****

***JAVA OBJECT ORİENTED***

***EREN ÖZEL***

**FEYZANUR TOYGAR-216501816**

**ŞULE NUR DOĞAN-216571207**

**RABİA YILMAZ-217462140**



**JAVA OBJECT ORİENTED**

***INHERITANCE (KALITIM):***

Kalıtım, bir sınıfın diğerinin özelliklerini (yöntemleri ve alanları) aldığı süreç olarak tanımlanabilir. Kalıtım kullanımıyla bilgiler hiyerarşik bir düzende yönetilebilir hale getirilir. Başkalarının özelliklerini kalıtım alan sınıf, alt sınıf (türetilmiş sınıf, alt sınıf) ve özellikleri kalıtım alınan sınıf, üst sınıf (temel sınıf, ana sınıf) olarak bilinir.

***Anahtar Kelimeyi Genişletir***

Genişletme, bir sınıfın özelliklerini kalıtım almak için kullanılan anahtar kelimedir. Extends anahtar sözcüğünün sözdizimi aşağıdadır.

**Syntax**

class Super {

.....

.....

}

class Sub extends Super {

.....

.....

}

***Basit kod***

Java kalıtımını gösteren bir örnek aşağıdadır. Bu örnekte, Hesaplama ve My\_Calculation adı verilen iki sınıfı gözlemleyebilirsiniz.

Extends anahtar sözcüğünü kullanarak My\_Calculation, Calculation sınıfının yöntemleri (() ve Çıkarma () yöntemlerini devralır.

Aşağıdaki programı My\_Calculation.java isimli bir dosyaya kopyalayıp yapıştırın

**Example**

class Calculation {

int z;

public void addition(int x, int y) {

z = x + y;

System.out.println("The sum of the given numbers:"+z);

}

public void Subtraction(int x, int y) {

z = x - y;

System.out.println("The difference between the given numbers:"+z);

}

}

public class My\_Calculation extends Calculation {

public void multiplication(int x, int y) {

z = x \* y;

System.out.println("The product of the given numbers:"+z);

}

public static void main(String args[]) {

int a = 20, b = 10;

My\_Calculation demo = new My\_Calculation();

demo.addition(a, b);

demo.Subtraction(a, b);

demo.multiplication(a, b);

}

}

Yukarıdaki kodu aşağıdaki gibi derleyin ve uygulayın.

javac My\_Calculation.java

java My\_Calculation

Programı yürüttükten sonra aşağıdaki sonucu verecektir

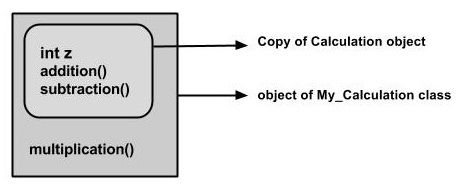
**Output**

The sum of the given numbers:30

The difference between the given numbers:10

The product of the given numbers:200

Verilen programda, My\_Calculation sınıfına bir nesne oluşturulduğunda, üst sınıfın içeriğinin bir kopyası bunun içinde yapılır. Bu nedenle, alt sınıfın nesnesini kullanarak bir üst sınıfın üyelerine erişebilirsiniz.



Üst sınıf referans değişkeni alt sınıf nesnesini tutabilir, ancak bu değişkeni kullanarak yalnızca üst sınıfın üyelerine erişebilirsiniz, bu nedenle her iki sınıfın üyelerine erişmek için her zaman alt sınıfa referans değişkeni oluşturmanız önerilir.

Yukarıdaki programı göz önünde bulundurursanız, sınıfı aşağıdaki gibi başlatabilirsiniz. Ancak, üst sınıf referans değişkenini (bu durumda cal) kullanarak, My\_Calculation alt sınıfına ait olan multiplic () yöntemini çağıramazsınız.

Calculation demo = new My\_Calculation();

demo.addition(a, b);

demo.Subtraction(a, b);

**Not** - Bir alt sınıf, tüm üyeleri (alanlar, yöntemler ve iç içe sınıflar) üst sınıfından miras alır. Yapıcılar üye değildir, bu nedenle alt sınıflar tarafından miras alınmazlar, ancak üst sınıfın yapıcısı alt sınıftan çağrılabilir.

