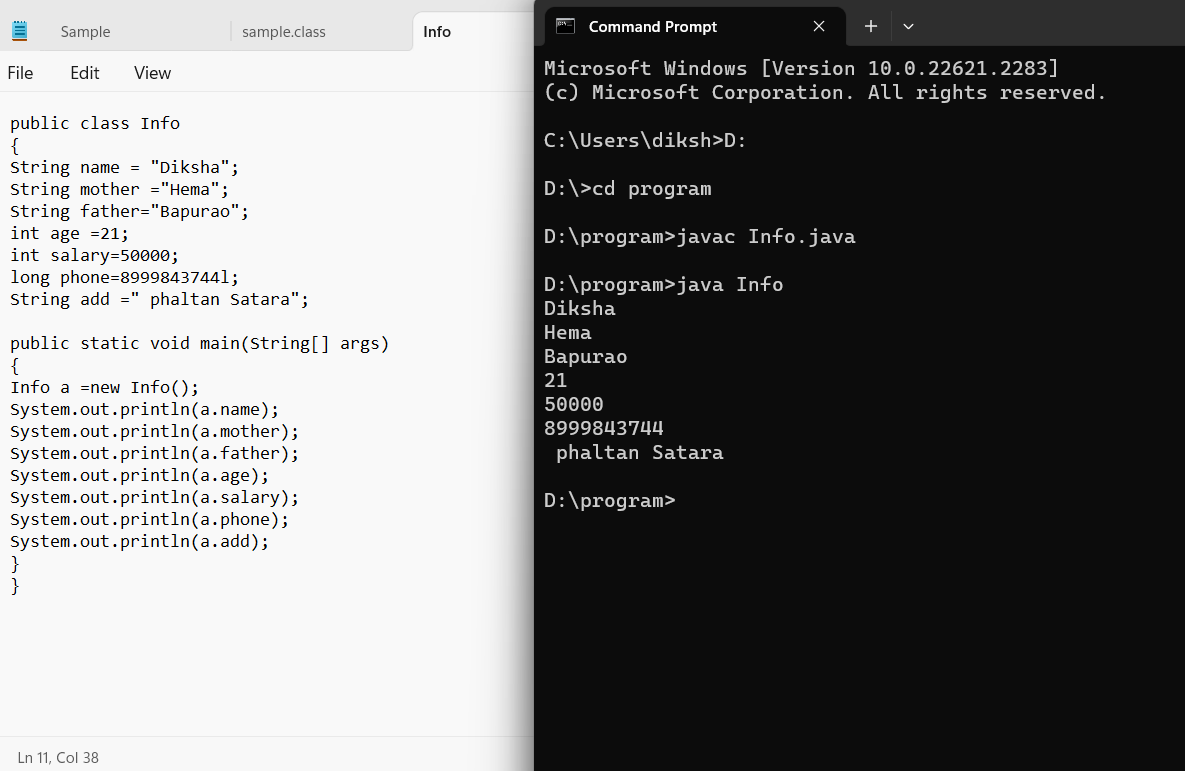
**Assignment No.1**

**Q1.create a java program to print your mother and father name, age, salary, phonenumber, address like city state country.**

