Day 5 Java8 Case Studies

Case Study 1: Lambda Expressions - Sorting and Filtering Employees

```
package day5_Assignment;

import java.util.*; public class

HRManager {
    public static void main(String[] args) { List<Employee> employees = new
        ArrayList<>(); employees.add(new Employee(101, "Alice", 50000));
        employees.add(new Employee(102, "Bob", 70000));
        employees.add(new Employee(103, "Charlie", 60000)); employees.add(new
        Employee(104, "David", 45000));

        employees.sort((e1, e2) -> e1.getName().compareTo(e2.getName())); employees.sort((e1, e2) ->
        Double.compare(e1.getSalary(), e2.getSalary()));

        List<Employee> highEarners = employees.stream()
            .filter(e -> e.getSalary() > 55000)
            .toList();

        highEarners.forEach(System.out::println);
    }
}

class Employee { private int id;
    private String name; private double
    salary;

    public Employee(int id, String name, double salary) { this.id = id;
        this.name = name; this.salary =
        salary;
    }

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

    public String toString() {
        return id + " " + name + " " + salary;
    }
}
```

Case Study 2: Stream API & Operators - Order Processing System

```
package day5_Assignment;


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

 public class OrderProcessor {
          public static void main(String[] args) { List<Order> orders =
                   Arrays.asList(
                            new Order(101, "Alice", "Electronics", 1200.00), new Order(102, "Bob",
                            "Books", 450.00),
                            new Order(103, "Alice", "Groceries", 200.00),
                            new Order(104, "Charlie", "Electronics", 1500.00), new Order(105, "Bob",
                            "Electronics", 900.00),
                            new Order(106, "Charlie", "Books", 300.00)
                   );

                   List<Order> filtered = orders.stream()
                            .filter(order -> order.getAmount() > 1000)
                            .toList();

                   Map<String, Long> count = orders.stream()
                                                                                    .collect(Collectors.groupingBy(Order::getCustomerName,
 Collectors.counting()));

                   Map<String, List<Order>> grouped = orders.stream()
                            .sorted(Comparator.comparing(Order::getAmount).reversed())
                            .collect(Collectors.groupingBy(Order::getCategory));

                   filtered.forEach(System.out::println);
                   count.forEach((k, v) -> System.out.println(k + ": " + v)); grouped.forEach((k, v) -> System.out.println(k + ": " + v));
          }
 }

 class Order {
          private int id;
          private String customerName; private String
          category; private double amount;

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

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

          public String toString() {
                   return id + " " + customerName + " " + category + " " + amount;
          }
 }
```

Case Study 3: Functional Interfaces – Custom Logger using both custom and built-in functional interfaces

```
package day5_Assignment;
@FunctionalInterface
public interface LogFilter {
   boolean shouldLog(String message);
}
```

This interface defines a single method for custom filter logic.

```
public class Logger {
   public static void log(String message, LogFilter filter) {
      if (filter.shouldLog(message)) {
         System.out.println("LOG: " + message);
      }
   }
}

package day5_Assignment;
public class Main {
   public static void main(String[] args) {
      // Log only ERROR messages
      LogFilter errorFilter = msg -> msg.startsWith("ERROR");
      Logger.log("ERROR: Disk full", errorFilter);
      Logger.log("INFO: User logged in", errorFilter);

      // Log messages longer than 15 characters
      LogFilter lengthFilter = msg -> msg.length() > 15;
      Logger.log("Short", lengthFilter);
      Logger.log("This is a detailed log message", lengthFilter);
   }
    }
```

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

```
package day5_Assignment;
public interface PaymentGateway {
   void pay(double amount); // Abstract method

   // Shared logic for all payment types
   default void logTransaction(double amount, String method) {
      System.out.println("Transaction of ₹" + amount + " processed via " + method);
   }
}

package day5_Assignment;
public class PayPalPayment implements PaymentGateway {
   public void pay(double amount) {
      System.out.println("Paying ₹" + amount + " using PayPal...");
      logTransaction(amount, "PayPal");
   }
}
```

```
package day5_Assignment;
public class UPIPayment implements PaymentGateway {
   public void pay(double amount) {
      System.out.println("Paying ₹" + amount + " using UPI...");
      logTransaction(amount, "UPI");
   }
}
```

```
package day5_Assignment;
public class CardPayment implements PaymentGateway {
   public void pay(double amount) {
      System.out.println("Paying ₹" + amount + " using Credit/Debit Card...");
      logTransaction(amount, "Card");
   }
}
```

```
package day5_Assignment;
public class PaymentTest {
   public static void main(String[] args) {
      PaymentGateway paypal = new PayPalPayment();
      PaymentGateway upi = new UPIPayment();
      PaymentGateway card = new CardPayment();

      paypal.pay(5000);
      upi.pay(1200);
      card.pay(2500);
   }
    }
```

