Amrita Vishwa Vidyapeetham

TIFAC-CORE in Cyber Security

20CYS312 - Principles of Programming Languages
Assignment-01: Exploring Programming Paradigms

Vinoth Kumar D

21st January, 2024

Paradigm 1: Object-Oriented Programming

- **Features:**

Classes and Objects:

- A class is a template that consists of the data members or variables and functions and defines the properties and methods for a group of objects.

- The compiler does not allocate memory whenever you define a class.

Example:

- You can define a class called Vehicle. Its data fields can be vehiclename, modelnumber, color, dateofmanufacture, etc.

- An object is nothing but an instance of a class. Each object has its values for the different properties present in its class. The compiler allocates memory for each object.

Abstraction:

- The literal meaning of abstraction is to remove some characteristics from something to reduce it to a smaller set. Similarly, Object Oriented Programming abstraction exposes only the essential information of an object to the user and hides the other details.

In real life, like when you toggle a switch, it simply turns on or off the lights. Here, we only know the functionality of the switch, but we don't know its internal implementation, like how it works.

Advantages of Abstraction

- The advantages of abstraction are as follows:
  - It enables code reuse by avoiding code duplication.
  - It enhances software security by making only necessary information available to the users and hiding the complex ones.

Inheritance:

- Inheritance is one of the most important features of object oriented programming. It allows a class to inherit the properties and methods of another class called the parent class, the base class, or the super-class.

The class that inherits is called the child class or sub-class. - You can have a parent class called "Shape" and other classes like Square, Circle, Rectangle, etc. Since all these are also shapes, they will have all the properties of a shape so that they can inherit the class Shape.

Polymorphism:

- The word polymorphism means to have many forms. So, by using polymorphism, you can add different meanings to a single component.

- There are two types of polymorphism:
  - Run-time polymorphism
  - Compile-time polymorphism

Method Overloading

- Methods overloading is a type of compile-time polymorphism using which you can define various functions with the same name but different numbers of arguments. The function call is resolved at compile time, so it's a type of compile-time polymorphism. Here resolution of the function call implies binding to the correct function definition depending on the arguments passed in the function call.

Example:

- You can create a function "add". Now, when you pass two integers to this function, it will return their sum, while on passing two strings, it will return their concatenation.
- So, the same function acts differently depending on the input data type.

Method Overriding

- Method Overriding is a type of run-time polymorphism. It allows overriding a parent class's method by a child class. Overriding means that a child class provides a new implementation of the same method it inherits from the parent class.

Example

- You can have a parent class called "Shape" with a method named "findArea" that calculates and returns the area of the shape. Several sub-classes inherit from the "Shape," like Square, Circle, Rectangle, etc. Each of them will define the function "findArea" in its way, thus overriding the function.

Encapsulation

- Encapsulation means enclosing the data/variables and the methods for manipulating the data into a single entity called a class. It helps to hide the internal implementation of the functions and state of the variables, promoting abstraction.

Example

- You can have some private variables in a class that you can't access outside the class for security reasons. Now, to read or change the value of this variable, you can define public functions in the class which will perform the read or writes operations.

• **Advantages of OOPS:**

Therefore the benefits of Object-Oriented Programming are:

- It makes troubleshooting easier and faster by making the code modular. So, you can look at the particular class or method whenever an error occurs instead of checking the entire code.
- It allows code reuse by inheritance.
- It enables flexibility through polymorphism as one function or object can adapt to several forms according to the requirement.
- It allows you to solve a problem efficiently by breaking a huge problem into smaller manageable parts like classes and objects.
- The upgrade of the OOP system of languages can be easily done from smaller systems to larger systems.
- Without interference, there might be multiple instances of the same object.

Disadvantages of OOPS:

- It has a steep learning curve because breaking down a problem into simple components requires long-term thinking, which comes with experience.
- Generally, the code becomes larger in object-oriented programming compared to procedural programming.

Difference Between Procedural and Object Oriented Programming:

Parameter	Procedural Programming	Object Oriented Programming
Security	Procedural Programming does not provide a mechanism to hide data, making it less secure compared to Object Oriented Programming.	Data hiding is possible in Object Oriented Programming due to abstraction, making it more secure than Procedural Programming.
Method	The main program is divided into smaller parts based on functions, treating them as separate programs.	OOP uses the concept of classes and objects, dividing the program into small chunks known as objects, which are instances of classes.
Approach	Top-Down approach.	Bottom-Up approach.
Importance	This programming paradigm prioritizes functions and the sequence of actions.	This programming paradigm emphasizes data over functions or procedures as it operates based on the real-world.
Orientation	It is Structure/Procedure oriented.	It is Object Oriented.
Inheritance	Inheritance is not provided in Procedural Programming.	Inheritance is achieved in three modes in Object Oriented Programming - protected, private, and public.
Reusability of Code	There is no feature to reuse code in Procedural Programming.	Object Oriented Programming allows the reuse of existing code through a feature called inheritance.
Important Attribute	It prioritizes function over data.	It prioritizes data over function.
Modes of Access	Procedural Programming does not have specific access modes for accessing functions or attributes in a program.	Object Oriented Programming provides three access modes - protected, private, and public for accessing functions or attributes.
Data Sharing	It shares global data among the functions in the program.	It shares data among objects through its member functions.
Examples	Examples of Procedural Programming languages include Assembly, C, Go, Shell, and MATLAB.	Examples of Object Oriented Programming languages include Ruby, Smalltalk, Eiffel, Kotlin, and Rust.

