Interface ve Abstraction (Soyutlama) Kavramı

Interface – Arayüz Kavramı

Nesneye dayalı programlama dünyası ve dolayısıyla Java' da bulunan bir özellikte Interface yani arayüz kavramıdır. Buradaki interface kelimesi user-interface ile karıştırılmamalıdır yaklaşım olarak nesne arayüzlerini ve soyutlamayı ifade eder. Bu aynı zamanda nesneler için ortak bir tanım oluşturma anlamınada gelebilir. Amaç nesnelerin ortak özelliklerini gerçekleştirimleri olmadan sadece tanım olarak bulundurmayı ifade eder.

Arayüz kullanımındaki en önemli sebeplerden biri polymorphism yani çok biçimlilik denebilir. Bunun yanında işaretleme içinde kullanım yöntemleri vardır. Örneğin Java içerisindeki java.io.Serializable interface' i byte array şeklinde yazılacak nesneleri işaretlemek için kullanılan bir interface' tir.

Inteface tanımları sınıflardaki gibi yine java dosyaları şeklinde ancak class yerine interface kelimesi ile tanımlanır. Bunu aşağıdaki gibi örneklendirebiliriz.

```
public interface Dokuman {

}
```

Buna göre Dokuman isminde bir Java dosyası oluşturup içerisine yukarıdaki kodu ekledim. Peki doküman kavramı benim için neden bir interface oldu ? Bunun nedeni dokümanın benim modelime göre soyut bir kavram olmasıdır. Bunu daha da soyutlaştıracak olursak benim için doküman dilekçe, sözleşme veya senet olabilir.

Peki Interaface kavramı içeriğini oluşturmak istediğim durumda Dokuman içerisinde neler olabilir ? Bu sefer içeriği örneklendirelim.

```
public interface Dokuman {

	int DOKUMAN_NO_KARAKTER_SAYISI = 8;

	public int dokumanNumarasiGetir();

	public void dokumanYazdir();
}
```

Şu anda karşımıza alıştığımız sınıf kavramından bambaşka bir şey çıktı. Birincisi daha önce metodlarımızın içerikleri süslü parantez ile belirtilirken burada hiç bir içerik ile karşılaşmadan direkt noktalı virgül ile sonlandırılıyor. Peki bunun nedeni nedir ? Çok basit içeriğin soyut yani abstract olması. Yeni buradaki

metodların tek başlarına bir anlamı bulunmamaktadır. Eğer bir anlam ifade edecekse uygulanmış hali yani implementasyonları bu Interface' i kullanan sınıflar içerisinde tanımlanmalıdır.

Not: Interface' ler soyut kavramlar olduklarından new kelimesi ile nesne oluşturamazlar. Örneğin yukarıdaki örneğe göre **new Dokuman()** ile bir doküman nesnesi oluşturamazsınız.

Not: Soyut yani abstract metodlar **abstract** olarak tanımlanır. Java' da bir Interface içerisindeki tüm metodlar otomatik olarak abstract tanımlanır. Bu durumda bir metodun başına abstract kelimesi eklemeseniz dahi abstract' tır. Yine aynı şey Interface' in kendisi içinde geçerlidir yani abstract' tır. Buna göre yukarıdaki Dokuman örneğini aşağıdaki gibi de yazıyor olabilirdik.

```
public abstract interface Dokuman {

	int DOKUMAN_NO_KARAKTER_SAYISI = 8;

	public abstract int dokumanNumarasiGetir();

	public abstract void dokumanYazdir();
}
```

Uygulama - Implementation

Dokuman Interface' i için örnek bir implementasyon yapalım.

```
public class Sozlesme implements Dokuman{

}
```

Bir sınıf içerisinden bir Interface çağırımı yapmak istiyorsanız yukarıdaki örnekteki gibi **implements** kelimesini kullanmanız gerekir. Bu bizim Interafece kullanımımızı sağlayacaktır. Ancak bu durumda ilgili interface içerisindeki soyut yani abstract metodları override etmeniz yani ezmeniz gerekecektir. Bu mecburi bir durumdur aksi halde kod derlenmez ve aşağıdaki gibi bir hata üretir.

The type Sozlesme must implement the inherited abstract method Dokuman.dokumanNumarasiGetir()

Buna göre doğru implementasyon yani uygulama aşağıdaki gibi olmalıdır.

```
public class Sozlesme implements Dokuman{

	@Override
	public int dokumanNumarasiGetir() {
		// Uygulama kod içeriği
		return 0;
	}

	@Override
	public void dokumanYazdir() {
		// Uygulama kod içeriği
	}
```

```
}
```

Artık bir interface için doğru bir uygulamaya sahip olduk sadece metod içerikleri bizim için henüz bir anlam içermiyor.

Dikkat : Interface içerisinde dikkat çeken bir başka durum ise değişken tanımının tamamen büyük harf olması. Peki bunun nedeni nedir ? Hatırlarsanız Java' da sabit tanımları tamamen büyük harf ile yapılıyordu ve değişkenler static ve final olarak tanımlanıyordu. Peki bu değişken static ve final' mı ? Kesinlikle evet. Eğer bir Interface içerisinde değişken tanımınız varsa bu siz desenizde demesenizde static ve final' dır. Bu yüzden çağırılan sınıflar içerisinde değerleri değiştirilemez ve nesneye ihtiyaç olmadan direkt olarak kullanılabilir.

Interface kullanım amacı

Peki bu durumda Interface kullanmak ve kullanmamak arasındaki fark ne oldu ? Aslında bunu en doğru açıklayacak yer polymorphism konusu olacaktır. Eğer biz tüm dokümanlar içerisinde numarası 2 ile başlayanları ayırmak isteseydik bu durumda Interface kavramı bizim oldukça işimize yarayacaktı.

