20CYS312 - Principles of Programming Languages
Exploring Programming Paradigms

Assignment-01

Presented by Alagu Soundarya G
CB.EN.U4CYS21005
TIFAC-CORE in Cyber Security
Amrita Vishwa Vidyapeetham, Coimbatore Campus

Feb 2024

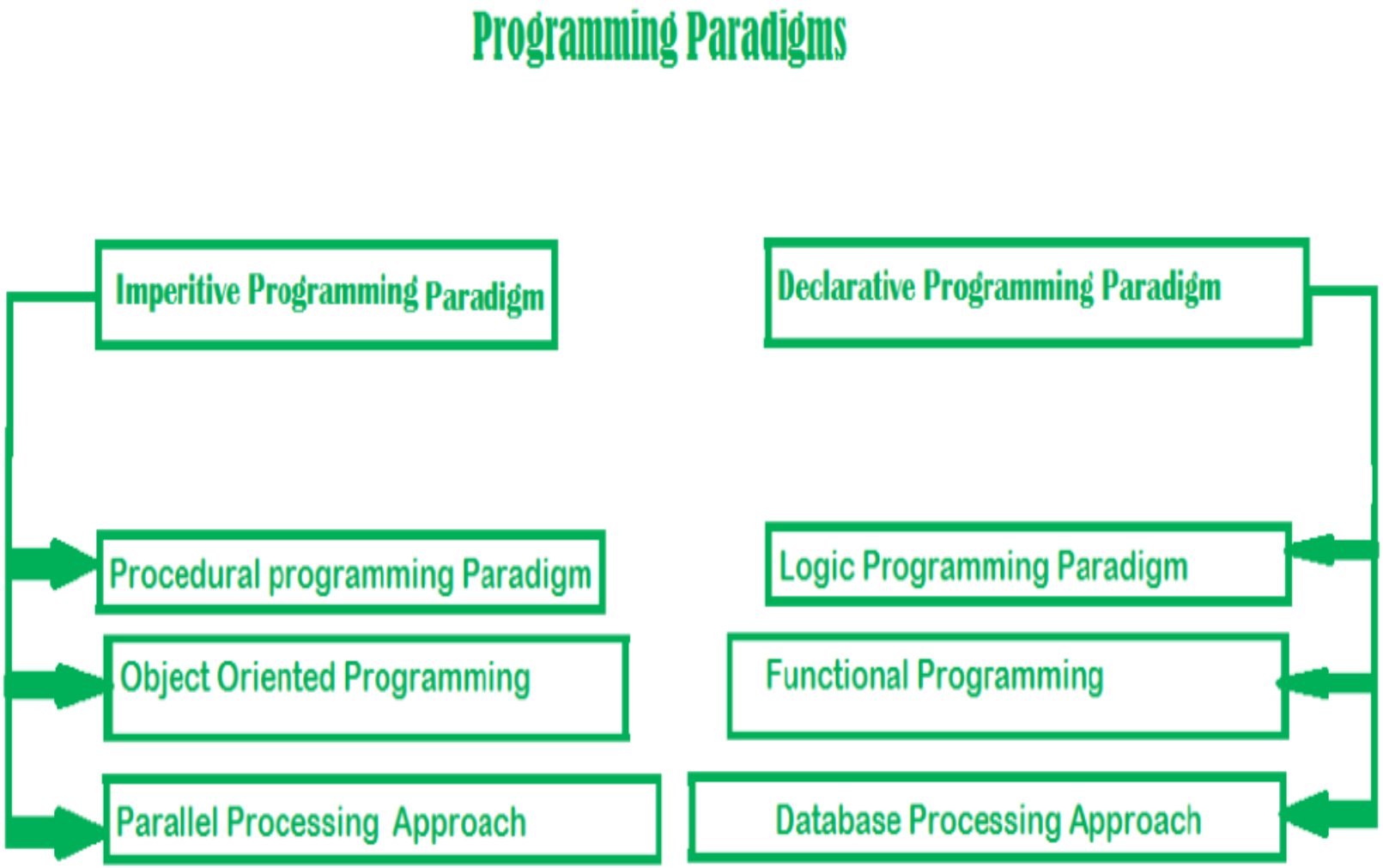
AMRITA
VISHWA VIDYAPEETHAM

Outline

What is programming paradigm

- A programming paradigm is indeed an approach or method to solve problems using a particular programming language or a combination of tools and techniques.
- It provides a framework and set of principles that guide how code is organized, structured, and executed.
- Different programming languages often support specific paradigms, and developers choose a language based on the paradigm that aligns with their problem-solving approach or the nature of the task at hand.

Types of Programming Paradigms



Types of Programming Paradigms

