7 The Collections Framework

Introduction

This chapter will introduce the reader to the collections framework – an important part of the standard Java library.

Objectives

By the end of this chapter you will be able to…

- Understand the basic concepts of the Java Collections Framework
- Contrast the characteristics of various Collections interfaces
- Understand the related concept of an iterators

Download free eBooks at bookboon.com
Click on the ad to read more

This chapter consists of twelve sections:-

1) An Introduction to Collections
2) Collection Interfaces
3) Old and New Collections
4) Lists
5) Sets
6) Maps
7) Collection Implementations
8) Overview of the Collections Framework
9) An Example Using Un-typed Collections
10) An Example Using Typed Collections
11) A Note About Sets
12) Summary

7.1 An Introduction to Collections

Most software systems need to store various groups of entities rather than just individual items. Arrays provide one means of doing this, but the Java Collections support much more varied and flexible forms of grouping.

In Java, collections (or 'containers') are classes which serve to hold together groups of other objects. The Java platform provides a 'Collections Framework', a consistent set of interfaces and implementations

- interfaces define the available functionality
- implementations influence other issues including performance

7.2 Collection Interfaces

At the core of the collections framework is an interface called 'Collection'. This defines methods that are at the core of the framework. List and Set are interfaces that are extensions of Collection – they inherit all its operations and add some more.

Note that because these are interfaces, not classes, they only define operation signatures, not any aspect of their implementation as methods.

An important additional interface is 'Map'. This is not an extension of Collection – this is because maps do not entirely fit in to the Collection hierarchy for reasons we will see later, although they are still part of the 'Collections Framework'.

Download free eBooks at bookboon.com

Thus we have the following interfaces…

- **Collection**: the most general grouping
    - **List:** a collection of objects (which can contain duplicates) held in a specific order
    - **Set**: a collection of unique objects in no particular order
        - **SortedSet**: a set with objects arranged in ascending order

- **Map**: a collection of unique 'keys' and associated objects
    - **SortedMap**: a map with objects arranged in ascending order of keys

Just as List and Set extend the Collection interface, SortedSet is an extension of Set and SortedMap is an extension of Map.

We will look at List, Sets and Maps in this chapter.

7.3 Old and New Collections

Up to Java SDK 1.4, collections held items of type Object – i.e. objects of any class since all are subclasses of Object. This allowed a mixture of objects in a collection. However when objects are taken out of a collection the compiler does not know what the object is and this can cause some additional complications.

In practise we almost always want only objects of some specific type in a particular collection. Thus in Java JDK 5.0 'generics' were introduced. 'Generics' allow a class to be defined without specifying the types of data items it handles. Instead the types are specified when the class is instantiated, and so are fixed for particular objects.

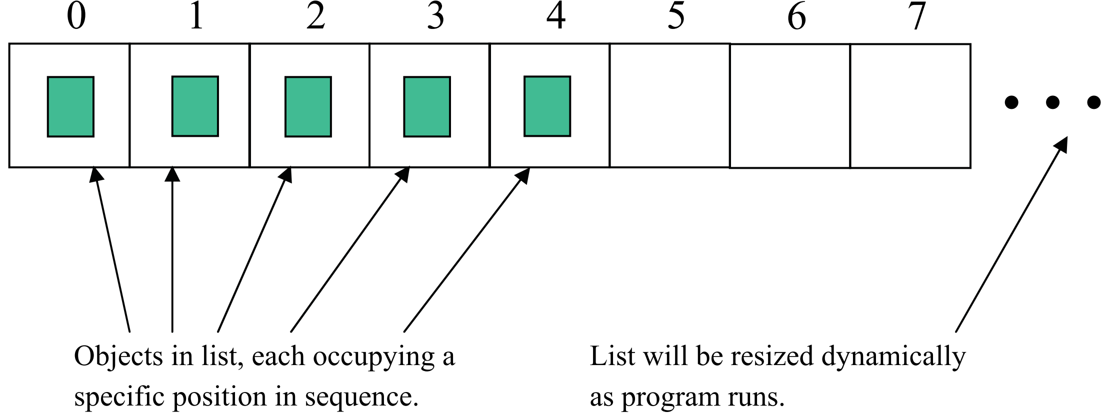
Thus using 'generics' we can create collections that only store objects of a specified type.

