Experiment 6

Student Name: Harshit Mishra **UID: 22BCS11592**

Branch: BE/CSE **Section/Group: IOT_618/B**

Semester: 6th **Date of Performance: 28/02/25**

Subject Name: Project based learning in Java

Subject Code: 22CSH-359

Easy Level

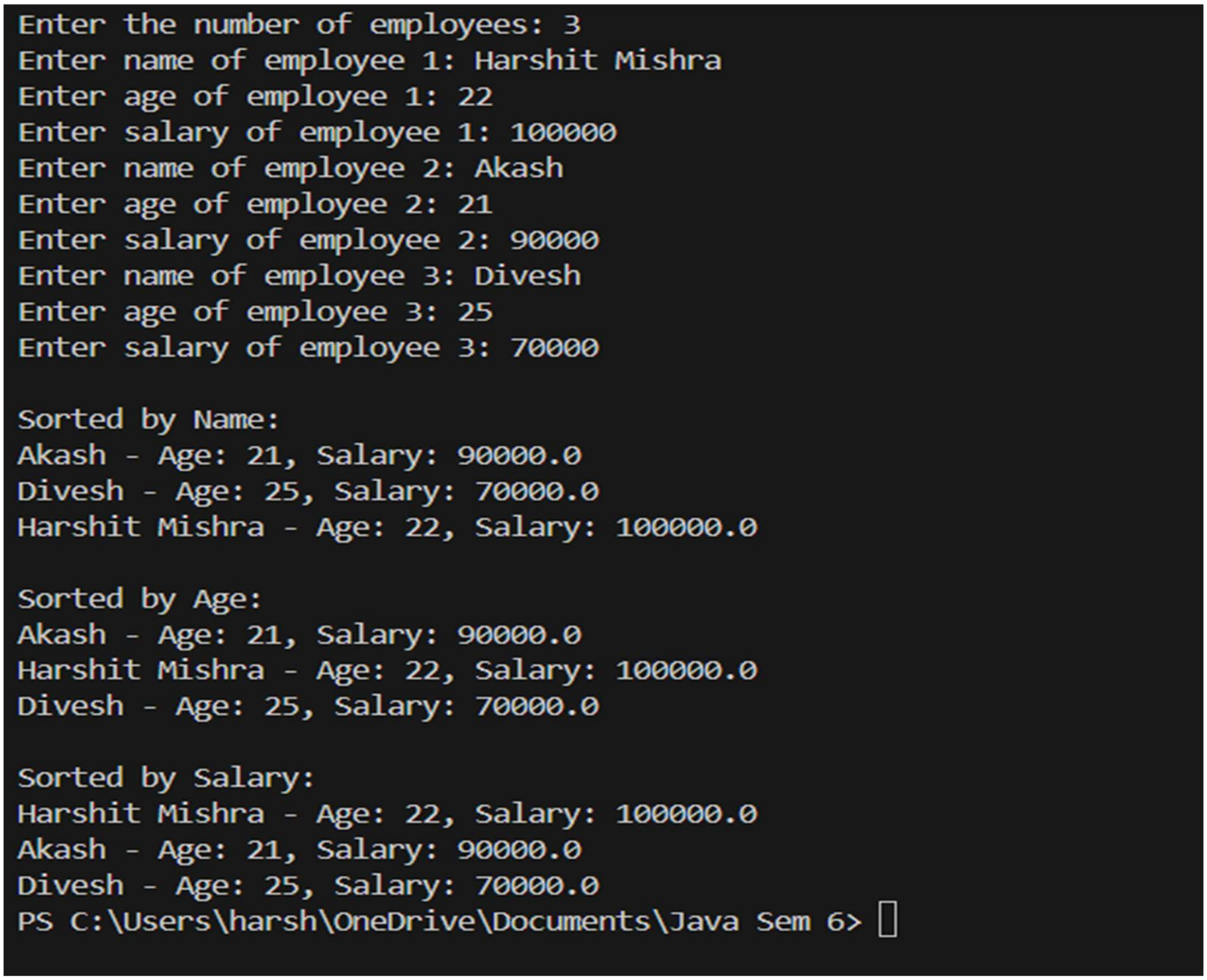
1. **Aim:** Write a program to sort a list of Employee objects (name, age, salary) using lambda expressions.

