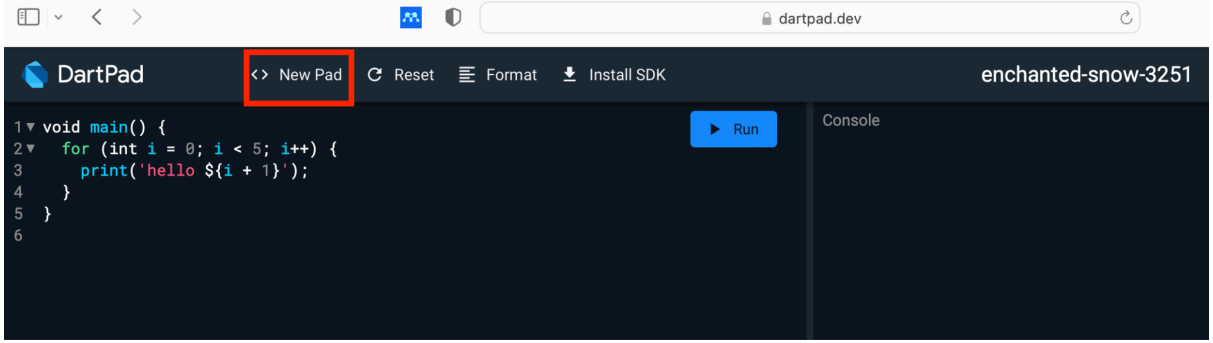


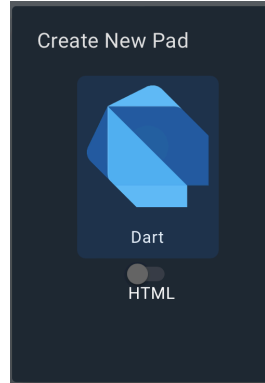
Dart ve Flutter'da Sınıf Yapısı

A) Hazırlık:

1. Sanal IDE olan <https://dartpad.dev/?> adresine **gidin** ve uygulama açıldığında **New Pad** butonuna tıklayın.



2. Dart dilinde kodlar yazacağımız için Create New Pad açılır penceresinde **Dart** seçeneğine tıklayın.



B) Temel Sınıf Yapısı

1. Diğer programlama dillerinde olduğu gibi Dart dilinde veri tutabileceğimiz en küçük yapı **variable (değişken)**'dir. Dartpad'de açılan örnekte de göreceğimiz üzere integer türündeki **i değişkenine** bir sayı atanmış ve bu sayı yazılan kod bloğunda içerisinde artırılarak kullanılmıştır.

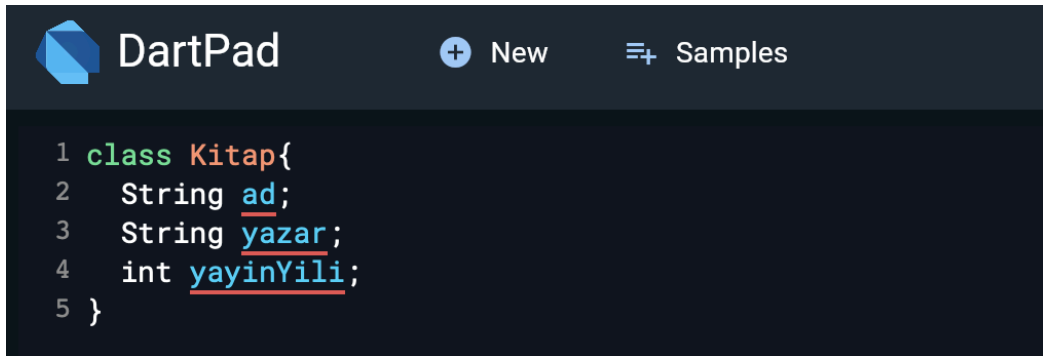
```
1 void main() {  
2   for (int i = 0; i < 10; i++) {  
3     print('hello ${i + 1}');  
4   }  
5 }  
6
```

2. Peki bu programlama dilinde verileri tutabileceğimiz başka yapılar nelerdir? **Sınıf (class)**, nesne yönelimli programlamanın temel yapı

taşlarından biridir ve benzer özelliklere ve davranışlara sahip nesneleri tanımlamak için kullanılır. Sınıflar, **veri (özellikler)** ve **işlevsellik (metotlar)** ile bir araya gelerek bir nesnenin temelini oluşturur.

