



## Getting started with Java language

---

The **Java Development Kit (JDK)** is a product of Sun Microsystems which is used to **develop** the Java software and is an extended subset of a Java **software development kit (SDK)**. JDK consists of the **API classes**, a **Java compiler**, and the **Java Virtual Machine interpreter**. JDK is used to compile Java applications and applets. It includes tools for developing Java applets and applications.

## Objectives

---

Upon the completion of this topic, we will be able to:

- Get started with Java programming language
- Understand various binary files involved in JDK
- Write a Java program, compile and run the same

## Java Development Kit (JDK)

---

A Java software development kit (SDK or "devkit") is a set of development tools that is used in developing applications for certain software package, software framework, hardware platform, computer system, video game console, operating system, or similar a platform. If J2SE 1.2 is used to develop applications, then you are using a platform known as the Java 2 Platform. The latest Java 5 or J2SE 1.5 is Sun's latest version that has added several enhancements to the Java language.

There are several programming tools and packages in JDK which includes the Java programming language core functionality, the Java Application Programming Interface (**API**) with multiple package sets and essential tools for developing Java programs.

The packages available in JDK are: **java.applet**, **java.awt**, **java.awt.image**, **java.awt.peer**, **java.io**, **java.lang**, **java.net** and **java.util**. These packages provide everything that is required to start creating powerful Java applications. The JDK also includes an additional package called **sun.tools.debug**, which is designed to make the application-debugging process easier.

Programming Tools available in the current version of JDK are used to create Java bytecode, view programs, debug code and are used for security purpose. Tools available in the current version of JDK are given in the table below.



### Tools in the JDK

<i>Executable</i>	<i>Tool Name</i>	<i>Description</i>
Appletviewer	The Java applet viewer	Used to view applets without a Web browser
Java	The Java interpreter	Runs Java bytecode
Javac	The Java compiler	Compiles Java programs into bytecode
Javadoc	The Java API documentation generator	Creates API documentation in HTML format from Java source code
Javah	The Java header and stub file generator	Creates C-language header and stub files from a Java class, which allows the Java and C code to interact
Javap	The Java class file disassemble	Disassembles Java files and prints out a representation of Java bytecode
Jdb	The Java language debugger	Helps to find and fix problems in Java code

There are some advanced commands also for debugging the programs that are available in JDK. They are:

- **up** – moves up the stack frame so that locals and print can be used to examine the program at the point before the current method was called
- **down** – moves down the stack frame to examine the program after the method call

Following commands can also be used while debugging:

- **classes** – lists the classes currently loaded into memory
- **methods** – lists the methods of a class
- **memory** – lists the total memory and the amount that is not currently in use
- **threads** – lists the threads that are executing
- **suspend thread** – suspends threads (by default all are suspended)
- **resume thread** – resume threads (by default all)
- **where** – dumps a thread's stack
- **threadgroups** – lists the threadgroups
- **print** – prints an object
- **locals** – print all the local variables in the current stack frame
- **dump** – print all the object information
- **cont** – continue execution from breakpoint



- **catch <class id>** - break for the specified exception
- **ignore <class id>** - ignore when the specified exception
- **gc** – free unused objects
- **load classname** – load Java class to be debugged
- **run <class> [args]** – start execution of a loaded Java class
- **!!** – repeat last command
- **help (or ?)** – list commands
- **exit (or quit)** – exit debugger

Along with the programming tools mentioned above there are three tools used for security purpose in the JDK. They are **Keytool**, **Jar** and **Jarsigner**.

### 1. *Keytool*

Keytool, the Java security key tool, is used to create and manage public keys, private keys and security certificates. It can be used to do the following:

- Manage their own public key / private key pairs
- Store the public keys of people and groups which it communicates with
- Use the certificates associated with these keys to authenticate the user to others
- Authenticate the source and integrity of data

### 2. *The jar Archival Tool and jarsigner*

The Java archival tool **jar** is used to package an archive file, a Java program and all the resource files that it requires.

One of the advantages of the **jar** tool is that it speeds up the loading time for an applet on the web, especially when the applet needs a group of other files in order to function. Jar is necessary to create a digitally signed Java program. All the class files which are needed to run a particular Java program need to be packaged into an archive jar file before being signed. When a jar file is included in the program it should be signed using a **jarsigner**.

