Week 1 - Basic Banking Application

# Project Overview

The Basic Banking Application is a simple console-based banking system that allows users to create an account, perform operations like deposit, withdrawal, transfer money, check balance, and delete accounts. It also includes features for displaying all the accounts present in the bank.

# Code: Key Sections

Bank.java

import java.util.ArrayList;

import java.util.Scanner;

/\*\*

 \* Bank Management System

 \* Author: MR.POSA

 \*/

public class Bank {

    public static void main(String[] args) {

        Scanner s = new Scanner(System.in);

        System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DATA SOFTIXS Bank\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

        ArrayList<BankAccounts> accounts = new ArrayList<>();

        while (true) {

            System.out.println("Create an account - Press 1");

            System.out.println("Bank operations - Press 2");

            System.out.println("Delete account - Press 3");

            System.out.println("Display all accounts - Press 4");

            System.out.println("Close the bank - Press 5");

            System.out.print("Enter your choice: ");

            int press = s.nextInt();

            switch (press) {

                case 1: {

                    System.out.println();

                    System.out.println("Enter user name: ");

                    s.nextLine(); // Consume the leftover newline

                    String name = s.nextLine();

                    System.out.println("Enter account number: ");

                    int accnum = s.nextInt();

                    System.out.println("Enter initial amount: ");

                    double amount = s.nextDouble();

                    System.out.println("Enter phone number: ");

                    int phone = s.nextInt();

                    accounts.add(new BankAccounts(name, accnum, amount, phone));

                    System.out.println("Account created successfully with the name of " + name);

                    break;

                }

                case 2: {

                    System.out.println("Bank operations menu:");

                    BankOperations bo = new BankOperations();

                    bo.Menu(accounts);

                    break;

                }

                case 3: {

                    System.out.println("Enter the account number or phone number to remove: ");

                    int identifier = s.nextInt();

                    BankAccounts accountToRemove = findAccountByIdentifier(accounts, identifier);

                    if (accountToRemove != null) {

                        accounts.remove(accountToRemove);

                        System.out.println("Account removed successfully.");

                    } else {

                        System.out.println("Account not found.");

                    }

                    break;

                }

                case 4: {

                    displayAccounts(accounts);

                    break;

                }

                case 5: {

                    System.out.println("Closing the bank. Goodbye!");

                    System.exit(1);

                }

                default: {

                    System.out.println("Enter a valid choice.");

                }

            }

        }

    }

    private static BankAccounts findAccountByIdentifier(ArrayList<BankAccounts> accounts, int identifier) {

        for (BankAccounts account : accounts) {

            if (account.getAccnum() == identifier || account.getPhone() == identifier) {

                return account;

            }

        }

        return null;

    }

    private static void displayAccounts(ArrayList<BankAccounts> accounts) {

        if (accounts.isEmpty()) {

            System.out.println("No accounts available to display.");

        } else {

            System.out.println("\nAccount Details:");

            for (BankAccounts account : accounts) {

                System.out.println("Name: " + account.getName() + ", Account Number: " + account.getAccnum()

                        + ", Balance: " + account.getAmount());

            }

        }

    }

}

Bankaccount.java

class BankAccounts {

    private String name;

    private int accnum;

    private double amount;

    private int phone;

    public BankAccounts(String name, int accnum, double amount, int phone) {

        this.name = name;

        this.accnum = accnum;

        this.amount = amount;

        this.phone = phone;

    }

    public String getName() {

        return name;

    }

    public int getAccnum() {

        return accnum;

    }

    public double getAmount() {

        return amount;

    }

    public int getPhone() {

        return phone;

    }

    public void deposit(double amount) {

        if (amount > 0) {

            this.amount += amount;

            System.out.println("Amount deposited successfully. New balance: " + this.amount);

        } else {

            System.out.println("Invalid deposit amount.");

        }

    }

    public void withdraw(double amount) {

        if (amount > 0 && this.amount >= amount) {

            this.amount -= amount;

            System.out.println("Amount withdrawn successfully. Remaining balance: " + this.amount);

        } else {

            System.out.println("Insufficient funds or invalid withdrawal amount.");

        }

    }

    public void transfer(BankAccounts targetAccount, double amount) {

        if (amount > 0 && this.amount >= amount) {

            this.amount -= amount;

            targetAccount.deposit(amount);

            System.out.println("Transferred " + amount + " to account number: " + targetAccount.getAccnum());

        } else {

            System.out.println("Transfer failed due to insufficient balance or invalid amount.");

        }

    }

    public void checkBalance() {

        System.out.println("Account balance for " + name + " is: " + amount);

    }

}

BankOperations.java

class BankAccounts {

    private String name;

    private int accnum;

    private double amount;

    private int phone;

    public BankAccounts(String name, int accnum, double amount, int phone) {

        this.name = name;

        this.accnum = accnum;

        this.amount = amount;

        this.phone = phone;

    }

    public String getName() {

        return name;

    }

    public int getAccnum() {

        return accnum;

    }

    public double getAmount() {

        return amount;

    }

    public int getPhone() {

        return phone;

    }

    public void deposit(double amount) {

        if (amount > 0) {

            this.amount += amount;

            System.out.println("Amount deposited successfully. New balance: " + this.amount);

        } else {

            System.out.println("Invalid deposit amount.");

        }

    }

    public void withdraw(double amount) {

        if (amount > 0 && this.amount >= amount) {

            this.amount -= amount;

            System.out.println("Amount withdrawn successfully. Remaining balance: " + this.amount);

        } else {

            System.out.println("Insufficient funds or invalid withdrawal amount.");

        }

    }

    public void transfer(BankAccounts targetAccount, double amount) {

        if (amount > 0 && this.amount >= amount) {

            this.amount -= amount;

            targetAccount.deposit(amount);

            System.out.println("Transferred " + amount + " to account number: " + targetAccount.getAccnum());

        } else {

            System.out.println("Transfer failed due to insufficient balance or invalid amount.");

        }

    }

    public void checkBalance() {

        System.out.println("Account balance for " + name + " is: " + amount);

    }

}

# Code Explanation

This banking application is divided into multiple classes to handle different operations and account details.

## 1. Bank Class

The main class of the project, which controls the overall flow of the application. It allows the user to create an account, perform bank operations, delete accounts, display accounts, and exit the bank. The class maintains a list of bank accounts and handles user interactions through a menu-driven interface.

## 2. BankAccounts Class

The BankAccounts class stores individual account details like name, account number, balance, and phone number. It includes methods for deposit, withdrawal, transfer, and balance checking. The class also has constructors to initialize account details and getters to retrieve them.

## 3. BankOperations Class

The BankOperations class handles all the banking operations. It allows the user to deposit money into an account, withdraw funds, transfer money between accounts, and check account balances. The class ensures that all operations are performed securely with proper validation.

# Tools and Technologies Used

1. Java: The main programming language used to build the banking application.  
2. ArrayList: Used to store the list of bank accounts.  
3. Scanner: Used for user input.  
4. Java Classes and Methods: Encapsulation and object-oriented principles are utilized to handle account and operations.

# Output Results

PS D:\dataSoftixs projects\week 1 project on bank application> & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\MAHA LAPTOPS\AppData\Roaming\Code\User\workspaceStorage\1c38b25221ec8c390ff19367ebd4b781\redhat.java\jdt\_ws\week 1 project on bank application\_1f650817\bin' 'Bank'

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DATA SOFTIXS Bank\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Create an account - Press 1

Bank operations - Press 2

Delete account - Press 3

Display all accounts - Press 4

Close the bank - Press 5

Enter your choice:

4

No accounts available to display.

