DevOps Certification Training

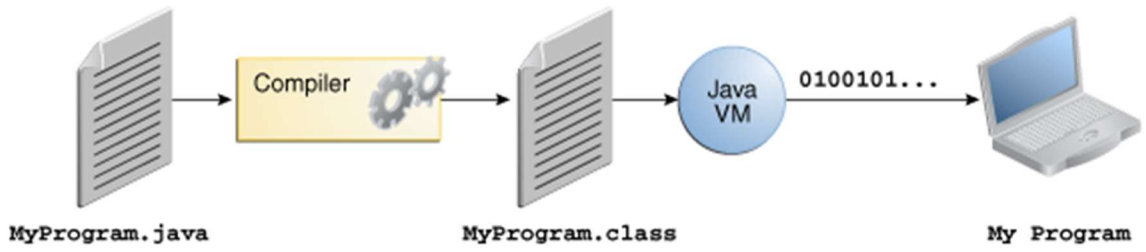
Study Material - Java

The Java Programming Language

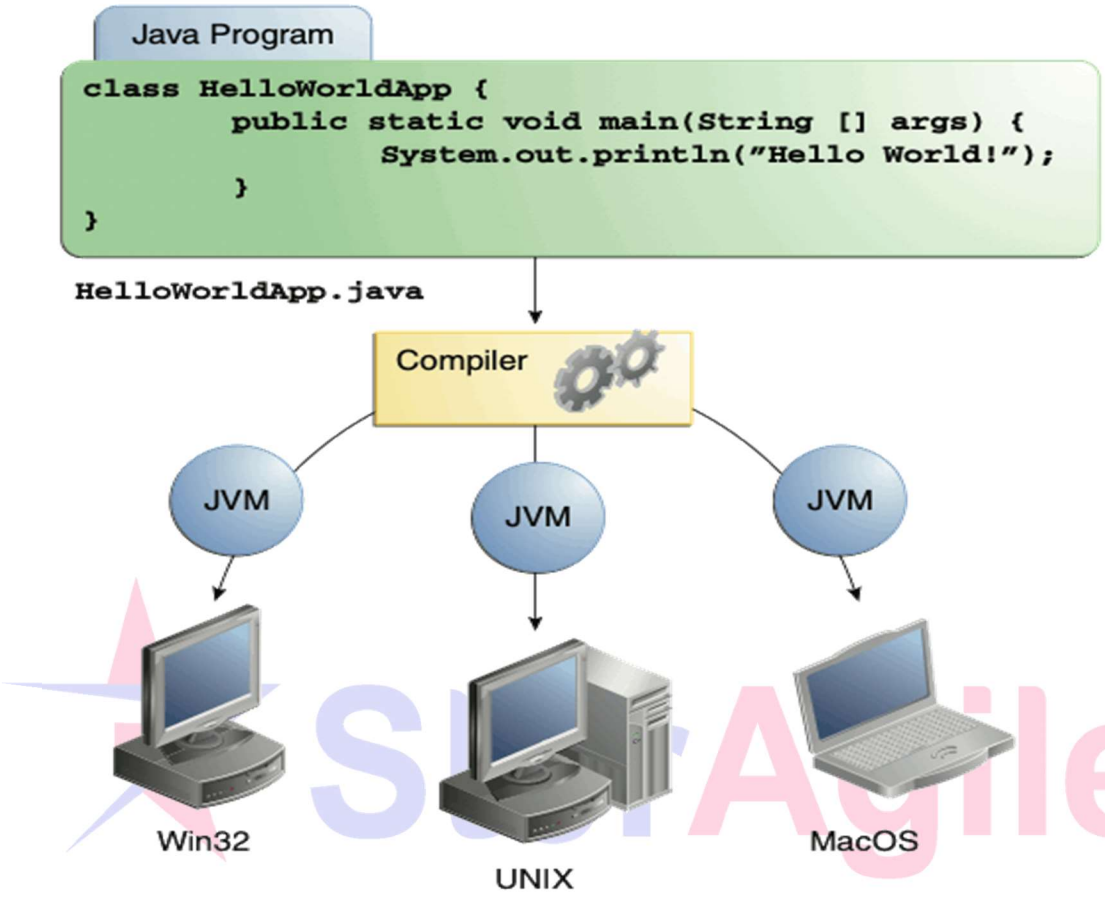
Java technology is both a programming language and a platform. The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

