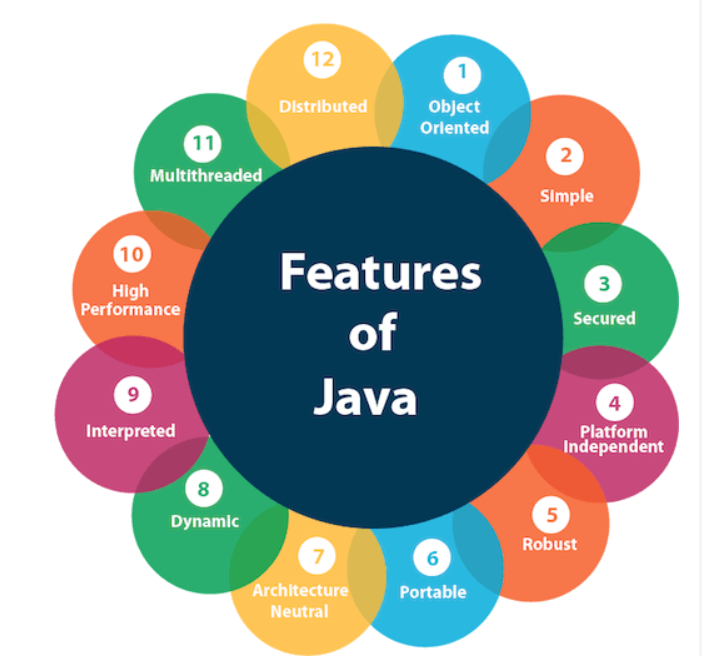
***PRACTICAL 1 : Questions***

Q1. What are the features of JAVA programming?



1. ***Platform Independent***



Java is platform independent because it is different from other languages like [C](https://www.javatpoint.com/c-programming-language-tutorial), [C++](https://www.javatpoint.com/cpp-tutorial), etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides a software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).

***2.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.

### *3.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.

### *4.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.

### *5.Portable*

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

Q2. Why do we learn and use JAVA programming for development of software application?

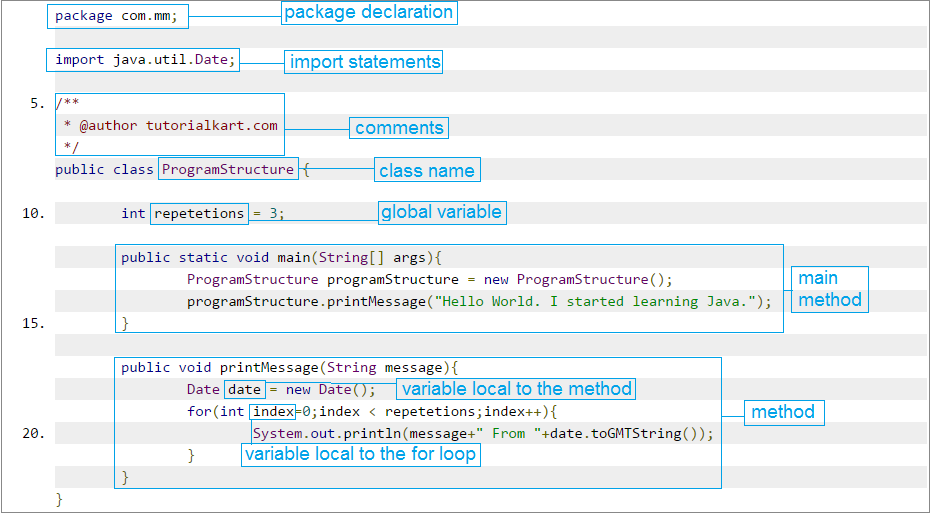
Q3. What is the structure of Java program and explain each statement in detail?

## ***Structure of a Java program***

A typical Structure of a Java program would contain the following elements:

* Package declaration
* Import statements
* Comments
* Class definition
* Class variables, Local variables
* Methods/Routines/Behaviours

The below picture would show the above mentioned elements that make out the structure of Java program.

