**Why Java?**

Object Oriented

Platform Independent

Write Once, Run Anywhere.

**Buzz Words**

* Simple
* Secure
* Portable
* Object-oriented
* Robust
* Multithreaded
* Architecture-neutral
* Interpreted
* High performance
* Distributed
* Dynamic

**Simple**

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:

* Java syntax is based on C++ (so easier for programmers to learn it after C++).
* Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
* There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

**Object-oriented**

Java is an object-oriented

programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.

Basic concepts of OOPs are:

* Object
* Class
* Inheritance
* Polymorphism
* Abstraction
* Encapsulation

**Platform Independent**



**Secured**

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

* **No explicit pointer**
* **Java Programs run inside a virtual machine sandbox**

**Simple Java Program**

public class MyFirstJavaProgram {

public static void main(String []args) {

System.out.println("Hello World"); // prints Hello World

}

}

Java program processing starts from the main() method which is a mandatory part of every Java program.

Running the programs:

javac MyFirstJavaProgram.java

java MyFirstJavaProgram

The javac command reads source files that contain module, package and type declarations written in the Java programming language, and compiles them into class files that run on the Java Virtual Machine

The java command starts a Java application. It does this by starting a Java runtime environment, loading a specified class, and calling that class's main method. By default, the first argument without an option is the name of the class to be called.

**Internal Details of the Program**

What happens at compile time?

At compile time, the Java file is compiled by Java Compiler (It does not interact with OS) and converts the Java code into bytecode.



**What happens at runtime?**

At runtime, the following steps are performed:



**Classloader:** It is the subsystem of JVM that is used to load class files.

**Bytecode Verifier:** Checks the code fragments for illegal code that can violate access rights to objects.

**Interpreter:** Read bytecode stream then execute the instructions.

**Can you save a Java source file by another name than the class name?**

Yes, if the class is not public. It is explained in the figure given below:



|  |  |
| --- | --- |
| **To compile:** | javac Hard.java |
| **To execute:** | java Simple |

Observe that, we have compiled the code with file name but running the program with class name. Therefore, we can save a Java program other than class name.

**How to set path in Java**

The path is required to be set for using tools such as javac, java, etc.

If you are saving the Java source file inside the JDK/bin directory, the path is not required to be set because all the tools will be available in the current directory.

However, if you have your Java file outside the JDK/bin folder, it is necessary to set the path of JDK.

Set Java path : <https://www.javatpoint.com/how-to-set-path-in-java>

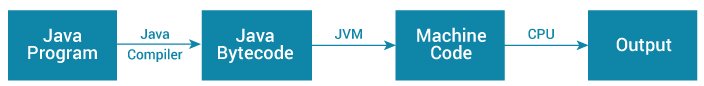
**Java JDK, JRE and JVM**

**What is JVM?**

JVM (Java Virtual Machine) is an abstract machine that enables your computer to run a Java program.

When you run the Java program, Java compiler first compiles your Java code to bytecode. Then, the JVM translates bytecode into native machine code (set of instructions that a computer's CPU executes directly).

Java is a platform-independent language. It's because when you write Java code, it's ultimately written for JVM but not your physical machine (computer). Since JVM ​executes the Java bytecode which is platform-independent, Java is platform-independent.



**Additional Learning on Java Architecture :** <https://dzone.com/articles/jvm-architecture-explained>

**What is JRE?**

JRE (Java Runtime Environment) is a software package that provides Java class libraries, Java Virtual Machine (JVM), and other components that are required to run Java applications.

JRE is the superset of JVM.



If we need to run Java programs, but not develop them, only JRE is needed.

**What is JDK?**

JDK (Java Development Kit) is a software development kit required to develop applications in Java. When you download JDK, JRE is also downloaded with it.

In addition to JRE, JDK also contains a number of development tools (compilers, JavaDoc, Java Debugger, etc).



**Relationship between JVM, JRE, and JDK.**



**The Primitive Types**

Java defines eight *primitive* types of data: **byte**, **short**, **int**, **long**, **char**, **float**, **double**, and **boolean**. The primitive types are also commonly referred to as *simple* types, and both terms will be used in this book. These can be put in four groups:

• Integers This group includes **byte**, **short**, **int**, and **long**, which are for whole-valued signed numbers.

