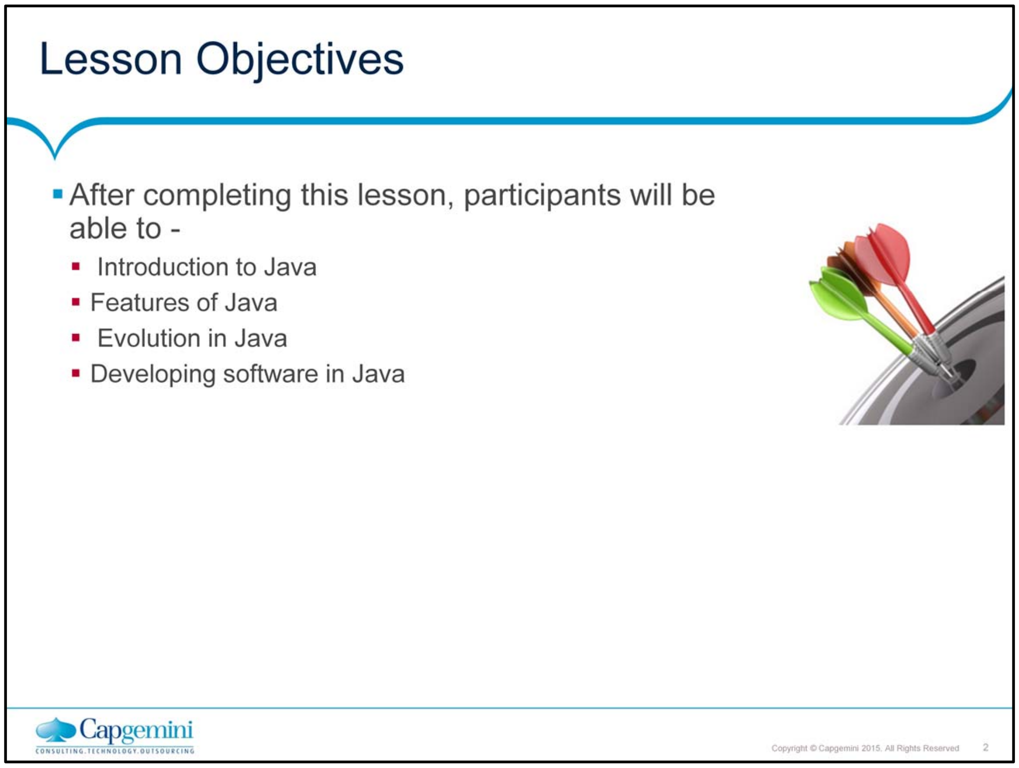
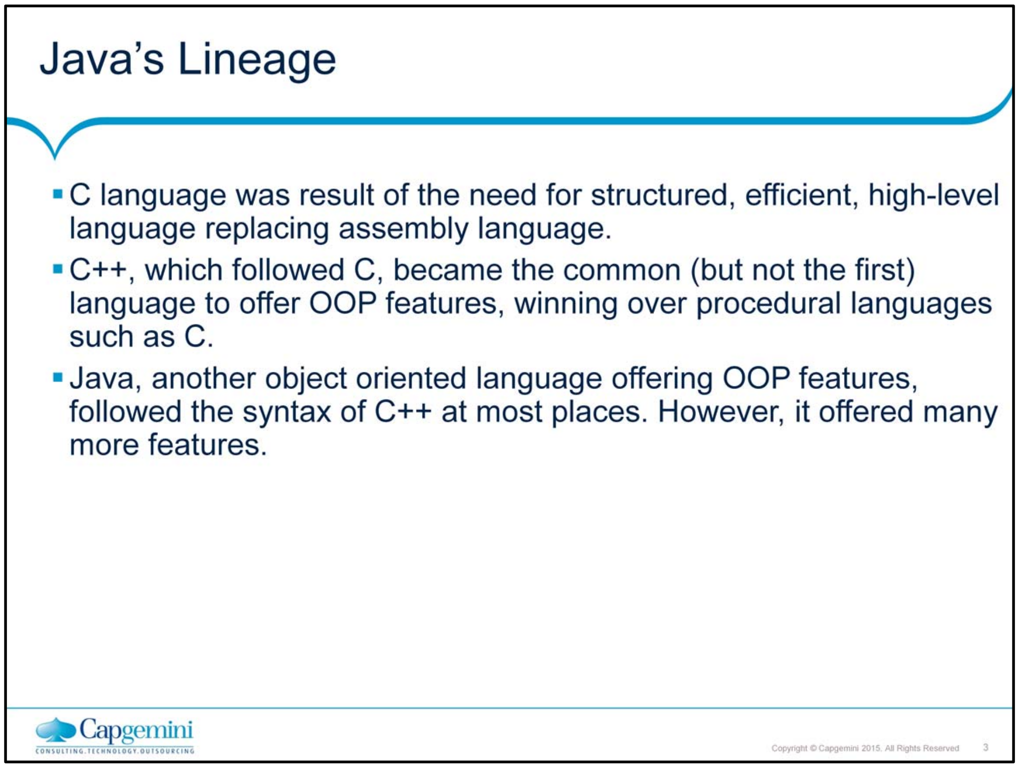
Core Java 8 and Development Tools