[](https://www.tutorialkart.com/wp-content/uploads/2016/12/java_program_structure.png)

### Package Declaration

Classes in java could be placed in different directories/packages based on the module they are used in or the functionality it provides. For all classes that belong to a single parent source directory, path from source directory is considered as package declaration.

### Import Statements

There would be classes written in other folders/packages of your working java project and also there are many classes written by individuals, companies, institutions, enthusiasts etc., which could be useful in our program. To use them in a class, we would need to import the class that we intend to use. Many classes could be imported in a single program and hence multiple import statements could be written.

### Class Name

A name should be given to class in java file. This name is used while creating an object of the class, in other classes/programs.

### Variables

Variables are the means of storing the values of parameters that are required during execution of the program. Variables declared with modifiers have different scopes, which define the life of a variable. We shall see in detail about the modifiers like global, local, static and private at Variables in Java.

### Main method

Execution of a java application starts from “main” method. In other words, its an entry point for the class or program that starts in Java Run-time.

### Method/Routine/Behaviour

This is a room for a set of instructions which form a purposeful functionality or a routine that would be required to run multiple times during the execution of the program. To not repeat the same set of instructions when the same functionality is required, the instructions are enclosed in a method. A method’s behaviour could be exploited by passing variable values to the method (also called arguments of a method).

Q4. What is the difference between JVM, JRE and JDK?

## **Difference Between JDK, JRE, and JVM**

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **JDK** | **JRE** | **JVM** |
| Full-Form | The JDK is an abbreviation for Java Development Kit. | The JRE is an abbreviation for Java Runtime Environment. | The JVM is an abbreviation for Java Virtual Machine. |
| Definition | The JDK (Java Development Kit) is a software development kit that develops applications in Java. Along with JRE, the JDK also consists of various development tools (Java Debugger, JavaDoc, compilers, etc.) | The Java Runtime Environment (JRE) is an implementation of JVM. It is a type of software package that provides class libraries of Java, JVM, and various other components for running the applications written in Java programming. | The Java Virtual Machine (JVM) is a platform-independent abstract machine that has three notions in the form of specifications. This document describes the requirement of JVM implementation. |
| Functionality | The JDK primarily assists in executing codes. It primarily functions in development. | JRE has a major responsibility for creating an environment for the execution of code. | JVM specifies all of the implementations. It is responsible for providing all of these implementations to the JRE. |
| Platform Dependency | The JDK is platform-dependent. It means that for every different platform, you require a different JDK. | JRE, just like JDK, is also platform-dependent. It means that for every different platform, you require a different JRE. | The JVM is platform-independent. It means that you won’t require a different JVM for every different platform. |
| Tools | Since JDK is primarily responsible for the development, it consists of various tools for debugging, monitoring, and developing java applications. | JRE, on the other hand, does not consist of any tool- like a debugger, compiler, etc. It rather contains various supporting files for JVM, and the class libraries that help JVM in running the program. | JVM does not consist of any tools for software development. |
| Implementation | **JDK** = Development Tools + JRE (Java Runtime Environment) | **JRE** = Libraries for running the application + JVM (Java Virtual Machine) | **JVM** = Only the runtime environment that helps in executing the Java bytecode. |
| Why Use It? | Why use JDK?  Some crucial reasons to use JDK are:   * It consists of various tools required for writing Java programs. * JDK also contains JRE for executing Java programs. * It includes an Appletviewer, Java application launcher, compiler, etc. * The compiler helps in converting the code written in Java into bytecodes. * The Java application launcher helps in opening a JRE. It then loads all of the necessary details and then executes all of its main methods. | Why use JRE?  Some crucial reasons to use JRE are:   * If a user wants to run the Java applets, then they must install JRE on their system. * The JRE consists of class libraries along with JVM and its supporting files. It has no other tools like a compiler or a debugger for Java development. * JRE uses crucial package classes like util, math, awt, lang, and various runtime libraries. | Why use JVM?  Some crucial reasons to use JVM are:   * It provides its users with a platform-independent way for executing the Java source code. * JVM consists of various tools, libraries, and multiple frameworks. * The JVM also comes with a Just-in-Time (JIT) compiler for converting the Java source code into a low-level machine language. Thus, it ultimately runs faster than any regular application. * Once you run the Java program, you can run JVM on any given platform to save your time. |
| Features | Features of JDK   * Here are a few crucial features of JDK: * It has all the features that JRE does. * JDK enables a user to handle multiple extensions in only one catch block. * It basically provides an environment for developing and executing the Java source code. * It has various development tools like the debugger, compiler, etc. * One can use the Diamond operator to specify a generic interface in place of writing the exact one. * Any user can easily install JDK on Unix, Mac, and Windows OS (Operating Systems). | Features of JRE   * Here are a few crucial features of JRE: * It is a set of tools that actually helps the JVM to run. * The JRE also consists of deployment technology. It includes Java Plug-in and Java Web Start as well. * A developer can easily run a source code in JRE. But it does not allow them to write and compile the concerned Java program. * JRE also contains various integration libraries like the JDBC (Java Database Connectivity), JNDI (Java Naming and Directory Interface), RMI (Remote Method Invocation), and many more. * It consists of the JVM and virtual machine client for Java HotSpot. | Features of JVM  Here are a few crucial features of JVM:   * The JVM enables a user to run applications on their device or in a cloud environment. * It helps in converting the bytecode into machine-specific code. * JVM also provides some basic Java functions, such as garbage collection, security, memory management, and many more. * It uses a library along with the files given by JRE (Java Runtime Environment) for running the program. * Both JRE and JDK contain JVM. * It is easily customizable. For instance, a user can feasibly allocate a maximum and minimum memory to it. * JVM can also execute a Java program line by line. It is thus also known as an interpreter. * JVM is also independent of the OS and hardware. It means that once a user writes a Java program, they can easily run it anywhere. |

