

YENİ BAŞLAYANLAR İÇİN

KOTLIN İLE ANDROID PROGRAMLAMA

CEREN AKMAN

Ceren AKMAN

twitter.com/akmncrn



linkedin.com/in/akmanceren1/



instagram.com/ceren.akman/



github.com/akmancrn



akmancrn@gmail.com



Kitaba dair örnek uygulama ve kaynak kodları aşağıdaki linkten indirebilirsiniz.

<https://github.com/akmancrn>

İÇİNDEKİLER

BÖLÜM 1: KOTLIN'E GİRİŞ	1
Kotlin Nedir?	2
Kotlin Neden Geliştirildi?	2
Kotlin Neden Tercih Edilmelidir?	3
Desteklediği Platformlar Nelerdir?	4
Server Tarafı Uygulamalar	4
Android Uygulamalar	5
Kotlin için JavaScript	6
Kotlin için Native Yapı	7
Multi Platform	7
Neler Öğrendik?	9
BÖLÜM 2: ANDROID STUDIO VE SANAL CİHAZ KURULUMLARI	11
Android	12
JDK Nedir?	13
JDK Kurulum	13
Android Studio Nedir?	15
Android Studio Kurulumu	16
SDK Nedir?	20
SDK Kurulumu	20
Emülatör Kurulumu	22
Android Studio Kendi Emülatör Kurulumu	22
Genymotion Kurulumu	24
Neler Öğrendik?	29

BÖLÜM 3: KOTLIN TEMEL VERİ TİPLERİ **31**

Değişken Nedir?	33
Değişken Tanımlama	34
Variable (var) ve Value (val) Nedir ve Nasıl Kullanılır?	35
val ve var Anahtar Kelimeleri Arasındaki Fark	37
Kotlin Temel Veri Tipleri	38
Veri Tipi Nedir ve Ne İşe Yarar?	38
Sayı Veri Tipleri	38
String Veri Tipi	42
String'leri Birleştirme	44
String Metotları	46
Metot Nedir?	46
Char Veri Tipi	53
Boolean Veri Tipi	54
Kullanıcıdan Değer Alma	55
readLine() Metodu Kullanımı	55
scanner Sınıfı Kullanımı	57
Tip Dönüşümü	60
Neler Öğrendik?	63

BÖLÜM 4: NULL KULLANIMI **65**

Null Pointer Exception Hatası	66
Null-Null Safety	69
Non-Null Type (Null Olamayan Tip)	69
Nullable Type (Null Olabilen Tip)	69
Safe Call Operatörünün Kullanımı (?)	72
elvis Operatörü(?:)	78
!! Operatörü	81

Nullability ve Koleksiyonlar	82
Null Değer Alabilen Koleksiyonlar	83
Null Olabilen Koleksiyonlar	85
Hem Null Değer Alabilen Hem Null Olabilen Koleksiyonlar	86
Null Safety ve JAVA Birlikte Çalışılabilirliği	87
Neler Öğrendik?	89
BÖLÜM 5: KOŞULLU İFADELER	91
if Koşul Yapısı	92
if-else Koşul Yapısı	95
if, else if, else Koşullu Yapısı	98
when Koşul Yapısı	102
when Bir expression Olarak Kullanılırsa	103
When Bir Statement Olarak Kullanılırsa	110
When Yapısını İçinde Argüman Olmadan Kullanma	111
When Yapısı ile Smart Cast Kullanımı	115
Neler Öğrendik?	115
BÖLÜM 6: RANGE KULLANIMI	117
Range Nedir?	118
rangeTo() Metodunun Kullanımı	121
downTo() Metodunun Kullanımı	125
step() Metodunun Kullanımı	126
until() Metodunun Kullanımı	128
reversed() Metodunun Kullanımı	128
filter() Metodunun Kullanımı	130
Diğer Metotlar	131
Neler Öğrendik?	133

BÖLÜM 7: DÖNGÜLER VE HATA AYIKLAMA	135
Döngü Nedir?	136
for Döngüsü	136
İç İçe for Döngüsü	140
while Döngüsü	144
do-while Döngüsü	149
Break, Continue ve Label Kullanımı	151
break Kullanımı	151
continue Kullanımı	152
label Kullanımı	156
try-catch Blokları	160
Neler Öğrendik?	163
BÖLÜM 8: DİZİLER	165
Dizi Nedir?	166
Dizi Tanımlama	166
Dizi Değer Atama	167
Dizi Değer Okuma	168
arrayOf Kullanımı	173
Çift Boyutlu Diziler	175
Neler Öğrendik?	178
BÖLÜM 9: FONKSİYONLAR	181
Fonksiyon Nedir?	182
Fonksiyon Nasıl Tanımlanır?	182
Fonksiyon Nasıl Çağrılır?	183
extension Fonksiyonu	192
infix Fonksiyonu	194
nested (iç içe) Fonksiyonlar	195

Parametreler	196
Default (Varsayılan) Fonksiyon Parametreleri	196
named (İsmlendirilmiş) Fonksiyon Parametreleri	199
varargs (Variable Number Of Function Parameter)	201
Lambda İfadeler	202
Lambda Nedir?	202
high-order Fonksiyon	205
Neler Öğrendik?	207

BÖLÜM 10: SINIF VE NESNE YAPISI **209**

Nesne Yönelimli Programlama Nedir?	210
Sınıf Nedir?	210
Sınıf Nasıl Oluşturulur?	210
Nesne Nedir?	212
Nokta(.) Operatörünü Neden Kullanırız?	214
Kalıtım (Inheritance) Nedir?	214
Abstract (Soyutlama) Nedir?	217
Abstract Class Nedir?	217
Interface (Arayüz) Nedir?	219
Sınıf Tipleri	222
Nested Class	222
Inner Class	223
Data Class	224
Neler Öğrendik?	225

BÖLÜM 11: CONSTRUCTOR (YAPICI) **227**

Constructor (Yapıcı) Nedir?	228
Primary Constructor (Birincil Yapıcı/Kurucu)	229
Secondary Constructor (İkincil Yapıcı/Kurucu)	232
Neler Öğrendik?	237

BÖLÜM 12: ERİŞİM BELİRLEYİCİLER (VISIBILITY MODIFIERS) 239

Erişim Belirleyiciler Nedir?	240
Public Nedir?	240
Private Nedir?	241
Protected Nedir?	243
Internal Nedir?	244
Neler Öğrendik?	245

BÖLÜM 13: LAYOUT'LAR 247

Layout Nedir?	248
Linear Layout Yapısı	251
Relative Layout Yapısı	254
FrameLayout Yapısı	255
Constraint Layout Yapısı	257
Neler Öğrendik?	259

BÖLÜM 14: BUTON KONTROLLERİ 261

Buton Kullanımı	262
Buton Stilini Özelleştirme	266
ImageButton Kullanımı	271
ToggleButton Kullanımı	274
RadioButton Kullanımı	277
CheckBox Kullanımı	281
Neler Öğrendik?	283