3. Şimdi bu sınıf yapısına bir örnekle bakalım. Bunun için dartpad'te size sunulan tüm hazır kodları **silin**. Dart dilinde **Kitap** adında bir sınıf oluşturun. İsimlendirme için **PascalCase** isimlendirme standardı kullandığımızı unutmayın. Bu sınıf üç temel veri içermelidir:

- **ad**: Kitabın adını tutan String tipinde bir değişken.
- **yazar**: Kitabın yazarını tutan String tipinde bir değişken.
- **yayınYili**: Yayın yılını tutan int tipinde bir değişken (**camelCase**)



```
1 class Kitap{
2   String ad;
3   String yazar;
4   int yayınYili;
5 }
```

4. Sınıfımızı oluştururken bir hata ile karşılaştık. Bunun nedeni dart dilinin **Null safety** olmasıdır. Null safety değişkenlerin null değer alıp almayacağını belirleyen bir özelliktir ve **hatalarını önlemeyi** sağlayarak kodunuzu **daha güvenli** hale getirir. Bu hatanın giderilmesinin bir çok yolu vardır.

- a. **Null Olabilir Değişken Tanımlama**: Eğer bir değişkenin null olabileceğini belirtmek istiyorsanız, tipinin sonuna **soru işareti (?)** ekleyerek tanımlarsınız.



```
1 class Kitap{
2   String? ad;
3   String yazar;
4   int yayınYili;
5
6 }
```

- b. **Late Anahtar Kelimesi Kullanma:** Eğer bir değişkeni başta null tanımlayıp daha sonra değer vermek istiyorsanız, **late anahtar kelimesi** ile tanımlayabilirsiniz.

```
1 class Kitap{
2   String? ad;
3   late String yazar;
```

- c. **Yapıcı Metot (Constructor) Tanımlama:** Bir sınıftan (class) **yeni bir nesne** (object) oluşturulurken çağrılan özel bir metottur. Yapıcı metotlar, **nesnelerin başlatılması** ve **gerekli başlangıç değerlerinin** atanması için kullanılır.

```
1 class Kitap{
2   String ad;
3   String yazar;
4   int yayınYili;
5
6   Kitap(this.ad, this.yazar, this.yayınYili);
7 }
```

Not: Burada this anahtar kelimesi **sınıf üyelerini ayırt** etmek için kullanılır.

5. Şimdi sınıfa kitabın ad, yazar ve yayınYili özelliklerini ekrana yazdıran bir **yöntem (method)** ekleyin. Burada **fonksiyon yapısının sınıf yapısına benzediğini** ama farklı olduğunu, print gibi işlem yapan satırların sonunda **noktalı virgül (;)** noktalama işaretinin olduğunu, **interpolation** \$ad, \$yazar, ve \$yayınYili sınıfın özelliklerine (fields) işaret ettiğini ve string interpolation kullanarak bu değerlerin string içine yerleştirildiğini unutmayın.

```
6   Kitap(this.ad, this.yazar, this.yayınYili);
7
8   dynamic bilgiVer(){
9     print("Kitabın Adı: $ad, Yazarı: $yazar, Yayın Yılı: $yayınYili");
10  }
11 }
```

6. Dart derleyicisi (ya da çalışma zamanı), programı çalıştırırken ilk olarak **main fonksiyonunu** arar ve çalıştırır. Dolayısıyla şimdi yapacağımız işlem bu fonksiyonu oluşturmaktır. Bu fonksiyon herhangi **bir değer döndürmez**, bu nedenle dönüş tipi **void** olarak belirtilmelidir.

```
13 void main(){  
14  
15 }
```

