**Java Preparation for Interview**

1. **What is Java?**

* Java is a **programming language** and a **platform**.
* **Platform**: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.
* Java is a high level, robust ( मजबूत ) , object-oriented and secure programming language.
* Java was developed by **Sun Microsystems.**

1. **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.
* Main Features of java as follows:
* **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](https://www.javatpoint.com/java-oops-concepts) 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.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. [Object](https://www.javatpoint.com/object-and-class-in-java)
2. [Class](https://www.javatpoint.com/object-and-class-in-java#class)
3. [Inheritance](https://www.javatpoint.com/inheritance-in-java)
4. [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
5. [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
6. [Encapsulation](https://www.javatpoint.com/encapsulation)

* **Portable:**

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

* **Platform Independent:**

When we write java code ( source code ) it’s a .java file and execution of the java program is 1st compiler converters .java file to byte code and this byte code is platform independent. Any machine can execute this byte code just we need JRE ( java runtime environment).

JVM is platform dependent because it takes byte code and generate byte code for the current operating system.

* **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**

**Classloader:** Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.

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

**Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

* **Robust:**

The English mining of Robust is strong. Java is robust because:

It uses strong memory management.

There is a lack of pointers that avoids security problems.

Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.

There are exception handling and the type checking mechanism in Java. All these points make Java robust.

* **Architecture neutral:**

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

* **Interpreted:**
* **High Performance:**

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

* **Multithreaded:**

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. **It shares a common memory area**. Threads are important for multi-media, Web applications, etc.