- Simple
- Object oriented
- Distributed
- Multithreaded
- Dynamic
- Architecture neutral
- Portable
- High performance
- Robust
- Secure

In the Java programming language, all source code is first written in plain text files ending with the .java extension. Those source files are then compiled into .class files by the javac compiler. A .class file does not contain code that is native to your processor; it instead contains *bytecodes* — the machine language of the Java Virtual Machine[1] (Java VM). The java launcher tool then runs your application with an instance of the Java Virtual Machine.



Because the Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris™ Operating System (Solaris OS), Linux, or Mac OS. Some virtual machines, such as the Java SE HotSpot at a Glance, perform additional steps at runtime to give your application a performance boost. This includes various tasks such as finding performance bottlenecks and recompiling (to native code) frequently used sections of code.



The Java Platform

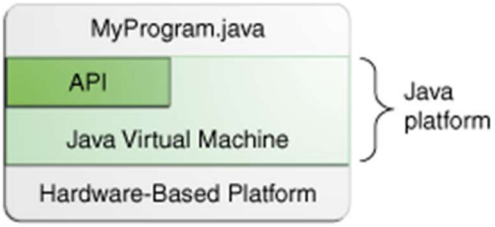
A *platform* is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Microsoft Windows, Linux, Solaris OS, and Mac OS. Most platforms can be described as a combination of the operating system and underlying hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

- The *Java Virtual Machine*
- The *Java Application Programming Interface* (API)

You've already been introduced to the Java Virtual Machine; it's the base for the Java platform and is ported onto various hardware-based platforms.

The API is a large collection of ready-made software components that provide many useful capabilities. It is grouped into libraries of related classes and interfaces; these libraries are known as *packages*. The next section, What Can Java Technology Do? highlights some of the functionality provided by the API.



The API and Java Virtual Machine insulate the program from the underlying hardware.

As a platform-independent environment, the Java platform can be a bit slower than native code. However, advances in compiler and virtual machine technologies are bringing performance close to that of native code without threatening portability.

The terms"Java Virtual Machine" and "JVM" mean a Virtual Machine for the Java platform.

What Can Java Technology Do?

The general-purpose, high-level Java programming language is a powerful software platform. Every full implementation of the Java platform gives you the following features:

- **Development Tools**: The development tools provide everything you'll need for compiling, running, monitoring, debugging, and documenting your applications. As a new developer, the main tools you'll be using are the javac compiler, the java launcher, and the javadoc documentation tool.
- **Application Programming Interface (API)**: The API provides the core functionality of the Java programming language. It offers a wide array of useful classes ready for use in your own applications. It spans everything from basic objects to networking and security, to XML generation and database access, and more.
- **Deployment Technologies**: The JDK software provides standard mechanisms such as the Java Web Start software and Java Plug-In software for deploying your applications to end users.
- **User Interface Toolkits**: The JavaFX, Swing, and Java 2D toolkits make it possible to create sophisticated Graphical User Interfaces (GUIs).

- **Integration Libraries**: Integration libraries such as the Java IDL API, JDBC API, Java Naming and Directory Interface (JNDI) API, Java RMI, and Java Remote Method Invocation over Internet Inter-ORB Protocol Technology (Java RMI-IIOP Technology) enable database access and manipulation of remote objects.

