Experiment 6

Student Name: Aditi Sharma **UID**: 22BCS10692

Branch: BE-CSE **Section/Group**: IOT-641/B

Semester: 6th **Date of Performance**: 5/03/2025

Subject Name: Project Based Learning Subject in Java with Lab **Code**: 22CSH-359

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

2. **Objective:**

   a) Understand lambda expressions in Java.
   b) Utilize the `Comparator` interface with lambda expressions.
   c) Implement sorting on custom objects.

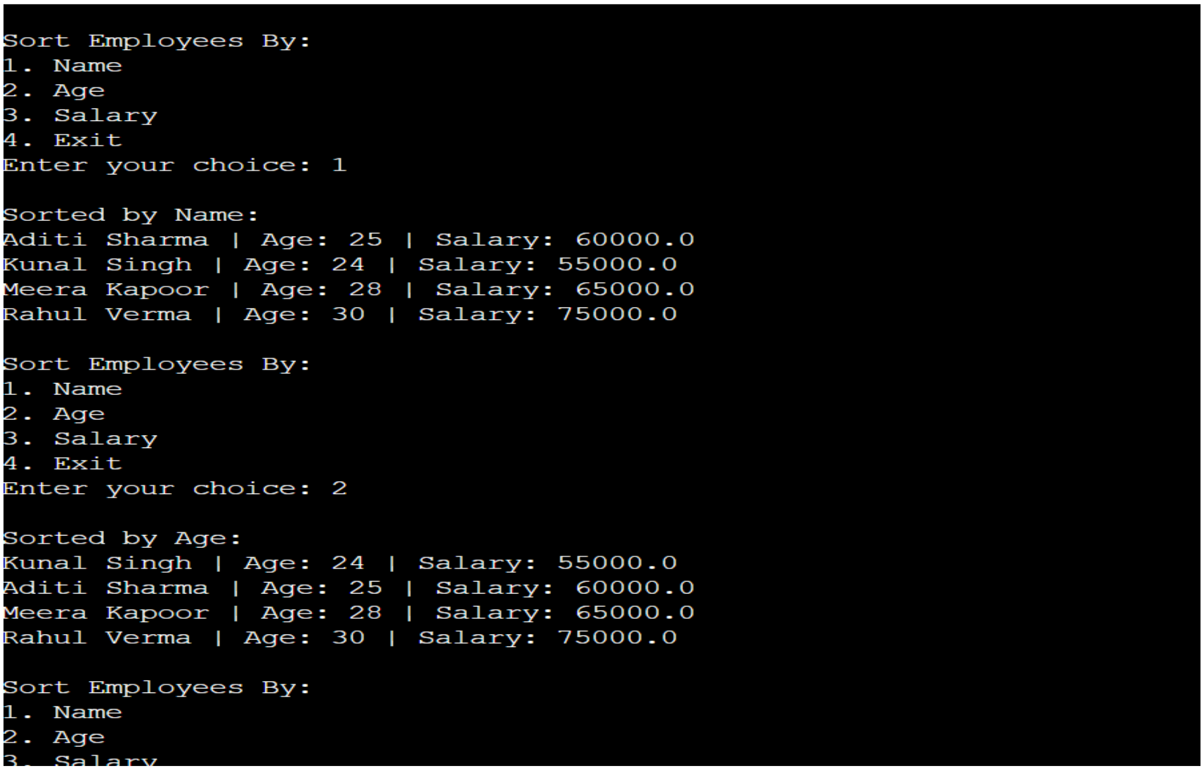
3. **Algorithm**:

   a) Create an Employee class with attributes: name, age, and salary.
   b) Create a list of Employee objects.
   c) Use lambda expressions to sort employees by:
   d) Name (alphabetically)
   e) Age (ascending order)
   f) Salary (descending order)
   g) Display the sorted results.

