CSC 3315 OBJECT-ORINTED PROGRAMMING-JAVA: (3 Units)

2023/2024 Academic Session

UNIT 1: Overview of Programming Languages

- Introduction to Computer Programming
- Programming Languages
- Programming Paradigms
- Programming Languages Translation
- Review Questions

1.1 Introduction to Computer Programming

Before getting into computer programming, let us first understand computer programs and what they do. A computer program is a sequence of instructions written using a Computer Programming Language to perform a specified task by the computer.

To understand these terms, consider a situation when someone asks you about how to go to a nearby restaurant at a certain point. What exactly do you do to tell him the way to go to the restaurant?

You will use Human Language to tell the way to go to restaurant, something as follows – "First go straight, after half kilometer, take right from the green light and then drive around hundred meters and you will find the restaurant at the left". Here, you have used English Language to give several steps to be taken to reach the restaurant. If they are followed in the following sequence, then you will reach restaurant –

1. Go straight
2. Drive half kilometer
3. Take right
4. Drive around one hundred meters
5. Look for the restaurant at your left side

Now, try to map the situation with a computer program. The above sequence of instructions is actually a **Human Program** written in **English Language**, which instructs on how to reach a nearby restaurant from a given starting point. This same sequence could have been given in Hausa, Arabic, or any other human language, provided the person seeking direction knows any of these languages.

Now, let's go back and try to understand a computer program, which is a sequence of instructions written in a Computer Language to perform a specified task by the computer. Following is a simple program written in **Java** programming Language −

```
System.out.print( "Hello, World!");
```

The above computer program instructs the computer to print "Hello, World!" on the computer screen.

If you understood what a **computer program** is, then we will say: the act of writing computer programs is called computer programming.

1.2 Programming Languages

A programming language is a set of instructions and syntax used to create software programs. It is a formal language that specifies a set of instructions for a computer to perform specific tasks. It's used to write software programs and applications, and to control and manipulate computer systems. There are many different programming languages, each with its own syntax, structure, and set of commands. Some of the most commonly used programming languages include Java, Python, C++, JavaScript, and C#. The choice of programming language depends on the specific requirements of a project, including the platform being used, the intended audience, and the desired outcome. Programming languages continue to evolve and change over time, with new languages being developed and older ones being updated to meet changing needs.

1.2.1 Categories of programming languages:

1. Machine languages.
2. Assembly languages.
3. High-level languages.

Machine Languages

Machine language is a type of low-level programming language. It is also called as **machine code or object code**. Machine language is easier to read because it is normally displayed in binary (0's and 1's), or hexadecimal form (base 16) form. It does not require a translator to convert the programs because computers directly understand the machine language programs.

The advantage of machine language is that it helps the programmer to execute the programs faster than the high-level programming language.

Assembly Language

Assembly language is also a type of low-level programming language that is designed for specific processors. It represents the set of instructions in a **symbolic and human-**

understandable form. It uses an assembler to convert the assembly language to machine language.

The advantage of assembly language is that it requires less memory and less execution time to execute a program.

High-level Languages

High-level programming language (HLL) is designed for **developing user-friendly software programs and websites**. This programming language requires a compiler or interpreter to translate the program into machine language (execute the program).

The main advantage of a high-level language is that it is **easy to read, write, and maintain**. However, it need to be translated to machine code.

High-level programming language includes **Python, Java, JavaScript, PHP, C#, C++, Objective C, Cobol, Perl, Pascal, LISP, FORTRAN, and Swift programming language**.

1.3 Programming Paradigms

Programming paradigm can also be termed as method to solve some problem or do some task. Programming paradigm is an approach to solve problem using some programming language or also we can say it is a method to solve a problem using tools and techniques that are available to us following some approach. There are lots for programming language that are known but all of them need to follow some strategy when they are implemented and this methodology/strategy is paradigms. A programming paradigm may consist of many programming languages. Apart from varieties of programming language there are lots of paradigms to fulfill each and every demand. They are discussed below:

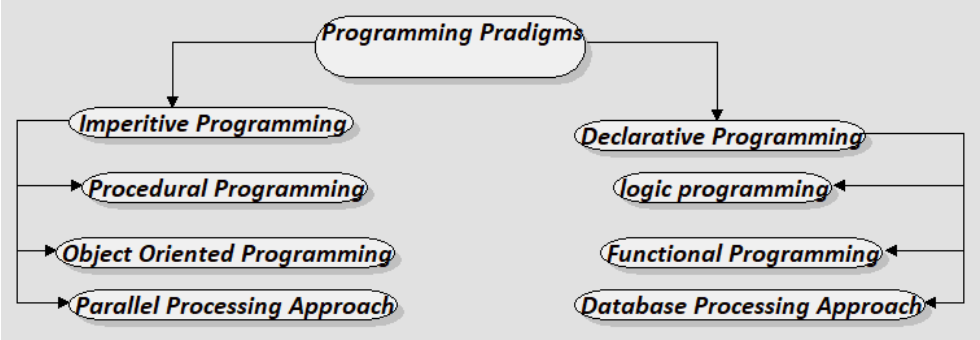


Fig. 1: Programming Paradigm

1. **Imperative programming paradigm:** It is one of the oldest programming paradigm. It features close relation to machine architecture. It is based on Von Neumann architecture. It works by changing the program state through assignment statements. It performs step by step task by changing state. The main focus is on how to achieve the goal. The paradigm consists of several statements and after execution of all the result is stored.

Advantages

- ❖ Very simple to implement
- ❖ It provides code reusability (contains loops, variables etc.)

Disadvantages

- ❖ Complex problem cannot be solved
- ❖ Parallel programming is not possible

Imperative programming is divided into three broad categories: Procedural, OOP and parallel processing. These paradigms are as follows:

- **Procedural programming paradigm –**
  This paradigm emphasizes on procedure in terms of under lying machine model. It has the ability to reuse the code and it was boon at that time when it was in use because of its reusability.