## **JDK vs. JRE vs. JVM: Key differences**

And now, for the differences:

* JDK is the development platform, while JRE is for execution.
* JVM is the foundation, or the heart of Java programming language, and ensures the program’s Java source code will be platform-agnostic.
* JVM is included in both JDK and JRE – Java programs won’t run without it.

Q5. What are the commands for compilation and execution of java program?

# ***Creating, Compiling, and Executing a Java Program***

# Step 1: Creating a Java Program

Let's use the command window prompt(cmd) and open a text editor such as **Notepad**to create the Java source-code file *Welcome.java*.

## Welcome.java

Let's create a Java program and name this file to *Welcome.java*:

public class Welcome {

public static void main(String[] args) {

System.out.println("Welcome to Java!");

}

}

# Step 2: Compiling a Java Program

A Java compiler translates a Java source file into a Java bytecode file. Let's use the following command to compile *Welcome.java* file:

javac Welcome.java

If there aren’t any syntax errors, the compiler generates a bytecode file with a *.class*extension. Thus, this command generates a file named *Welcome.class*.

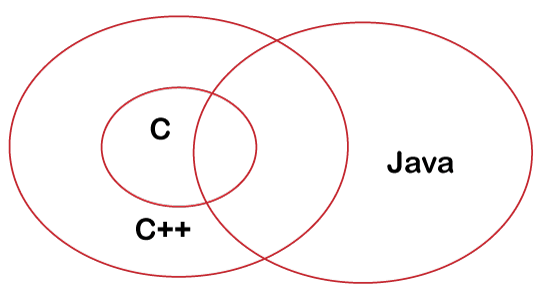
# Step 3: Executing a Java Program

Let's use the following command  to execute the bytecode:

java Welcome

The output of the above program displays the message **“Welcome to Java!”**

Q6. What is the difference between JAVA, C, C++?



