Nesne – Object

Durum ve davranışı olan herhangi bir varlık bir nesne olarak bilinir.

Örneğin: sandalye, kalem, masa, klavye, bisiklet vb.

Bir köpek bir nesnedir, çünkü renk, isim, cins vb. durumları ve kuyruk sallama, havlama, yemek yeme gibi davranışları vardır.

Fiziksel veya mantıksal herhangi bir şey nesne olabilir.

Nesne, bir sınıfın bir örneği olarak tanımlanabilir.

Bir nesne bir adres içerir ve hafızada yer kaplar.

Nesneler, birbirlerinin verilerinin veya kodlarının ayrıntılarını bilmeden iletişim kurabilir, sadece gerekli olan şey, kabul edilen mesajın türü ve nesneler tarafından döndürülen yanıtın tipidir.

Sınıf- Class

Nesneler topluluğu sınıf olarak adlandırılmaktadır.

Sınıf mantıksal (soyut) bir varlıktır.

Bir sınıf ayrıca, bireysel bir nesne oluşturabileceğiniz bir plan/şablon/taslak olarak da tanımlanabilir.

Sınıf soyut bir varlık olduğu için herhangi bir alanda saklamıyordur.

Kalıtım / Miras – Inheritance

Bir nesne ana nesnenin tüm özelliklerini ve davranışlarını aldığında, miras olarak bilinir.

Kod yeniden kullanılabilirliği sağlar.

Çalışma zamanı polimorfizmini (runtime polymorphism) başarmak için kullanılır.

Çok Biçimlilik – Polymorphism

Bir görev farklı yollarla gerçekleştirildiğinde polimorfizm olarak bilinir.

Örneğin: müşteriyi farklı şekilde ikna etmek, çokgen çizmek kare veya dikdörtgen gibi

Java'da, polimorfizmi başarmak için iki farklı yaklaşım bulunmaktadır.

1. Metot aşırı yükleme (overloading)
2. Metodu geçersiz kılma / çiğneme (overriding)

Örneğin; kedi konuşmak için miyavlarken, köpek havlamaktadır.

Soyutlama – Abstraction

Dahili detayları gizlemek ve işlevselliği göstermek soyutlama olarak bilinir.

Örneğin bir telefon görüşmesinde, iç işleyişi bilmiyoruz.

Java'da soyutlamayı sağlamak için soyut sınıf (abstract class) ve arayüz (interface) yapılarını kullanıyoruz.

Kapsülleme – Encapsulation

Kod ve verileri tek bir birim halinde birleştirmek (veya sarmak) Kapsülleme olarak bilinir.

Örneğin: kapsül, farklı ilaçlarla sarılır.

Bir java sınıfı Kapsülleme örneğidir.

Java çekirdeği tamamen kapsüllü sınıftır. Çünkü tüm veri üyeleri burada özeldir.

Nesne Tabanlı Programlama Bileşenleri Sırayla.

Object – Class – Inheritance – Polymorphism – Abstraction – Encapsulation

Prosedür Tabanlı Programalama Dillerine Göre OOP Avantajı

OOP, Prosedür odaklı dildeki gibi, proje boyutu büyürken kodun büyümesi durumunda yönetilmesi kolay olmadığı yerlerde, geliştirme ve bakım işlemlerini kolaylaştırır.

OOP veri gizleme sağlarken, prosedür dilleri global bir veridir ve her yerden erişilebilir.

OOP gerçek dünya olayını daha etkin bir şekilde simüle etme yeteneği sağlar. NYP kullanıyorsak gerçek dünya probleminin çözümünü sağlayabiliriz.

Nesneye yönelik programlama dili ve nesne tabanlı programlama dili arasındaki fark nedir? Nesne tabanlı programlama dili, kalıtım dışındaki OOP'ların tüm özelliklerini izler. JavaScript ve VBScript, nesne tabanlı programlama dillerinin örnekleridir.

Java İsimlendirme Kuralları

Sınıf, paket, değişken, sabit, yöntem gibi tanımlayıcılara nasıl ad vereceğinize ilişkin bir kuraldır. Ancak bu isim verme kuralları zorunluluk değil bir alışılagelmiş uygulama yaklaşımıdır. Standart Java adlandırma kurallarını kullanarak, kodunuzun daha kolay okunması sağlanabilir. Bir programın okunabilirliği çok önemlidir. Kodun ne yaptığını anlamak için daha az zaman harcandığını gösterir.

Sınıf (class): Büyük harfle başlamalı ve bir isim olmalıdır. String, Color, Button, System, Thread

Arayüz (interface): Büyük harfle başlamalı ve bir sıfat olmalıdır. Runnable, Remote, ActionListener

Yöntem (method): Küçük harfle başlamalı ve bir fiil olmalıdır. actionPerformed(), main(), print()

Değişken (varable): Küçük harfle başlamalıdır. ogrNumarasi, siparisNumarasi

Paket (package): Küçük harfle başlamalıdır. Java, lang, sql, util

Sabitler(constants): Büyük harf olmalıdır. KIRMIZI, SARI, EN_ONCELIKLI

CamelCase (Deve Sırtı) İsimlendirme Kuralı

Java, sınıf, arabirim, yöntem ve değişkeni adlandırmak için deve sırtı sözdizimini izler. Eğer isim iki kelimeyle birleştirilirse, ikinci kelime her zaman büyük harfle başlayacaktır. actionPerfomed(), firstName, ActionEvent, ActionListener

Java'da Nesne ve Sınıf

NYP tekniğinde, nesneleri ve sınıfları kullanarak bir program tasarlayacağız.

Nesne, mantıksal (soyut) varlık olduğu kadar fiziksel de (somut) bir varlık iken, sınıf ise sadece mantıksal bir varlıktır.

Nesne: Durum ve davranışı olan herhangi bir varlıktır. Bir nesne fiziksel veya mantıksal (somut veya soyut) olabilir. Somut nesne örnekleri araba, bilgisayar, kalem. Soyut olarak ise bankacılık sistemi.

Bir nesnenin üç özelliği vardır.

