Experiment - 6

Student Name: Shivam **UID: 22BCS50010**
Branch: B.ECSE **Section: IOT-643-A**
Semester: 6th **DOP: 24/02/25**
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.*;

public class Main {
   // Define the Employee class
   static class Employee {
      String name;
      int age;
      double salary;

      // Constructor
      public Employee(String name, int age, double salary) {
         this.name = name;
         this.age = age;
         this.salary = salary;
      }
```

```
    // Getter methods
    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public double getSalary() {
        return salary;
    }

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

public static void main(String[] args) {
    // Create a list of Employee objects
    List<Employee> employees = new ArrayList<>();
    employees.add(new Employee("Shivam", 28, 50000));
    employees.add(new Employee("Shivanshu", 35, 60000));
    employees.add(new Employee("Pranjal", 25, 45000));
    employees.add(new Employee("Agrim", 40, 70000));

    // Sorting by name (alphabetically)
    System.out.println("Sorting by name:");
    employees.sort((e1, e2) -> e1.getName().compareTo(e2.getName()));
    employees.forEach(System.out::println);

    // Sorting by age
    System.out.println("\nSorting by age:");
    employees.sort((e1, e2) -> Integer.compare(e1.getAge(), e2.getAge()));
    employees.forEach(System.out::println);

    // Sorting by salary
    System.out.println("\nSorting by salary:");
    employees.sort((e1, e2) -> Double.compare(e1.getSalary(), e2.getSalary()));
    employees.forEach(System.out::println);
}
}
```

b. Students scoring above 75%:

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

public class Main {  // Change class name to 'Main'
    String name;
    double marks;

    // Constructor to initialize Student object
    public Main(String name, double marks) {
        this.name = name;
        this.marks = marks;
    }

    // Getter methods
    public String getName() {
        return name;
    }

    public double getMarks() {
        return marks;
    }

    @Override
    public String toString() {
        return "Student{name='" + name + "', marks=" + marks + "}";
    }

    public static void main(String[] args) {
        // Creating a list of students
        List<Main> students = new ArrayList<>();
        students.add(new Main("Shivam", 85));
        students.add(new Main("Shivanshu", 70));
        students.add(new Main("Pranjal", 90));
        students.add(new Main("Pranav", 65));
        students.add(new Main("Kartiik", 80));

        // Filter students scoring above 75%, sort them by marks in descending order, and display
their names with marks
        System.out.println("Students scoring above 75%, sorted by marks:");
```

```
        students.stream()
                // Filter students with marks greater than 75
                .filter(student -> student.getMarks() > 75)
                // Sort by marks in descending order
                .sorted((s1, s2) -> Double.compare(s2.getMarks(), s1.getMarks()))
                // For each student, print name and marks
                .forEach(student -> System.out.println(student.getName() + " - " +
student.getMarks()));
    }
}
```

c. Large dataset of products using streams:

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

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

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

    // Getter methods
    public String getName() {
        return name;
    }

    public String getCategory() {
        return category;
    }

    public double getPrice() {
        return price;
    }

    @Override
    public String toString() {
        return "Product{name='" + name + "', category='" + category + "', price=" + price + "}";
    }
}
```

```
public class ProductProcessor {

    public static void main(String[] args) {
        // Creating a list of products
        List<Product> products = Arrays.asList(
            new Product("Laptop", "Electronics", 1200),
            new Product("Smartphone", "Electronics", 800),
            new Product("Tablet", "Electronics", 500),
            new Product("T-shirt", "Clothing", 30),
            new Product("Jeans", "Clothing", 40),
            new Product("Jacket", "Clothing", 100),
            new Product("Blender", "Home Appliances", 150),
            new Product("Microwave", "Home Appliances", 200),
            new Product("Washing Machine", "Home Appliances", 500)
        );

        // Group products by category
        Map<String, List<Product>> productsByCategory = products.stream()
            .collect(Collectors.groupingBy(Product::getCategory));

        // Print grouped products by category
        System.out.println("Products grouped by category:");
        productsByCategory.forEach((category, productList) -> {
            System.out.println(category + ": " + productList);
        });

        // Find the most expensive product in each category
        System.out.println("\nMost expensive product in each category:");
        productsByCategory.forEach((category, productList) -> {
            Product mostExpensiveProduct = productList.stream()
                .max(Comparator.comparingDouble(Product::getPrice))
                .orElseThrow(NoSuchElementException::new);  // In case the list is empty
            System.out.println(category + ": " + mostExpensiveProduct);
        });

        // Calculate the average price of all products
        double averagePrice = products.stream()
            .collect(Collectors.averagingDouble(Product::getPrice));

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

5) OUTPUT:

1. **Sort a list of Employee objects:**

```
Sorting by name:
Employee{name='Agrim', age=40, salary=70000.0}
Employee{name='Pranjal', age=25, salary=45000.0}
Employee{name='Shivam', age=28, salary=50000.0}
Employee{name='Shivanshu', age=35, salary=60000.0}

Sorting by age:
Employee{name='Pranjal', age=25, salary=45000.0}
Employee{name='Shivam', age=28, salary=50000.0}
Employee{name='Shivanshu', age=35, salary=60000.0}
Employee{name='Agrim', age=40, salary=70000.0}

Sorting by salary:
Employee{name='Pranjal', age=25, salary=45000.0}
Employee{name='Shivam', age=28, salary=50000.0}
Employee{name='Shivanshu', age=35, salary=60000.0}
Employee{name='Agrim', age=40, salary=70000.0}


...Program finished with exit code 0
Press ENTER to exit console.
```

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

```
Students scoring above 75%, sorted by marks:
Pranjal - 90.0
Shivam - 85.0
Kartiik - 80.0


...Program finished with exit code 0
Press ENTER to exit console.
```

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

```
Products grouped by category:
Clothing: [Product{name='T-shirt', category='Clothing', price=30.0}, Product{name='Jeans', category='Clothing', price=40.0}, Product{name='J
acket', category='Clothing', price=100.0}]
Electronics: [Product{name='Laptop', category='Electronics', price=1200.0}, Product{name='Smartphone', category='Electronics', price=800.0},
 Product{name='Tablet', category='Electronics', price=500.0}]
Home Appliances: [Product{name='Blender', category='Home Appliances', price=150.0}, Product{name='Microwave', category='Home Appliances', pr
ice=200.0}, Product{name='Washing Machine', category='Home Appliances', price=500.0}]

Most expensive product in each category:
Clothing: Product{name='Jacket', category='Clothing', price=100.0}
Electronics: Product{name='Laptop', category='Electronics', price=1200.0}
Home Appliances: Product{name='Washing Machine', category='Home Appliances', price=500.0}

Average price of all products: $391.1111111111111


...Program finished with exit code 0
Press ENTER to exit console.
```

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.