Unit I

Programming Paradigm of Languages

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. Apart from varieties of programming language there are lots of paradigms to fulfill each and every demand. They are discussed below:

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 consist of several statements and after execution of all the result is stored.

Advantages:
1. Very simple to implement
2. It contains loops, variables etc.

Disadvantage:
1. Complex problem cannot be solved
2. Less efficient and less productive
3. Parallel programming is not possible

- Object oriented programming –
  The program is written as a collection of classes and object which are meant for communication. The smallest and basic entity is object and all kind of computation is performed on the objects only. More emphasis is on data rather procedure. It can handle almost all kind of real life problems which are today in scenario.

Advantages:
- Data security
- Inheritance
- Code reusability
- Flexible and abstraction is also present

Examples of Object Oriented programming paradigm:

- C++
- Java
- Python3, C#, Javascript
  - 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:

It is divided as Logic, Functional, Database. In computer science 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 be 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.

- 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
- 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. Some of the languages like perl, javascript mostly uses this paradigm.

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
The next kind of approach is of Database.

- 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.

Difference between Logical programming Paradigm and Functional Programming Paradigm

Functional Programming	Logical Programming
It is totally based on functions.	It is totally based on formal logic.
In this programming paradigm, programs are constructed by applying and composing functions.	In this programming paradigm, program statements usually express or represent facts and rules related to problems within a system of formal logic.
These are specially designed to manage and handle symbolic computation and list processing applications.	These are specially designed for fault diagnosis, natural language processing, planning, and machine learning.
Its main aim is to reduce side effects that are accomplished by isolating them from the rest of the software code.	Its main aim is to allow machines to reason because it is very useful for representing knowledge.

Functional Programming	Logical Programming
Some languages used in functional programming include Clojure, Wolfram Language, Erland, OCaml, etc.	Some languages used for logic programming include Absys, Cycl, Alice, ALF (Algebraic logic functional programming language), etc.
It reduces code redundancy, improves modularity, solves complex problems, increases maintainability, etc.	It is data-driven, array-oriented, used to express knowledge, etc.
It usually supports the functional programming paradigm.	It usually supports the logic programming paradigm.
Testing is much easier as compared to logical programming.	Testing is comparatively more difficult as compared to functional programming.
It simply uses functions.	It simply uses predicates. Here, the predicate is not a function i.e., it does not have a return value.

Difference between Imperative programming and Declarative Programming

Imperative Programming	Declarative Programming
In this, programs specify how it is to be done.	In this, programs specify what is to be done.
It simply describes the control flow of computation.	It simply expresses the logic of computation.

Imperative Programming	Declarative Programming
Its main goal is to describe how to get it or accomplish it.	Its main goal is to describe the desired result without direct dictation on how to get it.
Its advantages include ease to learn and read, the notional model is simple to understand, etc.	Its advantages include effective code, which can be applied by using ways, easy extension, high level of abstraction, etc.
Its type includes procedural programming, object-oriented programming, parallel processing approach.	Its type includes logic programming and functional programming.
In this, the user is allowed to make decisions and commands to the compiler.	In this, a compiler is allowed to make decisions.
It has many side effects and includes mutable variables as compared to declarative programming.	It has no side effects and does not include any mutable variables as compared to imperative programming.
It gives full control to developers that are very important in low-level programming.	It may automate repetitive flow along with simplifying code structure.
In imperative programming, the programmer is responsible for optimizing the code for performance	In declarative programming, the system optimizes the code based on the rules and constraints specified by the programmer.
In imperative programming, variables can be mutable.	In declarative programming, variables are typically immutable.

Procedural and Non Procedural Programming Language

Procedural Language	Non-Procedural Language
It is command-driven language.	It is a function-driven language
It works through the state of machine.	It works through the mathematical functions.
Its semantics are quite tough.	Its semantics are very simple.
It returns only restricted data types and allowed values.	It can return any datatype or value
Overall efficiency is very high.	Overall efficiency is low as compared to Procedural Language.
Size of the program written in Procedural language is large.	Size of the Non-Procedural language programs are small.
It is not suitable for time critical applications.	It is suitable for time critical applications.
Iterative loops and Recursive calls both are used in the Procedural languages.	Recursive calls are used in Non-Procedural languages.

Proprietary standards

Proprietary standards are developed by experts within certain fields, industries or associations. Consensus standards are discussed and agreed upon by a consensus for impacted industries.