1. Durum (state): Bir nesnenin verilerini (değerini) temsil eder.
2. Davranış (behavior): Mevduat, para çekme vb. gibi bir nesnenin davranışını (işlevselliğini) temsil eder.
3. Kimlik (identity): Nesne kimliği tipik olarak benzersiz bir kimlik ile uygulanır. ID'nin değeri harici kullanıcı tarafından görülmez. Ancak, her bir nesneyi benzersiz bir şekilde tanımlamak için JVM tarafından dahili olarak kullanılır.

Nesne, bir sınıfın örneğidir. Sınıf nesnelerin oluşturulduğu bir şablon veya plandır. Nesne kavramı ile alakalı bazı tanımlamalar: Nesne bir gerçek dünya varlığıdır. Bir çalışma zamanı öğesidir. Durum ve davranışa sahip bir varlıktır.

Sınıf, ortak özelliklere sahip bir nesneler topluluğudur. Sınıf nesnelerin oluşturduğu bir şablon veya planıdır. Mantıksal bir varlıktır. Fiziksel olamaz.

Java'da bir sınıf şunları içerebilir:

1. Alanlar (fields)
2. Yöntemler (methods)
3. Yapıcılar (constructors)
4. Bloklar (blocks)
5. İç içe sınıf ve arayüz (nested class and interface)

Sınıf şablonu

Class <sınıf_ismi>{

Alanlar/özellikler (field);

Yöntemler (method);

}

Örnek Değişken (Instance Variable)

Sınıf içinde oluşturulan, ancak yöntemin dışında olan bir değişken, bir örnek değişkeni olarak bilinir. Örnek değişkeni derleme zamanında bellek almaz. Bir nesne veya örnek oluşturulduğunda çalışma zamanında bellek alır. Bu yüzden örnek değişken olarak bilinir.

new anahtar kelimesi; çalışma zamanında belleği ayırmak için kullanılır. Tüm nesneler Heap bellek alanında bellek alır.

Yöntemler (Methods)

Java'da, bir yöntem, bir nesnenin davranışını açığa çıkarmak için kullanılan bir işlev gibidir. Yöntem kullanmanın avantajları: Kod Yeniden Kullanılabilirliği (Reusability), Kod Optimizasyonu

Örnek Kod Bloğu

```
class Ogrenci{
//özelliklerin (fields) tanımlanması
İnt ogrenciNo;//field veya veri üyesi (data member) veya instance variable
String isim;
//Ogrenci sınıfının içerisinde main yöntemi tanımlanıyor
public static void main(String args[ ]){
//Bir nesne veya örneğin oluşturulması
Ogrenci nesne1=new Ogrenci();
//nesnenin değerlerine ulaşım
System.out.println(nesne1.ogrenciNo); //reference variable üyelere erişme
System.out.println(nesne1.isim);
}
}
```

Nesne ve Sınıf Örneği

Gerçek uygulamalar geliştirirken genellikle sınıflar oluştururuz ve onu başka bir sınıftan kullanırız. Bir yaklaşım sınıfın içerisinde main() yöntemini kullanmaktan daha iyi bir yaklaşımdır. Farklı java dosyalarında veya bir tek java dosyasında birden fazla sınıfa sahip olabiliriz. Tek bir java kaynak dosyasında birden fazla sınıf tanımlarsanız, dosya adını main() yöntemine sahip sınıf adıyla kaydetmek iyi bir fikirdir.

```
class Ogrenci{
int ogrenciNo;
String isim;
}
class TestOgrenci {
public static void main(String args[]){
Ogrenci nesne1=new Ogrenci();
System.out.println(nesne1.ogrenciNo);
System.out.println(nesne1.isim);
}
}
```

Çıktı: 0 (isim değişkeni Null)

Nesneleri Hazırlamanın Farklı Yolları

Java programlama dilinde nesneleri hazırlamak için üç farklı yol bulunmaktadır.

1. Referans değişkeni yardımıyla
2. Yöntem yardımıyla
3. Yapıcı tarafından

```
class Ogrenci{
int ogrenciNo;
String isim;
```

```
}
class TestOgrenci {
public static void main(String args[]){
//Nesneleri oluştur
Ogrenci nesne1=new Ogrenci();
Ogrenci nesne2=new Ogrenci();
//Nesneleri hazırla
nesne1.ogrenciNo=101;
nesne1.isim=”Hüseyin”;
nesne2.ogrenciNo=102;
nesne2.isim=”Kocaman”;
//Verileri yazdır
System.out.println(nesne1.ogrenciNo + “ “+nesne1.ogrenciNo);
System.out.println(nesne2.ogrenciNo+” “+nesne2.isim);
}
}
```

Çıktı: 101 Hüseyin
102 Kocaman

```
class Ogrenci{
int ogrenciNo;
String isim;
void kayitEkle(int ogr, String ad){
ogrenciNo=ogr;
isim=ad;
}
void bilgileriGoster(){System.out.println(ogrenciNo+” “+isim);}
}
class TestOgrenci {
public static void main(String args[]){
Ogrenci nesne1=new Ogrenci();
Ogrenci nesne2=new Ogrenci();
nesne1.kayitEkle(111,”Hüseyin”);
nesne2.kayitEkle(222,”Kocaman”);
nesne1.bilgileriGoster();
nesne2.bilgileriGoster();
}
}
```

Çıktı: 111 Hüseyin
222 Kocaman

```
class Dortgen{
int uzunluk;
int genislik;
void ekle(int u, int g){
uzunluk=u;
genislik=g;
}
void alanHesapla(){System.out.println(uzunluk*genislik);}
}
class TestDortgen{
public static void main(String args[]){
Dortgen nesne1=new Dortgen();
Dortgen nesne2=new Dortgen();
nesne1.ekle(11,5);
nesne2.ekle(3,15);
nesne1.alanHesapla();
nesne2.alanHesapla();
}
}
```

Çıktı: 55
45

Anonim Nesneler (Anonymous Object)

