JAVA8-CASE STUDY

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

```
package javacasestudy;
import java.util.*;
class Employee {
    private String name;
    private double salary;
    public Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    public String getName() { return name; }
    public double getSalary() { return salary; }

    @Override
    public String toString() {
        return "Employee{name='" + name + "', salary=" + salary + "}";
    }
}

public class EmployeeLambdaDemo {

    public static void main(String[] args) {
        List<Employee> employees = Arrays.asList(
            new Employee("Alice", 75000),
            new Employee("Bob", 55000),
            new Employee("Charlie", 90000),
            new Employee("David", 60000)
        );
```

```
        // Sort by name
        System.out.println(" Sorted by Name:");
        employees.stream()
                .sorted(Comparator.comparing(Employee::getName))
                .forEach(System.out::println);

        // Sort by salary
        System.out.println("\n Sorted by Salary:");
        employees.stream()
                .sorted(Comparator.comparingDouble(Employee::getSalary))
                .forEach(System.out::println);

        // Filter salary > 60000
        System.out.println("\n Employees with Salary > 60000:");
        employees.stream()
                .filter(e -> e.getSalary() > 60000)
                .forEach(System.out::println);
    }
}
```

2.Stream API and Operators – Case Study :order Processing System

```
package javacasestudy;

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

class Order {
    private int orderId;
    private String customerName;
    private String category;
    private double amount;
```

```
    public Order(int orderId, String customerName, String category, double amount) {
        this.orderId = orderId;
        this.customerName = customerName;
        this.category = category;
        this.amount = amount;
    }

    public int getOrderId() { return orderId; }
    public String getCustomerName() { return customerName; }
    public String getCategory() { return category; }
    public double getAmount() { return amount; }

    @Override
    public String toString() {
        return "Order{" +
                "ID=" + orderId +
                ", customer='" + customerName + '\'' +
                ", category='" + category + '\'' +
                ", amount=" + amount +
                '}';
    }
}

public class StreamAPIOperation {

    public static void main(String[] args) {
        List<Order> orders = Arrays.asList(
            new Order(101, "Alice", "Electronics", 2500),
            new Order(102, "Bob", "Clothing", 800),
            new Order(103, "Alice", "Books", 400),
```

```
            new Order(104, "David", "Electronics", 1200),
            new Order(105, "Bob", "Books", 200),
            new Order(106, "Charlie", "Clothing", 1500),
            new Order(107, "Alice", "Clothing", 900)
        );

        // 1. Filter orders above ₹1000
        System.out.println(" Orders Above ₹1000:");
        orders.stream()
              .filter(o -> o.getAmount() > 1000)
              .forEach(System.out::println);

        // 2. Count total orders per customer
        System.out.println("\n Total Orders per Customer:");
        Map<String, Long> orderCount = orders.stream()
              .collect(groupingBy(Order::getCustomerName, counting()));
        orderCount.forEach((customer, count) ->
              System.out.println(customer + ": " + count + " orders"));

        // 3. Sort and group orders by product category
        System.out.println("\n Grouped & Sorted Orders by Category:");
        Map<String, List<Order>> groupedByCategory = orders.stream()
              .sorted(Comparator.comparing(Order::getCategory))
              .collect(groupingBy(Order::getCategory));

        groupedByCategory.forEach((category, orderList) -> {
            System.out.println("\nCategory: " + category);
            orderList.forEach(System.out::println);
        });
    }
}
```