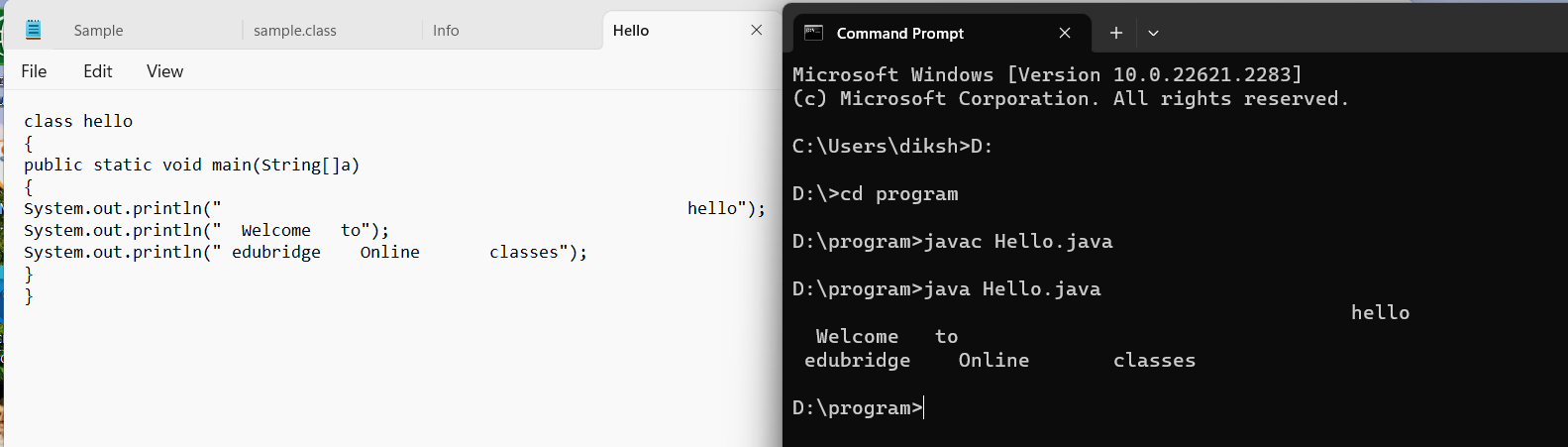
**Ans: **

**Q2.Write a java program to print the below msg in the given format**

**hello**

**welcome to**

**edubridge online classes**

**Ans:** ****

**Q3.What is default package in java?**

**Ans:** In Java, if you do not explicitly specify a package for your class, it belongs to the default package. Classes in the default package can be accessed from other classes within the same package, but they cannot be accessed from classes in named packages. It's generally recommended to avoid using the default package and instead organize your classes into named packages to promote better code organization and maintainability.

**Q4.Scanner class is in which package?**

**Ans:** The Scanner class in Java is part of **the java.util** package. To use the Scanner class in your Java program

**Q5.list the methods of Scanner class.**

**Ans:**

next(): Reads the next token (usually a word) from the input.

nextInt(): Reads the next integer value from the input.

nextDouble(): Reads the next double value from the input.

nextLine(): Reads the next line of text from the input.

hasNext(): Returns true if there is another token in the input.

hasNextInt(): Returns true if the next token is an integer.

hasNextDouble(): Returns true if the next token is a double.

useDelimiter(String pattern): Sets the delimiter pattern for the Scanner to use.

close(): Closes the scanner.

**Q6.list the types of datatypes with size and give an example of each by doing declaring and initializating the variables.**

**Ans:** There are two types of datatypes 1Primitive 2Non-primitive

In primitive there are 8 datatypes :-

datatypes size Default size

|  |  |  |
| --- | --- | --- |
| 1byte | 1 byte | 0 |
| 2 short | 2 byte | 0 |
| 3int | 4 byte | 0 |
| 4long | 8 byte | 0 |
| 5float | 4 byte | 0 |
| 6double | 8 byte | 0 |
| 7boolean | 1 bit | False |
| 8char | 2 byte | empty |

byte b=10;

short s=15;

int I = 100;

long = 124578963654l;

float f=55f;

double d = 271828;

boolean p = true;

char c= ‘A’;

**Non primitive datatypes**

**String:** A sequence of characters used to represent text String is a class in Java, not a primitive data type

String myString = "Hello, World!";

**Array:** A container object that holds a fixed number of values of a single type

int[] myIntArray = {1, 2, 3, 4, 5};

**Class Objects:** Objects created from user-defined classes

public class MyClass {

// class definition

}

MyClass myObject = new MyClass();

