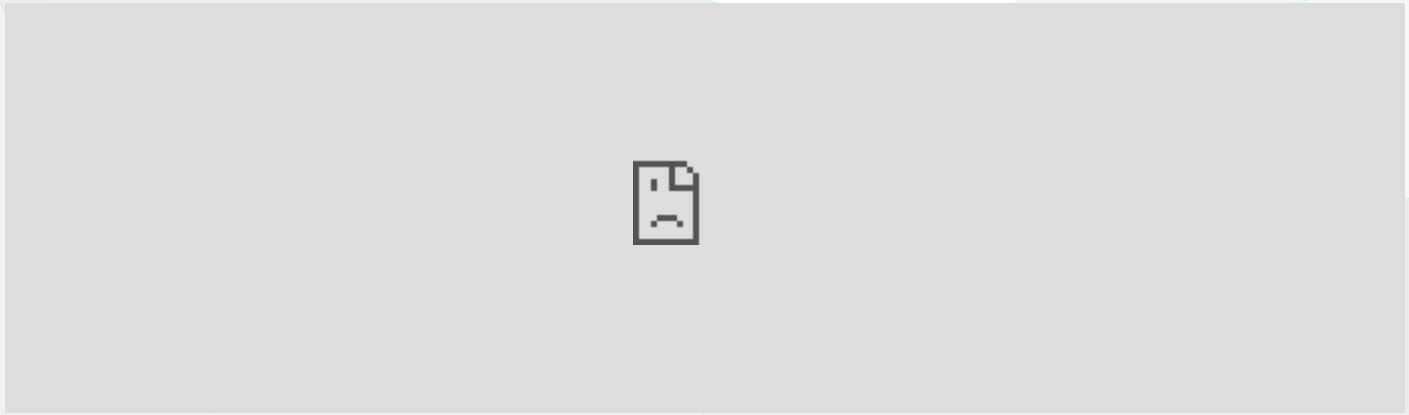
CEN206 Nesne Yönelimli Programlama

Hafta-14 (Vaka Çalışmaları - Tasarım Desenleri ve Yeniden Yapılandırma Pratikte)

Bahar Dönemi, 2025-2026

İndir BELGE-PDF, BELGE-DOCX, SLAYT



Hafta-14 Genel Bakış

Vaka Çalışmaları - Tasarım Desenleri ve Yeniden Yapılandırma Pratikte

Modül	Konu
A	Gerçek Dünya Tasarım Deseni Uygulamaları (Real-World Design Pattern Applications)
B	Vaka Çalışması 1 -- E-Ticaret Sipariş İşleme Sistemi
C	Vaka Çalışması 2 -- Geri Al/Yinele ile Metin Düzenleyici
D	Vaka Çalışması 3 -- Bildirim Sistemi Yeniden Yapılandırması
E	En İyi Uygulamalar ve Anti-Desenler (Anti-Patterns)

Neden Vaka Çalışmaları?

- Tasarım desenleri ve yeniden yapılandırma (refactoring) teknikleri güçlü araçlardır, ancak gerçek değerleri **gerçek dünya uygulamalarında** görülür.
- Vaka çalışmaları bize şunları anlamamıza yardımcı olur:
  - Bir deseni **ne zaman** uygulamalı
  - Desenlerin bir sistemde **nasıl** birlikte çalıştığı
  - Kod kalitesine **ne gibi** iyileştirmeler getirdikleri
  - Yeniden yapılandırmanın (refactoring) en büyük etkiyi **nerede** yarattığı

Modül A: Gerçek Dünya Tasarım Deseni Uygulamaları

Tasarım Desenleri Java Çerçevelerinde Nasıl Görünür

Modül A Taslağı

Gerçek Dünya Tasarım Deseni Uygulamaları (Real-World Design Pattern Applications)

Tasarım desenleri sadece akademik kavramlar değildir. Her gün kullandığınız üretim çerçevelerinde ve kütüphanelerde yaygın olarak kullanılırlar.

1. Singleton (Tekil): `Runtime.getRuntime()`
2. Iterator (Yineleyici): `java.util.Iterator`
3. Observer (Gözlemci): `EventListener`
4. Factory Method (Fabrika Yöntemi): `Calendar.getInstance()`
5. Decorator (Dekoratör): `java.io` Akışları
6. Strategy (Strateji): `java.util.Comparator`
7. Template Method (Şablon Yöntemi): `java.io.InputStream`
8. Adapter (Uyarlayıcı): `Arrays.asList()`

A1: Singleton (Tekil) -- `Runtime.getRuntime()`

Desen: Singleton, bir sınıfın yalnızca bir örneğe (instance) sahip olmasını ve genel bir erişim noktası sağlamasını garanti eder.

Nerede görülür: `java.lang.Runtime` -- her Java uygulamasında tam olarak bir `Runtime` nesnesi bulunur.

```
public class SingletonExample {
    public static void main(String[] args) {
        // Runtime, Singleton desenini kullanır
        Runtime runtime1 = Runtime.getRuntime();
        Runtime runtime2 = Runtime.getRuntime();

        // Her iki referans aynı nesneyi gösterir
        System.out.println(runtime1 == runtime2); // true

        // Singleton'ı kullanarak sistem bilgisi al
        System.out.println("Available processors: "
            + runtime1.availableProcessors());
        System.out.println("Free memory: "
            + runtime1.freeMemory() + " bytes");
    }
}
```

A1. Singleton (Tekil) -- `Runtime` Dahili Olarak Nasıl Çalışır

JDK, `Runtime` sınıfını klasik bir Singleton olarak uygular:

```
public class Runtime {
    // İstekli başlatma -- örnek sınıf yüklenirken oluşturulur
    private static final Runtime currentRuntime = new Runtime();

    // Özel yapıcı dışarıdan örneklemeyi engeller
    private Runtime() {}

    // Genel erişim noktası
    public static Runtime getRuntime() {
        return currentRuntime;
    }

    public int availableProcessors() {
        // yerel uygulama (native implementation)
    }

    public long freeMemory() {
        // yerel uygulama (native implementation)
    }
}
```

Temel çıkarım: JVM, yalnızca bir `Runtime` örneğinin var olmasını garanti eder; bu mantıklıdır çünkü uygulama başına yalnızca bir çalışma zamanı ortamı vardır.

Desen: Iterator, temel gösterimini açığa çıkarmadan bir koleksiyonun öğelerine sırayla erişmenin bir yolunu sağlar.

Nerede görülür: Her Java Collection `Iterable` arayüzünü uygular ve bir `Iterator` döndürür.

```
import java.util.*;

public class IteratorExample {
    public static void main(String[] args) {
        List<String> languages = new ArrayList<>(
            Arrays.asList("Java", "Python", "C++", "Rust")
        );

        // Iterator desenini açıkça kullanma
        Iterator<String> iterator = languages.iterator();
        while (iterator.hasNext()) {
            String lang = iterator.next();
            System.out.println(lang);
            if (lang.equals("C++")) {
                iterator.remove(); // yineleme sırasında güvenli silme
            }
        }

        // Geliştirilmiş for döngüsü arka planda Iterator kullanır
        for (String lang : languages) {
            System.out.println("Remaining: " + lang);
        }
```

A3. Observer (Gözlemci) -- `EventListener`

Desen: Observer, bire-çok bağımlılık tanımlar; böylece bir nesne durumunu değiştirdiğinde tüm bağımlıları bildirilir.

Nerede görülür: Java'nın olay işleme sistemi (AWT/Swing), `java.util.EventListener`, `PropertyChangeListener`.

```
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeSupport;

public class Stock {
    private String symbol;
    private double price;
    private PropertyChangeSupport support;

    public Stock(String symbol, double price) {
        this.symbol = symbol;
        this.price = price;
        this.support = new PropertyChangeSupport(this);
    }

    public void addObserver(PropertyChangeListener listener) {
        support.addPropertyChangeListener(listener);
    }

    public void setPrice(double newPrice) {
        double oldPrice = this.price;
        this.price = newPrice;
        // Tüm gözlemcileri bilgilendir
        support.firePropertyChange("price", oldPrice, newPrice);
    }

    public double getPrice() { return price; }
    public String getSymbol() { return symbol; }
}
```

```
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;

public class StockTrader implements PropertyChangeListener {
    private String traderName;

    public StockTrader(String name) {
        this.traderName = name;
    }

    @Override
    public void propertyChange(PropertyChangeEvent evt) {
        System.out.println(traderName + " notified: "
            + "Price changed from " + evt.getOldValue()
            + " to " + evt.getNewValue());
    }

    public static void main(String[] args) {
        Stock apple = new Stock("AAPL", 150.0);

        StockTrader trader1 = new StockTrader("Alice");
        StockTrader trader2 = new StockTrader("Bob");

        apple.addObserver(trader1);
        apple.addObserver(trader2);

        apple.setPrice(155.0); // Her iki yatırımcı bildirilir
        apple.setPrice(148.0); // Her iki yatırımcı tekrar bildirilir
    }
}
```

Çıktı:

```
Alice notified: Price changed from 150.0 to 155.0
Bob notified: Price changed from 150.0 to 155.0
Alice notified: Price changed from 155.0 to 148.0
```

A4: Factory Method (Fabrika Yöntemi) -- `Calendar.getInstance()`

Desen: Factory Method, bir nesne oluşturmak için bir arayüz tanımlar ancak hangi sınıfın örnekleneceğine alt sınıfların karar vermesine izin verir.

