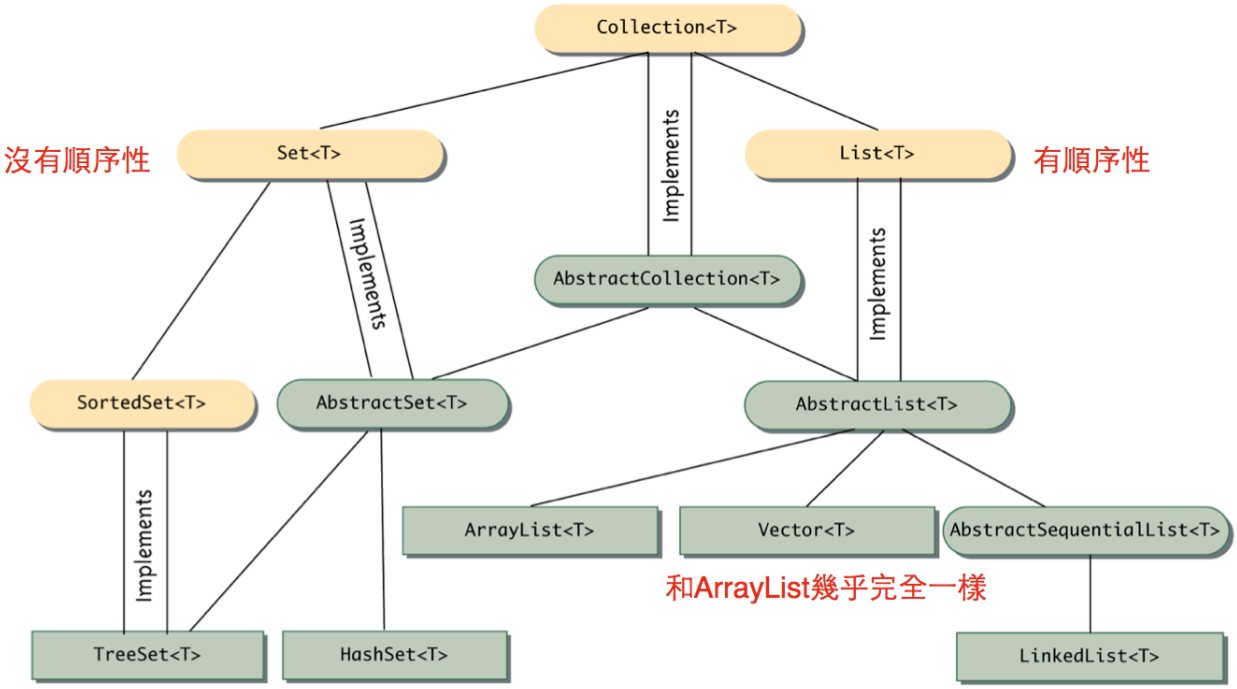
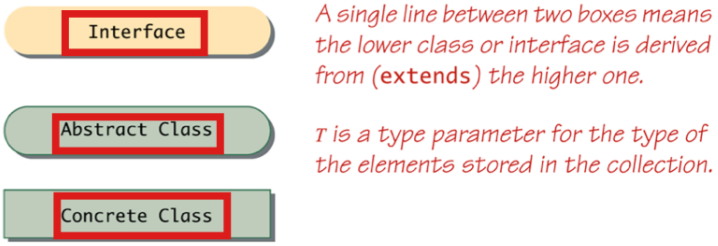
Chapter 16 Collections, Maps and Iterators

1. Java Collection
    a. any class that holds objects and implements the `Collection` interface
    b. `ArrayList<T>` implements all the methods in the `Collection` interface
    c. `Collection` is the highest level of Java's framework for collection classes