Anonim sadece isimsiz anlamına gelir. Referansı olmayan bir nesne, anonim nesne olarak bilinir. Sadece nesne oluşturma zamanında kullanılabilir. Bir nesneyi yalnızca bir kez kullanmamız gerekiyorsa, anonim bir nesne iyi bir yaklaşımdır.

```
class Hesapla{
void faktöriyel(int n){
int sonuc=1;
for(int i=1;i<=n;i++){
sonuc=sonuc*i;
}
System.out.println("Hesaplanan değer: "+sonuc);
}
public static void main(String args[]){
new Hesapla().faktoriyel(5); //anonim nesnenin metodu çağrılıyor
}
}
```

Çıktı: Hesaplanan değer: 120

Banka Örneği

Bir banka müşterisinin hesap numarası, müşteri adı ve bankada ne kadar parası olduğu bilgileri saklanmaktadır. Müşteri bankaya para yatırabilir veya para çekebilir. Para yatırma ve çekme işlemleri sonucunda hesaptaki para miktarının tutarlı olduğunu gösteren kontrol isimli bir yöntem olmalıdır. Göster metodu müşteri no, adı ve tutar bilgilerini ekrana yazdırmaktadır.

```
class Hesap{
int hesapNumarasi;
String hesapAdi;
float miktar;
void musteriEkle(int hn, String ha, float m){
hesapNumarası=hn;
hesapAdi=ha;
miktar=m;
}
Void paraYatir(float tutar)
{
miktar=miktar+tutar;
System.out.println(tutar+" hesaba yatırıldı");
}
Void paraCek(float tutar)
{
İf(tutar>miktar)
{
System.out.println("Hesapta yeterli bakiye yok");
}
else
{
miktar=miktar-tutar;
System.out.println(tutar+" hesaptan çekildi");
}
}
void kontrol()
{
System.out.println("Hesabın güncel bakiyesi: "+miktar);
}
 Void göster(){
System.out.println(hesapNumarasi+" "+hesapAdi+" "+miktar);
}
public class BankaHesap{
public static void main(String[] args){
Hesap musteri1=new Hesap();
```

```
musteri1.musteriEkle(1, “hüseyin kocaman”,10000);
musteri1.goster();
musteri1.paraYatir(5000);
musteri1.kontrol();
musteri1.paraCek(3000);
musteri1.kontrol();
musteri1.paraCek(3000);
musteri1.goster();
}
}
```

Çıktı:
```
1 hüseyin kocaman 10000.0
5000.0 hesaba yatırıldı
Hesabın güncel bakiyesi 15000.0
3000.0 hesaptan çekildi
Hesabın güncel bakiyesi 12000.0
3000.0 hesaptan çekildi
1 hüseyin kocaman 9000.0
```

Yapıcı Metotlar (Constructor Methods)

Java’da, bir yapıcı/kurucu, yönteme benzer bir kod bloğudur. Nesnenin bir örneği oluşturulduğunda ve nesne için bellek ayrıldığında çağrılır. Nesneyi başlatmak için kullanılan özel bir yöntem türüdür. Yapıcı, Oluşturucu veya Kurucu olarak adlandırılır. Çünkü nesne oluşturma zamanında değerleri oluşturur.

Ne zaman bir kurucu çağrılır?

new() anahtar sözcüğünü kullanarak bir nesne her oluşturulduğunda, en az bir kurucu çağrılır. Varsayılan bir kurucu çağırır.

Java Kurucu Metot Oluşturma Kuralları

Kurucu için tanımlanmış iki kural vardır:

1. Kurucu metot adı, sınıf adıyla aynı olmalıdır.
2. Bir kurucu açık bir dönüş türüne sahip olmaması gerekir.

Bir Java kurucusu soyut, statik, final ve senkronize olamaz. Bir yapıcıyı bildirirken erişim değiştiricilerini kullanabiliriz. Nesne oluşturmayı kontrol eder. Başka bir deyişle, Java’da özel, korumalı, genel veya varsayılan kurucu olabilir.

Java’da iki tür yapıcı metot vardır:

1. Varsayılan kurucu (parametresiz kurucu)
2. Parametreli kurucu

Varsayılan kurucunun amacı, türe bağlı olarak 0, null vb. gibi nesnelere varsayılan değerler sağlamak için kullanılır. Bir sınıf için bir kurucu yazmak gerekli değildir. Çünkü Java derleyiciniz, sınıfınızda hiç yoksa varsayılan bir kurucu oluşturur.

```
class Bike{                                  class Bike{
                        Derleyici ->         Bike(){}
]                                            }
```

Parametreli kurucu, belirli bir sayıda parametresi olan bir kurucuya denir. Farklı nesnelere farklı değerler sağlamak için kullanılır. Bununla birlikte, aynı değerleri de sağlayabilirsiniz.

```
class Ogrenci{
int ogrenciNo;
String isim;
Ogrenci(int ogr, String ad) // Parametreli yapıcı metot
ogrenciNo=ogr;
isim=ad;
}}
```

Kurucu Aşırı Yüklemesi

Java'da, bir kurucu sadece bir yöntem gibidir. Ancak dönüş türü yoktur. Java yöntemleri gibi aşırı yüklenebilir. Java'da aşırı yüklenici, farklı parametre listelerine sahip birden fazla kurucuya sahip olma tekniğidir. Her kurucu farklı bir görev gerçekleştirecek şekilde düzenlenmiştir. Derleyici tarafından listedeki parametrelerin sayısı ve türleri tarafından ayrıştırılır.

```
lass Ogrenci{
int ogrenciNo;
String isim;
Ogrenci(int ogr, String ad) // Parametreli yapıcı metot
ogrenciNo=ogr;
isim=ad;
}
Ogrenci(int ogr){
ogrenciNo=ogr;
}}
```

Parametrelere göre çağrılır. 2 parametre varsa ogr no, ad.. 1 parametre varsa ogr.. hiç yoksa yalnız değişken tanımları yapılır.

Kurucu ve Yöntem Arasındaki Farklar

Bir nesnenin durumunu başlatmak için bir kurucu kullanılır. Bir nesnenin davranışını ortaya çıkarmak için ise bir yöntem kullanılır.

Bir kurucunun bir dönüş türü olmamalıdır. Bir yöntemin bir dönüş türüne sahip olması gerekir.

