IANT Educom Private Limited

Module 1

Getting Started

Objectives

Upon completion of this module, you should be able to:

- Describe the key features of Java technology
- Write, compile, and run a simple Java technology application
- Describe the function of the Java Virtual Machine (JVM™)[1]
- Define garbage collection
- List the three tasks performed by the Java platform that handle code security

This module provides a general overview of Java technology, including the JVM, garbage collection, security features, and JVM Debug Interface tool.

Mayur Patel (mayurp5011@gmail.com) has a non-transferable license to use this Student Guide

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

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

Copyright 2008 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision G.2

IANT Educom Private Limited

Relevance

Discussion – The following questions are relevant to the material presented in this module:

- Is the Java programming language a complete language or is it useful only for writing programs for the web?

  ____
  ____
  ____

- Why do you need another programming language?

  ____
  ____
  ____

- How does the Java technology platform improve on other language platforms?

  ____
  ____
  ____

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Mayur Patel (mayurp391@gmail.com) has a non-transferable license to use this Student Guide.

IANT Educom Private Limited

Additional Resources

Additional resources – The following references provide additional details on the topics discussed in this module:

- Gosling, Joy, Bracha, and Steele. *The Java Language Specification, Second Edition.* Addison-Wesley. 2000. [Also online at: `http://java.sun.com/docs/books/jls/`].
- Lindholm and Yellin. *The Java Virtual Machine Specification, Second Edition.* Addison-Wesley. 1999. [Also online at: `http://java.sun.com/docs/books/vmspec/`].
- Yellin, Frank. Low-Level Security in Java, white paper. [Online]. Available: `http://java.sun.com/sfaq/verifier.html`.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Mayur Patel (mayurp391@gmail.com) has a non-transferable license to use this Student Guide.

Copyright 2008 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision G.2

IANT Educom Private Limited

What Is the Java™ Technology?

The Java technology is:

- A programming language
- A development environment
- An application environment
- A deployment environment

The syntax of the Java programming language is similar to C++ syntax. You can use the Java programming language to create all kinds of applications that you could create by using any conventional programming language.

As a development environment, Java technology provides you with a large suite of tools: a compiler, an interpreter, a documentation generator, a class file packaging tool, and so on.

The Java programming language is usually mentioned in the context of the World Wide Web (web) and browsers that are capable of running programs called *applets*. Applets are programs written in the Java programming language that reside on web servers, are downloaded by a browser to a client's system, and are run by that browser. Applets are usually small in size to minimize download time and are invoked by a Hypertext Markup Language (HTML) web page.

Java technology *applications* are standalone programs that do not require a web browser to execute. Typically, they are general-purpose programs that run on any machine where the Java runtime environment (JRE) is installed.

There are two main *deployment environments*. First, the JRE supplied by the Java 2 Software Development Kit (Java 2 SDK) contains the complete set of class files for all of the Java technology packages, which includes basic language classes, GUI component classes, an advanced Collections API, and so on. The other main deployment environment is on your web browser. Most commercial browsers supply a Java technology interpreter and runtime environment.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Mayur Patel (mayurp91@gmail.com) has a non-transferable license to use this Student Guide

IANT Educom Private Limited

Primary Goals of the Java Technology

Java technology provides the following:

- A language that is easy to program because it:
  - Eliminates many pitfalls of other languages, such as pointer arithmetic and memory management that affect the robustness of the code
  - Is object-oriented to help you visualize the program in real-life terms
  - Enables you to streamline the code
- An interpreted environment resulting in the following benefits:
  - Speed of development – Reduces the compile-link-load-test cycle
  - Code portability – Enables you to write code that can be run on multiple operating systems on any certified JVM
- A way for programs to run more than one thread of activity
- A means to change programs dynamically during their runtime life by enabling them to download code modules
- A means of ensuring security by checking loaded code modules

The Java technology architecture uses the following features to fulfill the previously listed goals:

- The JVM
- Garbage collection
- The JRE
- JVM tool interface

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Mayur Patel (mayur0391@gmail.com) has a non-transferable license to use this Student Guide.

