

Java程式設計進階

集合與泛型

鄭安翔

ansel_cheng@hotmail.com

課程大綱

1) Java 的集合架構

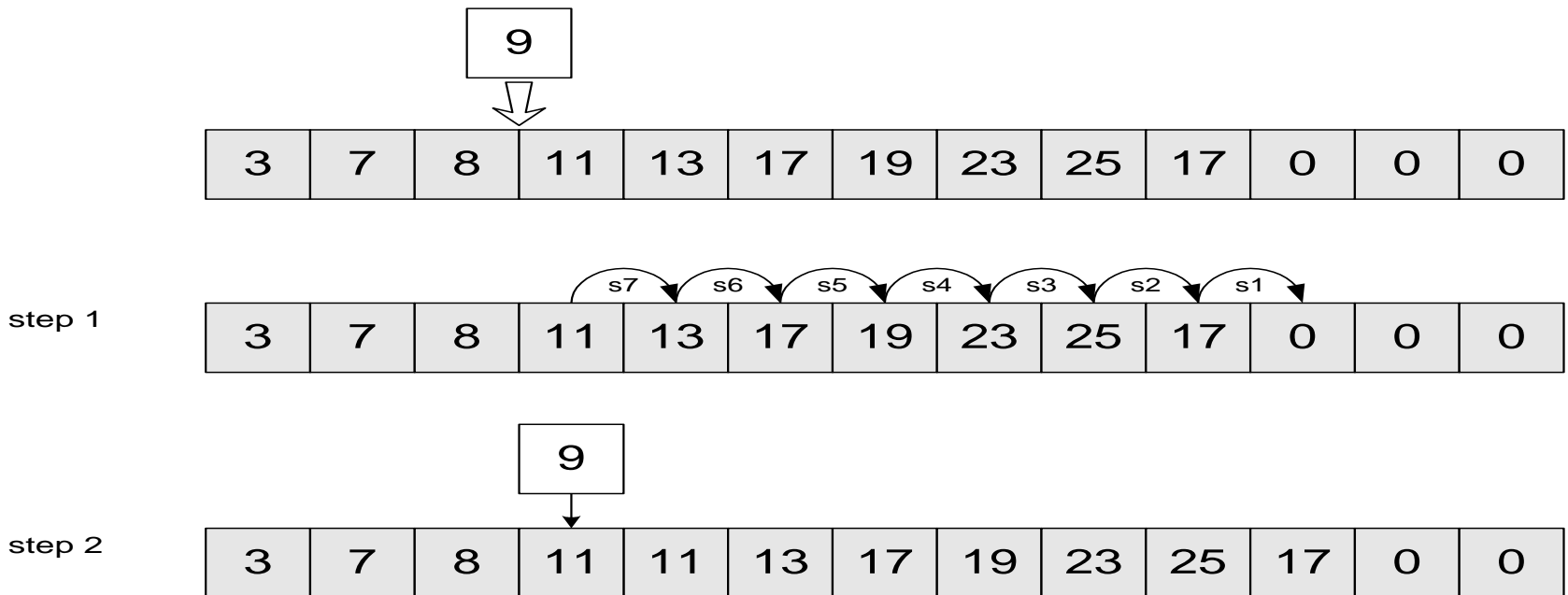
- 常用資料結構
- **Collection / Set / List / SortedSet** 介面
- **Map / SortedMap** 介面
- **Queue / Deque** 介面

2) 泛型

3) 集合進階

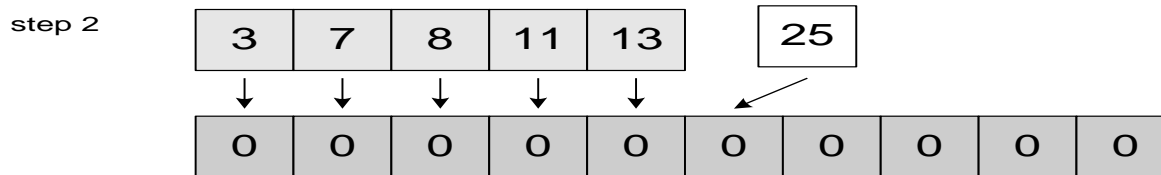
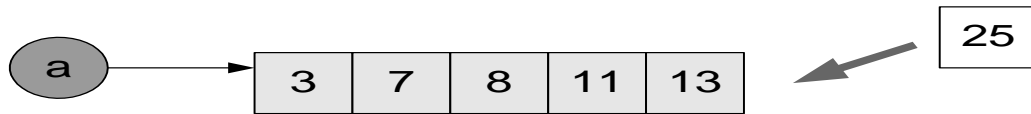
陣列 Array

- 存取元素的速度快。
- 插入或移除元素值不方便



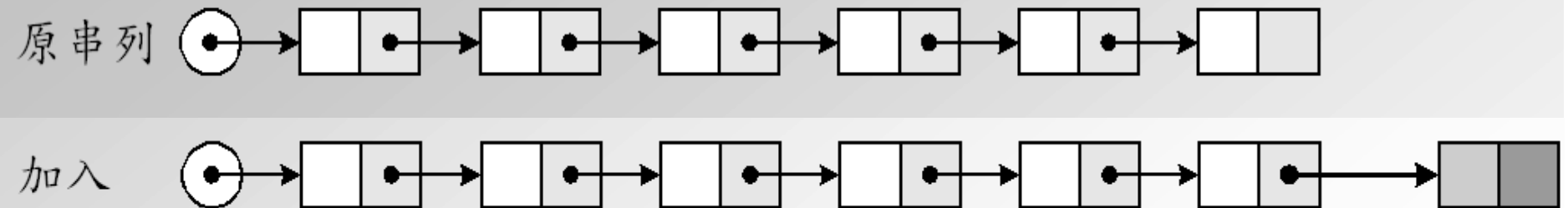
陣列 Array

- 陣列長度為固定，欲改變長度必須重新建立另一個陣列。



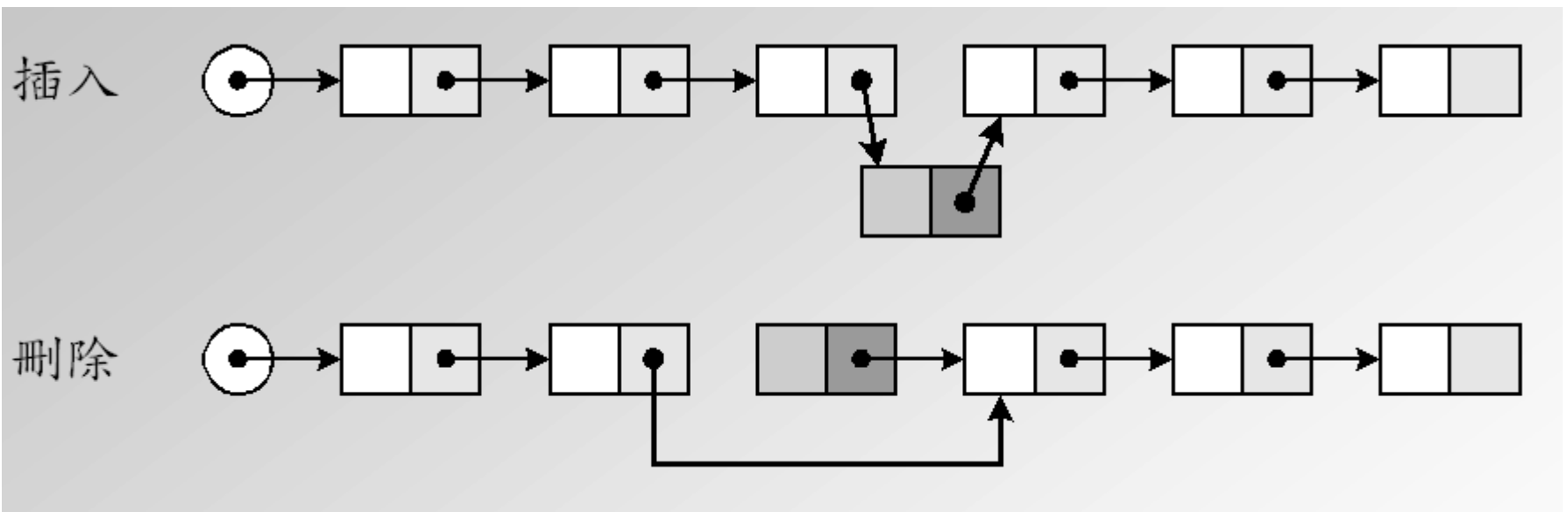
鏈結串列 Linked List

- 存取節點的速度較慢。
- 變更鏈結串列的長度很方便。



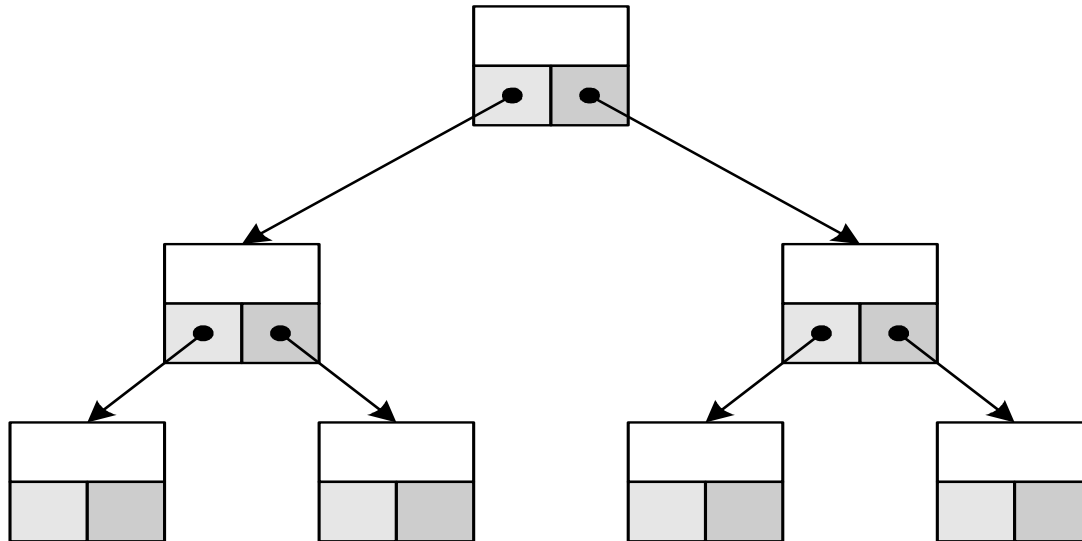
鏈結串列 Linked List

- 插入或刪除節點也很方便。



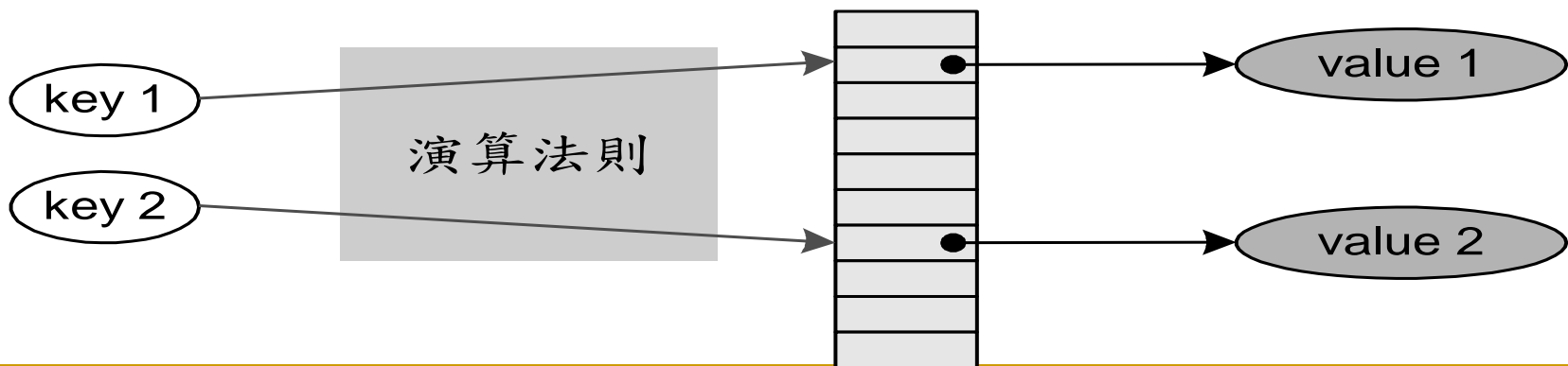
樹 Tree

- 有「排序」的功能。
- 節點資料的存取比較慢。
- Binary Tree



雜湊表 Hash Table

- 資料以key value paired的形式存在。
 - 一個元素包含一個key和一個value
- 透過key可以取得value。
- 存取資料的速度快。
- 較浪費記憶體空間。

