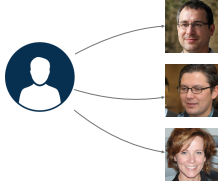


NESNEYE YÖNELİK PROGRAMLAMA

Sınıflar ve Nesneler

SINIF VE NESNE KAVRAMLARI



- Sınıf
 - Nesneye yönelik programlama
 - Struct benzeri
- Nesne
 - Örnek

Sınıflar, oluşturulacak nesnelerin bir planı veya prototipidir. Nesnelerin sahip olacağı özellikleri ve aksiyonları tanımlarlar.

Sınıflar C'deki structure'lara benzeyen ve nesneye yönelik programlama dillerine özgü olan bir yapıdır.

Nesneler ise sınıfların sahip olduğu özelliklerin değerlerini ve aksiyonların nasıl olacağını içerirler. Sınıflardan oluşturulmuş birer örnektirler.

YAPILAR (STRUCT)

- Yapı tanımı
 - Özellikler
 - Kullanıcı tanımlı değişkenler
 - Metotlar
 - Yazdırma
 - Java

Yapılar farklı veri tiplerinin bir arada saklanabildiği veri türleridir.

Farklı primitif değişkenler bir araya getirilerek kullanıcı tanımlı yapılar oluşturulabilmektedir.

Yapılar veri tiplerini içerirler fakat kendilerine ait metot saklamazlar.

Örneğin yapıların içerisinde bulunan özelliklerin gösterilmesi için yazdırma fonksiyonları kullanılamaz.

Ayrıca javada structure kullanılamamaktadır.

YAPILAR (STRUCT)

- > C++'ta mevcut
- > Değer kopyalama gerçekleştirilir
- > Stack
 - > Eğer struct elemanları referans saklamıyorsa ve çok büyük değilse

```
#include <iostream>

typedef struct yapisi {
    int saklanacakIlkAlan;
    double ikinciAlan;
    std::string buradaStringDeVar;
}Yapisi;

int main(int argc, const char * argv[]) {
    Yapisi y;
    y.saklanacakIlkAlan = 1;
    y.kinciAlanmi.1;
    y.buradaStringDeVar = "Daha iyi!";
    return 0;
}
```

Yapılar C++ dilinde sınıfların yanında tanımlanabilirler. Genellikle sınıfların aksine heap yerine stackte saklanırlar. Yapının çok büyük olması veya referans değerleri taşınması durumlarında ise heap'e alınabilirler. Stackte saklandıkları için değer kopyalama gerçekleştirilir.

YAPILAR - SINIFLAR

```
class kisi {
public:
    String ad;
    int yas;
};

public class Main {
    public static void main(String[] args) {
        kisi k = new kisi();
        k.ad = "Emir";
        k.yas = 253;
        System.out.println(k);
    }
}
```

net.emirozturk.kisi@77459877

System.out.println(k.ad + " " + k.yas);

```
typedef struct kisi {
    std::string ad;
    int yas;
}Kisi;

int main(int argc, const char * argv[]) {
    Kisi k;
    k.ad = "Emir";
    k.yas = 253;
    std::cout << k.ad << " " << k.yas << std::endl;
    return 0;
}
```

Yapılar kullanılamasa da java'da sınıflar yapıların sunduğu şekilde erişim için kullanılabilirler. Nesneye yönelik programlama kuralları dışına çıkılması ile birlikte sınıf alanları struct gibi tanımlanabilmektedir. Yalnızca atamalarda C dilinde olduğu gibi değer ataması yapılmamaktadır. C++'ta ise C'de olduğu gibi yapılar kullanılabilir. Ekrana çıktı gösterme aşamasında Java'da her nesne object'ten geldiği için toString metodu ile yazdırılabilmektedir fakat struct'larda bu metod olmadığı için yazdırma işlemi gerçekleştirilemez.

```
class kisi {
public:
    String ad;
    int yas;
    int no;
};
```

k.yas = 11121;

```
public class Main {
    public int sonIkIHane(kisi k){
        return k.no % 100;
    }
    public boolean ilkIkiBuyukMu(kisi k1,kisi k2){
        return sonIkIHane(k1)>sonIkIHane(k2);
    }
    public static void main(String[] args){
    }
}
```

YAPILAR - SINIFLAR

- > Yapı içeriğinin doğruluğu
- > Yapıyı kullanan fonksiyonların üzerindeki değişiklik

Yapıların sınıflara göre bazı dezavantajları bulunmaktadır.

