20CYS312 - Principles of Programming Languages
Exploring Programming Paradigms

Assignment-01: Exploring Programming Paradigms

Presented by Sudhir R.T
CB.EN.U4CYS21074
TIFAC-CORE in Cyber Security
Amrita Vishwa Vidyapeetham, Coimbatore Campus

Feb 2024

AMRITA
VISHWA VIDYAPEETHAM

Outline

Imperative Programming

- A paradigm is also a way of approaching a problem or carrying out a task.
- There are many well-known programming languages, but when they are used, they all need to adhere to a technique or strategy, and this approach is known as paradigms.
- The imperative programming paradigm is based on the idea that a program's state can be changed and manipulated by a sequence of statements.
- In imperative programming, functions are implicitly coded at every stage required to address a problem.
- Imperative programming instructs the computer "how" to perform a task
- An imperative program includes commands for the computer to execute.

Advantages:

- Very simple to implement
- It contains loops, variables etc.

Disadvantages:

- Complex problem cannot be solved
- Less efficient and less productive
- Parallel programming is not possible

Imperative - C++

- The predominant paradigm in C++ programming is imperative programming
- C++ allows the declaration of variables to store data, and these variables can be modified throughout the program, altering the program's state.
- C++ provides constructs for controlling the flow of execution, including if, else, for, while, and do-while statements.
- C++ supports procedural abstraction through functions. Functions encapsulate a sequence of statements and can be called with different arguments
- C++ allows the creation and manipulation of mutable data structures, such as arrays, vectors, and user-defined classes

```
int main() {
    // Mutable data structures in C++
    std::vector<int> numbers = {1, 2, 3, 4};
    numbers.push_back(5);  // Modify the vector
    return 0;
}
```

Figure: Code Snippet

Aspect-Oriented Programming

- Aspect oriented programming(AOP) as the name suggests uses aspects in programming. It can be characterized as the division of code into distinct modules.
- Features make it possible to implement overarching issues like transaction and logging that are not essential to business logic without complicating the code that is essential to its operation.
- It accomplishes this by advising the current code with new behavior.
- technique for building common, reusable routines that can be applied application wide.
- During development this facilitates separation of core application logic and common, repeatable tasks (input validation, logging, error handling, etc.)
- It works by adding behavior to existing code (an advice) without modifying the code itself, instead separately specifying which code is modified via a "pointcut" specification, such as "log all function calls when the function's name begins with 'set'".
- Security, for instance, is a cross-cutting concern since security rules can be applied to multiple methods inside an application across various different source codes.

Aspect-Oriented - AspectJ

- AspectJ is an aspect-oriented programming (AOP) extension created at PARC for the Java programming language.
- AspectJ has become a widely used de facto standard for AOP by emphasizing simplicity and usability for end users. It uses Java-like syntax, and included IDE integrations for displaying crosscutting structure.
- AspectJ enables the definition of pointcuts, which specify where the advice should be applied.
- AspectJ provides various types of advice to execute code at different points in the program's execution.
- AspectJ supports various join points like method executions, object instantiations, field access, etc.
- AspectJ supports both compile-time and runtime weaving.
  Compile-time weaving: This is typically done during the build process using the AspectJ compiler (ajc).
- AspectJ supports annotations for defining aspects and pointcuts, providing an alternative to the traditional syntax.

Comparison and Discussions

- Choosing between imperative programming and AOP often depends on the nature of the application and the development team's familiarity with the paradigms.
- In scenarios where low-level control and performance are crucial, imperative languages like C++ are appropriate.
- For applications with complex cross-cutting concerns and a need for modularity, AOP can be beneficial, especially when using languages like Java with mature AOP implementations like AspectJ.
- Imperative programming in C++ is characterized by its high performance, enabled by fine-grained control over memory and resources. Offering low-level features like pointers and manual memory management, it is well-suited for procedural programming tasks. However, this approach can lead to complexity and potential errors, especially with challenges related to concurrency. Despite its advantages, C++ code tends to be more verbose compared to higher-level languages.
- Aspect-Oriented Programming (AOP) with AspectJ enhances modularity by separating cross-cutting concerns from the core business logic. This paradigm provides a centralized approach to handling common concerns like logging and security, promoting reusability through aspects across different code sections. While AOP introduces a learning curve for developers unfamiliar with its concepts, AspectJ with its language extensions, dynamic weaving, and rich set of libraries, is a powerful tool for addressing and managing cross-cutting concerns in Java applications.

References

Respective Geeksforgeeks, wikipedia, javatpoint, science direct sites for each topic.