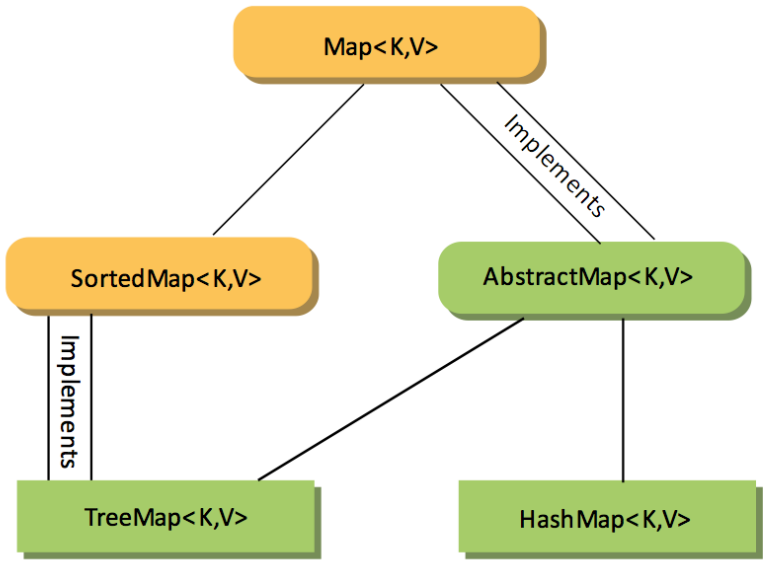


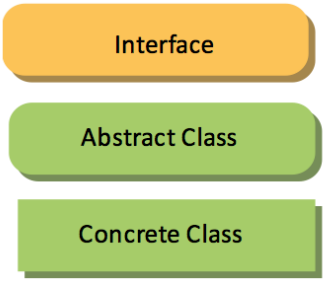
2. Wildcards (【電腦】萬用字元，通配符)
    a. no specified type parameter, use "?" to be a wide range of argument types
    b. syntax:
    ```
    public void method(String arg1, ArrayList<?> arg2)
    ```
    c. bounded wildcards: specifying the type to be an ancestor type or descendent type of some class or interface
    ```
    <? extends String>    //descendent class of String
    <? super String> //ancestor class of String
    ```
    d. <個人補充>wildcards vs type parameter
    ```
    <N extends Number> Collection<N> getThatCollection(Class<N> type)
    ```

```
Collection<? extends Number> getThatCollection(Class<? extends Number>)
```

   i. first declaration: passed argument and return type are the same type
   ii. second declaration: passed argument and return type doesn't have to be the same type

3. `Collection` Framework
   a. `Collection<T>` interface describes the basic operations that all collection classes should implement
   b. method headings example
      i. boolean isEmpty()
      ii. public boolean contains(Object target)
      iii. public boolean containsAll(Collection<?> collectionOfTargets)
      iv. public boolean equals(Object other)
      v. public int size()
      vi. Iterator<T> iterator()
      vii. public Object[] toArray()
      viii. public <E> E[] toArray(E[] a)
      ix. public int hashCode()
      x. public boolean add(T element) (Optional)
      xi. public boolean addAll(Collection<? extends T> collectionToAdd()) (Optional)
      xii. public boolean remove(Object element) (Optional)
      xiii. public void clear() (Optional)
      xiv. public boolean retainAll(Collection<?> saveElements) (Optional)
   c. "Optional" operation means the method does not completely implement its intended semantics and still has to be implemented!!
   d. If a trivial implementation is given, the method body should throw an `UnsupportedOperationException`
4. Concrete Collections Classes
   a. Set<T> interface has concrete classes such as…
      i. HashSet<T>
      ii. TreeSet<T>
   b. List<T> interface has concrete classes such as…
      i. ArrayList<T>
      ii. Vector<T>
      iii. LinkedList<T>
5. `Map` Framework
   a. deals with collections of ordered pairs(key and associated value)





A single line between two boxes means the lower class or interface is derived from (extends) the higher one.

K and V are type parameters for the type of the keys and elements stored in the map.

b. method headings
    i. boolean isEmpty()
    ii. public boolean containsValue(Object value)
    iii. public boolean containsKey(Object key)
    iv. public boolean equals(Object other)
    v. public int size()
    vi. public int hashCode()
    vii. public Set<Map.Entry<K,V>> entrySet()
    viii. public Collection<V> values()
    ix. public V get(Object key)
    x. public V put(K key, V value) (Optional)
    xi. public void putAll(Map<? extends K, ? extends V> mapToAdd) (Optional)
    xii. public V remove(Object key) (Optional)
c. unordered map
    i. HashMap<K,V>
        1. load factor(between 0 and 1): If number of elements in hash table exceeds the load factor, then the capacity of the hash table is automatically increased (default is 0.75)
        2. initial capacity is 16

```
1       // This class uses the Employee class defined in Chapter 7.
2       import java.util.HashMap;
3       import java.util.Scanner;
4       public class HashMapDemo
5       {
6         public static void main(String[] args)
7         {
8               // First create a hashmap with an initial size of 10 and
9               // the default load factor
10        創造物件 HashMap<String,Employee> employees =
11                      new HashMap<String,Employee>(10);

12              // Add several employees objects to the map using
13              // their name as the key
14              employees.put("Joe",
15                      new Employee("Joe",new Date("September", 15, 1970)));
16              employees.put("Andy",
17                      new Employee("Andy",new Date("August", 22, 1971)));
18        塞資料 employees.put("Greg",
19                      new Employee("Greg",new Date("March", 9, 1972)));
20              employees.put("Kiki",
21                      new Employee("Kiki",new Date("October", 8, 1970)));
22              employees.put("Antoinette",
23                      new Employee("Antoinette",new Date("May", 2, 1959)));
24              System.out.print("Added Joe, Andy, Greg, Kiki, ");
25              System.out.println("and Antoinette to the map.");

26              // Ask the user to type a name.  If found in the map,
27              // print it out.
28              Scanner keyboard = new Scanner(System.in);
29              String name = "";
30              do
31              {
32                System.out.print("\nEnter a name to look up in the map. ");
33                System.out.println("Press enter to quit.");
34                name = keyboard.nextLine();
35                if (employees.containsKey(name))
36                {
37          取資料  Employee e = employees.get(name);
38                  System.out.println("Name found: " + e.toString());
39                }
40                else if (!name.equals(""))
41                {
42                  System.out.println("Name not found.");
43                }
44              } while (!name.equals(""));
45      }
46  }
```

