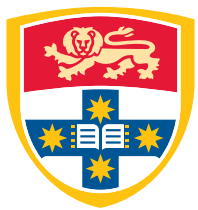
INFO1103: Introduction to Programming

School of Information Technologies, University of Sydney



Week 4: More loops, building software, Arrays

We will cover: Using `continue`, looping with `while`, `break` and `continue`, building software from scratch and introduction to arrays

You should read: §§4.2, 7.1 of Savitch

Lecture 7: Arrays and Iteration

Simple structures for storing data in your programs

Information of the same type

- Write a program that displays the name of 50 students
- Write a program that calculates and displays the total from 100 values
- Write a program that counts how many people in this room have a birthday today

What is the data type for each problem?

If we write these programs we need to use one variable to store each value

Arrays

An array is a *contiguous block of memory* containing multiple values of the same type.

Array solves the problem of having many variables.
e.g. you don't have to declare 100 int variables.

```
1    // bad style bro
2    int value001 = 32;
3    int value002 = 18;
4    int value003 = 4;
5    ...
6    int value100 = 6;
```

Instead of using value046, use the 46th value from the memory of the array

Arrays — ideas about memory

Arrays — creation

Here's how you create some arrays:

```
1    int [] x; // declare an array of integers of unknown length
2    int y = 15;
3    double [] z = new double[y];
4          // declare and initialise an array of doubles
5    String [] myStrings = new String[y];
6          // declare and initialise an array of Strings
```

Line by line,

1: *declares* an integer array variable, but does not initialize it: it's `null` so far.

2: declares and initializes the integer `y` with value 15.

3: declares and initializes the array `z` with `y` `doubles` (all initially 0)

4: declares and initializes the array `myStrings` with an array of `Strings`, all initially `null`.

Arrays — Initialisation

Initialise each value of the array at declaration

```
1  public class InitArrays {
2      public static void main(String [] args) {
3          String names [] = new String [] { "Bill", "Ted", "Larry" };
4          System.out.println(names[0]);
5          System.out.println(names[1]);
6          System.out.println(names[2]);
7      }
8  }
```

which creates an array of String objects, the contents of which are the three strings “Bill”, “Ted” and “Larry”. The length of the array is set *implicitly* to the number of elements in the strings provided in braces.

If you don't initialise your array

Then actually not much bad will happen.

When you create an array like this:

```
1        int [] data = new int[15];
```

then `data` is an array of length 15, and all the *values*, `data[0]`, `data[1]`, ..., `data[14]` are given their *default values*.

The compiler won't alert you that the values in the array are not initialised.

If you don't initialise your array

```
1       boolean [] flags = new boolean[10];
```

Now all the boolean variables get their default value of `false`.

Lastly,

```
1       char [] letters = new char[1024];
```

now `letters[0]`, ..., `letters[1023]` are all the (invisible) character 0.

Arrays — index

The array contains multiple values of the same type

Each element of the array is described by an index number.

The numbering starts at 0, not 1. The first element of the array is the "*the zero-th element*" or "0th element" of the array.

To access the *i*th element of an array `A` we just use `A[i]`.

If `i` is less than 0 then Java will throw an exception [1]

If `i` is bigger than the array size initialise, Java will throw an exception

[1] Write a program to test this behaviour: see what happens.

Using the array

```
1  public class SumArray
2  {
3      public static void main(String[] args) {
4          int [] values;
5          values = new int[4];
6
7          values[0] = 1;
8          values[1] = 2;
9          values[2] = 3;
10         values[3] = 4;
11
12         System.out.println("first value: " + values[0]);
13         System.out.println("last value: " + values[3]);
14
15         int sum = values[0] + values[1] + values[2] + values[3];
16         System.out.println("sum: " + sum);
17     }
18 }
```

Size of the Java array

There is a very nice feature of the Java array that is built-in: for any array x that you define, the value x.`length` is the length of the array.

That means if you declare an array of `char` items like this:

```
1    char myChars[] = new char[3];
```

the length of `myChars` is 3.

Accessing the last element

```
1    char myChars[] = new char[3];
2    myChars[0] = 'A';
3    myChars[1] = 'B';
4    myChars[myChars.length - 1]  = 'C';
5    System.out.println("last value: " + myChars[myChars.length - 1] )
```

ArrayIndexOutOfBoundsException

When you attempt to run a program that assumes you have some `String` arguments, but don't give it any, you will get an `ArrayIndexOutOfBoundsException`:

```
1 public class MissingArgs {
2     public static void main(String[] args) {
3         System.out.println("The third argument is " + args[2]);
4     }
5 }
```

```
~> javac MissingArgs.java
~> java MissingArgs
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2
    at MissingArgs.main(MissingArgs.java:3)
~>
```

See? Look back at the code: on line 3 the index assumed to work is 2 (`args[2]`) but the array `args` has *no elements in it*. So its length is 0 and there isn't an `args[2]`.

Arrays of Strings

You can have arrays of *anything*: the primitive types you've used for a while now, and of *reference types*, like `String`:

```
1        String [] words  = new String[1000];
```

The default value of a reference type is `null`, so all of the `String` objects in `words` are also `null`. You can't use them yet and trying to do so will usually cause an Exception.

ArrayOfStrings.java

```
1  public class ArrayOfStrings {
2      public static void main(String[] args) {
3          int i = 0;
4          while (i < args.length) {
5              String argument = args[i];
6              System.out.println(argument);
7              i = i + 1;
8          }
9      }
10 }
```

Running this gives the following behaviour:

```
~> java ArrayOfStrings one two
one
two
```

When to use Arrays (yes/no)

a todo list of tasks

stock market price of one company over a period of time

number of stairs at the Opera House

number of passengers in each of 8 train carriages

100 different variables used in automobile software e.g. fuel, passenger light on, odometer etc.

a catalogue of 100 book titles

a 3,000 word essay