2020 CI401
Introduction to programming

Week 1.05
Types, assignment and arrays

Dr Roger Evans
Module leader

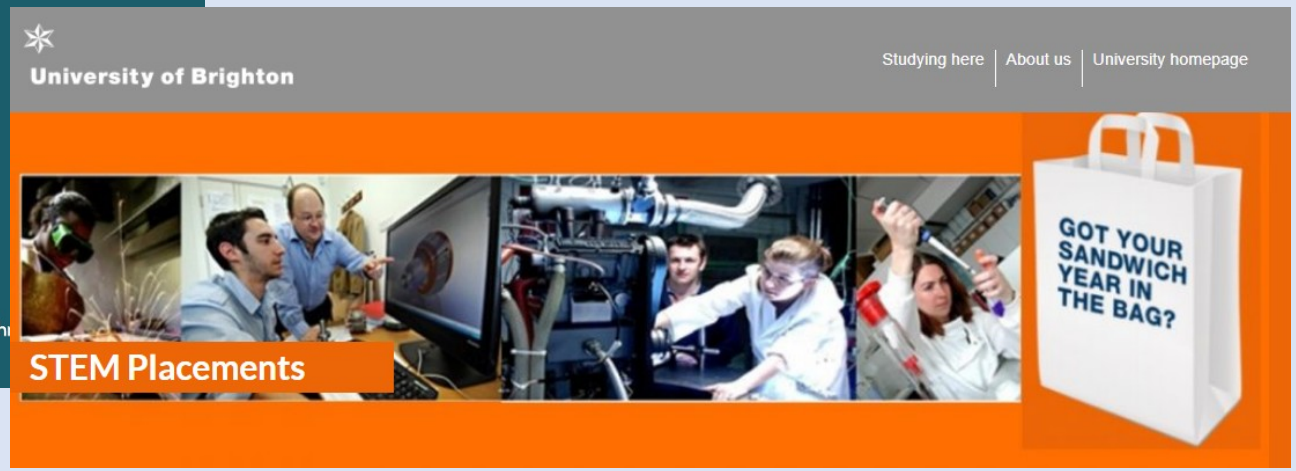
3rd November 2020

Lecture recording

- This lecture will be recorded and published in the module area
- The focus of recording is on the lecturer, not the audience
- If you are particularly concerned not to be part of the recording, turn off your microphone and camera.

- (This slide is really a reminder to me to turn recording on!)

Brighton Works Week



- https://www.brighton.ac.uk/careers/brighton-works-week.aspx
- https://blogs.brighton.ac.uk/sciengplacements/

Module structure (version 2)

Semester 1

Week	Topic	Theme
1.01	Introduction / Hello World	Coding
1.02	Variables, loops and choices	Coding
1.03	Input, more loops and choices	Coding
1.04	Variables and expressions	Coding
1.05	Types, assignment and arrays	Data
1.06	Let's play Top Trumps!	Data
1.07	Objects and classes	OO
1.08	Working with numbers	Data
1.09	Simple Algorithms	Dvp
1.10	Introduction to JavaFX	Dvp
1.11	Simple Animation	Dvp
	Xmas vacation 21 Dec – 8 Jan	
1.12	GUIs using MVC	OO
1.13		

Semester 2

Week	Topic	Theme	Project
2.01	Project topics and assessment	Project	Set
2.02	Simple Inheritance	OO	Lab
2.03	Scope, Visibility and Encapsulation	OO	Lab
2.04	Testing - JUnit	Testing	Lab
2.05	Documentation - Javadoc	Doc	Study
2.06	Collections and generic types	Data	Study
2.07	IO: files and streams	Dvp	Study
	Easter Vacation 29 Mar – 16 Apr		
2.08	Numbers – the computer's view	Data	Study
2.09	Java vs Python		Submit?
2.10	More algorithms – search and sort	Dvp	
2.11	How fast is my code?	Dvp	
2.12	Java 'under the hood'		
2.13	Revision week		Exam ↓

Assessment

Semester 1

- No formal assessment
- Tests and revision of semester 1 material

Semester 2

- Project (50%)
  - Published 8th February, hand-in 30th April (to be confirmed)
  - Coding project alongside lectures during February and March
  - Choice of projects and content
- Exam (50%)
  - During summer exam period (31st May -11th June)
  - Online, open-book
  - Mock exam will be provided

The story so far …

- Variables
  - Names for values in your code
  - Inputs or parameters
  - Used for calculation
  - Can update themselves
- Variable names
  - Start with a letter
  - Can include digits, $ and _
  - Meaningful – camelCase
  - Short – x,y,z (loop variables and indexes)
  - Constants – CAPS_WITH_UNDERSCORES
- Expressions
  - Literals – actual values 25, “hello”
  - Variables – current stored value
  - Complex – ‘doing sums’, 3*2, x+1
- Expression return a result
  - To store in a variable
  - To use in a bigger expression
  - To use to control a program (if and for statements)
- Boolean expressions
  - True/false tests using logic
  - Comparison operators <, >, <=, >=, ==
  - Boolean operators - &&, ||, !