Table 1: Comparison of Procedural Programming and Object Oriented Programming

Example:

Python

```
class Car:
    def __init__(self, make, model, year, color):
        self.make, self.model, self.year, self.color, self.speed = make, model, year, color, 0

    def accelerate(self):
        self.speed += 5
        print(f"{self.make} {self.model} is accelerating. Current speed: {self.speed} mph")

    def brake(self):
        if self.speed > 0:
            self.speed -= 5
            print(f"{self.make} {self.model} is braking. Current speed: {self.speed} mph")
        else:
            print(f"{self.make} {self.model} is already stopped.")
```

```
car1, car2 = Car("Toyota", "Camry", 2022, "Blue"), Car("Honda", "Accord", 2021, "Red")

car1.accelerate()
car2.accelerate()

car1.brake()
car2.brake()
```

Ruby

```
class Car
  attr_reader :make, :model, :year, :color, :speed

  def initialize(make, model, year, color)
    @make, @model, @year, @color, @speed = make, model, year, color, 0
  end

  def accelerate
    @speed += 5
    puts "#{@make} #{@model} is accelerating. Current speed: #{@speed} mph"
  end

  def brake
    if @speed > 0
      @speed -= 5
      puts "#{@make} #{@model} is braking. Current speed: #{@speed} mph"
    else
      puts "#{@make} #{@model} is already stopped."
    end
  end
end

car1 = Car.new("Toyota", "Camry", 2022, "Blue")
car2 = Car.new("Honda", "Accord", 2021, "Red")

puts "#{car1.make} #{car1.model} (#{car1.year}) in #{car1.color} color"
puts "#{car2.make} #{car2.model} (#{car2.year}) in #{car2.color} color"

car1.accelerate
car2.accelerate

car1.brake
car2.brake
```

Implementation of objet oriented programming

- Application Development:
  - OOP structures code into classes and objects for modularity and maintainability.
  - Encapsulation hides internal details, promoting a clear interface.
  - Inheritance and polymorphism enable code reuse and flexibility.
- Game Development:
  - Entities like characters and items are modeled as objects.
  - Inheritance organizes game objects into hierarchies.
  - Encapsulation hides internal logic; polymorphism aids flexibility.
- Web Development:

- OOP organizes web applications using classes for models, controllers, and views.
- Encapsulation ensures modular and reusable components.
- Inheritance supports code hierarchy and reuse.

- Operating Systems:
  - OOP principles structure operating systems with classes for processes, file systems, etc.
  - Inheritance creates hierarchies; encapsulation isolates internal details.
  - like Microsoft Windows uses OOPs Principle.
- Databases (ObjectDB, db4o):
  - Object-oriented databases map programming language classes to database objects.
  - Encapsulation ensures limited access to internal object details.
  - Inheritance models relationships; polymorphism aids flexibility.

Language for Paradigm 1: Ruby

0.0.1 Brief history

- Ruby is a dynamic, high-level, open source programming language with a focus on simplicity and productivity.

- It has an elegant syntax that is natural to read and easy to write. It is an interpreted programming language that was designed for the productivity of programmers in order to make programming fun.

- Ruby is a language of careful balance. Its creator, Yukihiro "Matz" Matsumoto, blended parts of his favourite languages Perl, Smalltalk, Eiffel, Ada, and Lisp to form a new language that balanced functional programming with imperative programming.

- It has gained popularity for its use in web application development using the Ruby on Rails framework, built with the Ruby language by [David Heinemeier Hansson].

- The first public release of Ruby 0.95 was announced on Japanese domestic newsgroups on December 21, 1995.

- As at the time of this writing the most current version of Ruby is version 2.6.3 which was released on April 17, 2019.

- Here's an *Hello World* code spinet in Ruby, Python and Java.
  - Ruby — ‘puts "Hello World"‘
  - Python — ‘print("Hello World")‘
  - Java —

```
public class Main {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

  - Doesn't talk directly to hardware
- Instead:
  - 1. written in a textfile
  - 2. parsed on the interpreter
  - 3. turned into code

0.0.2 Features and advantages of Ruby

Features

- Interpreted. That is, Ruby interpreter needs to parse the code and translate it into machine language understandable by a computer, but there is no previous compilation process like in C or Java.

- Dynamic and flexible. It can be used to alter code at runtime. This allows for greater flexibility in code writing and the creation of more expressive programs.

- Readable and expressive. Ruby takes pride in having a clean and readable syntax that resembles

natural language. This facilitates code understanding and maintenance, fostering rapid development.

- Open source and cross-platform. It can be freely downloaded from the official website and run on different operating systems.

- Object-oriented. Everything in Ruby is an object, even basic data types like strings, numbers, or even boolean values. This allows for the creation of classes, inheritance, polymorphism, and encapsulation, making it easier to build modular and reusable programs.

- Backed by a large community. Ruby has an active community of developers and a wide range of libraries and frameworks available. For instance, Ruby on Rails is a highly popular web framework built on Ruby, used for developing robust and scalable web applications.

Advantages

- Fast Development. Ruby on rails offers faster application development. Ready-to-use libraries and plugins (called gems) are available, allowing developers to build app functionalities quickly.

- Adherence to Coding Standards. Ruby on rails follows standard code and application design principles that enable developers to build high-performance software applications in less time.Ruby on rails also offers code optimization, where modules handle complex data requirements and perform complex computations.

- Scalability. Ruby on rails gives better performance with an increase in user requests, as server-side code can process user requests directly without any other web service as in client-server applications.

- Maintenance and Testing. Ruby on rails follows a model-view-controller (MVC) architecture. The separation of business logic as models, views as HTML, XML, etc., and the communication between both via controller improve code testing and maintenance.Ruby on rails supports test-driven development (TDD) and behavior-driven development (BDD), making the development process more cost-effective and efficient.

- Security. The current version is stable, with multiple built-in security mechanisms allowing developers to build web applications secure from cyber attacks and vulnerabilities.Ruby on Rails framework offers protection against cross-site scripting, SQL injections, insure direct object reference, social engineering attacks, etc.

- Strong Community. Being an open-source framework, ruby on rails enjoys a vibrant community of RoR developers. These developers contribute to the existing Ruby on Rails code base. It has the largest community of contributors on GitHub and is one of the biggest code repositories.

0.0.3 How Ruby differs from Python

Python	**LANGUAGE**	**Ruby**
Was created in 1991 by Guido Van Rossum		Was created in 1995 by Yukihiro Matsumoto
A diverse community with extensive ties to Linux and academia	**PROS**	Tons of features out of the box for web development
Often very explicit and inelegant to read	**CONS**	Can be tough to debug at times. Very web-focused
Django	**WEB FRAMEWORKS**	Ruby on Rails
Very stable and diverse but innovate slower	**COMMUNITY**	Innovate quicker but causes more things to break
Google, Youtube, Dropbox, The Washington Post	**USAGE**	Twitter, Airbnb, Github, Shopify

Table 2: Comparison of Python and Ruby

0.0.4 Limitations of Ruby

- Runtime Speed and Performance. One of the most frequent arguments against RoR is its 'slow' runtime speed, which makes it harder to scale your RoR applications.Performance considerations should be still kept in mind, though. Twitter, for example, struggled to improve RoR's performance that deteriorated after the social network became very popular. Although Twitter did not abandon RoR completely, it had to replace certain internal communication components and server daemons with Scala solutions.

- Lack of Flexibility. RoR is an opinionated framework with a lot of hard dependencies and modules included out of the box. To kickstart the project, your developers should configure routing, database migrations, and other modules shipped with the framework.At some point, you will have to make a difficult choice between giving a deep overhaul to your Rails application or using another framework that better suits your needs.

- High cost of wrong decisions in development. Since prototyping with Rails is incredibly fast, an engineering team inexperienced in Rails might make unobvious mistakes that will erode your application's performance in the future.These structural deficiencies will be hard to fix because Rails is an open framework, where all components are tightly coupled and depend on each other.

0.0.5 Code

Class