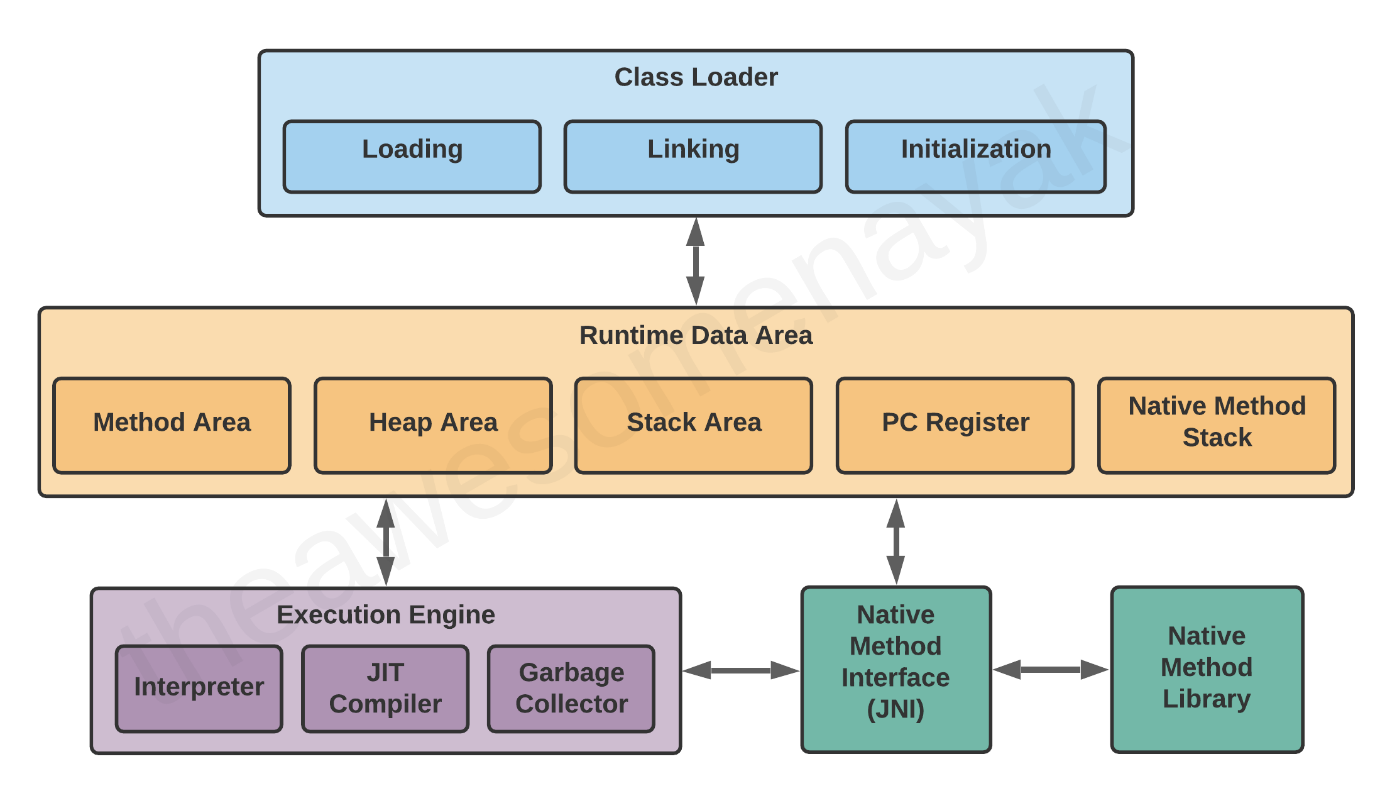
* **Distributed:**

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

* **Dynamic:**

Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

Java supports dynamic compilation and automatic memory management (garbage collection).

1. **JVM Architechture?**
2. **Oops in java?**

* **Object :**

Object is the entity ( **आस्तित्व** ) that has state and behavior.

* **Object-Oriented Programming:**

It is a methodology to design a program using class and objects.

It simplifies software development and maintenance by providing some concepts like:

**Object**

**Class**

**Inheritance**

**Polymorphism**

**Abstraction**

**Encapsulation**

* **Object:**

**Object is instance of class.**

**Entity that has state and behavior.**

**Object is runtime entity.**

Object is logical and physical entity.

Entity that has state and behavior is known as Object.

Object has three characteristics:

**State:** represents the data ( value ) of an object.

**Behavior:** represents the behavior ( functionality ) of an object

such as deposit, withdraw, etc.

**Identity:** An object identity typically implemented by unique ID.

The Value of the ID is not visible to the external user.

It is used internally by JVM to identify each object

Uniquely.

* **Class:**

A Class is group of objects which have common properties.

It is a blueprint from which object is created.

It is a logical entity. It can’t be physical.

A class in java contain:

**Fields ( variables )**

**Methods**

**Constructors**

**Blocks**

**Nested class and Interface**

**Fields ( instance variable):**

A variable which is created within the class but outside the method is known as Instance variable.

Instance variable doesn’t get memory at compile time. It get memory at runtime when an object or instance is created.

**Method:**

Method is like a function which is used to expose behavior of an object.

Advantage of Method is

**Code Reusability**

**Code Optimization**

**new keyword:**

new keyword is used to allocate memory at runtime. All objects get memory in Heap memory.

* **There are three ways to initialize the object:**

**By reference variable (** creating a object and with the help of that we can initialize)

Ex: Class Student{

Int id;

String name;

Psvm{

Student s = new Student();

s.id = 1;

s.name = “Vaibhav”;

}

}

**By method**

Ex: Class Student{

Int id;

String name;

void set(int i, String s ){

id = I;

name = s;

}

}

**By constructor**

Ex: Class Student{

Int id;

String name;

Student(int I, String s){

id = I;

name = s;

}

}

* **Different ways to create an object:**

**Using new keyword**

**Using new instance**

**Using clone() method**

**Using deserialization**

**Using newInstance() method of constructor class**

**Using new keyword:**

Ex:

class Student{

main(){

Student s = new Student();

}

}

**Using new instance:**

Ex:

// Java program to Illustrate Creation of Object

// Using new Instance

// Main class

class GFG {

// Declaring and initializing string

String name = "GeeksForGeeks";

// Main driver method

public static void main(String[] args)

{

// Try block to check for exceptions

try {

Class cls = Class.forName("GFG");

// Creating object of main class

// using instance method

GFG obj = (GFG)cls.newInstance();

// Print and display

System.out.println(obj.name);

}

// Catch block to handle the exceptions

// Catch block 1

// Handling ClassNotFound Exception

catch (ClassNotFoundException e) {

// Display the exception along with line number

// using printStacktrace() method

e.printStackTrace();

}

// Catch block 2

// Handling InstantiationException

catch (InstantiationException e) {

e.printStackTrace();

}

// Catch block 2

// Handling IllegalAccessException

catch (IllegalAccessException e) {

e.printStackTrace();

