Java Arrays

Intro to Java Arrays

- So far we have been working with variables that hold only one value.
- For example:
  - String myName;
  - myName = Bart;
  - Int myNumber;
  - myNumber = 12;
- There are times when we want a single variable to hold more than value. In this case we can use **An Array.**
- An Array is a container object that holds a fixed number of values of a single type.

Intro to Java Arrays

- An Array is like a list of items.
- The length of an array is established when the array is created.
- Think of an array as the columns in a spreadsheet.
- You can have a spreadsheet with only one column, or lots of columns. The data held in a single-list array might look like this:

	Array_Values
0	10
1	14
2	36
3	27
4	43
5	18

Arrays

- Like a spreadsheet, arrays have a position number for each row.
- The positions in an array **start at 0** and go up sequentially.
- Each position in the array can then hold a value.

	Array_Values
0	10
1	14
2	36
3	27
4	43
5	18

- In the image above Array position 0 is holding a value of 10, array position 1 is holding a value of 14, position 2 has a value of 36, and so on.

Arrays

- To set up an **Array**, you have to tell Java what kind of data is going into your array (integers, strings, boolean values, etc).
- You then need to say how many positions the array has.
- You set up an **Array** like this (this would be an Integer Array):
- **int[ ] arrayNumbers;**
- The only difference between setting up a normal integer variable and an array is a pair of square brackets **[ ]** after the data type.
- The name of the array above is **arrayNumbers**.
- Just like normal variables, you can call them almost anything you like

Arrays

- **int[ ] arrayNumbers;** Tells Java that you want to set up an integer array. It doesn't say how many positions the array should hold.
- To do that, you have to set up a new array object:
- You start with your array name, followed by the equals sign.
- After the equals sign, you need the Java keyword new, and then your data type again.
- **EX: int[ ] arrayNumbers ; arrayNumbers = new int [6];** (two lines)
- OR **int[ ] arrayNumbers = new int[6];** (one line)
- After the data type come a pair of square brackets.
- In between the square brackets you need the size of the array. The size is how many positions the array should hold. (**in this case 6 positions**)

Arrays

- **arrayNumbers = new int [6]; OR int[ ] arrayNumbers = new int[6];**
- We are telling Java to set up an array with 6 positions in it.
- To assign values to the various positions in an array, you do it in the normal way:
- **arrayNumbers[0]** = 10;
- Here, a value of 10 is being assigned to position 0 in the array called arrayNums.
- To assign a value of 14 to array position 1, the code would be this:
- **arrayNumbers[1]** = 14;
- And to assign a value of 36 to array position 2, the code is:
- **arrayNumbers[2]** = 36;

Arrays

- If you know what values are going to be in the array, you can set them up like this instead:
- **int[ ] arrayNumbers = { 5, 13, 26, 45 };**
- This method of setting up an array uses curly brackets after the equals sign.
- To set up strings, you can use this:
- **String[ ] arrayStrings = {"Autumn", "Spring", "Summer", "Winter" };**
- To Print an Array: System.out.println(**arrayNumbers[2]**);

Java Array List

Java Arrays Review

- **Which is correct?**
  - **arrayNumber{0} = 10;**
  - **arrayNumber[1] = 15;**
  - **arrayNumber(2) = 20;**
  - **arrayNumber<3> = 25;**
- **Which is correct?**
- **String[ ] arrayStrings = {‘Autumn’, ‘Spring’, ‘Summer’, ‘Winter’ };**
- **String[ ] arrayStrings = {Autumn, Spring, Summer, Winter };**
- **String[ ] arrayStrings = {"Autumn", "Spring", "Summer", "Winter" };**

Java Arrays Review

- Arrays are objects that are used to hold more than one value at a time.
- Arrays allow us to create positions and assign values to those positions
- `arrayNums[0] = 10;`
- `int[ ] arrayNums = { 1, 2, 3, 4 };`
- `String[ ] arrayStrings = {"Autumn", "Spring", "Summer", "Winter" };`

Java Array Lists

- Standard Java arrays are of a fixed length.
- After arrays are created, they cannot grow or shrink
- This means that we must know in advance how many elements an array will hold.
- Array lists supports dynamic arrays that can grow as needed.
- Array lists are created with an original size.
- When the size is exceeded, the array is automatically enlarged.
- When objects are removed, the array may be shrunk.

