Experiment-4

Student Name: Prateek Pratap Singh **UID: 22BCS10036**
Branch: BE-CSE **Section/Group: IOT_631-A**
Semester:6th **Date of Performance: 14/02/2025**
Subject Name: Project Based Learning in Java with Lab **Subject Code: 22CSH-359**

1. **(a) Aim:** Write a Java program to implement an ArrayList that stores employee details (ID, Name, and Salary). Allow users to add, update, remove, and search employees.

2. **Objective:**

- To implement an ArrayList that stores employee details (ID, Name, Salary).
- Provide functionalities to:
  - Add a new employee.
  - Update an existing employee's details.
  - Remove an employee by ID.
  - Search for an employee by ID.
  - Display a menu-based system for easy user interaction.

3. **Implementation:**

```
import java.util.ArrayList;
import java.util.Scanner;

class Employee {
    private int id;
    private String name;
    private double salary;

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

```
        this.salary = salary;
    }

    public int getId() {
        return id;
    }

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

    public void setSalary(double salary) {
        this.salary = salary;
    }

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

public class Main {
    public static void main(String[] args) {
        ArrayList<Employee> employees = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\n1. Add Employee\n2. Update Employee\n3.
Remove Employee\n4. Search Employee\n5. Exit");
            System.out.print("Enter choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    System.out.print("Enter ID: ");
                    int id = scanner.nextInt();
```

```
                scanner.nextLine();
                System.out.print("Enter Name: ");
                String name = scanner.nextLine();
                System.out.print("Enter Salary: ");
                double salary = scanner.nextDouble();
                employees.add(new Employee(id, name, salary));
                System.out.println("Employee added successfully!");
                break;
            case 2:
                System.out.print("Enter ID to update: ");
                int updateId = scanner.nextInt();
                scanner.nextLine();
                boolean updated = false;
                for (Employee emp : employees) {
                    if (emp.getId() == updateId) {
                        System.out.print("Enter new Name: ");
                        String newName = scanner.nextLine();
                        System.out.print("Enter new Salary: ");
                        double newSalary = scanner.nextDouble();
                        emp.setName(newName);
                        emp.setSalary(newSalary);
                        updated = true;
                        System.out.println("Employee updated successfully!");
                        break;
                    }
                }
                if (!updated) System.out.println("Employee not found!");
                break;
            case 3:
                System.out.print("Enter ID to remove: ");
                int removeId = scanner.nextInt();
                boolean removed = employees.removeIf(emp -> emp.getId() ==
removeId);
                if (removed) {
                    System.out.println("Employee removed successfully!");
                } else {
                    System.out.println("Employee not found!");
```

```
                    }
                    break;
                case 4:
                    System.out.print("Enter ID to search: ");
                    int searchId = scanner.nextInt();
                    boolean found = false;
                    for (Employee emp : employees) {
                        if (emp.getId() == searchId) {
                            System.out.println("Employee found: " + emp);
                            found = true;
                            break;
                        }
                    }
                    if (!found) System.out.println("Employee not found!");
                    break;
                case 5:
                    System.out.println("Exiting...");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid choice! Please enter a valid option.");
            }
        }
    }
}
```

4. **Output:**

```
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Enter choice: 1
Enter ID: 10036
Enter Name: Prateek Pratap Singh
Enter Salary: 60000
Employee added successfully!

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Enter choice: 1
Enter ID: 10047
Enter Name: Yash
Enter Salary: 55000
Employee added successfully!
```

```
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Enter choice: 2
Enter ID to update: 10045
Employee not found!

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Enter choice: 2
Enter ID to update: 10047
Enter new Name: YashVeer
Enter new Salary: 55000
Employee updated successfully!
```

```
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Enter choice: 3
Enter ID to remove: 10047
Employee removed successfully!

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Enter choice: 4
Enter ID to search: 10036
Employee found: Employee{id=10036, name='Prateek Pratap Singh', salary=60000.0}

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Exit
Enter choice: 5
Exiting...
```

5. Learning Outcomes:

- Java Collections (ArrayList) – How to store, retrieve, update, and delete objects dynamically.
- Object-Oriented Programming (OOP) – How to create and manage classes (Employee and Main).
- Encapsulation – Using private fields and public methods to control data access.
- Handling User Input – Using Scanner for reading integers, strings, and doubles.
- Looping & Conditional Logic – Implementing a menu-driven system with switch-case and loops.
- Search & Remove Operations – Finding and removing employees efficiently using for-each and removeIf().

1. **(b) Aim:** 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.

2. **Objective :**
   - To use a Collection (ArrayList) to store playing cards with Symbol & Value.
   - Provide functionalities to:
     - Add new cards to the collection.
     - Search for all cards with a given symbol.
     - Display all stored cards.
     - Allow smooth user interaction with a menu-driven approach.

3. **Implementation :**

```
import java.util.*;

class Card {
    private String symbol;
    private String value;

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

    public String getSymbol() {
        return symbol;
    }

    @Override
    public String toString() {
        return "Card{symbol='" + symbol + "', value='" + value + "'}";
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Collection<Card> cards = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\n1. Add Card\n2. Search Cards by Symbol\n3. Display All Cards\n4. Exit");
            System.out.print("Enter choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    System.out.print("Enter Symbol: ");
                    String symbol = scanner.nextLine();
                    System.out.print("Enter Value: ");
                    String value = scanner.nextLine();
                    cards.add(new Card(symbol, value));
                    System.out.println("Card added successfully!");
                    break;
                case 2:
                    System.out.print("Enter Symbol to search: ");
                    String searchSymbol = scanner.nextLine();
                    boolean found = false;
                    for (Card card : cards) {
                        if (card.getSymbol().equalsIgnoreCase(searchSymbol)) {
                            System.out.println(card);
                            found = true;
                        }
                    }
                    if (!found) System.out.println("No cards found with symbol: " + searchSymbol);
                    break;
                case 3:
                    if (cards.isEmpty()) {
                        System.out.println("No cards in the collection.");
```

```
                } else {
                    for (Card card : cards) {
                        System.out.println(card);
                    }
                }
                break;
            case 4:
                System.out.println("Exiting...");
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice! Please enter a valid option.");
        }
    }
}
}
```

4. **Output :**

```
1. Add Card
2. Search Cards by Symbol
3. Display All Cards
4. Exit
Enter choice: 1
Enter Symbol: Heart
Enter Value: Ace
Card added successfully!

1. Add Card
2. Search Cards by Symbol
3. Display All Cards
4. Exit
Enter choice: 1
Enter Symbol: Spade
Enter Value: King
Card added successfully!
```

```
1. Add Card
2. Search Cards by Symbol
3. Display All Cards
4. Exit
Enter choice: 3
Card{symbol='Heart', value='Ace'}
Card{symbol='Spade', value='King'}

1. Add Card
2. Search Cards by Symbol
3. Display All Cards
4. Exit
Enter choice: 2
Enter Symbol to search: Spade
Card{symbol='Spade', value='King'}

1. Add Card
2. Search Cards by Symbol
3. Display All Cards
4. Exit
Enter choice: 4
Exiting...
```

5. **Learning Outcomes:**
   - Java Collections (Collection Interface & ArrayList) – Storing and managing multiple objects dynamically.
   - Class & Object Handling – Designing a Card class with attributes and behaviors.
   - String Manipulation – Searching cards by symbol using equalsIgnoreCase().
   - Iterating Through Collections – Using for-each loops to traverse and search data.
   - Building Interactive Applications – Implementing a structured menu-driven approach.
   - Exception Handling (Indirect Learning) – Managing Scanner input properly to prevent errors.