Java – Arrays and Collections

Java has a number of ways of putting data into sets. Some of these are remarkably similar. This is because Java has evolved with new features leaving older features still valid. Some collections do, however, offer more versatility and functions at the expense of needing more code to take care of them.

Array

An array is a relatively straightforward set that has certain limitations

- All the data within an array must be of the same data type
- An array is of fixed length; its size needs to be set.

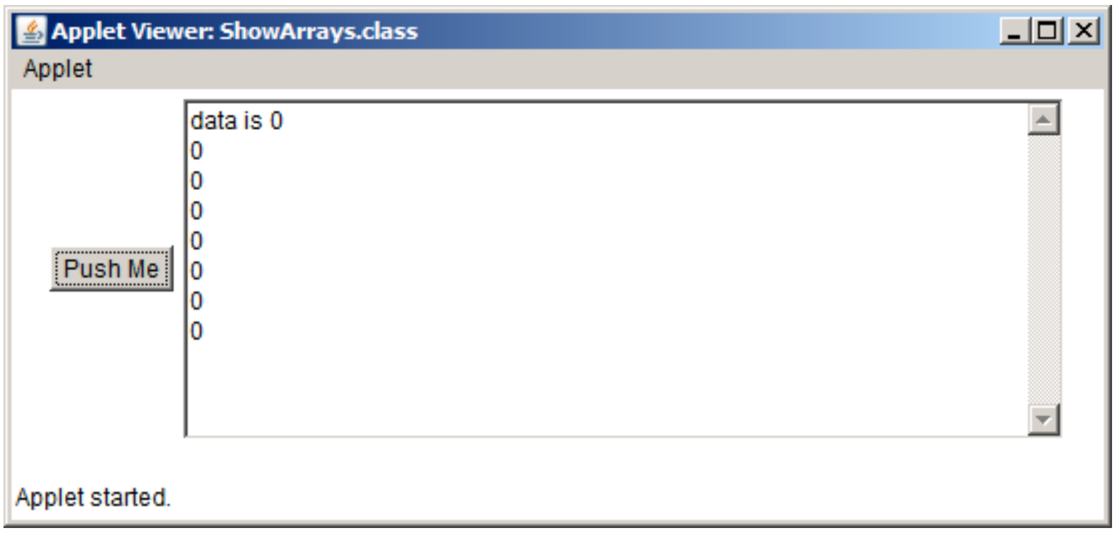
Here is a simple Applet that declares an array and outputs the values within it.

```
public class ShowArrays extends Applet implements
ActionListener

{

	private static final long serialVersionUID = 1L;
	private Button push;
	private TextArea output;
	private int []numbers;
	public void init()
	{
		push = new Button("Push Me");
		add(push);
		push.addActionListener(this);
		numbers = new int [8];
		output = new TextArea();
		add(output);
	}

	public void actionPerformed(ActionEvent event)
	{
		if(event.getSource()==push)
		{
			String data = "";
			for(int x=0; x<numbers.length;x++)
				data = data + numbers[x] + "\n";
			output.setText("data is " + data);
		}
	}
}
```



```
data is 0
0
0
0
0
0
0
0
```

The array could be filled up with a for loop:

```
for(int x=0;x<numbers.length; x++)
                    numbers[x] =x;
```

It could also be filled when instantiated:

```
private int []numbers={3,4,6,12,22,100,-3};
```

The only way to handle putting more data into an array than it has been set up for is to declare a new array and copy the old array into the new and then add more data.

Arrays

Arrays is a Java class that allows some popular array manipulation techniques through its methods.

```
//put 12 in all the elements
Arrays.fill(numbers, 12);
```

The only way to handle putting more data into an array than it has been set up for is to declare a new array and copy the old array into the new and then add more data. We can copy one array into another using a for loop but Arrays allows a short cut. Here it has been set up as a method that accepts an array together with the new value and passes back the slightly longer array.

```
private int []  addToArray(int [] set, int value)
{
  int[]moreNumbers = Arrays.copyOf(set, set.length + 1);
  moreNumbers[moreNumbers.length -1]=value;
  return moreNumbers;
}
```

The method is called as follows:

```
numbers = addToArray(numbers, 22);
```

An array can also be sorted;

```
Arrays.sort(numbers);
```

If sorted the array can be searched for data within that array. The method returns the 1st index of the searched for data or -1 if it is not present.

```
int position = Arrays.binarySearch(numbers,22);
if(position>= 0)
     data = data + " 22 is at position " + position;
```

ArrayList

ArrayList is a collection class. It acts like a turbo charged array. The ArrayList collection is extremely similar to the Vector collection. Some of the method names are, however, different.

- Objects of more than 1 class can be in the same ArrayList
- ArrayLists increase in size when full.
- ArrayList has a method to search for objects within the ArrayList.