Nerede görülür: `java.util.Calendar.getInstance()` yerel ayara özgü bir takvim uygulaması döndürür.

```
import java.util.Calendar;
import java.util.Locale;

public class FactoryMethodExample {
    public static void main(String[] args) {
        // Factory Method -- çoğu yerel ayar için
        // GregorianCalendar döndürür
        Calendar cal1 = Calendar.getInstance();
        System.out.println("Default: "
            + cal1.getClass().getName());

        // Japonca yerel ayar ile Factory Method
        // JapaneseImperialCalendar döndürür
        Calendar cal2 = Calendar.getInstance(
            new Locale("ja", "JP", "JP"));
        System.out.println("Japanese: "
            + cal2.getClass().getName());

        // İstemci kodu herhangi bir Calendar ile çalışır
        System.out.println("Year: "
            + cal1.get(Calendar.YEAR));
        System.out.println("Month: "
            + (cal1.get(Calendar.MONTH) + 1));
        System.out.println("Day: "
            + cal1.get(Calendar.DAY_OF_MONTH));
    }
}
```

A5. Decorator (Dekoratör) -- `java.io` Akışları

Desen: Decorator, bir nesneye dinamik olarak ek sorumluluklar ekler ve alt sınıflamaya esnek bir alternatif sağlar.

Nerede görülür: Tüm `java.io` akış hiyerarşisi Decorator desenini kullanır.

```
import java.io.*;

public class DecoratorExample {
    public static void main(String[] args) throws Exception {
        // Temel bileşen: FileReader
        // Dekoratör 1: BufferedReader tamponlama ekler
        // Her dekoratör önceki bileşeni sarar
        try (BufferedReader reader = new BufferedReader(
                new FileReader("example.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        }

        // Başka bir örnek: OutputStream dekoratörleri
        // FileOutputStream -> BufferedOutputStream -> DataOutputStream
        try (DataOutputStream out = new DataOutputStream(
                new BufferedOutputStream(
                    new FileOutputStream("data.bin")))) {
            out.writeInt(42);
            out.writeDouble(3.14);
            out.writeUTF("Hello, Decorator!");
        }

        // Karşılık gelen dekoratörlerle geri okuma
        try (DataInputStream in = new DataInputStream(
                new BufferedInputStream(
                    new FileInputStream("data.bin")))) {
            System.out.println("Int: " + in.readInt());
            System.out.println("Double: " + in.readDouble());
            System.out.println("String: " + in.readUTF());
        }
    }
}
```

A6. Strategy (Strateji) -- `java.util.Comparator`

Desen: Strategy, bir algoritma ailesi tanımlar, her birini kapsüller ve bunları birbirinin yerine kullanılabilir hale getirir.

Nerede görülür: `java.util.Comparator` çalışma zamanında sıralama algoritmasını değiştirmeye olanak tanır.

```
import java.util.*;

public class StrategyExample {
    public static void main(String[] args) {
        List<String> names = new ArrayList<>(
            Arrays.asList("Charlie", "Alice", "Bob", "Diana")
        );

        // Strateji 1: Doğal sıralama (alfabetik)
        names.sort(Comparator.naturalOrder());
        System.out.println("Alphabetical: " + names);

        // Strateji 2: Ters sıralama
        names.sort(Comparator.reverseOrder());
        System.out.println("Reversed: " + names);

        // Strateji 3: Dize uzunluğuna göre
        names.sort(Comparator.comparingInt(String::length));
        System.out.println("By length: " + names);

        // Strateji 4: Lambda ile özel karşılaştırıcı
        names.sort((a, b) -> {
            int lastCharA = a.charAt(a.length() - 1);
            int lastCharB = b.charAt(b.length() - 1);
            return lastCharA - lastCharB;
        });
        System.out.println("By last char: " + names);
    }
}
```

A7. Template Method (Şablon Yöntemi) -- `java.io.InputStream`

Desen: Template Method, bir yöntemde algoritmanın iskeletini tanımlar ve bazı adımları alt sınıflara bırakır.

Nerede görülür: `java.io.InputStream.read(byte[], int, int)` soyut `read()` yöntemini çağırır.

```
import java.io.InputStream;
import java.io.IOException;

// Template Method'u gösteren özel InputStream
public class RandomInputStream extends InputStream {

    private int remaining;

    public RandomInputStream(int size) {
        this.remaining = size;
    }

    // Alt sınıflar bu tek soyut yöntemi UYGULAMALIDIR
    @Override
    public int read() throws IOException {
        if (remaining <= 0) return -1;
        remaining--;
        return (int) (Math.random() * 256);
    }

    // InputStream'deki şablon yöntemi dahili olarak read() kullanır:
    // public int read(byte[] b, int off, int len) {
    //     for (int i = 0; i < len; i++) {
    //         int c = read();  <-- bizim uygulamamızı çağırır
    //         if (c == -1) break;
    //         b[off + i] = (byte) c;
    //     }
    // }

    public static void main(String[] args) throws IOException {
        try (RandomInputStream ris = new RandomInputStream(5)) {
            byte[] buffer = new byte[5];
            int bytesRead = ris.read(buffer, 0, 5);
            System.out.println("Read " + bytesRead + " bytes");
            for (byte b : buffer) {
                System.out.print((b & 0xFF) + " ");
            }
        }
    }
}
```

Desen: Adapter, bir sınıfın arayüzünü istemcilerin beklediği başka bir arayüze dönüştürür.

Nerede görülür: `Arrays.asList()` bir diziyi `List` arayüzüne uyarlar.

```
import java.util.*;

public class AdapterExample {
    public static void main(String[] args) {
        String[] array = {"Java", "Python", "C++"};

        // Arrays.asList() diziyi List arayüzüne uyarlar
        List<String> list = Arrays.asList(array);

        // Artık dizi üzerinde List işlemlerini kullanabiliriz
        System.out.println("Element at 1: " + list.get(1));
        System.out.println("Contains Java: "
            + list.contains("Java"));
        System.out.println("Index of C++: "
            + list.indexOf("C++"));

        // Listedeki değişiklikler orijinal diziyi etkiler
        list.set(0, "Kotlin");
        System.out.println("Array[0]: " + array[0]); // Kotlin

        // Not: Yapısal değişiklikler istisna fırlatır
        // list.add("Rust"); // UnsupportedOperationException
        // list.remove(0);   // UnsupportedOperationException
    }
```

Modül A -- Özet

Tasarım Desenleri Java'da Her Yerde

Desen	Java Örneği	Amaç
Singleton (Tekil)	`Runtime.getRuntime()`	Tek örnek garantisi
Iterator (Yineleyici)	`Collection.iterator()`	Sıralı erişim
Observer (Gözlemci)	`PropertyChangeListener`	Olay bildirimi
Factory Method (Fabrika Yöntemi)	`Calendar.getInstance()`	Esnek nesne oluşturma

Bu desenleri anlamak, Java API'lerini daha etkili kullanmanıza ve daha iyi yazılım tasarlamanıza yardımcı olur.

Modül B: Vaka Çalışması 1

E-Ticaret Sipariş İşleme Sistemi

Modül B Taslağı

E-Ticaret Sipariş İşleme Sistemi (E-Commerce Order Processing System)

Bu vaka çalışması, birden fazla tasarım deseninin gerçek iş sorunlarını çözmek için birlikte nasıl çalıştığını gösterir.

1. Problem Tanımı
2. Desenler Olmadan Tasarım (Naif Yaklaşım)
3. Strategy (Strateji) Deseni Uygulaması (Ödeme)
4. State (Durum) Deseni Uygulaması (Sipariş Durumu)
5. Observer (Gözlemci) Deseni Uygulaması (Bildirimler)
6. Factory Method (Fabrika Yöntemi) Deseni Uygulaması (Kargo)
7. Tam Entegre Sistem
8. Öncesi/Sonrası Karşılaştırması

B1. Problem Tanımı

Çevrimiçi Mağaza Gereksinimleri

Bir çevrimiçi mağaza için sipariş işleme sistemi oluşturuyorsunuz. Sistem şunları yapmalıdır:

- **Birden fazla ödeme yöntemi kabul etme** -- Kredi Kartı, PayPal, Banka Havalesi, Kripto Para
- **Sipariş durumunu takip etme** -- Yeni, Ödendi, Kargoya Verildi, Teslim Edildi, İptal Edildi
- **Bildirim gönderme** -- Sipariş durumu değiştiğinde E-posta, SMS, Anlık bildirim
- **Farklı kargo yöntemlerini destekleme** -- Standart, Ekspres, Ertesi Gün, Uluslararası

Sistem **genişletilebilir** olmalıdır -- yeni ödeme yöntemleri, kargo seçenekleri ve bildirim kanalları eklemek kolay olmalıdır.

