2023-2024 Bahar Dönemi İleri Java Final Projesi

Adı: Samet

Soyadı: İmirgi

Numarası: H5220064

Konusu: Nesne Tabanlı Programlama Temelleri

İçindekiler

Nesne Tabanlı Programlama Nedir?

1)En basit tabiri ile diğer programlama yöntemlerinden farklı olarak class-object mantığına dayanan ve bu mantık çerçevesinde projelerin daha dinamik, kompleks (karmaşık) olan yapıların önüne geçerek daha bütüncül bir şekli ile ortaya çıkmıştır.

2)Nesne tabanlı programlama mantığı artık diğer programlama yöntemlerinden ayrılmış olup bu mantığı insanların aklında hayal edebilmesini sağlamıştır.

3)Kısaca diğer programlama yöntemlerinden daha somut hale gelmiştir.

4)Ki eski programlama mantığı dediğimiz kompleks ve karmaşık olan yapılar yerini belli bir zaman'dır. Nesne tabanlı programlamaya bırakmıştır.

Nesne Tabanlı Programlamanın 4 Temel Başlığı Nelerdir?

1)Kapsülleme (Encapsulation)

- Verilerimizi, değişkenlerimizi tasarım aşamasında kapsüllemek yani dışına çerçeve giydirmek ve değişkenin içine atanan istenmeyen değeri koruma atına almak için kullandığımız bir yapıdır.

- Burada en önemli konu hususu güvenliktir. Paketler arasında verilerimize erişmek veya class'lar arasında verilerimize erişilmesine izin

vermek istemiyorsak bu başlık bizim için önemli olacaktır.

- Sözün Özü şu ki anlamsız, saçma verilerin değişkenimize atanmasını engellememiz gerekiyor. Bu da getter ve setter ile mümkündür.

- Encapsulation yaparken verimi, değişkenimi "public" erişim belirleyicisi ile tanımlamıyorum onun yerine verime ve değişkenime doğrudan bir şekilde erişilmesine izin vermeyip "private" erişim belirleyicisi ile güvenliğimi sağlıyorum.

- "Private" ile değişkenlerimizi tanımladık fakat farklı class'lardan erişemiyorum. Ne yapacağım tabi ki "Private" ile tanımlanan değişkenlere erişip değer atamanın 2 yolu var;
  - Constructor yapısı ile,

- Getter,setter yapısı ile (zaten bunlara ‘Encapsulation’ diyoruz.)

Örnek;

```
Public class calisan{
Private Long id;
Private String ad;
Private int maas;


Public calisan(Long id, String ad, int maas){
this.id=id;
this.ad=ad;
this.maas=maas;
}
}
```

Bu şekilde küçük bir örnek ile tanımlamış oldum.

2) Inheritance (Kalıtım), Miras Alma;

- Burada ise amacımız kod tekrarından kurtarıp karmaşık yapının önüne geçmek ve bütün class’ları tek bir yerden (ana class

üzerinden) değişkenleri ve methodları kontrol etmektir.

- Miras veren class'a "üst" class (süper class) diye tabir edilir. Yani "ana" class'ım olur. Kontrollerimi burada yapıyorum.

- Miras alan class'a "alt" class (sub class) diye tabir edilir.

- Kısaca Bir class miras alırsa mirası veren class'ın içindeki değişkenleri,methodları kendi method ve değişkeniymiş gibi kullanabiliyor.

- Inheritance kullanmak class'ımın bulunduğu package altındaki sub class'lardan da birden fazla yazmaktan kurtarıyor yani kod tekrarını önlüyor diyebiliriz.

- Ki miras almak istiyorsak **"extends"** kelimesi ile belirtmiş olmalıyız.

Şimdi kısa bir örnek üzerinden pekiştirmiş olalım Inheritance (kalıtım) başlığını.

