Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/JDK and First Java Program/5a9e9f444f99451ac0377ce6

TUTORIAL

JDK and First Java Program

Topics

Java Development Kit

Before you can compile and run those programs, however, you must have a Java development system installed on your computer. One is the standard JDK (Java Development Kit), which is available from Oracle. The JDK can be downloaded free of charge from http://www.oracle.com/technetwork/java/index.html. Just go to the download page and follow the instructions for the type of computer that you have. After you have installed the JDK, you will be ready to compile and run programs. The JDK supplies two primary programs. The first is javac.exe, which is the Java compiler. The second is java.exe, which is the standard Java interpreter, and is also referred to as the application launcher. One other point: the JDK runs in the command prompt environment. It is not a windowed application. However you can install other applications like Eclipse, Netbeans which provides the GUI environment for most of programming languages.

The commands are javac to compile and java to run your program. For example:

```
code@cq:~$ javac HelloWorld.java
code@cq:~$ java HelloWorld
Hello, World
code@cq:~$
```

First Java Program

Let's start by compiling and running the short sample program shown here.

```java
/* This is a simple Java program.
Save this file as Main.java. */
class Main
{
  // A Java program begins with a call to main().
  public static void main(String args[])
  {
    System.out.println("Code Quotient - Get Better at coding.");
  }
}
```

After saving the above file as Main.java you have to compile and run it as follows: -

To compile the Example program, execute the compiler, javac, specifying the name of the source file on the command line, as shown here:

```
code@cq:~$ javac filename.java
  Where filename.java is the name of file in which you stored the program.

code@cq:~$ javac Main.java
```

The javac compiler creates a file called **Main.class** that contains the bytecode version of the program. Remember, bytecode is not executable code. Bytecode must be executed by a Java Virtual Machine (JVM). Thus, the output of javac is not code that can be directly executed. To actually run the program, you must use the Java interpreter i.e. java. To do so, pass the class name Example as a command-line argument, as shown here:

```
code@cq:~$ java classname
  Where classname is the name of class which you want the
```

```
interpreter to run.

code@cq:~$ java Main
```

When the program is run, the following output is displayed:

```
Code Quotient - Get Better at coding.
```

When Java source code is compiled, each individual class is put into its own output file named after the class and using the .class extension. This is why it is a good idea to give your Java source files the same name as the class they contain—the name of the source file will match the name of the .class file. When you execute the Java interpreter as just shown, you are actually specifying the name of the class that you want the interpreter to execute. It will automatically search for a file by that name that has the .class extension. If it finds the file, it will execute the code contained in the specified class.

Let’s closely examine each part of the program.

The program begins with the following lines:

```
/* This is a simple Java program.
Save this file as Main.java. */
```

This is a comment. **Like** most other programming languages, **Java** lets you enter a remark **into** a program’s **source** file. The **contents of** a **comment are** ignored **by** the compiler. Instead, a **comment** describes **or** explains the operation **of** the program **to** anyone who **is** reading its **source** code. **In** this **case**, the **comment** describes the program **and** reminds you that the **source file** should be called **Main**.java. Of course, in real applications, comments generally explain how some part of the program works or what a specific feature does.

Java supports three styles of comments. The one shown at the top of the program is called a multiline comment. This type of comment must begin with /* and end with */. Anything between these two

comment symbols is ignored by the compiler. As the name suggests, a multiline comment may be several lines long.

The next line of code in the program is shown here:

```
class Main
{
```

This line uses the keyword class to declare that a new class is being defined. **Main** is the name of the class. The class definition begins with the opening curly brace ({) and ends with the closing curly brace (}).

The elements between the two braces are members of the class. For the moment, don't worry too much about the details of a class except to note that in Java, all program activity occurs within class. This is one reason why all Java programs are (at least a little bit) object-oriented. The next line in the program is the single-line comment, shown here:

```
// A Java program begins with a call to main().
```

This is the second type of comment supported by Java. A single-line comment begins with a // and ends at the end of the line. As a general rule, programmers use multiline comments for longer remarks and single-line comments for brief, line-by-line descriptions.

The next line of code is shown here:

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

This line begins the main( ) method. As the comment preceding it suggests, this is the line at which the program will begin executing. All Java applications begin execution by calling main(). We will examine each word written in this line later after knowing the concepts and will explore this line later.

All of the code included in a method will occur between the method's opening curly brace and its closing curly brace.

The next line of code is shown here. Notice that it occurs inside main( ).

```
System.out.println("Code Quotient - Get Better at coding.");
```

This line outputs the string "*Code Quotient - Get Better at coding.*" followed by a new line on the screen. Output is actually accomplished by the built-in println( ) method. In this case, println( ) displays the string which is passed to it. As you will see, println( ) can be used to display other types of information, too. The line begins with System.out. While too complicated to explain in detail at this time, briefly, System is a predefined class that provides access to the system, and out is the output stream that is connected to the console. Thus, System.out is an object that encapsulates console output. The fact that Java uses an object to define console output is further evidence of its object-oriented nature.

Notice that the println( ) statement ends with a semicolon. All statements in Java end with a semicolon.

The first } in the program ends main( ), and the last } ends the **Main** class definition.

Remember Java is case sensitive. Forgetting this can cause you serious problems. For example, if you accidentally type Main instead of main, or Println instead of println, the preceding program will be incorrect. Furthermore, although the Java compiler will compile classes that do not contain a main( ) method, it has no way to execute them. So, if you had mistyped main, the compiler would still compile your program. However, the Java interpreter would report an error because it would be unable to find the main( ) method with required signature.

Phases of a JAVA program

Phase 1: Program is created in the editor and stored on disk.

Phase 2: Compiler creates bytecodes and stores them on disk.

Phase 3: Class loader puts bytecodes in memory.

Phase 4: Bytecode verifier confirms that all bytecodes are valid and do not violate Java's security restrictions.

Phase 5: Interpreter reads bytecodes and translates them into a language that the computer can understand, possibly storing data values as the program executes.

Phase 1 consists of editing a file. This is accomplished with an editor program (normally known as an editor). The programmer types a Java program, using the editor, and makes corrections, if necessary. When the programmer specifies that the file in the editor should be saved, the program is stored on a secondary storage device, such as a disk. Java program file names end with the .java extension.

Phase 2 is to compile the program. the programmer gives the command javac to compile the program. The Java compiler translates the Java program into byte codes—the language understood by the Java interpreter.

Phase 3 is called loading. The program must first be placed in memory before it can be executed. This is done by the class loader, which takes the .class file (or files) containing the bytecodes and transfers it to memory. The .class file can be loaded from a disk on your system or over a network (such as your local university or company network or even the Internet). There are two types of programs for which the class loader loads .class files— applications and applets. These will be discussed in later sections.

In Phase 4 bytecode verifier takes place. Before the program is executed by the programs the bytecodes are verified by the bytecode verifier. This ensures that the bytecodes for classes that are loaded from the Internet (referred to as downloaded classes) are

valid and that they do not violate Java's security restrictions. Java enforces strong security, because Java programs arriving over the network should not be able to cause damage to your files and your system (as computer viruses might). Note that bytecode verification also occurs in applications that download classes from a network.

Finally, in Phase 5, the computer, under the control of its CPU, interprets the program one bytecode at a time, thus performing the actions specified by the program.