Intent'ler - Diğer Uygulamalarla Etkileşime Geçmek

Bir Android uygulaması birçok farklı Activity içerir. Her Activity yeni arayüzler gösterirken belirli bir görevi (haritanın gösterilmesi, fotoğraf çekilmesi gibi) yerine getirir. Kullanıcıyı bir Activity'den diğerine geçirmek için uygulamanızda Intent sınıfını kullanmalısınız. Uygulamanızın yapacağı herhangi bir işte "amacı" belirtmek için Intent sınıfını kullanmalısınız. Uygulamanızda startActivity() gibi bir metotla sisteme Intent geçirdiğinizde, sistem doğru uygulamayı ve eylemi belirlemek için bu Intent'i kullanır. Intent sınıfı, farklı uygulamalar tarafından kullanılan bir activity'yi çalıştırmaya da şans verir.

Bir Intent belli bir bileşeni başlatmak (örneğin: belirli bir Activity'yi) için açık (explicit) olabilir. Bununla birlikte hedeflenen amacı gerçekleştirebilecek (örn: fotoğraf çekilmesi) herhangi bir bileşeni başlatmak için örtülü (implicit) de olabilir.

Bu eğitimin devamında Intent kullanarak diğer uygulamalarla nasıl basitçe bağlantı kuracağınızı öğrenebilirsiniz. Eğitimin sonunda bir uygulamayı çağırmayı, o uygulamadan çeşitli sonuçlar almayı ve uygulamanızı başka uygulamalara tepki verebilir hale getirmeyi öğrenmiş olacaksınız.

Başka Bir Activity'yi Başlatmak

Bu eğitim içeriğinde kullanıcının bir düğmeye basmasıyla nasıl başka bir Activity'yi açacağınızı öğreneceksiniz. Esasında bu konu Android'in en önemli bileşenlerinden Intent'e giriş niteliği taşıyor.

Bu içeriği okurken uygulayarak devam etmek için öncelikle basit bir Android projesi oluşturun. Bu projenin MainActivity layout dosyasında (activity_main.xml) bir tane metin kutusu (EditText) ve düğme (Button) oluşturun. Bu belgenin sonuna geldiğinizde, bu uygulamada kullanıcının metin kutusuna girdiği değeri başka bir Activity'de gösterebileceksiniz. Öncelikle diğer Activity'ye geçmemize yarayacak düğme ile ilgili işlemleri halledelim.

Gönder düğmesine tepki verme

Uygulamanızın activity_main.xml isimli layout dosyasında bir düğme (Button) olduğunu farz edelim. Bu düğmenin tıklanma (dokunma) eylemine yanıt vermek için activity_main.xml dosyanızı açın veandroid:onClick özelliğini <Button> elemanına ekleyin:

```
<Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_gonder"
        android:onClick="mesajGonder" />
```

android:onClick özelliğinin değeri olan "mesajGonder" Activity'niz içinde, kullanıcı düğmeye bastığında sistemin çağıracağı metodun adı.

MainActivity.java dosyasını açın (/src dizinindedir) ve şu metodu MainActivity sınıfının içine ekleyin:

```
/** kullanıcı gönder düğmesine basınca çağırılır */
public void mesajGonder(View view) {
    // düğmeye yanıt verecek bir şeyler
}
```

Sistemin android:onClick'e verilen metod adıyla buradaki metod adını eşleştirebilmesi için metod imzası tıpkı burada gösterildiği gibi olmalıdır. Metod mutlaka

- public olmalı
- void dönüş değeri içermeli
- tek parametre olarak View almalı (bu View tıklanmış olan View oluyor)

Bundan sonraki aşamada, bu metodun içini metin kutusunun içeriğini okumak ve girilen metni diğer Activity'ye göndermek için dolduracağız.

Intent oluşturma

Bir Intent, farklı bileşenler arasında (iki Activity gibi) çalışma zamanında bağlama (runtime binding) sağlayan nesnedir. Intent, bir uygulamanın "bir şeyi yapmaya niyetlenmesini" ifade eder. Intent'leri çok çeşitli görevlerde kullanabilirsiniz fakat başka bir Activity'yi başlatma işlemlerinde sıkça kullanılırlar.

mesajGonder() metodunun içinde MesajGosterActivity isimli Activity'yi başlatmak için bir Intent oluşturun:

```
Intent intent = new Intent(this, MesajGosterActivity.class);
```

Bu işlem Intent sınıfını import etmenizi gerektirir:

```
import android.content.Intent;
```
Öneri: *Android Studio'dayken* ***Alt + Enter*** *tuşlarına basarak eksik sınıfları import edebilirsiniz.*

Burada kullanılan yapılandırıcı metod iki parametre alıyor:

- İlk parametre olarak Context (this kullandık çünkü Activity sınıfının, Context'in alt sınıfıdır)
- Sistemin Intent'i teslim edeceği bileşenin Class ismi. (bu örnekte düğmeye tıklayınca başlayacak olan Activity'nin sınıf adı)

NOT: *Henüz MesajGosterActivity isimli Activity'yi oluşturmadığınız için IDE'niz hata verecektir. İlerleyen bölümlerde oluşturacaksınız; okumaya ve uygulamaya devam edin.*

Bir Intent, "ekstra veri" olarak isimlendirilen ve anahtar-değer 8key-value) veri tiplerini tutan koleksiyonları taşır. putExtra() metoduyla ilk parametre olarak anahtarı, ikinci parametre olarak değeri alır.

Sıradaki Activity'nin ekstra veriyi sorgulayabilmesi(*) için Intent'inizin ekstra verisi olarak kullanmak için sabit bir anahtar (key) tanımlamalısınız. Bunun için MainActivity sınıfınızın en üstüne EXTRA_MESSAGEtanımlamasını yapabilirsiniz:

```
public class MainActivity extends ActionBarActivity {
    public final static String EXTRA_MESSAGE =
"org.tcellgy.android.kitaplik.MESAJ";
    ...
}
```

Anahtarlara uygulamanızın paket adını ön ek olarak vermek iyi bir alışkanlıktır. Böylece onları eşsiz (unique) yaparsınız ve uygulamanız diğer uygulamalarla etkileşime geçebilir.

İkinci Activity'yi başlatma

Bir Activity'yi başlatmak için startActivity() metodunu çağırın ve bu metoda Intent'inizi geçirin. Sistem bu çağrıyı alacak ve Intent'te belirttiğiniz Activity'nin bir örneğini (instance) başlatacaktır.

Bu yeni kodla beraber Gönder düğmesi tarafından çağrılan mesajGonder() metodu şöyle olacaktır:

```
/** kullanıcı gönder düğmesine basınca çalışır */
public void mesajGonder(View view) {
    Intent intent = new Intent(this, MesajGosterActivity.class);
```

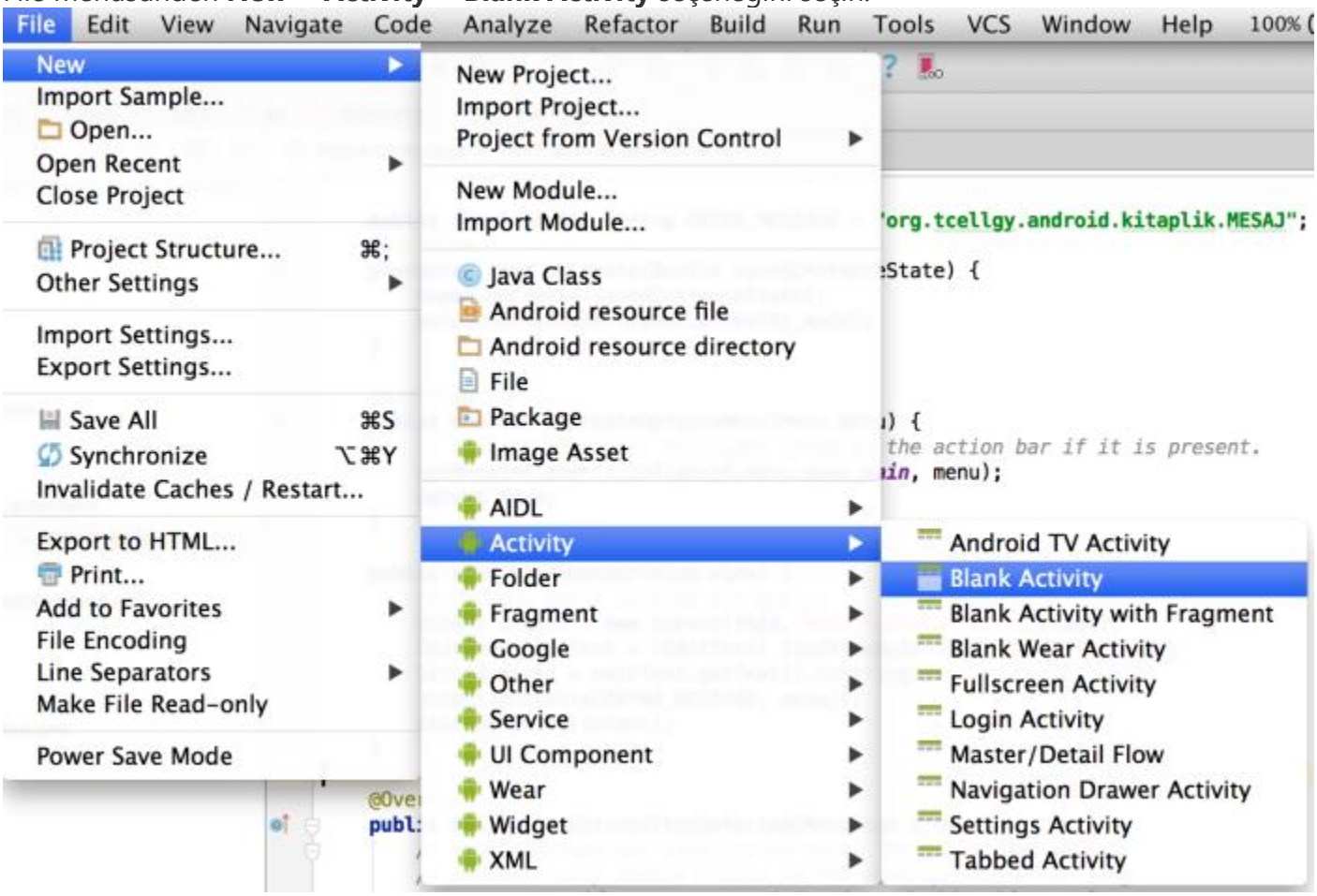
```
    EditText editText = (EditText) findViewById(R.id.edittext_mesaj);
    String mesaj = editText.getText().toString();
    intent.putExtra(EXTRA_MESSAGE, mesaj);
    startActivity(intent);
}
```

Bu kodun düzgün çalışması için MesajGosterActivity sınıfını oluşturmanın zamanı geldi.

İkinci Activity'yi oluşturma

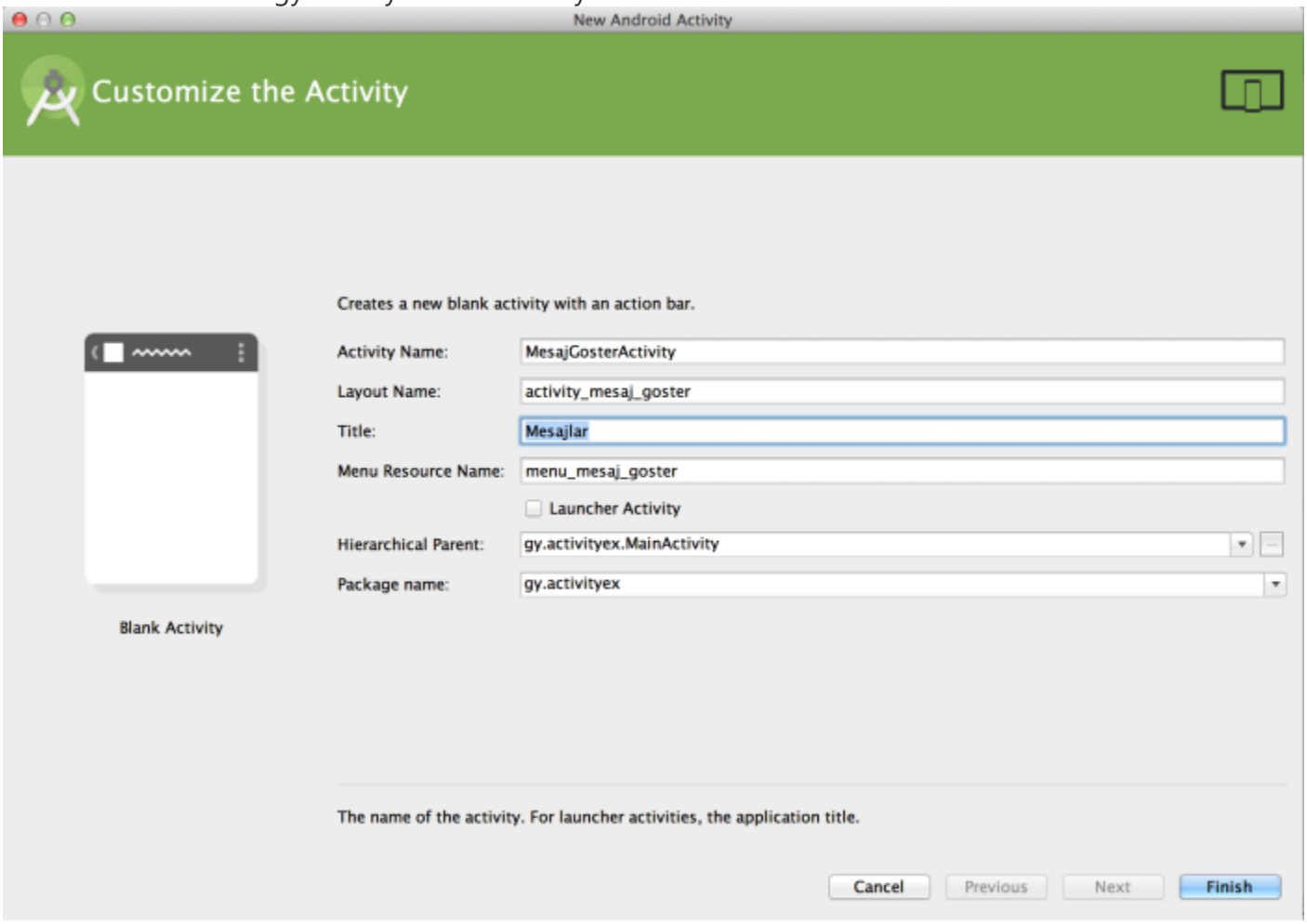
Android Studio kullanarak yeni Activity oluşturmak için:

1. File menüsünden **New > Activity > Blank Activity** seçeneğini seçin.



2. Activity ile ilgili ayrıntıları doldurun:
   - **Activity Name:** MesajGosterActivity
   - **Layout Name:** activity_mesaj_goster (Activity Name'i yazarken kendiliğinden değişir)
   - **Title:** Mesajlar

- **Hierarchical Parent:** gy.activityex.MainActivity



3. **Finish**'e basın.

/src dizininde oluşan MesajGosterActivity.java isimli dosya şunun gibi olacaktır:

```
package gy.activityex.MainActivity;


import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MesajGosterActivity extends ActionBarActivity {

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

```
	}

	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		// Inflate the menu; this adds items to the action bar if
it is present.
		getMenuInflater().inflate(R.menu.mesaj_goster, menu);
		return true;
	}

	@Override
	public boolean onOptionsItemSelected(MenuItem item) {
		// Handle action bar item clicks here. The action bar
will
		// automatically handle clicks on the Home/Up button, so
long
		// as you specify a parent activity in
AndroidManifest.xml.
		int id = item.getItemId();
		if (id == R.id.action_settings) {
			return true;
		}
		return super.onOptionsItemSelected(item);
	}
}
```

Activity'nin tüm alt sınıfları onCreate() metodunu mutlaka gerçeklemelidir - implemente etmelidir. Sistem bu metodu Activity'nin yeni bir örneğini (instance) oluştururken çağırır. Bu metot setContentView() metodu ile Activity layout'unu tanımlamanız gereken yerdir. Ayrıca Activity bileşenleriyle ilgili ilk ayarları yapmanızı önerdiğimiz yerdir.

Intent'i alma

Her Activity bir Intent tarafından çağrılır - kullanıcı nerede gezinirse gezinsin. Activity'nizde getIntent()metodunu kullanarak onu başlatan Intent'i elde edebilir ve içinde taşıdığı verilere ulaşabilirsiniz.

MesajGosterActivity sınıfının içindeki onCreate() metoduna gelip, az önce MainActivity'nin teslim ettiği Intent'i ve ekstra mesajı şu şekilde alabilirsiniz:

```
Intent intent = getIntent();
String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
```

Gelen mesajı gösterme

Artık ilk Activity'den gelen Intent'e erişebiliyoruz. Şimdi o Intent ile gelen mesajı MesajGosterActivity'de gösterelim. Bunun için MesajGosterActivity'nin layout dosyasını biraz düzenlememiz gerekiyor. Şu anki hali şöyle:

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="org.tcellgy.android.kitaplik.MesajGosterActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

Buradaki TextView üzerinde bazı değişiklikler yapacağız: Ona ulaşabileceğimiz bir kimlik numarası (ID) vereceğiz. Şöyle güncelleyebilirsiniz:

```
<TextView
        android:id="@+id/text_gelenmesaj"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20sp" />
```

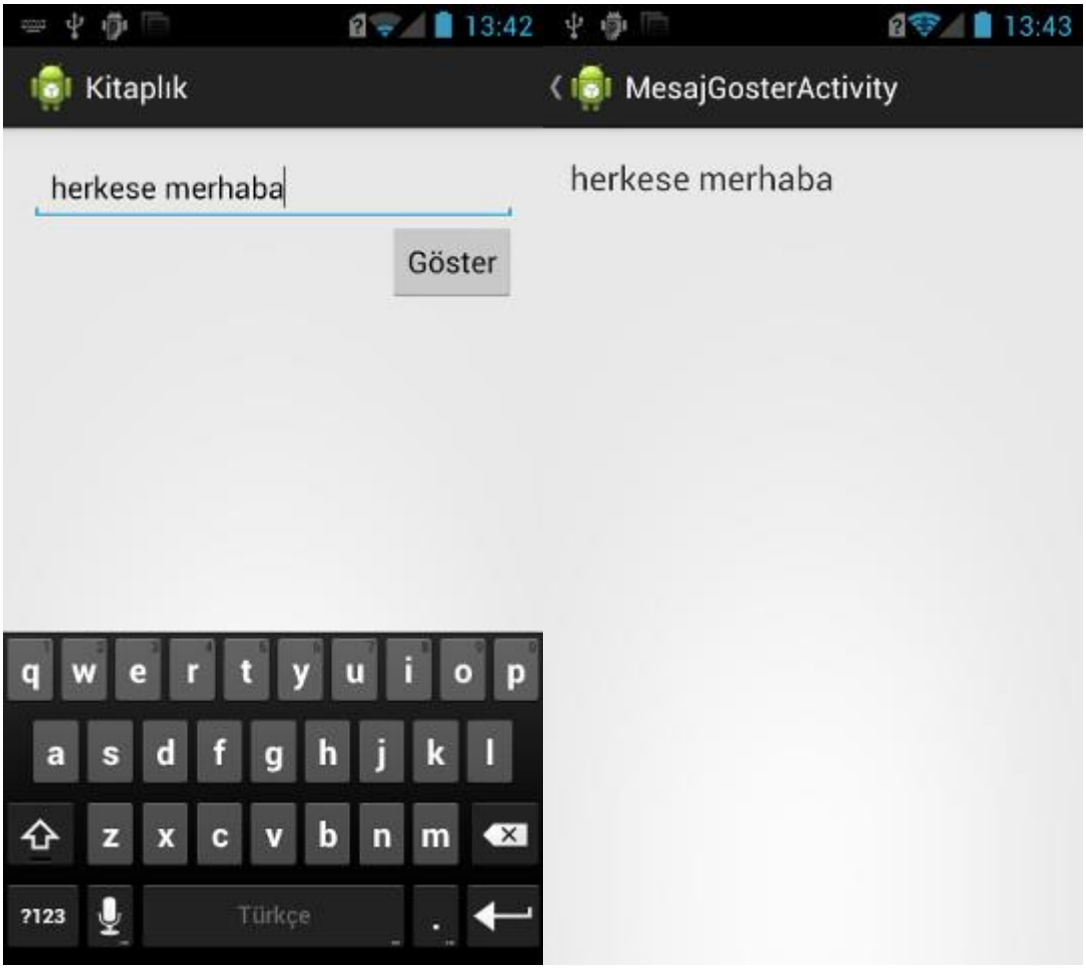
Buradaki android:id="@+id/text_gelenmesaj" satırına dikkat edin. Sadece bu TextView'ın ismi"text_gelenmesaj" oldu.

TextView'a isim verdiğimize göre artık Intent'ten aldığımız mesajı gösterebiliriz. Bunun içinMesajGosterActivity.java dosyanızdaki onCreate() metodunu şöyle özelleştirmek yeterli:

```
@Override
```

```
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_mesaj_goster);

		Intent intent=getIntent();
		String
mesaj=intent.getStringExtra(MainActivity.EXTRA_MESSAGE);

		TextView textView=(TextView)
findViewById(R.id.text_gelenmesaj);
		textView.setText(mesaj);
	}
```

Uygulamanızı çalıştırdığınızda gelen metin kutusuna bir şeyler yazın ve "Göster" düğmesine basın. Sonuç aşağıdaki gibi olacaktır:



Örnek projeyi ekranın yukarı sol tarafındaki bölümden indirebilirsiniz.

Kullanıcıyı Farklı Bir Uygulamaya Yönlendirmek

Android'in uygulamalara sağladığı önemli özelliklerinden biri de kullanıcıyı gerçekleştirmek istedikleri "eyleme göre" başka bir uygulamaya yönlendirme olanağı sunmasıdır. Örneklemek gerekirse, çeşitli resim işleme eylemleri yaptığınız bir uygulamada fotoğraf çekmek için bir activity yazmanıza gerek yoktur. Bunun yerine fotoğraf çekilmesi için bir Intent oluşturmanız ve çalıştırmanız yeterlidir. Sistem sizi kamera uygulamasına kendiliğinden yönlendirecektir.

Uygulamanızın Activity'leri arasında geçmek için Intent kullanmalısınız. Bunu genellikle **adresi belli intent**'ler **(explicit intent)** kullanarak, çalıştırmak istediğiniz bileşenin class ismini kullanarak yaparsınız, fakat başka uygulamada bir eylem gerçekleştirmek istediğinizde, örneğin "haritayı göster" gibi, **üstü kapalı intent (implicit intent)** kullanmak zorundasınız.

Bu eğitim içeriği özel bir eylem için nasıl *üstü kapalı (implicit) intent* oluşturacağınızı ve diğer uygulamanın ilgili eylemi gerçekleştirecek Activity'sini başlatmak için o Intent'i nasıl kullanacağınız üzerinde duracaktır:

1. Üstü kapalı (Implicit) intent oluşturma
2. Intent'i karşılayacak uygulama denetimi
3. Intent ile bir activity başlatma
4. Uygulama seçim diyalogu gösterme

Üstü kapalı (Implicit) intent oluşturma

Üstü kapalı intent, çalıştırmak istediğiniz bileşenin class'ını çağırmak yerine gerçekleştirmek istediğiniz eylemi belirttiğiniz intent türüdür. Belirttiğiniz eylem ne yapılacağını belirtir, örneğin bir şeyi göster (view), gönder (send), getir (get) gibi. Intentler eylemle ilgili çeşitli bilgiler de taşır. Örneğin göstermek istediğiniz adres, e-posta atılacak kişi, aranacak numara gibi. Oluşturmak istediğiniz Intent'e göre bu veri Uri veya diğer veri tiplerinin biçiminde olabilir veya Intent'in hiçbir veri taşımaya ihtiyacı da olmayabilir.

Uri nedir?
Uri (uniform resource identifier) yani nizami kaynak belirteci, bir kaynağı ya da veriyi isimlendirmek için kullanılan bir standarttır. Aynı zamanda kaynağı nitelendirir. Intent kullanırken bu türden bilgiler gereklidir.

Eğer veriniz Uri şeklindeyse, basit bir Intent() yapılandırıcı metoduyla (constructor) eylemi ve verileri tanımlayabilirisiniz.

Örneğimizde bir telefon araması yaparken Uri verisini nasıl kullandığımızı görebilirsiniz.

```
Uri number = Uri.parse("tel:5323334455");

Intent callIntent = new Intent(Intent.ACTION_DIAL, number);
```

Uygulamanız startActivity() ile callIntent isimli Intent'i çağırdığında Telefon uygulaması ilgili numarayı arar.

Aşağıda çeşitli Uri verileriyle bazı uygulamaların nasıl çağrıldığını görebilirsiniz.

Bir harita göstermek

```
// adrese göre harita noktası
Uri location =
Uri.parse("geo:0,0?q=1600+Amphitheatre+Parkway,+Mountain+View,+Californ
nia");
// ya da enlem ve boylam yerine göre konum bilgisi
// Uri location = Uri.parse("geo:37.422219,-122.08364?z=14"); // z
parametresi yaklaşma seviyesini belirler
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);
```

Bir web sayfası göstermek

```
Uri webpage =
Uri.parse("http://www.gelecegiyazanlar.turkcell.com.tr");
Intent webIntent = new Intent(Intent.ACTION_VIEW, webpage);
```

Diğer türlü üstü kapalı Intent'ler, farklı veri tipleri taşıyan "ekstra" veriler gerektirebilir, örneğin String gibi. Bir veya daha fazla ekstra veriyi veri tipine özel farklı putExtra() metotları kullanarak ekleyebilirsiniz.

Bilgi: *Intent'e istediğiniz veriyi ekleyemiyorsunuz. Veri tipine özel parametreler kabul eden putExtra() metotları var ve Intent'e ancak belli türden şartları kabul eden verileri ekleyebilirsiniz.*

Varsayılan olarak sistem, Intent tarafından gerek duyulan bir MIME tipini o Intent'e eklenen Uri'ye göre belirler. Eğer Intent'e bir Uri ilave etmediyseniz, Intent ile

ilişkilendirdiğiniz verinin tipini tanımlamak içinsetType() metodunu kullanmalısınız. MIME tipini tanımlamak Intent'i alması gereken Activity'lere ilave tanımlar yapmak anlamına gelir.

Aşağıda istenilen eylemi daha iyi tanımlamak için ekstra veri eklenen Intent örneklerini inceleyebilirsiniz.

Ekiyle birlikte e-posta göndermek

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);
//bu Intent'in bir URI'si yok. O halde MIME tipini "text/plain" tanımlayalım
emailIntent.setType(HTTP.PLAIN_TEXT_TYPE);
emailIntent.putExtra(Intent.EXTRA_EMAIL, new String[]
{"destek@gelecegiyazanlar.org", "ali@site.com"}); // alıcılar
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "E-posta başlığı");
emailIntent.putExtra(Intent.EXTRA_TEXT, "Merhaba, uygulama üzerinden e-posta gönderiyorum");
emailIntent.putExtra(Intent.EXTRA_STREAM,
Uri.parse("content://epostaya/ekleyeceginiz/dosyanin/adresi"));
// daha fazla eki, Uri'lerden oluşan ArrayList geçirerek de ekleyebilirsiniz.
```

Takvim etkinliği yaratmak

```
Intent calendarIntent = new Intent(Intent.ACTION_INSERT,
Events.CONTENT_URI);
Calendar baslamaZamani = Calendar.getInstance().set(2014, 3, 23, 9, 30);
Calendar bitisZamani = Calendar.getInstance().set(2014, 3, 23, 23, 30);
calendarIntent.putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME,
baslamaZamani.getTimeInMillis());
calendarIntent.putExtra(CalendarContract.EXTRA_EVENT_END_TIME,
bitisZamani.getTimeInMillis());
calendarIntent.putExtra(Events.TITLE, "Bayram kutlaması");
calendarIntent.putExtra(Events.EVENT_LOCATION, "Kasaba meydanı");
```

Not: *Bu Intent kullanımı API seviyesi olarak 14 ve üstlerinde geçerlidir.*

Not: Intent'i mümkün olduğunca kullanıma uygun tanımlamanız önemlidir. Örneğin ACTION_VIEWIntent'ini kullanarak bir resim göstermek istiyorsanız, MIME tipini image/* olarak tanımlamalısınız. Bunu yaptığınızda Intent tarafından tetiklenmiş olan,

diğer veri tiplerini gösterebilen (view) uygulamaların (harita uygulaması gibi) önüne geçersiniz.

Bunu biraz daha açıklayalım: Bir resim görüntülemek istediğini varsayalım. Bu resmi görüntüleyecek bir uygulama açmaya çalışırken Intent'inizi ACTION_VIEW eylemiyle yapılandırırsınız ancak ACTION_VIEW eylemi çok genel bir ifade. VIEW edebilen yani bir şeyler gösterebilen birçok uygulama olabilir. Eğer oluşturduğunuz Intent'in MIME tipini image/* olarak belirtmezseniz, sadece görselleri gösterebilen uygulamaları hedeflemiş olursunuz.

Intent'i karşılayacak uygulama denetimi

Android platformu belli Intent'lerin yerleşik uygulamalarca (Telefon, e-posta veya takvim uygulaması gibi) karşılık bulacağını garanti ediyor olsa da siz bir Intent'i çağırmadan önce mutlaka bir doğrulama adımı eklemelisiniz.

***Uyarı**: Eğer Intent'i çağırdığınızda cihazda belirttiğiniz işlemi yapacak bir uygulama yoksa, uygulamanız çökecektir.*

Intent'e yanıt verecek bir Activity'nin olup olmadığını doğrulamak için, queryIntentActivites() metodunu çağırarak Intent'inizi karşılama yeteneği olan Activity'lerin listesini alabilirsiniz. Dönen List boş değilse Intent'inizi güvenle kullanabilirsiniz. Örneğin:

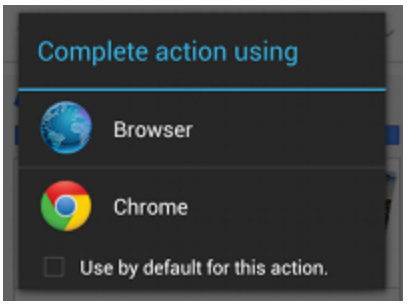
```
PackageManager packageManager = getPackageManager();
List<ResolveInfo> activities =
packageManager.queryIntentActivities(intent, 0);
boolean isIntentSafe = activities.size() > 0;
```

Eğer isIntentSafe değeri true ise en az bir uygulama Intent'e cevap verecektir. Eğer dönen cevap false ise almak istediğiniz eyleminizi gerçekleştirebilecek bir uygulama bulunmamaktadır.

***NOT:** Activity'niz ilk başladığında bu kontrolü yapmanız, belki de Intent kullanmaya çalışırken hata vermeden önce kapatmanız gereken özellikler olabileceği için makul bir işlem olabilir. Böyle bir durumda, yapmak istediğini eylemi yapan uygulamalara bağlantı verebilirsiniz. (Google Play'de uygulamanıza bağlantı vermek belgesine bakabilirsiniz.)*

Intent ile bir activity başlatma

Bir kere Intent'inizi oluşturup ekstra verileri ayarladığınızda startActivity() metodunu çağırarak bunu sisteme gönderebilirsiniz. Sistem işlemi gerçekleştirebilecek birden fazla Activity bulursa, kullanacağı uygulamayı kullanıcıya açtığı diyalog penceresinde gösterir. (Bkz Resim 1)



Resim 1: Intent'i karşılayabilecek birden fazla uygulama olduğunda çıkan seçim diyalogu

Eğer uygun sadece bir Activity varsa, sistem hemen şu metodu çalıştırır:

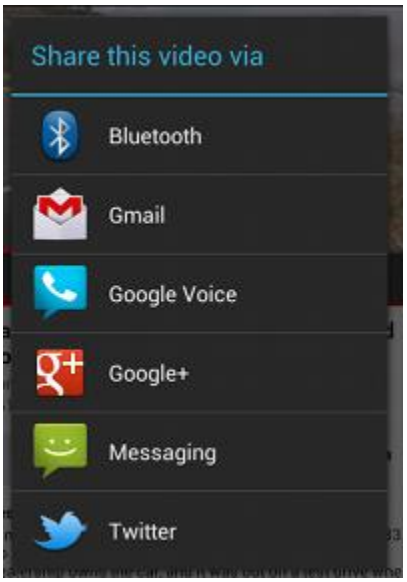
```
startActivity(intent);
```

Aşağıda bir uygulamada harita gösteriminin nasıl olacağını, ilgili uygulamanın doğrulamasını nasıl yapacağınızı gösteren kodu görebilirsiniz.

```
//Intent'i oluşturalım
Uri location =
Uri.parse("geo:0,0?q=1600+Amphitheatre+Parkway,+Mountain+View,+Califor
nia");
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);

// Karşılanacağının doğrulamasını yapalım
PackageManager packageManager = getPackageManager();
List<ResolveInfo> activities =
packageManager.queryIntentActivities(mapIntent, 0);
boolean isIntentSafe = activities.size() > 0;

// güvenli ise bir activity başlatalım
if (isIntentSafe) {
    startActivity(mapIntent);
}
```

Uygulama seçim diyalogu gösterme

Bir Activity'yi oluşturduğunuz Intent'i startActivity() metoduna geçirerek başlattığınızda ve Intent'inize yanıt verebilecek birden fazla uygulama olduğunda kullanıcı varsayılan olarak kullanılacak uygulamayı seçebilir. (Diyalog pencereciğinin altındaki seçenek kutusunu seçerek; Bkz: Resim 1) Bu durumlarda oluşan pencereden kullanıcı her zaman kullanacağı uygulamayı seçebilir. Web sitesi açılışlarında kullandığı web tarayıcıyı veya fotoğraf çekmek için kullandığı kamera uygulamasını insanlar genelde değiştirmek istemezler. Bu gibi durumlar için hoş bir özelliktir.



Resim 2: Seçim diyalogu

Bunun yanında bazı uygulamalarda bu tercih sık sık değişebilir, örneğin paylaşma aksiyonunu bazen Bluetooth, bazen e-posta bazen mesaj tercih edilebilir. Bunun gibi değişkenlik gösteren isteklere karşı bir seçim ekranı kullanmanız ve her zaman kullanmanız gereklidir.

Seçim pencereciğini göstermek için createChooser() metodunu kullanarak bir Intent oluşturup,startActivity()'ye geçirebilirsiniz. Örneğin,

```
Intent intent = new Intent(Intent.ACTION_SEND);

...

// arayüz metinleri için her zaman string kaynakları kullanın
// örneğin burada title değişkenine "Fotoğrafı şununla paylaş"
// metnini atamış oluyoruz
```

```
String title = getResources().getString(R.string.secici_basligi);
// seçici ekranı göstermek için Intent oluşturuyoruz
Intent chooser = Intent.createChooser(intent, title);

// Intent'in en az bir Activity çözümleneceğini doğruluyoruz
if (intent.resolveActivity(getPackageManager()) != null) {
    startActivity(chooser);
}
```

Bu kodla createChooser() metoduna geçirilmiş Intent'e karşılık gelebilecek uygulamaların listesini gösteren bir diyalog pencereciğinde gösterilir ve R.string.secici_basligi değişkeninden sağlanan metin diyalog başlığı olarak kullanılır.

