What Is a Java Program?

2

ORACLE

Objectives

After completing this lesson, you should be able to:

- Contrast the terms “platform-dependent” and “platform-independent”
- Describe the purpose of the JVM
- Explain the difference between a procedural program and an object-oriented program
- Describe the purpose of `javac` and `java` executables
- Verify the Java version on your system
- Compile and run a Java program from the command line

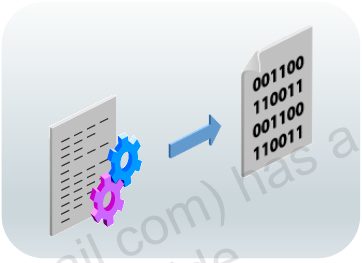
Topics

- Introduction to computer programs
- Introduction to the Java language
- Verifying the Java development environments
- Running and testing a Java program

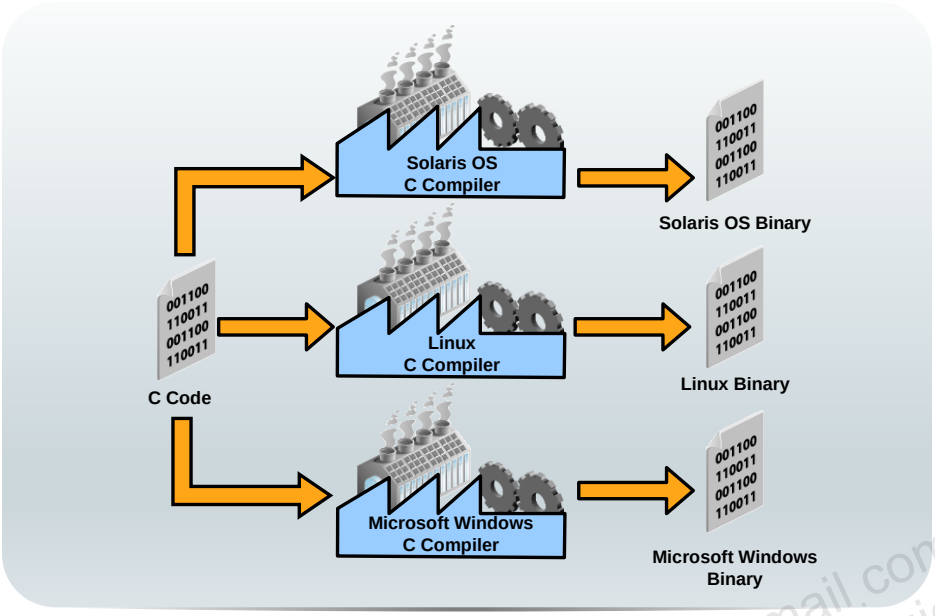
Purpose of a Computer Program

A computer program is a set of instructions that run on a computer or other digital device.

- At the machine level, the program consists of binary instructions (1s and 0s).
  - Machine code
- Most programs are written in *high-level* code (readable).
  - Must be translated to machine code



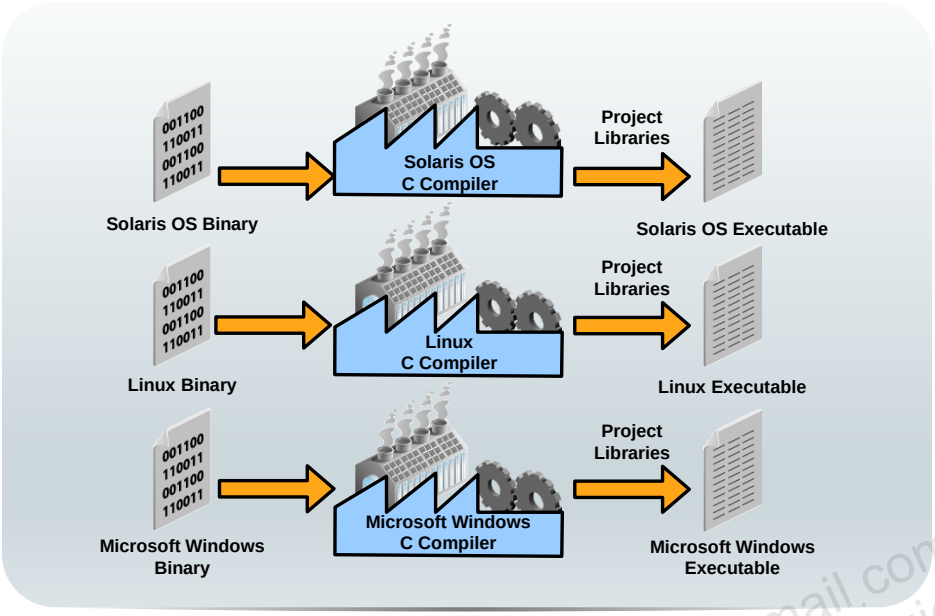
Translating High-Level Code to Machine Code



