Amrita Vishwa Vidyapeetham

TIFAC-CORE in Cyber Security

# 20CYS312 - Principles of Programming Languages

# Assignment-01: Exploring Programming Paradigms

Mittul R

21st January, 2024

## Paradigm 1: Imperative Programming Paradigm

- The word **"Imperative"** comes from the Latin word **"Impero"** meaning **"I Command"**
- Imperative programming is a Paradigm of Computer Programming in which the program describes a **sequence of steps that change the state of the computer.**
- It tells the Computer **"How"** to accomplish a task.
- Programs written this way often compile to **Binary Executables** that run more efficiently since all CPU instructions are themselves imperative statements.
- To make programs simpler to understand, Statements have been Grouped into Sections using Blocks.
- Procedural and Object-Oriented Programming belong under Imperative Paradigm includes
  - C
  - C++
  - C(Hash)
  - PHP
  - Java
  - Assembly

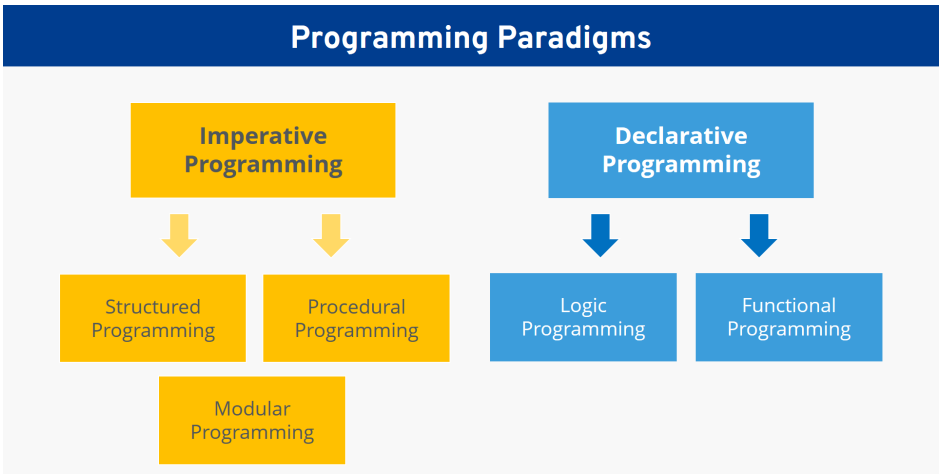



Figure 1: Programming Paradigm.

## Key Concepts of Imperative Programming :-

1. How Imperative Programming tells the Computer **How to Accomplish a Task ?**
   - Imperative programming involves giving explicit step-by-step instructions to the computer for performing a particular task.
   - Key elements of imperative programming include variables, assignments, control structures (like loops and conditionals), and procedures or functions.
   - In imperative programming, the focus is on providing precise instructions to the computer rather than on the overall logic or flow of the program.

2. What are the **Main Principles** that help developers articulate step-by-step instructions to the computer, allowing it to perform specific tasks ?
   - The Main Principles that help **Users and Developers** achieve specific results through step-by-step instructions are :
     - State and Variables
     - Assignment and Mutation
     - Sequential Execution
     - Control Structures
     - Procedures and Functions
     - Side Effects
     - Explicit Algorithms
     - Error Handling
     - Low-Level Control

3. **Side Effects :-**
    - The Side effects that are commonly associated with Imperative Programming, includes observable changes that occurs outside the function or code block.
    - These changes can include altering **global variables,** displaying the output on the console , etc..

4. **Explicit Algorithms :-**
    - In **Imperative Programming** , Developers / Users explicitly outline algorithms to solve problems by defining a sequence of statements.
    - In other words , we can say the users specify the **exact steps** for solving a problem.
    - Those Algorithms are expressed through a series of statements that transform the program's state.

5. **Error Handling :-**
    - Imperative Programming uses **exception handling mechanisms** that are commonly used for error handling in order to gracefully manage unexpected situations within programs.

6. **Low-Level Control :-**
    - Imperative languages provide low-level control over the hardware and memory, enabling developers to optimize code and enhance performance.

## Merits and De-Merits of Imperative Programming Paradigm:-

1. **Merits of Imperative Programming Paradigm :-**
    - Imperative programming is easy to read.
    - Imperative Programming are comparatively easy to learn.
    - Imperative Programming Conceptual model is very easy to understand.
2. **De-Merits of Imperative Programming Paradigm :-**
    - In Imperative Programming, the code gets confusing and quickly becomes very extensive
    - There are more chances to errors at the time of editing the code.
    - The maintenance blocks application development, which is the limitation of system-oriented programming.

