Experiment 1.1

Student Name: Alok Dangwal **UID: 22BCS11817**
Branch: CSE **Section: KRG_IOT_3_B**
Semester: 6th **Date of Performance: 10/01/2025**
Subject: Java **Subject Code:22CSH-359**

1. **AIM:** Create an application to save the employee information using arrays.

2. OBJECTIVE:

- Store Employee Information: Implement functionality to store employee details such as ID, Name, Salary, and Department using an array.
- Add and Display Employee Data: Provide options to add new employee data and display the list of employees in a readable format.
- Limit and Manage Employee Records: Restrict the number of employees based on a predefined array size, and allow the user to manage the records by adding or viewing employee details.

3. IMPLEMENTATION/CODE:

```
import java.util.Scanner;
class EmployeeManagement {
static class Employee {
int id;
String name;
double salary;
String department;
Employee(int id, String name, double salary, String department) {
this.id = id;
this.name = name;
this.salary = salary;
this.department = department;
}
}
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter the number of employees: ");
int totalEmployees = scanner.nextInt();
scanner.nextLine();
Employee[] employees = new Employee[totalEmployees];
int count = 0;
while (true) {
System.out.println("\nMenu:");
System.out.println("1. Add Employee");
System.out.println("2. Display Employees");
```

```
System.out.println("3. Exit");
System.out.print("Choose an option: ");
int choice = scanner.nextInt();
scanner.nextLine();
if (choice == 1) {
if (count < totalEmployees) {
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 Employee Salary: ");
double salary = scanner.nextDouble();
scanner.nextLine();
System.out.print("Enter Employee Department: ");
String department = scanner.nextLine();
employees[count] = new Employee(id, name, salary, department);
count++;
System.out.println("Employee added successfully!");
} else {
System.out.println("Employee list is full. Cannot add more employees.");
}
} else if (choice == 2) {
if (count == 0) {
System.out.println("No employees to display.");
} else {
System.out.println("\nEmployee Information:");
System.out.println("ID     Name            Salary      Department");
for (int i = 0; i < count; i++) {
Employee e = employees[i];
System.out.println(e.id + "     " + e.name + "       " + e.salary + "      " +
e.department);
}
}
} else if (choice == 3) {
System.out.println("Exiting the program. Goodbye!");
break;
} else {
System.out.println("Invalid choice. Please try again.");
}
}

scanner.close();
}
}
```

4. **OUTPUT**:

```
Enter the number of employees: 1

Menu:
1. Add Employee
2. Display Employees
3. Exit
Choose an option: 1
Enter Employee ID: 101
Enter Employee Name: Archish
Enter Employee Salary: 6000
Enter Employee Department: cse
Employee added successfully!

Menu:
1. Add Employee
2. Display Employees
3. Exit
Choose an option: 2

Employee Information:
ID     Name              Salary       Department
101      Archish         6000.0       cse

Menu:
1. Add Employee
2. Display Employees
3. Exit
Choose an option: 3
Exiting the program. Goodbye!


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

5. LEARNING OUTCOMES:

- **Learned about** creating and using arrays to store structured data in Java.
- **Learned about** managing user input and handling basic menu-driven interactions.
- **Learned about** defining and using nested classes in Java to structure related data.
- **Learned about** basic control flow, such as loops and conditionals, for interactive program design.

Experiment 2

Student Name: Alok Dangwal **UID:22BCS11817**
Branch: BE-CSE **Section/Group:KRG_IOT_3_B**
Semester:6th **Date of Performance:20.01.2025**
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: Alok Dangwal **UID: 22BCS11817**
Branch: BE-CSE **Section/Group: 22BCS_IOT-638**
Semester: 6th **Date of Performance: 24/01/25**
Subject Name: Project-Based Learning in Java **Subject Code: 22CSH-359**

1. **Aim:** Design and implement a simple inventory control system for a small video rental store

2. **Objective:** Calculate interest based on the type of the account and the status of the account holder. The rates of interest changes according to the amount (greater than or less than 1 crore), age of account holder (General or Senior citizen) and number of days if the type of account is FD or RD.

3. **Implementation/Code:**

```
abstract class Account {
protected double interestRate;
protected double amount;

        public Account(double amount) {
this.amount = amount;
        }

            abstract double calculateInterest();
}


  class FDAccount extends Account {
  private int noOfDays;
  private String accountHolderType; // "General" or "Senior"