Programs written in most languages usually require numerous modifications to run on more than one type of computing platform (a combination of a CPU and operating system). This platform-dependence is because most languages require you to write a code specific to the underlying platform. Popular programming languages, such as C and C++, require programmers to compile and link their programs, resulting in an executable program unique to a platform. A compiler is an application that converts a program that you write into a CPU-specific code called *machine code*. These platform-specific files (binary files) are often combined with other files, such as libraries of prewritten code, using a linker to create a platform-dependent program, called an *executable*, which can be executed by an end user. Unlike C and C++, the Java programming language is platform-independent.

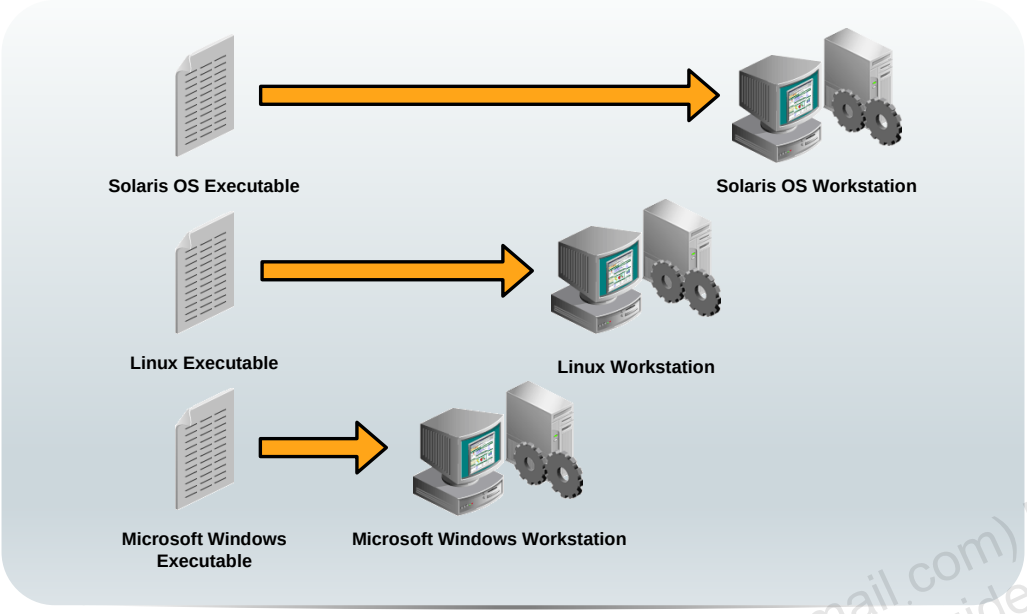
The image illustrates how a compiler creates a binary file.

Linked to Platform-Specific Libraries



The image illustrates how a binary file is linked with libraries to create a platform-dependent executable.

Platform-Dependent Programs



The image illustrates how platform-dependent executables can execute only on one platform.

Topics

- Introduction to computer programs
- Introduction to the Java language
- Verifying the Java development environment
- Running and testing a Java program

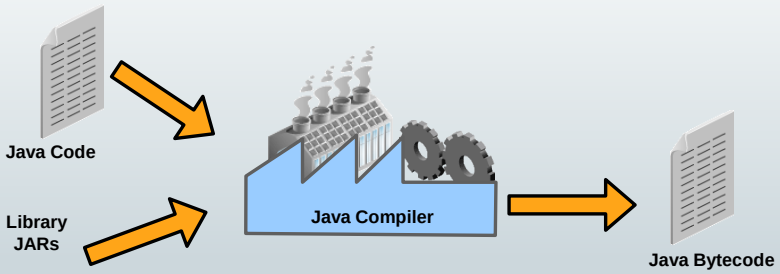
Key Features of the Java Language

Some of the features that set Java apart from most other languages are that:

- It is platform-independent
- It is object-oriented

There are several other key features of the Java language, but in this course, only the two mentioned above will be discussed.