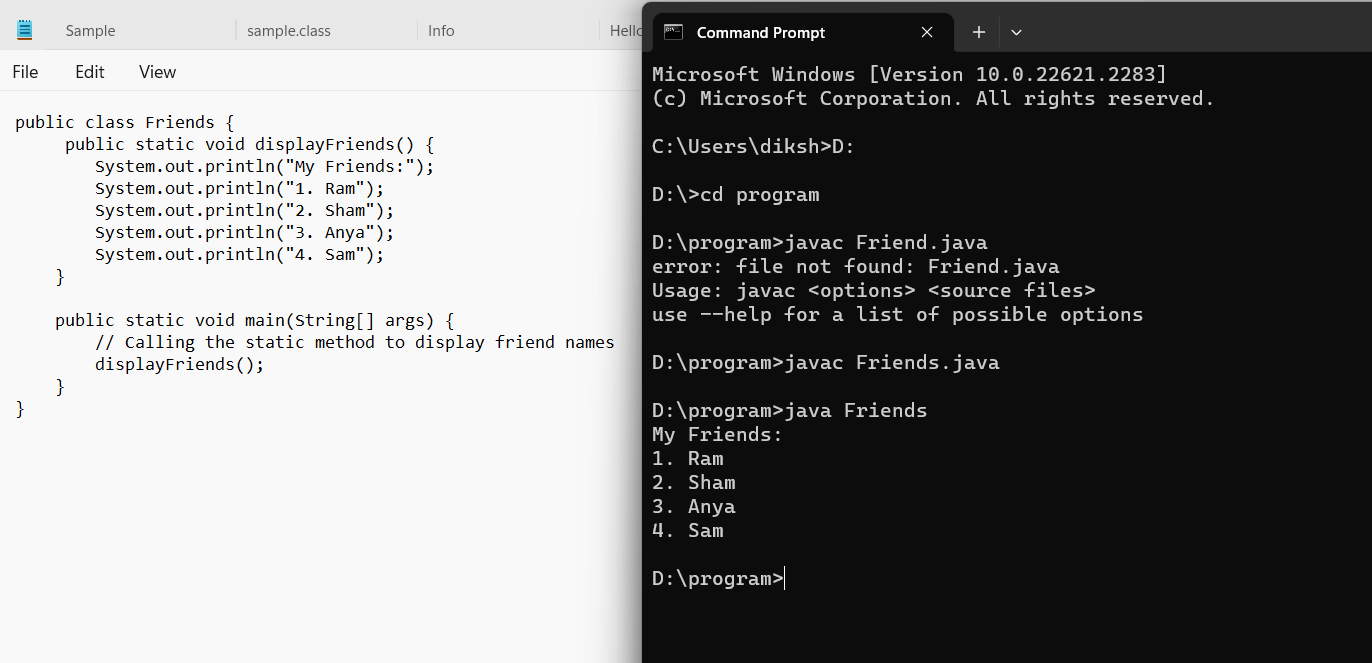
**Interfaces:** Similar to classes but they only contain method signatures and fields

public interface MyInterface {

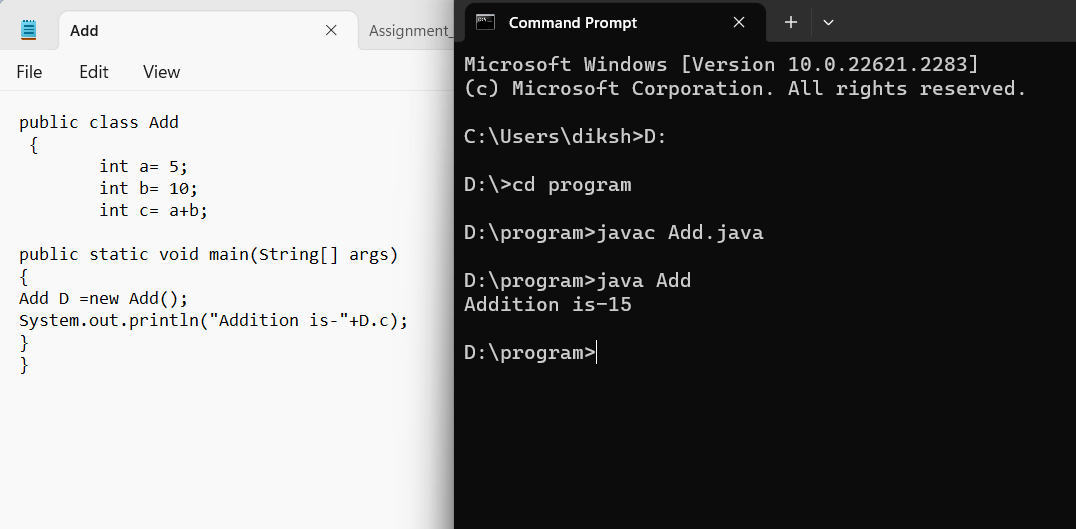
// interface definition

}

**Q7.Write a program to display the names of your friends by calling the static methods.**

**Ans:** 

**Q8.Write a program to add two number by using static method**

**Ans:** ****

**Q9Explain the public static void main(String[] args) in detail.**

**Ans:**

**- \*`public`\*:** It is an access modifier that specifies that the method is accessible from any other class. In Java, `main` method must be declared as `public` so that the JVM (Java Virtual Machine) can access and execute it.