```
class Color
  def initialize(name, hexcode)
    # Do cool things with name and hexcode
    puts "The color #{name} has hexcode #{hexcode}"
  end
end

color1 = Color.new("purple", "#8824a4")
color2 = Color.new("blue", "#4c6fcc")
```

Inheritance

```
class Person
  def initialize(name, age)
    @name = name
    @age = age
  end

  def to_s
    "#{@name} is #{@age} years old."
  end
end

class SuperHero < Person
  def initialize(name, age, superpower)
    super(name, age)
    @superpower = superpower
  end
end
```

Access Specifier

```
class Color
  # ... (other methods)
```

```
  protected

  def update_hexcode(hexcode)
    @hexcode = hexcode
  end
end

color = Color.new("purple", "#8824a4")
color.update_hexcode("#9427b2")
```

Encapsulation

```
class Demoencapsulation
  def initialize(id, name, addr)
    @cust_id = id
    @cust_name = name
    @cust_addr = addr
  end

  def display_details()
    puts "Customer id: #@cust_id"
    puts "Customer name: #@cust_name"
    puts "Customer address: #@cust_addr"
  end
end

cust1 = Demoencapsulation.new("1", "Mike", "Wisdom Apartments, Ludhiya")
cust2 = Demoencapsulation.new("2", "Jackey", "New Empire road, Khandala")

