
# Class Assignment #01

## Object Oriented Programming

Instructor: Mr. Mohsin Ahmed

| | |
|---|---|
| **Student Name:** | Syeda Aima Abbas |
| **Registration No:** | FA24-BSE-010 |
| **Department:** | Software engineering |
| **Submission Date:** | 18th oct 2025 |

COMSATS University Islamabad

## Question # 01:

**Class:**

A class is a blueprint used to create objects. It defines attributes and behaviors common to all objects of that type.

*Example:*

```
class Car {
    String color;
    void drive() {
        System.out.println("Car is driving");
    }
}
```

**Object:**

An object is an instance of a class that contains its own data and behavior.

*Example:*

```
Car myCar = new Car();
myCar.color = "Red";
myCar.drive();
```

**Data Members:**

These are variables declared within a class that hold the object's data.

*Example:*

```
class Student {
    String name;
    int age;
}
```

**Member Functions:**

Also called methods, these are functions defined inside a class that perform certain actions using class data.

*Example:*

```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }
}
```

## Question # 02:

**What are access specifiers in Java? List all four and explain how each of them affects visibility of class members.**

Access specifiers determine how members (variables, methods, and classes) can be accessed from other parts of the program.

| Access Specifier | Visibility | Description |
|---|---|---|
| **public** | Everywhere | Accessible from any package or class. |
| **protected** | Same package + subclasses | Restricted from non-subclass access outside the package. |
| **default (no modifier)** | Within same package | Accessible by all classes in the same package only. |
| **private** | Within the same class | Not accessible outside the class. |

## Question # 03:

**Define encapsulation and write a short example of a Java class that demonstrates encapsulation.**

Encapsulation is the concept of binding data and methods together while restricting direct access to some details of an object. It protects an object's internal state by controlling how data is accessed or modified.

```
class Employee {
    private double salary;

    public void setSalary(double s) {
        if (s > 0) {
            this.salary = s;
        }
    }

    public double getSalary() {
        return this.salary;
    }
}
```

This ensures outside code cannot modify salary directly, preserving object integrity.

## Question # 04:

**Differentiate between data hiding and data abstraction.**

| Aspect | Data Hiding | Data Abstraction |
|---|---|---|
| **Definition** | Restricting access to object details to protect data. | Showing only essential features and hiding implementation details. |
| **Purpose** | Security and integrity of data. | Simplification of complex systems. |
| **Implementation** | Achieved using access modifiers like private. | Achieved using abstract classes and interfaces. |
| **Example** | Making class fields private. | Providing an abstract method that must be implemented. |

## Question # 05:

**Scenario:**
A class bankAccount has attributes accountNumber, balance, and accountHolderName. The balance should not be accessible directly but can be modified through specific functions.

**Question 1:** Which OOP principle is being applied here?
**Answer:** The principle is **Encapsulation** since data is hidden, and access is controlled through methods.

**Question 2:** Which access specifier should be used for each attribute?
**Answer:**

- accountNumber → private (Sensitive detail)
- balance → private (To protect from direct modification)
- accountHolderName → private or protected (depending on subclass access requirement)

**Question 3:** Why is this principle important in real-world programming?
**Answer:** Encapsulation enables data protection, prevents unintended interference, improves code maintainability, and ensures consistent object states.

*Example Implementation:*

```
class BankAccount {
```

```
    private String accountNumber;
    private double balance;
    private String accountHolderName;

    public void deposit(double amount) {
        if (amount > 0) this.balance += amount;
    }

    public void withdraw(double amount) {
        if (amount <= this.balance) this.balance -= amount;
    }

    public double getBalance() {
        return this.balance;
    }

    public void display() {
        System.out.println("Account Number: " + this.accountNumber + ", Account Holder
Name: " + this.accountHolderName + ", Balance: " + this.balance);
    }
}
```