}

}

}

**Output:**

**GeeksForGeeks**

**By using clone() method:**

**Ex:**

// Java program to Illustrate Creation of Object

// Using clone() method

// Main class

// Implementing Cloneable interface

class GFG implements Cloneable {

// Method 1

@Override

protected Object clone()

throws CloneNotSupportedException

{

// Super() keyword refers to parent class

return super.clone();

}

// Declaring and initializing string

String name = "GeeksForGeeks";

// Method 2

// main driver method

public static void main(String[] args)

{

GFG obj1 = new GFG();

// Try block to check for exceptions

try {

// Using the clone() method

GFG obj2 = (GFG)obj1.clone();

// Print and display the main class object

// as created above

System.out.println(obj2.name);

}

// Catch block to handle the exceptions

catch (CloneNotSupportedException e) {

// Display the exception

// using printStackTrace() method

e.printStackTrace();

}

}

}

**Output**

GeeksForGeeks

**Note:**

Here we are creating the clone of an existing Object and not any new Object.

Class need to implement Cloneable Interface otherwise it will throw **CloneNotSupportedException**.

**By using deserialization:**

**Ex:**

**By using newInstance() method of the constructor class:**

**Ex:**

// Java program to illustrate creation of Object

// using newInstance() method of Constructor class

// Importing required classes from java.lang package

import java.lang.reflect.\*;

// Main class

class GFG {

// Member variables of this class

private String name;

// Constructor of this class

GFG() {}

// Method 1

// To set name ofthe string

public void setName(String name)

{

// This method refers to current object itself

this.name = name;

}

// Main driver method

public static void main(String[] args)

{

// Try block to check fo exceptions

try {

Constructor<GFG> constructor

= GFG.class.getDeclaredConstructor();

GFG r = constructor.newInstance();

// Custom passing

r.setName("GeeksForGeeks");

System.out.println(r.name);

}

// Catch block to handle the exceptions

catch (Exception e) {

// Display the exception on console

// using printStackTrace() method

e.printStackTrace();

}

}

}

**Output:**

**GeeksForGeeks**

* **Constuctor in java:**

A constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

Every time an object is created using the new() keyword, at least one constructor is called.

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.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

**Note:** It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

**Rules for creating Java constructor**

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

#### **Note: We can use [access modifiers](https://www.javatpoint.com/access-modifiers)**

#### while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Remember: Does constructor return any value?**

**There are no “return value” statements in the constructor, but the constructor returns the current class instance. We can write ‘return’ inside a constructor.**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### Q) What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

**If we declare the variable but not initialized it and there is no any constructor available in class, then JVM calls default constructor which initialize default value to declared variable.**

### Q) Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

## **Difference between constructor and method in Java**

There are many differences between constructors and methods. They are given below.

|  |  |
| --- | --- |
| **Java Constructor** | **Java Method** |
| A constructor is used to initialize the state of an object. | A method is used to expose the behavior of an object. |
| A constructor must not have a return type. | A method must have a return type. |
| The constructor is invoked implicitly. | The method is invoked explicitly. |
| The Java compiler provides a default constructor if you don't have any constructor in a class. | The method is not provided by the compiler in any case. |
| The constructor name must be same as the class name. | The method name may or may not be same as the class name. |

### What is the purpose of Constructor class?

Java provides a Constructor class which can be used to get the internal information of a constructor in the class. It is found in the java.lang.reflect package.

1. **Static keyword in java?**

The **static keyword** in [Java](https://www.javatpoint.com/java-tutorial)

is used for memory management mainly. We can apply static keyword with [variables](https://www.javatpoint.com/java-variables)

, methods, blocks and [nested classes](https://www.javatpoint.com/java-inner-class)

. The static keyword belongs to the class than an instance of the class.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class

**Static Variable:**

*\* static Variable:  
 \* - static variable allocate memory once and it is in method area.  
 \* - static variable is same for all the class if we change static variable value then it change actual value.  
 \*/*public class StaticVariable {  
 Integer id;  
 String name;  
 static String *college* = "PRMIT&R";  
  
 StaticVariable(Integer i, String s){  
 id = i;  
 name = s;  
 }  
  
 void display(){  
 System.*out*.println("id : "+id+", name: "+name+", college : "+*college*);  
 }  
  
 public static void main(String[] args) {  
 StaticVariable sv1 = new StaticVariable(1,"abc");  
 StaticVariable sv2 = new StaticVariable(2,"pqr");  
 StaticVariable sv3 = new StaticVariable(3,"xyz");  
  
 sv1.display(); // id : 1, name: abc, college : PRMIT&R  
 sv2.display(); // id : 2, name: pqr, college : PRMIT&R  
 sv3.display(); // id : 3, name: xyz, college : PRMIT&R  
 }  
}

**Static Method:**

There are two main restrictions for the static method. They are:

1. The static method can not use non static data member or call non-static method directly.
2. this and super cannot be used in static context.

