Texas College of Management & IT

Bachelor of Information Technology (BIT)

Project Title: Fun in Programming with Java and OOP

Submitted By:

Name: Anuraj Gautam

LCID: LC00017003334

Semester: 2nd Semester, 1st year

Submitted To:

Instructor Name: Rajan Yadav

Course: Java Programming

Date of Submission :2025/07/19

Texas College of Management &IT, Kathmandu

Table of Contents

Introduction:

Briefly introduce the important of Fullter for mobile development and java for backend logic. Mention the objective of combining both for cross-platform app development and logic modeling. Flutter is powerful open-source Ul toolkit by Google used to build natively mobile, web,and disktop applications from single codebase.It enables fast development with expressive and flexible UI designs. On ths other hand java is a backend programming language known for its portability and object-oriented capabilities.

Objects:

- To improves foundation of java programming .
- Help in buliding real world project.
- To represent real life entities in code like Car, Student,Employee.
- To Allow easy reuse of code through object instances
- Make programs easier to understand and maintain.

Task 1: OOP implementation -Banking Systems

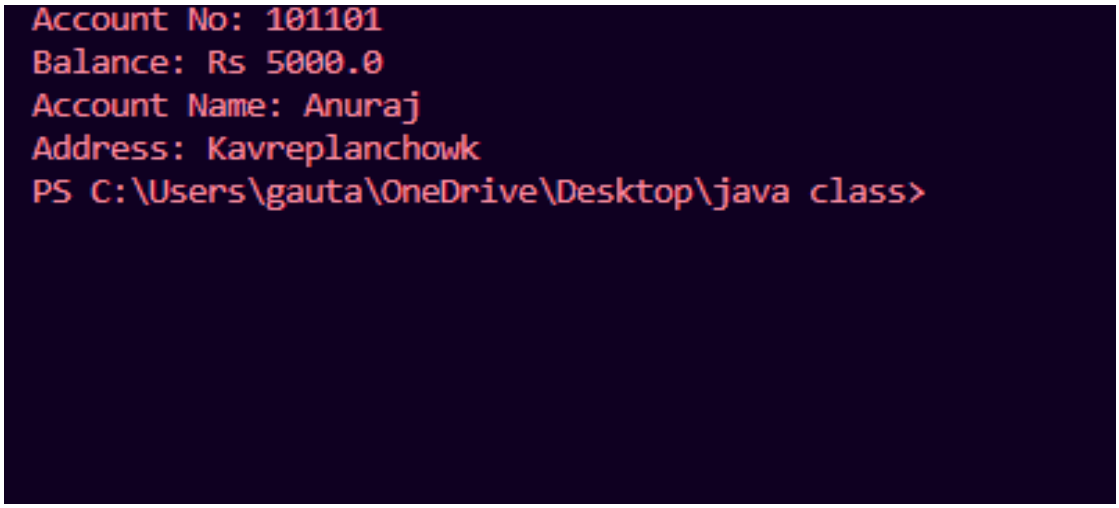
Bank.java

C: > Users > gauta > OneDrive > Desktop > java class > Bank.java

```java
/* Banking System */

class Customer {
    private String account_name;
    private String address;

    public Customer(String account_name, String address) {
        this.account_name = account_name;
        this.address = address;
    }

    public String getAccount_name() {
        return account_name;
    }

    public String getAddress() {
        return address;
    }


    public void displayCustomerDetails() {
        System.out.println("Account Name: " + account_name);
        System.out.println("Address: " + address);
    }
}

class Account {
    private int account_number;
    private double balance;
```

```java
    public Account(int account_number, double balance) {
        this.account_number = account_number;
        this.balance = balance;
    }

    public void displayAccount() {
        System.out.println("Account No: " + account_number);
        System.out.println("Balance: Rs " + balance);
    }

    @Override
    protected void finalize(){
        System.out.println("Account object is being deleted");

    }
}

public class Bank {
    public static void main(String[] args) {
        Customer c1 = new Customer("Anuraj", "Kavreplanchowk");
        Account a1 = new Account(101101, 5000);

        a1.displayAccount();
        c1.displayCustomerDetails();
    }
}
```