cust1.display_details()
cust2.display_details()
```

Paradigm 2: Aspect-Oriented

→ First aspect means a particular part or feature of something.

→ It can be defined as the breaking of code into different modules

→ Aspects enable the implementation of crosscutting concerns such as- transaction, logging not central to business logic without cluttering the code core to its functionality

0.0.6 Features

Cross-Cutting Concerns:
- AOP focuses on addressing cross-cutting concerns, which are functionalities that affect multiple modules or components in an application. Examples include logging, security, and error handling.

Aspect:
- In AOP, an aspect is a module that encapsulates cross-cutting concerns. Aspects define certain behaviors that can be applied to multiple parts of the application without modifying their source code.

Join Point:
- A join point is a specific point in the execution of a program where an aspect can be applied. Join points are identified by expressions that match specific points in the execution flow, such as method calls, object instantiations, or variable assignments.

Advice:

- Advice is the actual code that runs at a particular join point. There are different types of advice, such as "before," "after," "around," etc., specifying when the advice should be executed in relation to the join point.

Pointcut:

- A pointcut is a set of join points. It defines the criteria for where an aspect's advice should be applied. Pointcuts use expressions to match join points based on method signatures, class names, or other criteria.

Weaving:

- Weaving is the process of integrating aspects into the main business logic of the application. This can be done at compile-time, load-time, or runtime. Weaving allows the aspects to be applied to the appropriate join points.

Modularity and Reusability:

- AOP promotes modularity by separating cross-cutting concerns from the main application logic. Aspects can be reused across different parts of the application, enhancing maintainability and reducing code duplication.

Encapsulation:

- Cross-cutting concerns are encapsulated within aspects, providing a cleaner separation of concerns. This improves code organization and makes it easier to understand and maintain.

Dynamic Aspect:

- AOP allows for dynamic aspects, where aspects can be added or removed during runtime. This flexibility provides the ability to adapt the behavior of the system without modifying its source code.

AspectJ:

- AspectJ is a widely-used extension of the Java programming language that supports AOP. It introduces additional syntax and features to enable a more expressive and powerful AOP implementation.

0.0.7 Structure

- Core Modules
- Aspects
- Pointcuts
- Advice
- Weaver

0.0.8 Working of AOP

0.0.9 Identifying concerns

1. First, identify the different concerns or responsibilities of your program. For example, if you're writing a program to process orders, you might identify concerns such as order validation, payment processing, and order fulfillment.

0.0.10 Defining aspects

[resume]Once you've identified your concerns, you can define aspects for each one. An aspect is a modular unit of code that encapsulates a specific behavior or responsibility. For example, you might create an aspect for order validation, another for payment processing, and so on.

0.0.11 Determining join points

[resume]A join point is a specific point in your program's execution where an aspect can be applied. For example, a join point for the order validation aspect might be when a new order is submitted. You can define join points in your code using annotations or other markers.

0.0.12 Defining pointcuts

[resume]A pointcut is a set of join points where an aspect should be applied. For example, you might define a pointcut for the order validation aspect that includes all the join points where a new order is submitted. Pointcuts help narrow down the scope of an aspect so that it's only applied where needed.

0.0.13 Defining advice

[resume]Advice is the behavior an aspect provides at a join point. Several types of advice exist, including before, after, and around advice. Before advice is executed before the join point, after advice is executed after the join point, and around advice wraps the join point and can modify its behavior.

0.0.14 Weaving aspects

[resume]Weaving is the process of applying aspects to your program's execution. During weaving, the advice provided by aspects is injected into the appropriate join points in your code. This allows the behavior of aspects to be integrated into your program's execution.

0.0.15 Executing program

[resume]Once your aspects are woven into your program, you can execute the program as usual.

0.0.16 Advantages of AOP

1. **Modularity:** AOP helps to improve the modularity of software by allowing developers to separate cross-cutting concerns into modular units known as aspects. This can make it easier to understand, modify, and reuse code.
- **Maintainability:** By separating concerns, AOP can make it easier to maintain and modify code over time. This can be particularly useful when adding new features or behavior to an application without modifying the existing code.
- **Reusability:** Aspects can be reused across different projects, which can help to improve the efficiency and productivity of developers.
- **Performance:** AOP can be used to optimize the performance of an application by identifying and addressing bottlenecks or other inefficiencies.

0.0.17 Disadvantages of AOP

- **Complexity:** AOP can add complexity to an application, as it requires developers to learn and use an AOP framework and to understand the concepts of aspects, advice, pointcuts, and join points.
- **Overhead:** AOP can introduce additional overhead, as it requires the use of an AOP framework and the execution of additional code at runtime.
- **Compatibility:** AOP may not be compatible with all programming languages and environments, which can limit its use in some situations.
- **Debugging:** Debugging an application that uses AOP can be more challenging, as it may be difficult to understand the execution flow and the interactions between different aspects.

0.0.18 Features

0.0.19 Cross-Cutting Concerns