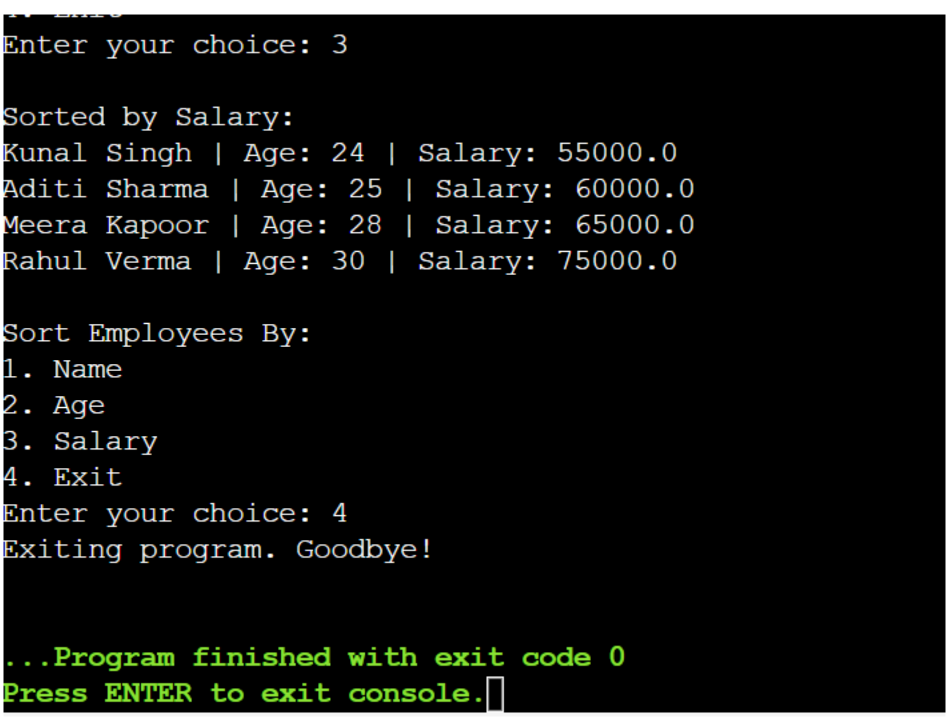
4. **Implementation/Code**:

```
import java.util.*;

// Employee class
class Employee {
    private String name;
    private int age;
    private double salary;

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

    // Getters
    public String getName() { return name; }
    public int getAge() { return age; }
    public double getSalary() { return salary; }

    // Method to display employee details
    public void display() {
        System.out.println(name + " | Age: " + age + " | Salary: " + salary);
    }
}

public class Main {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();

        // Adding employees
        employees.add(new Employee("Aditi Sharma", 25, 60000));
        employees.add(new Employee("Rahul Verma", 30, 75000));
        employees.add(new Employee("Meera Kapoor", 28, 65000));
        employees.add(new Employee("Kunal Singh", 24, 55000));

        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\nSort Employees By:");
            System.out.println("1. Name");
            System.out.println("2. Age");
            System.out.println("3. Salary");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    employees.sort(Comparator.comparing(Employee::getName));
                    System.out.println("\nSorted by Name:");
                    break;
                case 2:
                    employees.sort(Comparator.comparingInt(Employee::getAge));
                    System.out.println("\nSorted by Age:");
                    break;
                case 3:
                    employees.sort(Comparator.comparingDouble(Employee::getSalary));
                    System.out.println("\nSorted by Salary:");
                    break;
                case 4:
                    System.out.println("Exiting program. Goodbye!");
                    return;
                default:
                    System.out.println("Invalid choice! Please enter 1, 2, 3, or 4.");
                    continue;
            }

            // Display sorted employees
            for (Employee emp : employees) {
                emp.display();
            }
        }
    }
}
```