B2. Desenler Olmadan Tasarım (Naif Yaklaşım)

```
public class NaiveOrderProcessor {
    public void processPayment(String type, double amount) {
        if (type.equals("creditcard")) {
            System.out.println("Processing credit card: $" + amount);
            // credit card specific logic...
        } else if (type.equals("paypal")) {
            System.out.println("Processing PayPal: $" + amount);
            // paypal specific logic...
        } else if (type.equals("banktransfer")) {
            System.out.println("Processing bank transfer: $" + amount);
            // bank transfer specific logic...
        } else if (type.equals("crypto")) {
            System.out.println("Processing crypto: $" + amount);
            // crypto specific logic...
        }
    }

    public void updateStatus(String orderId, String newStatus) {
        if (newStatus.equals("paid")) {
            // validate payment...
            sendEmail(orderId, "Your order has been paid");
            sendSMS(orderId, "Your order has been paid");
        } else if (newStatus.equals("shipped")) {
            // validate transition...
            sendEmail(orderId, "Your order has been shipped");
            sendSMS(orderId, "Your order has been shipped");
        }
        // ... more conditions
    }

    public void createShipping(String type, String orderId) {
        if (type.equals("standard")) { /* ... */ }
        else if (type.equals("express")) { /* ... */ }
        // ... more conditions
    }

    private void sendEmail(String id, String msg) { /* ... */ }
    private void sendSMS(String id, String msg) { /* ... */ }
}
```

Sorunlar: Uzun if-else zincirleri, genişletmesi zor, Açık/Kapalı İlkesini ihlal eder, sıkı bağlılık (tight coupling).

B3. Strategy (Strateji) Deseni -- Ödeme Yöntemleri

Hedef: Her ödeme algoritmasını kapsülleyerek bağımsız olarak değiştirilebilmelerini sağlamak.

```
// Strateji arayüzü
public interface PaymentStrategy {
    boolean pay(double amount);
    String getPaymentMethod();
}

// Somut Strateji: Kredi Kartı
public class CreditCardPayment implements PaymentStrategy {
    private String cardNumber;
    private String cardHolder;

    public CreditCardPayment(String cardNumber, String cardHolder) {
        this.cardNumber = cardNumber;
        this.cardHolder = cardHolder;
    }

    @Override
    public boolean pay(double amount) {
        System.out.println("Paid $" + amount
            + " with Credit Card ending in "
            + cardNumber.substring(cardNumber.length() - 4));
        return true;
    }

    @Override
    public String getPaymentMethod() {
        return "Credit Card";
    }
}
```

B3. Strategy (Strateji) Deseni -- Diğer Ödeme Yöntemleri

```java
// Somut Strateji: PayPal
public class PayPalPayment implements PaymentStrategy {
    private String email;

    public PayPalPayment(String email) {
        this.email = email;
    }

    @Override
    public boolean pay(double amount) {
        System.out.println("Paid $" + amount
            + " via PayPal (" + email + ")");
        return true;
    }

    @Override
    public String getPaymentMethod() { return "PayPal"; }
}

// Somut Strateji: Kripto Para
public class CryptoPayment implements PaymentStrategy {
    private String walletAddress;

    public CryptoPayment(String walletAddress) {
        this.walletAddress = walletAddress;
    }

    @Override
    public boolean pay(double amount) {
        System.out.println("Paid $" + amount
            + " via Crypto wallet " + walletAddress);
        return true;
    }

    @Override
    public String getPaymentMethod() { return "Cryptocurrency"; }
}
```

B4. State (Durum) Deseni -- Sipariş Durumu Geçişleri

Hedef: Siparişin iç durumu değiştiğinde davranışını değiştirmesine izin vermek.

```
// Durum arayüzü
public interface OrderState {
    void next(Order order);
    void prev(Order order);
    void cancel(Order order);
    String getStatus();
}

// Somut Durum: Yeni Sipariş
public class NewOrderState implements OrderState {
    @Override
    public void next(Order order) {
        order.setState(new PaidOrderState());
        System.out.println("Order moved to PAID state.");
    }

    @Override
    public void prev(Order order) {
        System.out.println("Already at initial state.");
    }

    @Override
    public void cancel(Order order) {
        order.setState(new CancelledOrderState());
        System.out.println("Order has been CANCELLED.");
    }

    @Override
    public String getStatus() { return "NEW"; }
}
```

B4. State (Durum) Deseni -- Diğer Durumlar

```
public class PaidOrderState implements OrderState {
    @Override
    public void next(Order order) {
        order.setState(new ShippedOrderState());
        System.out.println("Order moved to SHIPPED state.");
    }

    @Override
    public void prev(Order order) {
        order.setState(new NewOrderState());
        System.out.println("Order moved back to NEW state.");
    }

    @Override
    public void cancel(Order order) {
        order.setState(new CancelledOrderState());
        System.out.println("Paid order CANCELLED. Refund initiated.");
    }

    @Override
    public String getStatus() { return "PAID"; }
}

public class ShippedOrderState implements OrderState {
    @Override
    public void next(Order order) {
        order.setState(new DeliveredOrderState());
        System.out.println("Order moved to DELIVERED state.");
    }

    @Override
    public void prev(Order order) {
        order.setState(new PaidOrderState());
        System.out.println("Order returned to PAID state.");
    }

    @Override
    public void cancel(Order order) {
        System.out.println("Cannot cancel shipped order.");
    }

    @Override
    public String getStatus() { return "SHIPPED"; }
}
```

B4. State (Durum) Deseni -- Son Durumlar (Terminal States)

```
public class DeliveredOrderState implements OrderState {
    @Override
    public void next(Order order) {
        System.out.println("Order already delivered.");
    }

    @Override
    public void prev(Order order) {
        System.out.println("Cannot revert a delivered order.");
    }

    @Override
    public void cancel(Order order) {
        System.out.println("Cannot cancel a delivered order.");
    }

    @Override
    public String getStatus() { return "DELIVERED"; }
}

public class CancelledOrderState implements OrderState {
    @Override
    public void next(Order order) {
        System.out.println("Cannot proceed from cancelled order.");
    }

    @Override
    public void prev(Order order) {
        System.out.println("Cannot revert a cancelled order.");
    }

    @Override
    public void cancel(Order order) {
        System.out.println("Order is already cancelled.");
    }

    @Override
    public String getStatus() { return "CANCELLED"; }
}
```

B5. Observer (Gözlemci) Deseni -- Bildirimler

Hedef: Sipariş durumu değiştiğinde birden fazla kanalı (e-posta, SMS, anlık bildirim) bilgilendirmek.

```
// Gözlemci arayüzü
public interface OrderObserver {
    void update(String orderId, String oldStatus, String newStatus);
}

// Somut Gözlemci: E-posta Bildirimi
public class EmailNotification implements OrderObserver {
    @Override
    public void update(String orderId, String oldStatus,
                       String newStatus) {
        System.out.println("[EMAIL] Order " + orderId
            + ": " + oldStatus + " -> " + newStatus);
    }
}

// Somut Gözlemci: SMS Bildirimi
public class SMSNotification implements OrderObserver {
    @Override
    public void update(String orderId, String oldStatus,
                       String newStatus) {
        System.out.println("[SMS] Order " + orderId
            + ": " + oldStatus + " -> " + newStatus);
    }
}

// Somut Gözlemci: Anlık Bildirim
public class PushNotification implements OrderObserver {
    @Override
    public void update(String orderId, String oldStatus,
                       String newStatus) {
        System.out.println("[PUSH] Order " + orderId
            + ": " + oldStatus + " -> " + newStatus);
    }
}
```

B6. Factory Method (Fabrika Yöntemi) -- Kargo

Hedef: Tam sınıflarını belirtmeden kargo nesneleri oluşturmak.

```
// Ürün arayüzü
public interface ShippingMethod {
    double calculateCost(double weight);
    String getEstimatedDelivery();
    String getMethodName();
}

// Somut Ürünler
public class StandardShipping implements ShippingMethod {
    @Override
    public double calculateCost(double weight) {
        return weight * 0.5;
    }
    @Override
    public String getEstimatedDelivery() {
        return "5-7 business days";
    }
    @Override
    public String getMethodName() { return "Standard"; }
}

public class ExpressShipping implements ShippingMethod {
    @Override
    public double calculateCost(double weight) {
        return weight * 1.5;
    }
    @Override
    public String getEstimatedDelivery() {
        return "2-3 business days";
    }
    @Override
    public String getMethodName() { return "Express"; }
}

public class OvernightShipping implements ShippingMethod {
    @Override
    public double calculateCost(double weight) {
        return weight * 3.0;
    }
    @Override
    public String getEstimatedDelivery() {
        return "Next business day";
    }
    @Override
    public String getMethodName() { return "Overnight"; }
}
```

