Experiment 1

Student Name: Sahil Wadhwa **UID: 22BCS10855**
Branch: BE-CSE **Section/Group: KRG-IOT-2(B)**
Semester:6th **Date of Performance:**
Subject Name: Project Based Learning in Java with Lab **Subject Code: 22CSH-359**

1. **Aim:** Given the following table containing information about employees of an organization, develop a small java application, which accepts employee id from the command prompt and displays the following details as output: Emp No Emp Name Department Designation and Salary
   You may assume that the array is initialized with the following details:

Emp No.	Emp Name	Join Date	Desig Code	Dept	Basic	HRA	IT
1001	Ashish	01/04/2009	e	R&D	20000	8000	3000
1002	Sushma	23/08/2012	c	PM	30000	12000	9000
1003	Rahul	12/11/2008	k	Acct	10000	8000	1000
1004	Chahat	29/01/2013	r	Front Desk	12000	6000	2000
1005	Ranjan	16/07/2005	m	Engg	50000	20000	20000
1006	Suman	1/1/2000	e	Manu factur ing	23000	9000	4400
1007	Tanmay	12/06/2006	c	PM	29000	12000	10000

   Salary is calculated as Basic+HRA+DA-IT. (DA details are given in the Designation table)

   Designation details :

Designation Code	Designation	DA
e	Engineer	20000
c	Consultant	32000
k	Clerk	12000
r	Receptionist	15000
m	Manager	40000

Use Switch-Case to print Designation in the output and to find the value of DA for a particular employee.

2. **Objective:**

i. Assuming that your class name is Project1, and you execute your code as java Project1 1003, it should display the following output :

```
Emp No.  Emp Name  Department  Designation  Salary
1003      Rahul       Acct          Clerk          29000
```

ii. java Project1 123

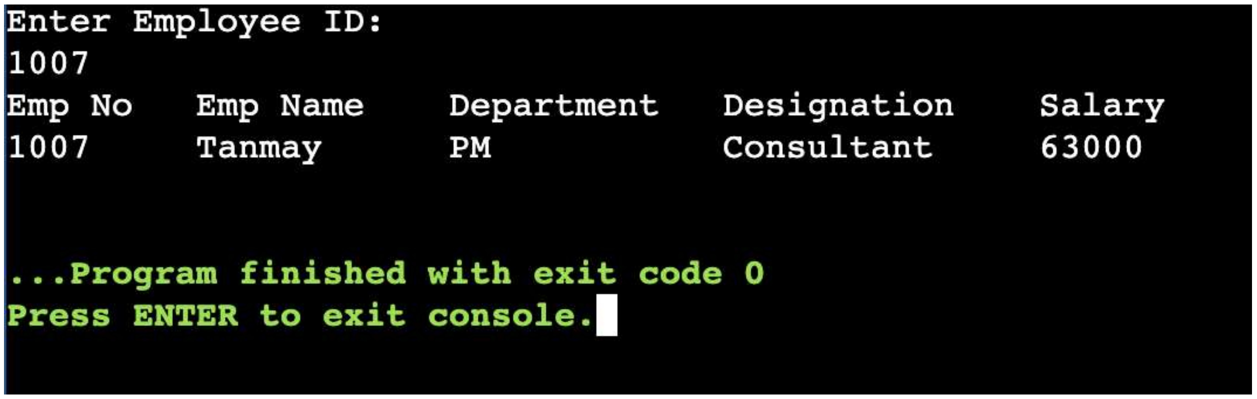
There is no employee with empid : 123

3. **Implementation/Code:**

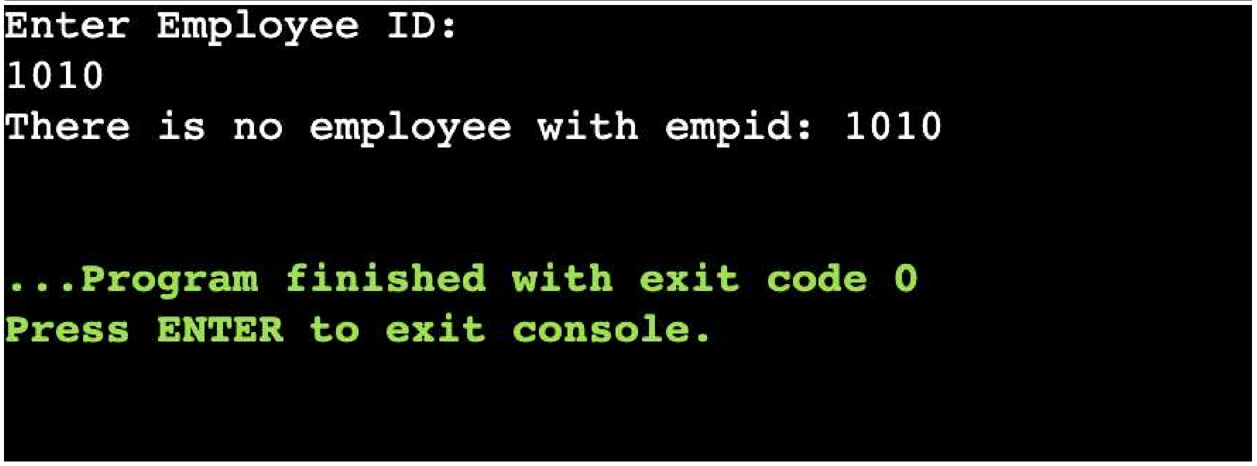
```
import java.util.Scanner;

public class Project1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        String[] empNo = {"1001", "1002", "1003", "1004", "1005", "1006",
"1007"};
```

```
        String[] empName = {"Ashish", "Sushma", "Rahul", "Chahat",
"Ranjan", "Suman", "Tanmay"};
        String[] joinDate = {"01/04/2009", "23/08/2012", "12/11/2008",
"29/01/2013", "16/07/2005", "01/01/2000", "12/06/2006"};
        String[] desigCode = {"e", "c", "k", "r", "m", "e", "c"};
        String[] dept = {"R&D", "PM", "Acct", "Front Desk", "Engg",
"Manufacturing", "PM"};
        int[] basic = {20000, 30000, 10000, 12000, 50000, 23000, 29000};
        int[] hra = {8000, 12000, 8000, 6000, 20000, 9000, 12000};
        int[] it = {3000, 9000, 1000, 2000, 20000, 4400, 10000};

        System.out.print("Enter Employee ID: ");
        Int EmpId = sc.nextInt();

        int index = -1;
        for (int i = 0; i < empNo.length; i++) {
            if (empNo[i].equals(inputEmpId)) {
                index = i;
                break;
            }
        }

        if (index == -1) {
            System.out.println("There is no employee with empid: " +
inputEmpId);
        } else {
            int da;
            String designation;
            switch (desigCode[index]) {
                case "e":
                    designation = "Engineer";
                    da = 20000;
                    break;
                case "c":
                    designation = "Consultant";
                    da = 32000;
                    break;
                case "k":
                    designation = "Clerk";
                    da = 12000;
                    break;
                case "r":
                    designation = "Receptionist";
                    da = 15000;
                    break;
                case "m":
                    designation  = "Manager";
                    da = 40000;
                    break;
                default:
                    designation  = "Unknown";
                    da = 0;
            }

            int salary = basic[index] + hra[index] + da  - it[index];

            System.out.println("Emp No      Emp Name     Department
Designation     Salary");
           System.out.println(empNo[index] + "     " + empName[index] + "     "
+
                    dept[index] + "     " + designation + "     " + salary);
        }

        sc.close();
    }
}
```

4. **Output:**



```
Enter Employee ID:
1007
Emp No     Emp Name     Department    Designation     Salary
1007       Tanmay       PM            Consultant      63000


...Program finished with exit code 0
Press ENTER to exit console.
```



```
Enter Employee ID:
1010
There is no employee with empid: 1010


...Program finished with exit code 0
Press ENTER to exit console.
```

5. **Learning Outcomes:**

i. Understand how to map employee details (like designation codes to roles) using efficient logic and structures.

ii. Learn to identify and address input mismatches or invalid entries through proper validation and error messages.

iii. Gain skills in presenting data in a well-structured and readable format for better user understanding.

Experiment 2

Student Name: Sahil Wadhwa **UID: 22BCS10855**
Branch: BE-CSE **Section/Group: KRG-IOT-2(B)**
Semester:6th **Date of Performance:**
Subject Name: Project Based Learning in Java with Lab **Subject Code: 22CSH-359**

1. **Aim:** The aim of this project is to design and implement a simple inventory control system for a small video rental store. Define least two classes: a class Video to model a video and a class VideoStore to model the

   actual store.

   Assume that an object of class Video has the following attributes:

   1. A title;
   2. a flag to say whether it is checked out or not;
   3. An average user rating.

   Add instance variables for each of these attributes to the Video class.

   In addition, you will need to add methods corresponding to the following:

   1. being checked out;
   2. being returned; 3. receiving a rating.

   The VideoStore class will contain at least an instance variable that references an array of videos (say of length 10). The VideoStore will contain the following methods:

   1. addVideo(String): add a new video (by title) to the inventory;
   2. checkOut(String): check out a video (by title);
   3. returnVideo(String): return a video to the store;
   4. receiveRating(String, int) : take a user's rating for a video; and 5. listInventory(): list the whole inventory of videos in the store.

2. **Objective:** Create a VideoStoreLauncher class with a main() method which will test the functionality of your other two classes. It should allow the following.

   1. Add 3 videos: "The Matrix", "Godfather II", "Star Wars Episode IV: A New Hope".
   2. Give several ratings to each video.
   3. Rent each video out once and return it.

   List the inventory after "Godfather II" has been rented out.

3. Implementation/Code:

1. **Video Class:-** class Video

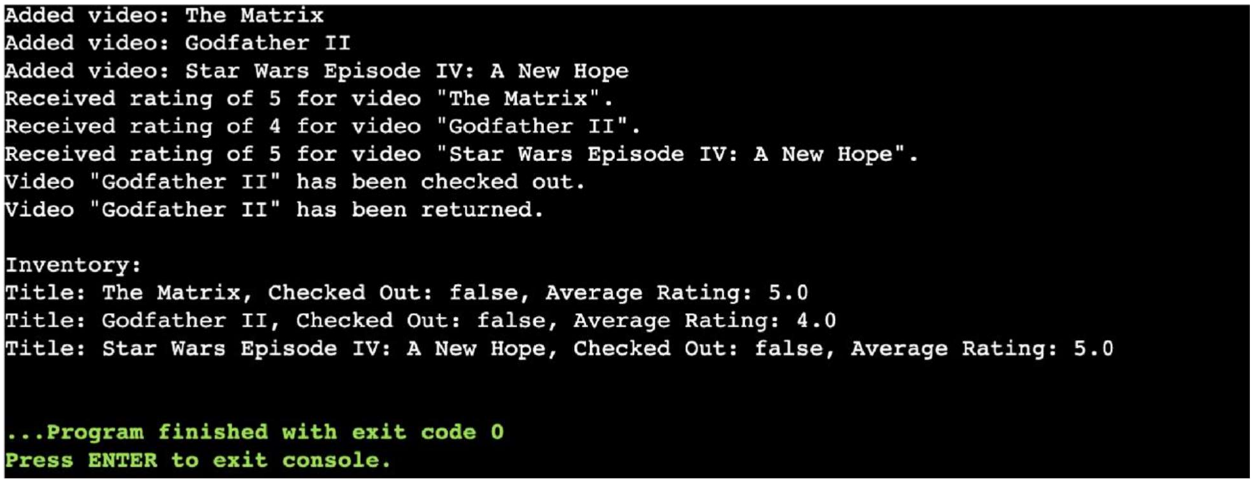
```
{ private String title; private
boolean checkedOut; private
double averageRating;
   private int ratingCount;

   public Video(String title)
{       this.title = title;
this.checkedOut = false;
this.averageRating = 0.0;
      this.ratingCount = 0;
   }

   public void checkOut()
{       if (!checkedOut)
{          checkedOut = true;
         System.out.println("Video \"" + title + "\" has been checked
out.");       } else {
         System.out.println("Video \"" + title + "\" is already checked out.");
      }
   }
```

```
    public void returnVideo()
{         if (checkedOut)
{             checkedOut = false;
            System.out.println("Video \"" + title + "\" has been
returned.");          } else {
            System.out.println("Video \"" + title + "\" was not checked out.");
        }
    }

    public void receiveRating(int rating)
{         if (rating < 1 || rating > 5) {
            System.out.println("Invalid rating. Please rate between 1 and 5.");
return;
        }
        averageRating = (averageRating * ratingCount + rating) /
(++ratingCount);
        System.out.println("Received rating of " + rating + " for video \"" + title +
"\".");
    }

     public String getTitle() {
        return title;
    }

    public boolean isCheckedOut() {
        return checkedOut;
    }

    public double getAverageRating() {
        return averageRating;
    }
}
```

