Java Definition

Java is an object-oriented programming language that produces software for multiple platforms. When a programmer writes a Java program, the compiled code (known as byte code) runs on most operating systems (OS), including Windows, Linux and Mac OS. Java derives much of its syntax from the C and C++ programming languages. A list of the most important features of the Java language is given below.

1. Simple:- Java is easy to learn and its syntax is quite simple, clean and easy to understand.
2. Object-Oriented:- In java, everything is an object which has some data and behavior.
3. Portable:- Java is portable because it facilitates you to carry the Java byte code to any platform. It doesn't require any implementation.
4. Platform independent:- Java is guaranteed to be write-once, run-anywhere language.
5. Secured:- Java program always runs in Java runtime environment with almost null interaction with system OS, hence it is more secure.
6. Robust :- The English mining of Robust is strong. Java is robust because: It uses strong memory management.
7. Architectural Neutral :- Compiler generates byte codes, which have nothing to do with particular computer architecture, hence a Java program is easy to interpret on any machine.
8. High Performance :- Java enables high performance with the use of compiler.
9. Multithreaded :- Java multithreading feature makes it possible to write program that can do many tasks simultaneously.
10. Distributed :- Java is also a distributed language. Programs can be designed to run on computer networks.
11. Dynamic :- Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand.

History of java

1) **James Gosling, Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.

2) Initially it was designed for small, embedded systems in electronic appliances like set-top boxes.

3) Firstly, it was called **"Greentalk"** by James Gosling, and the file extension was .gt.

4) After that, it was called **Oak** and was developed as a part of the Green project.

5) In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

What is Object Oriented Programming?

Object oriented programming (OOP) is a programming paradigm that relies on the concept of **classes** and **objects**. It is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes), which are used to create individual instances of objects. There are many object-oriented programming languages including JavaScript, C++, Java, and Python.

Characteristics of OOP

1. Class: - A group of objects that share common properties for data part and some program part are collectively called as class. In java class contains: methods, constructors, data types etc.
2. Object: - An entity that has state and behavior is known as an object. It is also called instance of class. e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible).
3. Inheritance in Java is a mechanism in which one class acquires all the properties and behaviors of a parent class. It is an important part of OOPs

(Object Oriented programming system). The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.

4. Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.
5. **Abstraction: -** Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essential units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.
6. Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.
7. A constructor is **a special type of function with no return type**. Name of constructor should be same as the name of the class. We define a method inside the class and constructor is also defined inside a class. A constructor is called automatically when we create an object of a class

Structured Programming

1. Structured Programming is designed which focuses on **process.**
2. Structured programming follows **top-down approach**.
3. In Structured Programming, Programs are divided into small self contained **functions**
4. Structured Programming provides **less reusability**, more function dependency.
5. Less abstraction and less flexibility.

Object Oriented Programming

1. Object Oriented Programming is designed which focuses on **data**.
2. Object oriented programming follows **bottom-up approach**.
3. In Object Oriented Programming, Programs are divided into small entities called **objects**
4. Object Oriented Programming provides more reusability, less function **dependency**.
5. More abstraction and more **flexibility**.

Different Types of Java Platforms

There are three different types of Java programming language platforms:

1**. Java Platform, Standard Edition (Java SE):** Java SE's API offers the Java programming language's core functionality. It defines all the basis of type and object to high-level classes. It is used for networking, security, database access, graphical user interface (GUI) development, and XML parsing.

2. **Java Platform, Enterprise Edition (Java EE):** The Java EE platform offers an API and runtime environment for developing and running highly scalable, large-scale, multi-tiered, reliable, and secure network applications.

3. **Java Programming Language Platform, Micro Edition (Java ME)**: The Java ME platform offers an API and a small-footprint virtual machine running Java programming language applications on small devices, like mobile phones.

COMPONENTS OF JAVA

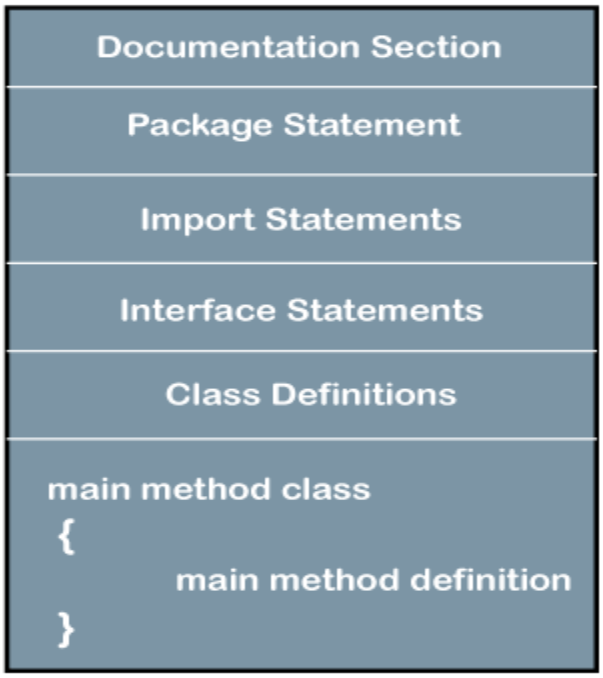
There are three main components of the Java programming language:

Java Virtual Machine (JVM): JVM is an engine that provides a runtime environment to drive the Java code or applications. It is the center of the programming language and performs the operation of converting Java code into machine language. It provides numerous libraries, frameworks, and tools.

