Full Stack Java Course:

JDK,JRE development:

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and [applets](https://www.javatpoint.com/java-applet). It physically exists. It contains JRE + development tools.



JRE:

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime

JVM:

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

Features of Java:

* The primary objective of [Java programming](https://www.javatpoint.com/java-tutorial) language creation was to make it portable, simple and secure programming language
* There are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as Java buzzwords.



JVM Architecture:



**Classloader** : is a subsystem of JVM which is used to load class files.

**Bootstrap ClassLoader:**  It loads the rt.jar file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes

**ExtensionClassLoader:** It loades the jar files located inside $JAVA\_HOME/jre/lib/ext directory.

**System/Application ClassLoader:** It loads the classfiles from classpath. By default, classpath is set to current directory.

Heap: It is the runtime data area in which objects are allocated.

Stack: It holds local variables and partial results, and plays a part in method invocation and return.

Program Counter: Contains the address of the Java virtual machine instruction currently being executed

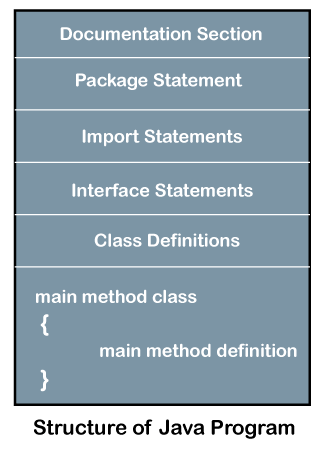
JDK and its Usage:

The Java Development Kit (JDK) is a cross-platformed software development environment that offers a collection of tools and libraries necessary for developing Java-based software applications and applets. It is a core package used in Java, along with the [**JVM (Java Virtual Machine)**](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/) and the JRE (Java Runtime Environment).

It's usage:

* Compiling the java code.
* Running java application.
* Provides Integrated environment and build tool like maven.
* Key JDK components like javac, java, Javadoc, jdb, jar.

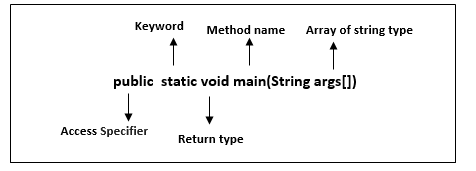
**Structure of Java Class:**



Writing a first java program:

1. **class** Simple{  //simple class declaration using class keyword
2. **public** **static** **void** main(String args[]){
3. //public represents visibility, static represents static method which does not need to create object to invoke method and void represents return type.//
4. System.out.println("Hello Java");
5. //system is class, out is object of printStream class and println() is method of printStream class//
6. }
7. }

**Main() Method :**



**main():** It is a default signature which is predefined in the JVM. It is called by JVM to execute a program line by line and end the execution after completion of this method. We can also overload the main() method.

**String args[]:** The main() method also accepts some data from the user. It accepts a group of strings, which is called a string array. It is used to hold the command line arguments in the form of string values

Different ways of writing main() method are:

1. **static** **public** **void** main(String []x)
2. **static** **public** **void** main(String...args)

Overloading the main() method:

1. **class** OverloadMain  {
2. **public** **static** **void** main(**int** a) { //overloaded main method
3. System.out.println(a);
4. }
5. **public** **static** **void** main(String args[]) {
6. System.out.println("main method incoked");
7. main(6);
8. }  }

Constructor:

In [Java](https://www.javatpoint.com/java-tutorial), a constructor is a block of codes similar to the method. It is called when an instance of the [class](https://www.javatpoint.com/object-and-class-in-java) is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

A Java constructor cannot be abstract, static, final, and synchronized

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Types of Constructors :

1. Default constructor (no-argument constructor): A constructor is called "Default Constructor" when it doesn't have any parameter**.**
2. Parameterized constructor: A constructor which has a specific number of parameters is called a parameterized constructor.

Constructor Overloading : Constructor [overloading in Java](https://www.javatpoint.com/method-overloading-in-java) is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task.

Class student{

Student(){//Default constructor

Sout(“Hello”) }

Student(String name){//parametrized constructor

Sout(“Welcome ”+name) }

}

Character set alphabet, digit, special symbol:

In Java, the character set used is based on Unicode, which is a standard for representing text in different writing systems.

Alphabet : **Alphabets in Java include both uppercase and lowercase letters from various languages.**

Digit : **Digits in Java are the numerical characters used to represent numbers.**

Special Symbol : **Special symbols in Java include punctuation marks, mathematical operators, and other miscellaneous symbols.**

Java Datatype:

Abstract data Type and Dsa

|  |  |
| --- | --- |
| Primitive Datatype | Non-primitive data type |
| Integer- int  long  short  byte | String  Array  List  Set  Stack |
| Floating point- float  double | Vector |
| Character- char | Dictionary |
| Boolean -boolean | All user defined classes |

Constant :

**Constant** is a value that cannot be changed after assigning it. Java does not directly support the constants. There is an alternative way to define the constants in Java by using the non-access modifiers static and final.

**static** **final** datatype identifier\_name=value;

* In the above statement, the **static** modifier causes the variable to be available without an instance of its defining class being loaded and the **final** modifier makes the variable fixed.
* If we declare a variable as **static**, all the objects of the class (in which constant is defined) will be able to access the variable and can be changed its value. To overcome this problem, we use the **final** modifier with a static modifier.

Enumeration (enum) as constant :

In the above statement, the **static** modifier causes the variable to be available without an instance of its defining class being loaded and the **final** modifier makes the variable fixed.

**public** **enum** Color {Red, Green, Blue, Purple, Black, White, Pink, Gray}

//traversing the enum

**for** (Colour c : Colour. Values())

System.out.println(c);

Java Literals :

Any constant value which can be assigned to the variable is called literal/constant.

