**Compiling and Executing Java Code**

If you needs to create an enterprise application, the creation process is more complex but at its core you will find these simple steps.

The first of these steps is to write some java code in a text editor.

Then this code has to be transformed to another format, which can be executed by your computer. This transformation is conducted by a special piece of software called a compiler.

The file produced by a compiler is often called a binary file or an executable file.

This code contains special binary codes called byte code.

Compiling some code may fail; your code has to be correct for the compiler to produce an executable version of it.

*These two steps: compilation and execution require two specific pieces of software that are part of the Java Development Kit, also known as the JDK.*

starting with Java SE 11 you can also merge these two steps into one, by executing a .java file directly. You can use these feature only if you are executing a program that is written in a single file. This way of executing your java application does not work if your java code spans more than one file.

**Creating a First Java Class**

The first step you need to know is that the Java code you are writing is saved in plain text files. In this tutorial, your application will be written in a single text file. Larger applications may require thousands of such files.

Java is an object-oriented language.

A Java class is created by a special declaration in a text file.

public class MyFirstClass {

}

 You need to save this text in a file named MyFirstClass.java. A Java class **must** be saved in a file that has the same name as your class with the extension .java.

You can give this class any name as long as it does not start with a number.

There is a convention though: the name of a Java class starts with a capital letter. This is not mandatory but all Java developers follow this convention.

**Preparing the Compilation of your First Class**

Compiling is the second step you need to follow after the creation of your first class.

It consists of transforming the Java code you wrote in your MyFirstClass.java file into another format that can be executed.

The result of this transformation will be stored in another file created by the compiler. The name of this file will be MyFirstClass.class.

**Source Code**: You write the Java code in a .java file.

**Compile**: When you run javac MyFirstClass.java, the compiler generates the .class file.

**Bytecode**: The content of the .class file is bytecode, which looks something like this when viewed in a tool that can display it:

CAFEBABE 00000034 0012 0A00 0300 0200 0400 0500 0600

0700 0800 0100 0900 0A00 0B00 0C00 0D00 0E00 0F00

10

...

So far the only tool you have been using is a plain text editor. Compiling this class requires a compiler; something you may not have on your computer.

As of now, downloading "Java" means downloading the Java Development Kit, also known as the JDK.

The JDK contains many tools and among them are the ones you will be using to compile and run a Java application. It is officially distributed by the OpenJDK project and by Oracle.

The JRE stands for Java Runtime Environment. It is a subset of the JDK that is not distributed by the OpenJDK or Oracle anymore. It only contained the tools needed to run a Java application. You cannot compile your code with the tools provided in the JRE.

You may also have heard about J2EE, Java EE or Jakarta EE. All these acronyms refer to the Java Enterprise Edition.  Java EE is different from the JDK.

You do not need Java EE to compile and run the simple application we are creating in this tutorial.

**Setting up a Java Development Kit**

There is a one-stop page that always refers to the latest version of the JDK: <https://jdk.java.net/>

Selecting the latest "Ready for use" JDK version takes you to a page where you can download the version of the JDK you need.

Let us download the Windows version. What you get is a ZIP file of about 200MB that you can open with any ZIP utility software. This ZIP file contains the JDK. You can unzip the content of this file anywhere on your computer.

Once this is done you need to create an environment variable called JAVA\_HOME that points to the directory where you unzipped the JDK. First you need to open a DOS prompt. If you unzipped a JDK 22 ZIP file in the D:\jdk\ directory then the command you need to type in this DOS prompt is the following:

> set JAVA\_HOME=D:\jdk\jdk-22

You can check that the JAVA\_HOME variable has been properly set by typing the following code:

> echo %JAVA\_HOME%

You then need to update your PATH environment variable to add the bin directory of your JDK directory to it. This can be done with the following command:

> set PATH=%JAVA\_HOME%\bin;%PATH%

**Compiling your First Class**

### **Compiling and Running Your First Java Program**

Whether you followed the Windows, the Linux or the macOS path, the remaining is the same.

1. Change to the directory where you saved your first class MyFirstClass.java. You can check that you are in the right directory by typing dir. It will show you the files you have in this directory. You should see your MyFirstClass.java file.
2. Check that your compiler is accessible from this directory by typing the following. This command is the same whether you are on Windows or Linux.

> java -version

1. Now you are all set to compile your first code. You can type the following.

> javac MyFirstClass.java

If the compiler remains silent and does not complain about anything: congratulations! It means that your Java code has been properly compiled. Checking the content of the directory again should show a new file in it: MyFirstClass.class

**Adding Code to Your Class to Run it**

So far your class is empty; there is no executable code in it. If you were able to compile it properly then you can advance to the next step and execute some code.

Just open your MyFirstClass.java file and copy the following code in it.

public class MyFirstClass {

public static void main(String... args) {

System.out.println("Hello, World!");

}

}

For compiling the code, type the following command:

>javac MyFirstClass.java

To run it, all you need to type is the following command:

> java MyFirstClass

**Running the Hello World Program as a Single File Application**

Starting with Java SE 11, you can run a Java application without going through the compilation step, as long as the program is written in a single file. This is the case of this simple Hello, World! application.

You can just type the following:

> java MyFirstClass.java

**Going Further**

This first Java program showed you the basic steps every Java developer follows to run an application.

1. Create a source in a set of .java text files
2. Compile these files to produce a set of corresponding .class binary files
3. Run them together as an application.

Developers that work on large applications do not use plain text editors to manage their source code; they use Integrated Development Environments. IDEs are complex software applications, specialized in software development.

