

Android Mobil Uygulama Geliştirme Modülü

İçindekiler

Mobil İşletim Sistemleri.....	2
Android İşletim Sistemi	3
Android Studio ve Mobil Uygulama Geliştirme Ortamı.....	4
Android Emülatör	5
Java Programlama Dili	8
Yorum Satırı	9
Veri Tipleri ve Değişkenler	9
İlkel Veri Tipleri.....	10
Sabit Değişkenler	11
Operatörler.....	11
Tip Dönüşümleri	13
Karar Yapıları (if/else)	14
Karar Yapıları (switch).....	16
Döngüler	17
Diziler	18
Metotlar	20
Sınıf Kavramı ve Erişim Belirleyiciler.....	21
Yeni Bir Proje Oluşturma.....	23
Aktivite Oluşturma.....	23
Activity Yaşam Döngüsü	25
Android Studio Proje Yapısı	26
Layoutlar ile Çalışmak	27
LINEARLAYOUT	29
LAYOUT WEIGHT	31
Viewler ile Çalışmak	33
DENSITY-INDEPENDENT PIXELS (dp).....	33
SCALABLE PIXELS (sp)	34
VIEW	34
VIEWGROUP	34
Firestore İşlemleri.....	35
Firestore Authentication	39
Firestore Storage	42
İzin İşlemleri	42
Firestore Firestore	43
Kaynaklar:.....	45

Mobil İşletim Sistemleri

Mobil cihazlar hayatımızda her geçen gün daha fazla yer alıyor. Kişisel asistan olarak adlandırılan el bilgisayarları ve dizüstü bilgisayarlar ile başlayan mobil cihazların kullanımı cep telefonlarının akıllanmasıyla Dünya genelinde milyonlarca bireysel kullanıcıya ulaşmış durumdadır. Günümüzde 3,8 milyardan fazla insan sosyal medya kullanıcısı durumdayken, Dünya nüfusunun %67'si cep telefonuna sahip. Akıllı cep telefonlarının başını çektiği mobil cihazlar içerisinde tabletler, televizyonlar, otomobil eğlence ve bilgi ekranları, akıllı bileklik ve saatler de gösterilebilir.

Bilgisayar, tablet, cep telefonu gibi elektronik donanımların kullanıcıyla etkileşime girebilmesi, kullanıcıdan veya diğer sistemlerden aldığı talimatları donanım üzerinde işleyebilmesi ve donanımlardan elde edilen sonuçların görsel, işitsel veya sinyal olarak çıktı verilmesi için işletim sistemi denilen bir yazılıma ihtiyaç duyarlar. Bu yazılım kullanıcı ve donanım arasında bir köprü vazifesi gördüğü için üzerinde çalıştığı donanıma yönelik geliştirilir. İşletim sistemleri donanımın çalışmasının yanında diğer yazılımların işlemci, ram, sabit disk gibi donanım kaynaklara olan erişimlerini ve paylaşımlarını da düzenler.

Mobil cihazlar için geliştirilen işletim sistemlerine mobil işletim sistemleri denir. Mobil işletim sistemleri üzerinde çalışacakları hedef ortam başta olmak üzere farklı özelliklere yönelik geliştirilmişlerdir. Mobil işletim sistemlerine örnek olarak aşağıdaki işletim sistemleri gösterilebilir:

- Android
- Apple IOS
- Windows Phone
- BADA
- Palm OS
- Symbian
- Meego
- Tizen
- Harmony OS
- Blackberry OS

Mobil işletim sistemleri içerisinde Pazar payı olarak Google tarafından geliştirilen Android ve Apple tarafından geliştirilen IOS ön plana çıkmaktadır. Her iki işletim sistemi akıllı telefon mobil işletim sistemi Pazar payının %99'una sahiptir. Günümüzdeki akıllı cep telefonları üzerinde çalışan mobil işletim sistemlerinin Pazar payına bakıldığında Ocak 2021 itibariyle Android işletim sistemi %71,93'lük Pazar payıyla liderliğini korumaktadır. Android işletim sisteminin açık kaynak olması ve Google desteğinin bulunması Pazar payının bu kadar yüksek olmasındaki en önemli iki etken olarak gösterilebilir.

Android işletim sisteminin lider konumda olması uygulama geliştiriciler açısından da güçlü ve fırsatlarla dolu bir Pazar alanı oluşturmaktadır. Android telefonlar üzerinde çalışacak yazılımların bulunduğu Google Play mağazası her geçen gün daha fazla uygulamaya ev sahipliği yapmaktadır. Google Play mağazasında 2020 Yılıının dördüncü çeyreği itibariyle 3,14 milyon uygulama bulunmaktadır.

Android İşletim Sistemi

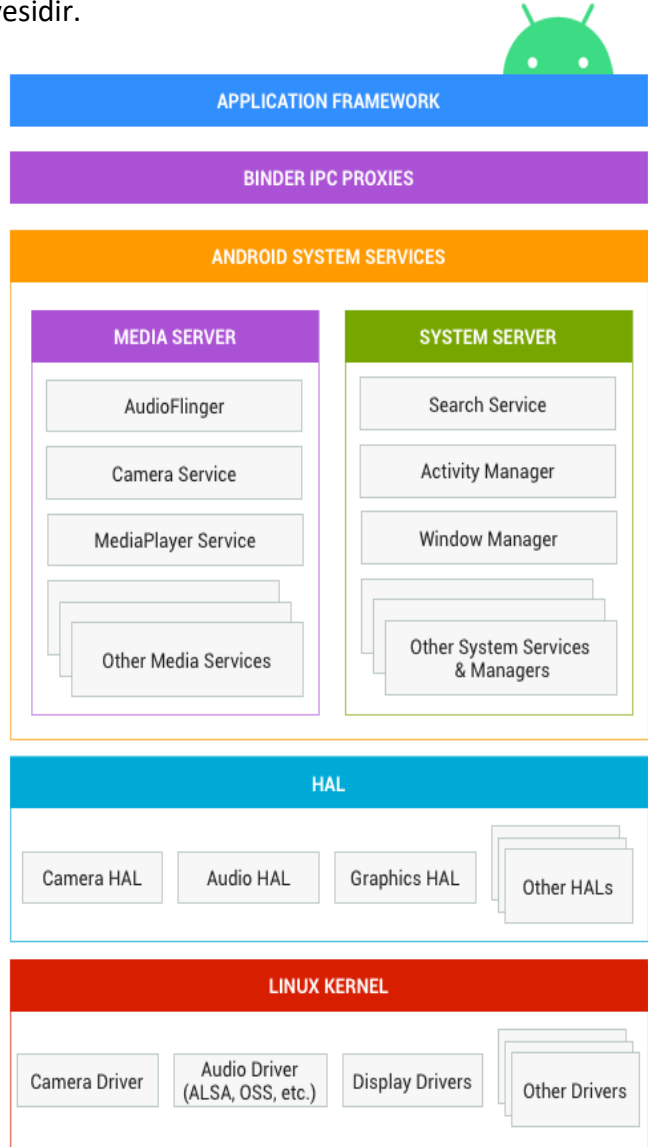
İlk olarak Android Inc. isimli bir şirkette ortaya çıkan Android, şirketin 2005 yılında Google tarafından satın alınmasıyla birlikte Google bünyesine geçmiştir. Android, Google ve Open Handset Alliance tarafından yürütülen açık kaynak mobil işletim sistemidir. Open Handset Alliance mobil Telekom operatör şirketlerinden, cep telefonu üreticilerinden, yarı iletken üreticilerinden oluşan bir topluluktur. Google, Huawei, Toshiba, Samsung, Oppo, Acer, Dell, Lenovo gibi üreticiler OHA topluluğunun üyesidir.

Android sistem mimarisi 5 temel bölümden meydana gelir. Mimarının en alt katmanı işletim sisteminin çekirdeğidir. Android işletim sistemi Linux çekirdeği üzerine inşa edilmiştir.

Çekirdeğin üzerinde donanım soyutlama katmanı (hardware abstraction layer) bulunur. Bu katman, donanımı üst katmanlardan soyutlayarak daha anlaşılır bir ara yüz sunar. Yazılım ile donanım arasında oluşturulan bu köprü sayesinde yazılımın farklı donanımlar üzerinde de çalışmasına olanak tanınır. Böylece üst düzey sistem değişikliklerini yapmaya gerek kalmaz.

Sistem hizmetleri medya ve sistem şeklinde 2 grup altında toplanmış modüler hizmet paketlerinden meydana gelir.

Bağlayıcı IPC (Inter Process Communication) , uygulamalar arası haberleşmeyi veya birden fazla iş yürüten bir uygulama içindeki farklı işlerin birbiriyle haberleşmesini sağlar. En üstteki uygulama çerçevesi ise uygulama geliştiricilerin muhatap olduğu katmandır.



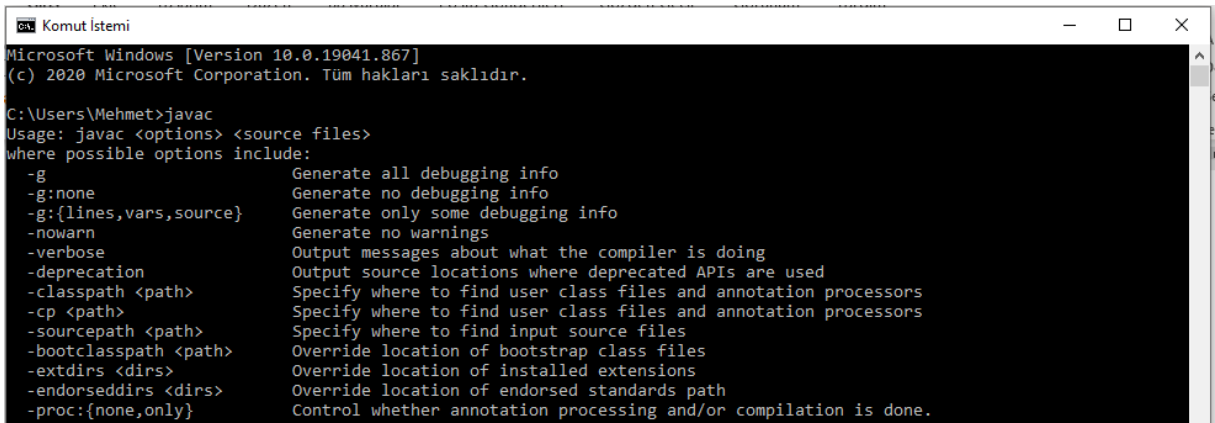
Android işletim sistemi gelişen teknolojiye paralel olarak güncellenmekte ve yeni teknolojiler içeren özelliklere sahip sürümler yayınlanmaktadır. Android işletim sistemi sürümleri farklı donanım özelliklerine ihtiyaç duyduğundan özellikle yeni sürümler güçlü donanım kaynakları içeren telefonlarda çalışır. Düşük donanımsal özelliklere sahip mobil telefonlar içinse Android işletim sisteminin önceki sürümleri kullanılmaktadır. Google Play mağazasında da özellikle Android 4.4 sürümü için kullanıcılara yönelik sosyal medya, haberleşme, bankacılık gibi pek çok uygulama bulunmaktadır. İlk sürümü 2008 yılında yayınlanan Android işletim sisteminin son sürümü ise Android 11'dir.

Android Studio ve Mobil Uygulama Geliştirme Ortamı

Akıllı telefon, tablet, akıllı saat veya smart televizyon gibi cihazlar üzerinde çalışan ve belirli bir görevi yerine getirmek için geliştirilmiş yazılımlara mobil uygulama ismi verilir. İster mobil cihazlarda isterse de bilgisayarlarda çalışacak herhangi bir yazılımı geliştirmek için IDE (Integrated Development Environment) adı verilen tümleşik geliştirme ortamları kullanılır. Kendisi de bir yazılım olan IDE, yazılımcıların daha hızlı ve verimli yazılım geliştirmelerine imkan tanır. Bir HTML web sayfası not defteri gibi bir editörde oluşturulabileceği gibi Adobe Dreamweaver üzerinde de oluşturulabilir. Fakat Dreamweaver üzerinde geliştirmek hataların daha kolay tespit edilmesi, ön izlemenin anlık görülebilmesi gibi süreci daha kolaylaştıran özelliklerin kullanılmasını sağlar.

Android Studio yazılımı, Android işletim sistemine sahip cihazlar üzerinde çalışacak mobil uygulamalar geliştirilmesine olanak tanıyan resmi IDE'dir. Google tarafından Mayıs 2013 tarihinde v0.1.x sürümüyle piyasa sürülen Android Studio'nun en son sürümü 4.1 sürüm numarası ile Ağustos 2020 tarihinde yayınlanmıştır. Android Studio üzerinde mobil uygulamalar Java veya Kotlin programlama dillerinden herhangi birisi ile yazılabilir. Android Studio native bir platformdur. Native platformlarda geliştirilen uygulamalar belirli bir platforma özel oluşur. Bu sebeple Android Studio ile yazılan uygulamalar sadece Android cihazlarda çalışacaktır. Xamarin gibi cross platformlar kullanılarak yazılan uygulamalar ise birden fazla platform üzerinde çalışabilir. Xamarin ortamında C# ile geliştirilen bir mobil uygulama hem Android telefonlar hem de Apple telefonlar için çalışacak şekilde çıktı alınabilir.

Android Studio kurulumuna başlamadan önce bilgisayarda Java Development Kit (JDK) son sürümü yüklenmelidir. JDK yükleme paketi <https://www.oracle.com/tr/java/technologies/javase-downloads.html> linkinden indirilebilir. JDK kurulduktan sonra ortam değişkenleri ayarlanır. Java kurulumunun başarılı bir şekilde gerçekleştiğini kontrol etmek için komut istemi ekranına (cmd) javac komutu yazılır. Komut çıktısı olarak ekranda parametreler listeleniyorsa kurulum başarıyla tamamlanmış demektir.



```
Komut İstemi
Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. Tüm hakları saklıdır.

C:\Users\Mehmet>javac
Usage: javac <options> <source files>
where possible options include:
  -g               Generate all debugging info
  -g:none          Generate no debugging info
  -g:{lines,vars,source} Generate only some debugging info
  -nowarn          Generate no warnings
  -verbose         Output messages about what the compiler is doing
  -deprecation     Output source locations where deprecated APIs are used
  -classpath <path> Specify where to find user class files and annotation processors
  -cp <path>       Specify where to find user class files and annotation processors
  -sourcepath <path> Specify where to find input source files
  -bootclasspath <path> Override location of bootstrap class files
  -extdirs <dirs>   Override location of installed extensions
  -endorseddirs <dirs> Override location of endorsed standards path
  -proc:{none,only} Control whether annotation processing and/or compilation is done.
```

Android Studio'nun son sürümü <https://developer.android.com/studio#downloads> adresinden indirilebilir. Kurulum yapılacak bilgisayardaki en düşük sistem özellikleri:

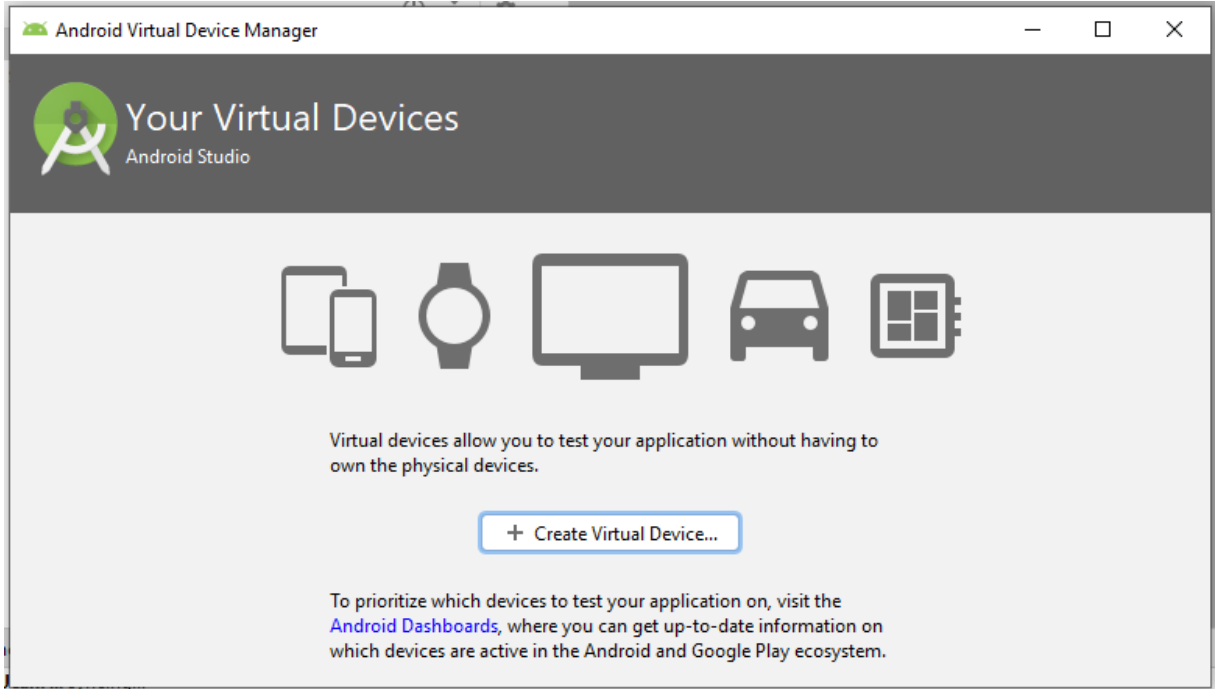
- Microsoft® Windows® 7/8/10 (64-bit)
- 4 GB RAM en düşük, 8 GB RAM önerilen

- 2 GB en düşük sabit diskte gereken boş alan, 4 GB önerilen (500 MB IDE için + 1.5 GB Android SDK ve emulator imajları için)
- 1280 x 800 en düşük ekran çözünürlüğü