**Decimal literals (Base 10):**In this form, the allowed digits are 0-9.

**Octal literals (Base 8):**In this form, the allowed digits are 0-7.

**Hexa-decimal literals (Base 16):**In this form, the allowed digits are 0-9, and characters are a-f.

Java Output :

System.out.println();//simply prints message and moves to next line

System.out.print();//simply print data and stay on same line

System.out.printf();//used to format the output of numbers and strings

Date and Time formatting :

We need to import

import java.time.LocalTime;

import java.time.format.DateTimeFormatter;

LocalTime time = LocalTime.now();//call built in function

// Predefined formatters

DateTimeFormatter formatter1 = DateTimeFormatter.ofPattern("HH:mm:ss");

DateTimeFormatter formatter2 = DateTimeFormatter.ofPattern("hh:mm a");

// Custom format

DateTimeFormatter customFormatter = DateTimeFormatter.ofPattern("HH:mm:ss.SSS");

import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

LocalDate date = LocalDate.now();

// Predefined formatters

DateTimeFormatter formatter1 = DateTimeFormatter.ofPattern("yyyy-MM-dd");

DateTimeFormatter formatter2 = DateTimeFormatter.ofPattern("dd/MM/yyyy");

DateTimeFormatter formatter3 = DateTimeFormatter.ofPattern("EEEE, MMM dd, yyyy");

// Custom format

DateTimeFormatter customFormatter = DateTimeFormatter.ofPattern("dd-MM-yyyy");

**Function in java:**

**Based on return type:**

* With return type//int ,char,float,double
* With void return type//which is not return anything



Also function can be :

* Predefined/built-in // already defined in the Java class libraries
* User defined // method written by the user or programmer

Also function based on parameter:

* With parameter //public void check();
* Withour parameter //public int check(int Num);

Also function based on scope:

* Static function //which does not need instance to call
* Instance function //need instance of class to call function

Also based on functionality:

* Utility function //to perform general purpose operation
* Getter /setter function //used to get /set values of private fields

|  |  |
| --- | --- |
| Function | Method |
| Normal function defined in class and invoked in the main function is called function | When function is invoked using the object of the class it becomes a method |
| Ex.  Class student{  Public static void sum(int a,int b){  Int c=a+b  }  Psvm(String[] args){  Sum(4,5);  }  } | Ex.  Class student{  Public void sum(int a,int b){  Int c=a+b  }  Psvm(String[] args){  Student s=new student();  s.Sum(4,5);  }  } |

Types of User defined function:

* Void function
* Function with return value
* Parametrized function
* Static function
* Instance function
* Constructor
* Event handler function

Calling a void function:

Public class student{

Public void printMessage(String msg){

Sout(msg);

}

Psvm(String[] args){

Student s=new student();

S.printMessage(“hello”);

}

}

Calling a function with return type:

Class add{

public int add(int a, int b) {

return a + b; }

Psvm(String[] args){

add a=new add();

a.add(a.add(5,4));

}

}

Flow of execution of function:

Class add{

public int add(int a, int b) {

return a + b; }

Psvm(String[] args){ Execution starts here

add a=new add(); creating a instance of class

a.add(a.add(5,4));//calling a function

}

}

Exercise1: Write a Calculator program without return statement using void method

Class calculator{

Public void sum(){

Int a=5,b=6;

Int sum=a+b;

Sout(“Sum :”+sum);

}

Public void multiply(){

Int a=5,b=6;

Int mul=a\*b;

Sout(“Multiplictaion :”+sum);

}

Public void divide(){

Int a=5,b=6;

Int div=a/b;

Sout(“Division :”+sum);

}Psvm(String[] args){

Calculator c=new calculator();

c.sum();

c.multiply();

c.divide();

}

}

Same program using Scanner class for Calculator

Import java.util.Scanner;

Class calculator{

Scanner sc=new Scanner(System.in);

Public void sum(){

Int a,b;

a=sc.nextInt();

b=sc.nextInt();

Int sum=a+b;

Sout(“Sum :”+sum);

}

Public void multiply(){

a=sc.nextInt();

b=sc.nextInt();

Int a=5,b=6;

Int mul=a\*b;

Sout(“Multiplictaion :”+sum);

}

Psvm(String[] args){

Calculator c=new calculator();

c.sum();

c.multiply();

c.divide();

}

}

Same program with return type for Calculator

Class calculator{

Public int sum(){

Int a=5,b=6;

Return a+b;

}

Public int multiply(){

Int a=5,b=6;

Return a\*b;

}

Psvm(String[] args){

Calculator c=new calculator();

c.sum();

c.multiply();

c.divide();

}

}

**Formal Parameter:** A variable and its type as they appear in the prototype of the function or method.

**Actual Parameter:** The variable or expression corresponding to a formal parameter that appears in the function or method call in the calling environment.

Pass by Value or Call by Value:

There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

1. **class** Operation{
2. **int** data=50;
3. **void** change(**int** data){
4. data=data+100;//changes will be in the local variable only
5. }
6. **public** **static** **void** main(String args[]){
7. Operation op=**new** Operation();
8. System.out.println("before change "+op.data);  //output: 50
9. op.change(500);
10. System.out.println("after change "+op.data);  //output :50
11. }
12. }

**By using objects:**

If we pass object in place of any primitive value, original value will be changed.

1. **class** Operation2{
2. **int** data=50;
3. **void** change(Operation2 op){
4. op.data=op.data+100;//changes will be in the instance variable
5. }
6. **public** **static** **void** main(String args[]){
7. Operation2 op=**new** Operation2();
9. System.out.println("before change "+op.data);
10. op.change(op);//passing object
11. System.out.println("after change "+op.data);
12. }
13. }