Örnek: SMS veya E-posta ile Paylaşmak

Eğer uygulama içinden e-posta atmak ya da SMS göndermek istersek, Android ile gelen uygulamaları bir**Intent** yardımıyla açarak gönderebiliriz. Bunun için yapmamız gereken, oluşturulan **Intent** sınıfının ne amaçla kullanılacağını belirtmek ve **startActivity** metodunu oluşturulan **Intent** ile çağırmaktır. Bundan sonra telefonda bu işlem için seçilmiş uygulama açılır ve kişinin karşısına belirtilen değerlerle bir SMS ya da e-posta gönderme penceresi gelir.

Şimdi bunu bir Android uygulamasında nasıl yapabileceğimizi görelim. Öncelikle ekranda iki düğmesi olan basit bir uygulama oluşturalım. Bununla ilgili layout dosyası aşağıdaki gibi olsun:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/androi
d"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/send_sms"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="SMS" />
```

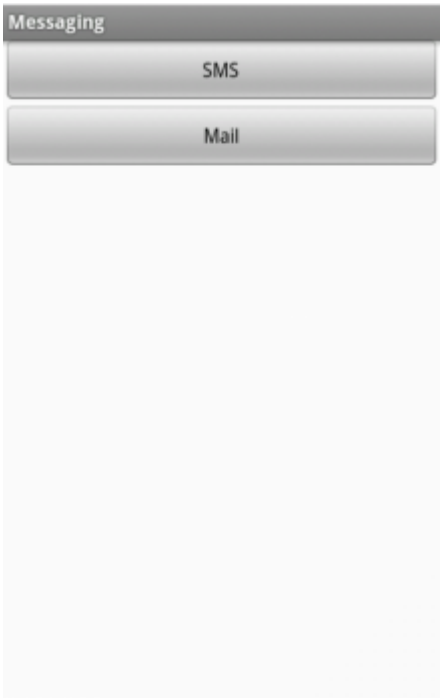
```
    <Button
        android:id="@+id/send_mail"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Mail" />

</LinearLayout>
```

Böylece ekranda **send_sms** ve **send_mail** *id* değeri olan iki adet düğme oluşturmuş olduk. Şimdi **onCreate**metoduna gelerek SMS gönderme amacıyla aşağıdaki kodu yazalım:

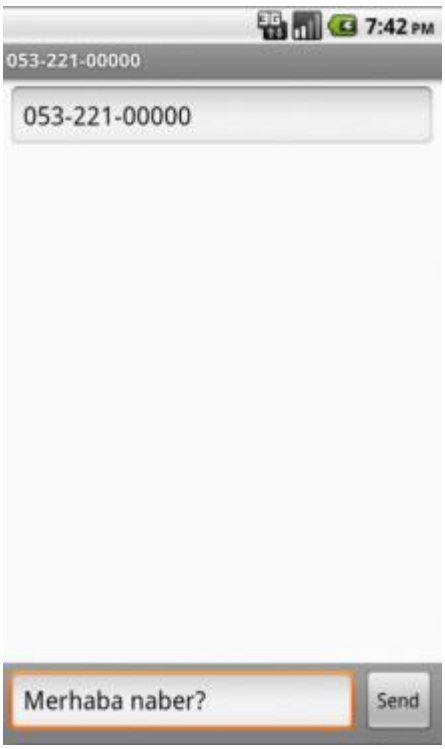
```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);

   ((Button) findViewById(R.id.send_sms)).setOnClickListener(new OnCli
ckListener() {

      @Override
      public void onClick(View v) {
         String smsNumber = "05322100000";
         String smsText = "Merhaba naber?";

         Uri uri = Uri.parse("smsto:" + smsNumber);
         Intent intent = new Intent(Intent.ACTION_SENDTO, uri);
         intent.putExtra("sms_body", smsText);
         startActivity(intent);
      }
   });
}
```

Burada yapılan işlemi açıklarsak, öncelikle **findViewById** metoduyla **send_sms** adlı düğmeye erişiyoruz. Burada daha önce açıklandığı

üzere **setOnClickListener** metoduyla düğmeye basıldığında gerçekleşecek işlemleri belirleyebiliyoruz. **onClick** metodu altında yazdığımız işlemler, kullanıcı düğmeye bastığı takdirde gerçekleşecek işlemlerdir.



onClick altında ise öncelikle **smsNumber** adında bir değişkene mesajı göndereceğimiz numarayı atıyoruz. Biz burada numarayı kodun içerisinde tanımladık ancak bu elbette bir **EditText** üzerinden ya da başka bir kaynaktan gelebilir. Aynı şekilde **smsText** değeri de SMS ile gönderilecek metnin ne olacağını belirtir.**Intent** ile SMS göndermek istediğimizde yaratacağımız **URI** değişkenine değer olarak **smsto:** vermemiz gerekiyor. Bu şekilde işletim sistemi **Intent** ile beraber mesaj atma uygulamasını çalıştırması gerektiğini anlayacaktır. Eğer mesaj uygulamasında gönderilecek telefon numarasının da yer almasını istersek, bu değeri **smsto:*telefon_no*** formatında kullanmamız gerekir. Gönderilecek SMS metnini ise **putExtra** metodu ve **sms_body** anahtarıyla oluşturulacak **Activity**'e gönderiyoruz. **startActivity** metodunu çağırdığımızda atanan değerlerle bir Intent oluşturulacak ve mesaj uygulaması açılacaktır.

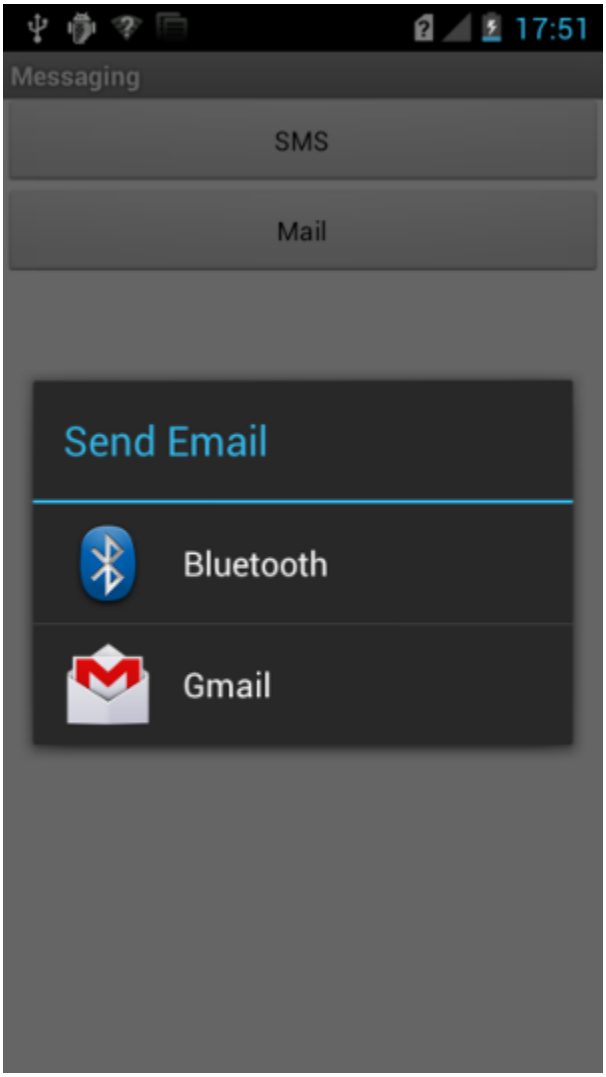


E-posta göndermek için yine benzer şekilde e-posta uygulamasını çağıran bir Intent hazırlamamız gerekiyor. Şimdi **onCreate** metodu içine aşağıdaki kodu ekleyerek devam edelim:

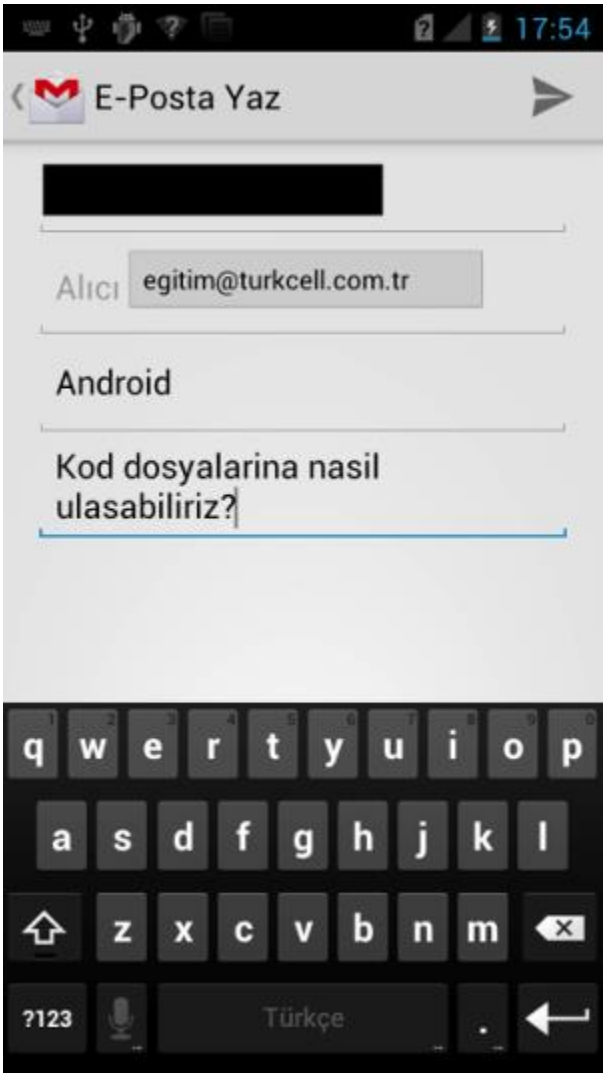
```
((Button) findViewById(R.id.send_mail)).setOnClickListener(new OnClick
Listener() {

   @Override
   public void onClick(View v) {
      Intent intent = new Intent(Intent.ACTION_SEND);
      intent.setType("text/html");
      intent.putExtra(Intent.EXTRA_EMAIL, new String[] { "egitim@turkc
ell.com.tr" });
      intent.putExtra(Intent.EXTRA_SUBJECT, "Android");
      intent.putExtra(Intent.EXTRA_TEXT, "Kod dosyalarina nasil ulasab
iliriz?");
      startActivity(Intent.createChooser(intent, "Send Email"));
   }
});
```

Oluşturulan Intent'e **putExtra** metotlarıyla gönderilecek e-posta ile ilgili çeşitli değişkenleri ekleyebiliriz.**EXTRA_MAIL** değeri alıcının e-posta adresini belirtirken, **EXTRA_SUBJECT** e-postanın konusunu belirlememize yardımcı olur. **EXTRA_TEXT** değeriyse e-posta ile birlikte gönderilecek metni ayarlamanızı sağlar. **EXTRA_MAIL** değeri bir **String** dizisi olduğundan birden fazla kişiye gönderilecek e-postalarda bunları dizi şeklinde belirtebiliriz.

startActivity içerisinde **createChooser** metodu ile bir Intent çağırdığımızda telefonda bu işlem için tanımlı diğer uygulamaların da seçilebileceği bir ekran açılacaktır. Bu şekilde kullanıcı telefonunda yer alan istediği e-posta uygulamasını kullanabilir.



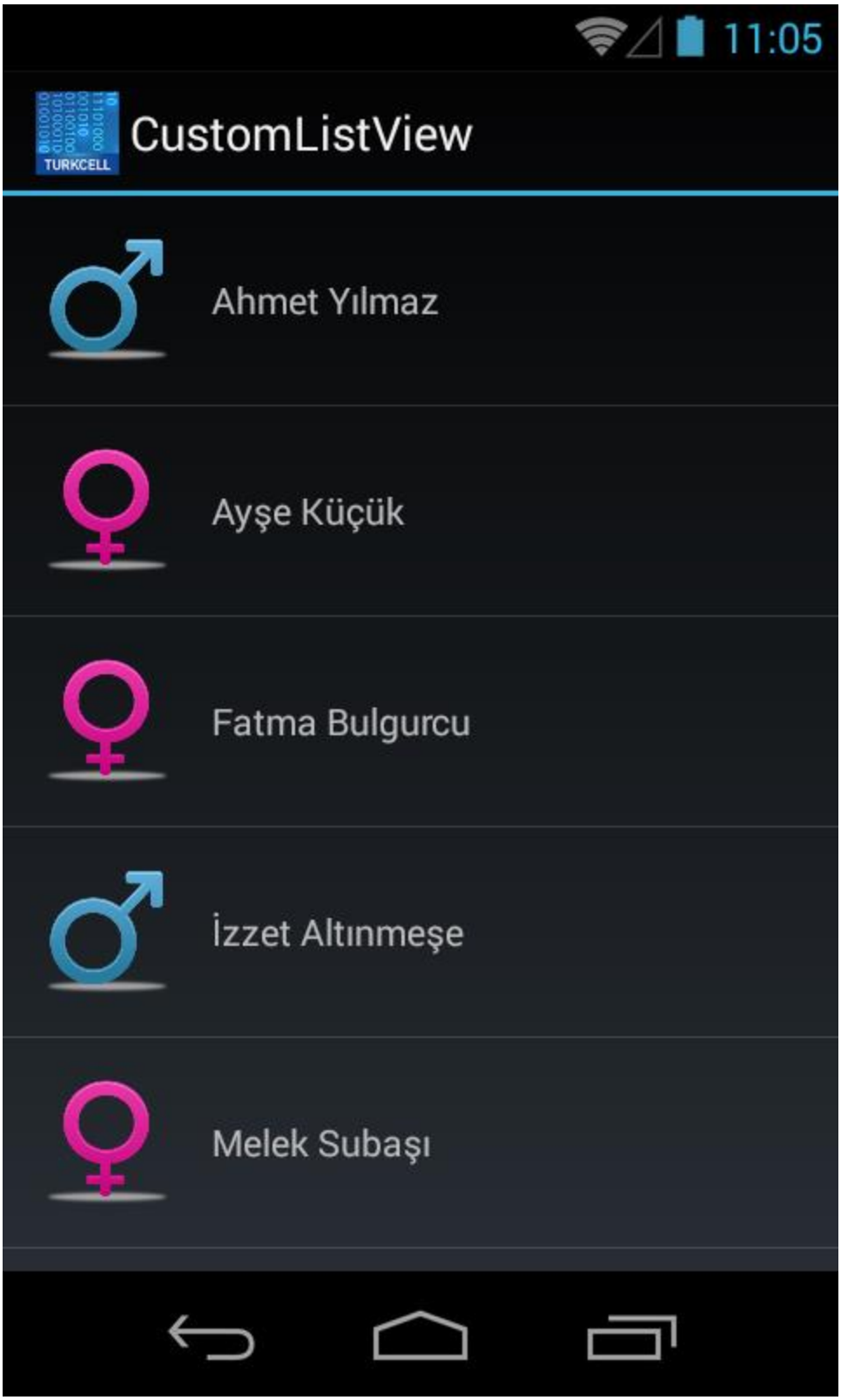
Kullanıcı ilgili uygulamayı seçtiğinde (biz Gmail uygulamasını seçtik) kod içerisinde belirtilen parametrelerle e-posta gönderme ekranı açılacaktır.



ListView Özelleştirme

Bildiğiniz üzere ListView nesnesi, TextView, EditText ya da Button nesneleri gibi basit bir View nesnesi değil,LinearLayout, GridView gibi taşıyıcı olarak görev yapan bir View nesnesidir. Bunun anlamı kendi içinde başka View'ları barındırıyor olduğudur. Önceki derste değinildiği gibi ListView, her satırında bir TextView bulundurur. Bu sıradan bir ListView için geçerli. İsterseniz her satırında daha farklı bir View düzeni kurabilirsiniz.

Bu bölümde kendi düzenimize ve tasarımımıza uygun, özel (Custom) bir ListView oluşturacağız. Bittiğinde şöyle görünecek:



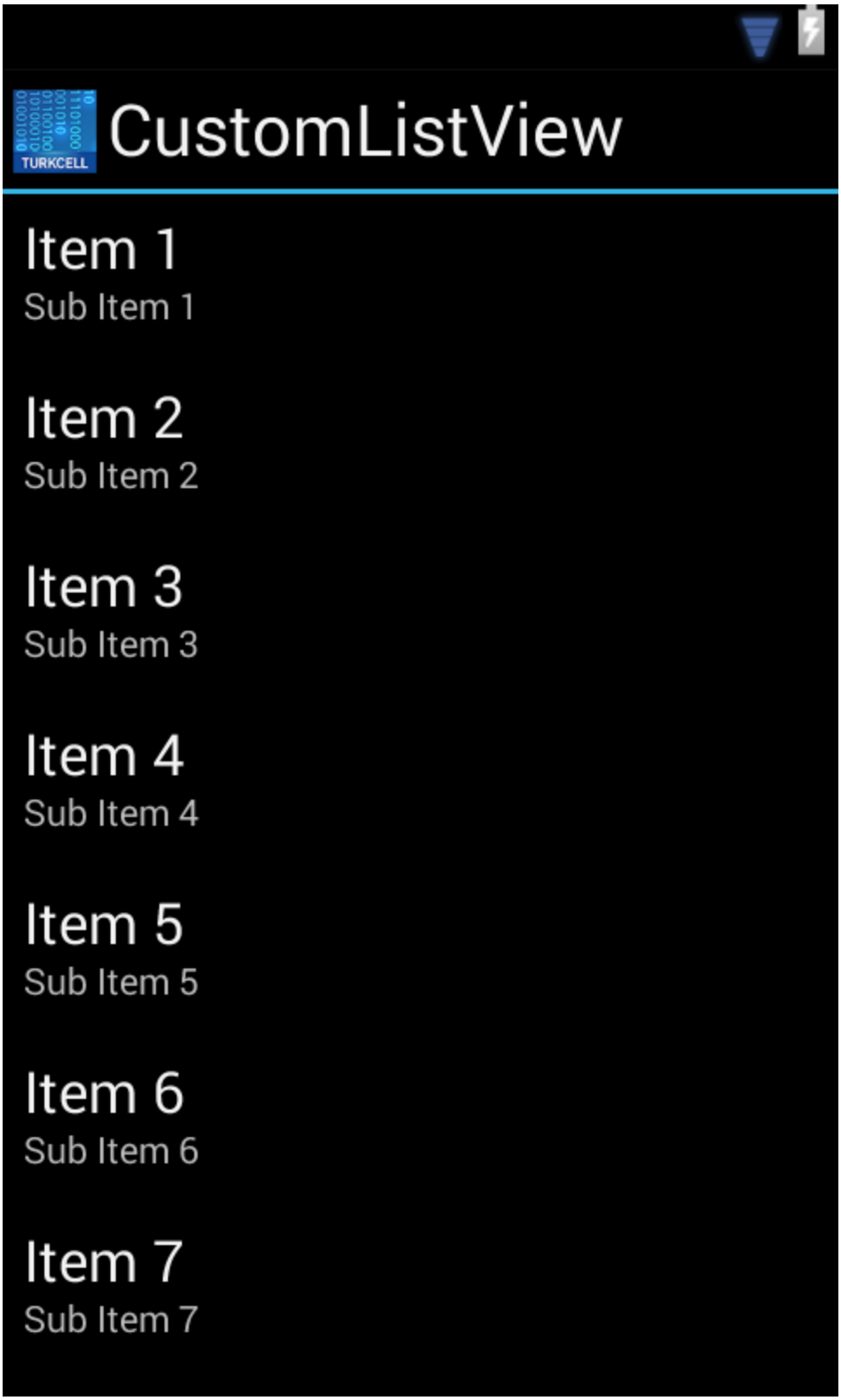
Takip edeceğimiz adımlar şunlar:

Ana yerleşimi tasarlamak

İlk olarak yeni bir proje oluşturalım ve **MainActivity.java** dosyasını ve buna ait **activity_main.xml** isimli layout dosyasını düzenleyelim. MainActivity'nin arayüzünü bir RelativeLayout olarak gösterelim ve içine de *liste* id'sini taşıyan bir ListView ekleyelim:

```
<RelativeLayout 
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

<ListView
    android:id="@+id/liste"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    />

</RelativeLayout>
```

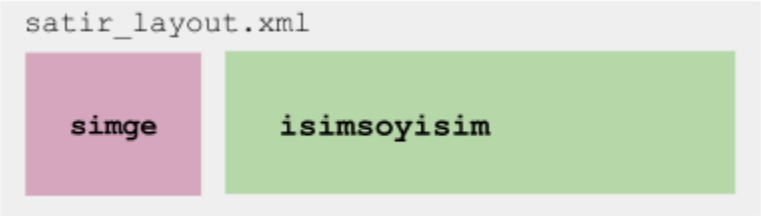
Bu kadarını tamamladığımz *layout* şöyle olacaktır:



Gördüğünüz gibi özelleştirilmemiş bir liste oluştu. Java kodu tarafında listeye erişebilmek için bu öğenin android:id özniteliğindeki liste değerini kullanacağız.

Satır yerleşimini tasarlamak

Android 201'de yer alan önceki örnekten farklı olarak, listedeki satırda TextView ile beraber bir ImageViewgöstereceğiz. Şunun gibi:



Bu yerleşim dosyasını da tıpkı activity_main.xml gibi projenizin **layout/** dizininde tutabilirsiniz.

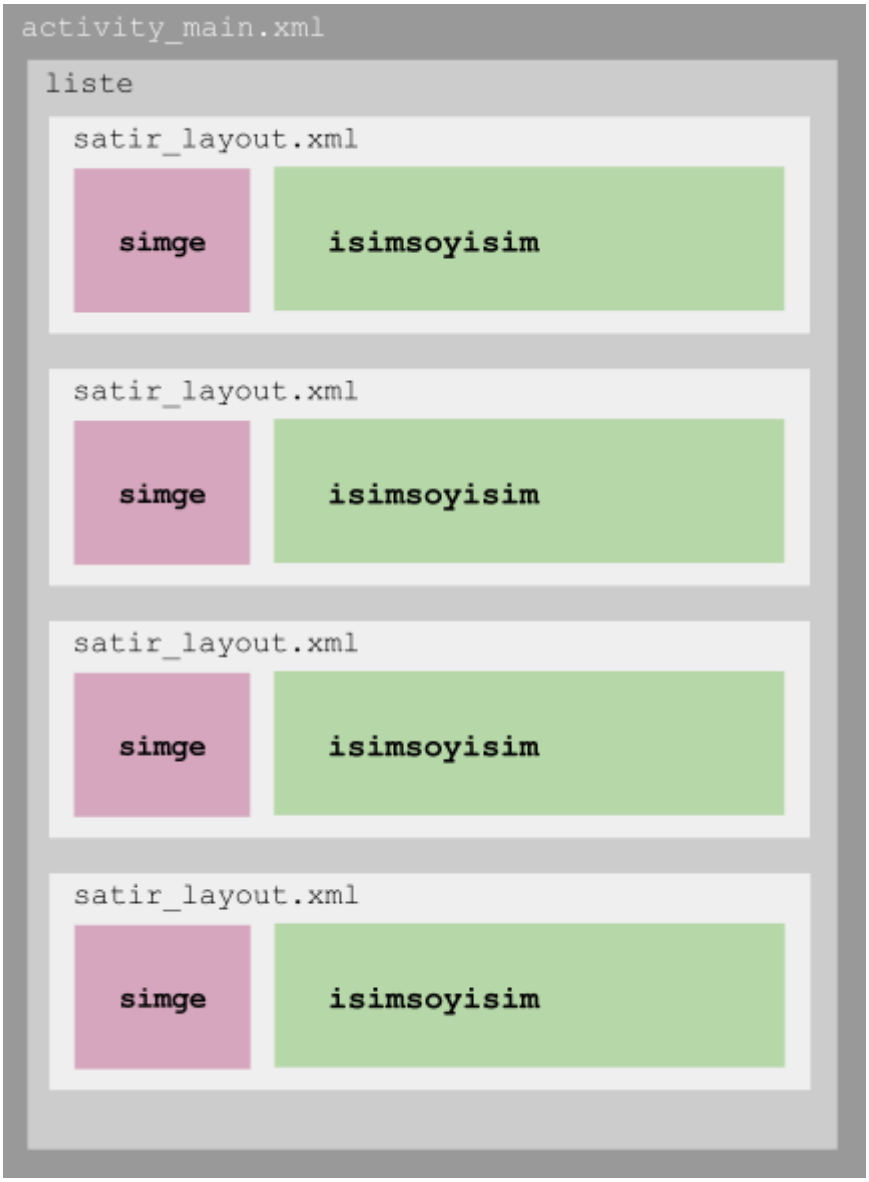
Bu görünümü sağlayacak ***satir_layout.xml*** yerleşim dosyası şöyle:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >

    <ImageView
       android:id="@+id/simge"
       android:layout_width="48dp"
       android:layout_height="48dp" />

    <TextView
       android:id="@+id/isimsoyisim"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content" />

</LinearLayout>
```

Birazdan bu **satir_layout.xml** dosyasını, gösterilecek listenin birer satırı olacak şekilde kullanacağız. Bunun için bir *inflating* yani (dilimize çevirmek gerekirse) şişirme işlemine başvuracağız.

NOT: Bu dosyada LinearLayout'un *android:orientation* özelliğinin *horizontal* olmasına dikkat ediniz. Böylece yavru elementler ImageView ve TextView, LinearLayout içinde yatay sırada konumlanabileceklerdir.

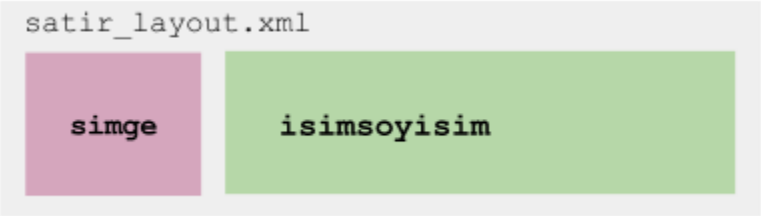
Yapmak istediğimiz yerleşimin son halinin hatları kabataslak şöyle olacaktır:



Veri modelini oluşturmak

Özel ListView'lar oluşturmadaki önemli bir husus, liste satırlarında gösterilecek **veri**lerdir. Android 201'deki örneğimizde ListView'ı doldururken sadece **String** tipli elemanlardan oluşan bir dizi kullanmıştık. Ayrıca ihtiyacımız basitti: verilerimizi (ulkeler) önceden belli bir listeye (android.R.layout.simple_list_item_1), önceden belli bir TextView'ın (android.R.id.text1) içine yerleştirip koymak ve kullanıcıya göstermek. İşte bu alma-gösterme işlemini tam olarak **ArrayAdapter** yapıyordu.
Şimdiki senaryoda verilerimizi, kendi oluşturduğumuz (*activity_main.xml'deki **liste**)* bir listede, kendi oluşturduğumuz bir düzende (Bkz: satir_layout.xml dosyası) göstereceğiz. Bu sebeple kendi Adapter'ımızı yazmak zorundayız. Bir Adapter ile uygun verileri, uygun yerleşimlere bağlayabiliyoruz. (Bkz: Adapter'ın tanımı)

Birer veri olarak ele alıp göstereceğimiz satırın tasarımına tekrar bakalım:



Sol tarafta bir cinsiyet simgesi olacak ve yanında da kişinin adı ve soyadı yazacak. Bu iki veriyi temsil eden bir sınıf yazmalıyız. Nesneye yönelik programlama yaklaşımında böyle sınıflara model sınıfları da denir. Aşağıda örnek bir sınıf görüyorsunuz:

```
package org.gelecegiyazanlar.customlistview;

public class Kisi {
    private String  isim;
    private boolean kadinMi;

    public Kisi(String isim, boolean kadinMi) {
        super();
        this.isim = isim;
        this.kadinMi = kadinMi;
    }

    @Override
    public String toString() {
        return isim;
    }

    public String getIsim() {
        return isim;
    }

    public void setIsim(String isim) {
        this.isim = isim;
    }

    public boolean isKadinMi() {
        return kadinMi;
    }

    public void setKadinMi(boolean kadinMi) {
        this.kadinMi = kadinMi;
    }
```

```
}
```

Burada **isim** alanı ile kullanıcını adını String tipinde tutmayı hedefliyoruz. **KadinMi** isimli boolean alan ile Kisi'nin cinsiyetini tutacağız. Kadınları temsil etmek için bu değeri *true* yapacağız. Nesne için oluşturduğumuz yapıcı (constructor) metot, isim ve cinsiyet değerlerini hızlı bir şekilde atamamızı sağlıyor. Buradaki sınıfa göre kaynak kod içinde **new Kisi("Ahmet Yılmaz", false)** şeklinde bir deyimle nesne oluşturursak, Ahmet Yılmaz adına sahip erkek cinsiyetinde birini temsil eden bir Kisi nesnesi oluşturmuş olacağız.

Listede gösterilecek verileri oluşturmak

Veri modelimize uygun bir sınıf oluşturduğumuza göre listemizde görüntülenecek veri bütününü-listeyi de oluşturabiliriz. Bunun için içini Kisi nesneleriyle dolduracağımız bir ArrayList yeterli gelecektir. Hemen MainActivity dosyamızı açalım ve bir ArrayList oluşturalım:

```
public class MainActivity extends Activity {
    final List<Kisi> kisiler=new ArrayList<Kisi>();

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

        kisiler.add(new Kisi("Ahmet Yılmaz", false));
        kisiler.add(new Kisi("Ayşe Küçük", true));
        kisiler.add(new Kisi("Fatma Bulgurcu", true));
        kisiler.add(new Kisi("İzzet Altınmeşe", false));
        kisiler.add(new Kisi("Melek Subaşı", true));
        kisiler.add(new Kisi("Selim Serdilli",false));
        kisiler.add(new Kisi("Halil İbrahim",false));
    }
}
```

kisiler isimli ArrayList'imizi MainActivity sınıfının bir alanı (field) şeklinde tanımladık. İsterseniz onCreate()metodunda da tanımlayabilirsiniz. **final** tanımlama yapmak da size kalmış. Bir kere oluşturup değer verdikten sonra yeniden bir atama (ArrayList'ten başka bir türe vs) yapmayacağımız için final olarak belirliyoruz.

Özel bir Adapter oluşturmak

Geliyoruz en önemli kısma. ListView'ı verilerle doldurabilmek için bir Adapter oluşturmamız gerekir. Kendi Adapter nesnemizi oluşturabilmek için Android SDK'de yer

alan **BaseAdapter** sınıfını temel sınıf olarak kullanabiliriz. Bu sınıftan *türetilen* sınıfların sahip olması/ezmesi gereken dört metot vardır:

- **getCount()**: int değer döner. ListView'da gösterilecek satır sayısını ifade eder. Verilerimizi barındıran ArrayList'in boyutu (size()) burada bize yarayacak.
- **getItem(int position)**: Object değer döner. position ile belirtilen satıra denk düşen nesneyi döndürür. Bu nesne satır olarak gösterilecek nesnedir. Bundan dolayı Object yerine doğrudan model sınıfınızdan oluştuğunuz nesneyi de dönüş türü olarak belirleyebilirsiniz.
- **getItemId(int position)**: long değer döndürmelidir. Veri listesinde position ile sırası belirtilen satırın kimlik numarasını (id) döndürür. Liste içeriğini veri tabanına kaydedecekseniz ya da orada eşlemeler yapacaksanız önem kazanır.
- **getView(int position, View convertView, ViewGroup parent)**: View değer döner. position ile sırası belirtilen satır için bir **View** döndürür. Bu metot içindeyken her satır için XML'i okuyup View haline getirme işlemi (inflating) yaparız. Bu hususta bize LayoutInflater servisi yardımcı olacaktır.

Örnek için oluşturduğumuz Adapter şöyle (MainActivity.java ile aynı dizinde yer alabilir):

```
public class OzelAdapter extends BaseAdapter {

    private LayoutInflater mInflater;
    private List<Kisi>     mKisiListesi;

    public OzelAdapter(Activity activity, List<Kisi> kisiler) {
        //XML'i alıp View'a çevirecek inflater'ı örnekleyelim
        mInflater = (LayoutInflater) activity.getSystemService(
                Context.LAYOUT_INFLATER_SERVICE);
        //gösterilecek listeyi de alalım
        mKisiListesi = kisiler;
    }

    @Override
    public int getCount() {
        return mKisiListesi.size();
    }

    @Override
    public Kisi getItem(int position) {
        //şöyle de olabilir: public Object getItem(int position)
        return mKisiListesi.get(position);
    }

    @Override
    public long getItemId(int position) {
```

```
        return position;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup
parent) {
        View satirView;

        satirView = mInflater.inflate(R.layout.satir_layout, null);
        TextView textView =
                (TextView) satirView.findViewById(R.id.isimsoyisim);
        ImageView imageView =
                (ImageView) satirView.findViewById(R.id.simge);

        Kisi kisi = mKisiListesi.get(position);

        textView.setText(kisi.getIsim());

        if (kisi.isKadinMi()) {
            imageView.setImageResource(R.drawable.kadin_simge);
        }
        else {
            imageView.setImageResource(R.drawable.adam_simge);
        }
        return satirView;
    }
}
```

İncelemeye yapılandırıcı metottan başlayalım:

```
public OzelAdapter(Activity activity, List<Kisi> kisiler) {
    //XML'i alıp View'a çevirecek inflater'ı örnekleyelim
    mInflater = (LayoutInflater) activity.getSystemService(
            Context.LAYOUT_INFLATER_SERVICE);
    //gösterilecek listeyi de alalım
    mKisiListesi = kisiler;
}
```

Yapılandırıcı metodumuzun ilk parametresi, Adapter'ın bağlı bulunacağı uygulama parçasıyla (ListView) ilgili: Context yani Bağlam. Böylece OzelAdapter'a çalışacağı ortamla ilgili temel bilgileri bu parametreyle geçirmiş oluyoruz.

List<Kisi> kisiler parametresi de adaptörümüzün ihtiyaç duyacağı verileri alacağımız yer. *<Kisi>* ifadesiyle içinde sadece Kisi bulunduran List'i kabul ettiğimizi belirtiyoruz. Verileri alma işlemini *mKisiListesi = kisiler* ifadesiyle hallediyoruz.

mInflater değişkenine activity üzerinden bir sistem servisini referans gösteriyoruz: Layout Inflater Service. Bu servis, yerleşimleri kullanıcıya gösterebilmek için onları önce XML'den okuyup ardından View'a çevirme işlemini yapar.

getCount() ve *getItemId()* metotları, yukarıda tanımladığımız işlemleri yapıyor. *getItem()* metodu da aynı şekilde mKisiListesi'nin position indisine sahip nesneyi döndürüyor.

Önemli bir metot olan **getView()**'a göz atalım:

```
@Override
public View getView(int position, View convertView, ViewGroup parent)
{
    View satirView;

    satirView = mInflater.inflate(R.layout.satir_layout, null);
    TextView textView =
            (TextView) satirView.findViewById(R.id.isimsoyisim);
    ImageView imageView =
            (ImageView) satirView.findViewById(R.id.simge);

    Kisi kisi = mKisiListesi.get(position);

    textView.setText(kisi.getIsim());

    if (kisi.isKadinMi()) {
        imageView.setImageResource(R.drawable.kadin_simge);
    }
    else {
        imageView.setImageResource(R.drawable.adam_simge);
    }
    return satirView;
}
```

Bu metodun satır olarak gösterilecek View'ı döndürdüğüne yukarıda değinmiştik. Bu amaçla satirView adında bir View oluşturuyoruz. Sonra bu View'a

```
mInflater.inflate(R.layout.satir_layout, null);
```

ile *satir_layout.xml* dosyasındaki yerleşimi veriyoruz.
Böylece *satirView* üzerinden *satir_layout.xml*'deki öğelere erişebiliyoruz:

```
TextView textView =
            (TextView) satirView.findViewById(R.id.isimsoyisim);
ImageView imageView =
            (ImageView) satirView.findViewById(R.id.simge);
```

Dikkat ederseniz findViewById() metodunu satirView üzerinde çalıştıyoruz.

