

**Nesne Yönelimli Programlama**

**Kalıtım:** Tasarlanan yeni türlerin tamamen bağımsız yeni türler olduğu düşünülemez. Mutlaka daha önce oluşturulmuş bazı türlerle benzerlikleri bulunur. Bu durumda benzer olan o özellikleri yeniden tanımlamak yerine, var olanları kullanıp kalıttan faydalanarak sadece yeni olan özellikleri yazmak zaman ve maliyetten kazanç sağlayacaktır.

Örnek Java : Kalıtımın desteklendiği Java programlama dilinde bir sıralama yapan sınıf yazıldı. Buna ek olarak sayıların ortalamasının da bulunduğu bir sınıf yazılmak isteniyor. Bu durumda yeni bir sınıf yazarak sıralama yapan sınıftan kalıtım alınırsa sayılar protected olarak tanımlandığı için kalıtlayan sınıflar bu sayılara erişebilir. Dolayısıyla ortalamayı da rahatlıkla bulacaktır. Java’da kalıtım denildiğinde **subclass** ve **superclass** ifadeleri göze çarpar. Aşağıdaki örnekte subclass Mean sınıfı, superclass ise Order sınıfı olmaktadır. Yani kalıtım alınan sınıf superclass kalıtlayan sınıf ise subclass’tır.

```
package SINIFLAR;

/**
 *
 * Order.Java
 */
public class Order {
    protected int[] Sayilar;
    public Order(int []sayilar){
        Sayilar = new int[sayilar.length];
        for(int i=0;i<sayilar.length;i++){
            Sayilar[i] = sayilar[i];
        }
    }
    public Order(int sayi){
        Sayilar = new int[String.valueOf(sayi).length()];
        int indeks=Sayilar.length-1;
        while(sayi > 0) {
            Sayilar[indeks--] = sayi % 10;
            sayi /= 10;
        }
    }
    public int[] Sirala(){
        for (int i = 0; i < Sayilar.length - 1; i++)
        {
            for (int j = 1; j < Sayilar.length - i; j++)
            {
                if (Sayilar[j] < Sayilar[j - 1])
                {
                    int tmp = Sayilar[j - 1];
                    Sayilar[j - 1] = Sayilar[j];
                    Sayilar[j] = tmp;
                }
            }
        }
        int []sirali = new int[Sayilar.length];
        for(int i=0;i<Sayilar.length;i++){
            sirali[i] = Sayilar[i];
        }
        return sirali;
    }
    @Override
    public String toString() {
```

```

        String ekran="";
        Sirala();
        for(int sayi : Sayilar){
            ekran += sayi + " ";
        }
        return ekran;
    }
}

```

```

package SINIFLAR;

/**
 *
 * Mean.Java
 */
public class Mean extends Order{

    public Mean(int[] sayilar) {
        super(sayilar);
    }
    public double Ortalama(){
        double toplam=0;
        for(int sayi : Sayilar){
            toplam+=sayi;
        }
        return toplam/Sayilar.length;
    }
    @Override
    public String toString() {
        return super.toString() + "\nOrtalama: "+Ortalama();
    }
}

```

Yukarıda sadece Mean sınıfı kullanılarak hem sıralama hem de ortalama alınabilir. Sıralama metodu public tanımlandığı için Mean sınıfının da bir alt alanı olacaktır. Burada çok dikkat edilmesi gereken yer Mean nesnesi oluşturduğu zaman arka planda Order nesnesi de oluşacağı için onun yapıcı fonksiyonları kontrol edilmelidir. Örneğin Order sınıfının sadece bir yapıcı metodu vardır ve parametre almaktadır. Bu durumda Mean nesnesi oluşunca bu parametre sağlanmalıdır. Bu da super(sayilar) şeklinde Mean sınıfında sağlanmıştır. Eğer parametre sağlama gerekmiyorsa bu durumda super() şeklinde bir ifadeyi yazmaya gerek yoktur. Çünkü üst sınıfın yapıcı metodu her türlü çağrılmaktadır.

Yukarıdaki örnek tekli kalıtıma (single inheritance) örnektir. Çünkü sadece bir sınıftan kalıtlamıştır. Fakat birden çok sınıftan kalıtlamayı destekleyen programlama dilleri de vardır. Örneğin C++ çoklu kalıtımı (multiple inheritance) destekler. Fakat Java çoklu kalıtımı desteklemez. Java'da çoklu kalıtıma uyacak bir yapı tasarlanmak isteniyorsa yapılacak işlem arayüz tasarımı yapıp arayüz üzerinden kalıtımı gerçekleştirmek. Çünkü Java **çoklu ara yüz** kalıtımını desteklemektedir.

