Abstract Soyut Sınıf Kavramı

Abstract Sınıf ?

Abstract sınıf nesne üretilmeyen ve soyut metotlar barındıran bir yapıdır. Metotların sadece tanımları bulunur ve bu sınıftan türetilen alt sınıflar bu metotların içeriklerini kendilerine göre oluşturulmak zorundadır.

Bir kalıtım hiyerarşisinde alt sınıftan üst sınıfa doğru gidildikçe daha genel ve özel alanların olmadığı bir yapıya doğru gidilir.

Canlı > Hayvan > Ayı > Kutup Ayısı burada Canlı sınıfı en genel, Kutup Ayısı ise en spesifik sınıftır. Canlı sınıfı altındaki sınıfların en genel özelliklerini taşır ve bazen bu sınıf o kadar soyut olur ki bundan bir nesne üretmek gerekmez. Sadece belli ortak özellikleri barındırmasını sağlamak için böyle bir sınıfı kullanırız.

Abstract sınıflar diğer sınıf yapısına oldukça benzer. Ama bazı farklılıkları bulunur.

- Abstract sınıflardan new ile yeni bir nesne üretemezsiniz.
- Abstract sınıflarda metotların tanımı olur, içerikleri olmaz. Bu içerikler abstract sınıftan türetilmiş sınıflarca yazılır.
- İçerikleri yazılmayan bu metotlar abstract metot olarak tanımlanır. Bir sınıfta bir tane abstract metotun olması o sınıfı abstract yapmaya yeterlidir.
- Bir metot abstract ise o sınıf mutlaka abstract olmalı. Ve bu metotlar static olamazlar.
- Her ne kadar new ile yeni nesne üretilmese de abstract sınıfların da constructorları vardır ve alt sınıftan bir nesne üretildiğinde bu kurucu metotlar sırasıyla calıstırılır.
- Abstract sınıflarda hiç metot da olmayabilir. Alt sınıfları bir üst sınıf tanımında toplamak için böyle bir durum olabilir.
- Üst sınıf normal bir sınıf iken alt sınıf abstract olabilir. Object sınıfı normal iken olusturacagımız bir sınıf abstract olabilir.

Bunların dışında soyut sınıflar normal sınıflara benzer. Birleştirici bir rol oynarlar. Polimorfizm ve geç bağlama sayesinde bu sınıfın değişşkenlerine alt sınıf nesneleri bağlanabilir.

Interface Arayüz Arabirim Kavramı

Interface Nedir ?

Javada çoklu kalıtım yoktur, işler karışıklaşmasın diye buna izin verilmez. Böyle bir ihtiyaç olduğunda interfaceler kullanılabilir.

Interface en soyut ifadeleri içeren yapılardır. Soyut sınıfların bir üst modeli gibi düşünülebilir. Bir sınıf bir interfaceyi gerçekleştirerek orada tanımlanmış gövdesiz metotları gerçekleştirir.

Interface içinde tanımlanan gövdesiz metotlar publictir, arayüzler içinde sabitler de tanımlanabilir, bunlar ise static ve finaldır.

Bir sınıf bir interfaceyi gerçekleştirirken implements anahtar kelimesi kullanılır. Class A implements B --> a sınıfı b interfacesini gerçekleştiriyor

Interfacelerde metot tanımları ve sabitler bulunur. Burada illaki kalıtım mantığı olmak zorunda değildir. Interfaceler ile ortak alanları bulunan ama aralarında bir kalıtım özelliği olmayan sınıfları bir çatı altında toplayabiliriz.

