Amrita Vishwa Vidyapeetham
TIFAC-CORE in Cyber Security

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

Saranesh E S

21st January, 2024

Paradigm 1: Object-Oriented Programming

Object-Oriented Programming (OOP) is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes), which are used to create individual instances of objects.Object-oriented programming is based on the notion of creating a model of the target problem in your programs. Object-oriented programming reduces programming errors and promotes the reuse of code. C++, Python, PHP, Java are some object-oriented programming languages.

Four main Principles of Object-Oriented Programming are Inheritance, Encapsulation, Abstraction and Polymorphism.

Inheritance:

1. The capability of a class to derive properties and characteristics from another class is called Inheritance.
2. Rather than defining new data functions while creating a class, you can inherit the existing class's data functions. The derived class takes over all the properties of the parent class and adds some new features to itself.
3. Types of Inheritance:
   - Single Inheritance: In this inheritance, a single class inherits the properties of a base class. All the data members of the base class are accessed

by the derived class according to the visibility mode that is specified during the inheritance.

- Multiple Inheritance: The inheritance in which a class can inherit or derive the characteristics of multiple classes, or a derived class can have over one base class. It specifies access specifiers separately for all the base classes at the time of inheritance.
- Multilevel Inheritance: The inheritance in which a class can be derived from another derived class is known as Multilevel Inheritance.
- Hierarchal Inheritance: The inheritance in which a single base class inherits multiple derived classes. This inheritance has a tree-like structure since every class acts as a base class for one or more child classes.
- Hybrid Inheritance: As the name suggests, it is the combination of two or over two types of inheritances.

4. Reusability, data hiding, reliability, maintainability are some advantages of Inheritance.

Encapsulation:

1. Encapsulation is the concept of binding fields and methods together as a single unit. A class allows programmers to create objects with variables (data) and behaviors (methods or functions). Since a class consists of data and methods packed into a single unit, it is an example of encapsulation.
2. It allows the construction of objects that encapsulate data and reveal only the methods or interfaces required for interaction with the outside world. This method encourages information concealment by prohibiting direct access to an object's internal state.
3. Types of encapsulations:
   - Access Modifiers: Access modifiers specify how visible or accessible class members (attributes and methods) are from outside the class.
     - Public: A public members access level is visible everywhere. Other classes, objects, or modules can access them.
     - Private: A private members access level is limited to the class. It isn't accessible outside of the class.

   - Protected: Protected members are available within the class and its subclasses. They must be reached from within the class hierarchy.

4. Data protection and security, modularity and reusability, code maintainability are some benefits of encapsulation programming.

Abstraction:

1. Abstraction is a methodology in which details of the programming codes are hidden away from the user, and only the essential things are displayed to the user. It's used to create a boundary between the application and the client programs.
2. Types of Abstraction:
   - Data Abstraction: Data abstraction is a technique for creating complicated data types and exposing only the actions that are necessary to interact with the data type, while keeping the implementation details hidden from outside activities.
   - Process Abstraction: Process abstraction is achieved by hiding the internal implementation of the many functions involved in a user operation.

Polymorphism

1. Polymorphism is the ability of any data to be processed in more than one form. It describes the concept that you can access objects of different types through the same interface. Each type can provide its own independent implementation of this interface.
2. Types of Polymorphism:
   - Compile time polymorphism: Method Overloading - multiple methods defined in the same class with the same name but different parameter lists. The compiler determines which method to call based on the number or types of arguments.
   - Runtime Polymorphism: Method Overriding - Inheritance allows a subclass to provide a specific implementation of a method that is already defined in its superclass.

Applications:

1. Stimulation and Modeling Systems: Complex systems are challenging to model because variables have different specifications. Complex systems stimulation requires precise modeling and knowledge of the interaction. OOP offers a suitable method for streamlining these complex structures.
2. Hypertext and Hypermedia: Hypertext is similar to regular text as it can be stored, searched, and edited easily. Hypermedia on the other hand is a superset of hypertext. OOP also helps in laying the framework for hypertext and hypermedia.
3. Client-Server systems: OOPs principles are quite helpful. To construct Object-oriented server internet, or OCSI, applications, the IT infrastructure is created using Object-oriented client-server systems.
4. Object-oriented Database: To preserve the object's integrity and identity, databases maintain a close correlation between database objects and their counterparts in the physical world.
5. Computer Aided Designs: OOP can be used in CAD software to develop classes that represent different design components including lines, curves, surfaces, and solids. Code reuse and modularity are made possible by inheritance, which can be used to construct relationships between various sorts of design elements.