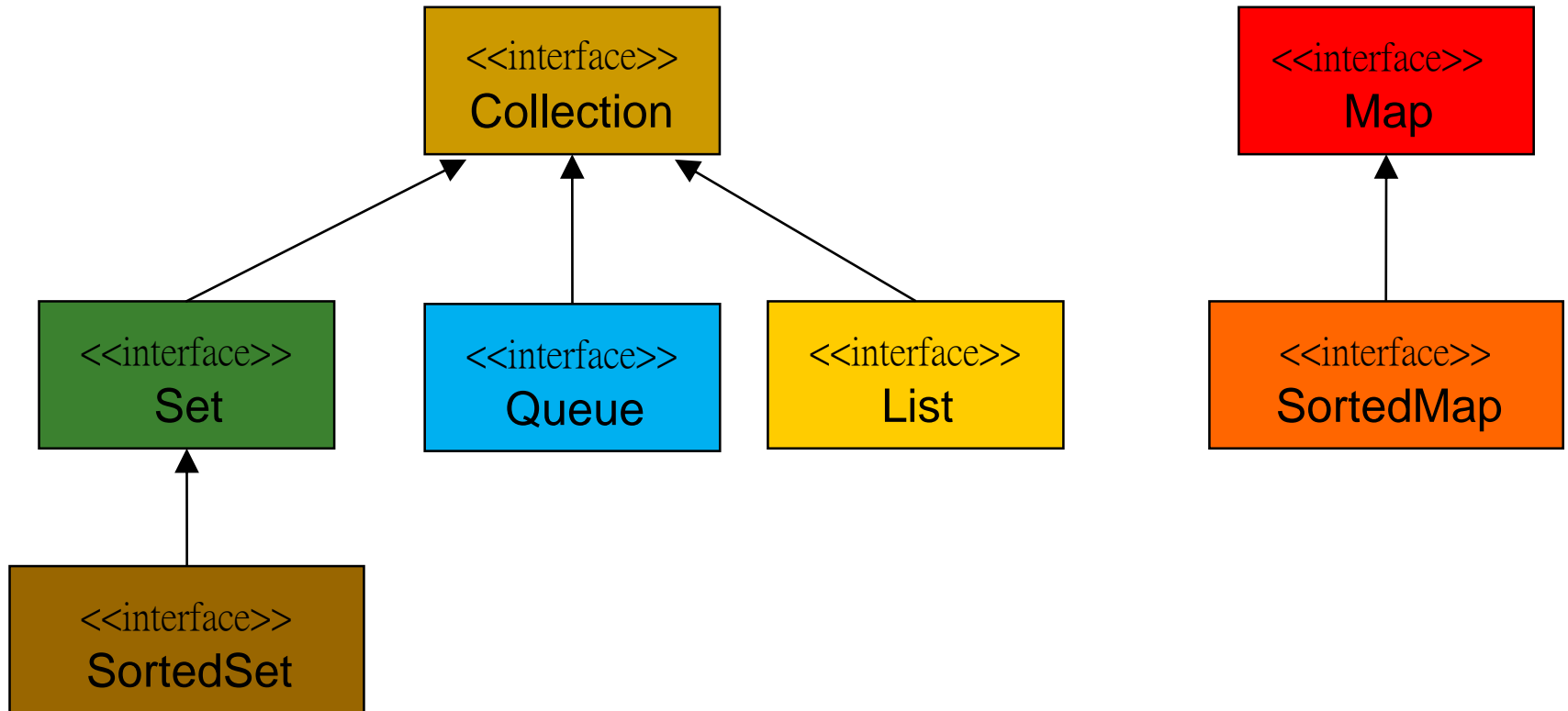


Collection Framework

- java.util package
- Array vs. Collection Framework

	陣列 array	集合 Collection
容量大小	需預先給定 無法於執行時期動態增加元素	可動態增加集合元素
資料型別	陣列內只能置入同型別資料	集合內可置入不同型別的資料
內容	可置入基本資料型別或物件	只能置入物件 基本型別需用包裝器類別
元素處理	基本讀取	提供進階的方法處理集合內元素

Collection 介面繼承架構



Collection interface

■ Collection 集合

- 使用一個物件描述(參照到)一群物件
- 集合中的物件稱為集合元素(**Elements**)

■ Collection 介面

- 是所有 **Java** 集合的根源。
- 子介面**List**、**Set**，分別代表不同特性的集合
 - 重複性 (**Duplicates**)
 - 順序性 (**Ordered**)
 - 遞增排序
 - 加入順序

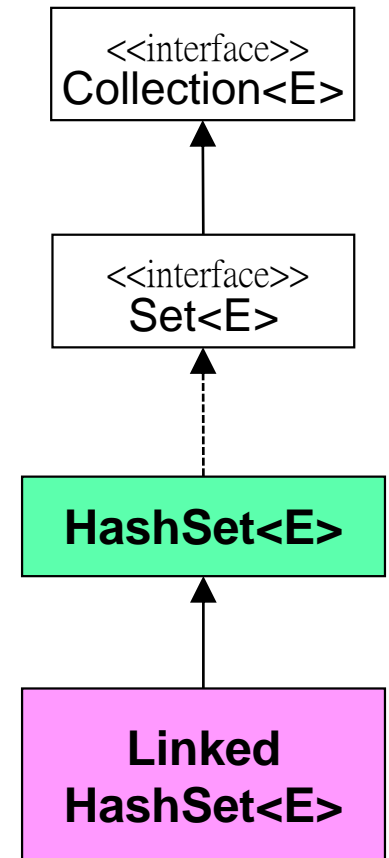
Collection 介面常用方法

方法名稱	傳回值	說明
Add(E e)	boolean	將指定的物件o加入集合中，若傳回值為true表示加入成功，false則加入失敗。
clear() throws UnsupportedOperationException	void	清除集合中的所有元素
contains(Object o)	boolean	用以判斷指定物件o是否屬於該集合物件的成員之一，屬於傳回true，不屬於則傳回false。
containsAll(Collection<?> c)	boolean	判斷指定集合物件c中所有成員是否都屬於該集合物件的成員，屬於傳回true，不屬於則傳回false。
equals(Object o)	boolean	比較集合物件與指定的物件o(if the specified object is equal to this collection)，當集合實作Set介面(非Map或List)且指定物件o也實作Set介面時，即傳回true。
isEmpty()	boolean	判斷此集合是否為空。若此集合是空的，將傳回true；反之則會傳回false。
remove(Object o)	boolean	移除在集合c中某一個元素，若該元素所參考的物件等於參數中的o物件即將該集合元素移除，移除成功傳回true，失敗則傳回false。
removeAll(Collection<?> c)	boolean	移除指定集合c中的所有成員(所指定的集合c可能包含了不只一種集合物件，但不論多少一律皆移除)。移除成功傳回true，失敗則傳回false。
retainAll(Collection<?> c)	void	除了留下集合c本身所包含的元素外，其餘的都將移除(也許在集合c中還包括了其他成員集合不過這些都會被刪除)。移除成功傳回true，失敗則傳回false。
size()	int	傳回此集合所有元素個數。
iterator()	Iterator <E>	將此集合回傳一個Iterator，此集合就可以使用Iterator中所有的方法如hasNext() next()與remove()。
hashCode()	int	傳回一個hash code數值
toArray()	Object[]	傳回一個包含所有集合成員的物件陣列

Set interface

■ Set :

- ❑ Set 是資料不可重複且無順序性的集合
- ❑ 繼承 Collection 介面
 - 包含 Collection 中所有的方法。
- ❑ Set 中放置的元素不可重複 (No Duplicates)
 - 用 equals() 方法來判斷加入的物件是否重複。
- ❑ Set 中放置的元素無順序(Unordered)
 - Set 元素沒有index的概念
 - 實作時,Set 的元素擺放位置是根據元素之 hashCode() 決定
 - hashCode() 是利用唯一數值並透過雜湊表(Hash tables)來尋找物件的方法



Set具體類別比較

■ HashSet

- ❑ 元素不可重複
- ❑ 元素沒有順序性(位置是根據 hashCode 決定)
- ❑ non thread-safe：方法並非宣告為 synchronized，在多執行緒時會出現 thread-safe 的問題。

■ LinkedHashSet

- ❑ 除了使用hashcode，還使用linked list結構實作Set
- ❑ 有序集合，元素順序為加入的順序

迭代器 – Iterator 介面

■ 用途

- ❑ 讀取 Collection 中的資料
- ❑ 利用 Collection 之 public Iterator iterator() 方法取得
- ❑ Iterator 物件之元素內容值將有順序性。

■ Iterator 常用方法

方法名稱	傳回值	說明
hasNext()	boolean	判斷是否還有下一個元素，若有，傳回 true，反之則傳回 false。
next()	E	取得下一個 Iteration 元素。
remove()	void	移除目前在 Iteration 中所指向的元素。

```
Iterator it = collection.iterator();  
while(it.hasNext()) {  
    Object o = it.next();  
}
```

HashSet 範例

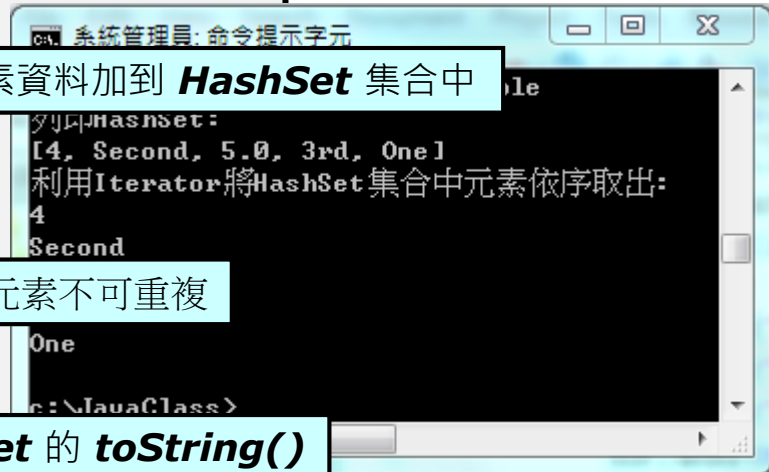
```
01 import java.util.*;
02 public class HashSetExample {
03     public static void main(String[] args) {
04         HashSet hs = new HashSet();
05         //加入元素
06         hs.add("One");
07         hs.add("Second");
08         hs.add("3rd");
09         hs.add(new Integer(4));
10         hs.add(new Float(5.0));
11         //加入重複元素
12         hs.add("Second");
13         hs.add(new Integer(4));
14         //列印HashSet,利用toString()方法
15         System.out.println("列印HashSet:");
16         System.out.println(hs);
17         //利用Iterator將HashSet集合中的所有元素依序取出"
18         System.out.println("利用Iterator將HashSet集合中元素依序取出:");
19         Iterator i= hs.iterator();
20         while(i.hasNext()) {
21             System.out.println(i.next());
22         }
23     }
24 }
```

