



MEDİPOL
UNV-ISTANBUL
ISTANBUL MEDIPOL UNIVERSITY

Adı Soyadı / Name Surname: Serra Vuslat AKYÜZ

Numarası/ Number: H5220055

MYO/YO/Fakülte/ Enstitü(School):Meslek Yüksekokulu

Bölüm/Program(Dept.):Bilgisayar Programcılığı

Dersin Adı/ Course Name: İleri Java

Öğretim Elemanı/ Instuctor: Hüseyin KINAY

Sınıf (class) nesnelerden oluşan geniş kapsamlı belirli kural ve yöntemleri olan kod topluluğudur. OOP 'da(Nesne Yönelimli Programlama) nesne sınıfın özelliklerini taşıyan yapı taşıdır.Sınıflar(classlar), metod ve belli başlı özelliklerden oluşur.

OOP nedir?

OOP,Objected Orianted Programming yani Nesneye Yönelik Programlama anlamına gelir .

Prosedürel programlama yani bir işte uyulması, tutulması gereken yol ve yöntemlerin izlediği kodlama ve yöntemlerin yazılmasıyla ilgili veriler üzerinde işlemler gerçekleştirilen prosedürel kodlarla ilgili durum; OOP ise içinde veri saklayan ve bu veriler üzerinde işlem yapacak olan metodlar bulunduran, her uygulamada tekrar kullanılabilir ve çağırılabilir yani zamandan ve enerjiden tasarruf sağlayan kodlama biçimidir.

OOP "Tekrar etmeme (DRY) ilkesi, kod tekrarının en aza indirilmesi tabiri şeklinde olabilir. Ortak kodları tekrarlamak yerine yeniden kullanma durumunu temsil eder.

Yazılım dünyasında gerçek dünyadaki nesnelerin tasarımları sınıf içinde yapılabilir.Örnek verecek olursak; sınıflar meyve, nesneler ise üzüm, greyfurt, mandalina şeklinde olabilir. Burada miras alma durumu mantığı gerçekleşir.

OOP 4 temel özellik altında toplanır:

1-Soyutlama (Abstraction)

2-Kapsülleme (Encapsulation)

3-Miras Alma (Inheritance)

4-Çok Biçimlilik (Polymorphism)

-Sınıftan nesne üretilip güncellemeye tabii tutulduğunda tüm programda güncelleme yapmak gerekmez, sadece oluşturulan nesnenin sınıf içinde değişiklik yapılması yeterli olacaktır.

-Kodları belirli bir proje ya da plan içerisinde topluyor şeklinde değerlendirebiliriz.

MainS değişkeniyle " " adında bir sınıf oluşturalım :

```
public class Main {  
    int S = 5;  
}
```

Önemli! Sınıf adları her zaman büyük harfle başlar ve java dosyasının adının sınıf adıyla eşleşmesi gerekir.

" " adında bir nesne oluşturalım myObjve S'in değerini yazdıralım:

```
public class Main {  
    int x = 5;  
  
    public static void main(String[] args) {  
        Main myObj = new Main();  
        System.out.println(myObj.x);  
    }  
}
```

Çıktı:5

Şekinde olacaktır.

Bir sınıftan birden fazla nesne oluşturabilir

```
public class Main {  
  
    int x = 5;  
  
    int y = 25;  
  
    int z = 45;  
  
  
    public static void main(String[] args) {  
  
        Main obj1 = new Main();  
  
        Main obj2 = new Main();  
  
        Main obj3 = new Main();  
  
  
        // Değerleri yazdırma  
  
        System.out.println("Object 1: x = " + obj1.x + ", y = " + obj1.y + ", z = " + obj1.z);  
  
        System.out.println("Object 2: x = " + obj2.x + ", y = " + obj2.y + ", z = " + obj2.z);  
  
        System.out.println("Object 3: x = " + obj3.x + ", y = " + obj3.y + ", z = " + obj3.z);  
  
    }  
}
```

Çıktı: Object 1: x = 5, y = 25, z = 45

Object 2: x = 5, y = 25, z = 45

Object 3: x = 5, y = 10, z = 45

Şeklinde olacaktır.