### C Dilinde Kalıtımın Benzetilmesi

Yukarıda Java dilinde yazılan örnek C dili için yazılmak istenirse geçen hafta bahsedilen benzetme yöntemi kullanılacak bu hafta da kalıtımın nasıl benzetilebileceği anlatılacaktır. İlk başta Order.h ve Order.c dosya içerikleri aşağıda verilmiştir.

```

#ifndef ORDER_H
#define ORDER_H
#include "stdio.h"
#include "stdlib.h"

struct ORDER{
    int *Sayilar;

```

```

        int uzunluk;
        int* (*Siralama)(struct ORDER*);
        void (*Yaz)(struct ORDER*);
        void (*Yoket)(struct ORDER*);
};
typedef struct ORDER* Order;

Order OrderOlustur(int sayilar[],int);
Order OrderOlustur_I(int,int);
int* _Siralama(const Order);
void EkranaYaz(const Order);
void OrderYoket(Order);

#endif
#include "Order.h"

Order OrderOlustur_I(int sayi,int uzunluk){
    Order this;
    this = (Order)malloc(sizeof(struct ORDER));
    this->Sayilar = malloc(sizeof(int)*uzunluk);
    this->uzunluk = uzunluk;
    int indeks=uzunluk-1;
    while(sayi > 0) {
        this->Sayilar[indeks--] = sayi % 10;
        sayi /= 10;
    }
    this->Siralama=&_Siralama;
    this->Yaz=&EkranaYaz;
    this->Yoket=&OrderYoket;
    return this;
}
Order OrderOlustur(int sayilar[],int uzunluk){
    Order this;
    this = (Order)malloc(sizeof(struct ORDER));
    this->Sayilar = malloc(sizeof(int)*uzunluk);
    this->uzunluk = uzunluk;
    int i;
    for(i=0;i<uzunluk;i++){
        this->Sayilar[i] = sayilar[i];
    }
    this->Siralama=&_Siralama;
    this->Yaz=&EkranaYaz;
    this->Yoket=&OrderYoket;
    return this;
}
int* _Siralama(const Order this){
    int i,j;
    for (i = 0; i < this->uzunluk - 1; i++){
        for (j = 1; j < this->uzunluk - i; j++){
            if (this->Sayilar[j] < this->Sayilar[j - 1]){
                int tmp = this->Sayilar[j - 1];
                this->Sayilar[j - 1] = this->Sayilar[j];
                this->Sayilar[j] = tmp;
            }
        }
    }
    return this->Sayilar;
}
void EkranaYaz(const Order this){
    this->Siralama(this);
}

```

```

        int i;
        for (i = 0; i < this ->uzunluk; i++){
            printf("%d ", this->Sayilar[i]);
        }
        printf("\n");
    }
    void OrderYoket(Order this){
        if(this == NULL) return;
        free(this ->Sayilar); // dizi de Heap'te oluşturulmuştu
        free(this);
    }
}

```

Struct içerisinde fonksiyon tanımlanamaz fakat **fonksiyon göstericisi** tanımlanabilir. Burada geçen haftadan farklı olarak yapı içerisinde fonksiyon göstericisi tanımlanmış ve yapının oluşturulduğu fonksiyon içerisinde ilgili fonksiyonlara göstericiler atanmıştır.

C dilinde Kalıtımın benzetilmesi için aşağıdaki gibi Mean yapısı içerisine Order yapısından bir değişken tanımlanmalıdır. İsmi Java diline benzetilmesi için super olarak verilmiştir.

```

#ifndef MEAN_H
#define MEAN_H
#include "Order.h"

struct MEAN{
    Order super;
    double (*Ortalama)(struct MEAN*);
    void (*Yaz)(struct MEAN*);
    void (*Yoket)(struct MEAN*);
};
typedef struct MEAN* Mean;

Mean MeanOlustur(int sayilar[],int);
double _Ortalama(const Mean);
void Yazdir(const Mean);
void MeanYoket(Mean);
#endif

#include "Mean.h"

Mean MeanOlustur(int sayilar[],int uzunluk){
    Mean this;
    this = (Mean)malloc(sizeof(struct MEAN));
    this->super = OrderOlustur(sayilar,uzunluk);
    this->Ortalama = &_Ortalama;
    this->Yaz = &Yazdir;
    this->Yoket = &MeanYoket;
    return this;
}

double _Ortalama(const Mean this){
    double toplam=0;
    int i;
    for(i=0;i< this ->super->uzunluk;i++){
        toplam += this ->super->Sayilar[i];
    }
    return toplam/ this ->super->uzunluk;
}

void Yazdir(const Mean this){
    this->super->Yaz(this->super);
    printf("\n");
    printf("Ortalama:%.2lf\n",m->Ortalama(m));
}

void MeanYoket(Mean this){