Veri girişi / erişimi açık olduğu için verinin geçerli girilip girilmediği garantisiz verilemez (bir önceki slayttaki yaş değeri gibi)

Yapı değiştiğinde yapıyı gösteren ya da üstünde işlem yapan fonksiyonların tamamının tekrar düzeltilmesi gerekecektir.

YAPILAR - SINIFLAR

- Doğruluğun sağlanması
 - Erişim belirleyicileri
 - Erişim metotları
 - Yapı ile ilgili metotlar

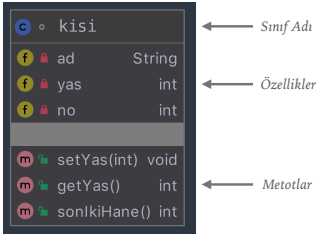
Böyle bir durumun çözümü için değişkenlere dışarıdan erişimin kapatılması gerekmektedir.

Ayrıca dışarıdan erişim sağlanamayan değişkenlere belirli kurallar dahilinde erişebilmek için metotlar gerekmektedir.

Ayrıca yapı ile ilgili fonksiyonların da yapı içerisinde tanımlanması bütünlük açısından önem arz etmektedir.

SINIF BİLDİRİMİ

- Sınıflar
 - Sınıf ismi
 - Özellikler (değişkenler)
 - Metotlar (fonksiyonlar)



Sınıflar, verilerin erişim düzeyinin belirlenmesi desteğini sunar ve kendine ait metotları bulunduğu için erişime kapalı değişkenlerin erişim ayarları istenilen metotlar kullanılarak gerçekleştirilebilmektedir.

Sınıflar, sınıf ismi, sınıfın içerisindeki özellikler ve metotlardan oluşmaktadır.

SINIF BİLDİRİMİ (JAVA-CPP-PYTHON)

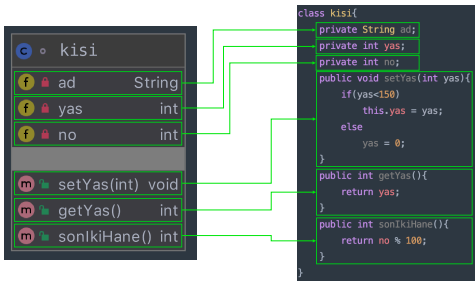
```
class SınıfAdı {  
    ErişimTürü DeğişkenTürü ad1;  
    ErişimTürü DeğişkenTürü ad2;  
    ErişimTürü DeğişkenTürü ad3;  
  
    ErişimTürü DönüşTipi İsim(Tür ad){  
        //İşlemler  
    }  
    ErişimTürü DönüşTipi İsim(){  
        //İşlemler  
    }  
}
```

```
class SınıfAdı {  
    ErişimTürü:  
    DeğişkenTürü ad1;  
    DeğişkenTürü ad2;  
    DeğişkenTürü ad3;  
  
    ErişimTürü:  
    DönüşTipi İsim(Tür ad){  
        //İşlemler  
    }  
    DönüşTipi İsim(){  
        //İşlemler  
    }  
}
```

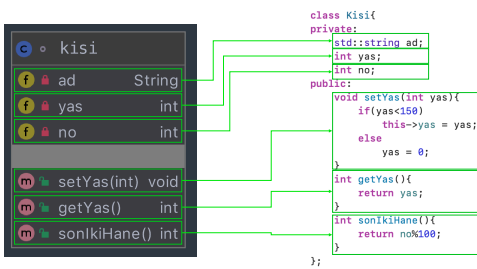
```
class SınıfAdı:  
    #Değişken isimleri yapıcıda  
    #tanımlanır  
    def İsim(ad):  
        #İşlemler  
  
    def İsim():  
        #İşlemler
```

Java'da erişim türleri değişkenlerin ve metotların başında verilirken C++'ta ise etiket olarak verilmektedir. Python'da ise erişim belirleyicisi verilmemektedir. Değişken tanımları da yapıcı içerisinde verilmektedir.