利用 **add()** 將元素資料加到 **HashSet** 集合中

HashSet 集合中元素不可重複

HashSet 的 **toString()**

用 **Iterator** 將 **HashSet** 集合中的所有元素依序取出



LinkedHashSet 範例

```
01 import java.util.*;
02 public class LinkedHashSetExample {
03     public static void main(String[] args) {
04         Set ls = new LinkedHashSet();
05         ls.add("One");
06         ls.add("Second");
07         ls.add("3rd");
08         ls.add(new Integer(4));
09         ls.add(new Float(5.0));
10         //加入重複元素
11         ls.add("Second");
12         ls.add(new Integer(4));
13         //列印LinkedHashSet,呼叫toString()方法
14         System.out.println("列印LinkedHashSet:");
15         System.out.println(ls);
16         //利用Iterator將LinkedHashSet集合中的所有元素依序取出"
17         System.out.println("利用Iterator將LinkedHashSet集合中元素依序取出:");
18         Iterator i = ls.iterator();
19         while(i.hasNext()) {
20             System.out.println(i.next());
21         }
22     }
23 }
```

利用 **add()** 將元素資料加到 **LinkedHashSet** 集合中

利用Iterator將LinkedHashSet集合中元素依序取出:

```
One
Second
3rd
"
```

LinkedHashSet 集合中元素不可重複

LinkedHashSet 的 **toString()**

Iterator將**LinkedHashSet** 集合中所有元素依序取出

SortedSet interface

■ SortedSet

□ 繼承Set介面

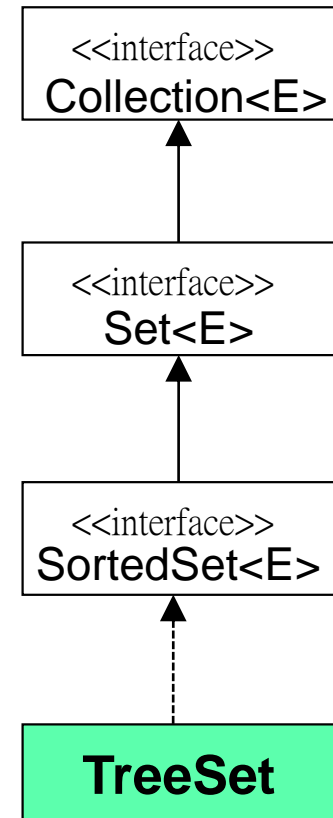
- 包含在 **Set** 類別所定義的方法。

□ 元素不可重複。

- **equals()**來判斷加入的物件是否重複中

□ 元素有順序性

- 元素資料型態必須相同。
- 將集合內容依物件邏輯作遞增排序。



SortedSet interface

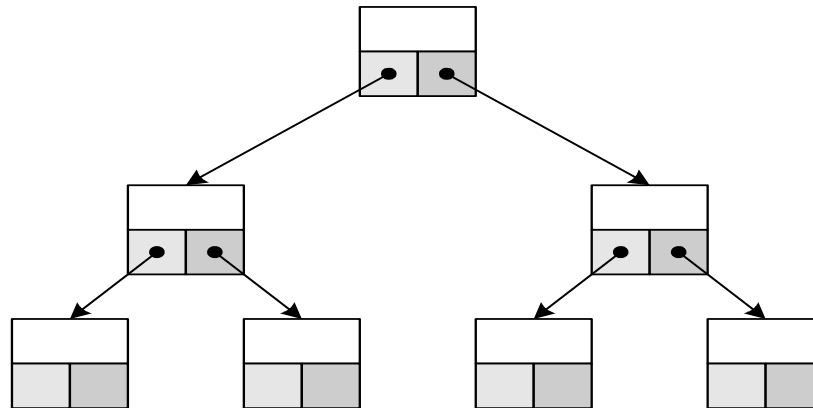
■ SortedSet 介面常用方法

方法名稱	傳回值	說明
Comparator<? super E>()	Comparator	傳回一個Comparator物件，若傳回值為null，表示以自然排序法排序。
first()	E	傳回第一個元素。
last()	E	傳回最後一個元素。
headSet(E toElement)	SortedSet<E>	回傳一個Set子集合，子集合元素內容都小於指定元素toElement。
tailSet(E fromElement)	SortedSet<E>	回傳一個Set子集合，子集合元素內容都大於等於指定元素fromElement。
subSet(E fromElement, E toElement)	SortedSet<E>	回傳一個Set子集合，子集合元素內容都介於指定元素fromElement與toElement之間，可包含fromElement但不包含toElement。

SortedSet 具體類別

■ TreeSet

- 使用tree結構實作SortedSet
- 元素在加入集合時，會和既有的元素比較，以擺放在適當的位置



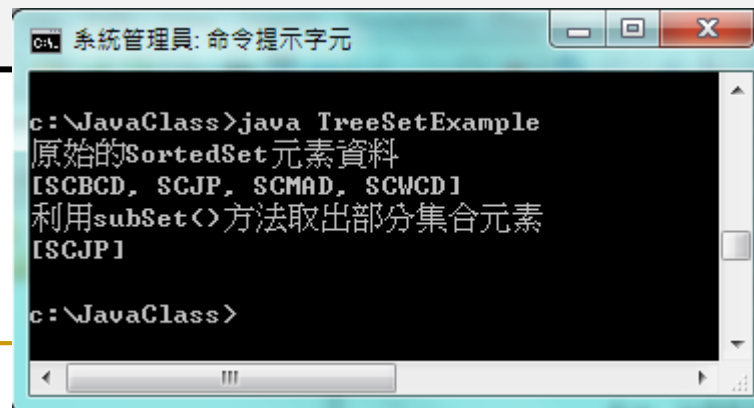
TreeSet 範例

```
01 import java.util.*;
02 public class TreeSetExample {
03     public static void main(String[] args) {
04         SortedSet ts = new TreeSet();
05         ts.add("SCWCD");
06         ts.add("SCJP");
07         ts.add("SCBCD");
08         ts.add("SCMAD");
09         ts.add("SCJP");
10         System.out.println("原始的SortedSet元素資料");
11         System.out.println(ts);
12         SortedSet sub_ts = ts.subSet("SCJP", "SCMAD");
13         System.out.println("利用subSet()方法取出部分集合元素");
14         System.out.println(sub_ts);
15     }
16 }
```

TreeSet 的宣告

利用 **add()** 將元素資料加到 **TreeSet** 集合中

利用 **subSet()** 取出部份 **TreeSet** 集合



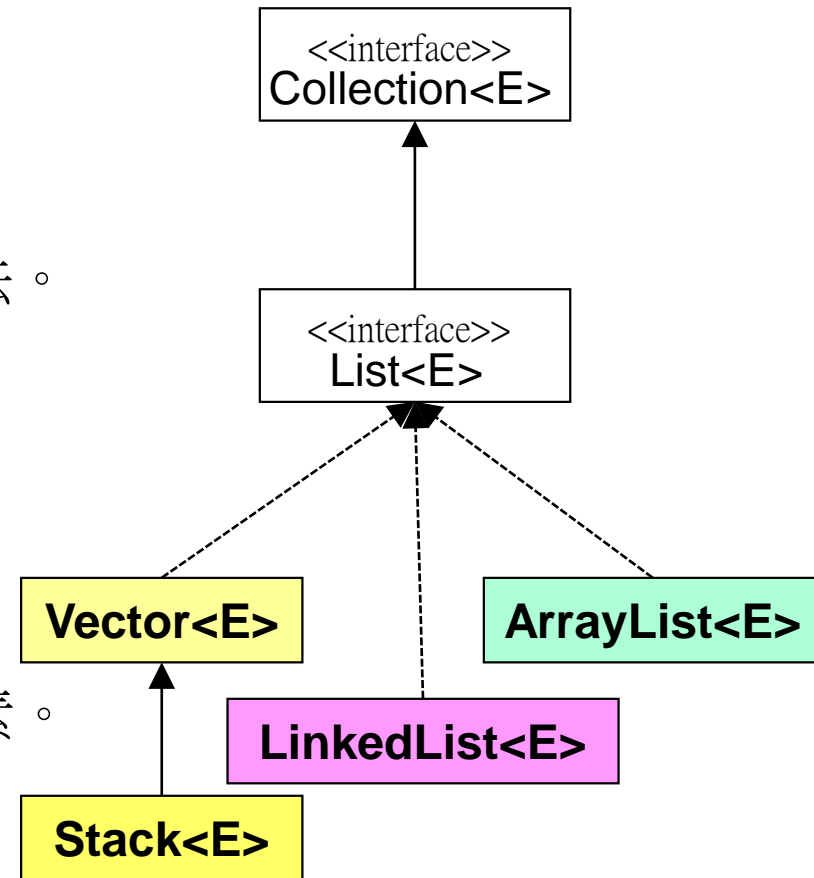
```
C:\JavaClass>java TreeSetExample
原始的SortedSet 元素資料
[SCBCD, SCJP, SCMAD, SCWCD]
利用subSet()方法取出部分集合元素
[SCJP]

C:\JavaClass>
```

List 介面

■ List 介面：

- 繼承 Collection 介面，
 - 包含 Collection 中所有的方法。
- List 中的資料可以重複
- List 中的元素有順序性
 - 按照 index (加入順序) 排序
 - 可插入或移除指定的集合元素。



List常用方法

方法名稱	傳回值	說明
<i>add(E e)</i>	<i>boolean</i>	從 List 的末端(<i>end of this list</i>)加入指定物件 e 。
<i>add(int index, E element)</i>	<i>void</i>	沒有傳回值。將指定物件 element 加到 List 所指定的索引(<i>index</i>)位置，並傳回此集合物件。
<i>addAll(Collection<? extends E> c)</i>	<i>boolean</i>	從 List 的末端(<i>end of this list</i>)加入集合 c
<i>get(int index)</i>	<i>E</i>	取得 List 中指定索引位置的元素
<i>set(int index, E element)</i>	<i>E</i>	將 List 中指定索引位置的元素以指定物件 element 取代
<i>indexOf(Object o)</i>	<i>int</i>	回傳指定物件 o 在 List 上的索引位置。傳回值若為 -1 表示 List 集合物件元素中並沒有包含指定物件 e 。
<i>listIterator()</i>	<i>ListIterator</i>	回傳一個 ListIterator 物件
<i>remove(int index)</i>	<i>E</i>	移除 List 中指定索引位置的元素，並傳回被移除的元素。
<i>subList(int fromIndex, int toIndex)</i>	<i>List<E></i>	在 List 上回傳一個索引位置，即從 fromIndex 到 toIndex 的 List 物件。該物件包含 formIndex 所指的元素但不包含 toIndex 所指的元素

ListIterator 介面

■ 用途

- ❑ Iterator讀取是單向的,讀過的資料無法再回頭讀一次
- ❑ List類型的物件可產生ListIterator物件,可以前後讀取資料
- ❑ 利用 List 之 Public ListIterator listIterator()方法取得

■ ListIterator常用方法

方法名稱	傳回值	說明
hasPrevious()	boolean	判斷List中是否有前一個 元素,若有,傳回 true,反之則傳回false。
previous()	E	取得List中前一個 元素。
previousIndex()	int	傳回前一個List元素的鍵值,並指向該list元素。
nextIndex()	int	傳回後一個List元素的鍵值,並指向該list元素。
add(E e)	void	加入一個指定元素e到list集合中
set (E e)	void	以指定元素取代目前所指向的list元素

