www.youtube.com/BMdersleri

MAKÜ
BURDUR MEHMET AKİF ERSOY ÜNİVERSİTESİ

Flutter ile Mobil Programlamaya Giriş

Flutter

5.HAFTA
Class&Objects ve Listeler

Hazırlayan : **Zeynep İrem KESLER 1911404048**

Tarih : 22/03/2022

Sürüm : v1

Ders Yürütücüsü : Doç. Dr. İsmail KIRBAŞ

İÇİNDEKİLER

Class & Objects Yapısı

Class kullanımı, bir nesnenin şablonunu oluşturmaya benzer. Oluşturulan bu şablon sayesinde, yeni nesneler türetip bu nesnelere verilen değerler sayesinde de istenilen özelliklerde nesneler oluşturulması sağlanır. Unutulmaması gerekir ki dart programlama dilinde her şey nesnedir. Bir class içinde olabilecek yapılar; "değişkenler (instances veriable) " , kendi adında " constructors (yapıcılar)" ve " metotlardır (fonksiyonlar) " .

```
class ClassAdi {
Komutlar
}
```

NOT: Class adları büyük harfle başlar.

Class & Objects Yapısı

```
void main() async {
Kitap kitap1 = Kitap() ;
          print(kitap1.turu) ;
          print(kitap1.sayfaSayisi) ;
}
class Kitap {
           String turu = "Roman" ;
           int sayfaSayisi = 150 ;
}
```

Çıktı:

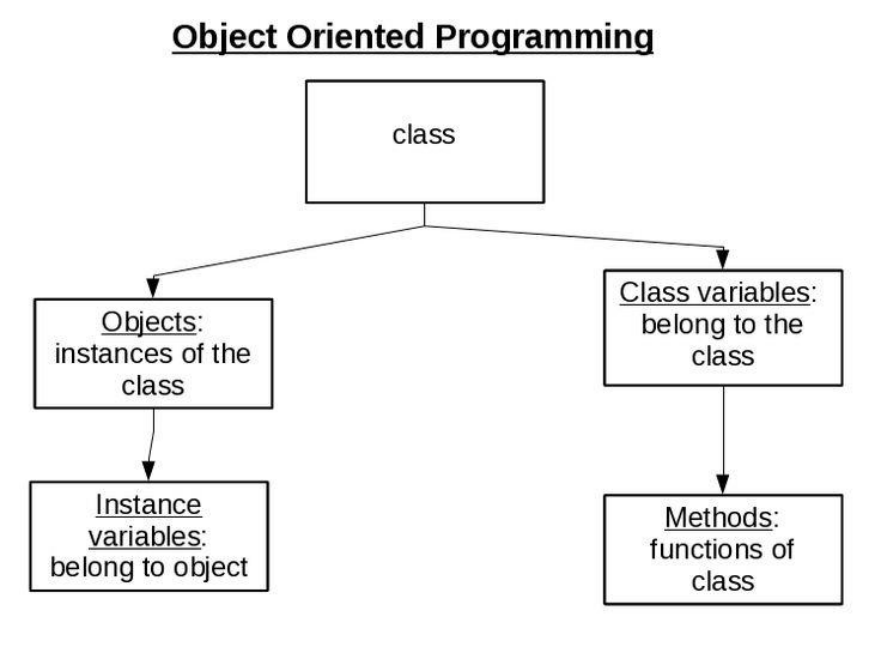
Roman

150

Class & Objects Yapısı

Yapıcılar – Constructor:

Constructor, sınıfların kurucu görevi taşıyan fonksiyonlarıdır. Bir sınıftan herhangi bir constructor kullanarak bir nesne oluşturulabilir. Bu nesneyi oluşturan fonksiyon, o sınıfın kurucu fonksiyonu olarak tanımlanır. Constructors ismi, class ismiyle aynı olmak zorundadır. Constructor sayesinde, sınıfın nesneleri kullanılarak birçok farklı veri oluşturulabilir. Class yapılarında kapsülleme işlemi de yapabilirsiniz.