Imperative Programming Pros & Cons

| Readability | Confusing Codes |
|---|---|
| You can easily adapt the structure of imperative programming. If you are a beginner learn and read imperative programming codes easily. | Your code will do more than one task and written in the imperative paradigm gets very extensive and wide. Thus it creates confusion. |
| **Comprehensibility** | **Erros in Writing** |
| No matter your experience level, you can easily follow the step-by-step solution process and implement the conceptual solution models. | There is a risk of editing errors when you change code. |
| **Application Features** | **Restriction on Extensions & Upgrades** |
| You can easily consider and use the features of an application to execute programs. | In imperative programming, it is very difficult to make upgrades and extensions. Therefore, you will have a tough time optimizing. |
| **Feasibility** | **Restriction in the Application Development** |
| You can easily compute through a source code by determining what a single-core processor can do with it. | Imperative programming is system-oriented programming. As a result, it can block your entire application development during maintenance. |
| **Widely Used** | **Prone to Data Race** |
| They teach imperative programming with authority in schools, colleges, and workplaces. Programmers and developers have a high familiarity with it. | Since imperative programs contain mutable variables, you have to keep in mind their vulnerability to data races. |

Figure 2: Source - Medium

## Applications of Imperative Programming Paradigm:-

1. System Programming and Middleware
2. Embedded Systems
3. Game Development
4. Device Drivers
5. Network Programming

## Language for Paradigm 1: Java


### What is Java ? History of Java Programming Language

1. **Java Programming Language**
   - Java is an object-oriented, platform-independent, and concurrent programming language that is widely used for various purposes.
   - Designed to be platform-independent, object-oriented, and concurrent, Java is a high-level programming language with broad applications.
   - Java stands out as a versatile and powerful programming language.
2. **History Of Java Programming Language**
   - **James Gosling, Mike Sheridan, and Patrick Naughton** initiated the Java language project in June 1991.
   - The language was initially called Oak after an oak tree. Later the project went by the name Green and was finally renamed Java, from Java coffee, a type of coffee from Indonesia.
   - Gosling designed **Java** with a **C/C++**-style syntax that system and application programmers would find familiar
   - **Sun Microsystems** released the first public implementation as **Java 1.0 in 1996.**
   - It promised write once, run anywhere (WORA) functionality, providing no-cost run-times on popular platforms.
   - The Latest Version of Java is **Java SE 21 LTS (19 - Sep - 2023)**

3. **Core Principles of Java Programming Langauge**

- The Five primary goals in the creation of the Java language are :
  (a) It must be simple, object-oriented, and familiar.
  (b) It must be robust and secure.
  (c) It must be architecture-neutral and portable.
  (d) It must execute with high performance.
  (e) It must be interpreted, threaded, and dynamic

4. **Merits and De-Merits of Java Programming Language**

- Merits of Java Programming Language
  (a) **Simple :-**
    – Java is a simple programming language , it is easy to learn and easy to understand. Its syntax is based on C++, and it uses automatic garbage collection.
  (b) **Object-Oriented :-**
    – Java uses an object-oriented paradigm. Java uses object-oriented concepts like **object, class, inheritance, encapsulation, polymorphism, and abstraction.**
  (c) **Secured**
    – Java is a secured programming language because it doesn't use Explicit pointers. Also, Java programs run inside the virtual machine sandbox.
  (d) **Robust**
    – Java is a robust programming language since it uses strong memory management. We can also handle exceptions through the Java code.
  (e) **Platform Independent**
    – Java code can run on multiple platforms directly
  (f) **Multi-Threaded**
    – Java uses a multi-threaded environment in which a bigger task can be converted into various threads and run separately.

- De - Merits of Java Programming Language
  (a) **Performance**
    – Java needs to be interpreted during runtime, which allows it to run on every operating system
  (b) **Memory consumption**
    – Java program consumes more memory since it runs on top of **Java Virtual Machine (JVM)**

(c) **Cost**

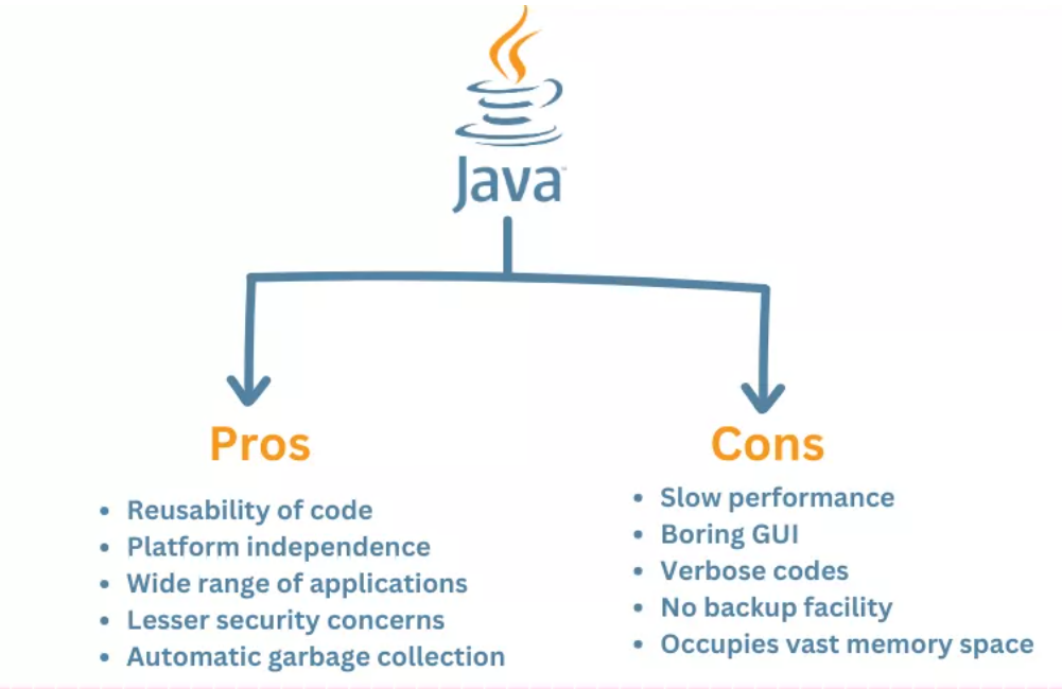
- Java programming language is a bit costly due to its higher processing and memory requirements

