# Institute of Geographical Information Systems

## CS-212 - Object Oriented Programming LAB

| | |
|---|---|
| **Semester: Fall 2025** | **Class: SCEE-IGIS - 2024** |
| **Name: Ali Nawaz** | **CMS ID : 00000526123** |
| **Submitted to: Ma'am Alvina Anjum** | **Due Date: Nov 19, 2025** |

# LAB 10: Constructor Overloading

### Task # 1: Basic Constructor Overloading

A construction company GeoBuild Pvt. Ltd. designs floor plans for small rooms and office cabins.

- Each room is modeled as a rectangle, and engineers often create new rooms by:
- Using standard default dimensions
- Entering custom dimensions
- Copying dimensions from an existing room template

However, the company has strict rules:

- No room can have non-positive dimensions.
- A copied room must have a valid area; if not, a default template is used instead.

You are hired to create a Rectangle Room Management System that helps engineers create valid room objects using constructor overloading.

**Question: Why Do We Need Constructor Overloading?**
**Answer:** We need constructor overloading because:

- Different objects need different types of starting information.
- Some objects need default values.
- Some need custom values.
- Some need to be copied from already existing objects.
- It makes object creation flexible and easier.
- It helps us write clean, readable, and organized code.

In real life, not every user, room, wallet, or staff member gives all details at the same time and constructor overloading solves this problem.

## Lab Task 1

Create a class Rectangle with the following private attributes:

- `double length`
- `double width`

## Constructor Requirements

## 1. Default Constructor

- Sets `length = 1` and `width = 1`
- Prints: `"Default constructor: Creating a standard 1x1 room."`

## 2. Parameterized Constructor

- Takes `double l, double w`
- **Validation rule**:
  If either `l ≤ 0` or `w ≤ 0`, set both to **1** and print:
  `"Invalid dimensions! Using default room size 1x1."`
- Otherwise, assign the values
- Print: `"Custom room created using parameterized constructor."`

## 3. Copy Constructor

- Takes another Rectangle object
- If the source object's **area = 0**, do not copy; instead set dimensions to 1 and print:
  `"Cannot copy a room with zero area. Using default dimensions."`
- Otherwise copy length and width
- Print: `"Copy constructor: Room copied successfully.”`

## Member Functions

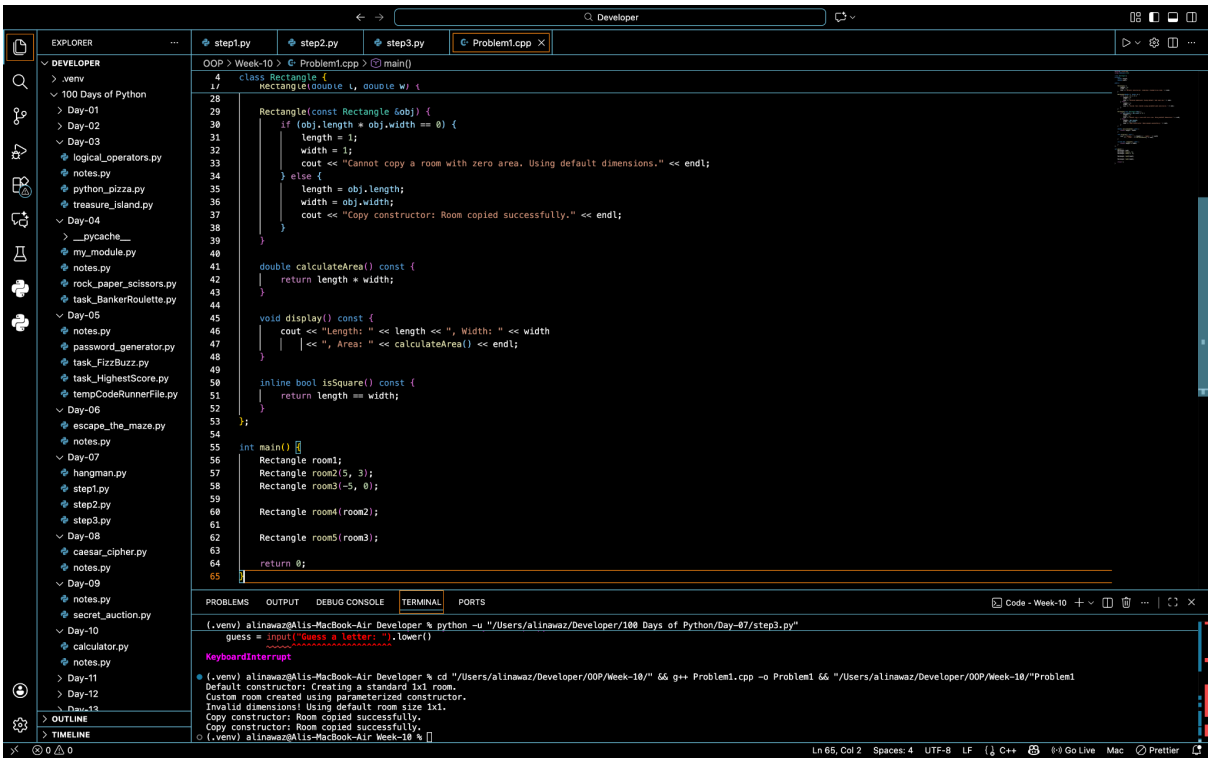
1. `double calculateArea() const`
2. `void display() const`
3. `bool isSquare() const` (inline function)