Output:

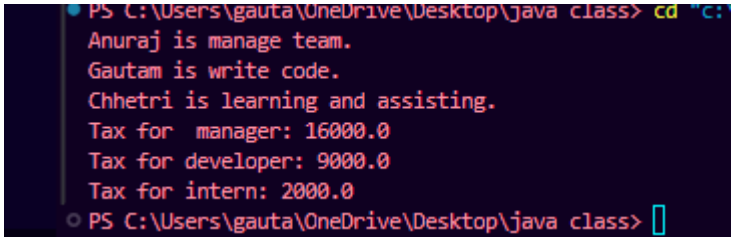

Account No: 101101
Balance: Rs 5000.0
Account Name: Anuraj
Address: Kavreplanchowk
PS C:\Users\gauta\OneDrive\Desktop\java class>

Task 2: inheritance & Polymorphism -Employee Management

```java
/* Emloyee management system*/
interface TaxPayer {
    double calculateTax();
}

abstract class Employee {
    String name;
    double salary;

    public Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    abstract void work();

    void showDetails() {
        System.out.println("Name: " + name + ", Salary: " + salary);
    }
}

class Manager extends Employee implements TaxPayer {
    public Manager(String name, double salary) {
        super(name, salary);
    }

    @Override
    void work() {
        System.out.println(name + " is manage team.");
    }

    public double calculateTax() {
        return salary * 0.2;
    }
}

class Developer extends Employee implements TaxPayer {
    public Developer(String name, double salary) {
        super(name, salary);
    }

    @Override
    void work() {
        System.out.println(name + " is write code.");
    }
```

```java
    public double calculateTax() {
        return salary * 0.15;
    }
}

class Intern extends Employee {
    public Intern(String name, double salary) {
        super(name, salary);
    }

    @Override
    void work() {
        System.out.println(name + " is learning and assisting.");
    }
     public double calculateTax() {
        return salary * 0.1;
    }
}

public class Employes{
    Run | Debug
    public static void main(String[] args) {
        Employee emp1 = new Manager(name:"Anuraj", salary:80000);
        Employee emp2 = new Developer(name:"Gautam", salary:60000);
        Employee emp3 = new Intern(name:"Chhetri", salary:20000);

        emp1.work();
        emp2.work();
        emp3.work();


        if (emp1 instanceof Manager ) {
            Manager  M= ( Manager) emp1;
            System.out.println("Tax for  manager: " + M.calculateTax());
        }
        if (emp2 instanceof Developer) {
            Developer  D= (Developer) emp2;
            System.out.println("Tax for developer: " + D.calculateTax());
        }
        if (emp3 instanceof Intern) {
            Intern  I= (Intern) emp3;
            System.out.println("Tax for intern: " + I.calculateTax());
        }
    }
}
```

Output:

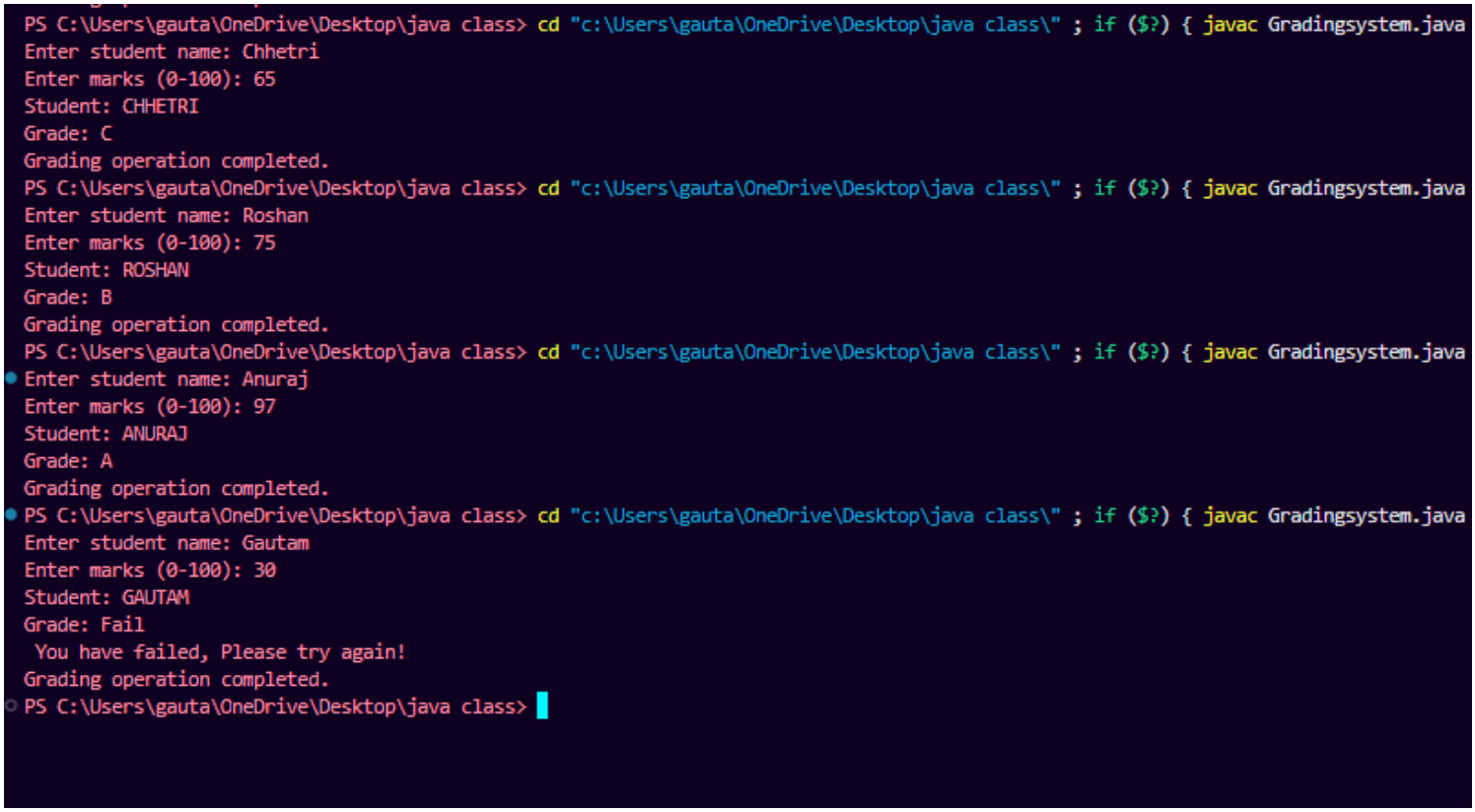


Task 3: String and Exception Handling -Student Grading

Gradingsystem.java

```java
import java.util.Scanner;

class InvalidMarkException extends Exception {
    public InvalidMarkException(String message) {
        super(message);
    }
}

public class Gradingsystem {
    Run | Debug
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        try {
            System.out.print(s:"Enter student name: ");
            String name = sc.nextLine();

            System.out.print(s:"Enter marks (0-100): ");
            String markStr = sc.nextLine();

            int marks = Integer.parseInt(markStr);
            if (marks < 0 || marks > 100) {
                throw new InvalidMarkException(message:"Marks must be between 0 and 100.");
            }

            String grade = getGrade(marks);
            System.out.println("Student: " + name.toUpperCase());
            System.out.println("Grade: " + grade);

            if (grade.equals(anObject:"Fail")) {
                System.out.println(x:" You have failed, Please try again!");
            }

        } catch (InvalidMarkException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (NumberFormatException e) {
            System.out.println(x:"Invalid input! Please enter numeric marks.");
        } finally {
            System.out.println(x:"Grading operation completed.");
            sc.close();
        }
    }

    static String getGrade(int marks) {
        if (marks >= 90) return "A";
        else if (marks >= 75) return "B";
        else if (marks >= 60) return "C";
        else if (marks >= 40) return "D";
        else return "Fail";
    }
}

