NESNEYE YÖNELİK PROGRAMLAMA

Referans Tabanlı ve Primitif Türler
Soyutlama ve Kapsülleme

Emir Öztürk

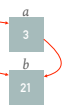
PRİMİTİF TÜRLER

- boolean, byte, char, short, int, long, float, double
- Tanımlandıkları zaman değere sahiptirler.
- Atama durumunda değer değişir.

Primitifler tanımlandığı an değere sahiptirler. Eğer ilk değer verilmezse dil içerisinde belirlenmiş varsayılan değere sahip olurlar.
Herhangi bir değere eşitleme (atama) işlemi gerçekleştirildiğinde değişkenin değeri değişmektedir.

PRİMİTİF TÜRLER

```
int a = 3;
int b = 5;
b = a;
b = 21;
System.out.println(a);
```



Ekran Çıktısı: 3

Her değişken tanımlandığında bellekten yeri alınır.
Daha sonra atama yapıldığında atama yapılan değeri değiştirilmiş olur.
Ekrana yazdırıldığında ilk değişkenin değeri sabit kalır.

REFERANS TÜRÜ

- Sınıflar, Arayüzler, enum, Diziler
- Değer saklamazlar
- Atama durumunda gösterilen alan değişir

Sınıflarda olduğu gibi arayüzler (interface), sayılabilir (enum) türler ve diziler de referans türündedir.
Değişken ismi heapte saklanan değerin bir referansını tutar. Bu sebeple değişkenleri birbirine eşitlemek işaret ettikleri yeri birbirine eşitlemek olduğu için değer değil, gösterdikleri değer değişir.

REFERANS TÜRÜ SINIF ÖRNEĞİ

- Sınıf:

```
class Kutu{
    private int yukseklik;
    private int genislik;
    private int derinlik;
    public void setGenislik(int genislik){
        this.genislik = genislik;
    }
    public int getGenislik(){
        return genislik;
    }
}
```

- Nesne Tanımlama:

```
Kutu kutu = new Kutu();
```

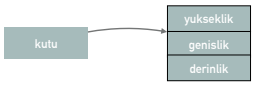
- Alternatif:

```
Kutu kutu;
kutu= new Kutu();
```

Örnek bir Kutu sınıfımız olsun. Bu sınıftan bir örnek oluşturmak için Kutu kutu = new Kutu(); ifadesini kullanmamız gerekmektedir.
Bu ifade “Kutu kutu; ve kutu = new Kutu();” şeklinde iki parçaya bölünebilir. İlk parça bellekte olan yeri gösterecek bir değişkeni tanımlarken, ikinci ifade bellekten bu yerin alınması işlemini gerçekleştirmektedir.

OLUŞTURMA

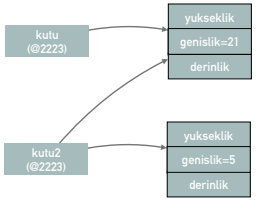
```
Kutu kutu;
kutu= new Kutu();
```



İlk satırda kutu değişkeni oluşturulur fakat değişken henüz alan alınmadığı için bir yeri göstermemektedir.
new ile alma işlemi gerçekleştirilmediğinde bu değişkenin alanlarına erişim hataya sebep olacaktır.

ATAMA

```
Kutu kutu = new Kutu();
kutu.setGenislik(3);
Kutu kutu2 = new Kutu();
kutu2.setGenislik(5);
kutu2 = kutu;
kutu2.setGenislik(21);
System.out.println(kutu.getGenislik());
```



Ekran çıktısı: 21

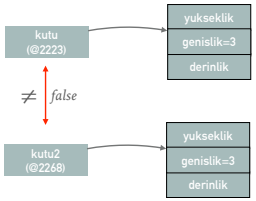
Kutu sınıfından bir nesne oluşturduğumuzda bu nesne heap'te yer alır ve bir referans (kutu) buna işaret eder. Daha sonra ikinci bir değişken tanımladığımızda (kutu2) bu da heap'te başka bir alanı gösterir ve atamalarını buna göre yapar. İki değişken birbirlerine eşitlendiğinde heap'te gösterdikleri alanlar birbirlerine eşitlenmezler. Bunun yerine kutu2= kutu yapıldığında kutu2'nin gösterdiği adres değeri kutu'nun gösterdiği adres değerine eşitlenir (örnek için 2223). Bu sebeple artık kutu2 de heap'te kutu'nun gösterdiği yeri gösterir ve birinde yapılan değişiklik aynı yerleri gösterdikleri için diğerinden okunduğunda gözükecektir.

KARŞILAŞTIRMA

```
int a = 3;
int b = 3;
System.out.println(a==b);
```

true

```
Kutu kutu = new Kutu();
kutu.setGenislik(3);
Kutu kutu2 = new Kutu();
kutu2.setGenislik(3);
System.out.println(kutu == kutu2);
```



Refererans türlerinde değerler karşılaştırılmadıkları için sağdaki durumda eşitlik true döndürmeyecektir.