Kurucu örtülü olarak çalıştırılır. Yöntem açıkça çağrılır.

Java derleyicisi, bir sınıfta herhangi bir kurucunuz yoksa varsayılan kurucu sağlar. Yöntem, derleyici tarafından her durumda sağlanmaz.

Yapıcının adı, sınıf adıyla aynı olmalıdır. Yöntem adı isse, sınıf adıyla aynı olabilir veya olmayabilir.

Static Anahtar Sözcüğü

Temel olarak bellek yönetimi için kullanılır. Değişkenler, yöntemler, bloklar ve iç içe sınıflara static anahtar sözcüğü uygulanabilir. Static anahtar kelimesi sınıfın örneğine ait olmaktan ziyade sınıfa aittir.

Aşağıdakiler statik olabilir:

1. değişken (sınıf değişkeni)
2. yöntem (sınıf yöntemi)

3. blok

4. iç içe geçmiş sınıf

Sabit Değişkenler

Herhangi bir değişkeni statik olarak bildirirseniz, statik değişken olarak bilinir. Statik değişken, tüm nesnelerin ortak özelliklerine (her nesne için benzersiz olmayan) atıfta bulunmak için kullanılabilir.

Örneğin, çalışanların şirket isimlerine, öğrencilerin fakülte adlarına vb.

Bir fakültede 500 öğrenci olduğunu varsayalım. Bu durumda tüm örnek değişkenlerin veri üyeleri, nesne oluşturulduğunda her zaman bellek alacaktır. Tüm öğrencilerin öğrenci numaraları ve isimleri vardır. Ancak tüm öğrencilerin fakülte isimleri aynıdır. Burada, "fakülte adı", tüm nesnelerin ortak özelliğini ifade eder. Statik yaparsak bu alan hafızada sadece bir kez yer alır.

Statik değişken, sınıf yükleme sırasında sınıf alanında sadece bir kez bellek alır. Avantajı ise programınızın belleğini verimli hale getirmesidir.

```
Class Ogenci{
İnt ogrenciNo;
String isim;
Static String fakulteAdi=”Mühendislik”;
Ogrenci(int ogr, String ad){
ogrenciNo=ogr;
isim=ad;
}
Void bilgileriGoster(){System.out.println(ogrenciNo+” “+isim);}
}
Class TestOgrenci{
Public static void main(String args[]){
Ogrenci nesne1=new Ogrenci(111,”Hüseyin”);
Ogrenci nesne2=new Ogrenci(222,”Kocaman”);
Nesne1.bilgileriGoster();
Nesne2.bilgileriGoster();
}}
```

Çıktı: 111 Hüseyin

222 Kocaman

Statik Değişken Kullanmadan Program Sayacı

Bu örnekte, yapıcıda artırılan sayaç adında bir örnek değişken oluşturulmuştur. Örnek değişkeni, nesne oluşturma zamanında belleği aldığından, her bir nesne, örnek değişkeninin kopyasına sahip olacaktır. Arttırılırsa, diğer nesneleri yansıtmaz. Yani her nesne, sayı değişkeninde 1 değerine sahip olacaktır.

```
Class Sayici{
İnt sayac=0;
Sayici(){
Sayaç++;
System.out.println(sayaac);
}
```

```
Public static void main(String args[]){
Sayici nesne1=new Sayici();
Sayici nesne2=new Sayici();
Sayici nesne3=new Sayici();
}}
```

Çıktı: 1
1
1

```
Class Sayici{
Static İnt sayac=0;
Sayici(){
Sayaç++;
System.out.println(sayaac);
}
Public static void main(String args[]){
Sayici nesne1=new Sayici();
Sayici nesne2=new Sayici();
Sayici nesne3=new Sayici();
}}
```

Çıktı: 1
2
3

Statik Yöntem

Herhangi bir yöntemle statik anahtar kelime uygularsanız, statik yöntem olarak bilinir. Statik yöntem, bir sınıfın nesnesi yerine sınıfa aittir. Bir sınıf örneğini oluşturmaya gerek kalmadan statik bir yöntem çağrılabilir. Statik veri üyesine statik bir yöntem erişebilir ve değerini değiştirebilir.

```
Class Ogenci{
İnt ogrenciNo;
String isim;
Static String fakulteAdi=”Mühendislik”;
Static void degistir(){fakulteAdi=”Teknoloji”;}
Ogrenci(int ogr, String ad){
ogrenciNo=ogr;
isim=ad;
}
Void bilgileriGoster(){System.out.println(ogrenciNo+” “+isim+fakulteAdi);}
}
Class TestOgrenci{
Public static void main(String args[]){
```

```
Ogrenci.degistir();
Ogrenci nesne1=new Ogrenci(111,”Hüseyin”);
Ogrenci nesne2=new Ogrenci(222,”Kocaman”);
Nesne1.bilgileriGoster();
Nesne2.bilgileriGoster();
}}
```

Çıktı: 111 Hüseyin Teknoloji
222 Kocaman Teknoloji

```
Class Hesapla{
Static int kupAl(int x){
Return x*x*x;}
Public static void main(String args[]){
İnt sonuc=Hesapla.kupAl(5);
System.out.println(sonuc);
}}
```

Çıktı: 125

Statik Yöntem İçin Kısıtlamalar

Statik yöntem için iki ana kısıtlama vardır:

1. Statik yöntem, statik olmayan veri üyesi kullanamaz veya doğrudan statik olmayan yöntemi çağıramaz.

2. this ve super statik bağlamda kullanılamaz.

```
Class A{
İnt a=40; //statik olarak tanımlanmamış
Public static void main(String args[]){
System.out.println(a);
}}
```

Çıktı: Output: Compile Time Error -> a değişkeni statik tanımlanmadığından

Main() yöntemi neden statik olarak tanımlanmaktadır? Nesnenin statik bir yöntem çağırması gerekmediği içindir. Statik olmayan bir yöntem olsaydı, JVM önce bir nesne oluşturur ve fazladan bellek ayırma sorununa yol açarak main() yöntemini çağırır.

