EXPERIMENT- 7

Student Name: Aryan   **UID:** 22BCS10190

Branch: CSE   **Section/Group:** KRG_IOT-3-B

Semester: 6   **Date of Performance:** 02.04.25

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

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:**

```java
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();
```
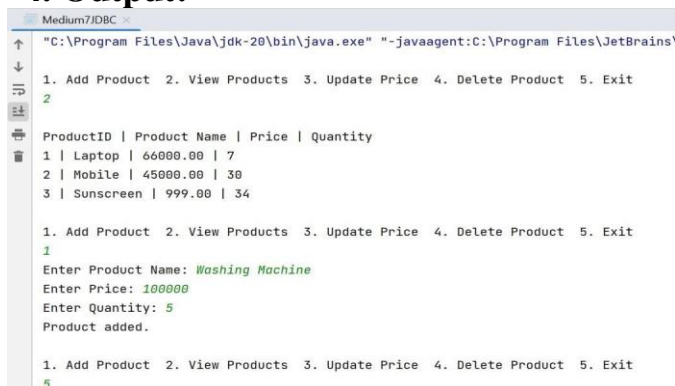
```java
        System.out.print("Enter Price: ");
        double price = sc.nextDouble();
        System.out.print("Enter Quantity: ");
        int quantity = sc.nextInt();
        PreparedStatement stmt = conn.prepareStatement
        ("INSERT INTO Product    (ProductName, Price, Quantity) VALUES (?, ?,
?)"); 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:

5. Learning Outcomes:

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

- Understand the use of **DriverManager and Connection objects** to interact with the database.

Experiment 1

Student Name: Aryan   UID:22BCS10190

Branch: BE-CSE   Section/Group:KRG_IOT-3-B

Semester:6th   Date of Performance:14-01-25

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

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;

```
class Employee {
    int empNo;
    String empName;
    String joinDate;
    char desigCode;
    String department;
    int basic;
    int hra;
    int it;

    Employee(int empNo, String empName, String joinDate, char desigCode, String department, int basic, int hra, int it) {
        this.empNo = empNo;
        this.empName = empName;
        this.joinDate = joinDate;
```

```java
        this.desigCode = desigCode;
        this.department = department;
        this.basic = basic;
        this.hra = hra;
        this.it = it;
    }

    int calculateSalary() {
        int da = 0; // Dearness Allowance based on desigCode
        switch (desigCode) {
            case 'e': da = 20000; break;
            case 'c': da = 32000; break;
            case 'k': da = 12000; break;
            case 'r': da = 15000; break;
            case 'm': da = 40000; break;
            default: break;
        }
        return basic + hra + da - it;
    }
}

public class exp_1_1 {
    public static void main(String[] args) {
        Employee[] employees = {
                new Employee(1001, "Ashish", "01/04/2009", 'e', "R&D", 20000, 8000, 3000),
                new Employee(1002, "Sushma", "23/08/2012", 'c', "PM", 30000, 12000, 9000),
                new Employee(1003, "Rahul", "12/11/2008", 'k', "Acct", 10000, 8000, 1000),
                new Employee(1004, "Chahat", "29/01/2013", 'r', "Front Desk", 12000, 6000, 2000),
                new Employee(1005, "Ranjan", "16/07/2005", 'm', "Engg", 50000, 20000, 20000),
                new Employee(1006, "Suman", "01/01/2000", 'e', "Manufacturing", 23000, 9000, 4400),
                new Employee(1007, "Tanmay", "12/06/2006", 'c', "PM", 29000, 12000, 10000)
        };