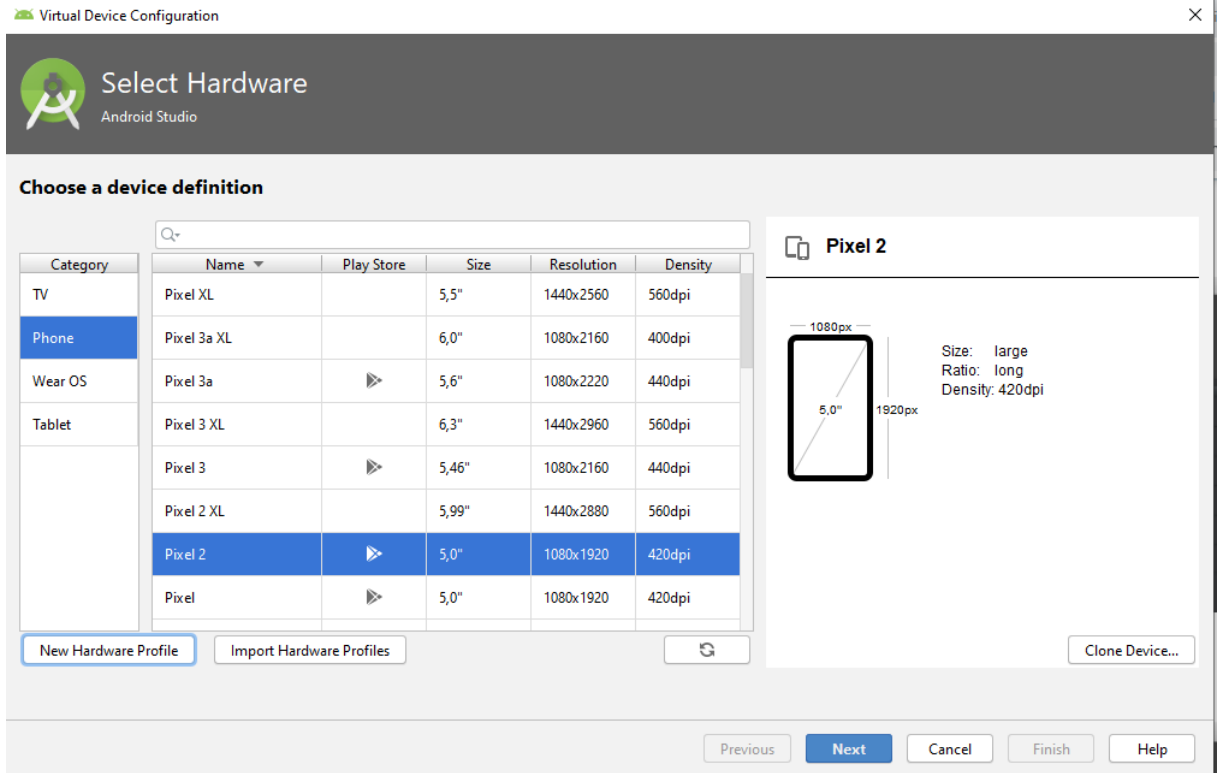
Android Emülatör

Mobil uygulamaların geliştirme aşamasında farklı donanımsal özelliklere sahip cihazlar üzerindeki çalışma biçiminin test edilmesi gerekebilir. Uygulamanın hedeflendiği her farklı platform için gerçek bir cihazın geliştirme sürecinde test için bulunması mümkün olmayabilir. Bu durumda uygulamanın gerçek bir cihaz üzerinde nasıl çalışacağını test edilmesi için simülasyon yapıları devreye girer. Android Emulator Android tabanlı cihazları gerçekte sahip oldukları özellikleriyle simule ederek uygulamanın test işlemlerinin eksiksiz yapılmasını sağlar.

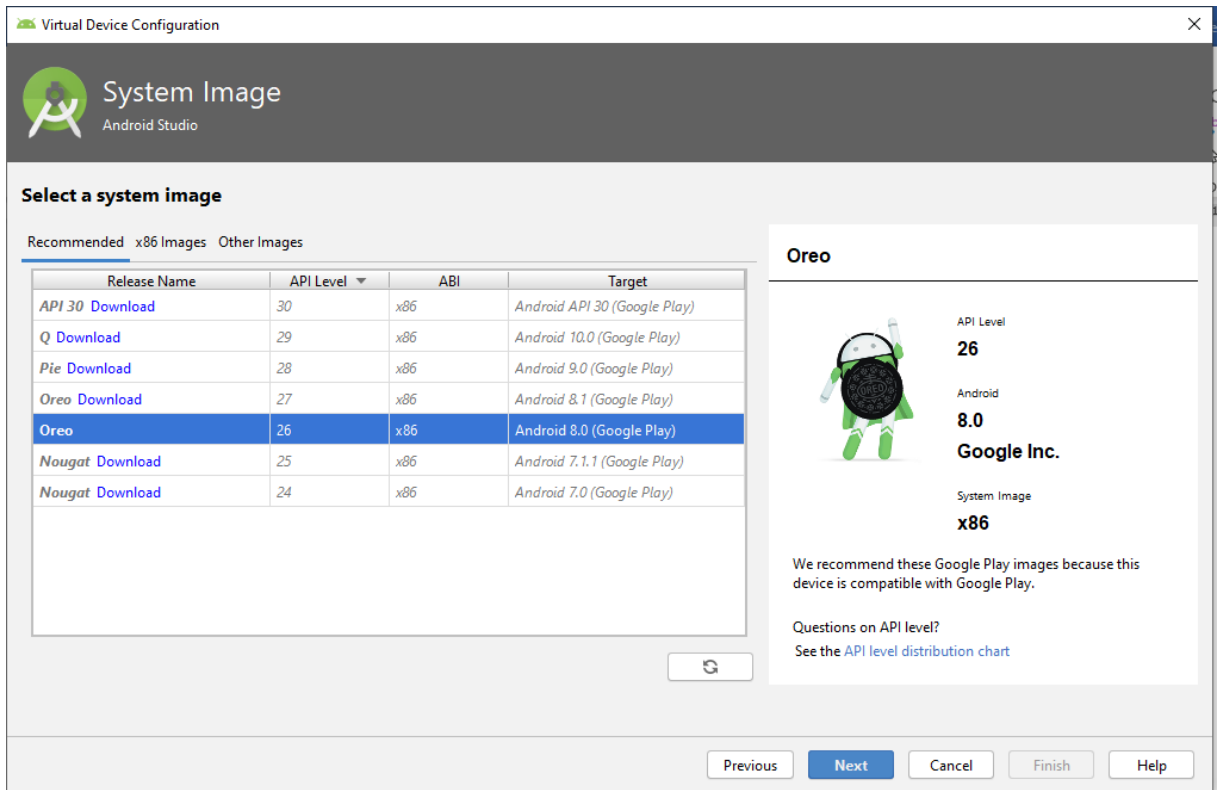
Android Emulatör, HAXM 6.2.1 veya üstünü destekleyen 64 bitlik işlemciye sahip sistemlerde çalışır. Android Studio’da sanal bir cihaz oluşturmak için Tools menüsü altındaki AVD Manager seçeneğine tıklanır. Açılan pencereden Create Virtual Device butonuna tıklanır.



Bu ekranda oluşturulacak sanal cihazın türü ve donanımsal özellikleri belirlenir. Eğer bir mobil telefon oluşturulacaksa Phone seçeneğine tıklanır. Listedeki önceden tanımlanmış özelliklerde standart mobil telefonlar seçilebileceği gibi New Hardware Profile butonuna tıklanarak kişisel tercihlere göre bir mobil telefonda tasarlanabilir.

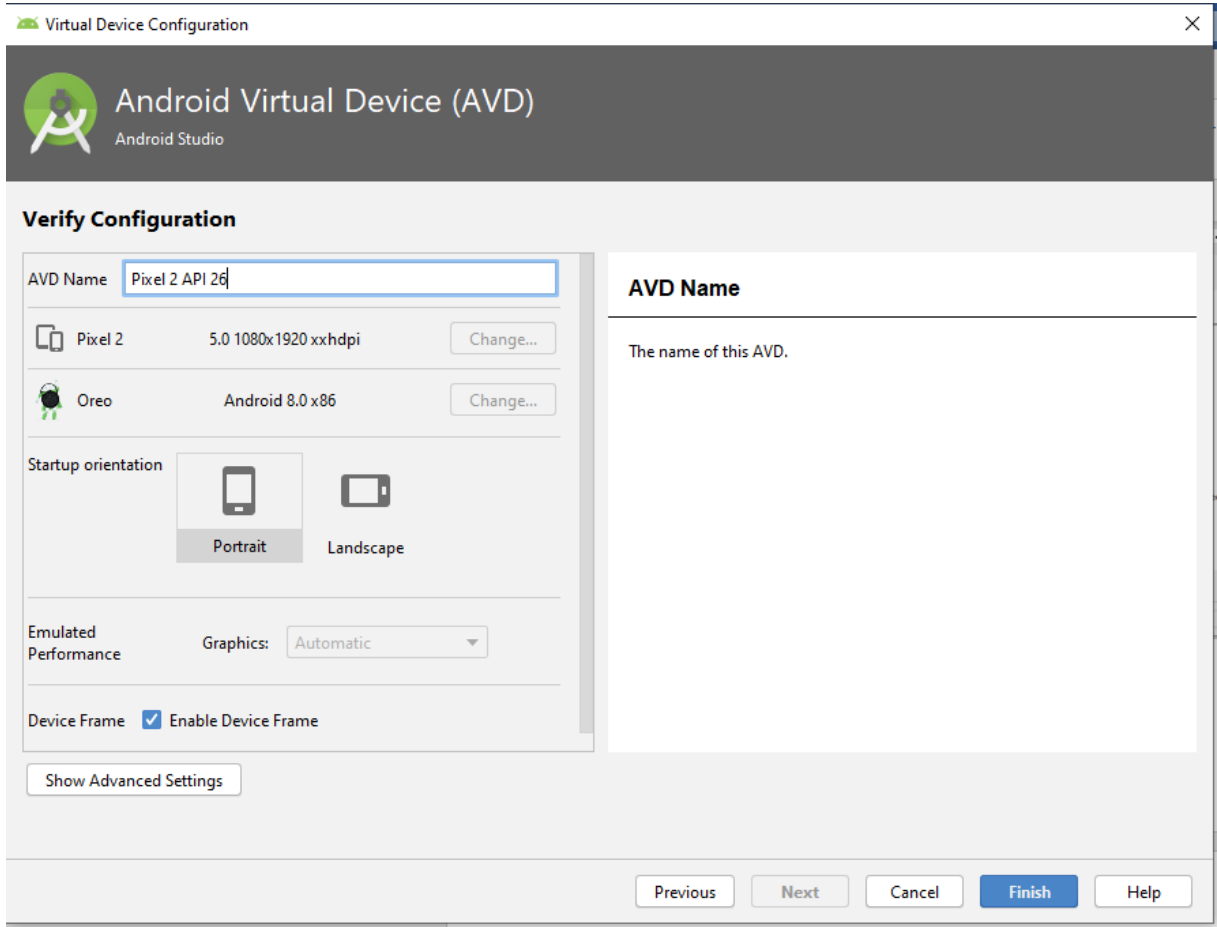


Cihaz türü seçildikten sonra Next butonuna basılır. Açılan ekranda seçilen mobil telefonda çalışacak olan Android sürümü belirlenir. Bilgisayarda istenilen sürümün imajı yoksa Download yazısına tıklanarak indirilmesi gerekmektedir.

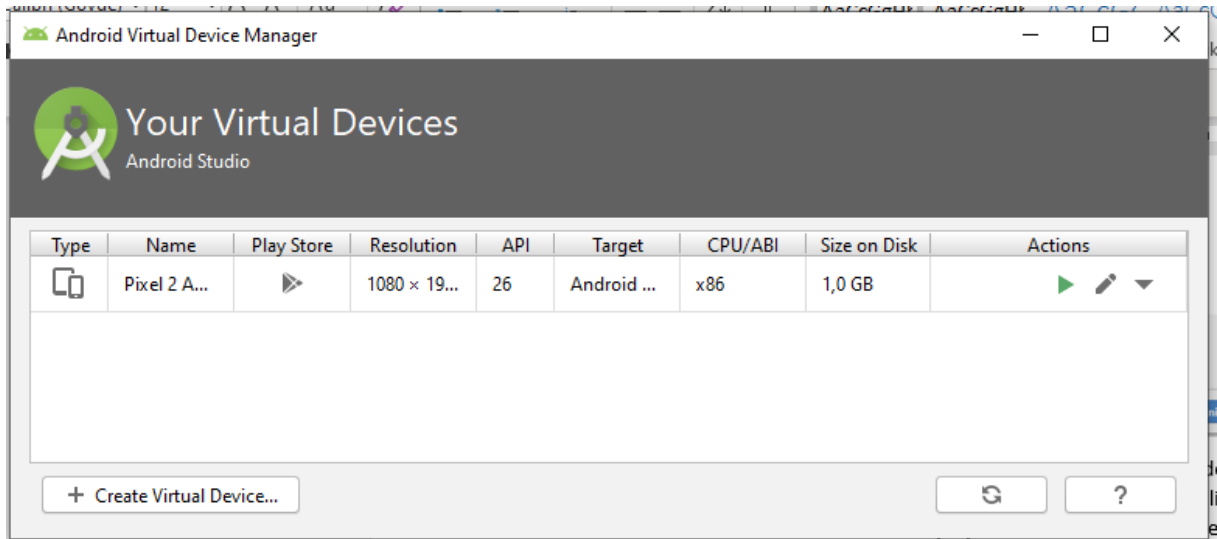


Android sürümü belirlendikten sonra Next butonu ile son aşamaya geçilir. AVD Name kısmında cihaza isim verilebilir. Cihazın başlangıçta ekran konumunun yatay veya dikey olarak açılması

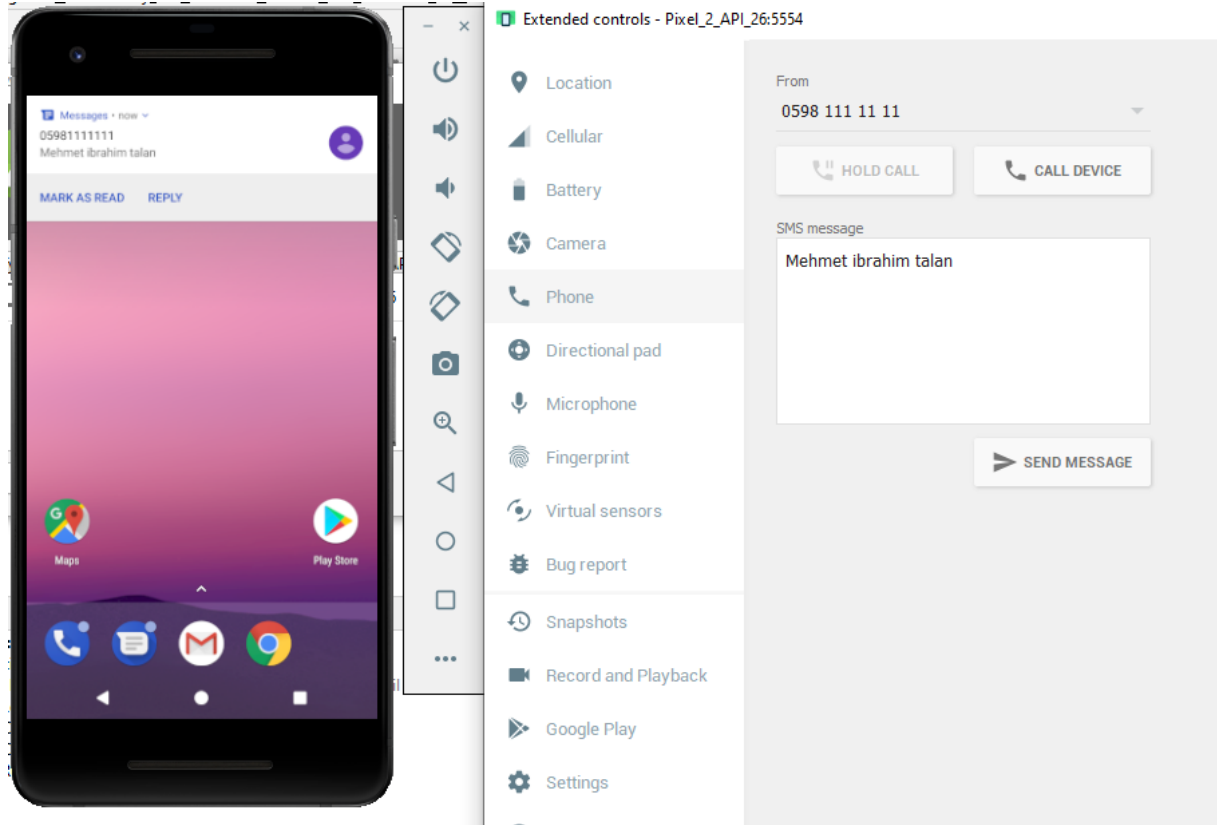
işlemi Startup Orientation bölümünden seçilir. Eğer cihazla ilgili ağ ayarları gibi ileri ayarların yapılması isteniyorsa Show Advanced Settings butonuna basılarak bu ayarlara ulaşılabilir.



Finish butonuna basılarak sanal cihaz oluşturma işlemi tamamlanır. Sistemde tanımlı olan sanal cihazlara Tools menüsü altındaki AVD Manager seçeneğinden ulaşılabilir. Bu ekranda mevcut sanal cihazlardan istenilen cihaz çalıştırılabilir (yeşil ok), kaldırılabilir, durdurulabilir veya özellikleri değiştirilebilir.



