Behavioral - Command Design Pattern

Komutların wrap edilip yani sarmalanıp tek bir nesne haline getirilmesini amaçlar. Herhangi bir methodu direk çağırıyor olmak bağımlılığı artıracaktır, bu bağımlılığı decoupling hale getirmek için kullanılır.Uzaktaki (RPC) bir methodu çalıştırmak içinde tavsiye edilmektedir.Client methodu çağırmak istediğin araya koyduğumuz aracıya gider ve aracı da methodu çalıştırır.Böylece client'i değiştirirsek method bundan etkilenmez,methodu değiştirirsek client bundan etkilenmez.Dolayısıyla decoupling sağlanmış olur.Aracı işini görecek olan nesneye **Invoker** adı veriliyor. "İşlemleri geri almak" konusuna nihayet gelebildik. COMMAND deseninin en yaygın kullanıldığı kısım bu başlıktır. Undo/Redo yapabilmek aslında işlemden etkilenen nesnelerin durumunu ileri/geri hareket ettirebilmek olarak düşünülebilir. Yani bir nesnenin durumunda (state) ileri geri gitmek istersek bize en uygun tasarım deseni; MEMENTO olacaktır. Ancak durum (state) temelli değil de süreci etkileyen işlemler penceresinden baktığımızda ise COMMAND deseni işimize çok yarayacaktır.

- Komut (Command) : Gerçekleştirilecek işlem için bir ara yüz tanımlar.
- Somut Komut (Concrete Command): Alıcı ve gerçekleştirilecek işlemler arasında bir bağ kurar, alıcıda karşılık düşen işlemleri çağırarak çalıştırma eylemini gerçekleştirir.
- İstemci (Client): Komut nesnesini oluşturur ve metodun sonraki zamanlarda çağrılabilmesi için gerekli bilgiyi sağlar.
- Çağırıcı (Invoker): Metodun ne zaman çağrılacağını belirtir.
- Alıcı (Receiver): Kullanıcı isteklerini gerçekleştirecek asıl metod kodlarını içerir.

Tüm electronic cihazların base class'ı

```
interface ElectronicDevice{
   void on();
   void off();
   void volumeUp();
   void volumeDown();
}
```

Command interface

```
//Command
interface Command{
   void execute();
   //void undo(); //kullanılabilir
}
```

Invoker

```
//Invoker
class RemoteControl{
   Command command;
   public RemoteControl(Command command){
      this.command = command;
   }

   public void pressButton(){
      command.execute();
   }
}

//Örnek kullanım
class TurnItAllOff implements Command{
   List<ElectronicDevice> electronicDevices;
   public TurnItAllOff(List<ElectronicDevice> electronicDevices) {
      this.electronicDevices = electronicDevices;
   }

   @Override
   public void execute() {
      for (ElectronicDevice device : electronicDevices){
         device.off();
      }
   }
}
```

Receiver class'lar

```
//Receiver
class Television implements ElectronicDevice{
    private int volume = 0;
    @Override
    public void on() {
        System.out.println("TV is ON");
    }

    @Override
    public void off() {
        System.out.println("TV is OFF");
    }

    @Override
    public void volumeUp() {
        volume++;
        System.out.println("TV volume is at : " +volume);
    }

    @Override
    public void volumeDown() {
        volume--;
        System.out.println("TV volume is at : " +volume);
    }
}

//Receiver
class Radio implements ElectronicDevice{
    private int volume = 0;
    @Override
    public void on() {
        System.out.println("Radio is ON");
    }

    @Override
    public void off() {
        System.out.println("Radio is OFF");
    }

    @Override
    public void volumeUp() {
        volume++;
        System.out.println("Radio volume is at : " +volume);
    }

    @Override
    public void volumeDown() {
        volume--;
        System.out.println("Radio volume is at : " +volume);
    }
}
```

Concrete Command

```
//Concrete Command
class TurnTvOn implements Command{
    private ElectronicDevice device;
    public TurnTvOn(ElectronicDevice device){
        this.device = device;
    }
    @Override
    public void execute() {
        device.on();
    }
}
class TurnTvOff implements Command{
    private ElectronicDevice device;
    public TurnTvOff(ElectronicDevice device){
        this.device = device;
    }
    @Override
    public void execute() {
        device.off();
    }
}
class TvVolumeUp implements Command{
    private ElectronicDevice device;
    public TvVolumeUp(ElectronicDevice device){
        this.device = device;
    }
    @Override
    public void execute() {
        device.volumeUp();
    }
}
class TvVolumeDown implements Command{
    private ElectronicDevice device;
    public TvVolumeDown(ElectronicDevice device){
        this.device = device;
    }
    @Override
    public void execute() {
        device.volumeDown();
    }
}
```

Radio içinde Concrete Command class'ı yaratılmalı. Yer kaplamasın diye yazmadım

```
public static void main(String[] args) {
    ElectronicDevice device = TVRemote.getDevice();
    TurnTvOn tvOn = new TurnTvOn(device);
    RemoteControl remoteControl = new RemoteControl(tvOn);
    remoteControl.pressButton();

    Television television = new Television();
    Radio radio = new Radio();
    List<ElectronicDevice> electronicDevices = new ArrayList<>();
    electronicDevices.add(television);
    electronicDevices.add(radio);
    TurnItAllOff turnOffAllDevices = new TurnItAllOff(electronicDevices);
    RemoteControl allDeviceOff = new RemoteControl(turnOffAllDevices);
    allDeviceOff.pressButton();
}
```