Lesson 01: Introduction to Java



Lesson Objectives

- After completing this lesson, participants will be able to -
  - Introduction to Java
  - Features of Java
  - Evolution in Java
  - Developing software in Java

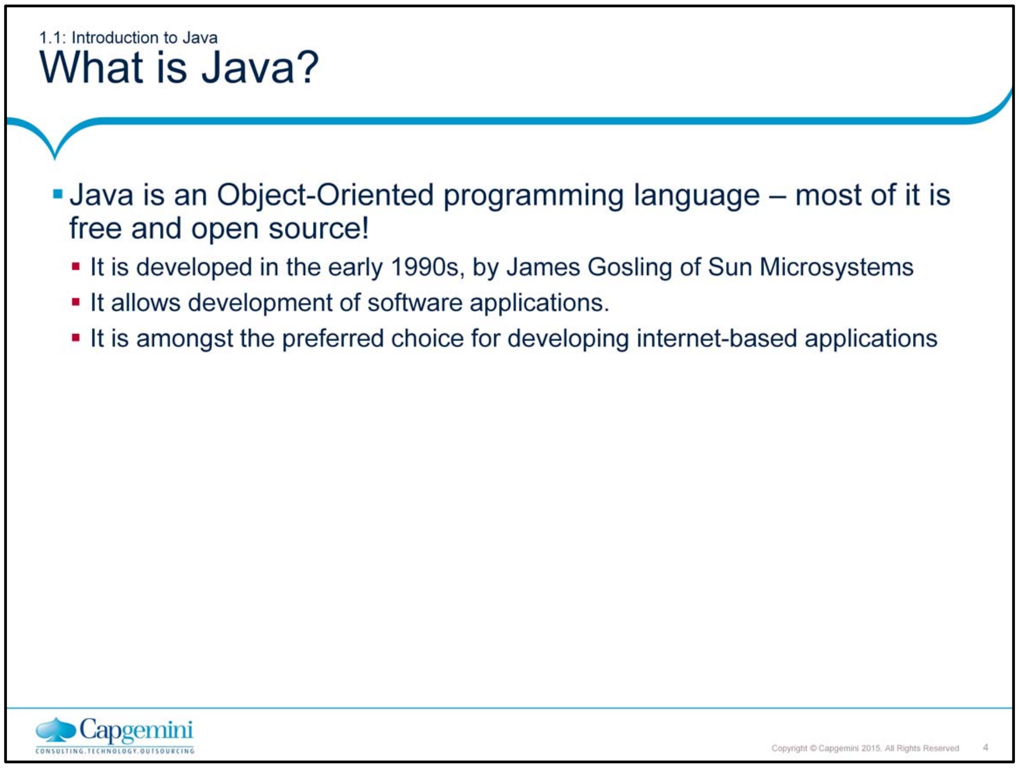


Java's Lineage

- 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".