(d) **Less machine interactive**

- Java lacks when it comes to interacting directly with machines.

(e) **Garbage collection**

- Java provides automatic garbage collection that cannot be controlled by the programmer.




5. **Applications of Java Programming Language**

(a) Web Development
(b) Mobile Applications (Android)
(c) Cloud-based Services (Java-based APIs)
(d) Embedded Systems
(e) Scientific Applications

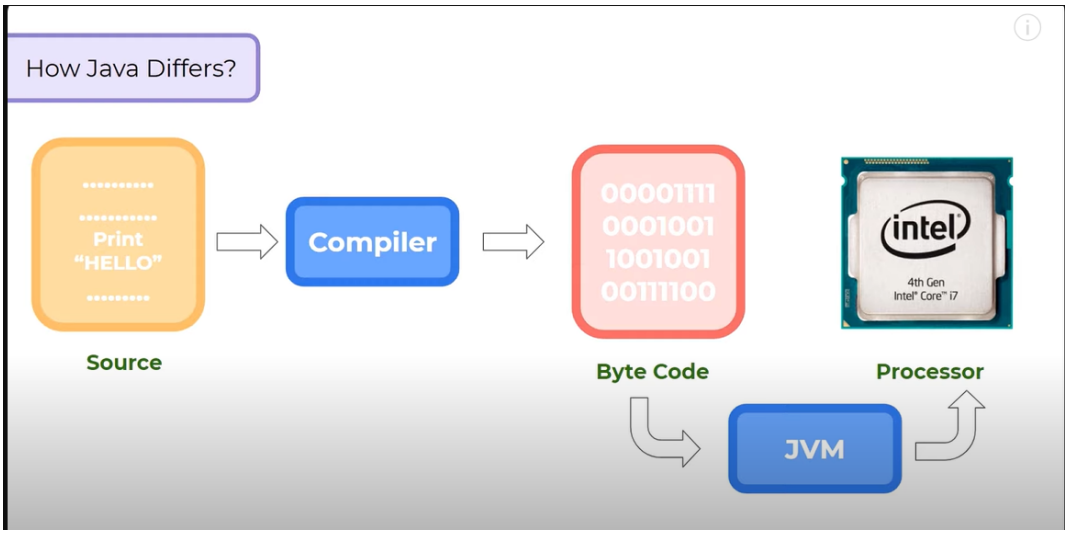
## How Does Java Programming Work ?

- Before getting into, **"How Java Program Works ?"** , we need to get familiarize in some concepts like :

  1. Byte Code
  2. JVM - **Java Virtual Machine**
  3. JIT - **Just In Time Compiler**
  4. JRE - **Java Run Time Environment**
  5. JDK - **Java Development Kit**

## Working Principle

- In **Java programming,** the source code is in a text file, and upon compilation, the code undergoes a process where it is translated by the compiler.
- Unlike some programming languages that directly convert source code into machine code **(binary code)**, ie in the form of **0's and 1's** understandable by the processor.
- Java's compiler produces an intermediary form known as **bytecode. Bytecode** is not directly comprehensible by the processor; instead, it serves as an intermediate representation of the source code.




- To facilitate the execution of bytecode on various hardware architectures, Java introduces the concept of the **Java Virtual Machine (JVM).**