Create an account - Press 1

Bank operations - Press 2

Delete account - Press 3

narendra

Enter account number:

101

Enter initial amount:

1000

Enter phone number:

9014293912

Exception in thread "main" java.util.InputMismatchException: For input string: "9014293912"

at java.base/java.util.Scanner.nextInt(Scanner.java:2290)

at java.base/java.util.Scanner.nextInt(Scanner.java:2238)

at Bank.main(Bank.java:34)

PS D:\dataSoftixs projects\week 1 project on bank application>

narendra

Enter account number:

101

Enter initial amount:

1000

Enter phone number:

9014293912

Exception in thread "main" java.util.InputMismatchException: For input string: "9014293912"

at java.base/java.util.Scanner.nextInt(Scanner.java:2290)

at java.base/java.util.Scanner.nextInt(Scanner.java:2238)

at Bank.main(Bank.java:34)

PS D:\dataSoftixs projects\week 1 project on bank application>

narendra

Enter account number:

101

Enter initial amount:

1000

Enter phone number:

9014293912

Exception in thread "main" java.util.InputMismatchException: For input string: "9014293912"

at java.base/java.util.Scanner.nextInt(Scanner.java:2290)

at java.base/java.util.Scanner.nextInt(Scanner.java:2238)

at Bank.main(Bank.java:34)

PS D:\dataSoftixs projects\week 1 project on bank application>

narendra

Enter account number:

101

Enter initial amount:

1000

Enter phone number:

9014293912

Exception in thread "main" java.util.InputMismatchException: For input string: "9014293912"

at java.base/java.util.Scanner.nextInt(Scanner.java:2290)

at java.base/java.util.Scanner.nextInt(Scanner.java:2238)

at Bank.main(Bank.java:34)

narendra

Enter account number:

101

Enter initial amount:

1000

Enter phone number:

narendra

Enter account number:

101

Enter initial amount:

narendra

Enter account number:

narendra

narendra

Enter account number:

narendra

Enter account number:

101

narendra

Enter account number:

narendra

Enter account number:

101

Enter initial amount:

1000

Enter phone number:

9014293912

Exception in thread "main" java.util.InputMismatchException: For input string: "9014293912"

at java.base/java.util.Scanner.nextInt(Scanner.java:2290)

at java.base/java.util.Scanner.nextInt(Scanner.java:2238)

at Bank.main(Bank.java:34)

PS D:\dataSoftixs projects\week 1 project on bank application> ^C

PS D:\dataSoftixs projects\week 1 project on bank application> d:; cd 'd:\dataSoftixs projects\week 1 project on bank application'; & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\MAHA LAPTOPS\AppData\Roaming\Code\User\workspaceStorage\1c38b25221ec8c390ff19367ebd4b781\redhat.java\jdt\_ws\week 1 project on bank application\_1f650817\bin' 'Bank'

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DATA SOFTIXS Bank\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Create an account - Press 1

Bank operations - Press 2

Delete account - Press 3

Display all accounts - Press 4

Close the bank - Press 5

Enter your choice: 1

Enter user name:

nani

Enter account number:

101

Enter initial amount:

100

Enter phone number:

Enter phone number:

9014293912

Account created successfully with the name of nani

Create an account - Press 1

Bank operations - Press 2

Delete account - Press 3

Display all accounts - Press 4

Close the bank - Press 5

Enter your choice: 1

Enter user name:

nann

Enter account number:

102

Enter initial amount:

1000

Enter phone number:

9876543210

Account created successfully with the name of nann

Create an account - Press 1

Bank operations - Press 2

Delete account - Press 3

Display all accounts - Press 4

Close the bank - Press 5

Enter your choice: 4

Account Details:

Name: nani, Account Number: 101, Balance: 100.0

Name: nann, Account Number: 102, Balance: 1000.0

Create an account - Press 1

Bank operations - Press 2

Delete account - Press 3

Display all accounts - Press 4

Close the bank - Press 5

Enter your choice: 2

Bank operations menu:

1. Deposit

2. Withdraw

3. Transfer

4. Check Balance

5. Back to Main Menu

Enter your choice: 1

Enter account number or phone number to deposit into:

101

Enter deposit amount:

2000

Amount deposited successfully. New balance: 2100.0

1. Deposit

2. Withdraw

3. Transfer

4. Check Balance

5. Back to Main Menu

Enter your choice: 4

Enter account number or phone number to check balance:

101

Account balance for nani is: 2100.0

1. Deposit

2. Withdraw

3. Transfer

4. Check Balance

5. Back to Main Menu

Enter your choice: 2

Enter account number or phone number to withdraw from:

101

Enter withdrawal amount:

100

Amount withdrawn successfully. Remaining balance: 2000.0

1. Deposit

2. Withdraw

3. Transfer

4. Check Balance

5. Back to Main Menu

Enter your choice: 3

Enter your account number or phone number:

101

Enter target account number or phone number:

102

Enter transfer amount:

100

Amount deposited successfully. New balance: 1100.0

Transferred 100.0 to account number: 102

1. Deposit

2. Withdraw

3. Transfer

4. Check Balance

5. Back to Main Menu

Enter your choice: 4

Enter account number or phone number to check balance:

101

Account balance for nani is: 1900.0

1. Deposit

2. Withdraw

3. Transfer

4. Check Balance

5. Back to Main Menu

Enter your choice: 14

Enter a valid choice.

1. Deposit

2. Withdraw

3. Transfer

4. Check Balance

5. Back to Main Menu

Enter your choice: 102

Enter a valid choice.

1. Deposit

2. Withdraw

3. Transfer

4. Check Balance

5. Back to Main Menu

Enter your choice: 4

Enter account number or phone number to check balance:

102

Account balance for nann is: 1100.0

1. Deposit

2. Withdraw

3. Transfer

4. Check Balance

5. Back to Main Menu

Enter your choice: 5

Returning to main menu.

Create an account - Press 1

Bank operations - Press 2

Delete account - Press 3

Display all accounts - Press 4

Close the bank - Press 5

Enter your choice: 4

Account Details:

Name: nani, Account Number: 101, Balance: 1900.0

Name: nann, Account Number: 102, Balance: 1100.0

Create an account - Press 1

Bank operations - Press 2

Delete account - Press 3

Display all accounts - Press 4

Close the bank - Press 5

Enter your choice: 3

Enter the account number or phone number to remove:

101

Account removed successfully.

Create an account - Press 1

Bank operations - Press 2

Delete account - Press 3

Display all accounts - Press 4

Close the bank - Press 5

Enter your choice: 4

Account Details:

Name: nann, Account Number: 102, Balance: 1100.0

Create an account - Press 1

Bank operations - Press 2

Delete account - Press 3

Display all accounts - Press 4

Close the bank - Press 5

Enter your choice: 5

Closing the bank. Goodbye!

# Week2 ****Student Management System****

## ****Code****

The code for the Student Management System is composed of the following classes:

1. **Main.java**
   * This is the entry point of the application that initializes the StudentService and StudentUI classes and displays the main menu.
2. **StudentService.java**
   * This class handles the logic of managing student records, including adding, viewing, updating, deleting, and sorting students by marks and name.
3. **StudentUI.java**
   * This class provides a user interface that allows interaction with the system via console input and output. It invokes methods from StudentService to perform actions like adding, updating, and deleting student records.
4. **Student.java**
   * This class defines the Student object, including attributes such as student ID, name, and marks, along with corresponding getters, setters, and a toString method for easy display.

## ****Key Sections of the Code****

### ****Main.java****

java

CopyEdit

public class Main {

public static void main(String[] args) {

StudentService studentService = new StudentService();

StudentUI studentUI = new StudentUI(studentService);

studentUI.displayMenu();

}

}

### ****StudentService.java****

java

CopyEdit

public class StudentService {

private final List<Student> students = new ArrayList<>();

public void addStudent(String id, String name, int marks) {

students.add(new Student(id, name, marks));

System.out.println("Student added successfully!");

}

public List<Student> viewAllStudents() {

return students;

}

public boolean updateStudent(String id, String newName, int newMarks) {

for (Student student : students) {

if (student.getId().equals(id)) {

student.setName(newName);

student.setMarks(newMarks);

return true;

}

}

return false;

}

public boolean deleteStudent(String id) {

return students.removeIf(student -> student.getId().equals(id));

}

public void sortByMarks() {

students.sort(Comparator.comparingInt(Student::getMarks).reversed());

}

public void sortByName() {

students.sort(Comparator.comparing(Student::getName));

}

}

### ****StudentUI.java****

java

CopyEdit

public class StudentUI {

private final StudentService studentService;

private final Scanner scanner = new Scanner(System.in);

public StudentUI(StudentService studentService) {

this.studentService = studentService;

}

public void displayMenu() {

int choice;

do {

System.out.println("\n--- Student Management System ---");

System.out.println("1. Add Student");

System.out.println("2. View All Students");

System.out.println("3. Update Student Details");

System.out.println("4. Delete Student");

System.out.println("5. Sort Students by Marks");

System.out.println("6. Sort Students by Name");

System.out.println("7. Exit");

System.out.print("Enter your choice: ");

choice = scanner.nextInt();

scanner.nextLine(); // Clear buffer

switch (choice) {

case 1 -> handleAddStudent();

case 2 -> handleViewStudents();

case 3 -> handleUpdateStudent();

case 4 -> handleDeleteStudent();

case 5 -> handleSortByMarks();

case 6 -> handleSortByName();

case 7 -> System.out.println("Exiting the program...");

default -> System.out.println("Invalid choice. Please try again.");

}

} while (choice != 7);

}

private void handleAddStudent() {

System.out.print("Enter Student ID: ");

String id = scanner.nextLine();

System.out.print("Enter Student Name: ");

String name = scanner.nextLine();

System.out.print("Enter Student Marks: ");

int marks = scanner.nextInt();

scanner.nextLine(); // Clear buffer

studentService.addStudent(id, name, marks);

