20CYS312 - Principles of Programming Languages
Exploring Programming Paradigms

Assignment-01

Presented by Swetha V
CB.EN.U4CYS21079
TIFAC-CORE in Cyber Security
Amrita Vishwa Vidyapeetham, Coimbatore Campus

Jan 2024

AMRITA
VISHWA VIDYAPEETHAM

Outline

Object Oriented Paradigm

- Object Oriented Paradigm works through the creation, utilization and manipulation of reusable objects to perform a specific task, process or objective.
- Objects as the building blocks: Treats software as a collection of objects that interact with each other to achieve a goal.

OOP-Pillars:

- Encapsulation: Bundles data (attributes) and the code that operates on that data (methods) together within objects, protecting data integrity and promoting modularity.
- Abstraction: Focuses on the essential features of objects, hiding implemen- tation details and making code easier to understand and maintain.
- Inheritance: Allows new objects (subclasses) to be created based on existing objects (superclasses), inheriting their properties and behaviors, promoting code reusability and extensibility.
- Polymorphism: Enables objects of different classes to be treated as objects of a common superclass, allowing for flexible and adaptable code.

Object-Oriented Paradigm

- OOP promotes modularity by organizing code into self-contained units (classes and objects), making it easier to understand, maintain, and extend.
- Objects communicate with each other by sending and receiving messages. Methods (functions) of one object can be invoked by another object to request a specific action.
- Dynamic binding allows the selection of a method or function to be determined at runtime, enabling flexibility and adaptability.
- Objects have both state (attributes or properties) and behavior (methods or functions).

Object Oriented Paradigm - C#

- C# syntax is similar to C and C++, making it familiar to developers from those backgrounds.
- C# is a strongly typed language with support for value types and reference types. Objects are instances of classes, and classes define the structure and behavior of object.
- C# is a fully object-oriented language where everything is treated as an object.
- Code snippets demonstrating OOP concepts in C#.
- C features automatic garbage collection, managing memory allocation and deallocation.

Object Oriented Paradigm - C#

- C can run on .NET Framework, providing a platform-independent and language-neutral environment for developing Windows application.
- C# supports encapsulation, allowing the bundling of data and methods that operate on the data within a class.
- C# supports inheritance, allowing a class to inherit properties and behaviors from another class.
  C# enables polymorphism, allowing objects to be treated as instances of their base class.
- C# allows abstraction by defining abstract classes and interfaces. Abstract classes provide a blueprint for derived classes, and interfaces define contracts for implementing classes.
  C promotes modularity, Dynamic Binding, Event-Driven Programming, Properties and Indexers.
- C# supports structured exception handling using try, catch, and finally blocks.

Aspect Oriented Paradigm

- AspectJ is a popular AOP framework for Java.
- Instead of scattering the code related to these concerns throughout the application, AOP allows developers to define aspects separately and then "weave" them into the main application at specific join points.
- Cross-cutting concerns are features or behaviors that affect multiple parts of an application, such as logging, security, and transaction management.
- An aspect is a module that encapsulates a cross-cutting concern. AOP aims to separate concerns, allowing developers to modularize cross-cutting concerns in dedicated aspects.
- A join point is a point in the execution of a program, such as method invocation or variable assignment.
  Advice is the code that is executed at a specified join point.
- A pointcut is a set of join points that share a common characteristic.
  Weaving is the process of integrating aspects into the main codebase at the specified join points.

Aspect Oriented Paradigm

- Eliminates the need to repeat cross-cutting code across different parts of the program.
- Enables cleaner separation of concerns, leading to more maintainable and testable code.
- Allows centralized implementation and consistent application of error handling logic across the program.
- Define aspects using high-level specifications, leaving the technical implementation details to the framework.
- This aspect can then be applied to specific join points, such as method executions, without modifying the original source code. In this way AOP provides modu-larity, maintainability and reusability of code.

Aspect Oriented Paradigm - Spring

- Spring is not a programming language like C#, but rather a framework built for Java.
- AOP in Spring is achieved through the use of aspects, pointcuts, and advice.
- Basic Spring configuration file with AOP support.
  <!– applicationContext.xml –> <beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/aop
  http://www.springframework.org/schema/aop/spring-aop.xsd">
  <!– AOP Configuration –> <aop:aspectj-autoproxy />
  <!– Define a bean –> <bean id="myService" class="com.example.MyService" />
  <!– Define an aspect for logging –> <bean id="loggingAspect"
  class="com.example.LoggingAspect" />
  </beans>

Aspect Oriented Paradigm - Spring

- Provides a dedicated module for AOP within the Spring framework. Allows developers to define aspects, pointcuts, and advice using XML configuration or annotations.
- Spring allows developers to configure AOP elements either through XML or annotations.
- AOP allows developers to apply different types of advice at specific points in the program execution.
  Spring supports various advice types, including @Before, @AfterReturning, @AfterThrowing, @After, and @Around.
- Developers can dynamically create and configure aspects, pointcuts, and ad- vice using the Spring AOP API.

Comparison and Discussions

- C# follows core OOP principles, including encapsulation, inheritance, and polymorphism.
  Spring AOP emphasizes modularization of cross-cutting concerns using aspects.
- Both C# and Spring support the concept of modularity and code organiza- tion.
- Strongly-typed language with support for classes, objects, and inheritance.
  ava-based, Spring supports AOP through AspectJ or Spring AOP.
- OOP in C# promotes modularity through classes and objects.
- Spring AOP enhances modularity by encapsulating cross-cutting concerns in aspects.
- Weaving in C is primarily performed at compile-time.
  Spring AOP supports both compile-time and runtime weaving.
- Traditional OOP in C may require manual configuration to address cross-cutting concerns.
  Spring AOP provides configuration options through annotations or XML, offering flexibility in defining aspects and pointcuts.

Comparison and Discussions

- C# is a standalone language, while Spring is a framework built on top of Java.
- Well-structured OOP code in C can be readable, but cross-cutting concerns may still lead to code tangling.
  AOP in Spring improves readability by separating core logic from cross-cutting concerns, enhancing maintainability
- C# has its own integrated development environment (IDE) called Visual Stu- dio, while Spring can be used with different IDEs, such as Eclipse or IntelliJ IDEA.
- Both C and Spring support the concept of modularity and code organization.
  They provide mechanisms for code reuse and promote good software engineering practices.
  Both languages offer features that enhance the development process and improve the overall quality of the code.
- C is a statically-typed language, meaning variables must have their types declared explicitly, while Spring is a dynamically-typed framework.
- Example: Discuss specific OOP design patterns or principles.

References

- "Object-Oriented Software Engineering: Practical Software Development Using UML and Java" by Timothy C. Lethbridge, Robert Laganier
- "Head First Object-Oriented Analysis and Design" by Brett D. McLaughlin, Gary Pollice, David West
- "Aspect-Oriented Software Development with Use Cases" by Ivar Jacobson, Pan-Wei Ng
- "Aspect-Oriented Programming with the E Verification Language: A Pragmatic Guide for Testbench Developers"