The following two activities should be performed:

- Digitally signing a java archive file
- Verifying the digital signature and contents of a java archive

To digitally sign a java archive file, the following should be specified, i.e. a name of the jar archive and the name of the private key to sign it with. The **jarsigner** tool has several command-line options that are identical to those offered by keytool, including **-keystore**. The keystore is used to specify the folder and the file location of a key store.



The jar tool can also be used to verify a digitally signed archive by adding the **-verify** option to the command. If the private key matches the Java archive and the archives' contents did not change after it was signed, it will be verified by the jarsigner.

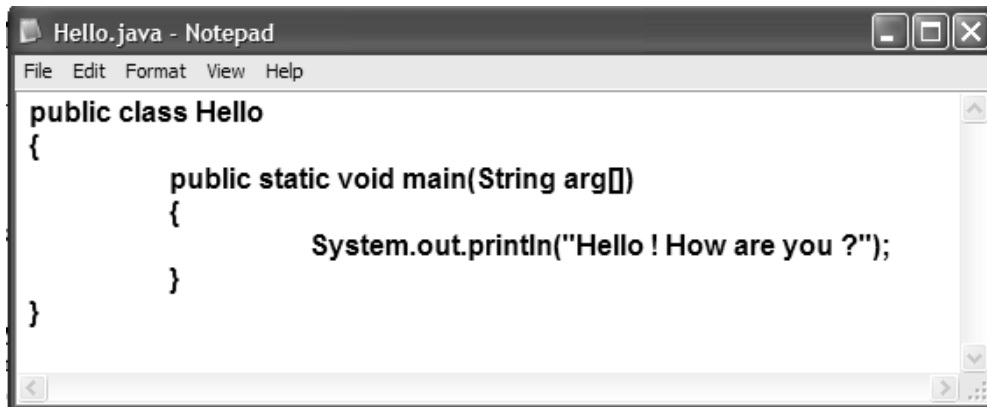
## Working of Java

---

A compiler converts the Java program into an intermediate language representation called **bytecode** which is platform independent. A Java file will have the extension **.java**, similar to a word file having the extension .doc, a Pascal file having the extension .pas and a text file having the extension .txt.

Let us assume that there exists a Java file named **Hello.java**. When this file is compiled, we get a file called **Hello.class**.

This class file is run using an interpreter as and when necessary. The figure below shows the java program saved as Hello.java.



```
public class Hello
{
    public static void main(String arg[])
    {
        System.out.println("Hello ! How are you ?");
    }
}
```

**A Sample Java Program**



The steps for compiling and running the program are shown in figure below. The program is stored in a subdirectory called java. When you run the above program, it prints the message “Hello! How are you?”

```

C:\WINDOWS\system32\cmd.exe

C:\java>javac Hello.java

C:\java>dir
Volume in drive C is ACADEMIC
Volume Serial Number is 4C19-52B1

Directory of C:\java

06/07/2007  12:44 AM    <DIR>        .
06/07/2007  12:44 AM    <DIR>        ..
06/07/2007  12:46 AM                159 Hello.java
06/07/2007  12:47 AM                425 Hello.class
               2 File(s)                584 bytes
               2 Dir(s)  6,100,946,944 bytes free

C:\java>java Hello
Hello ! How are you ?

C:\java>
```

### Compiling and Executing

The concept of “write once, run anywhere” is possible in Java. The Java program can be compiled on any platform having a Java compiler. The resulting *bytecode* can then be run on Window NT or Solaris or Macintosh or any other machine. The machine should have a Java platform to run a Java code. The Java platform consists of the Java Virtual Machine (JVM) and a package of readymade software components. This package is known as **Java Application Programming Interface (Java API)**. The compiled Java program can run on any hardware platform having a Java Virtual Machine (JVM) installed on it.

## Summary

Here are the key takeaways:

- A compiler converts the Java program into an intermediate language representation called **bytecode** which is platform independent.
- The concept of “write once, run anywhere” is possible in Java.
- The compiled Java program can run on any hardware platform having a Java Virtual Machine (JVM) installed on it.