Emul tor a ıldığında ger ek bir mobil telefonda yapılan i lemlerin tamamına yakını sanal cihaz  zerinde yapılabilir. B ylece uygulamaların test i lemleri her t rl  senaryo i in ger ekle tirilebilir. Emulator n saė alt kısmında bulunan    nokta simgesine basılarak a ılan ekranda cihazla ilgili i lemler ger ekle tirilir. SMS g ndermek, arama g ndermek, konum belirlemek gibi t m i lemler bu ekrandaki men ler altından yapılır.



Java Programlama Dili

Android mobil uygulamalar Android Studio ortamında Java programlama dili kullanarak yazılırlar. Java, 1995 yılında Sun Microsystems firması m hendislerinden James Gosling tarafından geli tirilmi  bir programlama dili ve bilgi i lem platformudur. Nesne y nelimli yapısal bir programlama dilidir. Bir kere yaz her yerde  alı tır felsefesini savunur. En  nemli  zelliėi platform baėımsız olmasıdır. Java programlama dili programlama dilleri i erisinde  nemli bir yere sahiptir. Pop lerliėini korumaya devam eden Java, 2019 yılında yapılan ara tırmaya g re %96,3'le en  ok kullanılan 2. dil konumundadır.

Java tabanlı uygulamalar geli tirmek veya Java tabanlı uygulamaları  alı tırmak i in bilgisayarda JDK (Java Geli tirme Seti) ve JRE (Java Runtime Environment) uygulamalarının y klenmi  olması gerekir. JRE, Java programlarını uygulamakta olan bir Java Sanal Makine uygulamasıdır. Java Runtime Environment, Java programlarını  alı tırmak i in gereken bir eklentidir. JVM, Core kitaplıkları ve Java yazılımında yazılan uygulamaları ve k   k uygulamaları  alı tırmak i in diėer ek bile enleri i erir. JDK ise Java tabanlı uygulamaları geli tirmek i in kullanabileceėiniz bir yazılım paketidir. Java uygulamalarını geli tirmek i in Java Geli tirme Seti gerekir. JRE, API sınıfları seti, Java derleyici, Webstart ve Java

uygulamalarını ve küçük uygulamalarını yazmak için gereken ek dosyaları içerir. Java ile yazılım geliştirmek için kullanılan IDE'lerin başında Netbeans gelmektedir.

Yorum Satırı

Program kodları arasına herhangi bir durum için açıklama yapmak gerekebilir. Örneğin bir kodla ilgili hatırlatıcı veya kodu sonradan okuyacaklar için çeşitli bilgiler verilmesi gerekebilir. Program dosyası içerisinde kod haricindeki ifadelerin olduğu gibi yazılması programın kırılmasına yol açacaktır. Java derleyicisi yazdığımız kod harici ifadelere geldiğinde o satırı çalıştırmaya kalkacak fakat kod olmadığı için hata verecektir. Bu sebeple kod harici satırların derleyici tarafından dikkate alınmaması için bu satırların başına // karakterleri konulur. Eğer açıklama satırları birden fazla ise /*....*/ karakterleri arasında yazılabilirler.

```
public static void main(String[] args) {  
    // Komut satırları noktalı virgül ile biter.  
    System.out.println("Merhaba Dünya");  
    /*Ekrana birşeyler yazdırmak için System.out.println  
    komutu kullandık.  
    */  
}
```

Veri Tipleri ve Değişkenler

Uygulama yazılımları amaçlarına bağlı olarak bir takım işlemleri yerine getirebilmek için verilere ihtiyaç duyabilir. Bu veriler program içerisinde kullanılan sabit değerler olabileceği gibi kullanıcıdan veya başka bir sistemden gelen veriler de olabilir. Kullanıcıdan veya başka bir sistemden alınan değerleri tahmin etmek, her ihtimali tanımlamak doğru bir yöntem olmayacağı gibi imkansızdır. Bunun için program içerisinde verileri temsil eden ifadeler kullanılır. Bu ifadelere değişken adı verilir. Değişkenlerin tuttukları verilere ise değişkenin değeri denilir. Bir öğrencinin iki yazılı notunun ortalamasını hesaplayıp ekranda gösteren küçük bir programı düşünelim. Bu programda öğrenciden alınacak yazılı notlarını önceden tahmin etmek mümkün değildir. Kaldı ki programı kullanan her öğrenci her kullanımında farklı notlar girerek programı çalıştırabilir. Bu sebeple program içerisinde sayısal bir değer belirtmek yerine öğrencinin 1.yazılısını temsil eden değişken olarak not1 ve 2.yazılısını temsil eden değişken olarak not2 değişkeni kullanılabilir. Program çalıştığında öğrenciden gelen 1.yazılı ve 2.yazılı bilgileri not1 ve not2 değişkenlerine atanarak işlem yapılabilir.

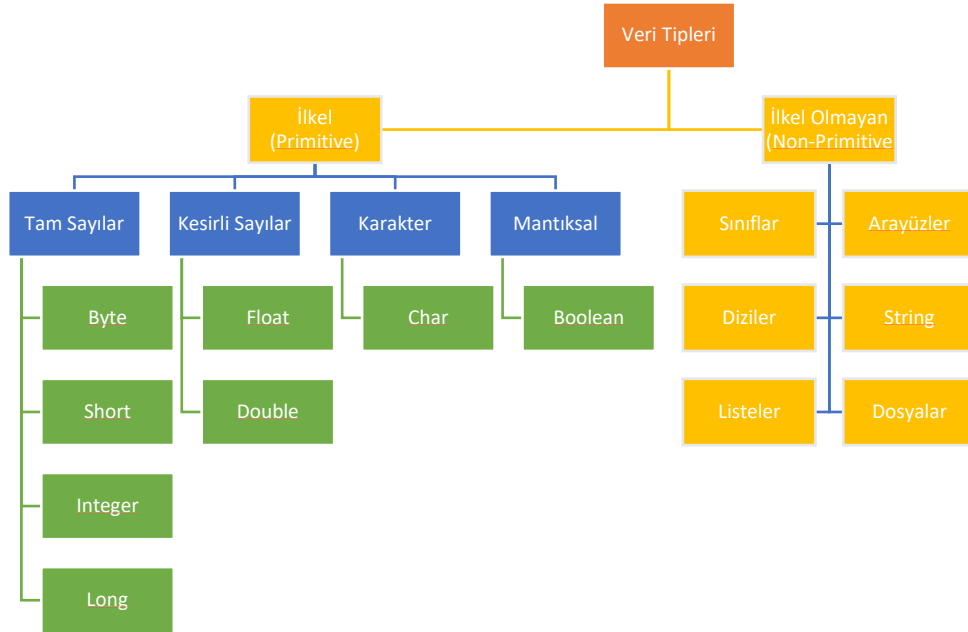
Değişkenlere değer atamadan önce hangi tipte değer tutacağı belirtilmek zorundadır. Değişkenlere değer atama eşittir (=) operatörü ile gerçekleştirilir. Değer atama işlemi sağdan sola doğrudur. Sabit olarak tanımlanmamış bir değişken her zaman kendisine en son atanan değişkeni saklar. Değişken isimleri küçük harfle başlar. Eğer birden fazla kelimedenden oluşuyor ise her yeni kelime boşluk bırakılmadan büyük harfle devam eder.

```
String isim="Mehmet";
```



Değişkenin Veri Tipi Değişkenin Adı Değişkenin Değeri

Değişkenler tuttukları veri tiplerine göre gruplara ayrılırlar. Kimi değişkenler sadece tam sayıları tutarken, ondalık basamak içeren sayıları farklı değişkenler tutar. Değişkenler program çalıştırıldığında RAM bellekte oluşturulurlar. Dolayısıyla geçici veri yapılarıdır. Bellekte kaplayacakları alan tanımlandıkları veri tipine göre değişir. Bu sebeple gereksiz büyüklükte bir değişken tanımlamak belleğin verimsiz kullanımına yol açar.



İlkel Veri Tipleri

Bu kavram kullanılmak için herhangi bir kütüphaneden çağrılmasına gerek olmayan veri tiplerini tanımlar. Diğer veri tiplerine temel oluşturan 4 temel ilkel veri tipi vardır.

- Tam Sayılar
- Kesirli Sayılar
- Mantıksal
- Karakterler

Tamsayılar ve Kesirli Sayı Değişken Tipleri

- ✓ Bellekte kapladıkları alana göre alt çeşitlere ayrılırlar.
- ✓ Bellekte kapladıkları alan büyüdükçe daha büyük sayıları değer olarak saklayabilirler.
- ✓ Float veri tipinde değer sonuna f karakteri, long veri tipinde ise değer sonuna l karakteri konulmalıdır.

- ✓ Float tip 6-7 ondalık basamak için yeterli iken Double veri tipi ise 15 ondalık basamağa kadar değer tutabilir.

Karakter ve Mantıksal Veri Tipleri

Char, tek bir Unicode karakteri veya Hex kodunu tutan veri tipidir. Bu tipte bir değişkene atanacak olan karakter tek tırnak işareti içerisinde yazılmalıdır. True veya False mantıksal değeri Boolean veri tipi değişkenlerde tutulur. Boolean tipindeki bir değişken True veya False değeri dışında başka bir değer tutamaz

Veri Tipi	Büyüklüğü	Açıklama
byte	1 byte	128 - 127
short	2 bytes	-32,768 to 32,767
int	4 bytes	-2,147,483,648 - 2,147,483,647
long	8 bytes	-9,223,372,036,854,775,808 - 9,223,372,036,854,775,807
float	4 bytes	6,7 Ondalık basamaklı sayılar
double	8 bytes	15 Ondalık basamaklı sayılar
boolean	1 bit	True False
char	2 bytes	Tek karakter veya ASCII kodu

Sabit Değişkenler

Değer atanmış bir değişken, tekrar değer atandığında son atanan değeri saklayacaktır. Eğer bir değişkenin değerinin sabit kalması isteniyorsa değişken tanımlanırken başına final kelimesi eklenir. Final ile sabit hale getirilen bir değişkene yeni bir değer atanamaz.

```
3 public class Kosullar {
4     public static void main(String[] args) {
5         final int sayi1=5;
6         sayi1=10;
7     }
8 }
```

Cannot assign a value to final variable 'sayi1'

Yandaki ekran görüntüsünde sayi1 değişkeni sabit olarak tanımlanmış ve değeri 5 olarak belirlenmiştir. 6 Nolu satırda sabit değişken olan sayi1'e yeniden değer atanmaya kalkıldığında hata mesajı verilmiştir.

Operatörler

Program içerisinde değişkenler veya sabit değerler ile aritmetiksel, mantıksal işlemler ile atama görevlerini yerine getiren karakterlere operatör denilir.

Matematiksel Operatörler aritmetik işlemleri yerine getirir.

Operatör	İsim	Açıklama	Örnek
+	Toplama	İki değeri toplar	$x + y$
-	Çıkarma	Bir değeri diğerinden çıkarır.	$x - y$
*	Çarpma	İki değeri çarpar.	$x * y$
/	Bölme	Bir değeri diğerine böler.	x / y
%	Mod	Bölümden kalanı verir.	$x \% y$
++	Artırma	Değeri bir artırır.	$++x$
--	Azaltma	Değeri bir azaltır.	$--x$

Karşılaştırma operatörleri değişkenler veya sabit değerler arasında karşılaştırma işlemlerini yerine getirir.

Operatör	İsim	Örnek
==	Eşit	$x == y$
!=	Eşit Değil	$x != y$
>	Büyüktür	$x > y$
<	Küçüktür	$x < y$
>=	Büyük veya eşit	$x >= y$
<=	Küçük veya eşit	$x <= y$

Mantıksal işlemleri yapan operatörler ile and, or gibi işlemler yapılabilir.

Operatör	İsim	Açıklama	Örnek
&&	Ve	Her iki şart doğru ise true değerini döndürür.	$x < 5 \ \&\& \ x < 10$
	Veya	Şartlardan en az birisi doğru ise true değerini döndürür.	$x < 5 \ \ x < 4$
!	Değil	Sonucu ters çevirir. True ise false, false ise true döndürür.	$!(x < 5 \ \&\& \ x < 10)$

Atama operatörleri işlemler yapılır ve aynı satırda sonuç bir değişkene aktarılır.

Operatör	Örnek	Denkliği
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
=	x = 3	x = x 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

```

1  class Main {
2      public static void main(String[] args)
3
4      int x=5;
5
6      x=x+5;
7      System.out.println(x);
8
9      x+=5;
10     System.out.println(x);
11
12     x++;
13     System.out.println(x);
14
15     ++x;
16     System.out.println(x);
17
18     System.out.println(x++);
19
20     System.out.println(++x);
21
22 }
23

```

Ekran Çıktısı

10

15

16

17

17

19

Atama operatörleri ile ilgili olarak yanda verilen örnek incelendiğinde operatörün değişkenin sağında veya solunda olması durumuna göre ekrana yazdırılan değişken değerinin farklı olduğu görülmektedir.

18 ve 20. Satırlarda yapılan işlem temelde aynı olmasına rağmen ekran çıktıları farklıdır. Her iki satırda da x değişkeninin değeri 1 artırılmıştır. Farklı sonuç çıkmasının sebebi artırma operatörünün konumudur. 18.Satırda x değişkeninin değeri ekrana yazdırılıp daha sonra 1 artırılmıştır. 20 Satırda ise önce 1 artırılıp sonra ekrana yazdırılmıştır.

Tip Dönüşümleri

Farklı tipteki sayısal değişkenler arası veri aktarımı tip dönüşümleri ile mümkün olmaktadır. Tip dönüşümünün gerçekleşmesi değişkenlerin sakladıkları verinin büyüklüğüne bağlıdır. Bir değişkenin kendisinden daha büyük bir değişken tipine dönüşümü derleyici tarafından otomatik yapılır (otomatic casting). Değişken kendisinden daha küçük bir değişkene dönüşecek ise dönüşeceği tip atama işlemi sırasında parantez içinde belirtilmelidir (manual casting). Manual casting işleminde değişkenin değeri dönüşeceği değişken türünün kapasitesinin üzerinde ise tüm değer saklanamaz. Ondalık bir değer tam sayıya dönüştürüldüğünde virgülden sonraki basamaklar saklanmaz.

```

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

        byte tavukSayisi=120;
        int kumesTavukSayisi=tavukSayisi;
        System.out.println(kumesTavukSayisi); // Ekran çıktısı 120

        int hindiSayisi=150;
        byte hindiMevcut= (byte) hindiSayisi;
        System.out.println(hindiMevcut); // Ekran çıktısı -106

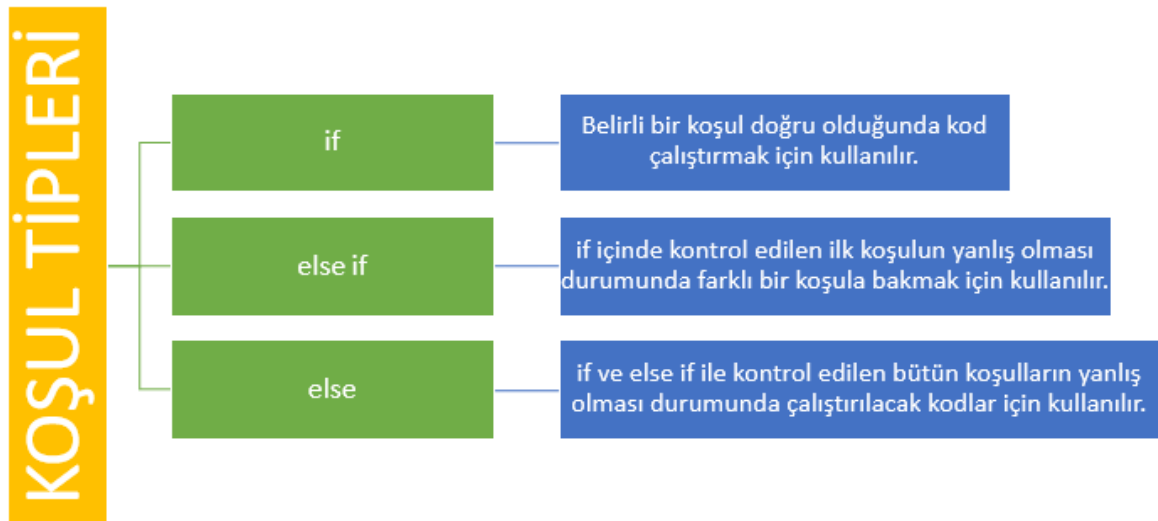
        double inekKilo=450.85d;
        short inekAgirlik= (short) inekKilo;
        System.out.println(inekAgirlik); //Ekran çıktısı 450
    }
}

```

Karar Yapıları (if/else)

Karar yapıları komutların belirli koşulların gerçekleşip gerçekleşmemesi durumlarına göre çalıştırılmasını sağlar. En yaygın kullanılan karar yapısı if-else blok yapısıdır. Karar yapıları ile programın önceden belirlenen durumlara karşı istenilen komut yapısını yürüterek tepki vermesi sağlanır. Örneğin sıcaklık değerine bağlı fan çalıştırılan bir uygulamada sıcaklık değerinin 30 dereceyi aşması durumunda fanın devreye girmesi isteniyorsa burada karar yapıları kullanılır.

Karar yapısı içerisinde bir koşul belirtilir. Program akışı sırasında gerçekleşen durumlar belirtilen koşulu sağlıyorsa bu duruma uygun kod bloğu, sağlamıyorsa başka bir kod bloğu çalıştırılır.