Interfacelerden nesne üretemeyiz ama değişken olarak kullanabiliriz. Bu değişkenlere interface'imizi implemente etmiş sınıf nesnelerini bağlayabiliriz.
InterfaceA a=new SinifAdi();

Soyut bir sınıf bir interfaceyi implemente edebilir. Ama soyut sınıflarda metotların gövdesi genelde olmayacağından, soyut sınıfların amacı genellikle birleştirici rol oynamak oldugundan interface'de bulunan metotu gerçekleştirmek zorunda değildir.

```
interface Yenilebilir{
    public void yenmeSekli();
}

abstract class Hayvan implements Yenilebilir{
    public abstract void sesCikar();
    //bu sınıf yenmeSekli metotunu gerçekleştirmek zorunda değildir.
}
```

class Tavuk extends Hayvan dediğimizde bu sınıfta hem yenmeSekli hem de ses cikar metotlarının tanımı olmalıdır.

Bir sınıf birden fazla interfaceyi implemente edebilir.
class A implements InterfaceA, InterfaceB{}

Bir interface başka bir interfaceden kalıtılabilir.
interface InterfaceA extends InterfaceB gibi bir tanım doğrudur.

Örnekler ve daha detaylı anlatımları video derste bulabilirsiniz.

Soyut Sınıfla Interface Farklılıkları

Farklılıklar

İki yapıyı da öğrendik, gelin hep beraber bunlar arasındaki ortak yanlardan ve farklardan bahsedelim.

Soyut sınıflarda değişken tanımlarken herhangi bir kısıtlama yoktur, normal bir sınıfta olduğu gibi farklı erişim belirleyicilere sahip değişkenler tanımlanabilir. Ama interfacelerde tanımlanan değişkenler mutlaka public, static ve final olmalıdır.

Soyut sınıfların normal sınıflar gibi kurucu metotları bulunur ve alt sınıftan bir nesne üretildiğinde soyut sınıftan alt sınıfa doğru kurucu metotlar tetiklenir.Kurucu metotu olmasına rağmen soyut sınıflardan new ile nesne üretemeyiz. Interfacelerde ise kurucu metot kavramı yoktur ve new ile nesne üretilemez.

Soyut sınıf metotlarını tanımlarken herhangi bir kısıtlama yoktur. Metotlar private veya public olabilir. Soyut yani gövdesi olmayan veya somut yani gövdesi olan metotlara sahip olabilir, interfacelerde ise tüm metotlar public ve soyut yani gövdesi olmayan metotlar olmalıdır.

Bir sınıf sadece tek bir soyut sınıftan extends olabilirken, birden fazla interfaceyi implemente edebilir.

Java 8 ile beraber gövdesi olan metotlar barındıran interfaceler olusturuluabilir.

```
interface Kisi{
    void yemekYe();
    void sporYap();
    default void adiniSoyle(){
        System.out.println("Ben kişi interfacesiyim");
    }//metotun basına default yazmak zorundayız
}
```

Java 8 ile beraber artık interfacelerde static metotlarda tanımlanabilir. Bu metotları override edemeyiz. Aynı interface içindeki değişkenleri kullanır gibi bunlarıda InterfaceAdi.metotAdi() şeklinde kullanabiliriz.

Bir interface birden fazla interfaceden extends olabilir.

```
interface A{
    void aa();
}
interface B{
    void bb();
}
interface C extends A,B{
    void cc();
}// bu kalıtım yapısı sınıflarda hata verirken, interfacelerde bir sorun oluşturmaz.
```

Interface Detaylar Özet ve Java 8 Yenilikleri

- Interfacelerin içinde somut yani içi dolu olan metotlar olamaz.

- Interfacelerde tanımladığınız değişkenler varsayılan olarak public,static ve final'dır. Bu değişkenlere bir kere değer atadıktan sonra değiştiremeyiz. Bu yüzden interfaceler sabit tanımlanırken de kullanılabilir.

