Java Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with **square brackets**:

String[] cars;

We have now declared a variable that holds an array of strings. To insert values to it, you can place the values in a comma-separated list, inside curly braces:

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

To create an array of integers, you could write:

```
int[] myNum = {10, 20, 30, 40};
```

Access the Elements of an Array

You can access an array element by referring to the index number.

This statement accesses the value of the first element in cars:

```
public class Main {
  public static void main(String[] args) {
    String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
    System.out.println(cars[0]);
  }
}
```

Change an Array Element

To change the value of a specific element, refer to the index number:

```
cars[0] = "Opel";
```

```
public class Main {
  public static void main(String[] args) {
    String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
    cars[0] = "Opel";
    System.out.println(cars[0]);
  }
}
```

Array Length

To find out how many elements an array has, use the `length` property:

```
public class Main {

  public static void main(String[] args) {

    String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

    System.out.println(cars.length);

  }

}
```

Java Arrays Loop

Loop Through an Array

You can loop through the array elements with the `for` loop, and use the `length` property to specify how many times the loop should run.

The following example outputs all elements in the **cars** array:

```
public class Main {

  public static void main(String[] args) {

    String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

    for (int i = 0; i < cars.length; i++) {

      System.out.println(cars[i]);

    }

  }

}
```

Loop Through an Array with For-Each

There is also a "**for-each**" loop, which is used exclusively to loop through elements in arrays:

```
for (type variable : arrayname) {

  ...

}
```

The following example outputs all elements in the **cars** array, using a "**for-each**" loop

```
public class Main {

  public static void main(String[] args) {

    String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

```
    for (String i : cars) {
      System.out.println(i);
    }
  }
```

The example above can be read like this: **for each** `String` element (called **i** - as in **i**ndex) in **cars**, print out the value of **i**.

If you compare the `for` loop and **for-each** loop, you will see that the **for-each** method is easier to write, it does not require a counter (using the length property), and it is more readable.

ava Multi-Dimensional Arrays

Multidimensional Arrays

A multidimensional array is an array of arrays.

Multidimensional arrays are useful when you want to store data as a tabular form, like a table with rows and columns.

To create a two-dimensional array, add each array within its own set of **curly braces**:

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
```

myNumbers is now an array with two arrays as its elements.

Access Elements

To access the elements of the **myNumbers** array, specify two indexes: one for the array, and one for the element inside that array. This example accesses the third element (2) in the second array (1) of myNumbers:

```
public class Main {
  public static void main(String[] args) {
    int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
    System.out.println(myNumbers[1][2]);
  }
}
```

Change Element Values

You can also change the value of an element:

```
public class Main {

  public static void main(String[] args) {

    int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

    myNumbers[1][2] = 9;

    System.out.println(myNumbers[1][2]); // Outputs 9 instead of 7

  }

}
```

Loop Through a Multi-Dimensional Array

We can also use a `for loop` inside another `for loop` to get the elements of a two-dimensional array (we still have to point to the two indexes):

```
public class Main {

   public static void main(String[] args) {

     int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

     for (int i = 0; i < myNumbers.length; ++i) {

        for(int j = 0; j < myNumbers[i].length; ++j) {

           System.out.println(myNumbers[i][j]);

        }

     }

   }

}
```

Java ArrayList

The `ArrayList` class is a resizable array, which can be found in the `java.util` package.

The difference between a built-in array and an `ArrayList` in Java, is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one). While elements can be added and removed from an `ArrayList` whenever you want. The syntax is also slightly different:

Example[Get your own Java Server]

Create an `ArrayList` object called **cars** that will store strings

```
import java.util.ArrayList; // import the ArrayList class



ArrayList<String> cars = new ArrayList<String>(); // Create an
ArrayList object
```

Add Items

The `ArrayList` class has many useful methods. For example, to add elements to the `ArrayList`, use the `add()` method:

```
import java.util.ArrayList;



public class Main {

  public static void main(String[] args) {

    ArrayList<String> cars = new ArrayList<String>();

    cars.add("Volvo");

    cars.add("BMW");

    cars.add("Ford");

    cars.add("Mazda");

    System.out.println(cars);

  }

}
```

Access an Item

To access an element in the `ArrayList`, use the `get()` method and refer to the index number:

```
import java.util.ArrayList;



public class Main {

  public static void main(String[] args) {

    ArrayList<String> cars = new ArrayList<String>();

    cars.add("Volvo");

    cars.add("BMW");
```

```
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars.get(0));
  }
}
```

Change an Item

To modify an element, use the `set()` method and refer to the index number:

```
import java.util.ArrayList;


public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    cars.set(0, "Opel");
    System.out.println(cars);
  }
}
```

Remove an Item

To remove an element, use the `remove()` method and refer to the index number:

```
import java.util.ArrayList;


public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
```

```
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    cars.remove(0);
    System.out.println(cars);
  }
}
```

To remove all the elements in the `ArrayList`, use the `clear()` method:

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    cars.clear();
    System.out.println(cars);
  }
}
```

ArrayList Size

To find out how many elements an ArrayList have, use the `size` method:

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
```

```
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars.size());
  }
}
```

Loop Through an ArrayList

Loop through the elements of an `ArrayList` with a `for` loop, and use the `size()` method to specify how many times the loop should run:

```
import java.util.ArrayList;


public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    for (int i = 0; i < cars.size(); i++) {
      System.out.println(cars.get(i));
    }
  }
}
```

You can also loop through an `ArrayList` with the **for-each** loop:

```
import java.util.ArrayList;


public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
```

```
    cars.add("Mazda");
    for (String i : cars) {
      System.out.println(i);
    }
  }
}
```

Other Types

Elements in an ArrayList are actually objects. In the examples above, we created elements (objects) of type "String". Remember that a String in Java is an object (not a primitive type). To use other types, such as int, you must specify an equivalent wrapper class: `Integer`. For other primitive types, use: `Boolean` for boolean, `Character` for char, `Double` for double, etc:

Create an `ArrayList` to store numbers (add elements of type `Integer`):

```
import java.util.ArrayList;


public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> myNumbers = new ArrayList<Integer>();
    myNumbers.add(10);
    myNumbers.add(15);
    myNumbers.add(20);
    myNumbers.add(25);
    for (int i : myNumbers) {
      System.out.println(i);
    }
  }
}
```

Sort an ArrayList

Another useful class in the `java.util` package is the `Collections` class, which include the `sort()` method for sorting lists alphabetically or numerically: import java.util.ArrayList;

```
import java.util.Collections;
```

```
public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");

    Collections.sort(cars);

    for (String i : cars) {
      System.out.println(i);
    }
  }
}
```

Example

Sort an ArrayList of Integers:

```
import java.util.ArrayList;
import java.util.Collections;

public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> myNumbers = new ArrayList<Integer>();
    myNumbers.add(33);
    myNumbers.add(15);
    myNumbers.add(20);
    myNumbers.add(34);
    myNumbers.add(8);
    myNumbers.add(12);
```

```
    Collections.sort(myNumbers);

    for (int i : myNumbers) {
      System.out.println(i);
    }
  }
}
```

Example

Sort an ArrayList of Integers in reverse:

```
import java.util.ArrayList;
import java.util.Collections;

public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> myNumbers = new ArrayList<Integer>();
    myNumbers.add(33);
    myNumbers.add(15);
    myNumbers.add(20);
    myNumbers.add(34);
    myNumbers.add(8);
    myNumbers.add(12);

Collections.sort(myNumbers, Collections.reverseOrder());
   // Collections.reverseOrder(myNumbers);

    for (int i : myNumbers) {
      System.out.println(i);
    }
  }
```

```
}
```