3.Functional Interface – Case Study :Custom Logger

```
package javacasestudy;


import java.util.function.Predicate;
import java.util.function.Consumer;



interface LogFilter {
    boolean shouldLog(String level, String message);
}

// 2. Logger class using functional interfaces
class Logger {
    private LogFilter filter;

    public Logger(LogFilter filter) {
        this.filter = filter;
    }

    public void log(String level, String message) {
        if (filter.shouldLog(level, message)) {
            System.out.println("[" + level.toUpperCase() + "] " + message);
        }
    }

    // Alternate method using built-in Predicate
    public void logWithPredicate(Predicate<String> predicate, Consumer<String> consumer, String
message) {
        if (predicate.test(message)) {
            consumer.accept(message);
        }
```

```
    }
}

// 3. Main class with main method
public class FunctionalInterfaceEx {

    public static void main(String[] args) {

        // Lambda: Only log ERROR messages
        Logger errorLogger = new Logger((level, msg) -> level.equalsIgnoreCase("ERROR"));
        errorLogger.log("INFO", "This is an info message");
        errorLogger.log("ERROR", "This is an error message");

        // Lambda: Log messages containing the word "urgent"
        Logger urgentLogger = new Logger((level, msg) -> msg.toLowerCase().contains("urgent"));
        urgentLogger.log("WARN", "This is a normal warning");
        urgentLogger.log("INFO", "This is an urgent update");

        // Using built-in Predicate and Consumer
        Logger predicateLogger = new Logger((level, msg) -> true); // dummy filter

        Predicate<String> containsError = msg -> msg.contains("error");
        Consumer<String> printConsumer = msg -> System.out.println("[PREDICATE LOG] " + msg);

        predicateLogger.logWithPredicate(containsError, printConsumer, "System running fine");
        predicateLogger.logWithPredicate(containsError, printConsumer, "Disk error occurred");
    }
}
```

4.Default Method in Interface -Case Study : Payment Gateway integration

```
package javacasestudy;
```

```
//1. PaymentGateway interface with a default method
interface PaymentGateway {
 void processPayment(double amount);

 default void logTransaction(String method, double amount) {
    System.out.println("Transaction logged: ₹" + amount + " via " + method);
 }
}

//2. PayPal implementation
class PayPalPayment implements PaymentGateway {
 public void processPayment(double amount) {
    System.out.println("Processing PayPal payment of ₹" + amount);
    logTransaction("PayPal", amount); // using default method
 }
}

//3. UPI implementation
class UPIPayment implements PaymentGateway {
 public void processPayment(double amount) {
    System.out.println("Processing UPI payment of ₹" + amount);
    logTransaction("UPI", amount); // using default method
 }
}

//4. Card implementation
class CardPayment implements PaymentGateway {
 public void processPayment(double amount) {
```

```
    System.out.println("Processing Card payment of ₹" + amount);
    logTransaction("Card", amount); // using default method
 }
}

//5. Main class
public class PaymentGatewayDemo {
 public static void main(String[] args) {
    PaymentGateway paypal = new PayPalPayment();
    PaymentGateway upi = new UPIPayment();
    PaymentGateway card = new CardPayment();

    paypal.processPayment(1500);
    upi.processPayment(750);
    card.processPayment(3000);
 }
}
```

5.Method Referance – Case Study : Notification System

```
package javacasestudy;


import java.util.function.Consumer;

// 1. Notification service with static methods
class NotificationService {
   public static void sendEmail(String message) {
      System.out.println("Email sent: " + message);
   }

   public static void sendSMS(String message) {
```

```
        System.out.println(" SMS sent: " + message);
    }

    public static void sendPushNotification(String message) {
        System.out.println(" Push Notification sent: " + message);
    }
}

// 2. Notification dispatcher
class NotificationDispatcher {
    public void dispatch(String message, Consumer<String> notificationMethod) {
        notificationMethod.accept(message);
    }
}

// 3. Main class to use method references
public class MethodReference {
    public static void main(String[] args) {
        NotificationDispatcher dispatcher = new NotificationDispatcher();

        // Using method references for existing methods
        dispatcher.dispatch("Your order has been shipped!", NotificationService::sendEmail);
        dispatcher.dispatch("Your OTP is 123456", NotificationService::sendSMS);
        dispatcher.dispatch("New offer just for you!", NotificationService::sendPushNotification);
    }
}
```