These applications handle the compilation of your source code automatically, they can help you to track errors in the syntax of your Java code and nail down bugs in its execution, among other things.

**Single-File Source-Code Program Execution**

In JDK 11, Java introduced the ability to launch a single-file source-code program with the java launcher, without first needing to explicitly compile the source code. This works by the java launcher automatically invoking the compiler and storing the compiled code in-memory. This can be a great way to learn how to use Java or explore new features within the Java API, without having to go through the cruft of compiling and then executing code.

**Executing Your First Single-File Source-Code Program**

To execute a single-file source-code program, the first class defined in the source file must contain public static void main(String[]) like in HelloWorld below:

public class HelloWorld {

public static void main(String[] args) {

System.out.println("Hello World!");

}

}

From the command line, HelloWorld can be launched with

> java HelloWorld.java

### **Passing in Arguments**

Arguments can also be passed in like with a normally compiled class, so in the below:

public class HelloJava {

public static void main(String[] args) {

System.out.println("Hello " + args[0]);

}

}

Passing in an argument can be done like this:

$ java HelloJava.java World!

**Multiple Classes in Same File**

Multiple classes can be defined within the same source file if needed for encapsulation purposes,

public class MultipleClassesInSameFile {

public static void main(String[] args) {

System.out.println(GenerateMessage.generateMessage());

System.out.println(AnotherMessage.generateAnotherMessage());

}

}

class GenerateMessage {

static String generateMessage() {

return "Here is one message";

}

}

class AnotherMessage {

static String generateAnotherMessage() {

return "Here is another message";

}

}

> java MultipleClassesInSameFile.java

Here is one message

Here is another message

**Reference JDK Classes and Non-JDK Classes**

A class that is part of the core JDK does not need to be added to the classpath to be executed. So this example, referencing the Scanner and MatchResult classes, can be executed simply with the java launcher:

import java.util.Scanner;

import java.util.regex.MatchResult;

public class ScannerExample {

public static void main(String... args) {

String wordsAndNumbers = """

Longing rusted furnace

daybreak 17 benign

9 homecoming 1

freight car

""";

try (Scanner scanner = new Scanner(wordsAndNumbers)) { scanner.findAll("benign").map(MatchResult::group).forEach(System.out::println);

}

}

}

> java ScannerExample.java

**INTRODUCTION**

**Java is a high-level object oriented programming language, it enables developer to create a application that can be executed on any platform that has JVM.**

**Java Features :**

Simple – Java provides simple syntax for easy to learn.

Object Oriented – Java supports objects and class which maintains a

Organization of code.

Platform Independent – Java code is compiled into bytecode which can be executed on any platform with JVM.

Portable – Java code compiled can be moved to any environment without need for re compilation.

Secured – Java has features like customizable security manager and bytecode verification which makes it a preferred choice for buildind secured application.

High Performance – Interpreted language may be slow but java use JIT, which convert frequently called method to native code, allowing for a faster execution.

**Working :**

In Java Programming language, all code are written in a text file ending with .java .

These .java files are compiled and converted into byte code .class , which is not a native code.

Then the .class files are interpreted by JVM and converted into native code.

JVM monitors the frequently called methods, these methods are compiled into native code, allowing for faster execution.

Java has two key components :

JVM

Java API

JVM stands for Java virtual machine, responsible for interpreting bytecode and execution of the program , memory management and provides runtime services.

Java API is a set of classes and interfaces that provides core functionalities for developers to create a java application. It includes libraries and functionalities such as input/output , datastructures, networking and GUI.

**OBJECT ORIENTED PROGRAMMING**

**OBJECT -** An Object is a software bundle of state(data) and behaviour(method) . Objects are often used to model real world objects.

Why? Encapsulation, model real-world entities, code reusability.(used to perform actions)

**CLASS -** A class is a prototype from which object is created. In otherwords, instance of a class.

**INTERFACE-** A Interface is a set of methods without method-implementation(that methods must be implemented).

**PACKAGES-** A Package is a organised set of related classes and interfaces.(folders)

**BASICS**

**Variables –** A Variable is a container that stores the data that may or may not change during program execution.

**Instance variables-** A Variable declared inside class and outside any method. Instance variable which are shared by each object of a class,.(Non-static Variable)

**Class Variables-** A Variable declared with a ‘static’ keyword. Unlike static variable are shared by all the objects of class. (Static Variable)

**Local Variables-** A Variable declared inside a method.(Limited within method, no default value)

**Parameter-** A Variable used to pass data into a method/constructor.

**Naming**

Variable names are case-sensitive. (Use Camel Notation)

Unlimited sequence of unicode letters and digits.

Begin with letter , $ , \_ (always begin with letter).

White space is not permitted.

**Datatype**

**Byte-** The Byte datatype is a 8 bit signed integer.(-128 to 127)

**Short-** The Short datatype is a 16-bit signed integer.(-32,768 to 32,767)

**Int-** The Int datatype is a 32-bit signed integer.(-2,147,483,648 to 2,147,483,648)

**Long-** The Long is a 64-bit signed integer.(-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)

**Float-** The Float is a 32-bit signed integer. .(-2,147,483,648 to 2,147,483,648)

**Long-** The Long is a 64-bit signed integer. (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)

**Boolean-** The Boolean is a 1-bit datatype that return a true or false.

**Char-** The Char is a 16-bit datatype.(0 to 65,535 unicode ‘\u0000’ to ‘\uffff’)

ASCII Code (A to Z - 65 to 90 , a to z - 97 to 122 , 0 to 9 – 48 to 57)