• Floating-point numbers This group includes **float** and **double**, which represent numbers with fractional precision.

* Characters This group includes **char**, which represents symbols in a character set, like letters and numbers.
* Boolean This group includes **boolean**, which is a special type for representing true/false values.

You can use these types as-is, or to construct arrays or your own class types. Thus, they form the basis for all other types of data that you can create.

1. The width and ranges of these integer types vary widely, as shown in this table:

page67image56291328

|  |  |  |
| --- | --- | --- |
| **Name** | **Width** | **Range** |
| **long** | 64 | –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| **int** | 32 | –2,147,483,648 to 2,147,483,647 |
| **shor t** | 16 | –32,768 to 32,767 |
| **byte** | 8 | –128 to 127 |

**Primitive Data Types**

byte - 8-bit signed

Range: -128 to 127 (-2^7 to 2^7-1)

Eg: byte a = -50

1 1 1 1 1 1 1 1

(64)(32)(16)(8)4)(2)(1)

short - 16-bit signed

Range: -32,768 to 32,767 (-2^15 to 2^15-1)

Eg: short a = 10000

int - 32-bit signed

Range: -2,147,483,648 to 2,147,483,647 (-2^31 to 2^31-1)

Eg: int height = 100000;

long - 64-bit signed

Range: (-2^63 to 2^63-1)

Eg: int height = 100000L;

float - single-precision floating point

Eg: float f1 = 234.5f

double - double-precision floating point

Range: (-2^64 to 2^64-1)

Eg: int height = 12.34;

boolean - represents one bit of information

Eg: boolean flag = true;

char - 16-bit Unicode character

Range: (-2^64 to 2^64-1)

Eg: char initial = ‘A’;

Variables and declarations

Some examples:

int a, b, c; // Declares three ints, a, b, and c.

int a = 10, b = 10; // Example of initialization

byte B = 22; // initializes a byte type variable B.

double pi = 3.14159; // declares and assigns a value of PI.

Char a = ‘a’; // the char variable a iis initialized with value ‘a’

**Using Blocks of Code**

if(x < y) { // begin a block

x = y;

y = 0;

} // end of block

**Lexical Issues**

**Whitespace**

Java is a free-form language. This means that you do not need to follow any special indentation rules. For instance, the **Example** program could have been written all on one line or in any other strange way you felt like typing it, as long as there was at least one whitespace character between each token that was not already delineated by an operator or separator. In Java, whitespace is a space, tab, or newline.

**Identifiers**

Some examples of valid identifiers are

page63image22615872

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AvgTemp | count | a4 | $test | this\_is\_ok |

Some examples of invvalid identifiers are

page63image22615872

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2AvgTemp | Count/1 | a-4 | 2$test | Thisisnot/ok |

**Comments**

// or /\* \*/

**Java Keywords**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| abstract | continue | for | new | switch |
| assert | default | goto | package | synchronized |
| boolean | do | if | private | this |
| break | double | implements | protected | throw |
| byte | else | impor t | public | throws |
| case | enum | instanceof | return | transient |
| catch | extends | int | shor t | try |
| char | final | inter face | static | void |
| class | finally | long | strictfp | volatile |
| const | float | native | super | while |

**String Literals**

String literals in Java are specified like they are in most other languages—by enclosing a sequence of characters between a pair of double quotes. Examples of string literals are

“Hello World” “two\nlines”  
“\”This is in quotes\”“

**Variables**

**Declaring a Variable**

*type identifier* [ = *value*][, *identifier* [= *value*] ...] ;

int a, b, c;

int d = 3, e, f = 5;

byte z = 22;

double pi = 3.14159;

char x = 'x';

// declares three ints, a, b, and c.

// declares three more ints, initializing

// d and f.

// initializes z.

// declares an approximation of pi.

// the variable x has the value 'x'.

**The Scope and Lifetime of Variables**

// Demonstrate block scope.

class Scope {

public static void main(String args[]) {

int x; // known to all code within main

x = 10;

if(x == 10) { // start new scope

int y = 20; // known only to this block

// x and y both known here.

System.out.println("x and y: " + x + " " + y);

x = y \* 2;

}

// y = 100; // Error! y not known here

// x is still known here.

System.out.println("x is " + x);

}

}