Ardından View olarak döndürülecek satır için Kisi nesnesini
yine position indisiyle mKisiListesi üzerinden alıyoruz ve kisi değişkenine atıyoruz.

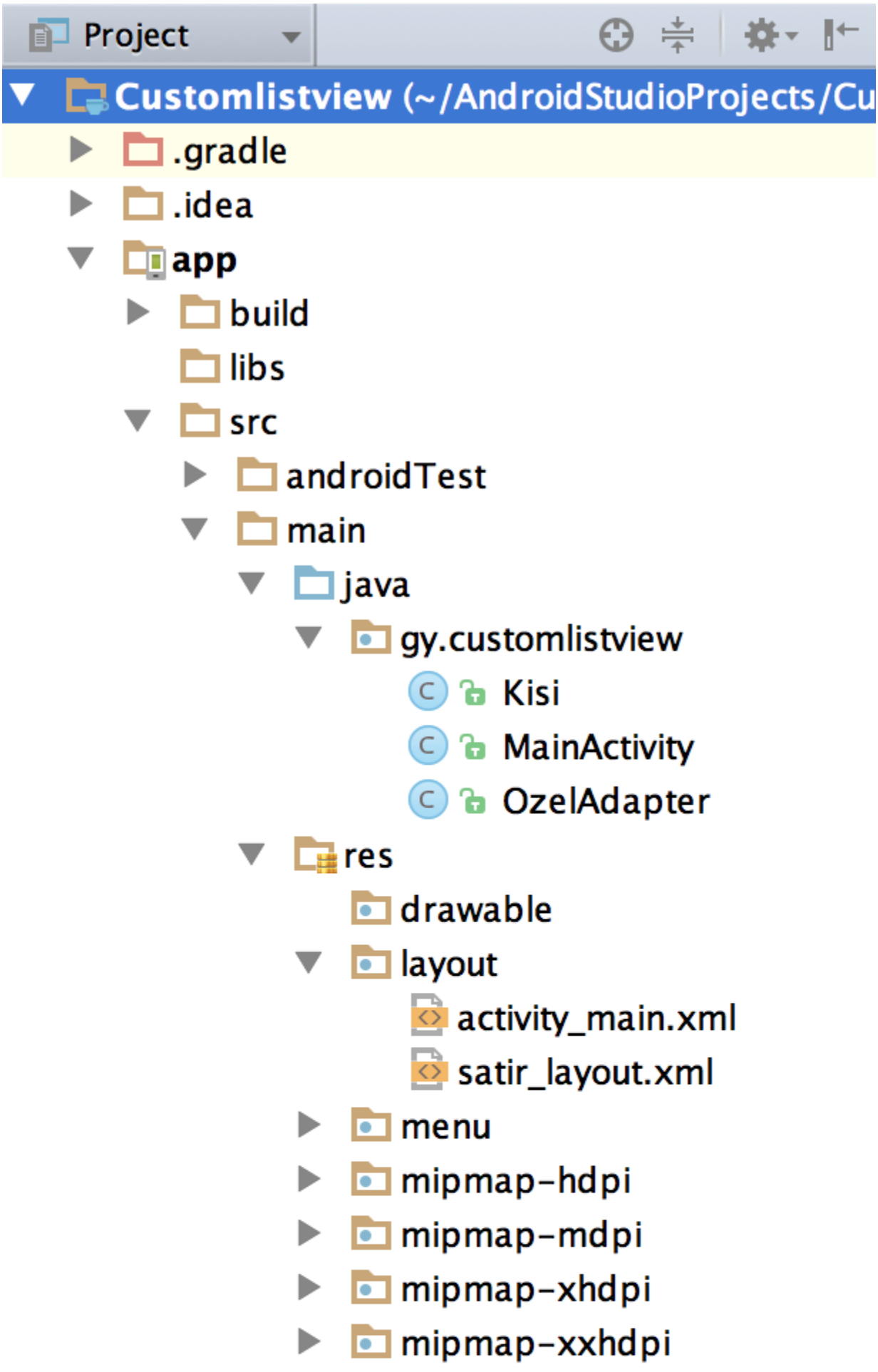
Satırdaki ImageView'da cinsiyet simgesini gösterebilmek için önce kisi referansı üzerinden nesnenin isKadinMi() metoduna erişiyoruz:

```
if (kisi.isKadinMi()) {
    imageView.setImageResource(R.drawable.kadin_simge);
}
    else {
    imageView.setImageResource(R.drawable.adam_simge);
}
```

isKadinMi() **true** sonuç dönerse ImageView'a kadın simgesini, **false** dönerse erkek simgesini veriyoruz.

Tüm bu işlemlerin ardından satirView hazır oluyor ve onu return ediyoruz.

Buraya kadar oluşturduklarımızla beraber projemiz Android Studio'nun package explorer bileşeninde şöyle yer edinecektir:


Project
Customlistview (~/AndroidStudioProjects/Cu
.gradle
.idea
app
build
libs
src
androidTest
main
java
gy.customlistview
Kisi
MainActivity
OzelAdapter
res
drawable
layout
activity_main.xml
satir_layout.xml
menu
mipmap-hdpi
mipmap-mdpi
mipmap-xhdpi
mipmap-xxhdpi

Listeyle adaptörü bağlamak

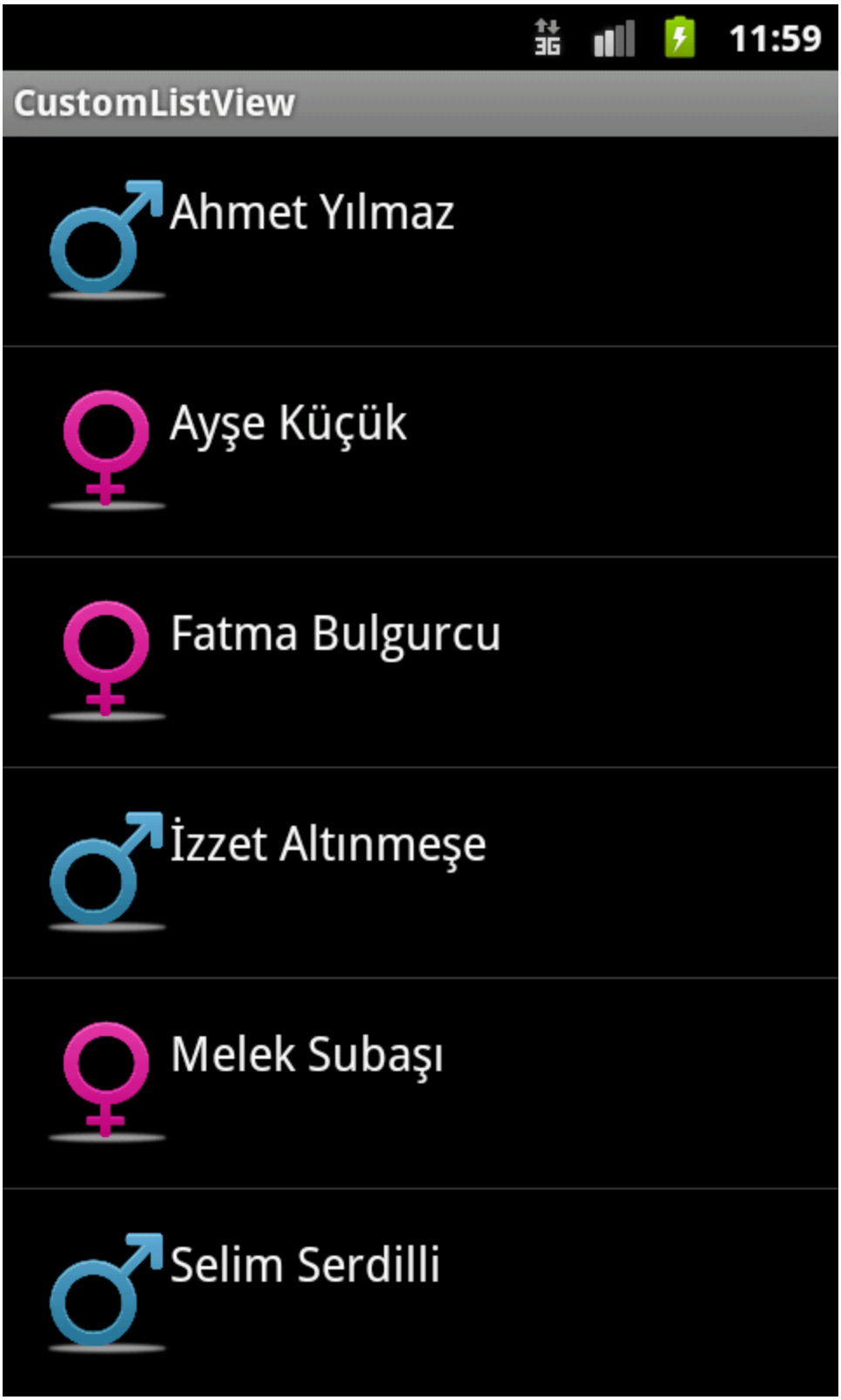
Yukarıda kisiler ArrayList'ini hazırlamıştık fakat onu adaptörümüze göndermemiştik. Artık verileri kendi OzelAdapter sınıfımıza gönderebiliriz.

Önce ListView'ımızı ana yerleşim dosyasından alalım. Ardından da OzelAdapter sınıfından bir nesne oluşturalım. Sonra da listenin adaptörünü gösterelim:

```
final ListView listemiz = (ListView) findViewById(R.id.liste);
    OzelAdapter adaptorumuz=new OzelAdapter(this, kisiler);
    listemiz.setAdapter(adaptorumuz);
```

OzelAdapter sınıfından nesne oluştururken parametre olarak this ve kisiler geçiyoruz. Buradaki **this**, MainActivity'de olduğumuz için MainActivity'yi işaret ediyor. Böylece OzelAdapter'a çalışacağı *bağlam* konusunda bir parametre geçmiş oluyoruz. Hatırlayın: Adapter, veri ile arayüz arasında bir köprü idi. Bu iki parametreyle ona veriye (*kisiler* ArrayList'i) ve arayüze (*listemiz*) ilişkin parametre geçmiş oluyoruz.

Özel bir ListView oluşturduk. Görselleri de bulunduran bir örnek için sayfadaki projenin kaynak kodlarını indirebilirsiniz. Yukarıda Android 4.0 üzerinde ekran görüntüsünü verdiğimiz örnek Android 2.3.5 üzerinde şöyle görünecektir:


11:59
CustomListView
Ahmet Yılmaz
Ayşe Küçük
Fatma Bulgurcu
İzzet Altınmeşe
Melek Subaşı
Selim Serdilli

Verileri Kaydetmek

Çoğu Android uygulaması, kullanıcısının ilerlemesi kaybolmasın diye onPause() olayları sırasında uygulama durumuna ilişkin bilgileri kaydediyor olsa bile veri kaydetme ihtiyacı duyar. Anlaşılması zor olan çoğu uygulama da kullanıcı ayarlarını kaydetmeye ihtiyaç duyar. Bazı uygulamalar büyük miktarda bilgiyi dosyalarla ve veritabanlarıyla yönetmek zorundadır. Bu ve sonraki birkaç eğitim içeriği, size Android'teki başlıca veri depolama seçeneklerini gösterecektir:

- Basit veri tiplerini anahtar-değer çiftleri şeklinde "shared preferences" dosyasına kaydetmek
- İstediğiniz şekilde dosyayı Android'in dosya sistemine kaydetmek
- SQLite ile yönetilen veritabanlarını kullanmak

Bu doğrultuda sıradaki eğitim içerikleri şunlar olacak:

- Anahtar-Değer Çiftlerini Kaydetmek: Küçük boyutlardaki verileri anahtar-değer çiftleri şeklinde kaydetmeye yarayan "shared preferences" dosyalarını nasıl kullanacağınızı burada öğrenebilirsiniz.
- Dosyaları Kaydetmek: Genellikle sıra sıra okunan uzun veri parçalarını basit dosyalar şeklinde nasıl kaydedeceğinizi burada öğrenebilirsiniz.
- Verileri SQL Veritabanına Kaydetmek: Yapılandırılmış veriyi yazmak ve okumak için bir SQLite veritabanını nasıl kullanacağınızı öğrenebilirsiniz.

Dosyaları Kaydetmek

Android diğer platformlardaki dosya sistemlerine benzer şekilde disk temelli bir dosya sistemi kullanır. Bu eğitim içeriğinde File API'lerini kullanarak Android dosya sisteminde dosya okumayı ve dosyaya yazmayı nasıl yapacağınızı bulabilirsiniz.

Bir File nesnesi başından sonuna büyük boyutlu verileri okumak veya yazmak için ideal metotlar sunar. Örneğin resim dosyaları veya ağ üzerinde taşınabilen her şey için idealdir.

Bu eğitim içeriğinden uygulamanızda dosyalarla ilişkili temel görevleri gerçekleştirmeyi öğrenebilirsiniz. Bu sırada Linux dosya sisteminin temellerine ve java.io kitaplığındaki standart dosya giriş/çıkış API'lerine aşina olduğunuzu varsayıyoruz. Kafanıza takılan bir konu olduğunda soru-cevap bölümünü kullanabilirsiniz.

Dâhili ya da harici depolama alanını seçmek

Her Android cihazının iki dosya depolama alanı vardır: "dâhili" ve "harici". Bu isimler Android'in ilk günlerinden, çoğu cihazın yerleşik olarak gelen kalıcı hafızasından (**dâhili**) ve buna ek olarak çıkarılabilir mikro SD kart (**harici**) gibi depolama alanlarından geliyor. Bazı cihaz üreticileri, cihazlarının çıkarılabilir depolama alanı olmamasına rağmen kalıcı depolama alanlarını "dâhili" ve "harici" olarak bölümlemeyi tercih ediyor. Böylece gerçekten harici depolama alanı çıkarılmış olsun ya da hiç olmasın, her zaman iki depolama alanı oluyor ve API'nin davranışı da değişmiyor. Şu iki liste her depolama alanı türünü özetlemeye yetecektir:

Dâhili Depolama Alanı	Harici Depolama Alanı
Her zaman kullanılabilir.	Her zaman kullanılabilir değildir; kullanıcı USB depolama aygıtı şeklinde bilgisayarına bağlamış (mount) olabilir veya bazı durumlarda doğrudan cihazla olan bağlantısını kaldırabilir.
Buraya kaydedilen dosyalar varsayılan olarak sadece uygulamanız tarafından erişilebilir durumdadır.	Herkesçe okunabilir. Bir başka deyişle, buraya kaydedilen dosyalar kontrolünüz dışında okunabilir.
Kullanıcı uygulamanızı cihazından kaldırdığında, sistem uygulamanızla ilgili tüm dosyaları dâhili depolama alanından siler.	Kullanıcı uygulamanızı cihazından kaldırdığında, sistem uygulamanızla ilgili dosyaları eğer onlarıgetExternalFilesDir() metodunun döndürdüğü dizine kaydettiyseniz silecektir.

Harici depolama alanı erişim kısıtlaması gerektirmeyen diğer uygulamalarla paylaşmak istediğiniz veya kullanıcının bilgisayarıyla erişmesine izin verdiğiniz dosyalar için en iyi yerdir.

***ÖNERİ**: Uygulamalar varsayılan olarak dâhili depolama alanına kuruluyor olsa da, manifest dosyanızda <application> elementinin android:installLocation özniteliğine uygun tanımlamayı yaparak uygulamanızın harici depolama alanına kurulmasını da sağlayabiliyorsunuz. Kullanıcılar bu seçeneği uygulamanın APK boyutu çok büyük olduğunda ve harici depolama alanları dâhili olandan daha büyükse takdirle karşılıyorlar.*

Harici depolama alanı için izinlerin alınması

Harici depolama alanına yazabilmek için WRITE_EXTERNAL_STORAGE iznini manifest dosyanızda istemeniz gerekiyor:

```
<manifest ...>
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    ...
</manifest>
```

UYARI: Şu an tüm uygulamalar harici depolama alanını özel bir izin almadan okuma olanağına sahip. Buna rağmen bu durum ileriki sürümlerde değişecektir. Eğer uygulamanız harici depolama alanını okuyacaksa (yazma yapmayacaksa)READ_EXTERNAL_STORAGE iznini beyan etmeniz gerekiyor. Uygulamanızın beklendiği gibi çalışmasına devam ettiğinden emin olmak için bu izni şimdi beyan etmelisiniz:

```
<manifest ...>
    <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
    ...
</manifest>
```

Yine de uygulamanız WRITE_EXTERNAL_STORAGE iznini kullanırsa aynı zamanda harici depolama alanını da okuyabilirsiniz.

Dâhili depolama alanına dosya kaydetmek için herhangi bir izin almanıza gerek yok. Uygulamanız dâhili depolama alanındaki dizinine istediği zaman dosya yazabilir ve o dosyaları okuyabilir.

Bir dosyayı dâhili depolama alanına kaydetmek

Bir dosyayı dâhili depolama alanına kaydederken aşağıdaki metotlardan birini çağırarak uygun dizini File nesnesi şeklinde elde edebilirsiniz:

getFilesDir() - Uygulamanızın dâhili depolama alanındaki dizinini ifade eden bir File nesnesi döndürür.

getCacheDir() - Uygulamanızın dâhili depolama alanındaki geçici dosyalarının saklandığı dizini ifade eden bir File nesnesi döndürür. Buradaki her dosyayı ihtiyacınız kalmadığında sildiğinizden ve 1 MB gibi kabul edilebilir bir boyut sınırıyla kullandığınız yeri sınırladığınızdan emin olmalısınız. Sistemin depolama alanı küçülmeye başladığında sizi uyarmadan buradaki dosyaları silebilir.