List具體類別比較

JDK 1.1以後版本(Non-Thread-safe)

■ ArrayList

- 使用array資料結構實作List。
- 可以自由的擴增容量，實作時使用 `add()`與 `get()` 這二個方法來置入與取出元素物件即可。
- 隨機存取元素的效率佳。
- 但插入、移除元素或重定陣列長度效率較差。

■ LinkedList

- 使用linked list實作List。
- 隨機存取元素的效率較差。
- 插入元素、移除元素和改變長度效率佳。

ArrayList 範例

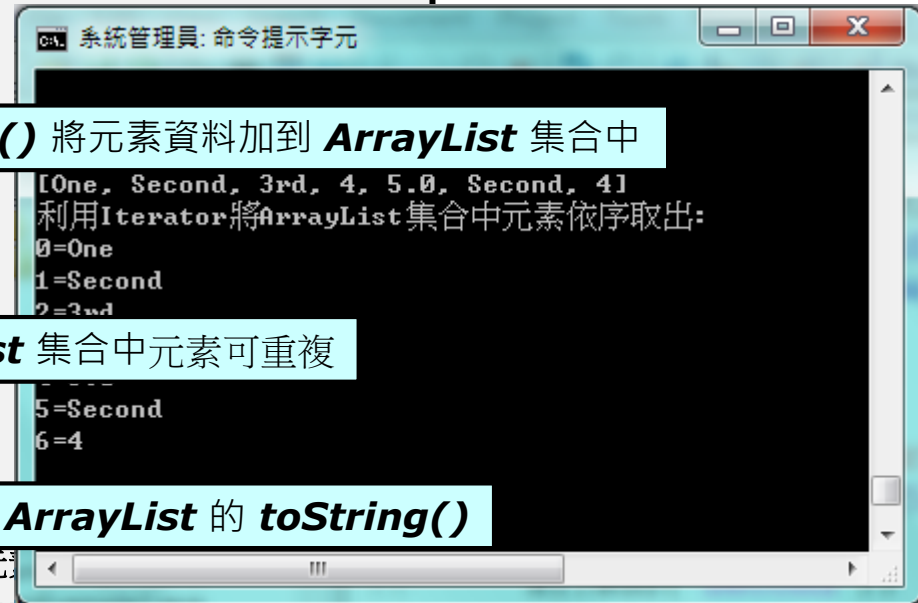
```
01 import java.util.*;
02 public class ArrayListExample {
03     public static void main(String[] args) {
04         ArrayList al = new ArrayList();
05         al.add("One");
06         al.add("Second");
07         al.add("3rd");
08         al.add(new Integer(4));
09         al.add(new Float(5.0));
10         //加入重複元素
11         al.add("Second");
12         al.add(new Integer(4));
13         //列印ArrayList,利用toString()方法
14         System.out.println("列印ArrayList:");
15         System.out.println(al);
16         //利用Iterator將ArrayList集合中的所有元素依序取出
17         System.out.println("利用Iterator將ArrayList集合中元素依序取出:");
18         ListIterator it = al.listIterator();
19         while(it.hasNext()) {
20             int index = it.nextIndex();
21             System.out.println(index + "=" + it.next());
22         }
23     }
24 }
```

利用 **add()** 將元素資料加到 **ArrayList** 集合中

ArrayList 集合中元素可重複

ArrayList 的 **toString()**

用 **ListIterator** 將 **ArrayList** 集合中的所有元素依序取出



LinkedList 範例

```
01 import java.util.*;
02 public class LinkedListExample {
03     public static void main(String[] args) {
04         LinkedList ll = new LinkedList();
05         ll.add("SCJP");
06         ll.add("SCWCD");
07         ll.add("SCBCD");
08         ll.add("SCMAD");
09         ll.add("SCJP");
10         //列印LinkedList
11         System.out.println("將 LinkedList 列印出來");
12         System.out.println(ll);
13         //抓取 LinkedList 中 index=1 資料
14         System.out.println("抓取 LinkedList 中 index=1");
15         System.out.println(ll.get(1));
16         //移除 LinkedList 中第一個元素資料
17         System.out.println("移除 LinkedList 中第一個元素資料:" + ll.removeFirst());
18         System.out.println("將變更後 LinkedList 列印出來");
19         System.out.println(ll);
20     }
21 }
```

利用 **add()** 將元素資料加到 **LinkedList** 集合中

LinkedList 的 **toString()**

取得 **List** 集合中特定元素

移除 **List** 集合中的特定元素

```
C:\JavaClass>java LinkedListExample
將 LinkedList 列印出來
[SCJP, SCWCD, SCBCD, SCMAD, SCJP]
抓取 LinkedList 中 index=1 資料
SCWCD
將變更後 LinkedList 列印出來
[SCWCD, SCBCD, SCMAD, SCJP]
```

List具體類別比較

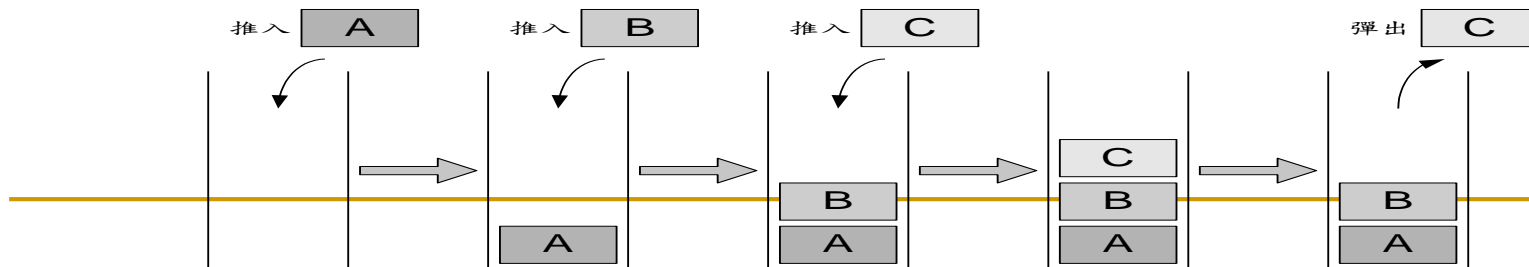
JDK 1.0 版本(執行緒同步Thread Safe)

■ Vector

- 使用array資料結構實作List。
- Vector為「執行緒同步類別」，而ArrayList則否

■ Stack

- Vector的子類別。同樣為有序、執行緒同步類別
- 先進先出(LIFO, Last-in-first-out)原則
- push()方法加入元素，pop() 方法取得元素資料



列舉 – Enumeration 介面

■ Enumeration 介面

- ❑ 讀取JDK1.1之前之原始集合中的資料 (Vector, Stack, Hashtable, Properties)
- ❑ 原始集合提供 `public Enumeration elements()`
- ❑ 所取出元素內容值是各自獨立，沒有順序性的。
- ❑ Enumeration 常用的方法

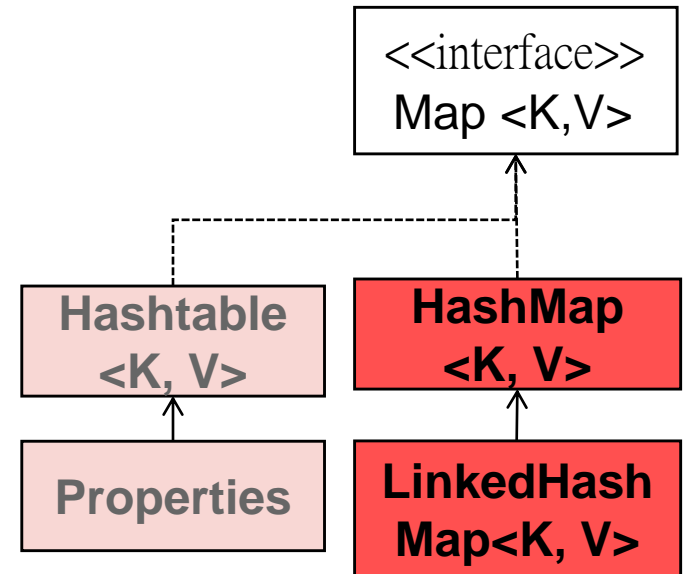
方法名稱	傳回值	說明
hasMoreElements()	boolean	測試是否還有下一個 Enumeration 元素，有的話則會傳回 true，反之則傳回 false。
nextElement()	E	指向並回傳下一個 Enumeration 元素。

```
Enumeration e = collection.elements();  
while(e.hasMoreElement()) {  
    Object o = e.nextElement();  
}
```

Map Interface

■ Map

- 不繼承Collection介面
- 資料是一組key-value pair
 - key和value皆為參照變數，
 - key不能重複，value可以重複。
 - 一個key對應(mapping)到一個value



Map Interface 常用方法

方法名稱	傳回值	說明
get (Object key)	V	利用 key 指向 Map 中指定的元素並傳回
put (K key, V value)	V	將所指定的 key 與 value 放入到 Map 中
putAll (Map<? extends K, ? extends V> m)	void	將所指定的 t (Map 物件)整份複製到 Map 中
remove (Object key)	V	利用 key 指向 Map 中指定的元素並移除
clear ()	void	清除 Map 所有資料
size ()	int	傳回 key-value pair 個數
containsKey (Object key)	boolean	若指定的 key 物件若存在於 Map 中則會傳回 true
containsValue (Object value)	boolean	若指定的 value 物件若存在於 Map 中則會傳回 true
keySet ()	Set<K>	將 Map 中包含的 key 以 Set 型態傳回
values ()	Collection<V>	將 Map 中包含的 value 以 Collection 型態傳回
entrySet ()	Set<Map.Entry<K,V>>	將 Map 中包含成對的 key 和 value 以 Set 型態傳回
equals (Object o)	boolean	將指定物件 o 與 Map 做比較
hashCode ()	int	傳回 Map 中的 hash code
isEmpty ()	boolean	判斷 Map 中是否是空的

Map具體類別比較

JDK 1.1以後版本(Non-Thread-safe)

■ HashMap

- 使用hash table實作Map。
- key無法重複 (HashSet)，Value可以重複。
- key-value pairs順序和放入的順序無關。
- Key及value皆可為null

■ LinkedHashMap

- 使用hash table和linked list實作Map。
- key無法重複 (LinkedHashSet)，Value可以重複。
- key-value pairs順序就是放入的順序。
- Key及value皆可為null

Map具體類別比較

JDK 1.0版本(Thread-safe)

■ Hashtable

- 和HashMap的功能類似
- Hashtable為執行緒同步
- Key及value皆不可為null

■ Properties

- 為Hashtable的子類別。
- Properties的key和value應該為String型別。
- 使用setProperty()和getProperty()取代put()和get()。

HashMap 範例

```
01 import java.util.*;
02 public class HashMapExample {
03     public static void main(String[] args) {
04         Map map = new HashMap();
05         map.put("one", "1st");
06         map.put("second", new Integer(2));
07         map.put("third", "3rd");
08         System.out.println(map.toString());
09         //覆寫前面的指定
10         map.put("third", "III");
11         System.out.println(map.toString());
12         //回傳Key集合
13         Set set1 = map.keySet();
14         System.out.println("Key集合:");
15         //回傳value的集合
16         Collection collection = map.values();
17         System.out.println("value的集合:");
18         //回傳Key與value的集合
19         Set set2 = map.entrySet();
20         System.out.println("Key與value的集合"+set2);
21     }
22 }
```

