人员
{
   private String firstName;        //名字
   private String surName;          //姓氏
   public Person(String firstName,String surName)
   {
      this.firstName=firstName;
      this.surName=surName;
    }
    public String toString()  {
      return firstName+"  "+surName;     }
}
```

```
class Crowd      //人群
{
   private Vector people;
   public Crowd(){people=new Vector();}
   public Crowd(int numPersons)
   {  people=new Vector(numPersons);   }
   public boolean add(Person someone)
   {  return people.add(someone);  }
   public Person get(int index)
   {  return (Person)people.get(index);  }
   public int size() {  return people.size();  }
   public int capacity() {return people.capacity(); }
}
```

```
public class TryVector
{
        public static void main(String[] args)
        {
          Person aPerson;
          Crowd filmCast=new Crowd();
          for (;;)  {
            aPerson=readPerson();    if (aPerson==null)  break;
            filmCast.add(aPerson);
          }
          int count=filmCast.size();
          System.out.println("You added "+count+(count==1?" person":" people")
+" to the cast.\n");
          for (int i=0;i<count;i++)    {
            aPerson=filmCast.get(i);      System.out.println(aPerson);
          }
    }
    public static Person readPerson()     //输入每个人的姓名
    {
       FormattedInput in=new FormattedInput();
       System.out.println("\nEnter first name or ! to end:");
       String firstName=in.stringRead().trim();
       if (firstName.charAt(0)=='!')  return null;
       System.out.println("Enter surname:");
       String surName=in.stringRead().trim();
       return new Person(firstName,surName);
    }
}
```

8.2.4 字符串类

在前面的应用程序中，大都使用的都是**String**类。**String**类是用来描述字符串常量的。即利用**String**类对象表示的字符串不能进行任何编辑操作。比如，在字符串中插入一个字符，删除字符串中的某个字符，更改字符串中某个位置的字符内容等等。下面我们介绍**Java**类库中提供的另外一个字符串标准类**StringBuffer**。它与**String**类的主要区别就是可以对**StringBuffer**类表示的字符串实施各种各样的编辑操作，因此，又被称为可变字符串。

StringBuffer类被定义在java.lang包中。其中含有三个private成员变量。它们分别是：

private char value[]：用来存放字符串内容的缓冲区。

private int count：缓冲区中存放字符的个数。

private boolean shared：缓冲区是否共享的标志。

面向对象程序设计

StringBuffer提供的三个构造方法

public StringBuffer()：无参的构造方法。默认开辟存放**16**个字符的缓冲区。

public StringBuffer(int length)：开辟存放**length**个字符的缓冲区。

public StringBuffer(String str)：开辟存放字符串的缓冲区大小为**str**的长度加上**16**，并将**str**字符串内容存入缓冲区。

StringBuffer提供的部分

public int length()

public int capacity()

public char charAt(int index)

public void getChars(int srcBegin, int srcEnd,

char dst[], int dstBegin)

public void setCharAt(int index, char ch)

public StringBuffer append(obj)

public StringBuffer delete(int start, int end)

面向对象程序设计

StringBuffer提供的部分