7.4 Lists

Lists are the most commonly used type of collection – where you might have used an array, a List will often provide a more convenient method of handling the data.

Lists are very general-purpose data structures, with each item occupying a specific position. They are in many ways like arrays, but are more flexible as they are automatically resized as data is added. They are also much easier to manipulate than arrays as many useful methods have been created that do the bulk of the work for you. Lists store items in a particular sequence (though not necessarily sorted into any meaningful order) and duplicate items are permitted.

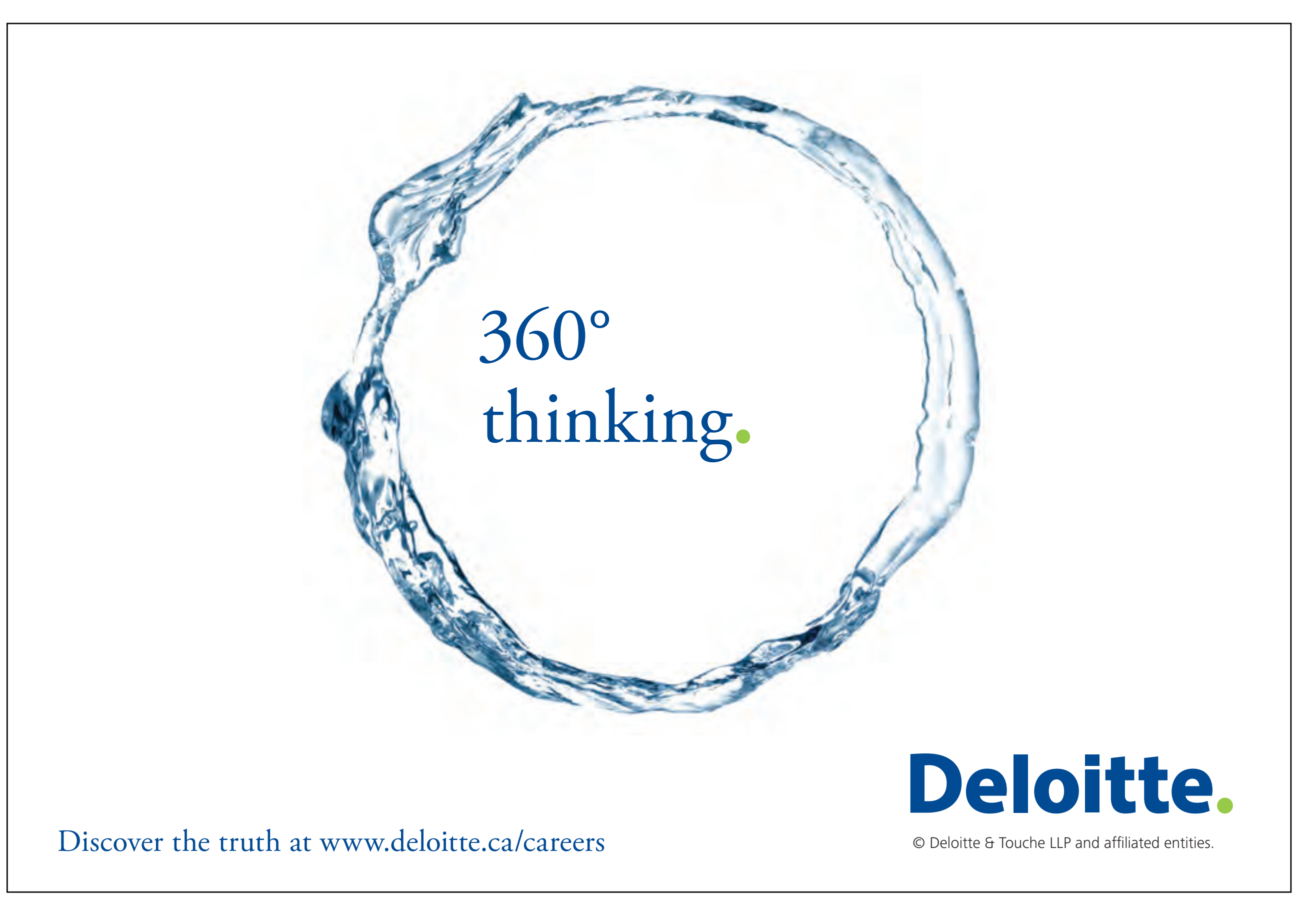
Download free eBooks at bookboon.com



7.5 Sets

A set is like a 'bag' of objects rather than a list. They are based on the mathematical idea of a 'set' – a grouping of 'members' that can contain zero, one or many distinct items.