利用 **put()** 將元素資料加到 **HashMap** 集合中

在 **HashMap** 集合中利用 **put()** 將已存在的 **Key** 對應到新的 **value**

利用 **keySet()** 取得 **HashMap** 集合中的所有的 **Key** 值

利用 **values()** 取得 **HashMap** 集合中的所有的 **value** 值

利用 **entrySet()** 取得 **HashMap** 集合中的所有的 **key** 與 **value** 值

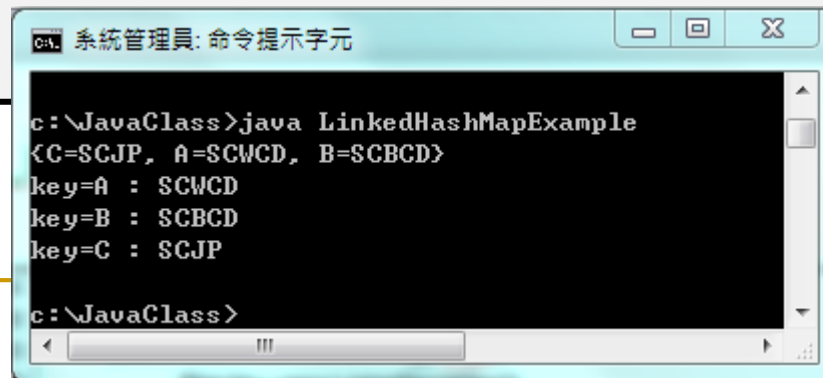
LinkedHashMap 範例

```
01 import java.util.*;
02 public class LinkedHashMapExample {
03     public static void main(String[] args) {
04         Map lm = new LinkedHashMap();
05         lm.put("C", "SCJP");
06         lm.put("A", "SCWCD");
07         lm.put("B", "SCBCD");
08
09         System.out.println(lm);
10
11         System.out.println("key=A : " + lm.get("A"));
12         System.out.println("key=B : " + lm.get("B"));
13         System.out.println("key=C : " + lm.get("C"));
14     }
15 }
```

利用 **put()** 將元素資料加到 **LinkedHashMap** 集合中

System.out.println(lm); **key-value pairs** 順序就是放入的順序

利用 **get()** 取得 **Map** 集合中
指定 **Key** 對應的 **value** 內容



The screenshot shows a Windows command prompt window titled "系統管理員: 命令提示字元". The command executed is `c:\JavaClass>java LinkedHashMapExample`. The output is as follows:

```
c:\JavaClass>java LinkedHashMapExample
{C=SCJP, A=SCWCD, B=SCBCD}
key=A : SCWCD
key=B : SCBCD
key=C : SCJP
c:\JavaClass>
```

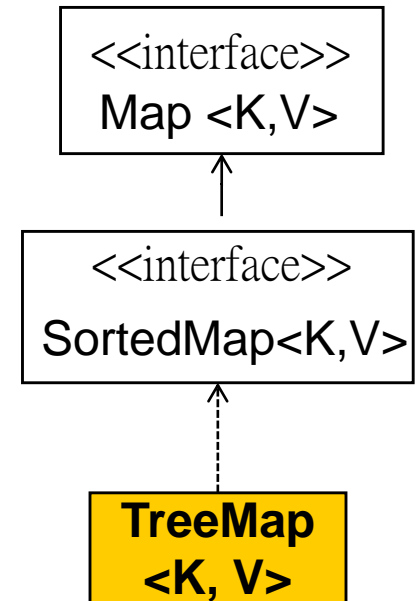
SortedMap Interface

■ SortedMap 介面

- Map的子介面
- Key值有順序性的集合

■ TreeMap 類別

- 使用tree實作Map。
- key無法重複(TreeSet)，Value可以重複。
- key-value pairs順序是依照key在集合中的排序而定
- Key及value皆不可為null



SortedMap Interface 常用方法

方法名稱	傳回值	說明
<i>comparator()</i>	<i>Comparator</i> <i><? super K></i>	傳回一個 <i>comparator</i> 物件，若傳回值為 <i>null</i> 表示將以自然排序法排序。
<i>firstKey()</i>	<i>K</i>	傳回目前鍵值最低(小)的物件。
<i>lastKey()</i>	<i>K</i>	傳回目前鍵值最高(大)的物件。
<i>headMap(K toKey)</i>	<i>SortedMap</i> <i><</i> <i>K,V></i>	回傳一個 <i>Map</i> 的子集合，所產生出來的子集合其鍵值必須都小於指定鍵值 <i>toKey</i> 。
<i>subMap(K fromKey, K toKey)</i>	<i>SortedMap</i> <i><</i> <i>K,V></i>	回傳一個 <i>Map</i> 的子集合，所產生出來的子集合其鍵值範圍必須介於 <i>formKey</i> 與 <i>toKey</i> 之間，但不包含 <i>toKey</i> 。
<i>tailMap(K fromKey)</i>	<i>SortedMap</i> <i><</i> <i>K,V></i>	回傳一個 <i>Map</i> 的子集合，所產生出來的子集合其鍵值範圍必須大於或等於 <i>formKey</i> 。

TreeMap 範例

```
01 import java.util.*;
02 public class TreeMapExample {
03     public static void main(String[] args) {
04         SortedMap sm = new TreeMap();
05
06         sm.put(new Integer(2), "SCJP");
07         sm.put(new Integer(1), "SCWCD");
08         sm.put(new Integer(3), "SCBCD");
09         sm.put(new Integer(1), "SCJP");
10
11         System.out.println(sm);
12     }
13 }
```

TreeMap 中的 **put()** 方法會依照所設定的 **Key** 排序,來排定元素順序
Key需是相同型態物件,且是可比較的

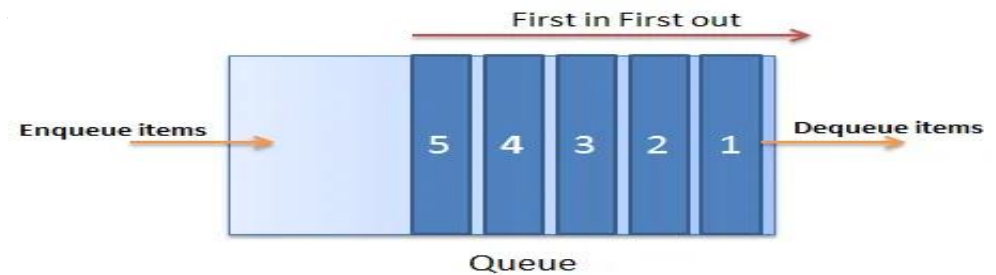


The screenshot shows a Windows command prompt window titled "系統管理員: 命令提示字元". The command executed is "c:\JavaClass>java TreeMapExample". The output displayed is "<1=SCJP, 2=SCJP, 3=SCBCD>". The prompt then returns to "c:\JavaClass>".

線性資料結構

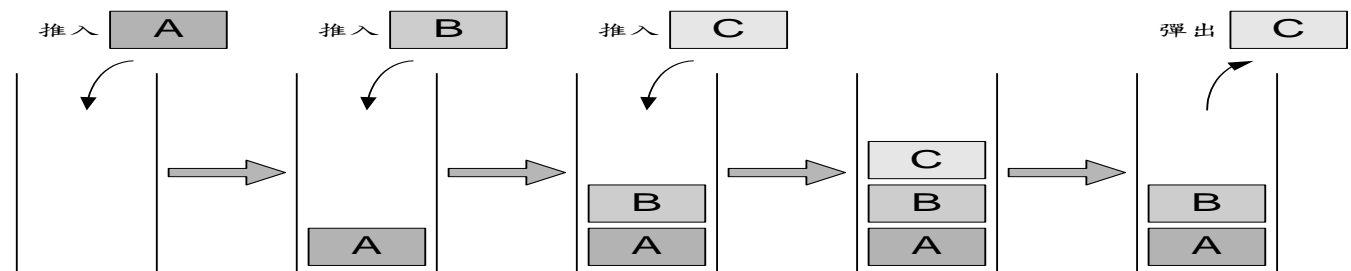
■ Queue 佇列結構

□ First In First Out(FIFO)



■ Stack 資料結構

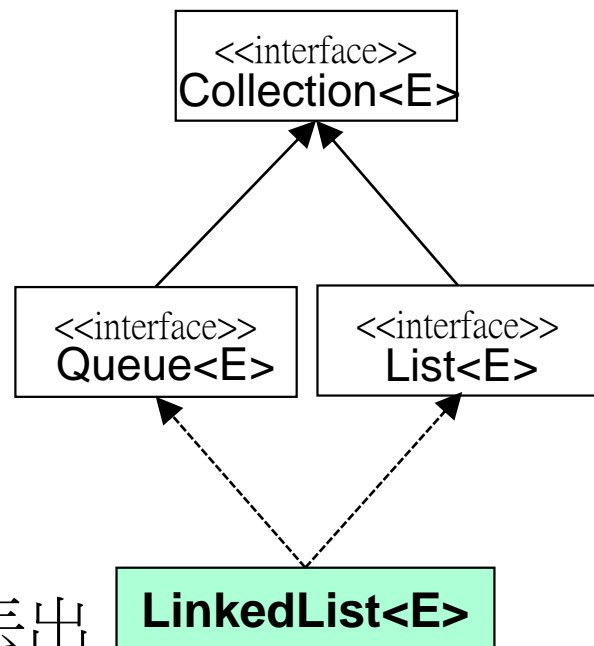
□ Last In First Out(LIFO)



Queue 介面

■ Queue 介面：

- ❑ Java SE 5.0 新增
- ❑ 繼承 Collection 介面
- ❑ 先進先出(FIFO, First-in-first-out)
- ❑ add(E e), remove(), element()方法丟出 IllegalStateException及NoSuchElementException



■ Queue 常用方法

方法名稱	傳回值	說明
offer(E e)	boolean	在 Queue 中加入指定元素。成功傳回true，失敗傳回false。
peek()	E	從 Queue 中取得起始指定元素內容，但不移除。若 Queue 是空的則傳回null。
poll()	E	從 Queue 中取得起始指定元素內容，並移除該元素。若 Queue 是空的則傳回null。

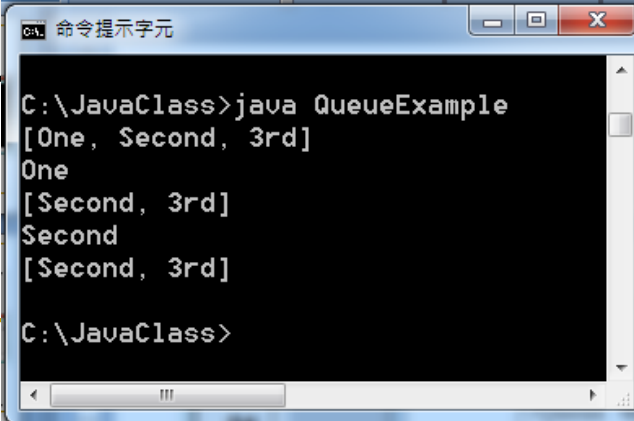
Queue 範例

```
01 import java.util.*;
02 public class QueueExample {
03     public static void main(String[] args) {
04         Queue q = new LinkedList();
05         q.offer("One");
06         q.offer("Second");
07         q.offer("3rd");
08         System.out.println(q.toString());
09         //Queue 中取得起始指定元素內容，並移除該元素
10         System.out.println(q.poll());
11         System.out.println(q.toString());
12         //Queue 中取得起始指定元素內容，但不移除該元素
13         System.out.println(q.peek());
14         System.out.println(q.toString());
15     }
16 }
```

利用 **offer()** 將元素資料加到 **Queue** 集合中

利用 **poll()** 取得 **Queue** 集合中
起始元素內容，並移除該元素

利用 **peak()** 取得 **Queue** 集合
中起始元素內容，不移除該元素



```
C:\JavaClass>java QueueExample
[One, Second, 3rd]
One
[Second, 3rd]
Second
[Second, 3rd]
C:\JavaClass>
```