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter Employee ID: ");
        int empNoInput = scanner.nextInt();

        boolean found = false;
        for (Employee emp : employees) {
            if (emp.empNo == empNoInput) {
                found = true;
                int salary = emp.calculateSalary();
                System.out.println("Emp No: " + emp.empNo);
                System.out.println("Emp Name: " + emp.empName);
                System.out.println("Department: " + emp.department);
                System.out.println("Designation Code: " + emp.desigCode);
                System.out.println("Salary: " + salary);
                break;
            }
```
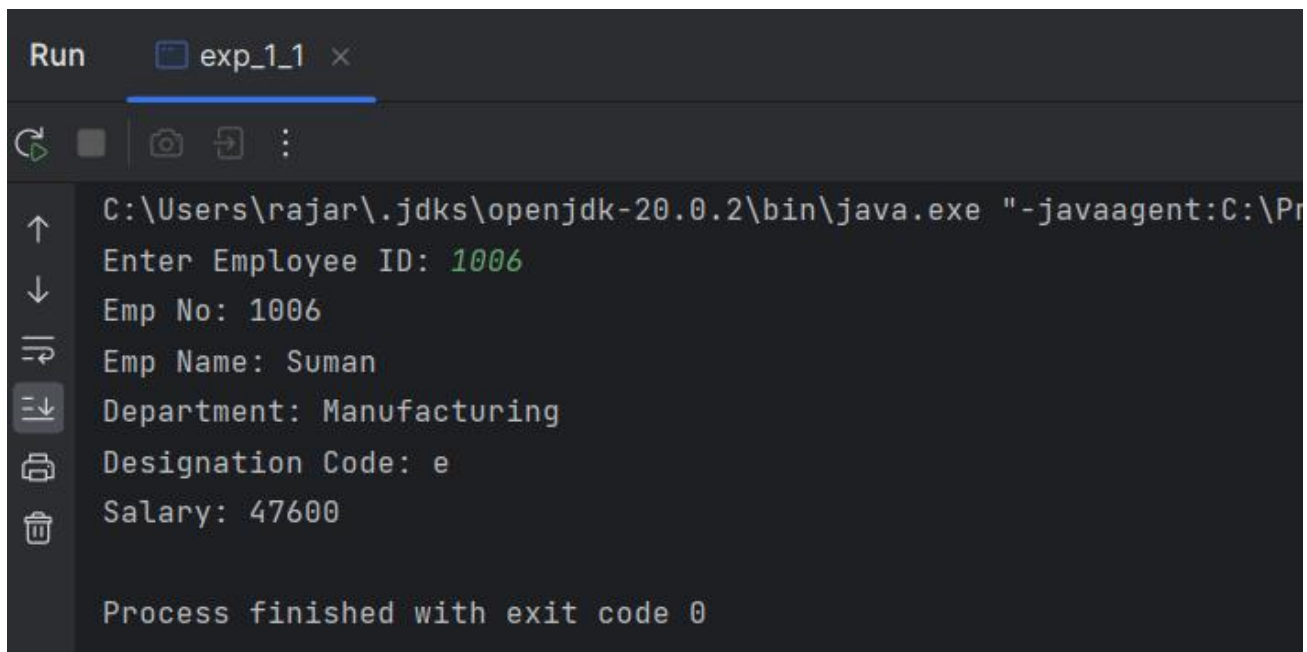
```
        }

        if (!found) {
            System.out.println("Employee not found!");
        }

        scanner.close();
    }
}
```

4. Output:

```
Run        exp_1_1  ×

C:\Users\rajar\.jdks\openjdk-20.0.2\bin\java.exe "-javaagent:C:\Pr
Enter Employee ID: 1006
Emp No: 1006
Emp Name: Suman
Department: Manufacturing
Designation Code: e
Salary: 47600

Process finished with exit code 0
```

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: Aryan   **UID: 22BCS10190**
Branch: BE CSE   **Section/Group: KRG-IOT-3B**
Semester: 6th   **Date of Performance: 15-01-2025**
Subject Name: JAVA   **Subject Code:22CSH-359**

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

2. **Objective:** To design and implement a user-friendly inventory control system for a small video rental store, enabling efficient management of video inventory, including functionalities for adding, renting, and returning videos.

3. **Implementation/Code:**

```java
import java.util.*;

class Item {
    String name;
    String genre;