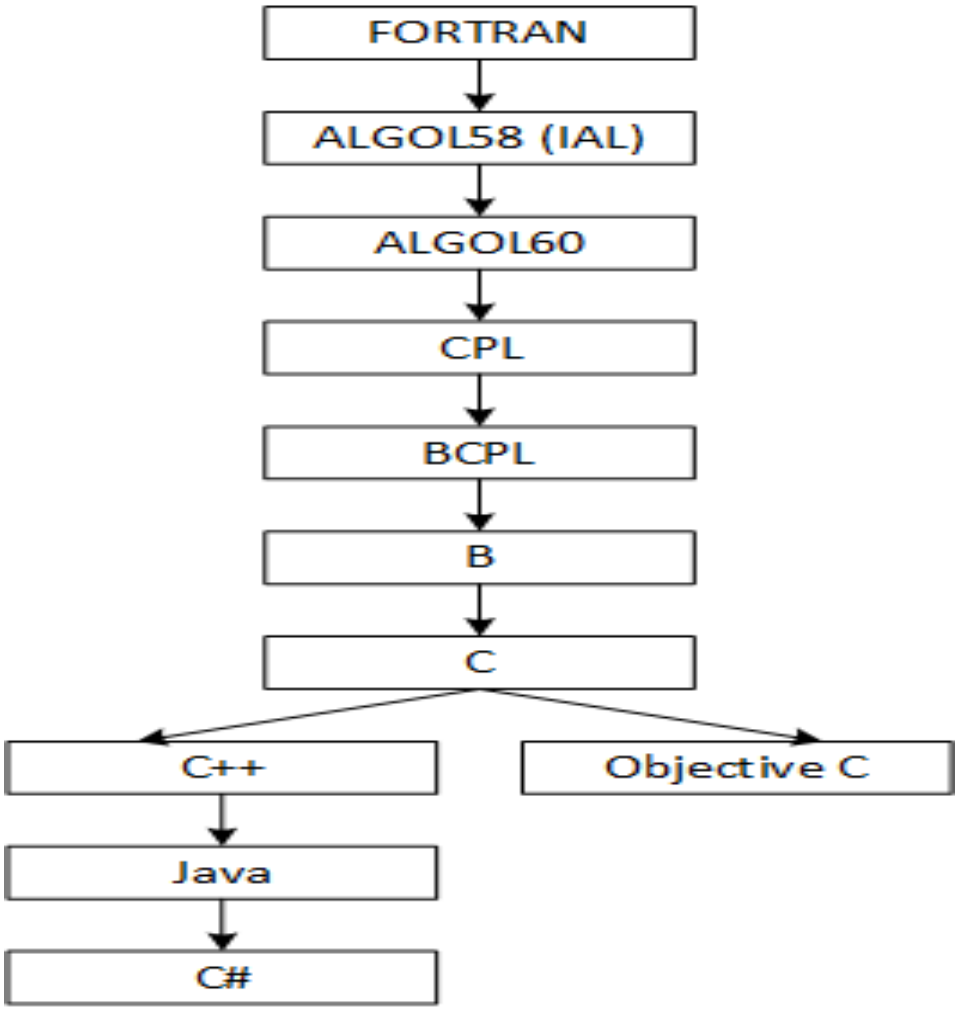
- Imperative Programming: Focuses on describing how a program operates in terms of statements that change a program's state.
- Declarative Programming: Concentrates on describing what a program should accomplish without specifying how to achieve it. Functional programming is a form of declarative programming.
- Object-Oriented Programming (OOP): Organizes code into objects, which encapsulate data and behavior. OOP promotes concepts like encapsulation, inheritance, and polymorphism.
- Functional Programming: Treats computation as the evaluation of mathematical functions.
- Procedural Programming: Organizes code into procedures, routines, or functions. It focuses on procedures that operate on data.
- Aspect-Oriented Programming (AOP): Aims to modularize cross-cutting concerns in software development.