B6. Factory Method (Fabrika Yöntemi) -- Kargo Fabrikası

```
// Fabrika
public class ShippingFactory {
    public static ShippingMethod createShipping(String type) {
        switch (type.toLowerCase()) {
            case "standard":
                return new StandardShipping();
            case "express":
                return new ExpressShipping();
            case "overnight":
                return new OvernightShipping();
            default:
                throw new IllegalArgumentException(
                    "Unknown shipping type: " + type);
        }
    }
}

// Kullanım
public class ShippingDemo {
    public static void main(String[] args) {
        ShippingMethod shipping =
            ShippingFactory.createShipping("express");
        double cost = shipping.calculateCost(2.5); // 2.5 kg
        System.out.println("Method: "
            + shipping.getMethodName());
        System.out.println("Cost: $" + cost);
        System.out.println("Delivery: "
            + shipping.getEstimatedDelivery());
    }
}
```

B7. Entegre Sipariş Sınıfı (Integrated Order Class)

```
import java.util.*;

public class Order {
    private String orderId;
    private double totalAmount;
    private double weight;
    private OrderState state;
    private List<OrderObserver> observers = new ArrayList<>();

    public Order(String orderId, double totalAmount, double weight) {
        this.orderId = orderId;
        this.totalAmount = totalAmount;
        this.weight = weight;
        this.state = new NewOrderState();
    }

    public void addObserver(OrderObserver observer) {
        observers.add(observer);
    }

    public void setState(OrderState newState) {
        String oldStatus = this.state.getStatus();
        this.state = newState;
        notifyObservers(oldStatus, newState.getStatus());
    }

    private void notifyObservers(String oldStatus, String newStatus) {
        for (OrderObserver observer : observers) {
            observer.update(orderId, oldStatus, newStatus);
        }
    }

    public void processPayment(PaymentStrategy paymentStrategy) {
        if (paymentStrategy.pay(totalAmount)) {
            state.next(this); // NEW'den PAID'e geçiş
        }
    }

    public void ship(String shippingType) {
        ShippingMethod shipping = ShippingFactory.createShipping(shippingType);
        double cost = shipping.calculateCost(weight);
        System.out.println("Shipping via " + shipping.getMethodName()
            + " | Cost: $" + cost
            + " | ETA: " + shipping.getEstimatedDelivery());
        state.next(this); // PAID'den SHIPPED'e geçiş
    }

    public void deliver() { state.next(this); }
    public void cancel() { state.cancel(this); }
    public String getStatus() { return state.getStatus(); }
    public String getOrderId() { return orderId; }
}
```

```
        order.addObserver(new EmailNotification());
        order.addObserver(new SMSNotification());
        order.addObserver(new PushNotification());

        System.out.println("Initial status: " + order.getStatus());
        System.out.println("---");

        // Ödemeyi işle (Strategy Deseni)
        PaymentStrategy payment = new CreditCardPayment(
            "4111111111111234", "John Doe");
        order.processPayment(payment);
        System.out.println("---");

        // Siparişi kargola (Factory Method Deseni)
        order.ship("express");
        System.out.println("---");

        // Siparişi teslim et (State Deseni)
        order.deliver();
        System.out.println("---");

        System.out.println("Final status: " + order.getStatus());
    }
}
```

Çıktı:

```
Initial status: NEW
---
Paid $299.99 with Credit Card ending in 1234
Order moved to PAID state.
[EMAIL] Order ORD-001: NEW -> PAID
[SMS] Order ORD-001: NEW -> PAID
[PUSH] Order ORD-001: NEW -> PAID
---
Shipping via Express | Cost: $5.25 | ETA: 2-3 business days
Order moved to SHIPPED state.
[EMAIL] Order ORD-001: PAID -> SHIPPED
[SMS] Order ORD-001: PAID -> SHIPPED
```

B8. Öncesi/Sonrası Karşılaştırması

Öncesi (Desenler Olmadan)

- Ödeme, durum, kargo, bildirimler için uzun `if-else` zincirleri
- Yeni bir ödeme yöntemi eklemek `processPayment()` metodunu değiştirmeyi gerektirir
- Yeni bir bildirim kanalı eklemek her durum geçişini değiştirmeyi gerektirir
- Durum geçişleri hataya açıktır -- doğrulama yoktur

Sonrası (Desenlerle)

- **Strategy:** Yeni ödeme yöntemleri = `PaymentStrategy` uygulayan yeni sınıf
- **State:** Net geçişler, geçersiz durum değişiklikleri imkansız
- **Observer:** Yeni bildirim kanalı = yeni `OrderObserver` uygulaması
- **Factory Method:** Yeni kargo tipi = yeni sınıf + fabrika güncellemesi

Modül B -- Özet

E-Ticaret Vaka Çalışmasından Temel Dersler

1. **Desenler belirli sorunları çözer** -- Çözdüğü sorun yoksa bir desen uygulamayın.
2. **Desenler birlikte çalışır** -- Strategy, State, Observer ve Factory Method aynı sistemde farklı endişeleri ele alır.
3. **Açık/Kapalı İlkesi (Open/Closed Principle)** -- Sistem genişletmeye açık (yeni ödeme yöntemleri, kargo tipleri) ama değiştirmeye kapalıdır.
4. **Tek Sorumluluk (Single Responsibility)** -- Her sınıfın değişmek için tek bir nedeni vardır.

Modül C: Vaka Çalışması 2

Geri Al/Yinele ile Metin Düzenleyici

Modül C Taslağı

Geri Al/Yinele ile Metin Düzenleyici (Text Editor with Undo/Redo)

Bu vaka çalışması, Command (Komut) ve Memento (Hatıra) desenlerinin birlikte çalışmasını gösterir.

1. Problem Tanımı
2. Command (Komut) Deseni (Düzenleyici İşlemleri)
3. Memento (Hatıra) Deseni (Geri Al/Yinele)
4. Composite (Bileşik) Desen (Belge Yapısı)
5. Entegre Metin Düzenleyici
6. Adım Adım Yeniden Yapılandırma

C1. Problem Tanımı

Metin Düzenleyici Gereksinimleri

Aşağıdaki özellikleri destekleyen basit bir metin düzenleyici oluşturun:

- İmleç konumunda **metin yazma**
- İmleç konumunda **metin silme**
- Seçili metne **Kalın/İtalik biçimlendirme**
- Tüm işlemler için **Geri Al/Yinele (Undo/Redo)**
- Bölümler, paragraflar ve karakterlerden oluşan **belge yapısı**

Düzenleyici tüm işlemlerin geçmişini hatırlamalı ve kullanıcıların bunları sırayla geri alıp yinelemesine izin vermelidir.

C2. Command (Komut) Deseni -- Düzenleyici İşlemleri

Hedef: Her işlemi bir nesne olarak kapsülleyerek geri alma desteği sağlamak.

```
// Komut arayüzü
public interface EditorCommand {
    void execute();
    void undo();
    String getDescription();
}

// Alıcı (Receiver) -- asıl metin tamponu
public class TextBuffer {
    private StringBuilder content = new StringBuilder();
    private int cursorPosition = 0;

    public void insert(String text, int position) {
        content.insert(position, text);
        cursorPosition = position + text.length();
    }

    public String delete(int position, int length) {
        String deleted = content.substring(position,
            position + length);
        content.delete(position, position + length);
        cursorPosition = position;
        return deleted;
    }

    public void setCursorPosition(int position) {
        this.cursorPosition = Math.min(position,
            content.length());
    }

    public int getCursorPosition() { return cursorPosition; }
    public String getContent() { return content.toString(); }
    public int getLength() { return content.length(); }
}
```

C2. Command (Komut) Deseni -- Somut Komutlar

```
// Somut Komut: Metin Ekle
public class InsertCommand implements EditorCommand {
    private TextBuffer buffer;
    private String text;
    private int position;

    public InsertCommand(TextBuffer buffer, String text,
                         int position) {
        this.buffer = buffer;
        this.text = text;
        this.position = position;
    }

    @Override
    public void execute() {
        buffer.insert(text, position);
    }

    @Override
    public void undo() {
        buffer.delete(position, text.length());
    }

    @Override
    public String getDescription() {
        return "Insert '" + text + "' at position " + position;
    }
}

// Somut Komut: Metin Sil
public class DeleteCommand implements EditorCommand {
    private TextBuffer buffer;
    private int position;
    private int length;
    private String deletedText; // geri alma için kaydedilir

    public DeleteCommand(TextBuffer buffer, int position,
                         int length) {
        this.buffer = buffer;
        this.position = position;
        this.length = length;
    }

    @Override
    public void execute() {
        deletedText = buffer.delete(position, length);
    }

    @Override
    public void undo() {
        buffer.insert(deletedText, position);
    }

    @Override
    public String getDescription() {
        return "Delete " + length + " chars at position "
            + position;
    }
}
```

