//Method Overloading

//Method Overriding

Problem Statement 1

Design a small object-oriented system for a **library** that allows you to:

* Add different types of books (e.g., Fiction, NonFiction) to the library
* Each book has: title, author, and a borrow() method
* Fiction books have an extra attribute genre, and NonFiction books have subject
* Override the borrow() method in each subclass to print a specific message like:  
  Borrowing Fiction: The Alchemist by Paulo Coelho  
  Borrowing NonFiction: A Brief History of Time by Stephen Hawking

SQL problem statement 1:

**Tables:**

sqlCopyEditCustomers(customer\_id, name)  
Orders(order\_id, customer\_id, order\_date, amount)

**Problem Statement:**

Write a SQL query to fetch the **top 3 customers** who spent the **most money** in total.  
Output should include:

* customer\_id
* name
* total\_spent

Java Problem statement 2:

Build a system with:

* A base class Employee with attributes name, salary
* Subclasses: Manager, Developer
* Each subclass should have a calculateBonus() method:
  + Manager: 20% of salary
  + Developer: 10% of salary

SQL statement 2:

Write a query to find the **most recent order date** placed by **each customer**. Show: customer\_id, name, and last\_order\_date.

Java Problem statement 3:

Design a banking system with:

* Base class: Account with methods deposit() and withdraw()
* Subclasses: SavingsAccount, CheckingAccount
  + SavingsAccount has a withdrawal limit of 3 transactions
  + CheckingAccount allows overdraft up to -5000

SQL question 3:

You are given a Products table:

**Table: Products**

| **product\_id** | **name** | **price** |
| --- | --- | --- |
| 1 | Pen | 10 |
| 2 | Pencil | 5 |
| 3 | Pen | 10 |
| 4 | Notebook | 20 |
| 5 | Pencil | 5 |
| 6 | Pen | 10 |

Write a query to **delete duplicate products**, keeping only **one row for each unique name**, and retaining the one with the **lowest product\_id**.

Java Problem 4:

Create a system to manage student grades:

* Create a base class Student with attributes: name, rollNo, and a method calculateGrade()
* Subclasses: UndergraduateStudent, PostgraduateStudent
* Each subclass should override calculateGrade() based on marks:
  + UG: A (>=90), B (>=75), C (>=60), D (<60)
  + PG: A (>=85), B (>=70), C (<70)

SQL problem statement 4

From Customers(customer\_id, name) and Orders(order\_id, customer\_id), find customers who have **never placed an order**.

Java problem 5:

* Create an abstract class Shape with a method getArea()
* Subclasses: Circle, Rectangle
* Implement the getArea() method in each
* Write a main() method that creates a list of shapes and prints their areas using polymorphism

**🔹 Table: Employees**

| **emp\_id** | **name** | **salary** |
| --- | --- | --- |
| 1 | Alice | 45000 |
| 2 | Bob | 85000 |
| 3 | Charlie | 120000 |
| 4 | David | 30000 |
| 5 | Eve | 105000 |

**❓ Question:**

Write a SQL query that creates a new column salary\_bracket with the following logic:

* Salary < 50,000 → **Low**
* Salary between 50,000–100,000 → **Medium**
* Salary > 100,000 → **High**

Java problem 6:

* Create an interface Payment with a method pay(double amount)
* Implement classes: CreditCard, UPI, Cash
* Each class should override pay() to print how the payment was made

**Table: Sales**

| **sale\_id** | **sale\_date** | **amount** |
| --- | --- | --- |
| 1 | 2024-01-01 | 100 |
| 2 | 2024-01-02 | 200 |
| 3 | 2024-01-03 | 150 |
| 4 | 2024-01-04 | 50 |

**❓ Question:**

Write a query to display a **running total** of sales amounts ordered by sale\_date.

