Experiment - 6

Student Name: HARJOT **UID: 22BCS16214**
Branch: B.ECSE **Section: IOT-643-A**
Semester: 6th **DOP: 03/03/25**
Subject: PBLJ **Subject Code: 22CSH-359**

1) **Aim:** Develop Java programs using core concepts such as data structures, collections, and multithreading to manage and manipulate data.

2) **Problem Statement:**
   a. Write a program to sort a list of Employee objects (name, age, salary) using lambda expressions.
   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.
   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.

3) **Algorithm:**

a. Sort a list of Employee Objects:

- Create an Employee class: Define the class Employee with fields: name, age, and salary.
- Create a list of Employee objects: Create a list of Employee objects with different names, ages, and salaries.
- Use Comparator with lambda expressions:
- For sorting by name, use a lambda expression to compare the names.
- For sorting by age, use a lambda expression to compare the ages.
- For sorting by salary, use a lambda expression to compare the salaries.
- Sort the list: Use the Collections.sort() method or List.sort() method with the corresponding lambda expression.
- Print the sorted list: After sorting, print the list of employees.

b. Students scoring above 75%:

- **Create a Student class**: Define the Student class with fields: name and marks.
- **Create a list of students**: Initialize a list of Student objects.
- **Use Streams**:
- **Filter** students who have marks greater than 75%.
- **Sort** the filtered students by marks in descending order.
- **Map** the students to their names.
- **Display the names**: Print the names of students who satisfy the conditions.

c. Large dataset of products using streams:

- **Create a Product class**: Define the Product class with fields: name, category, and price.
- **Create a list of products**: Initialize a list of Product objects.
- **Group products by category**: Use Collectors.groupingBy() to group the products by their category.
- **Find the most expensive product**: Use Collectors.maxBy() to find the most expensive product in each category.
- **Calculate average price**: Use Collectors.averagingDouble() to calculate the average price of all products.
- **Display the results**: Print the most expensive product in each category and the average price.

4) Program:

a. Sort a list of Employee objects:

```
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;
   }

   public String toString() {
      return "Name: " + name + ", Age: " + age + ", Salary: " + salary;
   }
}
```

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


        System.out.print("Enter the number of employees: ");
        int numEmployees = scanner.nextInt();
        scanner.nextLine();


        List<Employee> employees = new ArrayList<>();


        for (int i = 0; i < numEmployees; 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();


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


        employees.sort((e1, e2) -> Double.compare(e1.salary, e2.salary));


        System.out.println("\nEmployees sorted by salary:");
        for (Employee e : employees) {
            System.out.println(e);
        }

        scanner.close();
    }
}
```

b. Students scoring above 75%:

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

class Student {
    String name;
    double marks;
    public Student(String name, double marks) {
        name = name;
        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);

        System.out.print("Enter the number of students: ");
        int numStudents = scanner.nextInt();
        scanner.nextLine();
        List<Student> students = new ArrayList<>();
        for (int i = 0; i < numStudents; i++) {
            System.out.print("Enter student's name: ");
            String name = scanner.nextLine();
            System.out.print("Enter " + name + "'s marks: ");
            double marks = scanner.nextDouble();
            scanner.nextLine();

            students.add(new Student(name, marks));
        }

        students.stream()
            .filter(s -> s.marks > 75)
            .sorted((s1, s2) -> Double.compare(s1.marks, s2.marks))
            .forEach(s -> System.out.println(s.name));
```

```
        scanner.close();
    }
}
```

c. Large dataset of products using streams:

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

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;
    }

    public String getName() {
        return name;
    }

    public String getCategory() {
        return category;
    }

    public double getPrice() {
        return price;
    }

    @Override
    public String toString() {
        return "Name: " + name + ", Category: " + category + ", Price: $" + price;
    }
}

public class ProductStreamProcessor {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Product> products = new ArrayList<>();
```

```
        System.out.println("Enter the number of products: ");
        int numberOfProducts = scanner.nextInt();
        scanner.nextLine();

        for (int i = 0; i < numberOfProducts; i++) {
            System.out.println("\nEnter details for product " + (i + 1) + ":");

            System.out.print("Name: ");
            String name = scanner.nextLine();

            System.out.print("Category: ");
            String category = scanner.nextLine();

            System.out.print("Price: ");
            double price = scanner.nextDouble();
            scanner.nextLine();

            products.add(new Product(name, category, price));
        }

        Map<String, List<Product>> groupedByCategory = products.stream()
                .collect(Collectors.groupingBy(Product::getCategory));

        System.out.println("\nProducts grouped by category:");
        groupedByCategory.forEach((category, productList) -> {
            System.out.println(category + ":");
            productList.forEach(System.out::println);
        });

        System.out.println("\nMost expensive product in each category:");
        groupedByCategory.forEach((category, productList) -> {
            productList.stream()
                    .max(Comparator.comparingDouble(Product::getPrice))
                    .ifPresent(product -> System.out.println(category + ": " + product));
        });

        double averagePrice = products.stream()
                .mapToDouble(Product::getPrice)
                .average()
                .orElse(0);

        System.out.println("\nAverage price of all products: $" + averagePrice);
    }
```

}

5) OUTPUT:Sort a list of Employee objects:

2. **Students scoring above 75%:**

```
Enter the number of students: 4
Enter student's name: Alice
Enter Alice's marks: 85
Enter student's name: Bob
Enter Bob's marks: 72
Enter student's name: Charlie
Enter Charlie's marks: 90
Enter student's name: Dave
Enter Dave's marks: 65
```

```
2

Enter details for product 1:
Name: trt
Category: trophy
Price: 1200

Enter details for product 2:
Name: iphone
Category: smartphone
Price: 129000

Products grouped by category:
smartphone:
Name: iphone, Category: smartphone, Price: $129000.0
trophy:
Name: trt, Category: trophy, Price: $1200.0

Most expensive product in each category:
smartphone: Name: iphone, Category: smartphone, Price: $129000.0
trophy: Name: trt, Category: trophy, Price: $1200.0

Average price of all products: $65100.0
(base) harjotsingh@HARJOTs-MacBook-Pro exp 5n %
```

3. **Large dataset of products using streams**

6) Learning Outcomes:

- **Understanding Lambda Expressions**: Students will learn how to use lambda expressions in Java to write concise and readable code for sorting collections.
- **Using `Comparator` Interface**: Students will gain an understanding of how to implement custom sorting logic using the `Comparator` interface with lambda expressions.
- **List Sorting Techniques**: Learners will understand different sorting strategies (e.g., by name, age, and salary) and how to apply them in real-world use cases.
- **Enhanced Use of Java Collections**: Students will learn how to work with Java collections, particularly `List`, and the methods like `Collections.sort()` and `List.sort()` to modify the order of elements.