2. **VideoStore Class:-** class

```
VideoStore   {   private
Video[] videos;
   private int count;

   public VideoStore(int capacity)
{        videos = new Video[capacity];
      count = 0;
   }

   public void addVideo(String title)
{        if (count < videos.length) {
         videos[count++] = new Video(title);
System.out.println("Added video: " + title);
      } else {
         System.out.println("Inventory is full. Cannot add more videos.");
      }
   }

   public void checkOut(String title)
{        Video video = findVideo(title);
      if (video != null)
{           video.checkOut();
      } else {
         System.out.println("Video \"" + title + "\" not found.");
      }
   }

   public void returnVideo(String title)
{        Video video = findVideo(title);
      if (video != null)
{           video.returnVideo();
      } else {
            System.out.println("Video \"" + title + "\" not found.");
        }
    }

    public void receiveRating(String title, int rating) {
        Video video = findVideo(title);
if (video != null) {
            video.receiveRating(rating);
        } else {
            System.out.println("Video \"" + title + "\" not found.");
        }
    }

    public void listInventory()
{        System.out.println("\nInventory:");
for (int i = 0; i < count; i++) {
            Video video = videos[i];
            System.out.println("Title: " + video.getTitle() + ", Checked Out: " +
video.isCheckedOut() +
                    ", Average Rating: " + video.getAverageRating());
        }
    }

    private Video findVideo(String title)
{        for (int i = 0; i < count; i++) {
            if (videos[i].getTitle().equalsIgnoreCase(title))
{                return videos[i];
            }
}
        return null;
    }
}
```

3. **VideoStoreLauncher Class:-**
```
public class VideoStoreLauncher { public static void main(String[] args) { VideoStore store = new VideoStore(10);

        store.addVideo("The Matrix");
store.addVideo("Godfather II");
        store.addVideo("Star Wars Episode IV: A New Hope");

        store.receiveRating("The Matrix", 5);
store.receiveRating("Godfather II", 4);
        store.receiveRating("Star Wars Episode IV: A New Hope", 5);

        store.checkOut("Godfather II");
        store.returnVideo("Godfather II");

        store.listInventory();
    }
}
```
4. **Output:**



```
Added video: The Matrix
Added video: Godfather II
Added video: Star Wars Episode IV: A New Hope
Received rating of 5 for video "The Matrix".
Received rating of 4 for video "Godfather II".
Received rating of 5 for video "Star Wars Episode IV: A New Hope".
Video "Godfather II" has been checked out.
Video "Godfather II" has been returned.

Inventory:
Title: The Matrix, Checked Out: false, Average Rating: 5.0
Title: Godfather II, Checked Out: false, Average Rating: 4.0
Title: Star Wars Episode IV: A New Hope, Checked Out: false, Average Rating: 5.0


...Program finished with exit code 0
Press ENTER to exit console.
```

5. **Learning Outcomes:**
   1. Designed a functional system to manage video rentals, demonstrating the use of classes and objects in Java.
   2. Implemented methods for operations like adding videos, renting out, returning, and recording user ratings.
   3. Applied arrays to store and efficiently manage the video inventory within the store.
   4. Learned to integrate multiple classes and enable seamless interaction among them in a structured program.
   5. Strengthened understanding of object-oriented programming concepts like encapsulation and method abstraction.

Experiment 3

Student Name: Sahil Wadhwa **UID: 22BCS10855**
Branch: CSE **Section: KRG-IOT-2(B)**
Semester: 6th **DOP:17/1/2025**
Subject: Java Lab **Subject Code: 22CSH-359**

Aim: Create an application to calculate interest for FDs, RDs based on certain conditions using inheritance.

Objective: To develop a Java application that calculates interest for Fixed Deposits (FDs) and Recurring Deposits (RDs) using object-oriented programming principles. The application will use inheritance to define common properties and methods for accounts while providing specific implementations for FDs and RDs based on their respective conditions.

Algorithm:

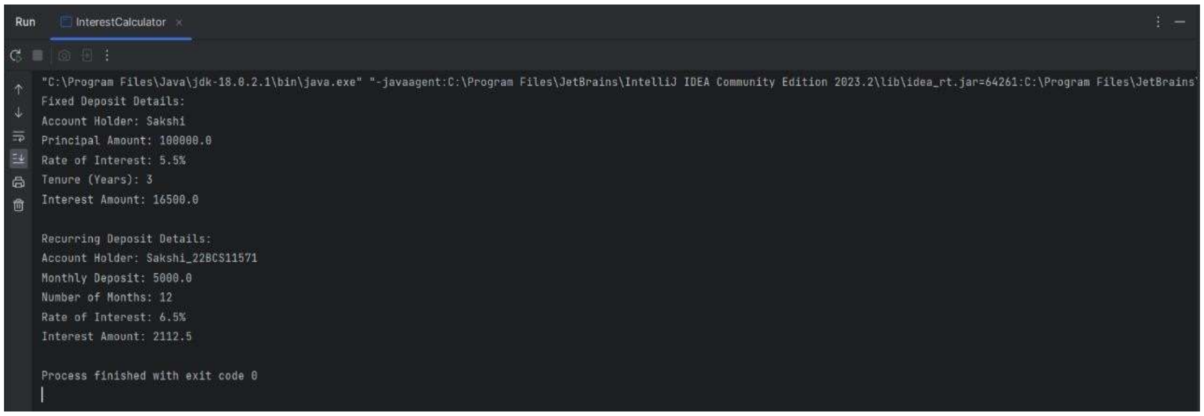
- **Create Account class** with attributes: `accountHolderName`, `principal`, `rateOfInterest`. Include methods for calculating interest (to be overridden) and displaying details.
- **Create FixedDeposit subclass** that calculates FD interest using: `principal * rateOfInterest * tenureInYears / 100`. Display FD details.
- **Create RecurringDeposit subclass** that calculates RD interest using: `(monthlyDeposit * months * (months + 1) / 2) * (rateOfInterest / (12 * 100))`. Display RD details.
- **In main method**, create instances of `FixedDeposit` and `RecurringDeposit` and display their details.

Code:

```
class Account {

String accountHolderName;

 double principal;

 double rateOfInterest;

 public Account(String accountHolderName, double principal, double rateOfInterest) {

    this.accountHolderName = accountHolderName;

    this.principal = principal;

    this.rateOfInterest = rateOfInterest;}

    public double calculateInterest() {

    return 0;

}
```

```
        public void displayDetails() {
        System.out.println("Account Holder: " + accountHolderName);
        System.out.println("Principal Amount: " + principal);
        System.out.println("Rate of Interest: " + rateOfInterest + "%");
    }
}
  class FixedDeposit extends Account {
   int tenureInYears;
   public FixedDeposit(String accountHolderName, double principal, double rateOfInterest, int
    tenureInYears) {
        super(accountHolderName, principal, rateOfInterest);
        this.tenureInYears = tenureInYears;
    }
   public double calculateInterest() {
        return principal * rateOfInterest * tenureInYears / 100;
    }
        public void displayDetails() {
        super.displayDetails();
        System.out.println("Tenure (Years): " + tenureInYears);
        System.out.println("Interest Amount: " + calculateInterest());}}
   class RecurringDeposit extends Account {
   int months;
   double monthlyDeposit;
   public RecurringDeposit(String accountHolderName, double monthlyDeposit, double rateOfInterest,
    int months) {
        super(accountHolderName, 0, rateOfInterest);
        this.monthlyDeposit = monthlyDeposit;
        this.months = months;
    }
    public double calculateInterest() {
        // RD interest formula: P(n(n+1)/2) * (r / 12 * 100)
        double n = months;
        return (monthlyDeposit * n * (n + 1) / 2) * (rateOfInterest / (12 * 100));
    }
    public void displayDetails() {
        System.out.println("Account Holder: " + accountHolderName);
        System.out.println("Monthly Deposit: " + monthlyDeposit);
        System.out.println("Number of Months: " + months);
        System.out.println("Rate of Interest: " + rateOfInterest + "%");
        System.out.println("Interest Amount: " + calculateInterest());
    }
}
public class InterestCalculator {
    public static void main(String[] args) {
        // Example FD account
        FixedDeposit fd = new FixedDeposit("Sakshi", 100000, 5.5, 3);
        System.out.println("Fixed Deposit Details:");
        fd.displayDetails();
        System.out.println();
        RecurringDeposit rd = new RecurringDeposit("Sakshi_22BCS11571", 5000, 6.5, 12);

        System.out.println("Recurring Deposit Details:");

        rd.displayDetails();

    }

}
```

Output:



```
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2\lib\idea_rt.jar=64261:C:\Program Files\JetBrains
Fixed Deposit Details:
Account Holder: Sakshi
Principal Amount: 100000.0
Rate of Interest: 5.5%
Tenure (Years): 3
Interest Amount: 16500.0

Recurring Deposit Details:
Account Holder: Sakshi_22BCS11571
Monthly Deposit: 5000.0
Number of Months: 12
Rate of Interest: 6.5%
Interest Amount: 2112.5

Process finished with exit code 0
```

Learning Outcomes:

- **Inheritance**: Use of base and derived classes for shared attributes and methods.
- **Method Overriding**: Custom implementation of methods in subclasses.
- **Constructor**: Initializing object attributes using constructors.
- **Encapsulation**: Storing and manipulating data within objects.
- **Polymorphism**: Different behavior of `calculateInterest()` based on object type.
- **Interest Calculation**: Implementing FD and RD interest formulas.
- **Class Interaction**: Creating objects and calling methods to display details.

Experiment 4

Student Name: Sahil Wadhwa **UID: 22BCS10855**
Branch: BE-CSE **Section: KRG-IOT-2(B)**
Semester:6th **Date of Performance:14/02/25**
Subject Name: PBLJ **Subject Code: 22CSH-359**

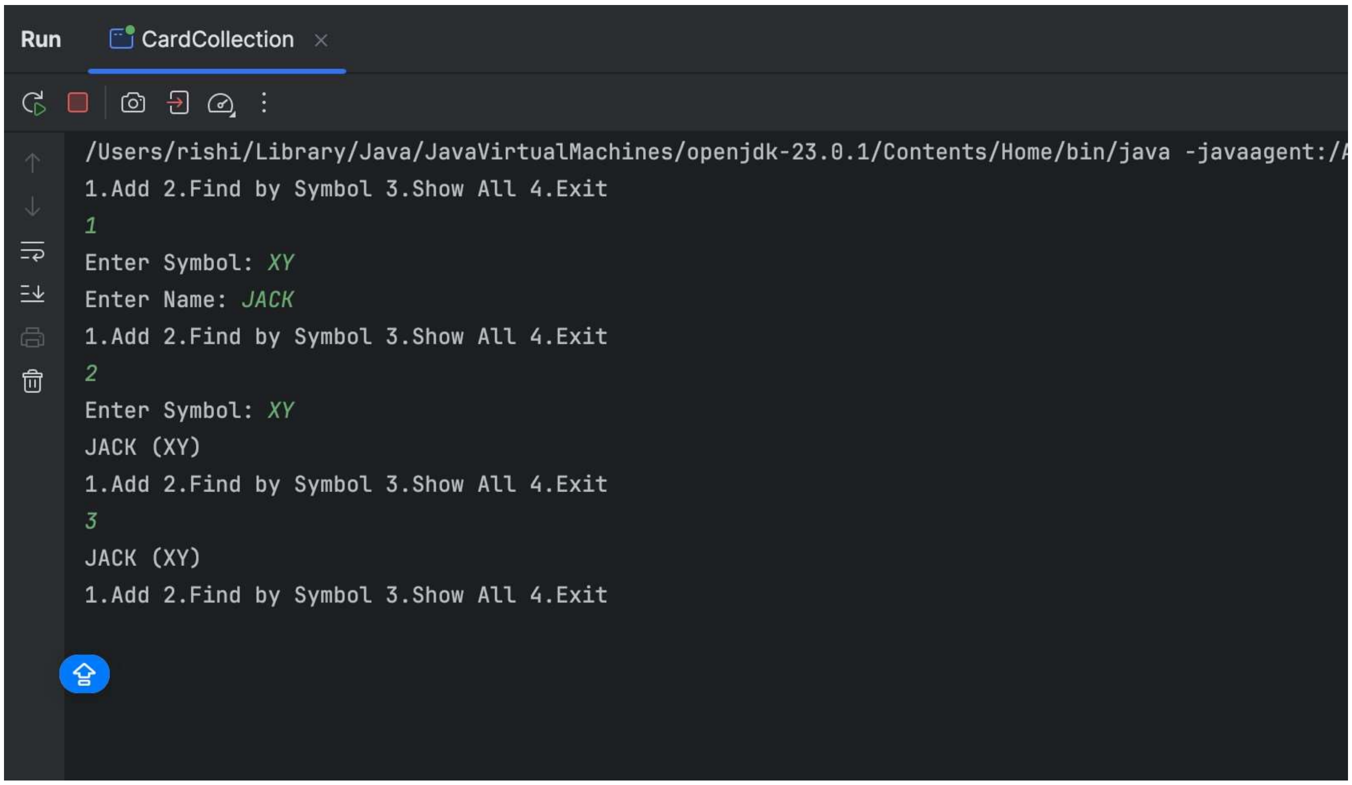
1. **Aim :** Write a program to collect and store all the cards to assist the users in finding all the cards in a given symbol.This cards game consist of N number of cards. Get N number of cards details from the user and store the values in Card object with the attributes symbol and Number. Store all the cards in a map with symbols as its key and list of cards as its value. Map is used here to easily group all the cards based on their symbol. Once all the details are captured print all the distinct symbols in alphabetical order from the Map.

