TextView

TextView Attributes

You can use XML attributes for a `TextView` to control:

- Where the `TextView` is positioned in a layout (like any other view)
- How the `TextView` itself appears, such as with a background color
- What the text looks like within the `TextView`, such as the initial text and its style, size, and color

For example, to set the width, height, and initial text value of the view:

```
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Hello World!"
   <!-- more attributes -->
/>
```

You can extract the text string into a string resource (perhaps called `hello_world`) that's easier to maintain for multiple-language versions of the app, or if you need to change the string in the future. After extracting the string, use the string resource name with `@string/` to specify the text:

```
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/hello_world"
   <!-- more attributes -->
/>
```

In addition to `android:layout_width` and `android:layout_height` (which are required for a `TextView`), the most often used attributes with `TextView` are the following:

- `android:text`: Set the text to display.
- `android:textColor`: Set the color of the text. You can set the attribute to a color value, a predefined resource, or a theme. Color resources and themes are described in other chapters.
- `android:textAppearance`: The appearance of the text, including its color, typeface, style, and size. You set this attribute to a predefined style resource or theme that already defines these values.
- `android:textSize`: Set the text size (if not already set by `android:textAppearance`). Use `sp` (scaled-pixel) sizes such as `20sp` or `14.5sp`, or set the attribute to a predefined resource or theme.

- `android:textStyle` : Set the text style (if not already set by `android:textAppearance`). Use `normal` , `bold` , `italic` , or `bold | italic` .
- `android:typeface` : Set the text typeface (if not already set by `android:textAppearance`). Use `normal` , `sans` , `serif` , or `monospace` .
- `android:lineSpacingExtra` : Set extra spacing between lines of text. Use `sp` (scaled-pixel) or dp (device-independent pixel) sizes, or set the attribute to a predefined resource or theme.
- `android:autoLink` : Controls whether links such as URLs and email addresses are automatically found and converted to clickable (touchable) links.

Use one of the following with `android:autoLink` :

- `none` : Match no patterns (default).
- `web` : Match web URLs.
- `email` : Match email addresses.
- `phone` : Match phone numbers.
- `map` : Match map addresses.
- `all` : Match all patterns (equivalent to web|email|phone|map).

For example, to set the attribute to match web URLs, use `android:autoLink="web"` .

Referring to a TextView in code

To refer to a `TextView` in your Java code, use its resource `id` . For example, to update a `TextView` with new text, you would:

1. Find the `TextView` and assign it to a variable. You use the `findViewById()` method of the `View` class, and refer to the view you want to find using this format:

   ```
   R.id.view_id
   ```

   In which *view_id* is the resource identifier for the view (such as `show_count`) :

   ```
   mShowCount = (TextView) findViewById(R.id.show_count);
   ```

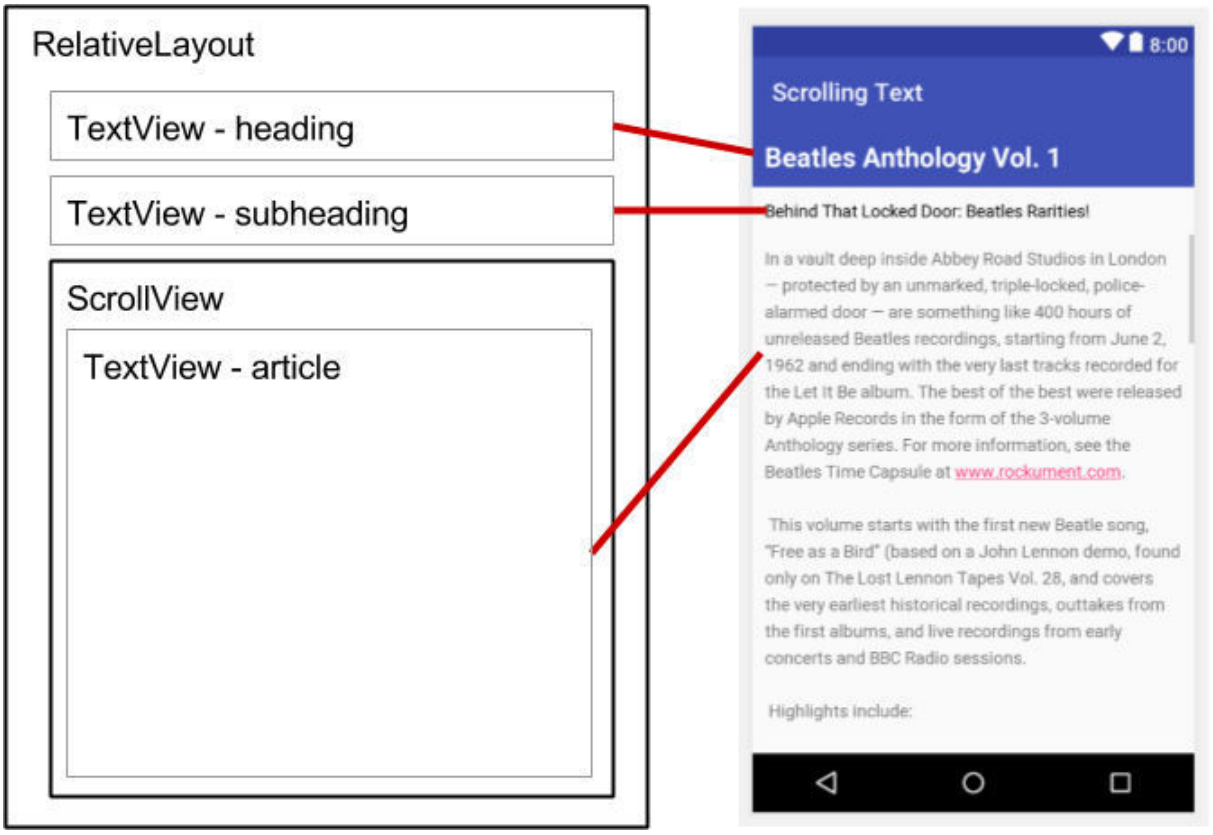
2. After retrieving the `View` as a `TextView` member variable, you can then set the text to new text (in this case, `mCount_text`) using the `setText()` method of the `TextView` class:

   ```
   mShowCount.setText(mCount_text);
   ```

Scrolling views

ScrollView with a TextView

To display a scrollable magazine article on the screen, you might use a `RelativeLayout` that includes a separate `TextView` for the article heading, another for the article subheading, and a third `TextView` for the scrolling article text (see the figure below), set within a `ScrollView` . The only part of the screen that would scroll would be the `ScrollView` with the article text.

