Chapter 22 – Collections

林偉川

1

Introduction

- Java collections framework
  - Provides reusable componentry
  - Common data structures
    - Example of code reuse
- Collection
  - Data structure (object) that can hold references to other objects
- Collections framework
  - Interfaces declare operations for various collection types
  - Belong to package java.util
    - Collection
    - Map
    - Set
    - List

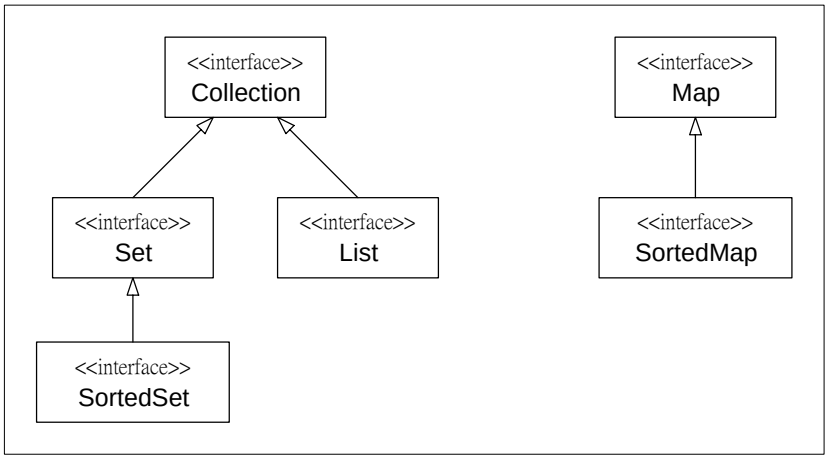
2

請注意Collections具體類別的相關特性：

- 類別實作的介面。
- 類別使用的資料結構。
- 元素可否重複。
- 無序或有序（加入的先後順序，或有排序功能）。
- 是否為執行緒同步（Thread safe class）。
- 有哪些常見的應用。

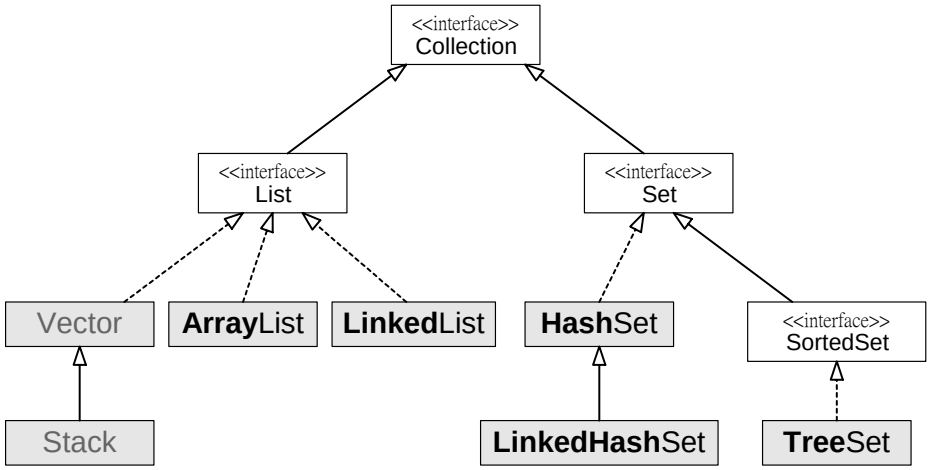
3

Collections Framework內的介面架構



4

實作Collections延伸介面的具體類別



5

Collection介面

- Collection介面定義的集合，其元素可以為無順序（non-ordered）和可重複（repetition allowed）。
- Collection兩個延伸介面List和Set，分別保有Collection的不同特性。
- **Set**的特點為，其元素不可重複。**SortedSet**介面繼承Set介面，且宣告了排序的方法。
- **List**介面繼承Collection介面，然而它是有順序的，而且是可以重複的。

6

Map介面

- Map介面和Collection介面沒有繼承的關係。
- Map介面的元素是「鍵值對」(key-value pairs)。
- Map中的key和value皆為參照變數，而且key不能重複，一個key對應到一個value。
- Map中的元素是沒有順序的。
- **SortedMap**介面為Map的子介面，故名思義，其為可排序的集合。

7

Collection中所使用的基本資料結構

- 陣列(array)
- 鏈結串列(linked list)
- 樹(tree)
- 雜湊表(hash table)

8

陣列的重要特性：

- 存取元素的速度快。
- 要在某位置插入或移除元素值時，並不方便。
- 陣列長度為固定，欲改變長度必須重新建立另一個陣列。

9

Class Arrays

- Class Arrays
  - Provides static methods for manipulating arrays
  - Provides “high-level” methods
    - Method binarySearch for searching sorted arrays
    - Method equals for comparing arrays
    - Method fill for placing values into arrays
    - Method sort for sorting arrays

10

Arrays

Arrays的靜態方法	說明
List **asList**(Object[] a)	將Object陣列a轉換成List物件，然後回傳。
int **binarySearch**(int[] a, int key)	在已排序的a陣列中以二次搜尋法，找出key的索引值。若找不到key則回傳 –1。此方法有許多因應不同型別陣列的多載方法。
boolean **equals**(int[] a, int[] a2)	比較兩陣列a和a2，若兩陣列中的元素值皆相等則回傳true。此方法有許多因應不同型別陣列的多載方法。
void **fill**(int[] a, int val)	將a陣列中的所有的元素值設定為val。此方法有許多因應不同型別陣列的多載方法。
void **sort**(int[] a)	對陣列a排序。此方法有許多因應不同參數的多載方法。

11

鏈結串列

- 可以鏈結的類別

```
class LinkedObj{
      Object data;          //節點資料
      LinkedObj next;       //下一個節點
}
```

- 鏈結串列的重要特性：
  - 插入或刪除節點很方便。
  - 變更鏈結串列的長度也很方便。
  - 存取節點的速度較慢。

12

Collections

- Collections中的靜態方法大都針對**List**，因為List不像Set和Map有可排序的類別。
- Collections另外提供將「非執行緒同步」集合包裹成「執行緒同步」集合的方法，這些方法在多執行緒程式裡會顯得相當方便。