  public FDAccount(double amount, int noOfDays, String accountHolderType) {
super(amount);      this.noOfDays = noOfDays;
    this.accountHolderType = accountHolderType;
setInterestRate();
  }
```

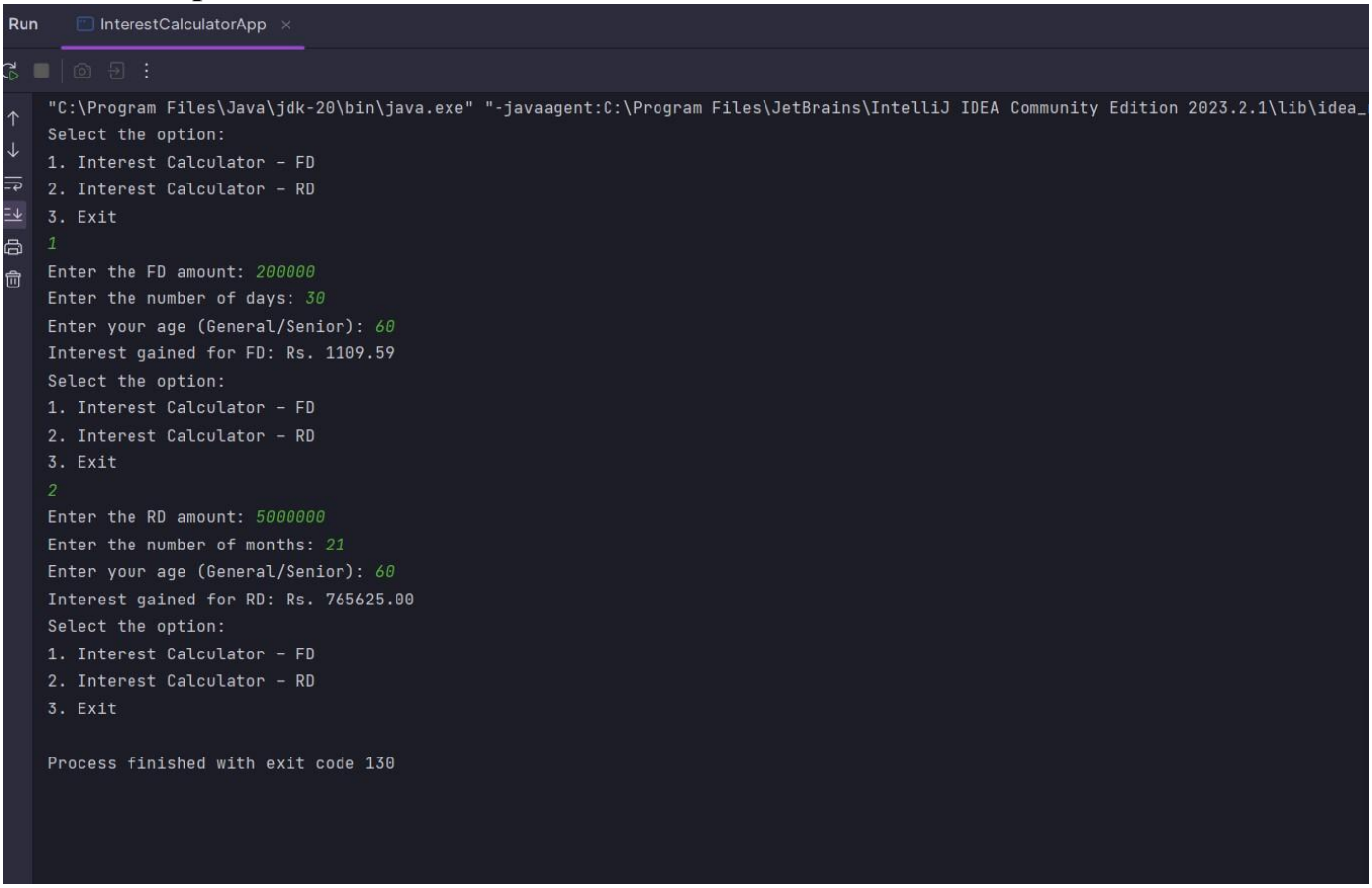
```
    private void setInterestRate() {        if (amount
< 1_00_000) { // Below 1 Crore            if
(noOfDays >= 7 && noOfDays <= 14) {
                interestRate = accountHolderType.equals("Senior") ? 5.00 : 4.50;
            } else if (noOfDays >= 15 && noOfDays <= 29) {
                interestRate = accountHolderType.equals("Senior") ? 5.25 : 4.75;
            } else if (noOfDays >= 30 && noOfDays <= 45) {
                interestRate = accountHolderType.equals("Senior") ? 6.00 : 5.50;
            } else if (noOfDays >= 46 && noOfDays <= 60) {
                interestRate = accountHolderType.equals("Senior") ? 7.50 : 7.00;
            } else if (noOfDays >= 61 && noOfDays <= 184) {
                interestRate = accountHolderType.equals("Senior") ? 8.00 : 7.50;
            } else if (noOfDays >= 185 && noOfDays <= 365) {
                interestRate = accountHolderType.equals("Senior") ? 8.50 : 8.00;
            }
        } else { // Above 1 Crore
            if (noOfDays >= 7 && noOfDays <= 14) {
interestRate = 6.50;
            } else if (noOfDays >= 15 && noOfDays <= 29) {
interestRate = 6.75;
            } else if (noOfDays >= 30 && noOfDays <= 45) {
interestRate = 6.75;
            } else if (noOfDays >= 46 && noOfDays <= 60) {
interestRate = 8.00;
            } else if (noOfDays >= 61 && noOfDays <= 184) {
interestRate = 8.50;
            } else if (noOfDays >= 185 && noOfDays <= 365) {
interestRate = 10.00;
            }
        }
    }

    @Override
    double calculateInterest() {
        return (amount * interestRate * noOfDays) / 36500; // Simple interest formula
    }
}


class RDAccount extends Account {
private int noOfMonths;
    private String accountHolderType; // "General" or "Senior"

    public RDAccount(double amount, int noOfMonths, String accountHolderType) {
super(amount);
```