2. **Objective :** This program collects and stores N cards, grouping them by symbol in a map for easy retrieval. It displays distinct symbols in alphabetical order along with their associated cards, total count, and sum of numbers, ensuring efficient organization and user-friendly output.

3. **Code**

```
import java.util.*;

class Card {
    String symbol;
    String name;

    Card(String symbol, String name) {
        this.symbol = symbol;
        this.name = name;
    }

    public String toString() {
        return name + " (" + symbol + ")";
    }
}

public class CardCollection {
    static Collection<Card> cards = new ArrayList<>();
    static Scanner sc = new Scanner(System.in);
    public static void main(String[] args) {
        while (true) {
            System.out.println("1.Add 2.Find by Symbol 3.Show All 4.Exit");
            int choice = sc.nextInt();
            switch (choice) {
                case 1 -> addCard();
                case 2 -> findBySymbol();
                case 3 -> cards.forEach(System.out::println);
                case 4 -> { return; }
                default -> System.out.println("Invalid");
            }
        }
    }

    static void addCard() {
        System.out.print("Enter Symbol: ");
        String symbol = sc.next();
        sc.nextLine();
        System.out.print("Enter Name: ");
        String name = sc.nextLine();
        cards.add(new Card(symbol, name));
    }

    static void findBySymbol() {
        System.out.print("Enter Symbol: ");
        String symbol = sc.next();
        cards.stream().filter(c ->
c.symbol.equals(symbol)).forEach(System.out::println);
    }
}
```

4. Code



```
/Users/rishi/Library/Java/JavaVirtualMachines/openjdk-23.0.1/Contents/Home/bin/java -javaagent:/
1.Add 2.Find by Symbol 3.Show All 4.Exit
1
Enter Symbol: XY
Enter Name: JACK
1.Add 2.Find by Symbol 3.Show All 4.Exit
2
Enter Symbol: XY
JACK (XY)
1.Add 2.Find by Symbol 3.Show All 4.Exit
3
JACK (XY)
1.Add 2.Find by Symbol 3.Show All 4.Exit
```

5. Learning Outcomes

- Understand how to use maps (dictionaries) for efficient data storage and retrieval.
- Learn to group and organize data based on a key attribute.
- Gain experience in handling user input and storing objects dynamically.
- Develop skills in sorting and displaying structured data in a meaningful

Experiment 5.1

Student Name: Sahil Wadhwa **UID: 22BCS10855**
Branch: CSE **Section: KRG IOT-2(B)**
Semester: 6
Subject: PBLJ **Subject Code:22CSH-359**

Aim: Write a Java program to calculate the sum of a list of integers using autoboxing and unboxing. Include methods to parse strings into their respective wrapper classes (e.g., Integer.parseInt()).

Objective: Demonstrate **autoboxing** and **unboxing** in Java by converting string numbers into Integer objects, storing them in a list, and computing their sum.

Algorithm:

Step 1: Initialize the Program

1. Start the program.
2. Import ArrayList and List classes.
3. Define the AutoboxingExample class.

Step 2: Convert String Array to Integer List

1. Define the method parseStringArrayToIntegers(String[] strings).
2. Create an empty ArrayList<Integer>.
3. Iterate through the string array:
   - Convert each string to an Integer using Integer.parseInt(str).
   - Add the integer to the list (**autoboxing** happens here).
4. Return the list of integers.

Step 3: Calculate the Sum of Integers

1. Define the method calculateSum(List<Integer> numbers).
2. Initialize a variable sum to 0.
3. Iterate through the list:
   - Extract each integer (**unboxing** happens here).
   - Add it to sum.
4. Return the total sum.

Step 4: Execute Main Function

1. Define main(String[] args).
2. Create a string array with numeric values.
3. Call parseStringArrayToIntegers() to convert it into a list of integers.
4. Call calculateSum() to compute the sum.
5. Print the result.

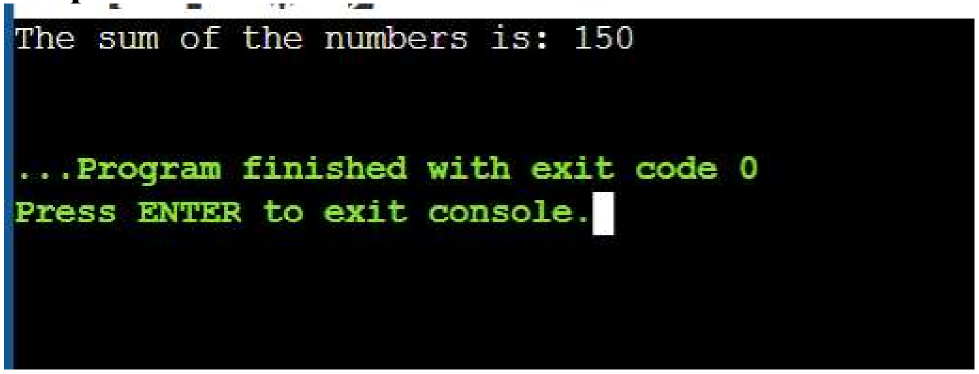
Step 5: Terminate the Program

1. End the execution.

Code:

```
import java.util.ArrayList;
import java.util.List;

public class AutoboxingExample {
    public static void main(String[] args) {
        String[] numberStrings = {"10", "20", "30", "40", "50"};

        List<Integer> numbers = parseStringArrayToIntegers(numberStrings);

        int sum = calculateSum(numbers);

        System.out.println("The sum of the numbers is: " + sum);
    }

    public static List<Integer> parseStringArrayToIntegers(String[] strings) {
        List<Integer> integerList = new ArrayList<>();
        for (String str : strings) {
            integerList.add(Integer.parseInt(str));
        }
        return integerList;
    }

    public static int calculateSum(List<Integer> numbers) {
        int sum = 0;
        for (Integer num : numbers) {
            sum += num;
        }
        return sum;
    }
}
```

Output:



```
The sum of the numbers is: 150


...Program finished with exit code 0
Press ENTER to exit console.
```

Learning Outcomes:

- Understand the concept of **autoboxing and unboxing** in Java and how primitive types are automatically converted to their wrapper classes and vice versa.
- Learn how to **convert string values into Integer objects** using Integer.parseInt() and store them in a list.
- Gain experience in **working with ArrayLists** to store and manipulate a collection of numbers dynamically.
- Develop proficiency in **iterating through collections** and performing arithmetic operations like summation.

Experiment 5.2

1. **Aim:** Create a Java program to serialize and deserialize a Student object.
The program should:

- Serialize a Student object (containing id, name, and GPA) and save it to a file.
- Deserialize the object from the file and display the student details.
- Handle FileNotFoundException, IOException, and ClassNotFoundException using exception handling.

2. **Objective:** The objective is to serialize and deserialize a Student object, store and retrieve its id, name, and GPA from a file, and handle exceptions like FileNotFoundException, IOException, and ClassNotFoundException.

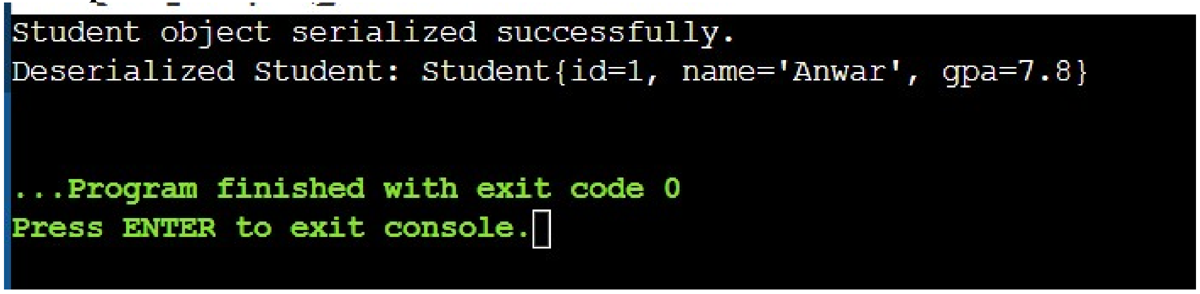
3. Algorithm:

```
Step 1: Initialize the Program
    1. Start the program.
    2. Import the necessary classes (java.io.*).
    3. Define a Student class implementing Serializable.
    4. Declare attributes:
        o id (int)
        o name (String)
        o gpa (double)
    5. Define a constructor to initialize Student objects.
    6. Override toString() to display student details.
Step 2: Define the Serialization Method
    1. Create serializeStudent(Student student).
    2. Use a try-with-resources block to create an ObjectOutputStream:
        o Open a FileOutputStream to write to student.ser.
        o Write the Student object to the file using writeObject().
    3. Handle exceptions:
        o FileNotFoundException → Print error message.
        o IOException → Print error message.
    4. Print a success message if serialization is successful.
Step 3: Define the Deserialization Method
    1. Create deserializeStudent().
    2. Use a try-with-resources block to create an ObjectInputStream:
        o Open a FileInputStream to read student.ser.
        o Read the Student object using readObject().
    3. Handle exceptions:
        o FileNotFoundException → Print error message.
        o IOException → Print error message.
        o ClassNotFoundException → Print error message.
    4. Print the deserialized student details.
Step 4: Execute Main Function
    1. Define main(String[] args).
    2. Create a Student object with sample data.
    3. Call serializeStudent() to save the object.
    4. Call deserializeStudent() to read and display the object.
Step 5: Terminate the Program
    1. End execution.
```

4. Implementation Code:

```
import java.io.*;

class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
    private double gpa;

    public Student(int id, String name, double gpa) {
        this.id = id;
        this.name = name;
        this.gpa = gpa;
    }

    @Override
    public String toString() {
        return "Student{id=" + id + ", name='" + name + "', gpa=" + gpa + "}";
    }
}

public class StudentSerialization {
    private static final String FILE_NAME = "student.ser";

    public static void main(String[] args) {
        Student student = new Student(1, "Anwar", 7.8);
        serializeStudent(student);
        deserializeStudent();
    }

    public static void serializeStudent(Student student) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
            oos.writeObject(student);
            System.out.println("Student object serialized successfully.");
        } catch (FileNotFoundException e) {
            System.err.println("File not found: " + e.getMessage());
        } catch (IOException e) {
            System.err.println("IOException occurred: " + e.getMessage());
        }
    }

    public static void deserializeStudent() {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME)))
{
            Student student = (Student) ois.readObject();
            System.out.println("Deserialized Student: " + student);
        } catch (FileNotFoundException e) {
            System.err.println("File not found: " + e.getMessage());
        } catch (IOException e) {
            System.err.println("IOException occurred: " + e.getMessage());
        } catch (ClassNotFoundException e) {
            System.err.println("Class not found: " + e.getMessage());
        }
    }
}
```

5. Output



```
Student object serialized successfully.
Deserialized Student: Student{id=1, name='Anwar', gpa=7.8}


...Program finished with exit code 0
Press ENTER to exit console.
```

6. Learning Outcomes:

- Understand object serialization and deserialization in Java.
- Learn how to use ObjectOutputStream and ObjectInputStream for file operations.
- Implement exception handling for FileNotFoundException, IOException, and ClassNotFoundException.
- Gain hands-on experience in storing and retrieving objects from a file.
- Develop skills in data persistence and file management using Java.

Experiment 5.3

1. **Aim:** Create a menu-based Java application with the following options.
   1. Add an Employee
   2. Display All
   3. Exit If option 1 is selected, the application should gather details of the employee like employee name, employee id, designation and salary and store it in a file. If option 2 is selected, the application should display all the employee details. If option 3 is selected the application should exit.
2. **Objective:** The objective is to develop a menu-based Java application that allows users to **add employee details**, **store them in a file**, and **display all stored employee records**, with an option to exit the program.

3. **Algorithm:**

Step 1: Initialize the Program
1. Start the program.
2. Import java.util.* and java.util.concurrent.* for thread handling.
3. Define a class TicketBookingSystem with:
   - A List<Boolean> representing seat availability (true for available, false for booked).
   - A synchronized method bookSeat(int seatNumber, String passengerName) to ensure thread safety.