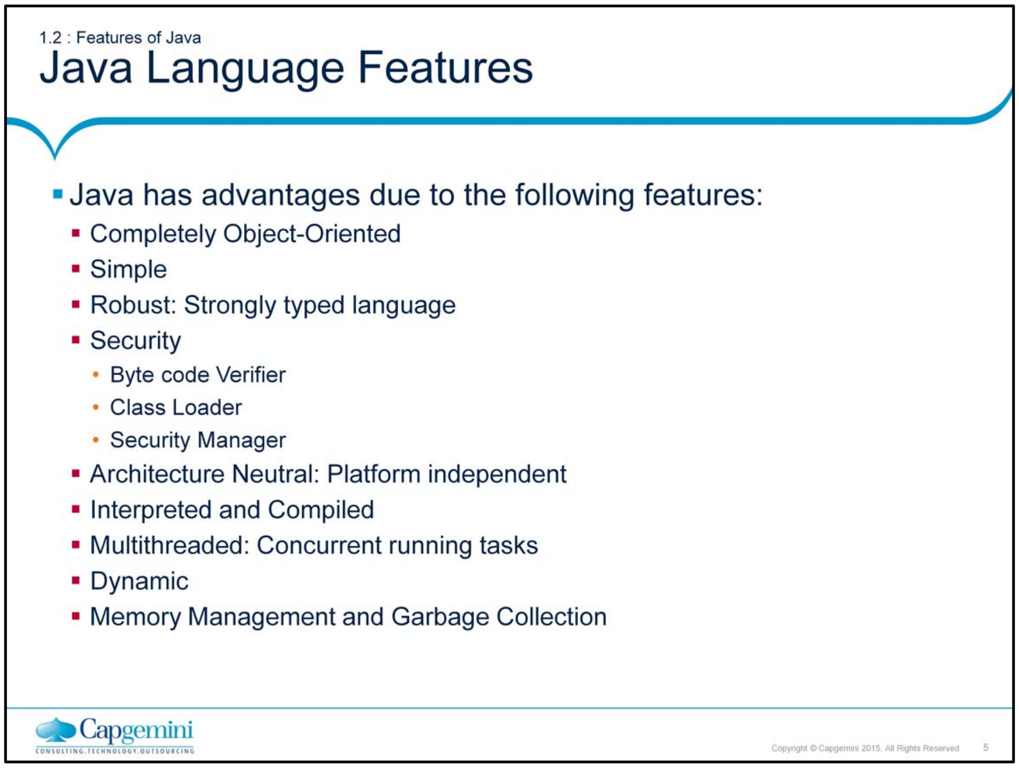
1.1: Introduction to Java

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

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

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.

Features of Java (contd.):

Security:
Java is intended to be used in networked / distributed environments. Thus a lot of emphasis is placed on security. Normally two things affect security:

- confidential information may be compromised, and
- computer systems are vulnerable to corruption or destruction by hackers

Java's security model has three primary components:

- **Byte code Verifier**: The byte code verifier ensures the following:
  - ➢ the Java programs have been compiled correctly,
  - ➢ they will obey the virtual machine's access restrictions, and
  - ➢ the byte codes will not access private data when they should not
- **Class Loader:** When the loader retrieves classes from the network, it keeps classes from different servers separate from each other and from local classes. Through this separation, the class loader prevents a class that is loaded off the network from pretending to be one of the standard built-in classes, or from interfering with the operation of classes loaded from other servers.
- **Security Manager:** It implements a security policy for the VM. The security policy determines which activities of the VM is allowed to perform and under what circumstances, operation should pass.

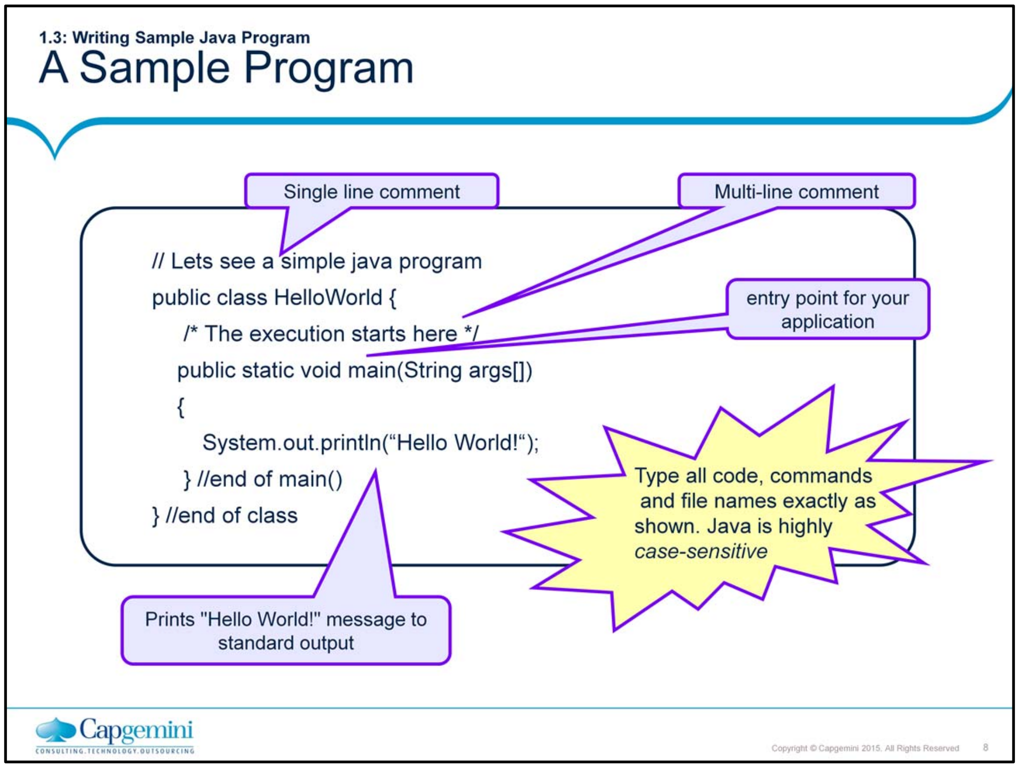
Architecture-Neutral:
A central issue for the Java designers was of code longevity and portability. One of the main problems facing programmers is that there is no guarantee that when you write a program today, it will run tomorrow — even on the same machine. Operating system upgrades, processor upgrades, and changes in core system resources can all combine to make a program malfunction. The Java designers made several hard decisions in the Java language and the Java Virtual Machine (JVM) in an attempt to alter this situation. Their goal was "write once; run anywhere, any time, forever".
To a great extent, this goal was accomplished.