Bu dizinlerden birinde yeni bir dosya oluşturmak isterseniz File() yapılandırıcı metodunu kullanabilirsiniz. Bunu yaparken yapılandırıcı metoda dâhili depolama alanını veren yukarıdaki metotlardan birini geçmeniz yeterli. Örneğin:

```
File file = new File(context.getFilesDir(), filename);
```

Alternatif olarak openFileOutput() metodunu çağırabilir ve böylece dâhili depolama alanınıza dosya yazabilecek bir FileOutputStream alabilirsiniz. Aşağıdaki örnekte bazı metinlerin bir dosyaya nasıl yazıldığını görebilirsiniz:

```
String dosyaAdi = "dosyam";
String string = "merhaba herkese!";
FileOutputStream outputStream;

try {
  outputStream = openFileOutput(dosyaAdi, Context.MODE_PRIVATE);
  outputStream.write(string.getBytes());
  outputStream.close();
} catch (Exception e) {
  e.printStackTrace();
}
```

Bunun yerine geçici olan bir önbellek dosyası oluşturmak istiyorsanız createTempFile() metodunu kullanmalısınız. Örneğin aşağıdaki örnek metod, uygulamanızın dâhili depolama alanındaki kendine has önbellek dizininde, ismini URL'den çıkardığı bir dosya oluşturur:

```
public File getTempFile(Context context, String url) {
    File file;
    try {
        String dosyaAdi = Uri.parse(url).getLastPathSegment();
        file = File.createTempFile(dosyaAdi, null,
context.getCacheDir());
    catch (IOException e) {
        // Error while creating file
    }
    return file;
```

```
}
```

NOT: Uygulamanızın dâhili depolama alanındaki dizini, uygulamanızın paket adına özel bir şekilde Android dosya sisteminin özel bir yerinde bulunur. Teknik olarak, eğer dosya modunu "okunabilir" yaptıysanız diğer uygulamalar sizin dâhili dosyalarını okuyabilir. Yine de bunu yapabilmek için paket adınızı ve dosya isimlerini bilmeleri gerekir. Aynı şekilde diğer uygulamalar sizin dâhili dizinlerinize göz atamaz ve siz dosyaları okunabilir veya yazılabilir olarak açıkça belirtmediyseniz okuma ya da yazma yapamazlar. Bundan dolayı dâhili depolama alanındaki dosyalarınız için kullanabildiğiniz kadar MODE_PRIVATE modunu kullanın ki diğer uygulamalar tarafından hiçbir zaman erişilemesinler.

Bir dosyayı harici depolama alanına kaydetmek

Harici depolama alanları her zaman kullanılabilir olamayabileceğinden dolayı -kullanıcı tarafından cihaz bilgisayara bağlanmış olabilir veya SD kart çıkartılmış olabilir- erişmeye çalışmadan önce o bölümün kullanılabilir olduğunu doğrulamanız gerekir.

Harici depolama alanının durumunu getExternalStorageState() metoduyla sorgulayabilirsiniz. Metodun döndürdüğü sonuç MEDIA_MOUNTED değeri olursa bu dosya okuma ve yazma yapabileceğiniz anlamına gelir. Aşağıdaki örnek depolama alanının müsait olup olmadığını sorguluyor:

```
/* harici depolama alanının okuma/yazmaya müsaitliğine bakar */
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}

/* harici depolama alanının sadece okumaya müsaitliğe bakar */
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}
```

Her ne kadar harici depolama alanı kullanıcılar ve diğer uygulamalar tarafından değiştirilebilir durumda olsa da şu iki kategorideki dosyalarınızı buraya kaydedebilirsiniz:

- **Genel dosyalar**: Kullanıcı ve diğer uygulamalar tarafından özgürce kullanılabilecek dosyalar. Kullanıcı uygulamanızı cihazından kaldırsa bile bu dosyalar kullanılmaya devam edilebilir. Örneğin uygulamanız tarafından çekilen fotoğraflar veya İnternet'ten indirilmiş dosyalar bu şekilde sınıflandırılabilir.
- **Özel dosyalar**: Uygulamanıza ait ve uygulamanız kaldırıldığında silinmiş olması gereken dosyalar. Bu dosyalar harici depolama alanında olduğundan teknik olarak kullanıcı ve diğer uygulamalar tarafından erişilebilir olsa da uygulamanız dışında kullanıcı için bir değer ifade etmeyen dosyalardır. Kullanıcı uygulamanızı kaldırdığında sistem harici depolama alanındaki tüm özel dosyaları da siler. Örneğin uygulamanız tarafından indirilmiş uygulamanıza has ek dosyalar veya geçici medya dosyalar.

Genel dosyalarınızı harici depolama alanına kaydetmek isterseniz, getExternalStoragePublicDirectory()metodunu çağırarak uygun dizini ifade eden bir File nesnesi elde edebilirsiniz. Bu metod kaydetmeyi istediğiniz dosyanın tipiyle ilgili bir parametre alır. Böylece diğer genel dosyalarla birlikte mantığa uygun bir şekilde (DIRECTORY_MUSIC veya DIRECTORY_PICTURES gibi) organize edilirler. Örnek:

```
public File getAlbumStorageDir(String albumName) {
    // kullanıcının genel resimler dizini için kullanılan dizini
alalım
    File file = new
File(Environment.getExternalStoragePublicDirectory(
            Environment.DIRECTORY_PICTURES), albumName);
    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "dizin oluşturulamadı");
    }
    return file;
}
```

Uygulamanız için özel dosyaları kaydetmek istiyorsanız getExternalFilesDir() metodunu çağırarak uygun dizini alabilir ve istediğiniz dizinin tipini belirten bir ismi de parametre olarak geçebilirsiniz. Bu yolla oluşturulan her dizin, kullanıcı uygulamayı kaldırdığında silinen ve uygulamanın harici depolama alanındaki tüm dosyalarını taşıyan bir tepe dizine eklenir.

Aşağıdaki örnek metodu bireysel fotoğraf albümü için kullanılabilecek bir dizin oluşturmakta kullanabilirsiniz:

```
public File getAlbumStorageDir(Context context, String albumName) {
```

```
    // uygulamamızda kullanmak üzere özel resim dizini için bir dizin
alalım
    File file = new File(context.getExternalFilesDir(
            Environment.DIRECTORY_PICTURES), albumName);
    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "dizin oluşturalamadı");
    }
    return file;
}
```

Eğer dosyalarınız için önceden tanımlı uygun bir alt dizin ismi yoksa getExternalFilesDir() metodunu "null" değer geçerek çağırabilirsiniz. Bu işlem size uygulamanızın harici depolama alanında yer alan özel dizinini döndürecektir.

Şunu unutmamalısınız: getExternalFilesDir() metodu, bir dizin oluşturur ki o dizin kullanıcı uygulamayı kaldırdığında silinecek bir dizinin içinde oluşturulur. Eğer buradaki dosyaların kullanıcı uygulamanızı kaldırdığında bile kalmasını istiyorsanız (örneğin, uygulamanız bir kamera uygulamasıdır ve çekilen fotoğrafların kalması gerekir), bu metod yerine getExternalStoragePublicDirectory() metodunu kullanmalısınız.

İster ortak dosyalar için getExternalStoragePublicDirectory() metodunu kullanmış olun, ister uygulamanıza özel dosyalar için getExternalFilesDir() metodunu kullanmış olun, kullanacağınız dizin isimlerininDIRECTORY_PICTURES gibi API sabitleri tarafından sağlanacağını unutmayın. Bu dizin isimleri, içindeki dosyaların sistem tarafından doğru şekilde muamele görmesini sağlar. Örneğin DIRECTORY_RINGTONESdizinine kaydedilen dosyalar, sistemin medya tarayıcısı tarafından müzik yerine zil sesi olarak kategorize edilir. Dolayısıyla buraya kaydedeceğiniz dosyalar zil sesi dosyası olmalıdır ki kullanıcı cihazını amacına uygun dosyalarla kullanabilsin.

Boş alanı sorgulamak

Kaydedeceğiniz verilerin ne kadar yer tuttuğunu erkenden öğrenmek isterseniz, hiç IOException oluşmasına meydan vermeden getFreeSpace() veya getTotalSpace() metotlarını kullanarak yeterli alan olup olmadığına bakabilirsiniz. Bu metotlar sırasıyla, geçerli depolama biriminde ne kadar boş alan kaldığı bilgisini ve alanın toplam boyutu bilgisini verir. Bu bilgiler depolama alanını belli bir eşiğin üstünde tıka basa doldurmaktan kaçınmak için de yardımcıdırlar.

Ancak şunu unutmamak gerekir ki getFreeSpace() metodunun size döndürdüğü yer kadar byte'ı yazmak için kullanabileceğinizin garantisini sistem veremez. Eğer döndürülen MB değeri kaydetmek istediğinizden fazlaysa veya dosya sisteminin doluluk oranı %90'dan azsa muhtemelen işlem sorunsuzca gerçekleşir. Öteki türlü muhtemelen depolama alanına yazamayacaksınız.

NOT: Dosya kaydetmeden önce ne kadar boş alan kaldığını denetlemeniz gerekmiyor. Bunun yerine dosyanızı hemen kaydetmeyi deneyin ve bir sorun çıkarsa bir IOException hatası fırlatın. Eğer gerçekten ne kadar alan kaldığınızı bilmenize gerek yoksa bunu böyle yapın. Örneğin bir resim dosyanızın olduğunu varsayın ve kodlamasını PNG'den JPEG'e çevireceksiniz. Böyle bir işlemde önceden dosyanın boyutunu bilmenize gerek olmayacaktır.

Bir dosyayı silmek

İhtiyacınız kalmayan dosyaları silmeye daima özen göstermelisiniz. Bir dosyayı silmenin en pratik yolu, açılan dosyanın referans değişkeni üzerinden delete() metodunu çağırmaktır.

```
dosyaAdi.delete();
```

Silinecek dosya dâhili depolama alanına kaydedilmişse, Context sınıfından yararlanarak yerini öğrenebilir ve deleteFile() metodunu kullanarak silebilirsiniz.

```
mContext.deleteFile(dosyaAdi);
```

Verileri SQL Veritabanına Kaydetmek

Sürekli tekrarlayan veya kişi bilgileri gibi belli bir yapısı olan verileri, veritabanına kaydetmek iyi bir karardır. Veritabanı sayesinde verileriniz üzerinde şartlı işlemler yapabilir, onları daha iyi yönetebilirsiniz. Bu eğitim içeriğinde SQL veritabanlarının geneline aşina olduğunuzu varsayıyoruz. Android üzerinde SQLite veritabanlarını öğrenmeye başlarken bu aşinalık size yardımcı olacaktır.

- Şemayı ve kontratı tanımlamak
- SQL Helper kullanarak veritabanı oluşturmak
- Veritabanına bilgileri kaydetmek

- Veritabanından bilgileri okumak
- Veritabanından bilgileri silmek
- Veritabanında güncelleme yapmak

Android üzerinde SQLite veritabanı için ihtiyacınız olan API'ler android.database.sqlite paketi içinde olacaktır.

Şemayı ve kontratı tanımlamak

SQL veri tabanlarının ana ilkelerinden biri de **şema**dır. Şema, veritabanının nasıl organize edildiğinin biçimsel bir açıklamasıdır. Şema, veritabanınızı oluşturmakta kullanacağınız SQL ifadelerinden oluşan bir yansımadır. Şemanızın yapısını sistematik bir yolla ifade eden bir "kontrat" sınıfı yazmak bu konuda yararınıza olacaktır.

Bir **kontrat sınıfı*** tablolar, sütunlar ve URI'ler için isim tanımlamaları yapan sabitleri bulunduran bir**taşıyıcı** gibidir. Kontrat sınıfı* sayesinde aynı sabitleri uygulama paketiniz içindeki diğer sınıflarda da kullanabilirsiniz. Bu size örneğin bir sütun ismini değiştirdiğinizde bunu tüm kodunuzda değiştirmenize gerek vermez ve zahmetsizce kullanabilmenizi sağlar.

Kontrat sınıfı oluşturmanın iyi bir yolu da veritabanınızın global tanımlamalarını sınıf seviyesinde değişkenler şeklinde koymaktır. Ardından her tablo için bir dâhili (inner) sınıf oluşturmak ve bu sınıflara sütun isimlerine karşılık gelecek şekilde numaralandırma yapmak gayet iyi bir kullanım şeklidir. Bunun ayrıntılarını belgenin ilerleyen bölümlerinde görebilirsiniz.

***NOT**: BaseColumns arayüzünü (interface) gerçeklerseniz (implementation), dâhili sınıflarınızın **_ID** olarak anılan birincil anahtar alanlarını miras almasını sağlayabilirsiniz. Android'in cursor adapter gibi bazı sınıfları **_ID** gibi alanlar ile çalıştığından bu uyumluluk açısından size kolaylık sağlar. Elbette şart değildir, ancak veritabanınızın Android geliştirme çatısı (framework) ile uyumlu çalışması epey işinize gelecektir.*

Aşağıdaki örnekte, tek bir tablonun adını ve sütun adlarını tanımlayan bir örnek kod görüyorsunuz:

```
public final class FeedReaderContract {
    // birinin bu sınıfı yanlışlıklar örneklemesinin önüne geçmek için
yapılandırıcı metodu boş bırakıyoruz
    public FeedReaderContract() {}

    /* tablo içeriklerini tanımlayan dahili sınıflar */
    public static abstract class FeedEntry implements BaseColumns {
        public static final String TABLE_NAME = "entry";
```

```
        public static final String COLUMN_NAME_ENTRY_ID = "entryid";
        public static final String COLUMN_NAME_TITLE = "title";
        public static final String COLUMN_NAME_SUBTITLE = "subtitle";
        ...
    }
}
```

SQL Helper kullanarak veritabanı oluşturmak

Veritabanınızın nasıl olacağını tanımladığınıza göre artık veritabanınızı (tablolarını) oluşturacak ve onun çalışmasını sürdürecek metotları da gerçeklemelisiniz. Aşağıda bu amaçla yazılmış tipik SQL ifadelerini görebilirsiniz. Bir tablo oluşturan ve eğer o tablo varsa önce kaldıran iki String ifadesi oluşturuluyor:

```
private static final String TEXT_TYPE = " TEXT";
private static final String COMMA_SEP = ",";
private static final String SQL_CREATE_ENTRIES =
    "CREATE TABLE " + FeedEntry.TABLE_NAME + " (" +
    FeedEntry._ID + " INTEGER PRIMARY KEY," +
    FeedEntry.COLUMN_NAME_ENTRY_ID + TEXT_TYPE + COMMA_SEP +
    FeedEntry.COLUMN_NAME_TITLE + TEXT_TYPE + COMMA_SEP +
    ... // CREATE komutuyla ilgili diğer seçenekler
    " )";

private static final String SQL_DELETE_ENTRIES =
    "DROP TABLE IF EXISTS " + FeedEntry.TABLE_NAME;
```

Tıpkı cihazın dâhili depolama alanına kaydettiğiniz dosyalar için olduğu gibi veritabanlarınız da uygulamayla ilişkili özel bir disk alanına kaydedilir. Verileriniz güvenli bir yerdedir çünkü bu alan diğer uygulamalar tarafından erişilebilir bir konumda değildir.

Veritabanı işleri için kullanışlı ve işinizi hızlandıracak bir API serisi SQLiteOpenHelper sınıfı içinde bulunuyor. Veritabanınız için bu sınıftan bir referans aldığınızda yani onu kullanmaya başladığınızda, sistem muhtemelen uzun sürecek veritabanı oluşturma ve güncelleme gibi işlemleri sadece ihtiyaç olduğunda gerçekleştirecektir, uygulama başlangıcında değil. Tüm ihtiyacınız olan şey, getWritableDatabase() veyagetReadableDatabase() metotlarını çağırmak.

***NOT**: Uzun süren işlemler yaptıkları için getWritableDatabase() veya getReadableDatabase() metotlarını arkaplandaki bir thread'te (AsyncTask veya IntentService ile) çalıştırdığınızdan emin olmalısınız.*

SQLiteOpenHelper'ı kullanmak için onCreate(), onUpgrade() ve onOpen() gibi callback metotlarını ezen (override) bir alt sınıf oluşturmalısınız. onDownGrade() metodunu da gerçeklemek isteyebilirsiniz ancak gerek olmayacaktır.

Aşağıdaki örnekte SQLiteOpenHelper'ın temel komutlarını kullanan bir sınıfı görebilirsiniz:

```
public class FeedReaderDbHelper extends SQLiteOpenHelper {
// eğer veritabanı şemanızı değiştirirseniz bu sürüm değişkenini
mutlaka artırmalısınız
   public static final int DATABASE_VERSION = 1;
   public static final String DATABASE_NAME = "FeedReader.db";

   public FeedReaderDbHelper(Context context) {
       super(context, DATABASE_NAME, null, DATABASE_VERSION);
   }

   public void onCreate(SQLiteDatabase db) {
       db.execSQL(SQL_CREATE_ENTRIES);
   }

   public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
   // örnekteki bu veritabanı çevrimiçi sitedeki verileri
önbelleklemek için kullanılacak. bu yüzden yükseltme gerektiğinde var
olan verileri atıp, yenileriyle başlayabilmeliyiz
       db.execSQL(SQL_DELETE_ENTRIES);
       onCreate(db);
   }

   public void onDowngrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
       onUpgrade(db, oldVersion, newVersion);
   }
}
```

Veritabanınıza erişmek için SQLiteOpenHelper'ın bir alt sınıfını örneklemeniz yeterli:

```
FeedReaderDbHelper mDbHelper = new FeedReaderDbHelper(getContext());
```

Veritabanına bilgileri kaydetmek

Veritabanına verileri eklemek için bir ContentValues nesnesini parametre olarak insert() metoduna geçmeniz yeterli:

```
// veri depomuza yazılabilir modda ulaşalım
SQLiteDatabase db = mDbHelper.getWritableDatabase();

// ekleyeceğimiz verileri bir map nesnesi şeklinde, sütun adları
anahtar olarak oluşturalım
ContentValues values = new ContentValues();
values.put(FeedEntry.COLUMN_NAME_ENTRY_ID, id);
values.put(FeedEntry.COLUMN_NAME_TITLE, title);
values.put(FeedEntry.COLUMN_NAME_CONTENT, content);

// yeni bir satır ekleyelim ve yeni satırın birincil anahtar değerini
döndürelim

long newRowId;
newRowId = db.insert(
        FeedEntry.TABLE_NAME,
        FeedEntry.COLUMN_NAME_NULLABLE,
        values);
```

insert() metodunun ilk parametresi tablo adıdır. İkinci parametre sütun adı yerine gelir ve veritabanı altyapısı eğer ContentValues boşsa buraya NULL değer verebilir ve böylece eklenecek değer olmadığında buraya yeni bir satır eklemez.