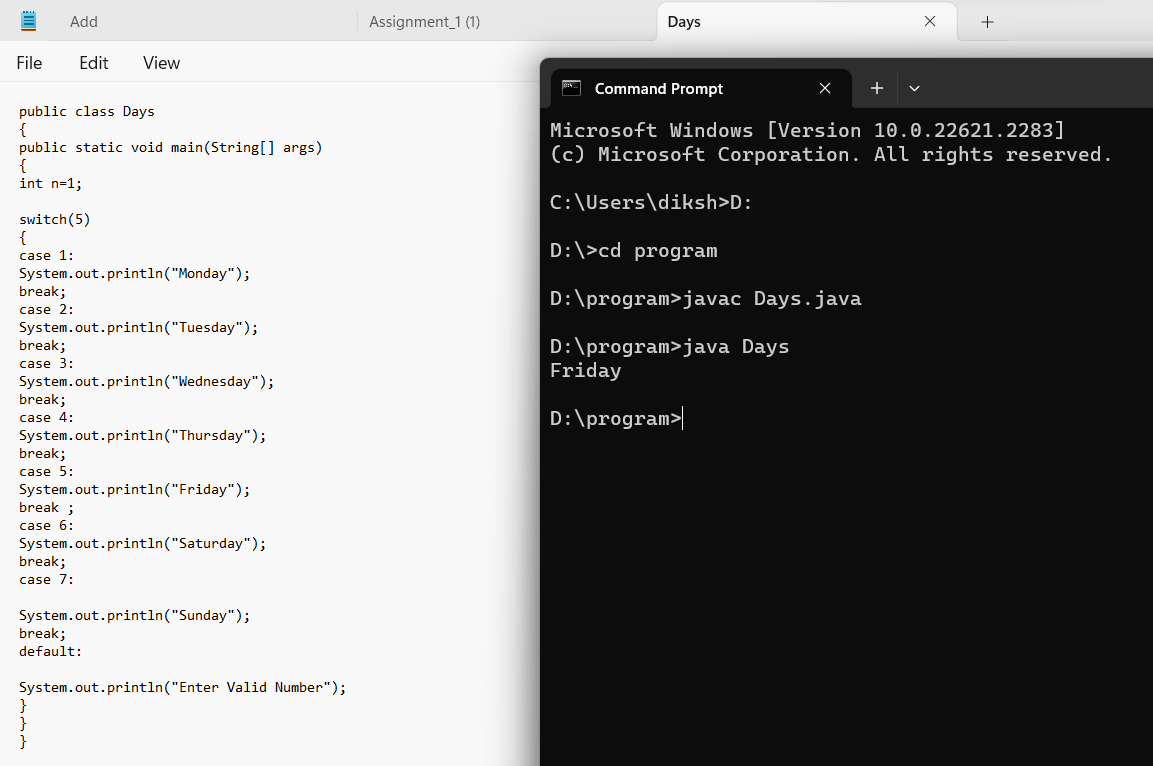
**- \*`static`\*:** It is a keyword that indicates that the method belongs to the class itself, rather than to any specific instance of the class. This means the method can be called without creating an instance of the class. In the context of the `main` method, it needs to be `static` because the JVM calls this method without creating any objects of the class.

**- \*`void`\*:** It is the return type of the method. `void` means the method does not return any value. The `main` method does not return anything, hence it's declared as `void`.

**- \*`main`\*:** It is the name of the method. The Java Virtual Machine looks for this specific method signature (`public static void main(String[] args)`) when starting the execution of a Java program.

**- \*`String[] args`\*:** It is a parameter of the `main` method. When you run a Java program from the command line, you can pass some values as arguments. These arguments are passed to the `main` method as an array of strings. `args` is the name of the array, and you can access the command-line arguments inside the `main` method using this parameter.

**Q10Write a program to display names of days by using commandLine arguments.**

**Ans:** ****

**Q11.What are the rules of naming the java file.**

**Ans:**

1. \*File Name Must Match the Class Name\*: The name of the Java file must exactly match the name of the public class it contains. For example, if your class is named `MyClass`, the file must be named `MyClass.java`.

2. \*Use CamelCase\*: Class names should be in CamelCase. CamelCase means the first letter of each word is capitalized (except the first word). For example, `MyClass`, `CalculatorService`, etc.