This Anahtar Sözcüğü

Geçerli nesneyi ifade eden bir referans değişkenidir. Kullanım alanlar:

1. Geçerli sınıf örnek değişkenini belirtmek için kullanılır.
2. Mevcut sınıf yöntemini çağırmak için kullanılır. (örtülü)
3. Mevcut sınıf yapıcısını çağırmak için kullanılır.
4. Metot çağrısında bir argüman olarak iletilebilir.
5. Yapıcı çağrısında argüman olarak iletilebilir.

6. Geçerli sınıf örneğini metottan döndürmek için kullanılabilir

Geçerli bir sınıf örnek değişkenini belirtmek için kullanımı:

```
Class Ogenci{
İnt ogrenciNo;
String isim;
Ogrenci(int ogrenciNo, String isim){
This.ogrenciNo=ogrenciNo; //Aynı değişken adı olduğundan burada this kullanıldı
This.isim=isim;
}
Void bilgileriGoster(){System.out.println(ogrenciNo+" "+isim+fakulteAdi);}
}
Class TestOgrenci{
Public static void main(String args[]){
Ogrenci.degistir();
Ogrenci nesne1=new Ogrenci(111,"Hüseyin");
Ogrenci nesne2=new Ogrenci(222,"Kocaman");
Nesne1.bilgileriGoster();
Nesne2.bilgileriGoster();
}}
```

Mevcut sınıf yöntemini çağırmak için kullanımı (örtülü):

```
Class A{                                  class A{
Void m(){}                                void m(){}
M();                                      void n(){
}                                         this.m();}
Public static void main(string args[]){   Public static void main(string args[]){
New A().n();                              New A().n();
}}                                        }}
```

```
Class A{
Void m(){System.out.println("merhaba m");
Void n(){
System.out.println("merhaba n");
This.m(); //m() yazmak ile this.m() yazmak arasında fark yok
}}
Class Test{
Public static void main(Strings args[]){
A a=new A();
a.n();
}}
```

Çıktı: merhaba n

Merhaba m

Mevcut yapıcıyı çağırmak için kullanımı:

```
Class A{
A(){
System.out.println(“merhaba a”);}
A(int x){
This();
System.out.println(x);
}}
Class Test{
Public static void main(String args[]){
A a=new A(10);
}}
```

Çıktı: merhaba a

10

```
Class A{
A(){
This(5);
System.out.println(“merhaba a”);}
A(int x){
System.out.println(x);
}}
Class Test{
Public static void main(String args[]){
A a=new A();
}}
```

Çıktı: 5

merhaba a;

```
Class Ogenci{
İnt ogrenciNo;
String isim,ders;
Float burs;
Ogrenci(int ogrenciNo, String isim, String ders){
This.ogrenciNo=ogrenciNo;
This.isim=isim;
This.ders=ders;
}
```

```
Ogrenci(int OgrenciNo, String isim, String ders, float burs){
This(ogrenciNo, isim, ders); //öncelikle çağrılmalı yoksa hata verir
This.burs=burs;}
Void bilgileriGoster(){System.out.println(ogrenciNo+" "+isim+" "+ders+" "+burs);}
}
Class TestOgrenci{
Public static void main(String args[]){
Ogrenci nesne1=new Ogrenci(111,"Hüseyin","java");
Ogrenci nesne2=new Ogrenci(222,"Kocaman","java",6000f);
Nesne1.bilgileriGoster();
Nesne2.bilgileriGoster();
}}
```

Çıktı: 111 Hüseyin java
222 Kocaman java 6000.0

Kalıtım / Miras (Inheritance)

Kalıtım, bir nesnenin bir üst nesneden tüm özelliklerini ve davranışlarını edindiği bir mekanizmadır. NYP'nin önemli bir parçasıdır. Java'daki mirasın ardındaki fikir, mevcut sınıflara dayanan yeni sınıflar oluşturabilmenizdir. Var olan bir sınıftan miras aldığınızda, üst sınıfın metotlarını ve alanlarını yeniden kullanabilirsiniz. Ayrıca, mevcut sınıfınızda yeni metotlar ve alanlar da ekleyebilirsiniz. Miras, ebeveyn-çocuk ilişkisi olarak da bilinen IS-A ilişkisini temsil eder.

Java'da metot geçersiz kılma için (bu yolla çalışma zamanı polimorfizmi elde edilebilir) ve kod yeniden kullanılabilirliği için miras kullanılır.

Kalıtımda Kullanılan Terimler

Sınıf (Class): Sınıf, ortak özelliklere sahip nesneler grubudur. Nesnelerin oluşturulduğu bir şablon veya plandır.

Alt Sınıf / Çocuk Sınıfı (Sub Class / Child Class): Diğer sınıfı devralan bir sınıftır. Ayrıca türetilmiş bir sınıf veya alt sınıf olarak da adlandırılır.

Süper Sınıf / Üst Sınıf(Super Class / Parent Class): Bir alt sınıfın özellikleri miras aldığı sınıftır. Ayrıca bir temel sınıf veya bir üst sınıf olarak da adlandırılır.

Yeniden Kullanılabilirlik (Reusability): Adın belirttiği gibi, yeni bir sınıf oluştururken var olan sınıfın alanlarını ve metotlarını yeniden kullanmanızı kolaylaştıran bir mekanizmadır. Önceki sınıfta önceden tanımlanan aynı alanları ve metotları kullanabilirsiniz.

extends Anahtar Sözcüğü (Kaltım Sözdizimi)

Bu anahtar sözcük, var olan bir sınıftan türeyen yeni bir sınıf oluşturduğunuzu gösterir. "Uzanır"'ın anlamı, işlevselliği arttırmaktır. Java'da, miras alınan bir sınıf ebeveyn veya üst sınıf olarak adlandırılır ve yeni bir sınıf çocuk veya alt sınıf olarak adlandırılır.

```
Class Subclass-ismi extends Superclass-ismi[
//metotlar ve alanlar
}

Class Calisan{
Float maas=40000;}
```

```
Class Programci extends Calisan{
İnt ekUcret=10000;
Public static void main(String args[]){
Programci p=new Programci();
System.out.println("Programcının ücreti:"+p.maas);
System.out.println("Programcının ek ücreti:"+p.ekUcret);
}}
```

Çıktı: Programcının maaşı: 40000.0
Programcının ek ücreti: 10000.0

Burada gösterildiği gibi Programci alt sınıftır ve Calisan süper sınıftır. İki sınıf arasındaki ilişki, Programci IS-A Calisan'ıdır. Bu, Programci'nin bir tür Calisan olduğu anlamına gelir.

