Behavioral - Chain of Responsibility Design Pattern (2. Örnek mutlaka bakılmalı)

Birbirini takip eden iş dizisine ait process'leri redirect ve handle etmek, yada istekte bulunan-confirm eden süreçleri için çözüm olarak ortaya çıkmış bir tasarım desendir. Yani iş client tarafından receiver'lara request edilir, receiver çözemeyeceği bir şey ise zincirin 2. Elemanına request'i gönderir bu böyle çözüm buluncaya kadar devam eder. Ancak pratikte CoR deseni, zincirdeki eleman sayısının 10'u aşmadığı durumlar için uygundur

DoFactory açıklaması : **Chain of Responsibility** deseni, ortak bir **mesaj** veya **talebin(Request),** birbirlerine **zayıf bir şekilde bağlanmış(Loosly Coupled)** nesneler arasında gezdirilmesi ve bu zincir içerisinde asıl sorumlu olanı tarafından ele alınması gerektiği vakalarda kullanılmaktadır.

Bir örnek ile anlatmak gerekirse;

1. Müşteri 480 bin TL lik para çekme isteğini veznedar'a iletir.
2. Veznedar bu isteği alır ve kontrol eder eğer onaylayabileceği bir tutar ise onaylar, onaylayabileceği bir tutar değilse yöneticisine gönderir,
3. Yönetici isteği alır onaylayabileceği bir tutar değilse müdüre iletir,
4. Müdür kontrol eder eğer onaylayabileceği bir tutar değilse bölge sorumlusunun onayına gönderir,
5. Bölge sorumlusu onaylar ve para müşteriye verilir.

Bir başka örnek;
Otomatik ürün makinelerine ait jeton slotları verilmektedir. Her tip jeton için bir slot oluşturmak aslında arka arkaya if blokları yazarak, gelen talebin anlaşılmaya çalışılmasına benzetilebilir. Bunun yerine makine üzerinde, her bir jetonu ele alan tek bir slot tasarlanır**(Handler).** Ürünü satın almak isteyen kişinin attığı jeton, verdiği komuta(Cola, çikolata vs istemek gibi) ve jetonun tipine göre, içeride uygun olan saklama alanına**(ConcreteHandler)** düşecektir. Sonrasında ise süreç, jetonun uygun olan saklama alanında değerlendirilerek istenilen ürünün teslim edilmesiyle tamamlanacaktır

Örnek ; Bir service'in hangi lokasyonda çalıştığı bilgisi zincir içerisinden bulunacak

```
enum ServiceLocation{
   LocalMachine,
   Intranet,
   Internet,
   SecureZone
}

//Chain içerisinde ki nesnelerde dolaşacak olan class
class ServiceInfo{
   private String name;
   private ServiceLocation serviceLocation;
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public ServiceLocation getServiceLocation() {
      return serviceLocation;
   }
   public void setServiceLocation(ServiceLocation serviceLocation) {
      this.serviceLocation = serviceLocation;
   }
}

//Handler
abstract class ServiceHandler{
   protected ServiceHandler successor;
   public void setSuccessor(ServiceHandler successor) {
      this.successor = successor;
   }
   public abstract void processRequest(ServiceInfo serviceInfo);
}
```

```
//Concrete Handler
// Servisin yerel makineye ait olma durumunu ele alır.
class LocalMachineHandler extends ServiceHandler{
    @Override
    public void processRequest(ServiceInfo serviceInfo) {
        if (serviceInfo.getServiceLocation() == ServiceLocation.LocalMachine){
            System.out.println("Yerel makinada yer alan bir servis");
        } else if(successor!=null){
            successor.processRequest(serviceInfo);
        }
    }
}
//Concrete Handler
// Servisin Intranet üzerinde olma durumunu ele alır.
class IntranetHandler extends ServiceHandler{
    @Override
    public void processRequest(ServiceInfo serviceInfo) {
        if (serviceInfo.getServiceLocation()==ServiceLocation.Intranet){
            System.out.println("Intranet üzerinde çalışan bir servis");
        } else if(successor!=null){
            successor.processRequest(serviceInfo);
        }
    }
}
//Concrete Handler
// Servisin Internet üzerinde olma durumunu ele alır.
class InternetHandler extends ServiceHandler{
    @Override
    public void processRequest(ServiceInfo serviceInfo) {
        if (serviceInfo.getServiceLocation()==ServiceLocation.Internet){
            System.out.println("Internet üzerinde çalışan bir servis");
        } else if(successor!=null){
            successor.processRequest(serviceInfo);
        }
    }
}
//Concrete Handler
// Servisin Secure Zone üzerinde olma durumunu ele alır.
//next successor son servis olduğu için yoktur
class SecureZoneHandler extends ServiceHandler{
    @Override
    public void processRequest(ServiceInfo serviceInfo) {
        if (serviceInfo.getServiceLocation()==ServiceLocation.SecureZone){
            System.out.println("Secure zone üzerinde çalışan bir servis");
        } else {
            System.out.println("servis bulunamadı");
        }
    }
}
```