What is Imperative Paradigm

- It is one of the oldest programming paradigm. It features close relation to machine architecture.
- It focuses on describing how a program operates in terms of statements that change a program's state.
- It is closely associated with the architecture of the von Neumann machine, which is the fundamental architecture for most computers.
- It is executed step by step, with each statement modifying the program's state.
- It often has a clear distinction between the input, the sequence of operations, and the final result. The state changes are made, and the final result is stored in memory or a designated variable after the execution of all statements.

Imperative Paradigm in C++

C++ is an imperative programming language with its roots tracing back to FORTRAN, the first high-level programming language.



Imperative Paradigm in C++

- C++ is an object-oriented programming language. It is an extension to C programming.
- C++ is a general purpose, case-sensitive, free-form programming language that supports object-oriented, procedural and generic programming.
- In C++, the imperative programming paradigm is employed, where the focus is on specifying a sequence of steps or commands that explicitly change the state of the program through actions such as assignment statements, loops, and conditionals.

Imperative Paradigm in C++ Code

```cpp
#include <iostream>
int calculateFactorial(int n) {
    int result = 1;

        result *= i;
    }
    return result;
}

int main() {
    int number;
    std::cout << "Enter a number: ";
    std::cin >> number;
    int factorial = calculateFactorial(number);
    std::cout << "Factorial of " << number << " is: " << factorial << std::endl;
    return 0;
}
```

Output

```
Enter a number: 5
Factorial of 5 is: 120
```

Imperative Paradigm in C++ Code Explanation

- Initialization: The imperative paradigm often involves explicit initialization of variables. In this case, result is initialized to 1.
- Looping: The for loop is a classic imperative construct. It iterates from 1 to n, and in each iteration, the value of result is multiplied by the loop variable i. This illustrates the imperative approach of specifying step-by-step instructions to achieve the desired result.
- State Modification: The result *= i; statement modifies the state of the result variable in each iteration, capturing the essence of imperative programming where state changes are explicit.
- User Input: The program takes user input using std::cin, which is common in imperative programming for interacting with the user.
- Function Call: The main function makes an imperative call to calculateFactorial to perform the factorial calculation.
- Output: The program outputs the result using std::cout, demonstrating the imperative nature of explicitly specifying output statements.

Need for Imperative Paradigm

1. Sequential Execution
2. Efficient Resource Management
3. Readability and Simplicity
4. State Modification
5. Control Flow
6. Procedural Abstraction