Kalıtım Türleri:

Sınıf bazında üç tip kalıtım olabilir. Tek, çok düzeyli ve hiyerarşik.

Tek: ClassA ← ClassB

Çok düzeyli: ClassA ←Class B ← ClassC

Hiyerarşik: ClassB → ClassA ← ClassC

Java Programlamasında çoklu ve karma kalıtım yalnızca arayüzle desteklenir.

Çoklu: ClassA← ClassC →ClassB

Karma: ClassB → ClassA ← ClassC ← ClassD →ClassB

Tekli Kalıtım (Single Inheritance)

```
Class Hayvan{
Void beslenme(){System.out.println("yemek yiyor");}}
Class Kopek extends Hayvan{
Void havla(){System.out.println(""köpek havlıyor);}}
Class TestKalitim{
Public static void main(String args[]){
Kopek k=new Kopek();
k.havla();
k.beslenme();
}}
```

Çıktı: köpek havlıyor
yemek yiyor

Çok Düzeyli (Multilevel Inheritance)

```
Class Hayvan{
Void beslenme(){System.out.println("yemek yiyor");}}
Class Kopek extends Hayvan{
```

```
Void havla(){System.out.println(“”köpek havlıyor);}}
Class YavruKopek extends Kopek{
Void acik(){System.out.println(“acıktı”);}}
Class TestKalitim{
Public static void main(String args[]){
YavruKopek yk=new YavruKopek();
yk.acik();
yk.havla();
yk.beslenme();
}}
```

Çıktı: acıktı
köpek havlıyor
yemek yiyor

Hiyerarşik Kalıtım (Hierarchical Inheritance)

```
Class Hayvan{
Void beslenme(){System.out.println(“yemek yiyor”);}}
Class Kopek extends Hayvan{
Void havla(){System.out.println(“”köpek havlıyor);}}
Class Kedi extends Hayvan{
Void miyavla(){System.out.println(“kedi miyavlıyor”);}}
Class TestKalitim{
Public static void main(String args[]){
Kedi k=new Kedi();
k.miyavla();
// k.havla(); C.T. Error
k.beslenme();
}}
```

Çıktı: kedi miyavlıyor
yemek yiyor

Java’da Neden Çoklu Kalıtım Desteklenmiyor?

Karmaşıklığı azaltmak ve dili basitleştirmek için, java’da birden fazla kalıtım desteklenmez. A, B ve C gibi üç sınıf olduğu bir senaryo düşünün. C sınıfı A ve B sınıflarını miras alır. A ve B sınıfları aynı metoda sahipse ve siz onu sınıf nesnesinden çağırırsanız, A veya B sınıfı metodunu çağırmak için belirsizlik olacaktır. Derleme zamanı hataları çalışma zamanı hatalarından daha iyi olduğu için Java, iki sınıfı devraldığında derleme zamanı hatası oluşturur. Yani aynı metoda veya farklı bir metoda sahip olsanız, derleme zamanı hatası olacaktır.

```
Class A{
Void mesajYaz(){System.out.println(“Java”);}}
Class B{
Void mesajYaz(){System.out.println(“Phyton”);}}
```

```
Class C extends A,B{ //olabileceğini varsayalım
Public static void main(String args[]){
C nesne=new C();
nesne.mesajYaz(); // hangi mesajYaz metodu çağrılacak?? Hata:
}}
```

Metot Aşırı Yüklenmesi (Method Overloading)

Bir sınıfın aynı ada sahip, ancak parametreleri farklı olan birden çok metodu varsa metot aşırı yüklenir. Tek bir işlem yapmanız gerekirse, metotların aynı adı taşıyan programın okunabilirliğini artırır. Verilen sayıların eklenmesini gerçekleştirmeniz gerektiğini varsayalım. Ancak, iki parametre için a(int, int) ve üç parametre için b(int, int, int) gibi bir metot yazdıysanız, herhangi bir sayıda argüman olabilir. Diğer programcıların yanı sıra, metodun davranışını anlaması zor olabilir çünkü adı farklıdır. Bu nedenle, programı çabucak anlamak için metot aşırı yükleme yapıyoruz.

Metodun aşırı yüklenmesi avantajı, programın okunabilirliğini arttırır. Java metodunu aşırı yüklemenin iki yolu vardır: Parametre (argüman) sayısını değiştirerek ve veri türünü değiştirerek. Java'da metodun geri dönüş türünü değiştirerek metot aşırı yüklemesi mümkün değildir.

Parametre sayısını değiştirerek:

```
Class Toplayici{
Static int topla(int a, int b){return a+b;}
Static int topla(int a, int b, int c){return a+b+c;}}
Class TestOverloading{
Public static void main(String [] args){
System.out.println(Toplayici.topla(11,11));
System.out.println(Toplayici.topla(11,11,11));
}}
```

Veri türünü değiştirerek:

```
Class Toplayici{
Static int topla(int a, int b){return a+b;}
Static double topla(double a, double b){return a+b;}}
Class TestOverloading{
Public static void main(String [] args){
System.out.println(Toplayici.topla(11,11));
System.out.println(Toplayici.topla(12.3, 12.6));
}}
```

Neden metot aşırı yükleme, sadece dönüş tipini değiştirerek mümkün değildir?