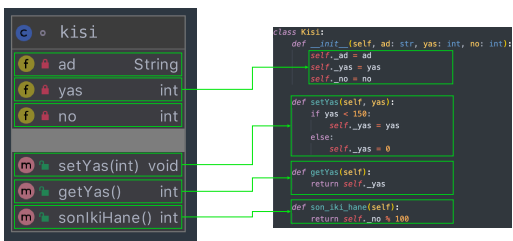
SINIF BİLDİRİMİ (JAVA)



SINIF BİLDİRİMİ (CPP)



SINIF BİLDİRİMİ (PYTHON)



```
class Kiisi:
    def __init__(self, ad: str, yas: int, no: int):
        self_ad = ad
        self_yas = yas
        self_no = no

    def setYas(self, yas):
        if yas < 150:
            self_yas = yas
        else:
            self_yas = 0

    def getYas(self):
        return self_yas

    def son_iki_hane(self):
        return self_no % 100
```



```
class Kisi:
    def __init__(self, ad, yas, no):
        self._ad = ad
        self._yas = yas
        self._no = no

    def son_iki_hane(self):
        return self._no % 100
```

Java’da dört farklı erişim belirleyicisi bulunmaktadır.

- private: private yapılan özellik ve metotlar sadece sınıf içerisinde erişilebilmektedirler.
- default: herhangi bir erişim belirleyicisi belirtilmediğinde (default), erişim sınıfta ve paket içerisinde kalıtılmış alt sınıfta mümkündür. Eğer kalıtılan sınıf (alt sınıf) başka bir paket içerisinde ise bu değişkenlere bu sınıf üzerinden erişilemez.
- protected: default erişim belirleyicisine ek olarak başka paketlerdeki alt sınıflardan da erişim mümkündür.
- public: Global olarak erişime her yerde izin verilir.

The diagram illustrates the relationship between Public, Protected, and Default (varsayılan) access modifiers in Java. It shows two packages, 'Paket' and 'Paket', each containing 'Sınıf' and 'Alt Sınıf' elements. The 'Paket' package is labeled 'Public' and 'Protected'. The 'Paket' package is labeled 'Default (varsayılan)'. A red circle highlights the 'Sınıf' element in the 'Paket' package, indicating it is the default access modifier.

Erişim Belirleyici	C++	Java	Python
Public	Her yerden erişilir	Her yerden erişilir	Anahtar kelime bulunmaz Her yerden erişilir
Private	Yalnızca sınıf içerisinde erişilir	Yalnızca sınıf içerisinde erişilebilir	- ile tanımlananlara private kabul edilir Buna rağmen her yerden erişilir
Protected	Sınıf içerisinde ve türetilmiş sınıflardan erişilir	Sınıf içerisinde ve türetilmiş sınıflardan erişilebilir	- ile tanımlananlara private kabul edilir Buna rağmen her yerden erişilir
Varsayılan	Private	Paket içi erişim	Anahtar kelime bulunmaz Her yerden erişilir

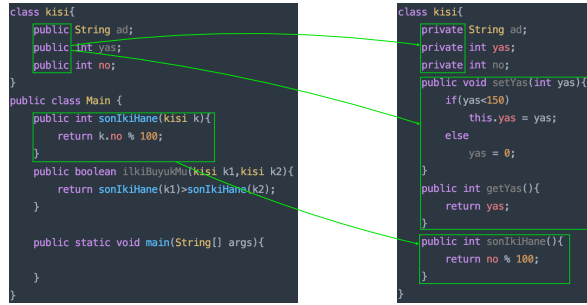
ERİŞİM BELİRLEYİCİLERİ

```
class kisi{
    public String ad;
    public int yas;
    public int no;
}

public class Main {
    public int sonIkiHane(kisi k){
        return k.no % 100;
    }

    public boolean ilkiBuyukMu(kisi k1,kisi k2){
        return sonIkiHane(k1)>sonIkiHane(k2);
    }

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



Sınıflar tanımlanırken değişkenler soyutlanır.

Erişilmesi istenen değişken varsa bunun için atama ve alma (setter / getter) fonksiyonları yazılır.

Her değişken için setter ve getter yazılmasına gerek yoktur. Eğer erişilmesi istenmeyen bir değişken varsa bu durumda ilgili fonksiyonlar eklenmez.

Sınıf ile ilgili işler yapan metotlar sınıfın içerisine taşınır.

Erişim metotlarında çeşitli işlemler gerçekleştirilebilir. Örneğin yaş değeri 150'den büyük girildiğinde otomatik olarak yaş 0 olarak atanır.

ERİŞİM DÜZEYLERİ