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.N.** | **Basis** | **C** | **C++** | **Java** |
| **1** | **Origin** | The C language is based on BCPL. | The C++ language is based on the C language. | The Java programming language is based on both C and C++. |
| **2** | **Programming Pattern** | It is a procedural language. | It is an object-oriented programming language. | It is a pure object-oriented programming language. |
| **3** | **Approach** | It uses the top-down approach. | It uses the bottom-up approach. | It also uses the bottom-up approach. |
| **4** | **Dynamic or Static** | It is a static programming language. | It is also a static programming language. | It is a dynamic programming language. |
| **5** | **Code Execution** | The code is executed directly. | The code is executed directly. | The code is executed by the JVM. |
| **6** | **Platform Dependency** | It is platform dependent. | It is platform dependent. | It is platform-independent because of byte code. |
| **7** | **Translator** | It uses a compiler only to translate the code into machine language. | It also uses a compiler only to translate the code into machine language. | Java uses both compiler and interpreter and it is also known as an interpreted language. |
| **8** | **File Generation** | It generates the .exe, and .bak, files. | It generates .exe file. | It generates .class file. |
| **9** | **Number of Keyword** | There are **32** keywords in the C language. | There are **60** keywords in the C++ language. | There are **52** keywords in the Java language. |
| **10** | **Source File Extension** | The source file has a .c extension. | The source file has a .cpp extension. | The source file has a .java extension. |
| **11** | **Pointer Concept** | It supports pointer. | It also supports pointer. | Java does not support the pointer concept because of security. |
| **12** | **Union and Structure Datatype** | It supports union and structure data types. | It also supports union and structure data types. | It does not support union and structure data types. |
| **13** | **Pre-processor Directives** | It uses pre-processor directives such as #include, #define, etc. | It uses pre-processor directives such as #include, #define, #header, etc. | It does not use directives but uses packages. |
| **14** | **Constructor/ Destructor** | It does not support constructor and destructor. | It supports both constructor and destructor. | It supports constructors only. |
| **15** | **Exception Handling** | It does not support exception handling. | It supports exception handling. | It also supports exception handling. |
| **16** | **Memory Management** | It uses the calloc(), malloc(), free(), and realloc() methods to manage the memory. | It uses new and delete operator to manage the memory. | It uses a garbage collector to manage the memory. |
| **17** | **Overloading** | It does not support the overloading concept. | Method and operator overloading can be achieved. | Only method overloading can be achieved. |
| **18** | **goto Statement** | It supports the goto statement. | It also supports the goto statement. | It does not support the goto statements. |
| **19** | **Used for** | It is widely used to develop drivers and operating systems. | It is widely used for system programming. | It is used to develop web applications, mobile applications, and windows applications. |
| **20** | **Array Size** | An array should be declared with size. For example, int num[10]. | An array should be declared with size. | An array can be declared without declaring the size. For example, int num[] |

Q7. What are the valid signature of main method in JAVA?

The method signature consists of the method name and the parameter list.

## **Example**

public class MethodSignature {

public int add(int a, int b){

int c = a+b;

return c;

}

public static void main(String args[]){

MethodSignature obj = new MethodSignature();

int result = obj.add(56, 34);

System.out.println(result);

}

}

## **Output**

90

Method signature does not include the return type of the method. A class cannot have two methods with same signature. If we try to declare two methods with same signature you will get a compile time error.

public class MethodSignature {

public int add(int a, int b){

int c = a+b;

return c;

}

public double add(int a, int b){

double c = a+b;

return c;

}

public static void main(String args[]){

MethodSignature obj = new MethodSignature();

int result = obj.add(56, 34);

System.out.println(result);

}

}

## **Error**

C:\Sample>javac MethodSignature.java

MethodSignature.java:7: error: method add(int,int) is already defined in class MethodSignature

public double add(int a, int b){

^

1 error

Q8. Explain the significance of System.out.println statement?

Java **System.out.println()** is used to print an argument that is passed to it. The statement can be broken into 3 parts which can be understood separately as:

1. [System](https://www.geeksforgeeks.org/java-lang-system-class-java/)**:** It is a final class defined in the [java.lang package](https://www.geeksforgeeks.org/java-lang-package-java/).
2. **out:** This is an instance of [PrintStream](https://www.geeksforgeeks.org/java-io-printstream-class-java-set-1/)type, which is a public and static member field of the [System class](https://www.geeksforgeeks.org/java-lang-system-class-java/).
3. [println()](https://www.geeksforgeeks.org/difference-between-print-and-println-in-java/)**:** As all instances of [PrintStream class](https://www.geeksforgeeks.org/java-io-printstream-class-java-set-1/) have a public method println(), hence we can invoke the same on out as well. This is an upgraded version of print(). It prints any argument passed to it and adds a new line to the output. We can assume that System.out represents the Standard Output Stream.