Copyright 2008 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision G.2

IANT Educom Private Limited

The Java Virtual Machine

The Java Virtual Machine Specification defines the JVM as:

> *An imaginary machine that is implemented by emulating it in software on a real machine. Code for the JVM is stored in* `.class` *files, each of which contains code for at most one public class.*

The Java Virtual Machine Specification provides the hardware platform specifications to which you compile all Java technology code. This specification enables the Java software to be platform-independent because the compilation is done for a generic machine, known as the JVM. You can emulate this *generic machine* in software to run on various existing computer systems or implement it in hardware.

The compiler takes the Java application source code and generates *bytecodes*. Bytecodes are machine code instructions for the JVM. Every Java technology interpreter, regardless of whether it is a Java technology development tool or a web browser that can run applets, has an implementation of the JVM.

The JVM specification provides concrete definitions for the implementation of the following: an instruction set (equivalent to that of a central processing unit [CPU]), a register set, the class file format, a runtime stack, a garbage-collected heap, a memory area, fatal error reporting mechanism, and high-precision timing support.

The code format of the JVM machine consists of compact and efficient bytecodes. Programs represented by JVM bytecodes must maintain proper type discipline. The majority of type checking is done at compile time.

Any compliant Java technology interpreter must be able to run any program with class files that conform to the class file format specified in *The Java Virtual Machine Specification*.

The JVM design enables the creation of implementations for multiple operating environments. For example, Sun Microsystems provides implementations of the JVM for the Solaris OS and the Linux and Microsoft Windows operating environments.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Mayur Patel (mayurp301@gmail.com) has a non-transferable license to use this Student Guide.

IANT Educom Private Limited

Garbage Collection

Many programming languages permit the memory to be allocated dynamically at runtime. The process of allocating memory varies based on the syntax of the language, but always involves returning a pointer to the starting address of a memory block.

After the allocated memory is no longer required (the pointer that references the memory has gone *out of extent*), the program or runtime environment should de-allocate the memory.

In C, C++, and other languages, you are responsible for de-allocating the memory. This can be a difficult exercise at times, because you do not always know in advance when memory should be released. Programs that do not de-allocate memory can crash eventually when there is no memory left on the system to allocate. These programs are said to have *memory leaks*.

The Java programming language removes you from the responsibility of de-allocating memory. It provides a system-level thread that tracks each memory allocation. During idle cycles in the JVM, the garbage collection thread checks for and frees any memory that can be freed.

Garbage collection happens automatically during the lifetime of a Java technology program, eliminating the need to deallocate memory and avoiding memory leaks. However, garbage collection schemes can vary dramatically across JVM implementations.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Mayur Patel (mayurp351@gmail.com) has a non-transferable license to use this Student Guide.

Copyright 2008 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision G.2

IANT Educom Private Limited

The Java Runtime Environment

Figure 1-1 illustrates the JRE and how it enforces code security.