    Item(String name, String genre)
        { this.name = name;
        this.genre = genre;
    }
    void display() {
        System.out.println("Name: " + name + ", Genre: " + genre);
    }
}

class Movie extends Item
    { String director;
    Movie(String name, String genre, String director)
        { super(name, genre);
        this.director = director;
    }

    @Override
    void display() {
```

```java
      super.display();
      System.out.println("Director: " + director);
   }
}

class TVShow extends Item
   { String season;
   TVShow(String name, String genre, String season)
      { super(name, genre);
      this.season = season;
   }
   @Override
   void display() {
      super.display();
      System.out.println("Season: " + season);
   }
}

public class Main {
   public static void main(String[] args)
      { Scanner sc = new Scanner(System.in);
      ArrayList<Item> inventory = new ArrayList<>();

      System.out.print("Enter number of movies: ");
      int numMovies = sc.nextInt();
      sc.nextLine();
      for (int i = 0; i < numMovies; i++)
         { System.out.print("Enter movie name: ");
         String name = sc.nextLine();
         System.out.print("Enter movie genre: ");
         String genre = sc.nextLine();
         System.out.print("Enter director: ");
         String director = sc.nextLine();
         inventory.add(new Movie(name, genre, director));
      }

      System.out.print("Enter number of TV shows: ");
      int numShows = sc.nextInt();
      sc.nextLine();
      for (int i = 0; i < numShows; i++)
         { System.out.print("Enter TV show name: ");
         String name = sc.nextLine();
         System.out.print("Enter TV show genre: ");
         String genre = sc.nextLine();
```
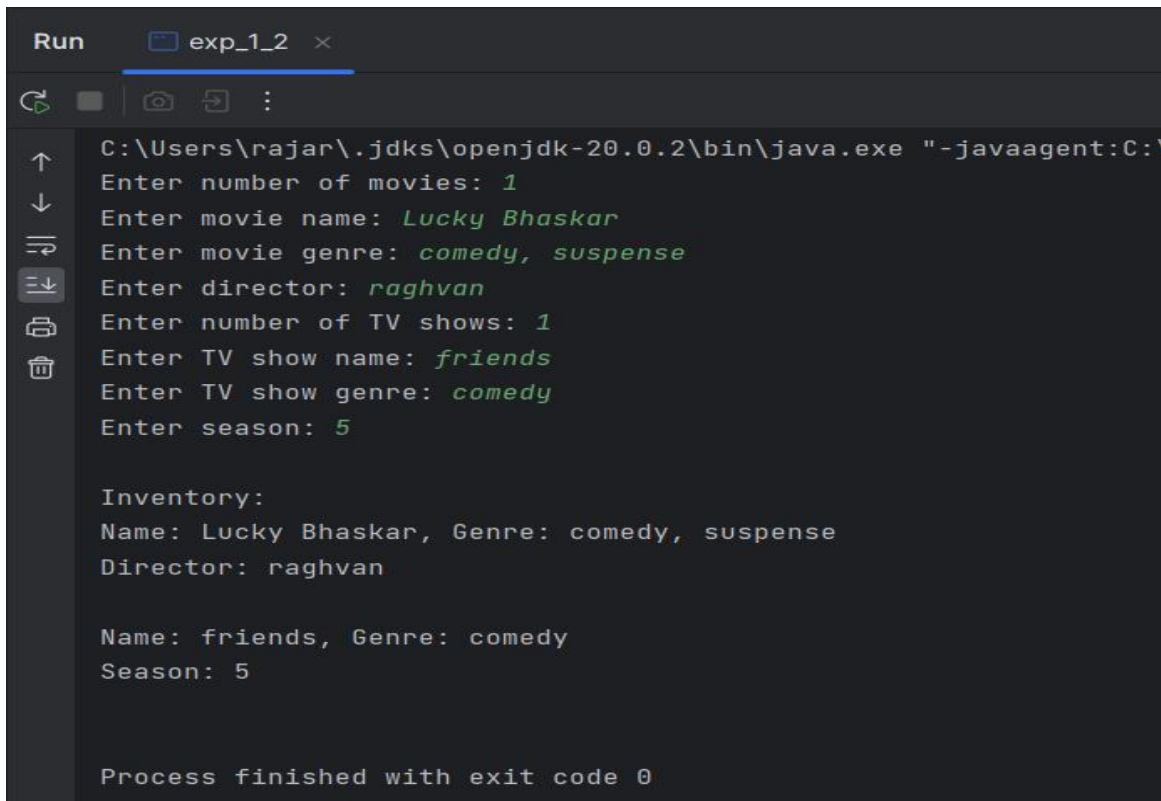
```java
                System.out.print("Enter season: ");
                String season = sc.nextLine();
                inventory.add(new TVShow(name, genre, season));
            }
            System.out.println("\nInventory:");
            for (Item item : inventory) {
                item.display();
                System.out.println();
            }