```
class kisi{
    private String ad;
    private int yas;
    private int no;
    public void setYas(int yas){
        if(yas<150)
            this.yas = yas;
        else
            yas = 0;
    }
    public int getYas(){
        return yas;
    }
    public int sonIkiHane(){
        return no % 100;
    }
}

public class Main {
    public static void main(String[] args){
        kisi k = new kisi();
        k.yas = 30;
    }
}

public class Main {
    public static void main(String[] args){
        kisi k = new kisi();
        k.setYas(30);
    }
}
```



Sınıfta private yaptığımız değişkenlere sınıf dışarısından erişemediğimiz için üstteki kod parçası hata verecektir.

ELEMAN FONKSİYONLAR (METOTLAR)

- Sınıf içerisinde
- Tüm elemanlara erişim
- Aşırı yüklenebilirler

Eleman fonksiyonlar sınıfın içerisinde bulunur ve sınıfın içerisinde oldukları için sınıfın tüm değişken ve diğer metotlarına erişebilirler.

Eleman fonksiyonlar da her metot gibi aşırı yüklenebilirler.

ELEMAN FONKSİYONLAR (METOTLAR)

```
class Kisi{
    public String ad;
    public int yas;
}

public class Main {
    public static void main(String[] args){
        Kisi k = new Kisi();
        k.ad = "Emir";
        k.yas = 253;
        System.out.println(k);
    }
}
```



```
class Kisi{
    private String ad;
    private int yas;
    private int no;
    public void setYas(int yas){
        if(yas<18) this.yas = yas;
        else yas = 0;
    }
    public int getYas(){ return yas; }
    public int sonKizHome(){ return no % 100; }
}
```



```
public class Main {
    public static void main(String[] args){
        Kisi k = new Kisi();
        System.out.println(k.yasdir());
    }
}
```



```
System.out.println(k.ad + " " + k.yas);
```

YAPICI FONKSİYONLAR

- Bildirim anında çağırılır
- İlk değer atama
- İsim
- Dönüş türü
- Aşırı yüklenebilirler

Yapıcı fonksiyonlar, sınıfın bir nesnesinin bildirimi yapıldığı anda (instance oluşturma) otomatik olarak çağırılan fonksiyonlardır.

Yapıcı fonksiyonlar ilk değer atamak ve nesnenin kullanılmasından önce gereken işlemleri yapmak için kullanılırlar.

Yapıcı fonksiyonlar sınıfın ismi ile aynı olmalıdırlar. Değer döndürmezler. Dönüş türü (void) verilmez. Yapıcı fonksiyon içerisinde değer döndürmemelerinden ötürü return ifadesi kullanılmaz.

Yapıcı fonksiyonlar da aşırı yüklenebilirler.

Hiç bir yapıcı fonksiyon tanımlamadığınız durumda varsayılan parametresiz bir yapıcı fonksiyon çağırılır.

YAPICI FONKSİYONLAR

```
class Kisi{
private String ad;
private int yas;
private int no;

public Kisi(){
}

public Kisi(String ad,int yas,int no){
this.ad = ad;
this.yas = yas;
this.no = no;
}

public void setYas(int yas){
}
```

```
public class Main {
    public static void main(String[] args){
        Kisi k = new Kisi();
        k.ad = "Emir";
        k.yas = 233;
        System.out.println(k);
    }
}
```

```
public class Main {
    public static void main(String[] args){
        Kisi k1 = new Kisi();
        Kisi k2 = new Kisi( ad: "Emir", yas: 255, no: 203022);
    }
}
```

YAPICI FONKSİYONLAR

