**Lesson 1**

OOP: driven by modeling the code around objects

Objects have Fileds, and are capable of performing actions (methods)

Primitive variables – primitive datatypes e.g int,float etc

Objects are enhanced datatypes

**Fields**

The fields of an object are all the data variables that make up that object. They are also sometimes referred to as **attributes** or **member variables**.

These fields are usually made up of primitive types like integers or characters, but they can also be objects themselves.

For example a book object may contain fields like title, author and numberOfPages.

Then a library object may contain a field named books that will store all book objects in an array.

**Accessing fields:**

Accessing a field in an object is done using the dot modifier ‘.’

For example, if we had an object called book that contains these fields:

String title;

String author;

**int** numberOfPages;

To access the title field you would use

book.title

This expression is just like any other string, which means you can either store it in a string variable:

String myBookTitle = book.title;

Or use it directly as a string itself and perform operations like printing it:

System.out.println(book.title);

**Setting Fields**

You can also change a field’s value. Say you want to set the number of pages in a book to 234 pages:

book.numOfPages = 234;

# Methods

You might have also noticed that running actions in objects look very much like calling a function. That’s because that’s exactly what it is.

Methods in Java are functions that belong to a particular object. When we get to creating our own object types later in this lesson we will be creating methods the same way we used to created functions.

### Calling a method

To use a method you call it (just like calling a function). This is also done using the dot modifier .

Methods, just like any function can also take in arguments. For Example: Assume that our book object has a method called setBookmark that takes the page number as a parameter:

**void** **setBookmark**(**int** pageNum);

If you wanted to set a bookmark at page 12, you can call the method and pass in the page number as an argument:

book.setBookmark(12);

## Summary

**Fields** and **Methods** together are what make an object useful, fields store the object's data while methods perform actions to use or modify those data.

However some objects might have no fields and are just made up of a bunch of methods that perform various actions.

Other objects might only have fields that act as a way to organize storing data but not include any methods!

## Next Step

Now that we’ve seen how to use objects and access their fields as well as call their methods, let’s set up your computer so you can start using objects straight away.

**Classes** and **Objects** are two different terms and should not be used interchangeably, they can sometimes seem like they both refer to the same thing but each has a different meaning.

Here's a comparison that illustrates when to use which:

|  |  | **Class** | **Object** |
| --- | --- | --- | --- |
| **What:** |  | A Data Type | A Variable |
| **Where:** |  | Has its own file | Scattered around the project |
| **Why:** |  | Defines the structure | Used to implement to logic |
| **Naming convention:** |  | CamelCase (starts with an upper case) | camelCase (starts with a lower case) |
| **Examples:** |  | Country | australia |
|  |  | Book | lordOfTheRings |
|  |  | Pokemon | pikachu |

In summary, **objects** are to **Classes** what **variables** are to **Data types**.

# Strings

You've probably already noticed that (unlike all primitive types) Strings start with an upper case 'S'! That's because a String is in fact a class and not a primitive type

A String variable is made up of an array of characters (char []) as its field, but being an object means that it also offers some powerful methods like length() that counts and returns the number of characters in that array, and equals(String s) that compares the characters in this string with another string.

# Everything is an object in Java

Because Java is an OOP language, it includes classes that simply wrap around all the primitive types themselves to offer some extra functionality through their methods:

| **Class** | **Primitive type** |
| --- | --- |
| Integer | int |
| Long | long |
| Double | double |
| Character | char |
| String | char[] |

Each of those classes is made up of the corresponding primitive type as its field, but usually also comes with some powerful methods.

It also allows you to forget about primitive types and treat everything in Java as an object. However, it is still recommended to use primitive types when writing a simple piece of code.

**The main method**

A Java program can be as small as a single class, but usually a single program will be made up of tens or even hundreds of classes!

A good Java program is one that divides the logic appropriately so that each class ends up containing everything related to that class, and nothing more!

Classes would be calling each other's methods and updating their fields to make up the logic of the entire program all together!

BUT, where should the program start from exactly? In other words, if a method can call another method and that method can call another, which method will start this sequence the very first time?

The answer is the main method! It looks like this:

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

*// Start my program here*

}

Let's break it down:

* **public**: Means you can run this method from anywhere in your Java program (we will talk more about public and private methods later
* **static**: Means it doesn't need an object to run, which is why the computer starts with this method before even creating any objects (we will also talk more about static methods later on)
* **void**: Means the main method doesn't return anything, it just runs when the program starts, and once it's done the program terminates
* **main**: Is the name of the method
* **String [] args**: Is the input parameter (array of strings) which we will cover how to use it later in this lesson as well!

This main method is the starting point for any Java program, when a computer runs a Java program, it looks for that main method and runs it.

Inside it you can create objects and call methods to run other parts of your code. And then when the main method ends the program terminates.

If this main method doesn't exist, or if there's more than one, the Java program won't be able to run at all!

The main method can belong to any class, or you can create a specific class just for that main method which is what most people do.

Let's have a look at an example next.