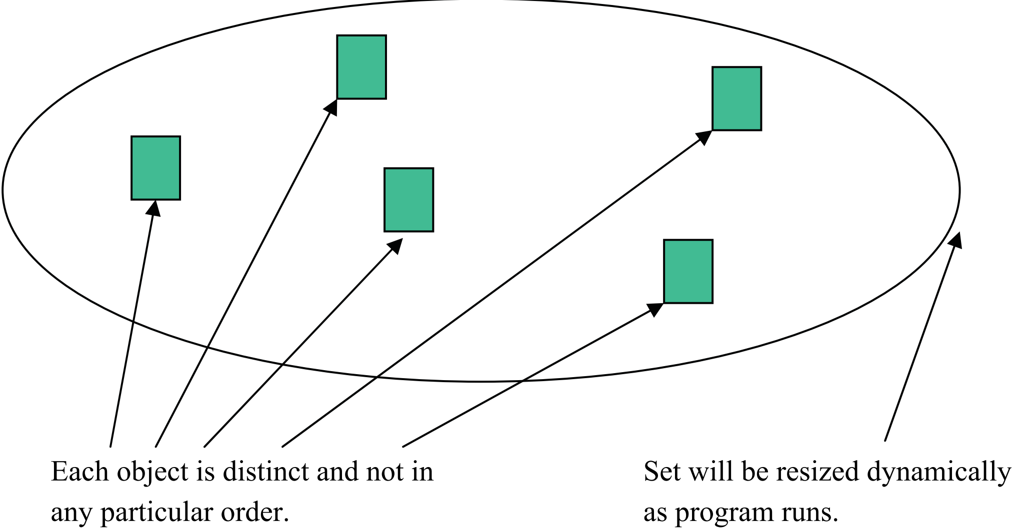
Unlike a List, duplicate items are not permitted in a set and a Set does not arrange items in order (but a SortedSet does).



Download free eBooks at bookboon.com

Click on the ad to read more

Like lists, sets are resized automatically as items are added



Many of the operations available for a List are also available for a Set, but

- we CANNOT add an object at a specific position since the elements are not in any order
- we CANNOT 'replace' an item for the same reason (though we can add one and delete another)
- retrieving all the items is possible but the sequence is indeterminate
- it is meaningless to find what position an element is in.

7.6 Maps

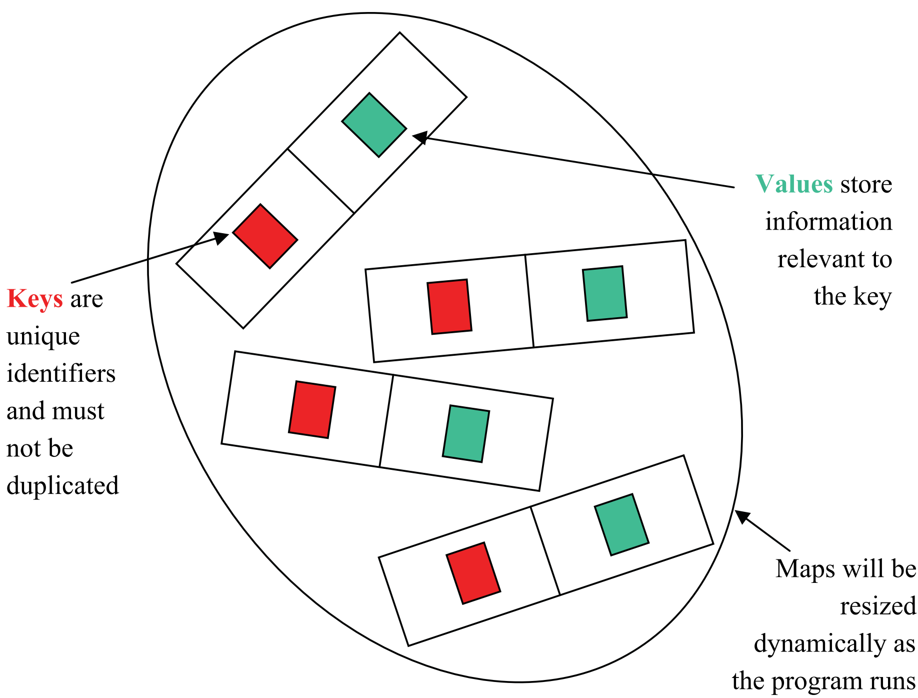
Maps are rather different from Lists and Sets because instead of storing individual objects they store pairs of objects. The pair is made up of a 'key' and a 'value'. The key is something which identifies the pair. The value is a piece of information or an object associated with the key.