Case Study 5: Method References – Notification System

```
package day5_Assignment;
public 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 Notification: " + msg);
   }
}
```

```
package day5_Assignment;
import java.util.function.Consumer;

public class NotificationDispatcher {
   public void dispatch(String message, Consumer<String> notifier) {
      notifier.accept(message);
   }
}


package day5_Assignment;
public class Main {
    public static void main(String[] args) {
        NotificationDispatcher dispatcher = new NotificationDispatcher();

        // Using method references to static methods
        dispatcher.dispatch("Welcome to our service!", NotificationService::sendEmail);
        dispatcher.dispatch("Your OTP is 123456", NotificationService::sendSMS);
        dispatcher.dispatch("You have a new message", NotificationService::sendPush);
    }
}
```

Case Study 6: Optional Class – User Profile Management

```
package day5_Assignment;
import java.util.Optional;

public class UserProfile {
    private String name;
    private Optional<String> email;
    private Optional<String> phone;

    public UserProfile(String name, String email, String phone) {
        this.name = name;
        // Wrap possibly null values in Optional
        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;
    }
}



package day5_Assignment;
public class ProfileManager {
    public static void main(String[] args) {
        UserProfile user1 = new UserProfile("Alice", "alice@example.com", null);
        UserProfile user2 = new UserProfile("Bob", null, "9876543210");

        showUserDetails(user1);
        System.out.println("------");
        showUserDetails(user2);
    }
    public static void showUserDetails(UserProfile user) {
        System.out.println("Name: " + user.getName());

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

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

Case Study 7: Date and Time API – Booking System

```
package day5_Assignment;
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
import java.time.Period;

public class Booking {
    private String guestName;
    private LocalDate checkIn;
    private LocalDate checkOut;

    public Booking(String guestName, LocalDate checkIn, LocalDate checkOut) {
        this.guestName = guestName;
        this.checkIn = checkIn;
        this.checkOut = checkOut;
    }

    public void displayBookingDetails() {
        System.out.println("Guest: " + guestName);
        System.out.println("Check-in: " + checkIn);
        System.out.println("Check-out: " + checkOut);

        if (checkOut.isBefore(checkIn)) {
            System.out.println(" Invalid booking: Check-out date is before check-in.");
            return;
        }

        long days = ChronoUnit.DAYS.between(checkIn, checkOut);
        System.out.println("Stay Duration: " + days + " nights");

        Period period = Period.between(checkIn, checkOut);
        System.out.println("Period Object: " + period.getDays() + " days");
    }
}
```

```
package day5_Assignment;

import java.time.LocalDate;

public class BookingSystem {
    public static void main(String[] args) {
        // Valid booking
        Booking booking1 = new Booking("Alice", LocalDate.of(2025, 7, 28), LocalDate.of(2025, 8, 2));
        booking1.displayBookingDetails();

        System.out.println("\n---\n");

        // Invalid booking
        Booking booking2 = new Booking("Bob", LocalDate.of(2025, 8, 5), LocalDate.of(2025, 8, 3));
        booking2.displayBookingDetails();

        System.out.println("\n---\n");

        // Recurring event (e.g., weekly maintenance)
        scheduleRecurringEvent(LocalDate.now(), 4); // 4 weeks from now
    }

    public static void scheduleRecurringEvent(LocalDate startDate, int weeks) {
        System.out.println("Maintenance schedule:");
        for (int i = 0; i < weeks; i++) {
            LocalDate eventDate = startDate.plusWeeks(i);
            System.out.println("Week " + (i + 1) + ": " + eventDate);
        }
    }
}
```

Case Study 8: ExecutorService – File Upload Service

```
package day5_Assignment;
public class FileUploader implements Runnable {
    private String fileName;

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

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

```
package day5_Assignment;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

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

        // Simulate user uploading 5 files
        String[] files = {
            "profile.jpg",
            "resume.pdf",
            "video.mp4",
            "report.docx",
            "invoice.xlsx"
        };

        for (String file : files) {
            FileUploader task = new FileUploader(file);
            executor.submit(task);
        }

        // Shut down after all tasks are submitted
        executor.shutdown();

        System.out.println("Main thread continues to respond while uploads happen in background.");
    }
}
```