Advantages:

1. Modularity through Breakdown: OOP excels in modularity by dividing complex systems into self-contained units, enhancing code reusability and maintainability.
2. Inheritance Benefits: OOP supports inheritance, allowing new classes to inherit properties and behaviors from existing classes, reducing repetition and promoting efficient code maintenance.
3. Clearer Code Structure: Component-based approach results in clear, comprehensible code that is easier to understand and test.
4. Enhanced Privacy and Security: Encapsulation conceals implementation details, with well-defined interfaces controlling access, reducing unintended changes and enhancing data privacy.

Language for Paradigm 1: PHP

PHP (recursive acronym for PHP: Hypertext Preprocessor) is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML. The Object-oriented programming feature was added in PHP5.

Features of PHP:

1. Simplicity: Easy to learn the language as the syntax is similar to that of 'C' or Pascal language. language is very logical and well organized general-purpose programming language. The core distribution helps the developers implement dynamic websites very easily with secured data. PHP applications are very easy to optimize.
2. Objective oriented: PHP supports object-oriented programming features like data encapsulation, inheritance, abstraction, polymorphism, etc.
3. Flexibility: it can be well integrated with HTML, XML, Javascript and many more. PHP can run on multiple operating systems like Windows, Unix, Mac OS, Linux, etc. It is very comfortably integrated with various Databases.
4. Open Source: All PHP frameworks are open sources, No payment is required for the users and its completely free. It supports a popular range of databases like MySQL, SQLite, Oracle, Sybase, Informix, and PostgreSQL.
5. Error reporting and exceptions: PHP has many pre-defined functions and reporting constants that generate errors at runtime. There are 16 levels of error in PHP5, representing the category and severity of an error in PHP.
6. Object oriented features: PHP supports object-oriented programming features, resulting in increased speed and introducing added features like data encapsulation and inheritance at many levels.
7. Real time access monitoring: PHP also provides a summary of user's recent logging accesses.

Implementation of OOP in PHP:

```php
<?php

class Car {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }

  function get_name() {
    return $this->name;
  }

  function set_color($color){
    $this->color = $color;
  }

  function get_color(){
    return $this->color;
  }
}

// Subclass extending Car
class SportsCar extends Car {
  // Additional property specific to SportsCar
  public $top_speed;

  // Constructor for SportsCar
  function __construct($name, $color, $top_speed) {
    $this->set_name($name);
    $this->set_color($color);
    $this->top_speed = $top_speed;
  }

  // Additional method specific to SportsCar
  function get_top_speed() {
    return $this->top_speed;
  }
}

// Create instances of the classes
$first_car = new Car();
$first_car->set_name('Maruti');
$first_car->set_color('Red');

$second_car = new SportsCar('Ferrari', 'Yellow', '250 mph');

// Accessing methods from the parent class
echo "The name of the first car is: " . $first_car->get_name();
echo "<br>";
echo "The color of the first car is: " . $first_car->get_color();
echo "<br>";

// Accessing methods from the subclass
echo "The name of the sports car is: " . $second_car->get_name();
echo "<br>";
echo "The color of the sports car is: " . $second_car->get_color();
echo "<br>";
echo "The top speed of the sports car is: " . $second_car->get_top_speed()
?>
```

Analysing how the principles of Object-Oriented Programming are implemented in the PHP code:

1. Inheritance: The SportsCar class extends the Car class using the extends keyword. This demonstrates the principle of inheritance as SportsCar inherits properties and methods from the Car class.
2. Abstraction: The class Car provides a set of methods (set name, get name,

set color, get color) that allow interaction with the object without revealing the internal details. This demonstrates the principle of abstraction by hiding the implementation details and exposing only what is necessary.