```
        this.noOfMonths = noOfMonths;
this.accountHolderType = accountHolderType;
setInterestRate();
    }

    private void setInterestRate() {
if (noOfMonths == 6) {
            interestRate = accountHolderType.equals("Senior") ? 8.00 : 7.50;
        } else if (noOfMonths == 9) {
            interestRate = accountHolderType.equals("Senior") ? 8.25 : 7.75;
        } else if (noOfMonths == 12) {
            interestRate = accountHolderType.equals("Senior") ? 8.50 : 8.00;
        } else if (noOfMonths == 15) {
            interestRate = accountHolderType.equals("Senior") ? 8.75 : 8.25;
        } else if (noOfMonths == 18) {
            interestRate = accountHolderType.equals("Senior") ? 9.00 : 8.50;
        } else if (noOfMonths == 21) {
            interestRate = accountHolderType.equals("Senior") ? 9.25 : 8.75;
        }
    }

    @Override
    double calculateInterest() {
        return (amount * interestRate * noOfMonths) / 1200; // Simple interest formula
    }
}
```

```
import java.util.Scanner;

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

        do {
            System.out.println("Select the option:");
            System.out.println("1. Interest Calculator – FD");
            System.out.println("2. Interest Calculator – RD");
            System.out.println("3. Exit");
            choice = scanner.nextInt();

            switch (choice) {
                case 1: // FD Interest Calculation
```

```
                System.out.print("Enter the FD amount: ");
double fdAmount = scanner.nextDouble();
System.out.print("Enter the number of days: ");                int
noOfDays = scanner.nextInt();
                System.out.print("Enter your age (General/Senior): ");
                String fdAccountHolderType = scanner.next();

                // Create FDAccount object
                FDAccount fdAccount = new FDAccount(fdAmount, noOfDays, fdAccountHolderType);
double fdInterest = fdAccount.calculateInterest();
                System.out.printf("Interest gained for FD: Rs. %.2f%n", fdInterest);
break;

            case 2: // RD Interest Calculation
                System.out.print("Enter the RD amount: ");
double rdAmount = scanner.nextDouble();
System.out.print("Enter the number of months: ");
int noOfMonths = scanner.nextInt();
                System.out.print("Enter your age (General/Senior): ");
String rdAccountHolderType = scanner.next();

                // Create RDAccount object
                RDAccount rdAccount = new RDAccount(rdAmount, noOfMonths, rdAccountHolderType);
double rdInterest = rdAccount.calculateInterest();
                System.out.printf("Interest gained for RD: Rs. %.2f%n", rdInterest);
break;

            case 3: // Exit
                System.out.println("Exiting the application.");
break;

default:
                System.out.println("Invalid option. Please try again.");
            }
        } while (choice != 3);

        scanner.close();
    }
}
```

4. Output:



5. Learning Outcomes:

- **Abstraction & Inheritance** – Uses an abstract class `Account` with method overriding in subclasses.
- **Financial Logic** – Implements interest calculations for SB, FD, and RD accounts based on conditions.
- **Input Validation** – Ensures valid inputs and handles errors using `IllegalArgumentException`.
- **User Interaction** – Provides a menu-driven interface for seamless user experience.

EXPERIMENT- 4

Student Name: Alok Dangwal **UID:** 22BCS11817

Branch: CSE **Section/Group:** 22BCS_KRG_IOT_3_B

Semester: 6th **Date of Performance:** 10.03.25

Subject Name: Project Based Learning in Java **Subject Code:**22CSH-359

1. Aim: To develop a Card Collection System in Java that allows users to store, organize, and retrieve cards based on their symbol using the Collection interface. The program will enable users to efficiently search for all cards of a specific symbol, ensuring easy management and retrieval of card data.

2. Objective: Card Collection System Create a program to collect and store all the cards to assist the users in finding all the cards in a given symbol using Collection interface.

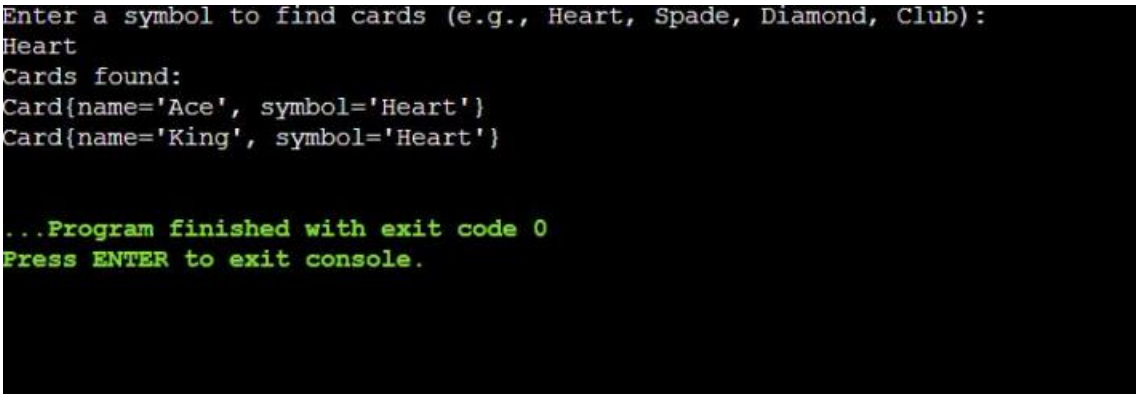
3. Implementation/Code:

```
importjava.util.*;
class Card {
private String symbol;
private String name;
public Card(String symbol, String name)
{ this.symbol = symbol;
this.name = name;
}
public String getSymbol()
{ return symbol;
}
public String getName()
```

