**Java8 - Case Study**

**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.

**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.

**3. Functional Interfaces – Case Study: Custom Logger**

**Scenario:**

You want to create a logging utility that allows:

• Logging messages conditionally.

• Reusing common log filtering logic.

**Use Case:**

You define a custom LogFilter functional interface and allow users to pass behavior using

lambdas. You also utilize built-in interfaces like Predicate and Consumer.

**4. Default Methods in Interfaces – Case Study: Payment Gateway Integration**

**Scenario:**

You're integrating multiple payment methods (PayPal, UPI, Cards) using interfaces.

**Use Case:**

You use default methods in interfaces to provide shared logic (like transaction logging or currency

conversion) without forcing each implementation to re-define them.

**5. Method References – Case Study: Notification System**

**Scenario:**

You’re sending different types of notifications (Email, SMS, Push). The methods for sending are

already defined in separate classes.

**Use Case:**

You use method references (e.g., NotificationService::sendEmail) to refer to

existing static or instance methods, making your event dispatcher concise and readable.

**6. Optional Class – Case Study: User Profile Management**

**Scenario:**

User details like email or phone number may be optional during registration.

**Use Case:**

To avoid NullPointerException, you wrap potentially null fields in Optional. This

forces developers to handle absence explicitly using methods like orElse, ifPresent, or

map.

**7. Date and Time API (java.time) – Case Study: Booking System**

**Scenario:**

A hotel or travel booking system that:

• Calculates stay duration.

• Validates check-in/check-out dates.

• Schedules recurring events.

**Use Case:**

You use the new LocalDate, LocalDateTime, Period, and Duration classes to

perform safe and readable date/time calculations.

**8. Executor Service – Case Study: File Upload Service**

**Scenario:**

You allow users to upload multiple files simultaneously and want to manage the processing

efficiently.

**Use Case:**

You use ExecutorService to handle concurrent uploads by creating a thread pool, managing

background tasks without blocking the UI or main thread.

**1. Lambda Expressions – Sorting and Filtering Employees**

import java.util.\*;

class Employee {

String name;

double salary;

Employee(String name, double salary) {

this.name = name;

this.salary = salary;

}

public String toString() {

return name + " - " + salary;

}

}

public class EmployeeManager {

public static void main(String[] args) {

List<Employee> employees = Arrays.asList(

new Employee("Alice", 75000),

new Employee("Bob", 50000),

new Employee("Charlie", 80000)

);

employees.sort((e1, e2) -> e1.name.compareTo(e2.name));

System.out.println("Sorted by name:");

employees.forEach(System.out::println);

employees.sort(Comparator.comparingDouble(e -> e.salary));

System.out.println("\nSorted by salary:");

employees.forEach(System.out::println);

System.out.println("\nFiltered (salary > 60000):");

employees.stream()

.filter(e -> e.salary > 60000)

.forEach(System.out::println);

}

}

**2. Stream API & Operators – Order Processing System**

import java.util.\*;

import java.util.stream.\*;

import java.util.function.\*;

import java.util.Map.Entry;

class Order {

String customer;

String category;

double amount;

Order(String customer, String category, double amount) {

this.customer = customer;

this.category = category;

this.amount = amount;

}

public String toString() {

return customer + " - " + category + " - $" + amount;

}

}

public class OrderProcessor {

public static void main(String[] args) {

List<Order> orders = Arrays.asList(

new Order("Alice", "Electronics", 1200),

new Order("Bob", "Books", 150),

new Order("Alice", "Books", 90),

new Order("Charlie", "Clothing", 200),

new Order("Bob", "Electronics", 1300)

);

System.out.println("Orders above $100:");

orders.stream()

.filter(o -> o.amount > 100)

.forEach(System.out::println);

System.out.println("\nTotal orders per customer:");

Map<String, Long> countMap = orders.stream()

.collect(Collectors.groupingBy(o -> o.customer, Collectors.counting()));

countMap.forEach((k, v) -> System.out.println(k + ": " + v));

System.out.println("\nGrouped by category:");

Map<String, List<Order>> grouped = orders.stream()

.sorted(Comparator.comparing(o -> o.category))

.collect(Collectors.groupingBy(o -> o.category));

grouped.forEach((k, v) -> {

System.out.println(k + ": " + v);

});

