



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 6

Student Name: Rishi Gupta

UID: 22BCS14938

Branch: CSE

Section: 22KPIT-902/B

Semester: 6th

Date of Performance: 28/02/2025

Subject: Project Based Learning in Java

Subject Code: 22CSH-359

1. Aim: Develop Java programs using lambda expressions and stream operations for sorting, filtering, and processing large datasets efficiently.

2. Objective 1: Easy Level

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

3. Code/Implementation:

```
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
Experiment6A {
    public static void main(String[] args) {
        List<Employee> employees = new
        ArrayList<>(Arrays.asList(
            new Employee("Alice", 30,
            60000),
            new Employee("Bob", 25, 50000),
            new Employee("Charlie", 35, 70000)
        ));

        // Sorting by salary using Lambda
        employees.sort((e1,
        e2) -> Double.compare(e1.salary, e2.salary));
        // Display sorted
        employees
        employees.forEach(System.out::println);
    } }
```

**Output:**

```
Bob | Age: 25 | Salary: 50000.0
Alice | Age: 30 | Salary: 60000.0
Charlie | Age: 35 | Salary: 70000.0
```

4. Objective 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.

5. Code/Implementation:

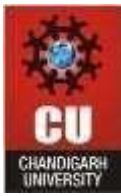
```
import java.util.*; import
java.util.stream.*;

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

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

public class Experiment6B {
    public static void main(String[] args)
    {
        List<Student> students =
        Arrays.asList(
            new
            Student("Alice", 80),
            new
            Student("Bob", 70),
            new
            Student("Charlie", 85),
            Student("David", 60)
        );

        // Filter students scoring above 75%, sort by marks, and
        display names        students.stream()
            .filter(s -> s.marks > 75)
            .sorted((s1, s2) -> Double.compare(s2.marks, s1.marks)) //
            Descending order
            .forEach(System.out::println);
    }
}
```



```
}  
}
```

Output:

```
Charlie | Marks: 85.0  
Alice | Marks: 80.0
```

6. Objective 3 : Hard Level

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.

7. Code/Implementation:

```
import java.util.*; import  
java.util.stream.Collectors;  
  
class Product {  
    String name, 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  
String.format("%-10s | %-12s | $%-8.2f", name, category, price);  
    }  
} public class  
Experiment6C { public  
static void main(String[]  
args)  
{  
    List<Product>  
products =  
Arrays.asList(  
new Product("Laptop",  
"Electronics", 800),  
new Product("Phone",  
"Electronics", 500),  
new Product("Shirt",  
"Clothing", 40),  
new Product("Jeans",  
"Clothing", 60),
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

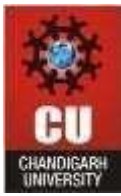
```
new Product("TV",
"Electronics", 1200)
);

// Grouping products by category
Map<String, List<Product>> groupedByCategory = products.stream()
    .collect(Collectors.groupingBy(p -> p.category));
// Finding the most expensive product in each category
Map<String, Product> mostExpensiveByCategory = products.stream()
    .collect(Collectors.groupingBy(
p -> p.category,
    Collectors.collectingAndThen(
        Collectors.maxBy(Comparator.comparingDouble(p ->
p.price)),
        Optional::get
    )
));

// Calculating the average price of all products
double avgPrice = products.stream()
    .mapToDouble(p -> p.price)
    .average()
    .orElse(0);

// Display results with symmetric formatting
System.out.println("\nProducts grouped by category:");
System.out.println("-----");
---");
System.out.printf("%-10s | %-12s | %-10s\n", "Name", "Category",
"Price ($)");
System.out.println("-----");
---");
groupedByCategory.forEach((category, productList) ->
{
    productList.forEach(System.out::println);
});

System.out.println("\nMost expensive product in each category:");
System.out.println("-----");
--");
System.out.printf("%-12s | %-10s | %-10s\n", "Category", "Name",
"Price ($)");
System.out.println("-----");
-----");
---");
mostExpensiveByCategory.forEach((category, product) ->
{
    System.out.printf("%-12s | %-10s | $%-8.2f\n", category,
product.name, product.price);
});
```



```
        System.out.println("\nAverage price of all products: $" +  
String.format("%.2f", avgPrice));  
    } }
```

Output:

```
Products grouped by category:  
-----  
Name      | Category    | Price ($)  
-----  
Shirt     | Clothing    | $40.00  
Jeans     | Clothing    | $60.00  
Laptop    | Electronics  | $800.00  
Phone     | Electronics  | $500.00  
TV        | Electronics  | $1200.00  
  
Most expensive product in each category:  
-----  
Category  | Name        | Price ($)  
-----  
Clothing  | Jeans       | $60.00  
Electronics | TV          | $1200.00  
  
Average price of all products: $520.00
```

8. Learning Outcomes:

- Understand and apply lambda expressions for sorting and filtering data efficiently.
- Utilize Java Streams to process and manipulate large datasets with ease.
- Implement grouping, aggregation, and transformation operations on collections.
- Analyze and extract meaningful insights using functional programming in Java.