C3. Memento (Hatıra) Deseni -- Düzenleyici Durum Anlık Görüntüleri

Hedef: Geri al/yinele için düzenleyicinin iç durumunu yakalamak ve geri yüklemek.

```
// Memento -- düzenleyici durumunun değişmez anlık görüntüsü
public class EditorMemento {
    private final String content;
    private final int cursorPosition;
    private final long timestamp;

    public EditorMemento(String content, int cursorPosition) {
        this.content = content;
        this.cursorPosition = cursorPosition;
        this.timestamp = System.currentTimeMillis();
    }

    public String getContent() { return content; }
    public int getCursorPosition() { return cursorPosition; }
    public long getTimestamp() { return timestamp; }
}

// Caretaker (Bakıcı) -- memento geçmişini yönetir
public class EditorHistory {
    private final List<EditorMemento> undoStack = new ArrayList<>();
    private final List<EditorMemento> redoStack = new ArrayList<>();

    public void save(EditorMemento memento) {
        undoStack.add(memento);
        redoStack.clear(); // yeni eylem yinele geçmişini geçersiz kılar
    }

    public EditorMemento undo() {
        if (undoStack.size() <= 1) return null; // geri alınacak bir şey yok
        EditorMemento current = undoStack.remove(
            undoStack.size() - 1);
        redoStack.add(current);
        return undoStack.get(undoStack.size() - 1);
    }

    public EditorMemento redo() {
        if (redoStack.isEmpty()) return null;
        EditorMemento memento = redoStack.remove(
            redoStack.size() - 1);
        undoStack.add(memento);
        return memento;
    }

    public boolean canUndo() { return undoStack.size() > 1; }
    public boolean canRedo() { return !redoStack.isEmpty(); }
}
```

C4. Composite (Bileşik) Desen -- Belge Yapısı

Hedef: Belgeyi bir öğe ağacı olarak temsil etmek (bölümler, paragraflar, metin).

```
import java.util.*;

// Bileşen (Component)
public interface DocumentElement {
    String render();
    int getCharCount();
}

// Yaprak (Leaf): TextSpan
public class TextSpan implements DocumentElement {
    private String text;
    private boolean bold;
    private boolean italic;

    public TextSpan(String text) {
        this.text = text;
    }

    public void setBold(boolean bold) { this.bold = bold; }
    public void setItalic(boolean italic) {
        this.italic = italic;
    }

    @Override
    public String render() {
        String result = text;
        if (bold) result = "**" + result + "**";
        if (italic) result = "_" + result + "_";
        return result;
    }

    @Override
    public int getCharCount() { return text.length(); }
}

// Bileşik (Composite): Paragraph
public class Paragraph implements DocumentElement {
    private List<DocumentElement> children = new ArrayList<>();

    public void add(DocumentElement element) {
        children.add(element);
    }

    @Override
    public String render() {
        StringBuilder sb = new StringBuilder();
        for (DocumentElement child : children) {
            sb.append(child.render());
        }
        return sb.toString() + "\n";
    }

    @Override
    public int getCharCount() {
        return children.stream()
            .mapToInt(DocumentElement::getCharCount).sum();
    }
}

// Bileşik (Composite): Section
public class Section implements DocumentElement {
    private String title;
    private List<DocumentElement> children = new ArrayList<>();

    public Section(String title) { this.title = title; }

    public void add(DocumentElement element) {
        children.add(element);
    }

    @Override
    public String render() {
        StringBuilder sb = new StringBuilder();
        sb.append("=== ").append(title).append(" ===\n");
        for (DocumentElement child : children) {
            sb.append(child.render());
        }
        return sb.toString();
    }

    @Override
    public int getCharCount() {
        return children.stream()
            .mapToInt(DocumentElement::getCharCount).sum();
    }
}
```

C5. Entegre Metin Düzenleyici (Integrated Text Editor)

```
import java.util.*;

public class TextEditor {
    private TextBuffer buffer;
    private EditorHistory history;
    private List<EditorCommand> commandLog;

    public TextEditor() {
        this.buffer = new TextBuffer();
        this.history = new EditorHistory();
        this.commandLog = new ArrayList<>();
        // Başlangıç durumunu kaydet
        history.save(new EditorMemento(
            buffer.getContent(), buffer.getCursorPosition()));
    }

    public void type(String text) {
        EditorCommand cmd = new InsertCommand(
            buffer, text, buffer.getCursorPosition());
        executeCommand(cmd);
    }

    public void deleteAt(int position, int length) {
        EditorCommand cmd = new DeleteCommand(
            buffer, position, length);
        executeCommand(cmd);
    }

    private void executeCommand(EditorCommand cmd) {
        cmd.execute();
        commandLog.add(cmd);
        history.save(new EditorMemento(
            buffer.getContent(), buffer.getCursorPosition()));
        System.out.println("Executed: " + cmd.getDescription());
    }

    public void undo() {
        EditorMemento memento = history.undo();
        if (memento != null) {
            buffer = new TextBuffer();
            buffer.insert(memento.getContent(), 0);
            buffer.setCursorPosition(memento.getCursorPosition());
            System.out.println("Undo performed.");
        } else {
            System.out.println("Nothing to undo.");
        }
    }

    public void redo() {
        EditorMemento memento = history.redo();
        if (memento != null) {
            buffer = new TextBuffer();
            buffer.insert(memento.getContent(), 0);
            buffer.setCursorPosition(memento.getCursorPosition());
            System.out.println("Redo performed.");
        } else {
            System.out.println("Nothing to redo.");
        }
    }

    public String getContent() { return buffer.getContent(); }
    public int getCursorPosition() { return buffer.getCursorPosition(); }
}
```

C5. Metin Düzenleyiciyi Çalıştırma

```
public class TextEditorDemo {
    public static void main(String[] args) {
        TextEditor editor = new TextEditor();

        // Metin yaz
        editor.type("Hello");
        System.out.println("Content: '" + editor.getContent()
            + "'\n");

        editor.type(" World");
        System.out.println("Content: '" + editor.getContent()
            + "'\n");

        editor.type("!");
        System.out.println("Content: '" + editor.getContent()
            + "'\n");

        // Son işlemi geri al
        editor.undo();
        System.out.println("Content: '" + editor.getContent()
            + "'\n");

        // Tekrar geri al
        editor.undo();
        System.out.println("Content: '" + editor.getContent()
            + "'\n");

        // Yinele (Redo)
        editor.redo();
        System.out.println("Content: '" + editor.getContent()
            + "'\n");

        // Metin sil
        editor.deleteAt(0, 5); // "Hello" sil
        System.out.println("Content: '" + editor.getContent()
            + "'\n");

        // Silmeyi geri al
        editor.undo();
        System.out.println("Content: '" + editor.getContent()
            + "'");
    }
}
```

C5. Metin Düzenleyici Çıktısı

```
Executed: Insert 'Hello' at position 0
Content: 'Hello'

Executed: Insert ' World' at position 5
Content: 'Hello World'

Executed: Insert '!' at position 11
Content: 'Hello World!'

Undo performed.
Content: 'Hello World'

Undo performed.
Content: 'Hello'

Redo performed.
Content: 'Hello World'

Executed: Delete 5 chars at position 0
Content: ' World'

Undo performed.
Content: 'Hello World'
```

```
public class DocumentDemo {
    public static void main(String[] args) {
        // Composite deseni kullanarak bir belge oluştur
        Section intro = new Section("Introduction");
        Paragraph p1 = new Paragraph();
        p1.add(new TextSpan("This is a "));
        TextSpan boldText = new TextSpan("text editor");
        boldText.setBold(true);
        p1.add(boldText);
        p1.add(new TextSpan(" case study."));
        intro.add(p1);

        Section body = new Section("Implementation");
        Paragraph p2 = new Paragraph();
        p2.add(new TextSpan("We use "));
        TextSpan italicText = new TextSpan("Command");
        italicText.setItalic(true);
        p2.add(italicText);
        p2.add(new TextSpan(" and "));
        TextSpan boldItalic = new TextSpan("Memento");
        boldItalic.setBold(true);
        boldItalic.setItalic(true);
        p2.add(boldItalic);
        p2.add(new TextSpan(" patterns."));
        body.add(p2);

        // Belgeyi oluştur (render)
        System.out.println(intro.render());
        System.out.println(body.render());
        System.out.println("Intro chars: "
            + intro.getCharCount());
        System.out.println("Body chars: "
            + body.getCharCount());
    }
}
```

Çıktı:

```
=== Introduction ===
This is a **text editor** case study.

=== Implementation ===
We use  Command  and  **Memento**  patterns.
```

Modül C -- Özet

Metin Düzenleyici Vaka Çalışmasından Temel Dersler

1. **Command (Komut) Deseni** işlemleri nesnelere dönüştürür -- günlükleme, kuyruklama ve geri almayı mümkün kılar.
2. **Memento (Hatıra) Deseni** kapsüllemeyi (encapsulation) bozmadan durum anlık görüntüleri yakalar.
3. **Composite (Bileşik) Desen** bireysel öğelerin ve bileşimlerin tek tip işlenmesine olanak tanır.
4. **Desenler birbirini tamamlar** -- Command ne olduğunu izler, Memento işlerin nasıl olduğunu hatırlar.
5. **Adım adım yeniden yapılandırma** sıfırdan yeniden yazmaktan daha güvenlidir.