Step 2: Implement Seat Booking Logic
1. Define bookSeat(int seatNumber, String passengerName):
   - If the seat is available (true), mark it as booked (false).
   - Print confirmation: "Seat X booked successfully by Y".
   - If already booked, print: "Seat X is already booked."

Step 3: Define Booking Threads
1. Create a class PassengerThread extending Thread:
   - Store passenger name, seat number, and booking system reference.
   - Implement run() method to call bookSeat().

Step 4: Assign Thread Priorities
1. Create VIP and Regular passenger threads.
2. Set higher priority for VIP passengers using setPriority(Thread.MAX_PRIORITY).
3. Set default priority for regular passengers.

Step 5: Handle User Input & Simulate Booking
1. In main(), create an instance of TicketBookingSystem.
2. Accept number of seats and bookings from the user.
3. Create multiple PassengerThread instances for VIP and regular passengers.
4. Start all threads using start().

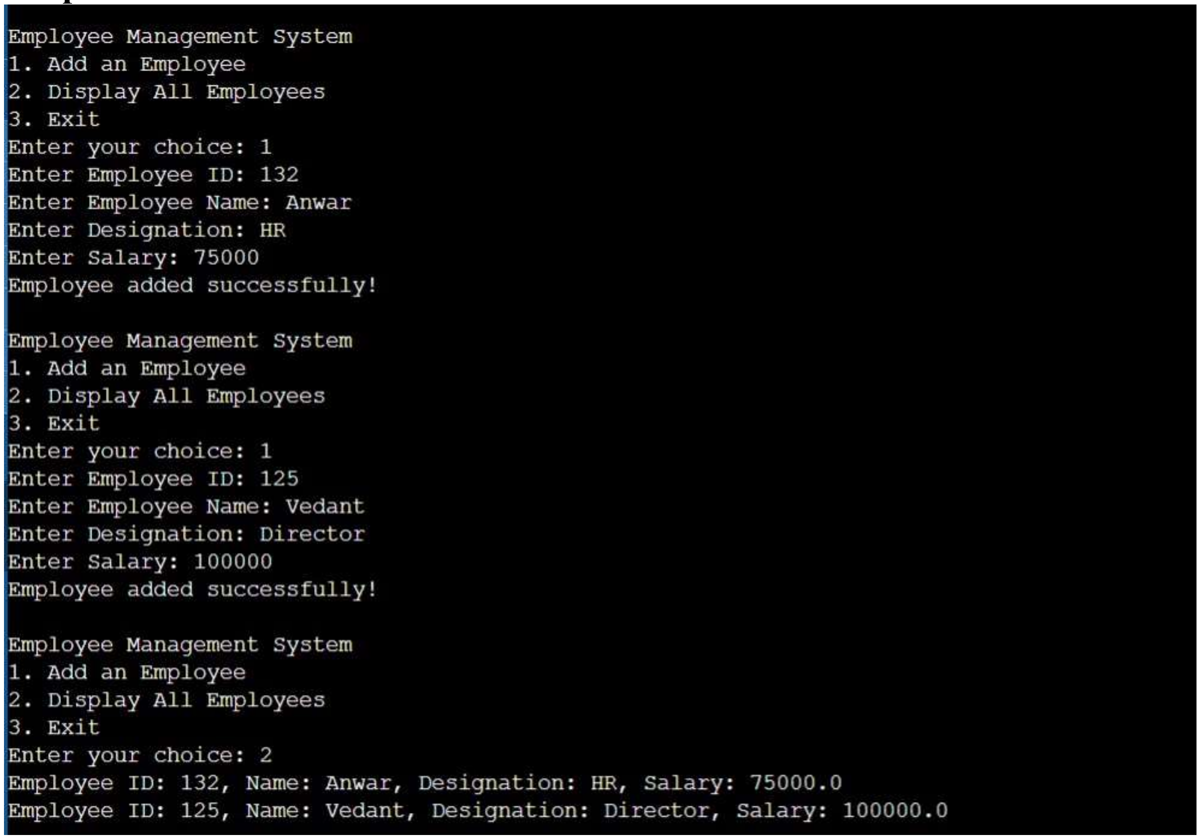
Step 6: Synchronization & Preventing Double Booking
1. Use the synchronized keyword in bookSeat() to ensure only one thread accesses it at a time.
2. Ensure thread execution order by assigning higher priority to VIP threads.

Step 7: Display Final Booking Status
1. After all threads finish execution, display the list of booked seats.
2. End the program with a message: "All bookings completed successfully."

4. **Implementation Code:**

```
i import java.io.*;
import java.util.*;

class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
    private String designation;
    private double salary;

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

    @Override
    public String toString() {
            return "Employee ID: " + id + ", Name: " + name + ", Designation: " + designation
+ ", Salary: " + salary;
    }
}

public class EmployeeManagementSystem {
    private static final String FILE_NAME = "employees.ser";
    private static List<Employee> employees = new ArrayList<>();

    public static void addEmployee() {
            Scanner scanner = new Scanner(System.in);
            System.out.print("Enter Employee ID: ");
            int id = scanner.nextInt();
            scanner.nextLine();
            System.out.print("Enter Employee Name: ");
            String name = scanner.nextLine();
            System.out.print("Enter Designation: ");
            String designation = scanner.nextLine();
            System.out.print("Enter Salary: ");
            double salary = scanner.nextDouble();

            Employee employee = new Employee(id, name, designation, salary);
            employees.add(employee);
            saveEmployees();
            System.out.println("Employee added successfully!");
    }

    public static void displayAllEmployees() {
            loadEmployees();
            if (employees.isEmpty()) {
                    System.out.println("No employees found.");
            } else {
                    for (Employee employee : employees) {
                            System.out.println(employee);
                    }
            }
    }
    private static void saveEmployees() {
            try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
                    oos.writeObject(employees);
            } catch (IOException e) {
                    System.err.println("Error saving employees: " + e.getMessage());
            }
    }

    @SuppressWarnings("unchecked")
    private static void loadEmployees() {
            try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(FILE_NAME))) {
                    employees = (List<Employee>) ois.readObject();
            } catch (FileNotFoundException e) {
                    employees = new ArrayList<>();
            } catch (IOException | ClassNotFoundException e) {
                    System.err.println("Error loading employees: " + e.getMessage());
            }
    }

    public static void main(String[] args) {
            Scanner scanner = new Scanner(System.in);
            while (true) {
                    System.out.println("\nEmployee Management System");
                    System.out.println("1. Add an Employee");
                    System.out.println("2. Display All Employees");
                    System.out.println("3. Exit");
                    System.out.print("Enter your choice: ");
                    int choice = scanner.nextInt();
                    scanner.nextLine();

                    switch (choice) {
                    case 1:
                            addEmployee();
                            break;
                    case 2:
                            displayAllEmployees();
                            break;
                    case 3:
                            System.out.println("Exiting...");
                            return;
                    default:
                            System.out.println("Invalid choice! Please try again.");
                    }
            }
    }
}
```

5. Output:



```
Employee Management System
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 132
Enter Employee Name: Anwar
Enter Designation: HR
Enter Salary: 75000
Employee added successfully!

Employee Management System
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 125
Enter Employee Name: Vedant
Enter Designation: Director
Enter Salary: 100000
Employee added successfully!

Employee Management System
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 2
Employee ID: 132, Name: Anwar, Designation: HR, Salary: 75000.0
Employee ID: 125, Name: Vedant, Designation: Director, Salary: 100000.0
```

6. Learning Outcomes:

- Understand file handling and serialization in Java to store and retrieve objects persistently.
- Learn how to implement a menu-driven console application using loops and conditional statements.
- Gain experience in object-oriented programming (OOP) by defining and managing Employee objects.
- Practice exception handling to manage file-related errors like FileNotFoundException and IOException.
- Develop skills in list manipulation and user input handling using ArrayList and Scanner.

Experiment 6

Student Name: Sahil Wadhwa **UID:22BCS10855**
Branch: CSE **Section/Group: KRG-IOT-2(B)**
Semester: 6 **Date of Performance:**
Subject Name: Java with Lab **Subject Code: 22CSH-359**

1. **Aim:** Develop Java programs using lambda expressions and stream operations for sorting, filtering, and processing large datasets efficiently.

2. **Objective:**

• Develop Java programs using lambda expressions and stream operations for sorting, filtering, and processing large datasets efficiently.

• Implement easy, medium, and hard-level tasks involving sorting employees, filtering and sorting students, and processing products using streams.

3. **Implementation/Code:**

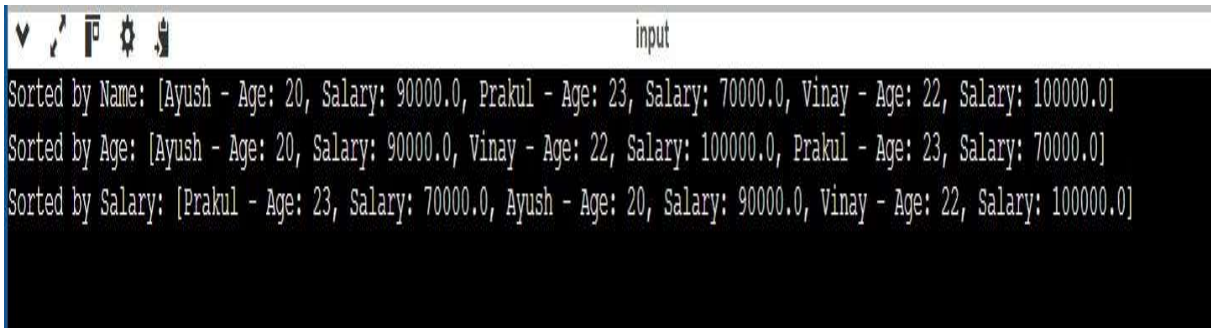
a.
```
import java.util.*;
class Employee {
    String name;
    int age;
    double salary;
    Employee(String name, int age, double salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }
    @Override
    public String toString() {
        return name + " - Age: " + age + ", Salary: " + salary;
    }
}
public class EmployeeSort {
    public static void main(String[] args) {
        List<Employee> employees = Arrays.asList(
            new Employee("Ayush", 20, 90000),
            new Employee("Vinay", 22, 100000),
            new Employee("Prakul", 23, 70000)
        );
        employees.sort(Comparator.comparing(emp -> emp.name));
        System.out.println("Sorted by Name: " + employees);
        employees.sort(Comparator.comparingInt(emp -> emp.age));
        System.out.println("Sorted by Age: " + employees);
        employees.sort(Comparator.comparingDouble(emp -> emp.salary));
        System.out.println("Sorted by Salary: " + employees);
    }
}
```

b. import java.util.*;

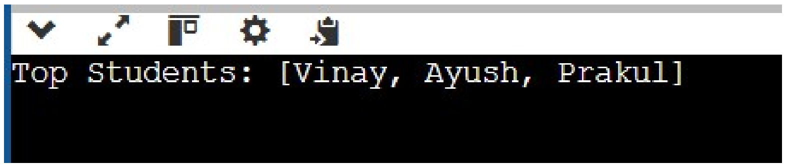
```
import java.util.stream.Collectors;
class Student {
    private String name;
    private double marks;
    public Student(String name, double marks) {
        this.name = name;
        this.marks = marks;
    }
    public String getName() {
        return name;
    }
    public double getMarks() {
        return marks;
    }
}
public class StudentFilter {
    public static void main(String[] args) {
        List<Student> students = List.of(
            new Student("Ayush", 85),
            new Student("Rajeev", 70),
            new Student("Vinay", 90),
            new Student("David", 60),
            new Student("Prakul", 80)
        );
        List<String> topStudents = students.stream()
            .filter(s -> s.getMarks() > 75)
            .sorted(Comparator.comparingDouble(Student::getMarks).reversed())
            .map(Student::getName)
            .collect(Collectors.toList());
        System.out.println("Top Students: " + topStudents);
    }
}
```

c.
```
import java.util.*;
import java.util.stream.Collectors;
class Product {
    String name;
    String category;
    double price;
    public Product(String name, String category, double price) {
        this.name = name;
        this.category = category;
        this.price = price;
    }
    @Override
    public String toString() {
        return name + " ($" + price + ")";
    }
}
public class ProductProcessor {
    public static void main(String[] args) {
        List<Product> products = Arrays.asList(
            new Product("Laptop", "Electronics", 1200),
            new Product("Phone", "Electronics", 800),
            new Product("TV", "Electronics", 1500),
            new Product("Shirt", "Clothing", 50),
            new Product("Jeans", "Clothing", 70),
            new Product("Blender", "Appliances", 200),
            new Product("Toaster", "Appliances", 100)
        );
        Map<String, List<Product>> productsByCategory = products.stream()
            .collect(Collectors.groupingBy(p -> p.category));
        System.out.println("Products grouped by category:");
        productsByCategory.forEach((category, productList) ->
            System.out.println(category + ": " + productList));
        Map<String, Optional<Product>> mostExpensiveByCategory = products.stream()
            .collect(Collectors.groupingBy(
                p -> p.category,
                Collectors.maxBy(Comparator.comparingDouble(p -> p.price))
            ));
        System.out.println("\nMost expensive product in each category:");
        mostExpensiveByCategory.forEach((category, product) ->
            System.out.println(category + ": " + product.orElse(null)));
        double averagePrice = products.stream()
            .mapToDouble(p -> p.price)
            .average()
            .orElse(0);
        System.out.println("\nAverage price of all products: $" + averagePrice);
    }
}
```