3. Encapsulation: It is demonstrated through the use of public properties and methods, allowing controlled access to the class members.This follows the principle of encapsulation by bundling the data (properties) and methods that operate on the data within a single unit (class).
4. Polymorphism: Polymorphism is not explicitly demonstrated in this code. While SportsCar could be used interchangeably with Car, true polymorphism would involve a common interface or method overloading.

Paradigm 2: Imperative

Imperative programming is a approach that employs statements to modify the state of a program. Functions are implicitly coded at every step necessary to solve a problem, hence pre-coded models aren't used. Imperative programming tells the machine "how" to do it.
Imperative Programming (IP) is one of the popular Programming Paradigms which executes a sequence of steps/instructions/statements in some order. Examples of IP Languages:- Java, C, C++, COBOL etc

Characteristics of Imperative Programming:

1. Variable and Storage
2. Commands:
   - Assignments
   - Procedure call: The effect of a procedure call is to apply a procedure abstraction to some arguments. The net effect of a procedure call is to update variables (local or global).
   - Sequential commands: Much of imperative languages are concerned with control flow, making sure that commands are executed in a specific order.

- Collateral commands: A computation is deterministic if we can predict in advance exactly which sequence of steps will be followed. Otherwise the sequence is nondeterministic.
- Conditional commands: A conditional command has a number of subcommands, from which exactly one is chosen to be executed.
- Iterative commands: An iterative command, also known as loop, has a set of commands that is to be executed repeatedly and some kind of phrases that determines when the iteration will stop.

Concepts of Imperative Programming:

1. State Modification:Imperative programming involves statements that change a program's state. Programs consist of commands for the computer to perform, much like imperative mood in natural languages expresses commands.
2. Step-by-Step Operation: Imperative programming focuses on describing how a program operates step by step. Emphasis is on the sequence of commands rather than high-level descriptions of expected results.
3. Procedural Programming: Procedural programming is a type of imperative programming. Programs are built from procedures (subroutines or functions).
4. Structured Programming: Heavy procedural programming, where state changes are localized to procedures, is a form of structured programming. Structured programming techniques improve maintainability and overall quality of imperative programs.
5. Object-Oriented Programming (OOP): Object-oriented programming extends structured programming concepts.

Techniques and Functions:

1. Assignment Statements: In imperative paradigm, assignment statements perform operations on information located in memory and store results in memory for later use.
2. Complex Expressions: High-level imperative languages permit the evaluation of complex expressions involving arithmetic operations and function evaluations.

3. Looping Statements: Looping statements (while loops, do-while loops, for loops) allow the execution of a sequence of statements multiple times.
4. Conditional Branching: Conditional branching statements allow the execution of a sequence of statements only if a certain condition is met.
5. Unconditional Branching: Unconditional branching statements allow the transfer of the execution sequence to another part of the program (e.g., jump/goto, switch, subprogram call).
6. Blocks and Subroutines: The introduction of the block and subroutines enables hierarchical decomposition, allowing the construction of complex structures.

Advantages of Imperative Programming:

1. Readability: Imperative programming employs a procedural, step-by-step approach that enhances code readability. This procedural clarity facilitates easier comprehension and understanding of the program's logic.
2. Maintenance and Optimization: Imperative programs are amenable to collaborative maintenance and optimization efforts by multiple developers. Ease of collaboration facilitates ongoing improvements, bug fixes, and performance optimizations.
3. Application-Specific Considerations: Imperative programming allows developers to explicitly address application-specific considerations during code construction. This flexibility permits tailoring programs to unique requirements.
4. Control Flow: We can direct the program's control flow with imperative programming. This gives us more power to change, modify, and optimize our program.

Disadvantages of Imperative Programming:

1. Code Complexity: As problems grow in complexity, imperative codebases can become intricate, challenging to manage, and prone to reduced readability.

