Komut(Command) Tasarım Deseni

Nedir?

Komut(Command) tasarım deseni, **behavioral** tasarım desenlerinden biridir. **Request-response** model arasında **lose coupling** ilişki sağlar. Bir isteği nesneye dönüştürerek, isteğin kullanıcı sınıfları tarafından rahatça erişilebilmesi sağlanır.

Ne zaman Kullanılır?

Diyelim ki 4 tane buton var ve her bir buton farklı işlemler yapıyor. Hangi butona tıklandığını **if-else** ile kontrol edip ilgili işlemin çalıştırılması yerine veya her buton tıklama olayı için ayrı 4 farklı metod yazmak yerine komut tasarım deseni kullanarak, yapılacak işlemler için(örneğin Açma, Kapama, Kaydetme gibi) sınıflar oluştururuz. Daha sonra, bu butonların onclick event'ları içerisinde bu sınıflardan nesne yaratılıp **execute()** metodu çağırabiliriz. Bu sayede karmaşık **if-else** yapısından da kurtulmuş oluruz.

Nasıl Kullanılır?

Her komut için bir sınıf yaratılır. Bu sınıfların ortak türe sahip olabilmesi için **Command** isminde bir **interface** tanımlanır. **Command interface** içerisinde tek bir tane metod gereklidir: **execute()** . Her komut için oluşturulan sınıflar **execute()** metodunu implement ederler. **execute()** metodu asıl işlevi yapacak olan sınıfın metodunu çağırır. **Asıl** işlevi yapan sınıf **Receiver** sınıfıdır. Daha sonra Komut sınıflarının çağrılması için **Invoker** sınıfı oluşturulur. Fakat **Invoker** sınıfı hangi komut nesnesini kullanacağını bilmez. Sadece nasıl kullanacağını bilir. Son olarak **Client** ise **Invoker** sınıfını çağırır.

Faydaları Nedir?

1. Uygulama daha modüler ve esnek olur.
2. Geriye alma işlemi(**undo**) ve anlık durumun **restorasyonu** gibi işlemler yapılabilir.
3. **Loglama** ve **transactional** işlemler için kullanılabilir.
4. Makro komutlar oluşturarak bu komutların bir anda çalıştırılması sağlanabilir.

Gerekenler

Türü **interface** olan **Command** sınıfı
Tek bir tane **somut Command** sınıfı
Receiver sınıfı
Invoker sınıfı
Test sınıfı

Örnek Kullanım Alanları

1. **java.lang.Runnable** interface'nin tüm implementasyonları
2. **javax.swing.Action** interface'nin tüm implementasyonları

Örnek Uygulama

Command interface

```
1  public interface Command {
2      public void execute();
3  }
```

LightOnCommand Sınıfı

```
 1  /**
 2   * Somut Command sinifimiz
 3   */
 4  public class LightOnCommand implements Command {
 5      Light light;
 6
 7      public LightOnCommand(Light light) {
 8          this.light = light;
 9      }
10
11      public void execute() {
12          light.on();
13      }
14  }
```

LightOffCommand Sınıfı

```
 1  /**
 2   * Somut Command sinifimiz
 3   */
 4  public class LightOffCommand implements Command {
 5      Light light;
 6
 7      public LightOffCommand(Light light) {
 8          this.light = light;
 9      }
10
11      public void execute() {
12          light.off();
13      }
14  }
```

SimpleRemoteControl Sınıfı

```
1  /**
2   * Invoker sinifimiz
3   */
4  public class SimpleRemoteControl {
```

```
    Command slot;

    public SimpleRemoteControl() {}

    public void setCommand(Command command) {
        slot = command;
    }

    public void buttonWasPressed() {
        slot.execute();
    }
}
```

