Fundamentals 2

Recap Session

- IntelliJ (debugging, step through IDE)
- Git
- GitHub

> USE RANDOMISER TO PICK SOMEONE TO ANSWER THESE QUESTIONS:
>
> - Where in IntelliJ would you reconfigure/configure the JDK version?
> - What are the steps to create a watch expression.
> - How do you make a conditional breakpoint?
> - In Git, how do to save changes and make them available to your fellow colleagues? Which commands are used
> - Is Git distributed or centralised?
> - If I wanted to copy a repository what command would I use?
> - What if I wanted to contribute to a project? What command would I use.
> - You are asked to work on a project. What are the steps you would take when setting up your environment/workspace based on what you learnt yesterday?

Make Up of a Java application

Packages

Filing cabinet analogy. Java uses a similar concept to a filing cabinet. Code is organised into packages/folders to contain related code. You can think of these as **LOGICAL GROUPS**.

Structure of Main/Test folders

- Explain how programs are organised

Main Method

- Describe the main method

Variable

What is a variable?

A variable is a name for a piece of memory that stores data.

When you declare a variable, you need to state the variable type along with giving it a name.

Declaring and Initialising a Variable

In Java, you declare a variable by simply using the following structure:

```
<data type> <variable_name>
```

For example:

```java
int numberOfAnimals;
```

This is declaring a variable *numberOfAnimals* of type integer.

Now that we've declared a variable, we can give it a value.

```java
int numberOfAnimals = 100;
```

Now you might want to at some point, declare and initialise multiple variables in one go? Well you can.

```java
int i1, i2, i3;

// what do you think would be initialised here?
int i1, i2, i3 = 0;

// only i3
```

One thing to note though is that you cannot declare variables and initialise them on the same line if they are not the same data types:

```
// this will not work:
int number, String name;
```

Other things to note when declaring variables:

```
boolean b1, b2;  // works
String s1 = "1", s2; // works - anyone want to guess why?
double d1, double d2; // won't work - anyone want to guess why?
int i1; int i2; // works
int i3; i4; // won't work
```

Data types

We all know that everything around us has some context and we can relate those objects to a particular type.

For example, age, is known as a number. Date of birth as a date etc.

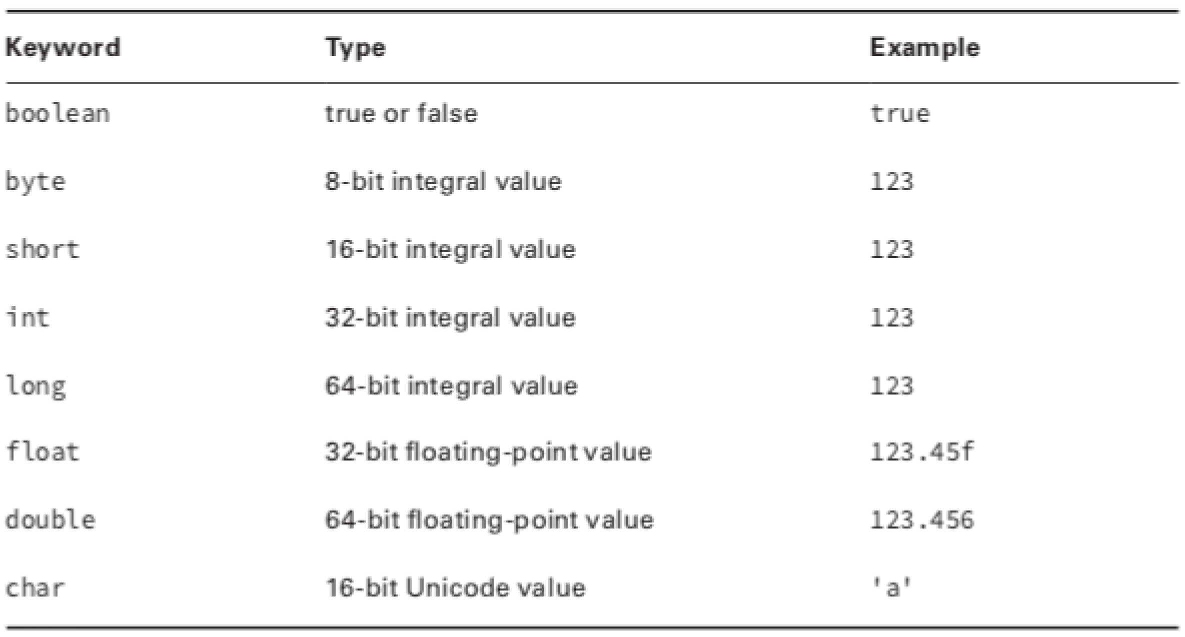
Java is a strongly typed language meaning that each type is predefined in the language and that all variables/constants defined must be of these types.