Examples of **Procedural** programming paradigm:
C : developed by Dennis Ritchie and Ken Thompson
Pascal : developed by Niklaus Wirth
FORTRAN: developed by John Backus
Basic: developed by John G. Kemeny and Thomas E. Kurtz

Advantages

- ❖ Low memory utilization
- ❖ Relatively efficient

Disadvantages

- ❖ Difficulty of reasoning about programs
- ❖ Difficulty of parallelization.

- **Object oriented programming (OOP)–**
  OOP language is **based upon the objects**. In this **programming language, programs are divided into small parts called objects**. It is used to implement real-world entities like inheritance, polymorphism, and abstraction in the program to makes the program reusable, efficient, and easy-to-use.

Advantages:

- ❖ Data security
- ❖ Inheritance
- ❖ Code reusability

- ❖ Flexible and abstraction is also present

Disadvantages

- ❖ Can have a steep learning curve, initially
- ❖ Doing I/O can be cumbersome

Examples of **Object Oriented** programming paradigm:
Simula : first OOP language
Java : developed by James Gosling at Sun Microsystems
C++ : developed by Bjarne Stroustrup
Objective-C : designed by Brad Cox
Visual Basic .NET : developed by Microsoft
Python : developed by Guido van Rossum
Ruby : developed by Yukihiro Matsumoto
Smalltalk : developed by Alan Kay, Dan Ingalls, Adele Goldberg

- **Parallel processing approach –**
  Parallel processing is the processing of program instructions by dividing them among multiple processors. A parallel processing system posses many numbers of processor with the objective of running a program in less time by dividing them. This approach seems to be like divide and conquer. Examples are NESL (one of the oldest one) and C/C++ also supports because of some library function.

2. **Declarative programming paradigm:**
   The *declarative programming* is a style of building programs that expresses logic of computation without talking about its control flow. It often considers programs as theories of some logic. It may simplify writing parallel programs. The focus is on what needs to be done rather how it should be done basically emphasize on what code is actually doing. It just declares the result we want rather how it has been produced. This is the only difference between imperative (how to do) and declarative (what to do) programming paradigms. Getting into deeper we would see logic, functional and database as the three categories of Declarative programing paradigm.

   - **Logic programming paradigms –**
     It can be termed as abstract model of computation. It would solve logical problems like puzzles, series etc. In logic programming we have a knowledge base which we know before and along with the question and knowledge base which is given to machine, it produces result. In normal programming languages, such concept of knowledge base is not available but while using the concept of artificial intelligence, machine learning we have some models like Perception model which is using the same mechanism.
     In logical programming the main emphasize is on knowledge base and the problem. The execution of the program is very much like proof of mathematical statement, e.g., Prolog

     Advantages

     - ❖ Good support for reasoning about programs

- Can lead to concise solutions to problems

Disadvantages

- Slow execution
- Limited view of the world--That means the system does not know about facts that are not its predicates and rules of inference.
- Difficulties in understanding and debugging large programs

- **Functional programming paradigms –**
  The functional programming paradigms has its roots in mathematics and it is language independent. The key principle of this paradigms is the execution of series of mathematical functions. The central model for the abstraction is the function which are meant for some specific computation and not the data structure. Data are loosely coupled to functions. The function hide their implementation. Function can be replaced with their values without changing the meaning of the program.

  Examples of **Functional** programming paradigm:

  JavaScript : developed by Brendan Eich
  Haskell : developed by Lennart Augustsson, Dave Barton
  Scala : developed by Martin Odersky
  Erlang : developed by Joe Armstrong, Robert Virding
  Lisp : developed by John Mccarthy
  ML : developed by Robin Milner
  Clojure : developed by Rich Hickey

  Advantages

  - Small and clean syntax
  - Better support for reasoning about programs
  - They allow functions to be treated as any other data values.
  - They support programming at a relatively higher level than the imperative languages

  Disadvantages

  - Difficulty of doing input-output
  - Functional languages use more storage space than their imperative cousins

- **Database/Data driven programming approach –**
  This programming methodology is based on data and its movement. Program statements are defined by data rather than hard-coding a series of steps. A database program is the heart of a business information system and provides file creation, data entry, update, query and reporting functions. There are several programming languages that are developed mostly for database application. For example SQL. It is applied to streams of structured data, for filtering, transforming, aggregating (such as computing statistics), or calling other programs. So it has its own wide application.

  Drill Question:

Which of these Paradigms is Best?

The most accurate answer is that there is no best paradigm.

- No single paradigm will fit all problems well.
- Human beings use a combination of the models represented by these paradigms.
- Languages with features from different paradigms are often too complex.
- So, the search of the ultimate programming language continues!

1.4 Programming Languages Translation

As mentioned earlier, one of the disadvantages of a high-level language is that it must be translated to machine language. High-level languages are translated using language translators.

A language translator translates a high-level language program or an assembly language program into a machine language program.

There are three types of translators:

1. Assemblers.
2. Compilers.
3. Interpreters.

Assemblers

An assembler is a program that translates an assembly language program, written in a particular assembly language, into a particular machine language.

Compilers

A compiler is a program that translates a high-level language program, written in a particular high-level language, into a particular machine language.

Interpreters

An interpreter is a program that translates a high-level language program, one instruction at a time, into machine language. As each instruction is translated it is immediately executed. Interpreted programs are generally slower than compiled programs because compiled programs can be optimized to get faster execution.

Note that: Some high-level languages are compiled while others are interpreted. There are also languages, like Java, which are first complied and then interpreted.