- AOP focuses on addressing cross-cutting concerns, which are functionalities that affect multiple modules or components in an application. Examples include logging, security, and error handling.

0.0.20 Aspect

- In AOP, an aspect is a module that encapsulates cross-cutting concerns. Aspects define certain behaviors that can be applied to multiple parts of the application without modifying their source code.

0.0.21 Join Point

- A join point is a specific point in the execution of a program where an aspect can be applied. Join points are identified by expressions that match specific points in the execution flow, such as method calls, object instantiations, or variable assignments.

1 Key Concepts of AOP

1.1 Advice

- Advice is the actual code that runs at a particular join point. There are different types of advice, such as "before," "after," "around," etc., specifying when the advice should be executed in relation to the join point.

1.2 Pointcut

- A pointcut is a set of join points. It defines the criteria for where an aspect's advice should be applied. Pointcuts use expressions to match join points based on method signatures, class names, or other criteria.

1.3 Weaving

- Weaving is the process of integrating aspects into the main business logic of the application. This can be done at compile-time, load-time, or runtime. Weaving allows the aspects to be applied to the appropriate join points.

1.4 Modularity and Reusability

- AOP promotes modularity by separating cross-cutting concerns from the main application logic. Aspects can be reused across different parts of the application, enhancing maintainability and reducing code duplication.

1.5 Encapsulation

- Cross-cutting concerns are encapsulated within aspects, providing a cleaner separation of concerns. This improves code organization and makes it easier to understand and maintain.

1.6 Dynamic Aspect

- AOP allows for dynamic aspects, where aspects can be added or removed during runtime. This flexibility provides the ability to adapt the behavior of the system without modifying its source code.

1.7 AspectJ

- AspectJ is a widely-used extension of the Java programming language that supports AOP. It introduces additional syntax and features to enable a more expressive and powerful AOP implementation.

2 Structure

- Core Modules
- Aspects
- Pointcuts
- Advice
- Weaver

3 Working of AOP

3.1 Identifying concerns

1. First, identify the different concerns or responsibilities of your program. For example, if you're writing a program to process orders, you might identify concerns such as order validation, payment processing, and order fulfillment.

3.2 Defining aspects

[resume]Once you've identified your concerns, you can define aspects for each one. An aspect is a modular unit of code that encapsulates a specific behavior or responsibility. For example, you might create an aspect for order validation, another for payment processing, and so on.

3.3 Determining join points

[resume]A join point is a specific point in your program's execution where an aspect can be applied. For example, a join point for the order validation aspect might be when a new order is submitted. You can define join points in your code using annotations or other markers.

3.4 Defining pointcuts

[resume]A pointcut is a set of join points where an aspect should be applied. For example, you might define a pointcut for the order validation aspect that includes all the join points where a new order is submitted. Pointcuts help narrow down the scope of an aspect so that it's only applied where needed.

3.5 Defining advice

[resume]Advice is the behavior an aspect provides at a join point. Several types of advice exist, including before, after, and around advice. Before advice is executed before the join point, after advice is executed after the join point, and around advice wraps the join point and can modify its behavior.

3.6 Weaving aspects

[resume]Weaving is the process of applying aspects to your program's execution. During weaving, the advice provided by aspects is injected into the appropriate join points in your code. This allows the behavior of aspects to be integrated into your program's execution.

3.7 Executing program

[resume]Once your aspects are woven into your program, you can execute the program as usual.

4 Advantages of AOP

1. **Modularity:** AOP helps to improve the modularity of software by allowing developers to separate cross-cutting concerns into modular units known as aspects. This can make it easier to understand, modify, and reuse code.
- **Maintainability:** By separating concerns, AOP can make it easier to maintain and modify code over time. This can be particularly useful when adding new features or behavior to an application without modifying the existing code.
- **Reusability:** Aspects can be reused across different projects, which can help to improve the efficiency and productivity of developers.
- **Performance:** AOP can be used to optimize the performance of an application by identifying and addressing bottlenecks or other inefficiencies.

5 Disadvantages of AOP

- **Complexity:** AOP can add complexity to an application, as it requires developers to learn and use an AOP framework and to understand the concepts of aspects, advice, pointcuts, and join points.
- **Overhead:** AOP can introduce additional overhead, as it requires the use of an AOP framework and the execution of additional code at runtime.
- **Compatibility:** AOP may not be compatible with all programming languages and environments, which can limit its use in some situations.
- **Debugging:** Debugging an application that uses AOP can be more challenging, as it may be difficult to understand the execution flow and the interactions between different aspects.

6 Difference Between Spring AOP, AspectJ, and JBoss

Join points	Spring AOP Compatibility	AspectJ Compatibility	JBoss AOP Compatibility
2*Calling Method	No	Yes	Yes
	No	Yes	Yes
2*Execution of Method	Yes	Yes	Yes
	No	Yes	Yes
2*Calling Constructor	No	Yes	Yes
	No	Yes	Yes
2*Execution of Constructor	No	Yes	No
	No	Yes	No
2*Execution of Static initialization	No	Yes	Yes
	No	Yes	Yes
2*Initialization of Object	No	Yes	Yes
	No	Yes	Yes
2*Field reference	No	Yes	Yes
	No	Yes	Yes
2*Field assignment	No	Yes	Yes
	No	Yes	Yes
2*Execution of Handler	No	Yes	Yes
	No	Yes	Yes
2*Execution of Advice	No	Yes	Yes
	No	Yes	Yes

Example

Spring AOP

