SOLID Yazılım Prensipleri

Nesneye yönelik programlamada geliştirilen yazılımın esnek, yeniden kullanılabilir, sürdürülebilir ve anlaşılır olmasını sağlayan, kod tekrarını önleyen tasarım prensipleridir. Robert C. Martin tarafından önerilmiştir.

1. 1.1. Interface segregation principle

> İstemciye özgü (client-specific) birçok arayüz, genel amaçlı bir arayüzden daha iyidir.
>
> Birkaç istemciye sahip bir sınıfınız mevcutsa, her istemci için spesifik arayüzler oluşturun ve bunları sınıfa miras alın. Robert C. Martin

Figure 1 bir sınıf ve birden çok istemciyi göstermekte.

- Character arayüzü, tüm karakter türlerine hizmet vermektedir.
- İlgili yapıda `FighterCharacter` sınıfının kullandığı `dealPhysicalDamage()` metodunun yapısı değiştiğinde, **bu metodu kullanan diğer iki sınıf da** ilgili değişiklikten etkilenecektir.
- İlgili değişiklik sonrası diğer sınıflara da değişiklik gerektirecek durumlar doğabilir.

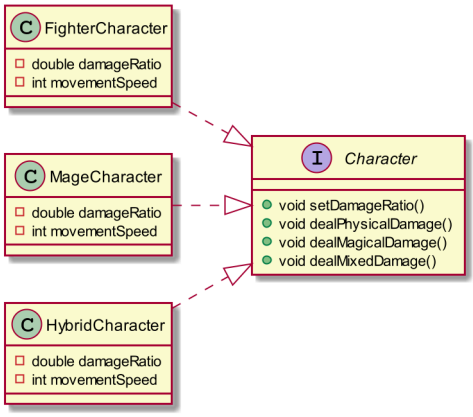

Figure 1. ISP uygulanmamış arayüz tasarımı

Daha iyi bir yaklaşım, Figure 2'de verilmiştir.

- Her client için gereken metotlar, o cliente özel arayüzlere yerleştirilmiştir.
- Eğer Warrior'un kullandığı arayüzün değişmesi gerekirse, yapılacak değişiklikten diğer clientler etkilenmeyecek, **yeniden derlenmeyecek** ve deploy edilmeyeceklerdir.

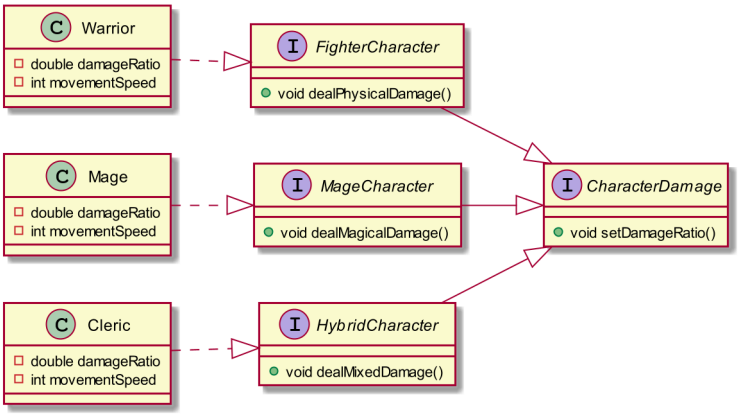

Figure 2. Ayrılmış (segregated) arayüzler

- Interface Segregation Principle (ISP), her servisin kendisine özel arayüze sahip olmasını önermez.
- Bunun yerine, istemciler türlerine göre kategorilere ayrılmalı ve her bir istemci kategorisi için arayüzler oluşturulmalıdır.
    - İlgili örnek için `FighterCharacter` arayüzünü implement eden bir başka `viking` sınıfı oluşturulabilir.
- OOP uygulamaları geliştirildikçe, sınıfların kullandığı arayüzler sıklıkla değişir.
    - Bu aşamada, ilgili değişikliklerin büyük bir etkiye sahip olduğu ve tasarımın **büyük bir bölümünün yeniden derlenmesini** ve **deploy edilmesini** zorunlu kıldığı durumlar olacaktır.
        - Bu etki, mevcut arayüzü değiştirmek yerine mevcut sınıflar için **yeni arayüzler oluşturularak** hafifletilebilir.

1.1. 1.1.1. ISP Uygulaması

Öncelikle ISP uygulanmamış örneği inceleyip, ilgili prensibi ihlal eden noktaları belirleyelim.

Character.java.

```
public interface Character { ❶
 void dealPhysicalDamage();
 void dealMagicalDamage();
 void dealMixedDamage();
}
```

❶ Tüm farklı hasar türleri, tek bir arayüzde toplandı, ISP prensibi uygulanmadı.

MageCharacter.java.

```
public class MageCharacter implements Character {

 @Override
 public void dealPhysicalDamage() { ❶
  throw new NotSupportedException();
 }

 @Override
 public void dealMagicalDamage() { ❷
  // mag. damage calculation...
 }

 @Override
 public void dealMixedDamage() { ❸
  throw new NotSupportedException();
 }

}
```

❶ Karakter büyücü olduğu için fiziksel saldırı gerçekleştiremez. Sınıfın ihtiyacı olmadığı bir metot implement edilmiştir. **IS prensipleri ihlal edilmiştir.**

❷ Sınıfın kullanacağı metot implement edilmektedir.

❸ Karakter büyücü olduğu için fiziksel saldırı gerçekleştiremez. Sınıfın ihtiyacı olmadığı bir metot implement edilmiştir. **IS prensipleri ihlal edilmiştir.**

İlgili projenin ISP gözetilerek tekrardan düzenlenmesi

CharacterDamage.java.

```
public interface CharacterDamage {
 void setDamageRatio(); ❶
```

```
}
```

❶ Karakter türlerine göre değişen hasar türlerini hesaplayan metotlar, alt arayüzler oluşturularak ilgili arayüzlere alındı.

FighterCharacter.java.

```
public interface FighterCharacter extends CharacterDamage{
 void dealPhysicalDamage(); ❶
}
```

❶ Fighter sınıfındaki karakterler, fiziksel hasar verir. Bu sınıfa ait karakterler, ilgili arayüzü implement edecektir.

Warrior.java.

```
public class Warrior implements FighterCharacter {

 @Override
 public void setDamageRatio() { ❶
  // check if character uses melee weapon
  // set it's damage ratio based on different criterias
 }

 @Override
 public void dealPhysicalDamage() { ❷
  // calculate phy damage with damage ratio.
  // then deal damage.
 }
}
```

❶ Karakter sınıfına özel hasar oranı hesaplaması gerçekleştirilir. Burada karakterin sahip olduğu silah ve hasar türü, hasar oranı hesabında değerlendirilir.

❷ Hasar oranı kullanılarak karakterin verebileceği hasar hesaplanır.

- Her bir karakter sınıfı, verdikleri hasar türlerine göre arayüzlere kategorize edilmiştir. Bu şekilde ISP uygulanmış olur.