Java'da, metodun geri dönüş türünü değiştirerek metot aşırı yüklenmesi mümkün değildir. Çünkü yalnızca belirsizlik getirmektedir. Ne kadar mı belirsiz olabilir?

```
Class Toplayici{
Static int topla(int a, int b){return a+b;}
Static double topla(int a, int b){return a+b;}}
Class TestOverloading{
```

```
Public static void main(String [] args){
System.out.println(Toplayici.topla(11,11)); //Hangisini çağıracak??
}}
```

Metot Geçersiz Kılma (Method Overriding)

Alt sınıf (çocuk sınıf), üst sınıfta bildirilenle aynı metoda sahipse, Java'da geçersiz kılma yöntemidir. Diğer bir deyişle, bir alt sınıf, ana sınıfından biri tarafından bildirilen yöntemin özel bir uygulamasını sağlarsa, bu yöntem geçersiz kılmadır. Metodun geçersiz kılınması, süper sınıfı tarafından halihazırda temin edilmiş olan bir yöntemin özel olarak uygulanmasını sağlamak için kullanılır. Çalışma zamanı Polimorfizmi için yöntem geçersiz kılma kullanılmaktadır.

Metot geçersiz kılma için üç kural vardır:

1. Metot, üst sınıftakiyle aynı ada sahip olmalıdır.
2. Metot, üst sınıftakiyle aynı parametreye sahip olmalıdır.
3. IS-A ilişkisi olmalıdır (miras)

```
Class Arac{
Void calistir(){System.out.println("Araç çalışıyor");}}
Class Bisiklet extends Arac{
Void calistir(){System.out.println("Bisiklet çalışıyor");}
Public static void main(String args[]){
Bisiklet nesne=new Bisiklet();
nesne.calistir();
}}
```

Çıktı: Bisiklet çalışıyor

```
Class Banka{
İnt faizOrani(){return 0;}}
Class YapiKredi extends Banka{
int faizOrani(){return 8;}}
Class Garanti extends Banka{
int faizOrani(){return 7;}}
Class isBankasi extends Banka{
int faizOrani(){return 9;}}
class Test{
public static void main(String args[]){
YapiKredi yb=new YapiKredi();
Garanti gb=new Garanti();
isBankasi ib=new isBankasi();
System.out.println("Yapı Kredi Bankası Faiz Oranı: "+yb.faizOrani());
System.out.println("Garanti Bankası Faiz Oranı: "+gb.faizOrani());
System.out.println("İş Bankası Faiz Oranı: "+ib.faizOrani());
}}
```

Çıktı: Yapı Kredi Bankası Faiz Oranı: 8

Garanti Bankası Faiz Oranı: 7

İş Bankası Faiz Oranı: 9

Statik yöntemi geçersiz kılabilir miyiz? Hayır, statik bir yöntem geçersiz kılınamaz. Çalışma zamanı Polimorfizmi ile kanıtlanabilir.

Statik yöntemi neden geçersiz kılamayız? Nedeni, statik yöntemin bir sınıfla bağlanması, örnek yönteminin bir nesneye bağlı olmasından kaynaklanır. Statik, sınıf alanına ait ve bir örnek yığın alanına aittir.

Java'da main() metodunu geçersiz kılabilir miyiz? Hayır, çünkü ana statik bir metottur.

super Anahtar Sözcüğü

Ana üst nesnesini ifade etmek için kullanılan bir referans değişkendir. Alt sınıf örneği oluşturduğunuzda, süper başvuru etiketiyle ifade edilen örtük bir alt sınıf örneği oluşturulur. Kullanımı:

1. super, üst sınıf örnek değişkenine başvurmak için kullanılır
2. super, ebeveyn sınıfı metodunu çağırmak için kullanılır.
3. super(), ana üst sınıf yapıcısını çağırmak için kullanılır.

```
Class Hayvan{
String renk=”beyaz”;}
Class Kopek extends Hayvan{
String renk=”siyah”;
Void renkYazdir(){
System.out.println(renk); //köpek sınıfının renk değeri
System.out.println(super.renk); //hayvan sınıfının renk değeri
}}
Class testSuper{
Public static void main(String args[]){
Kopek k=new Kopek();
k.renkYazdir();
}}
```

Çıktı: siyah

beyaz

```
Class Hayvan{
Void beslenme(){System.out.println(“yemek yiyor”);}}
Class Kopek extends Hayvan{
Void beslenme(){System.out.println(“ekmek yiyor”);}
Void havlama(){System.out.println(“”havlıyor);}
Void calis(){
Super.beslenme();
Havlama();
}}
Class TestSuper{
```

```
Public static void main(String args[]){
Kopek k=new Kopek();
k.calis();
}}
```

Çıktı: yemek yiyor
havlıyor

```
class Hayvan{
Hayvan(){System.out.println("Hayvan oluşturuldu");}}
Class Kopek extends Hayvan{
Kopek(){
super();
System.out.println("Köpek oluşturuldu");
}}
Class TestSuper{
Public static void main(String args[]){
Kopek k=new Kopek();
}}
```

Çıktı: Hayvan oluşturuldu
Köpek oluşturuldu

super() veya this() yoksa derleyici tarafından super() otomatik olarak her sınıf yapıcısına eklenir.

```
Class Bike{
}
```

```
class Bike{
Bike(){
süper();}}
```

Örnek

```
Class Personel{
İnt id;
String isim;
Personel(int id,, String isim){
This.id=id;
This.isim=isim;
}}
Class Emekci extends Personel{
Float ucret;
Emekci(int id, String isim, float ucret){
super(id,isim);
this.ucret=ucret;}
void göster(){System.out.println(id+" "+isim+" "+ucret);}}
```

```
class TestSuper{
public static void main(String args[]){
Emekci e1=new Emekci(1,”hüseyin”,45000f);
e1.goster();
}}
```
Çıktı: 1 hüseyin 45000

final Anahtar Sözcüğü

Kullanıcıyı kısıtlamak için kullanılır. Değişken, metot ve sınıf ile kullanılabilir.

```
Class Bisiklet{
Final int hizLimiti=90; //değeri değiştirilemez
Void calistir(){
hizLimiti=400;}
public static void main(String args[]){
Bisiklet obj=new Bisiklet();
obj.calistir();
}}
```
Çıktı: Output: Compile Time Error → final olduğu için değer değiştirilemez

```
Class Bisiklet{
Final void calistir(){System.out.println(“calisiyor”);}}
Class Honda extends Bisiklet{
Void calistir(){System.out.println(“100 km hızla çalışıyor”);}
Public static void main(String args[]){
Honda honda =new Honda();
honda.calistir();
}}
```
Çıktı: Output: Compile Time Error → final olduğu için calistir metodu farklı işlem yapamaz

```
Final class Bisiklet{}
Class Honda extends Bisiklet{
void calistir(){System.out.println(“calisiyor”);}
Public static void main(String args[]){
Honda honda =new Honda();
honda.calistir();
}}
```
Çıktı: Bisiklet değiştirilemez. Hata verecektir.