Java Is Platform-Independent

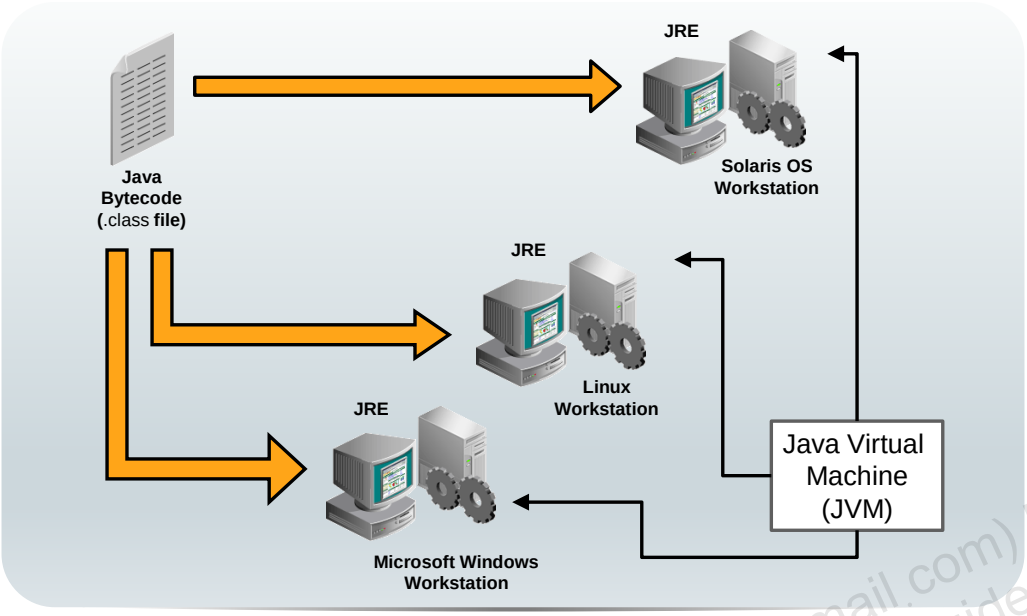


A Java program can run on several different CPUs and operating system combinations, such as the Solaris OS on a SPARC chip, Mac OS X on an Intel chip, and Microsoft Windows on an Intel chip, usually with few or no modifications.

As illustrated above, Java programs are compiled using a Java compiler. The resulting format of a compiled Java program is platform-independent Java bytecode instead of CPU-specific machine code.

After the bytecode is created, it is interpreted by a bytecode interpreter called the Java Virtual Machine or JVM. A virtual machine is a platform-specific program that understands platform-independent bytecode and can execute it on a particular platform. For this reason, the Java programming language is often referred to as an interpreted language, and Java technology programs are said to be portable or executable on any platform. Other interpreted languages include Perl.

Java Programs Run In a Java Virtual Machine



The image illustrates a Java bytecode file executing on several platforms where a Java runtime environment exists.

A virtual machine gets its name because it is a piece of software that runs code, a task usually accomplished by the CPU or hardware machine. For Java programs to be platform-independent, a virtual machine called the JVM is required on every platform where your program will run. The JVM is responsible for interpreting Java code, loading Java classes, and executing Java programs.

However, a Java program needs more than just a JVM to execute. A Java program also needs a set of standard Java class libraries for the platform. Java class libraries are libraries of prewritten code that can be combined with the code that you write to create robust applications.

Combined, the JVM software and Java class libraries are referred to as the Java Runtime Environment (JRE). Java Runtime Environments are available from Oracle for many common platforms.