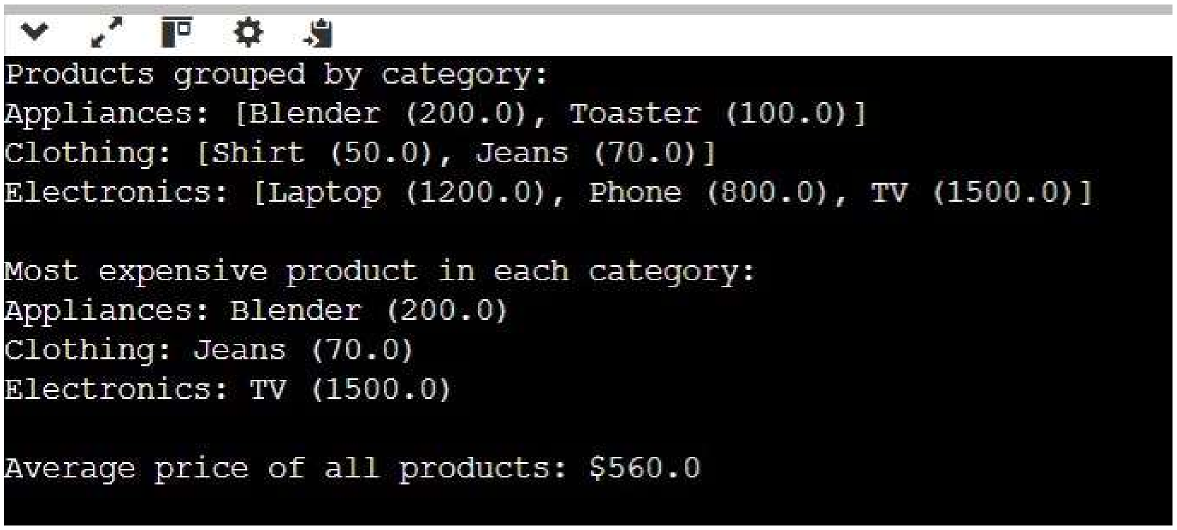
4. Output:



```
Sorted by Name: [Ayush - Age: 20, Salary: 90000.0, Prakul - Age: 23, Salary: 70000.0, Vinay - Age: 22, Salary: 100000.0]
Sorted by Age: [Ayush - Age: 20, Salary: 90000.0, Vinay - Age: 22, Salary: 100000.0, Prakul - Age: 23, Salary: 70000.0]
Sorted by Salary: [Prakul - Age: 23, Salary: 70000.0, Ayush - Age: 20, Salary: 90000.0, Vinay - Age: 22, Salary: 100000.0]
```



```
Top Students: [Vinay, Ayush, Prakul]
```



```
Products grouped by category:
Appliances: [Blender (200.0), Toaster (100.0)]
Clothing: [Shirt (50.0), Jeans (70.0)]
Electronics: [Laptop (1200.0), Phone (800.0), TV (1500.0)]

Most expensive product in each category:
Appliances: Blender (200.0)
Clothing: Jeans (70.0)
Electronics: TV (1500.0)

Average price of all products: $560.0
```

5. Learning Outcome:

- Understand and implement **lambda expressions** for sorting objects in a list based on different attributes.
- Utilize **Java Streams API** to perform operations like **filtering, sorting, and mapping** efficiently on large datasets.
- Learn **Comparator and method references** to simplify object comparisons for sorting.
- Apply **grouping and aggregation functions** using Collectors.groupingBy() and Collectors.maxBy() for processing categorized data.
- Gain hands-on experience in computing **statistical values** like the **average** from a dataset using mapToDouble() and average().
- Improve **code efficiency and readability** by using **functional programming** techniques in Java.

Experiment -7

Student Name: Sahil Wadhwa **UID:22BCS10855**
Branch: BE-CSE **Section/Group: KRG-IOT-2(B)**
Semester: 6 **Date of Performance:17/03/2025**
Subject : Project Based Learning in Java with Lab **Subject Code: 22CSH-359**

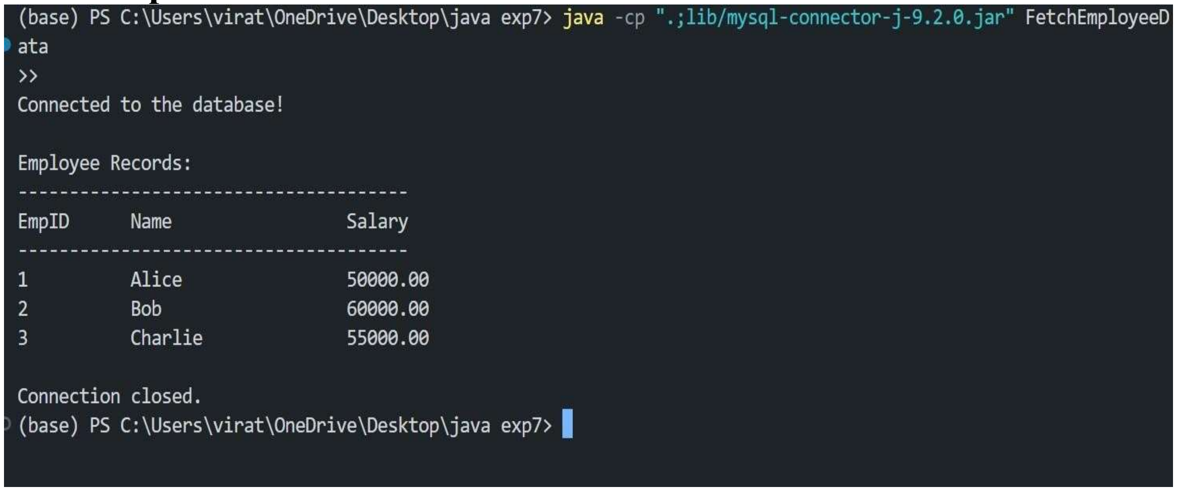
7.1.1.Aim: Create a Java program to connect to a MySQL database and fetch data from a single table. The program should: Use DriverManager and Connection objects. Retrieve and display all records from a table named Employee with columns EmpID, Name, and Salary..

7.1.2 Objective: To develop a Java program that connects to a MySQL database, retrieves data from the Employee table, and displays all records, demonstrating basic JDBC connectivity and data retrieval operations.

7.1.3 Code:

```
import java.sql.*;

public class FetchEmployeeData {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/testdb";
        String user = "root";
        String password = "password";

        String query = "SELECT EmpID, Name, Salary FROM Employee";

        try {
            // Load MySQL JDBC driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Establish connection
            Connection con = DriverManager.getConnection(url, user, password);
            System.out.println("Connected to the database!");

            // Create statement and execute query
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(query);

            // Display results
            System.out.println("\nEmployee Records:");
            System.out.println("------------------------------------------");
            System.out.printf("%-10s %-20s %-10s%n", "EmpID", "Name", "Salary");
            System.out.println("------------------------------------------");

            while (rs.next()) {
                int empID = rs.getInt("EmpID");
                String name = rs.getString("Name");
                double salary = rs.getDouble("Salary");

                System.out.printf("%-10d %-20s %-10.2f%n", empID, name, salary);
            }

            // Close resources
            rs.close();
            stmt.close();
            con.close();
            System.out.println("\nConnection closed.");

        } catch (ClassNotFoundException e) {
            System.out.println("MySQL Driver not found: " + e.getMessage());
        } catch (SQLException e) {
            System.out.println("SQL Error: " + e.getMessage());
        }
    }
}
```

7.1.4 Output:



```
(base) PS C:\Users\virat\OneDrive\Desktop\java exp7> java -cp ".;lib/mysql-connector-j-9.2.0.jar" FetchEmployeeD
ata
>>
Connected to the database!

Employee Records:
--------------------------------------
EmpID      Name                 Salary
--------------------------------------
1          Alice                50000.00
2          Bob                  60000.00
3          Charlie              55000.00

Connection closed.
(base) PS C:\Users\virat\OneDrive\Desktop\java exp7>
```

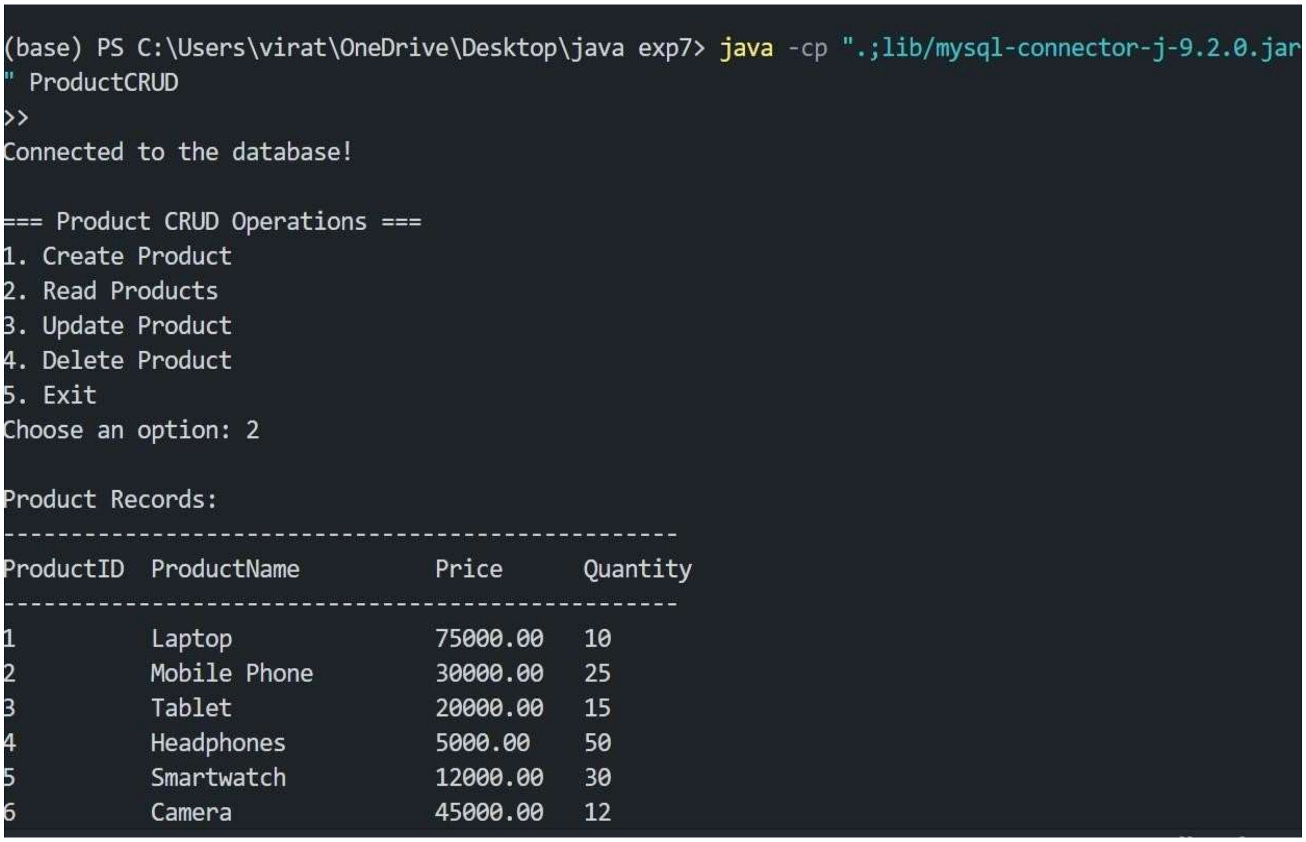
7.2.1 Aim: Build a program to perform CRUD operations (Create, Read, Update, Delete) on a database table Product with columns: ProductID, ProductName, Price, and Quantity. The program should include: Menu-driven options for each operation. Transaction handling to ensure data integrity.

7.2.2 Objective: To develop a Java program that connects to a MySQL database and performs CRUD operations (Create, Read, Update, Delete) on the Product table. The program ensures data integrity by using transaction handling and provides a menu-driven interface for user-friendly interaction.