3. if want to use custom class as parameterized type K, then custom class need to override…

```
public int hashCode();
public boolean equals(Object obj);
```

d. ordered map

i. TreeMap<K, V>
ii. LinkedHashMap<K, V>

6. `Iterators` Interface
   a. object that provide sequential access to the collection elements
   b. method headings:
      i. `public T next()`
      ii. `public boolean hasNext()`
      iii. `public void remove()` (Optional)
   c. `HashSet<T>` object imposes no order on the elements it contains, but iterator will impose an order on the elements in the hash set

```
1  import java.util.HashSet;
2  import java.util.Iterator;

3  public class HashSetIteratorDemo
4  {
5      public static void main(String[] args)
6      {
7          HashSet<String> s = new HashSet<String>();

8          s.add("health");
9          s.add("love");
10         s.add("money");

11         System.out.println("The set contains:");

12         Iterator<String> i = s.iterator();
13         while (i.hasNext())
14             System.out.println(i.next());

15         i.remove();

16         System.out.println();
17         System.out.println("The set now contains:");

18         i = s.iterator();
19         while (i.hasNext())
20             System.out.println(i.next());

21         System.out.println("End of program.");
22     }
23 }
```

You cannot "reset" an iterator "to the beginning." To do a second iteration, you create another iterator.

SAMPLE DIALOGUE

```
The set contains:
money
love
health

The set now contains:
money
love
End of program.
```

The HashSet<T> object does not order the elements it contains, but the iterator imposes an order on the elements.

d. For-Each Loops can serve the same purpose as an iterator, same results as above

```
1  import java.util.HashSet;
2  import java.util.Iterator;

3  public class ForEachDemo
4  {
5      public static void main(String[] args)
6      {
7          HashSet<String> s = new HashSet<String>();

8          s.add("health");
9          s.add("love");
10         s.add("money");

11         System.out.println("The set contains:");
12         String last = null;
13         for (String e : s)
14         {
15            last = e;
16            System.out.println(e);
17         }

18         s.remove(last);

19         System.out.println();
20         System.out.println("The set now contains:");

21         for (String e : s)
22             System.out.println(e);

23         System.out.println("End of program.");
24     }
25 }
```

The output is the same as in Display 16.8.

7. `ListIterator<T>` interface

a. extends `Iterator<T>` interface
b. designed to work with collections that satify the `List<T>` interface
c. can move in either direction
d. method headings
    i. public T next()
    ii. public T previous()
    iii. public boolean hasNext()
    iv. public int nextIndex()
    v. public int previoudIndex()
    vi. public void add(T newElement) (Optional)
    vii. public void remove() (Optional)
    viii. public void set(T newElement) (Optional)
e. for example:

The class `Date` *is defined in Display 4.13, but you can easily guess all you need to know about* `Date` *for this example.*

```
1  import java.util.ArrayList;
2  import java.util.Iterator;

3  public class IteratorReferenceDemo
4  {
5      public static void main(String[] args)
6      {
7          ArrayList<Date> birthdays = new ArrayList<Date>();


8          birthdays.add(new Date(1, 1, 1990));
9          birthdays.add(new Date(2, 2, 1990));
10         birthdays.add(new Date(3, 3, 1990));


11         System.out.println("The list contains:");
12         Iterator<Date> i = birthdays.iterator();
13         while (i.hasNext())
14             System.out.println(i.next());

15         i = birthdays.iterator();
16         Date d = null; //To keep the compiler happy.
17         System.out.println("Changing the references.");
18         while (i.hasNext())
19         {
20             d = i.next();
21             d.setDate(4, 1, 1990);
22         }
23         System.out.println("The list now contains:");


24         i = birthdays.iterator();
25         while (i.hasNext())
26             System.out.println(i.next());


27         System.out.println("April fool!");
28     }
29 }
```

SAMPLE DIALOGUE

```
The list contains:
January 1, 1990
February 2, 1990
March 3, 1990
Changing the references.
The list now contains:
April 1, 1990
April 1, 1990
April 1, 1990
April fool!
```

8. Programmer-defined Iterator
    a. suggested: derive from library collection classes
    b. if collection class must be customized in some other way, then an iterator class should be defined as an inner class of the collection class