STRING KARŞILAŞTIRMA

```
public static void main(String[] args) {
    String s1 = new String( original: "abc");
    String s2 = new String( original: "abc");

    if(s1 == s2)System.out.println("Eşit");
    else System.out.println("Değil");

    if(s1.equals(s2))System.out.println("Eşit");
    else System.out.println("Değil");
}
```

Değil
Eşit

```
public static void main(String[] args) {
    String s1 = "abc";
    String s2 = "abc";

    if(s1 == s2)System.out.println("Eşit");
    else System.out.println("Değil");

    if(s1.equals(s2))System.out.println("Eşit");
    else System.out.println("Değil");
}
```

Eşit
Eşit

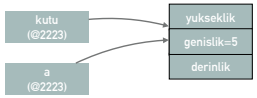
PARAMETRE AKTARIMI

```java
public class Main {
    public static void fonk(int a){
        a = 5;
    }
    public static void main(String[] args) {
        int a = 3;
        fonk(a);
        System.out.println(a);

    }
}
```

Ekran çıktısı: 3

```java
public class Main {
    public static void fonk(Kutu a){
        a.setGenislik(5);
    }
    public static void main(String[] args) {
        Kutu kutu = new Kutu();
        kutu.setGenislik(3);
        fonk(kutu);
        System.out.println(kutu.getGenislik());
    }
}
```



Ekran çıktısı: 5

Parametre aktarımında primitif türlerin bir kopyası fonksiyona alındığı için fonksiyonda yapılan değişiklik main'deki değişkeni etkilemeyecektir.
Nesnelerin fonksiyonlara iletilmesinde referanslarının kopyası alındığı için bu referansların işaret ettiği yerde yapılan değişiklikler diğer metotları da etkilemektedir.

SOYUTLAMA - KAPSÜLLEME

- Soyutlama - Dizayn
- Kapsülleme - İmplementasyon

Soyutlama dizayn seviyesindedir. Bir sınıf tanımladığınızda bu sınıfın detayı tamamen gözardı edilir. Sınıfın bir işi nasıl yaptığı bilinmez.
Kapsülleme ise tamamen implementasyon ile ilgilidir. Nesnelerin iç yapısının gizlenmesine hizmet eder ve dışarıdan erişimine engel olur.
Soyutlama bir nesnenin ne iş yaptığını söyler ama nasıl yaptığını söylemez.
Kapsüllemede ise amaç nasıl yapıldığının iç yapısını gizlemektir.

SOYUTLAMA

- Dizayn
  - Arayüzler
  - Soyut sınıflar

Soyutlamayı dizayn seviyesindeki gizleme sebebiyle ilgilendiren arayüzler ve soyut sınıflardır.

SOYUTLAMA

```
class Kisi{
}
class Karakter extends Kisi{
}
class KotuKarakter extends Kisi{
}
class Basrol extends Kisi{
}
public class Main {
    static void karakterAl(Kisi k){
    }
    public static void main(String[] args) {
        var k = new Kisi();
        var k1 = new Karakter();
        var k2 = new KotuKarakter();
        var b = new Basrol();
        karakterAl(k);
        karakterAl(k1);
        karakterAl(k2);
        karakterAl(b);
    }
}
```

Bir sınıftan kalıtım yoluyla türetilmiş üç farklı sınıf karakterAl() metodu içerisine parametre olarak verilebilmektedir. Burada yapılan soyutlama, karakterAl() metodunun aldığı nesnenin türünün soyutlanması olarak görülmektedir. Verilen sınıfın Kisi tipinde olması karakterAl fonksiyonu için yeterlidir.

KAPSÜLLEME

- İmplementasyon
  - Erişim belirleyicileri
  - İç Yapı

Kapsüllemeyi ilgilendiren ise private protected public gibi erişim belirleyicilerdir.

KAPSÜLLEME

```
class Kuyruk{
    private int[] dizi;
    int sayac;
    Kuyruk(){
        dizi = new int[10];
    }
    void ekle(int eleman){
        if(sayac<10)
            dizi[sayac++]=eleman;
    }
    int al(){
        int deger = dizi[0];
        for(int i=0;i<sayac;i++)
            dizi[i] = dizi[i+1];
        sayac--;
        return deger;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Kuyruk k = new Kuyruk();
        k.ekle( eleman: 3);
        k.ekle( eleman: 5);
        int alinan = k.al();
    }
}
```

```
class Kuyruk{
    private ArrayList<Integer> liste;
    Kuyruk(){
        liste = new ArrayList<Integer>();
    }
    void ekle(int eleman){
        if(liste.size()<10)
            liste.add(eleman);
    }
    int al(){
        int deger = liste.get(0);
        liste.remove( index: 0);
        return deger;
    }
}
```

Kapsülleme işleminde bir sınıfın iç yapısı değiştirildiğinde eğer kullanıcıya sunulan arayüz değiştirilimiyorsa kullanıcıdan bu gizlenmiş olur. Örneğin sol taraftaki kodda diziler ile 10 elemanlı bir kuyruk yapısı geliştirilmişken, sağ tarafta bu kuyruk yapısı bir liste ile gerçekleştirilmiştir. Main metodunda bu sınıfı çağıran kullanıcının hiç bir değişiklik yapmasına gerek kalmadan iki implementasyon arası değişim gerçekleştirilebilir. Oluşacak çıktı ve uygulama aynı olacaktır.