[Diagram

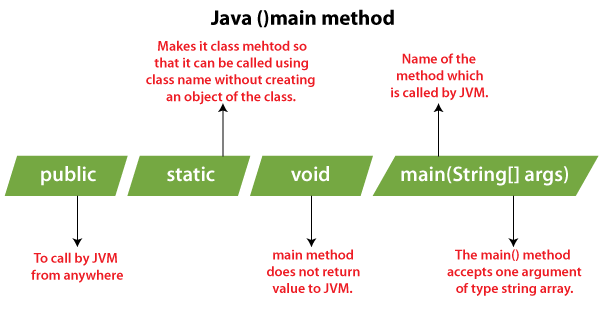
Description automatically generated](https://media.geeksforgeeks.org/wp-content/uploads/20191126171503/println1.png)

Q9. Why main method is static and public?

In Java, the **main()** method plays a vital role in program execution. The **main()** method is the first method that encounters first during execution. So, it is an entry point of a program. We cannot modify the syntax of the **main()** method. The only thing which we can change is the name of the String array argument. The syntax of the main() method is as follows:

**public** **static** **void** main(String[] args){

}



Let's divide the syntax of the **main()** method into several parts and understand each one of them:

### 1) public

It is not so complicated to understand. It is an access modifier of the **main()** method. We create main() method with **public** access specifier to execute it by any program. So, it is required to define main() method public and if we define the main() method as non-public, it will throw the following error:

**package** javaTpoint.MicrosoftJava;

// create TestMain class to check error for non-public main() method

**public** **class** TestMain {

        // define main() method as non-public

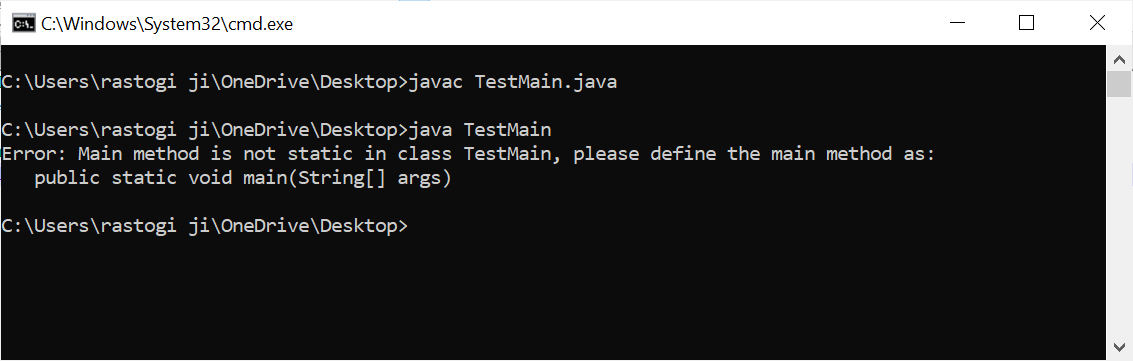
**static** **void** main(String[] args){

            System.out.println("Non-public main method.");

        }

}

**Output:**



### 2) static

The **static** is a keyword which we use in the main() method to define it as static. There is no object of the class available at the time of starting java runtime, and because of that, we have to define the main() method as **static**. By doing that, JVM can load the class into the main memory and call the main() method.

