JIT(just in time compiler) - just in time compiler present inside

JVM and it tells the JVM to do not convert same code into machine code again and again instead use it from JIT. For example some function has been already converted into machine code so no need to convert it into it again and again. Instead use the already converted machine code from JIT.

**Primitive variable** -

int a = 10;

Here, a is a primitive variable.

It directly stores the value 10 in memory (no reference, no object).

There is no object and no "reference value" here.

so a itself contains the number 10.

**Reference variable: -**

A **reference variable** stores the "address" (or reference) of an object in memory, not the actual object itself.

It’s like a remote control pointing to the actual TV.

Example:

String name = "Amit";

Here:

name is a reference variable.

It doesn’t directly hold the string "Amit", but rather the reference (memory address) of the String object stored in the heap.

String s = "Hello";

Here,

**Reference variable** = **s**

**Object** = Hello

**Primitive types** (int, char, double, boolean, etc.) → they are **variables**, they store actual values.

**Objects (classes, arrays, strings, etc.)** → are stored in heap and accessed using **reference variables**.

Variable Arguments (Varargs)

Normally, a method has a fixed number of parameters.  
But with **varargs**, you can pass **0 or more arguments** to a method.

👉 Syntax:

returnType methodName(type... varName) { }

* type... means variable arguments.
* Inside the method, varargs behave like an **array**.

**Method With Varargs**

import java.util.Arrays;  
  
public class VariableArgument {  
 public static void main(String[] args) {  
 *fun*(2,3,4,5,6,7,8,9); // return an array  
  
 }  
  
 // another method with multiple parameter.two integer one variable argument  
 static void multiple(int a, int b, String...v) {  
 // String...v works as an Array  
  
 }  
  
  
  
 // method with one variable argument  
 static void fun(int... v) {  
 System.*out*.println(Arrays.*toString*(v));  
 }  
}

👉 int... numbers can take **any number of int values**.  
Inside the method, numbers acts like an array.

**🔹 Key Rules**

1. Only **one varargs parameter** per method.  
   ✅ void test(int... x)  
   ❌ void test(int... x, int... y)
2. Varargs must be the **last parameter**.  
   ✅ void test(String name, int... x)  
   ❌ void test(int... x, String name)