2. **Objective:** The objective of this Java program is to demonstrate how to sort a list of Employee objects based on different attributes (name, age, salary) using **lambda expressions** and the Comparator interface

3. **Implementation/Code:**

```
import java.util.*;
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;
    }
    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
    public double getSalary() {
        return salary;
    }
    public void display() {
        System.out.println(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();  // Consume newline
        List<Employee> employees = new ArrayList<>();
        for (int i = 0; i < numEmployees; i++) {
            System.out.print("Enter name of employee " + (i + 1) + ": ");
            String name = scanner.nextLine();
            System.out.print("Enter age of employee " + (i + 1) + ": ");
            int age = scanner.nextInt();
            System.out.print("Enter salary of employee " + (i + 1) + ": ");
            double salary = scanner.nextDouble();
            scanner.nextLine();  // Consume newline
            employees.add(new Employee(name, age, salary));
        }
        // Sorting by name
        employees.sort(Comparator.comparing(Employee::getName));
        System.out.println("\nSorted by Name:");
        employees.forEach(Employee::display);
        // Sorting by age
        employees.sort(Comparator.comparingInt(Employee::getAge));
        System.out.println("\nSorted by Age:");
        employees.forEach(Employee::display);
        // Sorting by salary (Descending order)
        employees.sort(Comparator.comparingDouble(Employee::getSalary).reversed())
        System.out.println("\nSorted by Salary:");
        employees.forEach(Employee::display);
        scanner.close();
    }
}
```

4. Output:



```
Enter the number of employees: 3
Enter name of employee 1: Harshit Mishra
Enter age of employee 1: 22
Enter salary of employee 1: 100000
Enter name of employee 2: Akash
Enter age of employee 2: 21
Enter salary of employee 2: 90000
Enter name of employee 3: Divesh
Enter age of employee 3: 25
Enter salary of employee 3: 70000

Sorted by Name:
Akash - Age: 21, Salary: 90000.0
Divesh - Age: 25, Salary: 70000.0
Harshit Mishra - Age: 22, Salary: 100000.0

Sorted by Age:
Akash - Age: 21, Salary: 90000.0
Harshit Mishra - Age: 22, Salary: 100000.0
Divesh - Age: 25, Salary: 70000.0

Sorted by Salary:
Harshit Mishra - Age: 22, Salary: 100000.0
Akash - Age: 21, Salary: 90000.0
Divesh - Age: 25, Salary: 70000.0
PS C:\Users\harsh\OneDrive\Documents\Java Sem 6> 
```

5. Learning Outcomes:

- Learnt about Comparator interface
- Efficient Data Handling
- Learn to handle user input from the command line.
- Looping and Computation.
- Understanding Java Sorting and Filtration.

Medium Level

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.