13

Collections的常用靜態方法	說明
int **binarySearch**(List list, Object key)	以二次搜尋法，找出key的索引值。
void **copy**(List dest, List src)	將src序列的所元素複製給dest序列。
boolean **replaceAll**(List list, Object oldVal, Object newVal)	將list序列中的oldVal換成newVal物件。若oldVal不包含在list內則回傳false。
void **reverse**(List list)	將list序列的順序顛倒。
void **shuffle**(List list)	亂數重排list序列中元素的順序。
void **sort**(List list)	對list序列排序。
List **synchronizedList**(List list)	取得一執行緒同步的集合。
Map **synchronizedMap**(Map m)	
Set **synchronizedSet**(Set s)	
SortedMap **synchronizedSortedMap**(SortedMap m)	
SortedSet **synchronizedSortedSet**(SortedSet s)	

14

樹

- 實體可以做為二元樹的節點的類別

```
class TreeNode{
      Object data;
      TreeNode left;
      TreeNode right;
}
```

  - 樹的優點是有「排序」的功能，缺點和鏈結串列一樣，節點資料的存取都比較慢。

15

雜湊表

- 雜湊表（hash table）的資料是以「鍵值對」（key value paired）的形式存在。

- 加入一個元素時必須同時給予一個key和一個value，透過key就可以取得value。

- 雜湊表的特點：
  - 存取資料的速度快。
  - 較浪費記憶體空間。

16

是否相等

- Object.equals()的定義如下：

```
public boolean equals(Object obj){
     return (this == obj);
}
```

- 當兩個物件obj1和obj2使用equals()比較後認定為**相等時，他們由hashCode()方法所取得的雜湊碼也必須相同**。所以當兩物件相等時，以下兩個運算式的結果都必須為true。

```
obj1.equals(obj2)== obj2.equals(obj1)== true
obj1.hashCode()== obj2.hashCode()
```

17

是否相等

- hashCode()方法在使用hash table的集合中顯得相當重要，因為hash table是以key.hashCode()取得的hash code value。

- 當你覆蓋equals()方法時，也必須覆蓋hashCode()方法，讓兩者的行為一致。

18

是否相等

- Integer類別中定義的equals()

```
public boolean equals(Object obj){
   if(obj instanceof Integer){
      return value == ((Integer)obj).intValue();
   }
   return false;
}
```

- Integer類別中定義的hashCode()

```
public int hashCode(){
     return value;
}
```

19

String類別的equals()方法

```
public boolean equals(Object anObject){
   if (this == anObject){
      return true;                          //參照相同
   }
   if (anObject instanceof String){
      String anotherString = (String)anObject;
      int n = count;                        //字串長度
      if (n == anotherString.count){
         char v1[] = value;                 //字元陣列
         char v2[] = anotherString.value;
         int i = offset;                    //第一字元的位置
         int j = anotherString.offset;
         while (n-- != 0){
            if (v1[i++] != v2[j++])
               return false;                //有任何字元不同
         }
         return true;                       //所有字元都相同
      }
   }
   return false;                            //不是String型別
}
```

20

是否相等

- String類別的hashCode()方法

```
public int hashCode(){
    int h = hash;                 //取得字串的雜湊碼
    if(h == 0){                   //若雜湊碼為0表示尚未計算
        int off = offset;         //第一個字元的位置
        char val[] = value;       //取得字元陣列
        int len = count;          //字串的長度
        for (int i = 0; i < len; i++){
            //計算雜湊碼的規則
            h = 31*h + val[off++];
        }
        hash = h;
    }
    return h;
}
```

```
s[0]*31^(n-1)+ s[1]*31^(n-2)+ ... + s[n-1]
```

21

是否相等

- 自訂類別的equals()和hashCode()
  - 定義equals()時，可以一一比對各個屬性是否相等。屬性大多為現成的類別或基本資料型別，因此，就可以使用現成的類別的equals()方法。
  - 定義hashCode()時，只要將屬性的雜湊碼相互做位元互斥（XOR，^）運算即可。

22

較大或較小

- **Tree**Set和**Tree**Map都有自動排序的功能，既然要能排序必定有比較大小的規則。
- 使用tree實作的collection，其元素必須是相同型別的物件，而且必須實作**Comparable**介面。
- **String類別**和**基本型別的包裝器**（Boolean除外）實作了Comparable介面。

23

較大或較小

- Comparable介面只宣告一個抽象方法compareTo()

```
int compareTo(Object o);
```

- 實作compareTo()方法必須遵守
  - 不同型別的物件不能比較
  - 若物件本身比傳入的物件小時，則回傳負值。
  - 若物件本身比傳入的物件大時，則回傳正值。
  - 若不大也不小時，回傳0。

24

```
1  // Fig. 22.1: UsingArrays.java
2  // Using Java arrays.
3  import java.util.*;
4
5  public class UsingArrays {
6     private int intValues[] = { 1, 2, 3, 4, 5, 6 };
7     private double doubleValues[] = { 8.4, 9.3, 0.2, 7.9, 3.4 };
8     private int filledInt[], intValuesCopy[];
9
10    // initialize arrays
11    public UsingArrays()
12    {
13       filledInt = new int[ 10 ];
14       intValuesCopy = new int[ intValues.length ];
15
16       Arrays.fill( filledInt, 7 );   // fill with 7s
17
18       Arrays.sort( doubleValues );   // sort doubleValues ascending
19
20       // copy array intValues into array intValuesCopy
21       System.arraycopy( intValues, 0, intValuesCopy,
22          0, intValues.length );
23    }
24
```

Outline

UsingArrays.java

Line 16

Line 18

Lines 21-22

```
25    // output values in each array
26    public void printArrays()
27    {
28       System.out.print( "doubleValues: " );
29
30       for ( int count = 0; count < doubleValues.length; count++ )
31          System.out.print( doubleValues[ count ] + " " );
32
33       System.out.print( "\nintValues: " );
34
35       for ( int count = 0; count < intValues.length; count++ )
36          System.out.print( intValues[ count ] + " " );
37
38       System.out.print( "\nfilledInt: " );
39
40       for ( int count = 0; count < filledInt.length; count++ )
41          System.out.print( filledInt[ count ] + " " );
42
43       System.out.print( "\nintValuesCopy: " );
44
45       for ( int count = 0; count < intValuesCopy.length; count++ )
46          System.out.print( intValuesCopy[ count ] + " " );
47
48       System.out.println();
49
50    } // end method printArrays
51
```