In Java there are two forms of data types.

Primitive Data Types

The first type is known as a Primitive Data Type. This type has numerous types which we'll cover in this session.

In Java there are 8 primitive data types. These are:



Keyword	Type	Example
boolean	true or false	true
byte	8-bit integral value	123
short	16-bit integral value	123
int	32-bit integral value	123
long	64-bit integral value	123
float	32-bit floating-point value	123.45f
double	64-bit floating-point value	123.456
char	16-bit Unicode value	'a'

Let's go through examples using them:

boolean

This data type only has two possible outcomes. True or False.

```
boolean userRegistered = true;
boolean requiresValidation = true;
boolean isRainingToday = false;
```

char

2 bytes in size (16 bits). Range is 0 to 65,536 (unsigned).

```
char genderMale = 'M';
char genderFemale = 'F';
```

int

4 bytes in size (32 bits). Range is -2,147,483,648 to 2,147,483,647.

```
int age = 23;
int classroomSize = 36;
int year = 2009;
```

short

2 bytes in size. -32,768 to 32,767.

```
short age = 23;
short numberOfMonths = 12;
```

long

8 bytes in size. Range is -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

```
long epochTime = 1563201186L;
long num = -2333L;
```

In Java an 'L' is usually appended so that the compiler knows we are using a Long.

float

4 bytes in size.

```
float degrees = 40.5f;
```

In Java you generally put a 'f' on the end to allow the compiler to know we're using a float. You'll see why in a moment.

double

Doubles are more precise than a float. So prefer to use doubles over floats.

```
double interestRate = 0.3d;
double chancesOfWinning = 0.5d; // 50%;
```

In Java you generally put a 'd' on the end so the compiler can distinguish between a float and double.

Use **RANDOMISER** to choose someone:

Data Type Lottery:

- Number of items in a Shopping Basket
- Anniversary

- Birthday
- Amount of rainfall
- Mileage
- Percentage (double or integer) - depends on requirements
- Distance between two countries in miles.

Declaring Data Types

You can use underscores as of Java 7 which can help when displaying, say large numbers.

For example:

```
int million = 1000000;
int million2 = 1_000_000; // this is valid
```

You can add underscores anywhere EXCEPT

- at the beginning of a literal
- the end of a literal
- right before a decimal point
- right after a decimal point

For example, these are all invalid:

```
double notAtStart = _1000.00; // invalid
double notAtEnd = 1000.00_; // invalid
double notByDecimal = 1000_.00; // invalid

// This one is valid
double annoyingButLegal = 1_00_0.0_0; // this one compiles
```

Non Primitive Data Types

String

```
String firstName = "John";
String placeOfBirth = "Manchester";
```

We can also mix and match others with a String:

```
int numberOfBrothers = 3;
String message = "I have " + numberOfBrothers + " bro's";
```

As soon as you use the double quote you end up turning it into a String.

Data Types/ Variables and Best Practices

Best Practices

It's common sense but you should choose the appropriate date type for the target object you're are trying to store a value for.

Choosing the wrong data type can mean a whole host of extra code being written to just "work around" the data type rather than work with it.

Variables

We all know what a variable is? Don't we. If not explain.

