Sistemul de operare Android

Android este un sistem de operare “open source”, bazat pe Linux, pentru dispozitive mobile precum smartphone-urile și tablete. Android a fost dezvoltat de catre Alliance Open Handset, condus de către Google și alte companii.

Android oferă o abordare unificată a dezvoltării aplicațiilor pentru dispozitive mobile, ceea ce înseamnă că dezvoltatorii trebuie să se dezvolte numai pentru Android, iar aplicațiile lor ar trebui să poată rula pe diferite dispozitive alimentate de Android.

Prima versiune beta de Android Software Development Kit (SDK) a fost lansată de Google în 2007, iar prima versiune comercială, Android 1.0, a fost lansată în septembrie 2008.

Pe data de 27 iunie 2012, la conferința Google I/O, Google a anunțat următoarea versiune Android, 4.1 Jelly Bean. Jelly Bean este o actualizare incrementală, cu scopul principal de a îmbunătăți interfața utilizatorului, atât în ceea ce privește funcționalitatea, cât și performanța.

Codul sursă pentru Android este disponibil sub licențe software gratuite și open source. Google publică majoritatea codului sub licența Apache versiunea 2.0, iar restul, schimbari de Kernel Linux, sub licența GNU General Public Licence versiunea 2.

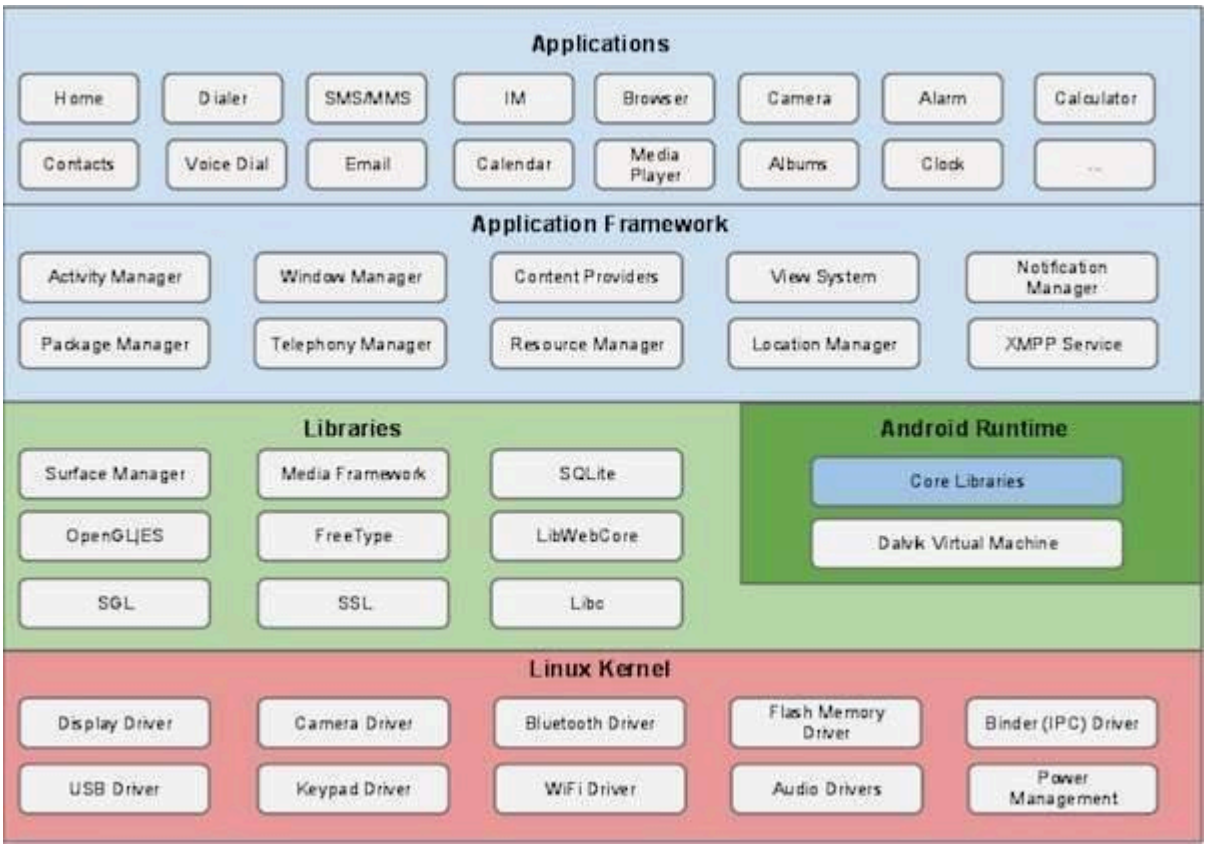
Caracteristicile Sistemului de Operare Android

