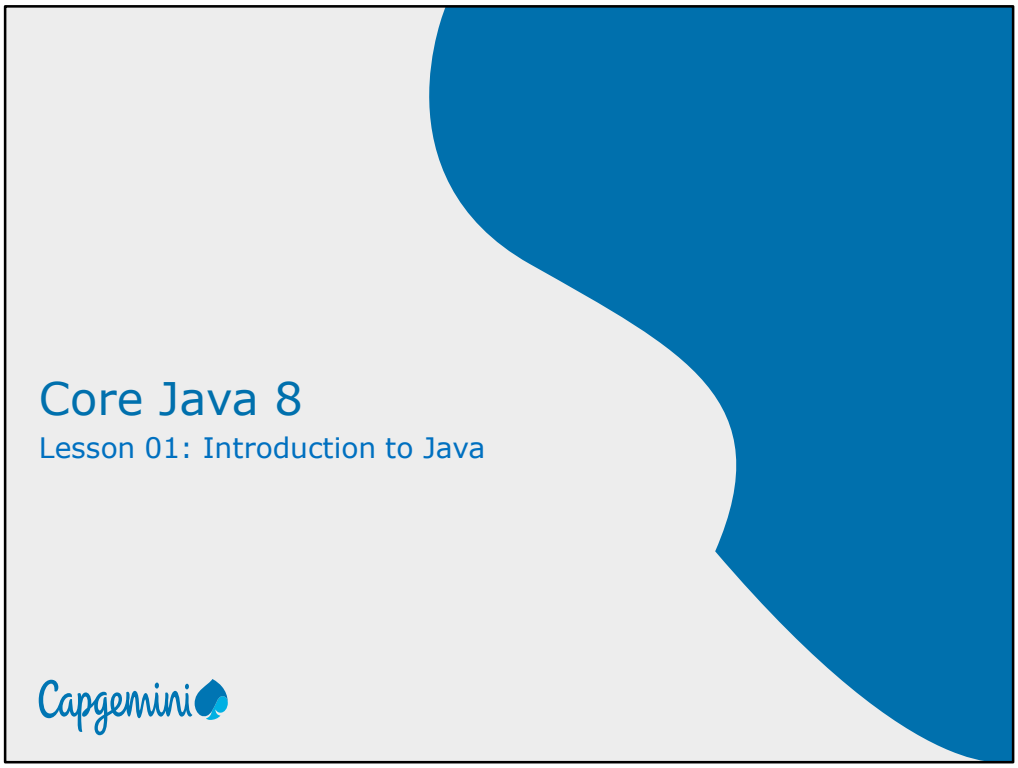

Core Java 8
Lesson 01: Introduction to Java
Capgemini

Lesson Objectives

After completing this lesson, participants will be able to -

- Introduction to Java
- Features of Java
- Evolution in Java
- Developing software in Java

Java's Lineage

C language was result of the need for structured, efficient, high-level language replacing assembly language.

C++, which followed C, became the common (but not the first) language to offer OOP features, winning over procedural languages such as C.

Java, another object oriented language offering OOP features, followed the syntax of C++ at most places. However, it offered many more features.

Introduction to Java:

- To understand Java, a new age Internet programming language, first it is essential to know the forces that drove to the invention of this kind of language. The history starts from a language called as **B,** which led to a famous one **C** and then to **C++**. However, the jump from C to C++ was a major development, as the whole approach of looking at an application changed from **procedural way** to **object oriented way**. The languages like, Smalltalk, were already using the Object Oriented features.
- By the time C++ got the real acknowledgement from the industry, there was a strong need for a language which creates architecture neutral, platform independent, and portable software. The reason behind this particular need was the Embedded software market, which runs on variety of consumer electronic devices. Hence a project called as "OAK" was started at Sun Microsystems.
- The second force behind this is WWW (World Wide Web). Now on WWW, most of computers over the Internet are divided into three major architectures namely Intel, Macintosh, and Unix. Thus, a program written for the Internet has to be a portable program, so that it runs on all the available platforms.
- All this led to the development of a new language in 1995, when they renamed the "OAK" language to "JAVA".

1.1: Introduction to Java

What is Java?

Java is an Object-Oriented programming language – most of it is free and open source!

- It is developed in the early 1990s, by James Gosling of Sun Microsystems
- It allows development of software applications.
- It is amongst the preferred choice for developing internet-based applications

Introduction to Java Features – What is Java?