2. Error-Prone Editing: Manual code editing in imperative programming introduces a higher likelihood of errors, particularly in larger codebases.
3. Susceptible to Ineffient Code: Since most of the instructions are given by the programmer, the code may become inefficient if the programmer uses inefficient algorithms and coding methods.
4. Side-Effects: Imperative programming uses mutable variables, i.e., data can be changed inside the program. This leads to many side-effects

Language for Paradigm 2: COBOL

COBOL: A Language for Business Applications
COBOL (1959) stands for "COmmon Business Oriented Language." While Fortran led advancements in scientific computing, yet another language rose to dominance elsewhere - COBOL. Emerging during the late '50s, COBOL became synonymous with business data processing due to its human-like syntax which made code easier to read and write. This helped non-programmers comprehend complex business rules embedded within programs, thus cementing COBOL's position in businesses worldwide. It is imperative, procedural, and object-oriented.

Key COBOL features

1. Global business language: Business-Centric Design, Early Adoption in Business Applications, Stability and Reliability were some reasons why it was a global business language.
2. Seamless integration with modern systems: COBOL is a legacy language that supports and integrates easily with most traditional deployments, architectures, modern technologies, and complex applications. COBOL has ensured that the language integrates and connects seamlessly with modern applications and expands its functionality to the web, mobile, and cloud.
3. Easy readability: COBOL came into existence with the motto of developing a language that communicates better with computers. The language does not use pointers, user-defined data types or functions, making it a simple language to understand.

4. Portable language: COBOL programs run on different platforms. The platform-agnostic aspect also allows developers to build, test, and deploy COBOL programs across various supported platforms, thereby speeding up the development and application execution process.

Application of Cobol

1. Financial Systems: widely used in the financial industry for applications such as banking systems, accounting software, and financial transaction processing. Its ability to handle complex calculations and manage large datasets is valuable in these contexts.
2. Insurance Applications: used for policy management, claims processing, and other critical functions. COBOL's robustness and ability to handle large amounts of data make it suitable for insurance-related applications.
3. Government Systems: used for applications like tax processing, social security systems, and public administration. The language's stability and longevity make it a reliable choice for maintaining essential government services.
4. Healthcare Systems: used for managing patient records, billing systems, and other administrative tasks. The language's suitability for processing and organizing large datasets is beneficial in healthcare applications.
5. Human Resources Management: used in HR systems for payroll processing, employee record-keeping, and other personnel management tasks. Its structured nature supports the organization of employee-related data.

Understanding characteristics of Cobol with Imperative Programming:

A simple interactive program that accepts employee details and displays them, incorporating key characteristics of the imperative programming paradigm.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SampleProgram.

DATA DIVISION.
```

```
WORKING-STORAGE SECTION.
01 Employee-Name    PIC X(30).
01 Employee-Age     PIC 9(3).
01 Employee-Salary  PIC 9(7)V99.
01 Employee-Department PIC X(20).

PROCEDURE DIVISION.
   DISPLAY 'Enter Employee Name: '.
   ACCEPT Employee-Name.

   DISPLAY 'Enter Employee Age: '.
   ACCEPT Employee-Age.

   DISPLAY 'Enter Employee Salary: '.
   ACCEPT Employee-Salary.

   DISPLAY 'Enter Employee Department: '.
   ACCEPT Employee-Department.

   DISPLAY 'Employee Details:'.
   DISPLAY 'Name: ' Employee-Name.
   DISPLAY 'Age: ' Employee-Age.
   DISPLAY 'Salary: ' Employee-Salary.
   DISPLAY 'Department: ' Employee-Department.

   IF Employee-Age > 60
      DISPLAY 'This employee is eligible for retirement benefits.'
   END-IF.