***Süper anahtar kelime***

Süper anahtar benzer bu anahtar kelime. Süper anahtar kelimenin kullanıldığı senaryolar aşağıdadır.

* Süper sınıf üyelerini, aynı isimlere sahiplerse, alt sınıf üyelerinden ayırmak için kullanılır .
* Üst sınıf yapıcısını alt sınıftan çağırmak için kullanılır .

### ***Üyeleri Farklılaştırmak***

Bir sınıf başka bir sınıfın özelliklerini miras alıyorsa. Süper sınıfın üyeleri alt sınıfla aynı isimlere sahipse, bu değişkenleri ayırt etmek için aşağıda gösterildiği gibi super anahtar sözcüğünü kullanırız.

super.variable

super.method();

### ***Basit kod***

Bu bölüm size **süper** anahtar kelimenin kullanımını gösteren bir program sunar .

Verilen programda, *Sub\_class* ve *Super\_class* adlı iki sınıfınız var , her ikisi de farklı uygulamalarla display () adında bir yönteme ve farklı değerlere sahip num adlı bir değişkene sahip. Her iki sınıfın display () yöntemini çağırıyoruz ve her iki sınıfın değişken sayısının değerini yazdırıyoruz. Burada, süper sınıf üyelerini alt sınıftan ayırmak için super anahtar kelimesini kullandığımızı gözlemleyebilirsiniz.

Programı kopyalayıp Sub\_class.java isimli bir dosyaya yapıştırın.

Aşağıdaki sözdizimini kullanarak yukarıdaki kodu derleyin ve yürütün.

javac Super\_Demo

java Super

Programı çalıştırırken aşağıdaki sonucu alırsınız –

**Çıktı**

This is the display method of subclass

This is the display method of superclass

value of the variable named num in sub class:10

value of the variable named num in super class:20

## *Süper Sınıf Oluşturucuyu Çağırma*

## Bir sınıf başka bir sınıfın özelliklerini miras alıyorsa, alt sınıf otomatik olarak üst sınıfın varsayılan yapıcısını alır. Ancak, üst sınıfın parametreli hale getirilmiş bir kurucusunu çağırmak istiyorsanız, super anahtar sözcüğünü aşağıda gösterildiği gibi kullanmanız gerekir.

super(values);

### ***Basit kod***

Bu bölümde verilen program, superclass'ın parametreli hale getirilmiş yapıcısını çağırmak için super anahtar sözcüğünün nasıl kullanılacağını gösterir. Bu program bir üst sınıf ve bir alt sınıf içerir, burada üst sınıf bir tamsayı değeri kabul eden parametreli bir kurucu içerir ve üst sınıfın parametreli hale getirilmiş yapıcısını çağırmak için super anahtar sözcüğünü kullanırız.

Aşağıdaki programı Subclass.java adlı bir dosyaya kopyalayıp yapıştırın.

***Örnek***

class Superclass {

int age;

Superclass(int age) {

this.age = age;

}

public void getAge() {

System.out.println("The value of the variable named age in super class is: " +age);

}

}

public class Subclass extends Superclass {

Subclass(int age) {

super(age);

}

public static void main(String argd[]) {

Subclass s = new Subclass(24);

s.getAge();

}

}

Aşağıdaki sözdizimini kullanarak yukarıdaki kodu derleyin ve yürütün.

javac Subclass

java Subclass

Programı çalıştırırken aşağıdaki sonucu alırsınız

***Çıktı***

The value of the variable named age in super class is: 24

## *IS-A İlişkisi*

IS-A, şöyle demenin bir yoludur: Bu nesne, o nesnenin bir türüdür. **Extends** anahtar sözcüğünün kalıtım elde etmek için nasıl kullanıldığını görelim .

genel sınıf Hayvan { }

genel sınıf Memeli , Hayvanı { } uzatır

genel sınıf Sürüngen , Hayvanı { } uzatır

genel sınıf Köpek Mammal'i uzatıyor { }

Şimdi, yukarıdaki örneğe göre, Nesne Yönelimli terimlerle, aşağıdakiler doğrudur -

* Hayvan, Mammal sınıfının üst sınıfıdır.
* Hayvan, Sürüngen sınıfının üst sınıfıdır.
* Memeli ve Sürüngenler, Hayvan sınıfının alt sınıflarıdır.
* Köpek, hem Memeli hem de Hayvan sınıflarının alt sınıfıdır.

