20CYS312 - Principles of Programming Languages
Exploring Programming Paradigms

Assignment-01

Presented by Nithin S
CB.EN.U4CYS21051
TIFAC-CORE in Cyber Security
Amrita Vishwa Vidyapeetham, Coimbatore Campus

Feb 2024

AMRITA
VISHWA VIDYAPEETHAM

Outline

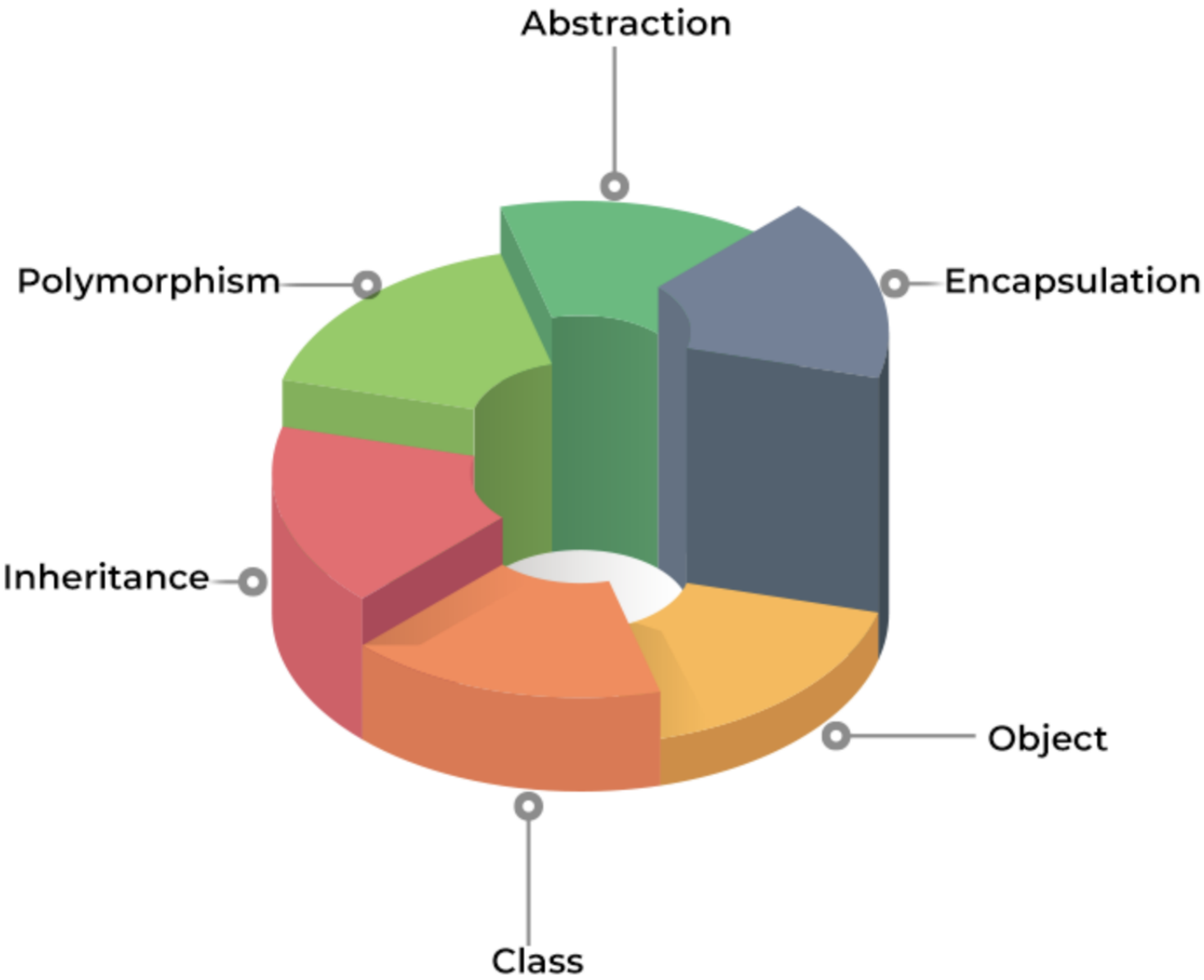
Object Oriented Programming

Definition

OOP is defined as a programming paradigm built on the concept of objects, i.e., a set of data contained in fields, and code, indicating procedures – instead of the usual logic-based system.