So, if we define main() method as non-static method, JVM would not be able to call it and throws the following error:

**package** javaTpoint.MicrosoftJava;

// create TestMain class to check error for non-static main() method

**public** **class** TestMain {

        // define main() method as non-public

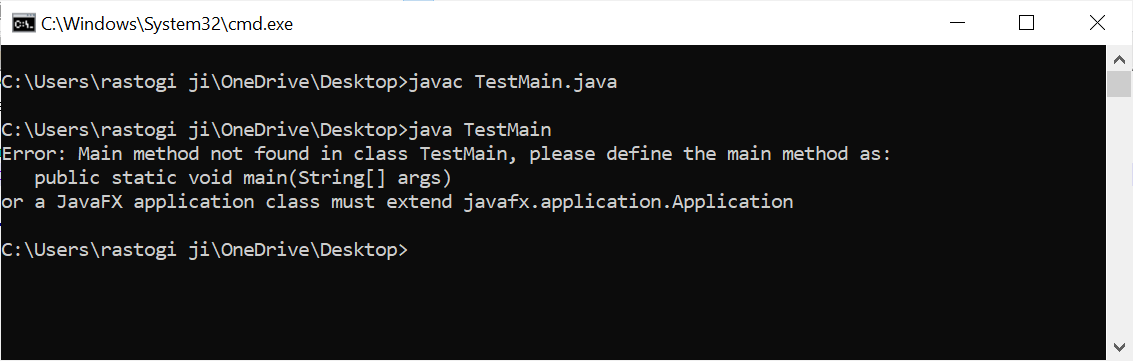
**public** **void** main(String[] args){

            System.out.println("Non-public main method.");

        }

}

**Output:**



### 3) void

As we know that, each method provides some return type such as String, Boolean, Integer etc. The Java main() method doesn't return anything, and its return type is **void.** The main() method doesn't return anything to make things simple. The program will be terminated after executing the **main()** method, and returning anything from the main() method is worthless because JVM will be done nothing for the returned object.

If we return something from the main() method, it will throw the following error:

// create TestMain class to check error when it doesn't have a return type

**public** **class** TestMain {

        // define main() method without return type

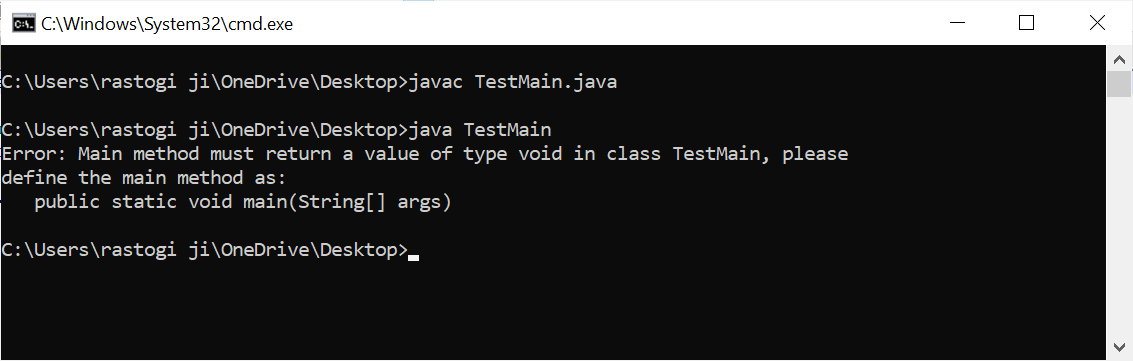
**public** **static** main(String[] args){

            System.out.println("Main method.");

        }

}

**Output:**



### 4) main()

It is the name of the **main()** method. The name of the method is static and we cannot change it. If we try to change the name of the method(), it will throw the following error:

// create TestMain class to check error for main() method having different name

**public** **class** TestMain {

        // define main() method as non-public

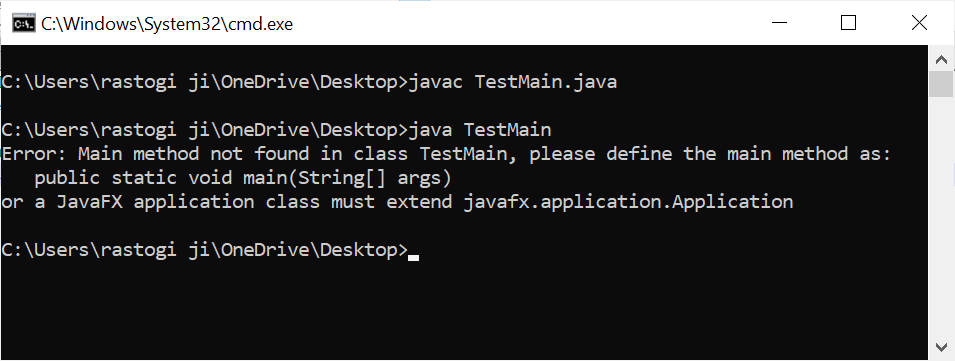
**public** **static** **int** newMain(String[] args){