Pros and Cons of Imperative Paradigm

Pros

- Very simple to implement.
- Popular and familiar

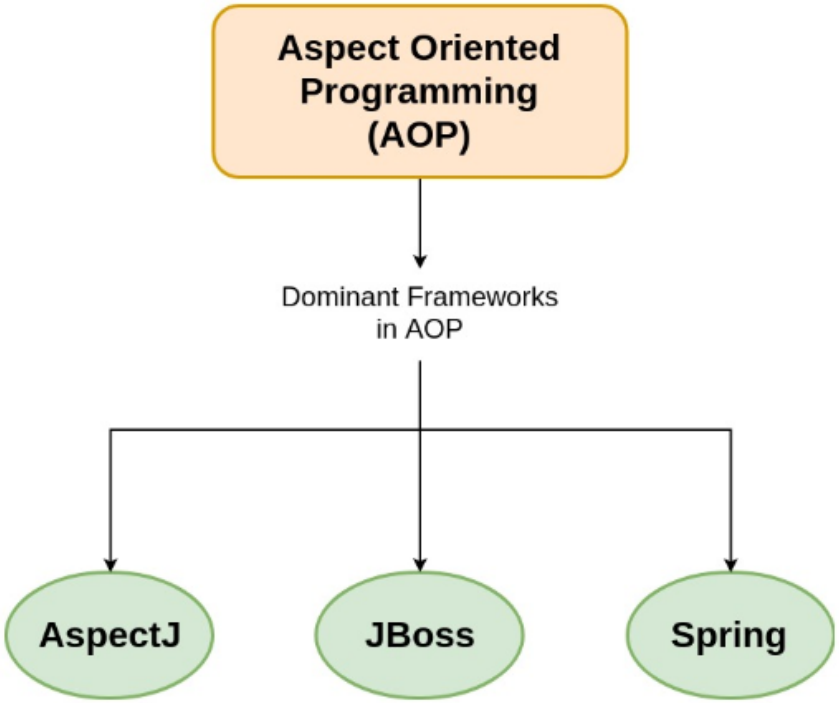
Cons

- Complex problem cannot be solved
- Less efficient and less productive

What is Aspect Paradigm

- Aspect-Oriented Programming (AOP) is a programming paradigm designed to enhance modularity by enabling the isolation of cross-cutting concerns.
- As the name suggests Aspect oriented programming(AOP) uses aspects in programming, where the aspect is the key unit of modularity.
- It does so by adding behavior to existing code (an advice) without modifying the code itself, instead separately specifying which code is modified via a "pointcut" specification.
- The three dominant frameworks in AOP are AspectJ, JBoss and Spring
- AspectJ is an extension for Java Programming. It implements both concerns and the weaving of crosscutting concerns using extensions of Java programming language.

What is Aspect Paradigm



Common terminologies in Aspect Paradigm

Aspect

- An aspect is a module or unit of code in AOP that encapsulates cross-cutting concerns. It contains advice, pointcuts, and inter-type declarations.
- Aspects allow the modularization of concerns that would otherwise be scattered across different parts of the codebase.

Weaving

- Weaving is the process of integrating aspects into the main program or application. It involves combining the aspect code with the base code.
- Weaving can occur at different stages: compile-time, load-time, or runtime.

Advice

- Advice is the code that gets executed at specified points in the program's execution. It represents the actual behavior associated with a cross-cutting concern.
- There are different types of advice:
  Before: Executed before a join point.
  After: Executed after a join point.
  Around: Completely surrounds a join point, allowing control over whether to proceed with the join point's execution.

Common terminologies in Aspect Paradigm

Join points

- Join points are specific points in the program's execution where advice can be applied. Examples of join points include method calls, object instantiations, or exception handling.
- Join points represent the points at which the aspect code can be woven into the base code.

Pointcuts

- A pointcut is a set of join points. It defines the conditions or criteria for selecting specific join points where the advice should be applied.
- Pointcuts allow developers to specify when and where the aspect code should be executed.

Inter-type Declarations

- Inter-type declarations allow aspects to declare new members (fields, methods, or nested types) in existing classes or types.
- This feature enables aspects to add functionality to existing classes without modifying their source code.

Aspect Paradigm in AspectJ Code

LoggingAspect.aj

```
import java.util.Date;

public aspect LoggingAspect {
    // Pointcut definition for methods in a specific package
    pointcut loggableMethods() :
        execution(* com.example.myapp.*.*(..));

    // Advice to be executed before the selected methods
    before() : loggableMethods() {
        System.out.println("Logging: Method called at " + new Date());
    }
}
```

Aspect Paradigm in AspectJ Code

MyApp.java

```
public class MyApp {
    public static void main(String[] args) {
        // Main application logic
        performTask();
    }

    static void performTask() {
        System.out.println("Executing the main task.");
    }
}
```

Output

sql Copy code

```
Logging: Method called at [current date and time]
Executing the main task.
```

Aspect Paradigm in AspectJ Code Explanation

Aspect

- LoggingAspect is an aspect, a module in AspectJ that encapsulates cross-cutting concerns, in this case, logging.
- The aspect keyword introduces the definition of the aspect.