- OOP approach identifies classes of objects that share a close relationship with each other related to methods with which they are associated.
- It arranges a computer program into fundamental, reusable code structures called "classes." Then, new and distinct objects with related functions are created by reusing these classes.
- Languages like Java, C++, and Python provide support for OOP.

OOP Concepts



Concepts of OOP

1. **Class:** It is a blueprint or template for creating objects. It defines the properties and methods that objects of that class will have.
2. **Objects:**Objects are instances of classes. They are created from a class blueprint and have their own state (values of attributes) and behavior (methods to perform actions).
3. **Inheritance:** It allows the creation of new classes based on existing classes. It enables the reuse of code and the creation of class hierarchies.
4. **Polymorphism:**It allows objects of different classes to be treated as objects of a common superclass and they can take on more than one form. It is achieved through method overriding and method overloading. Two Types: Compile time and run time.
5. **Abstraction:** It focuses on defining the essential characteristics and behavior of an object, while hiding the implementation details.
6. **Encapsulation:**It is the practice of hiding the internal details of an object and exposing only the necessary information and functionality. Access to object attributes and methods can be controlled through access modifiers.

Key difference between Abstraction and Encapsulation:
1.Encapsulation primarily shields internal workings from other code within the program, while abstraction implementation from users of a class or interface. 2. Encapsulation is often achieved through access modifi (public, private, protected), while abstraction is often achieved through abstract classes and interfaces.

Advantages of OOP

1. **Enables code reusability:**
   - The idea of inheritance in object-oriented programming allows attributes of a class to be inherited, eliminating the need for duplication of effort.
   - This prevents the problems associated with repeatedly writing the same code.
2. **Increases productivity in software development:**
   - Programs can be created from pre-written, interconnected modules, saving time and increasing productivity.
   - This eliminates the need to start from scratch for each project.
3. **Reinforces security:**
   - The concept of data abstraction in OOP allows only a small amount of data to be displayed to the user, enhancing security.
   - This feature ensures that only essential information is accessible, strengthening security measures.
4. **Results in flexible code:**
   - One advantage of OOP is polymorphism, allowing a piece of code to exist in more than one version.
   - This flexibility enables adapting and extending code functionality without modifying the existing codebase.

Object-Oriented - C-Sharp

It is a modern, general-purpose programming language developed by Microsoft's Anders Hejlsberg in the year 2000.
It can be used to perform a wide range of tasks and objectives that span over a variety of professions. C-Sharp is primarily used on the Windows .NET framework.
C-Sharp supports Object Oriented, Imperative programming, Functional Programming paradigms.

By the help of C-Sharp programming language, we can develop different types of secured and robust applications:

- Window applications
- Web applications
- Distributed applications
- Web service applications
- Database applications, etc.

Apps made with C-sharp are : Microsoft Visual Studio,Paint.NET, FlashDevelop, NMath, Pinta, OpenRA.

C-Sharp

Program for Calculating **Factorial**in C-Sharp:

```
using System;

class FactorialCalculator{
    static long CalculateFactorial(int n){
        if (n == 0 || n == 1)
            return 1;
        return n * CalculateFactorial(n - 1);
    }
    static void Main(){
        Console.WriteLine("Enter a number to calculate its factorial:");
        if (int.TryParse(Console.ReadLine(), out int number)&&number>= 0){
            long factorial = CalculateFactorial(number);
            Console.WriteLine($"Factorial of {number} is: {factorial}");
        }
        else{
            Console.WriteLine("Invalid input.");
        }
    }
}
```

Advantages of C-Sharp

Advantages of C-Sharp Programming:

1. **Time Efficiency:** It offers static typing and readability, reducing debugging time. It emphasizes simplicity and efficiency, saving developers from writing complex code.
2. **Ease of Learning:** It has a low learning curve, making it beginner-friendly. Aspiring developers can comfortably enter the programming field without feeling overwhelmed.
3. **Scalability and Maintenance:** It is highly scalable and easy to maintain due to its strict static code nature. Old projects remain consistent, making adjustments and maintenance straightforward.
4. **Supportive Community:** It boasts a strong community on platforms like StackOverflow and Meetup.com, providing a valuable resource for collaboration, problem-solving, and knowledge sharing.