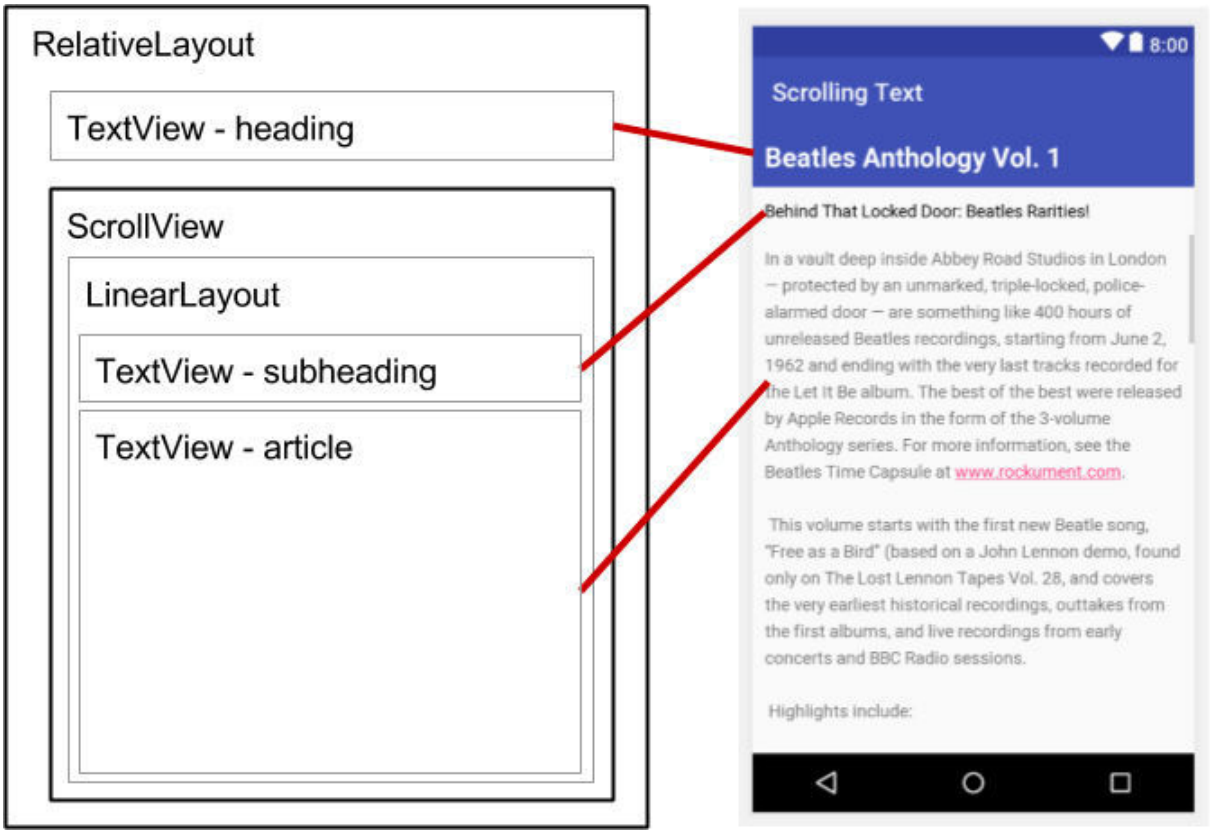


The layout with a ScrollView

ScrollView with a LinearLayout

A `ScrollView` can contain only one child `View` ; however, that `View` can be a `ViewGroup` that contains several `View` elements, such as `LinearLayout` . You can *nest* a `ViewGroup` such as `LinearLayout` *within* the `ScrollView` , thereby scrolling everything that is inside the `LinearLayout` .

For example, if you want the subheading of an article to scroll along with the article even if they are separate `TextView` elements, add a `LinearLayout` to the `ScrollView` as a single child `View` as shown in the figure below, and then move the `TextView` subheading and article elements into the `LinearLayout` . The user scrolls the entire `LinearLayout` which includes the subheading and the article.



A LinearLayout Inside the ScrollView

When adding a `LinearLayout` inside a `ScrollView` , use `match_parent` for the `LinearLayout` `android:layout_width` attribute to match the width of the parent `ScrollView` , and use `wrap_content` for the `LinearLayout` `android:layout_height` attribute to make it only large enough to enclose its contents.

Since `ScrollView` only supports vertical scrolling, you must set the `LinearLayout` orientation attribute to vertical (`android:orientation="vertical"`), so that the entire `LinearLayout` will scroll vertically. For example, the following XML layout scrolls the `article` `TextView` along with the `article_subheading` `TextView` :

```
<ScrollView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_below="@id/article_heading">

   <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
```

```
9        android:orientation="vertical">
10
11       <TextView
12          android:id="@+id/article_subheading"
13          android:layout_width="match_parent"
14          android:layout_height="wrap_content"
15          android:padding="@dimen/padding_regular"
16          android:text="@string/article_subtitle"
17          android:textAppearance=
18                        "@android:style/TextAppearance.DeviceDefault" />
19
20       <TextView
21          android:id="@+id/article"
22          android:layout_width="wrap_content"
23          android:layout_height="wrap_content"
24          android:autoLink="web"
25          android:lineSpacingExtra="@dimen/line_spacing"
26          android:padding="@dimen/padding_regular"
27          android:text="@string/article_text" />
28    </LinearLayout>
29
30 </ScrollView>
```

XML örneği

🚧 Layout

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.example.android.hellotoast.MainActivity">

    <Button
        android:id="@+id/button_toast"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:background="@color/colorPrimary"
        android:onClick="showToast"
        android:text="@string/button_label_toast"
        android:textColor="@android:color/white" />

    <TextView
        android:id="@+id/show_count"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_vertical"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:background="#FFFF00"
        android:text="@string/count_initial_value"
        android:textAlignment="center"
        android:textColor="@color/colorPrimary"
        android:textSize="160sp"
        android:textStyle="bold"
        android:layout_weight="1"/>

    <Button
        android:id="@+id/button_count"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:background="@color/colorPrimary"
        android:onClick="countUp"
        android:text="@string/button_label_count"
        android:textColor="@android:color/white" />
</LinearLayout>
```

📜 ScrollView

```
<ScrollView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_below="@id/article_heading">

   <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="vertical">

      <TextView
         android:id="@+id/article_subheading"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:padding="@dimen/padding_regular"
         android:text="@string/article_subtitle"
         android:textAppearance=
                       "@android:style/TextAppearance.DeviceDefault" />

      <TextView
         android:id="@+id/article"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:autoLink="web"
         android:lineSpacingExtra="@dimen/line_spacing"
         android:padding="@dimen/padding_regular"
         android:text="@string/article_text" />
   </LinearLayout>