```

Output:

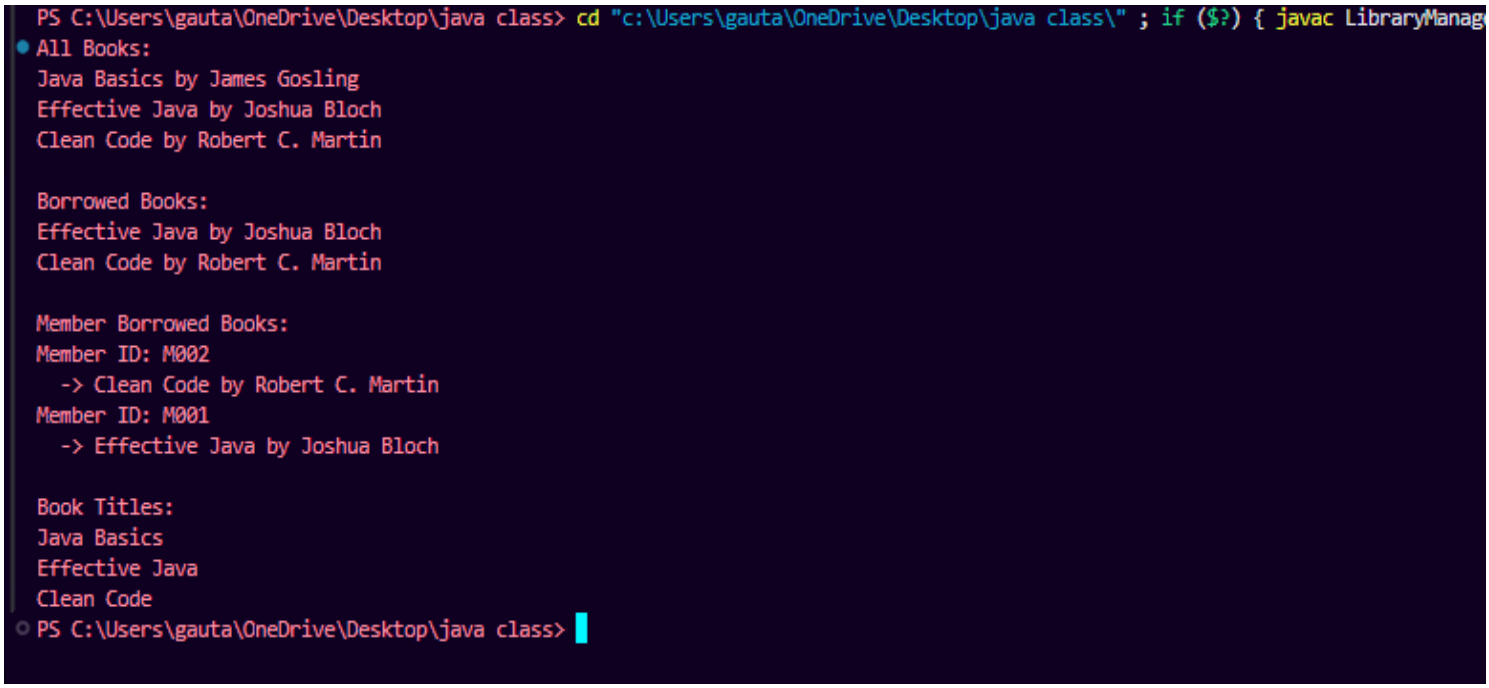


Task 4: Java Collections -Library Management System

LibraryManagementSystem.java

```java
import java.util.*;


class Book {
    String title;
    String author;

    public Book(String title, String author) {
        this.title = title;
        this.author = author;
    }

    @Override
    public String toString() {
        return title + " by " + author;
    }
}

public class LibraryManagementSystem {

    Run | Debug
    public static void main(String[] args) {


        List<Book> allBooks = new ArrayList<>();
        allBooks.add(new Book(title:"Java Basics", author:"James Gosling"));
        allBooks.add(new Book(title:"Effective Java", author:"Joshua Bloch"));
        allBooks.add(new Book(title:"Clean Code", author:"Robert C. Martin"));

        Set<Book> borrowedBooks = new HashSet<>();
        borrowedBooks.add(allBooks.get(index:1));
        borrowedBooks.add(allBooks.get(index:2));

        Map<String, List<Book>> memberBooksMap = new HashMap<>();
        memberBooksMap.put(key:"M001", Arrays.asList(allBooks.get(index:1)));
        memberBooksMap.put(key:"M002", Arrays.asList(allBooks.get(index:2)));


        System.out.println(x:"All Books:");
        for (Book book : allBooks) {
