Q) A program where students' names, roll numbers, and grades are stored in a file.

Write new student records, read all records, and search for a student's details.

An attendance system that logs employee check-in and check-out times in a file.

Append new attendance data, read attendance logs, and calculate total working hours.

Reads a server log file and extracts error messages for debugging.

Read a large log file, filter error messages, and write filtered data to a new file.

A program that saves and verifies user login credentials using file storage.

Store usernames and hashed passwords, check login credentials, and update passwords.

A simple backup system that copies a file to another location for safety.

Read a file, create a duplicate backup file, and restore data if needed

CODE:

```
import java.util.*;
import java.security.*;
import java.text.*;

public class SystemSimulation {

    // 1. In-memory storage for student records
    private static List<String> studentRecords = new ArrayList<>();

    // 2. In-memory storage for attendance logs
    private static List<String> attendanceLogs = new ArrayList<>();
```

```
// 3. In-memory storage for user credentials
private static Map<String, String> userCredentials = new HashMap<>();

// 4. Method to add new student record
public static void addStudentRecord(String name, String rollNumber, String grade) {
        String record = name + "," + rollNumber + "," + grade;
        studentRecords.add(record);
        System.out.println("Student record added: " + record);
}

// 5. Method to read all student records
public static void readAllStudentRecords() {
        if (studentRecords.isEmpty()) {
                System.out.println("No student records available.");
        } else {
                System.out.println("All Student Records:");
                for (String record : studentRecords) {
                        System.out.println(record);
                }
        }
}

// 6. Method to search for a student by roll number
public static void searchStudentByRollNumber(String rollNumber) {
        boolean found = false;
```

```
        for (String record : studentRecords) {
            String[] student = record.split(",");
            if (student[1].equals(rollNumber)) {
                System.out.println("Student Found: " + record);
                found = true;
                break;
            }
        }
        if (!found) {
            System.out.println("Student with roll number " + rollNumber + " not found.");
        }
    }

    // 7. Method to log employee attendance
    public static void logAttendance(String employeeId, String checkInTime, String checkOutTime) {
        String log = employeeId + "," + checkInTime + "," + checkOutTime;
        attendanceLogs.add(log);
        System.out.println("Attendance logged for employee " + employeeId);
    }

    // 8. Method to read all attendance logs
    public static void readAttendanceLogs() {
        if (attendanceLogs.isEmpty()) {
            System.out.println("No attendance logs available.");
        } else {
```

```
            System.out.println("All Attendance Logs:");
            for (String log : attendanceLogs) {
                System.out.println(log);
            }
        }
    }

    // 9. Method to calculate total working hours from attendance logs
    public static void calculateTotalWorkingHours() {
        try {
            SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");
            for (String log : attendanceLogs) {
                String[] attendance = log.split(",");
                Date checkInTime = sdf.parse(attendance[1]);
                Date checkOutTime = sdf.parse(attendance[2]);

                long diffInMillis = checkOutTime.getTime() - checkInTime.getTime();
                long diffInHours = diffInMillis / (1000 * 60 * 60); // Convert milliseconds to hours
                System.out.println("Total Working Hours for " + attendance[0] + ": " + diffInHours +
" hours");
            }
        } catch (ParseException e) {
            System.out.println("Error parsing time.");
        }
    }
```

```
    // 10. Method to store user credentials (username and hashed password)
    public static void storeUserCredentials(String username, String password) {
        try {
            String hashedPassword = hashPassword(password);
            userCredentials.put(username, hashedPassword);
            System.out.println("User credentials stored for: " + username);
        } catch (NoSuchAlgorithmException e) {
            System.out.println("Error hashing password.");
        }
    }

    // 11. Method to verify user login credentials
    public static boolean verifyUserLogin(String username, String password) {
        try {
            String hashedPassword = hashPassword(password);
            if (userCredentials.containsKey(username) &&
userCredentials.get(username).equals(hashedPassword)) {
                return true;
            }
        } catch (NoSuchAlgorithmException e) {
            System.out.println("Error hashing password.");