Şimdi, eğer IS-A ilişkisini düşünürsek, şunu söyleyebiliriz:

* Memeli IS-A Hayvan
* Sürüngen IS-A Hayvan
* Köpek IS-A Memeli
* Dolayısıyla: Köpek IS-A Hayvanı

Extends anahtar sözcüğünün kullanılmasıyla, alt sınıflar, üst sınıfın özel özellikleri dışında, üst sınıfın tüm özelliklerini devralabilir.

Mammal'in, örnek operatörün kullanımıyla aslında bir Hayvan olduğundan emin olabiliriz.

***Örnek***

class Animal {

}

class Mammal extends Animal {

}

class Reptile extends Animal {

}

public class Dog extends Mammal {

public static void main(String args[]) {

Animal a = new Animal();

Mammal m = new Mammal();

Dog d = new Dog();

System.out.println(m instanceof Animal);

System.out.println(d instanceof Mammal);

System.out.println(d instanceof Animal);

}

}

Bu aşağıdaki sonucu üretecektir -

***Çıktı***

true

true

true

Extends anahtar sözcüğünü iyi anladığımızdan , uygulamalı anahtar kelimenin IS-A ilişkisini elde etmek için nasıl kullanıldığına bakalım .

Genel olarak, implements anahtar sözcüğü bir arabirimin özelliklerini miras almak için sınıflarla birlikte kullanılır. Arayüzler asla bir sınıf tarafından genişletilemez.

***Örnek***

public interface Animal {

}

public class Mammal implements Animal {

}

public class Dog extends Mammal {

}

***Anahtar Kelime İfadesi***

Bize kullanalım instanceof Memeli aslında bir hayvan olduğunu ve köpek aslında bir hayvan olup olmadığını belirlemek kontrol etmesine.

***Örnek***

interface Animal{}

class Mammal implements Animal{}

public class Dog extends Mammal {

public static void main(String args[]) {

Mammal m = new Mammal();

Dog d = new Dog();

System.out.println(m instanceof Animal);

System.out.println(d instanceof Mammal);

System.out.println(d instanceof Animal);

}

}

Bu aşağıdaki sonucu üretecektir -

***Çıktı***

true

true

true

***HAS-A ilişkisi***

Bu ilişkiler esas olarak kullanıma dayanmaktadır. Bu, belirli bir sınıfın belirler HAS-bir belirli şey. Bu ilişki, hataların yanı sıra kod çoğaltılmasını da azaltmaya yardımcı olur.

Bir örneğe bakalım -

***Örnek***

public class Vehicle{}

public class Speed{}

public class Van extends Vehicle {

private Speed sp;

}

Bu, Van HAS-A Hızı sınıfını gösteriyor. Hız için ayrı bir sınıfa sahip olarak, hız sınıfına ait tüm kodu Van sınıfının içine koymak zorunda değiliz; bu da Hız sınıfını birden fazla uygulamada yeniden kullanabilmeyi mümkün kılıyor.

Nesneye Yönelik özelliğinde, kullanıcıların gerçek nesneyi hangi nesnenin yaptığını rahatsız etmeleri gerekmez. Bunu başarmak için, Van sınıfı, uygulama ayrıntılarını Van sınıfı kullanıcılarından gizler. Yani, temelde olan, kullanıcıların Van sınıfından belirli bir eylemde bulunmasını isteyecekleri ve Van sınıfından işi kendi başlarına yapacak veya başka bir sınıftan eylemi gerçekleştirmesini isteyecek olmasıdır.

***Miras Çeşitleri***

Aşağıda gösterildiği gibi çeşitli kalıtım türleri vardır.



Hatırlanması gereken çok önemli bir gerçek, Java'nın çoklu mirası desteklememesidir. Bu, bir sınıfın birden fazla sınıfı uzatamayacağı anlamına gelir. Bu nedenle aşağıdakiler yasaktır -

***Örnek***

public class extends Animal, Mammal{}

Bununla birlikte, bir sınıf, Java'nın çoklu kalıtımın imkansızlığından kurtulmasına yardımcı olan bir veya daha fazla arayüz uygulayabilir.

***OVERRIDING***

 Bir sınıf, üst sınıfından bir yöntemi miras alırsa, nihai olarak işaretlenmemesi koşuluyla, yöntemi geçersiz kılma şansı vardır.