Outline

UsingArrays.java

```
52    // find value in array intValues
53    public int searchForInt( int value )
54    {
55       return Arrays.binarySearch( intValues, value );
56    }
57
58    // compare array contents
59    public void printEquality()
60    {
61       boolean b = Arrays.equals( intValues, intValuesCopy );
62
63       System.out.println( "intValues " + ( b ? "==" : "!=" ) +
64          " intValuesCopy" );
65
66       b = Arrays.equals( intValues, filledInt );
67
68       System.out.println( "intValues " + ( b ? "==" : "!=" ) +
69          " filledInt" );
70    }
71
```

Outline

UsingArrays.java

Line 55

Lines 61 and 66

```
72    public static void main( String args[] )
73    {
74       UsingArrays usingArrays = new UsingArrays();
75
76       usingArrays.printArrays();
77       usingArrays.printEquality();
78
79       int location = usingArrays.searchForInt( 5 );
80       System.out.println( ( location >= 0 ? "Found 5 at element " +
81          location : "5 not found" ) + " in intValues" );
82
83       location = usingArrays.searchForInt( 8763 );
84       System.out.println( ( location >= 0 ? "Found 8763 at element " +
85          location : "8763 not found" ) + " in intValues" );
86    }
87
88 } // end class UsingArrays
```

```
doubleValues: 0.2 3.4 7.9 8.4 9.3
intValues: 1 2 3 4 5 6
filledInt: 7 7 7 7 7 7 7 7 7 7
intValuesCopy: 1 2 3 4 5 6
intValues == intValuesCopy
intValues != filledInt
Found 5 at element 4 in intValues
8763 not found in intValues
```

Outline

UsingArrays.java

```
1  // Fig. 22.2: UsingAsList.java
2  // Using method asList.
3  import java.util.*;
4
5  public class UsingAsList {
6     private static final String values[] = { "red", "white", "blue" };
7     private List list;
8
9     // initialize List and set value at location 1
10    public UsingAsList()
11    {
12       list = Arrays.asList( values );   // get List
13       list.set( 1, "green" );           // change a value
14    }
15
16    // output List and array
17    public void printElements()
18    {
19       System.out.print( "List elements : " );
20
21       for ( int count = 0; count < list.size(); count++ )
22          System.out.print( list.get( count ) + " " );
23
24       System.out.print( "\nArray elements: " );
25
```

Outline

UsingAsList.java

Line 12

Line 13

Line 21

Line 22

```
26       for ( int count = 0; count < values.length; count++ )
27          System.out.print( values[ count ] + " " );
28
29       System.out.println();
30    }
31
32    public static void main( String args[] )
33    {
34       new UsingAsList().printElements();
35    }
36
37 } // end class UsingAsList
```

```
List elements : red green blue
Array elements: red green blue
```

Outline

UsingAsList.java

Interface Collection and Class Collections

- Interface Collection
  - Contains *bulk operations*
    - Adding, clearing, comparing and retaining objects
  - Interfaces Set and List extend interface Collection
- Class Collections
  - Provides static methods that manipulate collections
  - Collections can be manipulated polymorphically

31

Lists

- List
  - Ordered Collection that can contain duplicate elements
  - Sometimes called a *sequence*
  - Implemented via interface List
    - ArrayList
    - LinkedList
    - Vector

32

```
1  // Fig. 22.3: CollectionTest.java
2  // Using the Collection interface.
3  import java.awt.Color;
4  import java.util.*;
5
6  public class CollectionTest {
7     private static final String colors[] = { "red", "white", "blue" };
8
9     // create ArrayList, add objects to it and manipulate it
10    public CollectionTest()
11    {
12       List list = new ArrayList();
13
14       // add objects to list
15       list.add( Color.MAGENTA );     // add a color object
16
17       for ( int count = 0; count < colors.length; count++ )
18          list.add( colors[ count ] );
19
20       list.add( Color.CYAN );        // add a color object
21
22       // output list contents
23       System.out.println( "\nArrayList: " );
24
25       for ( int count = 0; count < list.size(); count++ )
26          System.out.print( list.get( count ) + " " );
27
```

Outline

CollectionTest.java

Lines 15-20

Line 26

```
28       // remove all String objects
29       removeStrings( list );
30
31       // output list contents
32       System.out.println( "\n\nArrayList after calling removeStrings: " );
33
34       for ( int count = 0; count < list.size(); count++ )
35          System.out.print( list.get( count ) + " " );
36
37    } // end constructor CollectionTest
38
39    // remove String objects from Collection
40    private void removeStrings( Collection collection )
41    {
42       Iterator iterator = collection.iterator(); // get iterator
43
44       // loop while collection has items
45       while ( iterator.hasNext() )
46
47          if ( iterator.next() instanceof String )
48             iterator.remove();  // remove String object
49    }
50
```

Outline

CollectionTest.java

Line 29

Line 42

Line 45

Line 47

Line 48

```
51    public static void main( String args[] )
52    {
53       new CollectionTest();
54    }
55
56 } // end class CollectionTest
```

```
ArrayList:
java.awt.Color[r=255,g=0,b=255] red white blue java.awt.Color [r=0,g=255,b=255]

ArrayList after calling removeStrings:
java.awt.Color[r=255,g=0,b=255] java.awt.Color[r=0,g=255,b=255]
```

Outline

CollectionTest.java

```
1  // Fig. 22.4: ListTest.java
2  // Using LinkLists.
3  import java.util.*;
4
5  public class ListTest {
6     private static final String colors[] = { "black", "yellow",
7        "green", "blue", "violet", "silver" };
8     private static final String colors2[] = { "gold", "white",
9        "brown", "blue", "gray", "silver" };
10
11    // set up and manipulate LinkedList objects
12    public ListTest()
13    {
14       List link = new LinkedList();
15       List link2 = new LinkedList();
16
17       // add elements to each list
18       for ( int count = 0; count < colors.length; count++ ) {
19          link.add( colors[ count ] );
20          link2.add( colors2[ count ] );
21       }
22
23       link.addAll( link2 );          // concatenate lists
24       link2 = null;                  // release resources
25
```