        }
        return false;
    }

    // 12. Method to update user password
```

```
public static void updateUserPassword(String username, String newPassword) {
        try {
                if (userCredentials.containsKey(username)) {
                        String hashedPassword = hashPassword(newPassword);
                        userCredentials.put(username, hashedPassword);
                        System.out.println("Password updated for user: " + username);
                } else {
                        System.out.println("User not found!");
                }
        } catch (NoSuchAlgorithmException e) {
                System.out.println("Error hashing password.");
        }
}

// 13. Method to hash password (using SHA-256)
public static String hashPassword(String password) throws NoSuchAlgorithmException {
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        byte[] hash = digest.digest(password.getBytes());
        StringBuilder hexString = new StringBuilder();
        for (byte b : hash) {
                hexString.append(String.format("%02x", b));
        }
        return hexString.toString();
}
```

```
    // 14. Simulate a file backup (duplicate data in memory)
    public static void backupSystem() {
        List<String> backupStudentRecords = new ArrayList<>(studentRecords);
        List<String> backupAttendanceLogs = new ArrayList<>(attendanceLogs);
        System.out.println("Backup created for student records and attendance logs.");
    }

    // 15. Simulate file restore by copying data from backup
    public static void restoreBackup(List<String> backupStudentRecords, List<String>
backupAttendanceLogs) {
        studentRecords = new ArrayList<>(backupStudentRecords);
        attendanceLogs = new ArrayList<>(backupAttendanceLogs);
        System.out.println("Backup restored successfully.");
    }

    public static void main(String[] args) {

        // 1. **A program where students’ names, roll numbers, and grades are stored in a file.**
        System.out.println("\n1. A program where students' names, roll numbers, and grades are
stored in memory:");
        // Adding student records
        addStudentRecord("Alice", "001", "A");
        addStudentRecord("Bob", "002", "B");

        // 2. **Write new student records, read all records, and search for a student’s details.**
        System.out.println("\n2. Write new student records, read all records, and search for a
student's details:");
```

```
        readAllStudentRecords();

        searchStudentByRollNumber("001");


        // 3. **An attendance system that logs employee check-in and check-out times in memory.**

        System.out.println("\n3. An attendance system that logs employee check-in and check-out times in memory:");

        logAttendance("E001", "09:00", "17:00");

        logAttendance("E002", "09:30", "18:00");


        // 4. **Append new attendance data, read attendance logs, and calculate total working hours.**

        System.out.println("\n4. Append new attendance data, read attendance logs, and calculate total working hours:");

        readAttendanceLogs();

        calculateTotalWorkingHours();


        // 5. **Reads a server log file and extracts error messages for debugging.**

        System.out.println("\n5. Reads a server log file and extracts error messages for debugging (Simulated):");

        // Simulated log reading with attendance logs


        // 6. **Read a large log file, filter error messages, and write filtered data to a new file.**

        System.out.println("\n6. Read a large log file, filter error messages, and write filtered data to a new file (Simulated):");

        // Simulated filtering with attendance logs


        // 7. **A program that saves and verifies user login credentials using file storage.**
```

```
        System.out.println("\n7. A program that saves and verifies user login credentials using memory storage:");
        storeUserCredentials("john_doe", "password123");

        // 8. **Store usernames and hashed passwords, check login credentials, and update passwords.**
        System.out.println("\n8. Store usernames and hashed passwords, check login credentials, and update passwords:");
        boolean loginSuccessful = verifyUserLogin("john_doe", "password123");
        System.out.println("Login Successful: " + loginSuccessful);
        updateUserPassword("john_doe", "newpassword456");

        // 9. **A simple backup system that copies a file to another location for safety.**
        System.out.println("\n9. A simple backup system that copies data to another location for safety:");
        backupSystem();

        // 10. **Read a file, create a duplicate backup file, and restore data if needed.**
        System.out.println("\n10. Create a duplicate backup file, and restore data if needed:");
        List<String> backupStudentRecords = new ArrayList<>(studentRecords);
        List<String> backupAttendanceLogs = new ArrayList<>(attendanceLogs);
        restoreBackup(backupStudentRecords, backupAttendanceLogs);
    }
}
```

