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

```
package Day5Task;
import java.util.Arrays;
import java.util.List;
class Employee {
    String name;
    double salary;

    // Constructor
    public Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    // toString method to print employee details
    @Override
    public String toString() {
        return "Name: " + name + ", Salary: " + salary;
    }
}

public class HRManagement {
    public static void main(String[] args) {
        List<Employee> employees = Arrays.asList(
            new Employee("Ravi", 55000),
            new Employee("Amit", 70000),
            new Employee("Kiran", 45000),
            new Employee("Zara", 90000)
        );
```

```
        // Sort employees by name
        employees.sort((e1, e2) -> e1.name.compareTo(e2.name));

        // Print all employee names (using lambda)
        System.out.println("Employees sorted by name:");
        employees.forEach(e -> System.out.println(e.name));

        // Filter salary > 50000
        System.out.println("\nEmployees with salary > 50000:");
        for (Employee e : employees) {
            if (e.salary > 50000) {
                System.out.println(e); // toString() is used here
            }
        }
    }
}
```

2.Stream API & Operators – Case Study: Order Processing System

```
package Day5Task;
import java.util.*;
import java.util.stream.*;
class Order {
    String orderId;
    String customerName;
    String category;
    double amount;

    public Order(String orderId, String customerName, String category, double amount) {
```

```
        this.orderId = orderId;
        this.customerName = customerName;
        this.category = category;
        this.amount = amount;
    }

    @Override
    public String toString() {
        return "OrderID: " + orderId + ", Customer: " + customerName +
                ", Category: " + category + ", Amount: " + amount;
    }
}

public class OrderProcessingSystem {
    public static void main(String[] args) {
        List<Order> orders = Arrays.asList(
            new Order("O001", "Alice", "Electronics", 12000),
            new Order("O002", "Bob", "Fashion", 3000),
            new Order("O003", "Alice", "Electronics", 15000),
            new Order("O004", "Charlie", "Books", 800),
            new Order("O005", "Bob", "Electronics", 7000),
            new Order("O006", "Alice", "Books", 900)
        );

        // Filter orders above 5000
        System.out.println("Orders with amount > 5000:");
        orders.stream()
              .filter(order -> order.amount > 5000)
              .forEach(System.out::println);

        // Count total orders per customer
```

```
        System.out.println("\nTotal orders per customer:");
        Map<String, Long> orderCountPerCustomer = orders.stream()
            .collect(Collectors.groupingBy(
                order -> order.customerName, Collectors.counting()));

        orderCountPerCustomer.forEach((customer, count) ->
            System.out.println(customer + " -> " + count + " orders"));

        //  Sort and group orders by category
        System.out.println("\nOrders grouped by category (sorted by amount):");
        Map<String, List<Order>> ordersByCategory = orders.stream()
            .sorted(Comparator.comparingDouble(order -> order.amount))
            .collect(Collectors.groupingBy(order -> order.category));

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

3.Functional Interfaces – Case Study: Custom Logger

```
package Day5Task;
import java.util.function.Predicate;
import java.util.function.Consumer;
// Custom Functional Interface
@FunctionalInterface
interface LogFilter {
    boolean shouldLog(String level, String message);
```

```
}

public class CustomLogger {

   // Using Custom Functional Interface
   public static void log(String level, String message, LogFilter filter) {
      if (filter.shouldLog(level, message)) {
         System.out.println("[" + level + "] " + message);
      }
   }

   // Using Predicate + Consumer (Built-in Functional Interfaces)
   public static void log(String message, Predicate<String> condition, Consumer<String> action) {
      if (condition.test(message)) {
         action.accept(message);
      }
   }

   public static void main(String[] args) {

      System.out.println("Using Custom Functional Interface:");
      LogFilter errorOrWarnFilter = (level, msg) ->
            level.equalsIgnoreCase("ERROR") || level.equalsIgnoreCase("WARN");

      log("INFO", "Application started successfully", errorOrWarnFilter); // Won't log
      log("WARN", "Memory usage is high", errorOrWarnFilter);           // Will log
      log("ERROR", "NullPointerException occurred", errorOrWarnFilter);  // Will log

      System.out.println("\ Using Predicate and Consumer:");
      Predicate<String> keywordFilter = msg -> msg.contains("fail") || msg.contains("error");
      Consumer<String> customLogger = msg -> System.out.println("[CUSTOM LOG] " + msg);
```

```
        log("User login failed due to timeout", keywordFilter, customLogger); // Will log
        log("Everything is running fine", keywordFilter, customLogger);       // Won't log
    }
}
```

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

```
package Day5Task;
interface PaymentGateway {
    void pay(double amount); // Abstract method to be implemented

    // Default method for logging transaction
    default void logTransaction(String method, double amount) {
        System.out.println("Payment of ₹" + amount + " done via " + method);
    }
}

// PayPal implementation
class PayPalPayment implements PaymentGateway {
    public void pay(double amount) {
        logTransaction("PayPal", amount);
        System.out.println("Payment of ₹" + amount + " processed via PayPal.");
    }
}