### Scenario-Based Requirements

In the `main()` function, simulate real room design steps used by engineers at GeoBuild Pvt. Ltd.:

1. Create a standard room using the default constructor.
2. Create a custom office cabin using valid dimensions (e.g., 5 × 3).
3. Attempt to create an invalid meeting room using negative or zero values.
4. Create a copied room template based on the office cabin.
5. Create another room by copying the invalid room (should use default values due to zero area).

**Screenshot:**




**Output:**


## Task 2: Real-life Scenario – Bank Account Initialization

A digital payments company named **SwiftPay** is developing a lightweight **E-Wallet Management System** for users who want to maintain their electronic wallets for daily transactions such as food delivery, shopping, and ride-hailing.

Every SwiftPay user has:

- A **wallet ID**,
- A **user name**,
- A **wallet balance**.

When users sign up, they may register with or without an initial balance. Some users' wallet information may be pre-loaded from promotional campaigns.

Your job is to create the **EWallet** class using **constructor overloading** to support all these onboarding variations.

**Task Description**

**Class: EWallet**

Private data members:

- `string walletID`
- `string userName`
- `double balance`

## Constructor Requirements

## 1. Default Constructor

- Sets `balance = 0`
- Prints:
  `"Default wallet created with zero balance."`

## 2. Parameterized Constructor (Name + Wallet ID)

- Initializes `userName` and `walletID`
- Sets `balance = 0`
- Prints:
  `"Wallet created for user <name> with no initial balance."`

## 3. Parameterized Constructor (Name + Wallet ID + Initial Balance)

- Initializes all attributes
- **If initial balance < 0** → Set balance = 0 and print warning:
  `"Invalid initial amount! Balance set to 0."`
- Otherwise print:
  `"Wallet created with initial balance for user <name>."`

## Member Functions

### 1. `void deposit(double amount)`

- Adds amount to balance
- Reject negative or zero deposits
- Print appropriate success or error message

### 2. `void withdraw(double amount)`

- Allows withdrawal only if sufficient balance
- Prints:
  - `"Withdrawal successful!"`
  - OR `"Insufficient balance!"`

### 3. `void showDetails() const`

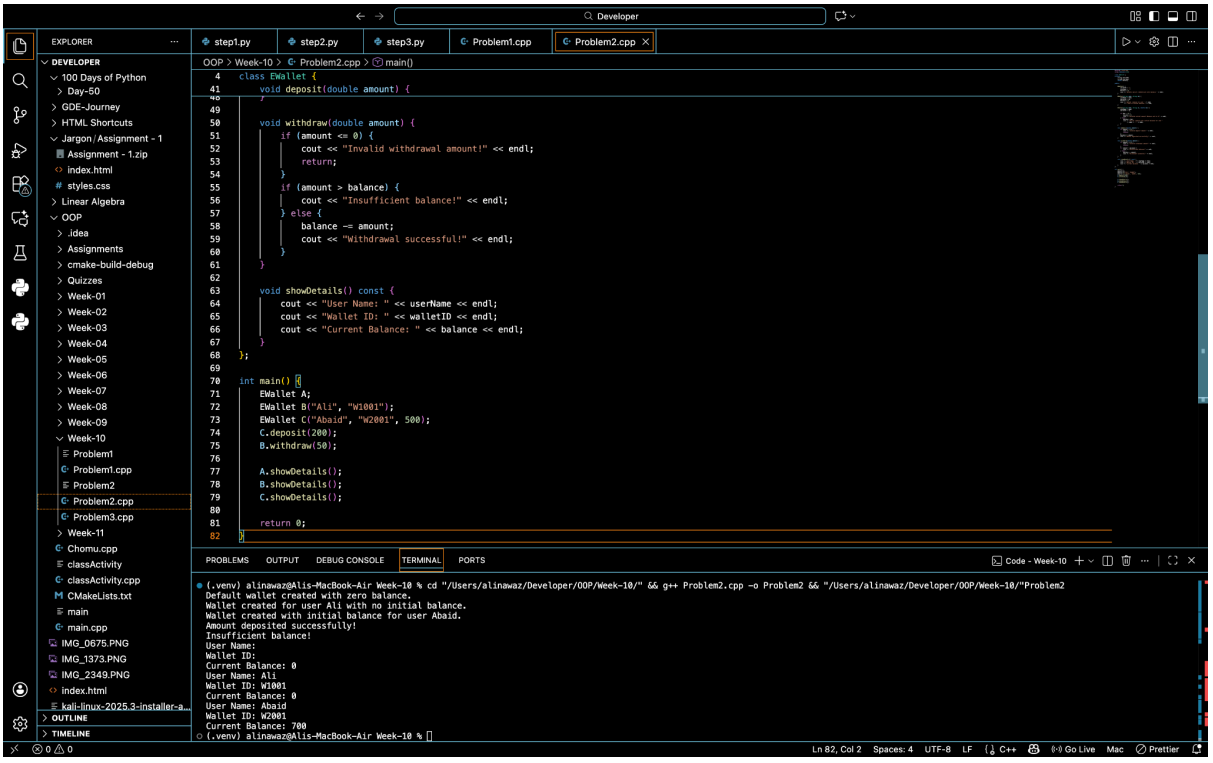
Prints:

```
User Name: _______
Wallet ID: _______
Current Balance: _______
```