Final metot miras alabilir, anccak override edilemez. Deklarasyon sırasında başlatılmamış final değişken boş final değişken olarak bilinir. Final değişkenler nesne oluşturma sırasında başlatılan bir değişken oluşturmak istiyorsanız

ve başlatıldıktan sonra değiştirilemezse, yararlıdır. Boş final değişkeni yalnız kurucuda başlatılabilir. Bir kurucu metot final alamaz. Çünkü kurucu asla miras alınamaz.

Soyut Sınıf (Abstract Class)

Soyut anahtar kelimesi ile bildirilen bir sınıf, Java'da soyut sınıf olarak bilinir. Soyut ve soyut olmayan yöntemlere sahip olabilir. Java'da soyut sınıf kavramını öğrenmeden önce Java'daki soyutlama anlaşılmalıdır.

Java'da Soyutlama (Abstraction)

Soyutlama, uygulama ayrıntılarını gizleme ve yalnızca kullanıcıya işlevselliği gösteren bir süreçtir. Başka bir şekilde, kullanıcı için sadece önemli şeyleri gösterir ve iç detayları gizler. Örneğin metni yazıp SMS mesajı gönderirken mesaj dağıtımıyla ilgili dahili işlemleri bilmiyorsunuz. Soyutlama, nesnenin bunun nasıl yapıldığına değil, ne yaptığı üzerine odaklanmanıza izin verir.

Java'da soyutlamayı başarmanın iki yolu vardır. Soyut sınıf (Abstract class), Arayüz(interface)

Java'da Soyut Sınıf (Abstract Class)

Soyut olarak ilan edilen bir sınıf, abstract bir sınıf olarak bilinir. Soyut ve soyut olmayan yöntemlere sahip olabilir. Genişletilmesi ve yönteminin uygulanması gerekiyor. Örneklendirilemez.

Soyut bir sınıf, abstract anahtar kelimesi ile bildirilmelidir. Soyut ve soyut olmayan yöntemlere sahip olabilir. Ayrıca kurucu ve statik yöntemleri de olabilir. Metodun gövdesini değiştirmemesi için alt sınıfı zorlayacak final yöntemlere sahip olabilir.

Soyut bir metoda sahip soyut sınıf: Bu örnekte, Bisiklet sınıfı sadece bir soyut metot çalıştırmayı içeren soyut bir sınıftır. Uygulaması Honda sınıfı tarafından sağlanmaktadır.

```
abstract class Bisiklet{
abstract void calistir();}
class Honda extends Bisiklet{
void calistir(){System.out.println("güvenli çalışıyor");}
public static void main(String args[]){
Bisiklet nesne=new Honda();
Nesne.calistir();
}}
```

Çıktı: güvenli çalışıyor

```
abstract class Sekil{
abstract void ciz();}
class Dikdortgen extends Sekil{
void ciz(){System.out.println("dikdörtgen çiziliyor");}}
class Cember extends Sekil{
void ciz(){System.out.println("çember çiziliyor");}}
class TestAbstraction{
public static void main(String args[]){
Sekil s=new Cember();
s.ciz();}}
```

Çıktı: çember çiziliyor

```
Abstract Class Banka{
Abstract İnt faizOrani();}
Class YapiKredi extends Banka{
int faizOrani(){return 7;}}
Class Garanti extends Banka{
int faizOrani(){return 8;}}
class TestBanka{
public static void main(String args[]){
Banka b;
b=new YapiKredi();
System.out.println("Faiz Oranı: "+b.faizOrani());
b=new Garanti();
System.out.println("Faiz Oranı: "+b.faizOrani());
}}
```

Çıktı: Faiz Oranı: 7

Faiz Oranı: 8

Java'da Arayüz (Interface)

Java'da arayüz bir sınıfın bir planıdır. Statik sabitlere ve soyut yöntemlere sahiptir. Java'daki arayüz, soyutlamayı elde etmek için bir mekanizmadır. Java arayüzünde metot gövdesi değil, sadece soyut metotlar olabilir. Java'da soyutlama ve çoklu kalıtım sağlamak için kullanılır. Başka bir deyişle, arayüzlerin soyut metotlara ve değişkenlere sahip olduğunu söyleyebiliriz. Bir metot gövdesi olamaz. Java interface de IS-A ilişkisini temsil ederr. Soyut (abstract) sınıf gibi soyutlanamaz. Bir arayüzde varsayılan (default) ve statik metotlara sahip olabiliriz. Bir arayüzde özel(private) metotlarımız olabilir.

Arayüzü kullanmanın üç nedeni vardır.

1. Soyutlamayı sağlamak için kullanılır
2. Arabirim ile çoklu kalıtım işlevselliğini destekleyebiliriz
3. Gevşek bağlama elde etmek için kullanılabilir.

Sınıf ve arayüz arasındaki ilişki: bir sınıf başka bir sınıfı genişletir, bir arabirim başka bir arabirimi genişletir ve ancak bir sınıf bir arabirimi uygular.

```
interface yazdirilabilir{
void yaz();}
class A6 implements yazdirilabilir{
public void yaz(){System.out.println("Merhaba");}
public static void main(String args[]){
A6 nesne=new A6();
nesne.yaz();
}}
```

```
İnterface Cizilebilir{
Void ciz();}
class Dikdortgen implements Cizilebilir{
public void ciz(){System.out.println(“dikdörtgen çiziliyor”);}}
class Cember implements Cizilebilir{
public void ciz(){System.out.println(“çember çiziliyor”);}}
class TestInterface{
public static void main(String args[]){
Cizilebilirl nesne=new Cember();
nesne.ciz();}}
```