- Interfacelerde yazdığımız metot tanımları varsayılan olarak abstract ve publictir.
- Interface dosyaları derlendiğinde .class uzantılı dosyalar oluşturulur.
- Bir sınıf bir interfaceyi implemente ettiğinde oradaki metotların içini doldurmak zorunda kalır ve bu metotlar da public olmalıdır. Çünkü metot overriding olayında üst sınıftaki metotun alt sınıfta görünürlüğü azaltılamaz.
  Yani interfacedeki metotA(); metotunu bir sınıf override ettiğinde;

```
class A implements InterfaceA{
    @Override
    public void metotA(){
    //hata yok
    }

    @Override
    void metotA(){
    //hata verir.public erişim belirleyici defaulta düşürüldüğü için
    }

}
```

- Varsayılan olarak bir interface abstracttır ama public değildir.
- Interfacelerde static ilkleme ve nesne ilkleme blockları kullanılmaz.
- Bir sınıf birden fazla interfaceyi implement edebilir.
- Bir interface baska bir interfaceden extends diyerek kalıtılabilir.
- Java 8 ile beraber default ve static metot tanımları yapabiliyoruz.
- Default metotu override edebiliyorken statik metotu override edemeyiz.

Erişim Belirleyiciler

public

Bu belirleyici ile belirlenen sınıf, metot ve sınıfa ait değişkenlere heryerden erişilebilir.

private

Bu belirleyici ile belirlenen sınıf üyelerine ve dahili sınıflara başka sınıflar ve paketlerden erişilemez. Sınıflar private olmaz(inner sınıflar hariç) Private tanımlanan metotlar sadece bu sınıfa aittir. Sınıfın kurucusu private yapılırsa o sınıftan nesne üretilemez ve o sınıftan alt sınıflar kalıtılamaz.

protected

Bu belirleyici ile belirlenen sınıf üyelerine ve dahili sınıflara aynı paket içerisindeki diğer sınıflar erişebilir. Farklı paket söz konusu ise bu durumda kalıtım kullanılarak oluşturulmuş alt sınıflar erişebilir. Sınıflar protected olamazlar.

friendly-default

Herhangi bir erişim belirleyici kullanılmazsa java o sınıf veya metotlara aynı paket içinden erişilecek bir sınır koyar. Bu sınıflara paket dışından erişilemez.

Bir metot ve değişken için;

	default-friendly	private	protected	public
Aynı sınıf	EVET	EVET	EVET	EVET
Aynı pakette alt sınıf	EVET	HAYIR	EVET	EVET
Aynı pakette alt sınıf olmayan	EVET	HAYIR	EVET	EVET
Farklı pakette alt sınıf	HAYIR	HAYIR	EVET	EVET
Farklı paket alt sınıf olmayan	HAYIR	HAYIR	HAYIR	EVET

Non Access Modifiers

Tanım

Javada 2 tip modifier vardır. Birincisini gördük : access modifier yani erişim belirleyiciler. Bunlar private public defaut ve protected idi.

Bunların dışında non access modifier diyebileceğimiz ve erişim dışında farklı işlevleri olan belirleyiciler vardır.

static : Bir sınıfa ait tüm nesnelerin erişebileceği, sınıfa özgü olan nesne ve değişkenleri belirler. Bunlar bellekte sınıfa özgü olan farklı bir alanda saklanırlar. Statik değişkenler, statik metotlar ve static kod bloğu...

final : Bir değişken, metot veya sınıf için kullanılır. Final belirtilen bir değişkene ilk değer atandıktan sonra güncelleme yapılamaz. Final belirtilen bir metot alt sınıflarca override edilemez. Final belirtilen bir sınıf kalıtılamaz.

abstract : Sınıf veya metot için kullanılır. Kurucu metot veya değişkenlere abstract tanımı yapılamaz. Bir metot abstract olarak tanımlanırsa alt sınıflar bunu gerçekleştirmek zorundadır. Bir sınıfta bir tane metot abstract ise o sınıf abstract olmalıdır ve abstract sınıflardan new ile nesne oluşturulamaz.

synchronized ve volatile thread konusunda işlenecektir.
transient nesnelerin serileştirilmesi konusunda işlenecektir.

strictfp : Noktalı ifadelerin yani float değerlerin hesaplanmasında tüm platformlarda aynı sonucun çıkması için daha tutarlı sonuçlar almak için kullanılır. Sınıflara metotlara ve interfacelere uygulanabilir.

