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.

Step 1: Define the Notification Classes

```
public class EmailService {
    public void sendEmail(String message) {
        System.out.println("[EMAIL] " + message);
    }
}

public class SMSService {
    public void sendSMS(String message) {
        System.out.println("[SMS] " + message);
    }
}
public class PushService {
    public static void sendPush(String message) {
        System.out.println("[PUSH] " + message);
    }
}
```

Step 2: Define Functional Interface (if needed)

```
@FunctionalInterface
public interface Notifier {
    void notify(String message);
}
```

Step 3: Use Method References in Dispatcher

```
public class NotificationDispatcher {
    public void dispatch(String message, Notifier notifier) {
        notifier.notify(message);
    }
}
```

Step 4: Main – Using Method References

```
public class NotificationApp {
    public static void main(String[] args) {

        NotificationDispatcher dispatcher = new NotificationDispatcher();

        EmailService emailService = new EmailService();
        SMSService smsService = new SMSService();

        // Instance Method Reference
        dispatcher.dispatch("Order Confirmed!", emailService::sendEmail);
        dispatcher.dispatch("OTP: 123456", smsService::sendSMS);

        // Static Method Reference
        dispatcher.dispatch("New Offer Available!", PushService::sendPush);
    }
}
```

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.

Step 1: Define the UserProfile **Class**

```
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;
        this.email = Optional.ofNullable(email);  // wrap nullable field
        this.phone = Optional.ofNullable(phone);  // wrap nullable field
    }
    public String getName() {
        return name;
    }
    public Optional<String> getEmail() {
        return email;
    }
    public Optional<String> getPhone() {
        return phone;
    }
}
```

Step 2: Usage Example – Main Application

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

        // Access email safely using ifPresent
        user1.getEmail().ifPresent(email ->
            System.out.println(user1.getName() + "'s email: " + email));
```

```
        // Use orElse to provide fallback
        String phone1 = user1.getPhone().orElse("Phone not provided");
        System.out.println(user1.getName() + "'s phone: " + phone1);

        String email2 = user2.getEmail().orElse("Email not provided");
        System.out.println(user2.getName() + "'s email: " + email2);

        //  Use map to transform if present
        user2.getPhone().map(p -> "+91 " + p)
                        .ifPresent(formatted -> System.out.println(user2.getName() + "'s phone (formatted): " +
formatted));
    }
}
```

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.

```
public class BookingSystem {
    public static void main(String[] args) {

        //  Booking dates
        LocalDate checkIn = LocalDate.of(2025, 8, 5);
        LocalDate checkOut = LocalDate.of(2025, 8, 10);
```

```
        //  1. Validate check-in/check-out dates
        if (checkOut.isBefore(checkIn) || checkOut.equals(checkIn)) {
            System.out.println("Invalid booking: Check-out must be after check-in.");
        } else {
            System.out.println(" Booking is valid.");
        }

        //  2. Calculate stay duration
        long stayDays = ChronoUnit.DAYS.between(checkIn, checkOut);
        System.out.println("Stay Duration: " + stayDays + " days");

        //  3. Schedule recurring event: Daily room cleaning at 10:00 AM
        LocalDateTime cleaningStart = checkIn.atTime(10, 0);
        for (int i = 0; i < stayDays; i++) {
            LocalDateTime cleaningTime = cleaningStart.plusDays(i);
            System.out.println("Scheduled cleaning on: " + cleaningTime);
        }

        // 4. Use Period to calculate date difference
        Period period = Period.between(checkIn, checkOut);
        System.out.println("Period: " + period.getDays() + " days");

        // 5. Time-based duration example
        LocalDateTime maintenanceStart = LocalDateTime.of(2025, 8, 6, 1, 0);
        LocalDateTime maintenanceEnd = LocalDateTime.of(2025, 8, 6, 4, 30);
        Duration maintenanceDuration = Duration.between(maintenanceStart, maintenanceEnd);
        System.out.println("Maintenance Duration: " + maintenanceDuration.toHours() + " hours and "
                + maintenanceDuration.toMinutesPart() + " minutes");
    }
}
```

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.

Step 1: Simulate File Upload Task

```
public class FileUploadTask implements Runnable {
    private String fileName;

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

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

Step 2: Main Class with ExecutorService

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

public class FileUploadService {
    public static void main(String[] args) {

        // Create a thread pool with 3 threads
        ExecutorService executor = Executors.newFixedThreadPool(3);

        //  Submit multiple upload tasks
        executor.submit(new FileUploadTask("photo1.jpg"));
        executor.submit(new FileUploadTask("resume.pdf"));
        executor.submit(new FileUploadTask("invoice.docx"));
        executor.submit(new FileUploadTask("video.mp4"));
        executor.submit(new FileUploadTask("audio.mp3"));

        //  Shutdown executor after tasks are submitted
        executor.shutdown();
        System.out.println("All file uploads initiated...");
    }
}
```