Durum(State) Tasarım Deseni

Nedir?

Durum(State) tasarım deseni, **behavioral** tasarım desenlerinden biridir. Nesnenin durumu değiştiğinde, davranışının değişmesine imkan tanır.

Ne zaman Kullanılır?

Nesne farklı durumlarda, farklı davranışlar gösteriyorsa bu tasarım desenini kullanabiliriz. Farklı durumlara sahip nesnenin, yeni bir duruma geçtiğinde davranışının yani yaptığı işlemin değişmesidir. Örneğin, bir öğrencinin ders başarı durumu kötüyken iyi olarak değişti. Yani yeni bir başarı durumuna geçti. Kötü durumdayken, öğrenci çok çalışarak derslerini düzeltmeye çalıştı. Öğrencinin ders durumu başarılı olunca ise, arkadaşları ile daha fazla vakit geçirmeye başladı. Şimdi bu örneği programlama diline dökersek; Öğrenci, durumu değişen nesneyi temsil etmektedir. Her durumda farklı işler yaptı: Durumu kötü iken çok ders çalıştı, iyi iken arkadaşları ile daha fazla vakit geçirdi.

Nasıl Kullanılır?

Öncelikle bir **State interface**'i oluşturulur. Daha sonra oluşturulan **interface**'i implement edecek somut **State** sınıfları yaratılır. Genelde somut **State** sınıfları **Context** sınıfı türünden **constructor'a** sahip olurlar. **Context** sınıfı yaratılır. Bu sınıf içerisinde state **interface** türünden durumların set edilmesi için bir **metod** bulunur. Bu metod sayesinde **Context** sınıfı state nesnesini tutmuş olur. Son olarak bir **Client** sınıfı oluşturulur. Bu sınıf **Context** sınıfından ve state sınıflardan nesneler üreterek işlemlerin yapılmasını sağlar.

Faydaları Nedir?

Nesnelerin duruma bağlı davranışlarının karmaşık **if-else** veya **switch** ifadeleri ile kontrol edilmesini önler. Örneğin, bir insanın eğitim süreçlerini düşünelim. Süreçler ilkokul, ortaokul,lise, üniversite şeklinde gitmektedir. Her aşama için bir metod yazdığımızı düşünelim ve bu metodlar içerisinde if-else kullanarak eğitim durumuna göre gidebileceği okulu yazdıracağız. Eğer **State** tasarım deseni olmasaydı metodlara şunları yazmak gerekirdi:

```
1   public class Student {
2       private int ELEMENTARY = 0;
3       private int SECONDARY = 1;
4       private int HIGH = 2;
5       private int COLLAGE = 3;
6       private int state;
7
8       public Student(int state) {
9           this.state= state;
10      }
11
12      public void elementarySchool() {
13          if (state == ELEMENTARY) {
14              System.out.println("İlkokula gidemez");
15          } else if (state == SECONDARY) {
16              System.out.println("Ortaokula gidebilir");
17          } else if (state == HIGH) {
18              System.out.println("Liseye gidebilir");
19          } else if (state == COLLAGE) {
20              System.out.println("Üniversiteye gidebilir");
21          }
22      }
23
24      public void secondarySchool() {
25          if (state == ELEMENTARY) {
26              System.out.println("İlkokula gidemez");
27          } else if (state == SECONDARY) {
28              System.out.println("Ortaokula gidemez");
29          } else if (state == HIGH) {
30              System.out.println("Liseye gidebilir");
31          } else if (state == COLLAGE) {
32              System.out.println("Üniversiteye gidebilir");
```

```
            }
        }
    }
    .....
```

Görüldüğü gibi, her bir metod için state kontrolu yapmak gerekiyor. Bu tarz karmaşık **if-else** yapısından kurtulmak için **State** tasarım deseni geliştirilmiştir.