- Java programming language is a high level programming language offering Object-Oriented features. James Gosling developed it at Sun Microsystems in the early 1990s. It is on the lines of C/C++ syntax, however, it is based on an object model. Sun offers much of Java as free and open source. Today we have the Java version 6 in the market.
- The Java programming language forms a core component of Sun's Java Platform. By platform, we mean the mix of hardware and software environment in which a program executes – such as MS Windows and Linux. Sun's Java is a "software-only" platform which runs on top of other hardware platforms. We will learn more about this on subsequent slides.
- Today, Java has become an extremely popular language. The platform is amongst the preferred choices for developing internet based applications. Why is it so? Let us explore!

1.2 : Features of Java

Java Language Features

Java has advantages due to the following features:

- Completely Object-Oriented
- Simple
- Robust: Strongly typed language
- Security
  - Byte code Verifier
  - Class Loader
  - Security Manager
- Architecture Neutral: Platform independent
- Interpreted and Compiled
- Multithreaded: Concurrent running tasks
- Dynamic
- Memory Management and Garbage Collection

Features of Java:

Java is completely object oriented. C++, being more compatible to C, allows code to exist outside classes too. However in Java, every line of code has to belong to some or other class. Thus it is closer to true object oriented language.
Java is simpler than C++, since concepts of pointers or multiple inheritance do not exist.
Let us discuss these features in detail.

Object-Oriented:

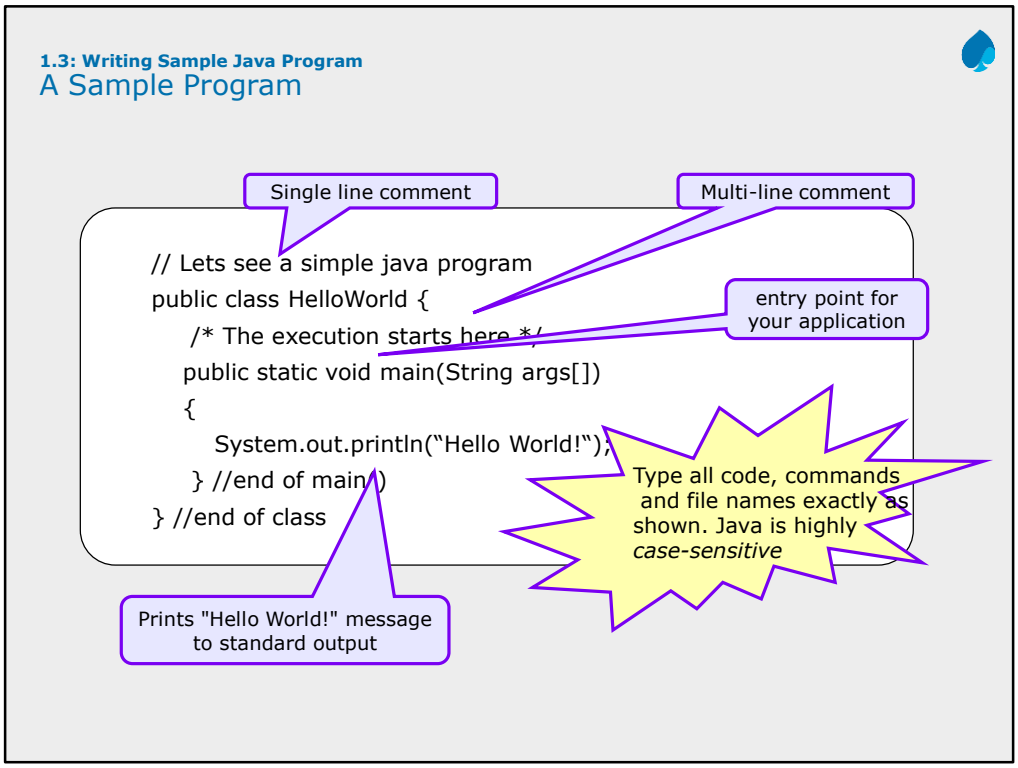
To stay abreast of modern software development practices, Java is Object-Oriented from the ground up. Many of Java's Object-Oriented concepts are inherited from C++, the language on which it is based, plus concepts from other Object-Oriented languages as well.

Simple:

Java omits many confusing, rarely used features of C++. There is no pointer level programming or pointer arithmetic. Memory management is automatic. There are no header files, structures, unions, operator overloading, virtual base classes, and multiple inheritance.