7. Şimdi de main() fonksiyonu içinde oluşturduğumuz sınıftan **bir nesne** (object) oluşturalım. Bunun için öncelikle **Kitap1** isminde **bir sınıf örneği** (instance) oluşturalı ve **yapıcı metodu çağırarak** bu nesnenin ad, yazar, ve yayınYili özelliklerini sırasıyla belirtmeliyiz.

```
13 void main(){  
14  
15     Kitap Kitap1=Kitap("Sefiller", "Victor Hugo", 1962);  
16  
17  
18 }
```

8. Artık **kitap1** nesnesi üzerinde **bilgiVer()** yöntemini çağırabiliriz. Bu yöntem, nesneye ait ad, yazar, ve yayınYili özelliklerini kullanarak konsola bir mesaj yazar.

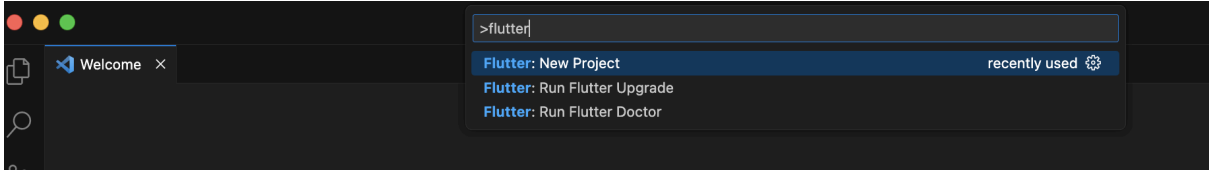
```
13 void main(){  
14  
15     Kitap Kitap1=Kitap("Sefiller", "Victor Hugo", 1962);  
16     Kitap1.bilgiVer();  
17  
18 }
```

C) Flutter'da Sınıf Kullanımı

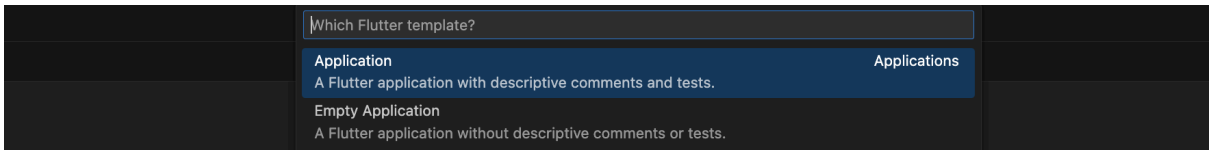
Eğer bilgisayarınızda Flutter için gerekli kurulumları henüz yapmadıysanız <https://dartpad.dev/?> adresine gidin ve **New Pad** diyerek yeni bir **flutter** projesi oluşturun. Kurulumları tamamlayanlar için:

Yeni Flutter Projesi Oluşturma ve Ortam Hazırlığı

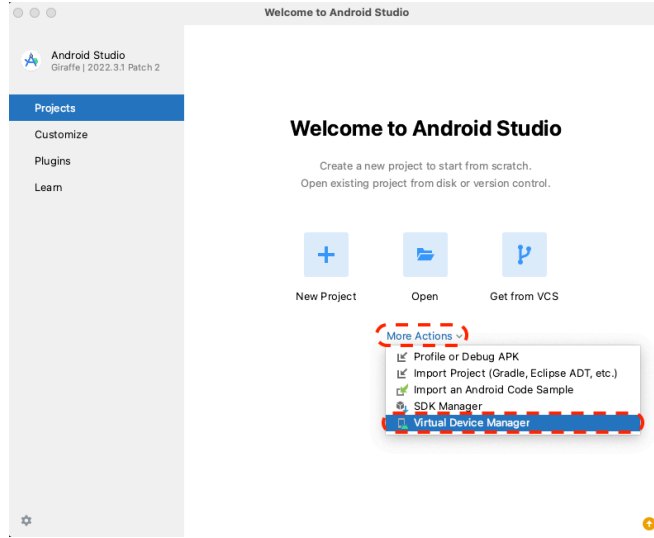
1. Bilgisayarınızda **Visual Studio Code** uygulamasını başlatın. Klavyenizde **Ctrl + Shift + P** tuşlarına aynı anda basın. Bu, **VS Code'un komut paletini** açar. Açılan komut paletinin arama çubuğuna **"Flutter"** yazın. Arama sonuçlarında **"Flutter: New Project"** seçeneğine tıklayın. Bu işlem yeni bir Flutter projesi oluşturmanızı sağlayacaktır.