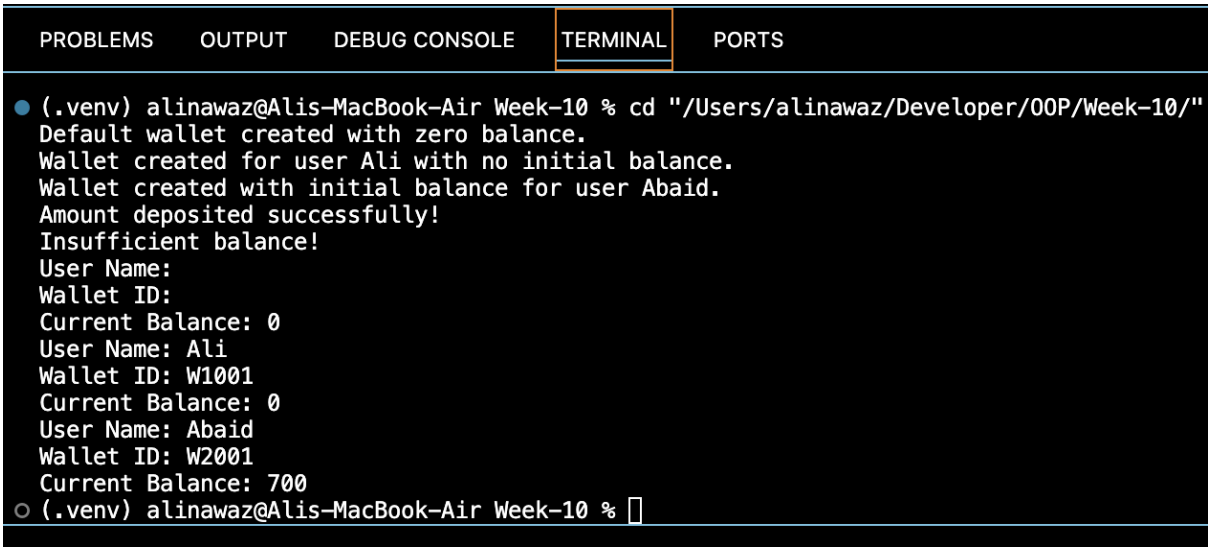
Simulate following onboarding and transactions for SwiftPay:

1. **Create Wallet A using default constructor**
   (represents a user who signed up through the mobile app without entering personal information yet)
2. **Create Wallet B using name + walletID**
   (user has created an ID but added no money yet)
3. **Create Wallet C with complete details**
   (user signed up through partner bank integration with initial funds)
4. **Deposit money into Wallet C**
   (promotional cashback deposit)
5. **Withdraw money from Wallet B**
   (should show insufficient balance)
6. **Display details of all wallets**
   (view for admin monitoring)

**Screenshot:**



**Output:**




## Task 3: Employee Database

A robotics research lab called TechRobo Innovations organizes technical workshops every month. To manage workshop staff members (instructors, lab assistants, and technical support), they maintain a Staff Registration System.

Each staff member has:

- A **name**
- A **unique ID number**
- A **stipend/salary** (depending on their role)

However, staff may register in different ways:

- Some staff are added informally, with minimal details.
- Some staff only give their name initially.
- Others give full details at once.
- Occasionally, staff forget to report their salary and must be assigned one later.

You are required to create the **Staff** class using constructor overloading to support all these registration types.