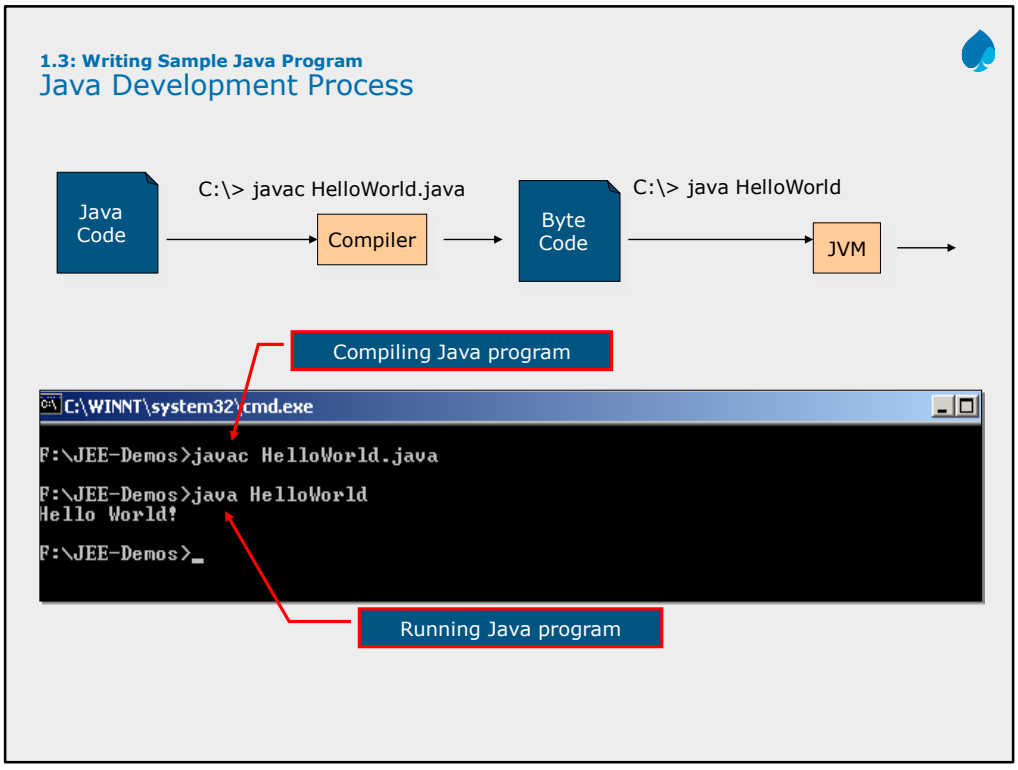
Robust:

Java programs are reliable. Java puts a lot of emphasis on early checking for potential problems, dynamic checking and eliminating situations that are error-prone. Java has a pointer model that eliminates possibility of overwriting memory and corrupting data.



Introduction to Java Features – A Sample Program:

- Here is our first Java program..
- Let us have a closer Look at the "Hello World!" program:
  - **class HelloWorld** begins the class definition block for the "Hello World!" program. Every Java program must be contained within a "class" block.
  - **public static void main(String[] args) { …}** is the entry point for your application. Subsequently, it invokes all the other methods required by the program.
- This program when executed will display "Hello World" on the screen. We shall see this in the next slide.



Introduction to Java Features – Java Development Process:

The Java Development Process involves the following steps:

1. Write the Java code in a text file with a .java extension, namely "HelloWorld.java". By convention, source file names are named by the public class name they contain.
2. At the command line, compile the code using the Java compiler (javac). The javac compiler converts the source into Byte Codes and stores it in a file having .class extension. What is special about byte codes? Unlike traditional compilers, javac does not produce processor specific native code. Instead it generates code which is in the language of JVM (Java Virtual Machine).
   Note: To have access to the **javac** and the **java** commands, you must set your path first. To do so, you may type the following at the command prompt:
   Set path=<your java-home directory>\bin
   For example: *Set path= C:\Program Files\Java\jdk5.8.0_25 \bin*
   We can also set the environment variables (path and classpath) through the Control Panel.
3. To run this program, use the Java command with class file name as the command parameter as shown on the above slide. The output of the program would, of course, be "Hello World!"