BÖLÜM 15: METİN KONTROLÜ 285

TextView Kullanımı	286
EditText Kullanımı	289
Neler Öğrendik?	293

BÖLÜM 16: SCROLLVIEW KULLANIMI	295
ScrollView Nedir?	296
Neler Öğrendik?	305
BÖLÜM 17: LISTVIEW KULLANIMI	307
ListView Nedir?	308
Neler Öğrendik?	311
BÖLÜM 18: İKON İŞLEMLERİ	313
İkon Nedir?	314
Adaptive İkon Nedir?	315
Neler Öğrendik?	317
BÖLÜM 19: MENÜ İŞLEMLERİ	319
Action Bar Nedir?	320
Action Bar Kullanımı	320
Pop-Up Menü Kullanımı	329
AlertDialog Kullanımı	336
Neler Öğrendik?	339
BÖLÜM 20: MEDYA İŞLEMLERİ	341
Media Player Sınıfı Nedir?	342
Nasıl Oynatılır?	342
Media Recorder Sınıfı Nedir ve Nasıl Kullanılır?	348
Neler Öğrendik?	352
BÖLÜM 21: INTENT KAVRAMI	355
Intent Nedir?	356
Sayfalar Arası Geçiş	356
Sayfalar Arası Veri Taşıma	360
Neler Öğrendik?	365

BÖLÜM 22: SPLASHSCREEN VE ANIMAYON OLUŞTURMA 367

SplashScreen Nedir?	368
Animasyon Oluşturma	372
Neler Öğrendik?	375

BÖLÜM 23: VERİTABANI İŞLEMLER 377

Veritabanı Nedir?	378
Veritabanı Ne İşe Yarar?	378
SQLite Nedir?	379
SQLite Neden Tercih Edilmelidir?	379
SQLite Veritabanı İşlemleri (CRUD)	379
Neler Öğrendik?	393

BÖLÜM 24: REKLAM EKLEME 395

Banner Reklam	396
Interstitial (Geçişli) Reklam	398
admob Hesap Açma	399
Neler Öğrendik?	401

BÖLÜM 25: GOOGLE PLAY'DE UYGULAMA YAYINLAMA 403

Ön Hazırlık	404
İmzalama (Sign) ve Derleme (Build)	405
Yayınlama (Publish)	407
Neler Öğrendik?	409
Son Söz	410
Dizin	411

1

KOTLIN'E GİRİŞ

BU BÖLÜMDE

Kotlin Nedir?	2
Kotlin Neden Geliştirildi?	2
Kotlin Neden Tercih Edilmelidir?	3
Desteklediği Platformlar Nelerdir?	4
Neler Öğrendik?	9

Bu bölümde, Kotlin dilinin ne olduğu, kim tarafından neden ve hangi amaçla geliştirildiği, desteklediği platformları ve avantaj, dezavantajlarını öğreneceğiz.

KOTLIN NEDİR?

Kotlin, IntelliJ IDEA'nın geliştiricisi Rusya merkezli bir şirket olan **JetBrains** firması tarafından 2011 yılında geliştirilmeye başlanıldığı duyuruldu ve 2016 senesinde ilk sürümünü yayınladı. Daha sonra Google ile birlikte IntelliJ IDEA tabanlı Android Studio'yu geliştirdiler ve Android'in resmi geliştirici IDE'si oldu.

Kotlin, JAVA ve C++'dan sonra üçüncü resmi Android geliştirici dil olduğu Google tarafından 2017 senesinde duyurulmasından sonra çok daha fazla ilgi görmeye başladı. Kotlin dili JVM (JAVA Virtual Machine) üzerinde çalışabilen, JavaScript ve Native uygulamaları geliştirmek için oluşturulan statik, ücretsiz ve açık kaynak koda sahip aynı zamanda JAVA kütüphanelerini kullanabilen bir programlama dilidir. Kotlin dili Apple'ın kullandığı Swift diline benzemektedir ve bu dille masaüstü, web ve mobil uygulamalar geliştirebilirsiniz tabi bunları yapabilmek için Kotlin temellerini öğrenmeniz gerekli Kotlin temellerini ilerleyen bölümlerde işleyeceğiz.

KOTLIN NEDEN GELİŞTİRİLDİ?

Birçok kişinin düşündüğü gibi sizler de düşünebilirsiniz JAVA varken neden Kotlin geliştirildi? Zaten Android programlamada kullandığımız JAVA varken Kotlin'e ne gerek vardı? gibi birçok sorular soruldu ve bu soruların cevabı olarak geliştirici firma olan **JetBrains** "*Performans ve Güvenliği Feda Etmeden JAVA'dan daha özlü kodlar yazmayı sağlamak için bu dili geliştirdik*" cevabını verdi.

Kotlin, JAVA dilinden daha iyi bir dil olarak tasarlanmıştır ama bu demek değildir ki JAVA'yı ortadan kaldıracak. Zaten hali hazırda böyle bir durum söz konusu olamaz. Uydu cihazları, akıllı telefonlar, drone'lar akıllı televizyonlar yani akıllı olarak tasarlanmış olan şeyler JAVA programlama dili ile yazılmıştır. İş böyle olunca JAVA'nın ortadan kalkması gibi bir düşünce olamaz. Tabi her ne kadar JAVA dili şuan egemen olsa da yavaş yavaş firmalar Kotlin'e geçişlerini gerçekleştirmeye başlamışlardır. Bu geçişin gerçekleşmesinin sebebi ise Kotlin dilinin bize kazandırdığı birçok avantajın söz konusu olmasıdır.

Örneğin; İnsanlar arası kullanılan iletişim diline daha yakın olduğu için, geliştiricilerin işini kolaylaştırmış olması, sade, pratik, güvenliği sağlıyor olması vs. gibi özellikler Kotlin'in tercih edilmesine vesile oluyor. Yani kısaca şöyle diyebiliriz Kotlin programlama dilinin geliştirilmesindeki asıl amacın JAVA'ya göre kod satırlarını azaltmak ve daha güvenli kod yazmayı amaçlandığı söylenebilir.

KOTLIN NEDEN TERCİH EDİLMELİDİR?

Kotlin programlama dilinde geliştirilen bütün uygulamalar tüm Android cihazlarda yani daha eski cihazlarda da çalıştırılmaya uygundur. Kotlin nesne yönelimli statik tipli bir programlama dilidir. Yani Kotlin programlama dilinde yazılmış bir kodda derleme esnasında tip kontrolü yapıldığı için program çalıştığı zaman tip kontrolü yapılmaz. Bu da uygulamaların daha hızlı ve verimli çalışmasını sağlar. Aynı zamanda Kotlin programlama dili uygulamaların ara yüz görünümünü de iyi yönde geliştirilmesini sağlar çünkü programcılara daha hızlı çalışma fırsatı sunan bir dil yapısı olduğu için programcılara ara yüz geliştirmek için ekstra zaman kalacaktır.