Geçersiz kılmanın yararı şudur: alt sınıf türüne özgü bir davranış tanımlama yeteneği, yani bir alt sınıfın gereksinimine göre bir üst sınıf yöntemi uygulayabileceği anlamına gelir.

Nesne yönelimli terimlerle geçersiz kılma, mevcut bir yöntemin işlevselliğini geçersiz kılmak anlamına gelir.

### ***Örnek***

Bir örneğe bakalım.

class Animal {

public void move() {

System.out.println("Animals can move");

}

}

class Dog extends Animal {

public void move() {

System.out.println("Dogs can walk and run");

}

}

public class TestDog {

public static void main(String args[]) {

Animal a = new Animal(); // Animal reference and object

Animal b = new Dog(); // Animal reference but Dog object

a.move(); // runs the method in Animal class

b.move(); // runs the method in Dog class

}

}

Bu aşağıdaki sonucu üretecektir -

### ***Çıktı***

Animals can move

Dogs can walk and run

Yukarıdaki örnekte, b'nin bir Hayvan türü olmasına rağmen , Köpek sınıfındaki taşıma yöntemini çalıştırdığını görebilirsiniz. Bunun nedeni şudur: Derleme zamanında, kontrol referans tipinde yapılır. Bununla birlikte, çalışma zamanında, JVM nesne türünü belirler ve o belirli nesneye ait olan yöntemi çalıştırır.

Bu nedenle, yukarıdaki örnekte, Animal sınıfı yöntemi hareket ettiğinden program düzgün bir şekilde derlenecektir. Ardından, çalışma zamanında, bu nesneye özgü yöntemi çalıştırır.

Aşağıdaki örneği düşünün -

### ***Örnek***

class Animal {

public void move() {

System.out.println("Animals can move");

}

}

class Dog extends Animal {

public void move() {

System.out.println("Dogs can walk and run");

}

public void bark() {

System.out.println("Dogs can bark");

}

}

public class TestDog {

public static void main(String args[]) {

Animal a = new Animal(); // Animal reference and object

Animal b = new Dog(); // Animal reference but Dog object

a.move(); // runs the method in Animal class

b.move(); // runs the method in Dog class

b.bark();

}

}

Bu aşağıdaki sonucu üretecektir.

### ***Çıktı***

TestDog.java:26: error: cannot find symbol

b.bark();

^

symbol: method bark()

location: variable b of type Animal

1 error

Bu program derleme zamanı hatası verecektir, çünkü b'nin referans türü Animal, kabuk adında bir metoda sahip değildir.

## *Metot Geçersiz Kılma Kuralları*

* Argüman listesi, geçersiz kılınan yöntemin aynısı olmalıdır.
* Dönüş tipi, üst sınıftaki orijinal geçersiz kılma yönteminde belirtilen dönüş tipiyle aynı veya alt tipte olmalıdır.
* Erişim seviyesi, geçersiz kılınan yöntemin erişim seviyesinden daha kısıtlayıcı olamaz. Örneğin: Süper sınıf yöntemi genel ilan edilirse, alt sınıftaki geçersiz kılma yöntemi özel veya korumalı olamaz.
* Örnek yöntemleri, yalnızca alt sınıf tarafından miras alınması durumunda geçersiz kılınabilir.
* Son olarak ilan edilen bir yöntem geçersiz kılınamaz.
* Statik olarak ilan edilen bir metot geçersiz kılınamaz ancak yeniden ilan edilebilir.
* Bir yöntem miras alınamazsa, geçersiz kılınamaz.
* Örneğin üst sınıfıyla aynı paketteki bir alt sınıf, özel veya nihai olarak bildirilmeyen tüm üst sınıf yöntemlerini geçersiz kılabilir.
* Farklı bir paketteki bir alt sınıf, yalnızca genel olarak ilan edilen veya korunan nihai olmayan yöntemleri geçersiz kılabilir.
* Geçersiz kılma yöntemi, geçersiz kılınan yöntemin istisnalar atıp atmamasına bakılmaksızın, işareti olmayan herhangi bir istisna olabilir. Ancak, geçersiz kılma yöntemi, geçersiz kılınan yöntem tarafından bildirilenden daha yeni veya daha geniş olan kontrol edilen istisnalar atmamalıdır. Geçersiz kılma yöntemi, geçersiz kılma yönteminden daha dar veya daha az istisna atabilir.
* Yapıcılar geçersiz kılınamaz.