Example: in an address book the keys would be people's names and the values their address, phone, email etc. There is only one value for each key, but since values are objects they can contain several pieces of data.

Duplicate keys are not permitted, though duplicate values are. So in the previous example if you looked up two people in the address book you may find them living at the same address – but one person would not have two homes.

Like a set, a Map does not arrange items in order (but a SortedMap does, in order of keys) and like lists and sets, maps are resized automatically as items are added or deleted.

Download free eBooks at bookboon.com



Activity 1

For each of the following problems write down which would be most appropriate:- a list, a set or a map.

1) We want to record the members of a club.
2) We want to keep a record of the money we spend (and what it was spent on).
3) We want to record bank account details – each identified by a bank account number

Feedback 1

1) For this we would use a Set. Members can be added and removed as required and the members are in no particular order.
2) For this we would use a List – this would record the items bought in the order in which they were purchased. Importantly as lists allows duplicate items we could buy two identical toys (perhaps for birthday presents for two different children),
3) We would not have two identical Bank accounts so a Set seems appropriate however as each is identified by a unique account number a Map would be the most appropriate choice.

7.7 Collection Implementations

Up to now we have been looking at the collections **interfaces** – specifications of functionality from which we can pick what we want to use.

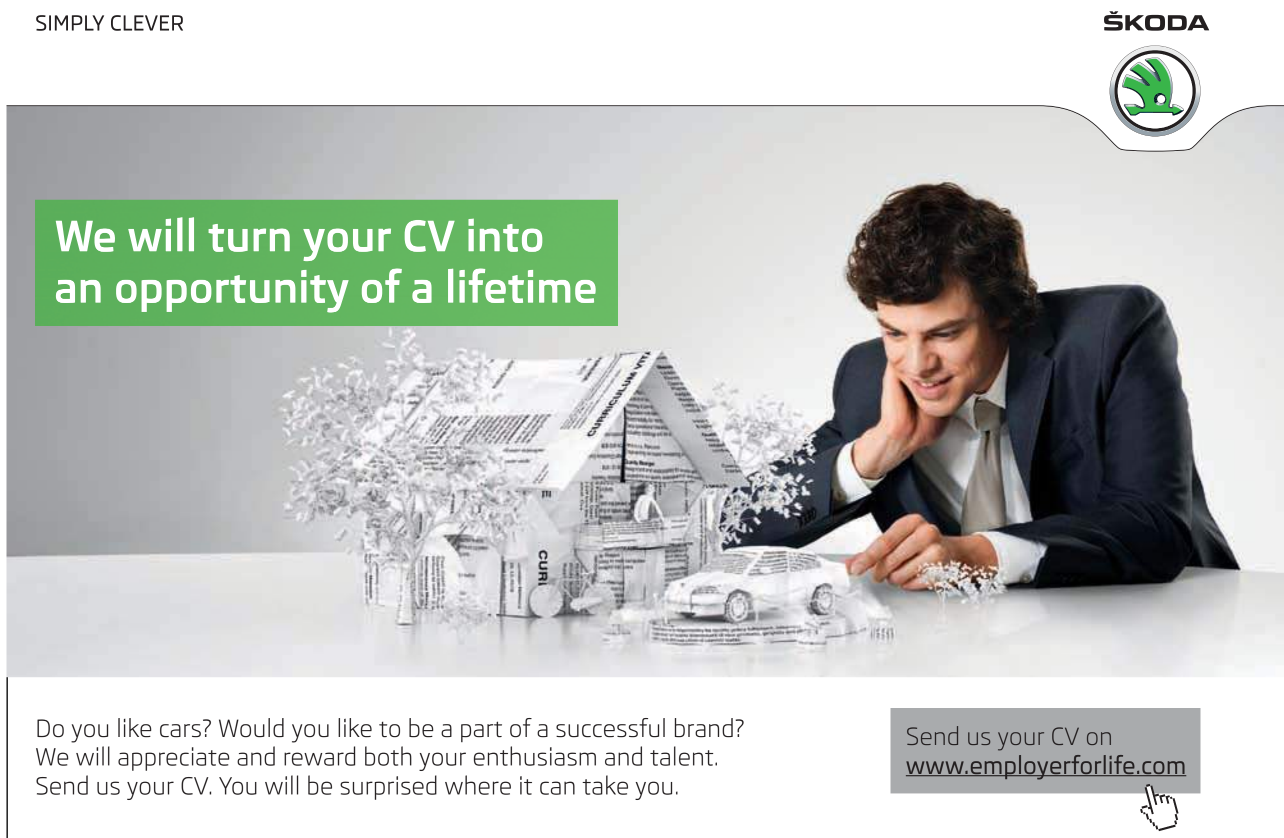
To actually use them we need **classes which implement** these interfaces.