Örnek;

- Hayvan class'ı bir üst class'dır yani süper class içinde hayvana ait özellikleri barındırmış olup bir de "nefesAl()" methodu vardır.

```
public class hayvan{

public Long id;
public String hayvanAdı;

public int yas;

public String renk;

public int ayaksayisi;
```

```
public void nefesAl(){
system.out.println("Hayvan  nefes alıyor.");
}
}
```

- Bu class ise kopek classı'dır. "hayvan" classını extends etmiş olup hayvan class'ı içinde olan tüm özelliklere erişecektir.

```
Public class kopek extends hayvan{ }
```

- Bu class ise kedi classı'dır. "hayvan" classını extends etmiş olup hayvan class'ı içinde olan tüm özelliklere erişecektir.

```
Public class kedi extends hayvan{

}
```

- Bu class ise test class'ımdır. Bu class üzerinden nesne türeteceğiz.

```
Public class test{
Public static void main (String[]args){

hayvan hayvan1 = new hayvan();
hayvan1.nefesAl();
```

```
kopek kopek1 = new kopek();
kopek1.nefesAl();

kopek kopek2 = new kopek();
kopek2.nefesAl();

kedi kedi1 = new kedi();
kedi1.nefesAl();

}
}
}
```

Bu şekilde küçük bir örnekle Inheritance (Miras Alma) başlığını da tamamlamış olduk.

3)Polymorphism (Çok Biçimlilik)

- Farklı farklı class'ların aynı arabirim üzerinden çalışabilmesi prensibidir.

- Tek bir method için farklı class'larda farklı şekillerde uygulanabilir.

- Buradaki **amacımız** kodun daha esnek yapıda olmasını sağlamak.

- Farklı bir çok nesneyi aynı şekilde işleyebilme yeteneği sağlar.

Küçük bir örnek yapısı ile pekiştirmiş olalım.

Örnek;

```
Class araba{
void sürmek(){
system.out.println(“Araba gidiyor.”)
}
}

Class mercedes extends araba{
@Override
void sürmek(){
system.out.println(“Mercedes EQB”)
}
}

Class bmw extends araba{
@Override
void sürmek(){
system.out.println(“BMW M5 Sport”)
}
}
```

```
Public class main{
Public static void main(String[] args){
araba araba1 = new araba();
araba araba2 = new mercedes();
araba araba3 = new bmw();

araba1.sürmek();
araba2.sürmek();
araba3.sürmek();
}
}
```

Burada yapılan örnek “araba1”, “araba2”, “araba3” nesneleri ‘araba’ sınıfına ait referans ile tanımlanmış olsalar dahi ‘sürmek()’ methodunu çağırdığımızda her bir nesne kendi sınıfına ait olan ‘sürmek()’ methodunu çalıştırır.

Bu şekilde Polymorphism (Çok Biçimcilik) başlığınıda bitirmiş olduk.

4) Abstraction (Soyutlama)

- Amaç sub class'larda belirli methodları kendilerine uygun olacak şekilde uygulamalarını sağlamaktır.

- Ortak methodlarımızı ve ortak değişkenlerimizi abstract class içerisinde tanımlıyoruz.

- Daha sonra tanımlamış olduğumuz abstract class'larımızı sub class'larda "extends" ediyoruz. Yani miras alıyoruz

- Miras alınan abstract class'ı içerisindeki abstract methodları'da sub class'larda "override" etmek zorundayız.

- Inheritance'da miras alan class içerisinde "override" etmek gibi zorunluluğumuz yoktu. Tercih meselesiydi.

- Biz abstract class'ından nesne türetemeyiz

- Abstract'lar tek başına bir anlam ifade etmiyor. Miras verdiğinde bir anlamı oluyor diyebiliriz.

Örnek;

```
Abstract class genel{
Public void sonuc(){
System.out.println("Akbank ATM'sine Hoş Geldiniz");
}
}
Class Main extends ornek{
Public static void main(String[] args){
Main main = new Main();
Main.sonuc();
}
}
```

Burada “genel” adında bir abstract class oluşturduk. Genel classımızda “sonuç()” methodunu oluşturduk. Sonra Main sınıfımızda bir nesne oluşturup bu nesneyi kullanarak “sonuç()” methodumuzu çağırdık.

Böylelikle Nesne Tabanlı Programlama’nın 4 temel başlığını bitirmiş olduk.

Constructor Nedir ve Nasıl Kullanılır?