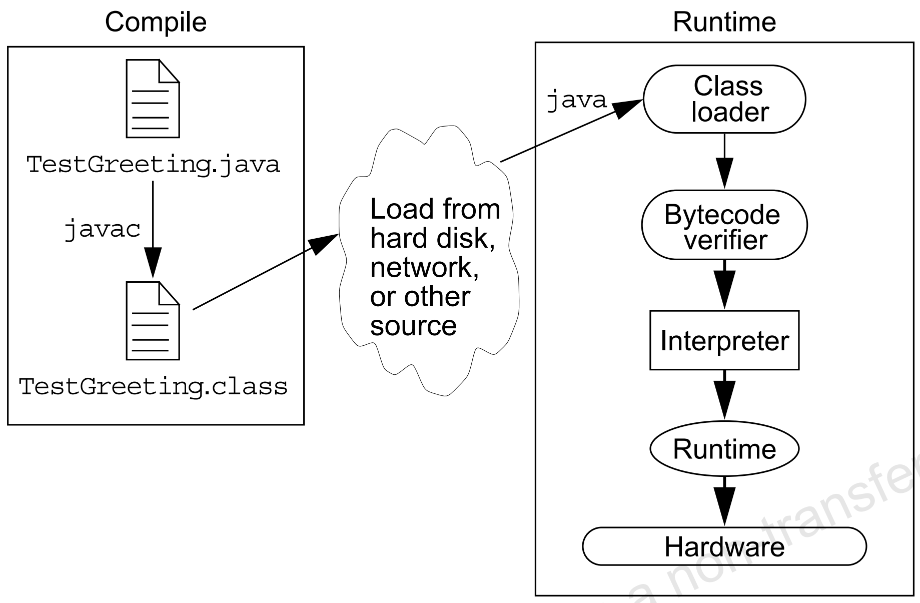

Figure 1-1 Operation of the JRE

Java software source files are *compiled* in the sense that they are converted into a set of bytecodes from the text format in which you write them. The bytecodes are stored in `.class` files.

At runtime, the bytecodes that make up a Java software program are loaded, checked, and run in an interpreter. In the case of applets, you can download the bytecodes, and then they are interpreted by the JVM built into the browser. The interpreter has two functions: it executes bytecodes and it makes the appropriate calls to the underlying hardware.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.
Mayur Patel (mayurp301@gmail.com) has a non-transferable license to use this Student Guide.

IANT Educom Private Limited

In some Java technology runtime environments, a portion of the verified bytecode is compiled to native machine code and executed directly on the hardware platform. This enables the Java software code to run close to the speed of C or C++ with a small delay at load time to enable the code to be compiled to the native machine code (see Figure 1-2).

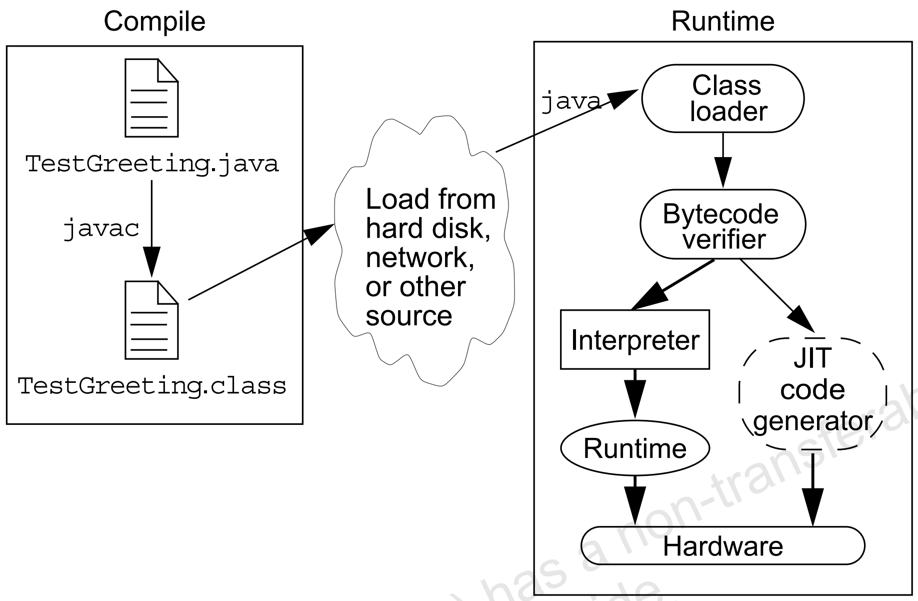

Figure 1-2 Operation of the JRE With a Just-In-Time (JIT) Compiler

Note – Sun Microsystems has enhanced the JVM machine by adding new performance-enabling technologies. One of these technologies is called the Java HotSpot™ virtual machine and has the potential to enable the Java programming language to run as fast as compiled C++ code.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.
Mayur Patel (mayurp301@gmail.com) has a non-transferable license to use this Student Guide.

IANT Educom Private Limited

JVM™ Tasks

The following section provides a more comprehensive discussion of the three main tasks performed by the JVM:

- Loads code – Performed by the class loader
- Verifies code – Performed by the bytecode verifier
- Executes code – Performed by the runtime interpreter

The Class Loader

The class loader loads all classes needed for the execution of a program. The class loader adds security by separating the namespaces for the classes of the local file system from those imported from network sources. This limits any Trojan Horse applications, because local classes are always loaded first.