```
public StringBuffer replace(int start,
                            int end, String str)
public String substring(int start, int end)
public StringBuffer insert(int index,
                           char str[], int offset, int len)
public StringBuffer insert(int offset,
                           Object obj)
public int indexOf(String str, int fromIndex)
public String toString()
```

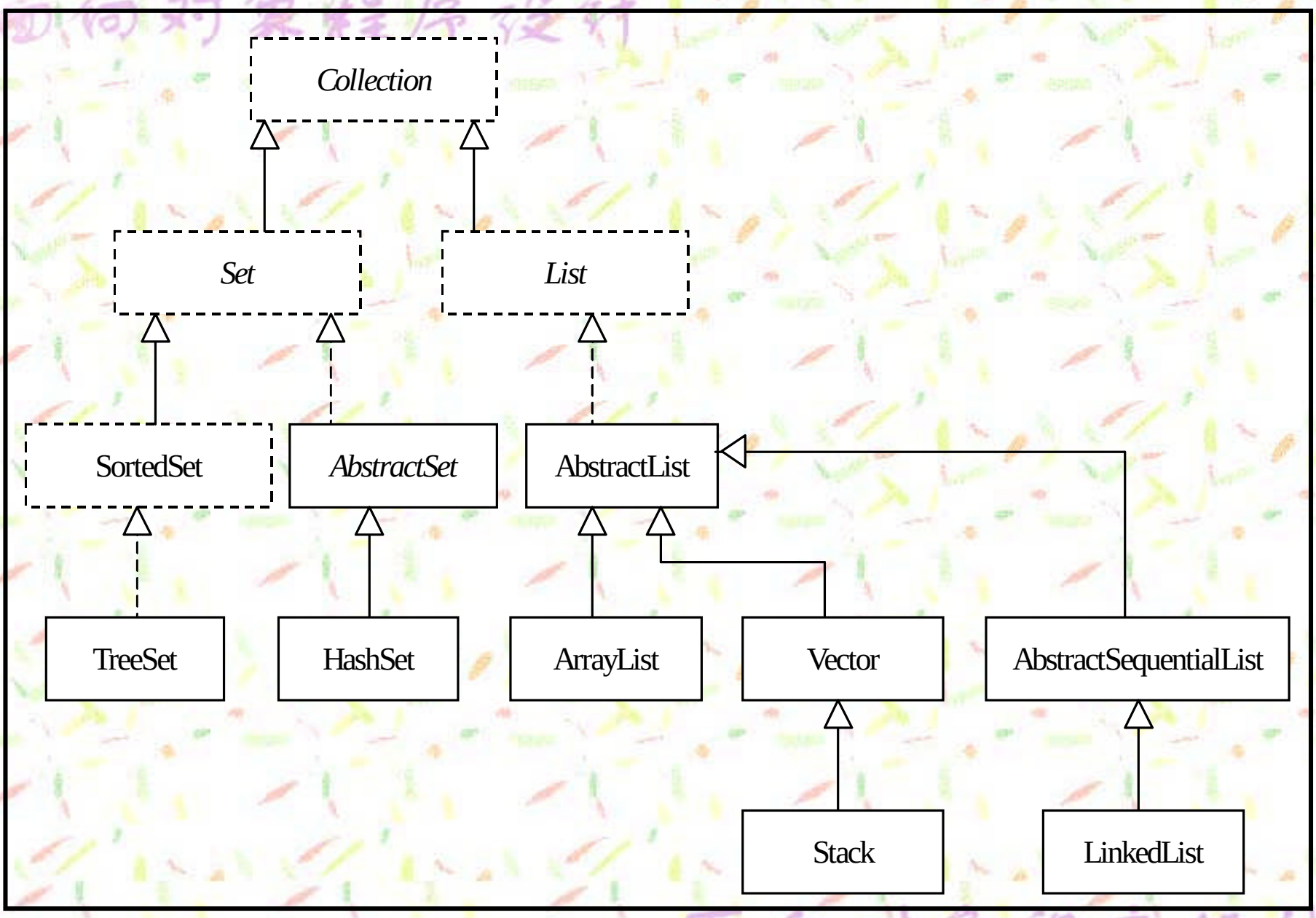
面向对象程序设计

