Experiment 6

Student Name: Viraj
UID: 22BCS13406
Branch: BE-CSE
Section/Group: DL_903_B
Semester: 6th
Date of Performance: 24-02-2025
Subject Name: Program Based Learning in Java with Lab
Subject Code: 22CSH-359

1.Aim:

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.

1.Implementation/Code:

1.) Easy: Write a program to sort a list of Employee objects (name, age, salary) using lambda expressions.

Code:

```
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 "Employee{name='" + name + "', age=" + age + ", salary=" + salary + "}";
      }
```

```
}

public class EmployeeSorter {
   public static void main(String[] args)
      { List<Employee> employees = new
      ArrayList<>();
      employees.add(new Employee("Manvendra", 30, 50000));
      employees.add(new Employee("Aniket", 25, 60000));
      employees.add(new Employee("Vatsalya", 35, 55000));

      // Sorting by name
      employees.sort(Comparator.comparing(emp -> emp.name));
      System.out.println("Sorted by name: " + employees);

      // Sorting by age
      employees.sort(Comparator.comparingInt(emp -> emp.age));
      System.out.println("Sorted by age: " + employees);

      // Sorting by salary
      employees.sort(Comparator.comparingDouble(emp -> emp.salary));
      System.out.println("Sorted by salary: " + employees);
   }
}
```

2.) Medium Level: Create a program to use lambda expressions and stream operations to filter students scoring above 75%, sort them by marks, and display their names.

Code:

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

class      Student
   {          String
   name;  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 StudentFilter {
    public static void main(String[] args) {
        List<Student> students = Arrays.asList(
            new Student("Manvendra", 85),
            new Student("Aniket", 72),
            new Student("Vatsalya", 90),
            new Student("Parit", 65),
            new Student("Braburam M", 78)
        );

        // Filter, sort, and display
        names students.stream()
                .filter(s -> s.getMarks() > 75) // Filter students scoring above 75%
                .sorted(Comparator.comparingDouble(Student::getMarks).reversed()) // Sort by marks descending
                .map(Student::getName) // Extract names
                .forEach(System.out::println); // Print names
    }
     }
```

3).Hard: 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.

Code:

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

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

    public double getPrice()
        { return price;
    }

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

public class ProductStreamProcessing
    { public static void main(String[] args) {
        List<Product> products = Arrays.asList(
            new Product("Laptop", "Electronics", 1200.00),
            new Product("Phone", "Electronics", 800.00),
            new Product("TV", "Electronics", 1500.00),
            new Product("Sofa", "Furniture", 700.00),
            new Product("Chair", "Furniture", 150.00),
            new Product("Table", "Furniture", 350.00),
            new Product("T-Shirt", "Clothing", 30.00),
            new Product("Jeans", "Clothing", 60.00),
            new Product("Jacket", "Clothing", 120.00)
        );

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

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

        // Find the most expensive product in each category
        Map<String, Optional<Product>> mostExpensiveByCategory = products.stream()
            .collect(Collectors.groupingBy( Product::getCat
                egory,
                Collectors.maxBy(Comparator.comparingDouble(Product::getPrice))
            ));
```

```
        System.out.println("\nMost expensive product in each category:");
        mostExpensiveByCategory.forEach((category, product) ->
            System.out.println(category + ": " + product.orElse(null))
        );

        // Calculate the average price of all products
        double averagePrice = products.stream()
            .mapToDouble(Product::getPrice)
            .average()
            .orElse(0.0);

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

2. Output

1.) Easy problem output

```
Sorted by name: [Employee{name='Aniket', age=25, salary=60000.0}, Employee{name='Manvendra', age=30, salary=50000.0}, Employee{name='Vatsalya', age=35,
 salary=55000.0}]
Sorted by age: [Employee{name='Aniket', age=25, salary=60000.0}, Employee{name='Manvendra', age=30, salary=50000.0}, Employee{name='Vatsalya', age=35,
salary=55000.0}]
Sorted by salary: [Employee{name='Manvendra', age=30, salary=50000.0}, Employee{name='Vatsalya', age=35, salary=55000.0}, Employee{name='Aniket', age=2
5, salary=60000.0}]


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

2.) Medium problem output

```
Vatsalya
Manvendra
Braburam M


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

3.) Hard problem output

```
Clothing: [T-Shirt (Clothing, ₹30.0), Jeans (Clothing, ₹60.0), Jacket (Clothing, ₹120.0)]
Electronics: [Laptop (Electronics, ₹1200.0), Phone (Electronics, ₹800.0), TV (Electronics, ₹1500.0)]
Furniture: [Sofa (Furniture, ₹700.0), Chair (Furniture, ₹150.0), Table (Furniture, ₹350.0)]

Most expensive product in each category:
Clothing: Jacket (Clothing, ₹120.0)
Electronics: TV (Electronics, ₹1500.0)
Furniture: Sofa (Furniture, ₹700.0)

Average price of all products: ₹545.5555555555555


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

4 Learning Outcomes

1. Mastering Lambda Expressions – Learn to use lambda functions for sorting and processing collections efficiently.
2. Working with Java Streams API – Gain expertise in filtering, sorting, and grouping data using streams.
3. Implementing Functional Programming – Understand concepts like method references, map(), filter(), and collectors for concise and readable code.
4. Handling Large Data Processing – Learn how to group, aggregate, and find max/min values in datasets efficiently.
5. Enhancing Problem-Solving with Java Collections – Improve understanding of Lists, Maps, Optionals, and Comparators for real-world applications.