# Object Oriented Programing Lab
# Lab Assignment 1
# Digital Art Gallery Management
# Date 19-09-2024
# Maximum Marks: 10

**CLO 4**

## Objective

The objective of this lab assignment is to implement a simple digital art gallery management system using Java. This task will help you understand and apply the concepts of:

- **Composition:** Building complex objects from simpler ones.
- **Constructor Overloading:** Providing flexibility in object creation.
- **Encapsulation:** Protecting data integrity and enhancing code maintainability.
- **Deep vs. Shallow Copy:** Understanding the implications of object copying.
- **Version Control:** Using Git and GitHub for collaboration and code management.

## Scenario

Imagine you are developing a system for a digital art gallery. This system needs to manage information about artists and their artworks. Each artwork is essentially linked to its creator. Your task is to design and implement the core classes for this system, ensuring they follow to object-oriented principles.

## Task Requirements

### 1. Implement the Classes

#### Artist Class

- **Fields:**
  - `name` (String): The artist's full name.
- **Methods:**
  - **Constructor:** Initialize the artist's name.
  - **Getter:** Retrieve the artist's name.
  - **Display Information:** Print the artist's name in a user-friendly format.

#### Artwork Class

- **Fields:**
  - `title` (String): The title of the artwork.

    - `year` (int): The year the artwork was created.
    - `artist` (Artist): The artist who created the artwork.
- **Constructor Overloading:**
    - **Constructor 1:** Takes the title, year, and an Artist object as parameters.
    - **Constructor 2:** Takes only the title and year, assigning a default "Unknown Artist" to the `artist` field.
- **Methods:**
    - **Getters:** Retrieve the title, year, and artist of the artwork.
    - **Display Information:** Print the artwork's details (title, year, artist name) in a user-friendly format.
    - **Shallow Copy:** Create a new `Artwork` object that shares the same `Artist` object as the original.
    - **Deep Copy:** Create a new `Artwork` object with a completely independent copy of the `Artist` object.

**2. Demonstrate the Concepts**

- **Main Class:** Create a `Main` class to demonstrate the functionality of your `Artist` and `Artwork` classes.
- **Object Creation:**

    - Instantiate **one** `Artist` object.
    - Instantiate **two** `Artwork` objects:
        - One using the constructor that takes the title, year, and the `Artist` object you created.
        - Another using the constructor that takes only the title and year (the default artist will be assigned).

- **Encapsulation:**

    - Set the title and year of the second artwork (the one with the default artist) using the appropriate setter methods.
    - Print the details of both artworks to show that encapsulation allows controlled modification.

- **Copy Operations:**

    - Create a shallow copy and a deep copy of the first artwork.
    - Print the details of the original artwork, the shallow copy, and the deep copy.
    - Change the artist's name in the original artwork.
    - Print the details of the original artwork, the shallow copy, and the deep copy again. This will demonstrate how deep and shallow copies behave differently when a shared object is modified.

**3. Compile and Run**

- **Notepad and Command Prompt:** Use Notepad for writing your Java code and Command Prompt for compilation and execution.

**4. Push to GitHub**

- **GitHub Repository:** Create a new public repository on GitHub named "Lab-Assignment-1".
- **Version Control:**
  - Initialize a local Git repository: `git init`
  - Stage your changes: `git add .`
  - Commit your changes with a descriptive message: `git commit -m "Completed Lab Assignment 1"`
  - Link your local repository to the remote repository on GitHub (follow GitHub's instructions).
  - Push your changes to GitHub: `git push origin main`

**5. Submission**

- **GitHub Link:** Submit the link to your GitHub repository through the designated platform (e.g., learning management system).
- **Code Comments:** Ensure your code is well-commented, explaining the purpose of classes, methods, and key logic.

## Evaluation Criteria

- **Functionality:** Correct implementation of classes, methods, constructors, and copy operations.
- **OOP Principles:** Effective use of composition, encapsulation, and constructor overloading.
- **Code Style:** Readability, indentation, meaningful variable names, and comments.
- **Version Control:** Proper use of Git and GitHub, including a clear commit history.