            sc.close();
        }
    }
```

4. Output:

```
Run        exp_1_2  ×

C:\Users\rajar\.jdks\openjdk-20.0.2\bin\java.exe "-javaagent:C:\
Enter number of movies: 1
Enter movie name: Lucky Bhaskar
Enter movie genre: comedy, suspense
Enter director: raghvan
Enter number of TV shows: 1
Enter TV show name: friends
Enter TV show genre: comedy
Enter season: 5

Inventory:
Name: Lucky Bhaskar, Genre: comedy, suspense
Director: raghvan

Name: friends, Genre: comedy
Season: 5


Process finished with exit code 0
```
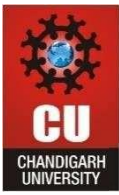
5. Learning Outcome:

- Learnt about Inheritance in java.
- Learnt about array List.
- Practice loops, conditionals, and methods for inventory operations.

Experiment-3

Student Name: Aryan   **UID:22BCS10190**

Branch: BE-CSE   **Section/Group: 22BCS-KRG-3B**

Semester: 6ᵗʰ   **Date of Performance: 25.1.25**

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

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

2.Objective: To design and implement a Java program that calculates interest for various account types (FD, RD, SB) using object-oriented principles, focusing on abstraction, method overriding, and dynamic input validation.

3.Implementation/Code:

```java
abstract class Account {
double interestRate;
 double amount;
abstract double calculateInterest();
}
class FDAccount extends Account {
int noOfDays;
int ageOfACHolder;

FDAccount(double amount, int noOfDays, int ageOfACHolder) {
this.amount = amount;
this.noOfDays = noOfDays;
 this.ageOfACHolder = ageOfACHolder;
}
@Override
double calculateInterest() {
if (amount < 10000000) { // Less than 1 crore
if (noOfDays >= 7 && noOfDays <= 14) interestRate = ageOfACHolder >= 60 ? 5.0 : 4.5;
else if (noOfDays >= 15 && noOfDays <= 29) interestRate = ageOfACHolder >= 60 ? 5.25: 4.75;
else if (noOfDays >= 30 && noOfDays <= 45) interestRate = ageOfACHolder >= 60 ? 6.0 : 5.5;
else if (noOfDays >= 45 && noOfDays <= 60) interestRate = ageOfACHolder >= 60 ? 7.5 :7.0;
else if (noOfDays >= 61 && noOfDays <= 184) interestRate = ageOfACHolder >= 60 ? 8.0: 7.5;
 else if (noOfDays >= 185 && noOfDays <= 365) interestRate = ageOfACHolder >= 60 ?8.5 : 8.0;
} else { // Greater than or equal to 1 crore
if (noOfDays >= 7 && noOfDays <= 14) interestRate = 6.5;
else if (noOfDays >= 15 && noOfDays <= 29) interestRate = 6.75;
 else if (noOfDays >= 30 && noOfDays <= 45) interestRate = 6.75;
```