```
{ return name;
}
@Override
public String toString() {

DEPARTMENT OF
COMPUTERSCIENCE &ENGINEERING
return "Card{name='" + name + "',symbol='" + symbol + "'}";
}
}
class CardCollection {
private Collection<Card> cards;
public CardCollection()
{ cards = new ArrayList<>();
}
public void addCard(String symbol, String name)
{ cards.add(new Card(symbol, name));
}
public List<Card> findCardsBySymbol(String symbol)
{ List<Card> result = new ArrayList<>();
for (Card card : cards) {
if (card.getSymbol().equalsIgnoreCase(symbol))
{ result.add(card);
}
}
return result;
}
public void displayAllCards()
{ for (Card card : cards) {
System.out.println(card);
```

```
}
}
}
public class CardSystem {
public static void main(String[] args)
{ Scanner scanner = new
Scanner(System.in);
CardCollection collection = new CardCollection();
collection.addCard("Heart", "Ace");
collection.addCard("Heart", "King");
collection.addCard("Spade", "Queen");
collection.addCard("Diamond", "Jack");
System.out.println("Enter a symbol to find cards (e.g., Heart, Spade, Diamond, Club): ");
String symbol = scanner.nextLine();
List<Card> foundCards = collection.findCardsBySymbol(symbol);
if (foundCards.isEmpty()) {
System.out.println("No cards found for symbol: " + symbol);

} else {
System.out.println("Cards found:");
for (Card card : foundCards) {
System.out.println(card);
}
}
scanner.close();
}
}
```

4. Output:



5. Learning Outcomes:

1. Learn how to perform basic CRUD (Create, Read, Update, Delete) operations on a List of String objects in Java.
2. Understand how to use the ArrayList classfor dynamically storing and manipulating a collection of items.
3. Practice handling user input using the Scanner class for interaction with the program.
4. Implement methodsfor searching, deleting, and displaying items in a list efficiently.
5. Gain familiarity with control flow and loopsto allow for continuous user interaction until the program is exited.

Experiment 5.1

Student Name: Alok Dangwal **UID: 22BCS11817**
Branch: CSE **Section: KRG_IOT_3B**
Semester: 6th **DOP: 18/02/25**
Subject: PBLJ **Subject Code:22CSH-359**

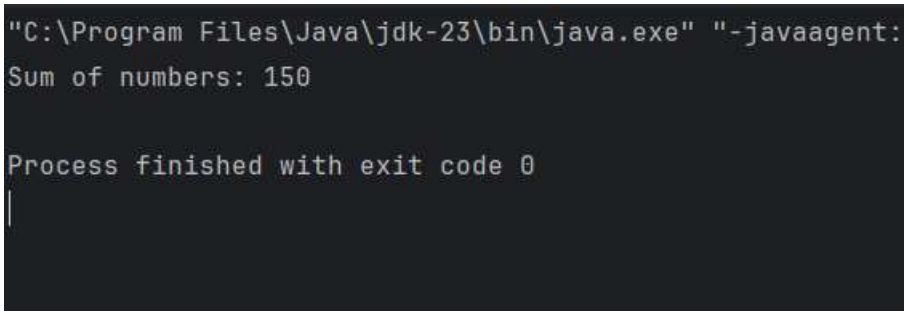
1.Aim: Write a Java program to calculate the sum of a list of integers using autoboxing and unboxing. Include methods to parse strings into their respective wrapper classes (e.g., Integer.parseInt()).

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

3.IMPLEMENTATION/CODE

```
import java.util.*;

public class AutoboxingUnboxingExample {

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

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

    public static void main(String[] args) {
        String[] stringNumbers = {"10", "20", "30", "40", "50"};
        List<Integer> numberList = parseStringToIntegerList(stringNumbers);
        int sum = calculateSum(numberList);
        System.out.println("Sum of numbers: " + sum);
    }
}
```

4.Output:



5.Learning Outcomes:

1) Learn how to use **autoboxing and unboxing** to automatically convert between `int` and `Integer`.
2) Learn how to use **Integer.parseInt()** to convert string numbers into Integer objects.
3) Learn how to use **ArrayList** to store and manipulate a list of integers.
4) Learn how to use **methods** to parse strings into integers and calculate the sum.
5) Learn how to use **loops** to iterate through a list and compute the sum efficiently.

Experiment 5.2

1.Aim: 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.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;
    }

    public void display() {
        System.out.println("Student ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("GPA: " + gpa);
    }
}

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

    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 (IOException e) {
            System.err.println("Error during serialization: " + e.getMessage());
        }
    }
```

```
    public static Student deserializeStudent() {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
            return (Student) ois.readObject();
        } catch (FileNotFoundException e) {
            System.err.println("File not found: " + e.getMessage());
        } catch (IOException e) {
            System.err.println("Error during deserialization: " + e.getMessage());
        } catch (ClassNotFoundException e) {
            System.err.println("Class not found: " + e.getMessage());
        }
        return null;
    }

    public static void main(String[] args) {
        Student student = new Student(101, "John Doe", 3.8);
        serializeStudent(student);
        Student deserializedStudent = deserializeStudent();
        if (deserializedStudent != null) {
            System.out.println("Deserialized Student Details:");
            deserializedStudent.display();
        }
    }
}
```

4.Output

```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Fil
Student object serialized successfully.
Deserialized Student Details:
Student ID: 11735
Name: prakhar
GPA: 3.8

Process finished with exit code 0
```

1. Learning Outcomes:

1) Understand serialization & deserialization in Java.
2) Use ObjectOutputStream & ObjectInputStream for object storage.
3) Implement Serializable interface and serialVersionUID.
4) Handle file operations & exceptions (IOException, FileNotFoundException).