Types

Java types

- We have mentioned from time to time that things in Java are of different types
- This is not just a casual usage (as in different kinds, or different sorts) – types are a specific and important part of Java
- Every piece of data that you use in a Java program has a type
- Every expression has a type, and every variable has a type
- And Java expects us to keep track of types and use them carefully and tells us off if we get it wrong (Java is called a ‘strongly-typed’ language)

Types we have seen already

- We have seen a few types in our examples already:
  - String – the type of string objects
  - String[] – the type of string arrays (lists of strings)
  - int – the type of whole numbers (integers, in maths)
  - boolean – the type of the values true and false, used in tests
  - Scanner – the type of a library to help us get user input

Types in Java

There are four kinds of type in Java:

- Primitive types – these are types for different sorts of numbers, and for booleans, which are the building blocks of larger data objects
- Classes – these are types associated with class definitions which define their state (what data they hold) and behaviour (what you can do with them), and are the basis for programming in Java. The main examples we have seen so far are the String and Scanner classes. Each of our example programs is also a type (eg QueenBot, EstateAgent).
- Arrays – these are types which represent lists of objects of another type, as we have already seen in our examples with String[].
- Generic types – these are types which are built up out of other types (like arrays, but more general). We will talk more about them later in the course.

Primitive types

- Java has eight primitive types:
  - `byte` – 8 bit integer
  - `short` – 16 bit integer
  - `int` – 32 bit integer
  - `long` – 64 bit integer
  - `float` – 32 bit floating point (decimal)
  - `double` – 64 bit floating point (decimal)
  - `boolean` – true/false
  - `char` – 16 bit Unicode character

- Primitive types mostly support numbers at different precisions
- They distinguish between integers (whole numbers) and floating point (decimals)
- For now, we will mostly be concerned with int and double numbers.
- boolean values and characters (char) are also primitive.

Class types

- A class type is created when a class is defined in a program.
- There is a large library of pre-defined classes (such as String and Scanner).
- Class type names conventionally start with a capital letter, to distinguish them from variables (notice that primitive types do not)
- Classes are used to define and create objects of a particular type
- We will learn more about how to make objects using classes soon – the only example we have seen so far is making a Scanner object with:

```
myInput = new Scanner(System.in);
```

Array types

- Arrays make lists of other things (for example a list of Strings or ints)
- An array has its own type depending on what kind of thing it is a list of
- So while String is a single string object, String[] is a list (array) of string objects
- And int[] is an array of ints
- You can always tell when you have an array type because it uses […] to identify it and to access data inside it.
- We will talk more about arrays below.

Generic types

- The last kind of type that you see in Java are generic types
- We will talk about these in more detail later in the module
- But we will mention one example now, and you may see others
- ArrayList<String> is a generic type. It's a bit like an Array only more useful, and we will introduce it in detail soon.
- Just as you can identify arrays because they have square brackets ([..]), you can identify a generic type because it has 'pointy brackets' (< ... >)
- Between the pointy brackets you write one or more other types

Declaring and assigning to variables

Variables and types

- In Java, a variable can only contain values of one type
- Whenever you want to start using a new variable, you have to tell Java its name, and what type of value it can hold
- We call this declaring the variable
- And once a variable has a type, you can only store values of that type in the variable

Variable declarations

- In its simplest form a variable declaration looks like this:

```
type variable;
```

- For example we can say

```
String home;
int x;
Scanner myInput;
```

Variable declarations with initialisation

- We can also give our variable a value in the same statement:

```
type variable = expression;
```

- `expression` is an expression which calculates a value of the specified type.

```
String home = "Brighton";
Scanner myScanner = new Scanner(System.in);
```

- This is called initialising the variable and is generally good practice, to make sure that your variables have an sensible initial value
- Variables always have values, so if you don't initialise one, it will set to a default value (0 for integer types, 0.0 for floating point, null for class types).

Variable assignment statements

- After a variable has been declared (and maybe initialised), you may want to change its value at some point in your program. You do this with an assignment statement:

```
variable = expression;
```

- Again, the expression must calculates a value of the type specified in the variable declaration.