Bu neden polymorphism konusundaki **Interface ler ve Polymorphism** konusunu incelemenizi tavsiye ederim.

Birden fazla Interface kullanımı

Java içerisinde sadece tek bir sınıfın miras alınması sağlanabiliyordu ancak aynı anda birden fazla Interface çağırımı yapabilme şansına sahipsiniz. Bunu aşağıdaki gibi örneklendirelim.

```
public class Sozlesme implements Dokuman, Evrak{

}
```

Örneğe göre aynı anda Dokuman ve Evrak isminde iki adet Interface çağırımı yapmış olduk. Bu durumda her ikisi içerisindeki abstract metodların gerçekleştirimini yapmak zorundayız. Ancak artık Sozlesme sınıfı hem bir doküman hemde değerli evrak şeklinde bir tip ifade edecektir. Bu bizim genişletilebilirliği sağlayan oldukça yetenekli bir özellik. En nihayetinde biz burada birden fazla tip ifade edebiliriz.

Interface' lerin genişletilebilmesi

Bir Interface başka bir Interface' ten türeyebilir mi ? Bu aşağıdaki örnekteki gibi mümkün olabilir.

```
public interface Dokuman extends Kayit, BasiliNesne{

}
```

Burada genişletme amacıyla kullandığımız kelime **extends** ancak sınıflardakinin aksine bir Interface birden fazla Interface' ten türeyebiliyor. Tabi burada soyut metod çakışmasına yine dikkat etmek gerekecektir. Yani aynı isimde yanlız farklı tanımlanmış metodlar karmaşıklığa yol açabilir.

Abstraction – Soyutlama Kavramı

Interface' ler zaten soyutlamayı ifade ederken Abstract sınıf tanımı neyi ifade eder ? Bu yarı soyut olarakta düşünülebilir. Interface' lerden farkı yine soyut olması ancak sınıf olarak tanımlanmasıdır. Bu durumda içeriğindeki metodların tamamı soyut olmak zorunda değildir ki hatta içeriğinde yine hiç soyut yani abstract metodta bulunmayabilir.

Abstract yani soyut sınıf tanımları aşağıdaki gibi yapılır.

```
public abstract class Dokuman {

	public abstract int dokumanNumarasiGetir();

	public void dokumanYazdir(){
		// Yazdirma islemleri
	}
}
```

Bu örnekteki abstract olan sınıf içerisinde iki adet metod bulunmaktadır. Bunlar abstract ve abstract olmayan normal metodlardır. Bu durumda yukarıdaki sınıf örneğinin gerçekleştirimi aşağıdaki gibi olacaktır.

```
public class Sozlesme extends Dokuman {

	@Override
	public int dokumanNumarasiGetir() {
		// Metod gerçekleştirimi...
		return 0;
	}

}
```

Burada uygulaması olması gereken metod sadece dokumanNumarasiGetir metodudur ve yine Interface' lerdeki gibi override edilmesi yani ezilmesi zorunludur.

Not : Abstract yani soyut sınıflar aynı Interface' ler gibi tek başlarına nesne oluşturamazlar.

Abstract – soyut sınıf kullanım amacı

Soyut sınıf kullanımını gerçek hayata uygulayacak olursak bunu aşağıdaki gibi örnekleyebiliriz.

Öncelikle senaryomuza göre veritabanı operasyonlarımızın olduğu bir uygulama tasarladık ve buna göre bir çok işlemimiz veritabanı ile haberleşecek. Bu durumda öncelikle bazı hazır işlemlerin olduğu aşağıdaki gibi ortak bir yardımcı soyut sınıf hazırladım. Bu sınıfın soyut olmasının nedeni tüm metodlarının içeriğinin henüz belirli olmamasıdır ki belirli olmayan metodumuz her işlemde değişecek sorgu metodudur.

```
public abstract class VeritabaniYardimciSinifi {

	void baglan(){
```

```
            // Veritabani baglantisi yapiliyor
      }

      abstract void sorgu();

      void calistir(){
            baglan();
            sorgu();
            baglantiKes();
      }

      void baglantiKes(){
            // Veritabani baglantisi yapiliyor
      }
}
```

Şimdi bunun için bir gerçekleştirim yapalım.

```
public class SozlesmeSorgu extends VeritabaniYardimciSinifi {

      @Override
      void sorgu() {
            // Sozlesme sorgu kodu
      }

}
```

Örneğe göre sözleşme işlemlerinin olacağı bir sorgu sınıfı hazırlamış olduk. Peki bu işimizi ne kadar kolaylaştırdı ? Çok basit, artık sadece aşağıdaki kod ile veritabanından sorgu getirme imkanına sahip olduk.

```
public class Ornek {

      public static void main(String[] args) {
            SozlesmeSorgu sozSorgu = new SozlesmeSorgu();
            sozSorgu.calistir();
      }
}
```

Tüm bunların yanında yine polymorphism kavramından yararlanma şansına sahibiz ki bunu bir sonraki bölümde görüyor olacağız.

Not : Az önceki yorumlardan birinde abstract tanımlanmış sınıf içerisinde hiç abstract metod olmayabileceğini yani böyle bir zorunluluk olmadığını belirtmiştik. Peki böyle bir sınıf bizim ne işimize yarayabilir ? Çok basit olarak nesne oluşturulmasını istemediğimiz sınıfları abstract olarak tanımlayabiliriz. Bu durumda **new** ile bir nesne oluşturma imkanını engellemiş oluruz yani sadece miras alınabilirler.