3. \*Use .java Extension\*: Java source files must have a `.java` extension. For example, `MyClass.java`.

4. \*Use Letters and Digits\*: File names should only contain letters (uppercase or lowercase) and digits. They should not contain spaces or special characters.

5. \*Start with a Letter\*: The file name must start with a letter (uppercase or lowercase). It should not start with a digit.

6. \*Avoid Java Keywords\*: Do not use Java keywords as file names. For example, `int.java` is not a valid file name because `int` is a keyword in Java.

Following these rules ensures that your Java files are named correctly and can be compiled without any issues. It also helps maintain a consistent and readable codebase.

**Q12.what are the rules of naming the class?**

**Ans:**

1. \*Use CamelCase\*: Class names should be in CamelCase, where the first letter of each word is capitalized. For example, `MyClass`, `CalculatorService`, `StudentRecord`, etc.

2. \*Start with a Capital Letter\*: Class names always start with a capital letter. This helps distinguish class names from variable names and method names, which typically start with a lowercase letter.

3. \*Avoid Underscores\*: Do not use underscores (\_) to separate words in class names. Use CamelCase instead. For example, prefer `StudentRecord` over `Student\_Record`.

4. \*Be Descriptive\*: Choose meaningful and descriptive names for your classes. The name should indicate the purpose of the class. For example, if your class represents a car, a suitable name might be `Car` or `Vehicle`.

5. \*Noun or Noun Phrase\*: Class names should be nouns or noun phrases, representing objects or concepts in your application. For example, `Person`, `Customer`, `FileReader`, etc.

6. \*Avoid Abbreviations\*: Try to avoid abbreviations in class names unless the abbreviation is well-known and widely used (e.g., `URL`, `HTML`, etc.). Using descriptive names improves the readability of your code.

7. \*Use Singular Form\*: Class names should be in singular form rather than plural. For example, use `Car` instead of `Cars`, `Book` instead of `Books`.

**Q13.what are the rules for naming the methods?**

**Ans:**

1. \*Use CamelCase\*: Method names should be in camelCase, where the first letter of the name is lowercase and the first letter of each subsequent concatenated word is capitalized. For example, `calculateTotalAmount()`, `getUserInfo()`, `printInvoice()`, etc.

2. \*Be Descriptive\*: Choose meaningful and descriptive names for your methods. A method name should clearly indicate what the method does. Avoid single-character or vague names. For example, use `calculateArea()` instead of `calc()`, and `getUserDetails()` instead of `getInfo()`.

3. \*Use Verbs\*: Method names should typically start with a verb that describes the action performed by the method. For example, `calculateTotal()`, `validateUser()`, `printReport()`, etc.

4. \*Avoid Abbreviations\*: Similar to class names, try to avoid abbreviations in method names unless the abbreviation is well-known and widely used. Using descriptive names improves the readability of your code. For example, use `initializeDatabase()` instead of `initDB()`.

5. \*Follow Java Naming Conventions\*: Adhere to the Java naming conventions. For example, accessor methods (methods used to retrieve the value of an instance variable) should start with "get" or "is" for boolean properties (e.g., `getName()`, `isComplete()`).

6. \*Use Parameters\*: If a method requires parameters, use descriptive names for the parameters. Parameter names should also follow camelCase. For example, `calculateArea(int length, int width)`. Descriptive parameter names make it clear what values are expected.

7. \*Avoid Overloading\*: Avoid creating methods with the same name but different parameter lists (method overloading) unless they serve a similar purpose. Overuse of method overloading can lead to confusion.

8. \*Consistency\*: Be consistent with the naming style throughout your codebase. If you follow a specific naming convention for methods, stick to it to maintain consistency.

**Q14.what are the rules for naming variables in java?**

**Ans:**

1. \*Use CamelCase\*: Variable names should be in camelCase, where the first letter of the name is lowercase and the first letter of each subsequent concatenated word is capitalized. For example, `firstName`, `totalAmount`, `numberOfStudents`, etc.

2. \*Be Descriptive\*: Choose meaningful and descriptive names for your variables. A variable name should clearly indicate the purpose of the variable. Avoid single-character or vague names. For example, use `customerName` instead of `cName`, and `totalSales` instead of `total`.

3. \*Use Letters and Digits\*: Variable names can consist of letters (uppercase or lowercase) and digits. They should not start with a digit.

4. \*Avoid Special Characters\*: Do not use special characters such as @, $, and % in variable names.