7.2.3 Code:

```
import java.sql.*;
import java.util.Scanner;

public class ProductCRUD {

    private static final String URL = "jdbc:mysql://localhost:3306/ProductDB";
    private static final String USER = "root";
    private static final String PASSWORD = "password";

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD)) {
            Class.forName("com.mysql.cj.jdbc.Driver");
            System.out.println("Connected to the database!");

            boolean exit = false;

            while (!exit) {
                System.out.println("\n=== Product CRUD Operations ===");
                System.out.println("1. Create Product");
                System.out.println("2. Read Products");
                System.out.println("3. Update Product");
                System.out.println("4. Delete Product");
                System.out.println("5. Exit");
                System.out.print("Choose an option: ");

                int choice = scanner.nextInt();
                scanner.nextLine();
                switch (choice) {
                    case 1 -> createProduct(conn, scanner);
                    case 2 -> readProducts(conn);
                    case 3 -> updateProduct(conn, scanner);
                    case 4 -> deleteProduct(conn, scanner);
                    case 5 -> exit = true;
                    default -> System.out.println("Invalid option. Try again.");
                }
            }

        } catch (ClassNotFoundException e) {
            System.out.println("MySQL Driver not found: " + e.getMessage());
        } catch (SQLException e) {
            System.out.println("SQL Error: " + e.getMessage());
        }

        scanner.close();
    }

    private static void createProduct(Connection conn, Scanner scanner) throws SQLException {
        System.out.print("Enter product name: ");
        String name = scanner.nextLine();
        System.out.print("Enter price: ");
        double price = scanner.nextDouble();
        System.out.print("Enter quantity: ");
        int quantity = scanner.nextInt();

        String query = "INSERT INTO Product (ProductName, Price, Quantity) VALUES (?, ?,
?)";

        try (PreparedStatement pstmt = conn.prepareStatement(query)) {
            conn.setAutoCommit(false);

            pstmt.setString(1, name);
            pstmt.setDouble(2, price);
            pstmt.setInt(3, quantity);

            int rows = pstmt.executeUpdate();
            conn.commit();

            System.out.println(rows + " product(s) inserted successfully!");
        } catch (SQLException e) {
            conn.rollback();
            System.out.println("Transaction rolled back due to error: " + e.getMessage());
        } finally {
            conn.setAutoCommit(true);
        }
    }

    private static void readProducts(Connection conn) throws SQLException {
        String query = "SELECT * FROM Product";

        try (Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(query)) {

            System.out.println("\nProduct Records:");
            System.out.println(" -------------------------------------------------- ");
            System.out.printf("%-10s %-20s %-10s %-10s%n", "ProductID", "ProductName",
"Price", "Quantity");
            System.out.println("-----------------------------------------------------");

            while (rs.next()) {
                int id = rs.getInt("ProductID");
                String name = rs.getString("ProductName");
                double price = rs.getDouble("Price");
                int quantity = rs.getInt("Quantity");

                System.out.printf("%-10d %-20s %-10.2f %-10d%n", id, name, price, quantity);
            }
        }
    }


    private static void updateProduct(Connection conn, Scanner scanner) throws SQLException
{
        System.out.print("Enter product ID to update: ");
        int id = scanner.nextInt();
        scanner.nextLine();

        System.out.print("Enter new name: ");
        String name = scanner.nextLine();
        System.out.print("Enter new price: ");
        double price = scanner.nextDouble();
        System.out.print("Enter new quantity: ");
        int quantity = scanner.nextInt();

        String query = "UPDATE Product SET ProductName = ?, Price = ?, Quantity = ? WHERE
    ProductID = ?";

        try (PreparedStatement pstmt = conn.prepareStatement(query)) {
            conn.setAutoCommit(false);

            pstmt.setString(1, name);
            pstmt.setDouble(2, price);
            pstmt.setInt(3, quantity);
            pstmt.setInt(4, id);

            int rows = pstmt.executeUpdate();
            conn.commit();

            System.out.println(rows + " product(s) updated successfully!");

        } catch (SQLException e) {
            conn.rollback();
            System.out.println("Transaction rolled back due to error: " + e.getMessage());
        } finally {
            conn.setAutoCommit(true);
        }
    }

    private static void deleteProduct(Connection conn, Scanner scanner) throws SQLException {
        System.out.print("Enter product ID to delete: ");
        int id = scanner.nextInt();

        String query = "DELETE FROM Product WHERE ProductID = ?";

        try (PreparedStatement pstmt = conn.prepareStatement(query)) {
            conn.setAutoCommit(false);

            pstmt.setInt(1, id);
            int rows = pstmt.executeUpdate();
            conn.commit();

            System.out.println(rows + " product(s) deleted successfully!");
        } catch (SQLException e) {
            conn.rollback();
            System.out.println("Transaction rolled back due to error: " + e.getMessage());
        } finally {
            conn.setAutoCommit(true);
        }
    }
}
```

7.2.4 Output:



```
(base) PS C:\Users\virat\OneDrive\Desktop\java exp7> java -cp ".;lib/mysql-connector-j-9.2.0.jar
" ProductCRUD
>>
Connected to the database!

=== Product CRUD Operations ===
1. Create Product
2. Read Products
3. Update Product
4. Delete Product
5. Exit
Choose an option: 2

Product Records:
--------------------------------------------------
ProductID  ProductName         Price      Quantity
--------------------------------------------------
1          Laptop              75000.00   10
2          Mobile Phone        30000.00   25
3          Tablet              20000.00   15
4          Headphones          5000.00    50
5          Smartwatch          12000.00   30
6          Camera              45000.00   12
```

7.3.1 Aim: Develop a Java application using JDBC and MVC architecture to manage student data. The application should: Use a Student class as the model with fields like StudentID, Name, Department, and Marks. Include a database table to store student data. Allow the user to perform CRUD operations through a simple menu-driven view. Implement database operations in a separate controller class.

7.3.2 Objective: The objective of this program is to develop a menu-driven Java application that allows users to add employee details, display all stored employees, and exit the program. Employee details, including ID, name, designation, and salary, are stored persistently in a file using serialization.

7.3.3 Code:

StudentController.java

```
package controller;

import model.Student;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class StudentController {

    private static final String URL = "jdbc:mysql://localhost:3306/StudentDB";
    private static final String USER = "root";
    private static final String PASSWORD = "rishuraman1@V";

    // Method to create a new student
    public void createStudent(Student student) throws SQLException {
        String query = "INSERT INTO Student (Name, Department, Marks) VALUES (?, ?, ?)";

        try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
             PreparedStatement pstmt = conn.prepareStatement(query)) {

            pstmt.setString(1, student.getName());
            pstmt.setString(2, student.getDepartment());
            pstmt.setDouble(3, student.getMarks());

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

    // Method to retrieve all students
    public List<Student> getAllStudents() throws SQLException {
        List<Student> students = new ArrayList<>();
        String query = "SELECT * FROM Student";

        try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(query)) {

            while (rs.next()) {
                students.add(new Student(
                        rs.getInt("StudentID"),
                        rs.getString("Name"),
                        rs.getString("Department"),
                        rs.getDouble("Marks")
                ));
            }
        }
        return students;
    }

    // Method to update student data
    public void updateStudent(Student student) throws SQLException {
        String query = "UPDATE Student SET Name = ?, Department = ?, Marks = ? WHERE
StudentID = ?";

        try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
             PreparedStatement pstmt = conn.prepareStatement(query)) {

            pstmt.setString(1, student.getName());
            pstmt.setString(2, student.getDepartment());
            pstmt.setDouble(3, student.getMarks());
            pstmt.setInt(4, student.getStudentID());

            int rows = pstmt.executeUpdate();
            if (rows > 0) {
                System.out.println("Student updated successfully!");
            } else {
                System.out.println("Student not found.");
            }
        }
    }
    // Method to delete a student
    public void deleteStudent(int studentID) throws SQLException {
        String query = "DELETE FROM Student WHERE StudentID = ?";

        try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
             PreparedStatement pstmt = conn.prepareStatement(query)) {

            pstmt.setInt(1, studentID);
            int rows = pstmt.executeUpdate();
            if (rows > 0) {
                System.out.println("Student deleted successfully!");
            } else {
                System.out.println("Student not found.");
            }
        }
    }
}
```

Student.java

```
package model;

public class Student {
    private int studentID;
    private String name;
    private String department;
    private double marks;

    public Student(int studentID, String name, String department, double marks) {
        this.studentID = studentID;
        this.name = name;
        this.department = department;
        this.marks = marks;
    }

    // Getters and Setters
    public int getStudentID() {
        return studentID;
    }

    public void setStudentID(int studentID) {
        this.studentID = studentID;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDepartment() {
        return department;
    }

    public void setDepartment(String department) {
        this.department = department;
    }

    public double getMarks() {
        return marks;
    }

    public void setMarks(double marks) {
        this.marks = marks;
    }

    @Override
    public String toString() {
        return String.format("ID: %d, Name: %s, Dept: %s, Marks: %.2f",
                studentID, name, department, marks);
    }
}
```

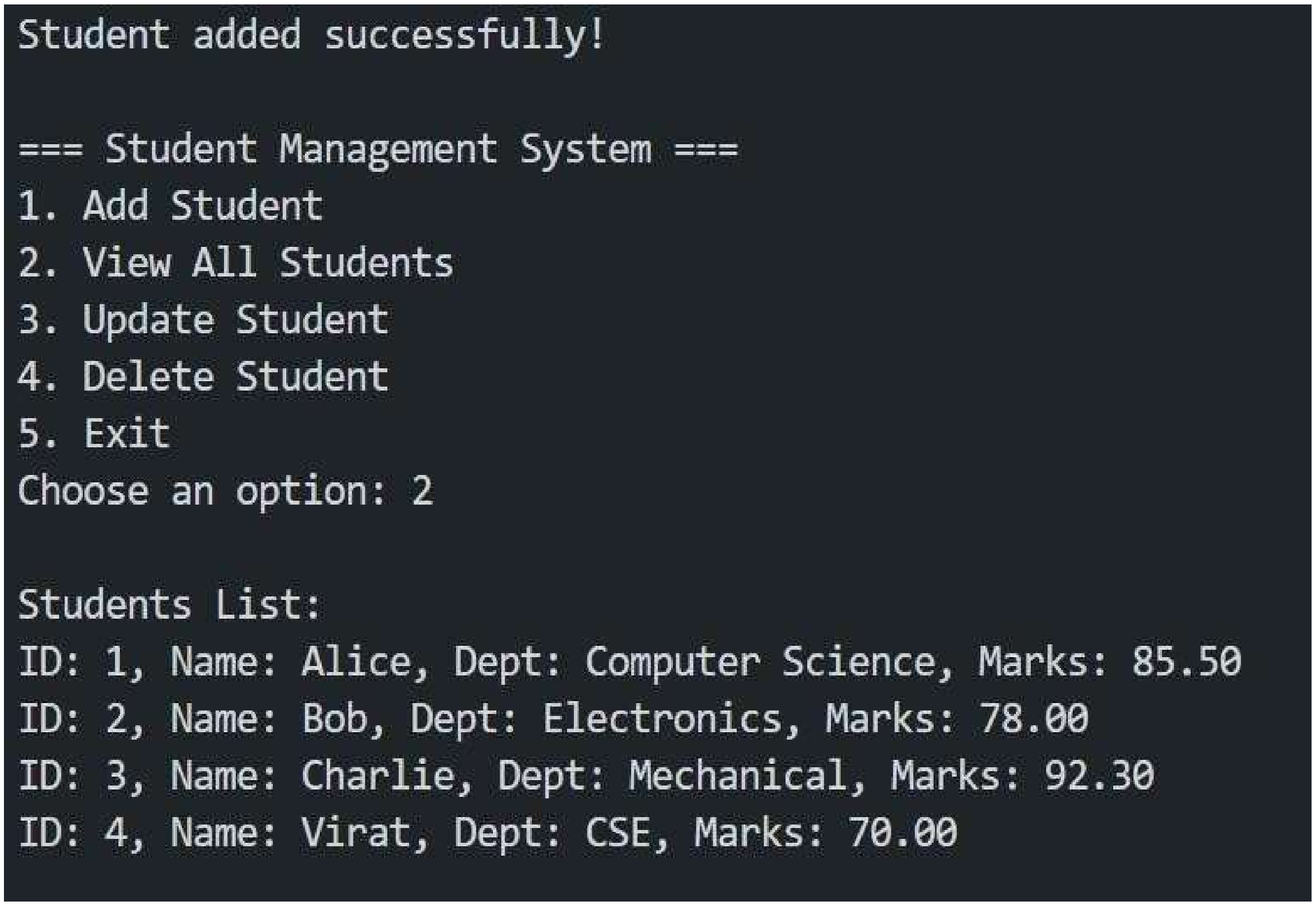
StudentView.java

```
package view;

import controller.StudentController;
import model.Student;

import java.util.List;
import java.util.Scanner;

public class StudentView {

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

    public void displayMenu() {
        boolean exit = false;

        while (!exit) {
            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");
            System.out.println("4. Delete Student");
            System.out.println("5. Exit");
            System.out.print("Choose an option: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            try {
                switch (choice) {
                    case 1 -> addStudent();
                    case 2 -> viewStudents();
                    case 3 -> updateStudent();
                    case 4 -> deleteStudent();
                    case 5 -> exit = true;
                    default -> System.out.println("Invalid option. Try again.");
                }
            } catch (Exception e) {
                System.out.println("Error: " + e.getMessage());
            }
        }
        scanner.close();
    }
    private void addStudent() throws Exception {
        System.out.print("Enter name: ");
        String name = scanner.nextLine();
        System.out.print("Enter department: ");
        String department = scanner.nextLine();
        System.out.print("Enter marks: ");
        double marks = scanner.nextDouble();

        Student student = new Student(0, name, department, marks);
        controller.createStudent(student);
    }
    private void viewStudents() throws Exception {
        List<Student> students = controller.getAllStudents();
        System.out.println("\nStudents List:");
        for (Student student : students) {
            System.out.println(student);
        }
    }
    private void updateStudent() throws Exception {
        System.out.print("Enter student ID to update: ");
        int id = scanner.nextInt();
        scanner.nextLine();

        System.out.print("Enter new name: ");
        String name = scanner.nextLine();
        System.out.print("Enter new department: ");
        String department = scanner.nextLine();
        System.out.print("Enter new marks: ");
        double marks = scanner.nextDouble();

        Student student = new Student(id, name, department, marks);
        controller.updateStudent(student);
    }

    private void deleteStudent() throws Exception {
        System.out.print("Enter student ID to delete: ");
        int id = scanner.nextInt();
        controller.deleteStudent(id);
    }
}
```