Consensus standard

Consensus standards are developed in cooperation with all parties with an interest in participating in the development or use of the standard. To achieve consensus, all views and objections must be considered and a demonstrated effort must be made toward resolution.

What Are Data Types and Why Are They Important?

Data Types : A data type is an attribute associated with a piece of data that tells a computer system how to interpret its value.

Data type	Used for	Examples
String	Alphanumeric characters	hello world Alice Bob123
Integer	Whole numbers	7 12 999
Float (floating point)	Numbers with a decimal point	3.15 9.06 00.13
Character	Encoding text numerically	97 (in ASCII, 97 indicates a lowercase *a*)

Data type	Used for	Examples
Boolean	Representing logical values	TRUE FALSE

Explain Premitive Data Types and Non Premitive Data Types

Primitive	Non-Primitive
Primitive Data types are predefined.	Non-Primitive data types are created by the programmer
Primitive Data types will have certain values.	Non-Primitive data types can be NULL.
Size depends on the type of data structure.	Size is not fixed
Examples are numbers and strings.	Examples are Array and Linked List.
It can start with a lowercase.	It can start with uppercase.

Composite Data Types

Composite data types are data types that have one or more fields dynamically linked to fields in another data type. Composite data types are useful for creating a single data type that references information in more than one data source or that references more than one table or other structure in a single data source.

Composite data type examples

Example of composite data type is array. **int a[ ] = {1,2,3,4,5}**; In above example, is composite because an array-of-int value is comprised of some number of element values chosen from the int type. Using composite data types, we can manage multiple pieces of related data as a single datum.

Record and objects are composite data type.

Functions are technically a type of object and are therefore considered composite data,

List the Steps involved in analysis of program

analysis consists of three phases:

1. Lexical Analysis
2. Syntax Analysis
3. Semantic Analysis

1.Lexical analysis:

In a compiler linear analysis is called lexical analysis or scanning. The lexical analysis phase reads the characters in the source program and grouped into them tokens that are sequence of characters having a collective meaning.

2.Syntax analysis:

Hierarchical Analysis is called parsing or syntax analysis. It involves grouping the tokens of the source program into grammatical phrases that are used by the complier to synthesize output. They are represented using a syntax tree

A **syntax tree** is the tree generated as a result of syntax analysis in which the interior nodes are the operators and the exterior nodes are the operands.

3.Semantic analysis :

This phase checks the source program for semantic errors and gathers type information for subsequent code generation phase. An important component of semantic analysis is type checking. Here the compiler checks that each operator has operands that are permitted by the source language specification.

Structure of Java Virtual Machine

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 adjustment. 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.

Include Diagram for Java Virtual Machine

Class Loader Subsystem
It is mainly responsible for three activities.

- Loading
- Linking
- Initialization

Loading: The Class loader reads the ".class" file, generate the corresponding binary data and save it in the method area. For each ".class" file, JVM stores the following information in the method area.

- The fully qualified name of the loaded class and its immediate parent class.
- Whether the ".class" file is related to Class or Interface or Enum.
- Modifier, Variables and Method information etc.

After loading the ".class" file, JVM creates an object of type Class to represent this file in the heap memory. Please note that this object is of type Class predefined in java.lang package. These Class object can be used by the programmer for getting class level information like the name of the class, parent name, methods and variable information etc. To get this object reference we can use getClass() method of Object class

About JDK

The Java Development Kit (JDK) is a cross-platformed software development environment that offers a collection of tools and libraries necessary for developing Java-based software applications and applets. It is a core package used in Java, along with the **JVM (Java Virtual Machine)** and the JRE (Java Runtime Environment).
Beginners often get confused with JRE and JDK, if you are only interested in running Java programs on your machine then you can easily do it using Java Runtime Environment. However, if you would like to develop a Java-based software application then along with JRE you may need some additional necessary tools, which is called JDK.

JDK=JRE+Development Tools

JDK contains:

- Java Runtime Environment (JRE),
- An interpreter/loader (Java),
- A compiler (javac),
- An archiver (jar) and many more.

The Java Runtime Environment in JDK is usually called Private Runtime because it is separated from the regular JRE and has extra content. The Private Runtime in JDK contains a JVM and all the class libraries present in the production environment, as well as additional libraries useful to developers, e.g, internationalization libraries and the IDL libraries.