Kotlin dilinin diğer bir iyi tarafı ise JAVA'ya oranla aynı işi daha kısa kod blokları ile yapma daha sade ve anlaşılır bir dil yapısına sahip olması ve bunların dışında kendine özgü yapısı geliştiricileri kendine çeker. Kotlin dilinde;

- » Type Inference
- » Extension Methods
- » Lambda Function
- » Data Class
- » Operator Overloading

özellikleri mevcuttur. Bu özelliklerin ne işe yaradığını nasıl kullanılacağını ilerleyen bölümlerde işleyeceğiz.

Kotlin dili bizler için daha fazla güvenilirlik sunar. Bu güvenilirlik şu anlama gelmektedir. Nesne tabanlı programlamalarda **null** hatası yüksek maliyetli zararlara yol açabilir. Ama Kotlin dilinde **nullable** ve **non-nullable type** sistemin birer parçasıdır. İşler böyle olunca Kotlin de compiler **null** olabilecek referanslar ile asla **null** olamayacak referansları ayırt edebilir.

Şuanda da **NETFLIX**, **Trello**, **Pinterest**, **Corla** vb. gibi pek çok uygulama Kotlin programlama dili geliştirilmiştir.

NOT

Null: Türkçe karşılığı **YOK** anlamına gelmektedir. Programlama jargonunda ise; genellikle program tarafından bulunamayan bilgileri simgeler.

4

NULL KULLANIMI

BU BÖLÜMDE

Null Pointer Exception Hatası	66
Null–Null Safety	69
Nullability ve Koleksiyonlar	82
Null Safety ve JAVA Birlikte	
Çalışılabilirliği	87
Neler Öğrendik?	89

Bu bölümde Null yapısının ne olduğunu, Null kullanımından dolayı çıkan hatalar ve bu hataların Kotlin dili ile nasıl engellendiğini işleyeceğiz.

Öncelikle Null nedir?

Null, Türkçe karşılığı **boş, yok** anlamında çevirebiliriz. Bu yapının bilgisayardaki karşılığı da pek farklı değildir. Null anahtar kelimesi boş özellik demektir. Yani değışkene veya nesneye ilk değer atanmamıştır.

Bu tür durumlar JAVA'da dahil olmak üzere bir çok programlama dilinde hatalara yol açar çünkü değeri olmayan bir değışkenin ya da nesnenin işlem yapması beklenemez. Ama olası bir durumda böyle bir istek gerçekleşirse hatalara yol açacaktır ve projenin kapsamına göre yüksek maliyetlere neden olabilir. Bu yüzden Kotlin dilinde amaç bu durumu yok etmektir ve bunun içinde önlemler alınmıştır. Şimdi null yapısının yanlış kullanımında ortaya çıkabilecek hataları ve Kotlin dilinde bu hataların nasıl önlendiğine bakalım.

NULL POINTER EXCEPTION HATASI

Tabi bu yapıdan kaynaklı en yaygın hata **Null Pointer Exception** hatasıdır. Bu hatayı şöyle düşünebiliriz; bir sınıf düşünün bu sınıfın nesnesini üretiyoruz ve daha sonra bazı sebeplerden dolayı bu nesne ile olan bağımız kopuyor, siliyoruz vb. durumlar yaşıyoruz sonra biz bu nesneye erişmeye çalışıyoruz ya da nesneye eleman yüklemeye çalışıyoruz o zaman bu hatayı alırız. Biliyorsunuz bizim ürettiğimiz ya da yazdığımız her nesne her değışken bellekte yer tutmaktadır. Eğer bellekte olmayan bir nesne ya da değışkene erişmeye çalışırsak işte bu hatayı alırız.

Neden kendi yazdığım kodda olmayan bir değışkene erişmeye çalışayım ya da neden ürettiğim nesne ile olan bağımı koparayım veya bağımı kopardığım nesneye erişmeye çalışayım diye düşünebilirsiniz. Ama işler öyle yürümüyor maalesef birçok şey neden olabilir bu hatayı almamıza örneğin;

- » Yazılımcıdan kaynaklı bir hata olabilir.
- » Server'dan kaynaklı bir hata olabilir.
- » Program içinde kullanılan kütüphanelerden dolayı da olabilir.
- » Programın birden fazla iş yaptığını düşünelim (multitasking) ve yapılması gereken görevin gerçekleşmemiş olması da neden olur. (Multitasking: Aynı anda birden fazla iş yapmak demektir. Örneğin Android Studio da kod yazarken bir yandan da internette müzik dinlemek gibi...)

Özellikle JAVA ile çalışılırsa bu hatayı runtime (çalışma) zamanında alırız ve bu hatanın nerede olduğunu bulmak hem zaman kaybına yol açar aynı zamanda yüksek maliyette zararlara dahi dönüşebilir.

NOT

Yazdığınız kodları on ya da on beş satırlık kod olarak düşünmeyin bazen bin satırdan fazla projeler yazıyoruz.

Kotlin dili burada devreye giriyor ve avantajını gözler önüne seriyor. Peki, nedir bu avantaj JAVA dilinde bu hatayı runtime'da (çalışma zamanı) alıyorduk ama Kotlin'de compiler time'da (derleme zamanı) alıyoruz ve bu şekilde bu hatanın önüne geçiyoruz. Peki, çalışma zamanı ve derleme zamanı arasındaki fark nedir? diye düşünebilirsiniz.

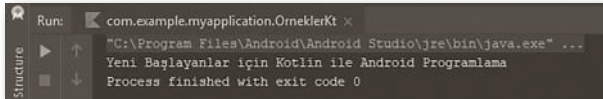
Bir hatayı çalışma zamanında almak demek son parantezide yazıp kodu çalıştırdığınız takdirde yanlış çalışan bir uygulama ya da hiç çalışmadan hata veren bir uygulama elde etmek demektir. Bu proje kapsamlı ve uzun bir proje ise hatanın nereden kaynaklandığını bulmak için epeyce çaba sarf etmeniz gerekir ki bu da pek hoş olmaz. Derleme zamanında hata almak demek hatalı kod satırını yazdığınız zaman anında hatalı yazdığınıza dair bir bildirim alırsınız. Bu da daha fazla ilerleme yapmadan hatayı çözmemizi sağlar.

Örneğin; Bir String değişken tanımlayalım daha sonra bu değişkene önce bir değer verelim yazdıralım daha sonra bu değişkene Null değeri atayalım o zaman **Null can not be a value of a non-null type String** hatasını derleme zamanında alacağız ve çalıştırmayacağız.