Android este un sistem de operare puternic care concurează cu Apple 4GS și facilitează funcții excelente. Cateva dintre acestea sunt enumerate mai jos:

Nr	Functie si Descriptie
1	**Beautiful UI / Interfata Stilizata** Ecranul de baza Android OS furnizeaza interfata stilizata si intuitiva.
2	**Connectivity / Conectivitate** GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.
3	**Storage / Stocare** SQLite, o bază de date relațională ușoară, utilizată în scopul de stocare a datelor.
4	**Media support / Suport Media** H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP.
5	**Messaging / Mesagerie** SMS si MMS
6	**Web browser** Bazat pe motorul open-source de layout WebKit, cuplat cu motorul V8 JavaScript din Chrome care acceptă HTML5 și CSS3.
7	**Multi-touch** Android are suport nativ pentru multi-touch, care a fost inițial disponibilă în telefoane, cum ar fi HTC Hero.
8	**Multi-tasking** Utilizatorul poate sări de la o sarcină la alta și diverse aplicații pot funcționa simultan.
9	**Resizable widgets / Widget-uri redimensionabile** Widget-urile sunt redimensionabile, astfel încât utilizatorii le pot extinde pentru a arăta mai mult conținut sau le pot micșora pentru a economisi spațiu.
10	**Multi-Language / Multi-lingvistic** Suporta text unidirectional si bidirectional.

Arhitectura - Android

Sistemul de operare Android este un grup de componente software care este împărțit în patru straturi, după cum se arată mai jos în diagrama de arhitectură.



Kernelul Linux

În ultimele straturi este kernelul Linux **- versiunea 3.6** cu peste 115 patch-uri. Acesta oferă un nivel de abstractizare între hardware-ul dispozitivului și conține toate driverele de hardware esențiale cum ar fi driverele pentru camera, tastatura, afișaj etc. De asemenea, kernelul se ocupă de toate lucrurile la care Linuxul este foarte bun, cum ar fi rețelistică și o gamă largă de drivere de dispozitiv, care usureaza interfațarea cu hardware-ul periferic.

Librarii

Peste Kernelul Linux există un set de biblioteci, inclusiv Webkit Open-Source Web Browser Webkit, Baza de date SQLite, care este un depozit pentru datele din aplicații, bibliotecile de redare și înregistrare audio și video, Librarii SSL responsabile pentru securitatea internetului etc.

Librarii Android

Aceasta categorie incapsuleaza acele librarii bazate pe Java specifice pentru dezvoltarea Android. Exemple de librarii in aceasta categorie include librariile de framework pentru aplicații în plus față de cele care facilitează clădirea interfeței utilizatorilor, desenul grafic și accesul la baza de date. Un rezumat al unor biblioteci cheie Android de bază disponibile dezvoltatorului Android este după cum urmează:

- **android.app** − Oferă acces la modelul de aplicație și este piatra de temelie a tuturor aplicațiilor Android.
- **android.content** − Facilitează accesul, publicarea și mesageria conținutului între aplicații și componente ale aplicațiilor.
- **android.database** − Folosit pentru a accesa datele publicate de furnizorii de conținut și include clase de gestionare a bazelor de date SQLite.
- **android.opengl** − O interfață Java a API-ului grafic 3D OpenGL ES 3D.
- **android.os** − Oferă aplicații cu acces la serviciile standard ale sistemului de operare, inclusiv mesaje, servicii de sistem și comunicare interprocesată.
- **android.text** − Folosit pentru a crea și manipula textul pe un afișaj al dispozitivului.
- **android.view** − Blocurile fundamentale ale interfețelor utilizatorilor de aplicații.
- **android.widget** − O colecție bogată de componente de interfață de utilizator pre-construite, cum ar fi butoane, etichete, vizualizări de liste, manageri de layout, butoane radio etc.
- **android.webkit** − Un set de clase destinate să permită construirea capabilităților de navigare pe web în aplicații.

Având în vedere bibliotecile de bază bazate pe JAVA din Runtime Android, este momentul să ne îndreptăm atenția asupra bibliotecilor bazate pe C/C ++ conținute în acest strat al stivei de software Android.

