Experiment 6

Student Name: Sumit Sharma **UID: 22BCS12980**
Branch: CSE **Section: 641/B**
Semester: 6th **DOP:05/03/25**
Subject: PBLJ **Subject Code:22CSH-359**

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

Objective: The objective of this program is to demonstrate how to use **lambda expressions** to sort a list of Employee objects based on different attributes (name, age, salary). It showcases the **concise and functional** approach to sorting in Python using the sorted() function.

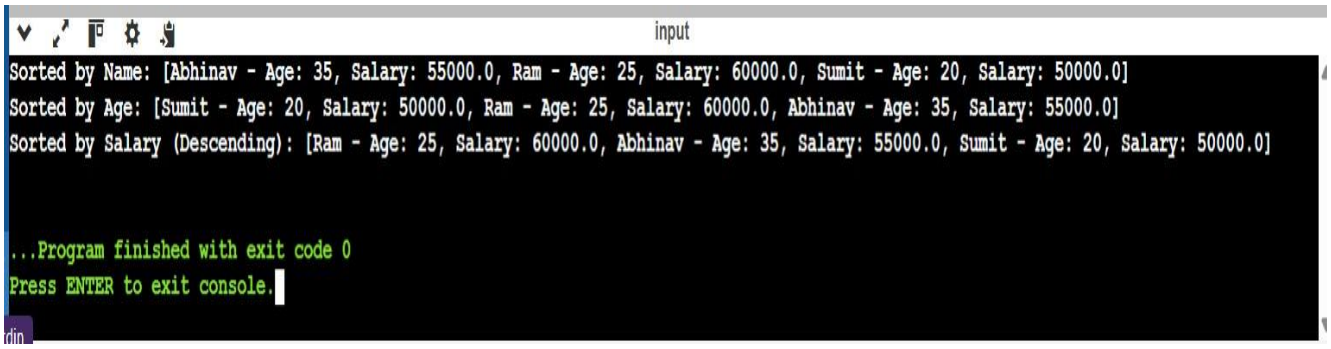
Algorithm:

1. Define the Employee class with attributes: name, age, and salary.
2. Create a list of Employee objects with sample data.
3. Use the sorted() function with lambda expressions to sort:
   - By name (alphabetically).
   - By age (ascending).
   - By salary (descending).
4. Print the sorted lists to verify results**.**

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;
    }

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

public class EmployeeSort {
    public static void main(String[] args) {
        List<Employee> employees = Arrays.asList(
            new Employee("Sumit", 20, 50000),
            new Employee("Ram", 25, 60000),
            new Employee("Abhinav", 35, 55000)
        );

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

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

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

Output:



```
Sorted by Name: [Abhinav - Age: 35, Salary: 55000.0, Ram - Age: 25, Salary: 60000.0, Sumit - Age: 20, Salary: 50000.0]
Sorted by Age: [Sumit - Age: 20, Salary: 50000.0, Ram - Age: 25, Salary: 60000.0, Abhinav - Age: 35, Salary: 55000.0]
Sorted by Salary (Descending): [Ram - Age: 25, Salary: 60000.0, Abhinav - Age: 35, Salary: 55000.0, Sumit - Age: 20, Salary: 50000.0]


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

Learning Outcomes:

i. Understand how to use **Comparator.comparing()** to sort objects based on different attributes.

ii. Learn how to use lambda expressions for concise and readable sorting in Java.

iii. Understand how **functional programming concepts** apply to Java using Comparator and lambda functions.

Question 2:

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.

Objective:

The program demonstrates the use of **lambda expressions and stream operations** in Java to:

1. **Filter students** who have scored above **75%**.
2. **Sort the filtered students** in descending order based on their marks.
3. **Display only their names**, showcasing **functional programming** with Java Streams.

Algorithm:

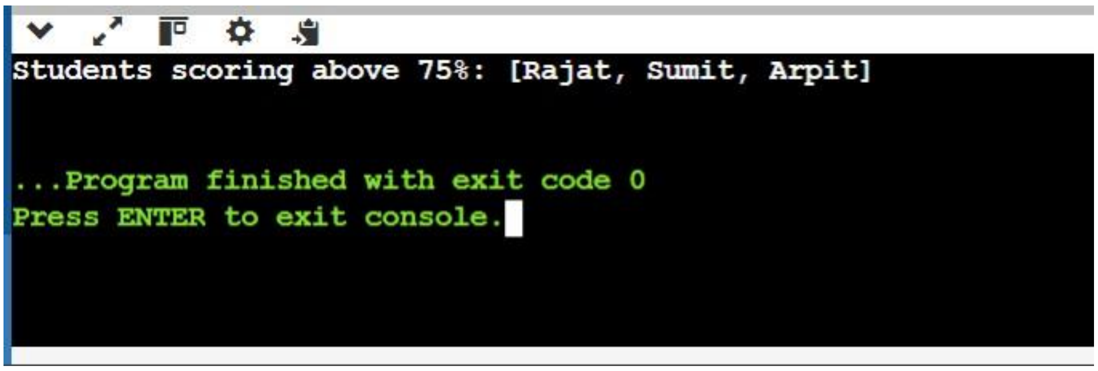
1. Define the Student class with attributes: name and marks.
2. Create a list of Student objects with sample data.
3. Use Java Streams to:
   - Filter students who scored more than 75%.
   - Sort the filtered students in descending order based on marks.
   - Extract and display their names.
4. Print the results to verify the output.

Code:

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

class  Student  {
   String   name;
   double marks;

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

   @Override
   public String toString() {
      return name + " - Marks: " + marks;
   }
}

public class StudentFilter {
   public static void main(String[] args) {
      List<Student> students = Arrays.asList(
         new Student("Sumit", 80),
         new Student("Rajat", 70),
         new Student("Abhinav", 72),
         new Student("Arpit", 76),
         new Student("David", 65)
      );
      List<String> topStudents = students.stream()
```

```
            .filter(s -> s.marks > 75)
            .sorted(Comparator.comparingDouble(s -> -s.marks))
            .map(s -> s.name)
            .collect(Collectors.toList()); // Collect names into a list

        System.out.println("Students scoring above 75%: " + topStudents);
    }
  }
```