Modül D: Vaka Çalışması 3

Bildirim Sistemi Yeniden Yapılandırması

Modül D Taslağı

Bildirim Sistemi Yeniden Yapılandırması (Notification System Refactoring)

Bu vaka çalışması, kod kokularını (code smells) tespit etmeye ve yeniden yapılandırma tekniklerini uygulamaya odaklanır.

1. İlk Monolitik Tasarım
2. Kod Kokularını Tespit Etme
3. Extract Method (Metod Çıkarma) Yeniden Yapılandırması
4. Replace Conditional with Polymorphism (Koşullu İfadeyi Çok Biçimlilikle Değiştirme)
5. Strategy (Strateji) ve Template Method (Şablon Yöntemi) Desenlerini Uygulama
6. Öncesi/Sonrası Kod Karşılaştırması

D1. İlk Monolitik Tasarım (Kötü Kokan Kod)

```
public class NotificationService {
    public void sendNotification(String type, String recipient,
            String subject, String message, boolean urgent,
            String templateName, Map<String, String> params) {

        String finalMessage = message;

        // Apply template if specified
        if (templateName != null && !templateName.isEmpty()) {
            if (templateName.equals("welcome")) {
                finalMessage = "Dear " + params.get("name")
                    + ",\nWelcome to our platform! " + message;
            } else if (templateName.equals("password_reset")) {
                finalMessage = "Hello " + params.get("name")
                    + ",\nClick here to reset your password: "
                    + params.get("resetLink") + "\n" + message;
            } else if (templateName.equals("order_confirmation")) {
                finalMessage = "Dear " + params.get("name")
                    + ",\nYour order #" + params.get("orderId")
                    + " has been confirmed.\nTotal: $"
                    + params.get("total") + "\n" + message;
            } else if (templateName.equals("shipping_update")) {
                finalMessage = "Dear " + params.get("name")
                    + ",\nYour order #" + params.get("orderId")
                    + " has been shipped.\nTracking: "
                    + params.get("trackingNumber") + "\n"
                    + message;
            }
        }

        // Send based on type
        if (type.equals("email")) {
            if (urgent) {
                System.out.println("[URGENT EMAIL] To: " + recipient);
                System.out.println("Subject: !!! " + subject + " !!!");
            } else {
                System.out.println("[EMAIL] To: " + recipient);
                System.out.println("Subject: " + subject);
            }
            System.out.println("Body: " + finalMessage);
            // Connect to SMTP server, authenticate, send...
            System.out.println("Email sent successfully.");

        } else if (type.equals("sms")) {
            String smsMessage = finalMessage;
            if (smsMessage.length() > 160) {
                smsMessage = smsMessage.substring(0, 157) + "...";
            }
            if (urgent) {
                smsMessage = "URGENT: " + smsMessage;
            }
            System.out.println("[SMS] To: " + recipient);
            System.out.println("Message: " + smsMessage);
            // Connect to SMS gateway, send...
            System.out.println("SMS sent successfully.");

        } else if (type.equals("push")) {
            if (urgent) {
                System.out.println("[PUSH - HIGH PRIORITY] To: "
                    + recipient);
            } else {
                System.out.println("[PUSH] To: " + recipient);
            }
            System.out.println("Title: " + subject);
            String pushBody = finalMessage.length() > 100
                ? finalMessage.substring(0, 97) + "..."
                : finalMessage;
            System.out.println("Body: " + pushBody);
            // Connect to push service, send...
            System.out.println("Push notification sent.");

        } else if (type.equals("slack")) {
            System.out.println("[SLACK] Channel: " + recipient);
            String slackMsg = "*" + subject + "*\n" + finalMessage;
            if (urgent) {
                slackMsg = ":rotating_light: " + slackMsg;
            }
            System.out.println("Message: " + slackMsg);
            // Connect to Slack API, send...
            System.out.println("Slack message sent.");
        }

        // Log the notification
        System.out.println("[LOG] " + type + " notification sent to "
            + recipient + " at " + new java.util.Date());
    }
}
```

D2. Kod Kokularını Tespit Etme (Identifying Code Smells)

`NotificationService` İçinde Bulunan Kokular

Koku (Smell)	Açıklama	Konum
Long Method (Uzun Metod)	`sendNotification()` 70 satırın üzerinde	Tüm metod
Feature Envy (Özellik Kıskançlığı)	Şablon işleme bildirim servisine ait değil	Şablon bölümü
Switch Statements (Koşullu İfadeler)	`type` ve `templateName` tabanlı çoklu if-else zincirleri	Her yerde
Divergent Change (Iraksak	Yeni bildirim tipi eklemek bu sınıfı	Tip tabanlı

D2. Yeniden Yapılandırma Planı

Adım Adım Yaklaşım

1. **Extract Method (Metod Çıkarma)** -- Uzun metodu daha küçük metodlara böl
2. **Replace Conditional with Polymorphism (Koşullu İfadeyi Çok Biçimlilikle Değiştir)** -- Tip tabanlı if-else'i ortadan kaldır
3. **Introduce Parameter Object (Parametre Nesnesi Tanıt)** -- Parametre sayısını azalt
4. **Strategy (Strateji) Deseni Uygula** -- Bildirim kanalları için
5. **Template Method (Şablon Yöntemi) Deseni Uygula** -- Mesaj biçimlendirmesi için

Adım adım yeniden yapılandıralım.

D3. Adım 1 -- Parametre Nesnesi Tanıtma (Introduce Parameter Object)

Öncesi: Metod imzasında 7 parametre.

Sonrası: Tek bir `NotificationRequest` nesnesi.

```
import java.util.*;

public class NotificationRequest {
    private final String type;
    private final String recipient;
    private final String subject;
    private final String message;
    private final boolean urgent;
    private final String templateName;
    private final Map<String, String> params;

    public NotificationRequest(String type, String recipient,
            String subject, String message, boolean urgent,
            String templateName, Map<String, String> params) {
        this.type = type;
        this.recipient = recipient;
        this.subject = subject;
        this.message = message;
        this.urgent = urgent;
        this.templateName = templateName;
        this.params = params != null
            ? Collections.unmodifiableMap(params)
            : Collections.emptyMap();
    }

    // Getter'lar
    public String getType() { return type; }
    public String getRecipient() { return recipient; }
    public String getSubject() { return subject; }
    public String getMessage() { return message; }
    public boolean isUrgent() { return urgent; }
    public String getTemplateName() { return templateName; }
    public Map<String, String> getParams() { return params; }
}
```

D3. Adım 2 -- Şablon İşlemeyi Çıkarma (Extract Template Processing)

Şablon mantığını kendi sınıf hiyerarşisine **çıkar**.

```
// Şablon arayüzü
public interface MessageTemplate {
    String apply(String baseMessage,
                 Map<String, String> params);
}

// Somut Şablonlar
public class WelcomeTemplate implements MessageTemplate {
    @Override
    public String apply(String baseMessage,
                        Map<String, String> params) {
        return "Dear " + params.get("name")
            + ",\nWelcome to our platform! " + baseMessage;
    }
}

public class PasswordResetTemplate implements MessageTemplate {
    @Override
    public String apply(String baseMessage,
                        Map<String, String> params) {
        return "Hello " + params.get("name")
            + ",\nClick here to reset your password: "
            + params.get("resetLink") + "\n" + baseMessage;
    }
}

public class OrderConfirmationTemplate
        implements MessageTemplate {
    @Override
    public String apply(String baseMessage,
                        Map<String, String> params) {
        return "Dear " + params.get("name")
            + ",\nYour order #" + params.get("orderId")
            + " has been confirmed.\nTotal: $"
            + params.get("total") + "\n" + baseMessage;
    }
}

// Şablon Kaydı (Template Registry)
public class TemplateRegistry {
    private static final Map<String, MessageTemplate> templates
        = new HashMap<>();

    static {
        templates.put("welcome", new WelcomeTemplate());
        templates.put("password_reset",
            new PasswordResetTemplate());
        templates.put("order_confirmation",
            new OrderConfirmationTemplate());
    }

    public static String applyTemplate(String templateName,
            String baseMessage, Map<String, String> params) {
        if (templateName == null || templateName.isEmpty()) {
            return baseMessage;
        }
        MessageTemplate template = templates.get(templateName);
        if (template == null) {
            return baseMessage;
        }
        return template.apply(baseMessage, params);
    }
}
```

D4. Adım 3 -- Koşullu İfadeyi Çok Biçimlilikle Değiştirme

Bildirim tipleri için if-else zincirini bir sınıf hiyerarşisi ile **değiştir**.

```
// Template Method deseni kullanan soyut bildirim kanalı
public abstract class NotificationChannel {

    // Template Method -- algoritma iskeletini tanımlar
    public final void send(NotificationRequest request) {
        String message = prepareMessage(request);
        if (request.isUrgent()) {
            message = applyUrgencyFormatting(request, message);
        }
        deliverMessage(request, message);
        logNotification(request);
    }

    // Alt sınıfların uyguladığı adımlar
    protected String prepareMessage(NotificationRequest request) {
        return TemplateRegistry.applyTemplate(
            request.getTemplateName(),
            request.getMessage(),
            request.getParams()
        );
    }

    protected abstract String applyUrgencyFormatting(
        NotificationRequest request, String message);

    protected abstract void deliverMessage(
        NotificationRequest request, String message);

    protected void logNotification(NotificationRequest request) {
        System.out.println("[LOG] " + getChannelName()
            + " notification sent to "
            + request.getRecipient()
            + " at " + new java.util.Date());
    }

    protected abstract String getChannelName();
}
```