**Felix, Arockia (Cognizant)**

Table 1: Customers customer\_id name 101 John Smith 102 Alice Brown 103 Bob Martin 104 Carol Lee 105 David Foster 🔹 Table 2: Orders order\_id customer\_id 1 101 2 102 3 103 4 104

**Table: Customers**

| **customer\_id** | **name** |
| --- | --- |
| 101 | John Smith |
| 102 | Alice Brown |
| 103 | Bob Martin |
| 104 | Carol Lee |

**Table: Orders**

| **order\_id** | **customer\_id** | **order\_date** | **amount** |
| --- | --- | --- | --- |
| 1 | 101 | 2024-01-10 | 300 |
| 2 | 101 | 2024-01-11 | 500 |
| 3 | 102 | 2024-01-09 | 450 |
| 4 | 102 | 2024-01-15 | 200 |
| 5 | 103 | 2024-01-13 | 150 |
| 6 | 104 | 2024-01-08 | 600 |
| 7 | 104 | 2024-01-20 | 700 |

SQL problem statement 4:

**🔹 Table 1: Customers**

| **customer\_id** | **name** |
| --- | --- |
| 101 | John Smith |
| 102 | Alice Brown |
| 103 | Bob Martin |
| 104 | Carol Lee |
| 105 | David Foster |

**🔹 Table 2: Orders**

| **order\_id** | **customer\_id** |
| --- | --- |
| 1 | 101 |
| 2 | 102 |
| 3 | 103 |
| 4 | 104 |

Use a List<Book> to store the books and iterate over them to call borrow()

Answer for Problem Statement 4: // Abstract base class

abstract class Student {

String name;

int rollNo;

double marks; // Changed to double

public Student(String name, int rollNo, double marks) {

this.name = name;

this.rollNo = rollNo;

this.marks = marks;

}

public abstract String calculateGrade();

}

// Undergraduate subclass

class UndergraduateStudent extends Student {

public UndergraduateStudent(String name, int rollNo, double marks) {

super(name, rollNo, marks);

}

@Override

public String calculateGrade() {

if (marks >= 90.0) return "A";

else if (marks >= 75.0) return "B";

else if (marks >= 60.0) return "C";

else return "D";

}

}

// Postgraduate subclass

class PostgraduateStudent extends Student {

public PostgraduateStudent(String name, int rollNo, double marks) {

super(name, rollNo, marks);

}

@Override

public String calculateGrade() {

if (marks >= 85.0) return "A";

else if (marks >= 70.0) return "B";

else return "C";

}

}

// Main class to test the system

public class Main {

public static void main(String[] args) {

Student ugStudent = new UndergraduateStudent("Felix", 101, 99.5);

Student pgStudent = new PostgraduateStudent("Karthikeyan", 201, 82.0);

System.out.println(ugStudent.name + " (UG) Grade: " + ugStudent.calculateGrade());

System.out.println(pgStudent.name + " (PG) Grade: " + pgStudent.calculateGrade());

}

}

Sql query:

mysql> use orders\_db;

Database changed

mysql> create table Customers( customer\_id INT PRIMARY KEY, name VARCHAR(100) );

Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TABLE Orders (order\_id INT PRIMARY KEY, customer\_id INT, FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id));

Query OK, 0 rows affected (0.04 sec)

mysql> INSERT INTO Customers (customer\_id, name) VALUES (101, 'John Smith'), (102, 'Alice Brown'), (103, 'Bob Martin'), (104, 'Carol Lee'), (105, 'David Foster');

Query OK, 5 rows affected (0.01 sec)

Records: 5 Duplicates: 0 Warnings: 0

mysql> INSERT INTO Orders (order\_id, customer\_id) VALUES (1, 101), (2, 102), (3, 103), (4, 104);

Query OK, 4 rows affected (0.01 sec)

Records: 4 Duplicates: 0 Warnings: 0

mysql> SELECT c.customer\_id, c.name FROM Customers c LEFT JOIN Orders o ON c.customer\_id = o.customer\_id WHERE o.customer\_id IS NULL;

+-------------+--------------+

| customer\_id | name |

+-------------+--------------+

| 105 | David Foster |

+-------------+--------------+

1 row in set (0.00 sec)