</ScrollView>
```

Activities and intents

🚧 Create the Activity

When you create a new project in Android Studio and choose the **Backwards Compatibility (AppCompat)** option, the `MainActivity` is, by default, a subclass of the `AppCompatActivity` class. The `AppCompatActivity` class lets you use up-to-date Android app features such as the app bar and Material Design, while still enabling your app to be compatible with devices running older versions of Android.

Here is a skeleton subclass of `AppCompatActivity`:

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

📄 Declare the Activity in AndroidManifest.xml

```
<activity android:name=".MainActivity" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

📤 Starting an Activity with an explicit Intent

Yeni aktivity oluşturulduğunda eskisi **Paused** olur

```
Intent messageIntent = new Intent(this, ShowMessageActivity.class);
startActivity(messageIntent);
```

💾 Add data to the Intent

```
Intent messageIntent = new Intent(this, ShowMessageActivity.class);
// A web page URL
messageIntent.setData(Uri.parse("http://www.google.com"));
// a Sample file URI
messageIntent.setData(Uri.fromFile(new File("/sdcard/sample.jpg")));
// A sample content: URI for your app's data model
messageIntent.setData(Uri.parse("content://mysample.provider/data"));
// Custom URI
messageIntent.setData(Uri.parse("custom:" + dataID + buttonId));

startActivity(messageIntent);
```

Add extras to the Intent

```
Intent messageIntent = new Intent(this, ShowMessageActivity.class);

// Tek tek koyma
messageIntent.putExtra(EXTRA_MESSAGE, "this is my message");
messageIntent.putExtra(EXTRA_POSITION_X, 100);
messageIntent.putExtra(EXTRA_POSITION_Y, 500);

// Bundle yapısı
Bundle extras = new Bundle();
extras.putString(EXTRA_MESSAGE, "this is my message");
extras.putInt(EXTRA_POSITION_X, 100);
extras.putInt(EXTRA_POSITION_Y, 500);

messageIntent.putExtras(extras);

startActivity(messageIntent);
```

Retrieve the data from the Intent in the started Activity

```
Intent intent = getIntent();

// Data alma
Uri locationUri = intent.getData();

// Extra alma
String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
int positionX = intent.getIntExtra(MainActivity.EXTRA_POSITION_X);
int positionY = intent.getIntExtra(MainActivity.EXTRA_POSITION_Y);

// Bundle alma
Bundle extras = intent.getExtras();
String message = extras.getString(MainActivity.EXTRA_MESSAGE);
```

Getting data back from an Activity

Use startActivityForResult() to launch the Activity

```
startActivityForResult(messageIntent, TEXT_REQUEST);

// İstek tipleri
public static final int PHOTO_REQUEST = 1;
public static final int PHOTO_PICK_REQUEST = 2;
public static final int TEXT_REQUEST = 3;
```

Return a response from the launched Activity

```
Intent returnIntent = new Intent();

public final static String EXTRA_RETURN_MESSAGE =
                                  "com.example.mysampleapp.RETURN_MESSAGE";

messageIntent.putExtra(EXTRA_RETURN_MESSAGE, mMessage);
setResult(RESULT_OK,replyIntent);

// Activity'i sonlandırma
finish();
```

> To avoid confusing sent data with returned data, use a new `Intent` object rather than reusing the original sending `Intent` object.

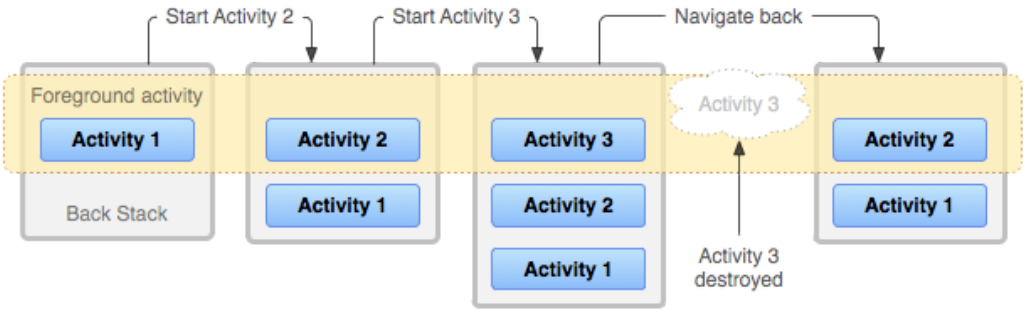
Read response data in onActivityResult()

- Aktivity'den yanıt geldiğinde çalışır

```
public void onActivityResult(int requestCode, int resultCode,  Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == TEXT_REQUEST) {
        if (resultCode == RESULT_OK) {
            String reply =
               data.getStringExtra(SecondActivity.EXTRA_RETURN_MESSAGE);
               // process data
        }
    }
}
```

Activity Navigation

Back navigation, tasks, and the back stack



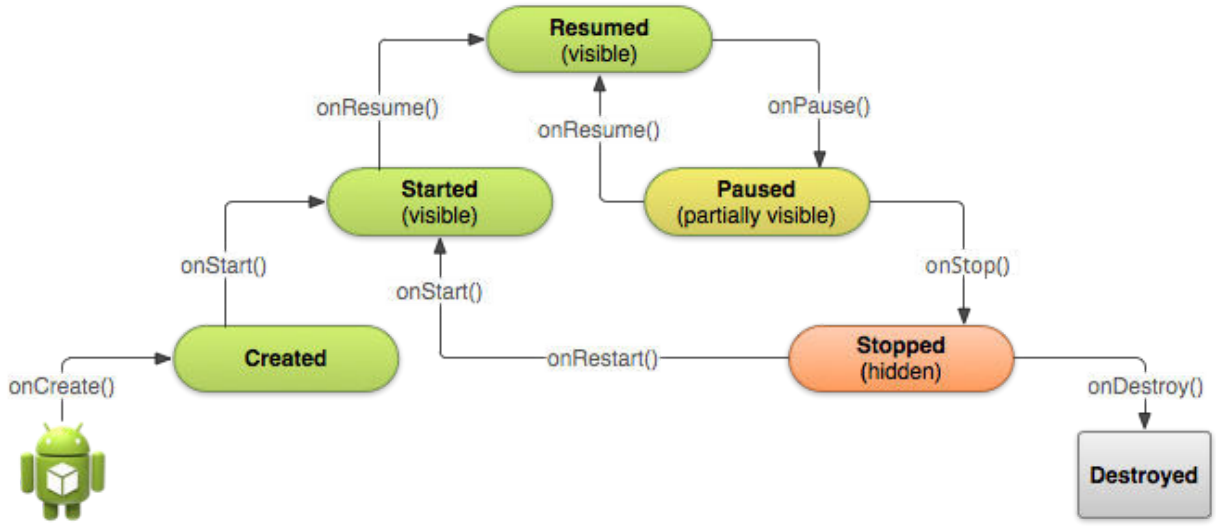
Implement Up navigation with a parent Activity