```
public class TryStringBuffer
{
    public static void main(String[] args) throws IOException
    {
      BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
      for (;;)  {
        System.out.print("\n> ");      String line=in.readLine();
        if ((line==null)||line.equals("quit"))  break;
        StringBuffer buf=new StringBuffer(line);
        for (int i=0;i<buf.length();i++)buf.setCharAt(i,code(buf.charAt(i)));
        System.out.println(buf);
      }
    }
    public static char code(char c)
    {   if ((c>='A')&&(c<='Z')|| (c>='a')&&(c<='z')) {  c+=13;  }
       return c;
     }
}
```

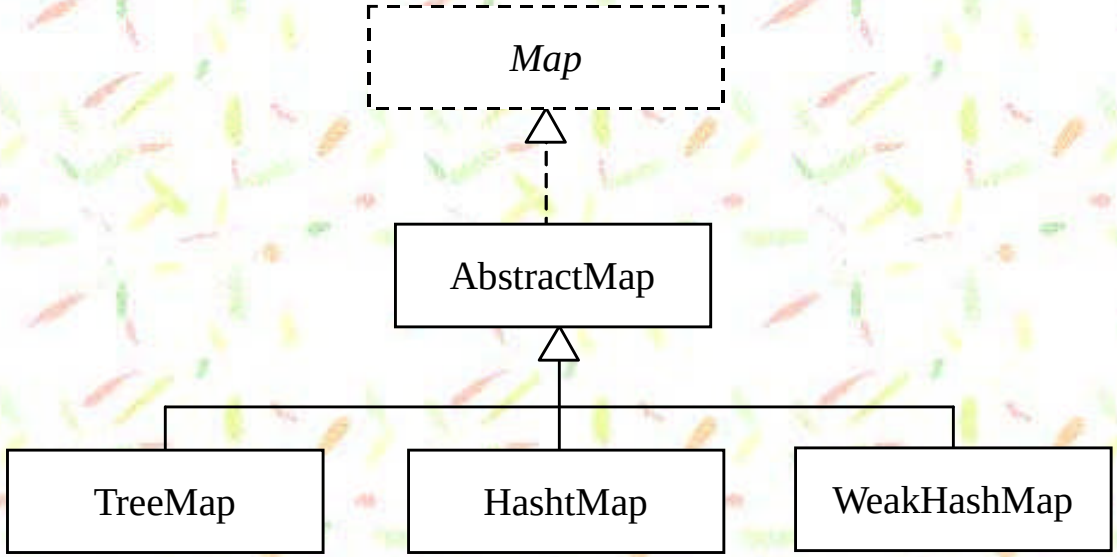

面向对象程序设计

8.3 基本的数据结构接口

集合是一组对象的整体，其中的每个对象被称为集合的元素。从严格意义上说，集合中没有重复的元素，每个元素之间也没有任何顺序关系。但随着计算机应用范围的不断扩大，人们将程序中所提及的集合概念加以扩展，即集合中也可以有重复的元素，元素之间也可以人为地规定某种顺序关系。这样就形成了今天应用十分广泛的集合、链表和映射几种数据结构。在**Java**语言的标准类库中，提供了丰富的数据结构接口和类，从而使得人们可以很轻松地在程序中操纵各种过去看起来比较复杂的数据结构。


Collection
Set
List
SortedSet
AbstractSet
AbstractList
TreeSet
HashSet
ArrayList
Vector
AbstractSequentialList
Stack
LinkedList

与Map接口有关的类关系图



面向对象程序设计

8.3.1 Collection接口

```
public interface Collection {
    int size();                  //返回集合中的元素个数
    boolean isEmpty();           //判集合是否为空
    boolean contains(Object o);  //判一个对象是否包含在集合中
    Iterator iterator();         //返回集合的迭代器
    Object[] toArray();          //将集合中的所有元素存入一个数组中返回
    Object[] toArray(Object a[]);  //将集合中的元素存入数组中返回
    boolean add(Object o);       //将对象o添加到集合中
    boolean remove(Object o);    //从集合中删去对象o
    boolean containsAll(Collection c); //判本集合是否包含集合c
    boolean addAll(Collection c);  //将集合c中的所有元素添加到本集合中
    boolean removeAll(Collection c);  //删除所有包含在集合c中的元素
    boolean retainAll(Collection c);  //删除没有包含在集合c中的元素
    void clear();                //清空集合
    boolean equals(Object o);    //比较集合与对象o是否相等
    int hashCode();              //返回对象的hash码
}
```

8.3.2 Set接口

- Set接口是由Collecton派生而来的，它被用来描述无重复元素的集合，其内部并没有声明新的成员方法，只是限定不能存在重复的元素。
- Set接口派生了一个接口SortedSet和一个抽象类AbstractSet。
- SortedSet接口用来描述有序集合，TreeSet类实现了这个接口，这个类描述了一个按升序排列的集合。而抽象类AbstractSet实现了部分Collection接口，并有一个子类HashSet，它以散列方式表示集合内容。

面向对象程序设计

```
public class TryTreeSet
{
        public static void main(String[] args)
        {
            Random Rvalue=new Random();
            TreeSet  tree=new TreeSet();
             Integer data;