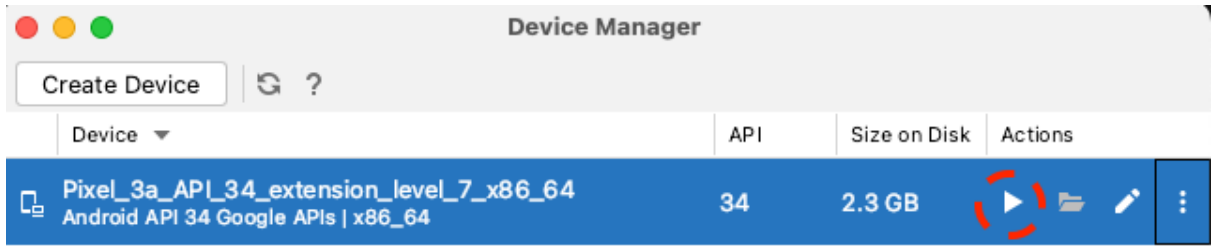
2. Which Flutter template? Sorusuna **uygulama (Application)** seçeneğini seçerek devam edin. Projeniz için uygun bir **isim girin** ve **"Enter"** tuşuna basın. Ardından, projenizin kaydedileceği **klasörü seçin**. Projeniz açıldığında "main.dart" dosyasının boş olmadığını, içerisinde çeşitli kodların bulunduğu göreceksiniz.



3. Ortam hazırlığına VS Code'da hazır olan bu kodların sonuçlarını görebilmek için **emülatörü** açmak ile devam edin. Bunun için bilgisayarınızda kurulu olan **android studio'yu** açın ve karşılama ekranında önce **More Actions** açılır menüsüne ardından da **Virtual Device Manager'a** tıklayın.

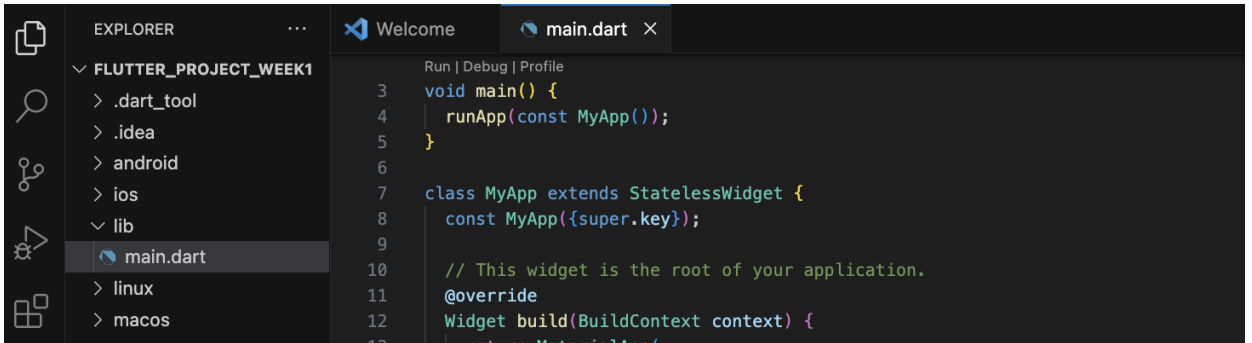


4. Açılan device manager ekranında çalışma yapmak istediğiniz emülatörü seçin ve **launch AVD butonuna** tıklayın. Bilgisayarınızın performansına bağlı olarak emülatörünüz bir süre sonra açılacaktır.