}

private void handleViewStudents() {

var students = studentService.viewAllStudents();

if (students.isEmpty()) {

System.out.println("No students found.");

return;

}

System.out.println("\n--- Student Records ---");

students.forEach(System.out::println);

}

private void handleUpdateStudent() {

System.out.print("Enter Student ID to update: ");

String id = scanner.nextLine();

System.out.print("Enter new Name: ");

String newName = scanner.nextLine();

System.out.print("Enter new Marks: ");

int newMarks = scanner.nextInt();

scanner.nextLine(); // Clear buffer

if (studentService.updateStudent(id, newName, newMarks)) {

System.out.println("Student details updated successfully!");

} else {

System.out.println("Student not found.");

}

}

private void handleDeleteStudent() {

System.out.print("Enter Student ID to delete: ");

String id = scanner.nextLine();

if (studentService.deleteStudent(id)) {

System.out.println("Student deleted successfully!");

} else {

System.out.println("Student not found.");

}

}

private void handleSortByMarks() {

studentService.sortByMarks();

System.out.println("Students sorted by marks in descending order.");

handleViewStudents();

}

private void handleSortByName() {

studentService.sortByName();

System.out.println("Students sorted by name in alphabetical order.");

handleViewStudents();

}

}

### ****Student.java****

java

CopyEdit

public class Student {

private String id;

private String name;

private int marks;

// Constructor

public Student(String id, String name, int marks) {

this.id = id;

this.name = name;

this.marks = marks;

}

// Getters and Setters

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getMarks() {

return marks;

}

public void setMarks(int marks) {

this.marks = marks;

}

// toString method for easy display

@Override

public String toString() {

return "ID: " + id + ", Name: " + name + ", Marks: " + marks;

}

}

## ****Explanation****

The Student Management System (SMS) is a console-based Java application designed to manage student records with basic functionalities such as adding, viewing, updating, deleting, and sorting students. This project offers a simple yet effective way of managing student data, making it easier for an administrator or user to perform CRUD operations (Create, Read, Update, and Delete) on student records. The system is structured into multiple classes to ensure modularity, ease of maintenance, and scalability.

### ****Key Features of the System****

1. **Add Student:** The user can add a new student by entering the student's ID, name, and marks. This action is performed by the StudentService class, where the input values are used to create a new Student object, which is then added to an ArrayList of students. The system acknowledges the successful addition of the student by printing a confirmation message.
2. **View All Students:** The system allows users to view the list of all students currently stored in the system. The viewAllStudents() method in the StudentService class retrieves all students from the internal list and displays them in a readable format through the StudentUI class. If no students exist, an appropriate message is displayed to inform the user.
3. **Update Student Details:** The user can update the details of an existing student by entering the student's ID, new name, and new marks. The StudentService class provides the updateStudent() method, which iterates through the list of students, checks if the student with the given ID exists, and if so, updates the student's name and marks. If the student is not found, the system notifies the user that the update could not be performed.
4. **Delete Student:** The system supports the deletion of a student based on their ID. The user provides the student's ID, and the deleteStudent() method removes the corresponding student from the ArrayList if the ID matches. If the student is not found, a message is displayed indicating the student does not exist.
5. **Sort Students by Marks:** Sorting students by marks allows the user to view the students in descending order of their academic performance. The sortByMarks() method uses Java’s built-in Comparator to sort the list of students based on their marks. This feature helps administrators quickly identify the best-performing students.
6. **Sort Students by Name:** Similar to sorting by marks, students can also be sorted alphabetically by their names using the sortByName() method. This function allows users to view students in alphabetical order, making it easier to find a student by name or just for an organized listing.

### ****System Architecture****

The system follows a basic architecture comprising three primary classes: Student, StudentService, and StudentUI. The separation of concerns ensures the program is modular and maintainable:

* **Student Class:** The Student class is a simple data model that encapsulates the properties of a student, including id, name, and marks. It also includes getter and setter methods for each attribute, which allows easy modification and access. Additionally, it overrides the toString() method to provide a string representation of the student object, which is used when displaying student information in the UI.
* **StudentService Class:** The StudentService class is responsible for handling all the business logic related to students. It manages a list of Student objects and provides methods to add, update, delete, and sort students. This class acts as a bridge between the data storage (in-memory ArrayList) and the user interface, ensuring that all operations on student records are carried out correctly.
* **StudentUI Class:** The StudentUI class provides the user interface, allowing the user to interact with the system through the console. It displays a menu with various options and prompts the user to enter their choice. Based on the user’s input, the corresponding method from the StudentService class is called to perform the requested operation. The UI also provides clear feedback to the user, informing them about the success or failure of their actions.

### ****User Interaction Flow****

1. When the program starts, the StudentUI class presents the user with a menu that offers different options for managing students.
2. Based on the user’s input, the relevant operation is executed, such as adding a new student, viewing all students, or updating a student’s details.
3. The user is continuously presented with the menu until they choose to exit the program.

### ****Why This Approach?****

This modular approach of separating concerns ensures that each class is responsible for a specific aspect of the system. The Student class handles data representation, the StudentService class manages data operations, and the StudentUI class provides interaction with the user. This structure not only keeps the code clean and organized but also allows for easy expansion and modification in the future.

For example, if the system needed to be extended to use a database for storing student records instead of using an in-memory list, only the StudentService class would need to be modified. The StudentUI class would remain unchanged, and the Student class would still represent the same data structure. This separation of concerns also makes the code more testable, as each class can be tested independently.

## ****Tools and Technologies Used****

* Java (JDK 8 or higher)
* IDE (IntelliJ IDEA, Eclipse, or NetBeans)
* Collections (ArrayList, List)
* Comparator for sorting.

## ****Output Result Screenshots****

PS D:\dataSoftixs projects\Week 2 project on student management system> & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\MAHA LAPTOPS\AppData\Roaming\Code\User\workspaceStorage\c25d168c660d38d4b98be11f52a77fc0\redhat.java\jdt\_ws\Week 2 project on student management system\_741cb83b\bin' 'Main'

--- Student Management System ---

1. Add Student

2. View All Students

3. Update Student Details

4. Delete Student

5. Sort Students by Marks

6. Sort Students by Name

7. Exit

Enter your choice: 2

No students found.

--- Student Management System ---

1. Add Student

2. View All Students

3. Update Student Details

4. Delete Student

5. Sort Students by Marks

6. Sort Students by Name

7. Exit

Enter your choice: 1

Enter Student ID: 101

Enter Student Name: nani

Enter Student Marks: 98

Student added successfully!

--- Student Management System ---

1. Add Student

2. View All Students

3. Update Student Details

4. Delete Student

5. Sort Students by Marks

6. Sort Students by Name

7. Exit

Enter your choice: 1

Enter Student ID: 102

Enter Student Name: raju

Enter Student Marks: 89

Student added successfully!

--- Student Management System ---

1. Add Student

2. View All Students

3. Update Student Details

4. Delete Student

5. Sort Students by Marks

6. Sort Students by Name

7. Exit

Enter your choice: 1

Enter Student ID: 103

Enter Student Name: ramu

Enter Student Marks: 78

Student added successfully!

--- Student Management System ---

1. Add Student

2. View All Students

3. Update Student Details

4. Delete Student

5. Sort Students by Marks

6. Sort Students by Name

7. Exit

Enter your choice: 1

Enter Student ID: 104

Enter Student Name: sai

Enter Student Marks: 85

Student added successfully!

--- Student Management System ---

1. Add Student

2. View All Students

3. Update Student Details

4. Delete Student

5. Sort Students by Marks

6. Sort Students by Name

7. Exit

Enter your choice: 1

Enter Student ID: 105

Enter Student Name: tharun

Enter Student Marks: 63

Student added successfully!

--- Student Management System ---

1. Add Student

2. View All Students

3. Update Student Details

4. Delete Student

5. Sort Students by Marks

6. Sort Students by Name

7. Exit

Enter your choice: 2

--- Student Records ---

ID: 101, Name: nani, Marks: 98

ID: 102, Name: raju, Marks: 89

ID: 103, Name: ramu, Marks: 78

ID: 104, Name: sai, Marks: 85

ID: 105, Name: tharun, Marks: 63

--- Student Management System ---

1. Add Student

2. View All Students

3. Update Student Details

4. Delete Student

5. Sort Students by Marks

6. Sort Students by Name

7. Exit

Enter your choice: 3

Enter Student ID to update: 101

Enter new Name: vinni Nani

Enter new Marks: 100

Student details updated successfully!

--- Student Management System ---

1. Add Student

2. View All Students

3. Update Student Details

4. Delete Student

5. Sort Students by Marks

6. Sort Students by Name

7. Exit

Enter your choice: 4

Enter Student ID to delete: 105

Student deleted successfully!