Procedural Programming Languages

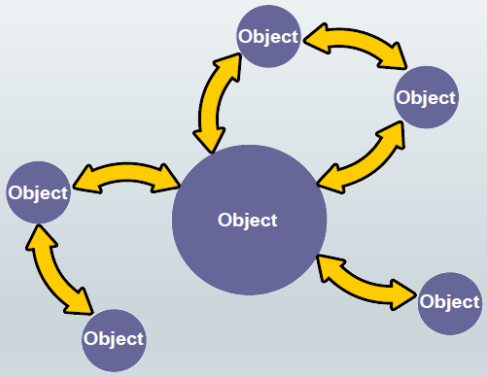
- Many early programming languages followed a paradigm called *Procedural Programming.*
- These languages use a sequential pattern of program execution.
- Drawbacks to procedural programming:
  - Difficult to translate real-world use cases to a sequential pattern
  - Difficult to maintain programs
  - Difficult to enhance as needed

Earlier programming languages were based on a programming paradigm called “procedural.” Procedural languages use a sequential pattern of program execution such as you see in the diagram above. Control is passed between procedures, where each procedure contains a series of steps to execute. Some examples of procedural programming languages are COBOL, Fortran, C, and Pascal.

This style of programming has become less popular due to the difficulty of designing real-world applications using sequential pattern. It has also proven difficult to maintain and enhance programs structured in this way.

Java Is an Object-Oriented Language

- Interaction of objects
- No prescribed sequence
- Benefits:
  - Modularity
  - Information hiding
  - Code reuse
  - Maintainability



Object-oriented programming differs from procedural programming, because procedural programming stresses the sequence of coding steps required to solve a problem, whereas object-oriented programming stresses the interaction of objects. Java is an object-oriented programming (OO) language. One of the main goals of an OO language is to create objects—pieces of autonomous code—that can interact with other objects to solve a problem. OO programming languages began in 1967 and have led to popular programming languages such as C++, upon which Java is loosely based.

This provides many benefits:

- **Modularity:** The source code for an object can be written and maintained independently of the source code for other objects. After it is created, an object can be easily passed around inside the system.
- **Information hiding:** By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.
- **Code reuse:** If an object already exists (perhaps written by another software developer), you can use that object in your program.
- **Maintainability:** If a particular object is found to be problematic, you can create another, slightly modified one and simply replace the original one in your application. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, you replace the bolt, not the entire machine.

The diagram illustrates an object-oriented program's focus on objects and object interactions.

Topics

- Introduction to computer programs
- Introduction to the Java language
- Verifying the Java development environment
- Running and testing a Java program

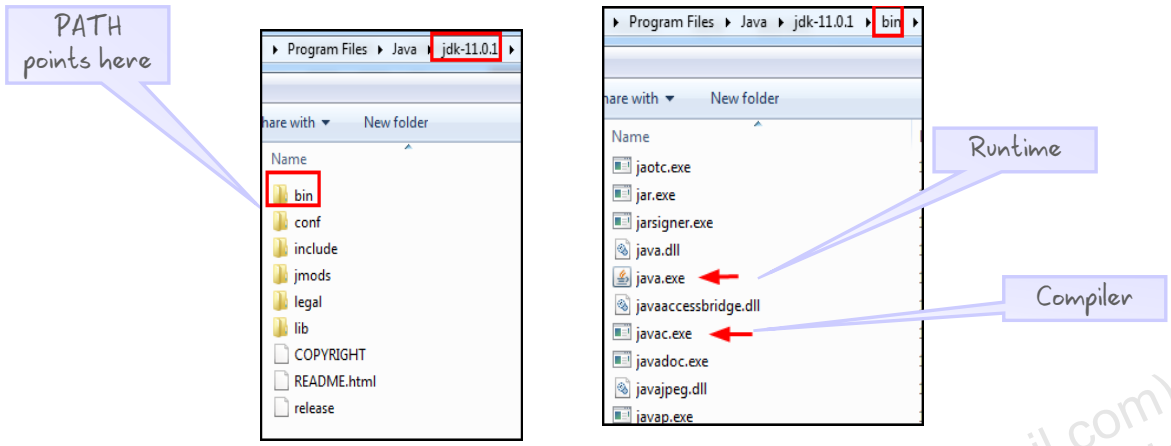
Verifying the Java Development Environment

1. Download and install the Java Development Kit (JDK) from oracle.com/java.
2. Examine the environment.
3. Compile and run a Java application by using the command line.

Setting up your Java development environment is a simple task. The JDK is available for free from the Oracle Java website.