Deque 介面實作線性資料結構

- Deque 介面：
 - Java SE 6.0 新增, LinkedList實作
 - Double Ended Queue
 - 先進先出(FIFO, First-in-first-out) Queue
 - 後進先出(LIFO, Last-in-first-out) Stack



Deque 介面

■ Deque 介面：

- Java SE 6.0 新增, LinkedList實作
- Double Ended Queue

	First Element (Head)		Last Element (Tail)	
	<i>Throws exception</i>	<i>Special value</i>	<i>Throws exception</i>	<i>Special value</i>
Insert	addFirst(e) : boolean	offerFirst(e) : boolean	addLast(e) : boolean	offerLast(e) : boolean
Remove	removeFirst() : E	pollFirst() : E	removeLast() : E	pollLast() : E
Examine	getFirst() : E	peekFirst() : E	getLast() : E	peekLast() : E

Queue Method	Deque Method
add(e) : boolean	addLast(e) : boolean
offer(e) : boolean	offerLast(e) : boolean
remove() : E	removeFirst() : E
poll() : E	pollFirst() : E
element() : E	getFirst() : E
peek() : E	peekFirst() : E

Stack Method	Deque Method
push(e) : boolean	addFirst(e) : boolean
pop() : E	removeFirst() : E
peek() : E	peekFirst() : E

Deque 範例

```
01 import java.util.*;
02 public class DequeExample {
03     public static void main(String[] args) {
04         //Queue
05         Deque q = new LinkedList();
06         q.offer("One");
07         q.offer("Second");
08         q.offer("3rd");
09         System.out.println("Offer: One, Second, 3rd");
10         System.out.println(q.toString());
11         //Queue 中取得元素內容，並移除該元素
12         System.out.println("Poll: " + q.poll());
13         System.out.println(q.toString());
14         //Queue 中取得元素內容，但不移除該元素
15         System.out.println("Peek: " + q.peek());
16         System.out.println(q.toString());
```

利用 **offer()** 將元素資料
加到 **Deque** 集合中

利用 **poll()** 取得並移除
Deque 集合第一個元素內容

利用 **peek()** 取得
Deque 集合中第一個
元素，不移除該元素

利用 **pop()** 取得並移除
Deque 集合最後加入元素

利用 **peek()** 取得 **Deque** 集合
中最後加入元素，不移除該元素

```
Push: One, Second, 3rd
[3rd, Second, One]
Pop: 3rd
[Second, One]
Peek: Second
[Second, One]

C:\JavaClass>
```

```
System.out.println();
```

//Stack

```
Deque s = new LinkedList();
```

```
s.push("One");
s.push("Second");
s.push("3rd");
```

利用 **push()** 將元素資料
加到 **Deque** 集合中

```
System.out.println("Push: One, Second, 3rd");
System.out.println(s.toString());
```

//Stack 中取得元素內容，並移除該元素

```
System.out.println("Pop: " + s.pop());
System.out.println(s.toString());
```

//Queue 中取得元素內容，但不移除該元素

```
System.out.println("Peek: " + s.peek());
System.out.println(s.toString());
```

```
C:\JavaClass>java DequeExample
Offer: One, Second, 3rd
[One, Second, 3rd]
Poll: One
[Second, 3rd]
Peek: Second
[Second, 3rd]
```

23
24
25

30
31
32

Collection具體類別比較

介面	實作類別	可重複	加入順序	排序	Thread safe
List	Vector	Yes		No	Yes
	Stack				No
	ArrayList				No
List Queue Deque	LinkedList	Yes		No	No
Set	HashSet	No	No	No	No
	LinkedHashSet		Yes		
	TreeSet			Yes	
Map (鍵值)	Hashtable	No	No	No	Yes
	Properties				
	HashMap		Yes	Yes	No
	LinkedHashMap				
	TreeMap				

課程大綱

- 1) Java 的集合架構
- 2) 泛型
 - 泛型應用
 - 自訂泛型類別
 - 泛型與多型
- 3) 集合進階

泛型 Generics

- 泛型提供了參數化指定型別的功能
 - JavaSE 5.0 加入的功能
 - 提供更有彈性的型別安全機制
 - 防止將不適當的資料加入集合中
 - 去除累贅的型別轉換敘述
 - 將潛在執行時期的錯誤，在編譯時期檢查出來
- 實務上，經常使用在集合型態的物件
 - 集合中大多會儲存相同類別的物件

沒有泛型之前

```
01 public class ShirtPackage {  
02     public int packageID = 1001;  
03     public Shirt shirt;  
04  
05     public void pack(Shirt s) {  
06         shirts = s;  
07     }  
08  
09     public Shirt open() {  
10         return shirt ;  
11     }  
12 }  
13
```

```
01 public class PhonePackage {  
02     public int packageID = 2001;  
03     public Phone phone;  
04  
05     public void pack(Phone p) {  
06         phone = p;  
07     }  
08  
09     public Phone open() {  
10         return phone;  
11     }  
12 }  
13
```

類別邏輯性質相似,但需撰寫多個相似邏輯的類別

沒有泛型之前

01	public class Package{	01	public class OrderTest {
02	public int packageID = 1001;	02	
03	public Object item;	03	public static void main(String[] args) {
04		04	Package pack1 = new Package();
05	public void pack(Object o) {	05	Package pack2 = new Package();
06	item = o;	06	
07	}		pack1.pack(new Shirt());
08	public Object open() {		pack2.pack("Hello World!");
09	return item;	09	
10	}		Shirt s0 = (Shirt)pack1.open();
11	}		Shirt s1 = (Shirt)pack2.open();
12		13	
13		14	}

語法上沒有錯誤，
編譯器檢查不出

要轉換資料型態

執行時會丟出
ClassCastException

如果改以共同父類別Object實作，
需做型別轉換，並有潛在的執行時期的錯誤

泛型 Generics

- Java 5.0後 Collections API 均加入泛型機制
- 使用泛型類別

泛型類別 <指定的型別參數>

- 宣告此類別中,成員為某種指定型別

- 使用泛型類別語法

- Java 7 以前

```
Collection <Type> data = new Collection<Type> ();
```

- Java 7 Diamond 型別推論

```
Collection <Type> data = new Collection <> ();
```

泛型 Generics

- 使用泛型類別的優點
 - 編譯時會做型別檢查
 - 由集合中取出資料,不必再轉型
 - 泛型檢查資訊只在編譯時期,不會帶到執行時期
 - 泛型的設定資訊,編譯之後,不會保留在類別檔之中
 - 類別供他人使用時,無法限制他人的使用方式

泛型 Generics

■ 注意事項

- 型別參數必須是參考型別,不可是基本型別
- 方法的覆載或多載,不考慮不同的型別參數

```
Vector<String> getMyVector() {....}
```

```
Vector<Integer> getMyVector() {....}
```

泛型List 範例

```
01 import java.util.*;
02
03 public class GenericExample1 {
04     public static void main(String[] args) {
05         List<String> data = new ArrayList<>();
06         data.add("One");
07         data.add("Second");
08         data.add("3rd");
09
10         for (String str: data) {
11             System.out.println(str);
12         }
13     }
14 }
```

List 宣告名稱
時指定型態

實體化時指定型態

加入元素時檢查
型態是否符合

使用時不需要
再轉換型態

泛型List 範例-編譯時型別檢查

```
01 import java.util.*;
02
03 public class GenericExample2 {
04     public static void main(String[] args) {
05         List<String> data = new ArrayList<>();
06         data.add("One");
07         data.add("Second");
08         data.add("3rd");
09         data.add(new Integer(4));
10
11         for (String str: data) {
12             System.out.println(str);
13         }
14     }
15 }
```

加入其他型態元素發生編譯錯誤

泛型Set 範例

```
01 import java.util.*;
02
03 public class GenSetExample {
04     public static void main(String[] args) {
05         Set<Integer> data = new LinkedHashSet<>();
06         data.add(new Integer(1));
07         data.add(new Integer(2));
08         data.add(new Integer(3));
09
10         Iterator<Integer> it = data.iterator();
11         while (it.hasNext()) {
12             int i = it.next();
13             System.out.println(i);
14         }
15     }
16 }
```

Set 宣告名稱
時指定型態

實體化時指定型態

Iterator宣告名
稱時指定型態

使用時不需要
再轉換型態

泛型Set 範例-編譯時型別檢查

```
01 import java.util.*;
02
03 public class GenSetExample2 {
04     public static void main(String[] args) {
05         Set<Integer> data = new LinkedHashSet<>();
06         data.add(new Integer(1));
07         data.add(new Integer(2));
08         data.add(new Integer(3));
09         data.add("Four");
10
11         Iterator<Integer> it = data.iterator();
12         while (it.hasNext()) {
13             int i = it.next();
14             System.out.println(i);
15         }
16     }
17 }
```

編譯發生錯誤

泛型 Map 範例

宣告一個**HashMap**集合, 指定
Key為**String**, **Value**為**Account**型別

```
01 public class Account {
02     private static int nextID = 1000;
03     private String accountID;
04     private double balance;
05     private String customer;
06
07     public Account(String name){ ..... }
08
09     public String getAccountID(){ .... }
10     public void deposit(double amount){ .... }
11     public void withdraw(double amount){ .... }
12     public double getBalance(){ .... }
13     public String getDetails(){ .... }
14     public String getCustomer(){
15         return customer;
16     }
17
18 }
```

```
01 Import java.util.*;
02 public class MapAcctRepository {
03     HashMap<String, Account> accounts;
04
05     public MapAcctRepository(){
06         accounts = new HashMap<> ();
07     }
08
09     public void put(Account account) {
10         String locator = account.getCustomer();
11         accounts.put(locator, account);
12     }
13
14     public Account get(String locator) {
15         Account acct = accounts.get(locator);
16         return acct;
17     }
18 }
```

泛型集合與非泛型集合比較

分類	非泛型類別	泛型類別
類別宣告	<code>public class ArrayList extends AbstractList implements List</code>	<code>public class ArrayList <E> extends AbstractList <E> implements List<E></code>
建構子宣告	<code>public ArrayList (int capacity)</code>	<code>public ArrayList (int capacity)</code>
方法宣告	<code>public void add(Object o)</code> <code>public Object get(int index)</code>	<code>public void add(E o)</code> <code>public E get(int index)</code>
變數宣告	<code>ArrayList list1;</code> <code>ArrayList list2;</code>	<code>ArrayList <String> list1;</code> <code>ArrayList <Date> list2;</code>
實例宣告	<code>list1 = new ArrayList (10);</code> <code>list2 = new ArrayList (10);</code>	<code>list1 = new ArrayList <String> (10);</code> <code>list2 = new ArrayList <Date> (10);</code>

已存在的非泛型集合程式

```
01 import java.util.*;
02 public class GenericsWarning {
03     public static void main(String[] args) {
04         List list = new ArrayList();
05         list.add(0, new Integer(42));
06         int total = ((Integer)list.get(0)).intValue();
07         System.out.println("total = " + total);
08     }
09 }
```

```
系統管理員: 命令提示字元

c:\JavaClass>javac GenericsWarning.java
Note: GenericsWarning.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

```
系統管理員: 命令提示字元

c:\JavaClass>javac -Xlint:unchecked GenericsWarning.java
GenericsWarning.java:5: warning: [unchecked] unchecked call to add(int,E) as a member of the raw type List
        list.add(0, new Integer(42));
            ^
    where E is a type-variable:
      E extends Object declared in interface List
1 warning

c:\JavaClass>
```

自訂泛型類別

■ 自訂泛型類別

- 類別宣告後使用<>宣告一個參數化型別變數
- 常用參數化型別變數名稱
 - `<T> : Type` 一般型別
 - `<E> : Element` 集合內的元素
 - `<K, V> : (Key, Value)` 鍵-值對
- 程式中使用參數化型別變數來宣告變數、方法、傳入參數、傳回值的型別

泛型宣告及使用