Inner Nested Dahili Sınıflar 1

Member Inner Class

Inner veya nested classlar bir başka sınıfın tanım aralıgında(scope, kapsam) olan sınıflardır.

İlk olarak member inner classlardan bahsedelim.

Nasılki bir sınıfın değişkenleri ve metotları olabiliyor ve onlara nesne oluşturup ulasabiliyorsak, bir sınıfın içinde baska bir sınıf oluşturup bu sınıflara da nesne üzerinden erişip işlem yaptırabiliriz.

```
class DisCerceveSinif
{
    int i;       //Nesne değişkeni

    void methodOne()
    {
        //Nesne Metotu
    }

    class InnerNestedDahiliSinif
    {
        //Nesne üzerindne erişilebilen Sınıf
    }
}
```

Bu sınıflar static nested sınıf veya non-static nested sınıf(inner class) olarak çeşitlendirilebilir..

Bunlara erişmek için önce dış sınıftan bir nesne üretilir sonrasında iç sınıfa erişilir.

```
DisCerceveSinif.InnerNestedDahiliSinif a=new DisSinif().new InnerSinif();
```

Bunlara member inner class diyoruz. Bu sınıflarda static olmayan member yani alanlar olmalıdır. Statik bir değişken veya metot tanımına izin verilmez.

NOT: Eğer değişken statik ve final olursa hata çıkmaz.

Member classlarda isimsiz kod blokları olabilir ama static bloklara izin verilmez.

```
class DisSinif{

    class IcSinif{

        {
        //sorun yok
        }
        static{
            //compile time hata
        }
    }

}
```

Inner Nested Dahili Sınıflar 2

Member Inner Class Devam

İçerdeki sınıf dış sınıfın tüm alanlarına (statik veya değil) erişebilir. Aynı şey dıştaki sınıf içinde geçerlidir ama öncesinde iç sınıftan nesne üretmelidir. Çerçeve sınıf iç sınıfın tüm alanlarına erişebilir.

Iç sınıflar abstract veya final olabilir, her ikisi aynı anda olamaz.

Local Inner Class

Local inner class static olmayan inner classların bir diğer türüdür. Bir metot veya bir blogun içinde tanımlanan sınıflara denir.

Local inner sınıflar statik olamaz. Local değişken olarak düşünülürse, local değişkenler statik olamaz. Ayrıca bu sınıflar static metot veya değişkene de sahip olamazlar. Ama istisna olarak static final değerleri saklayabilirler.

Bunları kullanmak için nesne üretmek yeterlidir.

```
class CerceveSinif
{
    void methodOne()
    {
        class LocalInnerClass
        {
            int i;     //Non-static alan

            static final int j = 10; //static ve final alan

            void methodOne()
            {
                System.out.println("Local inner sınıftan yazıyorum");
            }
        }

        System.out.println(LocalInnerClass.j);  //static ve final alana sınıf üzerinden erişilebilir.

        LocalInnerClass inner = new LocalInnerClass();

        System.out.println(inner.i);     //nesne üzerinden değişkene eriştik

        inner.methodOne();   //nesne üzerinden metota eriştik
    }
}

public class InnerClasses
{
    public static void main(String args[])
    {
        CerceveSinif disSinif = new CerceveSinif();
        disSinif.methodOne();
    }
}
```

Bu sınıflar sadece tanımlandıkları yerde geçerlidir.Bu sınıflar tanımlandıkları metotun içinde bulunan değişkenleri kullanabilir.

Bu sınıfları belirtirken erişim belirleyici kullanılmaz, ama bu sınıfın içinde tüm erişim belirleyicilere sahip değişken ve metotlar olabilir.

Bu sınıflar abstract veya final olabilir, ikisi aynı anda olamaz.

Anonymous Inner Class

İsimsiz Dahili Sınıflar

Belki de en çok kullanılacak , bu bölümün en önemli kısmı burasıdır. Bu sınıflar isimsiz iç sınıf veya statik olmayan iç sınıf olarak adlandırılır.