```java
    else if (noOfDays >= 45 && noOfDays <= 60) interestRate = 8.0;
    else if (noOfDays >= 61 && noOfDays <= 184) interestRate = 8.5;
    else if (noOfDays >= 185 && noOfDays <= 365) interestRate = 10.0;
}
return amount * interestRate / 100;
}
}
class RDAccount extends Account {
 int noOfMonths;
double monthlyAmount;
 int ageOfACHolder;

RDAccount(double monthlyAmount, int noOfMonths, int ageOfACHolder) {
 this.monthlyAmount = monthlyAmount;
this.noOfMonths = noOfMonths;
this.ageOfACHolder = ageOfACHolder;
}
@Override
double calculateInterest() {
if (noOfMonths == 6) interestRate = ageOfACHolder >= 60 ? 8.0 : 7.5;
else if (noOfMonths == 9) interestRate = ageOfACHolder >= 60 ? 8.25 : 7.75;
 else if (noOfMonths == 12) interestRate = ageOfACHolder >= 60 ? 8.5 : 8.0;
 else if (noOfMonths == 15) interestRate = ageOfACHolder >= 60 ? 8.75 : 8.25;
 else if (noOfMonths == 18) interestRate = ageOfACHolder >= 60 ? 9.0 : 8.5;
else if (noOfMonths == 21) interestRate = ageOfACHolder >= 60 ? 9.25 : 8.75;
 return monthlyAmount * noOfMonths * interestRate / 100;
}
}
class SBAccount extends Account {
String accountType;
SBAccount(double amount, String accountType) {
 this.amount = amount;
 this.accountType = accountType;
}
@Override
double calculateInterest() {
interestRate = accountType.equalsIgnoreCase("NRI") ? 6.0 : 4.0;
return amount * interestRate / 100;
}
```

4.Output:

```
Select the option:
1. Interest Calculator  SB
2. Interest Calculator  FD
3. Interest Calculator  RD
4. Exit
1
Enter the Average amount in your account:
50000
Enter account type (Normal/NRI):
normal
Interest gained: Rs. 2000.0
Select the option:
1. Interest Calculator  SB
2. Interest Calculator  FD
3. Interest Calculator  RD
4. Exit
```

5. Learning outcomes:

1. Understand the concept of abstract classes and method overriding in Java.

2. Learn to implement real-world scenarios using object-oriented principles.

3. Develop skills to validate user input for different account types.

4. Gain knowledge of calculating interest dynamically based on conditions.

5. Enhance problem-solving abilities by applying conditional logic effectively.

Experiment 4

Student Name: Aryan   **UID: 22BCS10190**

Branch: BE-CSE   **Section/Group: 22BCS_KRG-3-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**