```

```

        if(this == NULL)return;
        if(this->super != NULL) this->super->Yoket(this->super);
        free(this);
    }

```

Tekrar aynı fonksiyonlar ve alt elemanlar tanımlanmamış ve kalıtıma benzer olarak super değişkeni yardımıyla Order yapısının elemanlarına erişilmiştir. Bu yapıları test eden kod aşağıda görülebilir.

```

#include "Mean.h"

int main(){
    int dizi[]={16,95,1,18,3,7,10};
    Mean mean = MeanOlustur(dizi,7);
    mean->Yaz(mean);
    mean->Yoket(mean);
    return 0;
}

```

Tabi burada atlanmaması gereken Java'daki kalıtıma benzer olarak C dilindeki test programında mean değişkeninin gösterdiği yapı heap bellek bölgesinde oluşmakta ve içerisindeki super Order göstericisinin de Heap bellek bölgesinde aynı anda oluşmasıdır. Dolayısıyla Mean değişkenini yokeden fonksiyona dikkatli bakıldığında super'in gösterdiği bellek bölgesinin de iade edildiği görülecektir. Bu yapının derlenmesi için gerekli make dosyası içeriği aşağıda verilmiştir.

hepsi: derle calistir

derle:

```

gcc -I ./include/ -o ./lib/Order.o -c ./src/Order.c
gcc -I ./include/ -o ./lib/Mean.o -c ./src/Mean.c
gcc -I ./include/ -o ./bin/Test ./lib/Mean.o ./lib/Order.o ./src/test.c

```

calistir:

```
./bin/Test
```

### Nesne Karşılaştırması

Java'da Sınıflardan üretilen nesneler aslında heap bellek bölgesinde oluşan yerlerini referans olarak gösterirler. Eğer bu nokta gözden kaçırılırsa aşağıdaki gibi bir karşılaştırmada beklenenin tam tersi bir sonuç alınacaktır.

```

public class Sayi {
    private int deger;
    public Sayi(int dgr){
        deger=dgr;
    }

    @Override
    public String toString() {
        return String.valueOf(deger);
    }
}

public static void main(String[] args) {
    Sayi s1 = new Sayi(100);
    Sayi s2 = new Sayi(100);
    if(s1 == s2) System.out.println("Sayılar eşittir");
    else System.out.println("Sayılar eşit değil");
}

```

Yukarıdaki kod bloğunda beklenenin dışında bir sonuç alınmasının sebebi nesnelerin içindeki değerlerin değil de referans karşılaştırması yapılmış olmasıdır. Bunun yerine Sayi sınıfını aşağıdaki gibi yazmak istenen sonucu verecektir.

```

public class Sayi {
    private int deger;

```

```

public Sayi(int dgr){
    deger=dgr;
}

@Override
public boolean equals(Object obj) {
    if (obj == null) { // Karşılaştırılan nesne null olmamalı
        return false;
    }
    if (getClass() != obj.getClass()) { // aynı sınıf nesneleri olmalı
        return false;
    }
    final Sayi sy = (Sayi) obj;

    return this.deger == sy.deger;
}

@Override
public int hashCode() {
    return super.hashCode();
}

@Override
public String toString() {
    return String.valueOf(deger);
}
}

public static void main(String[] args) {
    // TODO code application logic here
    Sayi s1 = new Sayi(100);
    Sayi s2 = new Sayi(100);
    if(s1.equals(s2)) System.out.println("Sayılar eşittir");
    else System.out.println("Sayılar eşit değil");
}

```

Burada açıklanması gereken metotlar equals ve hashCode metotlarıdır. Her ikisi de Object sınıfında bulunduğu için override yapılarak kendi sınıfımıza yeniden tanımlamışızdır. equals metodu iki nesnenin nasıl eşit olabileceğinin cevabıdır. hashCode metodu ise Java'da bazı koleksiyon sınıflarında örneğin Hash Tablosu, nesnenin hangi yere koyulacağı hashCode döndürdüğü değer ile belirlenir. Eğer hashcode iyi yazılmaz ise aynı iki nesnenin hashcode farklı dönebilme ihtimalinden dolayı iki nesne eşit olmadığı kabul edilip hata yapılacaktır.