D4. Somut Bildirim Kanalları (Concrete Notification Channels)

```
public class EmailChannel extends NotificationChannel {
    @Override
    protected String applyUrgencyFormatting(
            NotificationRequest request, String message) {
        return message; // aciliyet teslimat sırasında işlenir
    }

    @Override
    protected void deliverMessage(
            NotificationRequest request, String message) {
        if (request.isUrgent()) {
            System.out.println("[URGENT EMAIL] To: "
                + request.getRecipient());
            System.out.println("Subject: !!! "
                + request.getSubject() + " !!!");
        } else {
            System.out.println("[EMAIL] To: "
                + request.getRecipient());
            System.out.println("Subject: "
                + request.getSubject());
        }
        System.out.println("Body: " + message);
        System.out.println("Email sent successfully.");
    }

    @Override
    protected String getChannelName() { return "Email"; }
}

public class SMSChannel extends NotificationChannel {
    private static final int MAX_LENGTH = 160;

    @Override
    protected String applyUrgencyFormatting(
            NotificationRequest request, String message) {
        return "URGENT: " + message;
    }

    @Override
    protected void deliverMessage(
            NotificationRequest request, String message) {
        if (message.length() > MAX_LENGTH) {
            message = message.substring(0, MAX_LENGTH - 3)
                + "...";
        }
        System.out.println("[SMS] To: "
            + request.getRecipient());
        System.out.println("Message: " + message);
        System.out.println("SMS sent successfully.");
    }

    @Override
    protected String getChannelName() { return "SMS"; }
}
```

D4. Diğer Bildirim Kanalları

```
public class PushChannel extends NotificationChannel {
    private static final int MAX_BODY_LENGTH = 100;

    @Override
    protected String applyUrgencyFormatting(
            NotificationRequest request, String message) {
        return message; // teslimat sırasında işlenir
    }

    @Override
    protected void deliverMessage(
            NotificationRequest request, String message) {
        if (request.isUrgent()) {
            System.out.println("[PUSH - HIGH PRIORITY] To: "
                + request.getRecipient());
        } else {
            System.out.println("[PUSH] To: "
                + request.getRecipient());
        }
        System.out.println("Title: " + request.getSubject());
        String pushBody = message.length() > MAX_BODY_LENGTH
            ? message.substring(0, MAX_BODY_LENGTH - 3) + "..."
            : message;
        System.out.println("Body: " + pushBody);
        System.out.println("Push notification sent.");
    }

    @Override
    protected String getChannelName() { return "Push"; }
}

public class SlackChannel extends NotificationChannel {
    @Override
    protected String applyUrgencyFormatting(
            NotificationRequest request, String message) {
        return ":rotating_light: " + message;
    }

    @Override
    protected void deliverMessage(
            NotificationRequest request, String message) {
        System.out.println("[SLACK] Channel: "
            + request.getRecipient());
        String slackMsg = "*" + request.getSubject()
            + "*\n" + message;
        System.out.println("Message: " + slackMsg);
        System.out.println("Slack message sent.");
    }

    @Override
    protected String getChannelName() { return "Slack"; }
}
```

D5. Yeniden Yapılandırılmış Bildirim Servisi (Refactored Notification Service)

```
import java.util.*;

public class NotificationService {
    private final Map<String, NotificationChannel> channels;

    public NotificationService() {
        channels = new HashMap<>();
        channels.put("email", new EmailChannel());
        channels.put("sms", new SMSChannel());
        channels.put("push", new PushChannel());
        channels.put("slack", new SlackChannel());
    }

    public void send(NotificationRequest request) {
        NotificationChannel channel =
            channels.get(request.getType());
        if (channel == null) {
            throw new IllegalArgumentException(
                "Unknown channel: " + request.getType());
        }
        channel.send(request);
    }

    // Mevcut kodu değiştirmeden yeni kanallar eklemek kolay
    public void registerChannel(String name,
                                NotificationChannel channel) {
        channels.put(name, channel);
    }
}
```

Dikkat: Orijinal 70+ satırlık metod artık basit bir 5 satırlık yönlendirmedir. Her kanal kendi mantığını bağımsız olarak işler.

D5. Yeniden Yapılandırılmış Servisi Kullanma

```
public class NotificationDemo {
    public static void main(String[] args) {
        NotificationService service = new NotificationService();

        // Şablon ile e-posta bildirimi gönder
        Map<String, String> params = new HashMap<>();
        params.put("name", "Alice");
        params.put("orderId", "ORD-456");
        params.put("total", "149.99");

        NotificationRequest emailReq = new NotificationRequest(
            "email", "alice@example.com",
            "Order Confirmed",
            "Thank you for your purchase!",
            false, "order_confirmation", params
        );
        service.send(emailReq);
        System.out.println("---");

        // Acil SMS gönder
        NotificationRequest smsReq = new NotificationRequest(
            "sms", "+1234567890",
            "Alert", "Server is down!",
            true, null, null
        );
        service.send(smsReq);
        System.out.println("---");

        // Slack mesajı gönder
        NotificationRequest slackReq = new NotificationRequest(
            "slack", "#engineering",
            "Deploy Complete",
            "Version 2.5.0 deployed to production.",
            false, null, null
        );
        service.send(slackReq);
    }
}
```

D6. Öncesi/Sonrası Karşılaştırması

Öncesi

Metrik	Değer
Sınıf Sayısı	1
Ana metod satır sayısı	70+

Sonrası

Metrik	Değer
Sınıf Sayısı	10+ (her biri tek sorumluluğa sahip)
Metod başına satır sayısı	5-15

Modül D -- Özet

Bildirim Sistemi Yeniden Yapılandırmasından Temel Dersler

1. **Kod kokuları semptomlardır** -- daha derin tasarım sorunlarına işaret ederler.
2. **Yeniden yapılandırma artımlıdır (incremental)** -- küçük, güvenli adımlar büyük iyileştirmelere yol açar.
3. **Replace Conditional with Polymorphism** en güçlü yeniden yapılandırmalardan biridir.
4. **Template Method** alt sınıflar bir algoritma yapısını paylaşıp belirli adımlarda farklılaştığında iyi çalışır.
5. **Parametre nesneleri** karmaşıklığı azaltır ve okunabilirliği artırır.
6. **Öncesi/Sonrası metrikleri** yeniden yapılandırma çabasını paydaşlara gerekçelendirmeye yardımcı olur.

Modül E: En İyi Uygulamalar ve Anti-Desenler

Desenler Ne Zaman Yardımcı Olur, Ne Zaman Zarar Verir

Modül E Taslağı

En İyi Uygulamalar ve Anti-Desenler (Best Practices & Anti-Patterns)

1. Tasarım Desenlerini Ne Zaman KULLANMAMALI
2. Yaygın Anti-Desenler
3. Basitlik ve Genişletilebilirlik Dengesini Kurma
4. SOLID İlkeleri Bağlamda Yeniden İnceleme
5. Pratik Yönergeler

E1. Tasarım Desenlerini Ne Zaman KULLANMAMALI

Aşırı Mühendislik (Over-Engineering) Uyarı İşaretleri

Tasarım desenleri araçtır, amaç değildir. İhtiyaç olmadığında desen kullanmak gereksiz karmaşıklık yaratır.

Şu durumlarda desen KULLANMAYIN:

- **Problem basitse** -- 2-3 durum için basit bir `if-else` yeterlidir.
- **Kodun değişeceğinden emin değilseniz** -- YAGNI (You Aren't Gonna Need It / Buna İhtiyacınız Olmayacak).
- **Desen tasarruf ettiğinden fazla kod ekliyorsa** -- Soyutlama problemden daha karmaşıksa, atlayın.
- **Ekibiniz deseni anlamıyorsa** -- Okunabilirlik zekilikten daha önemlidir.

```
// AŞIRI MÜHENDİSLİK: 2 basit tip için Factory
public class AnimalFactory {
    public static Animal create(String type) {
        if (type.equals("dog")) return new Dog();
        if (type.equals("cat")) return new Cat();
        throw new IllegalArgumentException(type);
    }
}

// DAHA BASİT: Nesneleri doğrudan oluşturun
Dog dog = new Dog();
Cat cat = new Cat();
```

E2. Anti-Desen: God Object (Tanrı Nesnesi)

Nedir: Çok fazla bilen ve çok fazla yapan tek bir sınıf.