Örnek

Açtığımız kod sayfasına aşağıdaki kod parçasını yazdığımızda ekrana str'nin değerini yazar.

```
fun main (args : Array<String>){  
    var str : String = "Yeni Başlayanlar için Kotlin ile Android Programlama "  
    print(str)  
}
```



Yukarıda ki kod parçasında normal bir şekilde String değişkenimizi tanımladık ve değer atayıp ekrana yazdırdık ve yukardaki resim gibi bir çıktı elde ettik. Şimdi bir null değer vererek yapalım.

6

RANGE KULLANIMI

BU BÖLÜMDE

Range Nedir?	118
Diğer Metotlar	131
Neler Öğrendik?	133

Bu bölümde, Ranges yapısının ne olduğunu ne anlama geldiğini nasıl ve nerelerde kullanıldığını ve ne gibi özel metotlara sahip olduğunu işleyeceğiz.

RANGE NEDİR?

Range Türkçe karşılığı **Aralık** anlamına gelmektedir. Kotlin dilindeki anlamı da bundan farklı değildir **Aralık** anlamına gelmektedir. Bir başlangıç ve son değerin olduğu bir aralık `..` (operatörü) kullanarak tanımlanır. Bu yapı daha çok döngülerde kullanılır. Nedenini daha detaylı olarak döngüler bölümünde göreceğiz ama yinede değinelim. Döngüler ile yapılan işlemler tekrarlı işlemler olduğu için belirli bir değeri baz alarak isteğe göre bu değer üzerinde artırma veya azaltma işlemleri yapılır. Ve bu işlemleri kontrol altında tutmak için **range** yapısı ile belirlediğimiz bir aralık olmalıdır. Daha sonra bu değer range yapısı ile belirlediğimiz aralıktaki ise döngü içinde yer alan işlemler yapılır. Eğer değer aralıktaki değilse o zaman döngü içindeki işlem yapılmaz. Bu değer aralıktaki olup olmadığının kontrolünü ise `in` anahtar kelimesi ile yaparız.

Ayrıca range yapısı ile belirlediğimiz aralığı sayılar (`int`) ile yapabildiğimiz gibi karakterler (`char`) ile de yapabiliriz.

Şimdi örnekler üzerinde inceleme yapalım.

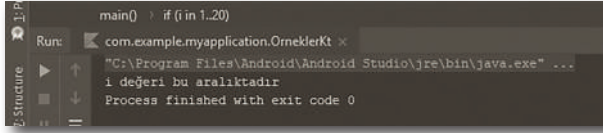
Örneğin bir değişkenimiz var ve değeri 5 olsun. Bu değişken 1 ile 20 arasında var mı kontrolünü yapalım.

Örnek

```
fun main(args: Array<String>) {
    var i: Int = 5
    if(i in 1..20){
        print("i değeri bu aralıktadır")
    }
    else
    {
        print("i değeri bu aralıktaki değildir")
    }
}
```

Yukarıdaki örneğimizde `i` değişkeni tanımladık ve 5 değerini atadık. Daha sonra `if` yapısı içinde 1 ile 20 arasında bir range (aralık) tanımladık ve dahası `in` anahtar kelimesini kullanarak `i` değişkeni bu aralıktaki mı kontrol ettik. Kontrolün sonucu `true` döndüğü için `if` içindeki işlemi yaptı. Artık biliyoruz eğer `false` dönsenydi `else` içindeki işlemi yapacaktı.

Kodun çıktısı aşağıdaki gibidir.



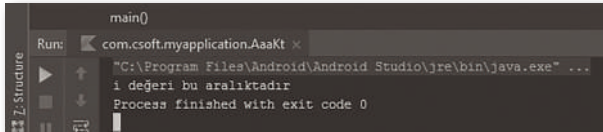
```
main() / if (in 1..20)
Run: com.example.myapplication.OrneklerKt x
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...
i deđeri bu aralıktadır
Process finished with exit code 0
```

Yukarıdaki örneđi elbette range (aralık) yapısı kullanmadan da yapabilirsiniz. Aşađıdaki örneđi range yapısını kullanmadan yapalım.

Örnek

```
fun main(args: Array<String>) {
    var i: Int = 5
    if(1 <= i && i <= 20){
        print("i deđeri bu aralıktadır")
    }
    else
    {
        print("i deđeri bu aralıkta deđildir")
    }
}
```

Bu örnekte if içinde range kullanmadık ve daha önceden de yaptığımız gibi kontrolleri yaptık. i deđeri 1'e eşit ya da büyük mü kontrolü yaptık. Daha sonra 10'a eşit ya da küçük mü kontrolünü yaptık. Bu iki kontrolden de true deđer dönmesi lazım çünkü && (ve) operatörü kullandık. Bizim deđerimiz 5 olduđu ve 1'den büyük 10'dan küçük olduđu için if içindeki işlemi yaptırdık ve aşağıdaki çıktıyı elde ettik ama dikkat ederseniz range yapısını kullandığımız zaman daha sade bir kod elde etmiş oluyoruz. Daha uzun programlarda bu sadelik can kurtarıcı oluyor.



```
main()
Run: com.csoft.myapplication.AaaKt x
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...
i deđeri bu aralıktadır
Process finished with exit code 0
```

Bu aralığı tam sayılar ile yapabildiğimiz gibi karakter ile de yapabiliriz. Aşađıdaki örnekte char tipinde bir deđerken belirleyelim ve yine bir aralık tanımlayıp deđerkenimiz o aralıkta mı bakalım.

8

DİZİLER

BU BÖLÜMDE

Dizi Nedir?	166
Dizi Tanımlama	166
Dizi Değer Atama	167
Dizi Değer Okuma	168
Çift Boyutlu Diziler	175
Neler Öğrendik?	178

Bu bölümde, aynı tipten bir çok veriyi bir arada tutan dizi yapılarını işleyeceğiz.

Dizi NEDİR?

Dizi aynı tipten çok sayıda veriyi bir arada saklayan veri yapısına **dizi** denir.

Diziler ne işimize yaracak diye düşünebilirsiniz aynı tipten birden fazla veriyi tanımlamasam ne olur ne olmaz diye düşünebilirsiniz. Örneğin; 10 tane sayıyı toplayacaksınız bunun için 10 tane ayrı ayrı tam sayı tanımlamanız lazım ama eğer dizi kullanırsanız 10 kere ayrı ayrı tanımlamak yerine dizi içinde tanımlama yapıyoruz ve daha sonra dizi üzerinde işlemler yapıyoruz. İlerleyen kısımlarda örneklerle anlatım yaptığımızda konuyu daha iyi kavrayacaksınız.

Dizi TANIMLAMA

Daha önceki konularda da gördüğümüz gibi nasıl değişken tanımlıyorsak aynı şekilde dizide tanımlanır.

`val dizi_Adi = Array (diziUzunluğu){başlangıç değerleri}` şeklinde tanımlama yaparız. Şimdi birkaç tane dizi tanımlamasına bakalım.