Creating Your First Application

Your first application, HelloWorldApp, will simply display the greeting "Hello World!" To create this program, you will:

- **Create an IDE project**

  When you create an IDE project, you create an environment in which to build and run your applications. Using IDE projects eliminates configuration issues normally associated with developing on the command line. You can build or run your application by choosing a single menu item within the IDE.

- **Add code to the generated source file**

  A source file contains code, written in the Java programming language, that you and other programmers can understand. As part of creating an IDE project, a skeleton source file will be automatically generated. You will then modify the source file to add the "Hello World!" message.

- **Compile the source file into a .class file**

  The IDE invokes the Java programming language *compiler* (javac), which takes your source file and translates its text into instructions that the Java virtual machine can understand. The instructions contained within this file are known as *bytecodes*.

- **Run the program**

  The IDE invokes the Java application *launcher tool* (java), which uses the Java virtual machine to run your application.

High Level Language

A High level language is a language which are human readable and as similar as English, which makes it easier for developer to write the code.

Low Level Language

A Low level language is a language which are machine readable. Machine can not understand High Level Language. Low level language not easy to read and write for

humans. While running programs, high level language is converted into low level language and then it gets executed.

Object-Oriented Programming Concepts

If you've never used an object-oriented programming language before, you'll need to learn a few basic concepts before you can begin writing any code. This lesson will introduce you to objects, classes, inheritance, interfaces, and packages. Each discussion focuses on how these concepts relate to the real world, while simultaneously introducing the syntax of the Java programming language.

What Is an Object?

An object is a software bundle of related state and behavior. Software objects are often used to model the real-world objects that you find in everyday life. This lesson explains how state and behavior are represented within an object, introduces the concept of data encapsulation, and explains the benefits of designing your software in this manner.

What Is a Class?

A class is a blueprint or prototype from which objects are created. This section defines a class that models the state and behavior of a real-world object. It intentionally focuses on the basics, showing how even a simple class can cleanly model state and behavior.

What Is Inheritance?

Inheritance provides a powerful and natural mechanism for organizing and structuring your software. This section explains how classes inherit state and behavior from their superclasses, and explains how to derive one class from another using the simple syntax provided by the Java programming language.

What is Polymorphism ?

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

In Java polymorphism is mainly divided into two types:

- Compile-time Polymorphism (Method Overloading)
- Runtime Polymorphism (Method Overriding)

What is Encapsulation?

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates.

What is abstraction in OOP?

Abstraction is the process of hiding the internal details of an application from the outer world. Abstraction is used to describe things in simple terms. It's used to create a boundary between the application and the client programs.

What Is an Interface?

An interface is a contract between a class and the outside world. When a class implements an interface, it promises to provide the behavior published by that interface. This section defines a simple interface and explains the necessary changes for any class that implements it.

What Is a Package?

A package is a namespace for organizing classes and interfaces in a logical manner. Placing your code into packages makes large software projects easier to manage. This section explains why this is useful, and introduces you to the Application Programming Interface (API) provided by the Java platform.

Variables

You've already learned that objects store their state in fields. However, the Java programming language also uses the term "variable" as well. This section discusses this relationship, plus variable naming rules and conventions, basic data types (primitive types, character strings, and arrays), default values, and literals.

The Java programming language defines the following kinds of variables:

- **Instance Variables (Non-Static Fields)** Technically speaking, objects store their individual states in "non-static fields", that is, fields declared without the static keyword. Non-static fields are also known as *instance variables* because their values are unique to each *instance* of a class (to each object, in other words); the currentSpeed of one bicycle is independent from the currentSpeed of another.
-
- **Class Variables (Static Fields)** A *class variable* is any field declared with the static modifier; this tells the compiler that there is exactly one copy of this variable in existence, regardless of how many times the class has been instantiated. A field defining the number of gears for a particular kind of bicycle could be marked as static since conceptually the same number of gears will apply to all instances. The code static int numGears = 6; would create such a static field. Additionally, the keyword final could be added to indicate that the number of gears will never change.