Platform-Independent:
This refers to a program's capability of moving easily from one computer system to another. Java is platform-independent at both the source and the binary level.

- ➢ At the source level, Java's primitive data types have consistent sizes across all development platforms.
- ➢ Java binary files are also platform-independent and can run on multiple platforms without the need to recompile the source because they are in the form of byte codes.

Features of Java (contd.):

Interpreted and Compiled:
Java programs are compiled into an intermediate byte code format, which in turn will be interpreted by the VM at run time. Hence, any Java program is checked twice before it actually runs.
Multithreaded:
A multi-threaded application can have several threads of execution running independently and simultaneously. These threads may communicate and co-operate. To the user it will appear to be a single program. Java implements multithreading through a part of its class library. However, Java also has language constructs to make programs thread-safe.
Dynamic:
Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time. This makes it possible to dynamically link code in a safe and expedient manner. This is crucial to the robustness of the applet environment, in which small fragments of bytecode may be dynamically updated on a running system.
Memory Management and Garbage Collection:
Java manages memory de-allocation by using garbage collection. Temporary memory is automatically reclaimed after it is no longer referenced by any active part of the program. To improve performance, Java's garbage collector runs in its own low-priority thread, providing a good balance of efficiency and real-time responsiveness.



1.3: Writing Sample Java Program

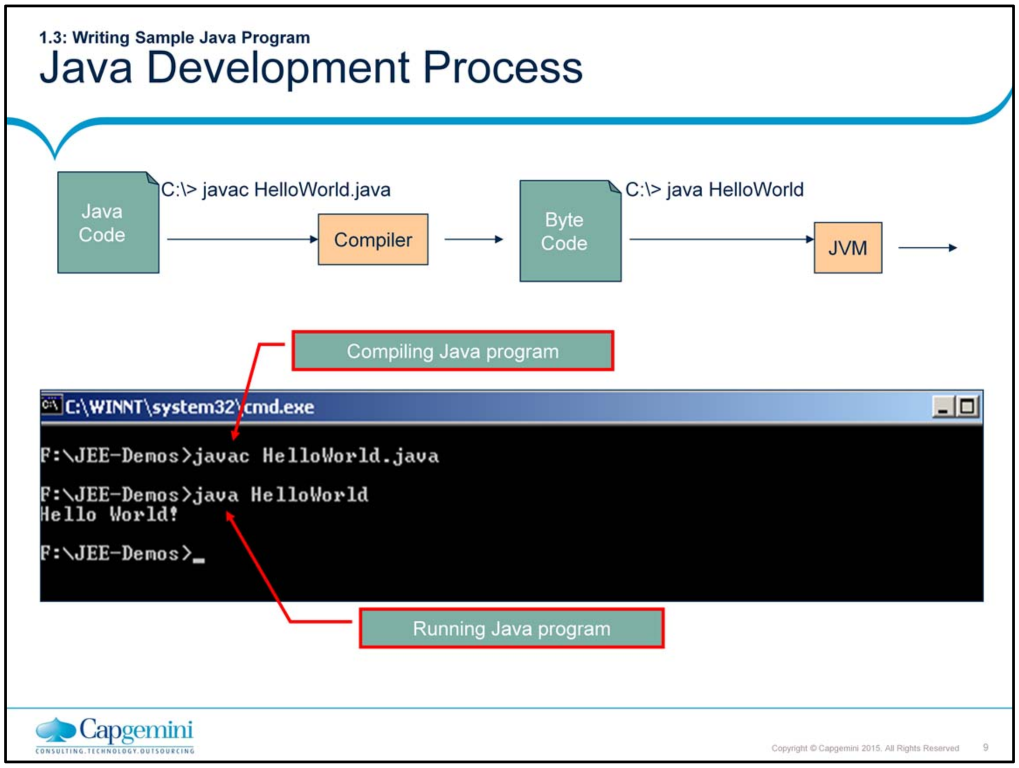
A Sample Program

```
// Lets see a simple java program
public class HelloWorld {
  /* The execution starts here */
  public static void main(String args[])
  {
    System.out.println(“Hello World!“);
  } //end of main()
} //end of class
```

Single line comment
Multi-line comment
entry point for your application
Type all code, commands and file names exactly as shown. Java is highly *case-sensitive*
Prints "Hello World!" message to standard output

<u>Introduction to Java Features – A Sample Program</u>:

- 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.