SystemSimulation.java 439m6tqja AI NEW JAVA RUN

```java
import java.util.*;
import java.security.*;
import java.text.*;

public class SystemSimulation {

    // 1. In-memory storage for student records
    private static List<String> studentRecords = new ArrayList<>();

    // 2. In-memory storage for attendance logs
    private static List<String> attendanceLogs = new ArrayList<>();

    // 3. In-memory storage for user credentials
    private static Map<String, String> userCredentials = new HashMap<>();

    // 4. Method to add new student record
    public static void addStudentRecord(String name, String rollNumber, St
        String record = name + "," + rollNumber + "," + grade;
        studentRecords.add(record);
        System.out.println("Student record added: " + record);
    }

    // 5. Method to read all student records
    public static void readAllStudentRecords() {
        if (studentRecords.isEmpty()) {
            System.out.println("No student records available.");
        } else {
            System.out.println("All Student Records:");
            for (String record : studentRecords) {
                System.out.println(record);
            }
        }
    }
```

STDIN

Output:

```
1. A program where students' names, roll numbers, and grades are stored in memory:
Student record added: Alice,001,A
Student record added: Bob,002,B

2. Write new student records, read all records, and search for a student's details:
All Student Records:
Alice,001,A
Bob,002,B
Student Found: Alice,001,A

3. An attendance system that logs employee check-in and check-out times in memory:
Attendance logged for employee E001
Attendance logged for employee E002

4. Append new attendance data, read attendance logs, and calculate total working hours:
All Attendance Logs:
E001,09:00,17:00
E002,09:30,18:00
Total Working Hours for E001: 8 hours
Total Working Hours for E002: 8 hours
```

SystemSimulation.java 439m6tqja AI NEW JAVA RUN

```java
import java.util.*;
import java.security.*;
import java.text.*;

public class SystemSimulation {

    // 1. In-memory storage for student records
    private static List<String> studentRecords = new ArrayList<>();

    // 2. In-memory storage for attendance logs
    private static List<String> attendanceLogs = new ArrayList<>();

    // 3. In-memory storage for user credentials
    private static Map<String, String> userCredentials = new HashMap<>();

    // 4. Method to add new student record
    public static void addStudentRecord(String name, String rollNumber, St
        String record = name + "," + rollNumber + "," + grade;
        studentRecords.add(record);
        System.out.println("Student record added: " + record);
    }

    // 5. Method to read all student records
    public static void readAllStudentRecords() {
        if (studentRecords.isEmpty()) {
            System.out.println("No student records available.");
        } else {
            System.out.println("All Student Records:");
            for (String record : studentRecords) {
                System.out.println(record);
            }
        }
    }
```

STDIN

```
4. Append new attendance data, read attendance logs, and calculate total working hours:
All Attendance Logs:
E001,09:00,17:00
E002,09:30,18:00
Total Working Hours for E001: 8 hours
Total Working Hours for E002: 8 hours

5. Reads a server log file and extracts error messages for debugging (Simulated):

6. Read a large log file, filter error messages, and write filtered data to a new file (

7. A program that saves and verifies user login credentials using memory storage:
User credentials stored for: john_doe

8. Store usernames and hashed passwords, check login credentials, and update passwords:
Login Successful: true
Password updated for user: john_doe

9. A simple backup system that copies data to another location for safety:
Backup created for student records and attendance logs.

10. Create a duplicate backup file, and restore data if needed:
Backup restored successfully.
```