Gerekenler

Türü **interface** veya **abstract** olan **State** sınıfı
En az iki tane **somut State** sınıfı
Context sınıfı: O anki durumu temsil eden somut **State** nesnesini yönetir.
Client sınıfı

Örnek Kullanım Alanları

1. javax.faces.lifecycle.LifeCycle#execute()

Örnek Uygulama

Yukarıda verilen örneği, **State** tasarım deseni ile yapacağız. Bu uygulamamızda her bir state için bir sınıf yazılacaktır.
State Interface

```java
/**
  * State interface
  */
public interface EducationState {
    public void elementarySchool() throws Exception;

    public void secondarySchool() throws Exception;

    public void highSchool() throws Exception;

    public void collage() throws Exception;
}
```

Somut State Sınıfları

```java
/**
  * Somut state sinifi
  */
public class Elementary implements EducationState {
    private Student state;

    public Elementary(Student state) {
        this.state= state;
    }

    @Override
    public void elementarySchool() {
        System.out.println("İlkokula gidiyor");
        //İlkokul bitince ortaokula geçildiginden state degistiriliyor
        state.setState(state.getSecondary());
    }

    @Override
    public void secondarySchool() throws Exception {
        throw new Exception("Ortaokula gidemez");
    }

    @Override
    public void highSchool() throws Exception {
```

```
            throw new Exception("Liseye gidemez");
        }

        @Override
        public void collage() throws Exception {
            throw new Exception("Üniversiteye gidemez");

        }
}
```

```
/**
  * Somut state sinifi
  */
public class Secondary implements EducationState {
        private Student state;

        public Secondary(Student state) {
            this.state= state;
        }

        @Override
        public void elementarySchool() throws Exception {
            throw new Exception("İlkokula gidemez");
        }

        @Override
        public void secondarySchool() throws Exception {
            System.out.println("Ortaokula gidiyor");
            //Ortaokul bitince liseye geçildiğinden state degistiriliyor
            state.setState(state.getHigh());
        }

        @Override
        public void highSchool() {
            System.out.println("Liseye gidiyor");
            state.setState(state.getCollage());
        }

        @Override
        public void collage() throws Exception {
            throw new Exception("Üniversiteye gidemez");

        }
}
```

```
/**
  * Somut state sinifi
  */
public class High implements EducationState {
        private Student state;

        public High(Student state) {
            this.state= state;
        }
        @Override
        public void elementarySchool()throws Exception {
            throw new Exception("İlkokula gidemez");
        }
        @Override
        public void secondarySchool()throws Exception {
            throw new Exception("Ortaokula gidemez");
        }

        @Override
        public void highSchool() {
            System.out.println("Liseye gidiyor");
            //Lise bitince universiteye gecildiginden state degistiriliyor
```

```
            state.setState(state.getCollage());
        }
        @Override
        public void collage() throws Exception {
            throw new Exception("Üniversiteye gidemez");

        }
    }
```

```
    /**
     * Somut state sinifi
     */
    public class Collage implements EducationState {
        private Student state;

        public Collage(Student state) {
            this.state= state;
        }
        @Override
        public void elementarySchool()throws Exception {
            throw new Exception("İlkokula gidemez");
        }
        @Override
        public void secondarySchool()throws Exception {
            throw new Exception("Ortaokula gidemez");
        }

        @Override
        public void highSchool() throws Exception {
            throw new Exception("Liseye gidemez");
        }

        @Override
        public void collage() {
            System.out.println("Üniversiteye gidiyor");
            //Son durak kendisi:)
            state.setState(state.getEducationState());
        }
    }
```

Context Sınıfı

```
    /**
     * Context sinifimiz
     */
    public class Student {
        private EducationState elementary;
        private EducationState secondary;
        private EducationState high;
        private EducationState collage;

        public Student() {
            elementary = new Elementary(this);
            secondary = new Secondary(this);
            high = new High(this);
            collage = new Collage(this);
            educationState = elementary;//ilk state atandi
        }
        public void elementary() throws Exception {
            educationState.elementarySchool();
        }
        public void secondary() throws Exception {
            educationState.secondarySchool();
        }
        public void high() throws Exception{
            educationState.highSchool();
        }
```

```
26      public void collage() throws Exception {
27          educationState.collage();
28      }
29      public void setState(EducationState educationState) {
30          this.educationState= educationState;
31      }
32
33      public EducationState getEducationState() {
34          return educationState;
35      }
36
37      public EducationState getElementary() {
38          return elementary;
39      }
40
41      public EducationState getSecondary() {
42          return secondary;
43      }
44
45      public EducationState getCollage() {
46          return collage;
47      }
48
49      public EducationState getHigh() {
50          return high;
51      }
52  }
```