```
01 public class Package <T> {  
02     public int packageID = 1001;  
03     public T item;  
04  
05     public void pack(T t) {  
06         item = t;  
07     }  
08  
09     public T open() {  
10         return item;  
11     }  
12 }
```

Package宣告
時指定型態

編譯發生錯誤

使用時不需要
再轉換型態

```
01 public class OrderTest {  
02  
03     public static void main(String[] args) {  
04  
05         Package <Shirt> pack1 = new Package<>();  
06         Package <Phone> pack2 = new Package<>();  
07  
08         pack1.pack( new Shirt() );  
09         pack2.pack("Hello World!");  
10  
11         Shirt s1 = pack1.open();  
12         Phone p1 = pack2.open();  
13     }  
14 }
```

diamond型別推論

Java 泛型的多型

- 基底型別可作多型自動轉型
- 型別參數不可作多型自動轉型

```
LinkedList<Integer> l1 = new LinkedList<Integer>();  
List<Integer> l2 = l1;
```



```
LinkedList<Integer> l1 = new LinkedList<Integer>();  
LinkedList<Number> l3 = l1;
```



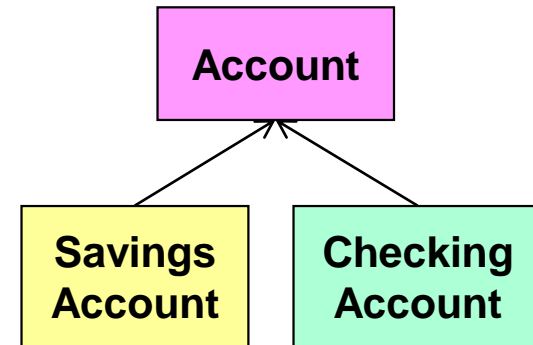
```
LinkedList<Integer> l1 = new LinkedList <Integer>();  
LinkedList l4 = l1; 14編譯時不會做型別檢查
```



Java 泛型的不變性 (Invariance)

```
List <CheckingAccount> lc = new LinkedList <CheckingAccount>();  
lc.add(new CheckingAccount("Fred")); //沒問題  
lc.add(new SavingsAccount("John")); //編譯錯誤! 不正確資料無法加入
```

```
CheckingAccount ca1 = lc.get(0); //安全, 取出時不需要型別轉換
```



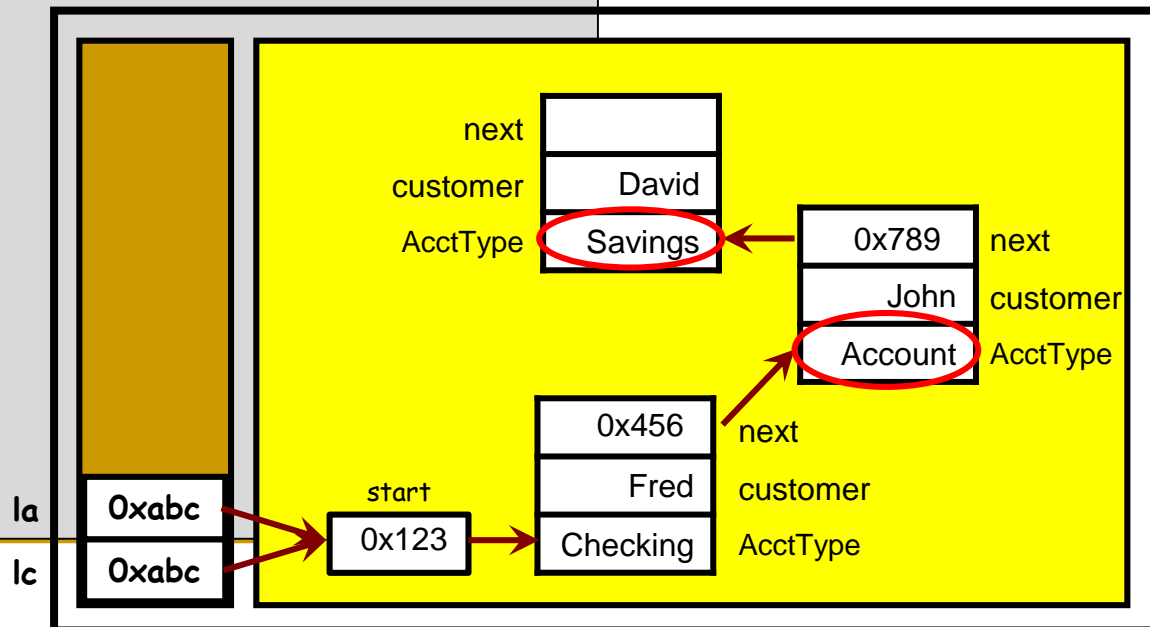
```
List <CheckingAccount> lc = new LinkedList <CheckingAccount>();  
List <Account> la = lc; //假設型別參數可以多形自動轉型
```

```
lc.add(new CheckingAccount("Fred"));  
la.add(new Account("John"));  
la.add(new SavingsAccount("David"));
```

```
CheckingAccount ca1 = lc.get(0);
```

```
CheckingAccount ca2 = lc.get(1);
```

```
CheckingAccount ca3 = lc.get(2);
```



未知型態型別參數 - Wildcard<?>

```
01 import java.util.*;
02 public class GenericsExample4 {
03     public static void main(String[] args) {
04         List<String> names = new ArrayList<String>();
05         names.add("Jacky");
06         names.add("Piggy");
07         print(names);
08
09         List<Integer> data = new LinkedList<Integer>();
10         data.add(1);
11         data.add(2);
12         print(data);
13     }
14
15     public static void print(List<?> data) {
16         Iterator<?> it = data.iterator();
17         while(it.hasNext()){
18             System.out.print( it.next()+"", " ");
19         }
20         System.out.println();
21     }
22 }
```

讀取未知型態傳入參數, 型態由方法呼叫者動態指定

未知型態
Iterator物件

Covariance & Contravariance

■ 共變性（Covariance）

- 如果B是A的子類別，則 `Collection` 可視為一種 `Collection<A>`
- Java的泛型不支援共變性 – 對集合新增元素時會造成問題
- 唯讀資料可用萬用字元?與`extends`來宣告變數，達到類似共變性
- 若宣告?不搭配`extends`，則預設為? `extends Object`

```
public void eat(Collection <? extends Fruit> basket) {  
    for(Fruit f : basket){  
        .....  
    }  
}
```

```
Collection<Apple> bask1 = new Collection {  
    new Apple(), new Apple(), new Apple() }  
Collection<Fruit> bask2 = new Collection {  
    new Apple(), new Pear(), new Banana() }  
eat (bask1);  
eat (bask2);
```

限制型別參數- <? extends XXX >

```
01 import java.util.*;
02 public class GenericsExample5 {
03     public static void main(String[] args) {
04         List<Integer> data1 = new LinkedList<Integer>();
05         data1.add(1);
06         data1.add(2);
07         sum(data1);
08
09         List<Double> data2 = new LinkedList<Double>();
10         data2.add(5.3);
11         data2.add(7.6);
12         sum(data2);
13     }
14
15     public static void sum(List<? extends Number> data) {
16         double sum = 0.0;
17         for(Number num : data)
18             sum += num.doubleValue();
19
20         System.out.println("SUM = " + sum);
21     }
22 }
```

保留資料的彈性與
處理上的相容性

讀取某種 Number
型態的物件

Covariance & Contravariance

■ 逆變性（Contravariance）

- 如果B是A的子類別，則 `Collection<A>` 可視為一種 `Collection`
- Java的泛型並不支援逆變性 – 對集合讀取元素時需要轉型
- 寫入的資料可用萬用字元? 與`super`來宣告變數，達到類似逆變性效果

```
public void pack(Collection <? super Apple> basket ){  
    for(int i=0; i<10; i++){  
        basket.add(new Apple());  
    }  
}
```

```
Collection<Apple> bask1= new Collection<>( );  
Collection<Fruit> bask2 = new Collection<>( );  
pack(bask1);  
pack(bask2);
```

限制型別參數- <? super XXX >

```
01 import java.util.*;
02 public class GenericsExample6 {
03     public static void main(String[] args) {
04         List<Integer> data1 = new LinkedList<Integer>();
05         addNumber(data1);
06
07         List<Number> data2 = new LinkedList<Number>();
08         addNumber(data2);
09
10     }
11
12     public static void addNumber(List<? super Integer> data) {
13         for(int i = 1; i <= 10; i++)
14             data.add(i);
15     }
16
17 }
```

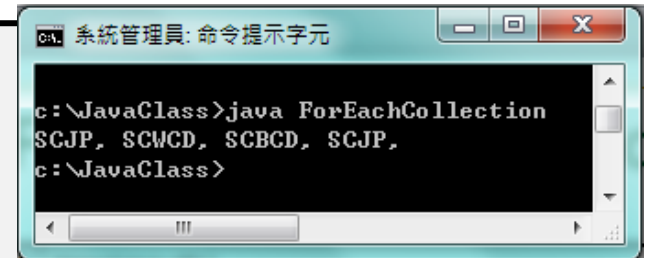
元素可加入泛型宣告為
Integer父類別的集合

課程大綱

- 1) Java 的集合架構
- 2) 泛型
- 3) 集合進階
 - ❑ **For Each** 與 集合
 - ❑ 集合排序
 - ❑ **Collections** 及 **Arrays** 類別

For Each 與集合

```
01 import java.util.*;  
02 public class ForEachCollection {  
03     public static void main(String[] args) {  
04         ArrayList al = new ArrayList();  
05         al.add("SCJP");  
06         al.add("SCWCD");  
07         al.add("SCBCD");  
08         al.add("SCJP");  
09  
10         for(Object obj : al) {  
11             String data = (String)obj ;  
12             System.out.print(data + ", " );  
13         }  
14     }  
15 }
```



```
系統管理員: 命令提示字元  
c:\JavaClass>java ForEachCollection  
SCJP, SCWCD, SCBCD, SCJP,  
c:\JavaClass>
```

for (元素資料型別 區域變數名稱 : 元素集合名稱)
不需額外定義索引值變數來控制集合的讀取
可自動擷取集合元素，直到全部擷取完畢為止

For Each 與泛型集合

```
01 import java.util.*;
02
03 public class GenericsExample3 {
04     public static void main(String[] args) {
05         List<String> data = new ArrayList<>();
06         data.add("Hello");
07         data.add("World");
08
09         for(String str : data)
10             System.out.println(str);
11     }
12 }
```

For Each 不需去控制迴圈使用變數
存取集合元素變得更簡單易懂也更安全

集合排序

■ 集合排序

- 可排序的集合：**TreeSet** 及 **TreeMap**
- 集合元素需為相同資料型態
- 需提供排序之邏輯：
 - 集合元素實作 **java.lang.Comparable** interface
 - 定義元素原生的排序規則
 - 使用**comparator** 物件
 - 幫未實作**comparable**的元素排序
 - 以自訂排序取代原生排序規則

集合排序

- `java.lang.Comparable<T>` 介面
 - `public int compareTo(T o)` 提供元素原生的排序規則
- `java.util.Comparator<T>` 介面
 - `public int compare(T o1, T o2)` 定義 comparator 物件的排序規則
 - 建立 `Comparator` 物件
 - 以 `Comparator` 物件為傳入參數建立 `TreeSet` / `TreeMap` 物件

Comparable 範例