            for (int i=0;i<10;i++){
                data=new Integer(Rvalue.nextInt()%1000);
                tree.add(data);
            }
            Iterator it=tree.iterator();
            while (it.hasNext())
              System.out.print(it.next()+"  ");
    }
}
```

面向对象程序设计

8.3.3 List接口

链表是有顺序关系的集合。这种顺序关系可以由插入的时间先后决定，也可以由元素值的大小决定。为了保证这种顺序关系，在插入或访问这种结构中的元素时，需要指定元素的位置。因此，**List**接口除了继承**Collection**接口的所有成员方法外，还声明了几个与位置有关的方法。

void add(int index, Object element)

Object remove(int index)

int indexOf(Object o)

int lastIndexOf(Object o)

面向对象程序设计

ArrayList类提供的部分成员方法

public ArrayList() 无参数的构造方法。

public ArrayList(int initialCapacity)带参数的构造方法。

public ArrayList(Collection c) 带参数的构造方法。

public Object clone() 覆盖拷贝方法。

public void ensureCapacity(int minCapacity)

重定义**ArrayList**对象存放链表元素的最小容量。

public void trimToSize()

将**ArrayList**对象中多余的空间释放。

LinkedList类提供的部分成员方法

public LinkedList() 无参数的构造方法。

public LinkedList(Collection c) 带参数的构造方法。

public void addFirst(Object o) 将对象o添加在链表的最前面。

public void addLast(Object o) 将对象o添加在链表的最后面。

public Object getFirst() 返回链表的第一个对象元素。

public Object getLast() 返回链表的最后一个对象元素。

public Object removeFirst() 删除第一个对象元素。

public Object removeLast() 删除最后一个对象元素。

public Object clone() 覆盖拷贝方法。

```
public class TryArrayList
{
    public static void main(String[] args)
    {
        ArrayList  List=new ArrayList(20);
        int i;
        List.add(new Integer(2));
        for (int primes=3;primes<1000;primes=primes+2)  {
          for (i=2;i<=Math.rint(Math.sqrt(primes));i++)
            if (primes%i==0) break;
          if (i>Math.rint(Math.sqrt(primes))){  List.add(new Integer(primes));}
        }
        for (i=0;i<List.size();i++){
           if (i%6==0) System.out.println();
           System.out.print(List.get(i)+"\t");
        }
  }
}
```


面向对象程序设计

```
public class TryLinkedList
{
    public static void main(String[] args)
    {
        LinkedList  Link=new LinkedList();
        int i;
        Integer data;

        Link.addLast(new Integer(2));
        for (int primes=3;primes<1000;primes=primes+2)   {
            for (i=2;i<=Math.rint(Math.sqrt(primes));i++)
              if (primes%i==0) break;
            if (i>Math.rint(Math.sqrt(primes))) {ink.addLast(new Integer(primes));  }
        }
        for (i=0;i<Link.size();i++){
          if (i%6==0) System.out.println();    System.out.print(Link.get(i)+"\t");
        }
    }
}
```

8.3.4 Map接口

- 映像是一种存储集合元素的方式。它将每个元素与一个键值对应，由键值决定每个元素所存放的位置，因此要求在集合中加入元素时同时给出键值和元素值。其主要优点是检索速度最快，即检索元素的平均比较次数最少。
- 在**Java**语言中，所有用映像方式表示数据结构的标准操作都声明在**Map**接口中。

面向对象程序设计

```
public interface Map
{
    int size();
    boolean isEmpty();
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    Object get(Object key);
    Object put(Object key, Object value);
    Object remove(Object key);
    void putAll(Map t);
    void clear();
    Set keySet();
    Collection values();
    Set entrySet();
    interface Entry {
      Object getKey();
        Object getValue();
        Object setValue(Object value);
boolean equals(Object o);
        int hashCode();
    }
    boolean equals(Object o);
    int hashCode();
}
```


面向对象程序设计

HashMap类提供的4个构造方法

HashMap() 无参的构造方法。

HashMap(int capacity)

创建容量为capacity，装填因子为0.75的散列表。

HashMap(int capacity,float loadFactor)

创建容量为capacity，装填因子为loadFactor的散列表。

HashMap(Map map)

创建一个散列表，其容量与装填因子为map对象的相应值。

面向对象程序设计

HashMap类提供的有关存储、检索和删除对象的几个成员方法

put(Object key,Object value)

将用键值**key**存储对象**value**。

putAll(Map map)

将**map**中的所有键值/对象传递给当前的散列表。

get(Object key)

将返回键值**key**对应的对象。

remove(Object key)

将从散列表中删除**key**键值所对应的对象。

HashMap类提供的有关处理元素的成员方法

KeySet() 将返回一个**Set**对象，其内容为所有的键值。

entrySet() 将返回一个**Set**对象。

values() 将返回一个**Collection**对象。

getKey() 将返回**Map.Entry**对象的键值。

getValue() 将返回**Map.Entry**所对应的对象。

setValue(Object new) 将**Map.Entry**对象设置为**new**。