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.Implementation Code

```
import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

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;
    }

    public void display() {
        System.out.println("Employee ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("Designation: " + designation);
        System.out.println("Salary: " + salary);
        System.out.println("--------------------------");
    }
}

public class ex3{
    private static final String FILE_NAME = "employees.dat";
    private static Scanner scanner = new Scanner(System.in);
```

```
    public static void main(String[] args) {
        while (true) {
            System.out.println("\nMenu:");
            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(); // Consume newline

            switch (choice) {
                case 1:
                    addEmployee();
                    break;
                case 2:
                    displayAllEmployees();
                    break;
                case 3:
                    System.out.println("Exiting the application...");
                    return;
                default:
                    System.out.println("Invalid choice! Please try again.");
            }
        }
    }
    private static void addEmployee() {
        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 Employee Designation: ");
        String designation = scanner.nextLine();
        System.out.print("Enter Employee Salary: ");
        double salary = scanner.nextDouble();

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

    private static void saveEmployeeToFile(Employee employee) {
        List<Employee> employees = loadEmployees();
        employees.add(employee);

        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(FILE_NAME))) {
            oos.writeObject(employees);
        } catch (IOException e) {
            System.err.println("Error saving employee: " + e.getMessage());
        }
    }
```

```
    private static List<Employee> loadEmployees() {
        List<Employee> employees = new ArrayList<>();
        File file = new File(FILE_NAME);

        if (!file.exists()) {
            return employees;
        }
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME)))
{
            employees = (List<Employee>) ois.readObject();
        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Error loading employees: " + e.getMessage());
        }
        return employees;
    }

    private static void displayAllEmployees() {
        List<Employee> employees = loadEmployees();
        if (employees.isEmpty()) {
            System.out.println("No employees found.");
        } else {
            System.out.println("\nEmployee Details:");
            for (Employee emp : employees) {
                emp.display();
            }}}}
```

4.Output:

```
Menu:
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 101
Enter Employee Name: prakhr
Enter Employee Designation: cse
Enter Employee Salary: 20000
Employee added successfully.

Menu:
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 2

Employee Details:
Employee ID: 101
Name: prakhr
Designation: cse
Salary: 20000.0
--------------------------
```

5.Learning Outcomes:

1) I have learned how to **serialize and deserialize objects** in Java.
2) I have learned how to **store and retrieve data from a file** using file handling.
3) I have learned how to **implement a menu-driven program** using loops and switch-case.
4) I have learned how to **handle exceptions** like `IOException` and `ClassNotFoundException`.
5) I have learned how to **use ArrayList to manage multiple employee records**.
6) I have learned how to **take user input and process it efficiently**.

Experiment 6

Student Name: Alok Dangwal UID: 22BCS11817
Branch: BE-CSE Section/Group: 22BCS_IOT-638(A)
Semester: 6th Date of Performance: 24/02/25
Subject Name: PBLJ Subject Code: 22CSH-359

1. Aim: Develop Java programs using lambda expressions and stream operations for sorting, filtering, and processing large datasets efficiently while utilizing wrapper classes, autoboxing and unboxing, byte streams, character streams, object serialization, cloning, functional interfaces, method references, and various stream operations such as sorting, filtering, mapping, reducing, and grouping.

2. Programming Problems:

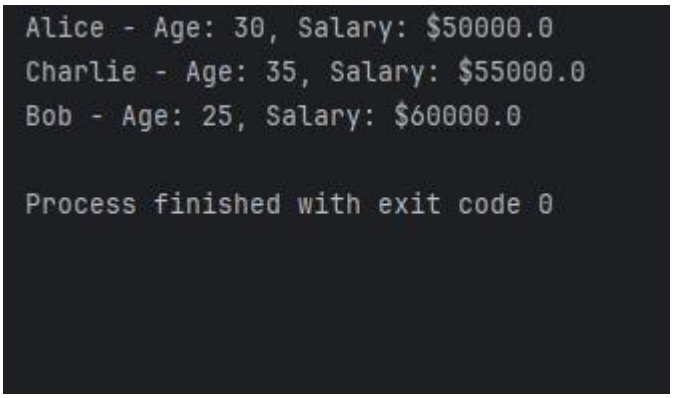
a) Write a program to sort a list of Employee objects (name, age, salary) using lambda expressions.

- Implementation/Code:

```
import java.util.*;

class Employee
   { String name;
   int age;
   double salary;

   public 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("Alice",
         30, 50000), new Employee("Bob", 25,
         60000),
         new Employee("Charlie", 35, 55000)
      );

      employees.sort(Comparator.comparingDouble(e -> e.salary));
      employees.forEach(System.out::println);
   }
}
```

- Output:



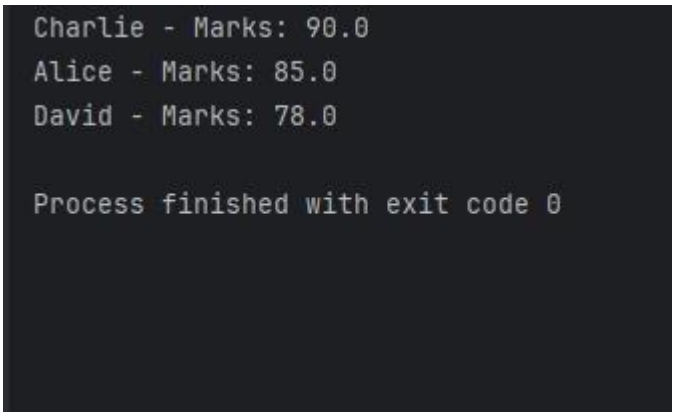
b)Create a program to use lambda expressions and stream operations to filter students scoring above 75%, sort them by marks, and display their names.