Flutter Dosyalama Yapısı

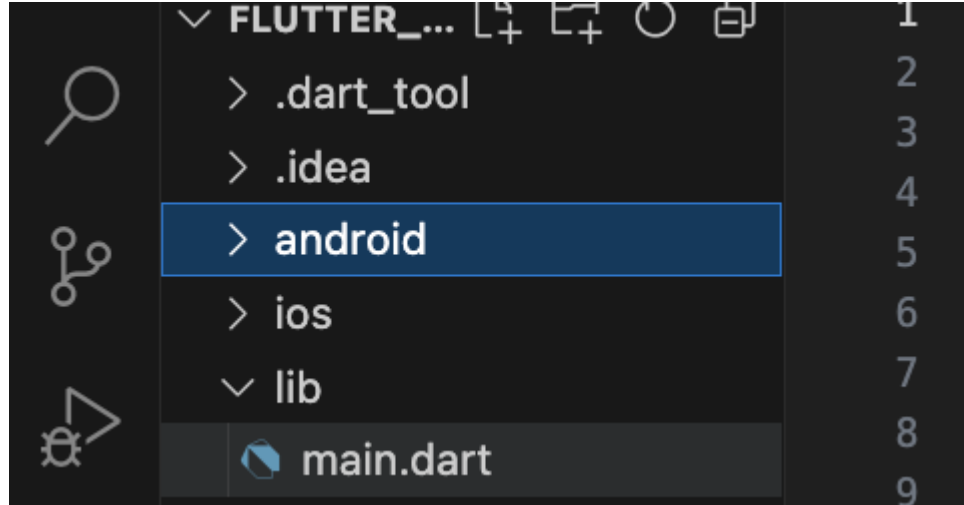
5. Kodları yazdığımız dosyanın ismi “**main.dart**” dosyasıdır ve bu dosya dizin ağacımızda **lib** klasörünün içinde bulunmaktadır. Bu dosya projenizi açtığınızda **otomatik** olarak açılmaktadır. Eğer **manuel** olarak açmanız gerekirse dizin ağacında lib klasörüne gidin ve main.dart dosyasına çift tıklayarak bu dosyayı açın.



6. Bir diğer önemli dosya proje yapılandırma dosyası olan **pubspec.yaml** dosyasını bulun ve açın. Bu dosya, projenizin genel bilgilerini ve proje için gerekli bağımlılıkları (dependencies) içerir. Dosyanın üst kısmında projenizin **adı**, **versiyonu**, ve **açıklaması** gibi temel bilgileri göreceksiniz. Bu bilgiler, projenizi tanımlamak ve versiyon kontrolünü sağlamak için kullanılır. Dosyanın alt kısmında **dependencies** başlığı bulunur. Bu bölüm, Flutter projenizde kullanacağınız **paket** ve **kütüphanelerin** listesini içerir. Varsayılan olarak burada Flutter SDK ve bazı temel kütüphaneler bulunur.

7. Flutter projesi oluşturduğunuzda proje dizininde **android** ve **ios** adlı iki klasör görürsünüz. Bu klasörler, projeye özgü yerel (native) projeleri içerir:

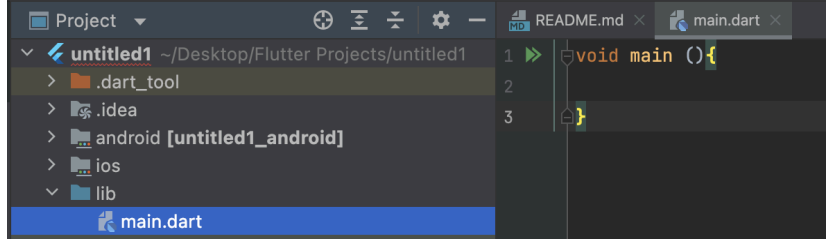
- ☐ **Android klasörü**, **Java** veya **Kotlin** dillerinde yazılmış Android işletim sistemiyle uyumlu native bir projeyi barındırır.
- ☐ **OS klasörü**, **Swift** veya **Objective-C** dillerinde yazılmış iOS işletim sistemiyle uyumlu native bir projeyi içerir.