Pointcut

- A pointcut (loggableMethods) defines a set of join points, which are specific points in the execution of the program.
- execution(* com.example.myapp.*.*(..)) is a pointcut expression that matches the execution of any method (*) in the package com.example.myapp with any name and any parameters.

Advice

- Advice is the actual code that gets executed at specified join points. In this case, the before advice is used.
- The advice specifies that it should run before the execution of methods matched by the loggableMethods pointcut.
- The code inside the advice prints a log message to the console indicating that a method is being called, along with the current date.

Aspect Paradigm in AspectJ Code Explanation

Sample Application Class (MyApp)

- MyApp is a simple Java class with a main method and a performTask method.
- The performTask method is called in the main method

Execution Flow

- When the performTask method is called in the MyApp class, the before advice in the LoggingAspect aspect is executed first (before the method call).
- The advice prints a log message to the console.
- After the advice, the actual logic in the performTask method is executed.

Output

- The output of running the MyApp class would include the log message printed by the before advice.

Need for Aspect Paradigm

- AOP provides a means to enhance the modularity of your application, particularly for functionality that extends across multiple boundaries.
- Enhanced flexibility and simplified management of our application.
- Elimination of redundant code patterns.
- Promotion of cleaner and more comprehensible code.
- Clear separation between core logic and cross-cutting concerns.

Pros and Cons of Aspect Paradigm

Pros

- Modularity and Separation of Concerns
- Code Reusability
- Improved Readability and Maintainability

Cons

- Increased Complexity
- Potential for Overuse
- Not supported by all programming languages

Imperative v/s Aspect

Imperative

- The focus is on "how" to achieve a goal through a sequence of statements.
- State changes are performed through assignments, loops, and conditionals.
- It closely aligns with the von Neumann architecture and the machine's instruction set.
- Common imperative languages include C, C++, Java, and Python.

Aspect

- The focus is on addressing cross-cutting concerns such as logging, security, and transaction management.
- AOP introduces constructs like aspects, pointcuts, advice, and weaving to separate concerns and enhance modularity.
- It allows for the encapsulation of behaviors that cut across different parts of the codebase.
- Common AOP frameworks include AspectJ.

Real Life Examples

Imperative Paradigm in C++

- Adobe Photoshop: Adobe Photoshop, a powerful image editing software, is written in C++. The imperative paradigm is heavily used to manage low-level details such as memory manipulation, pixel operations, and complex algorithms for image processing.
- Microsoft Windows Operating System: The Windows operating system kernel is primarily written in C++. It utilizes the imperative paradigm for tasks such as memory management, process scheduling, and hardware interactions.

Aspect Paradigm in AspectJ

- Spring Framework: The Spring Framework, a widely used Java framework for building enterprise applications, leverages AspectJ for aspect-oriented programming. AspectJ is employed to handle concerns such as transaction management, security, and logging
- Eclipse IDE: Eclipse, a popular integrated development environment (IDE), uses AspectJ for aspects related to workspace management, resource handling, and plug-in interactions. AspectJ is applied to enhance modularity and maintainabili a large and complex codebase.

References

- https://www.geeksforgeeks.org/introduction-of-programming-paradigms/
- https://icarus.cs.weber.edu/ dab/cs1410/textbook/1.Basics/programs.html
- https://www.javatpoint.com/cpp-tutorial
- https://en.wikipedia.org/wiki/Aspect-oriented_programming
- https://www.geeksforgeeks.org/aspect-oriented-programming-and-aop-in-spring-framework/
- https://eclipse.dev/aspectj/doc/released/progguide/starting-aspectj.html
- https://stackoverflow.com/questions/242177/what-is-aspect-oriented-programming
- ChatGpt