Java Array Lists

- To set up an ArrayList, we first have to import the package from the **java.util library**:
  - **import java.util.ArrayList;**

- We can then create a new ArrayList object as follow:
  - **ArrayList myList = new ArrayList( );**
- Pop question. What's missing from the Array? (hint: It's a symbol)
- Once we have a new ArrayList objects, we can add elements to it with the **add method**:
- **myList.add("Milk");**
- **myList.add("Butter");**
- **myList.add("third item");**
- **myList.add("fourth item");**
- **myList.add(7); (our list can also include integers)**

Java Array Lists

- Items in the list can be referenced (**printed**) by an Index number using the **get method**:
- **myList.get(3)**
- Pop question. What's the index position for this Array?
- This line will get the item at Index position 3 on the list.
- Remember, index numbers start counting at zero, so this will be which item?

1. **myList.add("Milk");**
2. **myList.add("Butter");**
3. **myList.add("third item");**
4. **myList.add("fourth item");**
5. **myList.add(7); (our list can also include integers)**

Java Array Lists

- You can also remove items from an ArrayList.
- You can either use the Index number: **myList.remove(2);**
- Or you can use the value on the list: **myList.remove(“Butter");**
- Removing an item will resize the ArrayList
- When using an Index number, we have to be careful when trying to get an item on the new resized list.
- If we've removed item number 2, then our list above will contain only 4 items.
- Trying to get the item with Index number 5 would result in an error. (**Why?**)

Java Array List Code Example

```
import java.util.ArrayList;
public class ArrayListExample {
    public static void main(String[] args){
        ArrayList listTest = new ArrayList( );
        listTest.add( "first item" );
        listTest.add( "second item" );
        listTest.add( "third item" );
        listTest.add( 7 );
        listTest.add("fifth item");

        System.out.println( "Whole list items = " + listTest );
        System.out.println( "Index 0 from the list = " + listTest.get(0));
        System.out.println( "Index 4 from the list = " + listTest.get(3));
        }
```

```
aryNumbers[0][0] = 10;      aryNumbers[1][0] = 20;
aryNumbers[0][1] = 12;      aryNumbers[1][1] = 45;
aryNumbers[0][2] = 43;      aryNumbers[1][2] = 56;
aryNumbers[0][3] = 11;      aryNumbers[1][3] = 1;
aryNumbers[0][4] = 22;      aryNumbers[1][4] = 33;

aryNumbers[2][0] = 30;      aryNumbers[3][0] = 40;
aryNumbers[2][1] = 67;      aryNumbers[3][1] = 12;
aryNumbers[2][2] = 32;      aryNumbers[3][2] = 87;
aryNumbers[2][3] = 14;      aryNumbers[3][3] = 14;
aryNumbers[2][4] = 44;      aryNumbers[3][4] = 55;
```

Java Multi-Dimensional Array

Java Multi-Dimensional Array

- The arrays we have used so far have only held one column of data.
- However, the great things about arrays is that you can set up an array to hold more than one column.
- These are called multi-dimensional arrays and can hold several values.
  - As an example, if you have a spreadsheet with 6 rows and 5 columns; then that spreadsheet can hold 30 numbers. It might look like this:

	Array_Values
0	10
1	14
2	36
3	27
4	43
5	18

	A	B	C	D	E
0	10	12	43	11	22
1	20	45	56	1	33
2	30	67	32	14	44
3	40	12	87	14	55
4	50	86	66	13	66
5	60	53	44	12	11

Java Multi-Dimensional Array

- You set up a multi-dimensional array in the same way as a normal array, except you have two sets of square brackets.
- **int[ ][ ] arrayNumber = new int[6][5];**
- The first set of square brackets is for the rows
- The second set of square brackets is for the columns.
- In the above line of code, we're telling Java to set up an array with 6 rows and 5 columns.
- To hold values in a multi-dimensional array you have to be careful to track the rows and columns.