```
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.journaldev.spring.service.EmployeeService;

public class SpringMain {

    public static void main(String[] args) {
        ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("spring.xml");
        EmployeeService employeeService = ctx.getBean("employeeService", EmployeeService.class);

        System.out.println(employeeService.getEmployee().getName());

        employeeService.getEmployee().setName("Pankaj");

        employeeService.getEmployee().throwException();

        ctx.close();
    }
}
```

AspectJ

```
public aspect LoggingAspect {

    before(): execution(* com.example.service.*.*(..)) {
        System.out.println("Logging before method execution in the service package.");
    }
}

public class MyService {

    public void doSomething() {
    }
}
```

JBoss AOP

```
import org.jboss.aop.advice.annotation.JoinPoint;
import org.jboss.aop.advice.annotation.Mixin;

@Mixin
public class LoggingMixin {

    public void logBeforeMethod(@JoinPoint Object target) {
        System.out.println("Logging before method execution in " + target.getClass().getName());
    }
}


@LoggingMixin
public class MyService {
```

```
    public void doSomething() {

    }
}
```

Implementation of Aspect-Oriented Programming

Logging

With AOP, a logging aspect can be applied to specific join points (e.g., method executions) to automatically log relevant information without modifying each method individually.

Security

Instead of embedding authorization logic within each method, an aspect can be applied to check user permissions at specific join points, ensuring that security concerns are centralized and easy to manage.

Transaction Management

In database interactions, managing transactions is a cross-cutting concern. AOP can be used to wrap methods with transactional behavior.

Exception Handling

An aspect can be applied to catch exceptions at specific join points, allowing for centralized error handling logic. This is particularly helpful for scenarios where consistent error reporting or logging is needed.

Performance Monitoring

Aspects can be applied to measure the execution time of methods or to collect other performance-related metrics. This information can be useful for identifying bottlenecks and optimizing critical parts of the system.

Audit Trails

AOP can be utilized to automatically generate audit trails by applying an aspect to specific join points, capturing relevant information such as who performed the action and when.

Resource Management

Managing resources like opening and closing connections, handling file operations, or releasing resources can be a cross-cutting concern.

Language for Paradigm 2: JBoss AOP

Brief history

2002: Inception and Early Development

- JBoss AOP was initiated by the JBoss Group as an open-source project to provide support for aspect-oriented programming in the JBoss Application Server.
- The project aimed to simplify the development and maintenance of enterprise Java applications by addressing cross-cutting concerns.

2003: First Releases

- JBoss AOP saw its early releases, offering features like dynamic AOP, which allowed aspects to be added or removed during runtime.
- The framework was integrated with the JBoss Application Server, making it an integral part of the JBoss ecosystem.

2004-2005: Enhancements and Improvements

- During this period, JBoss AOP underwent various enhancements and improvements, making it more robust and feature-rich.
- The framework continued to evolve to better meet the needs of developers dealing with complex enterprise applications.

2006: JBoss AOP Becomes JBoss Microcontainer

- JBoss AOP evolved into the JBoss Microcontainer project, which went beyond traditional AOP capabilities. The Microcontainer provided a lightweight, embeddable inversion of control (IoC) container with enhanced modularity features.

2008: Integration with JBoss AS 5

- JBoss AOP, now integrated into the JBoss Microcontainer, was a key component in JBoss Application Server 5 (AS 5).
- The Microcontainer and AOP capabilities were utilized to create a modular and extensible architecture for JBoss AS 5.

Overview of JBoss AOP

- JBoss AOP is a 100% Pure Java Aspect-Oriented Framework usable in any programming environment or tightly integrated with our application server.
- Aspects allow you to more easily modularize your code base when regular object-oriented programming just does not fit the bill.
- It can provide a cleaner separation from application logic and system code.
- Combined with Java Annotations, it also is a great way to expand the Java language in a clean pluggable way rather than using annotations solely for code generation.
- For example, metrics is one common aspect. To generate useful logs from your application, you have to (often liberally) sprinkle informative messages throughout your code.

Features and Advantages of JBoss

Features

1. **Dynamic AOP:** JBoss AOP supports dynamic aspects, allowing aspects to be added or removed during runtime. This provides flexibility in adapting the behavior of the application without requiring a restart.
2. **Aspect Instantiation and Lifecycle Management:** Aspects in JBoss AOP can be instantiated and managed throughout their lifecycle. The framework provides mechanisms for creating, reusing, and destroying aspect instances.

3. **Aspect Binding:** Aspects can be dynamically bound to specific join points at runtime. This dynamic binding allows for more granular control over when and where aspects are applied.
4. **Fine-Grained Pointcut Expressions:** JBoss AOP allows developers to define pointcuts using fine-grained expressions that specify join points based on method signatures, class names, or other criteria. This provides a high level of control over where aspects should be applied.
5. **Declarative Syntax:** The framework supports a declarative syntax for specifying aspects and pointcuts. Developers can define aspects in a concise and readable manner, making it easier to express cross-cutting concerns.
6. **Integration with JBoss Microcontainer:** JBoss AOP is integrated with the JBoss Microcontainer, which provides a lightweight, embeddable inversion of control (IoC) container. This integration enables a modular and extensible architecture for JBoss-based applications.
7. **Interceptors:** JBoss AOP uses interceptors to define the behavior associated with specific join points. Interceptors are the building blocks of aspects, and they can be chained together to form an aspect.
8. **Metadata Annotations:** Developers can use metadata annotations to express aspects, making it easier to apply aspects to code elements. Annotations provide a convenient and readable way to define aspects.
9. **Compatibility with JBoss Application Server (AS):** JBoss AOP was historically a key component of the JBoss Application Server. It played a crucial role in enhancing the modularity and flexibility of JBoss AS.
10. **Community Support:** JBoss AOP, being part of the JBoss ecosystem, benefited from community contributions and support. The community engagement helped in the improvement and maintenance of the framework.

Advantages

- JBoss deployment and configuration is easy and fast. This leads to lower cost and faster deliveries.
- JBoss gives you a lot of flexibility around performance tuning options to better suit your application needs.
- It is modular and cloud-ready. This can be installed on-premise and cloud with equal ease.
- Integration with modcluster:JBoss integrates well with modcluster, a smart httpd-based load balancing component that listens to incoming requests on the web server using httpd and then intelligently routes the request to JBoss hosts.

Disadvantages

- JBoss CLI is a great tool but had trouble using it to get values displayed on the JBoss GUI. It also has limitations parsing the applications.xml files, and a mix of jboss-cli and Linux bash commands had to be used to automate certain application administrative tasks.
- JBoss doesn't provide explicit performance tuning recommendations. It would have been nice if it could learn from the current demand vs current settings for things like connection pool, server configurations, garbage collection, etc.

Feature	JBoss AOP	Spring AOP
Goals	Full-fledged AOP solution	Simple AOP for Spring apps
Scope	Independent or JBoss integration	Spring beans only
Weaving	Compile-time & load-time	Runtime proxy-based
Joinpoints	Wider range (methods, fields, constructors)	Method calls only
Pointcut expressions	Powerful AspectJ language	Simpler matching language
Integration	Independent or JBoss	Tightly integrated with Spring
Performance	Compile-time weaving can be faster	Runtime weaving might have overhead
Complexity	More powerful, but complex	Simpler and easier to learn

Table 3: Comparison of JBoss AOP and Spring AOP Features

6.1 Limitations of JBoss AOP

- **1. Compile-time Weaving:**
  [label=–]
  - While JBoss AOP supports compile-time weaving, it relies on the AspectJ compiler, which can introduce additional complexity and potential compatibility issues with certain libraries or tools.
- **2. Load-time Weaving Overhead:**
  [label=–]Load-time weaving can have a performance impact, especially for large applications with many aspects.
- **3. Debugging Challenges:**
  [label=–]Debugging code that has been modified by load-time weaving can be more challenging, as traditional debuggers might not be able to fully understand the altered code structure.
- **4. Dependency on JBoss Application Server:**
  [label=–]While JBoss AOP can be used independently, it's tightly integrated with the JBoss application server, which can limit its flexibility in non-JBoss environments.
- **5. Limited Joinpoint Support:**
  [label=–]JBoss AOP doesn't support all possible joinpoints that AspectJ offers, such as exception handlers and static initialization blocks.
- **6. Configuration Complexity:**
  [label=–]Configuring JBoss AOP, especially with compile-time weaving, can be more complex compared to simpler AOP frameworks.
- **7. Deprecated Status:**
  [label=–]JBoss AOP has been deprecated in favor of AspectJ in newer versions of JBoss application server, indicating a shift in focus and potential lack of long-term support.

Code

Cross-Cutting Concern

–

```
public aspect LoggingAspect {
    pointcut methodCalls() : execution(* *(..));

    before() : methodCalls() {
        System.out.println("Entering method: " + thisJoinPoint.getSignature());
    }
```

