Experiment 6

Student Name: Omdev Kumar **UID:** 22BET10206
Branch: BE- IT **Section/Group:** 22BET-703/B
Semester: 6th **Date of Performance:** 25/02/25
Subject Name: JAVA LAB **Subject Code:** 22ITH-359

PROBLEM 1:

1. **Aim:** To develop a Java program that sorts a list of Employee objects based on different criteria (age, salary, and name) using Lambda Expressions.

2. **Objective:**

- To understand and implement Lambda Expressions in Java.
- To learn how to use Comparator with Lambda for sorting objects dynamically.
- To practice taking user input and storing it in a collection (ArrayList).

3. **Implementation/Code:** package Main; import java.util.*;

```
class Main {
   String name;
   int age;
   double salary;

   public Employee6(String name, int age, double salary)
      { this.name = name;
      this.age = age;
      this.salary = salary;
   }
   public void display() {
      System.out.println(name + " | Age: " + age + " | Salary: " + salary);

   }
}
```

```
public class main_1 {
   public static void main(String[] args)
      { Scanner sc = new Scanner(System.in);
      List<Employee6> employees = new ArrayList<>(); System.out.print("Enter
      number of employees: ");
      int n = sc.nextInt(); sc.nextLine(); //
      Consume the newline


      for (int i = 0; i < n; i++) {
        System.out.println("Enter details for Employee " + (i + 1) + ":");
         System.out.print("Name: ");
         String name = sc.nextLine();
         System.out.print("Age: "); int
         age = sc.nextInt();
         System.out.print("Salary: ");
         double salary = sc.nextDouble();
         sc.nextLine();


         employees.add(new Employee6(name, age, salary));
      }


      System.out.println("\nBefore sorting:");
      employees.forEach(Employee6::display);


      employees.sort((e1, e2) -> e1.age - e2.age);
      System.out.println("\nSorted by Age:");
      employees.forEach(Employee6::display);


      employees.sort((e1, e2) -> Double.compare(e2.salary, e1.salary));
      System.out.println("\nSorted by Salary (Descending):");
      employees.forEach(Employee6::display);
      employees.sort((e1, e2) ->
      e1.name.compareTo(e2.name));
      System.out.println("\nSorted by Name:");
      employees.forEach(Employee6::display);
```

```
        sc.close();
    }
}
```

4. Output:

```
Problems   Javadoc   Declaration   Console
<terminated> Main [Java Application] /Users/pranav/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.m
Enter number of employees: 3
Enter details for Employee 1:
Name: Pranav Anand
Age: 21
Salary: 1200000
Enter details for Employee 2:
Name: Himanshu Kumar
Age: 22
Salary: 1000000
Enter details for Employee 3:
Name: Omdev kumar
Age: 21
Salary: 800000

Before sorting:
Pranav Anand | Age: 21 | Salary: 1200000.0
Himanshu Kumar | Age: 22 | Salary: 1000000.0
Omdev kumar | Age: 21 | Salary: 800000.0

Sorted by Age:
Pranav Anand | Age: 21 | Salary: 1200000.0
Omdev kumar | Age: 21 | Salary: 800000.0
Himanshu Kumar | Age: 22 | Salary: 1000000.0

Sorted by Salary (Descending):
Pranav Anand | Age: 21 | Salary: 1200000.0
Himanshu Kumar | Age: 22 | Salary: 1000000.0
Omdev kumar | Age: 21 | Salary: 800000.0

Sorted by Name:
Himanshu Kumar | Age: 22 | Salary: 1000000.0
Omdev kumar | Age: 21 | Salary: 800000.0
Pranav Anand | Age: 21 | Salary: 1200000.0
```

Figure 1: Output of code on ECLIPSE

5. Learning Outcomes:

a) Gain an understanding of how Lambda Expressions simplify code by providing a concise way to define anonymous functions, reducing the need for separate comparator classes.

b) Implement sorting operations using Lambda Expressions and Comparator, enabling dynamic sorting of objects based on multiple attributes such as age, salary, and name.

c) Develop practical skills in using Java's Collection Framework, particularly ArrayList, to store and manipulate objects efficiently while integrating user input and sorting functionalities.

PROBLEM 2:

1. **Aim:** To develop a Java program that filters and sorts a list of Student objects using Lambda Expressions and Stream API to display students scoring above 75%.

2. **Objective:**

- To understand and implement Lambda Expressions and Stream API in Java.
- To implement symbol-based searching to allow users to find all cards linked to a specific symbol.
- To process and manage collections dynamically using ArrayList and stream operations.