In this example we define a variable by giving it an identifier (basically a name, identifier is the technical term used):

```
int numberOfEmployees = 100;
```

In Java an identified is a sequence of characters of unlimited length.

You can declare a variable with any format such as:

num1 // Can use a-z and 0-9 and start with a letter knsad // Only letters _abc // Can start with an underscore _ // Can have only one letter, which is an underscore sum_of_two_numbers // Can have letters and underscores Outer$Inner // Can have a-z, A-Z and $ $var // Can also start with $

However, all of these are not what you would call best practice. We'll come across better practices later.

We do however when create a variable and choosing a name, ensure that it starts with a lower case letter (i.e. be camelcase). Examples of camelcased variables are:

```
String firstName;
int classSize;
```

Apart from the first "word", every other word starts with a capital.

In Java there are several keywords we cannot use and are therefore reserved by the language itself. These are:

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

When you declare a variable a memory allocation is made for that variable to hold the data that is represented by it's data type. The memory location can be referred to by the identifier, name of the variable.

In Java there are 3 type of variables. We'll cover each of them with examples and an explanation of where we use them:

- Local variables
- Instance variables

- Class/Static variables

Local Variables

- These are declared in methods, constructors or blocks of code.
- They must be initialised before use.
- They are created within these areas and are destroyed by the compiler/garbage collector once you leave that statement, method etc.
- The thing with local variable is that they are not visible anywhere else other than within the method, constructor or block of code in which they are declared. I'll cover this soon with examples.

IN NOTEPAD:

```
public static void main(String [] args) {
  int y = 10;
  int x;

  int result = x + y;
  System.out.println("Result is: " + result);
}
```

What do you think this code does? PICK SOMEONE TO ANSWER. It won't compile.

So Let's look at local variables ...

Lets go through an example:

```
public class Variables {
  public static void main(String [] args) {
    int numberOfGoals = 60; // local variable and accessed inside

    System.out.println("Ronaldo scored " + numberOfGoals + "goals!!!");
  }
}
```

Run this and see what we get.

Instance Variables

- These are declared inside a Class, which we will visit further into the Bootcamp. They are different to local variables in that they are no declared inside a method, constructor or block of code.
- When space is allocated for objects on the memory heap, one is allocated for an instance variable of the size of it's data type. So if we declared a String, then space would be allocated for that amount of bytes.

- Instance variables can be accessed by any method, constructor or block of code in that class. There is however a much safer way to do this which we will cover in later sessions in the Bootcamp. I'm sure some of you may already know how.

Lets go through an example:

```
public class Student {
  // this is an instance variable
  public String studentName;

  public Student(String name) {
    studentName = name;
  }

  public void printDetails() {
    // Can access the instance variable here:
    System.out.println("Student name is " + name);
  }

  public static void main(String [] args) {
    Student student = new Student("John Doe");
    student.printDetails();
  }
}
```

Run the example code and look at the output. Feel free to debug it.

Class/Static Variables

- Static variables are also known as Class variables are declared using the **static** keyword in a class but outside of a method, constructor or block.
- When you declare a static variable there will only be one copy of that variable per class regardless of how many times you make that class. We'll see in an example soon.
- Static variables are created when a program starts and are destroyed when the program stops.
- Visibility of static variables is similar to *instance variables*_ but they most are declared public since they must be available to users of the class.
- You can access a **static** variable by using the Class name in which it is defined, again we'll see in an example.

Lets' see one in action:

```
public class Employee {
  // static variable:
  private static double salary;

  public static final String DEPARTMENT = "Marketing";

  public static void main(String [] args) {
    salary = 1000;
    System.out.println(DEPARTMENT + " average salary: " + salary);
  }
}
```

If we were to use this Employee class elsewhere, you could access the DEPARTMENT variable/constant like this:

```
Employee.DEPARTMENT
```

Demonstrate using Class called Company and call Employee and Employee.DEPARTMENT inside.

Quiz: https://www.flexiquiz.com/SC/N/2ddda8a5-6abe-431d-b2cd-80322b2f333a