Experiment 6

Student Name: Dharmesh Kumar UID: 22BCS10126
Branch: BE-CSE Section/Group: 22BCS_IOT-638(B)
Semester: 6th Date of Performance: 24/02/25
Subject Name: PBLJ Subject Code: 22CSH-359

1. Aim: Develop Java programs using lambda expressions and stream operations for sorting, filtering, and processing large datasets efficiently while utilizing wrapper classes, autoboxing and unboxing, byte streams, character streams, object serialization, cloning, functional interfaces, method references, and various stream operations such as sorting, filtering, mapping, reducing, and grouping.

2. Programming Problems:

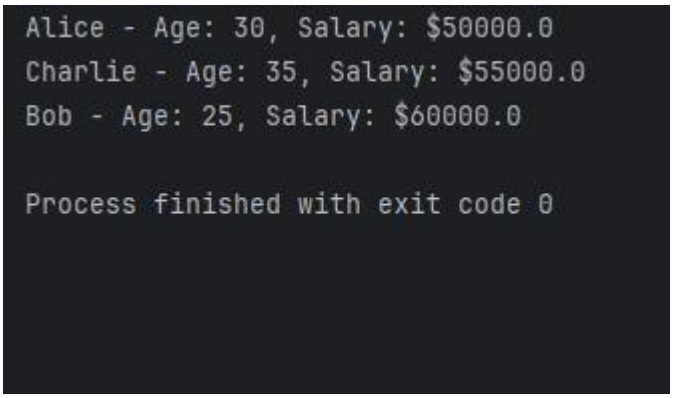
a) Write a program to sort a list of Employee objects (name, age, salary) using lambda expressions.

- Implementation/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("Alice",
            30, 50000), new Employee("Bob", 25,
            60000),
            new Employee("Charlie", 35, 55000)
        );

        employees.sort(Comparator.comparingDouble(e -> e.salary));
        employees.forEach(System.out::println);
    }
}
```

- Output:



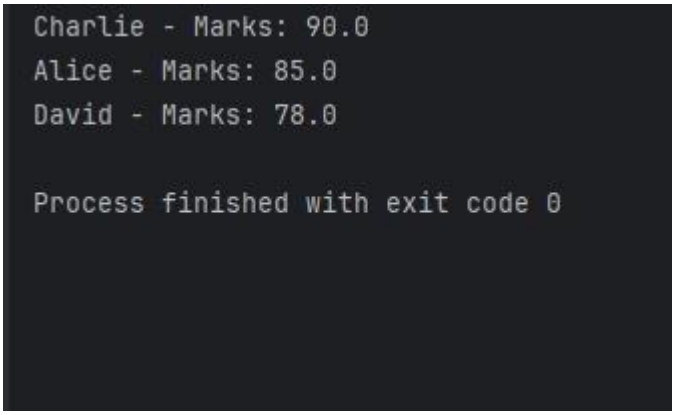
b)Create a program to use lambda expressions and stream operations to filter students scoring above 75%, sort them by marks, and display their names.

- Implementation/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("Alice",
            85), new Student("Bob", 72), new
            Student("Charlie", 90), new
            Student("David", 78)
        );

        List<Student> filteredStudents = students.stream()
            .filter(s -> s.marks > 75)
            .sorted(Comparator.comparingDouble(s -> -s.marks))
            .collect(Collectors.toList());

        filteredStudents.forEach(System.out::println);
    }
}
```

- Output:



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..

- Implementation/Code:

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

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;
   }

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

```
    }
}
public class ProductProcessor {
    public static void main(String[] args) {
        List<Product> products = Arrays.asList( new
            Product("Laptop", "Electronics", 1200),
            new Product("Phone", "Electronics", 800),
            new Product("TV", "Electronics", 1500),
            new Product("Shirt", "Clothing", 50), new
            Product("Jeans", "Clothing", 80), new
            Product("Sofa", "Furniture", 700), new
            Product("Table", "Furniture", 300)
        );

        Map<String, List<Product>> groupedByCategory = products.stream()
            .collect(Collectors.groupingBy(p -> p.category));

        Map<String, Product> mostExpensiveByCategory = products.stream()
            .collect(Collectors.toMap
                ( p -> p.category, p ->
                p,
                (p1, p2) -> p1.price > p2.price ? p1 : p2
            ));

        double averagePrice = products.stream()
            .mapToDouble(p -> p.price)
            .average()
            .orElse(0);

        System.out.println("Grouped Products:");
        groupedByCategory.forEach((category, list) -> System.out.println(category + ": " + list));

        System.out.println("\nMost Expensive Products in Each Category:");
        mostExpensiveByCategory.forEach((category, product) -> System.out.println(category + ": " +
        product));

        System.out.println("\nAverage Price of All Products: $" + averagePrice);
    }
}
```

- Output:

```
Grouped Products:
Clothing: [Shirt - Clothing - $50.0, Jeans - Clothing - $80.0]
Electronics: [Laptop - Electronics - $1200.0, Phone - Electronics - $800.0, TV - Electronics - $1500.0]
Furniture: [Sofa - Furniture - $700.0, Table - Furniture - $300.0]

Most Expensive Products in Each Category:
Clothing: Jeans - Clothing - $80.0
Electronics: TV - Electronics - $1500.0
Furniture: Sofa - Furniture - $700.0

Average Price of All Products: $661.4285714285714

Process finished with exit code 0
```

3. Learning Outcome:

- Understand the use of **wrapper classes** (`Integer`, `Character`, `Long`, `Boolean`) and apply **autoboxing & unboxing** in collections and stream operations.
- Learn **byte & character streams**, implement **object serialization**, and utilize **cloning** for efficient object handling.
- Explore **lambda expressions**, implement **functional interfaces**, and use **method references** to enhance functional programming.
- Master **stream operations** like **sorting, filtering, mapping, reducing, and grouping** for efficient data processing in large datasets.

DEPARTMENTOF

COMPUTER SCIENCE & ENGINEERING