--- Student Management System ---

1. Add Student

2. View All Students

3. Update Student Details

4. Delete Student

5. Sort Students by Marks

6. Sort Students by Name

7. Exit

Enter your choice: 5

Students sorted by marks in descending order.

--- Student Records ---

ID: 101, Name: vinni Nani, Marks: 100

ID: 102, Name: raju, Marks: 89

ID: 104, Name: sai, Marks: 85

ID: 103, Name: ramu, Marks: 78

--- Student Management System ---

1. Add Student

2. View All Students

3. Update Student Details

4. Delete Student

5. Sort Students by Marks

6. Sort Students by Name

7. Exit

Enter your choice: 6

Students sorted by name in alphabetical order.

--- Student Records ---

ID: 102, Name: raju, Marks: 89

ID: 103, Name: ramu, Marks: 78

ID: 104, Name: sai, Marks: 85

ID: 101, Name: vinni Nani, Marks: 100

--- Student Management System ---

1. Add Student

2. View All Students

3. Update Student Details

4. Delete Student

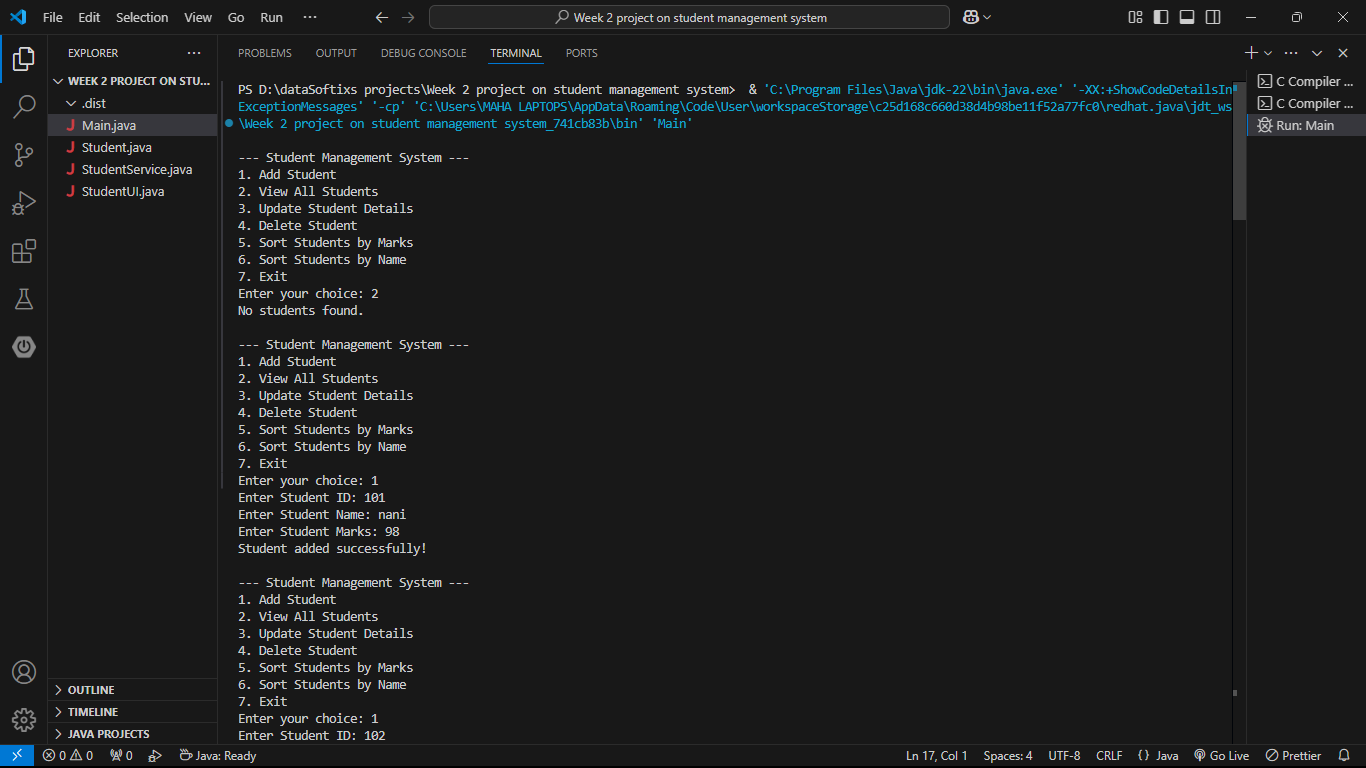
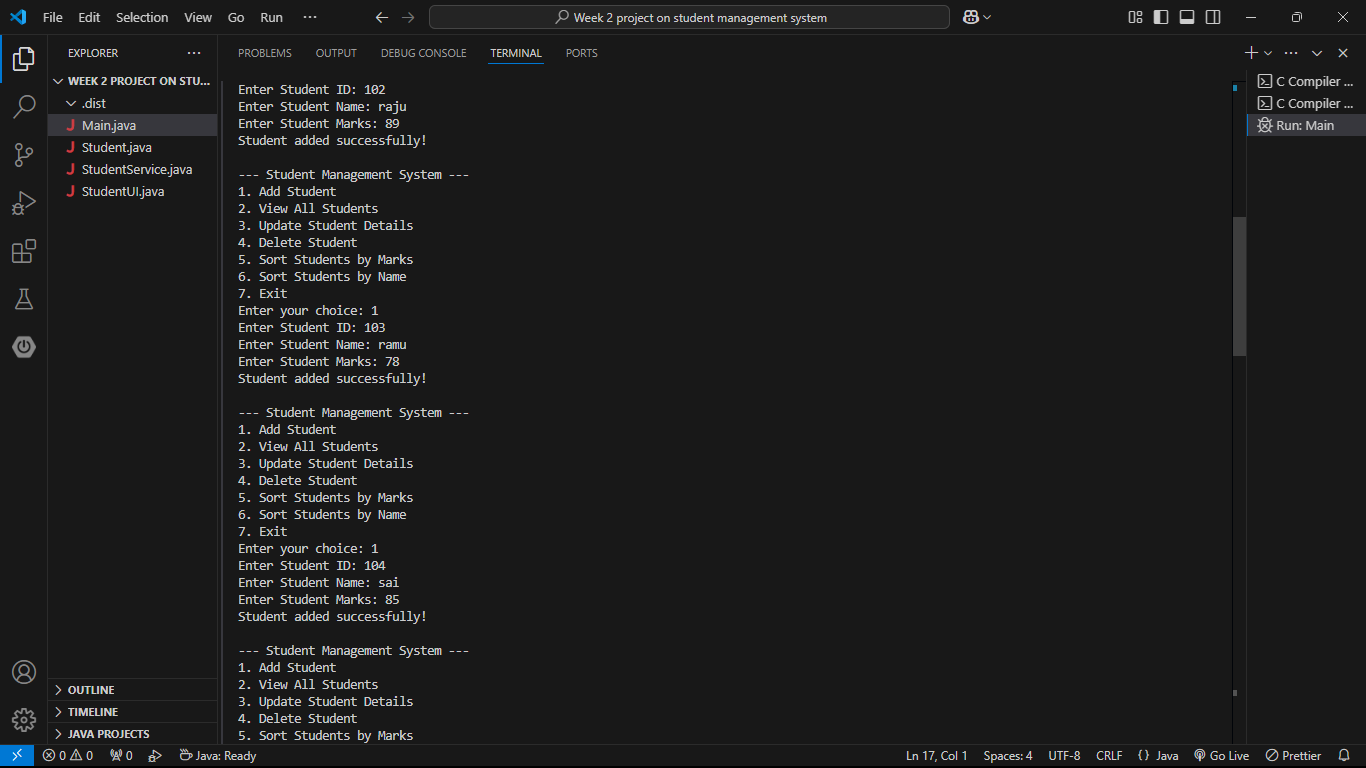
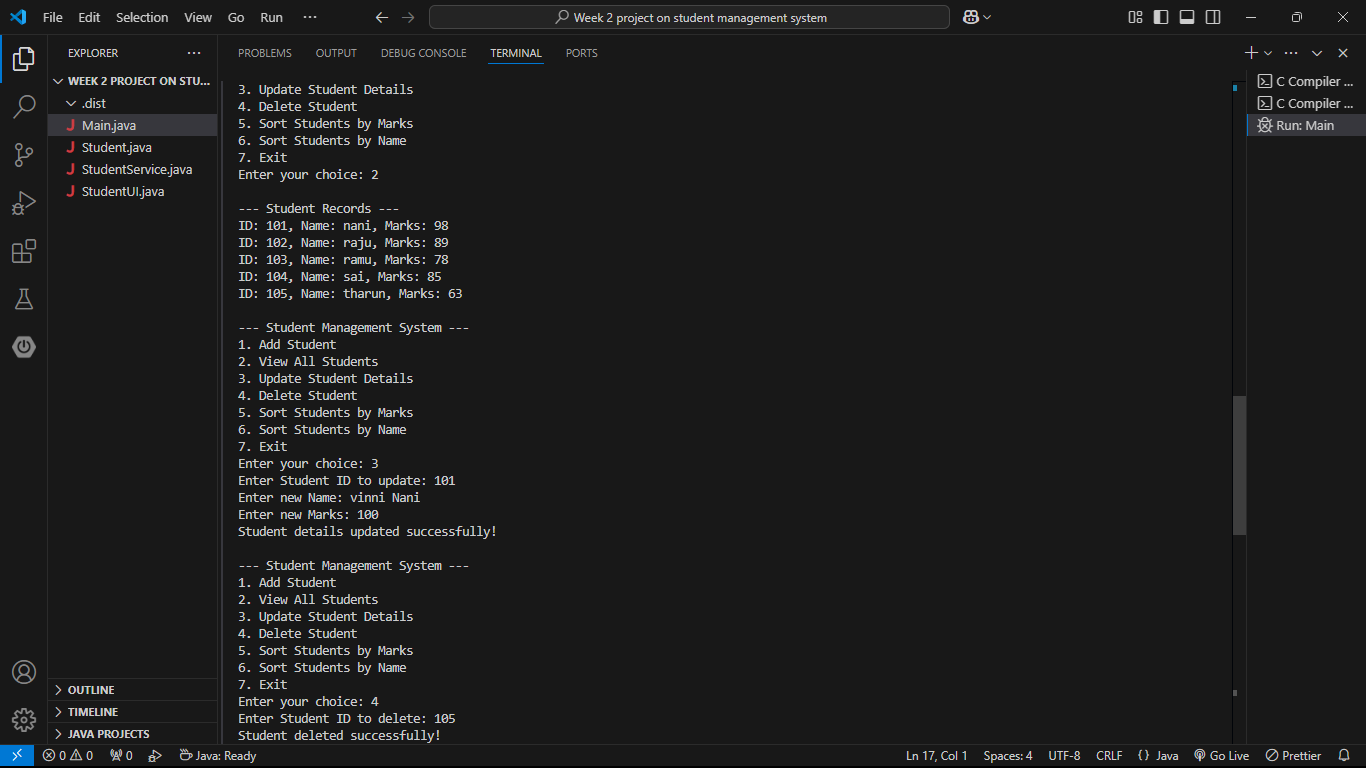
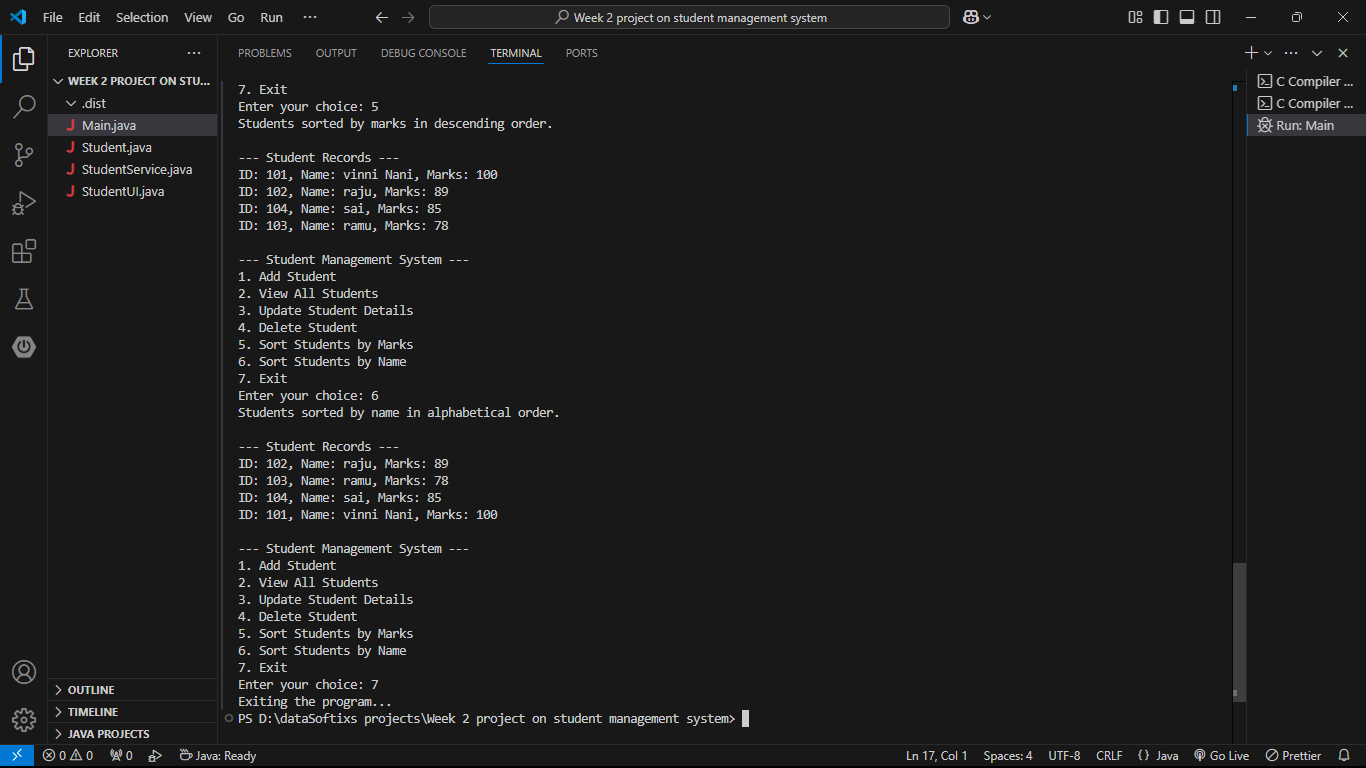
5. Sort Students by Marks