Main

```
public static void main(String[] args) {
    ServiceHandler localMachineHandler = new LocalMachineHandler();
    ServiceHandler intranetHandler = new IntranetHandler();
    ServiceHandler internetHandler = new InternetHandler();
    ServiceHandler secureZoneHandler = new SecureZoneHandler();
    localMachineHandler.setSuccessor(intranetHandler);
    intranetHandler.setSuccessor(internetHandler);
    internetHandler.setSuccessor(secureZoneHandler);
    ServiceInfo info = new ServiceInfo();
    info.setName("Order Process Service");
    info.setServiceLocation(ServiceLocation.Internet);
    localMachineHandler.processRequest(info);
}
```

Behavioral - Chain of Responsibility Design Pattern (2. Örnek)

Özellikle loglama,hata yönetimi vb. işlerde çeşitli kurallara bağlı kod yönlendirme işlemlerini gerçekleştirmek amacıyla kullanılan tasarım desenlerinden biridir

```
//harcama
class Expense{
    protected String detail;
    protected Double amount;
    public Expense(String newDetail,Double newAmount){
        detail = newDetail;
        amount = newAmount;
    }
}
```

Bu harcamayı yönetecek abstract class;

```
abstract class ExpenseHandlerBase{
    protected ExpenseHandlerBase successor;
    public abstract void HandleExpense(Expense expense);//harcamayı yönetme süreci

    public void setSuccessor(ExpenseHandlerBase newSuccessor){
        successor = newSuccessor;
    }
}
```

Expense base class'ının concrete'leri

```
//yönetici
class Manager extends ExpenseHandlerBase{
    @Override
    public void HandleExpense(Expense expense) {
        if (expense.amount<=100){
            System.out.println("Manager handled the expense"); //manager bu ödemeyi yetkisi dahilinde yapabilir
        } else if(successor!=null){
            //manager'in yetkisini aşan bir miktar gelirse bir üstüne yani successor'une gönderdik
            //manager'in bir üstü yani successor'ü varsa ödemeyi ona yönlendir demiş olduk
            successor.HandleExpense(expense);
        }
    }
}
//başkan yardımcısı
class VicePresident extends ExpenseHandlerBase{
    @Override
    public void HandleExpense(Expense expense) {
        if (expense.amount>100 && expense.amount<=1000){
            System.out.println("Vice president handled the expense");
        } else if(successor!=null){
            successor.HandleExpense(expense);
        }
    }
}
//başkan
class President extends ExpenseHandlerBase{
    @Override
    public void HandleExpense(Expense expense) {
        if (expense.amount>1000){
            System.out.println("President handled the expense"); //manager bu ödemeyi yetkisi dahilinde yapabilir
        }
    }
}
```

Main

```
public static void main(String[] args) {
    Manager manager = new Manager();
    VicePresident vicePresident = new VicePresident();
    President president = new President();
    manager.setSuccessor(vicePresident);
    vicePresident.setSuccessor(president);
    Expense expense = new Expense("Training",2000.5);
    manager.HandleExpense(expense);
}
```