Client Sınıfı

```
1   /**
2     * Client sinifi
3     */
4   public class Test {
5       public static void main(String[] args) throws Exception {
6           Student student=new Student();
7
8           student.elementary();
9           student.secondary();
10          student.high();
11          student.collage();
12
13          student.setState(new High(student));
14          student.high();
15          student.collage();
16      }
17  }
```

Ekran Çıktısı

```
1   İlkokula gidiyor
2   Ortaokula gidiyor
3   Liseye gidiyor
4   Üniversiteye gidiyor
5   Liseye gidiyor
6   Üniversiteye gidiyor
```

Örnekte görüldüğü gibi öncelikle bir **EducationState** isminde bir **State interface** tanımlandı. Bu **interface** içerisinde ne kadar state varsa o kadar metod kullanıldı. Ayrıca bu metodlar eklenirken, bir state'te iken geçilmemesi gereken state'e geçilmesini önlemek için Exception throw edebilmesi sağlandı. Fakat bu zorunlu değildir. Bunun yerine açıklayıcı mesajlar yazdırabilirsiniz. Bu metodları temsil için ise **Elementary**, **Secondary**, **High**, **Collage** isminde sınıflar yaratıldı. Bu sınıflara özel **State interface**'inde tanımlanan metodlar bulunduğu somut **State** sınıfına göre state değiştirme işlemi eklendi. **EducationState interface**'inde bulunan **elementarySchool()** metodu **Elementary** sınıfı içerisinde kullanıldığında Student ismindeki **Context** sınıfı nesnesine kullanarak aşağıdaki gibi state'i, secondary olarak set etti.

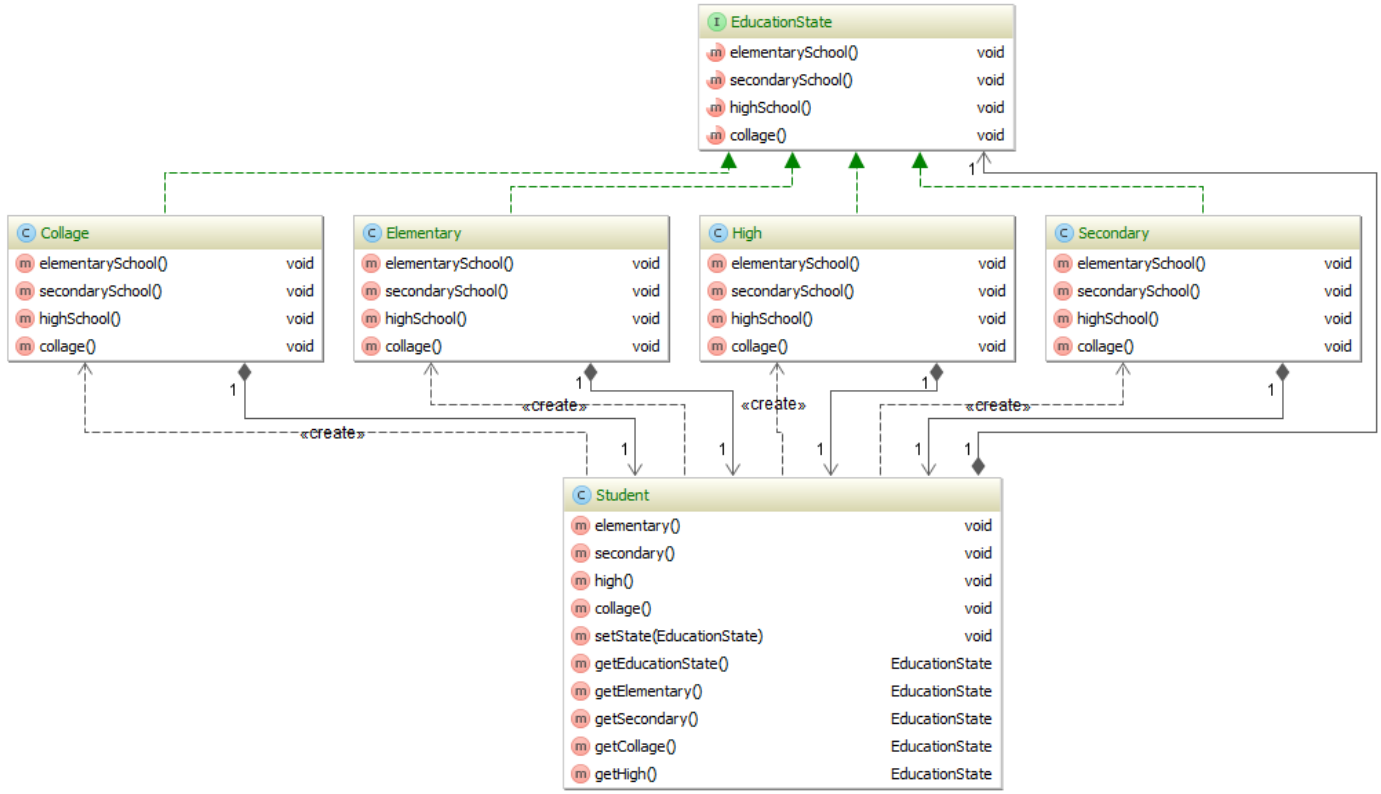
```
1    state.setState(state.getSecondary());
```