Örnek

```
fun main(args: Array<String>) {
    val intArray = Array<Int>(20){0}
    val stringArray = Array<String>(20){""}
    val doubleArray = Array<Double>(20){1.0}
    val charArray = Array<Char>(20){'A'}
}
```

Yukarıda birkaç tane dizi tanımlaması yaptık. Burada dizilerin uzunluklarını 20 olarak belirttik. Bu uzunluk belirleme işlemi bize kalan bir durumdur. Uzunluğu istediğimiz gibi ayarlayabiliriz. Uzunluk belirledikten sonra initial (başlangıç) değerleri tanımladık. Oluşan dizinin tüm elemanlarına ilk bu değer atanır. Örneğin; yukarıdaki `intArray` dizisinin uzunluğu 20 ve elemanlarının hepsi 0'dır.

NOT

Bu kısımda önemli olan bir adım var. Dizinin boyutunu yukarıda 20 olarak tanımladık bu durum program sonuna kadar aynı kalır. Yani program içinde dizinin boyutunda arttırma ya da azaltma yapamayız. Çünkü dizi oluşturulurken bellekte ona göre yer ayrılır ve dizinin uzunluğu hep sabit kalır.

Dizi DEĞER ATAMA

Dizi tanımlama işlemini yaptık daha sonra başlangıç değerlerini atadık ama elbette işlemlerimiz başlangıç değerleri yapmıyoruz yaptığımız işlemlere göre diziye değer atıyoruz.

Bu atama işlemini iki ayrı yolla yapıyoruz.

1. Yol

```
fun main(args: Array<String>) {
    val dizilistesi = Array<Int>(10){0}
dizilistesi[0] = 10
    dizilistesi[1] = 81
    dizilistesi[2] = 3
    dizilistesi[3] = 119
    dizilistesi[4] = 21
    dizilistesi[5] = 1
    dizilistesi[6] = 5
    dizilistesi[7] = -78
    dizilistesi[8] = 8
    dizilistesi[9] = 9
}
```

Yukarıdaki işlemde dizilistesi isminde bir dizi tanımladık ve dizinin boyutunu 10 olarak belirledik. Bu belirleme işleminden sonra dizinin elemanlarının ilk değerini 0 (sıfır) olarak atadık.

Bu tanımlama işleminden sonra sıra dizinin elemanlarına değer atamaya geldi. İstersek bütün elemanlara tek tek değer atarız istersek istediğimiz elemanlara değer atarız değer atamadığımız dizi elemanları ise başlangıç değerlerini kullanır. Yukarıdaki örnekte dizi elemanlarının hepsine tek tek değer atadık.

Bu şekil atama yapabildiğimiz gibi bir diğer atama şekli aşağıdaki gibidir.

2. Yol

Burada diziye eleman atama işlemini set() metodu ile yapıyoruz.

```
fun main(args: Array<String>) {
    val dizilistesi = Array<Int>(10){0}
dizilistesi.set(0,1)
    dizilistesi.set(1,10)
```

12

ERİŞİM BELİRLEYİCİLER (VISIBILITY MODIFIERS)

BU BÖLÜMDE

Erişim Belirleyiciler Nedir?

240

Neler Öğrendik?

245

Bu bölümde, değişkenler, fonksiyonlar, sınıflar, constructor vb. yapılar için kısıtlamaları işleyeceğiz.

ERİŞİM BELİRLEYİCİLER NEDİR?

Buraya kadar sınıflar tanımladık bu sınıflardan sınıflar türettik. Bu tanımladığımız ve türettiğimiz sınıflar içinde değişkenler, nesnelere, yapıcılar ve fonksiyonlar vardı. Peki, konumuzunda başlığı olan bu erişim belirleyicilerin görevi nedir? Erişim belirleyiciler bir varlığın yani sınıfın, metodun, değişkenin, arayüzlerin erişilebilirliğini belirtmek için kullanılan anahtar kelimelerdir. Çalıştığımız bir projede güvenlik vb. durumlardan dolayı yukarıda saydığımız varlıklara her yerden erişilmemesini veya tam tersi erişilmesini isteyebiliriz. İşte böyle durumlarda erişim belirleyicileri kullanılırız. Ayrıca projemiz büyüdükçe oluşabilecek sorunları da (Örneğin; çakışma) engelleyerek daha güvenli ve tutarlı bir kod yazmamızı sağlar. Kotlin'de 4 tane erişim belirleyici mevcuttur.

1. Public
2. Private
3. Protected
4. Internal

Kotlin de eğer fonksiyon, değişken vb. yapılar için erişim hakkında bir tanımlama yapılmaz ise default(varsayılan) olarak public dir.

PUBLIC NEDİR?

Bir öge public olarak tanımlandığı zaman hiçbir kısıtlama almaz. Kısıtlama almazdan kastımızı nedir? Yani hem kendi kod bloğu içinden hem de dışından ve hatta farklı paketlere sahip sınıflar üzerinden dahi çağrılabilirler. Yani tüm sınıflardan rahatça erişebilmek istediğimiz tüm öğeler için **public** anahtar kelimesini kullanırız. Ama burada dikkat etmemiz gereken çok önemli bir husus var güvelik ile ilgili öğeler için public anahtar kelimesini kullanmamaya dikkat etmeliyiz.

NOT

Eğer bir öge için herhangi bir erişim belirteci belirtilmediyse o zaman varsayılan olarak public'dir. Buraya kadar olan örneklerimiz de dikkat ederseniz herhangi bir erişim belirleyici tanımlamadık o yüzden hepsi public'dir.

Örnek

```
fun main(args: Array<String>) {
    var obj = publicExample()
    println("Dışardan erişim ile ekrana i değeri yazdırılıyor i = ${obj.i}")
    obj.yazdir()
}
```



```

}

class publicExample {
    public val i = 1
    fun yazdir() {

println("Sınıf içinden erişim ile public değişken olan i'nin değeri = $i")
    }

}

```

Bu örneğimizde publicExample isminde bir sınıf tanımladık ve bu sınıf içinde yer alan i değişkenini public olarak belirttik. Daha sonra bu değişkene hem sınıf içinden hem de sınıf dışından erişim sağlayarak ekrana yazdırdık. Ve sonuç olarak aşağıdaki gibi bir çıktı elde ettik.

```

main()
com.example.myapplication.OrneklerKt x
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...
Dışardan erişim ile ekrana i değeri yazdırılıyor i = 1
Sınıf içinden erişim ile public değişken olan i'nin değeri = 1
Process finished with exit code 0

```

DİKKAT

Unutmamalıyızki public olarak bir tanımlama yapmasaydık bile Kotlin derleyicisi varsayılan olarak public şeklinde tanımlar.

PRIVATE NEDİR?

Private belirteci public belirteci gibi esnek değildir. Aksine en katı erişim belirtecidir. Peki, en katı erişim belirtecinden kastımız nedir? Bu erişim belirteci ile tanımlanan ögenin ne zaman ve nasıl kullanılacağını kontrol edebiliriz. Ve bu belirteç ile tanımlanan öğeye sadece tanımlandığı sınıf içinden erişim sağlayabiliriz. Bu şekilde sınıf dışından gelebilecek erişim girişimlerinin önüne geçmiş oluruz. Bu yüzden güvenlik ile ilgili öge tanımlamalarında private kullanmayı tercih ederiz.