Veritabanından bilgileri okumak

Veritabanından bilgileri okumak için

Cursor içindeki satırlara bakmak için (değerleri okumaya başlamadan önce mutlaka çağırmanız gereken) nesnenin içinde gezmenize yarayan metotlardan birini kullanmalısınız. Genellikle moveToFirst() metodunu çağırarak başlamalısınız. Bu metot sonuçlar içindeki ilk girdinin "okuma pozisyonu"nun (read position) yerini sağlar. Her satırda Cursor'un get metotlarından birini (getString(), getLong() gibi) kulanarak sütunun oradaki değerini okuyabilirsiniz. Her get metoduna istediğiniz sütunun indeks posizyonunu parametre olarak geçmeniz gerekir. Bu indeks pozisyonunu getColumnIndex() veya getColumnIndexOrThrow()metotlarını kullanarak alabilirsiniz. Örnek:

```
cursor.moveToFirst();
long itemId = cursor.getLong(
   cursor.getColumnIndexOrThrow(FeedEntry._ID)
);
```

Veritabanından bilgileri silmek

Bir tablodaki satırları silmek için silmek o satırları ifade eden seçim kriterini de yazmanız gerekiyor. Android veritabanı API'si seçim kriterleri oluşturmak ve SQL sızmasını (injection) önlemek için bir mekanizma sağlıyor. Bu mekanizma sayesinde seçimi ifade eden kriter, parametreler ve seçim cümleciği olarak ayrılıyor. Seçim cümleciği, ilgilendiğiniz-silmek istediğiniz sütunları tanımladığınız yer oluyor. Parametreler ise değerler oluyor. Sonuç düzenli bir SQL ifadesiyle aynı şekilde yönetilmediği için SQL zehirlemesine karşı bir bağışıklık kazanılıyor.

```
// Sorgunun ‘where' kısmını tanımlayalım
String selection = FeedEntry.COLUMN_NAME_ENTRY_ID + " LIKE ?";
// Parametreleri sırasıyla tanımlayalım
String[] selectionArgs = { String.valueOf(rowId) };
// SQL ifadesini çalıştıralım
db.delete(table_name, selection, selectionArgs);
```

Veritabanında güncelleme yapmak

Veritabanınızdaki belli bir küme veriyi değiştirmek istediğinizde

Anahtar-Değer Çiftlerini Kaydetmek

Uygulamanızda kaydetmeyi istediğiniz veri ufak ya da başka bir deyişle küçük anahtar-değer çiftleri şeklindeyse SharedPreferences API'lerini kullanmalısınız.
Bir SharedPreferences nesnesi, anahtar-değer çiftleri içeren bir dosyaya karşılık gelir ve ona basit okuma-yazma işlemleri yapabileceğiniz metotları sunar. Her SharedPreferences dosyası geliştirme çatısı (framework) tarafından yönetilir ve gizli veya ortak olabilir.

Bu eğitim içeriğinde basit değerleri saklamak ve okumak için SharedPreferences API'lerini nasıl kullanacağınızı göreceksiniz.

NOT: *SharedPreferences API'leri sadece anahtar-değer çiftlerini okumak ve yazmak için kullanılır. İsim benzerliğinden dolayı onu Preference API'leriyle karıştırmamalısınız. Preference API'leriyle uygulamanızın ayar ekranı arayüzlerini oluşturursunuz.*

SharedPreferences'a hâkim olmak

İsterseniz yeni bir ortak tercih dosyası (shared preference file) oluşturabilir veya var olan bir ortak tercih dosyasına şu iki metotlardan birini kullanarak erişebilirsiniz:

- getSharedPreferences(): Eğer birden fazla ortak tercih dosyasına ihtiyacınız varsa bunu kullanın. Birden fazla ortak tercih dosyasıyla çalışmak için isimlerine ihtiyacınız vardır ve bu isimleri de metodun ilk parametresinde belirleyebilirsiniz. Bu metodu uygulamanızın içindeki herhangi birContext'ten çağırabilirsiniz.
- getPreferences(): Eğer Activity'niz için sadece bir tane ortak tercih dosyası kullanmanız gerekiyorsa bu metodu doğrudan Activity'den çağırarak kullanabilirsiniz.

Örneğin aşağıdaki kod bir Fragment içinde çalıştırılıyor. R.string.preference_file_key isminde bir string kaynağıyla belirlenmiş ortak tercih dosyasına erişiyor ve gizli modu kullanarak onu açıyor ki böylece dosya, sadece o uygulama tarafından erişilebilir oluyor:

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
        getString(R.string.preference_file_key),
Context.MODE_PRIVATE);
```

NOT: *getActivity() metodunu Fragment içinde çağırdığınızda, içinde o Fragment'ı barındıran Activity'nin referansını almış olursunuz.*

Ortak tercih dosyalarınızı isimlendirirken uygulamanız için belirlenmiş eşsiz isimler kullanmalısınız."com.example.myapp.PREFERENCE_FILE_KEY" gibi.

Alternatif olarak, eğer Activity'niz için sadece bir ortak tercih dosyasına ihtiyacınız varsa getPreferences()metodunu da kullanabilirsiniz:

```
SharedPreferences sharedPref =
getActivity().getPreferences(Context.MODE_PRIVATE);
```

UYARI: *Eğer bir ortak tercih dosyasını MODE_WORLD_READABLE veya MODE_WORLD_WRITEABLE modlarıyla oluşturuyorsanız, dosya belirtecinin (isminin) ne olduğunu bilen herhangi başka bir uygulama verilerinize erişebilir. Bu yüzden karışık isimler koymak tercih edilir.*

SharedPreferences'a yazmak

Bir ortak tercih (shared preference) dosyasına yazmak istiyorsanız, SharedPreferences nesnenizin edit() metodunu çağırarak bir SharedPreferenecs.Editor nesnesi elde etmelisiniz. Ardından yazmak istediğiniz anahtar ve değerleri putInt() ve putString() gibi uygun metotlara geçirmelisiniz. Yaptığınız değişiklikleri kaydetmek için de commit() metodunu çağırmalısınız. Örnek:

```
SharedPreferences sharedPref =
getActivity().getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putInt(getString(R.string.en_yuksek_skor), yeniYuksekSkor);
editor.commit();
```

SharedPreferences'tan okumak

Bir ortak tercih dosyasındaki değerleri getirmek için getInt() ve getString() gibi metotları çağırmalısınız. Bunu yaparken istediğiniz değerin anahtarını ve eğer anahtar bulunmazsa döndürülecek varsayılan değeri sağlamalısınız. Örnek:

```
SharedPreferences sharedPref =
getActivity().getPreferences(Context.MODE_PRIVATE);
int defaultValue =
getResources().getInteger(R.string.saved_high_score_default);
long highScore =
sharedPref.getInt(getString(R.string.saved_high_score), defaultValue);
```

Verileri Arka Planda Yüklemek

Göstermek istediğiniz veriler için ContentProvider'a sorgulama yapmak zaman alan bir işlemdir. Hele bir de bu sorguyu uygulamanızda doğrudan Activity üzerinden yapıyorsanız, Activity'nin "donmasına" ve büyük ihtimalle "Application Not Responding (Uygulama Yanıt Vermiyor)" sistem hatasına sebep olursunuz. Böyle hatalar olmasa bile mutlaka kullanıcı arayüzünde rahatsız edici yavaşlıklar yaşanacaktır. Kullanıcılar için "yağ gibi akan" uygulamalar yapmalısınız. Bu yüzden bu tür problemleri yaşamamak için sorgularınızı ayrı bir iş parçacığında (thread) başlatmalı, bitmesini beklemeli ve ondan sonra sonucu kullanıcı arayüzünde (UI) göstermelisiniz.

Veri sorgulama sırasında bile arayüzü yağ gibi akan uygulamaları çok basit bir nesneyle yapabilirsiniz. Bu nesne, sorguları eşzamansız (asenkron) bir şekilde arka planda çalıştıran ve bittiğinde sonuçları tekrar Activity'ye bağlayan CursorLoader nesnesidir. CursorLoader tüm bu "sorguları arka planda halletme" yeteneklerinin yanında, sorguyla ilişkili veri değiştiğinde sorguyu otomatik olarak yeniden başlatabilme yeteneğine sahiptir.

Takip eden eğitim içeriklerinde CursorLoader'ı arka planda nasıl kullanacağınız açıklanacaktır. ÖrneklerdeAndroid Destekleme Kütüphanesi v4'ü kullanıyoruz ki

uygulamalarınız Android 1.6'dan en yeni Android sürümüne kadar aynı sınıflarla yoluna devam edebilsin.

Sıradaki konular:

1. Bir Sorguyu CursorLoader ile Çalıştırma: CursorLoader kullanarak arka planda sorgular çalıştırmayı öğrenebilirsiniz.
2. Sonuçları Kullanmak: Sorgularınızdan dönen Cursor'leri nasıl kullanacağınızı öğrenebilirsiniz.

Bir Sorguyu CursorLoader ile Çalıştırma

Bir CursorLoader uygulamanızın arka planında bir ContentProvider'a (veri sağlayıcı) dayalı eşzamansız (asenkron) sorgular çalıştırmanıza yarar. Bu sayede ContentProvider'lar ile çalışırken uygulamanızın ön yüzünde herhangi bir performans kaybının önüne geçebilirsiniz. CursorLoader aynı zamanda yaptığı sorguların sonucunu, kendini çağıran bileşene (Activity ya da FragmentActivity gibi) döndürür. Bu sayede Activity veya FragmentActivity, o sırada sorgu çalışmaya devam ederken bile kullanıcıyla rahatlıkla etkileşime geçer.

CursorLoader'ı kullanan Activity'yi tanımlamak

CursorLoader'ı Activity veya FragmentActivity ile kullanmak için LoaderCallbacks<Cursor> arayüzünü (interface) kullanmanız gerekiyor. Bir CursorLoader, bu arayüzde tanımlanmış callback metotlarını çağırarak onu kullanan sınıfla haberleşir; bu eğitim içeriğinde ve sonrakinde bu callback metotlarını ayrıntılarıyla işleyeceğiz.

Örnek olarak CursorLoader'ın Android Destekleme Kütüphanesi'ndeki sürümünü kullanan bir FragmentActivity'nin nasıl tanımlanacağını görelim. FragmentActivity'yi türeterek (extend) CursorLoaderdesteği kazandığınız gibi Fragment desteği de kazanırsınız. FragmentActivity'nin normal Activity'den en önemli farkı budur desek, yanlış olmaz.

```
public class PhotoThumbnailFragment extends FragmentActivity
implements
        LoaderManager.LoaderCallbacks {
...
}
```

Sorguyu ilklendirmek (Initialization)

Bir sorguyu ilklendirmek için LoaderManager.initLoader() metodunu çağırmalısınız. Bu işlem arka planda ilgili framework'ü ilklendirir. Bu işlemin ardından kullanıcının girdiği veriyi sorguda kullanabilir veya kullanıcının verileriyle bir işiniz yoksa bu işlemi onCreate() ya da onCreateView() metotlarında gerçekleştirebilirsiniz. Bir örneğe bakalım:

```
// o anki bileşende (Activity gibi) kullanılacak
// özel Loader'ı tanımlar
private static final int URL_LOADER = 0;

...

/* Sistem Fragment'ı göstermeye hazır olduğunda
 * bu metot ile Fragment'ın View'ını gösterir
 */

public View onCreateView(
        LayoutInflater inflater,
        ViewGroup viewGroup,
        Bundle bundle) {

    ...

    /*
     * CursorLoader'ı ilklendirir. Buradaki URL_LOADER
     * değeri sonunda LoaderManager'ın onCreateLoader()
     * metoduna gider
     */

    getLoaderManager().initLoader(URL_LOADER, null, this);

    ...
}
```

NOT: *getLoaderManager() metodu sadece Fragment sınıfında geçerlidir. Eğer LoaderManager'ı bir FragmentActivityiçinde elde etmek isterseniz bu sefer getSupportLoaderManager()'ı çağırmalısınız.*

Sorguyu başlatmak

Loader'ları yöneten arka plan framework'ü ilklendiğinde yani başlamaya hazır olduğunda artıkonCreateLoader() metodundaki kodlarınızı çağırabilir. Sorguyu başlatmak için bu metot

ile bir CursorLoader döndürmelisiniz. Boş bir CursorLoader oluşturduktan sonra sonra, onun metotlarını sorgunuzu tanımlamak için kullanabilirsiniz veya oluşturup aynı anda sorgunuzu da tanımlayabilirsiniz:

```
/*
* Sistem Loader'ı ilklendirdiğinde çağrılacak ve sorguyu başlatmaya
* hazır callback metodudur. Bu metot genellikle initLoader()
çağrıldıktan
* sonra çağrılır. loaderID parametersi initLoader() çağrısına verilen
* ID değeriyle aynıdır.
*/

@Override
public Loader
onCreateLoader (int loaderID, Bundle bundle) {
    /*
     * oluşturulmuş olan Loader'ın ID değerine göre
     * aksiyon alıyoruz
     */

    switch (loaderID) {
        case URL_LOADER:
            // yeni bir CursorLoader döndürür
            return new CursorLoader(
                        getActivity(),   // Üst Activity'nin Context'i
                        mDataUrl,        // sorgulanacak tablo
                        mProjection,     // döndürülecek veriye
ilişkin şart
                        null,            // seçim cümleciği yok
                        null,            // seçim cüm. için parametre
yok
                        null             // varsayılan (ASC) sıralama
        );
        default:
            // içeri hatalı bir ID verilmişse
            return null;
    }
}
```

Arka plan framework'ü nesneyi ulaştıktan sonra sorguyu hemen arka planda başlatır. Sorgunun işi bitince de yine arka planda onLoadFinished() metodunu çağırır. Bu metoda ilişkin bilgileri bir sonraki eğitim içeriğinde bulabilirsiniz.

Diğer Fragment'lar ile İletişime Geçmek

Fragment arayüz bileşenlerini tekrar tekrar kullanabilmek istiyorsanız her birini, kendi kendini taşıyabilen, kendi layout'unu ve davranışını tanımlayabilen modüler bir yapıda inşa etmelisiniz. Tekrar kullanılabilir Fragment'ları bir kere tanımladıktan sonra, onları bir Activity ile ilişkilendirmeli ve genel arayüz mantığında yerine koymak için uygulama mantığına oturtmalısınız.

Genellikle bir Fragment'ı diğeriyle iletişime sokmak isteyeceksiniz. Örneğin bir kullanıcı olayına göre diğerinin içeriğini değiştirmek isteyeceksiniz. Fragment'tan Fragment'a yapılacak tüm iletişim, ilişkili oldukları Activity üzerinden gerçekleşir. Activity, burada trafik polisi gibidir. İki Fragment onsuz asla doğrudan iletişime geçmez.

Bu eğitim içeriğinde şunları göreceğiz:

Fragment'tan mesaj göndermek

Arayüz sınıfının tanımlaması

Bir Fragment'ın bağlı olduğu Activity ile iletişime geçmesi için Fragment sınıfınızın içinde bir arayüz sınıfı tanımlayabilir ve onu da Activity içinde gerçekleyebilirsiniz. Fragment, onAttach8) yaşam döngüsü olayı boyunca bu arayüz (interface) sınıfının gerçeklemesini (implementation) yakalar ve bu arayüz sınıfının metotlarını Activity ile haberleşmek için çağırır.

Aşağıda Fragment'tan Activity'ye doğru bir iletişim örneği var:

```
public class HeadlinesFragment extends ListFragment {
    OnHeadlineSelectedListener mCallback;

    // Taşıyıcı durumdaki Activity bu interface'i mutlaka implemente
etmeli
    public interface OnHeadlineSelectedListener {
        public void onArticleSelected(int position);
    }

    @Override
```

```
    public void onAttach(Activity activity) {
        super.onAttach(activity);

        // bununla taşıyıcı activity'nin bu callback interface'ini
        // gerçeklediğinden emin oluruz. Etmemişse hata fırlatırız
        try {
            mCallback = (OnHeadlineSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString()
                    + " mutlaka OnHeadlineSelectedListener");
        }
    }

    ...
}
```

Bu kodla birlikte artık Fragment, mCallback'in onArticleSelected() metodu yardımıyla Activity'ye mesaj teslim edebilir. mCallBack, OnHeadlineSelectedListener interface'inin bir örneği (instance) oluyor.

Örneğin aşağıdaki metod, kullanıcı bir liste öğesine tıklayınca çağırılıyor. Burada Fragment, callback interface'ini (mCallback) kullanarak olayı üstteki Activity'ye iletiyor.

```
@Override
    public void onListItemClick(ListView l, View v, int position, long
id) {
        // olayı üstteki activity'ye iletelim
        mCallback.onArticleSelected(position);
    }
```

Arayüz sınıfının gerçeklemesi

Fragment'tan olaylarla ilişkili callback'leri alabilmek için ev sahibi Activity'nin Fragment sınıfı içinde tanımlanmış arayüz (interface) sınıfını gerçeklemesi gerekiyor.

Aşağıdaki örnekte gördüğünüz Activity, yukarıdaki örnekte yer alan interface'i gerçekliyor.