## *Süper Anahtar Kelimeyi Kullanma*

Geçersiz kılınan bir yöntemin üst sınıf bir sürümünü çağırırken, **super** anahtar sözcüğü kullanılır.

### ***Örnek***

class Animal {

public void move() {

System.out.println("Animals can move");

}

}

class Dog extends Animal {

public void move() {

super.move(); // invokes the super class method

System.out.println("Dogs can walk and run");

}

}

public class TestDog {

public static void main(String args[]) {

Animal b = new Dog(); // Animal reference but Dog object

b.move(); // runs the method in Dog class

}

}

Bu aşağıdaki sonucu üretecektir -

### ***Çıktı***

Animals can move

Dogs can walk and run

***POLYMORPHSIM (POLİMORFİZM)***

* Polimorfizm kelimesi birçok biçime sahip olmak demektir.
* “Poly” kelimesi çok, “morphs” ise form anlamına gelir, yani birçok form anlamına gelir.
* Basit bir deyişle, polimorfizmi bir mesajın birden fazla biçimde gösterilebilmesi olarak tanımlayabiliriz.
* **Gerçek hayat polimorfizmi örneği:** Aynı anda bir insan farklı özelliklere sahip olabilir. Aynı zamanda bir erkek gibi bir baba, bir koca, bir çalışan. Yani aynı kişi, farklı durumlarda farklı davranışlara sahiptir. Buna polimorfizm denir.

1. **Çalışma Zamanı Polimorfizmi örneği:**

public class Cat extends Animal{

@Override

public void sound(){

System.out.println("Meow");

}

public static void main(String args[]){

Animal obj = new Cat();

obj.sound();

}

}

Output:

Meow

1. **Derleme zamanı Polimorfizmi**

* Statik polimorfizm olarak da bilinir. Bu tip polimorfizm, fonksiyon aşırı yüklenmesi veya operatör aşırı yüklenmesi ile sağlanır.
* Java, operatörleri aşırı yükleme seçeneği de sunar. Örneğin, dize sınıfının iki dizgiyi birleştirmesi için işlecini ('+') yapabiliriz.
* Java'da, yalnızca “+” operatörü aşırı yüklenebilir:

class Overload

{

void demo (int a)

{

System.out.println ("a: " + a);

}

void demo (int a, int b)

{

System.out.println ("a and b: " + a + "," + b);

}

double demo(double a) {

System.out.println("double a: " + a);

return a\*a;

}

}

class MethodOverloading

{

public static void main (String args [])

{

Overload Obj = new Overload();

double result;

Obj .demo(10);

Obj .demo(10, 20);

result = Obj .demo(5.5);

System.out.println("O/P : " + result);

}

}

**Output:**

a: 10

a and b: 10,20

double a: 5.5

O/P : 30.25

* Burada yöntem demo()3 kez aşırı yüklenmiştir: ilk yöntem 1 int parametresine, ikinci yöntem 2 int parametrelere ve üçüncü yöntem çift parametreye sahiptir. Hangi yöntemin çağrılacağı, yöntemleri çağırırken aktardığımız argümanlarla belirlenir. Bu tür polimorfizm, derleme zamanı polimorfizmi olarak bilinir.

# ***Java – Abstraction (Soyutlama)***

* Nesneye yönelimli programlamadaki en önemli kavramlardan biriside soyutlamadır. Büyük ve kapsamlı program birçok sistem parçası ile mesajlaşarak çalışmaktadır. Yapılan bu çalışmaların hepsini bir programcı olarak bilmek zor, zaman alıcı ve gereksizdir. Böyle bir sistemin detaylarına odaklanmak yerine girdi ve çıktılarını bakmak sistemi daha iyi anlamıza olanak sağlar.

