1. Lambda Expressions – Case Study: Sorting and Filtering Employees

Scenario: You are building a human resource management module. You need to:

• Sort employees by name or salary.

• Filter employees with a salary above a certain threshold.

Use Case: Instead of creating multiple comparator classes or anonymous classes, you use Lambda expressions to sort and filter employee records in a concise and readable manner.

```
package Day5_Java8_CaseStudy;
import java.util.Arrays;
import java.util.List;
class Employee{
	String name;
	double salary;

	public Employee(String name, double salary) {
		this.name = name;
		this.salary = salary;
	}
};
public class LambdaExpressions {
	public static void main(String[] args) {
		List<Employee> empList = Arrays.asList(
					new Employee("Ravi", 30000),
					new Employee("Ajay", 20000),
					new Employee("Murali", 40000)
					);

		//sort using employee name
		System.out.println("Sort using employee name:");
		empList.sort((e1,e2) -> e1.name.compareTo(e2.name));
		empList.forEach(e -> System.out.println(e.name +" - "+e.salary));

		//sort using employee salary
		System.out.println("\nSort using employee salary:");
		empList.sort((e1,e2)->Double.compare(e1.salary, e2.salary));
		empList.forEach(e -> System.out.println(e.name + " - " + e.salary));

		//filter employees by salary
		System.out.println("\nEmplyees List having salary greater than 25000:");
		empList.forEach(e -> {
			if(e.salary>25000) {
				System.out.println(e.name+" - "+e.salary);
			}
		});
	}
}
```

2. Stream API & Operators – Case Study: Order Processing System
Scenario: In an e-commerce application, you must: • Filter orders above a certain value.
• Count total orders per customer.
• Sort and group orders by product category.
Use Case: Streams help to process collections like orders using operators like filter, map, collect, sorted, and groupingBy to build readable pipelines for data processing.

```
package Day5_Java8_CaseStudy;
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;
class Order{
	double orderValue;
	String customerName;
	int totalOrders;
	String productCategory;

	public Order(double orderValue, String customerName, int totalOrders, String productCategory)
{
		this.orderValue = orderValue;
		this.customerName = customerName;
		this.totalOrders = totalOrders;
		this.productCategory = productCategory;
	}
	public double getOrderValue() {
		return orderValue;
	}
	public String getCustomerName() {
		return customerName;
	}
	public int getTotalOrders() {
		return totalOrders;
	}
	public String getProductCategory() {
		return productCategory;
	}
};
public class StreamAPIandOperators {
	public static void main(String[] args) {
		List<Order> orders = Arrays.asList(
				new Order(1000, "Ravi", 2, "Clothing"),
				new Order(2000, "Bhanu", 2, "Clothing"),
				new Order(200, "Suresh", 1, "Food"),
				new Order(10000, "Ravi", 1, "Electronics")
				);
```

```
                List<Order> highOrderValues = orders.stream()
                                .filter(o -> o.getOrderValue()>1000)
                                .collect(Collectors.toList());

                System.out.println("Orders above value 1000: \nCustomerName ProductCategory
\tCount\tValue");
                highOrderValues.forEach(o ->
System.out.println(o.getCustomerName()+"\t\t"+o.getProductCategory()+"\t"+o.getTotalOrders()+"\t"+o.ge
tOrderValue()));

                Map<String, Integer> totalOrdersPerCustomer = orders.stream()
            .collect(Collectors.groupingBy(
                  Order::getCustomerName,
                  Collectors.summingInt(Order::getTotalOrders)
            ));
      System.out.println("Total orders per customer:");
      totalOrdersPerCustomer.forEach((customer, total) ->
         System.out.println(customer + ": " + total)
      );
      Map<String, List<Order>> ordersByCategory = orders.stream()
            .sorted(Comparator.comparing(Order::getProductCategory))
            .collect(Collectors.groupingBy(Order::getProductCategory));
      System.out.println("Orders grouped by category:");
      ordersByCategory.forEach((category, orderList) -> {
         System.out.println(category + ":");
         orderList.forEach(o ->
            System.out.println("  " + o.getCustomerName() + " - " + o.getOrderValue())
         );
      });

          }
}
```