// UPI implementation
class UpiPayment implements PaymentGateway {
    public void pay(double amount) {
        logTransaction("UPI", amount);
```

```
        System.out.println("Payment of ₹" + amount + " processed via UPI.");
    }
}

// Card implementation
class CardPayment implements PaymentGateway {
    public void pay(double amount) {
        logTransaction("Card", amount);
        System.out.println("Payment of ₹" + amount + " processed via Card.");
    }
}

public class PaymentGatewayIntegration {
    public static void main(String[] args) {
        PaymentGateway paypal = new PayPalPayment();
        PaymentGateway upi = new UpiPayment();
        PaymentGateway card = new CardPayment();

        paypal.pay(5000);
        upi.pay(1500);
        card.pay(10000);
    }
}
```

5.Method References – Case Study: Notification System

```
package Day5Task;
import java.util.function.Consumer;
import java.util.Arrays;
```

```
import java.util.List;
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 sendPush(String message) {
        System.out.println("Push Notification sent: " + message);
    }
}

public class NotificationSystem {
    public static void main(String[] args) {
        String msg = "Your order has been shipped.";

        // List of consumers using method references
        List<Consumer<String>> notifiers = Arrays.asList(
            NotificationService::sendEmail,
            NotificationService::sendSMS,
            NotificationService::sendPush
        );

        // Send notification via all channels
        for (Consumer<String> notifier : notifiers) {
            notifier.accept(msg);
        }
    }}
```

Optional Class – Case Study: User Profile Management

```
package Day5Task;



import java.util.Optional;

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;
    }
}

public class OptionalExample {
```

```
    public static void main(String[] args) {
        User user1 = new User("Ravi", "ravi@example.com", null);
        User user2 = new User("Meena", null, "9876543210");

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

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

        user.getEmail().ifPresentOrElse(
            email -> System.out.println("Email: " + email),
            () -> System.out.println("Email not provided")
        );

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

6.Date and Time API – Case Study: Booking System

```
package Day5Task;
import java.time.*;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;
import java.util.Scanner;
```

```
public class BookingSystem {

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

        // Input: Check-in and Check-out Dates
        System.out.print("Enter Check-in Date (yyyy-MM-dd): ");
        LocalDate checkIn = LocalDate.parse(sc.nextLine());

        System.out.print("Enter Check-out Date (yyyy-MM-dd): ");
        LocalDate checkOut = LocalDate.parse(sc.nextLine());

        // Validate Dates
        if (checkOut.isBefore(checkIn)) {
            System.out.println("Error: Check-out date cannot be before check-in date.");
//          return;
        }

        // Calculate Stay Duration
        Period stayPeriod = Period.between(checkIn, checkOut);
        long totalDays = ChronoUnit.DAYS.between(checkIn, checkOut);

        System.out.println("\n Booking Details:");
        System.out.println("Check-in Date : " + checkIn);
        System.out.println("Check-out Date: " + checkOut);
        System.out.println("Stay Duration : " + totalDays + " days (" + stayPeriod.getMonths() + " months
and " + stayPeriod.getDays() + " days)");

        // Assume booking confirmation time now
        LocalDateTime bookingConfirmedAt = LocalDateTime.now();
        System.out.println("Booking Confirmed At: " +
bookingConfirmedAt.format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")));
```

```
        // Calculate expected arrival time (e.g., 3 hours from now)
        Duration travelDuration = Duration.ofHours(3);
        LocalDateTime arrivalTime = bookingConfirmedAt.plus(travelDuration);
        System.out.println("Expected Arrival Time: " +
arrivalTime.format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")));

        // Schedule room cleaning every 2 days of stay
        System.out.println("\n 🧹 Scheduled Room Cleaning:");
        LocalDate cleanDate = checkIn.plusDays(2);
        while (!cleanDate.isAfter(checkOut.minusDays(1))) {
            System.out.println("Cleaning Scheduled On: " + cleanDate);
            cleanDate = cleanDate.plusDays(2);
        }

        sc.close();
    }
}
```

7.Executor Service – Case Study: File Upload Service

```
package Day5Task;
import java.util.concurrent.*;
class FileUploader implements Runnable {
    private String fileName;

    public FileUploader(String fileName) {
        this.fileName = fileName;
    }
```

```
    @Override
    public void run() {
        System.out.println("Uploading started for: " + fileName + " | Thread: " +
Thread.currentThread().getName());
        try {
            Thread.sleep(2000); // Simulate upload time (2 seconds)
        } catch (InterruptedException e) {
            System.out.println("Upload interrupted for: " + fileName);
        }
        System.out.println("Upload completed for: " + fileName);
    }
}

public class FileUploadService {
    public static void main(String[] args) {
        // Create a fixed thread pool with 3 threads
        ExecutorService executor = Executors.newFixedThreadPool(3);

        // Simulate file uploads
        String[] files = {"resume.pdf", "photo.jpg", "report.docx"};

        for (String file : files) {
            FileUploader task = new FileUploader(file);
            executor.submit(task); // Submit task to thread pool
        }

        // Shut down the executor service gracefully
        executor.shutdown();

        System.out.println("Main thread continues while files are uploading...");
    }}
```