            System.out.println("Main method.");

        }

}

**Output:**



### 5) String[] args or String args[]

The Java main() method takes a single argument of type String array as a parameter also referred to as **a command-line argument.** Let's take an example and understand how the command-line argument works.

// create TestMain class to understand command line argument

**public** **class** TestMain {

        // define main() method for it

**public** **static** **int** newMain(String[] args){

**for**(String s : args){

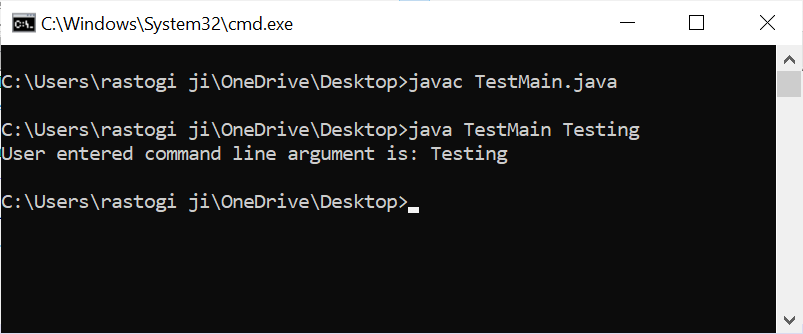
                System.out.println("User entered command line argument is: "+s);

            }

    }

}

**Output:**



### ****Reasons for defining main() method as static****

We cannot call a method without creating an instance of its class, and we already told you before that at the time of starting JVM, there is no object of a class. We create the main() method as static so that JVM can load the class into the main memory.

* The main() method is the entry point of each and every Java program. The main() method is required because the compiler starts executing a program from this entry point.
* The JVM needs to instantiate the class if the main() method is allowed to be non-static.
* JVM can call the static methods easily without creating an instance of the class by using the class name only.
* As discussed above, the main() method should be public, static, and have a return type void. If we do not define it as public and static or return something from the method, it will definitely throw an error.

Q10. What is a package and how to declare that?

**Package** in [Java](https://www.geeksforgeeks.org/java/) is a mechanism to encapsulate a group of classes, sub packages and interfaces. Packages are used for:

* Preventing naming conflicts. For example there can be two classes with name Employee in two packages, college.staff.cse.Employee and college.staff.ee.Employee
* Making searching/locating and usage of classes, interfaces, enumerations and annotations easier
* Providing controlled access: protected and default have package level access control. A protected member is accessible by classes in the same package and its subclasses. A default member (without any access specifier) is accessible by classes in the same package only.
* Packages can be considered as data encapsulation (or data-hiding).

All we need to do is put related classes into packages. After that, we can simply write an import class from existing packages and use it in our program. A package is a container of a group of related classes where some of the classes are accessible are exposed and others are kept for internal purpose.  
We can reuse existing classes from the packages as many time as we need it in our program.

Consider following two statements :