MainApp.java

```
import view.StudentView;

public class MainApp {
    public static void main(String[] args) {
        StudentView view = new StudentView();
        view.displayMenu();}}
```

7.3.4 Output:



```
Student added successfully!

=== Student Management System ===
1. Add Student
2. View All Students
3. Update Student
4. Delete Student
5. Exit
Choose an option: 2

Students List:
ID: 1, Name: Alice, Dept: Computer Science, Marks: 85.50
ID: 2, Name: Bob, Dept: Electronics, Marks: 78.00
ID: 3, Name: Charlie, Dept: Mechanical, Marks: 92.30
ID: 4, Name: Virat, Dept: CSE, Marks: 70.00
```

Learning Outcomes:

1. Understanding JDBC Integration**:**Gained practical experience in integrating JDBC with a Java application for database connectivity.
2. MVC Architecture Implementation**:**Learned how to implement the Model-View-Controller (MVC) architecture in Java for better code organization and separation of concerns.
3. Database CRUD Operations:Acquired the ability to perform CRUD operations (Create, Read, Update, Delete) using SQL queries in Java applications.
4. Transaction Handling**:**Understood the importance of transaction handling for maintaining data integrity during database operations.

Experiment-8

Student Name: Sahil Wadhwa Branch: BE-CSE
UID: 22BCS10855
Section/Group: KRG-IOT-2(B)
Semester: 6TH
Date of Performance:03/04/25
Subject Name: Project Based Learning in Java
Subject Code: 22CSH-359

1. **Aim:**

a. Write a servlet to accept user credentials through an HTML form and display a personalized welcome message if the login is successful.
b. Create a servlet integrated with JDBC to display a list of employees from a database. Include a search form to fetch employee details by ID.
c. Develop a JSP-based student portal. Include a form for entering attendance details and save them to the database using a servlet.

2. **Objective:** The objective is to develop web applications using Servlets and JSP for user input handling, database integration.

3. **Implementation/Code:**

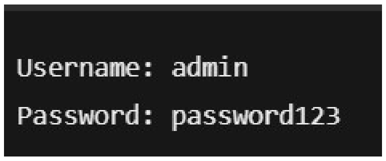
a) EASY LEVEL

HTML code:

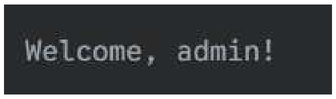
```
<!DOCTYPE html>
<html>
<head>
   <title>Login Page</title>
</head>
<body>
   <h2>Login</h2>
   <form action="LoginServlet" method="post">
      <label>Username:</label>
      <input type="text" name="username" required><br><br>

      <label>Password:</label>
      <input type="password" name="password" required><br><br>

      <input type="submit" value="Login">
   </form>
</body>
</html>
```

Java code:

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
       response.setContentType("text/html");
       PrintWriter out = response.getWriter();

       // Retrieve username and password
       String username = request.getParameter("username");
       String password = request.getParameter("password");

       // Hardcoded credentials for validation (Replace with DB authentication)
       if ("admin".equals(username) && "password123".equals(password)) {
           out.println("<h2>Welcome, " + username + "!</h2>");
       } else {
           out.println("<h2>Invalid Username or Password</h2>");
       }
       out.close();
   }
}
```



```
Username: admin
Password: password123
```



```
Welcome, admin!
```

(a)

b) MEDIUM LEVEL

Sql code:

```
CREATE DATABASE CompanyDB;
USE CompanyDB;

CREATE TABLE employees (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    position VARCHAR(100),
    salary DECIMAL(10,2)
);

INSERT INTO employees (name, position, salary) VALUES
('Alice Johnson', 'Software Engineer', 75000.00),
('Bob Smith', 'Manager', 90000.00),
('Charlie Brown', 'Analyst', 65000.00);
```

Java code:

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/EmployeeServlet")
public class EmployeeServlet extends HttpServlet {
    private static final String JDBC_URL =
        "jdbc:mysql://localhost:3306/CompanyDB";
    private static final String JDBC_USER = "root"; // Change as per your MySQL
        setup
    private static final String JDBC_PASS = "password"; // Change accordingly

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
```

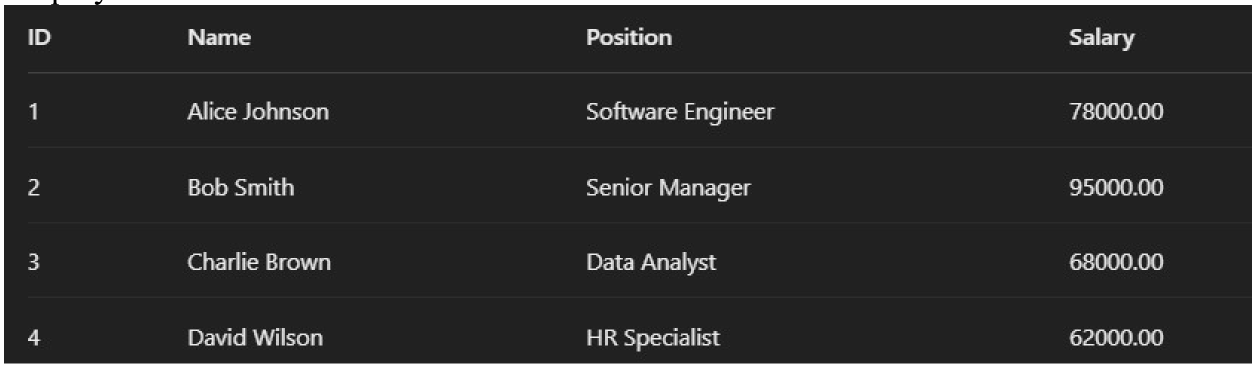
```
response.setContentType("text/html");
PrintWriter out = response.getWriter();

try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    Connection conn = DriverManager.getConnection(JDBC_URL,
JDBC_USER, JDBC_PASS);

    String searchId = request.getParameter("id");
    String query = "SELECT * FROM employees";

    if (searchId != null && !searchId.isEmpty()) {
        query += " WHERE id = ?";
    }

    PreparedStatement stmt = conn.prepareStatement(query);

    if (searchId != null && !searchId.isEmpty()) {
        stmt.setInt(1, Integer.parseInt(searchId));
    }

    ResultSet rs = stmt.executeQuery();

    out.println("<html><head><title>Employee List</title></head><body>");
    out.println("<h2>Employee Details</h2>");
    out.println("<form action='EmployeeServlet' method='GET'>");
    out.println("Search by ID: <input type='text' name='id'/> <input type='submit'
value='Search'/>");
    out.println("</form><br>");

    out.println("<table
border='1'><tr><th>ID</th><th>Name</th><th>Position</th><th>Salary</th>
</tr>");

    boolean found = false;
    while (rs.next()) {
        found = true;
        out.println("<tr><td>" + rs.getInt("id") + "</td>");
        out.println("<td>" + rs.getString("name") + "</td>");
        out.println("<td>" + rs.getString("position") + "</td>");
```

```
                out.println("<td>" + rs.getDouble("salary") + "</td></tr>");
            }

            if (!found) {
                out.println("<tr><td colspan='4'>No Employee Found</td></tr>");
            }

            out.println("</table></body></html>");

            rs.close();
            stmt.close();
            conn.close();
        } catch (Exception e) {
            out.println("<h3>Error: " + e.getMessage() + "</h3>");
        }
    }
}
```

XML code:

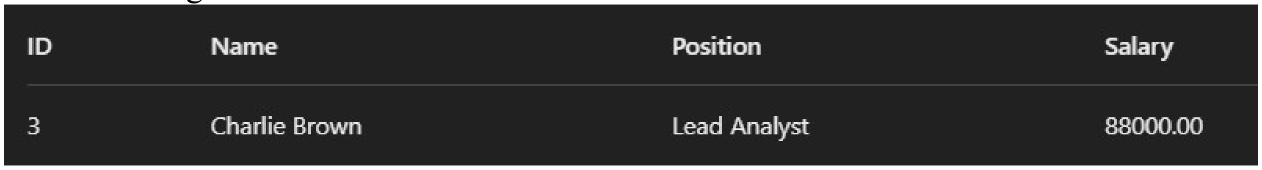
```
<web-app>
    <servlet>
        <servlet-name>EmployeeServlet</servlet-name>
        <servlet-class>EmployeeServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>EmployeeServlet</servlet-name>
        <url-pattern>/EmployeeServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

Employees List



ID	Name	Position	Salary
1	Alice Johnson	Software Engineer	78000.00
2	Bob Smith	Senior Manager	95000.00
3	Charlie Brown	Data Analyst	68000.00
4	David Wilson	HR Specialist	62000.00

ID 2 searching



ID	Name	Position	Salary
3	Charlie Brown	Lead Analyst	88000.00

(b)

c) HARD LEVEL

Sql code:

```
CREATE DATABASE StudentDB;
USE StudentDB;

CREATE TABLE student_attendance (
   id INT PRIMARY KEY AUTO_INCREMENT,
   student_name VARCHAR(100) NOT NULL,
   roll_number VARCHAR(20) UNIQUE NOT NULL,
   attendance_status ENUM('Present', 'Absent') NOT NULL,
   date DATE NOT NULL
);
```

Java code:

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/AttendanceServlet")
public class AttendanceServlet extends HttpServlet {
   private static final String JDBC_URL = "jdbc:mysql://localhost:3306/StudentDB";
   private static final String JDBC_USER = "root"; // Change as per your MySQL
      setup
   private static final String JDBC_PASS = "password"; // Change accordingly
```

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    String name = request.getParameter("studentName");
    String rollNumber = request.getParameter("rollNumber");
    String status = request.getParameter("attendanceStatus");
    String date = request.getParameter("date");

    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection conn = DriverManager.getConnection(JDBC_URL,
    JDBC_USER, JDBC_PASS);

        String query = "INSERT INTO student_attendance (student_name,
    roll_number, attendance_status, date) VALUES (?, ?, ?, ?)";
        PreparedStatement stmt = conn.prepareStatement(query);
        stmt.setString(1, name);
        stmt.setString(2, rollNumber);
        stmt.setString(3, status);
        stmt.setString(4, date);

        int rows = stmt.executeUpdate();
        if (rows > 0) {
            out.println("<h3>Attendance recorded successfully!</h3>");
        }

        stmt.close();
        conn.close();
    } catch (Exception e) {
        out.println("<h3>Error: " + e.getMessage() + "</h3>");
    }

    out.println("<br><a href='attendance.jsp'>Back to Attendance Form</a>");
}

protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
```

```
        PrintWriter out = response.getWriter();

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection conn = DriverManager.getConnection(JDBC_URL,
        JDBC_USER, JDBC_PASS);