```
public static class MainActivity extends Activity
        implements HeadlinesFragment.OnHeadlineSelectedListener{
    ...

    public void onArticleSelected(int position) {
        // kullanıcı HeadLinesFragment'tan bir yazının başlığını seçer
```

```
        // burada da yazıyı gösterecek işlemleri başlatırız
    }
}
```

Fragment'a mesaj göndermek

Ev sahibi Activity, bir örneğine (instance) findFragmentById() metoduyla eriştiği Fragment'a isterse mesaj da gönderebilir. Ardından o Fragment'ın public metotlarını doğrudan çalıştırabilir.

Örneğin, yukarıdaki örneklerde de geçtiği gibi bir Activity hayal edin. İçinde yazı başlıklarının listelendiği Fragment'tan (HeadLinesFragment) başka bir Fragment daha olsun ve seçilen başlığa göre bu Fragment'ta ilgili içerik gösterilsin. Bu Activity yukarıdaki callback metodunun döndürdüğü veriden yararlanarak ilgili içeriği diğer Fragment'ta gösterebilir. Aşağıdaki örnekte de bunu gerçekleştiriyoruz.

Activity, callback metoduyla gelen bilgiden yararlanarak bu bilgileri diğer Fragment'ta göstertiyor:

```
public static class MainActivity extends Activity
        implements HeadlinesFragment.OnHeadlineSelectedListener{
    ...

    public void onArticleSelected(int position) {
        // kullanıcı HeadlinesFragment'tan bir yazının başlığını seçti
        // şimdi seçtiği başlığa göre bir gösterme işlemi yapalım

        // önce detaylı bilgiyi göstereceğimiz ArticleFragment'a
erişelim
        ArticleFragment articleFrag = (ArticleFragment)

getSupportFragmentManager().findFragmentById(R.id.article_fragment);

        if (articleFrag != null) {
            // eğer articleFrag kullanılabilirse iki layout'u da
            // ekranda görebiliyoruz demektir

            // ArticleFragment'ın içideyken içeriğini güncelleyecek
            // metodu çağıralım
            articleFrag.updateArticleView(position);
        } else {
            // eğer articleFrag kullanılabilir değilse
```

```
            // tek parçalı bir layout'tayız demektir ve fragment'ları
yer
            // değiştirmemiz gerekir

            // bir Fragment oluşturalım ve seçilen başlığa göre ona
belli
            // argümanlar verelim
            ArticleFragment newFragment = new ArticleFragment();
            Bundle args = new Bundle();
            args.putInt(ArticleFragment.ARG_POSITION, position);
            newFragment.setArguments(args);

            FragmentTransaction transaction =
getSupportFragmentManager().beginTransaction();

            // fragment_container view'ını bu yeni fragment ile
değiştirelim
            // ve back stack'e bir işlem ekleyelim ki kullanıcı
            // geri döndüğünde daha önce gördüğü Fragment'ı görebilsin
            transaction.replace(R.id.fragment_container, newFragment);
            transaction.addToBackStack(null);

            // işlerimizi bitirelim
            transaction.commit();
        }
    }
}
```

Veritabanı Kullanımı

Android platformu da diğer mobil platformlar gibi veritabanı olarak **SQLite** kullanımını tercih etmektedir. Hem **SQL** komutlarını çalıştırabilmesi hem de mobil cihazlar gibi düşük kapasiteli ortamlarda kolayca çalışabilmesi *SQLite*'ı Android ve iOS platformlarında ilk seçenek haline getirmiştir.

Buradaki adresten Windows için indirebileceğiniz SQLite'ı komut satırından çalıştırabilirsiniz. Bunun için indirilen **.exe** dosyasını varsayılan şekilde kurmanız yeterli olacaktır. Aşağıda bulunan anlatımlar Mac OS X üzerinde anlatılmıştır fakat komutlar Windows işletim sistemi içinde aynıdır.

(Buradaki adresten Mac OS X için indirebileceğimiz SQLite'ı Terminal'den çalıştırabilirsiniz. Terminal penceresinde aşağıdaki komutu girdiğinizde yeni bir veritabanı yaratılacak ve diskte aynı isimle bir dosya oluşturulacaktır.)

```
Last login: Thu Apr  4 15:28:55 on ttys001
Ozans-MacBook-Pro:~ ozanuysal$ sqlite3 countries
```

Bu komutla birlikte diskte **countries** adında bir dosya oluşur ve biz veritabanı üzerinde SQL komutları kullanmaya başlayabiliriz. SQL komutları **MySQL** gibi karmaşık veritabanlarına göre daha basit olsa da**SQLite**, mobil uygulamalarda karşılaşabileceğimiz her türlü ihtiyacı giderecek kapasiteye sahiptir.

Şimdi aşağıdaki komutu girerek yeni bir tablo oluşturalım.

```
SQLite version 3.7.12 2012-04-03 19:43:07
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> create table country (id INTEGER PRIMARY KEY AUTOINCREMENT, ti
tle TEXT, code TEXT);
```

Yukarıdaki komutla **country** adında bir tablo oluşturmuş olduk. Burada ID değeri her satıra karşılık gelen anahtar değeridir ve ana anahtar (primary key) olarak tanımlanmıştır. **AUTOINCREMENT** ise her eklenen satırda otomatik olarak arttırılacağını gösterir. **title** ve **code** eklenen ülkenin adı ve ülke kodu için ayrılmıştır ve **TEXT** veri tipindedir.

Bu şekilde yeni bir tablo oluşturmuş olduk. Aşağıdaki komutla yeni bir satır ekliyoruz:

```
sqlite> INSERT INTO country VALUES (1,'Almanya','49');
```

Yukarıdaki **INSERT** ifadesi ile **country** tablosuna Almanya adında 49 koduna sahip bir ülke girmiş olduk. Bundan sonra diğer ülkeleri de girerek tablomuzu dolduralım.

```
sqlite> INSERT INTO country VALUES (1,'Almanya','49');
sqlite> INSERT INTO country VALUES (2,'Turkiye','90');
sqlite> INSERT INTO country VALUES (3,'Ingiltere','44');
sqlite> INSERT INTO country VALUES (4,'Amerika','49');
```

Bu şekilde tablomuza dört adet ülke girmiş olduk. Son adımda aşağıdaki komutu girerek veritabanından çıkıyoruz.

```
sqlite> .quit
Ozans-MacBook-Pro:~ ozanuysal$
```

Şimdi basit bir Android uygulamasıyla benzer işlemleri **Android SDK** üzerinden gerçekleştireceğiz. Android SDK, **SQLiteOpenHelper** sınıfı üzerinden SQLite ile ilgili işlemlerde bize yardımcı olacaktır. İlk olarak yeni bir proje oluşturarak **DBHelper** adında yardımcı bir sınıf yaratalım ve **SQLiteOpenHelper** sınıfına alt sınıf olarak belirleyelim.

```
package com.turkcell.sqlexample;

import java.util.ArrayList;
import java.util.List;

import android.content.ContentValues;
import android.content.Context;
```

```
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

public class DBHelper extends SQLiteOpenHelper {
  private static final String DATABASE_NAME   = "turkcellDB";
  // Contacts table name
  private static final String TABLE_COUNTRIES = "countries";
  public DBHelper(Context context) {
    super(context, DATABASE_NAME, null, 1);
   }

   @Override
   public void onCreate(SQLiteDatabase db) {
      String sql = "CREATE TABLE " + TABLE_COUNTRIES + "(id INTEGER PR
IMARY KEY,country_name TEXT,country_code TEXT" + ")";
      Log.d("DBHelper", "SQL : " + sql);
      db.execSQL(sql);
   }

   @Override
   public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVer
sion) {
      db.execSQL("DROP TABLE IF EXISTS " + TABLE_COUNTRIES);
      onCreate(db);
   }
}
```

Bu işlem yapıldığında *SQLiteOpenHelper* içerisinde yer alan **onCreate** ve **onUpgrade** metotlarını oluşturmamız gerekecektir. Aynı zamanda SQLiteOpenHelper (Context context, String name,SQLiteDatabase.CursorFactory factory, int version) yapıcısını da oluşturmamız gerekir.

onCreate metodu eğer uygulamayla ilgili SQLite veritabanı oluşturulmamışsa (örneğin uygulama ilk defa çalışıyorsa) bir SQLite veritabanı yaratır ve üzerinde metot içerisinde yer alan sorguları çalıştırır. Biz burada daha önce anlattığımıza benzer şekilde bir **CREATE sql** ifadesiyle ülke bilgilerine dair bir tablo oluşturduk.**execSQL** komutu da oluşturduğumuz SQL sorgusunu veritabanında çalıştırarak, ilgili tabloları yaratmamızda bize yardımcı oldu. Siz de uygulamanızda bu şekilde veritabanı yapısını oluşturabilirsiniz.

onUpgrade metodu, uygulamanın veritabanı güncellendiyse (örneğin yeni bir sürüm geldiyse) harekete geçecektir. SQLiteOpenHelper'ın yapıcısına dikkat edersek, son değerin veritabanı sürümü olduğunu görürüz. Veritabanında tablo yapısında bir değişiklik varsa ve eski sürümlerde güncelleme gerekiyorsa, bu metot içerisinde ilgili sorgular çalıştırılır. Hangi sürümler arasında geçiş olacağını **oldVersion** ve**newVersion** değişkenleriyle anlayabiliriz. Gerekli kontrol yapıları ile veritabanı güncelleme işlemlerini bu metod altında yapmalıyız. Biz burada basitçe tabloyu silip baştan yaratmayı tercih ettik.

Şimdi ülkelerle ilgili Country adında bir sınıf yaratalım:

```
package com.turkcell.sqlexample;

import java.io.Serializable;

public class Country implements Serializable {
  private static final long serialVersionUID = 1L;
  private int id;
  private String countryName;
  private String countryCode;

  public Country() {
    super();
  }

  public Country(String countryName, String countryCode) {
    super();
  this.countryName = countryName;
  this.countryCode = countryCode;
}

  public String getCountryCode() {
    return countryCode;
  }

  public void setCountryCode(String countryCode) {
    this.countryCode = countryCode;
  }

  public int getId() {
    return id;
```

```
  }

  public void setId(int id) {
    this.id = id;
  }

  public String getCountryName() {
    return countryName;
  }

  public void setCountryName(String countryName) {
    this.countryName = countryName;
  }
}
```

Yukarıdaki sınıfta ülkeye ait isim ve kod değerleri yer alıyor. ID ise veritabanındaki birincil anahtar değerine karşılık geliyor. Şimdi **DBHelper** sınıfına geri dönelim ve bir **insertCountry** metodu hazırlayalım. Bu metot veritabanına yeni bir ülke eklemeye yarayacaktır.

```
public void insertCountry(Country country) {
      SQLiteDatabase db = this.getWritableDatabase();

      ContentValues values = new ContentValues();
      values.put("country_name", country.getCountryName());
      values.put("country_code", country.getCountryCode());

      db.insert(TABLE_COUNTRIES, null, values);
      db.close();
   }
```

getWritableDatabase metodu veritabanıyla bağlantı kurarak yapacağımız sorgular için ortam hazırlar.**ContentValues** sınıfı, tabloda yer alan sütun adı ve bu sütunlara doldurulacak değerleri girdiğimiz**HashMap** sınıfına benzer bir yapıdır. **insert** metodu ise tablo adı

ve **ContentValues** içerisinde yer alan değerler ile bir **INSERT** sorgusu hazırlar ve veritabanına bir satır eklenmesini sağlar. Burada **country_name**adlı sütuna ülke adı girilirken, **country_code** sütununa ülke kodu giriliyor.

Şimdi de bütün ülkeleri listeleyen bir metot hazırlayalım:

```
public List<Country> getAllCountries() {
      List<Country> countries = new ArrayList<Country>();
      SQLiteDatabase db = this.getWritableDatabase();

      // String sqlQuery = "SELECT  * FROM " + TABLE_COUNTRIES;
      // Cursor cursor = db.rawQuery(sqlQuery, null);

      Cursor cursor = db.query(TABLE_COUNTRIES, new String[]{"id", "co
untry_name", "country_code"}, null, null, null, null, null);
      while (cursor.moveToNext()) {
         Country country = new Country();
         country.setId(cursor.getInt(0));
         country.setCountryName(cursor.getString(1));
         country.setCountryCode(cursor.getString(2));
         countries.add(country);
      }

      return countries;
   }
```

Burada veritabanından gelen sonuçları **List** tipinde bir dizi içerisinde saklamayı hedefliyoruz. Yine daha önceki örnekte olduğu gibi veritabanına erişim açarak bir **Cursor** üzerinden veritabanına sorgumuzu atıyoruz. **query** metodu basit bir **SELECT** sorgusu oluşturmak için idealdir ve veritabanından istenilen sütunları bir **String** dizisi şeklinde alır. Yukarıdaki metot aslında **SELECT id, country_name, country_code FROM country;** anlamında bir SQL sorgusu üretecektir. Sorgu sonucunda oluşan Cursor objesi ise bize sonuçlar içerisinde dolaşma olanağı sağlayacaktır. Burada ilk satırdan başlayarak bütün satırları dolaşma işlemini **moveToNext** metodu ile bir **while** döngüsü içerisinde yapıyoruz. Sonuç satırları bitene kadar *true*değeri dönülürken son satıra gelindiğinde **moveToNext** metodundan *false* değeri gelir ve döngü biter. Döngü içerisinde ise ilgili sütunlara sütun sırasıyla ulaşırız. Burada ID ilk sütun olduğundan **0** sıra sayısına

sahiptir. ID aynı zamanda sayısal bir şekilde tutulduğundan **getInt** metoduyla çağrılır. **country_name** ise ikinci değerdir ve **getString(1)** metodu ile çağırılır. Veritabanında **TEXT** tipinde saklandığından **getString**metodu yardımıyla alınır.

NOT: Alternatif olarak kendi oluşturduğumuz sorguları **rawQuery** metodu kullanarak veritabanına gönderebiliriz. Ancak kullanıcıdan girdi aldığımız ya da sorgu içerisindeki değişkenlerin kontrolümüz dışında değiştirildiği ortamlarda bu şekilde sorgu göndermemiz güvenlik açığı yaratacaktır.

Her satır bir **Country** nesnesi içerisinde saklandıktan sonra **countries** dizisine eklenir ve metodumuz bütün ülkeleri bir dizi şeklinde döndürür.

Şimdi daha önceki örneklerden hatırlayacağımız ve **ListView** tablosu doldurmak kullandığımız**MyListAdapter**'ı hatırlayalım:

```
package com.turkcell.sqlexample;

import java.util.List;

import android.app.Activity;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;

public class MyListAdapter extends BaseAdapter {

   private LayoutInflater inflater;
   private List<Country>  countryList;

   public MyListAdapter(Activity activity, List<Country> countries) {
      inflater = (LayoutInflater) activity.getSystemService(Context.LA
YOUT_INFLATER_SERVICE);
      countryList = countries;
   }

   @Override
   public int getCount() {
      return countryList.size();
   }
```

```
   @Override
   public Object getItem(int position) {
      return countryList.get(position);
   }

   @Override
   public long getItemId(int position) {
      return position;
   }

   @Override
   public View getView(int position, View convertView, ViewGroup paren
t) {
      View vi = convertView;
      if (convertView == null)
         vi = inflater.inflate(R.layout.listview_row, null); // create
 layout from

      TextView textView = (TextView) vi.findViewById(R.id.row_textview
); // user name

      Country country = countryList.get(position);

      textView.setText(country.getCountryName());
      return vi;
   }
}
```

Bu sınıf bize ülke listesinden bir tablo dolduracaktır. Tablo ile ilgili satır tasarımı ise şu şekildedir:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/androi
d"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:paddingBottom="@dimen/activity_vertical_margin"
```

```
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin" >

    <TextView
        android:id="@+id/row_textview"
        android:layout_width="fill_parent"
        android:layout_height="30dp"
        android:layout_gravity="center" />

</LinearLayout>
```

Son olarak **MainActivity** sınıfı içerisinde **onCreate** metodumuza göz atalım:

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

      final ListView customListView = (ListView) findViewById(R.id.lis
tview);
      DBHelper dbHelper = new DBHelper(getApplicationContext());

      SharedPreferences settings = getSharedPreferences("SQL", 0);
      boolean firstTime = settings.getBoolean("firstTime", true);

      if (firstTime) {
         dbHelper.insertCountry(new Country("Turkiye", "90"));
         dbHelper.insertCountry(new Country("Amerika", "1"));
         dbHelper.insertCountry(new Country("Ingiltere", "44"));
         dbHelper.insertCountry(new Country("Almanya", "49"));

         SharedPreferences.Editor editor = settings.edit();
         editor.putBoolean("firstTime", false);
         editor.commit();
      }

      List<Country> countries = dbHelper.getAllCountries();
      MyListAdapter myListAdapter = new MyListAdapter(MainActivity.thi
s, countries);
```

```
        customListView.setAdapter(myListAdapter);
    }
```

Uygulama çalıştığında yeni bir **DBHelper** yaratılarak veritabanı bağlantısı oluşturuluyor. Uygulamanın ilk defa çalışması ise (Google Play'den ilk yükleme) **SharedPreferences** üzerinden bir değişkenle kontrol ediliyor. Burada **firstTime** adında bir değerin cihazda daha önce bulunup bulunmadığı sorgulanıyor. Eğer yoksa değer doğru (*true*) dönüyor ve *if* içerisindeki kod çalışıyor. Bu kod ise dört adet ülkeyi veritabanına yukarıda oluşturduğumuz **insertCountry** metodu ile ekliyor. İşlem bittikten sonra **SharedPrefences.Editor**ile **firstTime** değeri yanlış (*false)* olarak değiştiriliyor ve **commit** ile kaydediliyor. Bu şekilde uygulama daha sonraki açılışlarında bu değerleri tekrar tekrar kaydetmeden çalıştıracaktır. Daha sonra ülkeler**getAllCountries** metoduyla veritabanından çağrılarak yine yukarıda oluşturduğumuz **MyListAdapter** ile listeleniyor.