   STOP RUN.
```

Characteristics Represented:

1. Statements as Commands: DISPLAY and ACCEPT
2. Command Order is Critical: The order in which commands are written is critical to the correct execution of the program.

3. Programmers Control All Aspects: Programmers explicitly define data structures (DATA DIVISION) and program logic (PROCEDURE DIVISION).
4. Statements Modify Program State: Each ACCEPT statement modifies the state of the program by capturing user input, and the subsequent DISPLAY statements modify the output state.
5. Imperative Style

Analysis

Strengths - PHP and Object-Oriented Programming:

1. Modularity and Code Reuse: OOP supports modularity by allowing developers to break down their code into smaller, manageable components called classes and objects. This makes it easier to maintain and update the code over time. Additionally, OOP enables code reuse, saving time and effort as developers can use existing classes and objects in new contexts.
2. Encapsulation: Referring to the ability to hide the internal properties and behavior of an object from external code. This ensures that changes made to the object have controlled consequences, promoting a cleaner and more organized code structure.
3. Inheritance: In PHP, OOP includes the concept of inheritance, allowing developers to create new classes that inherit properties and behavior from existing ones. This simplifies the creation of new objects with shared functionality, reducing redundancy and promoting a structured code hierarchy.
4. Polymorphism: Polymorphism in PHP means using the same interface to represent different types of objects. This flexibility is particularly useful when dealing with diverse datasets or building complex applications, as developers can interact with objects in a unified way.

Weakness - PHP and Object-Oriented Programming:

1. Security: Since it is open sourced, all people can see the source code. If there are bugs in the source code, it can be used by people to explore the weakness of it.

2. Not suitable of large applications: It will be difficult to use it for programming huge applications. Since the programming language is not highly modular, huge applications created out of the programming language will be difficult to maintain.
3. Weak type: Implicit conversion may surprise unwary programmers and lead to unexpected bugs. Confusion between arrays and hash tables. This is slow and could be faster. There are often a few ways to accomplish a task. It is not strongly typed. It is interpreted and uses curly braces.
4. Poor Error Handling Method: The framework has a bad error handling method. It is not a proper solution for the developers. Therefore, as a qualified PHP developer, you will have to overcome it.

Strengths - COBOL and Imperative Programming:

1. Efficient Data Handling: COBOL excels in managing complex data structures and processing large datasets, making it a top choice for applications dealing with extensive business data.
2. Reliable Transaction Processing: Industries like banking and finance, COBOL stands out for its reliable transaction processing capabilities. It ensures the integrity of critical business operations with its robust handling of high-volume transactions.
3. Optimized for Batch Processing: Optimized for efficient batch processing, particularly crucial for tasks like payroll processing. Its efficiency in handling scheduled batches of data sets it apart in scenarios requiring streamlined batch operations.
4. Precision in Decimal Arithmetic: COBOL's support for decimal arithmetic ensures precise control over numeric values, a crucial feature in financial applications where accurate handling of monetary data is essential.

Weaknesses - COBOL and Imperative Programming:

1. Limited Modern Features: COBOL lacks some of the advanced features present in modern programming languages. This includes support for sophisticated data structures and object-oriented programming constructs.

2. Verbose Code and Readability Challenges: COBOL code tends to be verbose, potentially impacting readability and increasing development time. While its syntax is designed for business comprehension, the extensive use of boilerplate code may hinder quick understanding.
3. Challenges in Parallel Processing: Traditional COBOL setups may face limitations in robust parallel processing support, impacting scalability in contemporary computing environments. This limitation becomes crucial in scenarios where concurrent task handling is essential.
4. Perception of Obsolescence: COBOL is sometimes seen as outdated, leading to a perception of obsolescence. This perception might discourage new developers from learning COBOL, potentially affecting the availability of skilled practitioners.

Comparison

Compare and contrast the two paradigms and languages, highlighting similarities and differences.
Comparison of the two paradigms and languages:
Similarities:

1. Programming Paradigm: Both PHP and COBOL share the imperative programming paradigm. This means they use statements to provide instructions for the computer to execute.
2. Platform Independence: Both PHP and COBOL can run on various operating systems without significant modifications, making them platform-independent.
3. Business Applications: Historically, both languages have been used for business applications. COBOL, especially, has found its place in financial, administrative, and other business-related systems.
4. Data Storage and File Handling: Both PHP and COBOL exhibit strong capabilities in data storage and file handling, essential aspects for managing information in diverse applications.