An ArrayList can contain any data type, scalars or objects but there is a problem with getting the data back out. The ArrayList could be of any length and contain any data type so what can this data be put into if we do not know what it is? The solution is to return the data as an Iteration and run through all the members until we run out.

```
list =new ArrayList();
list.add("Bill");
list.add(1);
list.add(1.34);
Iterator set = list.iterator();
int tempCounter = 0;
while(set.hasNext())
{
     String val = set.next().toString();
     System.out.println(val);
}
```

Although the ArrayList can contain any data type the program will be more robust if it is used for a specific data type only. Here a set of Car objects or objects of a subclass of car.

```
private ArrayList <Car> nums;
nums = new ArrayList<Car>();
```

This is a console class demonstrating an ArrayList. The ArrayList is used for a set of Strrings with data being replaced and searched for:

```
import java.util.*;
public class ArrayListNames {
     public static void main(String[] args) {
          //default length is 10, here set to 6
          ArrayList <String>names = new
          ArrayList<String>(6);
          names.add("Bill");  //add as next available
          element
          // ordinary array
          String []moreNames
          ={"Charlie","John","Fred","Janet"};
          for (int x=0;x<moreNames.length;x++)
          {  // add array to ArrayList
               names.add(moreNames[x]);
          }
          int count=0;
          if (names.contains("John"))
          {
               count = names.indexOf("John");
               System.out.println("John found by name at
               position " + count);
          }
          else
               System.out.println("John has gone out");
          try
          {// try to put at non-existent element
               names.set(31,"Gill");
          }
          catch (IndexOutOfBoundsException x)
          { // add at beginning instead
               names.set(0,"Gill");
          }
          count = names.indexOf("John");
          System.out.println("John found by index at
          position " + count);
          names.set(count,"Jack");
          if (names.contains("John"))
          { //John should now have been fired
               count = names.indexOf("John");
               System.out.println("John is here at
               position " + count);
          }
          else
          {
               System.out.println("John has been replaced
               by " +
               String.valueOf(names.get(count)));
          }
          names.add("Mary");
          names.add("Lucy");
          names.add("Tom");
          // put number of elements in Iterator
          Iterator<String> vectEnum = names.iterator();
          //there are now more than 6 names in the
          ArrayList
          while (vectEnum.hasNext())
          {
               try
               {
                    System.out.println
                    (String.valueOf(vectEnum.next()));
               }
               catch (NoSuchElementException s)
               {
                    break;
               }
          }
     }
}
```

Hashtables

This is another collection class, sometimes referred to as a dictionary. The Hashtable consists of a pair of elements the object and the key. Either of which can be of any class. The key is used to reference the object. A key can only be in the same Hashtable once but the same object can be referred to by more than 1 key.

Key	Object
1	Jones
22	Smith
33	Williams

The above Hashtable does not do anything stunning as the reference number and name could be attributes of a class and we could enumerate through an ArrayList to find the name that matched the number.

The following example demonstrates this code as a console class. The data from a Hashtable could be any length so has to go into an Enumeration.

```
public static void main(String[] args) {
 //this default Hashtable has a length of 11
 //any class can be put into a Hashtable
 //Hashtables do not like scalars - int etc
   Hashtable<String, String> collection  =
   new  Hashtable<String, String>();

  collection.put("1","Jones");
  collection.put("22","Smith");
  collection.put("33","Williams");
  Enumeration getKeys = collection.keys();
  //get the keys
  while(getKeys.hasMoreElements())
  {
     String data = (String) getKeys.nextElement();
     System.out.println(data);
  }
  getKeys = collection.elements();
  while(getKeys.hasMoreElements())
  {
    String data = (String) getKeys.nextElement();
    System.out.println(data);
  }
}
```

The code does not care what is used for the key or the element, either can be retrieved individually or as an Enumeration.

```
collection.put("33","Williams");
collection.put("33", "Rogers");
```

This code will replace Williams with Rogers at key 33.

```
collection.remove("33");
```

Sack key 33.

```
if(collection.containsKey("22"))
     System.out.println("There is a 22");
if(collection.containsValue("Jones"))
     System.out.println("Jones is here");
```

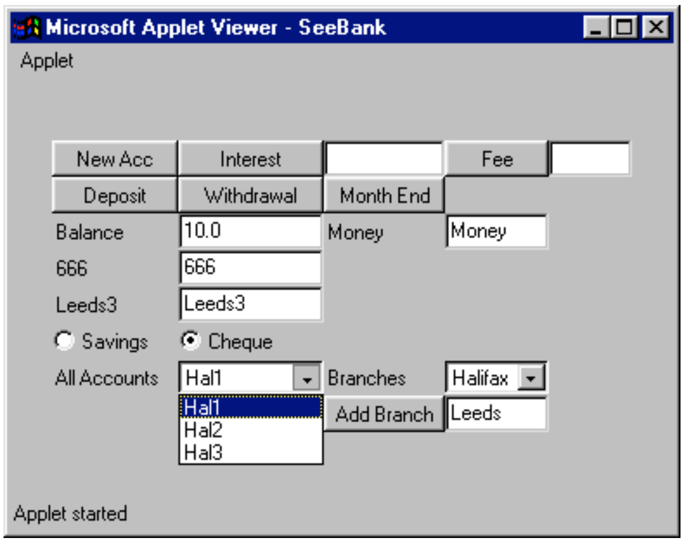
We can search for a key or a value in the HashTable

Assume that bank accounts are kept at branches. Each branch will have a number of accounts. The Hashtable will record the accounts at each branch as a ArrayList, the name of the branch will be the key.

Key	Object
Halifax	{Cheque1, Cheque2, Savings1}
Leeds	{Cheque3, Savings2, Savings3}
Bradford	{Savings4, Cheque5}

We have no idea how many branches there will be however only 1 ArrayList is required as we can shovel different sets of bank objects in and out of it. Any other sets of Bank objects are safe in the Hashtable.

Branches are recorded by a Choice. As a Choice is read only a TextField and Button will handle adding new branches.   Here a branch is just a String although it could be an Object defining some branch attributes and actions.



Balance: 10.0
Money: Money
666: 666
Leeds3: Leeds3
Savings [✓ Cheque]
All Accounts: Hal1 (Hal1, Hal2, Hal3)
Branches: Halifax
Leeds

Applet started