Nesne Tabanlı Programlama I

Öğr. Gör. Dr. Aysun ALTIKARDEŞ

AMAÇ

- Sınıf
- Paket
- Kapsülleme
- Kalıtım
- Çok Biçimlilik

üzerinde önemle durularak, temel programlama bilgilerinin kazandırılması amaçlanmıştır.

Kaynak: Yrd.Doç.Dr. Erbil AKBAY Ders Notları

Nesne Yönelimli Programlama ve Temel Kavramlar

Java dilinde modül veya altprogramlar, sınıf (class) ve yöntemler'den (method) oluşur. Nesne yönelimli programlamanın (Object Oriented Programming – OOP) temelini de, sınıf ve bu sınıfların örneklendirilmiş hali olan nesneler oluşturur.

UML Diyagramları

Nesne yönelimli programlama dillerinde analiz ve tasarım sürecinde modelleme için, çoğunlukla diyagramlardan oluşan UML (Unified Modeling Language – Tümleştirilmiş Modelleme Dili) dilinden yararlanılır. Bir sınıfın UML gösterimi şekildeki gibi üç bölümü olan bir kutu biçimindedir. En üstteki bölümde, sınıfın adı bulunur. Ortada sınıfın özellikleri yer alır. Alttaki kısmında ise, yöntem ve işlemlerle ilgili tanımlar bulunur.

Sınıf (Class) Kavramı

Sınıf, birşeyin soyut özellik ve davranışlarını tanımlar. Örneğin “Araba” sınıfının, tüm arabalarda ortak olarak görülen özellik (kapasite, renk, motor gücü vb.) ve davranışları (çalışmak, durmak) içermesi beklenir. Özellikler (Properties) ile davranışları tanımlayan yöntemlere (methods) topluca sınıf üyeleri (Class members) denir. Sınıflar nesne yönelimli programlamanın modülerliğini sağlayan en önemli öğeler konumundadır. Sınıf tanımlamak için aşağıdaki yapı kullanılır.

```
class Sınıf_adı
{
……../// Metot ve işlemler
}
```

Nesne (Object) Kavramı

Nesne yönelimli programlamada, nesneler bir sınıfı temel alarak oluşturulur. Bir sınıfın bir nesne oluşturma işlemine örnekleme (instantiating) denir. Sınıflar, temelde yalnızca nesne tanımlamalarıdır. Hafızada yer almaları ancak örneklendirilmeleri ile mümkündür. Bir başka ifade ile sınıflar örneklendirildiklerinde isimleri nesne olur diyebiliriz.

Nesne yönelimli programlamada her nesne kendi içinde bağımsız bir bütündür. Bir nesnenin diğer nesneler ile ilişkisi mesaj göndermekten ibarettir. Her nesne kendi ile ilgili verileri kendi içinde saklar, bazı verilere dışarıdan erişime izin vermeyebilir.

Nesne (Object) Kavramı

Bir sınıftan **new** komutu ile birlikte bir kopya oluşturulduğunda bu kopyaya nesne (object) denir. Bir sınıftan aynı anda, birbirinden bağımsız birçok sayıda nesne oluşturulabilir.

Nesne tanımlamak için aşağıdaki yapı kullanılır:

Sınıf_adı Nesne_adı= new Sınıf_adı();

Örneğin Araba sınıfından kamyon adlı bir nesne tanımlamak için aşağıdaki yapı kullanılır:

```
Araba kamyon = new Araba();
```

Bir nesne değişkeninin o an için hiçbir nesneyi işaret etmediğini belirtmek için null değeri kullanılır.

```
Kamyon=null;
…
if (kamyon!=null) kamyon .calis();
```

Nesne Üyelerine Erişim Belirteçleri

Bir sınıf veya nesnenin üyelerine erişim için dört etiket kullanılır.

Public (Açık): Public üyelere programdaki herhangi bir fonksiyon tarafından erişilebilir. Dolayısıyla public üye veya alanlara farklı paket ve sınıflardan erişilebilir.
Private (Özel): Bir sınıfın içinde tanımlanan private üyeye sadece o sınıfın içinden erişilebilir. Dolayısıyla private üye veya alanlara, aynı veya farklı paketten, farklı sınıflardan erişilemez.
Protected (Korumalı): Protected tanımlanmış bir üyeye aynı paket ve sınıf içerisinden erişilebilir fakat paket dışarısından yalnız kalıtım ile oluşturulmuş alt sınıflar erişilebilir.
Default: Bu üç belirteçten (public, private, protected) hiçbiri kullanılmamışsa default olarak tanımlanmış demektir. Default üyelere sadece tanımlandığı paket içerisinden erişilebilir.