3. **Implementation/Code:** package Main; import java.util.*; import java.util.stream.Collectors;

```
class Student6 {
   String name;
   double marks;

   public Student6(String name, double marks)
      { this.name = name; this.marks
      = marks;
   }
}

public class exp6_2 { public static void
   main(String[] args)
      { Scanner sc = new Scanner(System.in);
      List<Student6> students = new ArrayList<>(); System.out.print("Enter
      number of students: ");
      int n = sc.nextInt();
      sc.nextLine(); // Consume newline for

      (int i = 0; i < n; i++) {
```

```
        System.out.println("Enter details for

        Student " + (i + 1) + ":");

            System.out.print("Name: "); String name =
            sc.nextLine(); System.out.print("Marks: ");
            double marks = sc.nextDouble();
            sc.nextLine(); // Consume newline
            students.add(new Student6(name, marks));
        }
        List<String> filteredStudents = students.stream()
            .filter(s -> s.marks > 75)
            .sorted((s1, s2) -> Double.compare(s2.marks, s1.marks))
            .map(s -> s.name)
            .collect(Collectors.toList());
        System.out.println("\nStudents scoring above 75% (sorted by
        marks):"); filteredStudents.forEach(System.out::println); sc.close();
    }
}
```

4. Ouput:

```
Problems  @ Javadoc  Declaration  Console
<terminated> Main [Java Application] /Users/pranav/.p2/pool/plugins/org.eclipse.justj.openjdk.ho
Enter number of students: 3
Enter details for Student 1:
Name: Pranav Anand
Marks: 92
Enter details for Student 2:
Name: Himanshu Kumar
Marks: 93
Enter details for Student 3:
Name: Omdev Kumar
Marks: 85

Students scoring above 75% (sorted by marks):
Himanshu Kumar
Pranav Anand
Omdev Kumar
```

Figure 2: Output of code on ECLIPSE

5. Learning Outcomes:

a) Understand how to use Lambda Expressions and Stream API to filter and manipulate data efficiently, reducing the need for traditional loops.

b) Learn how to apply sorting mechanisms using Comparator and Stream API, enabling dynamic ordering of student records based on their marks.

c) Develop skills in handling real-world datasets using Java collections, improving their ability to process, filter, and display relevant information based on conditions.

PROBLEM 3:

1. **Aim:** To develop a Java program that processes a large dataset of products using the Streams API, performing operations such as grouping products by category, finding the most expensive product in each category, and calculating the average price of all products.

2. **Objective:**

- To understand and apply Java Streams API for data processing on large datasets.
- To implement efficient grouping, filtering, and aggregation operations using functional programming concepts.
- To enhance problem-solving skills by working with collections and stream-based computations.

3. **Implementation/Code:** package Experiments; import java.util.*;

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

class Product
  { String name;
  String
  category;
  double price;
  public Product(String name, String category, double price)
    { this.name = name;
    this.category = category;
    this.price = price;
  }
  @Override
  public String toString() {
    return name + " - $" +
    price;
  }
}

public class exp6_3 { public static void
  main(String[] args)
    { Scanner sc = new Scanner(System.in);
```

```
List<Product> products = new ArrayList<>(); System.out.print("Enter
number of products: ");
int n = sc.nextInt(); sc.nextLine();
// Consume newline

for (int i = 0; i < n; i++) {
    System.out.println("Enter details for Product " + (i + 1) + ":");
    System.out.print("Name: ");
    String name = sc.nextLine();
    System.out.print("Category: ");
    String category = sc.nextLine();
    System.out.print("Price: ");
    double price = sc.nextDouble();
    sc.nextLine(); // Consume
    newline

    products.add(new Product(name, category, price));
}
Map<String, List<Product>> productsByCategory = products.stream()
    .collect(Collectors.groupingBy(p -> p.category));

System.out.println("\nProducts grouped by category:");
productsByCategory.forEach((category, productList) -> {
    System.out.println(category + ": " + productList);
});
Map<String, Optional<Product>> mostExpensiveByCategory = products.stream()
    .collect(Collectors.groupingBy(
        p -> p.category,
        Collectors.maxBy(Comparator.comparingDouble(p -> p.price))
    ));
```

```
        System.out.println("\nMost expensive product in each category:");
        mostExpensiveByCategory.forEach((category, product) ->
            System.out.println(category + ": " + product.orElse(null))
    );
        double averagePrice = products.stream()
            .mapToDouble(p -> p.price)
            .average()
            .orElse(0.0);
        System.out.println("\nAverage price of all products: $" + averagePrice);
        sc.close();
    }
}
```

4. Output:

```
Enter number of products: 2
Enter details for product 1:
Name: Mouse
Category: Gaming
Price: 30000
Enter details for product 2:
Name: Keyboard
Category: Mechanical
Price: 800

Products grouped by category:
Mechanical: [Product{name='Keyboard', category='Mechanical', price=800.0}]
Gaming: [Product{name='Mouse', category='Gaming', price=30000.0}]

Most expensive product in each category:
Mechanical: Product{name='Keyboard', category='Mechanical', price=800.0}
Gaming: Product{name='Mouse', category='Gaming', price=30000.0}

Average price of all products: 15400.0
```

Figure 1: Output of code on ECLIPSE

5. Learning Outcomes:

a) Learn how to use Java Streams API to process large datasets efficiently, reducing the need for manual iteration and improving performance.

b) Gain hands-on experience in performing data grouping, aggregation, and filtering operations using Stream functions such as groupingBy(), maxBy(), and mapToDouble().

c) Develop practical skills in working with Java collections, particularly ArrayList and Map, while integrating stream operations to analyze and process product data effectively.