```
    after() : methodCalls() {
        System.out.println("Exiting method: " + thisJoinPoint.getSignature());
    }
}
```

Aspect

```
public aspect SecurityAspect {
    pointcut secureMethods() : execution(@Secure * *(..));

    before() : secureMethods() {

    }
}
```

Join Point

```
pointcut accountOperations() : execution(* com.example.AccountService.*(..));
```

Advice

```
before() : accountOperations() {
    System.out.println("Before account operation: " + thisJoinPoint.getSignature());
}
```

Pointcut

```
pointcut stringReturningMethods() : execution(String *(..));
```

Weaving

```
<aop>
    <pointcut name="allMethods" expression="execution(* *(..))"/>
    <aspect name="LoggingAspect" pointcut="allMethods">
        <advice name="before" type="before" bind-to="allMethods"/>
    </aspect>
</aop>
```

Analysis

Object Oriented Programming

Strengths:

- **Modularity:** OOP promotes code breakdown into self-contained, reusable modules (objects), enhancing code organization, maintainability, and testability.
- **Encapsulation:** Objects encapsulate data and behavior, protecting internal data from external interference and promoting data integrity.
- **Reusability:** Objects can be reused in different parts of an application, reducing code duplication and development time.
- **Abstraction:** OOP allows focusing on essential features while hiding implementation details, simplifying complex systems and making code easier to understand.

- **Inheritance:** Objects can inherit properties and behaviors from parent classes, promoting code reusability and reducing redundancy.
- **Polymorphism:** Objects can be treated as their parent types, allowing flexible and adaptable code that can handle objects of different subclasses.

Weaknesses:

- **Complexity:** OOP can introduce complexity, especially in large systems with many interacting objects. Understanding relationships and dependencies can be challenging.
- **Overhead:** OOP can have runtime and memory overheads due to object creation, method calls, and inheritance hierarchies.
- **Steep Learning Curve:** OOP concepts like encapsulation, inheritance, and polymorphism can be difficult to grasp for beginners.
- **Not a Silver Bullet:** OOP isn't suitable for all problems. Some tasks are better suited for procedural or functional paradigms.
- **Tightly Coupled Code:** Improper design can lead to tightly coupled objects that are difficult to modify or reuse independently.
- **Over-Engineering:** Excessive use of OOP concepts can create overly complex and abstract code that's difficult to maintain.

Ruby

Strengths

- **Developer Productivity:** Ruby is known for its conciseness and readability, thanks to features like blocks, metaprogramming, and flexible syntax. This leads to faster development and easier code maintenance.
- **Web Development:** Ruby shines in web development, with frameworks like Rails offering rapid prototyping and a convention-over-configuration approach.
- **Large Community and Libraries:** Ruby has a vibrant and supportive community, with a vast collection of open-source libraries covering various functionalities.
- **Flexibility and Dynamism:** Ruby is dynamically typed, allowing for flexibility and experimentation, and provides metaprogramming features that empower developers to manipulate the language itself.
- **Focus on Developer Happiness:** Ruby prioritizes developer experience, with clean syntax, helpful error messages, and a focus on elegance and simplicity.

Weaknesses

- **Performance:** Compared to compiled languages like C++, Ruby can be slower, especially for computationally intensive tasks. However, frameworks like JRuby and optimizations can mitigate this.
- **Concurrency:** Ruby's Global Interpreter Lock (GIL) limits true concurrency for multi-threaded applications. Alternatives like asynchronous programming and event-driven frameworks exist, but achieving high concurrency can be challenging.
- **Maturity:** While mature, Ruby is not as widely used as languages like Java or Python. This can lead to a smaller talent pool and potentially fewer enterprise-grade tools and libraries.
- **Gems and Library Management:** Managing numerous RubyGems can become complex, and compatibility issues might arise due to the dynamic nature of the ecosystem.
- **Security Concerns:** Some Ruby libraries may have security vulnerabilities, and the dynamic nature of the language requires careful coding practices to avoid security risks.

Aspect Oriented Programming

Strengths

- **Modularization of Cross-Cutting Concerns:** AOP excels at separating concerns that span multiple parts of an application, like logging, security, and transaction management. This promotes cleaner code, better maintainability, and reduced code duplication.
- **Improved Maintainability:** By isolating cross-cutting concerns, changes to those concerns can be made in a single place, affecting the entire application without modifying core business logic.
- **Encapsulation of Concerns:** Aspects encapsulate cross-cutting logic, making it easier to understand and manage.
- **Reusability:** Aspects can be reused across different applications or modules, reducing development time and effort.
- **Dynamic Behavior:** AOP can enable dynamic weaving of aspects at runtime, allowing for adaptive behavior and system modifications without recompilation.
- **Testing and Debugging:** AOP can simplify testing and debugging by isolating concerns and making it easier to focus on specific parts of the system.

Weaknesses

- **Complexity:** AOP concepts like aspects, join points, and pointcuts can introduce additional complexity, especially for developers unfamiliar with the paradigm.
- **Performance Overhead:** AOP can have performance overhead due to the weaving process and additional method calls introduced by aspects.
- **Debugging Challenges:** Debugging code affected by AOP can be more difficult, as traditional debuggers might not fully understand the altered code structure.
- **Tooling and Support:** AOP tools and frameworks may not be as mature or widely adopted as those for traditional OOP, potentially limiting support and resources.
- **Learning Curve:** AOP requires understanding new concepts and terminology, which can be a barrier for some developers.
- **Potential for Misuse:** Improper use of AOP can lead to overly complex and tangled code, making it difficult to understand and maintain.

JBoss AOP