```
01 public class Student implements Comparable<Student> {
02     String firstName, lastName;
03     int studentID=0;
04     double GPA = 0.0;
05     public Student(String first, String last, int ID, double gpa){
06         if(first==null || last==null || ID==0 || gpa==0.0 ) {
07             throw new IllegalArgumentException();
08         }
09         this.firstName = first;
10         this.lastName = last;
11         this.studentID = ID;
12         this.GPA = gpa;
13     }
14     public String firstName() { return firstName; }
15     public String lastName() { return lastName; }
16     public int studentID() { return studentID; }
17     public double GPA() { return GPA; }
18     public int compareTo(Student s){
19         double f = GPA-s.GPA;
20         if ((f==0.0))      return 0;  //0 表示相等
21         else if(f < 0.0)   return -1; // -1表示小於或排序在前
22         else               return 1; // +1表示大於或排序在後
23     }
24 }
```

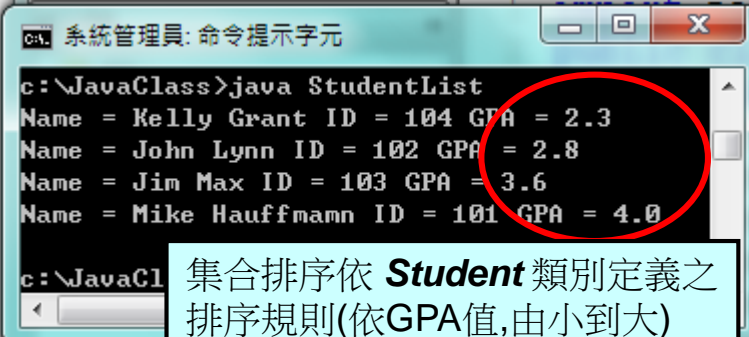
集合元素實作 **comparable** 介面

實作 **compareTo(Object o)** 方法
定義原生的排序規則

Comparable 範例

```
01 import java.util.*;
02 public class StudentList {
03     public static void main(String[] args) {
04         TreeSet studentSet = new TreeSet();
05         studentSet.add(new Student("Mike", "Hauffmann", 101, 4.0));
06         studentSet.add(new Student("John", "Lynn", 102, 2.8));
07         studentSet.add(new Student("Jim", "Max", 103, 3.6));
08         studentSet.add(new Student("Kelly", "Grant", 104, 2.3));
09         Object[] studentArray = studentSet.toArray();
10         Student s;
11         for(Object obj : studentArray){
12             s = (Student) obj;
13             System.out.println( "Name = " + s.firstName()
14                                 " " + s.lastName() +
15                                 " ID = " + s.studentID() +
16                                 " GPA = " + s.GPA());
17         }
18     }
19 }
```

將 **Student** 物件加入
可排序集合 **TreeSet**



```
c:\JavaClass>java StudentList
Name = Kelly Grant ID = 104 GPA = 2.3
Name = John Lynn ID = 102 GPA = 2.8
Name = Jim Max ID = 103 GPA = 3.6
Name = Mike Hauffmann ID = 101 GPA = 4.0
```

集合排序依 **Student** 類別定義之
排序規則(依GPA值,由小到大)

Comparator 範例

01	<code>public class Student2 {</code>	集合元素不實作 <i>comparable</i> 介面
02	<code> String firstName, lastName;</code>	
03	<code> int studentID=0;</code>	
04	<code> double GPA = 0.0;</code>	
05	<code> public Student2(String first, String last, int ID, double gpa){</code>	
06	<code> if(first==null last==null ID==0 gpa==0.0) {</code>	
07	<code> throw new IllegalArgumentException();</code>	
08	<code> }</code>	
09	<code> this.firstName = first;</code>	
10	<code> this.lastName = last;</code>	
11	<code> this.studentID = ID;</code>	
12	<code> this.GPA = gpa;</code>	
13	<code> }</code>	
14	<code> public String firstName() { return firstName; }</code>	
15	<code> public String lastName() { return lastName; }</code>	
16	<code> public int studentID() { return studentID; }</code>	
17	<code> public double GPA() { return GPA; }</code>	
18	<code>}</code>	

Comparator 範例

```
01 import java.util.*;
02 public class NameComp implements Comparator<Student2> {
03     public int compare(Student2 o1, Student2 o2){
04         return o1.firstName.compareTo(o2.firstName);
05     }
06 }
```

實作 **comparator** 介面
指定泛型型別參數為 **Student2**

實作 **compare(Student2 o1, Student2 o2)**
方法定義 **Comparator** 的排序規則

```
01 import java.util.*;
02 public class GPAComp implements Comparator<Student2> {
03     public int compare(Student2 o1, Student2 o2){
04         if (o1.GPA == o2.GPA)
05             return 0; //0 表示相等
06         else if(o1.GPA < o2.GPA)
07             return -1; //-1表示小於或排序在前
08         else
09             return 1; //+1表示大於或排序在後
10     }
11 }
```

實作 **comparator** 介面
指定泛型型別參數為 **Student2**

實作 **compare(Student2 o1, Student2 o2)**
方法定義 **Comparator** 的排序規則

Comparator 範例

建立 **Comparator** 物件 **c (GPAComp)**
建立可排序集合 **TreeSet** 傳入 **Comparator c**

建立 **Comparator** 物件 **c (NameComp)**
建立可排序集合 **TreeSet** 傳入 **Comparator c**

```
01 import java.util.*;  
02 public class StudentList2 {  
03     public static void main(String[] args) {  
04         Comparator<? super Student2> c = new GPAComp();  
05         TreeSet<Student2> studentSet = new TreeSet<>(c);  
06         studentSet.add(new Student2("Mike", "Hauffmamn", 101, 4.0));  
07         studentSet.add(new Student2("John", "Lynn", 102, 2.8));  
08         studentSet.add(new Student2("Jim", "Max", 103, 3.6));  
09         studentSet.add(new Student2("Kelly", "Grant", 104, 2.3));  
10  
11         for(Student2 s : studentArray){  
12             System.out.println( "Name = "+s.firstName()+  
13                 " "+s.lastName()+  
14                 " ID = "+s.studentID()+  
15                 " GPA = "+s.GPA());  
16         }  
17     }  
18 }  
19 }
```

```
c:\JavaClass>java StudentList2  
Name = Jim Max ID = 103 GPA = 3.6  
Name = John Lynn ID = 102 GPA = 2.8  
Name = Kelly Grant ID = 104 GPA = 2.3  
Name = Mike Hauffmamn ID = 101 GPA = 4.0
```

集合排序依 **NameComp** 類別定義之排序規則(依firstName屬性值由小到大排序)

```
c:\JavaClass>java StudentList2  
Name = Kelly Grant ID = 104 GPA = 2.3  
Name = John Lynn ID = 102 GPA = 2.8  
Name = Jim Max ID = 103 GPA = 3.6  
Name = Mike Hauffmamn ID = 101 GPA = 4.0
```

集合排序依 **GPAComp** 類別定義之排序規則(依GPA屬性值由小到大排序)

Arrays vs. Collections

■ java.util.Arrays

- ❑ 定義多種 **static methods**，提供陣列作排序、搜尋、比較等資料操控

■ java.util.Collections

- ❑ 定義多種 **static methods**，提供**Collection**作排序、搜尋、比較等資料操控
- ❑ 將非執行緒同步集合包裹成為執行緒同步集合的包覆方法
- ❑ 將集合包裹成為不可變更集合的包覆方法

Arrays 類別常用方法

方法名稱	傳回值	說明
asList(T... a)	static <T> List<T>	將指定型態陣列轉換為指定型態泛型的List後傳回
equals(int[] a, int[] a2)	static boolean	比較兩個 int 陣列是否相等
equals(Object[] a, Object[] a2)	static boolean	比較兩個物件陣列是否相等
fill(int[] a, int val)	static void	以指定的 int 值val填入指定 int 陣列中的每個元素
fill(Object[] a, Object val)	static void	以指定的物件參考val填入指定物件陣列中的每個元素
sort(int[] a)	static void	對指定的 int 陣列按數字進行升冪排序。
sort(Object[] a)	static void	對指定的物件陣列按物件排序規則進行升冪排序。
sort(T[] a, Comparator<? super T> c)	static <T> void	根據指定Comparator的排序規則，將指定的物件陣列進行升冪排序。
binarySearch(int[] a, int key)	static int	使用二元搜尋法搜尋指定的 int 陣列，傳回搜尋元素索引位置 執行前需對int 陣列進行升冪排序
binarySearch(Object[] a, Object key)	static int	使用二元搜尋法搜尋指定的物件陣列，傳回搜尋元素索引位置 執行前需對物件陣列進行升冪排序
binarySearch(T[] a, T key, Comparator<? super T> c)	static int	使用二元搜尋法搜尋指定的泛型陣列，傳回搜尋元素索引位置 執行前需對泛型陣列以Comparator的排序規則進行升冪排序

Collections 類別常用方法

方法名稱	傳回值	說明
copy(List<? super T> dest, List<? extends T> src)	static <T> void	所有元素從來源List複製到目標List
replaceAll(List<T> list, T oldVal, T newVal)	static <T> boolean	使用newVal值替換List中出現的所有oldVal值
reverse(List<?> list)	static void	反轉指定List中元素的順序。
shuffle(List<?> list)	static void	對指定List進行隨機順序調換(洗牌)。
fill(List<? super T> list, T obj)	static <T> void	使用指定元素替換指定List中的所有元素。
sort(List<T> list)	static <T extends Comparable<? super T>> void	根據元素的自然排序規則對指定List進行升冪排序
sort(List<T> list, Comparator<? super T> c)	static <T> void	根據Comparator的排序規則對指定List進行升冪排序
binarySearch(List<? extends Comparable<? super T>> list, T key)	static <T> int	使用二元搜尋法搜尋指定List，傳回搜尋元素索引位置 執行前需對List進行升冪排序
binarySearch(List<? extends T> list, T key, Comparator<? super T> c)	static <T> int	使用二元搜尋法搜尋指定List，傳回搜尋元素索引位置 執行前需對List以Comparator的排序規則進行升冪排序

Collections 類別常用方法

方法名稱	傳回值	說明
synchronizedCollection(Collection<T> c)	static <T> Collection<T>	將指定 Collection 轉換為執行緒安全的Collection傳回
synchronizedList(List<T> list)	static <T> List<T>	將指定 List 轉換為執行緒安全的List傳回
synchronizedSet(Set<T> s)	static <T> Set<T>	將指定 collection 轉換為執行緒安全的Set傳回
synchronizedSortedSet(SortedSet<T> s)	static <T> SortedSet<T>	將指定 collection 轉換為執行緒安全的SortedSet傳回
synchronizedMap(Map<K,V> m)	static <K,V> Map<K,V>	將指定 collection 轉換為執行緒安全的Map傳回
synchronizedSortedMap(SortedMap<K,V> m)	static <K,V> SortedMap<K,V>	將指定 collection 轉換為執行緒安全的SortedMap傳回
unmodifiableCollection(Collection<? extends T> c)	static <T> Collection<T>	將指定 Collection 轉換為不可變更的視圖View傳回
unmodifiableList(List<? extends T> list)	static <T> List<T>	將指定 List 轉換為不可變更的視圖View傳回
unmodifiableSet(Set<? extends T> s)	static <T> Set<T>	將指定 Set 轉換為不可變更的視圖View傳回
unmodifiableSortedSet(SortedSet<T> s)	static <T> SortedSet<T>	將指定 SortedSet 轉換為不可變更的視圖View傳回
unmodifiableMap(Map<? extends K,? extends V> m)	static <K,V> Map<K,V>	將指定 Map 轉換為不可變更的視圖View傳回
unmodifiableSortedMap(SortedMap<K,? extends V> m)	static <K,V> SortedMap<K,V>	將SortedMap 轉換為不可變更的視圖View傳回