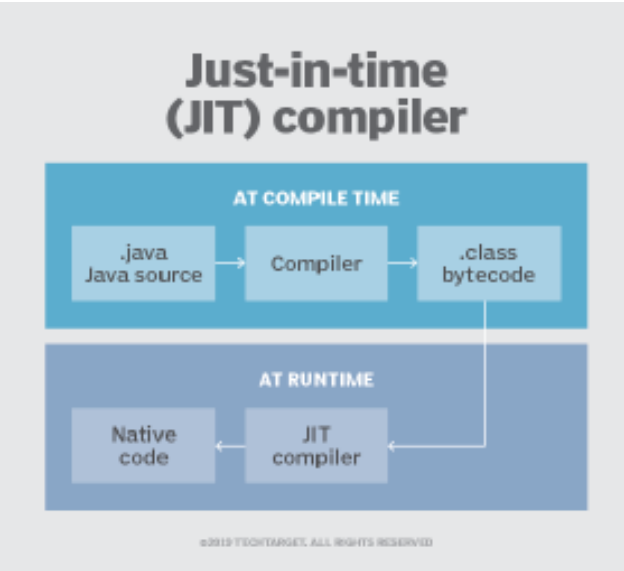
- The JVM acts as an abstraction layer between the **platform-independent bytecode** and the **underlying hardware**.
- When a Java program is executed, the bytecode is presented as **input to the JVM**, which dynamically translates it into a format compatible with the specific processor architecture.
- The compilation and execution of a Java program follow a two-step process.
    1. Initially, during the compilation phase, the source code is transformed into bytecode.
    2. Subsequently, during program execution, the JVM interprets and converts the bytecode into machine code, making it executable on the target processor.
- Java's adherence to this **bytecode-interpreted execution model** contributes to its reputation as a **Highly Portable Language.**
- The platform independence arises from the fact that the same bytecode can be executed on any device equipped with a compatible JVM.

## How Java is also called as Intrepreted Language ?

- Java is often characterized as an interpreted language due to its bytecode interpretation by the JVM.
- This implies that **each line of bytecode is read** and **executed sequentially** rather than processing the entire bytecode as a **single entity.**
- However, this interpretative approach may incur performance costs, leading to potentially slower execution.

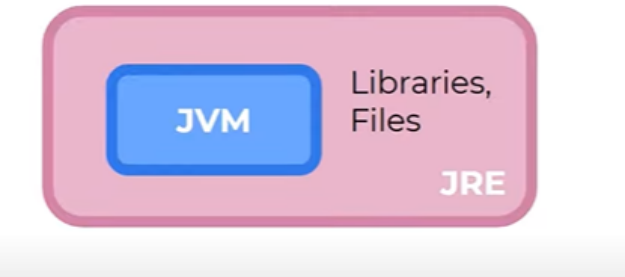
## JIT - Just In Time Compiler

- To address performance concerns associated with bytecode interpretation, Java incorporates a Just-In-Time (JIT) compiler.
- The JIT compiler identifies frequently executed portions of the bytecode and precompiles them into machine code. When these segments are reencountered during program execution, the JIT compiler efficiently delivers the precompiled machine code to the JVM, contributing to improved performance.



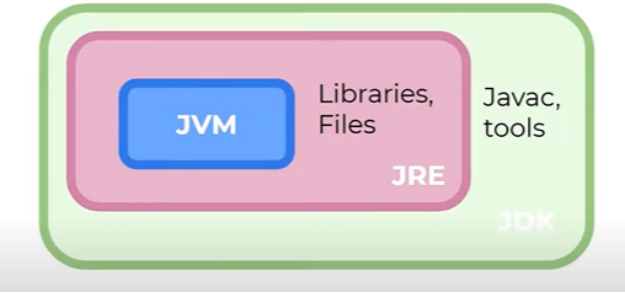
## JRE - Java Run-Time Environment

- JRE is the implementation of JVM.
- It provides a platform to execute java programs.
- JRE consists of **JVM, Java binaries, and other classes** to execute any program successfully.
- In short, a user needs **JRE** to run any **Java program.**



