Lecture 01

Outline

1. Procedural Programming
    a. Characteristics
        i. Based on concepts of,
            1. Procedures
            2. Functions
        ii. Step-by-step instructions
        iii. Focuses on functions
        iv. top-down approach for designing programs

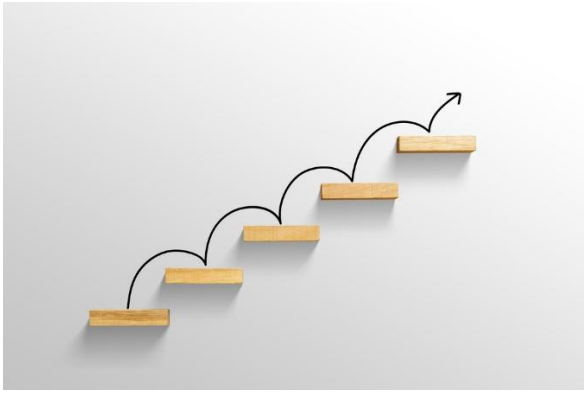
2. OOP
        i. Characteristics
            1. Programs are organized as a **collection of objects**.
            2. Allows us to solve more complex problems easily.
            3. Complex system developed using smaller **sub systems**
            4. Sub systems are,
                a. Independent
                b. Reusable
        ii. Process of developing an OOP
            1. **Identify objects** needed
            2. **Group the objects** and **creating classes** for similar objects
            3. **Create objects** using **classes**
            4. **Assemble** the objects
3. Definitions of keywords
    a. Class
        i. Attributes
        ii. Properties
        iii. Methods
        iv. Functions
    b. Encapsulation
    c. Public and Private access

Procedural Programming

- Based on the concept of procedures or functions.
    - **Procedure:** Perform a task but does not return a value.
    - **Functions:** Perform a task and return a value.

Feature	Function	Procedure
Returns a value?	✅ **Yes**	❌ **No**
Used for calculations?	✅ **Yes**	❌ **No**
Used for actions (e.g., printing)?	❌ **No**	✅ **Yes**

- Code is organized as a sequence of **step-by-step instructions**.
- It focuses on **functions** that operate on shared data.
- Programs are designed using a **top-down** approach.



Object Oriented Programming

- Object-Oriented Programming organizes programs as a collection of objects.
- These objects cooperate to solve problems.
- It helps solve complex problems more easily.
- A complex system consists of smaller subsystems.
- Independent units can be reused to solve different problems.

Process of Developing OOP

1. Identify Objects Needed
   The small subsystems that are used to create the main system.

2. Group the objects according to their characteristics
   Identify the similarities between objects and create classes that can create those objects.

3. Creating objects from classes
   Create the needed objects from classes that identified in first step.

4. Assemble the objects
   Connect these objects(sub-systems) together and make the main system.

Classes

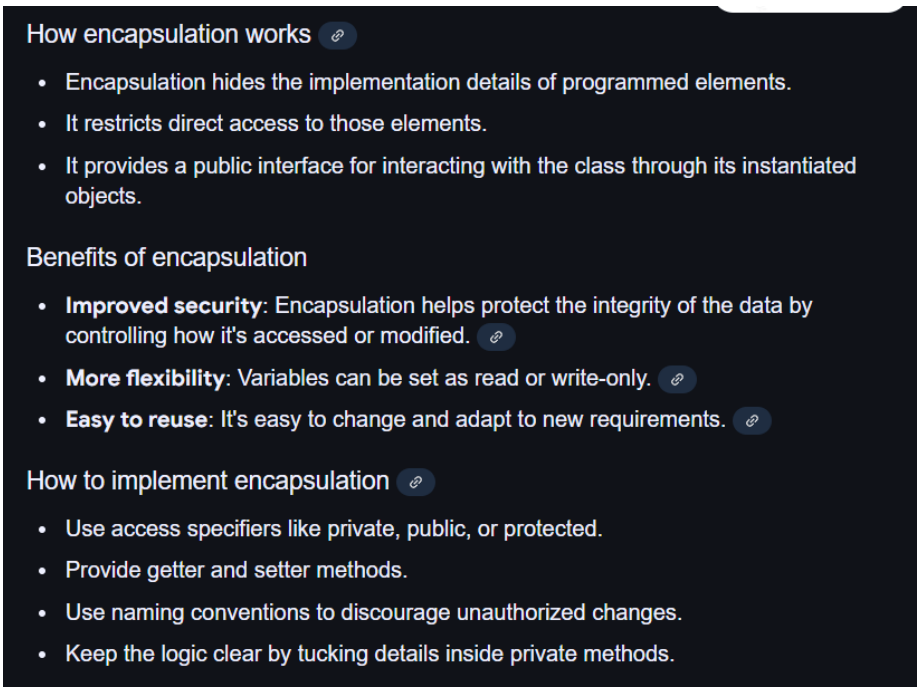
- Class is like a **blueprint** (Plan) for an object.
- It defines,
    - Attributes: Variables that store object data.
    - Properties: Characteristics accessed via getters/setters.
    - Methods: Actions an object can perform.
    - Functions: Reusable code blocks (same as methods in Java).

