1. **If there is no main method, what will happen?**

In modern versions of Java, a program will fail to execute if it does not have a main method. The main method serves as the entry point for the Java Virtual Machine (JVM), and without it, the JVM does not know where to start the program.

1. **Does main method return any value?**

No, the main method in Java does not return any value. This is indicated by the void keyword in its signature: public static void main(String[] args)

**3) what is main method?**

The main method in Java is the essential starting point for any standalone Java application. When you run a Java program, the Java Virtual Machine (JVM) looks for this specific method to begin execution. Without a main method that has the correct signature, the JVM will not be able to run the code, even if it compiles without errors.

**4) (String[] args) what is this in main method?**

Thinking of the main method as the entry point to your program, String[] args is the designated space to receive information from the outside world when your application starts. It is a parameter that allows you to pass command-line arguments to your Java program.

Here is a breakdown of what String[] args means:

**String[]** This declares that the parameter is an array of strings. An array is a collection of elements, and in this case, each element is a String.

**args** This is the name of the array. "args" is a widely used convention that stands for "arguments," but you can technically name it anything you want.

**5) main(String[] args) in this insted of args can i write something else like java, or Selenium like this (String[] Selenium)?**

Yes, you can absolutely change the name of the String[] parameter in the main method from args to something else, such as java, selenium, or any other valid Java identifier.

The Java Virtual Machine (JVM) does not care about the name of the parameter; it only cares about its type (String[]) and its position in the method signature. The name args is simply a convention, short for "arguments," that has been widely adopted by developers for clarity and consistency.

1. **public static void main(string args[]) { } is this right main method, is this will work? ([] position is changed)**

Yes, public static void main(String args[]) {} is a correct and valid main method, and it will work perfectly fine.

In Java, the placement of the square brackets [] for an array declaration is flexible. You can place them either after the type (String[]) or after the variable name (args[]). Both syntaxes are functionally identical and are understood by the Java compiler and the Java Virtual Machine (JVM)

**7) public static void main(string[]args) { }, is this right main method, is this will work? (i have removed the space between []args)**

No, that main method is not right and will not work as the entry point for a Java application. The issue is not the missing space between [] and args, which is perfectly fine. The critical error is the use of string with a lowercase "s".

In Java, language elements are case-sensitive. The correct type for the parameter is String (with a capital "S"), which is a built-in class in Java. Using string (lowercase) will cause a compilation error because the compiler will not recognize it as the required type.

**8) public static void main(string...args) { }, is this right main method, is this will work? ( i have removed [] and added ...)**

No, that is not a right main method and it will not work as the entry point for a Java application. You have made two changes, and it's important to understand both:

**Changing [] to ... (This part is correct):** Using ... (three dots) is a valid alternative to []. This syntax is called varargs (variable arguments), and it was introduced in Java 5. For the main method, String... args is functionally identical to String[] args. The JVM recognizes both as a valid parameter type.

**Changing String to string (This part is the error):** This is the critical mistake that will cause the program to fail. Java is a case-sensitive language. The parameter must be of the type String (with a capital "S"), which is a built-in class in Java. The compiler does not recognize string (with a lowercase "s") as a valid type, and it will result in a compilation error.

**9) static public void main(String[] args) { is this valid main mehtod ?**

Yes, static public void main(String[] args) is a valid main method and it will work correctly.

In Java, the order of the modifiers public and static can be interchanged without affecting the program's execution. The Java Virtual Machine (JVM) only requires that both modifiers are present; their specific order is not strict.

**10) public void static main(String[] args) {} is this valid main method?**

No, public void static main(String[] args) {} is not a valid main method and will cause a compilation error.

**Here's the rule that this example violates: The return type (void) must come immediately before the method name (main).**

While the order of the modifiers public and static can be switched, you cannot place a modifier between the return type and the method name.

1. **public static main void (String[] args) {} is this valid main method?**

No, that is not a valid main method and it will cause a compilation error.

The structure of a Java method declaration has a strict order that must be followed. The error in your example is swapping the method name (main) with the return type (void).

The Correct Order: The correct syntax for any method declaration in Java, including the main method, is :

[modifiers] [return\_type] [method\_name]([parameters])

Let's break down the required structure for the main method:

Modifiers: public static (or static public)

Return Type: void

Method Name: main

Parameters: (String[] args)

**12) Who Gets Highest Priority in Execution?**

In Java, threads with a higher priority are given preference for execution over threads with a lower priority by the thread scheduler. This system helps manage which tasks get access to the CPU when multiple threads are ready to run simultaneously.

How Thread Priority Works

Every thread in Java is assigned a priority, which is an integer value between 1 and 10. The Java Thread class provides three static constants to represent these levels :

Thread.MAX\_PRIORITY: The highest priority, with a value of 10.

Thread.NORM\_PRIORITY: The default priority for any new thread, with a value of 5.

Thread.MIN\_PRIORITY: The lowest priority, with a value of 1.