Compilation Process: Traditional Compilers

In the traditional compilation process, the compiler produces machine code for a specific family of processors. For example, given a source program, a compiler for the x86 family of processors will

produce binary files for this family of processors. A disadvantage of this compilation method is that the code produced in each case is not portable (it is not machine independent). To make the resulting code portable, we need the concept of a virtual machine as we discuss below:

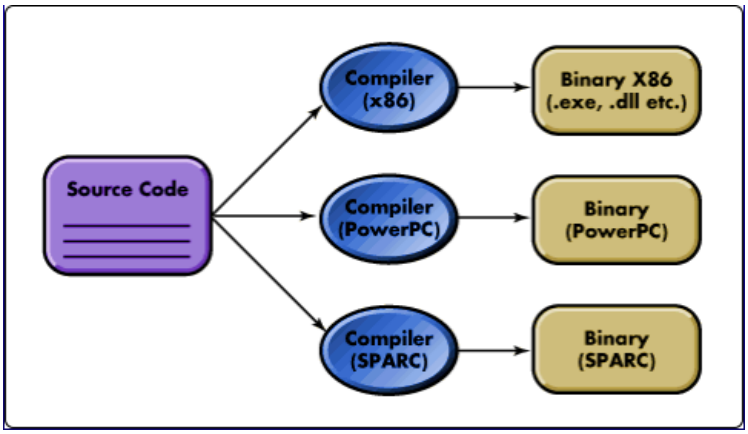


Fig.2: Machine dependent compilation process.

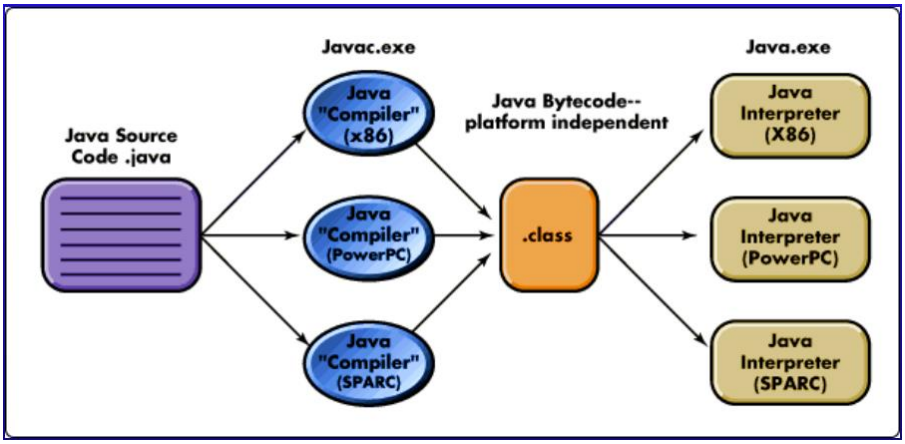


Fig.3: Java compilation process.

Instead of producing a processor-specific code, Java compilers produce an intermediate code called bytecode. The bytecode is also a binary code but is not specific to a particular CPU. A Java compiler will produce exactly the same bytecode no matter what computer system is used. The Java bytecode is then interpreted by the Java Virtual Machine (JVM) interpreter. We will discuss JVM and java execution phases in the subsequent unit.

1.5 Review Questions

1.List two advantages and two disadvantages of low-level languages.

2. Explain the similarities and differences between an assembly language and a machine language.

3. Mention the programming paradigm to which each of the following languages belongs: Visual Basic, Java, C#, Haskell, Lisp, Prolog, Pascal.

4. Which programming paradigms give better support for reasoning about programs?

5. Which programming paradigms give better support for doing I/O?

UNIT 2: Introduction to Java Fundamentals

2.1 History of Java

Java is an Object-Oriented programming language developed by **James Gosling** in the early 1990s. The team initiated this project to develop a language for digital devices such as set-top boxes, television, etc. Originally C++ was considered to be used in the project but the idea was rejected for several reasons (For instance C++ required more memory). Gosling endeavored to alter and expand C++ however before long surrendered that for making another stage called **Green**. James Gosling and his team called their project "**Greentalk**" and its file extension was **.gt** and later became to known as "**OAK**". Gosling and his team did a brainstorm session and after the session, they came up with several names such as **JAVA, DNA, SILK, RUBY, etc. Java** name was decided after much discussion since it was so unique. The name Java originates from a sort of **espresso bean**, Java. Gosling came up with this name while having a coffee near his office. Java was created on the principles like **Robust, Portable, Platform Independent, High Performance, Multithread, etc.** and was called one of the **Ten Best Products of 1995** by the **TIME MAGAZINE**. Currently, Java is used in **internet programming, mobile devices, games, e-business solutions, big Data Technologies, Internet of Things (IOT) etc.**

The Java language has experienced a few changes since **JDK 1.0** just as various augmentations of classes and packages to the standard library. In Addition to the language changes, considerably more sensational changes have been made to the Java Class Library throughout the years, which has developed from a couple of hundred classes in JDK 1.0 to more than three thousand in J2SE 5. There are many java versions that has been released. Current stable release of Java is **Java SE 21.**

2.2 Main Features of Java

I. Simple:

- Java is Easy to write and more readable.
- Java has a concise, cohesive set of features that makes it easy to learn and use.
- Most of the concepts are drawn from C++, thus making Java learning simpler.

II. **Secure:**

• Java program cannot harm other system thus making it secure.

• Java provides a secure means of creating Internet applications.

• Java provides secure way to access web applications.

III. **Portable:**

• Java programs can execute in any environment for which there is a Java run-time system.

• Java programs can run on any platform (Linux, Window, Mac)

• Java programs can be transferred over world wide web (e.g applets).

IV. **Object-oriented:**

• Java programming is object-oriented programming language.

• Like C++, java provides most of the object oriented features.

• Java is pure OOP Language. (while C++ is semi object oriented)

V. **Robust:**

• Java encourages error-free programming by being strictly typed and performing runtime checks.

VI. **Multithreaded:**

• Java provides integrated support for multithreaded programming.

VII. **Architecture-neutral:**

• Java is not tied to a specific machine or operating system architecture.

• Java is machine independent.

VIII. **Interpreted:**

• Java supports cross-platform code through the use of Java bytecode.

• Bytecode can be interpreted on any platform by JVM (Java Virtual Machine).

IX. **High performance:**

• Bytecodes are highly optimized.

• JVM can execute bytecodes much faster.

X. **Distributed:**

• Java is designed with the distributed environment.

• Java can be transmitted over internet.

XI. **Dynamic:**

• Java programs carry substantial amounts of run-time type information with them that is used to verify and resolve accesses to objects at run time.