            System.out.println(book);
        }


        System.out.println(x:"\nBorrowed Books:");
        Iterator<Book> it = borrowedBooks.iterator();
        while (it.hasNext()) {
            System.out.println(it.next());
        }

        System.out.println(x:"\nMember Borrowed Books:");
        for (Map.Entry<String, List<Book>> entry : memberBooksMap.entrySet()) {
            System.out.println("Member ID: " + entry.getKey());
            for (Book book : entry.getValue()) {
                System.out.println("  -> " + book);
            }
        }
        displayBookTitles(allBooks);
    }

```

Output:


PS C:\Users\gauta\OneDrive\Desktop\java class> cd "c:\Users\gauta\OneDrive\Desktop\java class\" ; if ($?) { javac LibraryManag
All Books:
Java Basics by James Gosling
Effective Java by Joshua Bloch
Clean Code by Robert C. Martin
Borrowed Books:
Effective Java by Joshua Bloch
Clean Code by Robert C. Martin
Member Borrowed Books:
Member ID: M002
-> Clean Code by Robert C. Martin
Member ID: M001
-> Effective Java by Joshua Bloch
Book Titles:
Java Basics
Effective Java
Clean Code
PS C:\Users\gauta\OneDrive\Desktop\java class>

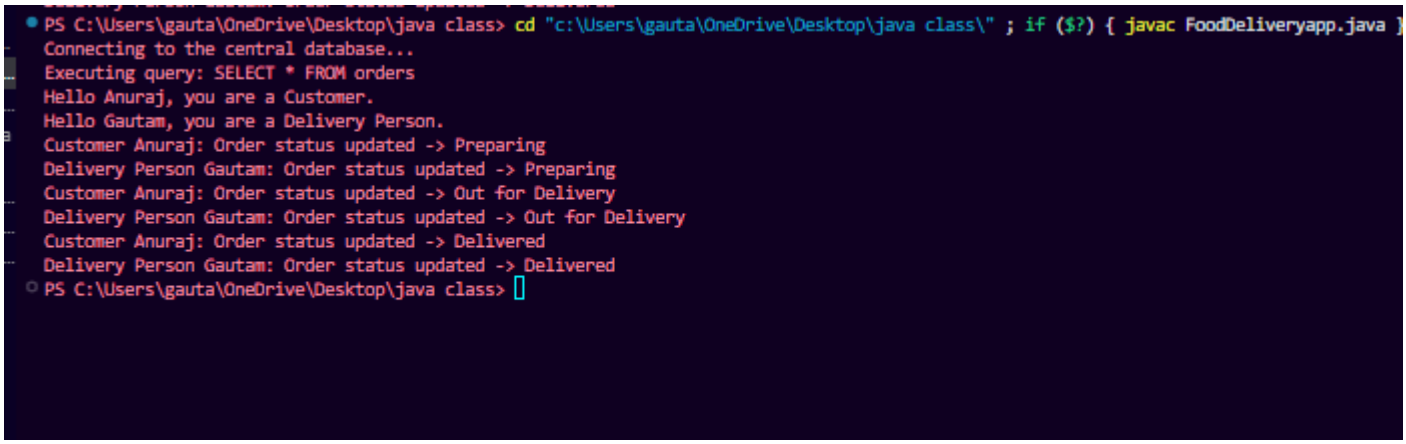
Task 5: Design Patterns – Online food Delivery App