Outline

ListTest.java

Lines 14-15

Line 23

Line 24

```
26      printList( link );
27
28      uppercaseStrings( link );
29
30      printList( link );
31
32      System.out.print( "\nDeleting elements 4 to 6..." );
33      removeItems( link, 4, 7 );
34
35      printList( link );
36
37      printReversedList( link );
38
39   } // end constructor ListTest
40
41   // output List contents
42   public void printList( List list )
43   {
44      System.out.println( "\nlist: " );
45
46      for ( int count = 0; count < list.size(); count++ )
47         System.out.print( list.get( count ) + " " );
48
49      System.out.println();
50   }
```

Outline

ListTest.java

Lines 42-50

```
51
52   // locate String objects and convert to uppercase
53   private void uppercaseStrings( List list )
54   {
55      ListIterator iterator = list.listIterator();
56
57      while ( iterator.hasNext() ) {
58         Object object = iterator.next();  // get item
59
60         if ( object instanceof String )   // check for String
61            iterator.set( ( ( String ) object ).toUpperCase() );
62      }
63   }
64
65   // obtain sublist and use clear method to delete sublist items
66   private void removeItems( List list, int start, int end )
67   {
68      list.subList( start, end ).clear();  // remove items
69   }
70
71   // print reversed list
72   private void printReversedList( List list )
73   {
74      ListIterator iterator = list.listIterator( list.size() );
75
```

Outline

ListTest.java

Lines 53-63

Line 68

```
76       System.out.println( "\nReversed List:" );
77
78       // print list in reverse order
79       while( iterator.hasPrevious() )
80          System.out.print( iterator.previous() + " " );
81    }
82
83    public static void main( String args[] )
84    {
85       new ListTest();
86    }
87
88 } // end class ListTest
```

```
list:
black yellow green blue violet silver gold white brown blue gray silver

list:
BLACK YELLOW GREEN BLUE VIOLET SILVER GOLD WHITE BROWN BLUE GRAY
SILVER

Deleting elements 4 to 6...
list:
BLACK YELLOW GREEN BLUE WHITE BROWN BLUE GRAY SILVER

Reversed List:
SILVER GRAY BLUE BROWN WHITE BLUE GREEN YELLOW BLACK
```

Outline

ListTest.java

Line 79

Line 80

```
1  // Fig. 22.5: UsingToArray.java
2  // Using method toArray.
3  import java.util.*;
4
5  public class UsingToArray {
6
7     // create LinkedList, add elements and convert to array
8     public UsingToArray()
9     {
10       String colors[] = { "black", "blue", "yellow" };
11
12       LinkedList links = new LinkedList( Arrays.asList( colors ) );
13
14       links.addLast( "red" );   // add as last item
15       links.add( "pink" );      // add to the end
16       links.add( 3, "green" );  // add at 3rd index
17       links.addFirst( "cyan" ); // add as first item
18
19       // get LinkedList elements as an array
20       colors = ( String [] ) links.toArray( new String[ links.size() ] );
21
22       System.out.println( "colors: " );
23
```

Outline

UsingToArray.java

Line 20

```
24      for ( int count = 0; count < colors.length; count++ )
25         System.out.println( colors[ count ] );
26   }
27
28   public static void main( String args[] )
29   {
30      new UsingToArray();
31   }
32
33 } // end class UsingToArray
```

```
colors:
cyan
black
blue
yellow
green
red
pink
```

Outline

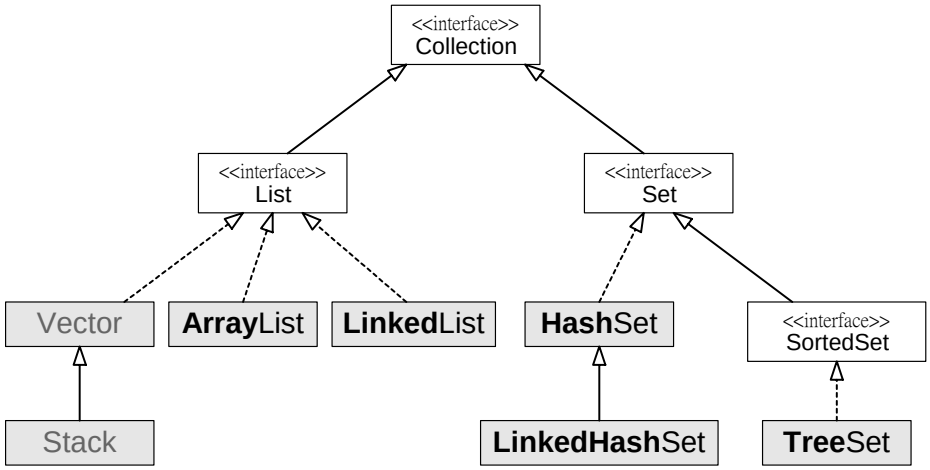
UsingToArray.java

請注意Collection具體類別的相關特性：

- 類別實作的介面。
- 類別使用的資料結構。
- 元素可否重複。
- 無序或有序（加入的先後順序，或有排序功能）。
- 是否為執行緒同步（Thread safe class）。
- 有哪些常見的應用。

42

實作Collection延伸介面的具體類別



43

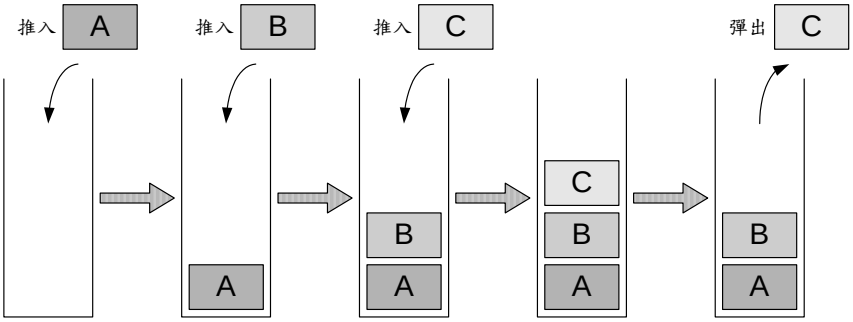
實作List介面的具體類別

- **ArrayList**使用array資料結構實作List。保有陣列的特性，存取元素的效率佳，但不利於插入或移除元素，且重定陣列長度效率差。
- **LinkedList**使用linked list實作List。保有linked list的特性，存取元素的效率較差，但利於插入元素、移除元素和改變長度。
- **Vector**和ArrayList很像，都是以array實作List。兩者最大的不同是在於：Vector為「執行緒同步類別」，而ArrayList則否。