            String query = "SELECT * FROM student_attendance";
            PreparedStatement stmt = conn.prepareStatement(query);
            ResultSet rs = stmt.executeQuery();

            out.println("<h2>Student Attendance Records</h2>");
            out.println("<table border='1'><tr><th>ID</th><th>Name</th><th>Roll
        Number</th><th>Status</th><th>Date</th></tr>");

            while (rs.next()) {
                out.println("<tr><td>" + rs.getInt("id") + "</td>");
                out.println("<td>" + rs.getString("student_name") + "</td>");
                out.println("<td>" + rs.getString("roll_number") + "</td>");
                out.println("<td>" + rs.getString("attendance_status") + "</td>");
                out.println("<td>" + rs.getString("date") + "</td></tr>");
            }

            out.println("</table>");
            out.println("<br><a href='attendance.jsp'>Back to Attendance Form</a>");

            rs.close();
            stmt.close();
            conn.close();
        } catch (Exception e) {
            out.println("<h3>Error: " + e.getMessage() + "</h3>");
        }
    }
}
```

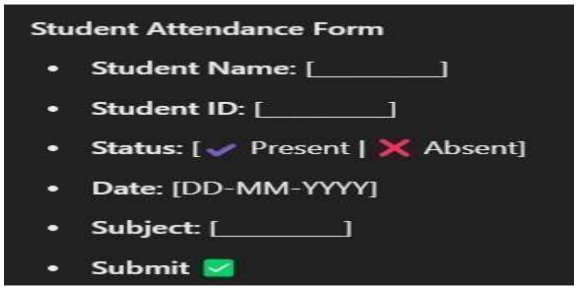
XML code:

```
<web-app>
    <servlet>
        <servlet-name>AttendanceServlet</servlet-name>
        <servlet-class>AttendanceServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>AttendanceServlet</servlet-name>
        <url-pattern>/AttendanceServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

JSP code:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <title>Student Attendance Portal</title>
</head>
<body>
    <h2>Enter Attendance Details</h2>
    <form action="AttendanceServlet" method="post">
        Name: <input type="text" name="studentName" required /><br><br>
        Roll Number: <input type="text" name="rollNumber" required /><br><br>
        Attendance:
        <select name="attendanceStatus">
            <option value="Present">Present</option>
            <option value="Absent">Absent</option>
        </select><br><br>
        Date: <input type="date" name="date" required /><br><br>
        <input type="submit" value="Submit Attendance">
    </form>

    <h3><a href="AttendanceServlet">View Attendance Records</a></h3>
</body>
</html>
```

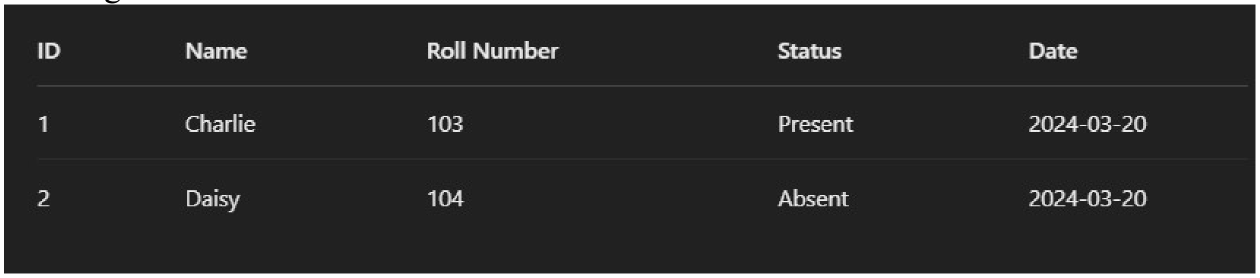
Attendance form



Student Attendance Form

- Student Name: [__________]
- Student ID: [__________]
- Status: [✓ Present | ✗ Absent]
- Date: [DD-MM-YYYY]
- Subject: [__________]
- Submit ✅

Viewing Attendance



ID	Name	Roll Number	Status	Date
1	Charlie	103	Present	2024-03-20
2	Daisy	104	Absent	2024-03-20

(c)

4. Learning Outcome:

- Servlet and JDBC Integration: Understanding how to connect a Java Servlet to a MySQL database.
- Handling HTTP Requests: Learning how to process GET and POST requests to retrieve and display data.
- Database Query Execution: Writing SQL queries in Java to fetch records dynamically.
- Form Handling & User Input: Implementing a search feature to filter employee records.
- Deploying on Tomcat: Deploying a Java web application using Apache Tomcat.
- Error Handling in JDBC: Managing SQL exceptions and debugging database connectivity issues.

Experiment-9

Student Name: Sahil Wadhwa **UID: 22BCS10855**
Branch: BE-CSE **Section/Group:** KRG-IOT-2(B)
Semester: 6TH **Date of Performance:**10/04/25
Subject Name: Project Based Learning in Java **Subject Code:** 22CSH-359

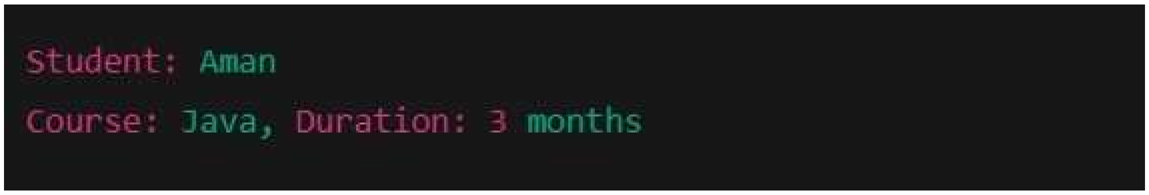
9.1.1 Aim: To demonstrate dependency injection using Spring Framework with Java-based configuration.

9.1.2 Objective:

Define Course and Student classes.
Use Configuration and Bean annotations to inject dependencies.
Load Spring context and print student details.

9.1.3 Code:
```
// Course.java
public class Course {
    private String courseName; private
    String duration;

    public Course(String courseName, String duration) { this.courseName =
        courseName;
        this.duration = duration;
    }

    public String getCourseName() { return courseName; } public
    String getDuration() { return duration; }

    @Override
    public String toString() {
        return "Course: " + courseName + ", Duration: " + duration;
    }
}

// Student.java
public class Student { private
    String name; private Course
    course;
    public Student(String name, Course course) {
        this.name = name;
        this.course = course;
    }

    public void showDetails() { System.out.println("Student: " +
        name); System.out.println(course);
    }
}// AppConfig.java
import org.springframework.context.annotation.*;

@Configuration
public class AppConfig { @Bean
    public Course course() {
        return new Course("Java", "3 months");
    }

    @Bean
    public Student student() {
        return new Student("Aman", course());
    }
}// MainApp.java
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class MainApp {
    public static void main(String[] args) { ApplicationContext
        context = new
AnnotationConfigApplicationContext(AppConfig.class); Student
        student = context.getBean(Student.class);
        student.showDetails();
    }
}
```

Output:



```
Student: Aman
Course: Java, Duration: 3 months
```

9.2.1 **Aim:** To perform CRUD operations on a Student entity using Hibernate ORM with MySQL.

Objective: Define Course and Student classes.

Use Configuration and Bean annotations to inject dependencies. Load Spring

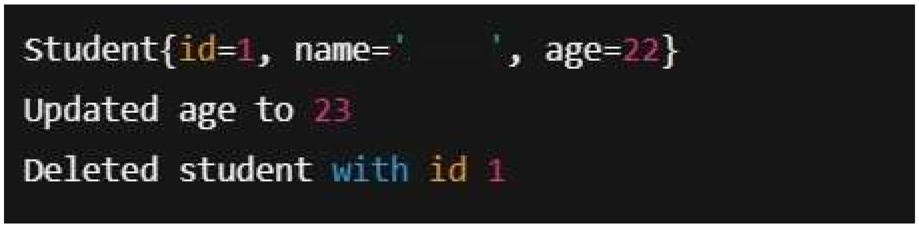
context and print student details.

9.2.2 Code:

```
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/testdb</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">password</property>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <mapping class="Student"/>
    </session-factory>
</hibernate-configuration> import

javax.persistence.*;

Entity
public class Student { Id
    GeneratedValue(strategy = GenerationType.IDENTITY) private int id;
    private String name; private
    int age;

    public Student() {}
    public Student(String name, int age) { this.name =
        name;
        this.age = age;
    }
    // Getters, setters, toString
}
import org.hibernate.SessionFactory; import
org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static final SessionFactory sessionFactory;

    static {
        sessionFactory = new Configuration().configure().buildSessionFactory();
    }

    public static SessionFactory getSessionFactory() { return
        sessionFactory;
    }
}

import org.hibernate.*; public

class MainCRUD {
    public static void main(String[] args) {
        Session session = HibernateUtil.getSessionFactory().openSession();

        // Create
        Transaction tx = session.beginTransaction();
        Student s1 = new Student("Aman", 22);
        session.save(s1);
        tx.commit();

        // Read
        Student student = session.get(Student.class, 1);
        System.out.println(student);

        // Update
        tx = session.beginTransaction();
        student.setAge(23);
        session.update(student); tx.commit();

        // Delete
        tx = session.beginTransaction(); session.delete(student);
```

```
            tx.commit();

            session.close();
        }
    }
```

9.2.3 Output:



```
Student{id=1, name='[redacted]', age=22}
Updated age to 23
Deleted student with id 1
```

9.3.1 Aim: To implement a banking system using Spring and Hibernate that ensures transaction consistency during fund transfers.

Objective:

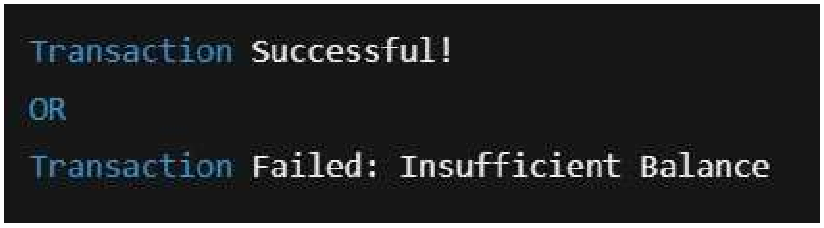
Integrate Spring + Hibernate.

Handle transactions atomically (rollback on failure). Demonstrate success and failure cases.

Code:

```
import javax.persistence.*;

Entity
public class Account { @Id
    private int accountId; private
    String holderName; private double
    balance;

    // Constructors, getters, setters
}

import javax.persistence.*; import
java.util.Date;

@Entity
public class BankTransaction { @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY) private int
    txnId;
    private int fromAcc; private
    int toAcc; private double
    amount;
    private Date txnDate = new Date();

    // Constructors, getters, setters
}
import org.hibernate.*;
import org.springframework.transaction.annotation.Transactional;

public class BankService {
    private SessionFactory sessionFactory;
    public BankService(SessionFactory sessionFactory) { this.sessionFactory
        = sessionFactory;
    }

    @Transactional
    public void transferMoney(int fromId, int toId, double amount) { Session session =
        sessionFactory.getCurrentSession();

        Account from = session.get(Account.class, fromId); Account to =
        session.get(Account.class, toId);

        if (from.getBalance() < amount) {
            throw new RuntimeException("Insufficient Balance");
        }

        from.setBalance(from.getBalance() - amount);
        to.setBalance(to.getBalance() + amount);

        session.update(from);
        session.update(to);

        BankTransaction txn = new BankTransaction(fromId, toId, amount); session.save(txn);
    }
}
@Configuration
@EnableTransactionManagement public
class AppConfig {
    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource ds = new DriverManagerDataSource();
        ds.setDriverClassName("com.mysql.cj.jdbc.Driver");
        ds.setUrl("jdbc:mysql://localhost:3306/testdb"); ds.setUsername("root");
        ds.setPassword("password");
        return ds;
    }

    @Bean
    public LocalSessionFactoryBean sessionFactory() { LocalSessionFactoryBean lsf = new
        LocalSessionFactoryBean(); lsf.setDataSource(dataSource());
        lsf.setPackagesToScan("your.package");
        Properties props = new Properties();
        props.put("hibernate.dialect", "org.hibernate.dialect.MySQL8Dialect");
        props.put("hibernate.hbm2ddl.auto", "update"); lsf.setHibernateProperties(props);
        return lsf;
    }

    @Bean
    public HibernateTransactionManager transactionManager(SessionFactory sf) { return new
        HibernateTransactionManager(sf);
    }

    @Bean
    public BankService bankService(SessionFactory sf) { return new
        BankService(sf);
    }
}

public class MainApp {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext ctx = new
AnnotationConfigApplicationContext(AppConfig.class);
        BankService service = ctx.getBean(BankService.class);

        try {
            service.transferMoney(101, 102, 500);
            System.out.println("Transaction Successful!");
        } catch (Exception e) {
            System.out.println("Transaction Failed: " + e.getMessage());
        }
        ctx.close();
    }
}
```

OUTPUT



```
Transaction Successful!
OR
Transaction Failed: Insufficient Balance
```

Learning Outcomes

- Demonstrated Dependency Injection using Spring with Java-based configuration via @Bean and @Configuration.
- Performed CRUD operations on Student entity using Hibernate ORM with MySQL database.
- Integrated Spring + Hibernate for seamless object-relational mapping and dependency management.
- Implemented transaction management using @Transactional to ensure atomicity in fund transfers.
- Handled transaction failures and rollbacks (e.g., insufficient balance) to maintain data consistency.