Nesne Üyelerine Erişim Belirteçleri

Erişebilme	Public	Protected	Default	Private
Aynı sınıf(class) içerisinden	Evet	Evet	Evet	Evet
Aynı paket içersindeki alt-sınıftan (sub-class)	Evet	Evet	Evet	Hayır
Aynı paket içerisindeki sınıftan	Evet	Evet	Evet	Hayır
Farklı paket içerisindeki alt-sınıftan (sub-class)	Evet	Evet (Türemeyle)	Hayır	Hayır
Farklı paket içerisindeki farklı sınıftan	Evet	Hayır	Hayır	Hayır

Yöntem (Method) Kavramı

Bir nesnenin yeteneklerine (kabiliyetlerine) yöntem veya metot adı verilir. Her bir yöntemi nesnenin yapabileceği bir davranışı simgeler. ÖRNEĞİN Kamyon, Araba sınıfına ait bir nesne olarak, o sınıfta tanımlı davranışları sergiler, yani çalışır, durur. Diğer bir ifade ile "çalışmak" Kamyon'un bir yöntemidir. Bir metot tanımlamak için aşağıdaki yapı kullanılır:

Nesne.Metot_adı();

Yöntem (Method) Kavramı

Örnek:

```
Class Ornek
{
public static void main(String [ ] args)
{
String ad;
int uz;
ad=new String(“Bulent”);
uz=ad.length();
System.out.println(ad  + “kelimesinin karakter uzunluğu: “ + uz);
```

Kalıtım (Inheritance) Kavramı

Miras alma olarak da isimlendirilen kalıtım eğer bir sınıf, bir üst (taban veya ebeveyn de denilebilir) sınıftan türetilirse yeni sınıf (alt veya çocuk sınıf olarak adlandırılabilir) bu üst sınıfın bütün özelliklerine ve metotlarına sahiptir.

Kalıtım ile özellikleri miras alan sınıf, atasının özelliklerini taşır. Bu durumda miras alınan sınıfa üst sınıf (super-class), miras alan sınıfa ise alt sınıf (sub-class) adı verilir.

Her alt sınıf ilerde bir üst sınıf olma adayıdır.

Bir alt-sınıf, üst-sınıfından taşıdığı özelliklere ve işlevlere ek olarak; kendine ait özellikleri ve işlevleri içerebilir.
Bir alt-sınıf aynı zamanda, üst-sınıfından taşıdığı işlevleri değiştirebilir.

Kapsülleme (Encapsulation) Kavramı

Farklı kaynaklarda paketleme veya sarmalama olarak da isimlendirilen kapsülleme, bir sınıfın içeriğinin, onun üyelerini kullananlar tarafından bilinmesine gerek duymadan sadece metodun verdiği hizmetin gösterilmesi işlemidir.
Her nesne, belli verileri tutar ve belli işlevler görür. Bir sınıf aslında başka sınıfların kullanımı için çeşitli özellik ve metotlar barındıran bir yapıdır. Ancak bir sınıftaki bütün özellik ve metotların dışarıdan bilinmesi veya kullanılması gerekmez. Hatta bazı özellik ve metotlara erişim sınıfın sağlamlığı açısından tehlikeli olabilir. Kapsülleme ile bir sınıf, kendi iç bütünlüğünü gizleyebilir ve koruyabilir. Bir sınıfın dışarıdan sadece gereken özellik ve metotlarıyla görülmesi ayrıca basitlik de sağlamaktadır. Kapsülleme, bir sınıfı kullanacak kişinin, ilgilendiği üyelerle sadece sınıfın kendisini ilgilendiren üyeleri bir arada görmesini de engeller.

```
class TV{
        private int boy;
        private int en;
        private int yukseklik;
        public void setUz(int p)
        {boy=p;}
        public void setEn(int p)
        {en=p;}
        public void setYuk(int  p)
        {yukseklik=p;}
        public int SesAyari()
        {return (boy*en*yukseklik);}
}
public class TvSesi {
        public static void main(String[] args) {
        TV obl=new TV();
        double ses;
        obl.setUz(120);
        obl.setEn(100);
        obl.setYuk(90);
        ses=obl.SesAyari();
        System.out.println("Ses Ayarı..:" + ses);}}
```