1.3: Writing Sample Java Program

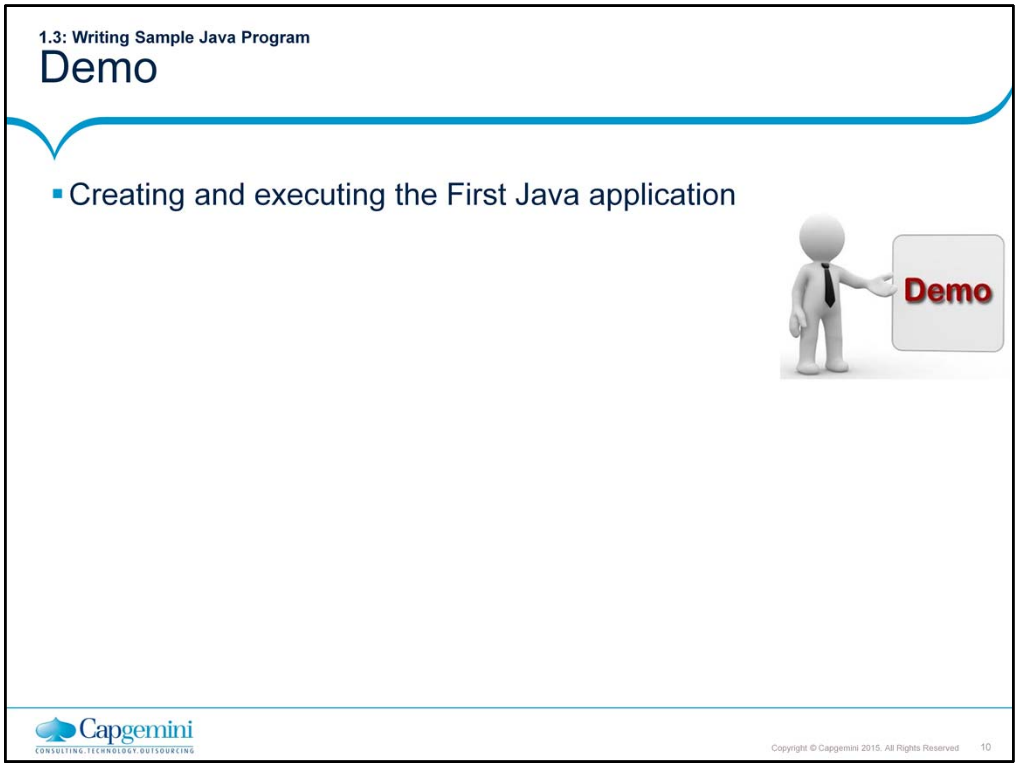
Java Development Process

```
F:\JEE-Demos>javac HelloWorld.java

F:\JEE-Demos>java HelloWorld
Hello World!

F:\JEE-Demos>_
```

<u>Introduction to Java Features – Java Development Process</u>:

The Java Development Process involves the following steps:

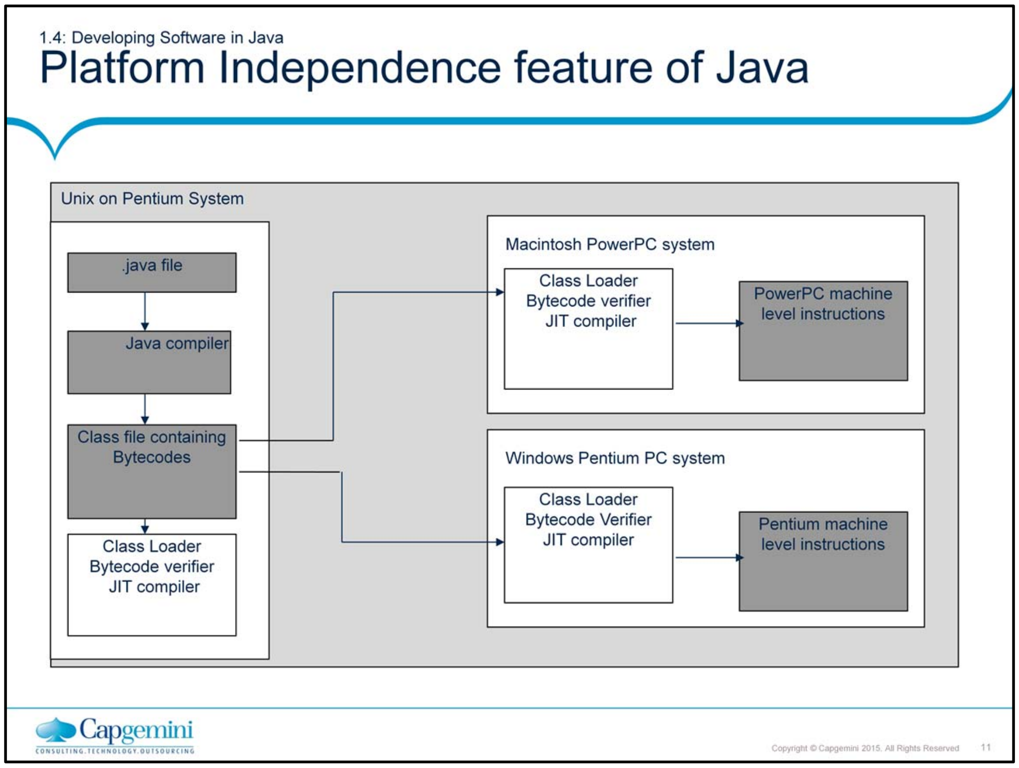
1. Write the Java code in a text file with a .java extension, namely “HelloWorld.java”. By convention, source file names are named by the public class name they contain.
2. At the command line, compile the code using the Java compiler (javac). The javac compiler converts the source into Byte Codes and 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\jdk1.8.0_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

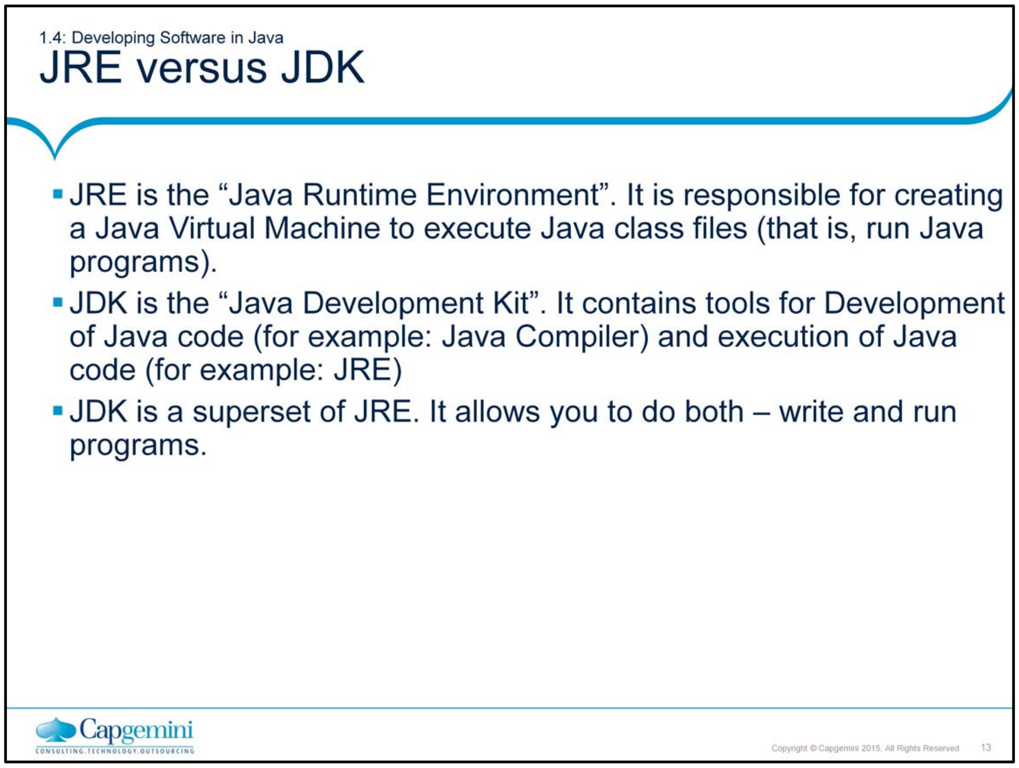
Platform Independence feature of Java

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.

Platform Independence (contd.):
JVM:
When you compile a Java program (which usually is a simple text file with .java extension), it is compiled to be executed under VM. This is in contrast to C/C++ programs, which are compiled to be run on a real hardware platform, such as a Pentium processor running on, say Win 95. The VM itself has characteristics very much like a physical microprocessor. However, it is entirely a software construct. You can think of the VM as an intermediary between **Java programs** and the underlying **hardware platform** on which all programs must eventually execute.