4. Output:



```
Sort Employees By:
1. Name
2. Age
3. Salary
4. Exit
Enter your choice: 1

Sorted by Name:
Aditi Sharma | Age: 25 | Salary: 60000.0
Kunal Singh | Age: 24 | Salary: 55000.0
Meera Kapoor | Age: 28 | Salary: 65000.0
Rahul Verma | Age: 30 | Salary: 75000.0

Sort Employees By:
1. Name
2. Age
3. Salary
4. Exit
Enter your choice: 2

Sorted by Age:
Kunal Singh | Age: 24 | Salary: 55000.0
Aditi Sharma | Age: 25 | Salary: 60000.0
Meera Kapoor | Age: 28 | Salary: 65000.0
Rahul Verma | Age: 30 | Salary: 75000.0

Sort Employees By:
1. Name
2. Age
3. Salary
```



```
4. Exit
Enter your choice: 3

Sorted by Salary:
Kunal Singh | Age: 24 | Salary: 55000.0
Aditi Sharma | Age: 25 | Salary: 60000.0
Meera Kapoor | Age: 28 | Salary: 65000.0
Rahul Verma | Age: 30 | Salary: 75000.0

Sort Employees By:
1. Name
2. Age
3. Salary
4. Exit
Enter your choice: 4
Exiting program. Goodbye!


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

5. Learning Outcomes:

   a) Using lambda expressions for sorting.
   b) Implementing `Comparator.comparing()`.
   c) Understanding sorting mechanisms with custom objects.

MEDIUM:

1. **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.

2. Objective:

a) Use Java streams to filter and sort data efficiently.
b) Learn lambda expressions for concise code.

3. Algorithm:

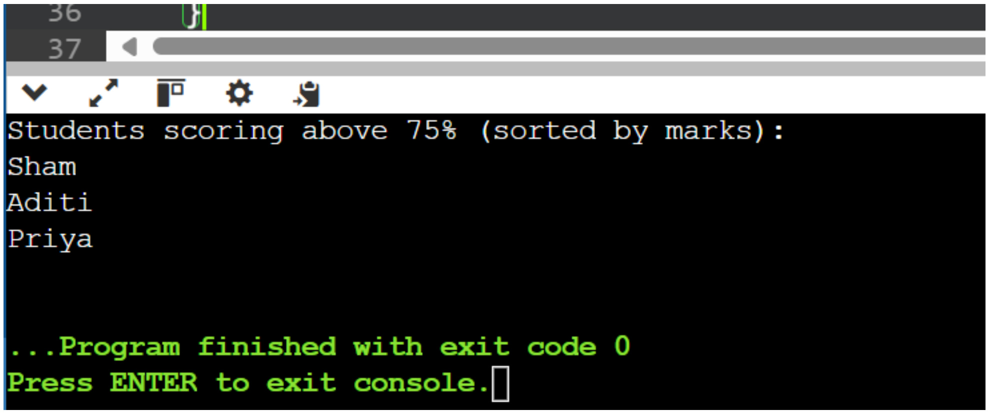
a) Create a `Student` class with attributes `name` and `marks`.
b) Create a list of students.
c) Use streams to:
- Filter students with marks > 75%.
- Sort them in descending order.
- Display their names.

4. **Implementation/Code**:

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

```
// Student class
class Student {
    private String name;
    private 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 Main {
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();

        // Adding students
        students.add(new Student("Aditi ", 85.5));
        students.add(new Student("Ram ", 72.0));
        students.add(new Student("Sham ", 90.3));
        students.add(new Student("Divya ", 65.0));
        students.add(new Student("Priya ", 78.8));