5. \*Avoid Java Keywords\*: Do not use Java keywords (reserved words) as variable names. For example, `int`, `class`, `while` are keywords and cannot be used as variable names.

6. \*Use Meaningful Names\*: Choose variable names that reflect the content or use of the variable. For example, use `userInput` instead of `data` if the variable stores user input.

7. \*Constants\*: If a variable is a constant (its value never changes), use all uppercase letters with underscores separating words. For example, `MAX\_VALUE`, `PI`, `DEFAULT\_SIZE`.

8. \*Class Member Variables\*: Class member variables (instance variables) should start with "this" to distinguish them from method parameters. For example, `this.age` refers to the member variable `age` of the class.

9. \*Local Variables\*: Local variables (variables declared inside methods) should be named in camelCase and be descriptive. For example, `int numberOfItems`.

10. \*Final Variables\*: If a variable is declared final (its value cannot be changed), it should be in all uppercase letters with underscores separating words. For example, `final int MAX\_LENGTH`.

**Q15.which is the main entry point of java program?**

**Ans:** In Java, the main method is the entry point of a Java program. When you run a Java program, the Java Virtual Machine (JVM) is responsible for executing the program. The JVM looks for the `public static void main(String[] args)` method in the class that you specify as the starting point of your program. The `main` method must be defined as `public` so that the JVM can access it.

So, to answer your question, the JVM is responsible for invoking the `main` method of a Java program when you run the program.

**Q16.who is invoking the main method of java ?**

**Ans: :** In Java, the main method is the entry point of a Java program. When you run a Java program, the Java Virtual Machine (JVM) is responsible for executing the program. The JVM looks for the `public static void main(String[] args)` method in the class that you specify as the starting point of your program. The `main` method must be defined as `public` so that the JVM can access it.

**Q17.why main has to be static method?**

**Ans:**

1. \*Entry Point:\* The `main` method serves as the entry point of a Java application. When you run a Java program, you don't create an object of the class containing the `main` method. Instead, the JVM directly calls the `main` method to start the program.

2. \*Static Context:\* The `main` method is static so that it can be called by the JVM before any objects are created for the class. In a static method, you cannot access instance-specific variables and methods directly because there is no instance associated with it. By declaring `main` as `static`, you indicate that it belongs to the class and not to any specific instance of the class.

3. \*Simplicity:\* Making `main` static simplifies the process of starting a Java program. You don't have to worry about creating an object of the class first. It can be called by the JVM directly from the class itself.

**Q18.components of java program are?**

**Ans**:

**1. \*Package Declaration (Optional):\*** You can organize your classes into packages. The package declaration, if used, appears at the beginning of the file.

**2. \*Import Statements (Optional):\*** If your program uses classes from other packages, you need to import them. Import statements, if used, come after the package declaration and before the class declaration.

**3. \*Class Declaration:\*** Every Java program consists of at least one class. The class declaration is where you define the properties and behaviors of objects. It contains methods and variables.

**4. \*Main Method:\*** The `main` method is the entry point of a Java program. It is mandatory in every Java application. The program execution starts from the `main` method.

**5. \*Variables:\*** Variables are used to store data values. They must be declared before they are used.

**6. \*Methods:\*** Methods define the behavior of objects. They contain a set of instructions that perform a specific task.

**7. \*Comments:\*** Comments are used to describe the code and improve its readability. There are single-line comments (`//`) and multi-line comments (`/\* \*/`).

**Q19.what is jvm,jre,jdk?**

**Ans:**

**1. \*JVM (Java Virtual Machine):\***

- JVM is a virtual machine that enables a computer to run Java programs as well as programs written in other languages and compiled to Java bytecode.

- It provides a runtime environment in which Java bytecode can be executed, enabling Java applications to be platform-independent.

**2. \*JRE (Java Runtime Environment):\***

- JRE is a software package that provides the Java runtime components, including JVM, libraries, and other files that JVM uses during runtime.

- It is used to run Java applications. If you just want to run Java programs, you need to install JRE on your system.

**3. \*JDK (Java Development Kit):\***

- JDK is a software development kit that provides the tools and resources needed for developing Java applications.

- It includes JRE, an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), and other tools required for Java development.