Download free eBooks at bookboon.com

The implementations shown here are all part of the Java platform library packages, but programmers can also write implementations of their own for special purposes.

- **ArrayList** implements the List interface
  - generally fast access for an ordered list
- **LinkedList** this also implements the List interface
  - may be faster than ArrayList in some less usual circumstances
- **HashSet** implements the Set interface
  - It provide fast access but the items are unordered
- **TreeSet** this also implements the Set interface
  - slower than the HashSet but it maintain elements in order ie. it provides us with a SortedSet
- **HashMap** implements the Map interface
  - It provide fast access but the items are unordered
- and **TreeMap** which also implements the Map interface
  - slower than a HashMap it maintain elements in order of the Key ie. it provides us with a SortedMap

You can't create a 'List object' as List is an interface – but you can create an ArrayList object or a LinkedList object, and either of these will provide the functionality defined by the List interface.



Download free eBooks at bookboon.com

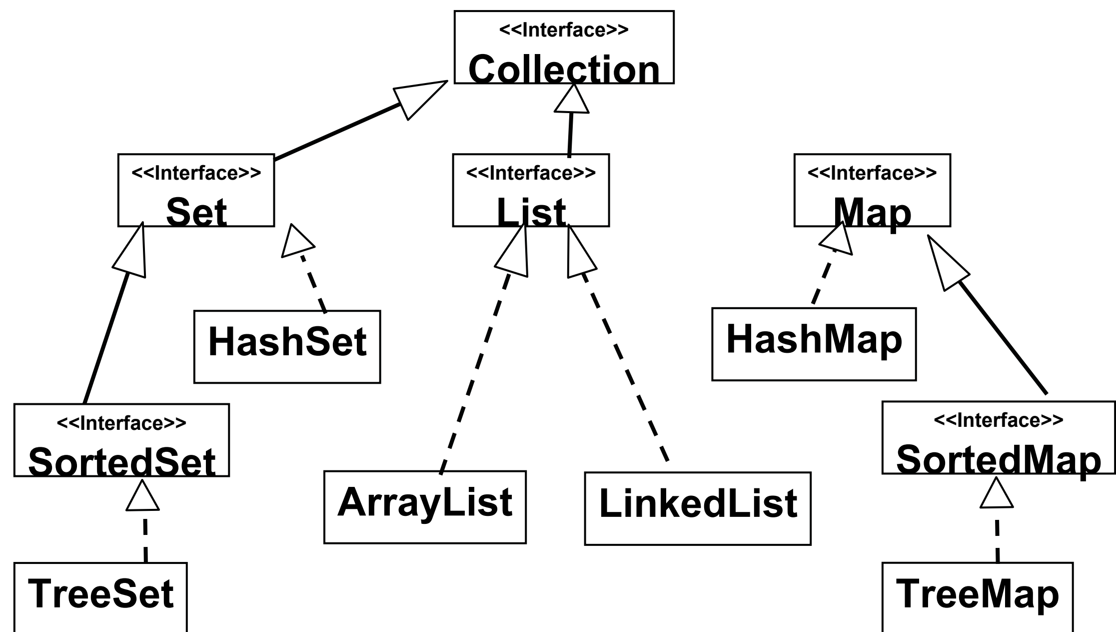
Click on the ad to read more

7.8 Overview of the Collections Framework

The diagram below gives us a slightly simplified overview of the collections framework. It shows the interfaces and the classes that implement the interfaces. Remember, interfaces have no implementation within them, they just contain public method signatures with must be fulfilled by any class implementing that interface.

There are two types of arrows shown. Arrows with solid lines denote inheritance; e.g. the **Set** interface extends the **Collection** interface. Arrows with dotted lines denote implementation; e.g. the **HashSet** class implements the **Set** interface

Note that Map is not a subinterface of Collection. This is because, unlike Lists and Sets which store individual objects, Maps store pairings of key objects and value objects. We still consider Map to be part of the collections framework.



Activity 2

Documentation for all the collections interfaces and implementations is available to browse online or download at:

http://java.sun.com/javase/6/docs/api/

The interfaces and implementation classes of the Collections Framework can all be found in the **java.util** package.

Use the online API documentation to find the methods for an ArrayList.

List three of the methods you think would be most useful.

Download free eBooks at bookboon.com

Feedback 2

Some of the clearly useful methods include…

add() – which adds an element into a specified position in the list
clear() – which clears the list
contains() – which returns true if this list contains the specified object.
get() – which returns the object at a specified position and
remove() – which removes an object from a list

Note some of these methods are overloaded such as remove() which can either remove an object at a specified position or remove the first instance of a specified object.

Click on the ad to read more

Download free eBooks at bookboon.com

7.9 An Example Using Un-typed Collections

The code below shows an example of an Un-typed list of Strings.

```
import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;
public class ListDemo
{
	List mAList;

	/**
	* Constructor
	*/
	public ListDemo ()
	{
		mAList = new ArrayList();
	}

	/**
	* Append a string at the end of the list
	* @param pStr string to be added to list
	*/
	public void appendString(String pStr)
	{
		mAList.add(pStr);
	}

	/**
	* Insert a string at a specified position in the list
	* @param pPos position at which string to be added
	*                                        (0 = before first)
	* @param pStr string to be added to list
	*/
	public void insertString(int pPos, String pStr)
	{
		mAList.add(pPos, pStr);
	}

	/**
	* Delete the string at a specified position in the list
	* @param pPos position at which string to be added
	*/
	public void deleteString(int pPos)
	{
		mAList.remove(pPos);
	}
```

Download free eBooks at bookboon.com

```
        /**
        * Display list of strings
        */
        public void display()
        {
                String nextItem;
                Iterator it = mAList.iterator();

                while (it.hasNext())
                {
                        nextItem = (String)it.next();
                        System.out.print(nextItem + " ");
                }
                System.out.println();
        }
}
```

Download free eBooks at bookboon.com
Click on the ad to read more

Mostly the code above is self explanatory however perhaps the display method needs some explanation. Firstly List, Set and Map classes implement the iterator() method this method returns an object of type Iterator and this object is capable of iterating around Lists, Sets or Maps. Note – this is not a simple process as each collection can be made up of different objects but, thankfully the hard work has been done for us.

Thus in the display() method above a variable 'it' is created of the general type Iterator. The method iterator() is then invoked on our specific collection thus returning an object capable of iterating around our List of Strings. The Iterator class has two very useful methods:- hasNext() which return true is another object exists in the collection and next() which returns the next object in the collection.

Thus the loop above iterates around the list rerunning and displaying each object.

There is one final thing to note: because we are using un-typed collection each object in the list could be a different type of object and hence when an object is retrieved from the list the JRE does not know what type of object it is – however as we are only using this list to store Strings we can cast each object retrieved onto a String. This will cause a compiler warning as this will fail if we add non String objects to the list.

While this program stores and manipulates a list of Strings the program could just as easily store a list of Publication – or any object constructed from a class we create.

7.10 An Example Using Typed Collections

Using typed collections this is simpler for two reasons:- 1) because we know what type of object will be returned from the list and hence we don't need to use the cast operator and 2) because Java 5.0 introduced an improved for loop that can loop through every item in a typed collection without using a loop counter (or iterator).

Download free eBooks at bookboon.com

Hence see example below….

```
import java.util.List;
import java.util.ArrayList;
public class ListDemo

{
	private List<String> mAList;

	/**
	* Constructor
	*/
	public ListDemo ()
	{
		mAList = new ArrayList<String>();
	}

	/**
	* Display list of strings
	*/
	public void display()
	{
		for (String nextItem: mAList)
		{
			System.out.print(nextItem + " ");
		}
		System.out.println();
	}
}
```

Creating a typed collection is very similar to creating an untyped collection except when we create the instance variable and create the actual collection we must specify what the collection contains (in this case it is simply a collection of Strings). You can think of < > syntax to mean 'of' ie in this case a list of strings

Appending, inserting and deleting elements is just the same as with untyped collections however displaying the collection is much simpler as we know what each element of the list contains. We don't need to cast the elements returned and in fact we don't even need an iterator (hence why there are only two import statements above.

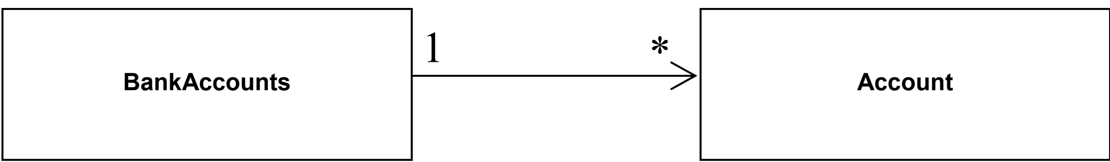
Download free eBooks at bookboon.com

nextItem is automatically assigned to the next item in the list. Note nextItem is defined as a String variable because here we are using a list of Strings. The compiler knows that and thus will ensure that the correct type of variable is used.

7.11 A Note About Sets

A thorough discussion of Sets is beyond the scope of this text however there is one complication with Sets that must be noted. Sets don't allow duplicate object to be stored. Thus we must ask what makes an object a duplicate. Are two object only the same if the name of the object is the same – or could they be the same if some of the data is identical?

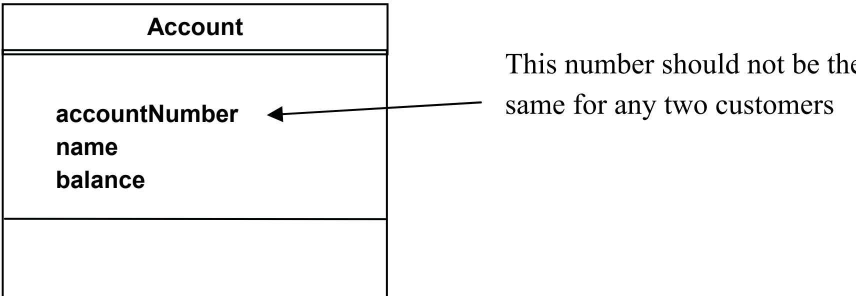
Consider a set of bank accounts:-



Download free eBooks at bookboon.com

Click on the ad to read more

Now consider two bank accounts one for Mr Smith and one for Mrs Jones



It should not be possible to create a second account with the same account number and add it to a set of bank accounts. However unless told otherwise Java will treat the two object below as different objects as both objects have different names (Account1 and Account2) and thus while Sets do not allow duplicates objects inside them both of these accounts could be created and added to a set.

Account1	Account2
1234 Mr Smith £1,200	1234 Mrs Jones £0

To overcome this problem we need to override the equals() method defined in the Object class to say these objects are the same if the accountNumber is the same.