Class & Objects Yapısı

Yapıcılar – Constructor:

```
void main() async {
          Kitap kitap1 = Kitap("Masal" , 20 ) ;
          print(kitap1.turu) ;
          print(kitap1.sayfaSayisi) ;
          print("***") ;
          Kitap kitap2 = Kitap("Ansiklopedi" , 300 ) ;
          print(kitap2.turu) ;
          print(kitap2.sayfaSayisi) ;
          print("***") ;
}
class Kitap {
           String? turu ;
           int? sayfaSayisi ;
Kitap(String turu , int sayfaSayisi) {       //Constructor
           this.turu = turu ;                                    //Yapıcıdaki değişken
değerlerini class içinde oluşturulan nesnelere atama
           this.sayfaSayisi = sayfaSayisi ;
           }
}
```

Çıktı:

```
Masal
20
***
Ansiklopedi
300
***
```

Class & Objects Yapısı

Kalıtım:

Bir alt sınıfın üst sınıfın özelliklerini kullanabilmesine olanak sağlayan durumlara **'kalıtım'** denir. Alt sınıf hangi sınıftan kalıtım alacaksa sınıfın isminden sonra **'extends'** yazılıp yanına da kalıtım aldığı sınıfın adı yazılır.

```
void main()  {
          SedanOtomobil oto1 = SedanOtomobil() ;
          oto1.boyaTuru() ;
}
class Otomobil {
           String? renk ;
           String? modelYili ;

           Otomobil( {this.renk , this.modelYili } ) ;
           boyaTuru() {
                print("Metalik boya") ;
           }
}
class SedanOtomobil extends Otomobil {
           sinifim() {
                 print ("Ben Otomobil sınıfından kalıtım alıyorum" ) ;
           }
}
```

Çıktı:

Metalik boya

Class & Objects Yapısı

Overriding:

Bir Overriding özelliği, kalıtım alan sınıfın üst sınıftan aldığı özelliği geçersiz sayıp kendi özelliğini kullanmasıdır. Örneğin üst sınıfta oluşturulmuş bir fonksiyon, kalıtım alan yani alt sınıfta da varsa o zaman kalıtım alan sınıf kendi fonksiyonunu kullanır. Üst Sınıfın fonksiyonu kendi içinde geçersiz olur.

```
void main() {
          SedanOtomobil oto1 = SedanOtomobil() ;
          oto1.tanimlama() ;
}
class Otomobil {
           String? renk ;
           String? modelYili ;

           Otomobil( {this.renk , this.modelYili } ) ;
tanimlama() {
           print("Ben otomobil sınıfıyım") ;
           }
}
class SedanOtomobil extends Otomobil {
           tanimlama() {
                 print ("Ben Otomobil sınıfının alt sınıfı Sedan Otomobil
sınıfıyım" ) ;
           }
}
```

Çıktı:

Ben Otomobil sınıfının alt sınıfı Sedan Otomobil sınıfıyım

Class & Objects Yapısı

İsimli Yapıcılar – (named constructor):

Bir sınıf içerisinde, sınıf ismi ile sadece bir tane yapıcı metot oluşturulabilir. Daha fazlası tanımlanırsa program hata verecektir. Bunun yerine dart dilinde **adlandırılmış yapıcı metotlar** kullanılır. İsimli yapıcı oluştururken önce sınıf adı yazılır, sonra nokta(.) koyarak isim verilir. Noktadan sonra yazılacak isim küçük harfle başlamalıdır.

```
void main() async {
          Otomobil oto1 = Otomobil(marka: "Renault" , model: "Clio" , modelYili: 2015 , renk: "Sarı") ;
          print(oto1.marka) ;
          print(oto1.model) ;
          print(oto1.modelYili) ;
          print(oto1.renk) ;
          print("***") ;

         Otomobil oto2 = Otomobil.sedan() ;
           print(oto2.marka) ;
          print(oto2.model) ;
          print(oto2.modelYili) ;
          print(oto2.renk) ;
          print("***") ;
}
 class Otomobil {
          String? marka ;
          String? model ;
          String? renk ;
          int? modelYili ;

          Otomobil( {this.marka , this.model , this.renk , this.modelYili } ) ;
          Otomobil.sedan() {
          marka = "Fiat";
          model = "Egea";
          renk = "Beyaz";
          modelYili = 2018;

          }
  }
```

Çıktı:

Renault
Clio
2015
Sarı
***
Fiat
Egea
2018
Beyaz

Class & Objects Yapısı

Factory Yapıcılar – (Yönlendirici Yapıcılar):

Factory, telefon santrali gibi düşünülebilir. Factory metodu, içinde bulunduğu sınıftan kalıtım alan diğer sınıflara yönlendirme yapıp türetilen nesne ya da nesneleri o sınıflardan oluşturmaya yarar. Bu yapıcıyı kullanabilmek için önce sınıfın yapıcısını oluşturmak gerekir.

```
void main() async {
        OtoGaleri galeri1 = OtoGaleri () ;
        OtoGaleri galeri2 = OtoGaleri.yonlendirme () ;

        print(galeri1) ;
        print(galeri2) ;
}
class OtoGaleri {
        OtoGaleri () ;
        factory OtoGaleri.yonlendirme() {
            return OtoGaleri() ;
        }
}
```

Çıktı:

Instance of 'OtoGaleri'

Instance of 'OtoGaleri'

Class & Objects Yapısı

Composition – Bir Sınıftan Başka Bir Sınıfa Erişim:

Bir sınıftan başka bir sınıfın içindeki nesnelere erişebilme olayın 'composition' denir. Factory yapıcılarından farkı alt sınıfı olmayan başka sınıflara erişebilmesidir.

```
void main() {
        Adres adres = Adres("Ankara" , "Çankaya") ;
        Musteri musteri = Musteri("Ali" , "Yıldız" , 25 , adres) ;

        print("Müşteri Adı: ${musteri.ad} ") ;
        print("Müşteri Soyadı: ${musteri.soyad} ") ;
        print("Müşterinin Yaşadığı İl: ${musteri.adres.il} ") ;
        print("Müşterinin Yaşadığı İlçe: ${musteri.adres.ilce} ") ;
}
class Adres {
         String il ;
         String ilce ;
         Adres(this.il , this.ilce) ;
}
class Musteri {
         String ad ;
         String soyad ;
         int yas ;
         Adres adres ;

         Musteri(this.ad, this.soyad, this.yas, this.adres) ;
}
```

Çıktı:

Müşteri Adı: Ali

Müşteri Soyadı: Yıldız

Müşterinin Yaşadığı İl: Ankara

Müşterinin Yaşadığı İlçe: Çankaya

Class & Objects Yapısı

'Const' ve 'Final' İfadeleri:

Her ikisi de tanımlandıktan sonra değişime uğramayan sabit değerler için kullanılan keywordlerdir. İkisinin temel farkı **zaman** diyebiliriz.

Const: Uygulamada derleme sırasında oluşturulan ve değerleri atanan sabit türlerdir. Bu değişken oluşturulurken değer ataması yapılır. Daha sonra bu değer değiştirilemez.

(Compile Time)

```
void main() {

const double pi = 3.14 ;

pi = 3 ; // değer değiştirilemez.
}
```

Class & Objects Yapısı

‘Const’ ve ‘Final’ İfadeleri:

Final: Uygulama çalışma sırasında oluşturulan sabit türdür. Final değişkeninin değeri sabit olarak doğrudan atanabilir, ya da sınıf yapıcı metodu içerisinde değeri atanmalıdır. Bir sınıftan yeni bir nesne oluştururken bu sabit değer gönderilerek değer atanabilir. Fakat bu değer atandıktan sonra değiştirilemez.

(Run Time)

```
void main() {

final time = DateTime.now() ;

print(" Zaman: $time") ;
}
```

Çıktı:

Zaman: 2022-03-15 21:34:37.138

Listeler

Birden fazla değeri ayrı ayrı değişkenler yerine tek bir değişken içinde tuttuğumuz yapılara liste denir. Çoğu programlama dillerinin aksine Dart dilinde dizi (array) kavramı yoktur. Dart dilinde dizi yerine de listeleri kullanırız.

List:

Listeler(List), belli bir veri türünden verileri bir arada tutan yapılardır.

List <Değişken_Türü> degisken_adi = [değerler]; → **Liste değerleri []** içinde girilir.