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <!-- The main activity (it has no parent activity) -->
    <activity android:name=".MainActivity">
       <intent-filter>
            <action android:name="android.intent.action.MAIN" />

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

    <!-- The child activity) -->
    <activity android:name=".SecondActivity"
       android:label = "Second Activity"
       android:parentActivityName=".MainActivity">
       <meta-data
          android:name="android.support.PARENT_ACTIVITY"
          android:value="com.example.android.twoactivities.MainActivity" />
       </activity>
</application>
```

> To support older versions of Android, include `<meta-data>` information to define the parent `Activity` explicitly

Activity lifecycle and state

Activity states and lifecycle callback methods



Temel Kullanım

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // The activity is being created.
}
```

Saving Activity instance state

- Lifecycle metodlarından değildir
- Kullanıcı `Activity` 'den ayrılıken çağırılır.
- Bazen `onStop()` 'tan önce çalışır

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    super.onSaveInstanceState(savedInstanceState);
```

```

    // Save the user's current game state
    savedInstanceState.putInt("score", mCurrentScore);
    savedInstanceState.putInt("level", mCurrentLevel);
}
```

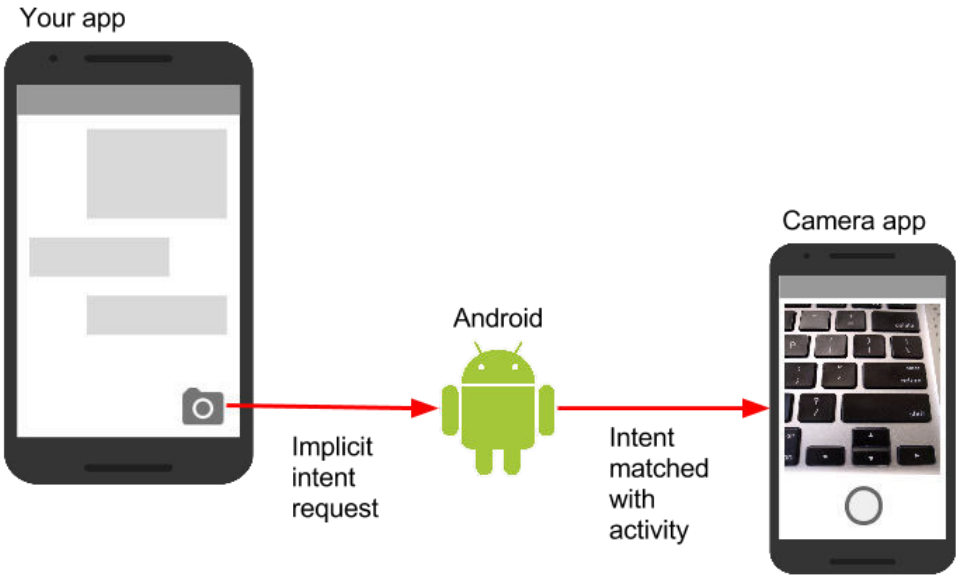
Restoring Activity instance state

- Kaydedilen `Bundle` verileri `onCreate()` callback metodunda kullanılmakta
- `Activity` oluşturulduktan sonra çalışan `onStart()` metodunun ardından çalışan `onRestoreInstanceState()` callback metodunda da kullanılabilir

> İlk kez uygulama oluşturulduğunda `Bundle` verisi olmayacağından `null` kontrolü yapılması gerekir.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    // Always call the superclass first
    super.onCreate(savedInstanceState);

    // Check if recreating a previously destroyed instance.
    if (savedInstanceState != null) {
        // Restore value of members from saved state.
        mCurrentScore = savedInstanceState.getInt("score");
        mCurrentLevel = savedInstanceState.getInt("level");
    } else {
        // Initialize members with default values for a new instance.
        // ...
    }
    // ... Rest of code
}
```

Implicit intents

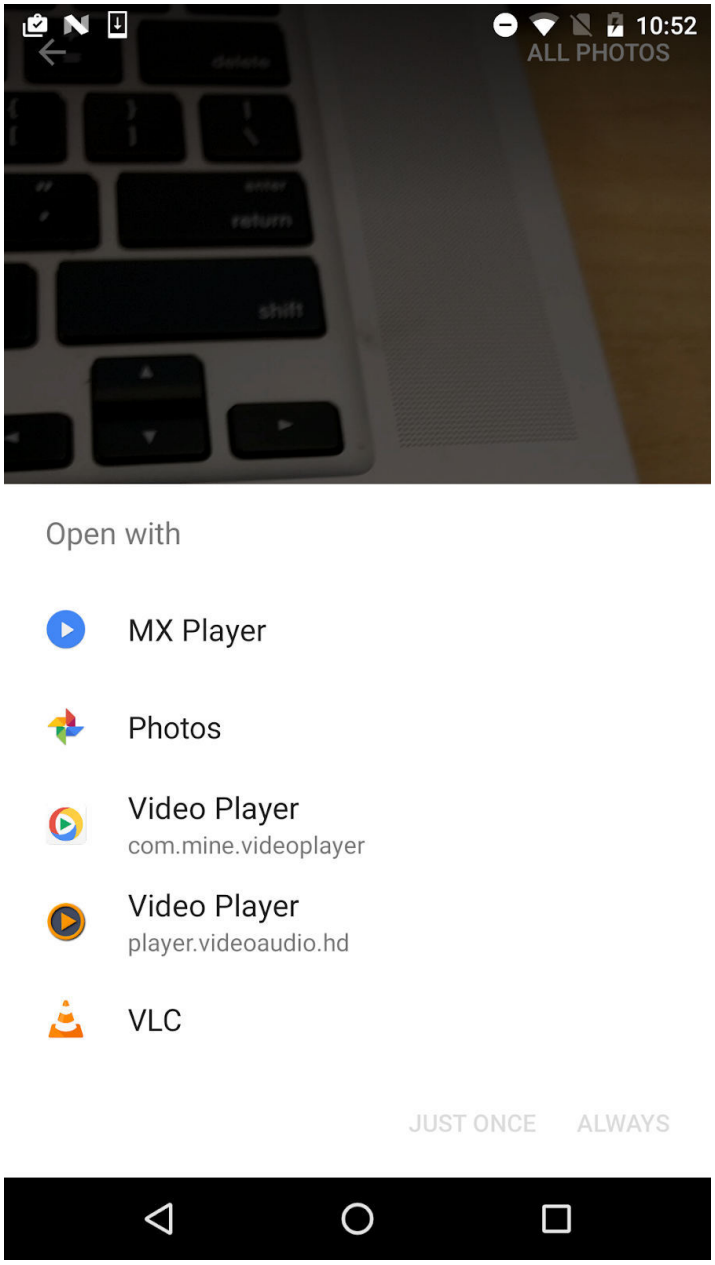


Create implicit Intent objects

- `Intent` oluşturmadan önce isteği karşılayabilecek `Activity` var mı kontrol edilmelidir.
- İsteklerini sağlayacak `Activity` olmazsa uygulama kapanır

```
// Implicit intent oluşturma
Intent sendIntent= new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Intent başlığını ayarlama
String title = getResources().getString(R.string.chooser_title);

// App Chooser oluşturma
Intent chooser = Intent.createChooser(sendIntent, title);

// İsteği sağlayacak Activity var mı kontrolü
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(chooser);
}
```