After all of the classes have been loaded, the memory layout of the executable file is determined. At this point, specific memory addresses are assigned to symbolic references and the lookup table is created. Because memory layout occurs at runtime, the Java technology interpreter adds protection against unauthorized access into the restricted areas of code.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Mayur Patel (mayurp391@gmail.com) has a non-transferable license to use this Student Guide.

Copyright 2008 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision G.2

IANT Educom Private Limited

The Bytecode Verifier

Java software code passes several tests before running on your machine. The JVM puts the code through a bytecode verifier that tests the format of code fragments and checks code fragments for illegal code, which is code that forges pointers, violates access rights on objects, or attempts to change object type.

Note – All class files imported across the network pass through the bytecode verifier.

Verification Process

The bytecode verifier makes four passes on the code in a program. It ensures that the code adheres to the JVM specifications and does not violate system integrity. If the verifier completes all four passes without returning an error message, then the following is ensured:

- The classes adhere to the class file format of the JVM specification.
- There are no access restriction violations.
- The code causes no operand stack overflows or underflows.
- The types of parameters for all operational codes are correct.
- No illegal data conversions, such as converting integers to object references, have occurred.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Mayur Patel (mayurp891@gmail.com) has a non-transferable license to use this Student Guide.

Copyright 2008 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision G.2

IANT Educom Private Limited

A Simple Java Application

Like any other programming language, you use the Java programming language to create applications. Code 1-1 and Code 1-2 show a simple Java application that prints a greeting to the world.

Code 1-1 The `TestGreeting.java` Application

```
//
// Sample "Hello World" application
//
public class TestGreeting {
   public static void main (String[] args) {
      Greeting hello = new Greeting();
      hello.greet();
   }
}
```

Code 1-2 The `Greeting.java` Class

```
public class Greeting {
    public void greet() {
       System.out.println("hi");
    }
}
```

Mayur Patel (mayurp391@gmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

IANT Educom Private Limited

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Mayur Patel (mayurp331@gmail.com) has a non-transferable license to use this Student Guide.

The `TestGreeting` Application

Code 1-3 Lines 1–3

```
1   //
2   // Sample "Hello World" application
3   //
```

Lines 1–3 in the program are comment lines.

Code 1-4 Line 4

```
4   public class TestGreeting  {
```

Line 4 declares the class name as `TestGreeting`. A class name specified in a source file creates a *`classname`*`.class` `file` when the source file is being compiled. If you do not specify a target directory for the compiler to use, this class file is in the same directory as the source code. In this case, the compiler creates a file called `TestGreeting.class`. It contains the compiled code for the public class `TestGreeting`.

Code 1-5 Line 5

```
5       public static void main (String args[]) {
```

Line 5 is where the program starts to execute. The Java technology interpreter must find this defined exactly as given or it refuses to run the program.

Other programming languages, notably C and C++, also use the `main()` declaration as the starting point for execution. The various parts of this declaration are briefly described here. The details are covered later in this course.

If the program is given any arguments on its command line, these are passed into the `main()` method in an array of `String` called `args`. In this example, no arguments are used.

The following describes each element of Line 5:

- `public` – The method `main()` can be accessed by anything, including the Java technology interpreter.
- `static` – This keyword tells the compiler that the `main()` method is usable in the context of the class `TestGreeting`. No instance of the class is needed to execute static methods.

Copyright 2008 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision G.2

IANT Educom Private Limited

- `void` – This keyword indicates that the method `main()` does not return any value. This is important because the Java programming language performs careful type-checking to confirm that the methods called return the types with which they were declared.
- `String` *`args[]`* – This method declares the single parameter to the `main` method, `args`, and has the type of a `String` array. When this method is called, the `args` parameter contains the arguments typed on the command line following the class name; for example:

  ```
  java TestGreeting  args[0] args[1] . . .
  ```

Code 1-6 Line 6

```
6       Greeting hello = new Greeting();
```

Line 6 illustrates how to create an object, referred to by the `hello` variable. The `new Greeting` syntax tells the Java technology interpreter to construct a new object of the class `Greeting`.