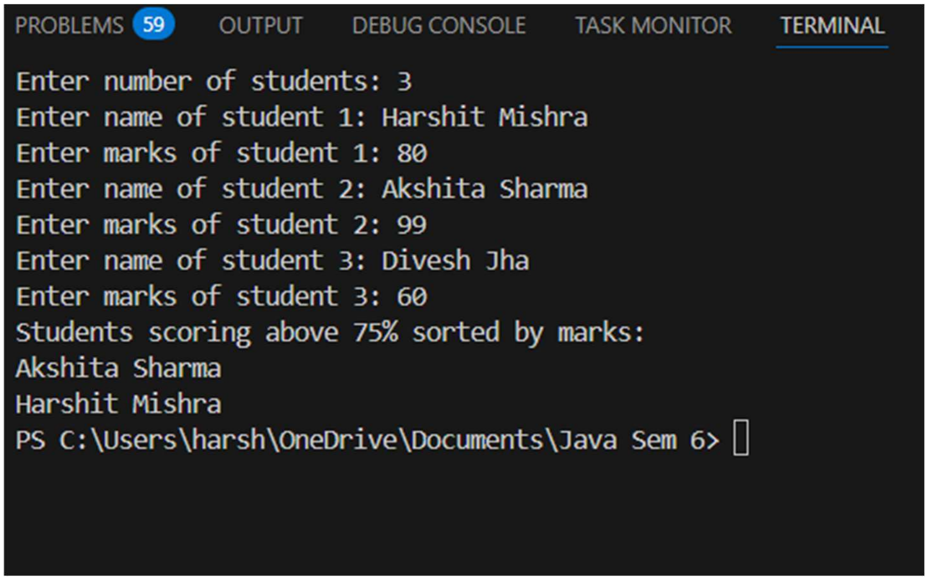
2. **Objective:** The objective of this Java program is to demonstrate the use of **lambda expressions** and **stream operations** to efficiently process and manipulate collections.

3. **Implementation/Code:**

```
import java.util.*;
import java.util.stream.Collectors;
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;
    }
    @Override
    public String toString() {
        return name + " - " + marks;
    }
}
public class StudentFilter {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Student> students = new ArrayList<>();
        System.out.print("Enter number of students: ");
        int n = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        for (int i = 0; i < n; i++) {
            System.out.print("Enter name of student " + (i + 1) + ": ");
            String name = scanner.nextLine();
            System.out.print("Enter marks of student " + (i + 1) + ": ");
            double marks = scanner.nextDouble();
            scanner.nextLine(); // Consume newline
            students.add(new Student(name, marks));
        }
        // Using Streams and Lambda expressions
        List<String> filteredSortedStudents = students.stream()
                .filter(student -> student.getMarks() > 75) // Filter students scoring
above 75%
```

```
                .sorted(Comparator.comparingDouble(Student::getMarks).reversed())
// Sort by marks in descending order
                .map(Student::getName) // Extract student names
                .collect(Collectors.toList());

        // Display the names of filtered and sorted students
        System.out.println("Students scoring above 75% sorted by marks:");
        filteredSortedStudents.forEach(System.out::println);
        scanner.close();
    }
}
```

4. Output:



```
Enter number of students: 3
Enter name of student 1: Harshit Mishra
Enter marks of student 1: 80
Enter name of student 2: Akshita Sharma
Enter marks of student 2: 99
Enter name of student 3: Divesh Jha
Enter marks of student 3: 60
Students scoring above 75% sorted by marks:
Akshita Sharma
Harshit Mishra
PS C:\Users\harsh\OneDrive\Documents\Java Sem 6>
```

5. Learning Outcomes:

- Understanding Java streams for Data Processing.
- Implement key-value storage.
- Add and retrieve elements dynamically without predefined limits.
- Use Scanner to take user input and process it efficiently.

Hard Level

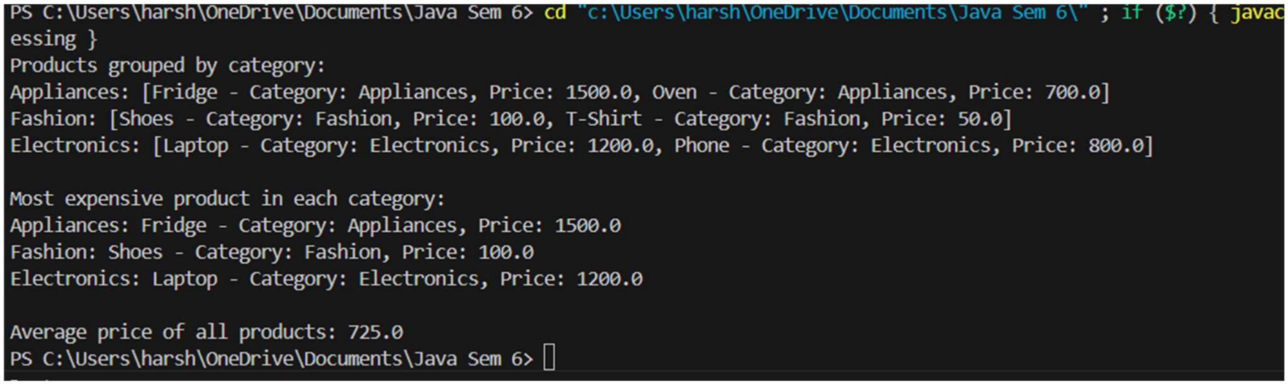
1. **Aim:** Write a Java program to process a large dataset of products using streams. Perform operations such as grouping products by category, finding the most expensive product in each category, and calculating the average price of all products.

