Experiment 6

Student Name: Kritika Sharma **UID: 22BCS14943**

Branch: IT **Section/Group: BET_KRG_IOT-3/B**

Semester: 6 **Date of Performance: 04-03-2025**

Subject Name: PBLJ **Subject Code: 22ITH-352**

1. **Aim:**
   - Write a program to sort a list of Employee objects (name, age, salary) using lambda expressions
2. **Objective:**
   - To Demonstrate Lamda expression
   - To sort the employees on the basis of name , age and salary

3. **Implementation/Code:**

```
import java.util.ArrayList;
import java.util.*; class
Employee {    private
String name;    private
int age;    private
double salary;


   public Employee(String name, int age, double salary)
{       this.name = name;       this.age = age;
this.salary = salary;
   }
   public String getName() {
return name;
   }
   public int getAge() {
return age;
   }
```

```
    public double getSalary() {
        return salary;
    }
    @Override     public
String toString() {
        return name + " - Age: " + age + ", Salary: " + salary;
    }
}

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

        System.out.print("Enter number of employees:
");         int n = scanner.nextInt();
scanner.nextLine(); // Consume newline

        for (int i = 0; i < n; i++) {
            System.out.print("Enter name: ");
            String name = scanner.nextLine();
System.out.print("Enter age: ");             int age
= scanner.nextInt();
System.out.print("Enter salary: ");
double salary = scanner.nextDouble();
scanner.nextLine(); // Consume newline

            employees.add(new Employee(name, age, salary));
        }
```

```
        // Sorting by name
        employees.sort(Comparator.comparing(Employee::getName));
    System.out.println("Sorted by Name: " + employees);

        // Sorting by age
        employees.sort(Comparator.comparingInt(Employee::getAge));
        System.out.println("Sorted by Age: " + employees);

        // Sorting by salary
        employees.sort(Comparator.comparingDouble(Employee::getSalary));
        System.out.println("Sorted by Salary: " + employees);

        scanner.close();
    }
}
```

5. Output:

```
<terminated> EmployeeSort [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe  (28 Feb 2025, 12:15:12 pm – 12:16:14 pm elapsed: 0:01:02.097) [pid: 15228]
Enter number of employees: 3
Enter name: A
Enter age: 43
Enter salary: 12000
Enter name: B
Enter age: 25
Enter salary: 10000
Enter name: C
Enter age: 45
Enter salary: 15000
Sorted by Name: [A - Age: 43, Salary: 12000.0, B - Age: 25, Salary: 10000.0, C - Age: 45, Salary: 15000.0]
Sorted by Age: [B - Age: 25, Salary: 10000.0, A - Age: 43, Salary: 12000.0, C - Age: 45, Salary: 15000.0]
Sorted by Salary: [B - Age: 25, Salary: 10000.0, A - Age: 43, Salary: 12000.0, C - Age: 45, Salary: 15000.0]
```

6. Learning Outcome:

- Learn how to use Java's Comparator with lambda expressions for sorting.
- Gain Understanding of how to take user input and store objects in a list
- Knowledge of using Collections.sort() and List.sort() methods efficiently.

Problem 2

4. **Aim:**
   - Create a program to use lambda expressions and stream operations to filter students scoring above 75%, sort them by marks, and display their names

5. **Objective:**
   - to filter a list of students based on specific conditions using Java Streams
   - Understand how to sort filtered results efficiently using Stream API
   - Display results in a structured and readable format

6. **Implementation/Code:**

```
import java.io.*; import
java.util.*; import
java.util.stream.Collectors;


class Student {
private String name;
private double marks;


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


   public String getName() {
return name;
   }
```

```
    public double getMarks() {
return marks;
    }
}

public class StudentFilterSort {     public
static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Student> students = new ArrayList<>();

        System.out.print("Enter number of students: ");
int n = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        for (int i = 0; i < n; i++) {
            System.out.print("Enter name: ");
            String name = scanner.nextLine();
System.out.print("Enter marks: ");
double marks = scanner.nextDouble();
scanner.nextLine(); // Consume newline

            students.add(new Student(name, marks));
        }
        List<String> topStudents = students.stream()
            .filter(s -> s.getMarks() > 75)
            .sorted(Comparator.comparingDouble(Student::getMarks).reversed())
            .map(Student::getName)
            .collect(Collectors.toList());
```

```
        System.out.println("Students scoring above 75% (sorted by marks): " + topStudents);
scanner.close();
    }
}
```

7. **Output:**

```
<terminated> StudentFilterSortt [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe
Enter number of students: 4
Enter name: A
Enter marks: 30
Enter name: B
Enter marks: 80
Enter name: C
Enter marks: 98
Enter name: D
Enter marks: 20
Students scoring above 75% (sorted by marks): [C, B]
```

8. **Learning Outcome:** • Mastery of filter(), sorted(), and map() functions in Stream API

• Ability to apply lambda expressions for data transformation and filtering

• improved understanding of functional programming concepts in Java

Problem 3

7. **Aim:**
   - 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.

8. **Objective:**
   - Learn how to group products based on categories using Stream API.
   - identify the most expensive product in each category using Java Streams
   - Calculate the average price of all products in the dataset

9. **Implementation/Code:**

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

class Product {     private
String name;     private
String category;
   private double price;

   public Product(String name, String category, double price) {
this.name = name;         this.category = category;
this.price = price;
   }
```

```
    public String getCategory() {
return category;
    }

    public double getPrice() {
return price;
    }

    public String getName() {
        return name;
    }
}

public class ProductProcessor {     public
static void main(String[] args) {
List<Product> products = Arrays.asList(
new Product("Laptop", "Electronics",
1200.99),            new Product("Phone",
"Electronics", 799.49),            new
Product("TV", "Electronics", 1500.00),
new Product("Shoes", "Clothing",
89.99),            new Product("Jacket",
"Clothing", 119.99),            new
Product("Apple", "Grocery", 1.49),
new Product("Milk", "Grocery", 2.99)
        );
        // Grouping products by category
        Map<String, List<Product>> groupedByCategory = products.stream()
            .collect(Collectors.groupingBy(Product::getCategory));
        System.out.println("Products grouped by category: " + groupedByCategory);

        // Finding the most expensive product in each category
        Map<String, Optional<Product>> mostExpensiveByCategory = products.stream()
            .collect(Collectors.groupingBy(
                Product::getCategory,
                Collectors.maxBy(Comparator.comparingDouble(Product::getPrice))
            ));
        System.out.println("Most expensive product in each category: " +
mostExpensiveByCategory);

        // Calculating the average price of all products
double averagePrice = products.stream()
```

```
                .mapToDouble(Product::getPrice)
                .average()
                .orElse(0.0);
        System.out.println("Average price of all products: " + averagePrice);
    }
}
```

9. Output:

```
<terminated> ProductProcessor [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe  (27 Feb 2025, 9:18:40 pm – 9:18:43 pm elapsed: 0:00:03.455) [pid: 21280]
Products grouped by category: {Grocery=[Apple (Category: Grocery, Price: 1.49), Milk (Category: Grocery, Price: 2.99)], Clothing=[Shoes (Category:
Most expensive product in each category: {Grocery=Optional[Milk (Category: Grocery, Price: 2.99)], Clothing=Optional[Jacket (Category: Clothing, Pr
Average price of all products: 530.7057142857143
```

10. Learning Outcome:

- Understanding of Collectors.groupingBy() and Collectors.maxBy() for aggregation
- Ability to use mapToDouble() and average() for numerical analysis

- Practical experience in handling large datasets efficiently using Java Streams