44

實作List介面的具體類別

- **Stack**為Vector的延伸類別，所以同樣為有序、執行緒同步類別。Stack類別是堆疊的抽象資料型別（Abstract Data Type）。



45

Algorithms

- Collections Framework provides set of algorithms
  - Implemented as static methods
    - List algorithms
      - sort
      - binarySearch
      - reverse
      - shuffle
      - fill
      - copy
    - Collection algorithms
      - min
      - max

46

Algorithm sort

- sort
  - Sorts List elements
    - Order is determined by natural order of elements' type
    - Relatively fast

47

```
3  import java.util.*;
4
5  public class Sort1 {
6     private static final String suits[] =
7        { "Hearts", "Diamonds", "Clubs", "Spades" };
8
9     // display array elements
10    public void printElements()
11    {
12       // create ArrayList
13       List list = new ArrayList( Arrays.asList( suits ) );
14
15       // output list
16       System.out.println( "Unsorted array elements:\n" + list );
17
18       Collections.sort( list ); // sort ArrayList
19
20       // output list
21       System.out.println( "Sorted array elements:\n" + list );
22    }
23
24    public static void main( String args[] )
25    {
26       new Sort1().printElements();
27    }
28 } // end class Sort1
```

```
Unsorted array elements:
[Hearts, Diamonds, Clubs, Spades]
Sorted array elements:
[Clubs, Diamonds, Hearts, Spades]
```

Outline

Sort1.java

Line 13

Line 18

```
1  // Fig. 22.7: Sort2.java
2  // Using a Comparator object with algorithm sort.
3  import java.util.*;
4
5  public class Sort2 {
6     private static final String suits[] =
7        { "Hearts", "Diamonds", "Clubs", "Spades" };
8
9     // output List elements
10    public void printElements()
11    {
12       List list = Arrays.asList( suits ); // create List
13
14       // output List elements
15       System.out.println( "Unsorted array elements:\n" + list );
16
17       // sort in descending order using a comparator
18       Collections.sort( list, Collections.reverseOrder() );
19
20       // output List elements
21       System.out.println( "Sorted list elements:\n" + list );
22    }
23    public static void main( String args[] ) {    new Sort2().printElements();     }
24 } // end class Sort2
```

```
Unsorted array elements:
[Hearts, Diamonds, Clubs, Spades]
Sorted list elements:
[Spades, Hearts, Diamonds, Clubs]
```

Outline

Sort2.java

Line 18

Line 18

```
1  // Fig. 22.8: Sort3.java
2  // Creating a custom Comparator class.
3  import java.util.*;
4
5  public class Sort3 {
6
7     public void printElements()
8     {
9        List list = new ArrayList(); // create List
10
11       list.add( new Time2(  6, 24, 34 ) );
12       list.add( new Time2( 18, 14, 05 ) );
13       list.add( new Time2(  8, 05, 00 ) );
14       list.add( new Time2( 12, 07, 58 ) );
15       list.add( new Time2(  6, 14, 22 ) );
16
17       // output List elements
18       System.out.println( "Unsorted array elements:\n" + list );
19
20       // sort in order using a comparator
21       Collections.sort( list, new TimeComparator() );
22
23       // output List elements
24       System.out.println( "Sorted list elements:\n" + list );
25    }
26
```

Outline

Sort3.java

Line 21

```
27    public static void main( String args[] )
28    {
29       new Sort2().printElements();
30    }
31
32    private class TimeComparator implements Comparator {
33       int hourCompare, minuteCompare, secondCompare;
34       Time2 time1, time2;
35
36       public int compare(Object object1, Object object2)
37       {
38          // cast the objects
39          time1 = (Time2)object1;
40          time2 = (Time2)object2;
41
42          hourCompare = new Integer( time1.getHour() ).compareTo(
43                new Integer( time2.getHour() ) );
44
45          // test the hour first
46          if ( hourCompare != 0 )
47             return hourCompare;
48
49          minuteCompare = new Integer( time1.getMinute() ).compareTo(
50                new Integer( time2.getMinute() ) );
51
```

Outline

Sort3.java

Line 32

Line 36

```
52          // then test the minute
53          if ( minuteCompare != 0 )
54             return minuteCompare;
55
56          secondCompare = new Integer( time1.getSecond() ).compareTo(
57                new Integer( time2.getSecond() ) );
58
59          return secondCompare; // return result of comparing seconds
60       }
61
62    } // end class TimeComparator
63
64 } // end class Sort3
```

```
Unsorted array elements:
[06:24:34, 18:14:05, 08:05:00, 12:07:58, 06:14:22]
Sorted list elements:
[06:14:22, 06:24:34, 08:05:00, 12:07:58, 18:14:05]
```

Outline

Sort3.java

Algorithm shuffle

- shuffle
  - Randomly orders List elements

Outline

Cards.java

```
1  // Fig. 22.9: Cards.java
2  // Using algorithm shuffle.
3  import java.util.*;
4
5  // class to represent a Card in a deck of cards
6  class Card {
7     private String face;
8     private String suit;
9
10    // initialize a Card
11    public Card( String initialface, String initialSuit )
12    {
13       face = initialface;
14       suit = initialSuit;
15    }
16
17    // return face of Card
18    public String getFace()
19    {
20       return face;
21    }
22
23    // return suit of Card
24    public String getSuit()
25    {
26       return suit;
27    }
```

```
28
29    // return String representation of Card
30    public String toString()
31    {
32       StringBuffer buffer = new StringBuffer( face + " of " + suit );
33       buffer.setLength( 20 );
34
35       return buffer.toString();
36    }
37
38 } // end class Card
39
40 // class Cards declaration
41 public class Cards {
42    private static final String suits[] =
43       { "Hearts", "Clubs", "Diamonds", "Spades" };
44    private static final String faces[] = { "Ace", "Deuce", "Three",
45       "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten",
46       "Jack", "Queen", "King" };
47    private List list;
48
49    // set up deck of Cards and shuffle
50    public Cards()
51    {
52       Card deck[] = new Card[ 52 ];
53
```

Outline

Cards.java

```
54       for ( int count = 0; count < deck.length; count++ )
55          deck[ count ] = new Card( faces[ count % 13 ],
56             suits[ count / 13 ] );
57
58       list = Arrays.asList( deck );   // get List
59       Collections.shuffle( list );    // shuffle deck
60    }
61
62    // output deck
63    public void printCards()
64    {
65       int half = list.size() / 2 - 1;
66
67       for ( int i = 0, j = half + 1; i <= half; i++, j++ )
68          System.out.println( list.get( i ).toString() + list.get( j ) );
69    }
70
71    public static void main( String args[] )
72    {
73       new Cards().printCards();
74    }
75
76 } // end class Cards
```

Outline

Cards.java

Line 59

```
King of Diamonds    Jack of Spades
Four of Diamonds    Six of Clubs
King of Hearts      Nine of Diamonds
Three of Spades     Four of Spades
Four of Hearts      Seven of Spades
Five of Diamonds    Eight of Hearts
Queen of Diamonds   Five of Hearts
Seven of Diamonds   Seven of Hearts
Nine of Hearts      Three of Clubs
Ten of Spades       Deuce of Hearts
Three of Hearts     Ace of Spades
Six of Hearts       Eight of Diamonds
Six of Diamonds     Deuce of Clubs
Ace of Clubs        Ten of Diamonds
Eight of Clubs      Queen of Hearts
Jack of Clubs       Ten of Clubs
Seven of Clubs      Queen of Spades
Five of Clubs       Six of Spades
Nine of Spades      Nine of Clubs
King of Spades      Ace of Diamonds
Ten of Hearts       Ace of Hearts
Queen of Clubs      Deuce of Spades
Three of Diamonds   King of Clubs
Four of Clubs       Jack of Diamonds
Eight of Spades     Five of Spades
Jack of Hearts      Deuce of Diamonds
```

Outline

Cards.java

Algorithms reverse, fill, copy, max **and** min

- reverse
  - Reverses the order of List elements
- fill
  - Populates List elements with values
- copy
  - Creates copy of a List
- max
  - Returns largest element in List
- min
  - Returns smallest element in List

58

```
1  // Fig. 22.10: Algorithms1.java
2  // Using algorithms reverse, fill, copy, min and max.
3  import java.util.*;
4
5  public class Algorithms1 {
6     private String letters[] = { "P", "C", "M" }, lettersCopy[];
7     private List list, copyList;
8
9     // create a List and manipulate it with methods from Collections
10    public Algorithms1()
11    {
12       list = Arrays.asList( letters );     // get List
13       lettersCopy = new String[ 3 ];
14       copyList = Arrays.asList( lettersCopy );
15
16       System.out.println( "Initial list: " );
17       output( list );
18
19       Collections.reverse( list );        // reverse order
20       System.out.println( "\nAfter calling reverse: " );
21       output( list );
22
23       Collections.copy( copyList, list );  // copy List
24       System.out.println( "\nAfter copying: " );
25       output( copyList );
26
```

Outline

Algorithms1.java

Line 19

Line 23

```
27       Collections.fill( list, "R" );      // fill list with Rs
28       System.out.println( "\nAfter calling fill: " );
29       output( list );
30
31    } // end constructor
32
33    // output List information
34    private void output( List listRef )
35    {
36       System.out.print( "The list is: " );
37
38       for ( int k = 0; k < listRef.size(); k++ )
39          System.out.print( listRef.get( k ) + " " );
40
41       System.out.print( "\nMax: " + Collections.max( listRef ) );
42       System.out.println( "  Min: " + Collections.min( listRef ) );
43    }
44
45    public static void main( String args[] )
46    {
47       new Algorithms1();
48    }
49
50 } // end class Algorithms1
```

Outline

Algorithms1.java

Line 27

Line 41

Line 42

```
Initial list:
The list is: P C M
Max: P  Min: C

After calling reverse:
The list is: M C P
Max: P  Min: C

After copying:
The list is: M C P
Max: P  Min: C

After calling fill:
The list is: R R R
Max: R  Min: R
```

Algorithm binarySearch

- binarySearch
  - Locates Object in List
    - Returns index of Object in List if Object exists
    - Returns negative value if Object does not exist

61

```
1  // Fig. 22.11: BinarySearchTest.java
2  // Using algorithm binarySearch.
3  import java.util.*;
4
5  public class BinarySearchTest {
6     private static final String colors[] = { "red", "white",
7        "blue", "black", "yellow", "purple", "tan", "pink" };
8     private List list;       // List reference
9
10    // create, sort and output list
11    public BinarySearchTest()
12    {
13       list = new ArrayList( Arrays.asList( colors ) );
14       Collections.sort( list );   // sort the ArrayList
15       System.out.println( "Sorted ArrayList: " + list );
16    }
17
18    // search list for various values
19    private void printSearchResults()
20    {
21       printSearchResultsHelper( colors[ 3 ] ); // first item
22       printSearchResultsHelper( colors[ 0 ] ); // middle item
23       printSearchResultsHelper( colors[ 7 ] ); // last item
24       printSearchResultsHelper( "aardvark" );  // below lowest
25       printSearchResultsHelper( "goat" );      // does not exist
26       printSearchResultsHelper( "zebra" );     // does not exist
27    }
28
```

Outline

BinarySearchTest.java

Line 14

```
29    // helper method to perform searches
30    private void printSearchResultsHelper( String key )
31    {
32       int result = 0;
33
34       System.out.println( "\nSearching for: " + key );
35       result = Collections.binarySearch( list, key );
36       System.out.println( ( result >= 0 ? "Found at index " + result :
37          "Not Found (" + result + ")" ) );
38    }
39
40    public static void main( String args[] )
41    {
42       new BinarySearchTest().printSearchResults();
43    }
44
45 } // end class BinarySearchTest
```

```
Sorted ArrayList: black blue pink purple red tan white yellow
Searching for: black
Found at index 0

Searching for: red
Found at index 4

Searching for: pink
Found at index 2

Searching for: aardvark
Not Found (-1)

Searching for: goat
Not Found (-3)

Searching for: zebra
Not Found (-9)
```