Notes on assignment statements

- An assignment statement looks like an initialisation statement, but without the type specifier. Don't mix them up, however, because you may only declare a variable once.
- A common editing error is to copy a declaration statement and paste it in where you want an assignment. This results in an error because you are re-declaring your variable.

```
String room = "hall";        // declaration with initial value
…
String room = myInput.next();  //  second declaration!!!
// Java will complain because it thinks room is being declared again
```

- What you need is just

```
room = myInput.next();          // assignment
```

Notes on assignment statements

- Assignment statements also look like equations, but they are not. They are instructions to (a) evaluate the `expression` and (b) assign its value to the `variable`. This is important because the expression may (often) include a reference to the variable itself, for example:

```
n = n+1;
```

- As an equation, this is nonsense. As an instruction it says:
  1. Take the (current) value of `n`
  2. Add one to it
  3. Save the result as the (new) value of `n`

Summary

- Variables let us save values and give them names, which allows us to use them in more than one place in a program
- Expressions let us write down/calculate values of different sorts (strings, numbers, booleans etc.)
- Variables, expressions and values have types
- In Java, whenever you make want to use a new variable, you have to tell Java what type it is, we call this declaring the variable
- And once a variable has a type, you can only store values of that type in the variable (approximately)
- This means that you can only use an expression of that type in an assignment statement for the variable

Arrays

Arrays – the story so far

- Our first array example looked like this:

```
String[] places = {"London", "Brighton", "Paris"};
```

- And we used them it for loops like this:

```
for (String place: places) {
    System.out.println(place);
}
```

- This is quite a limited way to use arrays, so let's look at some more things that we can do.

Declaring array variables

- A basic array variable declaration looks like this:

```
TypeName[] variableName;
```

- There's a second form like this:

```
TypeName variableName[];
```

- Notice the square brackets are in a different place
- You can use either, but it's the square brackets that tell you it's an array

Creating arrays

- The array declaration/initialisation we saw before looks like this:

```
String[] places = {"a", "b", "c"};
```

- This is actually an abbreviation for this:

```
String[] places = new String[] {"a", "b", "c"};
```

- Notice this uses the new keyword, just like the Scanner example. new is the standard way in Java to create new data objects.

- If you don't know what the array contents should be, you can just specify the array's size when you initialise it. You put the size of the array inside the square brackets:

```
String[] places = new String[3];
```

- This creates an array with three elements, each set to null (or 0 for number arrays).

Array indexing

- The only place we have used arrays so far is in for loops
- But we can also access each member (or element) of an array individually
- Elements of an array are numbered. The number of an element is called its index (plural indices), and the numbering goes from 0 (not 1!) up to the length of the array minus 1
- We access an element by putting its index in the square brackets – for example myArray[0] is the element with index zero (which we might naturally call the first element) in myArray
- We can use an indexing expression like this like a variable – we can access its value, or we can change its value by assigning to it

Array indexing example

- Declare an array:

  Indices: 0 1 2

```
String[] places = {"London", "Brighton", "Paris"};
```

- Print out the element with index 2 (which is Paris):

```
System.out.println(places[2]);
```

- Or take an element of an array and store it in a variable:

```
String myTown = places[1];
```

- Change the first element to Manchester:

```
places[0] = "Manchester";
```

- To find out how big your array is, use .length:

```
int i = places.length;
```

More examples

- Accessing the first element of an array:

```
myArray[0]
```

- Accessing the last element of an array:

```
myArray[myArray.length - 1]
```

- Swapping elements in an array – this doesn't work (why not?):

```
myArray[1] = myArray[2];
myArray[2] = myArray[1];
```

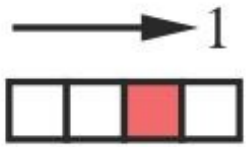
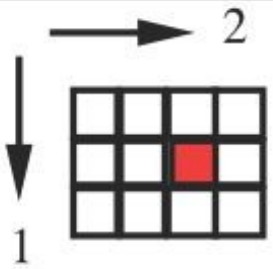
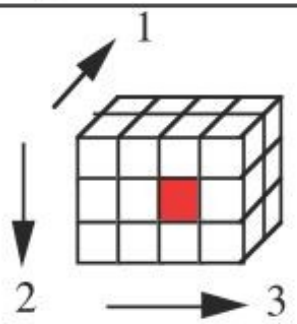
- You need an extra variable to store one of the values temporarily:

```
temp = myArray[1];
myArray[1] = myArray[2];
myArray[2] = temp;
```

Multi-dimensional arrays

- Arrays let you store a list of objects in a line
- Often in programming you want a 2 dimensional list (a table or a matrix), or a three dimensional list (a cube) or more
- In Java, we do this just making an array of arrays, or an array of arrays of arrays ...

Multi-dimensional arrays

Declaration of array handle	Conceptual model of storage	Allocation of physical storage for array an access to shaded element
`int vector[];`		`vector = new int[4];` `int v = vector[2];`
`int table[][];`		`table = new int[3][4];` `int v = table[1][2];`
`int cube[][][];`		`cube = new int[2][3][4];` `int v = cube[0][1][2];`

Lab exercises
Week 1.05

Lab exercises 1.05

- Lab1 – Some simple coding tasks using an array
- Lab2 – A task using a 2 dimensional array (a matrix)
- Lab3 – Challenge lab – write code to do matrix multiplication