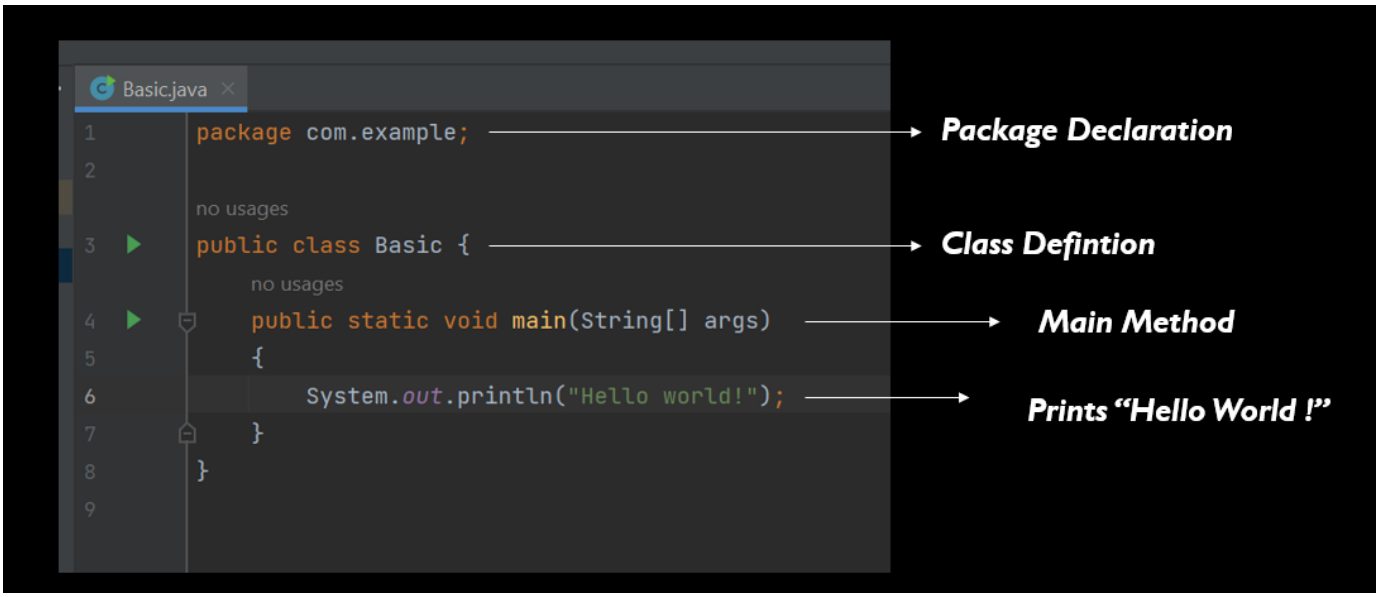

## JDK - Java Development Kit

- Java Development Kit aka JDK is the core component of Java Environment and provides **all the tools, executables, and binaries** required to **compile, debug, and execute a Java Program.**
- It is a software development environment which is used to develop |**Java applications** and **Applets.** It physically exists. It contains **JRE** + **development tools.**



# Example Programs

## 0.1 Basic "Hello World" Program




- Package Declaration - It is used to indicate the package to which the class belongs.
- Class Definition - Defines a class named **Basic**
- Main Method - Every Java Program Begins execution with the **Main Method.**
- Print Statement - **System.out.println("Hello, World!");** This line prints the text "Hello, World!" to the console.
- **System.out** is an object that represents the system's standard output (console).

## 0.2 Rules to be followed , while writing a Java Program :

- The Name of Class Definition , should be as same as File Name
- Main Class , should be always defined by the keyword **public**
- Main Method , should be always defined by the keyword **public static void**

## 0.3 Output of the Program

```
C:\Users\mitul\IdeaProjects\Popl\src\com\example>javac Basic.java

C:\Users\mitul\IdeaProjects\Popl\src\com\example>java com.example.Basic
```

- The Program is compiled using the keyword **javac <Filename.java>**
- Once if the program is compiled without any errors , then **java <Filename.class>** is used to print the output to the console

```
C:\jdk-19.0.2\bin\java.exe "-javaagent:C:\Program
Hello world!
```

## 0.4 Example 2 - Addition of Two Numbers

- To get two positive numbers as an input from the user and perform addition calculation

Main.java

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the Value for First Number: ");
        double num1 = scanner.nextDouble();
        System.out.print("Enter the Value for Second Number: ");
        double num2 = scanner.nextDouble();
        scanner.close();
        double sum = num1 + num2;
        System.out.println("Addition of " + num1 + " and " + num2 + " is: " + sum);
    }
}
```