```java
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);
```
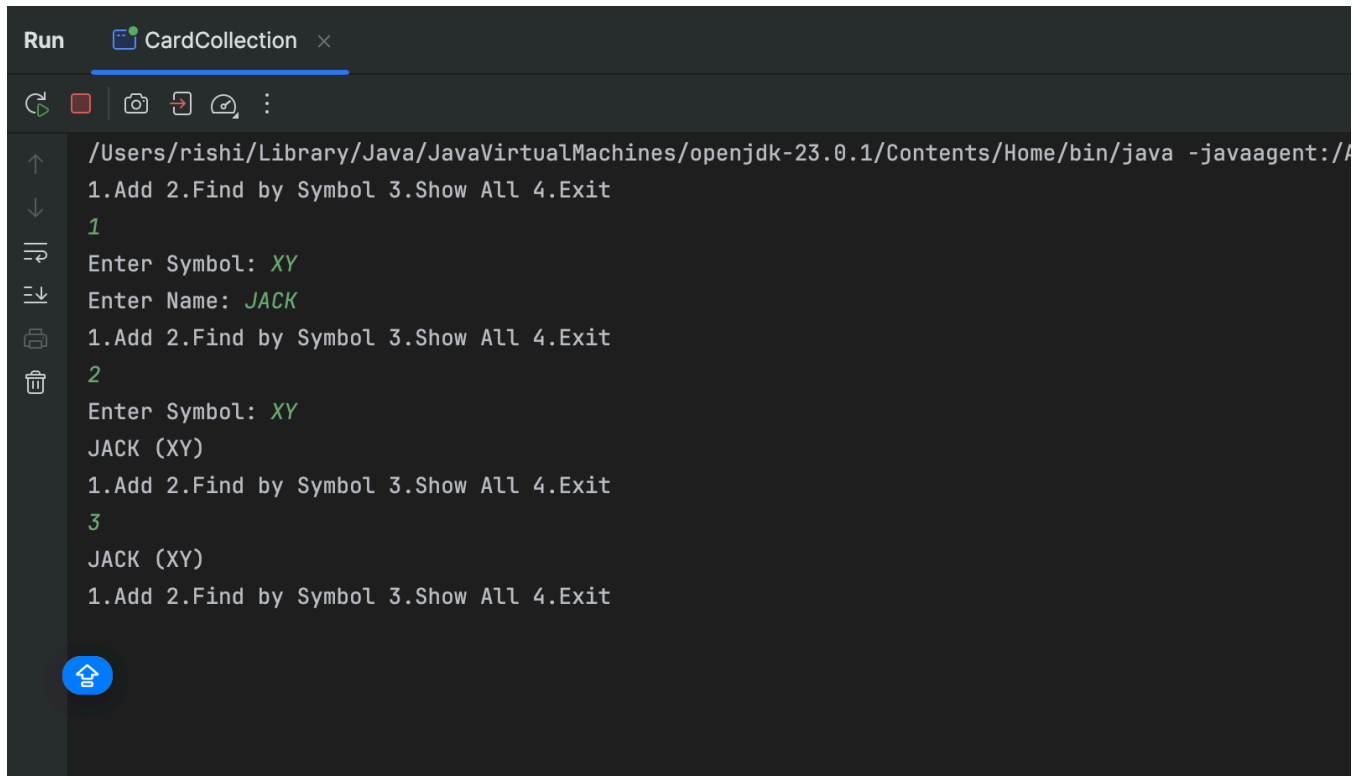
```java
public static void main(String[] args) {
    while (true) {
        System.out.println("1.Add 2.Find by Symbol 3.Show All 4.Exit");
        int choice = sc.nextInt();
        switch (choice) {
            case 1 -> addCard();
            case 2 -> findBySymbol();
            case 3 -> cards.forEach(System.out::println);
            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

```
Run      CardCollection  ×

/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

Student Name: Aryan

UID: 22BCS10190

Branch: CSE

Section: 22BCS_KRG-3-B

Semester: 6th

DOP: 28-2-25

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:
        o   Convert each string to an Integer using Integer.parseInt(str).
        o   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:
        o   Extract each integer (**unboxing** happens here).
        o   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:

```java
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;
    }
}
```
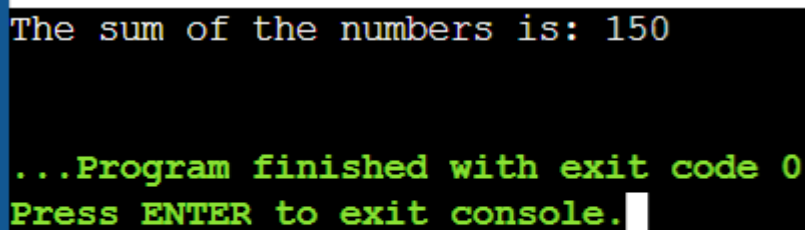
Output:

```
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 6

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:

```java
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 8

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:
   o A List<Boolean> representing seat availability (true for available, false for booked).
   o 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):
   o If the seat is available (true), mark it as booked (false).
   o Print confirmation: "Seat X booked successfully by Y".
   o If already booked, print: "Seat X is already booked."

Step 3: Define Booking Threads
1. Create a class PassengerThread extending Thread:
   o Store passenger name, seat number, and booking system reference.
   o 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 {
```

```java
        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);
                        }
                }
        }
```

```java
    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.