Employee	*Class Name*
EmpNo **Name** **Address** **BasicSalary** **OtHrs** **OtRate**	*Attributes* *Properties*
CalculateOTAmount() **CalculateNetSalary()** **PrintPaySlip**	*Methods*

```
// Employee class
class Employee {
  // Attributes
  int empNo;
  String name;
  String address;
  double basicSalary;
  double otHrs;
  double otRate;

  // Constructor
  Employee(int empNo, String name, String address, double basicSalary, double otHrs, double otRate) {
    this.empNo = empNo;
    this.name = name;
    this.address = address;
    this.basicSalary = basicSalary;
    this.otHrs = otHrs;
    this.otRate = otRate;
  }

  // Method to calculate OT amount
  double calculateOTAmount() {
    return otHrs * otRate;
  }

  // Method to calculate net salary
  double calculateNetSalary() {
    return basicSalary + calculateOTAmount();
  }

  // Method to print pay slip
  void printPaySlip() {
    System.out.println("Employee Pay Slip");
    System.out.println("-----------------");
    System.out.println("Emp No   : " + empNo);
    System.out.println("Name     : " + name);
    System.out.println("Address  : " + address);
    System.out.println("Basic Salary: $" + basicSalary);
    System.out.println("OT Amount   : $" + calculateOTAmount());
    System.out.println("Net Salary  : $" + calculateNetSalary());
  }
}

// Main class
public class EmployeeDemo {
  public static void main(String[] args) {
    // Creating an Employee object
    Employee emp1 = new Employee(101, "John Doe", "123 Street, City", 50000, 10, 200);

    // Printing the pay slip
    emp1.printPaySlip();
  }
}
```

Encapsulation

- The practice of bundling data and methods into a single unit.
- Help improve security and simplify data hiding.



How encapsulation works

- Encapsulation hides the implementation details of programmed elements.
- It restricts direct access to those elements.
- It provides a public interface for interacting with the class through its instantiated objects.

Benefits of encapsulation

- **Improved security**: Encapsulation helps protect the integrity of the data by controlling how it's accessed or modified.
- **More flexibility**: Variables can be set as read or write-only.
- **Easy to reuse**: It's easy to change and adapt to new requirements.

How to implement encapsulation

- Use access specifiers like private, public, or protected.
- Provide getter and setter methods.
- Use naming conventions to discourage unauthorized changes.
- Keep the logic clear by tucking details inside private methods.

Public And Private Access

- "private" access means a class member (variable or method) can only be accessed from within the class itself, while "public" access allows any code outside the class to access the member directly.

```
class BankAccount:
    def __init__(self, balance):
        self._balance = balance  # Private attribute, accessible only within the class

    def deposit(self, amount):  # Public method, can be called from outside
        self._balance += amount

    def withdraw(self, amount):  # Public method
        if amount <= self._balance:
            self._balance -= amount
            return True
        else:
            return False

    def get_balance(self):  # Public method to access the balance
        return self._balance

# Usage
account = BankAccount(100)
account.deposit(50)
print(account.get_balance())  # Output: 150

# Cannot directly access private attribute
# print(account._balance)  # This would raise an error in most languages
```