Önemli: Hashcode'larının aynı olması nesnelerin aynı olduğu anlamına gelmeyeceği gibi farklı nesnelerin farklı hashcode'lara sahip olma gibi bir zorunluluğu da yoktur. **Fakat equals metodunun true döndüğü nesneler mutlaka aynı hashcode'a sahip olmalıdırlar.**

Aşağıdaki hashCode metodu incelendiğinde nesnelerin hash değerleri sayının 101 sayısına bölümünden kalan değer ile belirleniyor. Dolayısıyla 101 ve 0 değerlerine sahip olan iki farklı sayı nesnesi aynı hash değere sahip olacaklardır. Fakat bu onların aynı nesne olduklarını göstermez. Zaten equals metodu ile test yapıldığında farklı sayılar olduğu görülecektir.

```

...
@Override
public int hashCode() {
    return deger % 101;
}
...

public static void main(String[] args) {
    Sayi s1 = new Sayi(0);
    Sayi s2 = new Sayi(101);
    System.out.println(s1.hashCode());
}

```

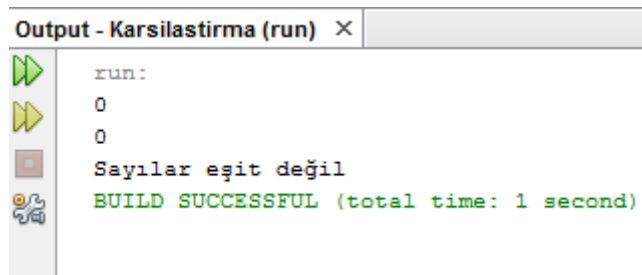
```

System.out.println(s2.hashCode());

if(s1.equals(s2)) System.out.println("Sayılar eşittir");
else System.out.println("Sayılar eşit değil");
}

```

Ekran Çıktısı



```

run:
0
0
Sayılar eşit değil
BUILD SUCCESSFUL (total time: 1 second)

```

== operatörü yerine equals kullanmamızın nedeni Java'da operator overloading yoktur. Dolayısıyla == operatörü kullanıldığı sürece referans karşılaştırması yapacaktır. Fakat C++'ta durum aynı değildir. C++ operator overloading olduğu için == operatörü yeniden tanımlandığı zaman yapılan karşılaştırma doğru bir karşılaştırma olacaktır. C dilinde de operatör overloading olmadığı için Java tarzı bir yazıma benzetilebilir.

```

#ifndef SAYI_H
#define SAYI_H
#include "stdio.h"
#include "stdlib.h"
typedef enum BOOL { false, true}bool;
struct SAYI{
    int deger;
    bool (*equals)(struct SAYI*,struct SAYI*);
    void (*Yoket)(struct SAYI*);
};
typedef struct SAYI* Sayi;

Sayi SayiOlustur(int);
bool Equals(const Sayi,const Sayi);
void SayiYoket(Sayi);
#endif

#include "Sayi.h"

Sayi SayiOlustur(int dgr){
    Sayi this;
    this = (Sayi)malloc(sizeof(struct SAYI));
    this->deger = dgr;
    this->equals = &Equals;
    this->Yoket = &SayiYoket;
    return this;
}

bool Equals(const Sayi this,const Sayi sag){
    if(this == NULL || sag == NULL) return false;
    if(this->deger == sag->deger)return true;
    else return false;
}

void SayiYoket(Sayi this){
    if(this == NULL) return;
    free(this);
}

#include "Sayi.h"

int main(){

```

<pre>Sayi s1 = SayiOlustur(100); Sayi s2 = SayiOlustur(100); if(s1 == s2) printf("Sayilar esit\n"); //Adres karşılaştırması else printf("Sayilar esit degil\n"); //Değer Karşılaştırması if(s1-&gt;equals(s1,s2)) printf("Sayilar esit\n"); else printf("Sayilar esit degil\n"); s1-&gt;Yoket(s1); s2-&gt;Yoket(s2); return 0; }</pre>
<p>hepsi: derle calistir</p> <p>derle:</p> <pre>gcc -I ./include/ -o ./lib/Sayi.o -c ./src/Sayi.c gcc -I ./include/ -o ./bin/Test ./lib/Sayi.o ./src/Test.c</pre> <p>calistir:</p> <pre>./bin/Test</pre>

**Hazırlayan**  
**Dr. Öğr. Üyesi M. Fatih ADAK**