2.3 Phases to execute Java Program

Fig 4. How java works.

Java program execution follows 5 major steps

- Edit - Here the programmer uses a simple editor or a notepad application to write the java program and in the end give it a ".java" extension.
- Compile - In this step, the programmer gives the javac command and the .java files are converted into bytecode which is the language understood by the Java virtual machine (and this is what makes Java platform independent language). Any compile time errors are raised at this step.
- Load - The program is then loaded into memory. This is done by the class loader which takes the .class files containing the bytecode and stores it in the memory. The .class file can be loaded from your hard disk or from the network as well.
- Verify - The bytecode verifier checks if the bytecode loaded are valid and do not breach java security restrictions.
- Execute - The JVM interprets the program one bytecode at a time and runs the program.

Java Development Kit (JDK) is a Kit that provides the environment to **develop and execute(run)** the Java program.

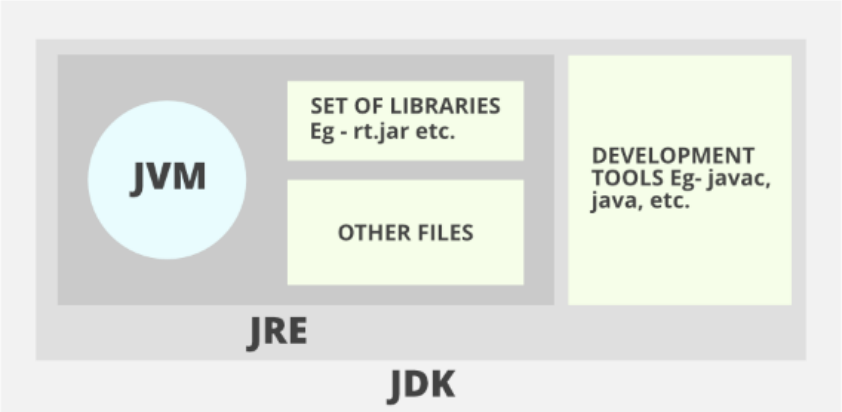


Fig 5. JDK

JDK is a kit (or package) that includes two things

- Development Tools (to provide an environment to develop your java programs)
- JRE (to execute your java program)

Java Runtime Environment(JRE) is an installation package that provides an environment to **only run (not develop)** the java program (or application) onto your machine. JRE is only used by those who only want to run Java programs that are end-users of your system.

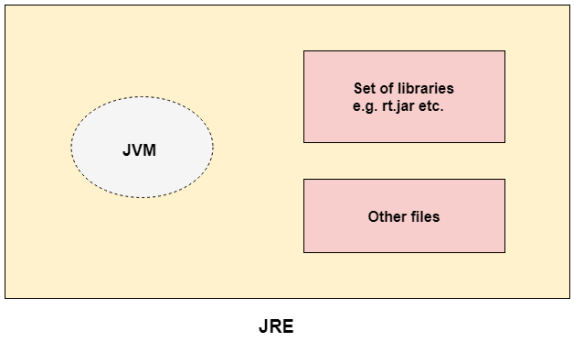


Fig.6: JRE

Now let us discuss the components of JRE in order to understand its importance and perceive how it actually works.

The components of JRE are as follows:

1. **Deployment technologies**, including deployment, Java Web Start, and Java Plug-in.
2. **User interface toolkits**, including *Abstract Window Toolkit (AWT), Swing, Java 2D, Accessibility, Image I/O, Print Service, Sound, drag, and drop (DnD)*, and *input methods*.
3. **Integration libraries**, including *Interface Definition Language (IDL), Java Database Connectivity (JDBC), Java Naming and Directory Interface (JNDI), Remote Method Invocation (RMI), Remote Method Invocation Over Internet Inter-Orb Protocol (RMI-IIOP)*, and *scripting*.
4. **Other base libraries**, including *international support, input/output (I/O), extension mechanism, Beans, Java Management Extensions (JMX), Java Native Interface (JNI), Math, Networking, Override Mechanism, Security, Serialization*, and *Java for XML Processing (XML JAXP)*.
5. **Lang and util base libraries**, including *lang and util, management, versioning, zip, instrument, reflection, Collections, Concurrency Utilities, Java Archive (JAR), Logging, Preferences API, Ref Objects*, and *Regular Expressions*.
6. **Java Virtual Machine (JVM)**, including *Java HotSpot Client* and *Server Virtual Machines*.

Java Virtual Machine (JVM) is a very important part of both JDK and JRE because it is contained or inbuilt in both. Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for executing the java program line by line, hence it is also known as an interpreter.

- JVM (Java Virtual Machine) acts as a run-time engine to run Java applications. JVM is the one that actually calls the **main** method present in a java code. JVM is a part of JRE (Java Runtime Environment).
- Java applications are called WORA (Write Once Run Anywhere). This means a programmer can develop Java code on one system and can expect it to run on any other Java-enabled system without any adjustments. This is all possible because of JVM.
- When we compile a *.java* file, *.class* files (contains byte-code) with the same class names present in *.java* file are generated by the Java compiler. This *.class* file goes into various steps when we run it. These steps together describe the whole JVM.

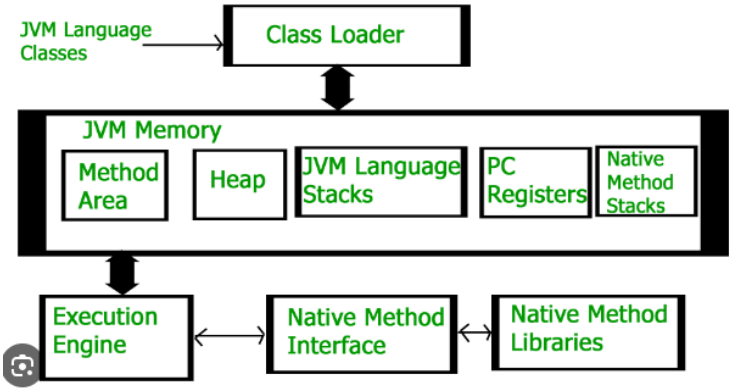


Fig.7: JVM Architecture.