Koşullar parantez içerisinde belirtilir. Bir if içerisinde birden fazla koşul mantıksal operatörlerle birbirine bağlanabilir. Koşulların sağlanması durumunda çalışacak olan kod birden fazla satırdan oluşuyor ise kod bloğu süslü parantez içerisinde yazılmalıdır. Eğer tek bir satır ise parantez kullanımına gerek yoktur. Else if koşullarından herhangi birisi doğru olduğunda diğer else if veya else blokları çalıştırılmaz.

```
class Main {
    public static void main(String[] args) {
        if(koşul1)
        {
            //Koşul1 doğru ise çalışacak kodlar.
        }else if(koşul2){
            //Koşul1 yanlış ve koşul 2 doğru ise çalışacak kodlar.
        }else if(koşul3){
            //Koşul 1 ve koşul 2 yanlış, koşul 3 doğru ise
            //çalışacak kodlar.
        }else{
            //Tüm koşullar yanlış ise çalışacak olan kodlar.
        }
    }
}
```

Sınav ortalamasına göre öğrencinin harf notunu belirleyen bir program örneği aşağıda verilmiştir.

Ortalama : 0-29 arasında ise FF , 30-29 arasında ise DD, 40-59 arasında ise DC, 60-69 arasında ise CC, 70-74 arasında ise CB, 75-84 arasında ise BB, 85-100 arasında ise AA

```
1 class Main {
2     public static void main(String[] args) {
3         byte sinav1=95,sinav2=90;
4         byte ortalama;
5         ortalama=(byte) ((sinav1+sinav2)/2);
6         if(ortalama<30)
7         {
8             System.out.println("Ortalamanız: "+ortalama+ " , Başarı notunuz: FF"
9         }else if(ortalama<40){
10            System.out.println("Ortalamanız: "+ortalama+ " , Başarı notunuz: DD"
11        }else if(ortalama<60){
12            System.out.println("Ortalamanız: "+ortalama+ " , Başarı notunuz: DC"
13        }else if(ortalama<70){
14            System.out.println("Ortalamanız: "+ortalama+ " , Başarı notunuz: CC"
15        }else if(ortalama<75){
16            System.out.println("Ortalamanız: "+ortalama+ " , Başarı notunuz: CB"
17        }else if(ortalama<85){
18            System.out.println("Ortalamanız: "+ortalama+ " , Başarı notunuz: BB"
19        }
20        else{
21            System.out.println("Başarı notunuz: AA");
22        }
23    }
24 }
25 }
```

Araç tipine göre köprü geçiş ücreti hesaplayan örnek uygulama

SINIF	ARAÇ TİPİ	KÖPRÜ GEÇİŞ ÜCRETİ(TL)
1	AKS ARALIĞI 3.20 m'DEN KÜÇÜK İKİ AKSLI ARAÇLAR	10,50
2	AKS ARALIĞI 3.20 m VE 3.20 m.'DEN BÜYÜK UKOME KARARI GEREĞİNCE GEÇEBİLEN VE GEÇEMEYEN HER TÜRLÜ İKİ AKSLI ARAÇLAR	13,50
3	3 AKSLI HER TÜRLÜ ARAÇLAR	29,50
4	4 VE 5 AKSLI HER TÜRLÜ ARAÇLAR	58,75
5	6 VE YUKARI AKSLI ARAÇLAR	78,25
6	MOTOSİKLETLER	4,25

```

1 class Main {
2     public static void main(String[] args) {
3         byte aksSayisi=4;
4         float aksAraligi=3.48f;
5         if(aksSayisi==2)
6         {
7             if(aksAraligi<3.20)
8             {
9                 System.out.println("Köprü Geçiş Ücreti: 10,50 TL");
10            }else{
11                System.out.println("Köprü Geçiş Ücreti: 13,50 TL");
12            }
13        }else if(aksSayisi==3)
14        {
15            System.out.println("Köprü Geçiş Ücreti: 29,50 TL");
16        }else if(aksSayisi==4||aksSayisi==5)
17        {
18            System.out.println("Köprü Geçiş Ücreti: 58,75 TL");
19        }else
20        {
21            System.out.println("Köprü Geçiş Ücreti: 78,25 TL");
22        }
23    }
24 }

```

```

.04.4)
javac -classpath ./run_dir/junit-4.1
java -classpath ./run_dir/junit-4.12
Köprü Geçiş Ücreti: 58,75 TL

```

Karar Yapıları (switch)

Switch ifadesi ile belirtilen değişkenle aynı değere sahip case kod bloğunun çalıştığı koşul yapısıdır. Case kod bloğu break kodu ile biter. Break kodu çalıştırıldığında diğer case bloklarına bakılmadan switch yapısından çıkılır. Koşul değişkenine eşit hiçbir case bloğu çalışmaz ise default bloğundaki kodlar çalışır. Koşul değişkeni int veya char tipinde olmalıdır.

```

3 public class Kosullar {
4     public static void main(String[] args) {
5         byte bolumKodu=1;
6         switch (bolumKodu) {
7             case 1:
8                 System.out.println("Bilişim");
9                 break;
10            case 2:
11                System.out.printf("Biyomedikal");|
12                break;
13            case 3:
14                System.out.println("Elektrik");
15                break;
16            case 4:
17                System.out.println("Makina");
18                break;
19            case 5:
20                System.out.printf("Metal");
21                break;
22            case 6:
23                System.out.printf("Mobilya");
24                break;
25            default:
26                System.out.printf("Hatalı Giriş");
27        }
28    }
29 }
30 }

```

Yandaki örnek bölüm koduna göre bölüm adını ekrana yazdıran programdır.

bolumKodu koşul değişkenidir. Koşul değişkeninin değerine eşit olan case bloğu altındaki kodlar çalışır.

Döngüler

Bir işlem belirli sayıda tekrar ederse burada döngüler kullanılır. Döngü içerisinde belirtilen kod döngüde belirtilen koşula göre istenilen sayıda tekrar çalıştırılır. Bir işlemi istenildiği kadar tekrar ettirmek için döngüler kullanılır. Diziler, listeler ve hashmap gibi birden fazla değer tutan yapılarda her değerde aynı işlemi yapmak için döngüler kullanılır. Döngü yapısında tekrarlanacak işlem ve işlemin ne kadar süre yapılacağı belirtilmelidir. Döngünün duracağı koşul belirtilmez ise sonsuz döngü meydana gelir. Sonsuz döngü programın kırılmasına yol açan istenmeyen bir durumdur.

For, while ve do while olmak üzere 3 tip döngü vardır. For döngüsü dizi yapıları için basit bir gösterime sahiptir. Bu sebeple dizilerde daha çok for döngüsü tercih edilir. While ve do while döngülerinin yapıları birbirine benzer.

FOR Döngüsü

Başlangıç bölümü döngü ilk kez çalıştığında döngünün başlangıç değerini belirler. Döngü her çalıştığında artış bölümüne göre başlangıç değişkeninin değeri artırılır. Koşul sağlandığı sürece döngü tekrar tekrar çalışmaya devam eder. Başlangıç değeri koşula uymadığında döngüden çıkılır.

for(başlangıç ; koşul ; artış) {

Tekrarlanacak kodlar

}

```
for(int i=1;i<10;i++){  
    System.out.println("Abidinpaşa Bilişim");  
}
```

```
Abidinpaşa, i değeri=1  
Abidinpaşa, i değeri=2  
Abidinpaşa, i değeri=3  
Abidinpaşa, i değeri=4  
Abidinpaşa, i değeri=5  
Abidinpaşa, i değeri=6  
Abidinpaşa, i değeri=7  
Abidinpaşa, i değeri=8  
Abidinpaşa, i değeri=9
```

```
public static void main(String[] args) {  
    int toplam=0;  
    for(int i=1;i<10;i++){  
        toplam=toplam+i;  
    }  
    System.out.println("Toplam="+toplam);  
}
```

Toplam=45

Yukardaki ilk örnekte Abidinpaşa Bilişim metni 9 kez ekrana yazdırılmıştır. İkinci örnekte ise 1 ile 9 arasındaki sayıların toplamı alınmıştır. Döngü yapısındaki başlangıç değeri, dizi içindeki elemanların indeks numarası olarak kullanılabilir. Bu sayede dizi içindeki elemanlara döngü içerisinde ulaşarak işlem yapılabilir.

For döngüsü dizilerde kullanım için özel bir gösterime sahiptir. Aşağıdaki örnekte verilen eleman değişkeni aylıkYagis dizisi içine girerek dizinin elemanlarını sırayla çeker. for(int eleman:dizinin adı)

```

public static void main(String[] args) {

    int[] aylıkYagis={39,35,39,41,51,34,13,11,17,27,31,44};

    int yıllıkToplam=0;
    double yıllıkOrtalama;

    for(int eleman:aylıkYagis){
        yıllıkToplam=yıllıkToplam+eleman;
    }

    yıllıkOrtalama = yıllıkToplam / aylıkYagis.length;
    System.out.println("Yıllık Yağış Ortalaması= "+yıllıkOrtalama);
}

```

While & Do While Döngüleri

While ve do while döngü içerisinde bir sayaç değişkeni barındırır. Döngü her çalıştığında döngü içerisindeki sayaç artar. Şart sağlandığı sürece döngü çalışmaya devam eder. While döngüsü şartı kodlar çalıştırılmadan kontrol ederken, do while döngüsü ise şartı kod çalıştırıldıktan sonra kontrol eder. Do while döngüsündeki komutlar şart sağlanmasa bile 1 kez çalıştırılır.

Do While Döngüsü

```

public static void main(String[] args) {

    int i=0;

    do{
        System.out.println("Abidinpaşa");
        i++;
    }while(i<5);

}

```

While Döngüsü

```

public static void main(String[] args) {

    int i=0;

    while(i<5){
        System.out.println("Abidinpaşa");
        i++;
    }

}

```

Diziler

İçinde aynı tipte birden fazla değer saklayabilen yapılara dizi denir. Aynı tipteki değerleri farklı isimlerdeki değişkenlerde ayrı ayrı saklamak yerine tek bir değişken içinde saklamak belleğin daha verimli kullanılmasını sağlar. Dizi oluşturmak için iki yöntem vardır.

1. Yöntemde dizinin tipi, kaç eleman barındıracağı ve dizinin ismi tanımlanır. Elemanlar daha sonra diziye atanır.

```

3 public class Kosullar {
4     public static void main(String[] args) {
5
6         //1.YOL
7         String[] apartmanl0c=new String[6];
8
9         apartmanl0c[0]="TAN";
10        apartmanl0c[1]="EMRE";
11        apartmanl0c[2]="KEMAL";
12        apartmanl0c[3]="MEHMET";
13        apartmanl0c[4]="FATMA";
14        apartmanl0c[5]="ÖMER";
15    }
16 }
17

```

2.Yöntemde ise dizinin tipi ve adı tanımlanırken elemanları da diziye atanır.

```

3 public class Kosullar {
4     public static void main(String[] args) {
5
6         //2.YOL
7         String[] apartmanl0c= {"TAN", "EMRE", "KEMAL", "MEHMET", "FATMA", "ÖMER"};
8     }
9 }
10

```

Dizinin bir elemanını değiştirmek veya boş bir elemana değer atamak için elemanın bulunduğu numaraya (index) yeni değer atanır. Dizi elemanında kendisine en son atanan değer saklanır. Dizideki bir elemanın değerini okumak için indeks numarası mutlaka köşeli parantez içinde belirtilmelidir.

```

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

        String[] apartmanl0c=new String[6];

        apartmanl0c[0]="TAN";
        apartmanl0c[1]="EMRE";
        apartmanl0c[2]="KEMAL";
        apartmanl0c[3]="MEHMET";
        apartmanl0c[4]="FATMA";
        apartmanl0c[5]="ÖMER";

        System.out.println(apartmanl0c[2]);

        apartmanl0c[2]="VOLKAN";

        System.out.println(apartmanl0c[2]);

    }
}

```

Bir dizinin eleman sayısını öğrenmek için length özelliği kullanılır.

```

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

        String[] uyeListesi={"Ali", "Ahmet", "Ayşe", "Mert", "Hasan", "Selim", "Feyza", "Ece", "Sefa"};
        System.out.println(uyeListesi.length);

        //ekran çıktısı: 9

    }
}

```

Metotlar

Bir kod bloğunu program içerisinde birden fazla yerde kullanmak gerekebilir. Böyle bir durumda aynı kodları birden fazla kez tekrarlamak program boyutunun büyümesine ve bellekte daha fazla alan kaplamasına yol açar. Bunu engellemek için metotlar kullanılabilir. Metotlar bir ismi olan dışarıdan parametre olarak adlandırılan veri alabilen ve istenildiği takdirde geriye değer dönebilen kod bloklarıdır. Bir metodun oluşturulması çalışacağı anlamına gelmez. Metotlar çağrılmadığı takdirde çalışmazlar.



- ✓ Bütün kodları main metodu içerisine yazmak programı karmaşık hale getirir.
- ✓ Metotlar erişilebilirlik seviyesine göre ihtiyaç duyulan her yerde çağrılarak kullanılırlar.
- ✓ Metotlar sayesinde programın farklı yerlerinde ihtiyaç duyulan aynı işler için tekrar tekrar aynı kodların yazılmasına gerek kalmaz.
- ✓ Metotlar farklı amaçlara yönelik kodların bir arada bulunarak programın kolay yönetilebilmesini ve anlaşılabilmesini sağlar.
- ✓ Metotlar program akışı içerisinde çağrılmadıkları sürece çalışmazlar.

```
public class Donguler {  
    public static void main(String[] args) {  
        5  
        okulYazdir();  
    }  
    1    2    3    4  
    public static void okulYazdir(){ ←  
        6  
        System.out.println("Abidinpaşa MTAL");  
        System.out.println("Bilişim Teknolojileri");  
    }  
}
```

1. Metoda nereden erişileceğini belirtir. Public ifadesi programın her yerinden metodun çağırabileceğini gösterir.
2. static kavramı metodun classa ait olduğunu gösterir. Aynı class içerisinde metodun örneği oluşturulmadan ismi yazılarak kullanılabilir. Static olarak tanımlanan metotlara başka bir sınıftan ulaşılamaz.
3. Metodun çağırıldığı yere değer döndürmeyeceğini gösterir. İşlem metodun içinde tamamlanır sonuçla başka bir işlem yapılmaz
4. Metodun adıdır.
5. Metot ismiyle çağırılır.
6. Metot içerisindeki kodlardır. { } parantezleri içerisine yazılmalıdır.

Metot içerisindeki değişkenlere değerleri metot içerisinde verilebileceği gibi metodun çağırıldığı yerden de gönderilebilir. Metodun dışardan aldığı değerlere parametre denilir. Metodun kullanacağı parametreler ve tipleri metot başlığında tanımlanır. Metot çalıştığında çağırıldığı yere bir sonuç gönderecekse gidecek sonucun tipi metot başlığında tanımlanır. Gidecek sonuç metodun en altında return komutu ile tanımlanır.

```
public static void main(String[] args) {  
  
    int toplamaSonucu=toplama(30,20);  
    System.out.println(toplamaSonucu);  
  
}  
  
public static int toplama(int sayi1,int sayi2){  
  
    int sonuc;  
    sonuc=sayi1+sayi2;  
    return sonuc;  
  
}
```

Yukardaki örnek incelendiğinde, toplama(30,20) kodu ile birlikte toplama metodu sayi1 değişkeni için 30, sayi2 değişkeni için ise 20 değerleri ile çağırılır. Toplama metodu çalışır sonuc değişkeninin değeri return komutu ile sonuç değişkeninin değeri metodun çağırıldığı yere gönderilir. Sonucun tipi metot başlığında int olarak belirtilmiştir.

Sınıf Kavramı ve Erişim Belirleyiciler

Java programlama dili nesne tabanlı (object-oriented) bir dildir. Java gibi programlama dillerinde her şey nesneler ve sınıflarla ilgilidir. Sınıf soyut bir kavramken nesne sınıfın somutlaşmış bir dışavurumudur. Sınıf kavramı ortak özelliklere sahip bir yapıyı temsil ederken, nesne ise sınıftan türetilen, sınıfa ait temel özelliklere sahip olmasının yanında kendisine has özellikleri de olan yapılardır. Bir sınıf başka bir sınıftan türetilir. Bu durumda alt sınıf, üst sınıfın tüm özelliklerini miras alır. Buna kalıtım denir. Sınıfların nitelikleri değişkenler ve özelliklerle belirtilirken, yapabildikleri ise metotlarla tanımlanır.

Öğrenci kavramı bir sınıf olarak düşünülebilir. Her öğrencinin sınıfı, okulu, not ortalaması, okul numarası gibi ortak bir takım özellikleri vardır. Öğrenci sınıfının içerisinde yer alan sınıf geçme metodu öğrencinin not ortalamasına ve koşullara göre sınıfı geçip geçmeyeceğini belirler.

Öğrenci sınıfından somut bir öğrenci oluşturulduğunda bu nesne olacaktır. Mehmet isimli öğrenci, sınıftan oluşturulan bir nesnedir. Dolayısıyla Mehmet öğrencisi, öğrenci sınıfının temel özelliklerini ve metotlarını kullanabilir.

```
package com.kodlis.myapplication;

import java.security.PublicKey;

public class Oğrenci {
    public String Adi;
    public int No;
    public String Sinif;
    public double NotOrt;

    public static void SinifGecme(double notOrt) {
        if(notOrt>50){
            System.out.println("Sınıfı geçer");
        }else{
            System.out.println("Sınıfta Kaldı");
        }
    }
}

package com.kodlis.myapplication;

class Main {
    public static void main(String[] args) {
        Oğrenci ogr=new Oğrenci();
        ogr.Adi="Mehmet";
        ogr.No=5885;
        ogr.Sinif="AMP 12/F";
        ogr.NotOrt=58.5;

        Oğrenci.SinifGecme(ogr.NotOrt);
    }
}
```

Sınıflara, metotlara, değişkenlere programın herhangi bir bölümünden erişimini denetim altına almak için erişim belirleyiciler kullanılır. Erişim belirleyiciler metot ve class tanımlanırken yazılır. Java'da kullanılan erişim belirleyiciler:

Public: Herhangi bir engel olmadan bütün sınıflar erişebilir.