Bir sınıf düşünün ve bu sınıftaki bir metotu küçük bir değişiklikle kullanmak istiyoruz. Böyle bir durumda bu sınıftan kalıtılan bir sınıf yapıp burda metotu override edip sonrasında bu yeni sınıf nesnesini kullanmamız gerekirdi.

Bir tane metotu biraz değiştirmek için çok fazla zorluk yaşadık. İşte burada isimsiz sınıflardan yararlanabiliriz.

```
GeometrikSekil nesne=new GeometrikSekil (){
    @Override
    void alanHesapla(){
        System.out.println("Isımsız sınıfta bu metot tekrar yazıldı");
    }
}
```

Bu ifade ile yukarıda anlatılanlar aslında yapılmış olur. Burada sanki isim verilmemiş bir sınıf bu GeometrikSekil isimli sınıftan kalıtılmış, isim verilmemiş sınıfımız bu üst sınıftaki alanHesapla isimli metotu override etmiş gibi düşünülebilir.

Bu değişiklik sadece nesne için geçerlidir. Buradaki gibi değişiklikle metotu kullanmak istiyorsak yukarıdaki gibi tekrardan isimsiz bir iç sınıf olusturmamız gerekir. Yani alanHesapla da içeriğin yeniden yazılması sonradan oluşturulacak diğer nesnelere aktarılmayacaktır.

Bu oluşturulan yapıda bir alt sınıf oluşturulur ama ismi yoktur ve başka bir sınıf içindedir. Ondan dolayı buna isimsiz iç sınıf denir.

Ve bu olurken aslında üst sınıfı GeometrikSekil olan bir alt sınıf ve bunun nesnesi oluştuurlur ve bu nesne üst sınıf tipindeki değişkene bağlanmış olur. Burda da polimorfizm kullanmış oluruz aslında.
GeometrikSekil nesne = new IsimsizInnerClass() gibi bir durum oldu.

İsimsiz iç sınıfları nesne üretilmeyen interface ve soyut sınıflarda da oldukça fazla kullanırız. Mantık aynıdır, interface için düşünürsek.

```
Interface a =new Interface(){

    @Override
    public void interfaceMetotu(){
        System.out.println("Metot");
    }
}
```

Sanki bir sınıf bu interfaceyi implemente etmiş ve biz ondan bir nesne oluşturmuşuz gibi düşünebilirsiniz.

Static Inner Class

Inner sınıfı statik olarak belirleyebiliriz. Bu sınıflarda hem statik hem statik olmayan metot ve değişkenler bulunabilir.

Statik inner sınıftan dışardaki sınıfın statik alanlarına erişilebilir, ama statik olmayan alanlara erişemeyiz.

Bir static inner sınıf abstract olabilir, ama statik metotlar abstract olamaz.

Statik inner sınıflar final olabilir.

```
class CerceveSinif
{
    int i = 10;

    static void cerceveStatikMetot()
    {
        System.out.println("Static metot CerceveSinif");
    }

    static class DahiliSinifBir
    {
        int i = 20;

        static void icSinifStatikMetot()
        {
            System.out.println("Static metot DahiliSinifBir");
        }
    }

    static class DahiliSinifIki
    {
        int i = 30;

        static void icSinifStatikMetot()
        {
            System.out.println("static methot NestedClassTwo");
        }
    }
}

public class NestedClasses
{
    public static void main(String[] args)
    {
        CerceveSinif.cerceveStatikMetot();
        CerceveSinif disSinif = new CerceveSinif();
        System.out.println(disSinif.i);

        CerceveSinif.DahiliSinifBir.icSinifStatikMetot();
        CerceveSinif.DahiliSinifBir icSinif1 = new CerceveSinif.DahiliSinifBir();
        System.out.println(icSinif1.i);

        CerceveSinif.DahiliSinifIki.icSinifStatikMetot();
        CerceveSinif.DahiliSinifIki icSinif2 = new CerceveSinif.DahiliSinifIki();
        System.out.println(icSinif2.i);
    }
}
```