Code 1-7 Line 7

```
7       hello.greet();
```

Lines 7 demonstrates an object method call. This call tells the `hello` object to `greet` the world. The implementation of this method is shown on Lines 3–5 of the `Greeting.java` file.

Code 1-8 Lines 8–9

```
8     }
9   }
```

Lines 8–9 of the program, the two braces, close the method `main()` and the class `TestGreeting`, respectively.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Mayur Patel (mayurp391@gmail.com) has a non-transferable license to use this Student Guide.

IANT Educom Private Limited

The `Greeting` Class

Code 1-9 Line 1

```
1   public class Greeting {
```

Line 1 declares the `Greeting` class.

Code 1-10 Lines 2–4

```
2   public void greet() {
3       System.out.println("hi");
4   }
```

Lines 2–4 demonstrate the declaration of a method. This method is declared `public`, making it accessible to the `TestGreeting` program. It does not return a value, so `void` is used as the return type.

The `greet` method sends a string message to the standard output stream. The `println()` method is used to write this message to the standard output stream.

Code 1-11 Line 5

```
5   }
```

Line 5 closes the class declaration for `Greeting`.

Mayur Patel (mayurp391@gmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2008 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision G.2

IANT Educom Private Limited

Compiling and Running the `TestGreeting` Program

After you have created the `TestGreeting.java` source file, compile it with the following line:

```
javac TestGreeting.java
```

If the compiler does not return any messages, the new file `TestGreeting.class` is stored in the same directory as the source file, unless specified otherwise. The `Greeting.java` file has been compiled into `Greeting.class`. This is done automatically by the compiler because the `TestGreeting` class uses the `Greeting` class.

To run your `TestGreeting` application, use the Java technology interpreter. The executables for the Java technology tools (`javac`, `java`, `javadoc`, and so on) are located in the `bin` directory.

```
java TestGreeting
```

Note – You must set the `PATH` environment variable to find `java` and `javac`; ensure that it includes *`java_root`*`/bin` (where *`java_root`* represents the directory root where the Java technology software is installed).

Mayur Patel (mayurp391@gmail.com) has a non-transferable license to use this Student Guide

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

IANT Educom Private Limited

Troubleshooting the Compilation

The following sections describe errors that you might encounter when compiling code.

Compile-Time Errors

The following are common errors seen at compile time, with examples of compiler or runtime messages. Your messages can vary depending on which version of the Java 2 SDK you are using.

- `javac: Command not found`

  The `PATH` variable is not set properly to include the `javac` compiler. The `javac` compiler is located in the `bin` directory below the installed Java Development Kit (JDK™) software directory.

-
  ```
  Greeting.java:4:cannot resolve symbol
  symbol   : method printl   (java.lang.String)
  location: class java.io.PrintStream
  System.out.printl("hi");
            ^
  ```

  The method name `println` is typed incorrectly.

- Class and file naming

  If the `.java` file contains a public class, then it must have the same file name as that class. For example, the definition of the class in the previous example is:

  ```
  public class TestGreeting
  ```

  The name of the source file must be `TestGreeting.java`. If you named the file `TestGreet.java`, then you would get the error message:

  ```
  TestGreet.java:4: Public class TestGreeting must be
  defined in a file called "TestGreeting.java".
  ```

- Class count

  You should declare only one top-level, non-static class to be public in each source file, and it must have the same name as the source file. If you have more than one public class, then you will get the same message as in the previous bullet for every public class in the file that does not have the same name as the file.

Mayur Patel (mayur1391@gmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2008 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision G.2

IANT Educom Private Limited

Runtime Errors

Some of the errors generated when typing `java TestGreeting` are:

- `Can't find class TestGreeting`

  Generally, this means that the class name specified on the command line was spelled differently than the *`filename`*`.class` file. The Java programming language is *case-sensitive*.