Private: Sadece tanımlandığı sınıf içerisinde kullanılabilir.

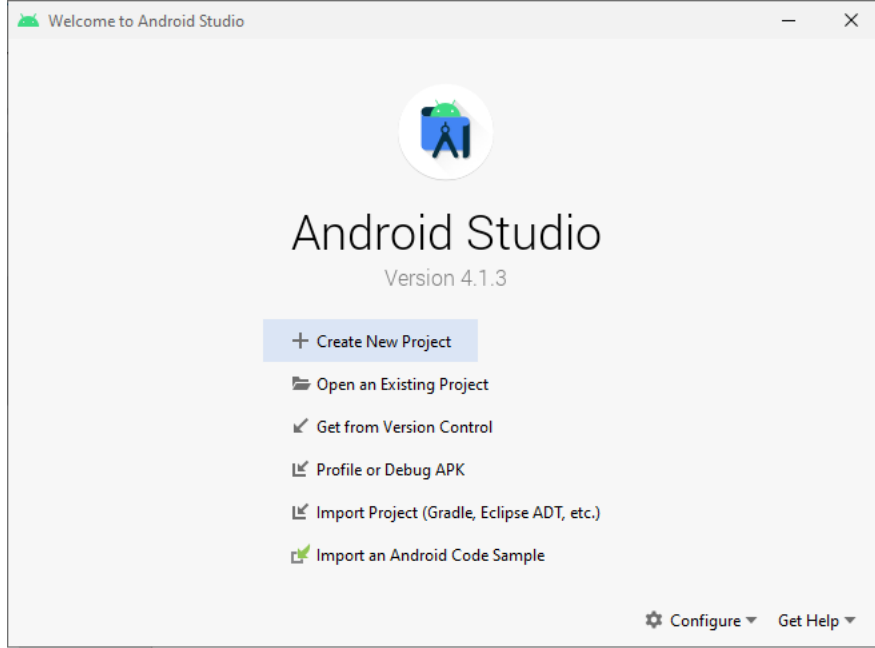
Protected: Alt sınıflar ve aynı paketdeki sınıflar erişebilir.

private protected: Sadece alt sınıflar erişebilir.

<default> (ön-tanımlı) : Eğer erişim belirteci yazılmazsa. Default geçerli olur. Bu durumda aynı paket içerisindeki sınıflar erişebilir.

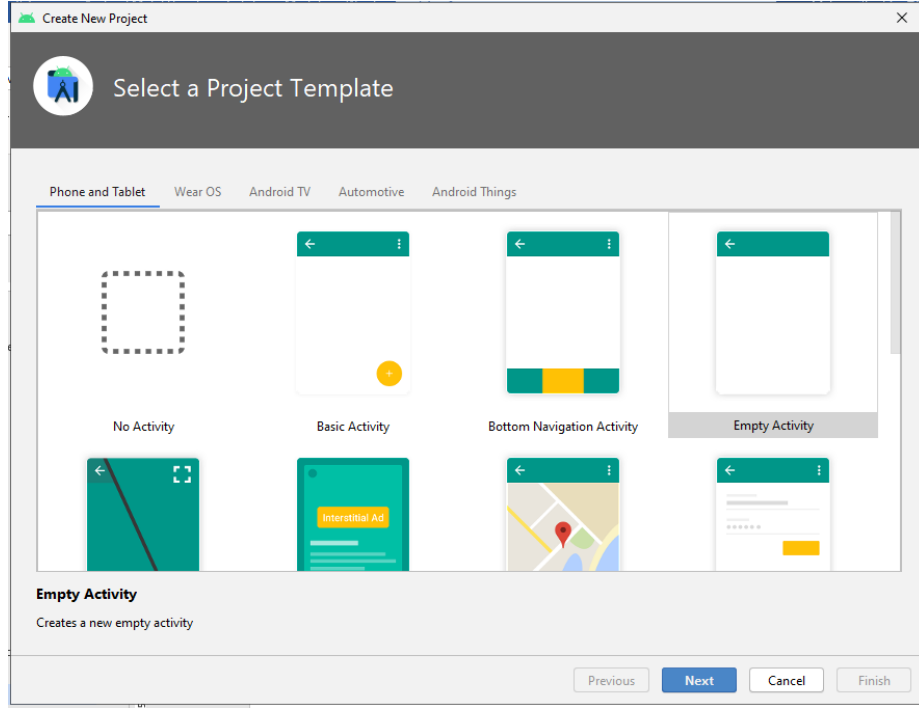
Yeni Bir Proje Oluřturma

Android Studio'da yeni bir proje oluřturmak iin Create New Project seeneėine tıklanır. Eėer mevcut bir proje aılacak ise Open an Existing Project seilir.



Aktivite Oluřturma

Mobil uygulamaların kullanıcılarla etkileřim kurduėu yapılar activity olarak isimlendirilir. Activity nesneleri iin uygulama ierisindeki ekranlar desek yanlış olmaz. Uygulamanızda 5 farklı ekran varsa 5 farklı activity nesnesinin olması gerekmektedir. Uygulamalar da yer alan ekranlar birer activity sınıfının rneėidir. Activityler zerlerinde kullanıcıların bilgi girmesini ya da iřlenmiř ıktılara eriřimlerini saėlayan view ėelerini barındırırlar. Android Studio ierisinde eřitli zellikte hazır řablon activityler bulunmaktadır. Projeye uygun olan activity hazır řablonu seilebileceėi gibi kendi tasarınızı yapabilmek iin ise Empty Activity seilir. Altteki grsel grleceėi gibi sadece mobil telefonlar iin deėil giyilebilir cihazlar ve televizyonlar iinde hazır řablon activityler bulunmaktadır.



Activity seçildikten sonra aşağıdaki ekran karşımıza gelecektir. Burası proje ile ilgili temel bilgilerin girildiği ekrandır. Proje oluştururken girilecek bilgiler şunlardır:

Name: Projenin ismi bu alana yazılır.

Package Name: Bu alan genellikle uzantılı bir domain adı barındırır. Buraya vereceğiniz isim sizin uygulamanızın Google Play üzerindeki diğer uygulamalardan ayırt edilmesini sağlayacaktır. Kodlis isimli bir şirketin Obezite, Kalp, Doktorum, Sağlık Bilgilerim isimli 4 uygulama geliştirdiğini varsayalım. Bu durumda bu uygulamaların package nameleri sırasıyla:

com.kodlis.obezite

com.kodlis.kalp

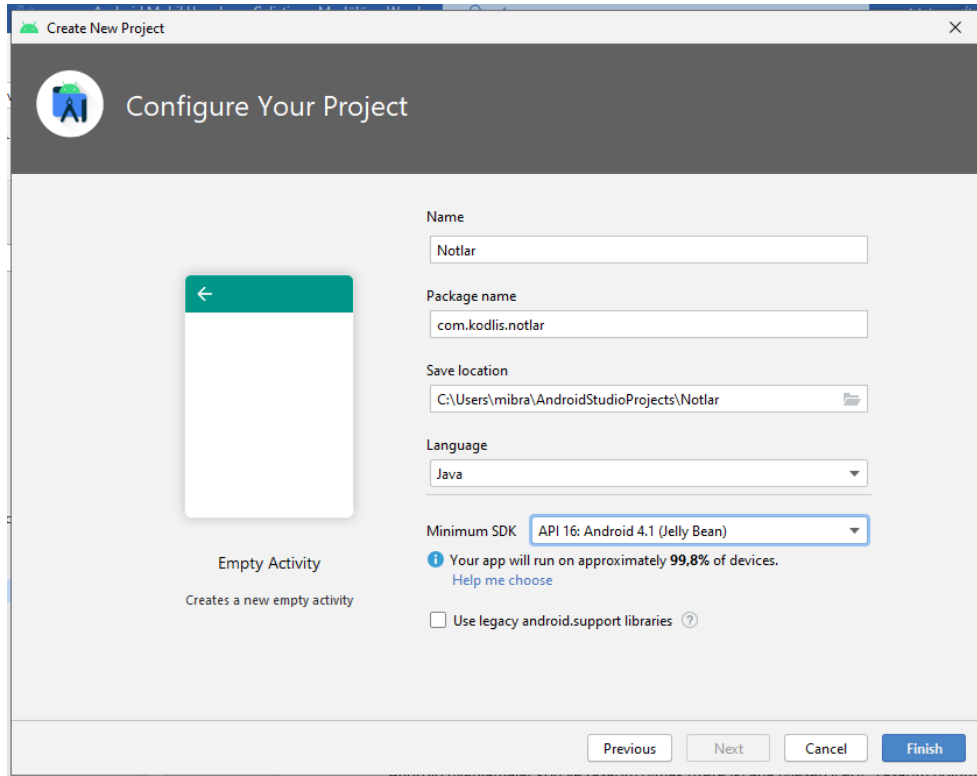
com.kodlis.doktorum

com.kodlis.saglikbilgilerim olacaktır.

Save Location: Bu alanda proje dosyalarının bilgisayarınıza kaydedileceği alan seçilir.

Language: Projenizde kullanacağınız programlama dilini belirlediğiniz yerdir. Java ve Kotlin olmak üzere 2 seçenekten oluşur.

Minimum SDK: Uygulamanızın çalışacağı en düşük SDK versiyonunu seçebilirsiniz. Burada dikkat edilmesi gereken husus en uygun seçimin yapılması gerekliliğidir. En yeni özellikleri kullanabilmek için yüksek SDK seçilirse bu durumda uygulamanızın hitap edeceği Android sürümüne sahip az sayıda bir telefon pazarı olacaktır. Daha çok telefona hitap etmek için düşük SDK seçilirse bu sefer de projenizde kullanmanız gerekebilecek birtakım özelliklere erişiminiz ortadan kalkar.



Activity Yaşam Döngüsü

Activtyler belirli bir yaşam döngüsüne sahiptir. Yapılacak olan uygulama içerisinde activtylerin yaşam döngülerinin doğru planlanması gereksiz bellek tüketiminin önüne geçecektir. Activitynin kullanıcının tercihleri doğrultusunda gireceği moda uygun çalışacak olan kodlar yaşam döngüsünde ilgili metodun içine yazılmalıdır.

onCreate: Activity ilk oluşturulduğunda çağırılan metottur. Bu metod içerisinde activitynin tasarımını barındıran layout dosyası yüklenerek ekran hazırlanmış olunur.

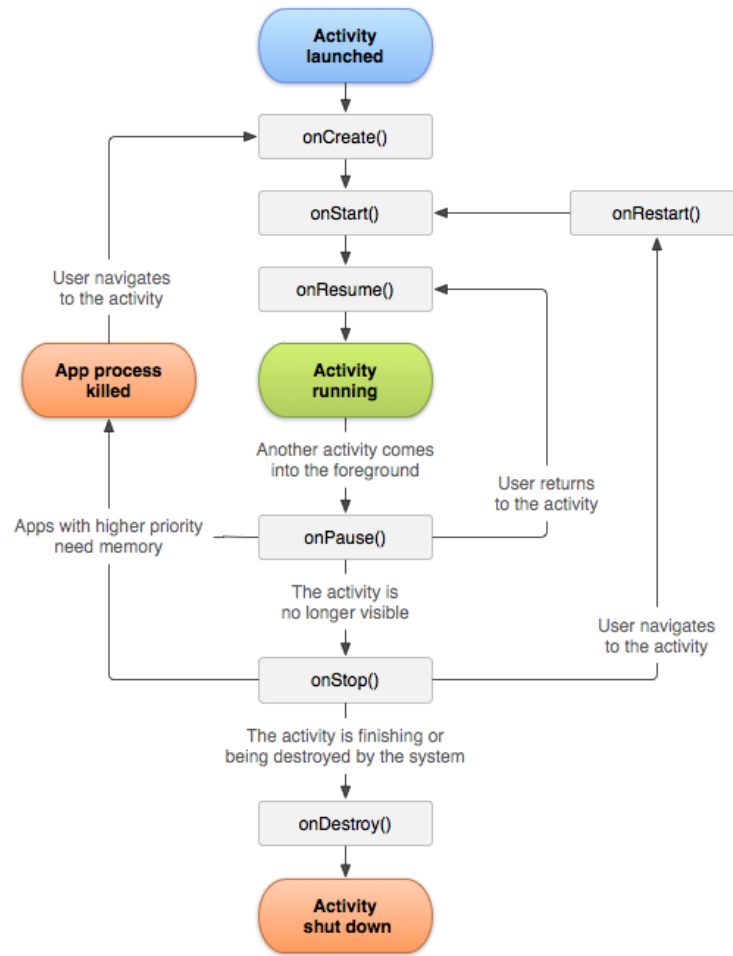
onStart: onCreate metodu ile activity ekran tasarımı tamamlandıktan sonra çağırılan metottur.

onResume: Herhangi bir sebepten dolayı durdurulan activity tekrar çalışmaya başladığında bu metod çalışır.

onPause: Activitynin durdurulması durumunda çalışan metottur.

onStop: Activity arka plana atıldığı anda işleme girer. Aşağıdaki şekilde görüleceği üzere kullanıcı uygulamaya tekrar geri dönüş yaparsa onRestart() metodu ile activity çalışmaya devam eder. Eğer kullanıcı uygulamaya dönmez ise onDestroy metodu ile tamamen kapatılır.

onDestroy: Activitynin kullandığı tüm kaynaklar geri alınır ve activity bellekten kaldırılarak tamamen kapatılmış olunur.



Aşağıdaki kod yapısı incelendiğinde bir activity nesnesine ait `onCreate` metodu görülmektedir. Activity nesnesine ait view öğelerinin bulunduğu xml formatındaki tasarım `onCreate` metodu içerisinde çağrılmaktadır.

```

package com.kodlis.notlar;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}

```

Android Studio Proje Yapısı

Proje oluşturulduktan sonra ekranın sol tarafında proje yapısı görüntülenir. Proje yapısı içerisinde yer alan dosya ve klasörler geliştirilecek mobil uygulamanın birer parçasıdır. Her bir klasör farklı özellikteki dosyaların depolanmasını sağlar.

AndroidManifest.xml: Uygulamanın kullanıcıdan alacağı izinler, uygulamanın özellikleri, uygulamanın bileşenleri gibi uygulamayla ilgili en önemli ve en temel yapıların tanımlandığı dosyadır.

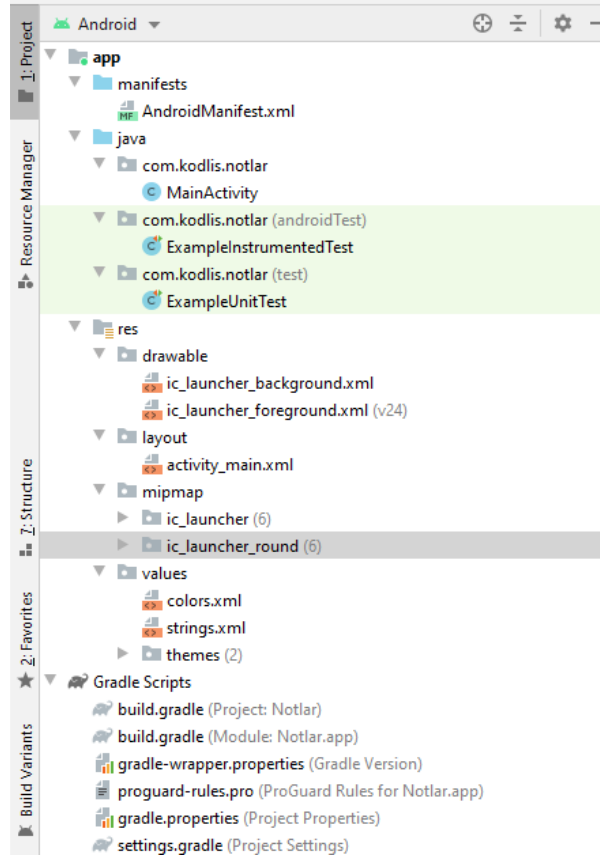
Java klasörü: Activitylerin Java veya Kotlin ile yazılmış kod dosyalarını içerir.

Res klasörü: Res klasörü uygulama içerisinde yer alacak grafik, metin ve ses gibi dosyaları barındırır. Bu klasör altında yer alan diğer klasörler şunlardır:

drawable klasörleri: Uygulama içerisinde kullanılacak resim dosyalarını barındırır.

mipmap klasörü: Uygulamanızın açılış logolarını içerir.

layout klasörü: Activitylerin tasarım dosyaları bu klasör altında bulunur.



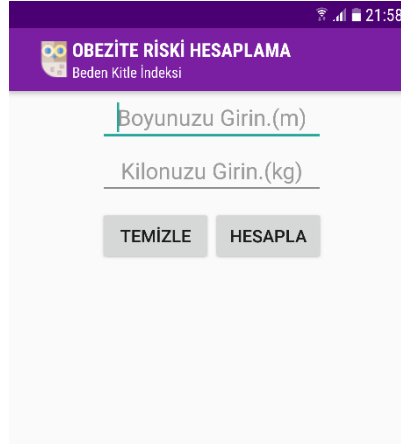
values klasörü: Uygulama içerisinde kullanılacak sabit metin, renk tanımlamaları, stil tanımlamaları gibi özellikleri barındıran dosyaları içerir.

Gradle Scripts: Derleme ile ilgili özellikleri içeren dosyaları barındırır.

Layoutlar ile Çalışmak

Android uygulamalar kod ve tasarım olmak üzere iki ana bileşen içerir. Tasarım bölümünde kullanıcıyla etkileşim kuracak butonlar, checkboxlar gibi öğeler yer alırken, kod bölümünde ise tasarım bölümünden gelen davranışlara ne cevap verileceğini işleyen kodlar yer alır.