6.Optional Class – Case Study : User Profile Management

```
package javacasestudy;

import java.util.Optional;
```

```
// 1. User class with optional fields
class User {
    private String name;
    private Optional<String> email;
    private Optional<String> phone;

    public User(String name, String email, String phone) {
        this.name = name;
        this.email = Optional.ofNullable(email);
        this.phone = Optional.ofNullable(phone);
    }

    public String getName() {
        return name;
    }

    public Optional<String> getEmail() {
        return email;
    }

    public Optional<String> getPhone() {
        return phone;
    }
}

// 2. Main class demonstrating Optional usage
public class OptionalUserProfileDemo {
    public static void main(String[] args) {
        User user1 = new User("Alice", "alice@example.com", null);
        User user2 = new User("Bob", null, "9876543210");
```

```
        printUserProfile(user1);
        System.out.println("----");
        printUserProfile(user2);
    }

    static void printUserProfile(User user) {
        System.out.println("Name: " + user.getName());

        // Handle optional email
        user.getEmail().ifPresentOrElse(
            email -> System.out.println(" Email: " + email),
            () -> System.out.println(" Email: Not provided")
        );

        // Handle optional phone
        String phone = user.getPhone().orElse("Phone not provided");
        System.out.println(" Phone: " + phone);
    }
}
```

7.Date and Time API(Java.time)-Case Study: Booking System

```
package javacasestudy;

import java.time.*;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;

public class BookSystemDemo {

    public static void main(String[] args) {
        // Booking dates
```

```
LocalDate checkIn = LocalDate.of(2025, 8, 1);
LocalDate checkOut = LocalDate.of(2025, 8, 5);

System.out.println(" Booking Dates:");
System.out.println("Check-in: " + checkIn);
System.out.println("Check-out: " + checkOut);

// Validate dates
if (checkOut.isAfter(checkIn)) {
   long days = ChronoUnit.DAYS.between(checkIn, checkOut);
   System.out.println("Valid booking. Stay Duration: " + days + " nights");
} else {
   System.out.println(" Error: Check-out must be after check-in");
}

// Schedule a recurring weekly housekeeping every Monday
System.out.println("\n Weekly Housekeeping Schedule for August 2025:");
LocalDate housekeepingDate = LocalDate.of(2025, 8, 1);
while (housekeepingDate.getMonth() == Month.AUGUST) {
   if (housekeepingDate.getDayOfWeek() == DayOfWeek.MONDAY) {
      System.out.println("Housekeeping on: " + housekeepingDate);
   }
   housekeepingDate = housekeepingDate.plusDays(1);
}

// Use LocalDateTime and Duration
LocalDateTime startTime = LocalDateTime.of(2025, 8, 1, 14, 0);
LocalDateTime endTime = LocalDateTime.of(2025, 8, 1, 18, 30);
Duration duration = Duration.between(startTime, endTime);

System.out.println("\n Time Between Events:");
```

```
        System.out.println("Start: " + startTime.format(DateTimeFormatter.ofPattern("dd-MM-yyyy
HH:mm")));
        System.out.println("End: " + endTime.format(DateTimeFormatter.ofPattern("dd-MM-yyyy
HH:mm")));
        System.out.println("Duration: " + duration.toHours() + " hours and " +
                duration.toMinutesPart() + " minutes");
    }
}
```

8.Executor Service – Case Study : File Upload Service

```
package javacasestudy;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

// Simulated FileUploadTask
class FileUploadTask implements Runnable {
    private final String fileName;

    public FileUploadTask(String fileName) {
        this.fileName = fileName;
    }

    @Override
    public void run() {
        System.out.println("Uploading: " + fileName + " [Thread: " + Thread.currentThread().getName()
+ "]");
        try {
            // Simulate upload time
            Thread.sleep(2000);
```

```
        } catch (InterruptedException e) {
            System.out.println("Upload interrupted: " + fileName);
        }
        System.out.println("Completed: " + fileName);
    }
}

public class FileUploadServiceDemo {

    public static void main(String[] args) {

        // 1. Create a fixed thread pool (e.g., 3 concurrent uploads)
        ExecutorService executor = Executors.newFixedThreadPool(3);

        // 2. Simulate uploading 6 files
        String[] files = {"img1.png", "img2.jpg", "doc1.pdf", "music.mp3", "video.mp4", "report.docx"};

        for (String file : files) {
            FileUploadTask task = new FileUploadTask(file);
            executor.submit(task);  // Submit task for execution
        }

        // 3. Shutdown executor after all tasks are submitted
        executor.shutdown();

        try {
            // Wait for all tasks to finish
            executor.awaitTermination(1, TimeUnit.MINUTES);
        } catch (InterruptedException e) {
            System.out.println("Upload interrupted");
        }
```

```
        System.out.println("\n All files uploaded.");
    }
}
```