}

}

**3. Functional Interfaces – Custom Logger**

import java.util.function.Predicate;

import java.util.function.Consumer;

@FunctionalInterface

interface LogFilter {

boolean shouldLog(String message);

}

class Logger {

private LogFilter filter;

Logger(LogFilter filter) {

this.filter = filter;

}

void log(String message) {

if (filter.shouldLog(message)) {

System.out.println("LOG: " + message);

}

}

}

public class CustomLogger {

public static void main(String[] args) {

Logger logger = new Logger(msg -> msg.contains("ERROR"));

logger.log("INFO: Everything is working.");

logger.log("ERROR: Something failed.");

Predicate<String> warnFilter = msg -> msg.contains("WARN");

Consumer<String> warnLogger = msg -> {

if (warnFilter.test(msg)) {

System.out.println("WARN: " + msg);

}

};

warnLogger.accept("WARN: Low disk space.");

warnLogger.accept("INFO: Startup complete.");

}

}

**4. Default Methods in Interfaces – Payment Gateway Integration**

interface Payment {

void pay(double amount);

default void logTransaction(double amount) {

System.out.println("Logging transaction of $" + amount);

}

default double convertToINR(double amountUSD) {

return amountUSD \* 83.0;

}

}

class PayPal implements Payment {

public void pay(double amount) {

logTransaction(amount);

System.out.println("Paid $" + amount + " via PayPal.");

}

}

class UPI implements Payment {

public void pay(double amount) {

logTransaction(amount);

System.out.println("Paid ₹" + convertToINR(amount) + " via UPI.");

}

}

public class PaymentGateway {

public static void main(String[] args) {

Payment paypal = new PayPal();

paypal.pay(100);

Payment upi = new UPI();

upi.pay(100);

}

}

**5. Method References – Notification System**

class NotificationService {

public static void sendEmail(String msg) {

System.out.println("Sending Email: " + msg);

}

public static void sendSMS(String msg) {

System.out.println("Sending SMS: " + msg);

}

public static void sendPush(String msg) {

System.out.println("Sending Push: " + msg);

}

}

public class NotificationSystem {

public static void main(String[] args) {

List<String> messages = Arrays.asList("Welcome!", "Your order shipped.", "Password reset");

messages.forEach(NotificationService::sendEmail);

messages.forEach(NotificationService::sendSMS);

}

}

**6. Optional Class – User Profile Management**

import java.util.Optional;

class User {

private Optional<String> email;

private Optional<String> phone;

User(String email, String phone) {

this.email = Optional.ofNullable(email);

this.phone = Optional.ofNullable(phone);

}

void printContactInfo() {

email.ifPresent(e -> System.out.println("Email: " + e));

System.out.println("Phone: " + phone.orElse("Not Provided"));

}

}

public class UserProfileManager {

public static void main(String[] args) {

User user1 = new User("alice@example.com", null);

User user2 = new User(null, "1234567890");

user1.printContactInfo();

System.out.println();

user2.printContactInfo();

}

}

**7. Date and Time API – Booking System**

import java.time.\*;

public class BookingSystem {

public static void main(String[] args) {

LocalDate checkIn = LocalDate.of(2025, 8, 1);

LocalDate checkOut = LocalDate.of(2025, 8, 7);

Period stay = Period.between(checkIn, checkOut);

System.out.println("Stay duration: " + stay.getDays() + " days");

if (checkOut.isAfter(checkIn)) {

System.out.println("Check-out is valid.");

} else {

System.out.println("Invalid check-out date.");

}

LocalDate start = LocalDate.now();

for (int i = 0; i < 4; i++) {

System.out.println("Cleaning scheduled on: " + start.plusWeeks(i));

}

}

}

**8. ExecutorService – File Upload Service**

import java.util.concurrent.\*;

class FileUploadTask implements Runnable {

private String fileName;

FileUploadTask(String fileName) {

this.fileName = fileName;

}

public void run() {

System.out.println("Uploading: " + fileName + " - " + Thread.currentThread().getName());

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Completed: " + fileName);

}

}

public class FileUploader {

public static void main(String[] args) {

ExecutorService executor = Executors.newFixedThreadPool(3);

for (int i = 1; i <= 5; i++) {

executor.execute(new FileUploadTask("File" + i + ".txt"));

}

executor.shutdown();

}}