- Birden fazla constructor yapım olabilir bir class içerisinde fakat aynı değerli veya aynı 2 parametresiz değer olamaz.

- Tek bir nesne komutu ile tek bir constructor yapısına erişebilirim.

- Tek bir nesne komutu ile birden fazla farklı constructor yapısını bağlayamam.

- Farklı nesne komut adları ile erişmek istediğim constructor yapısını belirterek bu şekilde bir çok constructor yapısına erişebilirim.

Örnek;

```
public Class ogrenci{
public ogrenci(){
}
public ogrenci(int id){
}
public ogrenci(String id){
}
public ogrenci(int id, String isim){
}
}
```

Burada 4 adet Contructor oluşturdum. Görüldüğü gibi hepsi farklı farklı tek bir adet method değilde birden fazla method (constructor)

Yazdım ve burada ayrıca ‘method overloading’ yapmış olduk yani methodlara aşırı yükleme anlamında.

Java da Final Anahtar Kelimesi Nedir?

- Final anahtar kelimesi bir değişken için method için veya sınıf için değiştirilemez olduğunu belirtir. Bir öğe “final” anahtar kelimesi ile bildirildiğinde bir daha o öğe değeri veya yapısı değiştirilemez.

Final Değişkenler: Bir değişkeni “final” olarak belirttiğimizde o değişkenin değeri zaten ilk başta

atandığı için sonradan değiştirilemez. Genel olarak sabit değerler ya da önemli özellikler için kullanılır.

Örnek: final int PI = 3.14;

Final Methodlar: Bir methodun önüne “final” ekleyerek, bu methodu alt sınıfların geçersiz kılmasını engelleyebiliriz.

Örnek:

```
class parent{
final void display(){
// İçerik
}
}
```

Final Classlar: Bir class’ı “final” olarak belirttiğimizde, bu sınıfın alt class oluşturulmasını engelleyebiliriz.

Örnek:

```
Final class örnek{
// İçerik
}
```

Super Anahtar Kelimesi Nedir?

- Alt sınıfdan bir üst sınıf için erişim sağlamak için kullanılır. Alt Sınıf, üst sınıfı miras aldığında “super” anahtar kelimesini kullanarak üst sınıf özelliklerine erişim sağlayabilir.

- Üst sınıfdan erişim sağlayabildiğimiz yapılar method olabilir constructor olabilir veya değişkenlerde olabilir.

Örnek:

```
class mevsimler{
   String mevsim;

   Mevsim(String mevsim) {
      this.mevsim =mevsim;
```

```
    }
    void start() {
        System.out.println(mevsim+  " mevsimi başladı.");
    }
}

class kis extends  mevsimler{
    kis() {
        super("Kış");
    }

    @Override
    void start() {
        super.start  ();
        System.out.println("Karlı  ve hastalıklı günler başladı ☺");
    }
}

public class Main {
    public static void main(String[] args) {
        kis kis1= new kis();
        kis1.start ();
    }
}
```

Böylelikle küçük bir örnek ile pekiştirmiş olduk super anahtar kelimesini.

Overloading Ne demek ve Nasıl Kullanılır?

- Herkesin bildiği ve halk ağzında da dolaşan söylem “aşırı yüklenme”.

- Tek bir class içinde olduğumuzu düşünelim ve burada bir çok method olsun **aynı isme** sahip parametresiz veya parametreli olacak şekilde bir işlevi farklı şekillerde gerçekleştirmek için kullanılır.

- Okul Hayatında herkese öğretilen ve klasikleşmiş bir örnek düşünelim bu ne olabilir? Tabi ki kullanıcıdan 2 tane sayı alıp bu methodlara dört işlem uygulatmak olacaktır.

  Örnek:

```
class hesapMakinesi{

   int topla(int a, int b) {
      return a + b;
   }

   int topla(int a, int b, int c) {
      return a + b + c;
```

```
    }

    double topla(double a, double b) {
        return a + b;
    }
}

public class Main {
    public static void main(String[] args) {
        Hesaplayici hesaplayici = new Hesaplayici();
        System.out.println(hesaplayici.topla(2, 3));
        System.out.println(hesaplayici.topla(1, 2, 3));
System.out.println(hesaplayici.topla(2.5, 3.5));
    }
}
```

Burada açıklayıcı olacak şekilde örnek kodumuz ile birlikte overloading başlığını da bitirmiş olduk.