- Implementation/Code:

```
import java.util.*; import
java.util.stream.Collectors; class
Student { String name; double
marks;
```

```
    public Student(String name, double marks) {
        this.name = name;
        this.marks = marks;
    }

    @Override public
    String toString() {
        return name + " - Marks: " + marks;
    }
}

public class StudentFilter {
    public static void main(String[] args) {
        List<Student> students =
            Arrays.asList( new Student("Alice",
            85), new Student("Bob", 72), new
            Student("Charlie", 90), new
            Student("David", 78)
        );

        List<Student> filteredStudents = students.stream()
            .filter(s -> s.marks > 75)
            .sorted(Comparator.comparingDouble(s -> -s.marks))
            .collect(Collectors.toList());

        filteredStudents.forEach(System.out::println);
    }
}
```

- Output:



c)Write a Java program to process a large dataset of products using streams. Perform operations such as grouping products by category, finding the most expensive product in each category, and calculating the average price of all products..

- Implementation/Code:

```
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 + " - " + category + " - $" + 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", 80), new
            Product("Sofa", "Furniture", 700), new
            Product("Table", "Furniture", 300)
        );

        Map<String, List<Product>> groupedByCategory = products.stream()
            .collect(Collectors.groupingBy(p -> p.category));

        Map<String, Product> mostExpensiveByCategory = products.stream()
            .collect(Collectors.toMap
                ( p -> p.category, p ->
                p,
                (p1, p2) -> p1.price > p2.price ? p1 : p2
            ));

        double averagePrice = products.stream()
            .mapToDouble(p -> p.price)
            .average()
            .orElse(0);

        System.out.println("Grouped Products:");
        groupedByCategory.forEach((category, list) -> System.out.println(category + ": " + list));

        System.out.println("\nMost Expensive Products in Each Category:");
        mostExpensiveByCategory.forEach((category, product) -> System.out.println(category + ": " +
        product));

        System.out.println("\nAverage Price of All Products: $" + averagePrice);
    }
}
```

- Output:

```
Grouped Products:
Clothing: [Shirt - Clothing - $50.0, Jeans - Clothing - $80.0]
Electronics: [Laptop - Electronics - $1200.0, Phone - Electronics - $800.0, TV - Electronics - $1500.0]
Furniture: [Sofa - Furniture - $700.0, Table - Furniture - $300.0]

Most Expensive Products in Each Category:
Clothing: Jeans - Clothing - $80.0
Electronics: TV - Electronics - $1500.0
Furniture: Sofa - Furniture - $700.0

Average Price of All Products: $661.4285714285714

Process finished with exit code 0
```

3. Learning Outcome:

- Understand the use of **wrapper classes** (`Integer`, `Character`, `Long`, `Boolean`) and apply **autoboxing & unboxing** in collections and stream operations.
- Learn **byte & character streams**, implement **object serialization**, and utilize **cloning** for efficient object handling.
- Explore **lambda expressions**, implement **functional interfaces**, and use **method references** to enhance functional programming.
- Master **stream operations** like **sorting, filtering, mapping, reducing, and grouping** for efficient data processing in large datasets.

DEPARTMENTOF

COMPUTER SCIENCE & ENGINEERING

EXPERIMENT- 7

Student Name: Alok Dangwal **UID:** 22BCS11817

Branch: CSE **Section/Group:** 22BCS_KRG_IOT_3_B

Semester: 6th **Date of Performance:** 10.03.25

Subject Name: Project Based Learning in Java **Subject Code:**22CSH-359

EASY LEVEL

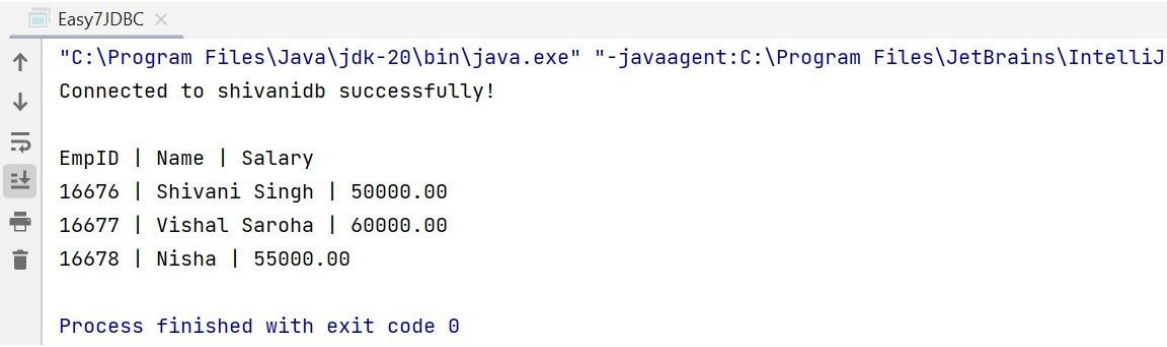
1. **Aim**: Create a Java program to connect to a MySQL database and fetch data from a single table.