Characteristics and features of the C-Sharp

Key characteristics and features of C-Sharp that align with OOP

- **Classes:** Blueprints for creating objects with properties (data) and methods (behavior).
- **Objects:** Instances of classes, representing real-world entities or concepts.
- **Inheritance:** Allows new classes (subclasses) to inherit properties and methods from existing classes (base classes), promoting code reuse and extensibility.
- **Polymorphism:** Enables objects of different classes to be treated as the same type, allowing for flexible and adaptable code.
- **Encapsulation:** Bundles data and methods within classes, protecting data integrity and promoting modularity.
- **Abstraction:** Hides implementation details and exposes only essential features, simplifying interfaces and making code more manageable.

Imperative Paradigm

Definition

It is the process of giving clearly-defined sequence of instructions to a computer to follow in a predetermined order. The reason it's termed "imperative" is that, as programmers, we specify exactly what the machine must do, and how.

Examples of IP Languages:- Java, C, C++ etc.. Say you want a chocolate coffee. The imperative approach to do might look like this:

- Brew coffee.
- Melt chocolate.
- Mix melted chocolate into coffee.
- Stir well.
- Add sugar or sweetener.

It's that simple.

Imperative Paradigm

Actual code example: Let's say we want to calculate the sum of the first 10 natural numbers.

```
int sum = 0;
for (int i = 1; i <= 10; i++){
    sum += i;
}
printf("Sum of the first 10 natural numbers: %d\n", sum);
return 0;
```

Here we're telling our program to iterate through 1 to 10 and add all the values to get the result and print it as the sum of all elements.
We're being detailed and specific in our instructions, and that's what imperative programming stands for.

Features and Concepts of IP

Key Features and Concepts of Imperative Programming

1. **State and Variables:** It involves managing and altering the program's state by utilizing variables. These variables serve as containers capable of holding values that can be dynamically changed during the course of program execution.
2. **Assignment Statements:** It make use of assignment statements to update the values held by variables. Through assignment statements, developers can manipulate and control the flow of data within the program, facilitating dynamic changes in variable values.
3. **Control Flow:** It employs control flow structures to dictate the sequence in which statements are executed. This includes the use of loops and conditionals for decision-making processes. It mechanisms enable the program to adapt and respond to different scenarios, allowing for efficient and flexible execution of statements.
4. **Procedures and Subroutines:** It encourages the organization of code into reusable procedures or subroutines. These are modular blocks of code that can be called and executed at various points within the program.
5. **Mutable State:** In IP, the program's state is subject to modification during execution. This implies that variables can be altered, leading to side effects that influence the behavior of the program.

Imperative - FORTRAN

- Fortran is a general-purpose programming language, primarily designed for mathematical computations in science applications.
- It was developed by John Backus for IBM to overcome the obstacles presented by machine code in the creation of complex programs.
- "Fortran" is an acronym for FORmula TRANslation.
- Fortran (1950) was the first high-level programming language.
- The work on Fortran started in the 1950s at IBM, and there have been many versions since.
- The most common Fortran version today is still Fortran 77, although Fortran 90 is growing in popularity.

