Chapter 14 Generics and the ArrayList Class

1. Generics: type parameter enabled coding
    a. applies to any class
2. `ArrayList`
    a. container that can grow and shrink
    b. has private instance variable: `Array`
        i. when Array is full, new larger `Array` is created and data is transferred
    c. less efficient than `Array`
    d. syntax

```
//BaseType can not be primitive type
ArrayList<BaseType> aList = new ArrayList<BaseType>();

//intial capacity of 20 items
ArrayList<BaseType> aList = new ArrayList<BaseType>(20);

aList.add("something");
int howMany = aList.size();
aList.set(index, "something else");//replace index
String thing = aList.get(index);
boolean hasString = aList.contains("something");
aList.remove(index);
aList.clear();
```

    e. can be used with for-each

```
for (String entry: aList)
      System.out.println(entry);
```

    f. `ArrayList` grows automatically but does not shrink automatically, so use `trimToSize` to save memory
    g. `clone` method makes a shallow copy
    h. `Vector` class is almost same as `ArrayList`, but `ArrayList` is newer and prefered
3. generic class (or parameterized class)

```
1  public class Sample<T>
2  {
3      private T data;

4      public void setData(T newData)
5      {
6          data = newData;
7      }

8      public T getData()
9      {
10         return data;
11     }
12 }
```

T is a parameter for a type.

    a. definition: class definition with type parameter

b. type parameter is included in angular brackets(<>) after the class name in the class definition heading

c. type parameter no needed to be included in the constructor

```
1  public class Pair<T>
2  {
3      private T first;
4      private T second;

5      public Pair()
6      {
7          first = null;
8          second = null;
9      }

10     public Pair(T firstItem, T secondItem)
11     {
12         first = firstItem;
13         second = secondItem;
14     }
```

Constructor headings do not include the type parameter in angular brackets.

不用特別再寫T了

d. type parameter cannot be used in expressions to create a new object當別人使用這個class的時候，必須要指定好型態！

```
T object = new T();   //illegal
T[] a = new T[10];    //illegal
```

e. generic class cannot be an `Array` base type

```
Pair<String>[] a = new Pair<String>[10];
```

f. multiple type parameters

```
1  public class TwoTypePair<T1, T2>
2  {
3      private T1 first;
4      private T2 second;

5      public TwoTypePair()
6      {
7          first = null;
8          second = null;
9      }

10     public TwoTypePair(T1 firstItem, T2 secondItem)
11     {
12         first = firstItem;
13         second = secondItem;
14     }
```

g. generic class cannot be an `Exception` class

```
public class Pair<T> extends Exception//illegal
```

h. bounds for type parameters

   i. restricting the possible types that can be plugged in for T

```
public class RClass<T extends Comparable>
public class ExClass<T extends Class1>
```

```
public class Two<T1 extends Class1, T2 extends Class2
                    & Comparable>
//不可以class & class，因為Java沒有多重繼承！
```

4. generic methods
    a. definition: type parameter used in the definitions of the methods for an ordinary class or a generic class
    b. type parameter of a generic method is local to that method, not to the class
    c. syntax

```
public static <T> T genMethod(T[] a)   //<T>和T順序不可以錯！

String s = NonG.<String>genMethod(c);
```

5. inheritance with generic class

```
1  public class UnorderedPair<T> extends Pair<T>
2  {
3      public UnorderedPair()
4      {
5          setFirst(null);
6          setSecond(null);
7      }

8      public UnorderedPair(T firstItem, T secondItem)
9      {
10         setFirst(firstItem);
11         setSecond(secondItem);
12     }
13     public boolean equals(Object otherObject)
14     {
15         if (otherObject == null)
16             return false;
17         else if (getClass() != otherObject.getClass())
18             return false;
19         else
20         {
21             UnorderedPair<T> otherPair =
22                             (UnorderedPair<T>)otherObject;
23             return (getFirst().equals(otherPair.getFirst())
24                && getSecond().equals(otherPair.getSecond()))
25                ||
26                    (getFirst().equals(otherPair.getSecond())
27                && getSecond().equals(otherPair.getFirst()));
28         }
29     }
30 }
```

繼承後T要綁在一起

```
1  public class UnorderedPairDemo
2  {
3     public static void main(String[] args)
4     {
5          UnorderedPair<String> p1 =
6               new UnorderedPair<String>("peanuts", "beer");
7          UnorderedPair<String> p2 =
8               new UnorderedPair<String>("beer", "peanuts");
9          if (p1.equals(p2))
10         {
11             System.out.println(p1.getFirst() + " and " +
12                       p1.getSecond() + " is the same as");
13             System.out.println(p2.getFirst() + " and "
14                                + p2.getSecond());
15         }
16    }
17 }
```

6.