Birden Çok Sınıfı Kullanma:

Başka bir sınıftan erişebilirsiniz , bir sınıfın nesnesini oluşturabilir durumda iken.Genellikle organizasyon optimizasyonu için kullanılır (bir sınıf tüm niteliklere sahipken, diğer sınıf main())yöntemi (run edilen ve görüntülenen kod) tutar).

-Main.java

-İkinci.java

```
public class Main {
```

```
    int x = 8;
```

```
}
```

```
class Second {
```

```
    public static void main(String[] args) {
```

```
        Main myObj = new Main();
```

```
        System.out.println(myObj.x);
```

```
    }
```

```
}
```

Her iki dosya da derlendiğinde:

C:\Users\Your Name>javac Main.java

C:\Users\Your Name>javac Second.java

Ve çıktı şöyle olacaktır:

8

Java Sınıfı Nitelikleri:

Main iki özelliğe sahip " " adında bir sınıf oluşturalım : x ve y:

```
public class Main {  
    int x = 7;  
    int y = 9;  
}
```

Niteliklere Erişim

Niteliklere, sınıfın bir nesnesini oluşturarak ve nokta sözdizimini () kullanarak erişilebilir .:

Örnek

" " adında bir nesne oluşturalım ve myObj'e değerini yazdıralım x:

```
public class Main {  
    int x = 6;  
  
    public static void main(String[] args) {  
        Main myObj = new Main();  
        System.out.println(myObj.x);  
    }  
}
```

Çıktı:6

Şeklinde olacaktır.

Nitelikleri Değiştir:

Özellik değerlerini de değiştirebilirsiniz:

Örnek

Değerini x80 olarak ayarlayın:

```
public class Main {  
    int x;  
  
    public static void main(String[] args) {  
        Main myObj = new Main();  
        myObj.x = 80;  
        System.out.println(myObj.x);  
    }  
}
```

çıktı:80

Şeklinde olacaktır.

-Eğer mevcut değerler geçersiz olmasın ve sabit kalsın istenilirse final anahtar kelimesini kullanabiliriz.

```
public class Main {  
    final int x = 15;
```

```
public static void main(String[] args) {  
    Main myObj = new Main();  
    myObj.x = 25;    System.out.println(myObj.x);  
}  
}
```

çıktı: ERROR

Şeklinde olacaktır.

Çoklu Nesneler:

Öznitelik değerleri değişmeden nesneleri tutabiliriz.

Örnek

değerini x36 olarak değiştirin myObj2 ve değiştirmeden x şeklinde myObj1 bırakalım:

```
public class Main {  
    int x = 6;  
  
    public static void main(String[] args) {  
        Main myObj1 = new Main(); // Object 1  
        Main myObj2 = new Main(); // Object 2  
        myObj2.x = 36;  
        System.out.println(myObj1.x);  
    }  
}
```



```
        System.out.println(myObj2.x);  
    }  
}
```

çıktı: 6

36

Şeklinde olacaktır.

Çoklu Nitelikler:

İstenilen sayıda özellik belirtilebilir.

```
public class Main {  
    String fname = "Serra";  
    String lname = "Vuslat";  
    int age = 22;  
  
    public static void main(String[] args) {  
        Main myObj = new Main();  
        System.out.println("Name: " + myObj.fname + " " + myObj.lname);  
        System.out.println("Age: " + myObj.age);  
    }  
}
```

Java Sınıfı Yöntemleri:

Bir yöntem oluşturalım:

```
public class Main {  
    static void myMethod() {  
        System.out.println("Serra Vuslat");  
    }  
}
```

myMethod arayalım ():

```
public class Main {  
    static void myMethod() {  
        System.out.println("Serra Vuslat");  
    }  
  
    public static void main(String[] args) {  
        myMethod();  
    }  
}
```

Çıktı: Serra Vuslat AKYÜZ

Şeklinde olacaktır.

Statik ve Genel:

Genelde statik veya public niteliklerine ve yöntemlerine göre değerlendireceğiz.