Outline

BinarySearchTest.java

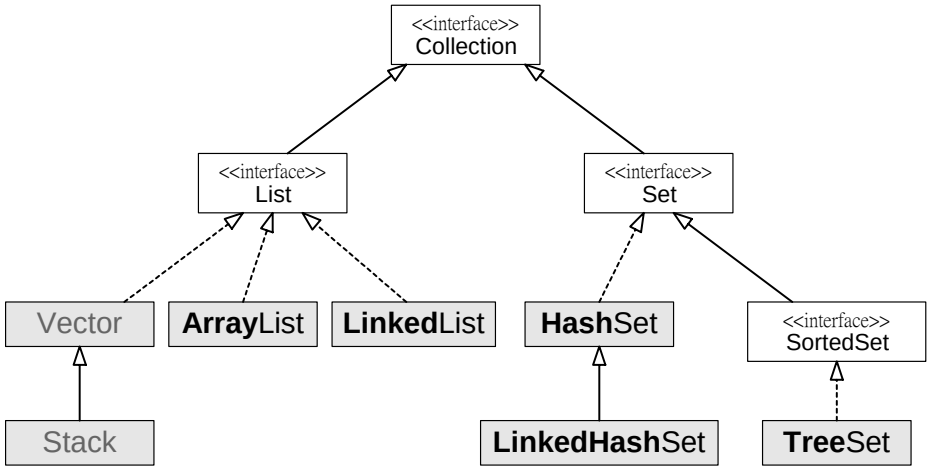
Line 35

Sets

- Set
  - Collection that contains unique elements
  - HashSet
    - Stores elements in hash table
  - TreeSet
    - Stores elements in tree

64

實作Collection延伸介面的具體類別



65

實作Set介面的具體類別

- **HashSet**使用hash table實作Set。元素沒有順序，而且元素不可重複存在HashSet物件內。其key和value是使用相同的物件，也就是被加入的物件。
- **LinkedHashSet**除了使用hash table，還使用linked list實作Set，使之成為有序集合。元素的順序是依照加入的順序。
- **TreeSet**使用tree實作SortedSet，所以元素在加入集合時，就會和既有的元素「比較」，以排放在適當的位置。

66

```
1  // Fig. 22.12: SetTest.java
2  // Using a HashSet to remove duplicates.
3  import java.util.*;
4
5  public class SetTest {
6     private static final String colors[] = { "red", "white", "blue",
7        "green", "gray", "orange", "tan", "white", "cyan",
8        "peach", "gray", "orange" };
9
10    // create and output ArrayList
11    public SetTest()
12    {
13       List list = new ArrayList( Arrays.asList( colors ) );
14       System.out.println( "ArrayList: " + list );
15       printNonDuplicates( list );
16    }
17
18    // create set from array to eliminate duplicates
19    private void printNonDuplicates( Collection collection )
20    {
21       // create a HashSet and obtain its iterator
22       Set set = new HashSet( collection );
23       Iterator iterator = set.iterator();
24
25       System.out.println( "\nNonduplicates are: " );
26
```

Outline

SetTest.java

Line 22

```
27       while ( iterator.hasNext() )
28          System.out.print( iterator.next() + " " );
29
30       System.out.println();
31    }
32
33    public static void main( String args[] )
34    {
35       new SetTest();
36    }
37
38 } // end class SetTest
```

```
ArrayList: [red, white, blue, green, gray, orange, tan, white, cyan, peach, gray, orange]

Nonduplicates are:
red cyan white tan gray green orange blue peach
```

Outline

SetTest.java

Lines 27-28

```
1  // Fig. 22.13: SortedSetTest.java
2  // Using TreeSet and SortedSet.
3  import java.util.*;
4
5  public class SortedSetTest {
6     private static final String names[] = { "yellow", "green",
7        "black", "tan", "grey", "white", "orange", "red", "green" };
8
9     // create a sorted set with TreeSet, then manipulate it
10    public SortedSetTest()
11    {
12       SortedSet tree = new TreeSet( Arrays.asList( names ) );
13
14       System.out.println( "set: " );
15       printSet( tree );
16
17       // get headSet based upon "orange"
18       System.out.print( "\nheadSet (\"orange\"):  " );
19       printSet( tree.headSet( "orange" ) );
20
21       // get tailSet based upon "orange"
22       System.out.print( "tailSet (\"orange\"):  " );
23       printSet( tree.tailSet( "orange" ) );
24
25       // get first and last elements
26       System.out.println( "first: " + tree.first() );
27       System.out.println( "last : " + tree.last() );
28    }
29
```

Outline

SortedSetTest.java

Line 12

Line 19

Line 23

Lines 26-27

```
30    // output set
31    private void printSet( SortedSet set )
32    {
33       Iterator iterator = set.iterator();
34
35       while ( iterator.hasNext() )
36          System.out.print( iterator.next() + " " );
37
38       System.out.println();
39    }
40
41    public static void main( String args[] )
42    {
43       new SortedSetTest();
44    }
45
46 } // end class SortedSetTest
```

```
set:
black green grey orange red tan white yellow

headSet ("orange"):  black green grey
tailSet ("orange"):  orange red tan white yellow
first: black
last : yellow
```

Outline

SortedSetTest.java

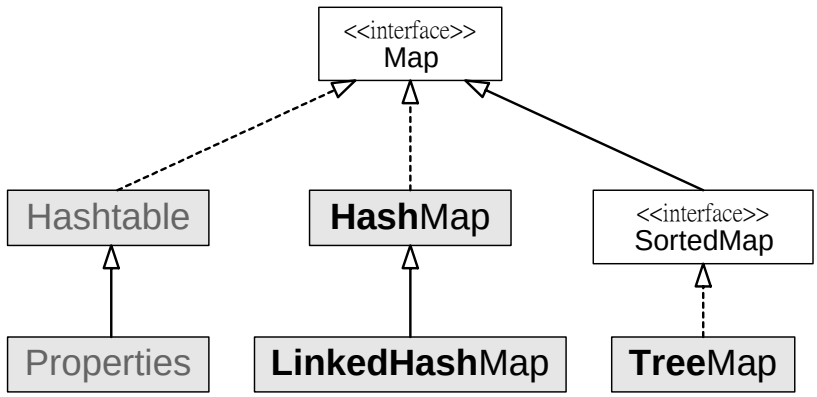
Lines 35-36

Maps

- Map
  - Associates keys to values
  - Cannot contain duplicate keys
    - Called *one-to-one mapping*

71

實作Map介面的具體類別



72

實作Map介面的具體類別

- **HashMap**使用hash table實作Map，此集合中key-value pairs順序和放入的順序無關，而且key無法重複。
- **LinkedHashMap**使用hash table和linked list實作Map，此集合中key-value pairs順序就是放入的順序，而key同樣無法重複。
- **TreeMap**使用tree實作Map，此集合中key-value pairs順序是依照key在集合中的排序而定，而key同樣無法重複。