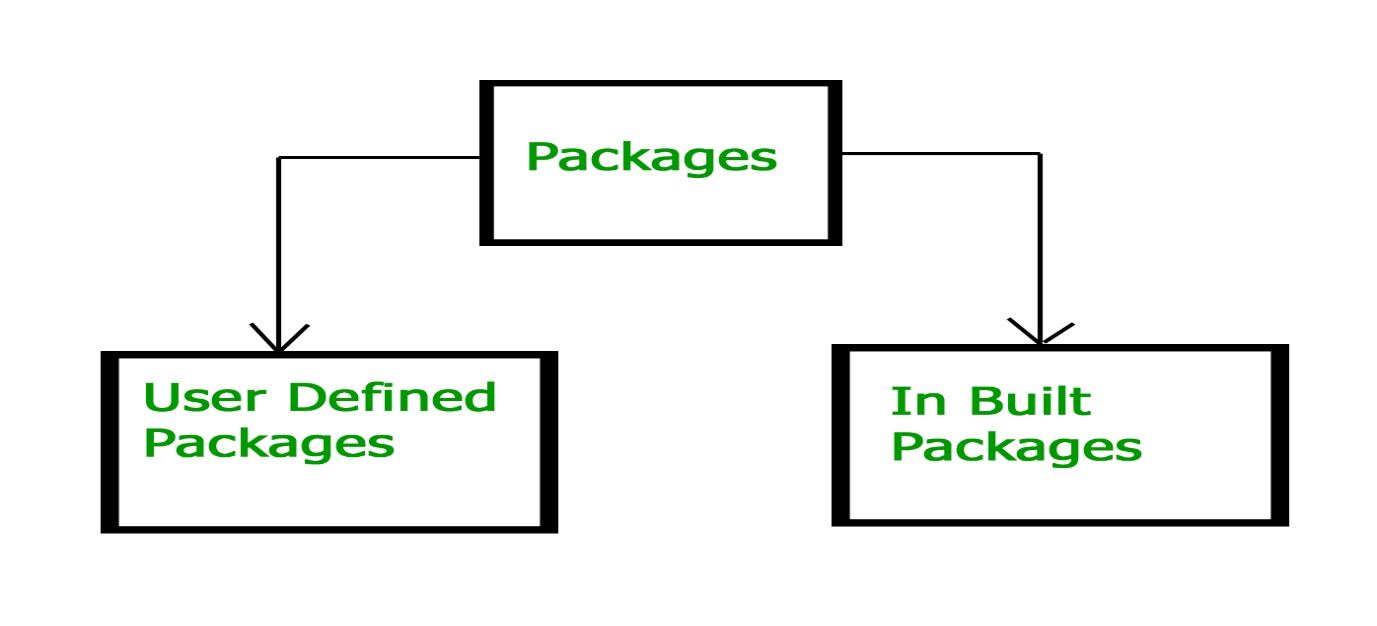
// import the Vector class from util package.

import java.util.vector;

// import all the classes from util package

import java.util.\*;

* First Statement is used to import **Vector** class from **util** package which is contained inside **java**.
* Second statement imports all the classes from **util** package.



**Built-in Packages**

These packages consist of a large number of classes which are a part of Java **API**.Some of the commonly used built-in packages are:

1) **java.lang:**Contains language support classes(e.g classed which defines primitive data types, math operations). This package is automatically imported.

2) **java.io:**Contains classed for supporting input / output operations.

3) **java.util:**Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.

4) **java.applet:**Contains classes for creating Applets.

5) **java.awt:**Contain classes for implementing the components for graphical user interfaces (like button , ;menus etc).

6) **java.net:**Contain classes for supporting networking operations.

**User-defined packages**  
These are the packages that are defined by the user. First we create a directory **myPackage** (name should be same as the name of the package). Then create the **MyClass** inside the directory with the first statement being the **package names**.

// Name of the package must be same as the directory

// under which this file is saved

package myPackage;

public class MyClass

{

public void getNames(String s)

{

System.out.println(s);

}

}

Now we can use the **MyClass** class in our program.

/\* import 'MyClass' class from 'names' myPackage \*/

import myPackage.MyClass;

public class PrintName

{

public static void main(String args[])

{

// Initializing the String variable

// with a value

String name = "GeeksforGeeks";

// Creating an instance of class MyClass in

// the package.

MyClass obj = new MyClass();

obj.getNames(name);

}

}

**Note :** **MyClass.java** must be saved inside the **myPackage** directory since it is a part of the package.

Q11. How to import any inbuilt class in JAVA?

Consider following two statements :

// import the Vector class from util package.

import java.util.vector;

// import all the classes from util package

import java.util.\*;

* First Statement is used to import **Vector** class from **util** package which is contained inside **java**.
* Second statement imports all the classes from **util** package.