JDBC CASE STUDY

Case Study 1: Online Course Registration System

SQL

```
CREATE DATABASE course_db;

USE course_db;


CREATE TABLE courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(100),
    faculty VARCHAR(100),
    credits INT
);
```

JDBC Operations

```
package jdbc.demo;



import java.sql.*;
import java.util.Scanner;


public class CourseRegistrationSystem {


    static final String DB_URL = "jdbc:mysql://localhost:3306/course_db";
    static final String USER = "root";
    static final String PASS = "Hema@1802"; // Change to your actual password
```

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    try (
        Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
        Statement stmt = conn.createStatement();
    ) {
        Class.forName("com.mysql.cj.jdbc.Driver");

        int choice;
        do {
            System.out.println("\n--- Course Registration System ---");
            System.out.println("1. Insert Course");
            System.out.println("2. View All Courses");
            System.out.println("3. Update Course");
            System.out.println("4. Delete Course");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");
            choice = sc.nextInt();
            sc.nextLine(); // clear buffer

            switch (choice) {
                case 1:
                    System.out.print("Enter Course ID: ");
                    int id = sc.nextInt();
                    sc.nextLine();
                    System.out.print("Enter Course Name: ");
                    String name = sc.nextLine();
                    System.out.print("Enter Faculty Name: ");
                    String faculty = sc.nextLine();
                    System.out.print("Enter Credits: ");
```

```
    int credits = sc.nextInt();

    String insertQuery = "INSERT INTO courses VALUES (?, ?, ?, ?)";
    try (PreparedStatement pstmt = conn.prepareStatement(insertQuery)) {
        pstmt.setInt(1, id);
        pstmt.setString(2, name);
        pstmt.setString(3, faculty);
        pstmt.setInt(4, credits);
        pstmt.executeUpdate();
        System.out.println(" Course inserted successfully.");
    }
    break;

case 2:
    String selectQuery = "SELECT * FROM courses";
    ResultSet rs = stmt.executeQuery(selectQuery);
    System.out.println("\n--- Course List ---");
    while (rs.next()) {
        System.out.println("ID: " + rs.getInt("course_id") +
                ", Name: " + rs.getString("course_name") +
                ", Faculty: " + rs.getString("faculty") +
                ", Credits: " + rs.getInt("credits"));
    }
    break;

case 3:
    System.out.print("Enter Course ID to Update: ");
    int updateId = sc.nextInt();
    sc.nextLine();
    System.out.print("Enter New Faculty Name: ");
    String newFaculty = sc.nextLine();
```

```
            System.out.print("Enter New Credits: ");
            int newCredits = sc.nextInt();

            String updateQuery = "UPDATE courses SET faculty = ?, credits = ? WHERE course_id =
?";
            try (PreparedStatement pstmt = conn.prepareStatement(updateQuery)) {
                pstmt.setString(1, newFaculty);
                pstmt.setInt(2, newCredits);
                pstmt.setInt(3, updateId);
                int rows = pstmt.executeUpdate();
                System.out.println(rows > 0 ? " Course updated." : " Course not found.");
            }
            break;

        case 4:
            System.out.print("Enter Course ID to Delete: ");
            int deleteId = sc.nextInt();

            String deleteQuery = "DELETE FROM courses WHERE course_id = ?";
            try (PreparedStatement pstmt = conn.prepareStatement(deleteQuery)) {
                pstmt.setInt(1, deleteId);
                int rows = pstmt.executeUpdate();
                System.out.println(rows > 0 ? " ✅ Course deleted." : " ❌ Course not found.");
            }
            break;

        case 5:
            System.out.println("Exiting...");
            break;

        default:
```

```
                System.out.println("Invalid choice.");
            }

        } while (choice != 5);