Along with declarations, expressions, and statements, it supported:

- Arrays
- Subroutines
- "Do" loops

FORTRAN Structure

A Fortran program generally consists of a main program (or driver) and possibly several subprograms (or procedures or subroutines). The STRUCTURE of the main program is:

- Program Name
- Declarations
- Statements
- Stop
- End

FORTRAN 77 Basics: A Fortran program is just a sequence of lines of text. The code has to follow a certain syntax to be a valid Fortran program. We will now look at a simple example where we calculate the area of a circle:

```
PROGRAM CIRCLE
      REAL R, AREA
C THIS PROGRAM READS A REAL NUMBER R AND PRINTS
C THE AREA OF A CIRCLE WITH RADIUS R.
      WRITE (*,*) 'GIVE RADIUS R:'
      READ (*,*) R
      AREA = 3.14159*R*R
      WRITE (*,*) 'AREA = ', AREA
      STOP
      END
```

Charcteristics and Features of FORTRAN:

Characteristics and Features or How does it embody IMPERATIVE Principles:

1. **Emphasis on Statements and Explicit Instructions:**
   - Fortran code consists of explicit statements (e.g., assignments, loops, conditionals).
   - The order of statements controls program flow, emphasizing explicit instruction.
2. **Mutable Variables and State Change:**
   - Fortran variables can be mutated during execution, allowing dynamic data manipulation.
   - Imperative, state-changing behavior is a fundamental characteristic.
3. **Control Flow Constructs and Algorithmic Steps:**
   - Fortran provides IF/ELSE and DO loops for precise control flow.
   - These constructs enable the expression of algorithms with well-defined steps.
4. **Procedural Organization and Modularization:**
   - Fortran uses procedures (SUBROUTINES, FUNCTIONS) for modularization.
   - Procedures enhance code organization and maintainability.
5. **Focus on Scientific Computation:**
   - Fortran aligns with imperative principles for precise calculations in scientific applications.
   - Suited for domains requiring step-by-step numerical algorithms.

Comparison and Discussions

Object Oriented Programming (OOP) is a super set of Imperative Programming. It follows all characteristics of IP with some extra features. Those extra features are:

- Everything is an Object.
- Each Object contains Some Data Fields and Methods.
- OOPs Concepts: Abstraction, Encapsulation, Inheritance, and Polymorphism.

Similarities

- Both define programs as sequences of instructions.
- Both use variables to store and manipulate data.
- Both can be used to solve a wide range of problems.

Differences:

1. Focus:
   - Imperative: How instructions change program state.
   - OOP: Data and operations bundled in objects
2. Code organization:
   - Imperative: Procedural, step-by-step.
   - OOP: Modular, object-oriented.

Comparison and Discussions

3. Data access
   - Imperative: Open access to variables.
   - OOP: Controlled access through object methods.
4. Code Reusability
   - Imperative: Limited through functions.
   - OOP: High through inheritance and polymorphism

Suitable Problems:

- Imperative: Procedural tasks, step-by-step calculations, simple algorithms.
- OOP: Real-world scenarios with complex entities and relationships, simulations, graphical user interfaces.

Which is better?

- There's no one-size-fits-all answer. Choose the paradigm that:
- **Matches the problem complexity:** OOP for complex, interrelated entities; imperative for simpler, procedural tasks.
- **Promotes code maintainability and reusability:** OOP offers greater modularity and reuse potential.
- **Suits your team's expertise and preferences:** Different paradigms have differe learning curves.

References

- https://www.freecodecamp.org/news/an-introduction-to-programming-paradigms/#what-is-a-programming-paradigm
- https://www.spiceworks.com/tech/devops/articles/object-oriented-programming/
- https://deb.ugc.ac.in/Uploads/SelfLearning/HEI-Exempted-U-0748/HEI-Exempted-U-0748_SelfLearning_20231026135924.pdf
- https://www.inscc.utah.edu/~krueger/6150/Fortran77_tutorial.pdf
- https://en.wikipedia.org/wiki/Imperative_programming
- https://iopscience.iop.org/article/10.1088/1757-899X/714/1/012001/pdf
- https://www.pluralsight.com/blog/softwaredevelopment/everything-you-need-to-know-about-c
- https://medium.com/@Ariobarxan/what-is-a-programming-paradigm-ec6c5879952b
- https://www.digitalocean.com/community/tutorials/functional-imperative-object-oriented-programming-comparison