Örnek:

List <String> renkler = [“ Mavi” , “ Yeşil” , “ Kırmızı” , “ Lacivert”];

```
void main() {

List<String> liste = ["Muz" , "Portakal" , "Mandalina" ,
"Limon"] ;

print(liste);
}
```

Çıktı:

Zam[Muz, Portakal, Mandalina, Limon]

Listeler

```
void main() {

List liste = ["Muz" , "Portakal" , "Mandalina" , "Limon"
, 5, 8, true] ;

print(liste[0]);
print(liste[2]);
print(liste[4]);
}
```

Çıktı:

Muz

Mandalina

5

Listeler

Listelerin Metotları:

1- indexOf: Listedeki elemanın kaçıncı index numarasında olduğunu verir.

2- add: Listeye eleman ekler.

3- length: Listenin eleman sayısını verir.

4- insert: Listenin istenilen pozisyonuna yeni eleman ekler.

5- reversed: Listenin elemanlarını sondan başa doğru yazdırır. Yani listeyi tersine çevirir.

6- removeAt: Listenin istenilen index numaralı elemanını siler.

7- clear(): Listenin tüm elemanlarını siler. (***listem.clear()***)

Listeler

Map:

Map yapısı bir koleksiyon yapısıdır. Aynı zamanda bir sözlük gibi de düşünülebilir. Bu yapı sayesinde key ve value değerleri ile kolay bir tanım sağlanabilmektedir. Bu yapıda key karşısında bir value bilgisi vardır.

Map <String,String> sozluk = {“ book”: “ kitap” , “ door”: “ kapı” , “ red”: “ kırmızı” } ;

```
void main() {

Map<int,String> plakalar = {55: "Samsun" , 15: "Burdur"
, 07: "Antalya" ,  34: "İstanbul"} ;

print(plakalar);
}
```

Çıktı:

{55: Samsun, 15: Burdur, 07: Antalya, 34: İstanbul}

Listeler

Map yapısındaki key ile değeri ekrana yazdırma:

```
void main() {

Map<int,String> plakalar = {55: "Samsun" , 15: "Burdur"
, 07: "Antalya" ,  34: "İstanbul"} ;

print(plakalar[15]);
}
```

Çıktı:

Burdur

Listeler

Map yapısına yeni eleman ekleme:

```
void main() {

Map<int,String> plakalar = {55: "Samsun" , 15: "Burdur"
, 07: "Antalya" ,  34: "İstanbul " , 35: "İzmir"} ;

plakalar[35] = "İzmir" ;
print(plakalar) ;
}
```

Çıktı:

{55: Samsun, 15: Burdur, 07: Antalya, 34: İstanbul, 35: İzmir}

Listeler

Map yapısında key değeriyle istenilen bir elemanı silme:

```
void main() {

Map<int,String> plakalar = {55: "Samsun" , 15: "Burdur"
, 07: "Antalya" ,  34: "İstanbul " } ;

plakalar.remove(07) ;
print(plakalar) ;
}
```

Çıktı:

{55: Samsun, 15: Burdur, 34: İstanbul}

Listeler

Get ve Set İfadeleri:

Değişken üzerinde işlem yapılabilmesini sağlayan fonksiyonlardır. **Get** fonksiyonlar değer döndürürken; **Set** fonksiyonlar ise değer alır. Set'lerin listelerden farkı, aynı öğeden sadece bir tane içerebilmesidir.

```
void main() async {
          Isim isim = Isim() ;
          print(isim.kisininIsmi) ;

          isim.kisininIsmi = "Ali" ;
          print(isim.kisininIsmi) ;
}
class Isim {
          String adi = "Ahmet" ;
          String get kisininIsmi {
              return adi ;
          }
          set kisininIsmi(String yeniIsim) {
              adi = yeniIsim ;
          }
}
```

Çıktı:

Ahmet
Ali

Yardımcı Kaynaklar

- Adım Adım Flutter İle Mobil Uygulamalar (Rakıcı Oğuz , 2021)

İlginiz için teşekkürler…

Hazırlayan : **Zeynep İrem KESLER 1911404048**
E-posta : zeynepiremkesler@gmail.com
Tarih : 22/03/2022
Ders Yürütücüsü : Doç. Dr. İsmail KIRBAŞ
E-posta : ismkir@gmail.com