2. **Objective:** To retrieve and display all records from a table named Employee with columns EmpID, Name, and Salary.

3. **Implementation/Code:**

```
package Project1; import java.sql.*; public
class Easy7JDBC {     public static void
main(String[] args) {
        // Database connection details
        String url = "jdbc:mysql://localhost:3306/shivanidb";
        String username = "root";
        String password = "Shivani@1234";
        // SQL Query
        String query = "SELECT * FROM Employee";         try (Connection
conn = DriverManager.getConnection(url, username, password);
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(query)) {
            System.out.println("Connected to shivanidb successfully!\n");
System.out.println("EmpID | Name | Salary");             while
(rs.next()) {
                System.out.printf("%d | %s | %.2f\n",
rs.getInt("EmpID"),                          rs.getString("Name"),
rs.getDouble("Salary"));}
        } catch (SQLException e) {
            System.err.println("Connection failed: " + e.getMessage());
        }
    }}
```

4. **Output:**



MEDIUM LEVEL

1. **Aim**: Build a program to perform CRUD operations

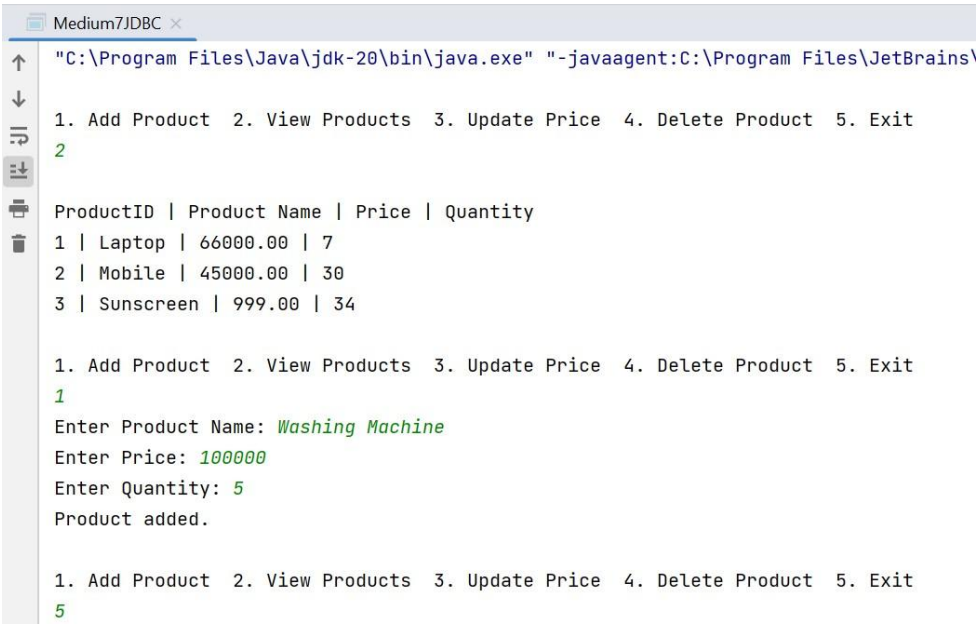
2. **Objective:** To perform Create, Read, Update, Delete on a database table Product with columns: ProductID, ProductName, Price, and Quantity. The program should include menu-driven options for each operation.

3. **Implementation/Code:**

```
package Project1;          import
java.sql.*;          import
java.util.Scanner; public class
Medium7JDBC {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/shivanidb";
        String user = "root";
        String password = "Shivani@1234";          Scanner sc = new
Scanner(System.in);          try (Connection conn =
DriverManager.getConnection(url, user, password)) {              while
(true) {
                System.out.println("\n1. Add Product  2. View Products
3. Update Price  4. Delete Product  5. Exit");
int choice = sc.nextInt();                  if (choice == 1)
addProduct(conn, sc);                 else if (choice == 2)
viewProducts(conn);                 else if (choice == 3)
updateProduct(conn, sc);                 else if (choice == 4)
deleteProduct(conn, sc);                 else if (choice == 5)
break;
                else System.out.println("Invalid choice.");
            }
        } catch (SQLException e) {
e.printStackTrace();}}     static void
addProduct(Connection conn, Scanner sc) throws
SQLException {
        System.out.print("Enter Product Name: ");
sc.nextLine();
```

```
        String name = sc.nextLine();
System.out.print("Enter Price: ");         double
price = sc.nextDouble();
System.out.print("Enter Quantity: ");         int
quantity = sc.nextInt();
        PreparedStatement stmt = conn.prepareStatement("INSERT INTO
Product (ProductName, Price, Quantity) VALUES (?, ?, ?)");
stmt.setString(1, name);         stmt.setDouble(2, price);
stmt.setInt(3, quantity);          stmt.executeUpdate();
        System.out.println("Product added.");
    }
    static void viewProducts(Connection conn) throws SQLException {
        ResultSet rs = conn.createStatement().executeQuery("SELECT * FROM
Product");
        System.out.println("\nProductID | Product Name | Price |
Quantity");
        while (rs.next()) {
            System.out.printf("%d | %s | %.2f | %d\n", rs.getInt(1),
rs.getString(2), rs.getDouble(3), rs.getInt(4));
        }
}
    static void updateProduct(Connection conn, Scanner sc) throws
SQLException {
        System.out.print("Enter ProductID to update: ");
int id = sc.nextInt();
        System.out.print("Enter new Price: ");
double price = sc.nextDouble();
        PreparedStatement stmt = conn.prepareStatement("UPDATE Product
SET Price=? WHERE ProductID=?");         stmt.setDouble(1, price);
stmt.setInt(2, id);         stmt.executeUpdate();
        System.out.println("Product updated.");
    }
    static void deleteProduct(Connection conn, Scanner sc) throws
SQLException {
        System.out.print("Enter ProductID to delete: ");
int id = sc.nextInt();
        PreparedStatement stmt = conn.prepareStatement("DELETE FROM
Product WHERE ProductID=?");         stmt.setInt(1, id);
stmt.executeUpdate();
        System.out.println("Product deleted.");
    }}
```