Private data members:

- `string name`
- `int id`
- `double salary`

## Constructor Requirements

### 1. Default Constructor

Sets:

- `name = "Not set"`
- `id = 0`
- `salary = 0`

Print:

`"Default staff profile created with placeholder values."`

### 2. Constructor with Only Name

Sets:

- `name = givenName`
- `id = 0`
- `salary = 0`

Print:

`"Staff registered with name only."`

### 3. Constructor with Name + ID

Sets:

- `name = givenName`
- `id = givenID`
- `salary = 0`

Print:

`"Staff registered with name and ID."`

### 4. Constructor with Name + ID + Salary

- Validate salary:
    - If salary < 0 → set salary = 0 and print warning:
        - `"Invalid salary entered. Setting salary to 0."`
    - Else set actual salary
- Print:
    - `"Staff registered with full details."`

## Member Functions

### 1. `void setSalary(double amount)`

- Sets salary only if `amount > 0`
- Otherwise print error message:
  - `"Salary not updated. Invalid amount."`
- Else:
  - `"Salary updated successfully."`

### 2. `void display() const`

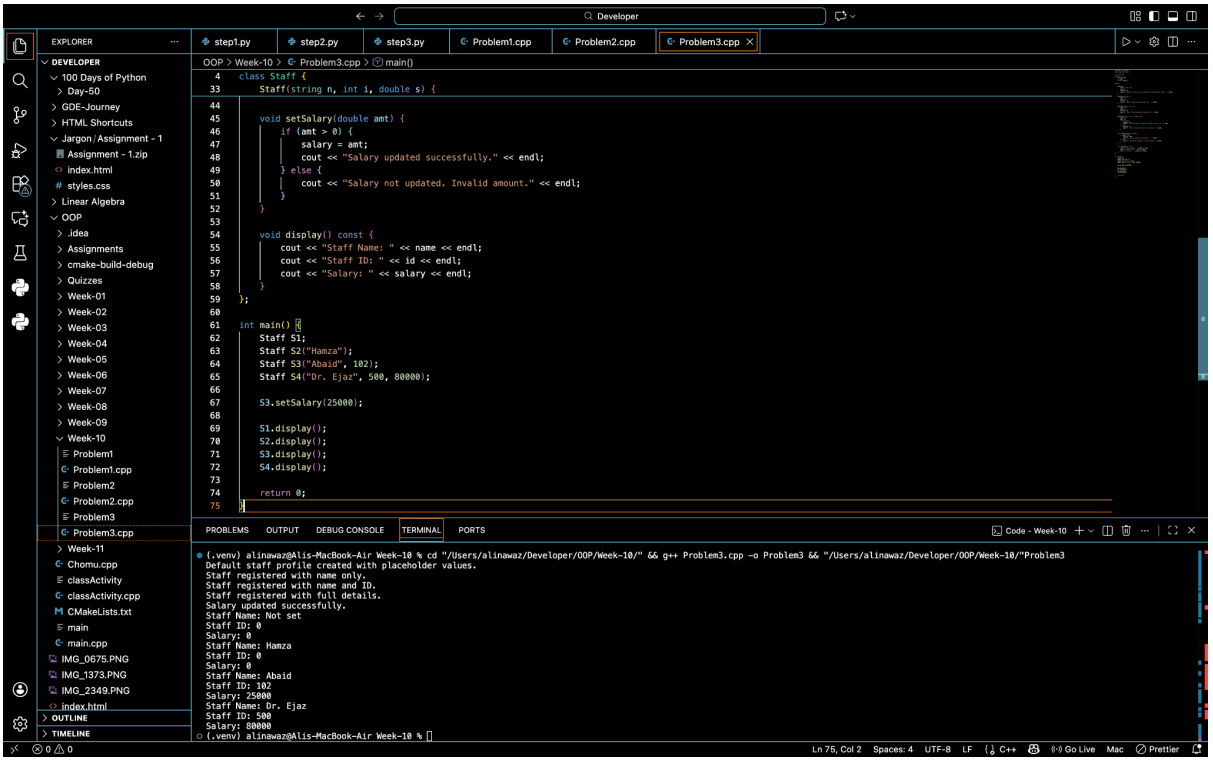
Display the staff profile:

```
Staff Name: _______
Staff ID: _______
Salary: _______
```

Simulate how TechRobo Innovations registers workshop staff:

1. **Create a default staff profile**
   (Used temporarily for unassigned roles)
2. **Register a staff member with name only**
   (A volunteer trainer arrives unexpectedly)
3. **Register another staff member with name and ID**
   (A lab assistant signs up formally but salary is decided later)
4. **Register a senior instructor with all details**
   (Expert invited from industry with predefined compensation)
5. **Update the salary** of the staff member who registered without salary
6. **Display all staff profiles** for workshop management

## Screenshot:




## Output: