The ArrayList class

One reason why Java is such a powerful and popular language is that, being both Object Orientated and open source there is a free library with thousands of professionally designed classes that can be used in packages.

One such class is the ArrayList at

http://java.sun.com/j2se/1.5.0/docs/api/index.html

The ArrayList has 3 important features

- it is able to increase its size automatically
- it counts how many objects it's holding
- it keeps the order of objects you insert into it.

One disadvantage of an ArrayList is that if an object is removed from inside the list the list compresses itself and so all the index numbers from that item on are reduced by 1

The fixed loop

```
for(int i=0; i<10; i++) {

        Do something here

}
```

For example

```
public void listFiles()
  {
    for(int i=0;i<files.size();i++)
    {
      System.out.println("Index: " + i + "->" + files.get(i));

    }
  }
```

The for-each loop

for each

in

```
for(Type <local variable> : <name of ArrayList>){

        Do something here

}
```

Came in later version of Java

- + Much easier to use
- - Loops over the whole collection. (Can't be stopped)
- - Has no index to refer to

An example

```
public void listAllFiles()
  {
     for(String filename:files){
        System.out.println(index + "-" +filename);
     }
  }
```

index ++;

int index =0;

Programmers have a strong preference for using a for-each loop or an Iterator instead of for-loops.

The traditional for-loop always uses a loop index - an integer that identifies each iteration of the loop. Loop indexes have always been a source of error.

For example, off-by-one errors are very common in programming, and they are often related to these loop indexes.

Since they are more error prone, for-loops should generally be avoided if there is an alternative.

Indefinite iteration

if you lose your keys ....

The while loop

```
while( boolean condition){
        do something
}
```

```
while( the keys are missing){
        look in next room
}
```

```
do{

do something

}
while( boolean condition)
```

Interfaces

An Interface is a public class that contains method signatures that are not implemented. i.e. the code for these methods has not been written.
So an Interface class tells other classes what it must do i.e. its behaviour but not how it's going to do it.
A class implements an Interface.

All Collection classes implement the Iterator interface

Iterators

http://docs.oracle.com/javase/6/docs/api/java/util/Iterator.html

The Iterator class is an Interface that is implemented by all Collection classes.

The Iterator interface specifies that its implementation must have 3 and only 3 methods.

Method Summary	
boolean	hasNext() Returns true if the iteration has more elements.
E	next() Returns the next element in the iteration.
void	remove() Removes from the underlying collection the last element returned by the iterator (optional operation).

It does not say how these methods are to be coded. Each class that implements the Iterator interface must code these three methods but each class will do it in its own way.

The Cllection classes is then provide the tree methods to their users. All the user has to do is invoke them.

Method Summary

boolean hasNext()
Returns true if the iteration has more elements.

Object next()
Returns the next element in the iteration.

void remove()
Removes from the underlying collection the last element returned by the iterator (optional operation).

A class that implements an Iterator must not remove elements using a method of its own. It must use the remove method provided by the Iterator.

The ArrayList implements the Iterator interface

http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html

java.util

Class ArrayList<E>

```
java.lang.Object
  └ java.util.AbstractCollection<E>
      └ java.util.AbstractList<E>
          └ java.util.ArrayList<E>
```

All Implemented Interfaces:
Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

Example of Iterator.bluej

Three (sometimes four) steps needed

1 Set up the Collection object

```
files = new Collection Type<Object Type>();
```

2 Invoke the Collection's iterator method

```
Iterator<Object Type> it= files.iterator();
```

3 Check if there is another element in the Collection

```
while(it.hasNext()){
        Object Type t = it.next();
                (Do something to t.......)
```

4 And sometimes if you need to

```
it.remove();
```

it is the Iterator object

hasNext() returns True/False

next() returns an object to play with

remove() removes the object

Do not use the Collections remove method if you are using an Iterator or you will confuse it poor thing.

```
import java.util.*;

private ArrayList<String> trees;

public void showTrees()
  {
      Iterator<String> it = trees.iterator();
      while (it.hasNext()) {
         System.out.println(it.next());
      }
  }
```

Attachments

package.bluej