Why Java is platform Independent

The meaning of Java platform-independent is that the Java compiled code(byte code) can run on all operating systems. A program is written in a language that is a human-readable language. It may contain words, phrases, etc which the machine does not understand. For the source code to be understood by the machine, it needs to be in a language understood by machines, typically a machine-level language. So, here comes the role of a compiler. The compiler converts the high-level language (human language) into a format understood by the machines.

Therefore, a compiler is a program that translates the source code for another program from a programming language into executable code. This executable code may be a sequence of machine instructions that can be executed by the CPU directly, or it may be an intermediate representation that is interpreted by a virtual machine. This intermediate representation in Java is the Java Byte Code.

Step-by-Step Execution of Java Program

- Whenever a program is written in JAVA, the javac compiles it.
- The result of the JAVA compiler is the .class file or the bytecode and not the machine's native code (unlike the C compiler).
- The bytecode generated is a non-executable code and needs an interpreter to execute on a machine. This interpreter is the JVM and thus the Bytecode is executed by the JVM.
- And finally, the program runs to give the desired output.
- **Include Diagram for program execution**

Why Java is platform-independent but JVM is platform dependent?

In Java, the main point here is that the JVM depends on the operating system – so if you are running Mac OS X you will have a different JVM than if you are running Windows or some other operating system. This fact can be verified by trying to download the JVM for your particular machine – when trying to download it, you will be given a list of JVMs

corresponding to different operating systems, and you will obviously pick whichever JVM is targeted for the operating system that you are running. So we can conclude that JVM is platform-dependent and it is the reason why Java is able to become "Platform Independent".

Important Points:

- In the case of Java, it is the magic of Bytecode that makes it platform-independent.
- This adds to an important feature in the JAVA language termed portability. Every system has its own JVM which gets installed automatically when the JDK software is installed. For every operating system separate JVM is available which is capable to read the .class file or byte code.
- An important point to be noted is that while JAVA is a platform-independent language, the JVM is platform-dependent. Different JVM is designed for different OS and byte code is able to run on different OS.

Language standardization and Issues with it

In computer programming, a programming language specification (or standard or definition) is a documentation artifact that defines a programming language so that users and implementors can agree on what programs in that language mean.

Specifications are typically detailed and formal, and primarily used by implementors, with users referring to them in case of ambiguity; the C++ specification is frequently cited by users, for instance, due to the complexity. documentation includes a programming language reference, which is intended expressly for users, and a programming language rationale, which explains why the specification is written as it is; these are typically more informal than a specification.

Unit II

String :

String is a class in Java, this class is having its own in built functions which can directly be accessed by using class name **[String.methodName()]**

String can be declared as

1) String str = new String("Hello Java"); OR

2) String str= "Java Programming";

str is String variable or object

Methods of String

Method	Description	Return Type
charAt()	Returns the character at the specified index (position)	char
compareTo()	Compares two strings lexicographically	int
compareToIgnoreCase()	Compares two strings lexicographically, ignoring case differences	int
concat()	Appends a string to the end of another string	String
contains()	Checks whether a string contains a sequence of characters	boolean
endsWith()	Checks whether a string ends with the specified character(s)	boolean
equals()	Compares two strings. Returns true if the strings are equal, and false if not	boolean
equalsIgnoreCase()	Compares two strings, ignoring case considerations	boolean
format()	Returns a formatted string using the specified locale, format string, and arguments	String
getChars()	Copies characters from a string to an array of chars	void
indexOf()	Returns the position of the first found occurrence of specified characters in a string	int
isEmpty()	Checks whether a string is empty or not	boolean
lastIndexOf()	Returns the position of the last found occurrence of specified characters in a string	int

length()	Returns the length of a specified string	int
matches()	Searches a string for a match against a regular expression, and returns the matches	boolean
replace()	Searches a string for a specified value, and returns a new string where the specified values are replaced	String
replaceFirst()	Replaces the first occurrence of a substring that matches the given regular expression with the given replacement	String
replaceAll()	Replaces each substring of this string that matches the given regular expression with the given replacement	String
split()	Splits a string into an array of substrings	String[]
startsWith()	Checks whether a string starts with specified characters	boolean
substring()	Returns a new string which is the substring of a specified string	String
toCharArray()	Converts this string to a new character array	char[]
toLowerCase()	Converts a string to lower case letters	String
toString()	Returns the value of a String object	String
toUpperCase()	Converts a string to upper case letters	String
trim()	Removes whitespace from both ends of a string	String