6. Sort Students by Name

7. Exit

Enter your choice: 7

Exiting the program...

PS D:\dataSoftixs projects\Week 2 project on student management system>    

Week 3 Document - File Organization and Duplicate File Deletion Project

# Project Overview

This project focuses on organizing files within a specified folder and deleting duplicate files based on their content using SHA-256 hashing. It includes:  
- File organization into folders like Documents, Images, and Others.  
- Logging of organized files.  
- Duplicate file handling using hash comparison for file deletion.

# Code: Key Sections

## Main.java

This is the entry point of the program. It prompts the user for a folder path and invokes methods for file organization, logging, and duplicate deletion.

public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 System.out.println("Enter the folder path to organize:");  
 String folderPath = scanner.nextLine();  
   
 File folder = new File(folderPath);  
 if (!folder.exists() || !folder.isDirectory()) {  
 System.out.println("Invalid folder path.");  
 return;  
 }  
  
 FileOrganizer.organizeFiles(folder);  
 FileLogger.logOrganizedFiles(folder);  
  
 System.out.println("Do you want to delete duplicate files? (yes/no):");  
 String deleteDuplicates = scanner.nextLine();  
 if (deleteDuplicates.equalsIgnoreCase("yes")) {  
 DuplicateFileHandler.deleteDuplicateFiles(folder);  
 }  
}

## FileOrganizer.java

This class organizes files into categories such as Documents, Images, and Others based on their file extension.

public static void organizeFiles(File folder) {  
 File documentsFolder = new File(folder, "Documents");  
 File imagesFolder = new File(folder, "Images");  
 File othersFolder = new File(folder, "Others");  
  
 documentsFolder.mkdir();  
 imagesFolder.mkdir();  
 othersFolder.mkdir();  
  
 File[] files = folder.listFiles();  
 if (files == null) return;  
  
 for (File file : files) {  
 if (file.isFile()) {  
 String fileType = getFileExtension(file);  
 switch (fileType) {  
 case "txt":  
 case "doc":  
 case "docx":  
 case "pdf":  
 moveFile(file, documentsFolder);  
 break;  
 case "jpg":  
 case "jpeg":  
 case "png":  
 case "gif":  
 moveFile(file, imagesFolder);  
 break;  
 default:  
 moveFile(file, othersFolder);  
 break;  
 }  
 }  
 }  
}

## DuplicateFileHandler.java

This class handles the deletion of duplicate files by comparing their SHA-256 hashes.

public static void deleteDuplicateFiles(File folder) {  
 Map<String, File> fileHashes = new HashMap<>();  
 File[] files = folder.listFiles();  
 if (files == null) return;  
  
 for (File file : files) {  
 if (file.isFile()) {  
 try {  
 String fileHash = getFileHash(file);  
 if (fileHashes.containsKey(fileHash)) {  
 System.out.println("Deleting duplicate file: " + file.getName());  
 file.delete();  
 } else {  
 fileHashes.put(fileHash, file);  
 }  
 } catch (IOException e) {  
 System.out.println("Error reading file: " + file.getName());  
 e.printStackTrace();  
 }  
 }  
 }  
}