2.4 Setting Up an Environment for Java

2.4.1 Local environment setup (Windows OS)

Download Java and run the .exe to install Java on the machine, you can follow the steps below:

Step 1: https://www.oracle.com/java/technologies/downloads/#jdk21-windows

copy the above address, paste in a browser and hit enter

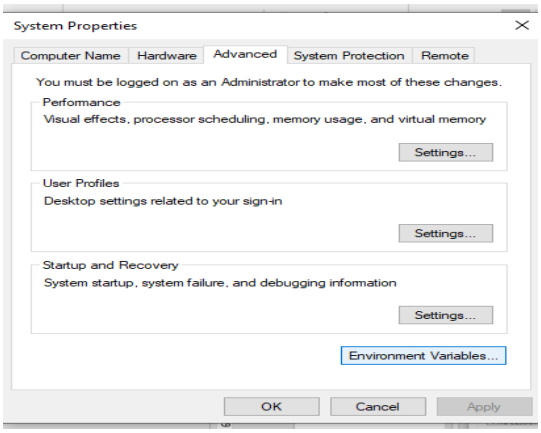
step 2: click on the download link under x64MSI installer to download the JDK

step 3: open the download folder in your PC, locate to the download file, double click on it and follow the on-screen instructions to install it.

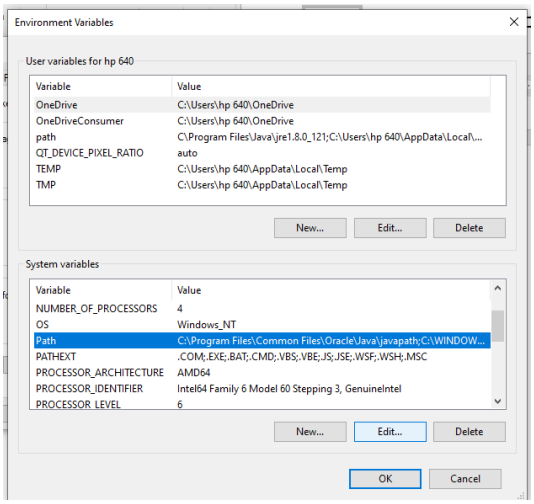
setting up the Path for Windows Assuming Java is installed in c:\Program Files\java\jdk directory

Step 4: Locate **Advanced system properties** in the **Control panel** of your PC and click on it to open.

Step 5: click on **environmental variables** then **ok**, like below:



Step 6: click on **path** then **edit,** like below:

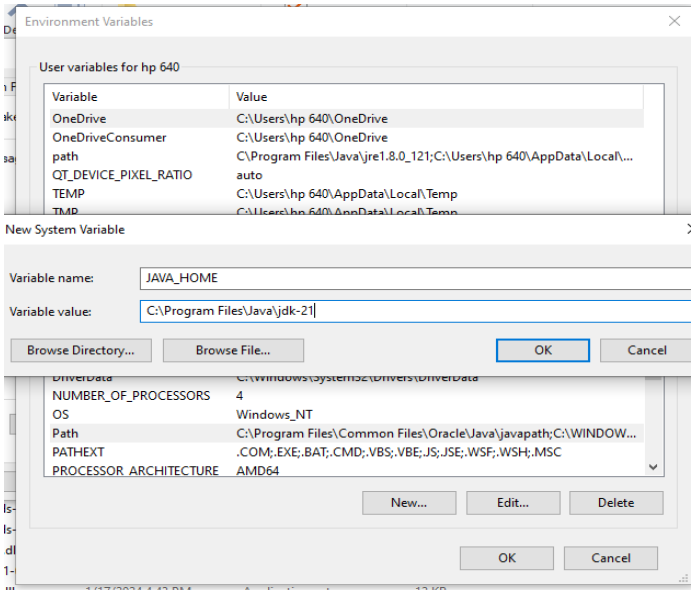


Step 7: minimize the window and locate the path of the installed java **bin** folder in your C drive then copy it. See below for example:

C:\Program Files\Java\jdk-21\bin

Step 8: go back to the minimize page, click on **new** then paste the copied path in step 7 . The path will appear the list then click **ok** .

Step 9: click on **new** under **system variables** and write JAVA_HOME in the **variable name,** then paste the copied path in **step 7** under the **variable value.** Note: you have to delete **bin folder** after pasting, see below for example.



Step 10: click **ok** to save all changes. Congratulations your java is ready !

2.4.2 Popular Java Editors

To write Java programs, we need any of the following:

- notepad − Text editor
- NetBeans − A Java IDE that is open-source and free
- eclipse − A Java IDE developed by the eclipse open-source community
- InteliJ- A Java IDE developed by Apache 2 licensed community

2.4.3 Structure of Simple Java Programs

The following shows a simplified structure of Java programs. We will consider a more detailed structure of Java programs later in this course.

```
public class ClassName {
        public static void main(String[] args ){
         statement 1;
         statement 2;
         * * *
        statement N
        }
}
```

A Java program consists of essential elements called **classes**. The classes in a program are used to create specific things called **objects.** An objects is any entity that has a **state** and **behavior** in the real word.

Source Program: Example

```
public class Greeting {
        public static void main(String[] args ){
                System.out.println("Good Morning.");
        }
}
```

- You must type this program and save it in a file named **Greeting.java**
- Java is case sensitive and has a free-form layout
- The words public, class, static, void, main and etc are called reserved or keyword words

The meaning of the words is fixed by the language. For now, they must appear in the places shown.

- By contrast, the word ***Greeting***, varies from program to program. What exactly goes there is chosen by the programmer.
- The first line, *public class Greeting*, starts a new class called Greeting.
- Classes are a fundamental concept in java. Their role is as factories for objects.

But here we are using the *Greeting class* as a container for our program's instructions.

- Java requires that all program instructions be placed inside **methods** and that every method must be placed inside a **class.** Thus we must define methods that would contain our instructions and a class that holds the methods.