- If you are a Java developer, you need JDK installed on your system to write, compile, and run Java programs

**Q20.Explain the components of compile time environments and run time environments.**

**Ans:** Let's delve deeper into the components of both compile-time and runtime environments in software development:

**Compile-time Environment**

1. \*Source Code:\*

- The source code is the human-readable code written by developers. It contains the instructions and logic of the program.

2. \*Compiler:\*

- The compiler is a software tool that translates the source code into machine code or bytecode.

- It checks the syntax and semantics of the code and generates an intermediate form of the program that can be executed by the computer.

- Compilers are language-specific; for example, Java source code is compiled by the Java compiler (`javac`).

3. \*Syntax Checker and Type Checker:\*

- These components of the compiler ensure that the source code follows the correct syntax of the programming language and enforces type rules.

- Syntax checker identifies grammatical errors, while type checker ensures variables and operations are used correctly according to their data types.

4. \*Intermediate Code or Object Code:\*

- The compiler produces intermediate code or object code, which is a lower-level representation of the source code.

- In some languages, this could be machine code directly; in others like Java, it's bytecode, an intermediate code interpreted by the runtime environment.

**Run-Time Environment:**

1. \*Compiled Code or Bytecode:\*

- The compiled code or bytecode is generated by the compiler and is used at runtime.

- In languages like Java, this is platform-independent and needs an interpreter (Java Virtual Machine) or a Just-In-Time (JIT) compiler to execute.

2. \*Loader:\*

- The loader loads the compiled code or bytecode into the computer's memory before execution.

- It prepares the program for execution by allocating memory and linking references to external libraries.

3. \*Interpreter or Just-In-Time (JIT) Compiler:\*

- For languages like Java, the bytecode is interpreted by the interpreter or compiled to native machine code by the JIT compiler at runtime.

- The interpreter reads the bytecode line by line and executes corresponding machine instructions.

- JIT compiler translates bytecode into native machine code just before execution, improving performance.

4. \*Runtime Libraries:\*

- Runtime libraries are collections of pre-written code that provide low-level operations used by the compiled code during execution.

- They handle tasks such as input/output operations, mathematical computations, and memory management.

5. \*Memory Manager and Garbage Collector:\*

- Memory manager allocates and deallocates memory dynamically during program execution.

- Garbage collector identifies and frees up memory occupied by objects that are no longer in use, ensuring efficient memory usage.

6. \*Operating System Interfaces:\*

- The runtime environment interacts with the operating system, utilizing its services for tasks such as file I/O, networking, and process management**.**

**Q21.what is JIT?**

**Ans**: JIT stands for \*Just-In-Time\* compilation. It's a technique used in computer programming where the bytecode of a program, often found in Java and .NET environments, is compiled into native machine code at runtime, just before the program is executed. This contrasts with ahead-of-time (AOT) compilation where code is compiled before the program is run.

Here's how JIT compilation works:

1. \*Bytecode Compilation:\*

- In languages like Java, source code is first compiled into an intermediate form called bytecode. Bytecode is platform-independent and is executed by a virtual machine.

2. \*Interpretation or Compilation at Runtime:\*

- Initially, the bytecode is interpreted by the virtual machine, which can be slower than native machine code execution.

- When the JVM (Java Virtual Machine) or similar runtime environment detects that a particular section of bytecode is frequently executed, it can decide to compile that specific part into native machine code.

3. \*Compilation to Native Code:\*

- The identified bytecode is compiled to the native machine code specific to the computer's architecture. This process is done just before the code is executed, hence the term "Just-In-Time."

4. \*Execution of Native Code:\*

- The compiled native code is executed directly by the CPU, which is much faster than interpreting bytecode.

The advantage of JIT compilation is that it combines the flexibility of interpretation (which makes it easy to run code on different platforms without modification) with the speed of native machine code execution. By identifying and compiling only the frequently executed parts of the code, JIT compilation optimizes the performance of the application.

**Q22.different types of memory in jvm.**

**Ans:**

1. \*Heap Memory:\*

- The heap is the runtime data area from which memory for all class instances and arrays are allocated.

- Objects in Java are dynamically allocated in the heap memory area and can be accessed by multiple threads.

- The Java garbage collector works in the heap space, removing unreferenced objects to free up memory.

2. \*Method Area (PermGen/Metaspace):\*

- The method area stores class structures like class and method information, static variables, constant pool, and method code.

- In older versions of Java (up to Java 7), this area was called PermGen (Permanent Generation). In Java 8 and later versions, it's replaced by Metaspace, which is not fixed in size and can grow or shrink dynamically.