# 3. Explanation

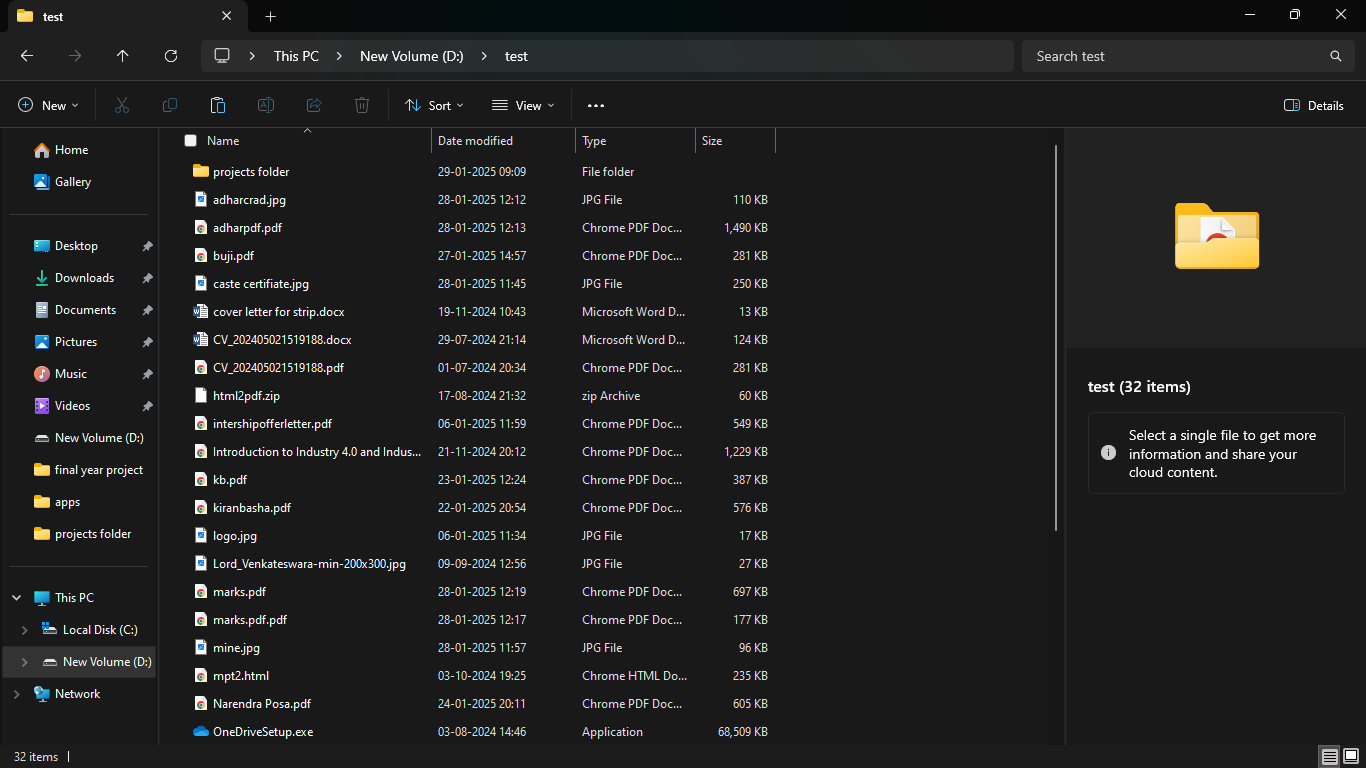
This project works by first organizing files in a folder based on their extensions. The files are sorted into respective directories such as 'Documents', 'Images', and 'Others.' The program checks the folder for any files and moves them to the appropriate folder based on file type.  
A log is then created to record which files were organized into which folder. The user is given the option to delete duplicate files. If they choose 'yes,' the program scans the folder, computes the hash of each file, and deletes any file that shares the same hash as another.

# Tools and Technologies Used

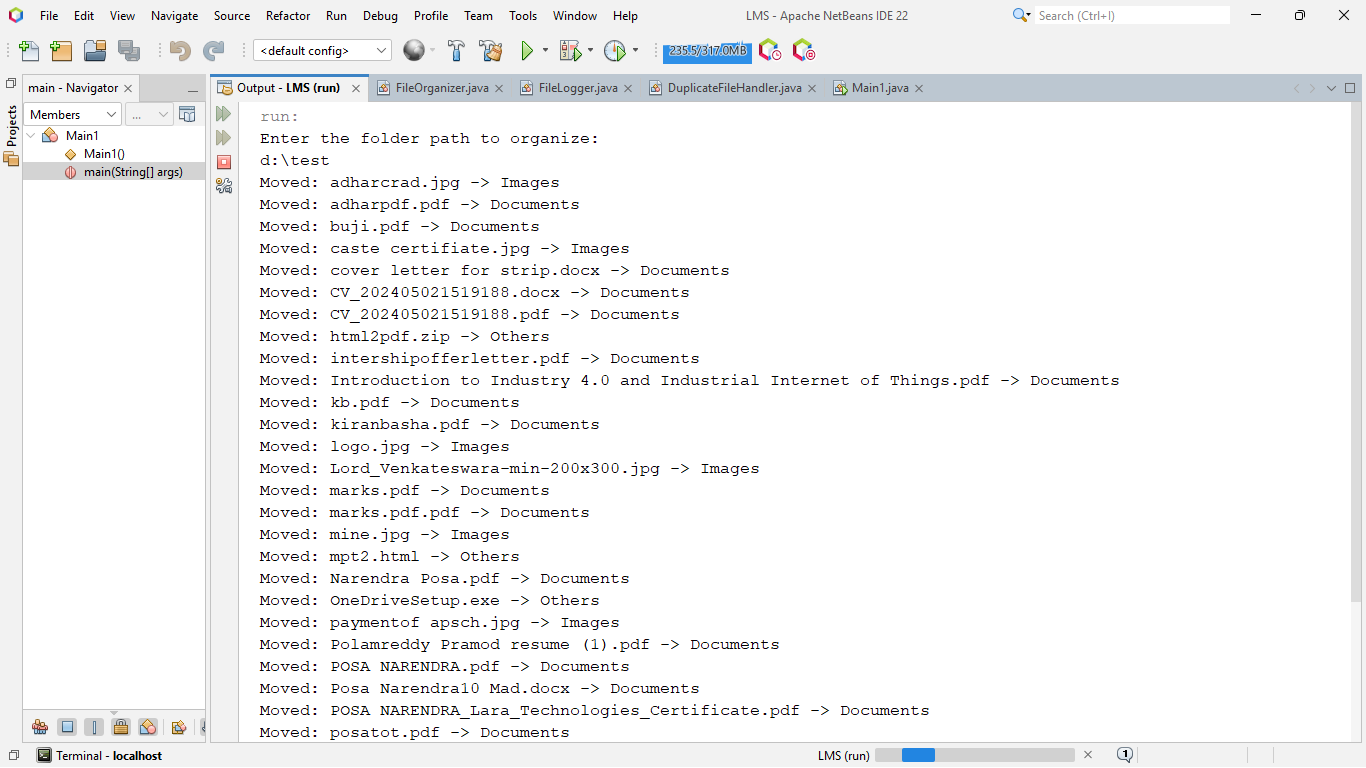
- Java: The primary programming language used for implementing the project.  
- Java.io package: For file operations such as reading, moving, and deleting files.  
- Java.nio.file package: For handling file movements.  
- SHA-256: Cryptographic hash function used for comparing file content.  
- Scanner: Used for user input.  
- PrintWriter: Used for logging organized files.