App Chooser



Receiving an implicit Intent

- `AndroidManifest.xml` dosyasında `intent-filter` ile tanımlanan uygulamalardan biri seçilir
- `intent-filter` 0 veya daha fazla `action` , `category` veya `data` içerir
- `intent-filter` içermeyen `Activity` 'ler sadece explicit intent ile çağrılabilir
- Birden fazla `intent-filter` veya bir `intent-filter` için birden fazla `action` , `category` veya `data` tanımlanabilir

```
<intent-filter>
    <!-- Açılış Activity'si olduğunu belirtlir -->
    <action android:name="android.intent.action.MAIN" />
    <!-- Launcher ekranında (telefon ana ekranında) gözükmesini sağlar -->
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
```

```
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

Actions

- Action yapısı Intent üzerinde `ACTION_` ön eki ile kullanılır

```
<intent-filter>
    <!-- Intent sendIntent = new Intent(Intent.ACTION_SEND); -->
    <action android:name="android.intent.action.EDIT" />
    <action android:name="android.intent.action.VIEW" />
</intent-filter>
```

Categories

- Category yapısı Intent üzerinde `CATEGORY_` ön eki ile kullanılır
- Tüm implicit intent objelerine varsayılan olarak `android.intent.category.DEFAULT` atanır

```
<intent-filter>
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
</intent-filter>
```

Data

- Alttaki yapıları vardır
  - URI Scheme
  - URI Host
  - URI Path
  - Mime type

```
<intent-filter>
    <data android:mimeType="video/mpeg" android:scheme="http" />
    <data android:mimeType="audio/mpeg" android:scheme="http" />
</intent-filter>
```

Sharing data using ShareCompat.IntentBuilder

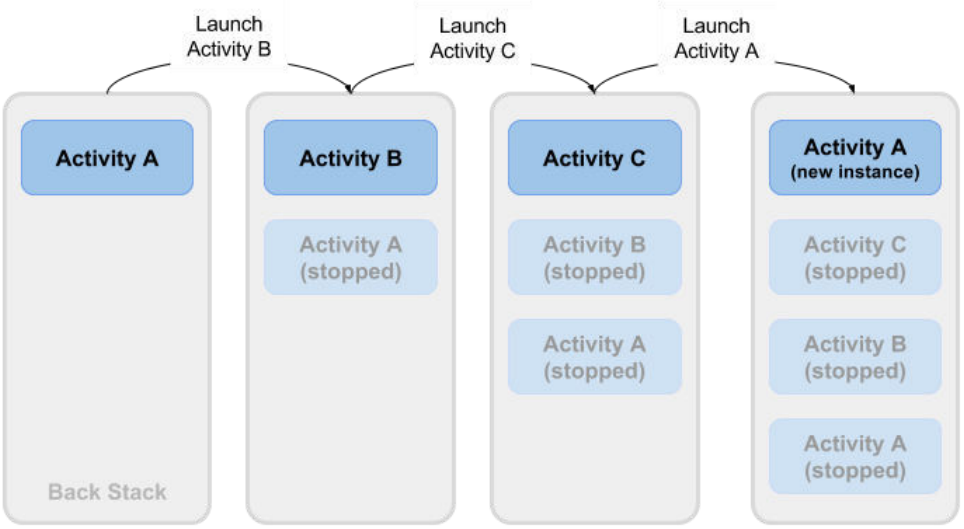
- Sosyal ağ uygulamalarında veri paylaşmak için kullanılan yöntemdir
- Implicit intent yerine, Android sunduğu bu yapı daha faydalıdır

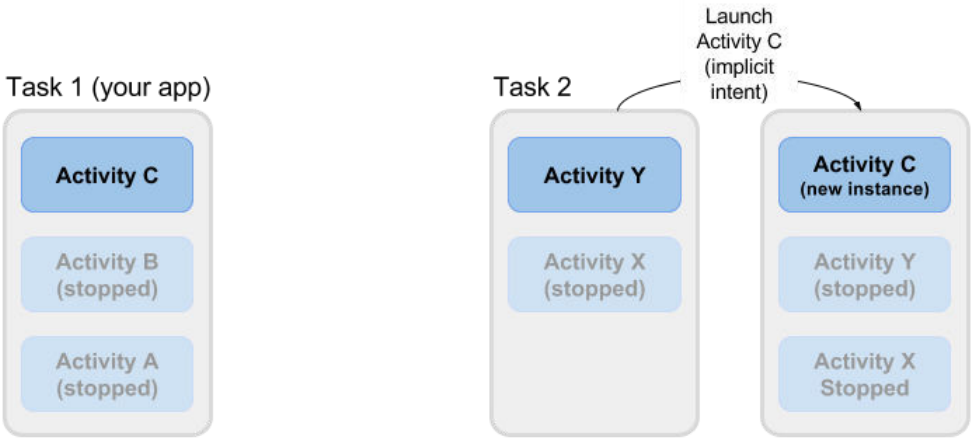
```
ShareCompat.IntentBuilder
    .from(this)         // information about the calling activity
    .setType(mimeType)  // mime type for the data
    .setChooserTitle("Share this text with: ") //title for the app chooser
    .setText(txt)       // intent data
    .startChooser();    // send the intent
```

Managing tasks

- Android'in çalışma yapısı gereği, `Activity` 'ler eski açık olanı kullanmak yerine kendileri yeni `Activity` oluştururlar (Şekil 1)
- Implicit intent ile açılan `Activity` 'ler de, asıl çalışan `Activity` 'den bağımsız olarak açılır (Şekil 2)

> Bu yapı **Android Launch Modes** ile değiştirilebilmektedir.





Activity launch modes

`AndroidManifest.xml` dosyası içerisindeki `<activity>` alanının değiştirilmesi ile yönetilir

Activity attributes

- Belirtilen `launchMode` değerlerinden biri kullanılır
- Varsayılan olarak `standart` değeri seçilir

Launch Mode	Anlamı
`standart`	Android'in varsayılan modu
`singleTop`	Activity, stack'te en tepede ise yeni işlerde yeni activity oluşturulmaz

`singleTask`	Activity için yeni bir işlem tanımlandığında, işlem yapan activity kullanılır, yeni oluşturulmaz
`singleInstance`	Activity yalnızca bir kez oluşturulur