3. \*Stack Memory:\*

- Each thread in a Java application has its own stack, which stores method call information, local variables, and partial results.

- Stack memory is used for method invocations and managing local variables.

- It operates in a Last In, First Out (LIFO) manner, meaning the most recently called method is always at the top of the stack.

4. \*PC Registers:\*

- PC (Program Counter) registers store the address of the Java virtual machine instruction currently being executed.

- Each thread has its own PC register, ensuring that it keeps track of which instruction to execute next.

5. \*Native Method Stacks:\*

- Native method stacks are used for native methods, which are methods written in a language other than Java (e.g., C or C++).

- When a Java method calls a native method, the JVM creates a new native method stack for that method.

6. \*Direct Memory (Off-Heap Memory):\*

- Direct memory is memory that is allocated outside of the Java heap in native memory.

- It is used when programs need memory that can be directly accessed by native libraries or components.

7. \*Code Cache:\*

- The code cache is used to store compiled native code generated by the Just-In-Time (JIT) compiler.

- Compiled methods are stored here to improve the performance of the application.

**Q23.In which area .class is stored?**

**Ans:** In a Java Virtual Machine (JVM), the class-related information is stored in the \*Method Area, which is also known as the \*\*Class Area\*. The Method Area is a part of the JVM's memory where the JVM stores class structures, metadata, method data, static variables, constant pool, and other class-related data.

When a Java program is executed, the JVM loads class files and stores information about classes and interfaces in the Method Area. This includes details such as method signatures, field names, method code, method and field references, and other metadata required by the program.

It's important to note that in Java 7 and earlier versions, the Method Area was part of the Permanent Generation (PermGen) space. However, starting from Java 8, PermGen was replaced by Metaspace, which also includes the memory allocated for class-related information. Metaspace can dynamically adjust its size, making it more flexible compared to the fixed-size PermGen space.

**Q24.In which area object are stored?**

**Ans**: In a Java Virtual Machine (JVM), objects are stored in the \*Heap Memory\* area. The heap is a runtime data area in which objects are dynamically allocated memory space. It's the area where all objects (instances of classes) and arrays are allocated at runtime.

When you create a new object in Java using the `new` keyword, the JVM allocates memory for that object on the heap. Objects stored in the heap can be accessed by multiple threads, making it a suitable storage area for objects in Java applications.

The heap memory is managed by the Java garbage collector, which is responsible for identifying and reclaiming memory occupied by objects that are no longer reachable or in use. This automatic memory management ensures efficient use of memory and helps prevent memory leaks in Java programs.

**Q25.Why do we call as java simple?**

**Ans:** Java is often considered a simple programming language due to several design choices and features that make it relatively easy to learn and use, especially for beginners. Here are some reasons why Java is often referred to as simple:

1. \*Simple Syntax:\* Java has a clean and straightforward syntax that resembles C++, making it easier for programmers to read, write, and maintain code. It avoids complex features found in other languages, which simplifies the learning curve.

2. \*Object-Oriented:\* Java is purely object-oriented, which means it revolves around objects and classes. This simplifies the organization of code into reusable components and encourages good programming practices.

3. \*Platform Independence:\* Java code is compiled into platform-independent bytecode, which can run on any Java Virtual Machine (JVM). This portability simplifies the development of cross-platform applications.

4. \*Automatic Memory Management:\* Java handles memory management automatically through its garbage collection mechanism. Programmers don't have to worry about manual memory deallocation, reducing the chances of memory-related errors.

5. \*Standard Libraries:\* Java provides a vast set of standard libraries and APIs (Application Programming Interfaces) that simplify complex tasks. These libraries cover everything from data structures to network communication, saving developers time and effort.

6. \*Strongly Typed Language:\* Java is a strongly typed language, which means variables must be declared with a specific data type. This helps catch type-related errors at compile-time, enhancing program reliability.

7. \*Rich Documentation:\* Java comes with extensive documentation and community support. Java's official documentation and numerous online resources simplify the process of finding solutions to common programming problems.

8. \*No Explicit Pointers:\* Java doesn't use explicit pointers, reducing the risk of pointer-related errors such as null pointer exceptions, which can be common in languages like C and C++.

9. \*Security Features:\* Java includes built-in security features like classloaders and bytecode verification, enhancing the overall security of Java applications.