Aşağıdaki şekilde kilonuzu boyunuzu girdiğinizde beden kitle indeksini veren bir uygulamanın ekran görüntüsü bulunmaktadır. Bu uygulamada kullanıcıdan veri almak için kullanılan metin kutucukları ve hesaplama işleminin yapılmasını sağlayan buton uygulamanın tasarım kısmıdır yani sizin gördükleriniz. Uygulamanın kod bölümü ise Hesapla butonuna basıldığında kullanıcıdan gelen verilerle matematiksel işlemi yapıp sonucu hesaplamaktır.

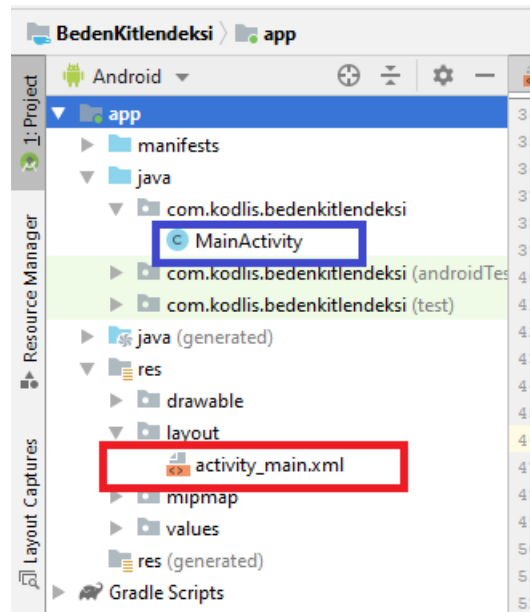


Şekil 1 Beden Kitle Hesaplama Uygulaması

Kullanıcıyla etkileşim kurulmasını sağlayan öğelere view adı verilir. Birden fazla view öğesinin bir araya gelerek oluşturduğu yapı ise viewgroup olarak isimlendirilmektedir. Şekil 1’deki viewleri (butonlar, edittextler) bir arada tutan, ekran üzerindeki yerleşimlerini düzenleyen yapılara layout denir. Layout kelimesinin Türkçe karşılığı yerleşim demektir. Layoutlar birden fazla view öğesini bir arada tuttuğu için bir bakıma viewgroup sayılır. Ekranda kullanıcıya gözükecek olan bütün öğelerimiz bir layout içinde yer alacağından tasarımın temelini layoutlar oluşturur diyebiliriz. Layout planlamanın doğru yapılması ilerde karşılaşılabilecek tasarım hatalarının minimum olmasını sağlar.

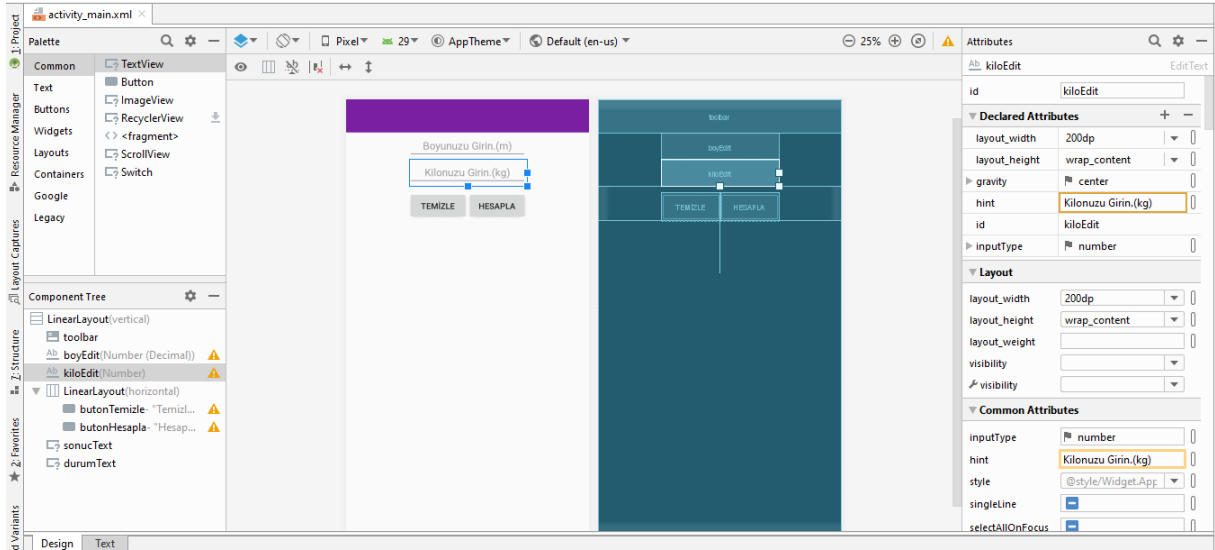
Layoutlar içlerinde view öğeleri tuttuğu gibi layoutlar da tutabilir. Yani bir layout diğerinin içinde olabilir. Böyle bir durumda içteki layout dıştaki layoutun çocuğu(child) olurken doğal olarak dıştaki layoutta içteki layoutun ebeveyni (parent) olacaktır.

Layout dosyaları xml uzantılı ve xml kodlarıyla yazılan dosyalardır. Bu dosyalardaki kodlara göre view öğelerinin özellikleri ve yerleşimleri ayarlanır. Mobil uygulamadaki sayfalara ait tasarımı barındıran bu layout dosyaları, res (resource) klasörü altında ki layout klasörü içerisinde yer alır.

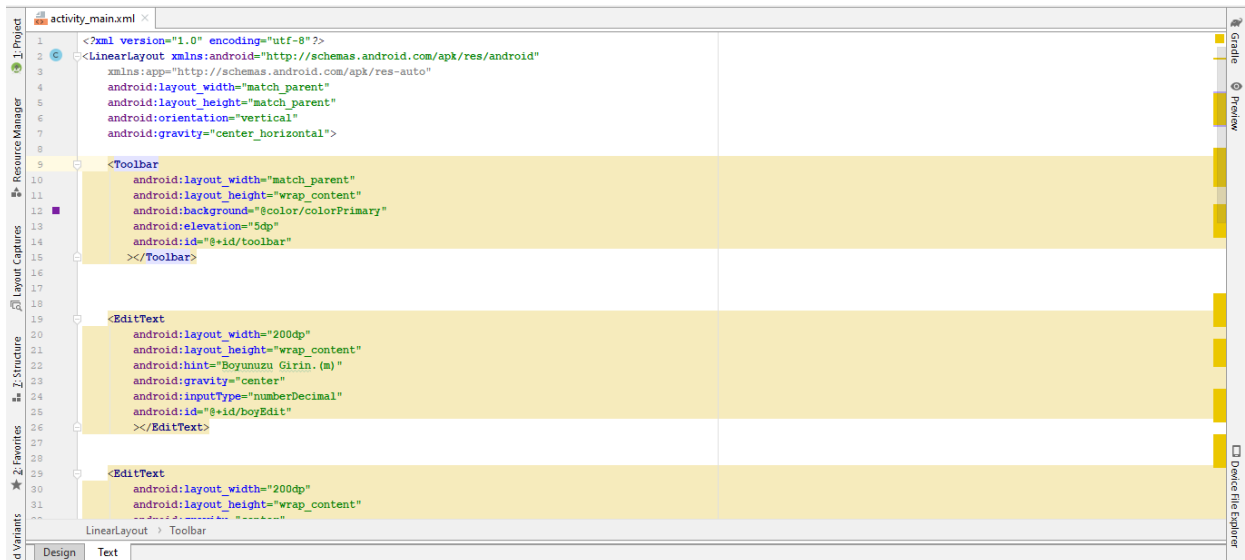


Yukardaki proje yapısına bakıldığında kırmızı ile çerçevelenmiş bölüm activity_main isimli sayfanın xml uzantılı layout tasarım dosyasıdır. Mavi çerçeve ile belirtilen dosya ise activity_main isimli sayfanın kod bölümünü içeren java dosyasıdır.

Farklı tasarım ihtiyaçlarına yönelik farklı layout türleri vardır. Uygulamalarda genellikle linearlayout relativelayout veya constraintLayout tercih edilir. Layout üzerine viewlerin yerleşimi ve özelliklerinin ayarlanması Android Studio’da sürükleyip bırak yöntemi ile kod yazmadan yapılabileceği gibi (design) tasarıma tam hakim olmak isteniyorsa işlemlerin xml kodlarıyla (Text) yapılması daha verimli olacaktır.



Şekil 2Layout Tasarımı Design



Şekil 3Layout Tasarım Text

LINEARLAYOUT

LinearLayout uygulama tasarımında sıkça kullanılan bir layout çeşididir. LinearLayoutta tüm view öğeleri programcının tercihi doğrultusunda yatayda veya dikeyde sırayla ekrana dizilirler. Bir sayfanın tasarımı ilk olarak layoutun yerleşimi ile başlar. Dolayısıyla xml tasarım dosyasına ilk olarak tüm öğeleri üzerinde tutacak olan layout yerleştirilir. LinearLayout oluşturmak için

aşağıdaki kodlar yazılır. Android Studio'yu açtığınızda layout dosyanızın içindekileri silip aşağıdaki kodları yapıştırdığınız takdirde LinearLayoutunuz hazır olacaktır. Şimdi bir linearlayoutu oluşturan kodları inceleyelim.

```
activity_main.xml x
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3   xmlns:app="http://schemas.android.com/apk/res-auto"
4   android:layout_width="match_parent"
5   android:layout_height="match_parent"
6   android:orientation="vertical" >
7
8
9 </LinearLayout>
```

Şekil 4 LinearLayout

XML kodlarında yeşil renkli olan bölümler özelliğe değeri veren parametrelerdir. 6.Satırda orientation özelliği layout içindeki öğelerin nasıl sıralanacağını belirler. Sıralamanın nasıl olacağını ise yeşil bölüm olan parametre belirler.

1. Layout dosyalarının xml kodlarından oluştuğunu hatırlayın. Encoding komutu karakter kodlamasının türünü belirtir. Utf-8 karakter kodlaması Türkçe karakterleri de içeren çok sayıda farklı karakteri barındıran bir kodlama türüdür.
2. LinearLayoutın başlangıcı olan satırdır. LinearLayout özellikleri buradaki küçük işareti ile başlar 6.satırda işaretlenmiş büyük işaretinin bulunduğu yere kadar devam eder. LinearLayout'a ait bütün özellikler bu iki işaret arasında yer almalıdır. Xmlns ile devam eden satır ise standart bir işlem olup XML dosyalarında olması gereken ad alanının (namespace) karşılığıdır. XML dosyalarındaki etiketlerin birbirleriyle çakışmasını engeller.
3. 3.Satırdaki ifade de bununla ilgilidir. Burada da namespace yerine ön ek (prefix) kullanılmıştır.
(Fazlası için : https://www.w3schools.com/xml/xml_namespaces.asp)
4. Layoutun genişliği yani yatayda ne kadar yer kaplayacağıdır.
5. Layoutun uzunluğu yani dikeyde ne kadar yer kaplayacağıdır.
6. LinearLayoutın viewleri sıraladığını biliyoruz. Bu komutla sıralama işleminin dikeyde mi yoksa yatayda mı yapılacağı belirlenir. "vertical" parametresi ile dikey sıralama, "horizontal" parametresi ile ise yatay da sıralama yapılır.
9. LinearLayoutu bitiren koddur. LinearLayout içinde olması gereken tüm viewler(buton, resim, text vb) bu satırdan önce 6.satırdan sonra olmalıdır.

Layout ve view öğelerinin genişlik ve uzunluk değerlerinin mutlaka verilmesi gerekir. Boyutları belirlemek için aşağıdaki 3 özellikten birini seçmelisiniz.

- a. **"match_parent"** : Ebeveynin özelliğini al. Mevcut linearlayoutumuz herhangi bir layoutun içinde değil. Bu durumda parentı yani ebeveyni telefon ekranı olacaktır. Bu seçeneği seçtiğinizde layoutun boyutu telefon ekranı kadar olacak demektir.

- b. **"wrap_content"** : İçerik kadar boyutlandır. Bu seçeneği seçerseniz layoutun boyutu içindeki öğeler kadar olacaktır. Öğeler telefon ekranınızın yarısına kadar uzanıyorsa layoutta orada bitecektir.
- c. **"250dp"** : Yukardaki seçenekler dışında sabit bir değer vermek için kullanabilirsiniz.

Layout dışında kalan öğeler ekran dışında kalırlar. Ekrana sığdırılması için otomatik olarak herhangi bir optimizasyon yapılmaz.

LAYOUT WEIGHT

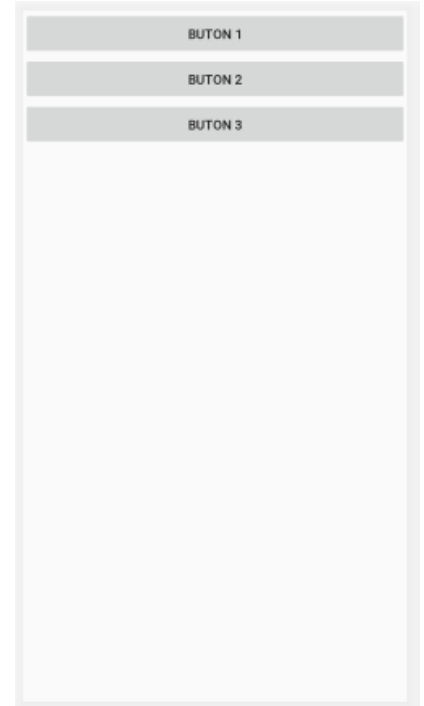
"layout_weight" özelliği view öğeleri için kullanılır. Öğenin layout içindeki ağırlığını belirler. Bu özellik belirtilmez ise layout içindeki tüm öğelerin weight değerleri 0 olacaktır. Aşağıdaki örnekte LinearLayout içerisinde dört adet buton yerleştirilmiştir. Butonlara weight değeri atanmadığı için hepsinin weight değeri varsayılan olarak 0'dır. orientation="vertical" olarak belirtildiği için 3 adet buton sırayla layout içerisinde dikey olarak sıralanmıştır.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".TestActivity"
    android:orientation="vertical">

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Buton 1">
    </Button>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Buton 2">
    </Button>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Buton 3">
    </Button>

</LinearLayout>
```

Şekil 6 Weight Özelliği Kullanılmayan



Şekil 5 Butonların Yerleşimi

Örneğe dikkat edilirse butonların genişliği match_parent olarak belirlenmiştir. Bunun anlamı butonların genişliğinin parentı komunundaki layoutun genişliği kadar olması demektir. Uzunlukları ise wrap_content olarak belirlendiği için bu değer butonların içindeki Buton 1 yazısı kadar olacaktır.

Şimdi aynı örneği farklı weight değerli ile yapalım. Buradaki amacımız butonların ağırlıklarına göre layout içerisinde kapladıkları alanın nasıl değiştiğini görmek.

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".TestActivity"
    android:orientation="vertical">

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Buton 1"
        android:layout_weight="1">

    </Button>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Buton 2"
        android:layout_weight="2">

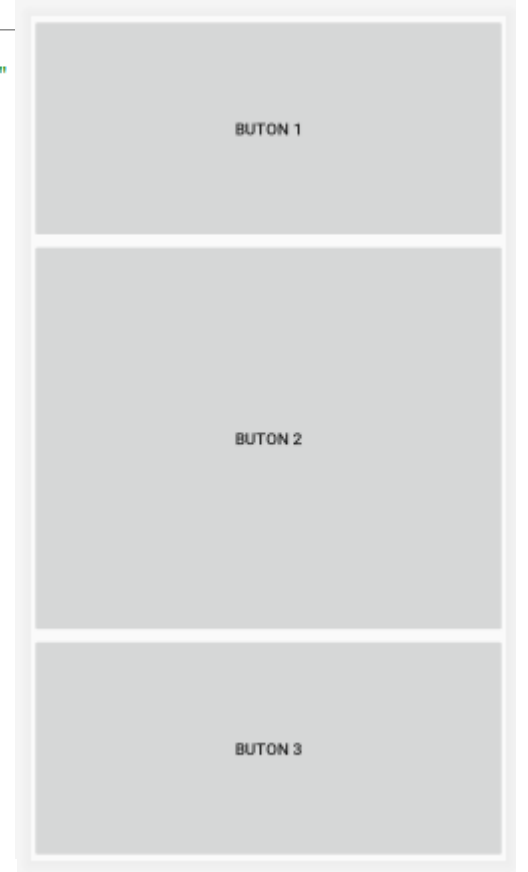
    </Button>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Buton 3"
        android:layout_weight="1">

    </Button>

</LinearLayout>

```

Şekil 7 Weight Özelliği



Şekil 8 Butonların yerleşimi

İlk örnekten farklı olarak her bir butona weight değeri verilmiştir. Birinci butonun weight değeri 1, ikinci butonun 2 ve üçüncü butonun ise 1 olarak belirlenmiştir. Layout bu weight değerlerine göre butonların dikeyde kaplayacakları alanları otomatik olarak ayarlamıştır. Bir bakıma layout 4 eşit parçaya bölünmüştür. Birinci parça weight 1 değeri ile birinci butona, ikinci ve üçüncü parçalar weight 2 değeri ile ikinci butona, son parça olan 4. Parça ise weight 3 değeri ile 3.butona verilmiştir.