```
<activity
   android:name=".SecondActivity"
   android:label="@string/activity2_name"
   android:parentActivityName=".MainActivity"
   android:launchMode="standard">
   <!-- More attributes ... -->
</activity>
```

Intent flags

- Activity attributes gibidir, ama çakışma durumunda bayraklar ele alınır
- `setFlag()` ve `getFlag()` ile kullanılır

Flag	Launch Mode karşılığı	Anlamı
`FLAG_ACTIVITY_NEW_TASK`	`singleTask`	İşlem için var olan Activity'i kullanır
`FLAG_ACTIVITY_SINGLE_TOP`	`singleTop`	Activity, stack'te en tepede ise yeni işlerde yeni activity oluşturulmaz
`FLAG_ACTIVITY_CLEAR_TOP`		Eğer activity stack'te varsa, onu tepeye alıp, üstündeki her activity'i `destroy` eder. FLAG_ACTIVITY_NEW_TASK ile kullanılırsa activity işlemlerini ön plana taşır

Handle a new Intent

- Genellikle `onResume()` 'den sonra çalışır
- `getIntent()` metodu her zaman, `Activity` 'nin kendi `intent` 'ini döndürdüğünden bu yapı kullanılır
- `setIntent()` ile Activity intent'i değiştirilir

```
@Override
public void onNewIntent(Intent intent) {
    super.onNewIntent(intent);
    // Use the new intent, not the original one
    setIntent(intent);
}
```

😵‍💫 Düzensiz Notlar

Task affinities

Task affinities indicate which task an `Activity` prefers to belong to when that `Activity` instance is launched. By default each `Activity` belongs to the app that launched it. An `Activity` from outside an app launched with an implicit `Intent` belongs to the app that sent the implicit `Intent` .

To define a task affinity, add the `android:taskAffinity` attribute to the `<activity>` element in the `AndroidManifest.xml` file. The default task affinity is the package name for the app (declared in `<manifest>`). The new task name should be unique and different from the package name. This example uses `"com.example.android.myapp.newtask"` for the affinity name.

```
<activity
   android:name=".SecondActivity"
   android:label="@string/activity2_name"
   android:parentActivityName=".MainActivity"
   android:taskAffinity="com.example.android.myapp.newtask">
   <!-- More attributes ... -->
</activity>
```

Task affinities are often used with the `singleTask` launch mode or the `FLAG_ACTIVITY_NEW_TASK` `Intent` flag to place a new `Activity` in its own named task. If the new task already exists, the `Intent` is routed to that task and that affinity.

Another use of task affinities is reparenting, which enables a task to move from the `Activity` in which it was launched to the `Activity` it has an affinity for. To enable task reparenting, add a task affinity attribute to the `<activity>` element and set `android:allowTaskReparenting` to true.

```
<activity
   android:name=".SecondActivity"
   android:label="@string/activity2_name"
   android:parentActivityName=".MainActivity"
   android:taskAffinity="com.example.android.myapp.newtask"
   android:allowTaskReparenting="true" >
   <!-- More attributes ... -->
</activity>
```

Responding to button-click events

Adding onClick() to the layout element

A quick way to set up an `OnClickListener` for a clickable element in your `Activity` code and assign a callback method is to add the `android:onClick` attribute to the element in the XML layout.

For example, a `Button` in the layout would include the `android:onClick` attribute:

```
<Button
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

When a user clicks the `Button`, the Android framework calls the `sendMessage()` method in the `Activity`：

```
public void sendMessage(View view) {
    // Do something in response to button click
}
```

The callback method for the `android:onClick` attribute must be `public`, return `void`, and define a `View` as its only parameter (this is the `View` that was tapped). Use the method to perform a task or call other methods as a response to the `Button` tap.

Using the button-listener design pattern

You can also handle the click event in your Java code using the button-listener design pattern, shown in the figure below. For more information on the "listener" design pattern, see Creating Custom Listeners.

Use the event listener `View.OnClickListener`, which is an interface in the `View` class that contains a single callback method, `onClick()`. The method is called by the Android framework when the view is triggered by user interaction.

The event listener must already be registered to the `View` in order to be called for the event. Follow these steps to register the listener and use it (refer to the figure below the steps):

1. Use the `findViewById()` method of the `View` class to find the `Button` in the XML layout file:

```
Button button = findViewById(R.id.button_send);
```

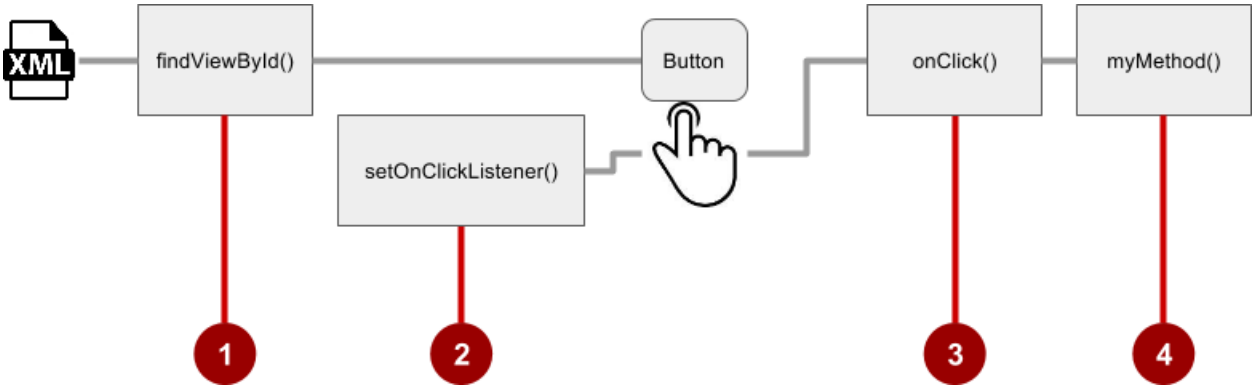
2. Get a new `View.OnClickListener` and register it to the `Button` by calling the `setOnClickListener()` method. The argument to `setOnClickListener()` takes an object that implements the `View.OnClickListener` interface, which has one method: `onClick()` .

```
button.setOnClickListener(new View.OnClickListener() {
   // ... The onClick method goes here.
}
```