- Even with the VM, at some point, all Java programs must be resolved to a particular underlying hardware platform. In Java, this resolution occurs within each particular VM implementation. The way this works is that Java programs make calls to the VM, which in turn routes them to appropriate native calls on the underlying platform. It is obvious that the **VM itself** is very much **platform dependent**.

How does the JIT compiler work?
- The VM instead of calling the underlying native operating system, it calls the JIT compiler. The JIT compiler in turn generates native code that can be passed on to the native operating system for execution. The primary benefit of this arrangement is that the JIT compiler is completely transparent to everything except VM. The neat thing is that a JIT compiler can be integrated into a system without any other part of the Java runtime system being affected.
- The integration of JIT compilers at the VM level makes JIT compilers a legitimate example of component software. You can simply plug in a JIT compiler and reap the benefits with no other work or side effects.
- A Java enabled browser contains its own VM. Web documents that have embedded Java applets must specify the location of the main applet class file. The Web browser then starts up the VM and passes the location of the applet class file to the class loader. Each class file knows the names of any additional class files that it requires. These additional class files may come from the network or from client machine. Supplement classes are fetched only if they are actually going to be used or if they are necessary for the verification process of the applets.



1.4: Developing Software in Java

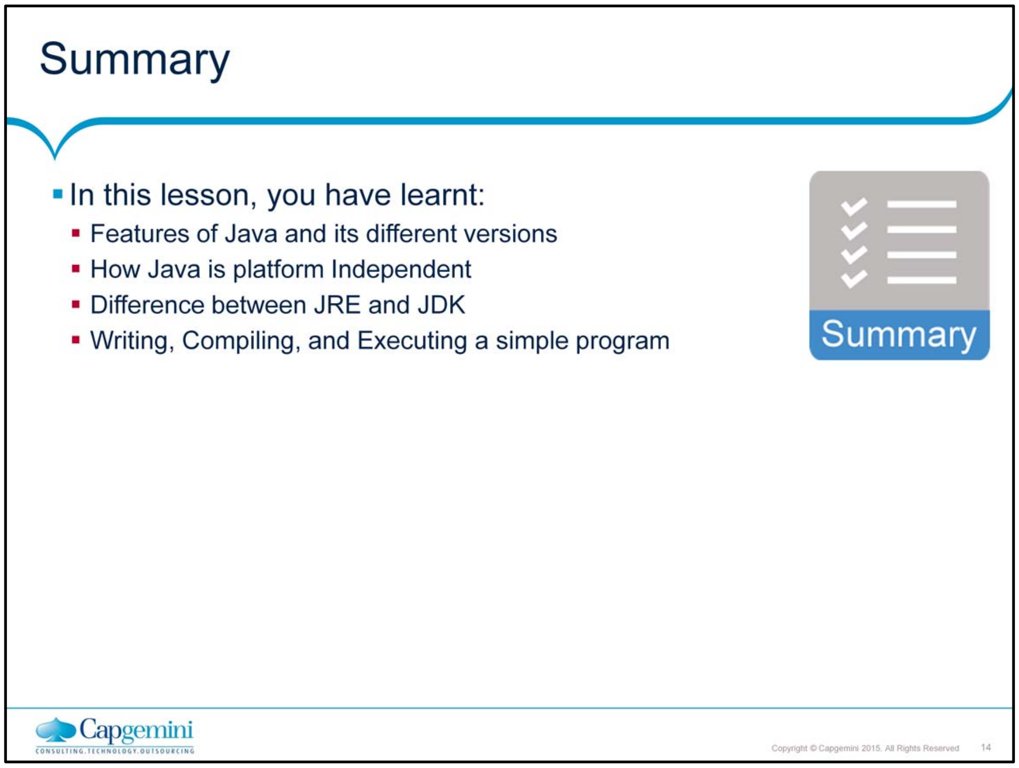
JRE versus JDK

- 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.