Overriding Nedir ve Nasıl Kullanılır?

- Öncelikle Overriding kavramı da polymorphism ile ilişkili'dir.

- Merkeziyetçi bir yapı sunan override biz yazılımcıları karmaşık yapıdan uzaklaştırıyor.

- Şöyle başlayabiliriz ilk olarak methodum parent class'da yani (üst, super class'da) olacak şekilde yapıyı tasarlamalıyım.

- Ardından alt ve üst class meselesi olduğu için üst class'ı alt class'da extends etmeliyim.

- Bizim burada amacımız methodun;
  - İsmi
  - Parametreleri
  - İmzası
  - Dönüş türü(Return type)

  Aynı olmasına dikkat etmeliyiz.

- Override işlemi aynı classta değil de farklı bir alt class'da gerçekleştirilebilir.

- Final methodlar override edilemez.

- Static olan methodlar override edilemez.

- Erişim belirteci private olan methodlar override edilemez.

- Constructor’lar override edilemez.

Örnek;

```
class sehirler {
    public void tanitim() {
        System.out.println("Bu  bir şehir ☺");
    }
}

class istanbul extends sehirler{
    @Override
    public void tanitim() {
        System.out.println("İstanbul,  Türkiye'nin metropol şehridir.");
    }
```

```
}


class van extends sehirler{
    @Override
    public void tanitim() {
        System.out.println("Van, Türkiye'nin en doğusunda bulunan bir şehridir.");
    }
}



public class  Main{
    public static void main(String[] args) {
        sehirler sehir1 = new sehirler();
        sehirler sehir2 = new istanbul();
        sehirler sehir3 = new van();
    }
}
```

Bu şekilde de Overriding kavramına değinmiş olduk.

ÖZGÜN ÖRNEK;

```
public class Calisan {
	private String ad;
	private String soyad;
	public static int id;
	public static int maas;

	public static void id() {
		System.out.println("İd  Değeri: "+id);
	}
	public static void maas() {
		System.out.println("Maaş  Değeri: "+maas);
	}

	public void calisanBilgileri() {
		System.out.println("İsminiz:  "+getAd());
		System.out.println("Soyisminiz:  "+getSoyad());

		// ***2. yol*** static ile ekrana çıktı çıkarma yöntemi.
		Calisan.id();
		Calisan.maas();
	}


	public String getAd() {
		return ad;
	}
	public void setAd(String ad) {
		this.ad = ad;
```

```
	}
	public String getSoyad() {
		return soyad;
	}
	public void setSoyad(String soyad) {
		this.soyad = soyad;
	}
}


import java.util.Scanner;

public class test {

	public static void main(String[] args) {

		Calisan calisan1=new Calisan();

		Scanner scanner=new Scanner(System.in);

		System.out.println("*******Çalışan Bilgileri Uygulamasına
Hoşgeldiniz*******\n");

		System.out.print("İsminizi Giriniz:");
		String isim=scanner.nextLine();

		System.out.print("Soyisminiz: ");
		String soyisim=scanner.nextLine();

		calisan1.setAd(isim);
```

```
                calisan1.setSoyad(soyisim);

                System.out.print("Maaş  Değeriniz:");
                int maas=scanner.nextInt();

                System.out.print("İd  Ddeğeriniz:");
                int id=scanner.nextInt();

                System.out.println("----------------------------------------");

                //Bu kod yazılmaz ise kullanıcıdan almış olduğum  değeri "Calisan" class'ı içine
atayamam.
                Calisan.id=id;
                Calisan.maas=maas;



                // ***1. yol*** static ile ekrana çıktı çıkarma yöntemi
                //Calisan.id();
                //Calisan.maas();

                calisan1.calisanBilgileri();
        }
}
```

Buraya kadar sabredip okuduğunuz için Teşekkür ederim.