Static ve public farklarını simgeleyen bir örnek :

```
public class Main {  
  
    static void myStaticMethod() {  
        System.out.println("Statik alan");  
    }  
  
    public void myPublicMethod() {  
        System.out.println("Public alan");  
    }  
  
    public static void main(String[] args) {  
        myStaticMethod();  
  
        Main myObj = new Main();  
        myObj.myPublicMethod();  
    }  
}
```

Bir Nesneyle Erişim Yöntemleri:

wealther()mycloud

```
public class Main {
```

```
    public void rain() {
```

```
        System.out.println("The weather is rainy today!");
```

```
    }
```

```
    public void wealther(int maxrain) {
```

```
        System.out.println("Wealther: " + weather);
```

```
    }
```

```
    public static void main(String[] args) {
```

```
        Main mywealther = new Main();
```

```
        mycloud.rain();
```

```
        mywealther.(1);
```

```
    }
```

```
}
```

Birden Çok Sınıfı Kullanma:

Main.java

İkinci.java

Main.java

```
public class Main {  
    public void car() {  
        System.out.println("the fast")  
    }  
  
    public void speed(int maxSpeed) {  
        System.out.println("Max speed is: " + maxSpeed);  
    }  
}
```

```
class Second {  
    public static void main(String[] args) {  
        Main myCar = new Main();  
        myCar.car();  
        myCar.speed(150);  
    }  
}
```

her iki dosya da derlendiğinde:

```
C:\Users\Your Name>javac Main.java
```

```
C:\Users\Your Name>javac Second.java
```

Ve çıktı şöyle olacaktır:

```
The car is going as fast as it can!
```

```
Max speed is: 150
```

Erişim Belirleyiciler:

Sınıfların, metodların ve alanların erişilebilirliğini kontrol etmek için kullanılan başlıca dört tür erişim belirleyici vardır:

public: Her yerden erişilebilir.

private: Sadece tanımlandığı sınıfın içinden erişilebilir.

protected: Aynı paket içerisindeki sınıflardan ve alt sınıflardan erişilebilir.

default (belirleyici yoksa): Aynı paket içerisindeki sınıflardan erişilebilir.

1-Public Erişim Belirleyici Örnek Kullanım:

```
public class PublicClass {  
  
    public int publicVar = 9;  
  
    public void publicMethod() {
```

```
        System.out.println("Public method");
    }
}
```

```
public class TestPublic {
    public static void main(String[] args) {
        PublicClass obj = new PublicClass();
        System.out.println(obj.publicVar);
        obj.publicMethod();
    }
}
```

2-Private Erişim Belirleyici:

```
public class PrivateClass {
    private int privateVar = 7;

    private void privateMethod() {
        System.out.println("Private method");
    }

    public void accessPrivate() {
        System.out.println(privateVar);
        privateMethod();
    }
}
```

```
    }  
}
```

```
public class TestPrivate {  
    public static void main(String[] args) {  
        PrivateClass obj = new PrivateClass();  
        obj.accessPrivate();  
    }  
}
```

3-Protected Erişim Belirleyici:

```
public class ProtectedClass {  
    protected int protectedVar = 5;  
  
    protected void protectedMethod() {  
        System.out.println("Protected method");  
    }  
}
```

```
public class TestProtected extends ProtectedClass {  
    public static void main(String[] args) {  
        ProtectedClass obj = new ProtectedClass();  
        System.out.println(obj.protectedVar);  
    }  
}
```



```
        obj.protectedMethod();  
    }  
}
```

4-Varsayılan (Default) Erişim Belirleyici:

```
class DefaultClass {  
    int defaultVar = 3;  
  
    void defaultMethod() {  
        System.out.println("Default method");  
    }  
}
```

```
public class TestDefault {  
    public static void main(String[] args) {  
        DefaultClass obj = new DefaultClass();  
        System.out.println(obj.defaultVar);  
        obj.defaultMethod();  
    }  
}
```

Abstarct:

Soyut Sınıflar ve Yöntemler:

Belirli ayrıntıların gizlenmesi ve istenilen bilgilerin gösterilmesi işlemidir.

-Erişim dışı bir değiştiricidir:

Soyut sınıf: Erişim için bir sınıftan miras alma durumu olması gerekir.

Soyut yöntem: Sadece devralınan sınıftan gövde sağlanır kendine ait gövdesi yoktur soyut kullanımı vardır.

```
abstract class Car {  
  
    public abstract void CarSpeed();  
  
    public void () {  
  
        System.out.println("150km/s");  
  
    }  
  
}
```

Yukarıdaki örnekte Car sınıfından bir nesne oluşturmak mümkün değildir:

Error verecektir.