Çok Biçimlilik (Polymorphism) Kavramı

Çok biçimlilik, bir nesnenin davranış şekillerini duruma göre değiştirebilme yeteneği, başka bir tanıma göre ise, nesnelerin içeride farklı çalışmalarına rağmen dışarıdan aynı biçimde görülmelerine verilen addır.

```
class Hayvan{
public void Konus(){
System.out.println("Ben bir hayvanım.");
}}
class Inek extends Hayvan{
public void Konus(){
System.out.println("MÖÖ");
}}
class Kedi extends Hayvan{
public void Konus(){
System.out.println("Miyav");
}}
class Kopek extends Hayvan{
public void Konus(){
System.out.println("Hav Hav");
}}
public class Alem {
public static void main(String [] args){
Hayvan[]a=new Hayvan [3];
int indis;
a[0]=new Kedi();
a[1] =new Kopek();
a[2]=new Inek ();
for (indis=0; indis <a.length;indis++){
a[indis].Konus();
}
}}
```

ALT PROGRAM VE FONKSİYONLAR

Büyük programların, alt programlara ayrılması (küçük program parçacıkları haline getirilmesi), programın denetimi başta olmak üzere yazımını, anlaşılabilirliğini, hata takibini ve program üzerinde olası değişiklikleri kolaylaştırır ve programa modülerlik kazandırır.

Alt program (Void Yöntem)

Bir programın içinde aynı işi gören bir grup deyimi programın çeşitli yerlerinde tekrar tekrar yazıp kullanmak yerine bu deyimlerden oluşan program parçalarını kullanmak hem kodlama tekrarını önler, hem de programın anlaşılabilirliğini artırır.
Bir alt program, bir ana program ya da alt program tarafından çağrılan ve kendi içinde bir bütün oluşturan program parçasıdır.
Java dilinde alt program ve fonksiyon kavramlarının karşılığı, metot tanımlamalarıdır.
Metotlar parametreli ya da parametresiz tanımlanabilir.
*Alt programlarda geriye dönüş değeri olmadığını belirtmek için **void** deyimi kullanılır.

Alt program (Void Yöntem)

```
class alt program_adı{
void metot_adı(parametre listesi)
{
Komut ve ifadeler;
}}
```

Alt program (Void Yöntem)

Örnek: Ekrana yazılan her mesajdan sonra digital imzanızı gösteren programı yazınız.(Digital imza olarak ad soyad gösterilecektir.)

```
class Imzanesnesi{
void imza(){
System.out.println("Nevzat Çelik");
}}
public class Anaprogram {
public static void main(String[] args) {
Imzanesnesi nesne=new Imzanesnesi();
System.out.println("İlk mesaj...");
nesne.imza();
System.out.println("İkinci mesaj...");
nesne.imza();
}}
```

Ekran Çıktısı:

```
İlk mesaj...
Nevzat Çelik
İkinci mesaj...
Nevzat Çelik
```

Fonksiyon (Function)

Bir alt program sadece komutları işlerken, fonksiyonlar işletilen komutlara ek olarak geriye bir değer gönderirler yani sonuç üretirler.
*Geri dönüş değeri **return** deyiminin yanında belirtilir. Ayrıca return deyimi, fonksiyonun çalışmasını sonlandırmak amacıyla da kullanılabilir.

Fonksiyon (Function)

```
Dönüş Tipi fonksiyon_adı [Parametreler]
{
        İfade ve işlemler;
}
```

Fonksiyon (Function)

Örnek: 0'dan 40'a kadar olan sayılar içerisinde çift olanları ekranda gösteren programı yazınız. (Sayının çift mi tek mi olduğu karar isimli fonksiyon ile belirlenecektir.)

```
class Cift{
boolean karar(int x){
if ((x%2) ==0) return true;
else return  false;
}}
public class Anaprogram {
public static void main(String[] args) {
Cift e =new Cift ();
for (int i=0;i<=40;i++){
if(e.karar(i)) System.out.println(i+ "\t");
}
}}
```

Fonksiyon (Function)

Örnek: Klavyeden girilen Fahrenayt (Fº) sıcaklığını dereceye (C º) çeviren programı yazınız. (C=(F-32)/1.8)

```
import java.util.Scanner;
public class Anaprogram {
public static void main(String[] args) {
derece d=new derece();
Scanner tara=new Scanner(System.in);
double f;
System.out.println("Fahrenayt sıcaklığını giriniz:");
f=tara.nextDouble();
System.out.println(d.cevir(f) + " derecedir.");
}}

class derece{
double cevir(double n) {
return ((n-32)/1.8);
}}
```

Yöntemlere Parametre Aktarımı

Ana program ve altprogramlarda kullanılan değişkenler birbirinden bağımsızdır. Yani ana programdaki bir değişkenin içeriği herhangi bir alt program tarafından bilinmez ve kullanılmaz.