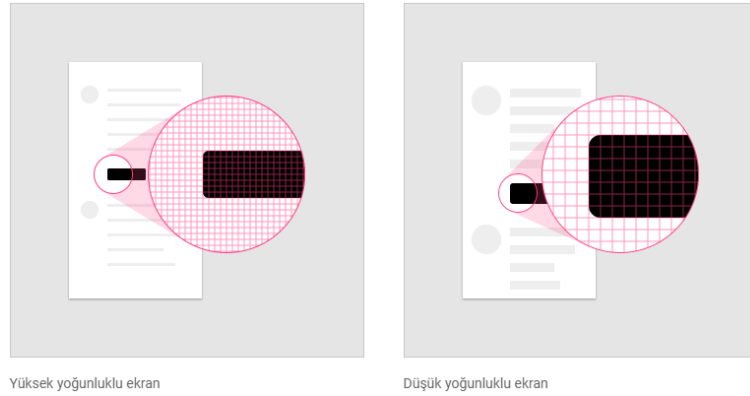
“layout_weight” özelliği kullanıldığında butonun uzunluğu için belirlediğimiz Wrap_content özelliği layout tarafından dikkate alınmamış, öncelik weight değerine verilmiştir. Bu özelliği kullanıp kullanmamak tamamen programcıya bağlıdır. Eğer uygulamanızdaki öğelerin layout üzerinde örneğe benzer bir dağılımla sıralamak isterseniz o zaman weight özelliğini kullanmanız gerekecektir.

Viewler ile Çalışmak

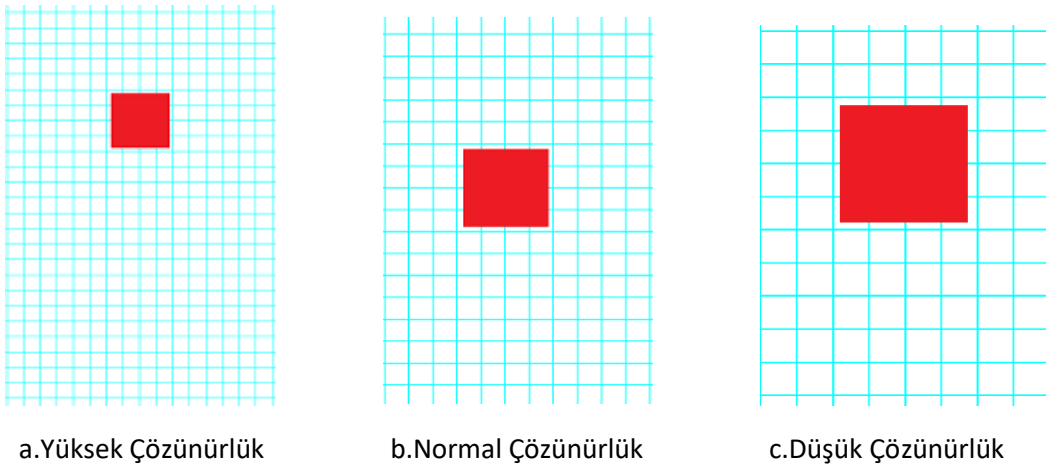
Mobil uygulama geliştirme sürecinin en önemli bölümlerinden birisi kullanıcıya görünen sayfa tasarımlarının işlevsel olmasıdır. Geliştirdiğiniz uygulama sadece belirli bir markanın modeline göre değilse geniş bir kullanıcı kitlesine hitap ediyorsunuz demektir. Mobil uygulamanızın farklı marka Android cihazlarda istediğiniz biçimde kullanıcıya görünmesi için tasarım ilkelerine dikkat edilmelidir. Aksi takdirde uygulama geliştirirken kullandığınız test cihazına göre her şey yolundayken farklı cihazlara uygulamanızı yüklediğinizde butonlarınız durması gereken yerden çok uzakta çıkabilir.

DENSITY-INDEPENDENT PIXELS (dp)

Mobil uygulamada kullanıcıyla etkileşime giren butonlar, checkboxlar veya text kutucukları gibi öğelerin boyutlandırılmasında ölçü birimi olarak pixel kullanılmaz. Bunun sebebi mobil uygulamanın çalışacağı farklı model Android cihazlarının farklı ekran çözünürlüğüne sahip olmasıdır.



Yukarıdaki görselde her bir kırmızı kutuyu pixel olarak düşünün. Yüksek çözünürlüklü ekranda çember içinde sığan kırmızı kutu sayısı, düşük çözünürlüklü ekrana göre daha fazladır. Yüksek çözünürlüklü ekranlarda inç başına daha fazla pixel yer alırken çözünürlük azaldıkça inç (2,54cm) başına düşen pixel sayısı da azalacaktır. Buna bağlı olarak 20 pixel genişliğindeki butonun görünümü düşük çözünürlüğe sahip telefonda daha büyük olurken, yüksek çözünürlüklü ekrana sahip telefonda ise daha küçük duracaktır.



Ekrana 4px*4px ölçülerinde bir buton yerleştirildiğinde, butonu boyutları aynı olmasına rağmen çözünürlüğe göre farklı görünümüler oluşacaktır. Bu durum tasarımcıların istedikleri bir durum değildir. Bu problemi gidermek için nesnelerin boyutlandırılmasında pixel yerine dp (**Yoğunluktan bağımsız pikseller**) değeri kullanılır. dp değeri ekran çözünürlüğüne göre değişken bir değer olduğundan 20 dp genişliğindeki bir butonun görünümü farklı marka cihazlarda ekrana oranla aynı olacaktır. Bir başka deyişle dp değeri sabit değil, ekran çözünürlüğüne göre değişen dinamik bir değerdir. Dp değeri aşağıdaki formülle hesaplanır:

$dp = (\text{piksel cinsinden genişlik} * 160) / \text{ekran yoğunluğu}$

Ekran fiziksel genişliği	Ekran yoğunluğu	Piksel cinsinden ekran genişliği	Dps cinsinden ekran genişliği
1.5 inç	120	180 piksel	240 dp
1.5 inç	160	240 piksel	
1.5 inç	240	360 piksel	

Dp değeri ile boyutlandırılan nesneler farklı marka cihazlarda tasarımcının istediği boyutlarda görüntülenecektir.

SCALABLE PIXELS (sp)

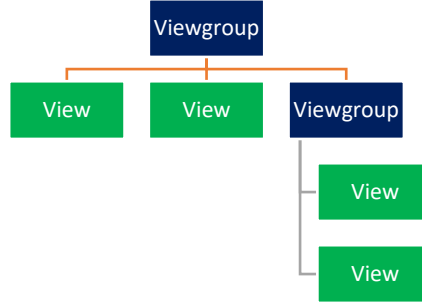
dp değeri nesnelerin boyutlandırılmasında esneklik ve ekrana göre ölçeklenebilirlik sağlarken ekrandaki yazıların da aynı nesnelerde olduğu gibi ekrana göre ölçeklenmesi gerekir. Metinlerin büyüklüğünü pixel olarak belirtmek aynı nesnelerdeki gibi problemlere yol açacaktır. Bu sebeple metinlerin büyüklüğü pixel yerine sp (ölçeklenebilir pixel) değeri ile belirtilir.

VIEW

Ekranda yer alan ve kullanıcıyla etkileşim sağlayan nesneler view olarak isimlendirilir. Kayıt olma işlemi için tıklanılan buton, cinsiyet seçimi için işaretlenen radiobutton veya isminizi yazdığınız metin kutusu birer view nesnesidir.

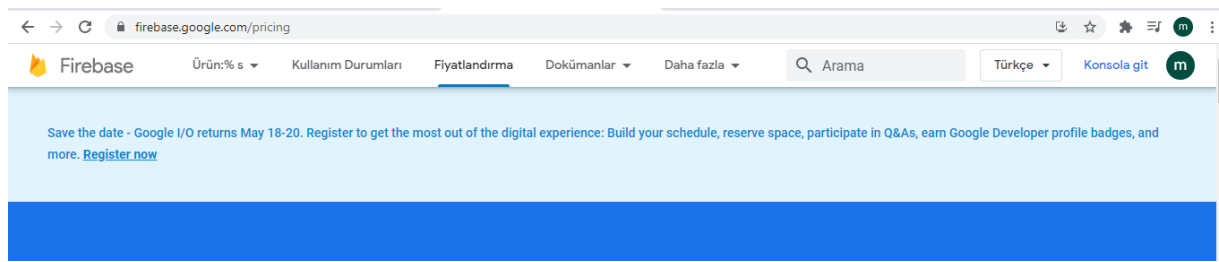
VIEWGROUP

Mobil uygulama ekranında birden fazla view yer alabilir. Bu noktada birden fazla view nesnesini bir arada tutan yapılara ViewGroup denilir. Layoutlar uygulama tasarımında sıkça kullanılan ViewGrouplardır. ViewGroup içerisinde viewler olduğu gibi ViewGrouplarda olabilir.

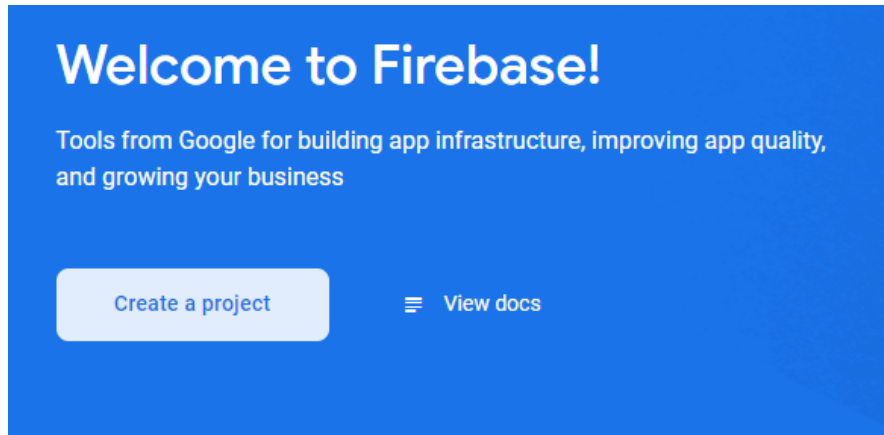


Firestore İşlemleri

Firestore projesi oluşturmak için firebase.google.com adresine Google hesabı ile giriş yaptıktan sonra aşağıdaki görselde görülen Konsola git butonuna tıklanır.



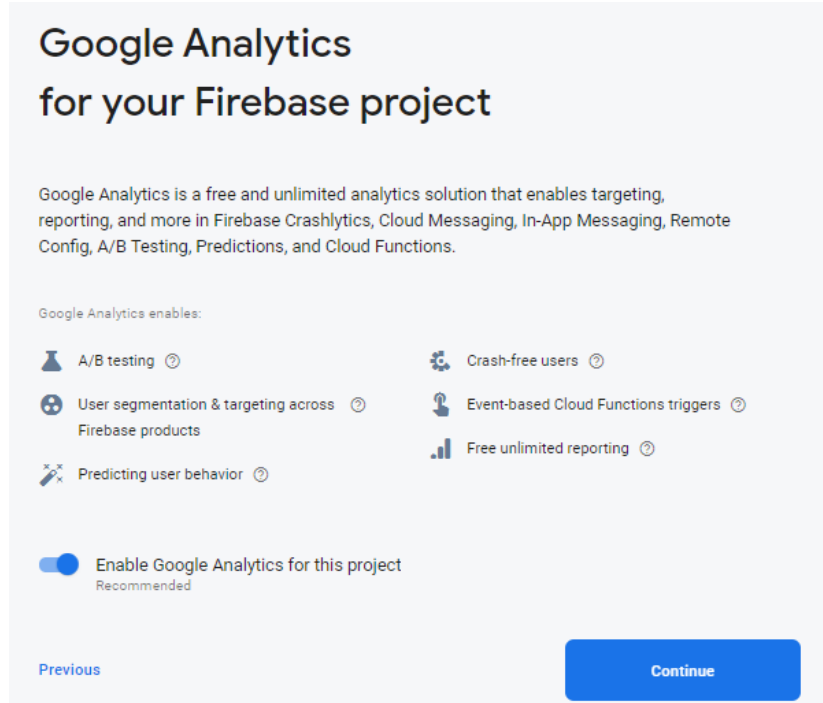
Yeni bir proje oluşturmak için Create a Project butonuna tıklanır. Açılan sayfada proje bilgilerinin girileceği 3 aşamalı bir ekran gelecektir.



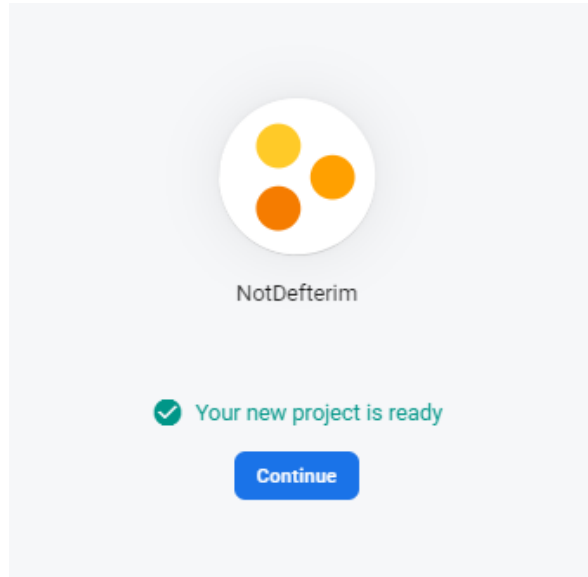
İlk olarak proje adı girilecek ekran gelir. Proje ismi en az 4 karakterden oluşmalıdır. Proje ismi Project Name bölümüne yazılır ve ekranın altında bulunan Continue butonuna tıklanarak bir sonraki aşamaya geçilir.



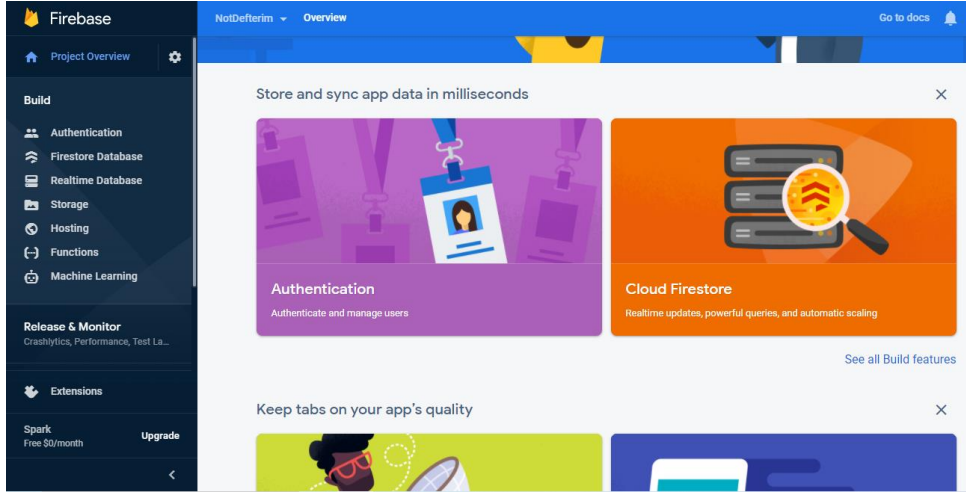
Bu aşamada projenize Google analytic özelliğini dahil edip etmeyeceğinizi seçmeniz istenir. Google Analytic uygulamanız hakkında raporlama yapmanıza imkan tanıyan ücretsiz bir özelliktir. Google Anaystic özelliğini eklemek isterseniz Enable Google Analytics for this Project seçeneğini seçmeniz gerekmektedir. İşleme devam etmek için Continue butonuna tıklanır.



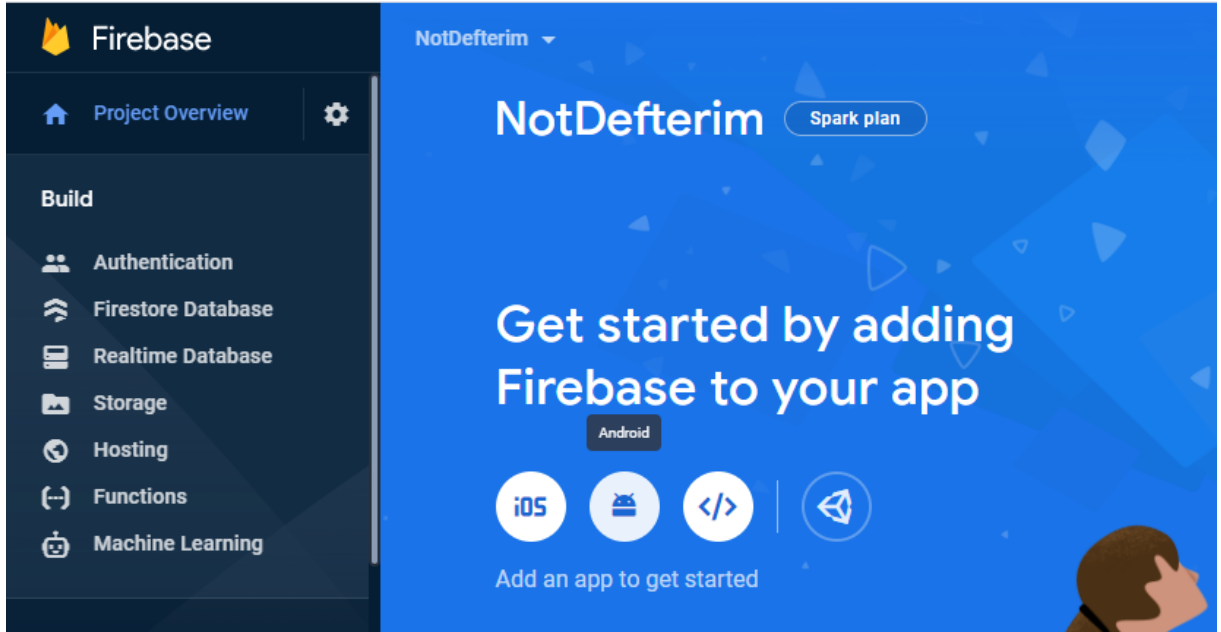
Create Project butonuna basılarak proje oluşturulur. Your new Project is ready mesajı, projenin başarıyla oluşturulduğunu gösterir.