3. Override the `onClick()` method:

```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Do something in response to button click
    }
}
```

4. Do something in response to the button click, such as perform an action.



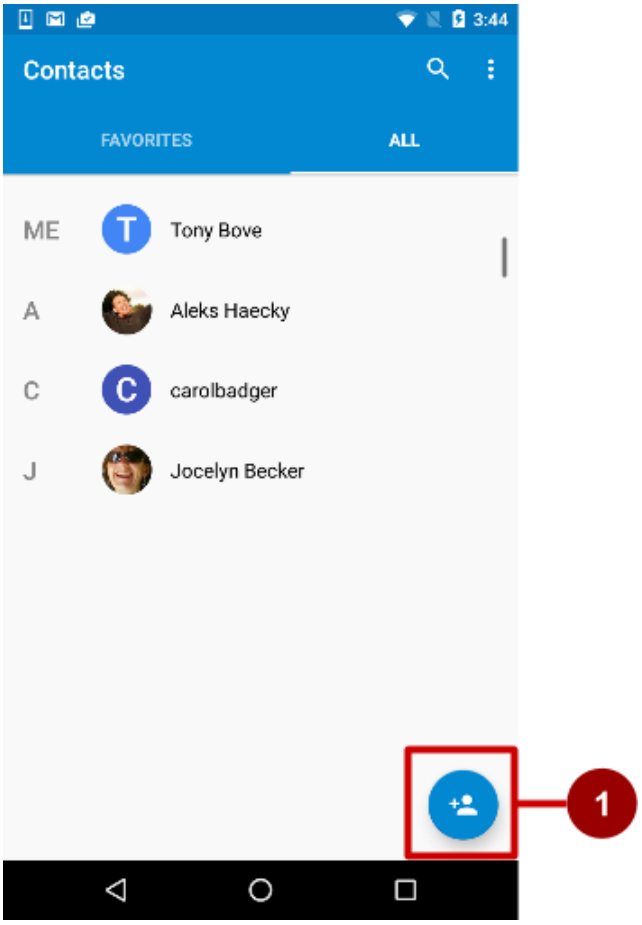
Button-Listener design pattern

Using clickable images

```
<LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal"
        android:layout_marginTop="260dp">
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/icecream_circle"
```

```
            android:onClick="orderIcecream"/>
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/donut_circle"
            android:onClick="orderDonut"/>
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/froyo_circle"
            android:onClick="orderFroyo"/>
</LinearLayout>
```

Using a floating action button



Floating Action Button (#1)

```
<android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:layout_margin="@dimen/fab_margin"
        android:src="@drawable/ic_fab_chat_button_white" />
```

Input Controls

Checkboxex

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

        <CheckBox android:id="@+id/checkbox1_chocolate"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/chocolate_syrup" />
        <CheckBox android:id="@+id/checkbox2_sprinkles"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/sprinkles" />
        <CheckBox android:id="@+id/checkbox3_nuts"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/crushed_nuts" />

</LinearLayout>
```

```
boolean checked = ((CheckBox) view).isChecked();
```

Radio buttons

```
<RadioGroup
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="24dp"
        android:layout_marginStart="24dp"
        android:orientation="vertical"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/delivery_label">

        <RadioButton
            android:id="@+id/sameday"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onRadioButtonClicked"
            android:text="Same day messenger service" />

        <RadioButton
            android:id="@+id/nextday"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onRadioButtonClicked"
            android:text="Next day ground delivery" />

        <RadioButton
            android:id="@+id/pickup"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onRadioButtonClicked"
            android:text="Pick up" />
</RadioGroup>
```

```
public void onRadioButtonClicked(View view) {
   // Check to see if a button has been clicked.
   boolean checked = ((RadioButton) view).isChecked();
   // Check which radio button was clicked.
   switch(view.getId()) {
      case R.id.sameday:
         if (checked)
            // Code for same day service ...
            break;
      case R.id.nextday:
         if (checked)
            // Code for next day delivery ...
            break;
      case R.id.pickup:
         if (checked)
            // Code for pick up ...
            break;
   }
}
```

Building the AlertDialog

```
import android.content.DialogInterface;
import android.support.v7.app.AlertDialog;

AlertDialog.Builder myAlertBuilder = new
                  AlertDialog.Builder(MainActivity.this);
myAlertBuilder.setTitle(R.string.alert_title);
myAlertBuilder.setMessage(R.string.alert_message);

myAlertBuilder.setPositiveButton("OK", new
                                 DialogInterface.OnClickListener() {
```

```
      public void onClick(DialogInterface dialog, int which) {
           // User clicked OK button.
           // ... Action to take when OK is clicked.
      }
});
myAlertBuilder.setNegativeButton("Cancel", new
                                 DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int which) {
         // User clicked the CANCEL button.
         // ... Action to take when CANCEL is clicked.
      }
});

alertDialog.show();
```

Navigation

```
@Override
public void onBackPressed() {
    // Add the Back key handler here.
    return;
}
```

```
<activity android:name=".OrderActivity"
   android:label="@string/title_activity_order"
   android:parentActivityName=
                        "com.example.android.droidcafe.MainActivity">
   <meta-data
      android:name="android.support.PARENT_ACTIVITY"
      android:value=".MainActivity"/>
</activity>
```

```
<activity android:name="com.example.android.droidcafeinput.OrderActivity"
    android:label="Order Activity"
    android:parentActivityName=".MainActivity">
    <meta-data android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity"/>
</activity>
```

7.1: AsyncTask and AsyncTaskLoader

AsyncTask

When `AsyncTask` is executed, it goes through four steps:

1. `onPreExecute()` is invoked on the UI thread before the task is executed. This step is normally used to set up the task, for instance by showing a progress bar in the UI.
2. `doInBackground(Params...)` is invoked on the background thread immediately after `onPreExecute()` finishes. This step performs a background computation, returns a result, and passes the result to `onPostExecute()` . The `doInBackground()` method can also call `publishProgress(Progress...)` to publish one or more units of progress.
3. `onProgressUpdate(Progress...)` runs on the UI thread after `publishProgress(Progress...)` is invoked. Use `onProgressUpdate()` to report any form of progress to the UI thread while the background computation is executing. For instance, you can use it to pass the data to animate a progress bar or show logs in a text field.
4. `onPostExecute(Result)` runs on the UI thread after the background computation has finished. The result of the background computation is passed to this method as a parameter.

Example of an AsyncTask

```java
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
            // Escape early if cancel() is called
            if (isCancelled()) break;
        }
        return totalSize;
    }