We can do this as below…

```
public boolean equals (Object pObj)
{
      Account account = (Account) pObj;
      return (accountNumber.equals(account.accountNumber);
}
```

This overrides Object.equals() for the Account class

Download free eBooks at bookboon.com

Even though it will always be an Account object passed as a parameter, we have to make the parameter an Object type (and then cast it to Account) in order to override the equals() method inherited from Object which has the signature

public boolean equals (Object obj)

(You can check this in the "API and Language" section of the JDK documentation.)

If we gave this method a signature with an Account type parameter it would not override Object.equals(). Therefore we need to cast the parameter to an Account before extracting its accountNumber to compare them with those of the current object.

One additional complication concerns how objects are stored in sets. To do this Java uses a hashcode based on the objects name. However two accounts with the same accountNumber should generate the same hashcode even if the object name is different. To make certain this happens we need to override the hashCode() method so that a hashcode is generated using the accountNumber rather than the object name (just as we needed to override the equals() method). We can ensure that the hashcode generated is based on the account number by overriding this method as shown below….

```
/**
 * Override hashCode() inherited from Object
 * @return hashcode based on accountNumber
 */
public int hashCode()
{
      return accountNumber.hashCode();
}
```

The simplest way to redefine hashCode for an object is to join together the instance values which are to define equality as a single string and then take the hashcode of that string. In this case equality is defined purely by the accountNumber which must be unique.

It looks a little strange, but we can use the hashCode() method on this String even though we are overriding the hashCode() method for objects of type Account.

hashCode() is guaranteed to produce the same hash code for the same String. Occasionally the **same** hash code may be produced for **different** key values, but that is not a problem.

Download free eBooks at bookboon.com

By overriding equals() and hashCode() methods Java will prevent objects with duplicate data (in this case with duplicate account numbers) from being added to sets.

Activity 3

The code below will find a String in a set of Strings (called stringSet). Amend this so this will work find a Publication in a set of Publications (called publicationSet).

```
public void findString(String pStr)
{
        boolean found;

        found = stringSet.contains(pStr);

        if (found)
        {
                System.out.println("Element " + pStr + " found in set");
        }
        else
        {
                System.out.println("Element " + pStr + " NOT found in set");
        }
}
```

Enhance your career opportunities

We offer practical, industry-relevant undergraduate and postgraduate degrees in central London

› Accounting and finance
› Business, management and leadership
› Oil and gas trade management
› Global banking and finance
› Luxury brand management
› Media communications and marketing

Contact us to arrange a visit
Apply direct for January or September entry

T +44 (0)20 7487 7505 E exrel@regents.ac.uk W regents.ac.uk

Download free eBooks at bookboon.com

Click on the ad to read more

Feedback 3

```
public void findPublication(Publication pPub)
{
        boolean found;

        found = publicationSet.contains(pPub);

        if (found)
        {
                System.out.println("Element " + pPub + " found in set");
        }
        else
        {
                System.out.println("Element " + pPub + " NOT found in set");
        }
}
```

Activity 4

Look at the code in the feedback to Activity 3 and answer the following questions.

1) Could the code above be used to store a collection of books?
2) Could it store a combination of books and magazines?
3) If a book was found in the set what would the following line of code do?

System.out.println("Element " + pPub + " found in set");

Feedback 4

1) Yes
2) Yes
3) pPub would invoke the toString() method on the publication. The JRE would determine at run time that this was in fact a book and assuming the toString() method had been overridden for book, to return the title and author, this would make up part of the message displayed.

Download free eBooks at bookboon.com

7.12 Summary

The Java 'Collections Framework' provides ready-made interfaces and implementations for storing collections of objects. This almost completely makes the use of arrays redundant.

There are 'untyped' collections, and as of JDK 5.0 'typed' collections also.

Collection interfaces include List, Set and Map, each defining appropriate operations.

Collection implementations include ArrayList, HashSet and HashMap which are implementations of the List, Set and Map interfaces respectively.

Special attention is required when defining objects to be stored in Sets (or as keys in Maps) to define the meaning of 'duplicate'. For these we need to override the methods equals() and hashCode() methods inherited from Object.

While we have not been able to provide a detailed discussion of collection in this chapter the case study, in Chapter 11, will demonstrate the use of Maps and Sets for storing our own objects (not just Strings). The code for this will be available to download and inspect if required.

Download free eBooks at bookboon.com