73

實作Map介面的具體類別

- **Hashtable**和HashMap的功能差不多，不同處在於Hashtable為「**執行緒同步**」。
- **Properties** 為Hashtable的延伸類別。Properties的key和value應該為String型別，而且使用setProperty()和getProperty()取代put()和get()。

74

具體類別的整體比較

具體類別	直接實作介面	資料結構	元素重複性	元素順序	執行緒同步
ArrayList	List	array	可	依加入順序	否
LinkedList	List	linked list	可	依加入順序	否
Vector	List	array	可	依加入順序	是*
Stack	List	array	可	依加入順序	是*
HashSet	Set	hash table	不可	無關加入順序	否
LinkedHashSet	Set	hash table linked list	不可	依加入順序	否
TreeSet	SortedSet	tree	不可	排序	否
HashMap	Map	hash table	key不可	無關加入順序	否
LinkedHashMap	Map	hash table linked list	key不可	依加入順序	否
TreeMap	SortedMap	tree	key不可	依key排序	否
Hashtable	Map	hash table	key不可	無關加入順序	是*
Properties	Map	hash table	key不可	無關加入順序	是*

75

```
1  // Fig. 21.14: WordTypeCount.java
2  // Program counts the number of occurrences of each word in a string
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.util.*;
6  import javax.swing.*;
7
8  public class WordTypeCount extends JFrame {
9     private JTextArea inputField;
10    private JLabel prompt;
11    private JTextArea display;
12    private JButton goButton;
13
14    private Map map;
15
16    public WordTypeCount()
17    {
18       super( "Word Type Count" );
19       inputField = new JTextArea( 3, 20 );
20
21       map = new HashMap();
22
23       goButton = new JButton( "Go" );
24       goButton.addActionListener(
25
```

Outline

MapTest.java

Line 21

```
26          new ActionListener() { // inner class
27
28             public void actionPerformed( ActionEvent event )
29             {
30                createMap();
31                display.setText( createOutput() );
32             }
33
34          } // end inner class
35
36       ); // end call to addActionListener
37
38       prompt = new JLabel( "Enter a string:" );
39       display = new JTextArea( 15, 20 );
40       display.setEditable( false );
41
42       JScrollPane displayScrollPane = new JScrollPane( display );
43
44       // add components to GUI
45       Container container = getContentPane();
46       container.setLayout( new FlowLayout() );
47       container.add( prompt );
48       container.add( inputField );
49       container.add( goButton );
50       container.add( displayScrollPane );
51
```

Outline

MapTest.java

```
52       setSize( 400, 400 );
53       show();
54
55    } // end constructor
56
57    // create map from user input
58    private void createMap()
59    {
60       String input = inputField.getText();
61       StringTokenizer tokenizer = new StringTokenizer( input );
62
63       while ( tokenizer.hasMoreTokens() ) {
64          String word = tokenizer.nextToken().toLowerCase(); // get word
65
66          // if the map contains the word
67          if ( map.containsKey( word ) ) {
68
69             Integer count = (Integer) map.get( word ); // get value
70
71             // increment value
72             map.put( word, new Integer( count.intValue() + 1 ) );
73          }
74          else // otherwise add word with a value of 1 to map
75             map.put( word, new Integer( 1 ) );
76
77       } // end while
78
79    } // end method createMap
```

Outline

MapTest.java

Line 69

Lines 72 and 75

```
80
81    // create string containing map values
82    private String createOutput() {
83       StringBuffer output = new StringBuffer( "" );
84       Iterator keys = map.keySet().iterator();
85
86       // iterate through the keys
87       while ( keys.hasNext() ) {
88          Object currentKey = keys.next();
89
90          // output the key-value pairs
91          output.append( currentKey + "\t" +
92                   map.get( currentKey ) + "\n" );
93       }
94
95       output.append( "size: " + map.size() + "\n" );
96       output.append( "isEmpty: " + map.isEmpty() + "\n" );
97
98       return output.toString();
99
100   } // end method createOutput
101
```

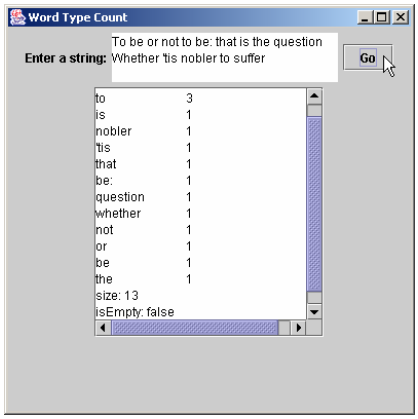
Outline

MapTest.java

```
102   public static void main( String args[] )
103   {
104      WordTypeCount application = new WordTypeCount();
105      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
106   }
107
108 } // end class WordTypeCount
```

Outline

MapTest.java



Synchronization Wrappers

- Built-in collections are unsynchronized
  - Concurrent access to a Collection can cause errors
  - Java provides *synchronization wrappers* to avoid this
    - Via set of public static methods

public static method header
`Collection synchronizedCollection( Collection c )`
`List synchronizedList( List aList )`
`Set synchronizedSet( Set s )`
`SortedSet synchronizedSortedSet( SortedSet s )`
`Map synchronizedMap( Map m )`
`SortedMap synchronizedSortedMap( SortedMap m )`

Fig. 22.15 Synchronization wrapper methods.

81

Unmodifiable Wrappers

- Unmodifiable wrappers
  - Converting collections to unmodifiable collections
  - Throw UnsorrtedOperationException if attempts are made to modify the collection

public static method header
`Collection unmodifiableCollection( Collection c )`
`List unmodifiableList( List aList )`
`Set unmodifiableSet( Set s )`
`SortedSet unmodifiableSortedSet( SortedSet s )`
`Map unmodifiableMap( Map m )`
`SortedMap unmodifiableSortedMap( SortedMap m )`

Fig. 22.16 Unmodifiable wrapper methods.

82

Abstract Implementations

- Abstract implementations
  - Offer “bare bones” implementation of collection interfaces
    - Programmers can “flesh out” customizable implementations
  - AbstractCollection
  - AbstractList
  - AbstractMap
  - AbstractSequentialList
  - AbstractSet

83