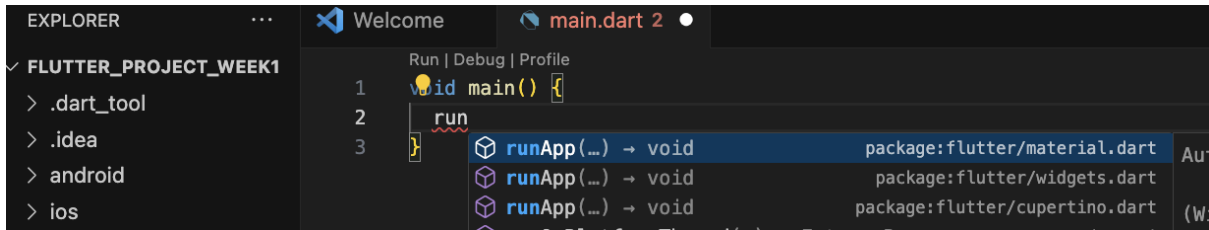
Flutter projesinde Dart dilinde yaptığınız değişiklikler, **android** ve **ios** klasörlerindeki native projelere otomatik olarak yansır. Örneğin, uygulamanızın arayüzü ve işlevselliği Dart kodu ile tanımlanır ve bu kod her platformda yerel bileşenler ile çalışacak şekilde Flutter tarafından güncellenir.

Flutter ile Sınıf Oluşturma

8. Proje dizininde bulunan **main.dart** dosyasını tekrar açın. Bu dosyasındaki mevcut tüm kodları **silerek** tamamen boş bir sayfa ile başlayın. Flutter uygulamaları, Dart dilinde yazıldığı için her şey Dart dilinde çalışır. Uygulamanın başlangıç noktası olan **main fonksiyonunu** yazarak işe başlayalım.



9. Flutter'da her şey bir **widget**dir. Bir **buton**, bir **metin**, hatta **uygulamanızın tamamı** bir widget olarak kabul edilir. Bu nedenle, uygulamanızın da bir widget olduğunu hatırlayarak, uygulama widgetımızı çalıştırmak için **runApp()** komutunu kullanıyoruz. main fonksiyonu içerisine **run yazınca** code.snippets'in size önerdiği runApp paketlerinden birini kullanın. Seçmiş olduğunuz paketin **material.dart** dosyasından çekildiğinden emin olun. Bu sizin bir android altyapısı üzerinde çalışmanız için gerekli tüm öğeleri getirecek olan kütüphanedir (Ps. **cupertino da benzer şekilde ios kütüphanesidir**, unutmayın ki hangi kütüphaneyi içe aktarırsak aktaralım sonuçta derleme sırasında her iki işletim sistemi için de programımızı basabiliriz).



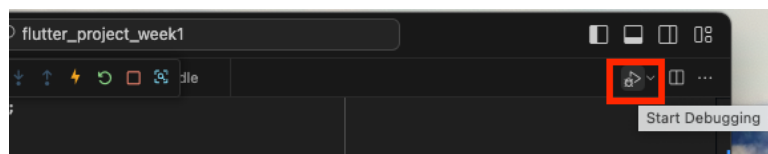
- 10. Stateless Widget (Durumsuz Widget) Ekleme:** `main.dart` dosyasındaki kodların altına gelin. **stw yazıp** otomatik tamamlama listesinden **StatelessWidget** seçeneğini seçin. Bu kısa yol, **durumsuz bir widget** sınıfı oluşturacaktır. Stateless widget'lar **değişmeyen (statik)** arayüzler için kullanılır.

```
7 class myWidget extends StatelessWidget {  
8   const myWidget({super.key});  
9  
10  @override  
11  Widget build(BuildContext context) {  
12    return Container();  
13  }  
14 }
```

- 11. runApp ile Sınıfı Çağırma:** Şimdi, tıpkı Dart uygulamasında yaptığımız gibi bu widget'ı uygulamanın başlangıç noktası olan **main()** fonksiyonunda çağırmamız gerekiyor. Bunun için **runApp** fonksiyonu içinde **sınıfın ismini** kullanacağız.

```
lib > main.dart > ...  
1 import 'package:flutter/material.dart';  
2  
   Run | Debug | Profile  
3 void main() {  
4   runApp(myWidget());  
5 }
```