In our program, *main()* is our only method while *Greeting* is our class. At this point, simply regard

```
public class ClassName{
...
}
```

as a necessary part of the plumbing that is required to write a java program.

The construction

```
public static void main (String[] args){
...
```

}

is where we define the method *main ()*

The parameter *String[] args* is a required part of the *main* method. At this time, simply consider

public class ClassName{

public static void main (String[] args){

...

}

}

as yet another part of the plumbing for the time being,

simply put all instructions between the curly braces {} of the *main()* method.

- There is no limit to the number of instructions that can be placed inside the body of the *main* method. Our program contains only one instruction, that is

The instruction,

System.out.println("Good Morning");

Which prints a line of text namely "Good Morning".

A program can send the string : to a window, to a file, to a networked computer. However, our program prints the string to the terminal window. That is the monitor.

Terminal window is represented in java by an object called ***out*** and ***out*** object is contained in the ***System class.***

The **System class** contains useful **objects** and **methods** to access System resources. To use the **out object** in the **System class**, you must to refer to it as *System.out.*

To use *System.out* object, specify what you want to do to it. In this case, you want to print a line of text. The ***println()*** method carries out this.

The ***println()*** method prints a string or a number and then start a new line. For example:

- The sequence Statements:

  System.out.println("Good");

  System.out.println("Morning.");

  Prints two lines of text

  Good

  Morning.

- The statement:

System.out.println(2+3);

Prints the number 5

There is a second method called ***print(),*** which print an item without starting a new line.

For example:

System.out.print("Good");

System.out.print("Morning.");

Prints a single line

Good Morning.

2.5 Access Modifiers, Comments, Data types, Variables & Constants

2.5.1 Access Modifiers

Access specifiers or access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class. It determines whether a data or method in a class can be used or invoked by other class or subclass. types of access Specifiers

There are 4 types of java access specifiers:

1. Private
2. Default (no specifier)
3. Protected
4. Public

The details about accessibility level for access specifiers are shown in following table.

Access Modifiers	Default	Private	Protected	Public
Accessible inside the class	Yes	Yes	Yes	Yes
Accessible within the subclass inside the same package	Yes	No	Yes	Yes
Accessible outside the package	No	No	No	Yes
Accessible within the subclass outside the package	No	No	Yes	Yes

In the subsequent sections, we will discuss access modifiers in details.

2.5.2 Java comments

The java comments are statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code for specific time.

There are 3 types of comments in java.

1. Single Line Comment
2. Multi Line Comment
3. Documentation Comment

1) Java Single line comment is used to comment only one line. A single-line comment begins with a // and ends at the end of the line.

Syntax	Example
//Comment	//This is single line comment

2) Java multi line comment This type of comment must begin with /* and end with */. Anything between these two comment symbols is ignored by the compiler. A multiline comment may be several lines long.

Syntax	**Example**
/*Comment starts continues continues . . . Commnent ends*/	/* This is a multi-line comment */

3)) Java Documentation comment This type of comment is used to produce an HTML file that documents our program. The documentation comment begins with a /** and ends with a */.

Syntax	**Example**
/**Comment start . . . comment ends*/	/** This is documentation comment */

2.5.3 Data types

Java is a statically typed and also a strongly typed language. Each type of data (such as integer, character, hexadecimal, etc.) is predefined as part of the programming language and all constants or variables defined within a given program must be described with one of the data types. Data

types represent the different values to be stored in the variable. In java, there are two categories of data types:

- Primitive data types
- Non-primitive data types

Java has eight **primitive data types** as described below.

Data Types	Default value	Default size	Range
Boolean	**False**	**1 bytes**	**True or false**
byte	**0**	**1 bytes**	**$(-2^{(n-1)})$ to $(2^{(n-1)}-1)$**
short	**0**	**2 bytes**	**$(-2^{(n-1)})$ to $(2^{(n-1)}-1)$**
int	**0**	**4 bytes**	**$(-2^{(n-1)})$ to $(2^{(n-1)}-1)$**
long	**0L**	**8 bytes**	**$(-2^{(n-1)})$ to $(2^{(n-1)}-1)$**
float	**0.0f**	**4 bytes**	**$\pm$3.4E38**
double	**0.0**	**8 bytes**	**$\pm$1.7E308**
char	**'\u0000'**	**2 bytes**	**A single character**

Non-primitive data types are called **reference types** because they refer to objects. Examples of non-primitive types are Strings, Arrays, Classes, Interface, etc. You will learn more about these in a later unit.

2.5.4 Variables

A variable is the holder that can hold the value while the java program is executed. A variable is assigned with a datatype. It is name of reserved area allocated in memory. In other words, it is a name of memory location.

A variable provides us with named storage that our programs can manipulate. Each variable in Java has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable. Before using any variable, it must be declared.

Example

int a, b, c; // Declaration of variables a, b, and c.

int a = 20, b = 30; // initialization

byte B = 22; // Declaratrion initializes a byte type variable B

Scope and Life Time of Variables

The scope of a variable defines the section of the code in which the variable is visible. As a general rule, variables that are defined within a block are not accessible outside that block. **The lifetime of a variable** refers to how long the variable exists before it is destroyed. Destroying variables refers to deallocating the memory that was allotted to the variables when declaring it. We have written a few classes till now. You might have observed that not all variables are the same. The ones declared in the body of a method were different from those that were declared in the class itself.

There are three types of variables in java:

- local variable
- instance variable
- static variable

1)Local Variable

- Local variables are declared inside the methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered
- Local variable will be destroyed once it exits the method, constructor, or block.
- Local variables are visible only within the declared method, constructor, or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.
- Access specifiers cannot be used for local variables.
- Scope of a local variable is within the block in which it is declared.
- The lifetime of a local variable is until the control leaves the block in which it is declared.

2) Instance Variable

- A variable declared inside the class but outside the method, is called instance variable. Instance variables are declared in a class, but outside a method, constructor or any block.
- A slot for each instance variable value is created when a space is allocated for an object in the heap.
- Instance variables are created when an object is created with the use of the keyword ‘new’ and destroyed when the object is destroyed.

- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. It is recommended to make these variables as private. However, visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values.
    - numbers, the default value is 0,
    - Booleans it is false,
    - Object references it is null.
- Values can be assigned during the declaration or within the constructor.
- Instance variables cannot be declared as static.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. ObjectReference.VariableName.
- The scope of instance variables is determined by the access specifier that is applied to these variables.
- The lifetime of these variables is the same as the lifetime of the object to which it belongs. Object once created do not exist for ever. They are destroyed by the garbage collector of Java when there is no more reference to that object.

3)Static/ class variable

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- Only one copy of each class variable per class is created, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.

- Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is same as instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables.
    - numbers, the default value is 0;
    - Booleans, it is false;
    - Object references, it is null.
- Values can be assigned during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks
- Static variables cannot be local.
- Static variables can be accessed by calling with the class name ClassName. VariableName.
- When declaring class variables as *public static final*, then variable names (constants) are all in upper case. If the static variables are not public and final, the naming syntax is the same as instance and local variables.
- The general scope of a static variable is throughout the class.
- The lifetime of a static variable is until the end of the program or as long as the class is loaded in memory.

Example to understand the types of variables in java

```
public class TypesOfVariables{
    int data=50;//instance variable
    static int m=100;//static variable
    public void method(){
        int n=90;//local variable
    }
}//end of class
```

2.5.4 **Constants in Java**

A constant is a variable which cannot have its value changed after declaration. It uses the 'final' keyword. Syntax :

modifier *final* dataType variableName = value; //global constant

2.6 Operators, Control Statements and Methods in Java

2.6.1 Operators

Operator in java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in java which are given below:

Unary Operator:

Postfix : expr++ expr- -

Prefix : ++expr - -expr +expr -expr

Arithmetic Operator:

Multiplicative : * / %

Additive : + -

Relational Operator:

Comparison : < > <= >=

instanceof Equality : == !=

Bitwise Operator :

bitwise AND : &

bitwise XOR : ^

bitwise OR : |

Shift operator: << >> >>>

Logical Operator:

logical AND : &&

logical OR : ||

logical NOT : ~ !

Ternary Operator : ? :

Assignment Operator:

Assignment Operators: =

1. Unary Operator

Unary operators need only one operand. They are used to increment, decrement, or negate a value.

- **– : Unary minus**, used for negating the values.

- **+ : Unary plus** indicates the positive value (numbers are positive without this, however). It performs an automatic conversion to int when the type of its operand is the byte, char, or short. This is called unary numeric promotion.
- **++ : Increment operator**, used for incrementing the value by 1. There are two varieties of increment operators.
    - **Post-Increment:** Value is first used for computing the result and then incremented.
    - **Pre-Increment:** Value is incremented first, and then the result is computed.
- **– – : Decrement operator**, used for decrementing the value by 1. There are two varieties of decrement operators.
    - **Post-decrement:** Value is first used for computing the result and then decremented.
    - **Pre-Decrement: The value** is decremented first, and then the result is computed.
- **! : Logical not operator**, used for inverting a boolean value.

```
// Java Program to implement Unary Operators
class Unary {
    public static void main(String[] args)
    {
        int a = 10;
        int b = 10;

        System.out.println("Post increment: " + (a++)); //Post increment: 10
        System.out.println("Pre increment: " + (++a)); // Pre increment:  12
        System.out.println("Post decrement: " + (b--)); //Post decrement: 10
        System.out.println("Pre decrement: " + (--b)); // Post decrement: 8
    }
}
```

2. Arithmetic Operator

   the arithmetic Operators Arithmetic operators are used to perform arithmetic operations in the same way as they are used in algebra.

```
// Java Program to implement Arithmetic Operators
class Arithmatic{
      // Main Function
    public static void main (String[] args) {

        // Arithmetic operators
        int a = 10;
        int b = 3;

        System.out.println("a + b = " + (a + b)); // a + b = 13
        System.out.println("a - b = " + (a - b)); // a - b = 7
        System.out.println("a * b = " + (a * b)); // a * b = 30
        System.out.println("a / b = " + (a / b)); // a / b = 3
        System.out.println("a % b = " + (a % b)); // a % b = 1
```

```
    }
}
```

3. Relational Operator

These operators are used to check for relations like equality, greater than, and less than. They return boolean results after the comparison and are extensively used in looping statements as well as conditional if-else statements. The general format is,

variable **relation_operator** value

Some of the relational operators are-

- **==, Equal to** returns true if the left-hand side is equal to the right-hand side.
- **!=, Not Equal to** returns true if the left-hand side is not equal to the right-hand side.
- **<, less than:** returns true if the left-hand side is less than the right-hand side.
- **<=, less than or equal to** returns true if the left-hand side is less than or equal to the right-hand side.
- **>, Greater than:** returns true if the left-hand side is greater than the right-hand side.
- **>=, Greater than or equal to** returns true if the left-hand side is greater than or equal to the right-hand side.

```
// Java Program to implement relational Operators
class Relational{
    public static void main(String[] args)
    {
        // Comparison operators
        int a = 10;
        int b = 3;
        int c = 5;

        System.out.println("a > b: " + (a > b)); // a > b: true
        System.out.println("a < b: " + (a < b)); // a < b: false
        System.out.println("a >= b: " + (a >= b)); // a >= b: true
        System.out.println("a <= b: " + (a <= b)); // a <= b: false
        System.out.println("a == c: " + (a == c)); // a == c: false
        System.out.println("a != c: " + (a != c)); // a != c: true
    }
}
```

4. Bitwise Operator

These operators are used to perform the manipulation of individual bits of a number. They can be used with any of the integer types. They are used when performing update and query operations of the Binary indexed trees.