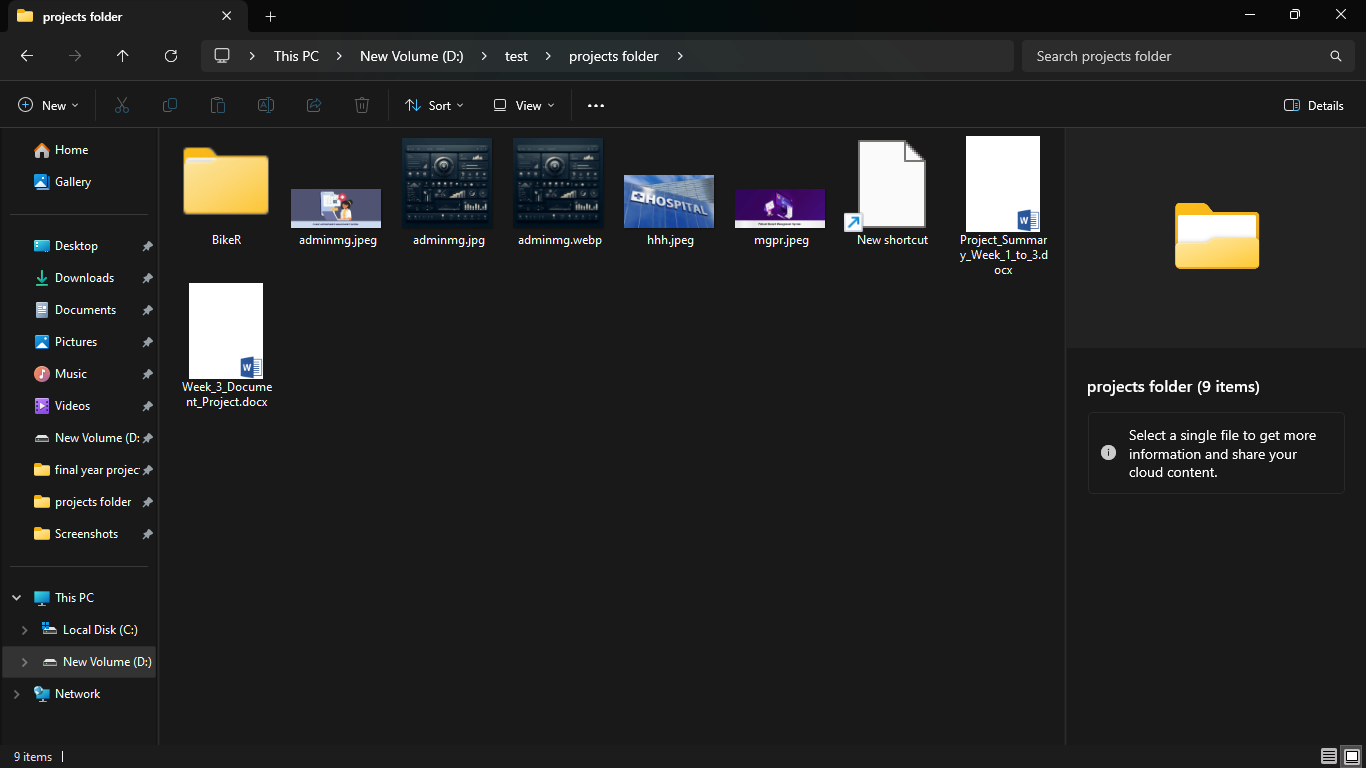
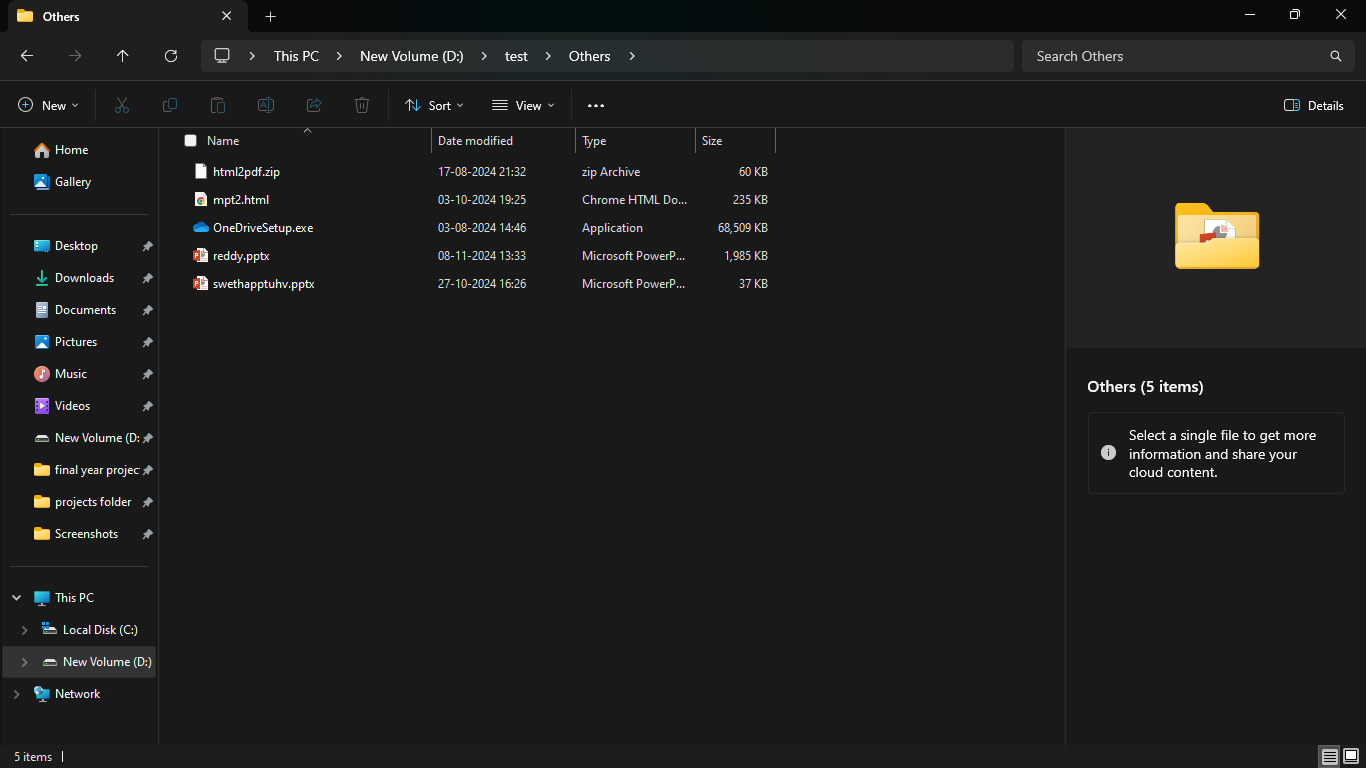
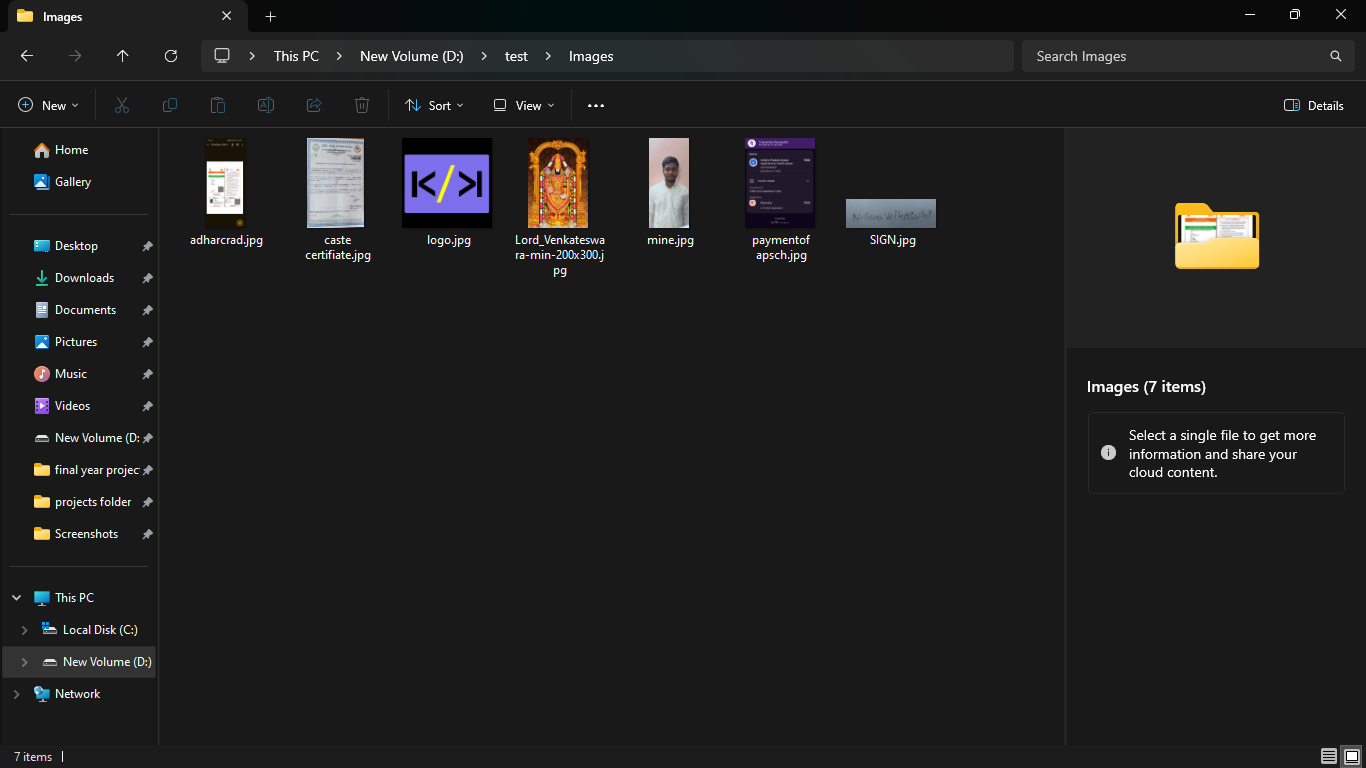
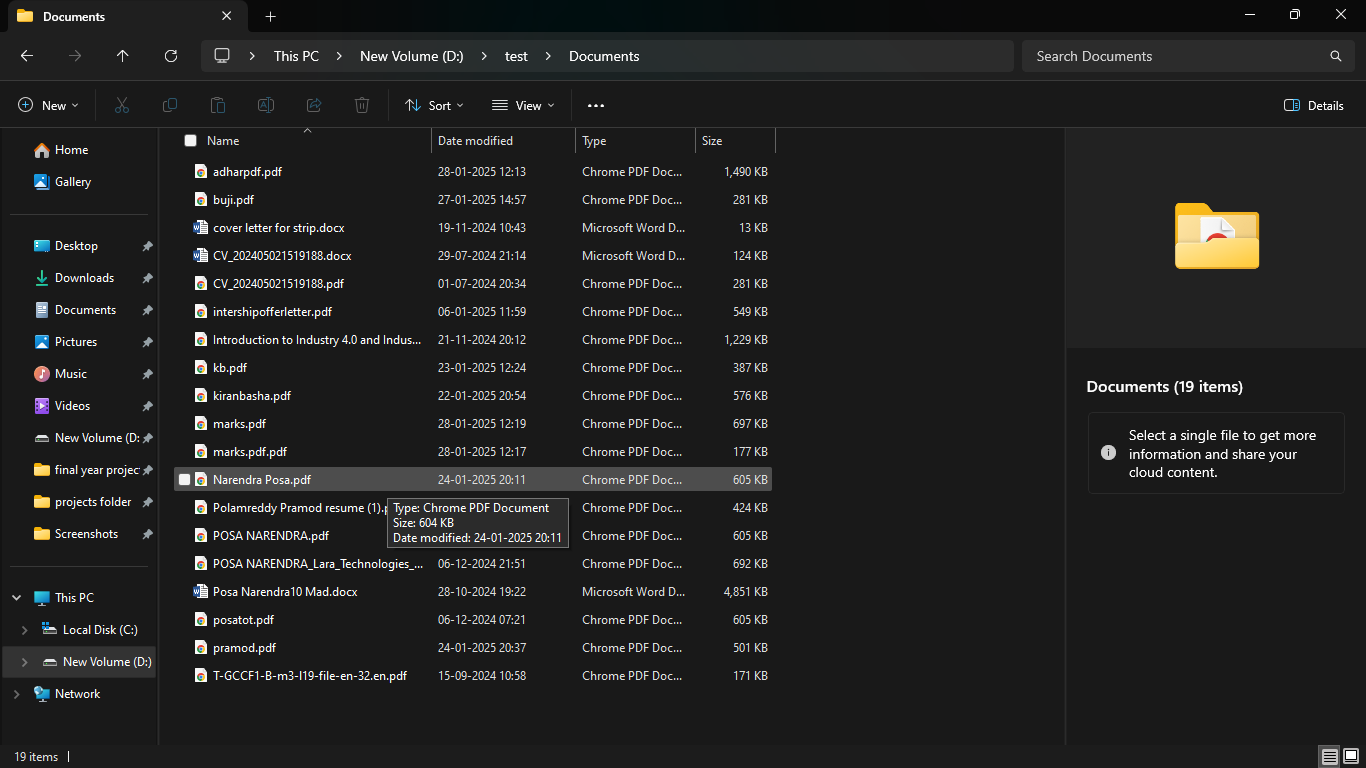
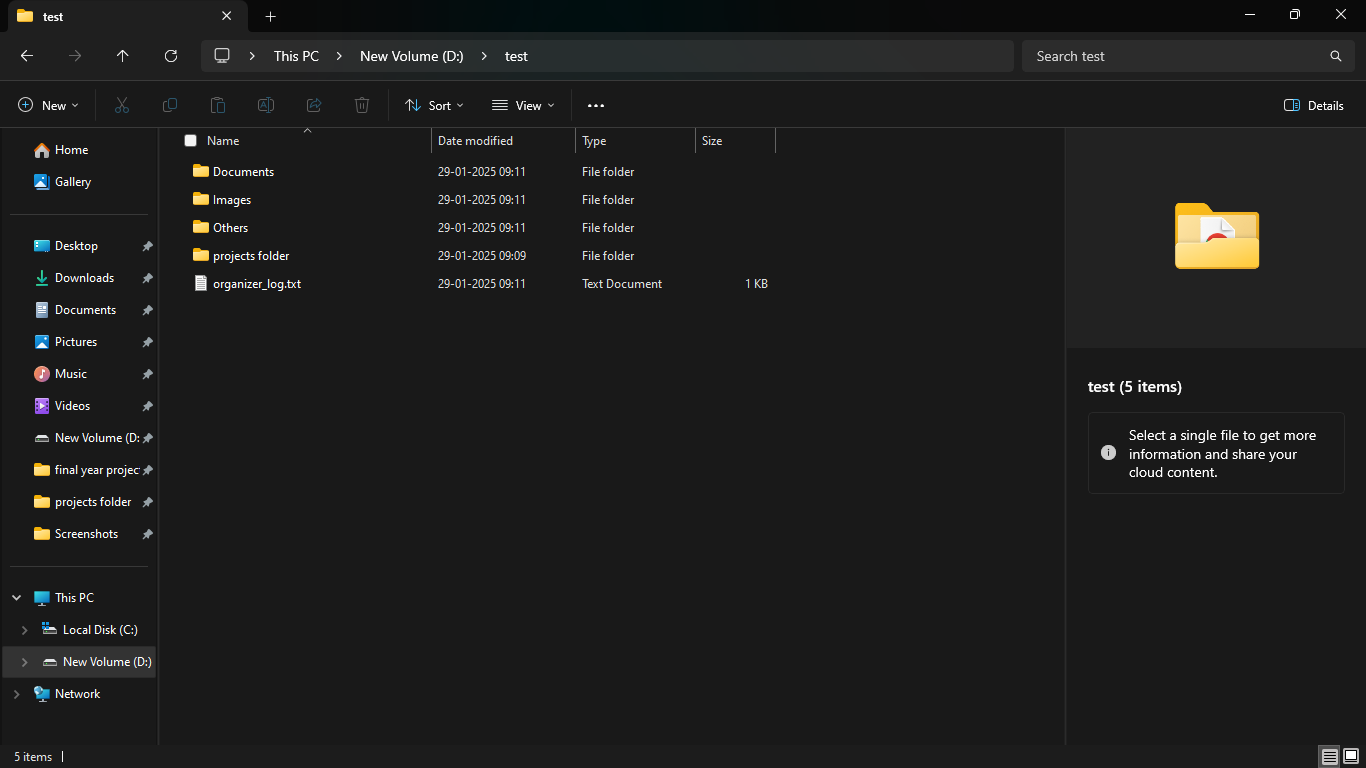
# Output Result Screenshots