Strengths

- **Powerful and Flexible:** JBoss AOP offers a comprehensive set of features for implementing AOP concepts, including diverse joinpoints, pointcuts, and advice types. It allows for fine-grained control over aspect application and supports both compile-time and load-time weaving.
- **Integration with JBoss Application Server:** JBoss AOP integrates seamlessly with the JBoss application server, offering additional features and advantages when used within that environment.
- **Community and Resources:** While smaller than AspectJ, JBoss AOP has a supportive community and well-documented resources available for learning and troubleshooting.
- **Open-Source and Free:** JBoss AOP is open-source and free to use, making it accessible to a broad range of developers and projects.

Weaknesses

- **Deprecated:** JBoss AOP has been deprecated in favor of AspectJ in newer versions of the JBoss application server. While still functional, future support and development might be limited.
- **Complexity:** Due to its feature richness, JBoss AOP can be more complex to learn and configure compared to simpler AOP frameworks.
- **Performance Overhead:** Load-time weaving in JBoss AOP can introduce performance overhead, especially for large applications with many aspects.
- **Limited Joinpoint Support:** JBoss AOP doesn't support all potential joinpoints available in AspectJ, such as exception handlers and static initialization blocks.
- **Integration Issues:** Integrating JBoss AOP with non-JBoss environments or libraries might require additional effort and may not be as smooth as other options.

Comparison

Feature	Object-Oriented Programming (OOP)	Aspect-Oriented Programming (AOP)
Focus	Organizing logic around objects (data + behavior)	Modularizing cross-cutting concerns (affecting multiple parts)
Strengths	• Modularity (reusable object modules) • Encapsulation (protects internal data) • Reusability (inheritance & polymorphism) • Abstraction (simplifies complex systems)	• Modularization of cross-cutting concerns (logging, security) • Improved maintainability (centralized changes) • Encapsulation of concerns (easier to manage) • Reusability (reusable aspects) • Dynamic behavior (runtime weaving)
Weaknesses	• Complexity (large systems with many objects) • Overhead (object creation & method calls) • Steep learning curve (inheritance & polymorphism) • Not a silver bullet (not for all problems) • Tightly coupled code (improper design)	• Complexity (learning aspects & joinpoints) • Performance overhead (weaving & extra calls) • Debugging challenges (altered code structure) • Limited tooling & support (compared to OOP) • Potential for misuse (overly complex code)
Relationship	AOP complements OOP by handling cross-cutting concerns	Objects can have aspects applied for extended functionality
Best Use	Core application logic structuring with objects and relationships	Cross-cutting concerns like logging, security, error handling

Table 4: Comparison of Object-Oriented Programming (OOP) and Aspect-Oriented Programming (AOP) Features

Feature	Ruby	JBoss AOP
Type	General-purpose programming language	Aspect-oriented programming framework (Java-based)
Focus	Building web applications, scripting, rapid prototyping	Modularizing cross-cutting concerns in Java applications
Paradigm	Object-oriented, dynamically typed	Aspect-oriented, builds on Java's OOP model
Syntax	Known for readability and conciseness	Java-based syntax with AOP-specific constructs
Strengths	• Developer productivity • Web development • Flexibility • Community support	• Powerful AOP features • Integration with JBoss server • Performance (compile-time weaving)
Weaknesses	• Performance (compared to compiled languages) • Concurrency limitations • Library management	• Complexity • Deprecated status • Performance overhead (load-time weaving)
Best Use Cases	• Web development • Prototyping • Scripting • General-purpose programming	• Modularizing concerns like logging, security, transactions in Java applications

Table 5: Comparison of Ruby and JBoss AOP

Challenges Faced

1. **Very low resources for JBoss AOP.**
   (a) Used the resources uploaded on GitHub.
2. **Too much resources for object-oriented paradigm and could not find the exact thing for aspect-oriented programming.**
   (a) For that, I used a few AI tools.
3. **Too much time-consuming to make a report in Overleaf. These are some of the challenges.**

Conclusion

In wrapping up, Object-Oriented Programming (OOP) and Aspect-Oriented Programming (AOP) offer unique approaches to software development. OOP focuses on creating organized and reusable code, providing a strong basis for structuring the core logic of applications. On the flip side, AOP shines in simplifying complex issues that span different parts of an application, making it easier to maintain and reuse code.Ruby, a dynamically typed language, stands out for its emphasis on making developers more productive and code more readable, especially in web development. Despite being deprecated, JBoss AOP continues to be a robust tool for handling concerns in Java applications.However, both paradigms come with their own set of challenges. OOP can get intricate in larger systems, while AOP requires careful consideration due to its learning curve and potential impact on performance. The choice between them depends on the unique needs of a project.Throughout our exploration, collaborative tools like GitHub and Overleaf, assisted by AI, made resource management and report generation more efficient. As the landscape of software development evolves, understanding these paradigms equips developers with valuable tools for creating resilient and maintainable applications.

References

1. https://www.trustradius.com/reviews/red-hat-jboss-enterprise-application-platform-2019-03-15-11-35-09
2. https://www.baeldung.com/spring-aop-vs-aspectj
3. https://access.redhat.com/documentation/en-us/jboss$_enterprise_application_platform/5/html/administration_and_con$
4. https://medium.com/hprog99/aspect-oriented-programming-b9a06ca256db: :text=One
5. https://www.iaeng.org/publication/WCECS2019/WCECS2019$_pp118-123.pdf$
6. https://chat.openai.com/share/da3d7a43-8423-4afe-a0b5-788eb1da917c
7. https://chat.openai.com/share/8e3209bb-1f48-4905-8960-ea413923814e
8. https://chat.openai.com/share/92b3f520-6cef-4591-873e-3080414ce1f9
9. https://arasopraza.medium.com/introduction-to-ruby-programming-language-7014c1a651d8