Bir altprogramı çağırırken yapılacak işlemleri belli değerlere göre gerçekleştirmesini isteyebilir. Veya ana programdaki bazı değişkenlerin altprogram tarafından da kullanılmasını isteyebiliriz. Bu gibi durumlarda ana program ile alt program arasında bilgi paylaşımı gerçekleştirecek parametre aktarımını sağlamamız gerekir.

Parametreler, metot ismini takip eden parantez içlerine yazılır. Parametre geçişi iki şekilde yapılabilir. Birincisi verinin doğrudan değerinin aktarılması ki buna **değeri ile aktarma (call-by-value);** ikincisi ise verinin adresinin aktarılması ki buna **referansı ile aktarma (call-by-reference)** denir.

Parametreli aktarımda değişkenler ana program ve alt program / fonksiyon içinde ayrı ayrı tanımlanır. Alt program kendisine gönderilen bu parametreleri alır, gerekli işlemleri yaptıktan sonra sonucu çağıran programa (ana programa) aktarır.

Değeri ile aktarma örneği

```
class Test {
void Altprogram(int i, int j){
i=i+j;
j=-j;
}}
public class DegeriileAktarma {
public static void main(String[] args) {
Test ob=new Test();
int a=15, b=20;
System.out.println("a ve b değişkenlerini çağırmadan önce: " + a + " " + b);
ob.Altprogram(a,b);
System.out.println("a ve b, değeri ile aktarıldıktan sonra: " + a + " " + b);
}}
```

Ekran Çıktısı:

a ve b değişkenlerini çağırmadan önce: 15 20
a ve b, değeri ile aktarıldıktan sonra: 15 20

Referansı ile aktarma örneği

```
class Test {
int a,b;
public Test(int i, int j){
            a=i;
            b=j;}
void change(Test ob){
            ob.a=ob.a + ob.b;
            ob.b=-ob.b;}
}
public class ReferansiileAktarma {
public static void main(String[] args) {
Test ob=new Test(15,20);
System.out.println("ob.a ve ob.b çağırılmadan önce: " + ob.a + " " + ob.b);
ob.change(ob);
System.out.println("ob.a ve ob.b çağırıldıktan sonra : " + ob.a + " " + ob.b);
}}
```

Ekran Çıktısı:

ob.a ve ob.b çağırılmadan önce: 15 20
ob.a ve ob.b çağırıldıktan sonra : 35 -20

Özyinelemeli Fonksiyon (Recursive Function)

Kendini doğrudan veya dolaylı olarak çağıran fonksiyonlara **özyinelemeli fonksiyonlar** adı verilir.

Örnek: Dışarıdan girilen sayının faktöryelini alan programı özyinelemeli fonksiyonu kullanarak yapınız.

```
import java.util.Scanner;
public class fakt {
static int fakt(int i){
if (i<=1) return (1);
else return (i*fakt(i-1));
}
public static void main(String[] args) {
Scanner tara=new Scanner(System.in);
int n;
System.out.println("Faktöryeli alınacak sayıyı giriniz");
n=tara.nextInt();
if (n<0) System.out.println("Negatif sayı girmeyiniz");
else System.out.println(n + "!=" + fakt(n));
}}
```

Ekran Çıktısı:

```
Faktöryeli alınacak sayıyı giriniz
5
5!=120
```

Örnek: Her elemanı kendisinden önceki iki elemanının toplamı şeklinde ifade edilen fibinocci serisinin n. eleman değerinin serisini özyinelemeli fonksiyon ile bulan programı yazınız.

```
import java.util.Scanner;
public class fib {
        static int fib(int i){
                if (i==0 || i==1) return (i);
                else return (fib(i-1) + fib(i-2));
        }
public static void main(String[] args) {
        Scanner tara=new Scanner(System.in);
        int n;
        System.out.println("Eleman değerini giriniz");
        n=tara.nextInt();
        if(n<0) System.out.println("Negatif sayı girmeyiniz");
        else for(int j=0; j<=n; j++)
        System.out.print(fib(j) + "\t");
        }}
```

Ekran Çıktısı:

```
Eleman değerini giriniz
11
0   1   1   2   3   5   8   13   21   34   55   89
```