Java Runtime Environment (JRE): JRE is a runtime environment that is required to execute Java programs and applications. If a user wants to run a Java program in their machine, they must have JRE installed on the machine. Its platform independent, meaning the JRE installed must be compatible with the user's operating system and architecture.

Java Development Kit (JDK): JDK is the core component of the Java environment. It contains JRE along with Java compiler, Java debugger, and other classes. It's used for Java development to provide the entire executables and binaries as well as the tools to compile and debug a Java program

Basic Structure of Java



Structure of Java Program

Eg.

WAP in java to print hello world

```
public class Hello
{
public static void main(String []args)
{
System.out.println("hello world");
}
}
```

Data Types in Java

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.

A. boolean − Stores 1-bit value representing true or, false.

B. byte − Stores twos compliment integer up to 8 bits.

C. char − Stores a Unicode character value up to 16 bits.

D. short − Stores an integer value upto 16 bits.

E. int − Stores an integer value upto 32 bits.

F. long − Stores an integer value upto 64 bits.

G. float − Stores a floating point value upto 32bits.

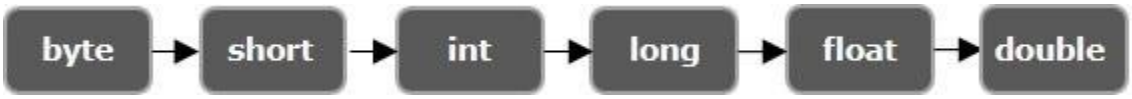
H. double − Stores a floating point value up to 64 bits.

2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

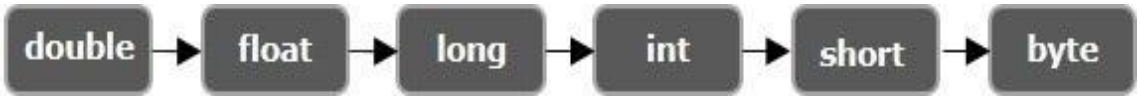
Type Casting/type conversion

Converting one primitive datatype into another is known as type casting (type conversion) in Java. You can cast the primitive datatypes in two ways namely, Widening and, Narrowing.

Widening − Converting a lower datatype to a higher datatype is known as widening. In this case the casting/conversion is done automatically therefore, it is known as implicit type casting. In this case both datatypes should be compatible with each other.



Narrowing − Converting a higher datatype to a lower datatype is known as narrowing. In this case the casting/conversion is not done automatically, you need to convert explicitly using the cast operator “( )” explicitly. Therefore, it is known as explicit type casting. In this case both datatypes need not be compatible with each other.



Int i=3;

Eg. `char ch = (char) i;`

Variables in Java

Variable in Java is a data container that saves the data values during Java program execution. Every variable is assigned a data type that designates the type and quantity of value it can hold. A variable is a memory location name for the data.

Creating variable int a;

Types of Variables in Java

Now let us discuss different types of variables which are listed as follows:

1. Local Variables
   These variables are created when the block is entered, or the function is called and destroyed after exiting from the block or when the call returns from the function.
   The scope of these variables exists only within the block in which the variables are declared, i.e., we can access these variables only within that block.

2. Instance Variables

As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.

3. Static Variables

These variables are declared similarly as instance variables. The difference is that static variables are declared using the static keyword within a class outside of any method, constructor or block.

Unlike instance variables, we can only have one copy of a static variable per class, irrespective of how many objects we create.

```
public class Variable

{

   public static int c=1;//static variable

      public int a;//instance variable

public int b=1;//instance variable

void display()

{

  c++;

b++;

System.out.println("instance variable  is"+b);

   System.out.println("static variable  is"+c);

}

public static void main(String []args)

{

   Variable v=new Variable();

Variable v1=new Variable();
```

```
    v.display();
    v1.display();
}
}
```

