Latest Java Version – Java SE 18

## Features –

1. Simple
2. Object Oriented
3. Platform Independent - as it uses JVM to run the program
4. Secured – no pointers, no direct interaction with OS/machine, bytecode is verified, access control.
5. Robust – Strong memory mgmt., no pointers, GC, exception handling
6. Data Types are not dependent on platform
7. Portable
8. Performance is high – used in enterprise
9. Java supports RMI/EJB that makes any resource available over network
10. Multi-threading supported
11. Classes can be loaded dynamically

JAVA\_HOME = Java installation directory

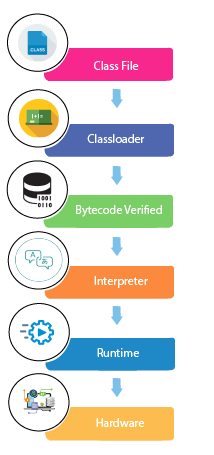
CLASSPATH= location of classes & packages used by program – JVM or compiler will use it

## Compilation -

Javac helloworld.java

## Execution-

Java <classname>

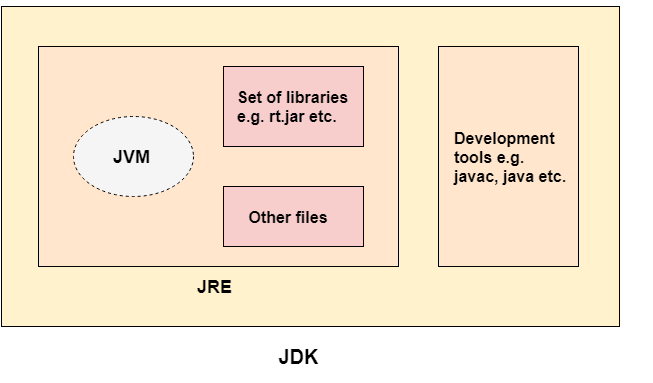
  
Note – public class must use the file name. If we don’t use public keyword, we can use any other file name as well.

## JDK vs JVM vs JRE:

JVM is a virtual machine that loads code, verifies bytecode, executes & provide runtime environment

JRE is set of tools used for development. It provides the RE & it is implementation of JVM. Basically it provides libraries and other s/w tools used by JVM

JDK is of three type – SE/EE/ME and it provides JRE and JVM as well as development tools like Javac, java etc



## JVM:

JVM is a specification and it is implemented by Oracle and many other companies. Mostly we use Oracle’s.

When we use java command, JVM is created and it –

* Loads the bytecode
* Verifies it
* Executes
* GC

Also it defines –

* Memory , class file, register set, GC heap, error reporting

Classloader –

* Bootstrap CL - Loads *rt.jar*
* Extension CL – loads – JAVAHOME/jre/lib/ext
* System CL – loads our class file

Program Counter register – points to the address of instruction currently being executed

Execution Engine – Interpreter & JIT Compiler

## Variables

* can use alphanumeric + $ & \_
* Three types –
  + Local – block level
  + Instance – object level
  + Static – class level

## Primitive Data Types –

* byte 1byte
* short 2byte
* int 4byte
* long 8byte
* float – 4 byte
* double – 8byte
* char – 2 byte – uses Unicode
* boolean – no size defined – can be 1 bit to 1 byte

## Operators:

|  |  |  |
| --- | --- | --- |
| **Operator Type** | **Category** | **Precedence** |
| Unary | postfix | *expr*++ *expr*-- |
| prefix | ++*expr* --*expr* +*expr* -*expr* ~ ! |
| Arithmetic | multiplicative | \* / % |
| additive | + - |
| Shift | shift | << >>  >>> sets msb to 0 for negative numbers |
| Relational | comparison | < > <= >= instanceof |
| equality | == != |
| Bitwise | bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive OR | | |
| Logical | logical AND | && |
| logical OR | || |
| Ternary | ternary | ? : |
| Assignment | assignment | = += -= \*= /= %= &= ^= |= <<= >>= >>>= |

## Constructor:

There are 2 types –

Default constructor – classname()

Parameterized –

Constructor do not have a return type, yet they return the current object.

Constructor can be overloaded

## Static Keyword:

* Static variables - are class variables
  + Are initialized once during class loading
  + Shared among all instances of objects and can be accessed directly from class
  + Saves memory – as it is kind of singleton
* Static Methods –
  + Belong to class and can not be called on object
  + However instance methods can call them using classname or if in scope – directly as a function.
  + Static method can not access the instance variables – only static variables
  + Main method is static so that we can call it directly
* Static Block –
  + Static block can initialize the static member
  + It is executed once, during class load, can be written anywhere in class
  + Its executed even before the main method
* Static class –
  + Will describe

## this keyword:

‘this’ represents the instance of the class

It can be used as

* To refer current object, its variables/methods
* To invoke constructor
* Can be passed as argument and can be returned from methods

## Inheritance: (IS-A)

class Child extends Parent{

}

Child can access the variables, methods of parent class

Java supports – Single , Hierarchical, Multilevel etc inheritance

Multiple Inheritance is not supported – to avoid confusion of where to find a method/variable – in which parent to choose. It makes the implementation easy even though a feature is missed here.

## Aggregation: (HAS-A)

Sometimes we use another class in the instance variable of our class. Example we created an engine class and using as below:

class Car{

    public String modelName;

    public String start(){

Engine e = new Engine();

}

}

Means – Car HAS A Engine

## Composition (Part-Of)

It is strong association -

class Car{

    public String modelName;

    private final Engine engine;

}

Both Aggregation & Composition are part of Association.

* + - Aggregation is - using other classes as local variables (HAS-A)
    - Composition is – using other class as static/instance (PART-OF)

## Method Overloading

We can overload methods – use the same name but change the signature –

1. By changing number of args
2. Or change the datatype of args

Note that we can not change the return type to avoid the confusion – which data type to store the result in. Hence it will throw compile time error.

## Method Overriding

Overriding is also known as runtime polymorphism. When we declare a method with same name but in child class with same signature. Else it is a normal method only.

* We can not override static methods – as they are bound to class area
* The access modifier can not have more restriction in child class. ie Parent method is public then child method must be public. If parent method is protected, child can be protected or public

## Access Modifiers:

* public (+)– accessible everywhere
* protected (#) – Accessible in same package & subclasses in different packages.
  + In other words, only same package (the author) and classes inheriting this class will be able to access the protected stuff.
* default – Accessible inside same package only – (only author)
* private (-): Accessible only inside its own class.