  For example,

  ```
  public class TestGreet {
  ```

  creates a `TestGreet.class`, which is not the class name (`TestGreeting.class`) that the compiler expected.

- `Exception in thread "main" java.lang.NoSuchMethodError: main`

  This means that the class you told the interpreter to execute does not have a static `main` method. There might be a main method, but it might not be declared with the static keyword or it might have the wrong parameters declared, such as:

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

  In this example, `args` is a single string, not an array of strings.

  ```
  public static void main() {
  ```

  In this example, the coder forgot to include any parameter list.

Mayur Patel (mayurp391@gmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

IANT Educom Private Limited

Figure 1-3 illustrates how Java technology programs can be compiled and then run on the JVM. There are many implementations of the JVM on different hardware and operating system platforms.

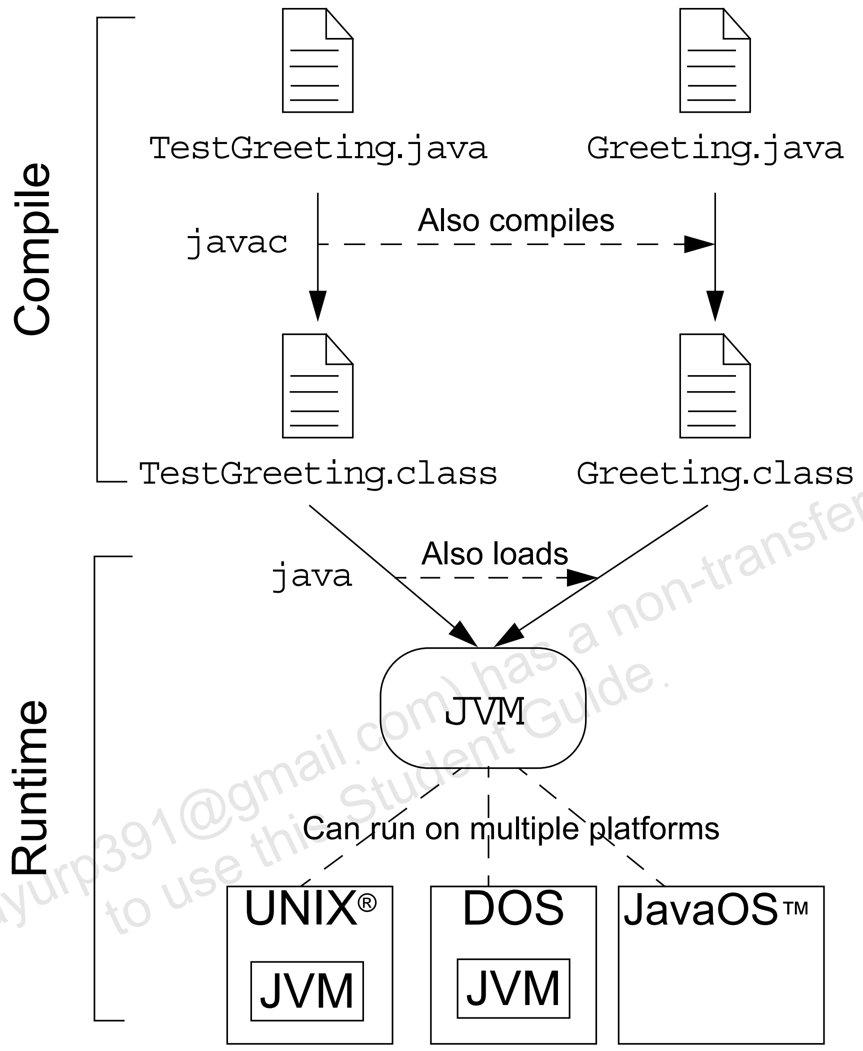

Figure 1-3 Java Technology Runtime Environment

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Mayur Patel (mayurp391@gmail.com) has a non-transferable license to use this Student Guide.

Copyright 2008 Sun Microsystems, Inc. All Rights Reserved. Sun Services, Revision G.2

IANT Educom Private Limited

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Mayur Patel (mayurp391@gmail.com) has a non-transferable license to use this Student Guide.