```
Enter the Value for First Number: 23
Enter the Value for Second Number: 27
Addition of 23.0 and 27.0 is: 50.0
```

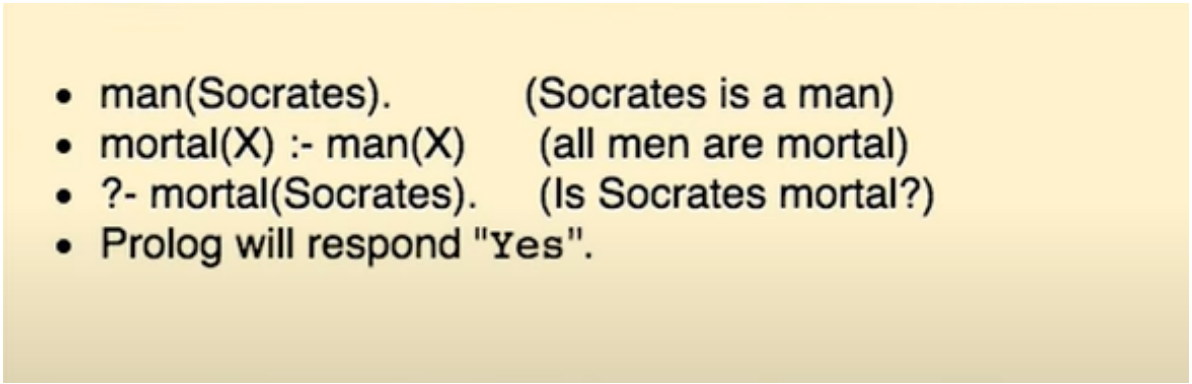
# Paradigm 1: Logic Programming Paradigm

## Introduction

- Logic programming is a **programming, database and knowledge** representation paradigm based on **formal logic**
- A logic program is a set of sentences in **Logical Form,** representing knowledge about some problem domain.
- Computation is performed by applying **Logical Reasoning** to that knowledge, to solve problems in the domain
- The Logic programming paradigm isn't made up of instructions - rather it's made up of **Facts and Clauses**
- In Logic oriented programming we will build an entire program based on the **notion of Logical Deduction.**
- The Logic programming using its knowledge base and tries to come up with a conclusion like **True or False**
- Logic programming languages that include **Negative Conditions** have the knowledge representation capabilities of a **Non-Monotonic Logic.**

## The Different Programming Languages that comes under Logical Programming Paradigm are :-

1. Prolog
2. Mercury
3. Datalog
4. Answer Set Programming (ASP)



## Timeline and Development of Logic Programming Paradigm

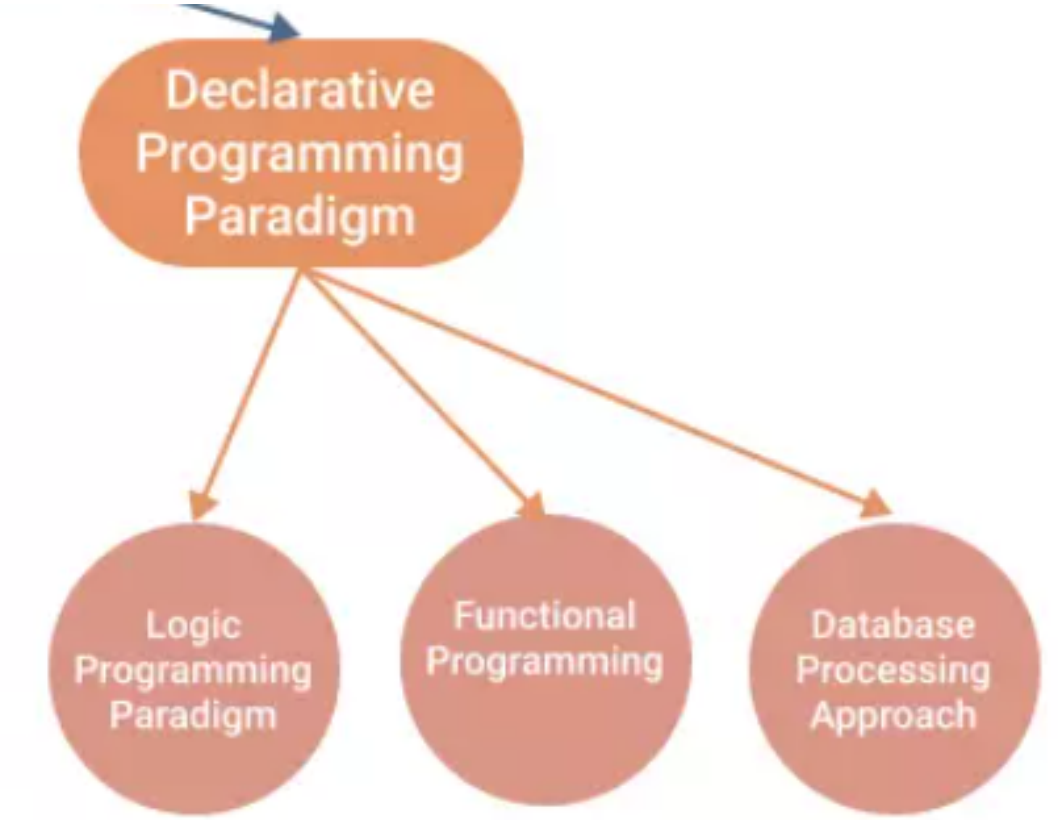
1. **1950s**: The concept of Logic Programming begins to take shape with the development of **symbolic logic** and early work on automated theorem proving.

2. **1970s:** The Logic Programming language **Prolog** is developed by **Alain Colmerauer** and **Phillipe Roussel,** providing a practical implementation of Logic Programming Concepts.

3. **1980s:** Prolog gains popularity and becomes widely used in academic and industrial settings for tasks such as **natural language processing, expert systems, and symbolic reasoning.**

4. **1990s:** Research and development in Logic Programming lead to the creation of new languages and extensions, such as **Answer Set Programming (ASP)** and **Constraint Logic Programming (CLP).**

5. **Present:** Logic programming remains an active area of research and development, with ongoing efforts to improve its efficiency, scalability, and integration with other paradigms.



## Key Concepts of Logic Programming Paradigm

1. **Logic-Based Representation :-**
   - In Logic Programming Paradigm, the programs are written as a collection of logical statements or rules. These rules define the relationships and conditions that the system must adhere to.
2. **Facts and Rules :-**
   - Logic programs are made up of facts and rules.
   - Facts are statements that are assumed to be true, while rules establish the connections between these facts.
   - By combining facts and rules, we can represent knowledge and specify how to derive new information.
3. **Predicates and Relationships :-**
   - Predicates play a crucial role in logic programming as they define the relationships between different entities.
   - Predicates are used to state facts or express rules about the connections between objects or concepts.
4. **Queries and Inference:**
   - In logic programming, the primary way of computing is through querying the knowledge base.
   - Users can ask questions to the system, and the logic programming engine uses logical inference to provide answers based on the defined facts and rules.
5. **Backtracking :**
   - Backtracking is a fundamental mechanism in logic programming.
   - When the system encounters a choice point or fails to find a solution, it can backtrack and explore alternative paths.
   - This allows the system to consider multiple possibilities and find all possible solutions.
6. **Unification :**
   - Unification is a critical process in logic programming that aims to make different logical terms equivalent. It enables variables to be assigned values, allowing for the matching and combination of different pieces of information.
7. **Declarative Nature :-**
   - Logic programming is considered declarative, meaning that programs express what needs to be achieved rather than how to achieve it.
   - The programmer specifies the relationships and constraints, and the logic programming engine takes care of the control flow.

8. **Horn Clauses :-**
    - Horn clauses are a special type of logical statements used in logic programming.
    - They consist of a **head and a body.**
      The head contains a **predicate,** while the body contains **conditions.**

9. **Recursion :-**
    - Logic programming languages often support recursion, allowing rules and predicates to refer to themselves.

10. **Non-Determinism :-**
    - One of the advantages of logic programming is its ability to handle non-deterministic computation.