Belirtiler: Yüzlerce veya binlerce satır, her yerden import, test etmesi zor.

```
// KÖTÜ: God Object (Tanrı Nesnesi)
public class ApplicationManager {
    public void handleUserLogin(String u, String p) { /* ... */ }
    public void processOrder(Order o) { /* ... */ }
    public void generateReport(String type) { /* ... */ }
    public void sendEmail(String to, String body) { /* ... */ }
    public void updateInventory(Item item) { /* ... */ }
    public void calculateTaxes(Order o) { /* ... */ }
    public void backupDatabase() { /* ... */ }
    public void processPayroll() { /* ... */ }
    // Uygulamanın her yönünü kapsayan 50 metod daha...
}

// İYİ: Sorumlulukları odaklanmış sınıflara ayır
public class AuthenticationService {
    public void login(String u, String p) { /* ... */ }
}
public class OrderService {
    public void processOrder(Order o) { /* ... */ }
}
public class ReportService {
    public void generateReport(String type) { /* ... */ }
}
public class NotificationService {
    public void sendEmail(String to, String body) { /* ... */ }
}
```

E3. Anti-Desen: Singleton Kötüye Kullanımı (Singleton Abuse)

Nedir: "Sadece bir örneğe ihtiyacımız var" diye her şey için Singleton kullanmak.

Sorunlar: Gizli bağımlılıklar, test etmesi zor, global değiştirilebilir durum.

```
// KÖTÜ: Global değişken olarak kullanılan Singleton
public class DatabaseConnection {
    private static DatabaseConnection instance;
    private Connection connection;

    private DatabaseConnection() {
        // veritabanına bağlan
    }

    public static DatabaseConnection getInstance() {
        if (instance == null) {
            instance = new DatabaseConnection();
        }
        return instance;
    }

    public void query(String sql) { /* ... */ }
}

// Her sınıf singleton'a bağımlı
public class OrderService {
    public void createOrder(Order order) {
        // Gizli bağımlılık -- test etmesi zor!
        DatabaseConnection.getInstance()
            .query("INSERT INTO orders ...");
    }
}

// İYİ: Bunun yerine bağımlılık enjeksiyonu kullan
public class OrderService {
    private final DatabaseConnection db;

    public OrderService(DatabaseConnection db) {
        this.db = db; // Açık bağımlılık, test edilebilir
    }

    public void createOrder(Order order) {
        db.query("INSERT INTO orders ...");
    }
}
```

E4. Anti-Desen: Desen Takıntısı (Pattern Obsession)

Nedir: Daha basit çözümler varken bile her yerde tasarım deseni uygulamak.

Sonuç: Her basit işlem soyutlama katmanlarına sarıldığı için takip etmesi zor kod.

```
// AŞIRI DESENLENMİŞ: Basit string biçimlendirme
// Strategy + Factory + Builder desenleri kullanarak
public interface FormatterStrategy {
    String format(String input);
}
public class UpperCaseFormatter implements FormatterStrategy {
    public String format(String input) {
        return input.toUpperCase();
    }
}
public class FormatterFactory {
    public static FormatterStrategy create(String type) {
        if ("upper".equals(type))
            return new UpperCaseFormatter();
        throw new IllegalArgumentException(type);
    }
}
// Kullanım: Önemsiz bir şey için 4 sınıf
String result = FormatterFactory.create("upper")
    .format("hello");

// BASİT: Yerleşik metodu kullanın
String result = "hello".toUpperCase();
```

Temel kural: Çözümü tek bir cümleyle açıklayabiliyorsanız, muhtemelen bir desene ihtiyacınız yoktur.

E5. Basitlik ve Genişletilebilirlik Dengesini Kurma

Karmaşıklık Spektrumu

```
Basit Kod   <-------|-------->  Genişletilebilir Kod
(Şimdi kolay)     (Tatlı Nokta)      (Sonra kolay)
```

Tatlı noktayı bulmak için yönergeler:

Soru	EVET ise	HAYIR ise
Bu kod sık değişecek mi?	Soyutlama ekle	Basit tut
3 veya daha fazla varyasyon var mı?	Desen kullan	if-else kullan
Ekip deseni tanıyor mu?	Kullan	Belgele veya basitleştir

E6. SOLID İlkeleri Yeniden İnceleme

SOLID Desen Kullanımına Nasıl Rehberlik Eder

İlke	Açıklama	Desen Bağlantısı
S - Single Responsibility (Tek Sorumluluk)	Bir sınıfın değişmek için tek bir nedeni olmalı	God Object'i önler
O - Open/Closed (Açık/Kapalı)	Genişletmeye açık, değiştirmeye kapalı	Strategy, Observer, Decorator
L - Liskov Substitution (Liskov Yerine Geçme)	Alt tipler temel tipler yerine kullanılabilir olmalı	Template Method, State
I - Interface Segregation (Arayüz Ayrımı)	Küçük, odaklanmış arayüzleri tercih et	Adapter, spesifik arayüzler
D - Dependency Inversion (Bağımlılık Tersine Çevirme)	Soyutlamalara bağımlı ol, somutlamalara değil	Factory, Dependency Injection

Tek Sorumluluk (Single Responsibility)

```
// Vaka Çalışması 1: Her desen bir endişeyi ele alır
PaymentStrategy    -> Yalnızca ödeme işleme
OrderState         -> Yalnızca durum geçişleri
OrderObserver      -> Yalnızca bildirimler
ShippingMethod     -> Yalnızca kargo lojistiği
```

Açık/Kapalı İlkesi (Open/Closed Principle)

```
// Vaka Çalışması 3: Yeni bildirim kanalı ekleme
// mevcut kodu DEĞİŞTİRMEZ
public class TeamsChannel extends NotificationChannel {
    @Override
    protected String applyUrgencyFormatting(
            NotificationRequest request, String message) {
        return "!! " + message + " !!";
    }

    @Override
    protected void deliverMessage(
            NotificationRequest request, String message) {
        System.out.println("[TEAMS] To: "
            + request.getRecipient());
        System.out.println("Message: " + message);
    }

    @Override
    protected String getChannelName() { return "Teams"; }
}
```

E7. Pratik Yönergeler

Desen Uygulama Karar Çerçevesi

1. **Basit başla** -- Çalışan en basit kodu yaz.
2. **Kokuyu bekle** -- Kod kokularının ne zaman yeniden yapılandırma yapılacağını söylemesine izin ver.
3. **Desenlere doğru yeniden yapılandır** -- Desenleri önceden değil, belirli sorunları çözmek için uygula.
4. **İyileştirmeyi ölç** -- Kod satır sayısı, test edilebilirlik, bağımlılık.
5. **Kararlarını belgele** -- Gelecekteki geliştiricilerin bir desenin *neden* seçildiğini anlaması gerekir.

Üç Vuruş Kuralı (Three Strikes Rule)

> *Bir şeyi ilk yaptığında, sadece yap. İkinci kez benzer bir şey yaptığında, yüzünü buruştur ama yine de yap. Üçüncü kez, yeniden yapılandır.*
> -- Martin Fowler

Modül E -- Özet

En İyi Uygulamalar Özeti

1. **Desenler araçtır, amaç değildir** -- Onları sorunları çözmek için kullanın, etkilemek için değil.
2. **Anti-desenlerden kaçının** -- God Object, Singleton kötüye kullanımı ve Desen takıntısı çözdüklerinden daha fazla sorun yaratır.
3. **Denge anahtardır** -- Çok az tasarım spagetti koda yol açar; çok fazlası lazanya koda (çok fazla katman) yol açar.
4. **SOLID ilkeleri kararları yönlendirir** -- Bir desenin ne zaman uygun olduğunu bilmenize yardımcı olur.
5. **Artımlı olarak yeniden yapılandırın** -- Küçük, güvenli adımlar büyük yeniden yazmalardan daha iyidir.
6. **Kod insanlar içindir** -- Okunabilirlik ve bakım kolaylığı her zaman zekilikten üstündür.

Hafta-14 Özeti

Neler Öğrendik

Modül	Temel Çıkarım
A	Tasarım desenleri Java çerçevelerinde yaygın olarak kullanılır
B	Birden fazla desen karmaşık sorunları çözmek için birlikte çalışır (E-Ticaret)
C	Command + Memento geri al/yineleyi sağlar; Composite hiyerarşileri modeller (Metin Düzenleyici)
D	Yeniden yapılandırma kötü kokan kodu temiz, genişletilebilir tasarımlara dönüştürür (Bildirimler)
E	Desenler SOLID ilkeleri rehberliğinde dikkatli uygulanmalıdır

Kaynaklar

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
- Freeman, E., Robson, E. (2020). *Head First Design Patterns*. O'Reilly Media.
- Fowler, M. (2018). *Refactoring: Improving the Design of Existing Code* (2. Baskı). Addison-Wesley.
- Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.
- Bloch, J. (2018). *Effective Java* (3. Baskı). Addison-Wesley.

Kaynaklar (devam)

- RefactoringGuru. *Design Patterns*. https://refactoring.guru/design-patterns
- RefactoringGuru. *Refactoring Techniques*. https://refactoring.guru/refactoring/techniques
- Oracle. *Java Platform SE Documentation*. https://docs.oracle.com/javase/
- Kerievsky, J. (2004). *Refactoring to Patterns*. Addison-Wesley Professional.
- Martin, R. C. (2003). *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall.

$$End - Of - Week - 14 - Module$$