    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }

    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}
```

```
new DownloadFilesTask().execute(url1, url2, url3);
```

The example above goes through three of the four basic `AsyncTask` steps:

- `doInBackground()` downloads content, a long-running task. It computes the percentage of files downloaded from the index of the `for` loop and passes it to `publishProgress()` . The check for `isCancelled()` inside the `for` loop ensures that if the task has been cancelled, the system does not wait for the loop to complete.
- `onProgressUpdate()` updates the percent progress. It is called every time the `publishProgress()` method is called inside `doInBackground()` , which updates the percent progress.
- `doInBackground()` computes the total number of bytes downloaded and returns it. `onPostExecute()` receives the returned result and passes it into `onPostExecute()` , where it is displayed in a dialog.

The parameter types used in this example are:

- `URL` for the "Params" parameter type. The `URL` type means you can pass any number of URLs into the call, and the URLs are automatically passed into the `doInBackground()` method as an array.
- `Integer` for the "Progress" parameter type.
- `Long` for the "Result" parameter type.

Cancelling Task

You can cancel a task at any time, from any thread, by invoking the `cancel()` method.

- The `cancel()` method returns `false` if the task could not be cancelled, typically because it has already completed normally. Otherwise, `cancel()` returns `true` .
- To find out whether a task has been cancelled, check the return value of `isCancelled()` periodically from `doInBackground(Object[])` , for example from inside a loop as shown in the example below. The `isCancelled()` method returns `true` if the task was cancelled before it completed normally.
- After an `AsyncTask` task is cancelled, `onPostExecute()` will not be invoked after `doInBackground()` returns. Instead, `onCancelled(Object)` is invoked. The default implementation of `onCancelled(Object)` calls `onCancelled()` and ignores the result.
- By default, in-process tasks are allowed to complete. To allow `cancel()` to interrupt the thread that's executing the task, pass `true` for the value of `mayInterruptIfRunning` .

Limitations of AsyncTask

`AsyncTask` is impractical for some use cases:

- Changes to device configuration cause problems.

When device configuration changes while an `AsyncTask` is running, for example if the user changes the screen orientation, the activity that created the `AsyncTask` is destroyed and re-created. The `AsyncTask` is unable to access the newly created activity, and the results of the `AsyncTask` aren't published.

- Old `AsyncTask` objects stay around, and your app may run out of memory or crash.
  If the activity that created the `AsyncTask` is destroyed, the `AsyncTask` is not destroyed along with it. For example, if your user exits the app after the `AsyncTask` has started, the `AsyncTask` keeps using resources unless you call `cancel()` .

When to use `AsyncTask` :

- Short or interruptible tasks.
- Tasks that don't need to report back to UI or user.
- Low-priority tasks that can be left unfinished.

For all other situations, use `AsyncTaskLoader` , which is part of the `Loader` framework described next.

Loaders

Starting a loader

Use the `LoaderManager` class to manage one or more `Loader` instances within an activity or fragment. Use `initLoader()` to initialize a loader and make it active. Typically, you do this within the activity's `onCreate()` method. For example:

```
// Prepare the loader. Either reconnect with an existing one,
// or start a new one.
getLoaderManager().initLoader(0, null, this);
```

```
getSupportLoaderManager().initLoader(0, null, this);
```

The `initLoader()` method takes three parameters:

- A unique ID that identifies the loader. This ID can be whatever you want.
- Optional arguments to supply to the loader at construction, in the form of a `Bundle` . If a loader already exists, this parameter is ignored.
- A `LoaderCallbacks` implementation, which the `LoaderManager` calls to report loader events. In this example, the local class implements the `LoaderManager.LoaderCallbacks` interface, so it passes a reference to itself, `this` .

The `initLoader()` call has two possible outcomes:

- If the loader specified by the ID already exists, the last loader created using that ID is reused.
- If the loader specified by the ID doesn't exist, `initLoader()` triggers the `onCreateLoader()` method. This is where you implement the code to instantiate and return a new loader.

Note: Whether `initLoader()` creates a new loader or reuses an existing one, the given `LoaderCallbacks` implementation is associated with the loader and is called when the loader's state changes. If the requested loader exists and has already generated data, then the system calls `onLoadFinished()` immediately (during `initLoader()`), so be prepared for this to happen. Put the call to `initLoader()` in `onCreate()` so that the activity can reconnect to the same loader when the configuration changes. That way, the loader doesn't lose the data it has already loaded.

Restarting a loader

When `initLoader()` reuses an existing loader, it doesn't replace the data that the loader contains, but sometimes you want it to. For example, when you use a user query to perform a search and the user enters a new query, you want to reload the data using the new search term. In this situation, use the `restartLoader()` method and pass in the ID of the loader you want to restart. This forces another data load with new input data.

About the `restartLoader()` method:

- `restartLoader()` uses the same arguments as `initLoader()` .
- `restartLoader()` triggers the `onCreateLoader()` method, just as `initLoader()` does when creating a new loader.
- If a loader with the given ID exists, `restartLoader()` restarts the identified loader and replaces its data.
- If no loader with the given ID exists, `restartLoader()` starts a new loader.

LoaderManager callbacks

The `LoaderManager` object automatically calls `onStartLoading()` when creating the loader. After that, the `LoaderManager` manages the state of the loader based on the state of the activity and data, for example by calling `onLoadFinished()` when the data has loaded.

To interact with the loader, use one of the `LoaderManager` callbacks in the activity where the data is needed:

- Call `onCreateLoader()` to instantiate and return a new loader for the given ID.
- Call `onLoadFinished()` when a previously created loader has finished loading. This is typically the point at which you move the data into activity views.
- Call `onLoaderReset()` when a previously created loader is being reset, which makes its data unavailable. At this point your app should remove any references it has to the loader's data.

AsyncTaskLoader

`AsyncTaskLoader` is the loader equivalent of `AsyncTask` . `AsyncTaskLoader` provides a method, `loadInBackground()` , that runs on a separate thread. The results of `loadInBackground()` are automatically delivered to the UI thread, by way of the `onLoadFinished()` `LoaderManager` callback.

AsyncTaskLoader usage

```
public StringListLoader(Context context, String queryString) {
   super(context);
   mQueryString = queryString;
}
```

```
@Override
public List<String> loadInBackground() {
    List<String> data = new ArrayList<String>;
    //TODO: Load the data from the network or from a database
    return data;
}
```

Implementing the callbacks

```
@Override
public Loader<List<String>> onCreateLoader(int id, Bundle args) {
   return new StringListLoader(this, args.getString("queryString"));
}
```

```
public void onLoadFinished(Loader<List<String>> loader, List<String> data) {
    mAdapter.setData(data);
}
```