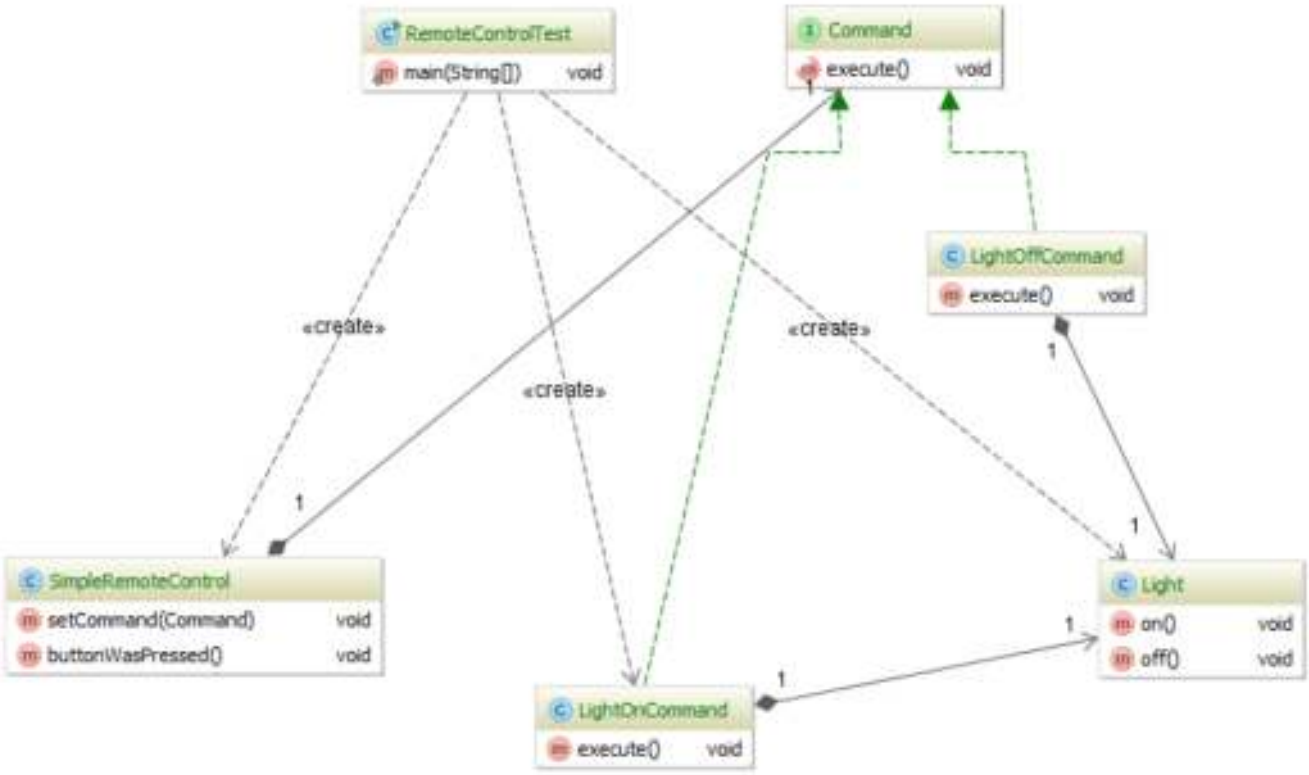
Light Sınıfı

```
/**
 * Receiver sınıfımız
 */
public class Light {

    public Light() {
    }

    public void on() {
        System.out.println("Light is on");
    }

    public void off() {
        System.out.println("Light is off");
    }
}
```

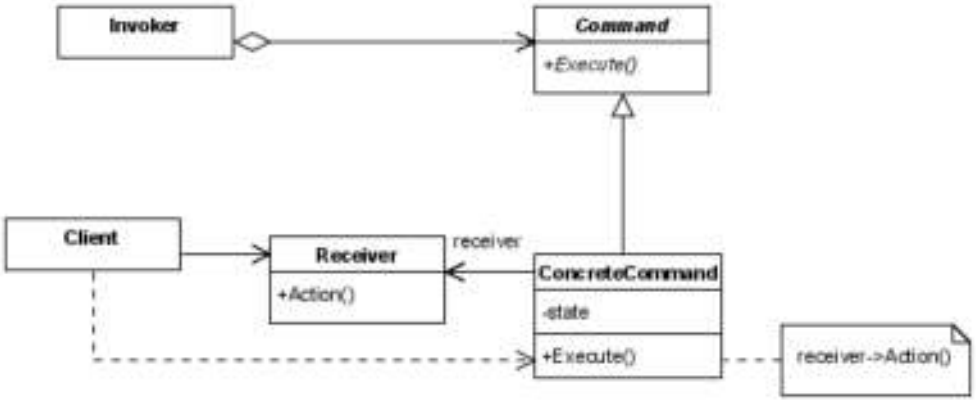
Test Sınıfı

```
/**
 * Client sınıfımız
 */
public class RemoteControlTest {
    public static void main(String[] args) {
        SimpleRemoteControl remote = new SimpleRemoteControl();
        Light light = new Light();
        LightOnCommand lightOn = new LightOnCommand(light);
        remote.setCommand(lightOn);
        remote.buttonWasPressed();
        remote.buttonWasPressed();
    }
}
```

Uygulamamızın UML Diagramı



Komut Tasarım Deseni'nin Şematik Gösterimi


Invoker
Command
+Execute()
Client
Receiver
+Action()
receiver
ConcreteCommand
-state
+Execute()
receiver->Action()