Static Method

The static keyword is used to construct methods that will exist regardless of whether or not any instances of the class are generated. Any method that uses the static keyword is referred to as a static method.

```
public class ExampleStatic{
    // static variable
    static int a = 40;

    // instance variable
    int b = 50;

    void simpleDisplay()
    {
        System.out.println(a);
        System.out.println(b);
    }

    // Declaration of a static method.
    static void staticDisplay()
    {
      System.out.println(a);
    }

    // main method
```

```
    public static void main(String[] args)
    {
        ExampleStatic obj = new ExampleStatic();
        obj.simpleDisplay();

        // Calling static method.
        staticDisplay();
    }
}
```

this keyword

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

```
public class Student{

int rollno;

String name;

float fee;

Student(int rollno,String name,float fee){

this.rollno=rollno;

this.name=name;

this.fee=fee;

}

void display(){

System.out.println(rollno+" "+name+" "+fee);

}

public static void main(String args[]){

Student s1=new Student(111,"ankit",5000f);

Student s2=new Student(112,"sumit",6000f);

s1.display();

s2.display();
```

}}

Data member and member function

The variables which are declared in any class by using any fundamental data types (like int, char, float etc) or derived data type (like class,interface etc.) are known as **Data Members**. And the functions which are declared in private section, public section ,protected and default are known as **Member functions**.

Access Modifiers in Java

The access modifiers in Java specify the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Constructors in Java

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

```
class Bike1{
  Bike1()
  {
  System.out.println("Bike is created");
  }
  public static void main(String args[])
  {
  Bike1 b=new Bike1();
  }
  }
```

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

```
class Student5{
      int id;
   String name;
```

```
    int age;
     //creating two arg constructor
     Student5(int i,String n){
     id = i;
     name = n;
     }
     //creating three arg constructor
     Student5(int i,String n,int a){
     id = i;
     name = n;
     age=a;
     }
     void display(){System.out.println(id+" "+name+" "+age);}
     public static void main(String args[]){
     Student5 s1 = new Student5(111,"Karan");
Student5 s2 = new Student5(222,"Aryan",25);
     s1.display();
     s2.display();
    }
 }
```

What is a Destructor?

A destructor is a special method that gets called automatically as soon as the life-cycle of an object is finished. A destructor is called to de-allocate and free memory.

Garbage Collector

A garbage collector is a program that runs on the Java virtual machine to recover the memory by deleting the objects which are no longer in use or have finished their life-cycle. An object is said to be eligible for garbage collection if and only if the object is unreachable.

Java Finalize() Method

It becomes fairly difficult for any developer to force the execution of a garbage collector, but there is an alternative to this. We can use the object. Finalize method which works exactly like a destructor in Java.

```
Import  java.lang.*;
public class DestructorExample
{
protected void finalize()
{
System.out.println("Object is destroyed by the Garbage Collector");
}
public static void main(String[] args)
{
DestructorExample de = new DestructorExample ();
de.finalize();
de = null;
System.gc();
System.out.println("Inside the main() method");
}
}
```

Constructor vs Destructor: Difference Between A Constructor And A Destructor

Constructor	Destructor
A constructor is used to initialize an instance of a class	A destructor is used to delete or destroy the objects when they are no longer in use
Constructors are called when an instance of a class is created	Destructors are called when an object is destroyed or released
Memory allocation	Releases the memory

Overloading is possible	Overloading is not allowed
They are allowed to have arguments	No arguments can be passed in a destructor

Constructor Chaining In Java

Constructor chaining is the process of calling one constructor from another constructor with respect to current object.

One of the main use of constructor chaining is to avoid duplicate codes while having multiple constructor (by means of constructor overloading) and make code more readable.

```
Java program to illustrate Constructor Chaining
// within same class Using this() method
class Temp
{
    // default constructor 1
    // default constructor will call another
constructor
    // using this keyword from same class
    Temp()
    {
        // calls constructor 2
        this(5);
        System.out.println("The Default constructor");
    }

    // parameterized constructor 2
    Temp(int x)
    {
        // calls constructor 3
        this(5, 15);
        System.out.println(x);
    }
```

```
    // parameterized constructor 3
    Temp(int x, int y)
    {
        System.out.println(x * y);
    }

    public static void main(String args[])
    {
        // invokes default constructor first
        new Temp();
    }
}
```

Output:

```
75

5

The Default constructor
```

Rules of constructor chaining :

1. The **this()** expression should always be the first line of the constructor.
2. There should be at-least be one constructor without the this() keyword (constructor 3 in above example).
3. Constructor chaining can be achieved in any order.

What happens if we change the order of constructors?

Nothing, Constructor chaining can be achieved in any order

```
// Java program to illustrate Constructor Chaining
// within same class Using this() keyword
// and changing order of constructors
class Temp
{
    // default constructor 1
    Temp()
    {
        System.out.println("default");
    }

    // parameterized constructor 2
    Temp(int x)
```

```
    {
        // invokes default constructor
        this();
        System.out.println(x);
    }
    // parameterized constructor 3
    Temp(int x, int y)
    {
        // invokes parameterized constructor 2
        this(5);
        System.out.println(x * y);
    }
    public static void main(String args[])
    {
        // invokes parameterized constructor 3
Temp t=  new Temp(8, 10);
}
}
```