NOT

NOT: Private ögenin kullanıldığı sınıftan türetilen sınıftan da erişim sağlanmaz. Yani kalıtım yolu ile erişim söz konusu değildir.

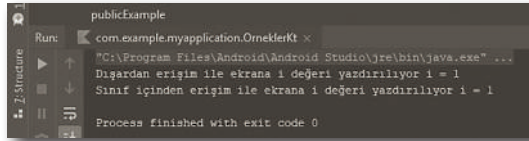
```

class privateExample {
    private var i = 1

    inner class Nested {
        private fun yazdir() {
            println("Sınıf içinden erişim ile ekrana i değeri yazdırılıyor i = ${i}")
        }
    }
}

```

Örneğimizde hem `i` değişkeni için hem de `yazdir` fonksiyonu için `private` tanımlama yaptık. Sonrada erişmeye çalıştık ve aşağıdaki gibi bir çıktı elde ettik.



Yani buradan çıkaracağımız sonuç eğer `private` bir tanımlama yapıyorsak o zaman sadece tanımlanan öğe hangi sınıfın içinde ise oradan erişim gerçekleşir.

PROTECTED NEDİR?

`Protected` ile tanımlanan öğeye tanımlandığı sınıfın içinden erişebiliriz. Bu yönü ile `private`'a benzer ama aralarında bir fark var. O da `protected` ile tanımlanan öğeye tanımlandığı sınıftan türetilen (miras alan) sınıf içinden de erişilebilir. Yani demek istediğimiz `protected` ile tanımlanmış öğe kalıtım yolu ile erişilebilir.

Örnek

```

fun main(args: Array<String>) {
    var obj = Test()
    obj.Yazdir()
}

open class protectedExample {
    protected val i = 1
}

class Test : protectedExample(){
    fun Yazdir(){
        println("Protected erişim belirtci ile i değişkenine erişim sağlanıyor. i nin değeri = $i")
    }
}

```


13

LAYOUT'LAR

BU BÖLÜMDE

Layout Nedir?	248
Neler Öğrendik?	259

Bu bölümde, ara yüz oluşturmamızı sağlayan uygulamalarımızda view (görünüm) elemanlarını düzenlememizi sağlayan layoutları işleyeceğiz.

Artık Android de ara yüz tasarım kısmına giriş yapıyoruz. Bundan sonra yapacağımız uygulamaları görsel olarak tasarlayacağız. Bunun için bilmeniz gereken temel öğeler var onlara hep birlikte bakalım.

LAYOUT NEDİR?

Layout arayüz oluşturmamızı sağlayan, uygulamamızda view (görünüm) elemanları düzenlememizi sağlayan kontrollerdir. Peki, Layout'un faydası nedir? Bu yapıların faydası tasarım yaptığımız zaman kullandığımız elemanların sağa, sola - yukarıya, aşağıya kaymasını engeller ve eklenen elemanların belirli bir düzende kalmasını sağlar.

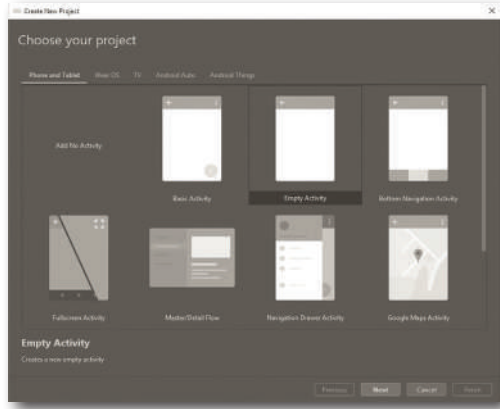
Android uygulamalar geliştirdiğimizde görsel kısımlar res klasörü altında yer alan Layout dosyalarında saklanır. Bu dosyalar .xml formatında olur. Hatta projeyi ilk çalıştırdığınızda otomatik olarak ilk gelen layout `activity_main.xml`'dir. Tasarım işlemlerini .xml dosyasından yapabileceğimiz gibi Design (tasarım) sayfasından da sürükleyip bırakarak yapılabilir.

Elbette layout'ların da belli başlı özellikleri mevcuttur. Bunlardan en çok kullanacaklarımızı birlikte inceleyelim.

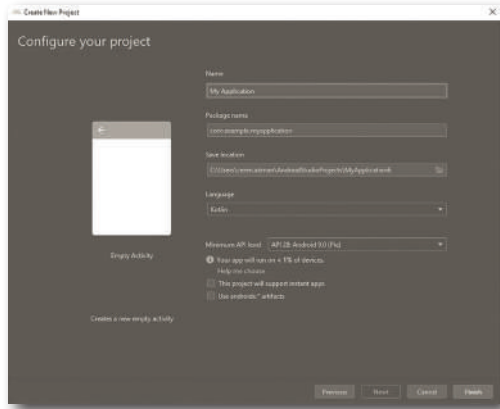
- 1. layout_width:** Kullandığımız görüntü elemanlarının uygulamada ne kadar yer kaplayacağını belirler. Bu özellikte kendi içinde 2'ye ayrılır.
 - » **wrap_content:** İhtiyaç duyulan alan kadar öğeyi genişletir. Yani bulunulan alan içerisinde yer alan metin, resim veya bileşen kadar yer kaplar.
 - » **match_parent:** Bulunduğu alan içerisinde yatay veya dikey olarak alanı kaplar.
- 2. gravity:** Nesnenin bulunduğu alan içinde nereye konumlanacağını belirler. (top, center, left, right vb.)

Birden fazla layout tipleri mevcuttur ama o tiplere geçmeden önce Android Studio'da tasarım yapacağımız ekran nasıl açılıyor onu hatırlatalım.

Bunun için Android Studio'yu açıyoruz **File>New>New Project** diyoruz. Daha sonra karşımıza aşağıdaki gibi bir ekran çıkıyor.

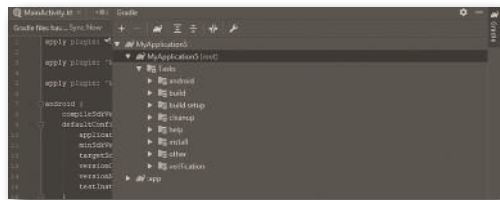


Bu ekranda **Empty Activity**'i seçiyoruz. Bir sonraki adımda karşımıza aşağıdaki gibi bir ekran daha geliyor.