Java Multi-Dimensional Array

- Here's some code to fill the first rows of numbers from our spreadsheet image:
- **arrayNumber[0][0] = 10;**
  arrayNumber[0][1] = 12;
  arrayNumber[0][2] = 43;
  arrayNumber[0][3] = 11;
  arrayNumber[0][4] = 22;
- So the first row is row 0. The columns then go from 0 to 4, which is 5 items. To fill the second row, it would be this:
- **arrayNumber[1][0] = 20;**
  arrayNumber[1][1] = 45;
  arrayNumber[1][2] = 56;
  arrayNumber[1][3] = 1;
  arrayNumber[1][4] = 33;
- The column numbers are the same, but the row numbers are now all 1.

Java Multi-Dimensional Array

```
int[][] arrayNumber =
    { {99, 42, 74, 83, 100},
      {90, 91, 72, 88, 95},
      {88, 61, 74, 89, 96},
      {61, 89, 82, 98, 93},
      {93, 73, 75, 78, 99},
      {50, 65, 92, 87, 94},
      {43, 98, 78, 56, 99} };
```

Java Multi-Dimensional Array

- To access all the items in a multi-dimensional array we can use one loop inside of another. Here's some code to access all our number from above.

```
public static void main(String[] args) {

    int[][] aryNumbers = new int[6][5];

    aryNumbers[0][0] = 10;      aryNumbers[1][0] = 20;
    aryNumbers[0][1] = 12;      aryNumbers[1][1] = 45;
    aryNumbers[0][2] = 43;      aryNumbers[1][2] = 56;
    aryNumbers[0][3] = 11;      aryNumbers[1][3] = 1;
    aryNumbers[0][4] = 22;      aryNumbers[1][4] = 33;

    aryNumbers[2][0] = 30;      aryNumbers[3][0] = 40;
    aryNumbers[2][1] = 67;      aryNumbers[3][1] = 12;
    aryNumbers[2][2] = 32;      aryNumbers[3][2] = 87;
    aryNumbers[2][3] = 14;      aryNumbers[3][3] = 14;
    aryNumbers[2][4] = 44;      aryNumbers[3][4] = 55;
```

```
    aryNumbers[4][0] = 50;      aryNumbers[5][0] = 60;
    aryNumbers[4][1] = 86;      aryNumbers[5][1] = 53;
    aryNumbers[4][2] = 66;      aryNumbers[5][2] = 44;
    aryNumbers[4][3] = 13;      aryNumbers[5][3] = 12;
    aryNumbers[4][4] = 66;      aryNumbers[5][4] = 11;

    int rows = 6;
    int columns = 5;
    int i, j;

    for ( i = 0; i < rows; i++) {
        for ( j = 0; j < columns; j++) {
            System.out.print(aryNumbers[i][j] + " ");
        }
        System.out.println( "" );
    }
}
```

- The first for loop is used for the rows; the second for loop is for the columns.
- The two loop system is used to go through all the values in a multi-dimensional array, row by row.

```
int[][] arrayNumber;
arrayNumber = new int [6][5];

arrayNumber[0][0] = 10;
arrayNumber[0][1] = 12;
arrayNumber[0][2] = 43;
arrayNumber[0][3] = 11;
arrayNumber[0][4] = 22;

arrayNumber[1][0] = 20;
arrayNumber[1][1] = 45;
arrayNumber[1][2] = 56;
arrayNumber[1][3] = 1;
arrayNumber[1][4] = 33;

arrayNumber[2][0] = 30;
arrayNumber[2][1] = 67;
arrayNumber[2][2] = 32;
arrayNumber[2][3] = 14;
arrayNumber[2][4] = 44;

arrayNumber[3][0] = 40;
arrayNumber[3][1] = 12;
arrayNumber[3][2] = 87;
arrayNumber[3][3] = 14;
arrayNumber[3][4] = 55;

arrayNumber[4][0] = 50;
arrayNumber[4][1] = 86;
arrayNumber[4][2] = 66;
arrayNumber[4][3] = 13;
arrayNumber[4][4] = 66;

arrayNumber[5][0] = 60;
arrayNumber[5][1] = 53;
arrayNumber[5][2] = 44;
arrayNumber[5][3] = 12;
arrayNumber[5][4] = 11;
int rows = 6;
int columns = 5;
int i, j;


for (i=0; i < rows ; i++) {for (j=0; j < columns ; j++) {
System.out.print( arrayNumber[ i ][ j ] + " " );
```