Continue butonuna basılarak projenin yönetim paneline geçiş yapılır.



Firebase projesinin hangi platform için kullanılacağı sayfanın en üstündeki iconlardan seçilir. Android uygulaması yaptığımız için burada Android simgesine basılması gerekir.



Android simgesine basıldığında bizden Android projemize ait bir takım bilgiler istenecektir. Bu bilgileri Android Studio'da bulunan projemizden alacağız. İlk olarak Android projemizi Firebase üzerine kaydedebilmemiz için Package name ve nickname bilgisi girilir ve register butonuna basılır.

Project-level build.gradle (<project>/build.gradle):

```
buildscript {
    repositories {
        // Check that you have the following line (if not, add it):
        google() // Google's Maven repository
    }
    dependencies {
        ...
        // Add this line
        classpath 'com.google.gms:google-services:4.3.5'
    }
}

allprojects {
    ...
    repositories {
        // Check that you have the following line (if not, add it):
        google() // Google's Maven repository
        ...
    }
}
```

App-level build.gradle (<project>/<app-module>/build.gradle):

```
apply plugin: 'com.android.application'
// Add this line
apply plugin: 'com.google.gms.google-services'

dependencies {
    // Import the Firebase BoM
    implementation platform('com.google.firebase:firebase-bom:27.1.0')

    // Add the dependencies for the desired Firebase products
    // https://firebase.google.com/docs/android/setup#available-libraries
}
```

Gradle dosyasına yapılan eklemelerden sonra ekranın üstünde çıkan uyarı satırında yer alan Sync Now linkine tıklanarak senkronizasyon işlemi yapılır ve süreç tamamlanır. Next butonuna basılarak Firebase Console a geri dönülür.

[Sync Now](#)

Firebase Authentication

Kullanıcıların kimlik doğrulama işlemlerinin Firebase sunucusu üzerinden yapılabilmesini sağlayan modüldür. Bu modülü kullanabilmek için ilgili modülün kütüphanesi Android projemize eklenmelidir. Bunun için build.gradle dosyasındaki dependencies bölümüne aşağıdaki kod satırı eklenir.

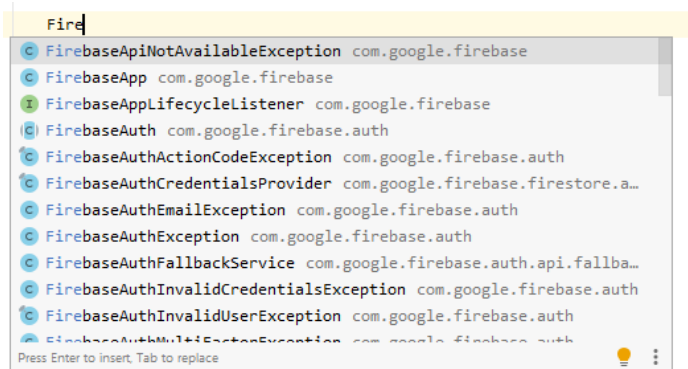
implementation 'com.google.firebase:firebase-auth'

```
dependencies {

    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'com.google.android.material:material:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
    implementation platform('com.google.firebase:firebase-bom:27.1.0')
    implementation 'com.google.firebase:firebase-auth'
    implementation 'com.google.firebase:firebase-firestore'
    implementation 'com.google.firebase:firebase-storage'
    implementation 'androidx.recyclerview:recyclerview:1.0.0'
    implementation 'com.squareup.picasso:picasso:2.71828'

}
```

Kütüphane ekleme işlemleri başarılı olduysa Firebase komutları aşağıdaki şekilde kullanılabilir durumda olmalıdır.



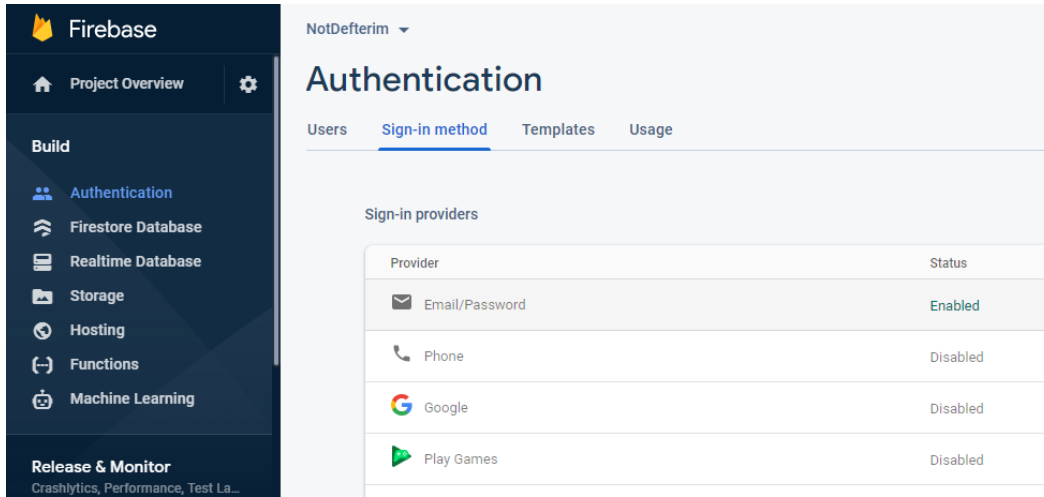
The screenshot shows a search bar with the text 'Fire'. Below it, a list of search results is displayed, including:

- FirestoreApiNotAvailableException com.google.firebase
- FirestoreApp com.google.firebase
- FirestoreAppLifecycleListener com.google.firebase
- FirestoreAuth com.google.firebase.auth
- FirestoreAuthActionCodeException com.google.firebase.auth
- FirestoreAuthCredentialsProvider com.google.firebase.firestore.a...
- FirestoreAuthEmailException com.google.firebase.auth
- FirestoreAuthException com.google.firebase.auth
- FirestoreAuthFallbackService com.google.firebase.auth.api.fallba...
- FirestoreAuthInvalidCredentialsException com.google.firebase.auth
- FirestoreAuthInvalidUserException com.google.firebase.auth
- FirestoreAuthMultiFactorException com.google.firebase.auth

Firebase Authentication yapısını kullanmak için ilk olarak FirebaseAuth.getInstance(); komutu ile FirebaseAuth nesnesi (instance) oluşturulur.

FirebaseAuth mAuth=FirebaseAuth.getInstance();

Oluşturduğumuz mAuth nesnesi sayesinde Firebase Authentication yapısına ait tüm özellikleri kullanabilir duruma geliyoruz. İlk olarak e-posta adres ve bir parola kullanarak yeni kullanıcı hesabı oluşturalım. Kod bölümüne geçmeden önce ilk olarak Firebase Console üzerinden kullanıcı hesabı oluşturma seçeneği olarak Email/Password aktif hale getirilmelidir.



Firebase ile yeni bir kullanıcı kaydı oluşturmak için oluşturduğumuz FirebaseAuth sınıfına ait `createUserWithEmailAndPassword` metodunu kullanabiliriz. Bu metod yeni kullanıcı kaydı için e-posta adresi ve parola olmak üzere iki parametre alır. İşlemin başarılı ya da başarısız olma durumlarını tespit edip bu durumlara göre farklı aksiyonlar göstermek için listenerlar kullanılır. `AddonSuccessListener` ile kayıt işleminin başarılı olma durumu tespit edilirken, işlemin herhangi bir sebeple başarısızlıkla sonuçlanması durumu `addOnFailureListener` ile tespit edilir.

```
mAuth.createUserWithEmailAndPassword(email,sifre).addOnSuccessListener(new OnSuccessListener<AuthResult>() {  
    @Override  
    public void onSuccess(AuthResult authResult) {  
        Toast.makeText( context: MainActivity.this, text: "Kayıt işlemi başarılı. :", Toast.LENGTH_SHORT).show();  
        Intent intent=new Intent( packageContext MainActivity.this,UserActivity.class);  
        startActivity(intent);  
        finish();  
    }  
}).addOnFailureListener(new OnFailureListener() {  
    @Override  
    public void onFailure(@NonNull Exception e) {  
        Toast.makeText( context: MainActivity.this, text: "İşlem başarısız. :("+" "+e.getMessage().toString(),  
            Toast.LENGTH_LONG).show();  
    }  
});
```

Kayıtlı bir kullanıcının uygulamamız üzerinde eposta adresi ve parolası ile oturum açabilme işlemin yapmak için FirebaseAuth sınıfının `signInWithEmailAndPassword` metodu kullanılır. Bu metoda ulaşabilmek için ilk adımda oluşturduğumuz FirebaseAuth nesnesini kullanacağız. Bu metod eposta adresi ve parola olmak üzere iki parametre alır. Kullanıcıdan aldığımız eposta adresi ve parola bilgisinin doğruluk firebase sunucusu üzerine kontrol edilir. Kullanıcının oturum açma işleminin başarılı ya da başarısız olarak sonuçlanması durumları bir önceki örnekte olduğu gibi burada da listenerlar ile takip edilir.

```

mAuth.signInWithEmailAndPassword(email,sifre).addOnSuccessListener(new OnSuccessListener<AuthResult>() {
    @Override
    public void onSuccess(AuthResult authResult) {

        Intent intent=new Intent( packageContext: MainActivity.this,UserActivity.class);
        startActivity(intent);
        finish();
    }
}).addOnFailureListener(new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception e) {
        Toast.makeText( context: MainActivity.this, text: "Olmadı :("+" "+e.getLocalizedMessage().toString(),
            Toast.LENGTH_LONG).show();
    }
});

```

Firebase Storage

Firebase sunucuları üzerinde resim, ses, video gibi dosyaların depolanabilmesi için Firebase Storage uygulaması kullanılır. Storage özelliğini kullanabilmek için ilk olarak kütüphanenin yüklenmesi amacıyla Build.Gradle dosyasına aşağıdaki kod satırının eklenmesi gerekir.

implementation 'com.google.firebase:firebase-storage'

FirebaseStorage sınıfına ait özellikleri kullanabilmek için FirebaseStorage sınıfından firebaseStorage=FirebaseStorage.getInstance(); komutu kullanılarak bir nesne türetilir. Kullanıcının seçmiş olduğu dosyayı storege alanına yüklemek için putFile metodu kullanılır. Bu metot seçilen dosyanın yol bilgisini içeren Uri formatında bir parametre alır. Metodu ise storageReference ile nesnesi ile kullanılır. Reference yükleme konumunu gösteren bir özellik olarak görev yapar. Yükleme işleminin başarılı ya da başarısız olma durumları diğer örneklerde olduğu gibi listenerlar kullanılarak tespit edilir.

```

String gorselAdi="images/"+uuid+".jpg";

storageReference.child(gorselAdi).putFile(secilenData).addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
    // ...
}

```

İzin İşlemleri

Bir mobil uygulamanın kullanıcı cihazı üzerinde dosya alanına erişmek, internet bağlantısını kullanmak, gelen SMS'lere erişmek gibi işlemleri yapabilmesi için kullanıcının bu işlemler için uygulamaya izin vermesi gerekir. Bu sebeple bir mobil cihazın galerisindeki resimleri kullanıcın seçim yapması amacıyla açabilmek için ilk olarak dosya alanına erişim izninin olup olmadığını kontrol etmek gerekir. Eğer izin varsa işleme devam edilirken izin yoksa bu işlemi gerçekleştirmek için kullanıcının izni talep edilir.

```

public void gorselSec(View view) {

    if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.READ_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions( activity: this, new String[]{Manifest.permission.READ_EXTERNAL_STORAGE}, requestCode: 1);
    } else {
        Intent intenttoGallery = new Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
        startActivityForResult(intenttoGallery, requestCode: 2);
    }
}

```

Yukardaki görsel metodunda ilk olarak dosya alanına erişim izin daha önce kullanıcıdan izin alıp alınmadığı if bloğunda sorgulanmaktadır. İzinler Manifest dosyası içerisinde yer alır. PackageManager.PERMISSION_GRANTED komutu izin olduğu anlamına gelmektedir. Eğer kullanıcı izni yoksa if şartı sağlanmış olacak ve if bloğu içerisindeki ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.READ_EXTERNAL_STORAGE}, 1); komutu çalışacaktır. Bu komut ile kullanıcıdan Manifest. Permission. READ_EXTERNAL_STORAGE izni istenmektedir. Kod satırı sonundaki 1 ise requestCode numarasıdır. Kullanıcıdan 1 den fazla izin talep edilebilir. Bu durumda her bir izin için farklı bir numara verilir ki bu numara izinleri birbirinden ayırt etmek için sonraki aşamalarda kullanılır. Eğer kullanıcı daha önceden bu izni veriyse o zaman else bloğu çalışır ve kullanıcı resim seçmek için galeriye geçiş yapar.

Kullanıcıdan istediğimiz izinlere yönelik kullanıcının vermiş olduğu yanıtları onRequestPermissionsResult metodu içerisinde takip edebiliriz. Aşağıdaki komut içerisinde kullanıcıdan 1 numarası ile istediğimiz dosya alanına erişim izninin sonucunu işleme alıyoruz.

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {

    if(requestCode==1){
        if(grantResults.length >0 && grantResults[0]==PackageManager.PERMISSION_GRANTED){
            Intent intenttoGallerri = new Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
            startActivityForResult(intenttoGallerri, requestCode: 2);
        }
    }

    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
}
```

requestCode==1 komutu ile baktığımız iznin 1 numarası ile talep ettiğimiz READ_EXTERNAL_STORAGE izni olduğunu kontrol ediyoruz. İf satırı içerisindeki ilk şartımız grantResults.length >0 komutu ile gösterilmiştir. Bu komut ile kullanıcının izin talebimize bir yanıt verdiğini kontrol ediyoruz. İzin talepleri ve bu izinlere verilen cevaplar bir dizi içerisinde yer alır. Bu sebeple length özelliği ile dizinin içerisinde bir yanıt olması durumunda dizinin eleman sayısı 0 dan büyük olacaktır. İkinci şartımız ise gelen cevabın izin verilmesi olduğunu kontrol eden grantResults[0]==PackageManager.PERMISSION_GRANTED satırıdır. Eğer kullanıcı 1 nolu izin talebimize bir cevap veriyse ve cevabı izin verilmesi şeklindeyse o zaman izin alınmış demektir ve galeri açılabilir.

Firestore

Firestore Firebase tarafından yayınlanan en yeni özelliklere sahip gerçek zamanlı bir veri tabanı özelliğidir. Bu veri tabanı içerisinde bilgiler bir ağaç yapısı şeklinde kaydedilir. Kök bölümüne collection adı verilir. Kayıtlar doküman biçiminde collection içerisine kaydedilir. Doküman içindeki kaydı oluşturan bilgiler ise field olarak adlandırılır. Örnek bir yapı aşağıdaki şekilde gösterilmektedir.

🏠 > Notlar > 1Of0HRf8E7ZOz...		
notdefterim-c2b0a	Notlar	1Of0HRf8E7ZOzYEsWa7f
+ Start collection	+ Add document	+ Start collection
Notlar >	1Of0HRf8E7ZOzYEsWa7f >	+ Add field
	4rVZH2gZxFvwM7d8BHWa THqMJkYNEgU6ai5WYqxd jkjQvKoYzcgKGtLjcxcl	gorselyolu: "https://firebasestorage.googleapis.com/v0/b/notdef c2b0a.appspot.com/o/images%2F90d09fd1-44f7-45 b2f7-9f93fcb64695.jpg?alt=media&token=afe44df9-9 4eb2-85d6-ea6e23d31b71" kullaniciposta: "biltek2015@gmail.com" not: "testt" tarih: May 10, 2021 at 1:07:46 PM UTC+3

Bir kaydın veri tabanına kaydedilmesi için ilk olarak bir FirebaseFirestore nesnesinin oluşturulması gerekir. `firebaseFirestore=FirebaseFirestore.getInstance();` komutu ile bu nesne oluşturulur. Kayıt eklemek için bilgiler field alanına bir isim ve bu isme ait bir değerle kaydedilmesi için bilgiler ilk olarak HashMap yapısına dönüştürülür. HashMap formatındaki bilgiler firebaseFirestore nesnesi içinde collection metodu ile yolunu öğrenir add metodu ile eklenir. Aşağıdaki kod bloğu bu duruma örnektir.

```
HashMap<String,Object> kayitData=new HashMap<>();
kayitData.put("kullaniciposta",userMail);
kayitData.put("gorselyolu",downloadUri);
kayitData.put("not",yorum);
kayitData.put("tarih", FieldValue.serverTimestamp());

firebaseFirestore.collection( collectionPath: "Notlar").add(kayitData).addOnSuccessListener(new OnSuccessListener<DocumentReference>() {
    @Override
    public void onSuccess(DocumentReference documentReference) {

        Intent intent=new Intent( packageContext: YuklemeActivity.this,UserActivity.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(intent);
    }
}).addOnFailureListener(new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception e) {
        Toast.makeText( context: YuklemeActivity.this, text: "Olmadı :(" +e.getMessage().toString(), Toast.LENGTH_SHORT).show();
    }
});
```

Kaynaklar:

http://www.megep.meb.gov.tr/mte_program_modul/moduller/Mobil%20Uygulamaya%20Giri%C5%9F.pdf

<https://wearesocial.com/digital-2020>

<https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/>

<https://www.statista.com/statistics/289418/number-of-available-apps-in-the-google-play-store-quarter/>

<http://www.openhandsetalliance.com/>

<https://www.mobiler.dev/post/ipc-yontemleri-1-aidl>

<https://source.android.com/devices/architecture>

<https://developer.android.com/studio/releases#4-1-0>

<https://www.w3schools.com/java/>

<https://stackoverflow.com/>

<https://developer.android.com/>

<https://material.io/design/layout/>

<https://atilsamancioglu.com/>