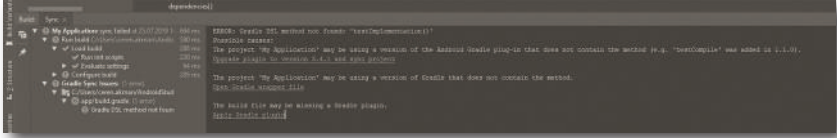
Bu sayfada istenilenleri kitabımızın ikinci bölümde öğrendiğimiz gibi dolduruyoruz. Daha sonra **Finish** butonuna tıklıyoruz ve projemizi açılıyor.

Bu açılan projede `app>src>main>res>layout` aşağıdaki resimde de gördüğünüz yolu izleyerek tasarım sayfamızı açıyoruz.



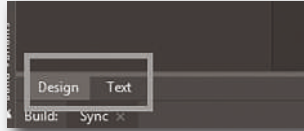
DİKKAT

Eğer işletim sisteminiz Türkçe ise Android Studio İngilizce olduğu için kuvvetli muhtemel aşağıdaki hatayı alacaksınız. Bu hatayı çözmek için **app** klasörü altında yer alan **build.gradle** klasörü içinde **testImplementation** satırları göreceksiniz. Burada **testImplementation**'da yer alan **I** harfini **I** ile değiştiriyoruz sonra tekrar senkronize ediyoruz ve bu hatadan kurtuluyoruz.

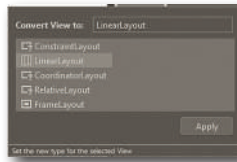


Evet, uygulamamızı tasarlayacağımız sayfamızı açtık. Açılan sayfada otomatik olarak karşımıza bir ekran gelir ve bu ekranda bir öge içinde Hello World yazdığını görürüz. Bu yazının yazdığı öge bir layout içinde yer alır. İçinde bulunduğu layout yapısının belli başlı tipleri mevcuttur. Şimdi sıra onların içinden en çok kullanacaklarımızı incelemeye geldi.

Design ve .xml sayfaları tasarım sayfasının sol alt tarafında yan yana yer alır. Eğer **Design** sayfasını seçerseniz **dizayn**, **Text** sayfasını seçerseniz **.xml** sayfasına geçersiniz.

NOT

Bu inceleme işleminden önce bir işlem daha yapmamız lazım. Android Studio varsayılan olarak **Constraint Layout** tipi ile gelir. Öncelikle bunu **Linear Layout** ile değiştirmemiz lazım. Bunun için tasarım sayfasında layout üzerinde sağa tıklıyoruz. Sağa tıkladığımız zaman karşımıza gelen ekranda **Convert view** seçeneğini seçiyoruz. Daha sonra karşımıza aşağıdaki gibi bir ekran geliyor. Bu ekranda hangi layout tipini kullanacaksak seçiyoruz. Biz şimdi **LinearLayout** tipini seçiyoruz ve **Apply** butonuna tıklıyoruz.



18

İKON İŞLEMLERİ

BU BÖLÜMDE

İkon Nedir?	314
Adaptive İkon Nedir?	315
Neler Öğrendik?	317

Bu bölümde, uygulamamız için nasıl ikon oluşturacağımızı işleyeceğiz.

İKON NEDİR?

Geliştirdiğimiz bir uygulamayı cihazlara yüklediğimizde uygulamamızı temsil eden simgeler olur bunlara ikon denir. Android Studio’da bir uygulama oluşturduğumuz zaman varsayılan olarak aşağıdaki iki resim ikon olarak gelir.



Bu ikonlar res klasörü altında yer alan 5 farklı klasörde depolanmaktadır. Neden 5 farklı dosyada ayrı ayrı bulduklarını düşünebilirsiniz.

NOT

İkon için belirlediğimiz resim 5 klasörde de aynı resim ve aynı isimde olmak zorunda.

mipmap-hdpi = 72 x 72

mipmap-mdpi = 48 x 48

mipmap-xhdpi = 96 x 96

mipmap-xxhdpi = 144 x 144

mipmap-xxxhdpi = 192 x 192

Yani bu dosyalardan anlayacağımız ikon için kullanacağımız resim rastgele bir şekilde dosyalara atılmıyor. Karşılarında yazan boyuta göre resimler ayarlanıp dosyalara atılıyor. Neden resimleri farklı boyutlandırıyoruz? diye düşünebilirsiniz. Nedeni ise şudur. Uygulamamız birden fazla cihazda olacak ve bu cihazların çözünürlükleri, boyutları birbirinden farklıdır. Bu yüzden ikonumuzu her boyutta tasarlamamız gerekiyor. Ayrıca her dosya da yer alan ikonların hepsinin resim ve isimlerinin aynı olması gerekir.

NOT

İkon olarak ekleyeceğimiz resimleri eklemeyen önce varsayılan ikonları silerek ve varsayılan ikon isimleri olan `ic_launcher.png`, `ic_launcher_round.png` şeklinde resimleri kayıt edersek programın kod kısmında bir değişim yapmaya gerek kalmaz. Eğer isim değişikliği yaparsak o zaman aşağıda gördüğünüz `AndroidManifest.xml` sayfasında ki `android:icon="@mipmap/ic_launcher"` ve `android:roundIcon="@mipmap/ic_launcher_round"` kod satırında `@mipmap/ikon_resim_ismi` şeklinde değişiklik yapmamız lazım.