- 12. Emülatörü Başlatma:** Artık oluşturduğunuz uygulamanın çıktısını emülatörde görebiliriz. Bunun için **VS Code** üzerinde **Start Debugging** butonunu kullanın. Bu buton, sağ üst köşede "play" (oynat) simgesi şeklinde yer alır veya **F5** tuşuna basarak aynı işlemi gerçekleştirebilirsiniz. Uygulamanızın emülatöre kurulması ve başlatılması biraz zaman alabilir. Bu süreç, emülatörün performansına ve bilgisayarınızın hızına bağlı olarak birkaç dakika sürebilir.



Android Temelli Uygulama Arayüzü Oluşturma

13. Flutter uygulamanızda sadece bir **Container** widget'ı kullanırsanız, genellikle **ekran siyah** olarak görüntülenir çünkü ekrana herhangi bir yapı ya da içerik yerleştirilmemiş olur. Bu siyah ekran çıktısından kurtulmanın en basit yolu, **Container** widget'ını **MaterialApp** widget'ı ile değiştirmektir. **MaterialApp** widget'ı, Flutter'da **Android temelli** uygulamalar için en temel yapı taşıdır (CupertinoApp'de ios için). Bu widget, uygulamanızın **genel ayarlarını** ve **görünümünü** kontrol eder. Container yerine yazacağınız **MaterialApp** widget'ı ile uygulamanızın ilk ekranını belirleyebiliriz.

```
0  @override
1  Widget build(BuildContext context) {
2    return MaterialApp();
3  }
```

14. Uygulamanızı emülatöre gönderdiğinizde **hala siyah** bir ekranla karşılaşacaksınız. Bunun nedeni uygulamanızda MaterialApp widget'ının **hiçbir özelliğinin** kullanılmamış olmasıdır. **MaterialApp** widget'ının en temel özelliklerinden biri **home property'sidir**. **home** property'si, uygulamanın başlangıç ekranını belirler. Ayrıca, bu widget'ın home özelliğini kullanarak içerisinde bir çok widgetı kullanabilirsiniz. Hemen bu widgetlardan biri olan ekrana bir şey yazdırabileceğimiz **Text widget'ı** ile işe başlayalım. Bu sonucu emülatörde görmek için de **start debugging** ile projemizi sanal cihazımıza kuralım.

```
10  @override
11  Widget build(BuildContext context) {
12    return const MaterialApp(
13      home: Text("Merhaba"),
14    ); // MaterialApp
15  }
16 }
```

15. Çirkin bir görüntünün oluştuğundan hepimiz eminiz. **home** property'si ile kullanacağımız bir **scaffold** widget'ı hemen bizim uygulamamıza çağ atlatacak ve android standartlarında bir layout'u önümüze sunacaktır. Şimdi **Text Widget'ını** silelim ve yerine **Scaffold Widget'ını** koyalım. **Scaffold** widget'ı, uygulamanız için tipik bir **Android arayüzü** sunar. Uygulamanızı emülatöre gönderdiğinizde siyah ekranın bir android uygulamasına benzemeye başladığını göreceksiniz.

```
7   class myWidget extends StatelessWidget {  
8     const myWidget({super.key});  
9  
10    @override  
11    Widget build(BuildContext context) {  
12      return MaterialApp(  
13        home: Scaffold(),  
14      ); // MaterialApp  
15    }  
16  }  
17
```

16. Daha da ileriye gitmek için **Scaffold** Widget'ının ilk özelliği olan **appBar'ı** kullanalım. Gördüğünüz gibi propertylerin yazım şekli **camelCase**. Bundan sonra gelecek olan **AppBar widgetı** ise **PascalCase** olarak yazılmalıdır. Bu AppBar widget'ı ile uygulamamızın üstüne bir uygulama çubuğu eklenecektir.

```
10    @override  
11    Widget build(BuildContext context) {  
12      return MaterialApp(  
13        home: Scaffold(  
14          appBar: AppBar(),  
15        ), // Scaffold  
16      ); // MaterialApp  
17    }  
18  }
```