Interface - Arayüz :

Soyut yapıların ve statik sabitlerin olduğu yapılardır. Arayüzdeki tüm yöntemler, default soyut ve geneldir.

Nesne oluşturmak için kullanılamaz.

Bir arayüzün uygulanmasında, onun tüm yöntemlerini geçersiz kılmalısınız.

Arayüzler Neden ve Ne Zaman Kullanılmalı?

-Güvenlik ve miras için kullanılmalıdır

```
interface Animal {  
    public void animalSound();  
    public void run();  
}
```

-

```
interface Animal {  
    public void animalSound();  
    public void sleep();  
}
```

```
class Snake implements Animal {  
    public void animalSound() {  
        System.out.println("Sss");  
    }  
    public void dog() {  
        System.out.println("Havhav");  
    }  
}
```

```
class Main {  
  
    public static void main(String[] args) {  
  
        Dog myDog = new Pig();  
  
        myDog.animalSound();  
  
        myDog.sleep();  
  
    }  
}
```

Çoklu Arayüzler:

```
interface FirstInterface {  
  
    public void myMethod();  
  
}
```

```
interface SecondInterface {  
  
    public void myOtherMethod();  
  
}
```

```
class DemoClass implements FirstInterface, SecondInterface {  
  
    public void myMethod() {  
  
        System.out.println("Life is short...");  
  
    }  
  
    public void myOtherMethod() {
```

```
        System.out.println("Birds fly...");
    }
}

class Main {

    public static void main(String[] args) {

        DemoClass myObj = new DemoClass();

        myObj.myMethod();

        myObj.myOtherMethod();

    }

}

çıktı: Life is short...

        Birds fly...
```

Dünyadaki Suları İçinde Barındıran Java Sınıfları Projesi

1-Main.java:

Suları temsil eden nesneleri oluşturur .

```

public class Main {
    public static void main(String[] args) {

        Sea sea = new Sea("Karadeniz Sea", 20000, 1000, true);

        Lake lake = new Lake("Como Lake ", 7000, 500, true);

        River river = new River("Van River", 80000, 60, 400);

        System.out.println(sea);
        System.out.println(lake);
        System.out.println(river);
    }
}

```

2-WaterBody.java:

Suları İçinde Barındıran gövdedir.

```

public class WaterBody {
    public String name;
    public String area;
    public String depth;

    public WaterBody(String name, String area, String depth) {
        this.Name = name;
        this.Area = area;
        this.Depth = depth;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.Name = name;
    }

    public double getArea() {
        return area;
    }

    public void setArea(double area) {
        this.Area = area;
    }

    public double getDepth() {
        return depth;
    }

    public void setDepth(double depth) {
        this.depth = depth;
    }

    public String toString() {
        return "Name: " + name + ", Area: " + area + " km², Depth: " + depth + " meters";
    }
}

```

3-Sea.java

Denizleri temsil eden sınıftır.

```
private class Sea extends WaterBody {
    private boolean saltWater;
    private Sea(String name, String area, String depth, boolean saltWater) {
        super(name, area, depth);
        this.saltWater = saltWater;
    }
    private boolean saltWater() {
        return saltWater;
    }
    private void setHasSaltWater(boolean saltWater) {
        this.saltWater = saltWater;
    }
    -Override
    private String toString() {
        return super.toString() + ", Salt Water: " + (saltWater ? "Yes" : "No");
    }
}
```

4-Lake.java

Göl sınıfını temsil eder.

```
private class Lake extends WaterBody {
    private boolean isWater;
    private Lake(String name, string area, string depth, boolean isFreshWater) {
        super(name, area, depth);
        this.isWater = isWater;
    }
    private boolean isWater() {
        return isWater;
    }
    public void setIsWater(boolean isWater) {
        this.isWater = isWater;
    }
    -Override
    private String toString() {
        return super.toString() + ",Water: " + (isWater ? "Yes" : "No");
    }
}
```

Her sınıf, temel sınıftan miras alır ve kendine özgü sınıfsal özelliklerini ekler.

Teşekkürler