    - This means that when there are multiple choices or solutions available, the system can explore different paths and provide multiple solutions.

## Merits and De-Merits of Logic Programming Paradigm

1. **Merits of Logic Programming Paradigm :-**
    - Declarative Nature
    - Natural Language Processing
    - Rule-Based Systems
    - Backtracking
    - Parallelism

2. **De-Merits of Logic Programming Paradigm :-**
    - Limited Efficiency for Some Tasks
    - Complexity in Debugging
    - Limited Support for Numeric Computation
    - Learning Curve
    - Not Universally Applicable

### Applications of Logic Programming Paradigm

1. Natural Language Processing (NLP)
2. Rule-Based and Expert Systems
3. Semantic Web and Knowledge Graphs
4. Database Querying
5. Decision Support Systems

## Language for Paradigm 2: Mercury

### What is Mercury ? and History of Mercury Programming Language

1. Mercury Programming Language
    - Mercury is a **Functional Logic Programming Language made for real-world uses.**
    - It is a purely **Declarative Logic Programming Language.**
    - It is related to both **Prolog and Haskell.**
    - It features a strong, static, polymorphic type system, and a strong mode and determinism system.
    - It has the same syntax and the same basic concepts such as the **Selective Linear Definite Clause Resolution (SLD) Algorithm.**
    - As such, it is often compared to its predecessor (Prolog) in features and run-time efficiency.
    - Unlike the original implementations of Prolog, it has a separate **compilation phase**, rather than being directly interpreted.
    - This allows a much **Wider Range of Errors** to be **detected** before running a program.
    - It features a strict static type and mode system and a module system


### Timeline of Mercury Programming Language :-

1. **1995** - The development of Mercury began in 1995 at the University of Melbourne, Australia, as a research project.

2. **1990** - Throughout the late 1990s, the language features of Mercury were defined. This included the design decisions that would enable the language to support both logic and functional programming constructs.

3. **2000** - The implementation of the Mercury language progressed. During this time, the Mercury team worked on improving the language's efficiency and expressiveness.

4. **2003** - The first public release of Mercury (version 0.10) occurred in 2003.

## Key Concepts of Mercury Programming Language

1. **Declarative and Logical Programming:**
    - Mercury allows developers to express Relationships and Rules Declaratively using logical statements.
2. **Purely Functional:**
    - Mercury supports purely functional programming, emphasizing immutability and the absence of side effect
3. **Strong, Static Typing:**
    - It has a strong, static type system that helps catch errors at compile-time.
4. **Constraint Logic Programming:**
    - Mercury allows us to specify constraints on variables, providing a powerful mechanism for expressing logical conditions.
5. **Pattern Matching:**
    - It supports pattern matching for concisely expressing relationships and making decisions based on data structures.
6. **Backtracking:**
    - Mercury supports backtracking, allowing the system to explore alternative solutions if the current one fails.
7. **Mode and Determinism Annotations:**
    - Mercury allows developers to annotate predicate modes and determinism, providing information to the compiler for optimization.
8. **High-Level Abstraction:**
    - Mercury provides high-level abstractions and features for expressing complex relationships and algorithms concisely.
9. **Modules and Namepaces:**
    - It supports modular programming, allowing developers to organize code into modules and namespaces.

## Merits and De-Merits of Mercury Programming Language

1. **Merits of Mercury Programming Language**
    - Declarative and Functional Features
    - Strong Typing and Safety
    - Parallelism and Concurrency Support
    - Memory Management
    - High-level Abstractions

2. **De-Merits of Mercury Programming Language**
    - Limited Adoption and Ecosystem
    - Performance Overhead
    - Less Tooling and Development Support
    - Debugging and Profiling
    - Niche Use Cases

## How does Mercury Program Works ?

- The File extension used for Mercury programming language source files is typically **".m"**
- The Latest Version of Mercury Programming Language is **Mercury 20.06** released in **June 2020.**

## How to Run Mercury Programm in Terminal ?

- Create a Mercury source file , for example **"hello.m"**
- Compile the Mercury program using the command ,