Before run the code



After run the code



The file are now 

### ****Conclusion****

In conclusion, the three projects—**Student Management System**, **Bank Operation System**, and **File Organization System**—represent a significant step forward in understanding and applying core software development concepts. Each project offers unique insights into various aspects of system design and functionality.

1. **Student Management System**: This project provides a hands-on experience with Java, focusing on CRUD operations and data manipulation. The implementation of sorting and updating functionalities further enhances the user experience and demonstrates the practical use of data structures like lists.
2. **Bank Operation System**: This project introduces banking concepts, where various operations such as account creation, deposit, withdrawal, and balance checking are executed. It enhances understanding of system transactions, security, and managing financial data.
3. **File Organization System**: Focused on file management, this project emphasizes the organization and cleanup of system files. The duplicate file deletion and file categorization functionalities provide a glimpse into how data can be effectively managed in a real-world system.

Each project was designed with a focus on practicality, ease of use, and efficient data management. They showcase different aspects of software development, including user interface design, data structure usage, and implementing business logic. These projects, together, provide a comprehensive understanding of core programming and system design principles.

### ****Acknowledgment****

I would like to express my heartfelt gratitude to **Datasoftixs** for the wonderful opportunity to intern with them. This internship has been an incredible learning experience, allowing me to apply my knowledge to real-world projects and hone my skills. I appreciate the guidance, feedback, and encouragement I received from the entire team. The knowledge and experience gained during this time will certainly contribute to my growth as a developer and help me in future endeavors.

Thank you for trusting me with these valuable projects, and I look forward to more opportunities for learning and contributing to the team.