17. Uygulamanızı test ettiğinizde bir şey değişmemiş gibi görünecektir. Bunun nedeni **AppBar**'ın **özelliklerinin** kullanılmamış olması. AppBar widgetının propertylerinden olan **title**'i kullanarak uygulamamıza bir başlık eklemeye ne dersiniz? Tüm yazma işlemini tamamladıktan sonra **hot reload** (ctrl+s) yaparak uygulama sonucunu görelim. Flutter'ın geliştirme sürecini hızlı ve verimli hale getiren en önemli özelliklerinden biri **hot reload** (anında yeniden yükleme) özelliğidir. Hot reload, kodda yapılan değişikliklerin anında uygulamaya yansıtılmasını sağlar ve bu sayede geliştiriciler hızlı bir şekilde geri bildirim alabilirler. Uygulamayı baştan başlatmaya gerek kalmadan kullanıcı arayüzündeki değişiklikler anında görülebilir.

```
appBar: AppBar(title: const Text("İlk Uygulamam"),),  
)); // Scaffold // MaterialApp
```

18. Bu başlığı ortalamak için **centerTitle** property'sini kullanalım. Boolean bir değişken olan bu özelliğin **true** olarak girilmesi, başlığın ortalanması anlamına geliyor.

```
title: const Text("İlk Uygulamam"),  
centerTitle: true,
```

19. Başka bir özellik olan **backgroundColor** özelliği ile uygulama çubuğunu renklendirebiliriz. Bunu uygulamak için de **backgroundColor** property'si içerisine **Colors** kütüphanesinden **deepOrange** rengini çağıralım.

```
14 appBar: AppBar(  
15   title: const Text("İlk Uygulamam"),  
16   centerTitle: true,  
17   backgroundColor: Colors.deepOrange,  
18 ), // AppBar
```

20. AppBar widget'ının önemli bir özelliği de **leading** property'sidir. Bu özellik, başlık çubuğunun sol tarafında yer alacak widget'ı belirtir. Örneğin, başlık çubuğunda "Menü" yazan bir metin göstermek için:

```
backgroundColor: Colors.deepOrange,  
leading: const Text("MENÜ"),  
, // AppBar
```

21. Scaffoldun ikinci özelliği olan **body**'e geçelim. **body** property'si, uygulamanızda görünen ana içerik alanını temsil eder. Bu alan, uygulamanızın görsel içeriğini oluşturmak için kullanılır. Örneğin, metin, resim, buton gibi widget'lar burada yer alabilir. **body alanı**, **Scaffold** içerisindeki en büyük ve belirgin alanı oluşturur. Bu özelliğin içerisine "Uygulamama Hoş Geldiniz" yazan bir **Text** widget'ı ekleyelim.

```
), // AppBar  
body: const Text("Uygulamama Hoş Geldiniz"),  
) , // Scaffold
```

22. Flutter'da bir yazıyı veya diğer widget'ları ortalamak için **Center** widget'ını kullanabiliriz. **Center** widget'ı **child (çocuk) property'si** ile kendisine verilen alt widget'ı, hem yatay hem de dikey olarak ortalamak için kullanılır. Flutter'daki **child** ve **children** özellikleri, widget'ların altındaki diğer widget'ları yerleştirmek için kullanılır. Ancak aralarındaki temel fark, **child** sadece **tek bir widget** kabul ederken, **children** bir **widget listesi** kabul eder. Hangi özelliği kullanmanız gerektiği, hangi widget ile çalıştığınıza ve altına kaç tane widget eklemek istediğinize bağlıdır.

```
body: const Center(  
  child: Text("Uygulamama Hoş Geldiniz"),  
) , // Center  
) , // Scaffold
```

23. Tebrikler, sınıf yapılarından birini kullanarak bir flutter projesi oluşturdunuz.