Context sınıfı olan **Student** sınıfında ise tüm somut state sınıfları aşağıdaki gibi constructor içerisinde yaratıldı:

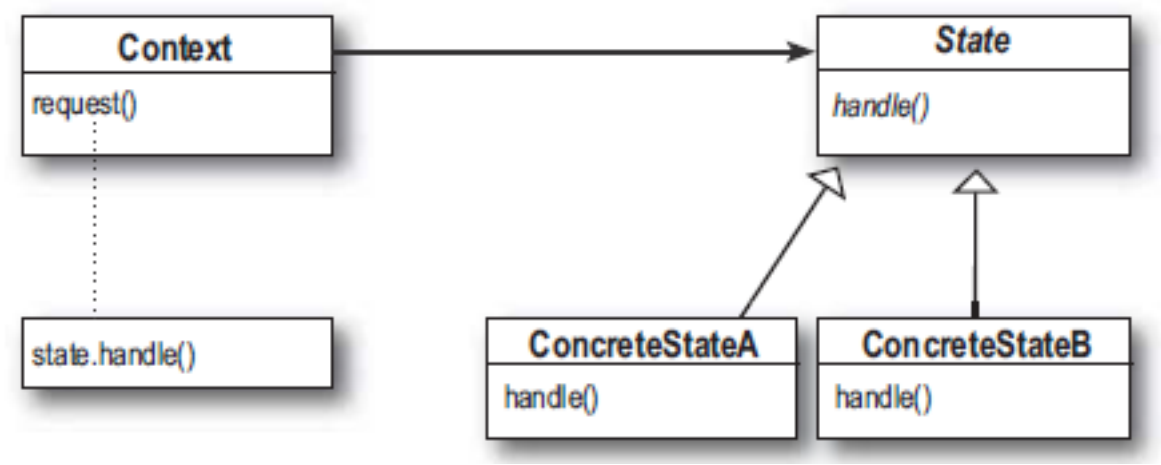
```
1    public Student() {
2         elementary = new Elementary(this);
3         secondary = new Secondary(this);
4         high = new High(this);
5         collage = new Collage(this);
6    }
```

Tüm yaratılan somut **State** nesnelerin **getter** metodları da eklendi. Çünkü bu metodlar somut **State** sınıfları içerisinde çağrılmaktadır.

Uygulamamızın UML Diagramı



State Tasarım Deseni'nin Şematik Gösterimi



Not: State tasarım deseni yapısı ile Strateji tasarım deseni aynıdır. Tek farkı kullanım amacıdır.