    } catch (Exception e) {
        System.out.println(" Error: " + e.getMessage());
    } finally {
        sc.close();
    }
  }
}
```

Case Study 2: Product Inventory System

```
CREATE DATABASE inventory_db;
USE inventory_db;

CREATE TABLE products (
  product_id INT PRIMARY KEY,
  product_name VARCHAR(100),
  quantity INT,
  price DECIMAL(10,2)
);
```

Java JDBC Code

```
package jdbc.demo;

import java.sql.*;
import java.util.Scanner;

public class ProductInventorySystem {
```

```
static final String DB_URL = "jdbc:mysql://localhost:3306/inventory_db";
static final String USER = "root";
static final String PASS = "Hema@1802"; // Replace with your actual MySQL password
        private static final String DB2_URL = null;

public static void main(String[] args) {
   Scanner sc = new Scanner(System.in);

   try (
      Connection conn = DriverManager.getConnection(DB2_URL, USER, PASS);
      Statement stmt = conn.createStatement();
   ) {
      Class.forName("com.mysql.cj.jdbc.Driver");

      int choice;
      do {
         System.out.println("\n--- Product Inventory System ---");
         System.out.println("1. Add Product");
         System.out.println("2. View All Products");
         System.out.println("3. Update Quantity");
         System.out.println("4. Delete Product");
         System.out.println("5. Exit");
         System.out.print("Enter your choice: ");
         choice = sc.nextInt();
         sc.nextLine(); // Clear buffer

         switch (choice) {
            case 1:
               System.out.print("Enter Product ID: ");
               int id = sc.nextInt();
```

```
        sc.nextLine();
        System.out.print("Enter Product Name: ");
        String name = sc.nextLine();
        System.out.print("Enter Quantity: ");
        int qty = sc.nextInt();
        System.out.print("Enter Price: ");
        double price = sc.nextDouble();

        String insertQuery = "INSERT INTO products VALUES (?, ?, ?, ?)";
        try (PreparedStatement pstmt = conn.prepareStatement(insertQuery)) {
            pstmt.setInt(1, id);
            pstmt.setString(2, name);
            pstmt.setInt(3, qty);
            pstmt.setDouble(4, price);
            pstmt.executeUpdate();
            System.out.println(" Product added successfully.");
        }
        break;

    case 2:
        String selectQuery = "SELECT * FROM products";
        ResultSet rs = stmt.executeQuery(selectQuery);
        System.out.println("\n--- Product List ---");
        while (rs.next()) {
            System.out.println("ID: " + rs.getInt("product_id") +
                    ", Name: " + rs.getString("product_name") +
                    ", Qty: " + rs.getInt("quantity") +
                    ", Price: ₹" + rs.getDouble("price"));
        }
        break;
```

```
case 3:
    System.out.print("Enter Product ID to Update: ");
    int updateId = sc.nextInt();
    System.out.print("Enter New Quantity: ");
    int newQty = sc.nextInt();

    String updateQuery = "UPDATE products SET quantity = ? WHERE product_id = ?";
    try (PreparedStatement pstmt = conn.prepareStatement(updateQuery)) {
        pstmt.setInt(1, newQty);
        pstmt.setInt(2, updateId);
        int rows = pstmt.executeUpdate();
        System.out.println(rows > 0 ? "Quantity updated." : " Product not found.");
    }
    break;

case 4:
    System.out.print("Enter Product ID to Delete: ");
    int deleteId = sc.nextInt();

    String deleteQuery = "DELETE FROM products WHERE product_id = ?";
    try (PreparedStatement pstmt = conn.prepareStatement(deleteQuery)) {
        pstmt.setInt(1, deleteId);
        int rows = pstmt.executeUpdate();
        System.out.println(rows > 0 ? " Product deleted." : " Product not found.");
    }
    break;

case 5:
    System.out.println("Exiting...");
    break;
```

```
                default:
                    System.out.println("Invalid choice.");
                }

            } while (choice != 5);

        } catch (Exception e) {
            System.out.println(" Error: " + e.getMessage());
        } finally {
            sc.close();
        }
    }
}
```