Output:



```
Students scoring above 75%: [Rajat, Sumit, Arpit]


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

Learning Outcomes:

1. Learn how to use **lambda expressions** for concise and functional programming in Java.
2. Understand how **Streams** work for filtering, sorting, and mapping data efficiently.
3. Learn how to filter elements based on a condition (marks > 75).
4. Use **Comparator.comparingDouble()** to sort students by marks in **descending order**.
5. Extract specific fields (e.g., student names) from objects using **map()**.
6. Learn how to perform operations without modifying the original list, improving code readability and efficiency.

Question 3:

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.

Objective: Objective of the Program:
1. **Organize Products by Category**: Group products based on their categories.
2. **Find Expensive Products:** Identify the most expensive product in each category.
3. **Calculate Average Price:** Find the average price of all products.

Algorithm:

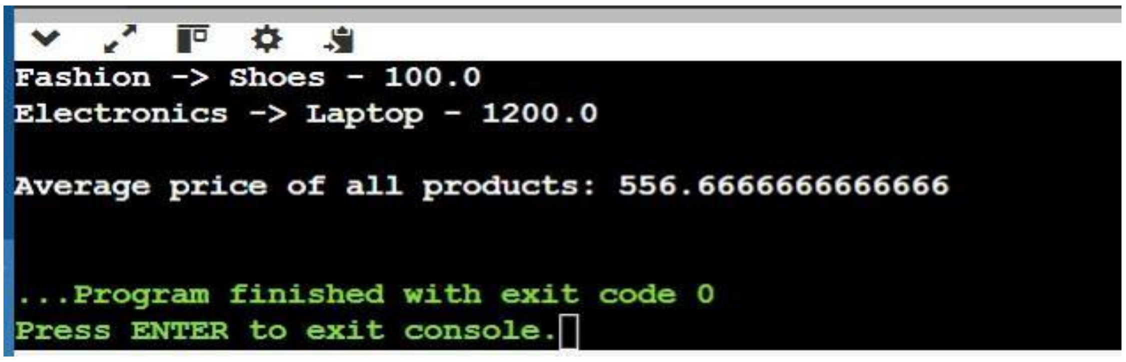
1. **Input Data:**
   Create a list of products with details like name, category, and price.
2. **Group Products by Category:**
   Use streams to group products based on their category.
3. **Find Most Expensive Product in Each Category:**
   For each category, find the product with the highest price using stream operations.
4. **Calculate Average Price of All Products:**
   Use streams to calculate the average price of all products.
5. **Display Results:**
   Show the grouped products, most expensive product in each category, and the average price.
6. **End Program**

Code:

```
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 getCategory() {
      return category;
   }

   public double getPrice() {
      return price;
   }
```

```
    @Override
    public String toString() {
        return name + " - " + price;
    }
}

public class ProductStreamProcessing {
    public static void main(String[] args) {
        List<Product> products = Arrays.asList(
                new Product("Laptop", "Electronics", 1200.0),
                new Product("Phone", "Electronics", 800.0),
                new Product("Shoes", "Fashion", 100.0),
                new Product("T-shirt", "Fashion", 40.0),
                new Product("Fridge", "Appliances", 900.0),
                new Product("Microwave", "Appliances", 300.0)
        );

        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));

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

        double averagePrice = products.stream()
                .collect(Collectors.averagingDouble(Product::getPrice));
        System.out.println("\nAverage price of all products: " + averagePrice);
    }
}
```

Output:



```
Fashion -> Shoes - 100.0
Electronics -> Laptop - 1200.0

Average price of all products: 556.6666666666666


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

Learning Outcomes:

i. Learn how to process lists of items easily using Java Streams.
ii. Understand how to group products by category automatically.
iii. Discover how to find the most expensive product in each group.
iv. Learn how to calculate the average price of all products.