4. Output:



HARD LEVEL

1. **Aim**: Develop a Java application using JDBC and MVC architecture to manage student data.

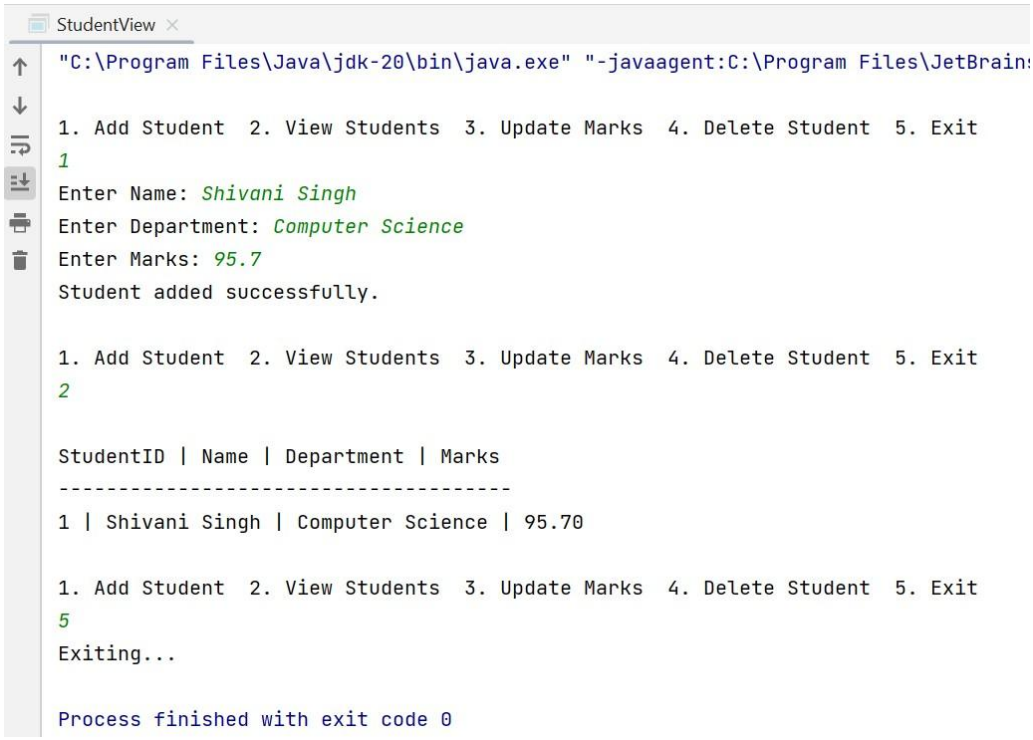
2. **Objective:** To Use a Student class as the model with fields like StudentID, Name, Department, and Marks. Include a database table to store student data.

3. **Implementation/Code:**

```
package Project1;
 import
java.sql.SQLException; import
java.util.List; import
java.util.Scanner;

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

            while (true) {
                System.out.println("\n1. Add Student  2. View Students
3. Update Marks  4. Delete Student  5. Exit");
                int choice = sc.nextInt();

                if (choice == 1) {
```

```
                    System.out.print("Enter Name: ");
sc.nextLine();
                    String name = sc.nextLine();
                    System.out.print("Enter Department: ");
                    String dept = sc.nextLine();
System.out.print("Enter Marks: ");                    double marks =
sc.nextDouble();                    controller.addStudent(new
Studentss(0, name, dept, marks));                }
                else if (choice == 2) {
                    List<Studentss> students = controller.getStudents();
                    System.out.println("\nStudentID | Name | Department |
Marks");
                    System.out.println("---------------------------------
-----");
                    for (Studentss s : students) {
                        System.out.printf("%d | %s | %s | %.2f\n",
s.getStudentID(), s.getName(), s.getDepartment(), s.getMarks());
                    }
}
                else if (choice == 3) {
                    System.out.print("Enter StudentID to update: ");
int id = sc.nextInt();
                    System.out.print("Enter new Marks: ");
double marks = sc.nextDouble();
controller.updateStudentMarks(id, marks);
                }
                else if (choice == 4) {
                    System.out.print("Enter StudentID to delete: ");
int id = sc.nextInt();                    controller.deleteStudent(id);
                }
                else if (choice == 5) {
                    System.out.println("Exiting...");
break;                  }                else {
                    System.out.println("Invalid choice.");
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

4. Output:



5. Learning Outcomes:

(i) Learn how to **establish a connection** between a Java application and a MySQL database using **JDBC**.

(ii) Understand the use of **DriverManager and Connection objects** to interact with the database.

(iii) Learn to use **PreparedStatement** to securely execute SQL queries.