- After you have installed the JDK, you can explore the Java environment by typing some commands at the command line. For example, open a terminal window and enter `java`.
- Review the command options displayed.
- Enter `java -version` to see what Java version is installed on your system.
- Examine the enviornment
- Compile and run a Java application using the command line.

Examining the Installed JDK: The Tools



Java SE (Standard Edition) Development Kit

The Java SE Development Kit includes both the tools and classes that you will use to develop a Java program. The tools and utilities are stored in the `bin` directory. These are shown in the screenshot on the right. They include:

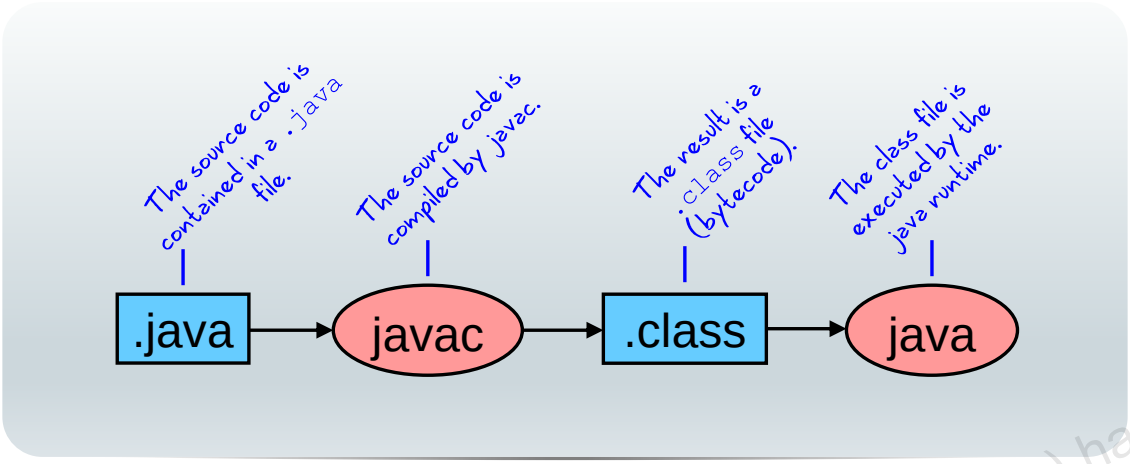
- A Java Virtual Machine (JVM) for the platform you choose. Here you see a Windows example. The runtime engine is started by running the `java` program.
- A Java compiler, started by running the `javac` program
- Additional utilities, such as utilities for creating Java archive files (`JAR` files) and for debugging Java programs
- The `bin` directory, which must be on the system PATH in order to run or compile a Java program. The Java installer automatically adds the bin to your system PATH.

Note: The Java Runtime Environment used in *production* (commonly called the JRE) is also included with Java SE Development Kit. This is found in the `jre` directory.

Topics

- Introduction to computer programs
- Introduction to the Java language
- Verifying the Java development environment
- Running and testing a Java program

Compiling and Running a Java Program



The above diagram shows what happens when you compile and run a Java program.

- The Java code is written in a file with the extension `.java`. This is called the "Java source code."
- You use the javac executable to compile the source code (the "c" stands for "compiler") into a bytecode file with the extension `.class`. This is called a Java class.
- You use the `java` executable to run the Java class. This is your Java program.

Compiling a Program

1. Go to the directory where the source code files are stored.
2. Enter the following command for each `.java` file you want to compile.

- Syntax:

```
javac <source file>
```

- Example:

```
javac SayHello.java
```

Compiling converts the source files that you write into bytecode that can be executed by a Java Virtual Machine. The source file has a `.java` extension. It also defines a public class of the same name. For example, the class, `SayHello`, must be saved in a file called `SayHello.java`. (You will learn more about classes later in this course.)

To compile the `SayHello` source code, perform the following steps:

1. Go to the directory where the source code files are stored.
2. Enter the following command for each `.java` file that you want to compile (note that the `.java` extension is required):

   Example: `javac SayHello.java`

After the compilation has finished, and assuming no compilation errors have occurred, you should have a new file called `<classname>.class` in your directory for each source code file that you compiled.