/\* File name : Employee.java \*/

public abstract class Employee {

private String name;

private String address;

private int number;

public Employee(String name, String address, int number) {

System.out.println("Constructing an Employee");

this.name = name;

this.address = address;

this.number = number;

}

public double computePay() {

System.out.println("Inside Employee computePay");

return 0.0;

}

public void mailCheck() {

System.out.println("Mailing a check to " + this.name + " " + this.address);

}

public String toString() {

return name + " " + address + " " + number;

}

public String getName() {

return name;

}

public String getAddress() {

return address;

}

public void setAddress(String newAddress) {

address = newAddress;

}

public int getNumber() {

return number;

}

}

***ENCAPSULATION-INTERFACES***

***ENCAPSULATION:*** Verilerin dış ortamdan olası aksaklıklar karşısında koruma mekanizması olarak adlandırılır.

-Yani oluşturulan bir class ın özelliklerini private yapmak(başka classlar tarafından erişebilirliğini kaldırmak) ve class ın detaylarını başka kullanıcılara olabildiğince gizlemek anlamına gelmektedir.

***ÖRNEK:***

* Örnekte göstereceğimiz araba kapı sayısının nasıl değiştiği, car classında kapı sayısı private yapılarak ve metodlarda bunları if koşulu sayesinde kontrol altına alınmış olunur.
* Araba kapı sayısını sadece 2 veya 4 olacak şekilde değiştirilmesini kontrol altına alınmak istenilmektedir.
* Bunu car class ından private olarak tanımladığımız doors yani kapı sayısını 2 veya 4 olacak şekilde if koşulunun yardımı ile kısıtlandırabiliriz.

**public class** Main {  
 **public static void** main(String[] args) {  
Car car1 = **new** Car();  
 car1.setDoors(**1**);  
 System.***out***.println(**"araba kapı sayısı:"** + car1.getDoors());

**Bu şekilde car1.setDoors(1); 1 yazarsak aşağıdaki kullanımdan dolayı kapı saysısının sadece 4 olabileceği uyarısını alırız.**

**Ancak buraya 2 veya 4 yazarsak kod sorunsuz bir şekilde çalışacaktır.**

**public class** Car {  
 *//fields;arabamıza 4 tane field yani özellik tanımladık.* **private** String **color**;  
 **private** String **model**;  
 **private double engine**;  
 **private int doors**; *//private : erişim belirleyicisi.*  
  
*//şuanki objemizin doors özelliğini, gönderdiğim doors a eşitlemek istiyoruz.(set)*

**public void** setDoors(**int** doors) {

if (doors == 2 || doors == 4 ) {

this.doors = doors;

}

else {

System.out.println("kapı sayısı sadece 2 veya 4 olabilir.");

}  
 **this**.**doors** = doors;  
 }  
*//setDoors a dışardan erişmek istersek, yani değerini almak istersek sadece, değer vermek istemezsek(get);*

**public int** getDoors(){  
 **return doors**;  
 }  
*//objenin kendi class ı içinden ulaşılmasını istiyorsak, yani referansını vermek istiyorsak, bu referansımız this olur.*   
  
}

**INTERFACES**

* Interface ler class lar ile benzerlik göstermektedir, interface ler aslında bir çeşit classtır. Fakat interfaces lerden herhangi bir obje oluşturulamaz.
* Java’da tanımlanmış bir interface sınıftan, normal bir sınıfta olduğu gibi new() operatörü ile  
  bir nesne oluşturulamaz
* Bir interface sınıfında sadece metotlar açıklanır. Bu metotların gövdeleri boştur. Alt sınıflar bu metotların gövdeleri için gerekli kodu oluştururlar **(implement ederler).** Metot gövdesi olmayan bir interface sınıftan nesne oluşturulamaz, çünkü sadece metot bildirimine sahip bir nesnenin hiçbir işlevsel görevi olamaz.
* Örneğin elimizde iki tane sınıf var. Ve ikisinde de ortak metod isimleri kullanacağız.
* Interface sınıfında metotların isimleri yazılır.
* Sınıflarda implement ederek kullanılır.
* Metotların içi istenilen şekilde sınıfta kullanılır.

**interface** TestiCozebilme{  
 **public void** testicoz();  
}  
  
**interface** TestiDogruCozebilme{  
 **public void** testidogrucoz();  
}  
  
**class** Ogrenci **implements** TestiCozebilme, TestiDogruCozebilme{  
 **public void** testicoz() {  
 System.***out***.println(**"Öğrenci testi çözüyor"**);  
 }  
 **public void** testidogrucoz() {  
 System.***out***.println(**"öğrenci testi doğru çözüyor"**);  
 }  
}