FoodDeliveryapp.java

```java
import java.util.*;
class Database {
    private static Database instance;

    private Database() {
        System.out.println(x:"Connecting to the central database...");
    }

    public static synchronized Database getInstance() {
        if (instance == null) {
            instance = new Database();
        }
        return instance;
    }

        public void query(String sql) {
        System.out.println("Executing query: " + sql);
    }
}


abstract class User {
    protected String name;
    public User(String name) { this.name = name; }
    abstract void showRole();
}

class Customer extends User {
    public Customer(String name) { super(name); }
    void showRole() {
        System.out.println("Hello " + name + ", you are a Customer.");
    }
}

class DeliveryPerson extends User {
    public DeliveryPerson(String name) { super(name); }
    void showRole() {
        System.out.println("Hello " + name + ", you are a Delivery Person.");
    }
}

class Admin extends User {
    public Admin(String name) { super(name); }
    void showRole() {
        System.out.println("Hello " + name + ", you are an Admin.");
    }
}

class UserFactory {
    public static User createUser(String role, String name) {
        switch (role.toLowerCase()) {
            case "customer": return new Customer(name);
            case "delivery": return new DeliveryPerson(name);
            case "admin": return new Admin(name);
            default: throw new IllegalArgumentException(s:"Invalid role!");
        }
    }
```

```java
}


interface Observer {
    void update(String status);
}

class CustomerObserver implements Observer {
    private String name;

    public CustomerObserver(String name) {
        this.name = name;
    }

    public void update(String status) {
        System.out.println("Customer " + name + ": Order status updated -> " + status);
    }
}

class DeliveryObserver implements Observer {
    private String name;

    public DeliveryObserver(String name) {
        this.name = name;
    }

    public void update(String status) {
        System.out.println("Delivery Person " + name + ": Order status updated -> " + status);
    }
}

class OrderStatus {
    private List<Observer> observers = new ArrayList<>();
    private String status;

    public void attach(Observer obs) {
        observers.add(obs);
    }

    public void setStatus(String newStatus) {
        this.status = newStatus;
        notifyAllObservers();
    }

    private void notifyAllObservers() {
        for (Observer obs : observers) {
            obs.update(status);
        }
    }
}

public class FoodDeliveryapp {
    Run | Debug
    public static void main(String[] args) {
```

Output:



Class Diagram

Database

-Instances

-Database ()

+getInstances

+query

Benefits

-Adds flexibility to user role creation.

-Easy to extend (e.g. add Admin, Restaurant Owner)

Limitations

-Requires code changes when new roles are added.

-Hard to manage if many subclasses are needed.

Conclusion

This project highlights the power of java Object-Oriented Programming in modeling real world systems. Each task demonstrates core concepts like encapsulation, inheritance, polymorphism, collection, and design patterns in practical scenarios. These concepts strengthen logical thinking, improve projects structure.

Reference:

This project is based on personal analysis and understanding of real-world platform challenges related to trust. For technical concepts and implementation resources,

-W3School

-Lecture notes and class assignments

-Personal experiment and testing environment.

-YouTube video-codewithmosh, codewithharry Apna College etc.

Declaration of Originality

I hereby declare that this assignment is my original hard work, and I have referenced all sources as required. I understand that failure to adhere to academic integrity will in disciplinary action.

Student Name: Anuraj Gautam

Student Id: LC00017003334

Signature: 

Date :2025/07/19