*/\*\* Static Method:   
 \*   
 \* If you apply static keyword with any method, it is known as static method.  
 \*  
 \* A static method belongs to the class rather than the object of a class.  
 \* A static method can be invoked without the need for creating an instance of a class.  
 \* A static method can access static data member only and can change the value of it.  
 \* we can call static method with the help of class name only.  
 \*  
 \* The static method is used that will exist independently of any instances created for the class.  
 \*/*public class StaticMethod {  
 Integer id;  
 String name;  
 static String *college*="ABC";  
  
 static void change(){  
 *college* = "PQR";  
 }  
  
 StaticMethod(Integer i, String s){  
 id=i;  
 name=s;  
 }  
  
 void display(){  
 System.*out*.println("Id: "+id+", name: "+name+", college: "+*college*);  
 }  
  
 public static void main(String[] args) {  
 StaticMethod sm1 = new StaticMethod(1,"abc");  
 StaticMethod sm2 = new StaticMethod(2,"pqr");  
 StaticMethod sm3 = new StaticMethod(3,"xyz");  
  
 sm1.display(); // Id: 1, name: abc, college: ABC  
 StaticMethod.*change*();  
  
 sm2.display(); // Id: 2, name: pqr, college: PQR  
 sm3.display(); // Id: 3, name: xyz, college: PQR  
  
 }  
}

//Java Program to get the cube of a given number using the static method

class Calculate{

  static int cube(int x){

  return x\*x\*x;

  }

  public static void main(String args[]){

  int result=Calculate.cube(5);

  System.out.println(result);

  }

}

**Output: 125**

**Static Block:**

Is used to initialize the static data member.

**It is executed before the main method at the time of classloading.**

public class StaticBlock {  
  
 static {  
 System.*out*.println("in static Block");  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println("in main method");  
 }  
  
}  
//output:  
// in static Block  
// in main method

### Q) Can we execute a program without main() method?

Ans) No, one of the ways was the static block, but it was possible till JDK 1.6. Since JDK 1.7, it is not possible to execute a Java class without the [main method](https://www.javatpoint.com/java-main-method)

1. **this keyword in java?**

## Usage of Java this keyword

Here is given the 6 usage of java this keyword.

1. [this can be used to refer current class instance variable.](https://www.javatpoint.com/this1)

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

this.rollno=rollno;

this.name=name;

this.fee=fee;

}

1. [this can be used to invoke current class method (implicitly)](https://www.javatpoint.com/this2)

void m(){System.out.println("hello m");}

void n(){

System.out.println("hello n");

//m();//same as this.m()

this.m();

}

1. [this() can be used to invoke current class constructor.](https://www.javatpoint.com/this3)

**we can call constructor inside constructor using this**

A(){System.out.println("hello a");}

A(int x){

this();

System.out.println(x);

}

A(){

this(5);

System.out.println("hello a");

}

A(int x){

System.out.println(x);

}

1. [this can be passed as an argument in the method call.](https://www.javatpoint.com/this4)

void m(S2 obj){

System.out.println("method is invoked");

}

void p(){

m(this);

}

1. [this can be passed as argument in the constructor call.](https://www.javatpoint.com/this5)

We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes. Let's see the example:

class B{

A4 obj;

B(A4 obj){

this.obj=obj;

}

void display(){

System.out.println(obj.data);//using data member of A4 class

}

}

class A4{

int data=10;

A4(){

B b=new B(this);

b.display();

}

public static void main(String args[]){

A4 a=new A4();

}

}

1. [this can be used to return the current class instance from the method.](https://www.javatpoint.com/this6)

class A{

A getA(){

return this;

}

void msg(){System.out.println("Hello java");}

}

class Test1{

public static void main(String args[]){

new A().getA().msg();

}

}

**Output:**

**Hello java**