- **Local Variables** Similar to how an object stores its state in fields, a method will often store its temporary state in *local variables*. The syntax for declaring a local variable is similar to declaring a field (for example, int count = 0;). There is no special keyword designating a variable as local; that determination comes entirely from the location in which the variable is declared — which is between the opening and closing braces of a method. As such, local variables are only visible to the methods in which they are declared; they are not accessible from the rest of the class.

- **Parameters** You've already seen examples of parameters, both in the Bicycle class and in the main method of the "Hello World!" application. Recall that the signature for the main method is public static void main(String[] args). Here, the args variable is the parameter to this method. The important thing to remember is that parameters are always classified as "variables" not "fields". This applies to other parameter-accepting constructs as well (such as constructors and exception handlers) that you'll learn about later in the tutorial.

Primitive Data Types

The Java programming language is statically typed, which means that all variables must first be declared before they can be used. This involves stating the variable's type and name, as you've already seen:

int gear = 1;

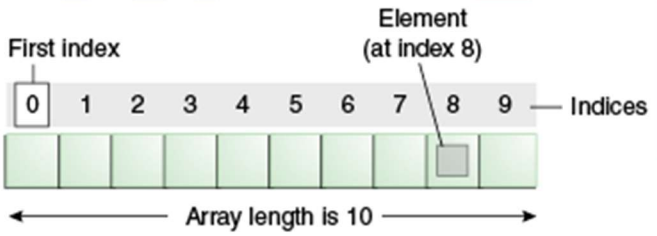
Data Type	Default Value (for fields)
Byte	0
Short	0
Int	0
Long	0L
Float	0.0f
Double	0.0d

Char	'\u0000'
String (or any object)	null
Boolean	false

In addition to the eight primitive data types listed above, the Java programming language also provides special support for character strings via the java.lang.String class. Enclosing your character string within double quotes will automatically create a new String object; for example, String s = "this is a string";. String objects are *immutable*, which means that once created, their values cannot be changed. The String class is not technically a primitive data type, but considering the special support given to it by the language, you'll probably tend to think of it as such.

Arrays

An *array* is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed. You have seen an example of arrays already, in the main method of the "Hello World!" application. This section discusses arrays in greater detail.



An array of 10 elements.

Each item in an array is called an *element*, and each element is accessed by its numerical *index*. As shown in the preceding illustration, numbering begins with 0. The 9th element, for example, would therefore be accessed at index 8.

Control Flow Statements

The statements inside your source files are generally executed from top to bottom, in the order that they appear. *Control flow statements*, however, break up the flow of execution by employing decision making, looping, and branching, enabling your program to *conditionally* execute particular blocks of code. This section describes the decision-making statements (if-then, if-then-else, switch), the looping statements

(for, while, do-while), and the branching statements (break, continue, return) supported by the Java programming language.

What is Tomcat ?

It is an open-source Java servlet container that implements many Java Enterprise Specs such as the Websites API, Java-Server Pages and last but not least, the Java Servlet. The complete name of Tomcat is "Apache Tomcat" it was developed in an open, participatory environment and released in 1998 for the very first time.

What does an application server do?

The function of the application server is to act as host (or container) for the user's business logic while facilitating access to and performance of the business application.

What does web server do ?

The basic objective of the web server is to store, process and deliver web pages to the users. This intercommunication is done using Hypertext Transfer Protocol (HTTP). These web pages are mostly static content that includes HTML documents, images, style sheets, test etc.

What is testing an application ?

Application testing is the process through which applications are tested for required quality, functionality, compatibility, usability, performance, and other characteristics.

Introduction to Spring Boot :

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need minimal Spring configuration.

Features

- Create stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
- Provide opinionated 'starter' dependencies to simplify your build configuration
- Automatically configure Spring and 3rd party libraries whenever possible

- Provide production-ready features such as metrics, health checks, and externalized configuration
- Absolutely no code generation and no requirement for XML configuration