- **&, Bitwise AND operator:** returns bit by bit AND of input values.
- **|, Bitwise OR operator:** returns bit by bit OR of input values.
- **^, Bitwise XOR operator:** returns bit-by-bit XOR of input values.
- **~, Bitwise Complement Operator:** This is a unary operator which returns the one's complement representation of the input value, i.e., with all bits inverted.

```
// Java Program to implement bitwise Operators
class Bitwise {
```

```
    public static void main(String[] args)
    {
        // Bitwise operators
        int d = 10;
        int e = 12;
        System.out.println("d & e: " + (d & e)); // d & e: 8

        System.out.println("d | e: " + (d | e)); // d | e: 14

        System.out.println("d ^ e: " + (d ^ e)); // d ^ e: 6

        System.out.println("~d: " + (~d)); // ~d: -11

    }
}
```

5. Shift Operator

   These operators are used to shift the bits of a number left or right, thereby multiplying or dividing the number by two, respectively. They can be used when we have to multiply or divide a number by two. General format-
   number **shift_op** number_of_places_to_shift;
   - **<<, Left shift operator:** shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as multiplying the number with some power of two.
   - **>>, Signed Right shift operator:** shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit depends on the sign of the initial number. Similar effect to dividing the number with some power of two.
   - **>>>, Unsigned Right shift operator:** shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit is set to 0.

```
// Java Program to implement shift Operators
class Shift {
    // main function
    public static void main(String[] args)
    {
        int a = 10;
        int b = 12;
        System.out.println("a<<2 : " + (a << 2)); // a<<2:40
        System.out.println("b>>1 : " + (b >> 1)); // b>>1:6
        System.out.println("b>>>1 : " + (b >>> 1)); // b>>>1: 6
    }
}
```

6. Logical Operator

These operators are used to perform “logical AND” and “logical OR” operations, i.e., a function similar to AND gate and OR gate in digital electronics. One thing to keep in mind is the second condition is not evaluated if the first one is false, i.e., it has a short-circuiting effect. Used extensively to test for several conditions for making a decision. Java also has “Logical NOT”, which returns true when the condition is false and vice-versa
Conditional operators are:

- **&&, Logical AND:** returns true when both conditions are true.
- **||, Logical OR:** returns true if at least one condition is true.
- **!, Logical NOT:** returns true when a condition is false and vice-versa

```
// Java Program to implement Logical operators
class Logical {
    public static void main (String[] args) {
        boolean x = true;
        boolean y = false;

        System.out.println("x && y: " + (x && y)); // x && y: false
        System.out.println("x || y: " + (x || y)); // x || y: true
        System.out.println("!x: " + (!x)); // !x: false
    }
}
```

7. Ternary Operator

   The ternary operator is a shorthand version of the if-else statement. It has three operands and hence the name Ternary.
   The general format is:
   condition **?** if true **:** if false

The above statement means that if the condition evaluates to true, then execute the statements after the '?' else execute the statements after the ':'.

```
// Java Program to implement ternary operators
public class Ternary {
    public static void main(String[] args)
    {
        int a = 20, b = 10, c = 30, result;

        // result holds max of three numbers
        result= ((a > b) ? (a > c) ? a : c : (b > c) ? b : c);
        System.out.println("Max of three numbers = "+ result);
    }
}
```

Output
Max of three numbers = 30

8. Assignment Operator

   '=' Assignment operator is used to assign a value to any variable. It has right-to-left associativity, i.e. value given on the right-hand side of the operator is assigned to the variable on the left, and therefore right-hand side value must be declared before using it or should be a constant.
   The general format of the assignment operator is:
   variable = value;

In many cases, the assignment operator can be combined with other operators to build a shorter version of the statement called a **Compound Statement**. For example, instead of a = a+5, we can write a += 5.

- **+=**, for adding the left operand with the right operand and then assigning it to the variable on the left.
- **-=**, for subtracting the right operand from the left operand and then assigning it to the variable on the left.
- ***=**, for multiplying the left operand with the right operand and then assigning it to the variable on the left.
- **/=**, for dividing the left operand by the right operand and then assigning it to the variable on the left.
- **%=**, for assigning the modulo of the left operand by the right operand and then assigning it to the variable on the left.

```
// Java Program to implement Assignment Operators
class Assignment {
    public static void main(String[] args)
    {

        // Assignment operators
        int f = 7;
        System.out.println("f += 3: " + (f += 3)); // f += 3: 10
        System.out.println("f -= 2: " + (f -= 2)); // f -= 2: 8
        System.out.println("f *= 4: " + (f *= 4)); // f *= 4: 32
        System.out.println("f /= 3: " + (f /= 3)); // f /= 3: 10
        System.out.println("f %= 2: " + (f %= 2)); // f %= 2: 0
        System.out.println("f &= 0b1010: " + (f &= 0b1010)); //f &=0b1010: 0
        System.out.println("f |= 0b1100: " + (f |= 0b1100));//f |= 0b1100:12
        System.out.println("f ^= 0b1010: " + (f ^= 0b1010));//f ^= 0b1010: 6
        System.out.println("f <<= 2: " + (f <<= 2)); // f <<= 2: 24
        System.out.println("f >>= 1: " + (f >>= 1)); // f >>= 1: 12
        System.out.println("f >>>= 1: " + (f >>>= 1)); // f >>>= 1: 6
    }
}
```

Operators Precedence Rules

Precedence and associative rules are used when dealing with hybrid equations involving more than one type of operator. In such cases, these rules determine which part of the equation to consider first, as there can be many different valuations for the same equation. The below table depicts the precedence of operators in decreasing order as magnitude, with the top representing the highest precedence and the bottom showing the lowest precedence.

Operators	Associativity	Type
++ --	Right to left	Unary postfix
++ -- + - ! (type)	Right to left	Unary prefix
/ * %	Left to right	Multiplicative
+ -	Left to right	Additive
< <= > >=	Left to right	Relational
== !=	Left to right	Equality
&	Left to right	Boolean Logical AND
^	Left to right	Boolean Logical Exclusive OR
|	Left to right	Boolean Logical Inclusive OR
&&	Left to right	Conditional AND
||	Left to right	Conditional OR
?:	Right to left	Conditional
= += -= *= /= %=	Right to left	Assignment

Precedence Table

2.6.2 Control Statements

2.6.3 Methods