xml Kodu

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

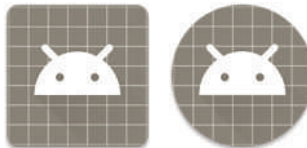
</manifest>

```

Şimdi Android 8.0 ve üzerindeki cihazlarda farklı şekilde ve efektte ikon kullanmamızı sağlayan Adaptive İkon Nedir? ona bakalım.

ADAPTİVE İKON NEDİR?

Android 8.0 ve üstü olan cihazlarda tüm simgeleri daha tutarlı hale getirmek amaçlanmıştır. Bunun içinde farklı şekilde ve efektte ikon kullanmamamızı sağlayan adaptive uygulama ikonunu bize sunar. Bunun için iki katman söz konusudur. Birincisi **foreground layer**, ikinci katman **background layer**'dir. Bu iki katman birbiri üzerine istiflenir ve background tamamen opak olmalıdır, foreground ise saydamlık içermelidir. Daha sonra üst üste istiflenen yapı bir input olur ve maskelenir burada bu maskelemeyen kastımız aşağıdaki şekillerde de gördüğümüz gibi yuvarlak ya da köşeleri yuvarlanmış bir kare şekli alır.



20

MEDYA İŞLEMLERİ

BU BÖLÜMDE

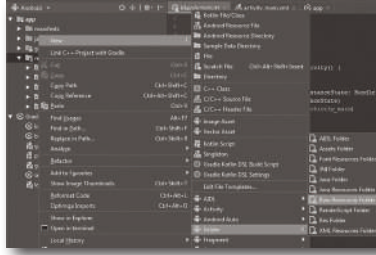
Media Player Sınıfı Nedir?	342
Nasıl Oynatılır?	342
Media Recorder Sınıfı Nedir ve Nasıl Kullanılır?	348
Neler Öğrendik?	352

Bu bölümde, uygulamamızda nasıl ses ve video oynatacağz onu öğreneceğiz.

MEDIA PLAYER SINIFI NEDİR?

Android Multimedia çatısı çeşitli medya formatlarını destekleyen ve uygulamamıza ses, video gibi yapıları eklememizi sağlayan sınıftır. Ayrıca bu sınıf dışarıdan ses ve video eklememizi sağladığı gibi istediğimiz uzunlukta video kaydetmemizi ve oynatmamızıda sağlar. Bu ses ve videoları oynatmamız için uygulamamıza eklememiz gerekli.

Bunun için uygulama dosyasında yer alan res klasörü altına raw isiminde klasörümüzü oluşturup ses ve video dosyalarımızı bu klasöre atmamız lazım. Aşağıdaki resimde gördüğümüz gibi res klasörü üzerinde faremizle Sağ tıklayıp New>Folder>Raw Resources Folder yolunu izliyoruz.



Oynatmak istediğimiz ses ve video dosyalarımız böyle ekleyebildiğimiz gibi aynı zamanda internet üzerinden de bu sınıf sayesinde veri aktarımı sağlayabilirsiniz.

NASIL OYNATILIR?

MediaPlayer sınıfının kod içinde nasıl kullanıldığına geçmeden önce bu sınıfı kullanırken çoğunlukla kullanmak zorunda olduğumuz bazı izinleri incelememiz lazım. Örneğin; internet bağlantısı sayesinde belli bir yerden veri aktarımı yapmak istersek **AndroidManifest.xml** dosyasında değişiklik yapıp internet izni almamız gerekiyor. Bunun içinde **AndroidManifest.xml** dosyamıza `<uses-permission android:name="android.permission.INTERNET"/>` kod satırını yazmamız lazım. Bu kod ile hem internete bağlanmasına izin verecek, hem de internet üzerinden rahatlıkla veri aktarımı sağlayabileceğiz.

Ekleme işlemi yapıldıktan sonra **Manifest** dosyası aşağıdaki gibidir.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.csoft.myapplication">
    <uses-permission android:name="android.permission.INTERNET"/>
```

```

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>

            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
</application>
</manifest>

```

Bir diğer izin ise uyku modudur. Medya oynatılırken ekranın kararmaması ve işlemcinin (CPU) kendisini uyku moduna almaması için ayarlama yapmamız lazım. Bunun içinde **AndroidManifest.xml** dosyasında izin vermemiz lazım.

İzin vermek için `<uses-permission android:name="android.permission.WAKE_LOCK"/>` kod satırını kullanırız. Bu kod sayesinde ekran kararmamasını ve uyku moduna geçişi engellemiş oluruz. Ekleme işleminden sonra **Manifest** dosyası aşağıdaki gibidir.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.csoft.myapplication">

    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">

```

25

GOOGLE PLAY'DE UYGULAMA YAYINLAMA

BU BÖLÜMDE

Ön Hazırlık	404
İmzalama (Sign) ve Derleme (Build)	405
Yayınlama (Publish)	407
Neler Öğrendik?	409
Son Söz	410
Dizin	411

Bu bölümde, tamamladığımız uygulamamızı kullanıcıların hizmetine sunmak için nasıl yayınlayacağımız hakkına bilgi sahibi olacağız.

Uygulamalarımızı kullanıcılara sunmak için hazırlarız. Bunun içinde belli başlı mağazalarda uygulamalarımızı kullanıcıların indirip kullanabileceği şekilde yayınlarız. İOS uygulamalarımızı AppStore'a yükleriz ve kullanıcılar bu mağazadan indirir. Ama Android uygulamalarımızı yükleyip indirebileceğimiz zamanlar birden fazladır. Ama Android'in ana mağazası Google Play'dir. Biz de bu bölümde uygulamalarımızı Google Play de nasıl yayınlayacağımızı ele alacağız. Şimdi uygulama yayınlamak için gerekli olan hazırlıklar neler onlara bakalım.

ÖN HAZIRLIK

Geliştirdiğimiz uygulamamızı yayınlamaya hazır hale getirmek için belli başlı 3 adım bulunmaktadır. Bunlar

- » İmzalama (Sign)
- » Derleme (Build)
- » Yayınlama (Publish)

İmzalama (sign) işleminden kastımız uygulamamızı geliştirici lisansımız ile imzalarız. Derleme(build) işleminden kastımız uygulamayı yayına hazır çalışır hale getiririz. Yayınlama(publish) bu işlem ise son adımdır hazır hale getirilen uygulama yayınlanır.

Şimdi imzalama işlemine geçmeden önce uygulamamızın build.gradle(Module:app) dosyamıza geliyoruz. Ve bu dosyamızda yer alan bazı satırların ne işe yaradığını bakalım. Bu sayfada yer alan versionCode 1

versionName "1.0" versionCode(Sürüm kodu) ve versionName(Sürüm ismi) değerlerini belirlemek bizim inisiyatifimize kalan durumlardır. Burada ister 1,1.5,2.0 şeklinde belirleyebildiğimiz gibi 100, 200 şeklinde de belirleyebiliriz. Ve uygulamamızın hangi sürüm olduğunu belirleyip uygulamada güncellemeler yaptıkça bu sürümü değiştiririz. Daha önceki bölümlerde de bahsettiğimiz gibi Google Play'de uygulama yayınlamak için targetSDKVersion'un 28 olması gerekiyor. Bunun dışında yer alan ve önemli olan diğer bir satır ise minSDKVersion burada belirleyeceğimiz değer ile uygulamanın çalışacağı en düşük SDK sürümünü belirleriz. Şimdi bu sayfada yer alan satırların ne işe yaradığını öğrendikten sonra sıra imzalama işlemine geldi.

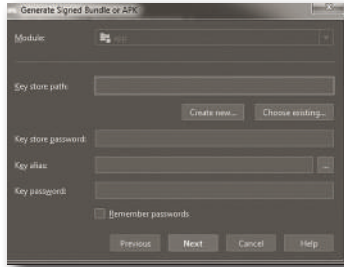
İMZALAMA (SIGN) VE DERLEME (BUILD)

Yazılmış olan uygulamamızı Google Play Store'da yayınlamak için imzalama işlemini gerçekleştirmemiz lazım. Yukarıda da bahsettiğimiz gibi imzalanmayan uygulamalar Google Play'de yayınlanmaz. Şimdi nasıl imzalayacağız öğrenelim.

Burada **Build>Generate Signed Bundle/APK** seçeneğini seçiyoruz. Daha sonra karşımıza aşağıdaki gibi bir ekran geliyor.



APK seçeneğini seçiyoruz ve **Next** ile devam ediyoruz.



Burada **Create New** butonuna basıyoruz. Ve karşımıza aşağıdaki gibi bir ekran gelecek gerekli bilgileri dolduruyoruz.