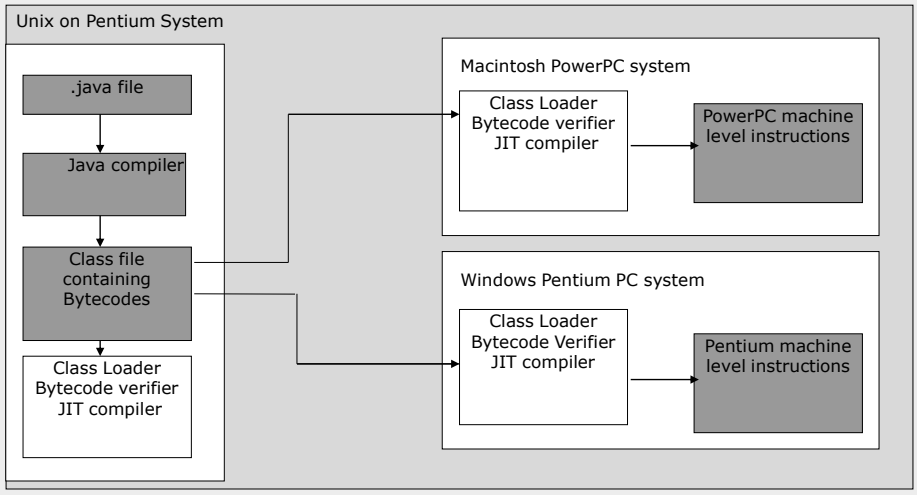
1.3: Writing Sample Java Program

Demo

Creating and executing the First Java application

1.4: Developing Software in Java

Platform Independence feature of Java



Platform Independence:

•The figure illustrates how platform independence is achieved using Java. Once you write Java code on a platform and run it through Java Compiler, the class file containing byte codes is obtained.

•Different JVMs are available for different platforms. So the JVM for Unix on Pentium will be different from the JVM for Mac or for Windows. Each of these JVMs take the same input, namely the Class File, and produce the machine level instructions for the respective platforms.

•One common grouse among developers is that Java programs take longer to execute because the compiled bytecodes are *interpreted* by the JVM. The Java just-in-time (JIT) compiler, compiles the bytecode into platform-specific executable code (**native code**) that is immediately executed, thus speeding up execution! Traditional native code compilers run on the developer's machine and are used by programmers, and produce non-portable executables. JIT compilers run on the user's machine and are transparent to the user. The resulting native code instructions do not need to be ported because they are already at their destination.

1.4: Developing Software in Java

JRE versus JDK

JRE is the "Java Runtime Environment". It is responsible for creating a Java Virtual Machine to execute Java class files (that is, run Java programs).

JDK is the "Java Development Kit". It contains tools for Development of Java code (for example: Java Compiler) and execution of Java code (for example: JRE)

JDK is a superset of JRE. It allows you to do both – write and run programs.

Difference between JRE and JDK:

- The **Java Development Kit (JDK)** is a superset which includes Java Compilers, Java Runtime Environments (JRE), Development Libraries, Debuggers, Deployment tools, and so on. One needs JDK to develop Java applications. We have different versions that include JDK 1.2, JDK 1.4, and so on.
- The **Java Runtime Environment (JRE)** is an implementation of JVM that actually executes the Java program. It is a subset of JDK. One needs JRE to execute Java applications.

Summary

In this lesson, you have learnt:

- Features of Java and its different versions
- How Java is platform Independent
- Difference between JRE and JDK
- Writing, Compiling, and Executing a simple program

Review Question

Question 1: A program written in the Java programming language can run on any platform because...

- **Option 1:** The JIT Compiler converts the Java program into machine equivalent
- **Option 2:** The Java Virtual Machine1(JVM) interprets the program for the native operating system
- **Option 3:** The compiler is identical to a C++ compiler
- **Option 4:** The APIs do all the work

Question 2: Java Compiler compiles the source code into ___ code, which is interpreted by ___ to produce Native Executable code.

Review Question

Question 3: Which of the following are true about JVM?

- **Option 1:** JVM is an interpreter for byte code
- **Option 2:** JVM is platform dependent
- **Option 3:** Java programs are executed by the JVM
- **Option 4:** All the above is true

Question 4 : _____ allows a Java program to perform multiple activities in parallel.

- **Option 1:** Java Beans
- **Option 2:** Swing
- **Option 3:** Multithreading
- **Option 4:** None of the above