Example: `SayHello.class`

Executing (Testing) a Program

1. Go to the directory where the class files are stored.
2. Enter the following for the class file that contains the main method:

- Syntax:

```
java <classname>
```

- Example:

Do not specify .class.

```
java SayHello
```

- Output:

```
Hello World!
```

When you have successfully compiled your source code files, you can execute and test them using the Java Virtual Machine.

To execute and test your program:

1. Go to the directory where the class files are stored.
2. Enter the following command for the class file that contains the `main` method. Note that here the file extension (`.class`) should *not* be included.

   Example: `java SayHello`

This command runs the `SayHello` class. The `SayHello` class contains the `main` method. This is the entry point to a Java application. The `java` executable only works with a class containing a main method. In the above example, the main method contains code that prints the string "Hello World!"

Output for a Java Program

A Java program can output data in many ways. Here are some examples:

- To a file or database
- To the console
- To a webpage or other user interface

In this course, we will be outputting data only to the console. You can learn more about writing to other destinations, such as a file, database, or webpage, by taking the *Java SE 8 Programming* course.

Exercise 2-1

- From a Terminal window, enter `java -version` to see the system's Java version.
- Look for `SayHello.java` in:
  `/labs/02-GettingStarted/Exercises/Exercise1`
- Compile it: `javac SayHello.java`
- Run the resulting class file: `java SayHello`
  - Did you see the output?

In this exercise, you look at the Java version installed on your system, and then you run a simple Java program from the command line.

- Open a terminal window by double-clicking the Terminal shortcut on your desktop. It will open at your home directory, which is `/home/oracle`.
  - **Note:** A handy shortcut to navigate to your home directory from anywhere is **~**. Example: `cd ~` to go to `/home/oracle`.
- Enter `java` to see the available command options.
- Enter `java -version` to verify that JDK 11 is installed on your system.
- Navigate to the folder containing the Java source file for this exercise:
  `cd labs/02-GettingStarted/Exercises/Exercise1`
- Enter `javac SayHello.java` to compile it.
- Enter `java SayHello` to run it. You should see a "Hello World!" message as output.

JDK 11: Launch Single-File Source-Code Programs

Benefits:

- Skip the compilation "ceremony".
- Run a program with one quick command:

  java *<source file>*

Requirements:

- Write the entire program as single source file.
- The file may contain any number of classes.
- The top-most class declares a `main` method.

Use Cases:

- Experiment quickly to learn Java.
- Write small utility or "shebang" files.

Circle.java

```
public class Test {
  public static void main(String args[]) {
    double area = Circle.findArea(7.5);
    System.out.print("Area of circle=" +area);
  }
}

public class Circle {
  public static double findArea(double radius){
    return Math.PI * radius * radius;
  }
}
```

```
java Circle.java
Area of circle=176.714…
```

If the order of the classes changes, the program will not execute.

Note on shebang files: Single-file programs are common when the task at hand needs a small utility program. In this context, it is desirable to be able to run a program directly from source using the "#!" mechanism on Unix-derived systems, such as macOS and Linux. This is a mechanism provided by the operating system which allows a single-file program (such as a script or source code) to be placed in any conveniently named executable file whose first line begins with #! and which specifies the name of a program to "execute" the contents of the "shebang" file.

Exercise 2-2

- In a terminal window:
- Look for `Circle.java` in:
  `/labs/02-GettingStarted/Exercises/Exercise2`
- Run the file: `java Circle.java`
- Did you see the output?
- Do you see any bytecode file produced?

In this exercise, you test how to launch a single-file source-code programs.

- Navigate to the folder containing the Java source file for this exercise:
  `cd labs/02-GettingStarted/Exercises/Exercise2`
- Enter `java Circle.java` to run it.
- You should see an output message and no bytecode file. The effect of launch a single-file source-code programs is as if the source file is compiled into memory.

Quiz

Which of the following is correct? (Choose all that apply.)

a. `javac OrderClass`
b. `java OrderClass`
c. `javac OrderClass.java`
d. `java OrderClass.java`

Answer: b, c, d

Summary

In this lesson, you should have learned how to:

- Describe the distinction between high-level language and machine code
- Describe what platform-independence means
- Describe how a Java program is compiled and to what format
- Explain what it means to say that Java is an object-oriented language
- Determine the version number of a Java install
- Compile and run a Java program from the command line