2. **Objective:** The objective of this Java program is to demonstrate the use of **Java Streams** to efficiently process a large dataset of products.

3. **Implementation/Code:**

```
import java.util.*;

import java.util.stream.Collectors;

class Product {

    String name;

    String category;

    double price;

    Product(String name, String category, double price) {

        this.name = name;

        this.category = category;

        this.price = price;

    }
    @Override
    public String toString() {
        return name + " - Category: " + category + ", Price: " + price;
    }
}
public class ProductProcessing {
    public static void main(String[] args) {
        List<Product> products = Arrays.asList(
                new Product("Laptop", "Electronics", 1200),
                new Product("Phone", "Electronics", 800),
                new Product("Shoes", "Fashion", 100),
                new Product("T-Shirt", "Fashion", 50),
                new Product("Fridge", "Appliances", 1500),
                new Product("Oven", "Appliances", 700)
        );
        // Grouping products by category
        Map<String, List<Product>> groupedByCategory = products.stream()
                .collect(Collectors.groupingBy(product -> product.category));
        // Finding the most expensive product in each category
            Map<String, Optional<Product>> mostExpensiveByCategory = products.stream()
                .collect(Collectors.groupingBy(product -> product.category,
                        Collectors.maxBy(Comparator.comparingDouble(product -> product.price))));
        // Calculating the average price of all products
        double averagePrice = products.stream()
                .mapToDouble(product -> product.price)
                .average()
                .orElse(0);
        // Display results
        System.out.println("Products grouped by category:");
        groupedByCategory.forEach((category, productList) ->
                System.out.println(category + ": " + productList));
        System.out.println("\nMost expensive product in each category:");
        mostExpensiveByCategory.forEach((category, product) ->
                System.out.println(category + ": " + product.orElse(null)));
        System.out.println("\nAverage price of all products: " + averagePrice);

    }

}
```

4. Output:



```
PS C:\Users\harsh\OneDrive\Documents\Java Sem 6> cd "c:\Users\harsh\OneDrive\Documents\Java Sem 6\" ; if ($?) { javac
essing }
Products grouped by category:
Appliances: [Fridge - Category: Appliances, Price: 1500.0, Oven - Category: Appliances, Price: 700.0]
Fashion: [Shoes - Category: Fashion, Price: 100.0, T-Shirt - Category: Fashion, Price: 50.0]
Electronics: [Laptop - Category: Electronics, Price: 1200.0, Phone - Category: Electronics, Price: 800.0]

Most expensive product in each category:
Appliances: Fridge - Category: Appliances, Price: 1500.0
Fashion: Shoes - Category: Fashion, Price: 100.0
Electronics: Laptop - Category: Electronics, Price: 1200.0

Average price of all products: 725.0
PS C:\Users\harsh\OneDrive\Documents\Java Sem 6> 
```

5. Learning Outcomes:

- Grouping Data.
- Using Java Streams.
- Handling race conditions in a multi-threaded environment.
- Taking user input for dynamic seat selection and priority assignment.
- Efficient Data Analysis.