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

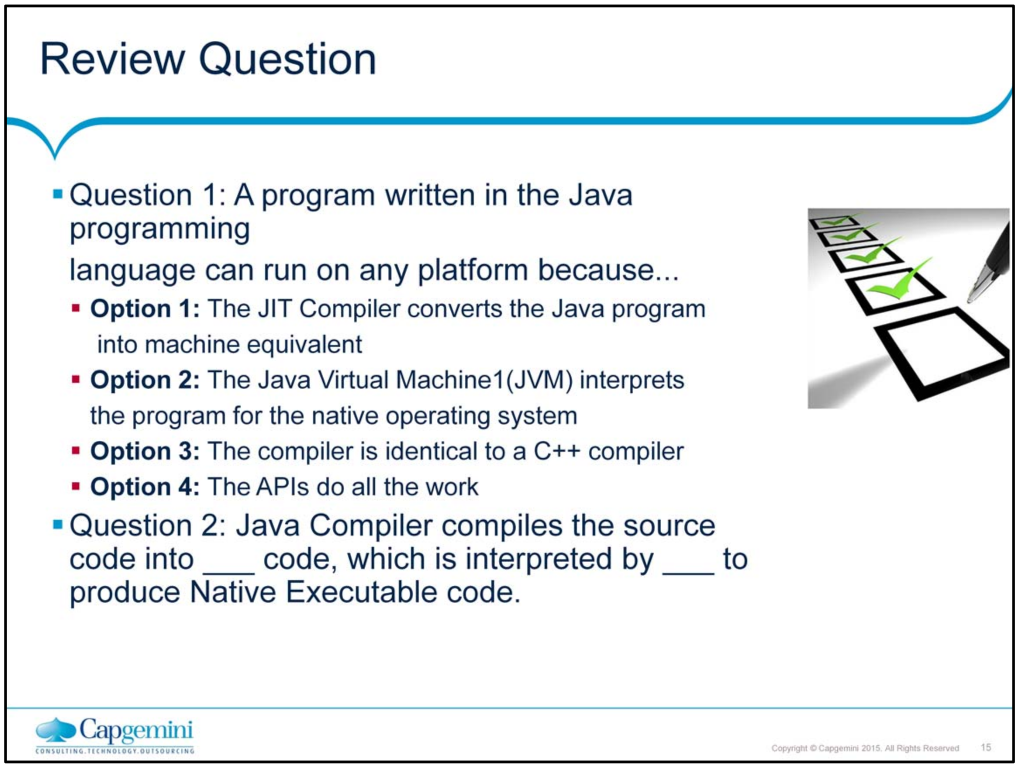
Difference between JRE and JDK:

- 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.



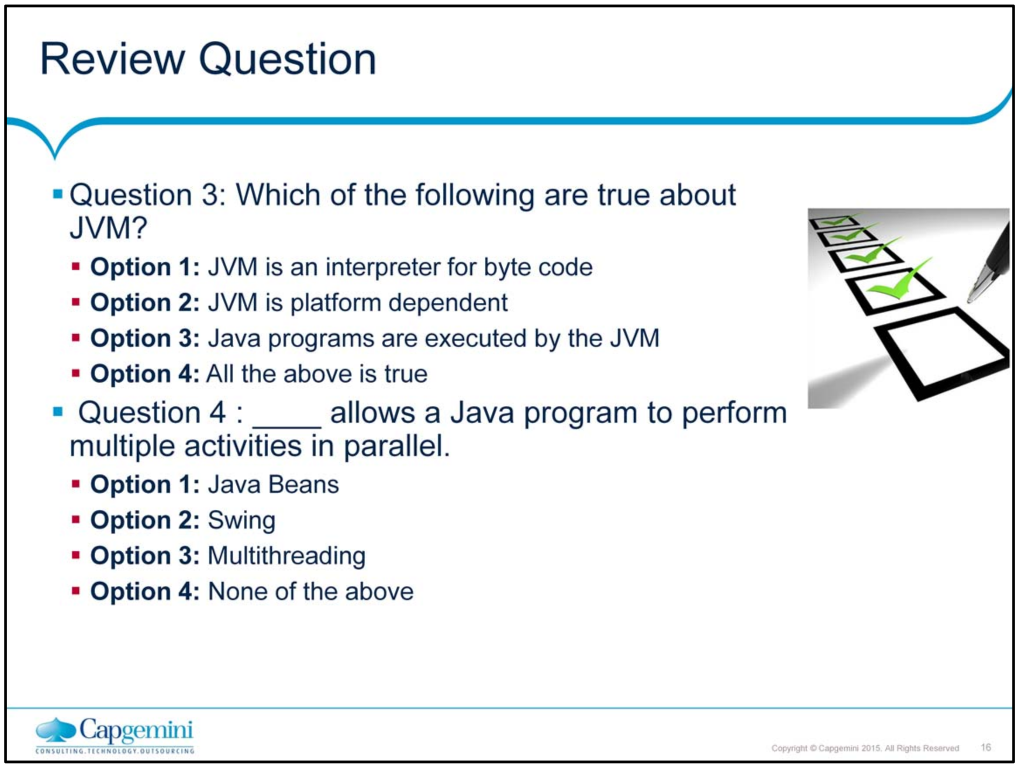
Summary

- 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

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

16