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

Output:



```
Students scoring above 75% (sorted by marks):
Sham
Aditi
Priya


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

Learning Outcome:

a) Understanding Java Streams for filtering and sorting.

b) Using lambda expressions effectively in stream operations.

c) Implementing sorting with sorted() in streams.

HARD:

1. **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.

2. Objective:

a) Implement **grouping**, **filtering**, and **aggregation** operations using Java Streams.

b) Learn how to work with large datasets efficiently.

3. Algorithm:

a) Create a `Product` class with attributes: `name`, `category`, and `price`.
b) Create a list of products with different categories.
c) Use **Streams API** to:

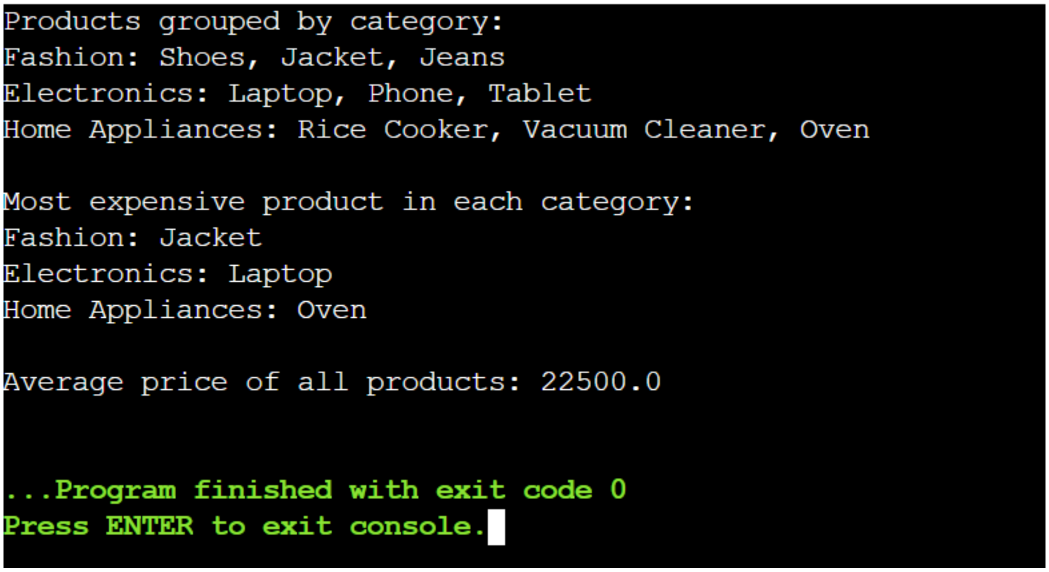
   a. Group products by category.
   b. Find the most expensive product in each category.
   c. Compute the average price of all products.

4. Implementation/Code:

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

// Product class
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; }
}

public class Main {
    public static void main(String[] args) {
        List<Product> products = Arrays.asList(
            new Product("Laptop", "Electronics", 80000),
            new Product("Phone", "Electronics", 50000),
            new Product("Tablet", "Electronics", 30000),
            new Product("Shoes", "Fashion", 4000),
            new Product("Jacket", "Fashion", 8000),
            new Product("Jeans", "Fashion", 2500),
            new Product("Rice Cooker", "Home Appliances", 6000),
            new Product("Vacuum Cleaner", "Home Appliances", 10000),
            new Product("Oven", "Home Appliances", 12000)
        );

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

        System.out.println("Products grouped by category:");
        groupedByCategory.forEach((category, prodList) -> {
            System.out.println(category + ": " + prodList.stream()
                    .map(Product::getName)
                    .collect(Collectors.joining(", ")));
        });

        // Finding the most expensive product in each category
        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.map(Product::getName).orElse("None"))
        );

        // Calculating 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);
    }
```

```
}
```

Output:



```
Products grouped by category:
Fashion: Shoes, Jacket, Jeans
Electronics: Laptop, Phone, Tablet
Home Appliances: Rice Cooker, Vacuum Cleaner, Oven

Most expensive product in each category:
Fashion: Jacket
Electronics: Laptop
Home Appliances: Oven

Average price of all products: 22500.0


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

Learning Outcome:

a) Implementing **grouping** operations using Collectors.groupingBy().

b) Using Collectors.maxBy() to find maximum values within groups.

c) Performing **aggregations** like calculating the average price.