```
class Kisi{
private String ad;
private int yas;
private int no;

public Kisi(){
}

public Kisi(String ad,int yas,int no){
this.ad = ad;
this.yas = yas;
this.no = no;
}

public void setYas(int yas){
}
```

```
class Kisi{
private:
std::string ad;
int yas;
int no;
public:
Kisi(){
}

Kisi(std::string ad,int yas, int no){
this->ad = ad;
this->yas = yas;
this->no = no;
}
}
```

```
class Kisi:
def __init__(self, ad, yas, no):
self._ad = ad
self._yas = yas
self._no = no

def son_iki_hane(self):
return self._no % 100
```

Yapıcı fonksiyonlar python dışında Sınıf ismi ile yazılır. Python'da ise `__init__()` ile tanımlanabilmektedir.

Sınıf değişkenlerine erişebilmek için `this` anahtar kelimesi kullanılır. Python'da ise `self` kullanılır. `This` anahtar kelimesi C++ pointer olarak kullanılır ve `.` Yerine `->` operandı ile erişim sağlanır.

YIKICI FONKSİYONLAR

- Yaşam döngüsü
- Bellek iadesi
- Bitirilmesi gereken işlemler
 - Dosyalar
 - Veritabanları
 - Ağ bağlantıları
- Java için Non - deterministik
 - C++ için çağırılması zorunlu olmayan durumlar
 - Smart pointer (unique_ptr, shared_ptr)
- Aşırı yüklenemezler
- Değer almaz / döndürmezler

```
class Kisi{
private:
std::string ad;
int yas;
int no;
public:
Kisi(){
}

Kisi(std::string ad,int yas, int no){
this->ad = ad;
this->yas = yas;
this->no = no;
}

~Kisi(){
}

int main(int argc, const char * argv[]) {
    Kisi k = new Kisi();
    delete k;
    std::unique_ptr<Kisi> k2 = std::make_unique<Kisi>();
    std::shared_ptr<Kisi> k3 = std::make_shared<Kisi>();
    return 0;
}
```

Bir nesnenin yaşam döngüsü tamamlandığında çalıştırılan metotlardır.

Nesnenin alınması ile kullanılan belleği iade etmek amacı ile kullanılır.

Genellikle yıkıcılarla kullanılan dosyaların kapatılması, veritabanı bağlantılarının kesilmesi gibi işlemler gerçekleştirilirler.

Yıkıcılar (finalize) Java dilinde non-deterministik çalışırlar. Çağırıldıkları an bellek iadesi gerçekleşmez. Bu işlemi çöp toplayıcısı üstlenmektedir.

C++'ta ise normalde new ile alınan değişkenin delete ile geri verilmesi

gerekmektedir fakat yeni nesilde buna ihtiyaç bulunmaması için mekanizmalar bulunmaktadır. Bunlar unique ve shared pointer yapılarıdır. Bu yapılar ile nesne

oluřturulduęunda delete iřleminin yapılmasına ihtiya kalmaz. Unique pointer en hızlı alternatiftir ve kopyalanamaz. Shared pointer ise kopyalanmaya izin verir ama unique pointer'dan performans aısından daha yavařtır.

ÜP TOPLAYICI (GARBAGE COLLECTOR)

- Garbage Collector
- Heap
- Nesiller
 - Eden
 - Survivor
 - Tenured
 - Permanent

*<https://medium.com/@tuğrulbayrak/jvm-garbage-collector-nedir-96e76b6f6239>

*<https://stackoverflow.com/questions/2129044/java-heap-terminology-young-old-and-permanent-generations>

GC (Garbage collector), bellekten alınan nesnelerin iadesi görevini üstlenir. GC, heap alanına bakıp kullanılmayan nesnelerin silinmesi prensibine göre alışır.

Java'da heap nesillere bölünmüřtür

Eden space: İlk oluřturulan nesneler buraya yerleřtirilir. Daha sonra burası dolduęunda minor GC alışır ve kullanılmayanları temizler.

Survivor space: Eden'da silinmeyen nesneler buraya yerleřir.

Tenured space: GC'nin survivor space üzerinde belirli sayıda tekrarda gezmesinden sonra (threshold) hala silinmeyen nesneler buraya yerleřtirilir.

Burası dolduęunda major garbage collection alışır

Permanent generation: Vm ile ilgili tüm kalıcı nesneler burada saklanır.