   - File Handling: Both languages offer features for efficient reading from and writing to files. While PHP excels in web applications, providing versatile file handling functions, COBOL is historically strong in

batch processing, efficiently managing large volumes of data stored in sequential files.

- Data Storage: PHP seamlessly integrates with various databases, enabling interaction with structured data. COBOL, on the other hand, often deals with flat files and databases in mainframe environments, showcasing its prowess in handling business data effectively.

Differences:

1. Domain and Usage:
   - PHP: Primarily employed in web development, PHP is a server-side scripting language designed for creating dynamic web pages and applications.
   - COBOL: Traditionally used in business, finance, and administrative systems, especially in mainframe computing and legacy environments.
2. Web vs. Mainframe:
   - PHP: Tailored for web applications, it's used to create server-side scripts generating HTML content and interacting with databases.
   - COBOL: Historically utilized for mainframe computing, it excels in large-scale
3. Syntax and Structure:
   - PHP: Features a C-like syntax and is designed to be embedded within HTML, making it well-suited for web development with support for modern programming features.
   - COBOL: Boasts a more verbose, English-like syntax, known for readability and self-documenting nature, which aids business analysts' comprehension.
4. Runtime Environments:
   - PHP: Operates as an interpreted language, executing code at runtime for flexibility and ease of development.
   - COBOL: Can be either compiled or interpreted, with programs often compiled into machine code for efficient execution, though interpreters also exist.

5. Community and Ecosystem:

    - PHP: Thrives with a large, active community, extensive documentation, and a wide array of third-party libraries and frameworks, especially for web development.
    - COBOL: While still in use for many legacy systems, its community has diminished relatively, and modern development practices may not be as prevalent.

Challenges Faced

Discuss any challenges you encountered during the exploration of programming paradigms and how you addressed them.

Imperative Programming Paradigm:

1. Code Duplication:

    - Challenge: Code duplication is a prevalent problem in imperative programming, causing maintenance issues and requiring changes in multiple places when modifications are needed.
    - Addressing: Enhance code re-usability by encapsulating common functionality in functions or procedures. Adopt modular design principles to minimize redundancy and improve maintainability.

2. Global State Management:

    - Challenge: Imperative programming often relies on mutable global state, making it challenging to track changes and understand the flow of data throughout the program.
    - Addressing: Minimize the use of global state where possible. Encourage the use of local variables and parameters to create more self-contained and predictable functions.

3. Procedural Complexity:

    - Challenge: Managing procedural complexity becomes a challenge as the code structure grows. Large programs with numerous procedures can become difficult to comprehend and maintain.

- Addressing: Break down complex procedures into smaller, modular functions. Encourage the use of functions with well-defined responsibilities, making the code structure more readable and maintainable.

Object-Oriented Programming Paradigm:

1. Complex Debugging:
   - Challenge: Debugging in an object-oriented paradigm can be more complex due to the interconnected nature of objects. Identifying the source of issues in a system with numerous interacting objects may require advanced debugging skills.
   - Addressing: Adopt good coding practices, modular design, and effective testing strategies. Proficient use of debugging tools.
2. Inheritance Overuse:
   - Challenge: Over-reliance on inheritance can lead to a rigid and inflexible code structure. Inappropriate use of inheritance hierarchies may result in code that is difficult to extend or modify.
   - Addressing : Minimize the use inheritance if not necessary.
3. Design Organisation:
   - Challenge: The design phase in OOP can sometimes take longer due to the need for careful consideration of class hierarchies, relationships, and encapsulation.
   - Address: Find a balance between a design architecture and program needs

Conclusion

The choice between OOP and Imperative Programming depends on the specific needs of the application and the preferences of the development team. OOP brings modularity, code reuse, and abstraction, making it suitable for complex systems. Imperative Programming, exemplified by COBOL, shines in data-centric, transactional applications where efficiency and reliability are paramount. Understanding the strengths and weaknesses of each paradigm allows developers to make informed decisions based on the requirements of the project. Ultimately, both paradigms have their place in the diverse landscape of programming languages, each addressing unique challenges and catering to distinct application domains.

References

Include any references or sources you consulted for your assignment.
https://www.educative.io/blog/object-oriented-programming
https://www.ibm.com/docs/en/watsonx-as-a-service?topic=language-object-oriented-programming
https://www.javatpoint.com/inheritance
https://www.scaler.com/topics/encapsulation-in-oops/
https://www.analyticsinsight.net/top-10-applications-of-object-oriented-programming-using-python-language/
https://www.php.net/manual/en/index.php
https://pbphpsolutions.com/oops-concepts-in-php-with-realtime-examples.html
https://www2.seas.gwu.edu/ bell/csci210/lectures/imperative$_{l}anguages.pdf$