```
mmc –make hello
```

- Then Run the compiled program ,

```
./hello
```

## Example Programs

### Program to check whether the given number is Odd or Even

1. **Program to check whether the given number is Odd or Even**

```
:- module odd_even.

:- interface.
:- import_module io.

:- pred main(io::di, io::uo) is det.

:- implementation.
main(!IO) :-
    io.write_string("Enter a number: ", !IO),
    io.flush_output(!IO),
    io.read_line_as_string(Result, !IO),
    (
        Result = ok(String),
        parse_int(String, Num),
        (
            Num < 0,
            io.write_string("Please enter a positive number.\n", !IO)
        ;
            Num mod 2 = 0,
            io.write_string("The number is even.\n", !IO)
        ;
            io.write_string("The number is odd.\n", !IO)
        )
    ;
        Result = eof,
        io.write_string("Unexpected end of input.\n", !IO)
    ;
        Result = error(Error, Line, Col),
        io.format("Error: %s at line %d, column %d.\n", [s(Error), i(Line), i(Col)], !IO)
    ).
:- pred parse_int(string::in, int::out) is semidet.
parse_int(String, Int) :-
    string.to_int(string.strip(String), IntResult),
    ( IntResult = ok(Int) ; IntResult = error(_) ).
```

Figure 3: Program

## Analysis

1. Strength and Weakness of both paradigms ie **Imperative and Logic** and their associated programming languages **Java and Mercury** are discussed in the **Merits and De-Merits State**

### Notable Feature of Both Paradigms and their Associated Languages are :

### Imperative Programming Paradigm :

1. Uses statements to change the program's state.
2. Focuses on how to perform tasks step by step.
3. Relies on variables and assignment statements to store and manipulate data.
4. Includes control structures like loops and conditional statements for flow control.
5. Commonly used in languages like C, C++, and Java.
6. Emphasizes the sequence of steps to achieve a specific goal.

### Logic Programming Paradigm :

1. Based on formal logic and rules.
2. Focuses on defining relations and rules rather than step-by-step instructions.
3. Uses logical inference to derive conclusions from given facts and rules.
4. Prolog is a popular language that follows the logic programming paradigm.
5. Declarative in nature, where the programmer specifies what should be achieved rather than how to achieve it.
6. Well-suited for problems involving complex relationships and constraints.

## Challenges Faced :

## In Java(Imperative Programming Paradigm)

- Java can be overwhelming due to its extensive features such as complexity , Memory Management , Threading and Concurrency, but we can start with core concepts and use tutorials and books. Memory management challenges can be addressed with memory profiling tools and best practices. Developing concurrent programs in Java requires understanding thread safety and synchronization, which can be tackled by learning about Java's concurrency utilities and best practices.

## In Mercury(Logic Programming Paradigm

- Exploring Mercury poses challenges in understanding logic programming, especially for us as we are familiar with imperative programming. Limited resources compared to mainstream languages like Java make finding learning materials challenging. Efficient execution and performance considerations in logic programming add to the complexity. To address these, utilize online tutorials, documentation, and community forums for learning. Engage in hands-on coding to overcome the learning curve.

## Similarities and Differences :

| Aspect | Java | Mercury |
|---|---|---|
| Paradigm | Imperative, Object-oriented | Logic, Functional |
| Typing | Strongly typed | Strongly typed |
| Memory Management | Manual memory management | Automatic memory management |
| Concurrency | Built-in threading support | Support for parallel execution |
| Learning Curve | Moderate | Steeper learning curve |
| Community Support | Extensive | Limited |
| Platform Independence | Platform-independent (via JVM | Limited platform independence |
| Usage | General-purpose | Specialized for logic programming |

## Conclusion

- Java is a powerful, object-oriented language that is widely used for general-purpose programming. It has a strong typing system, requires manual memory management, and offers built-in threading support. With its extensive community support and platform independence, Java is a great choice for various programming tasks.
- On the other hand, Mercury is a unique language that focuses on logic programming. It combines logic and functional programming paradigms and offers automatic memory management. While it supports parallel execution, it does have a steeper learning curve compared to Java. Additionally, Mercury has limited platform independence and community support, but it excels in its specialized field of logic programming.

## References


1. https://www.techtarget.com/whatis/definition/imperative-programming:
2. https://www.ionos.com/digitalguide/websites/web-development/imperative-programming/
3. https://www.lsraheja.org/wp-content/uploads/2019/09/Imperative-Programming.pdf
4. https://www.geeksforgeeks.org/java/
5. https://www.simplilearn.com/tutorials/java-tutorial/what-is-java
6. https://www.linode.com/docs/guides/logic-programming-languages/
7. https://www.allassignmenthelp.com/blog/logic-programming-what-are-its-techniques/
8. https://mercurylang.org/information/doc-latest/reference_manual.pdf
9. https://mercurylang.org/documentation/papers/mfug_talk.pdf