Android Runtime

Aceasta este a treia secțiune a arhitecturii și disponibilă pe cel de-al doilea strat. Această secțiune oferă o componentă cheie numită Dalvik Virtual Machine, care este un fel de mașină virtuală Java special concepută și optimizată pentru Android.

Dalvik VM utilizează caracteristicile de bază Linux, cum ar fi gestionarea memoriei și multi-threading, care este intrinsecă în limba Java. Dalvik VM permite fiecărei aplicații Android să ruleze în propriul proces, cu propria sa mașină virtuală Dalvik.

Alternativa este ART: https://intexsoft.com/blog/android-runtime-environment-dvm-vs-art/

Android Runtime oferă, de asemenea, un set de biblioteci de bază care permit dezvoltatorilor de aplicații Android să scrie aplicații Android folosind limba standard de programare Java.

Application Framework

Stratul-cadru de aplicații (The Application Framework) oferă multe servicii de nivel superior aplicațiilor sub formă de clase Java. Dezvoltatorii de aplicații li se permite să utilizeze aceste servicii în aplicațiile lor.

Framework-ul Android include urmatoarele servicii cheie:

- **Activity Manager** − Controlează toate aspectele ciclului de viață al aplicației și al stivei de activitate.

- **Content Providers** − Permite aplicațiilor să publice și să partajeze date cu alte aplicații.
- **Resource Manager** − Oferă acces la resurse încorporate non-cod, cum ar fi șiruri de caractere, setări de culori și layout-uri de interfață utilizator.
- **Notifications Manager** − Permite aplicațiilor să afișeze alerte și notificări către utilizator.
- **View System** − Un set extensibil de vizualizări utilizate pentru a crea interfețe de utilizare a aplicațiilor.

Componentele aplicațiilor sunt blocurile esențiale ale unei aplicații Android. Aceste componente sunt cuplate în mod liber de fișierul manifest de aplicație Androidmanifest.xml care descrie fiecare componentă a aplicației și modul în care interacționează.

Există următoarele patru componente principale care pot fi utilizate într-o aplicație Android:

No	Componente
1	**Activities / Activitati** Acestea dictează UI și manipulează interacțiunea utilizatorului cu ecranul telefonului inteligent.
2	**Services / Servicii** Se ocupă de procesarea de fundal asociată cu o aplicație.
3	**Broadcast Receivers / Receptoare de transmisii** Manipulează comunicarea dintre OS Android și aplicații.
4	**Content Providers / Furnizori de conținut** Manipulează problemele de gestionare a datelor și bazelor de date.

Activitati

O activitate reprezintă un singur ecran cu o interfață de utilizator, activitatea în scurt efectuează acțiuni pe ecran.De exemplu, o aplicație de e-mail ar putea avea o activitate care prezintă o listă de e-mailuri noi, o altă activitate pentru a compune un e-mail și o altă activitate pentru citirea e-mailurilor.Dacă o aplicație are mai mult de o activitate, atunci una dintre ele trebuie marcată ca activitate care este prezentată atunci când este lansată aplicația.

O activitate este implementată ca o subclasă a clasei de activitate după cum urmează -

```
public class MainActivity extends Activity {
}
```

Servicii

Un serviciu este o componentă care rulează în fundal pentru a efectua operații de lungă durată.De exemplu, un serviciu ar putea reda muzică în fundal, în timp ce utilizatorul se

află într-o aplicație diferită sau ar putea aduce date prin rețea fără a bloca interacțiunea cu o activitate.

Un serviciu este implementat ca o subclasă a clasei de servicii după cum urmează -

```
public class MyService extends Service {
}
```

Broadcast Receivers / Receptoare de transmisii

Receptoarele de transmisii răspund pur și simplu la mesajele transmise din alte aplicații sau din sistem.De exemplu, aplicațiile pot, de asemenea, să inițieze emisiunile pentru a permite alte aplicații să știe că unele date au fost descărcate pe dispozitiv și sunt disponibile pentru a le utiliza, deci acest receptor care va intercepta această comunicare și va iniția acțiuni adecvate.

Un receptor de transmisie este implementat ca o subclasă a clasei BroadcastReceiver și fiecare mesaj este transmis ca un obiect de intenție.

```
public class MyReceiver extends BroadcastReceiver {
   public void onReceive(context,intent){}
}
```

Furnizori de Continut

O componentă furnizor de conținut furnizează date de la o aplicație la altele la cerere. Astfel de solicitări sunt gestionate de metodele clasei ContentSolver. Datele pot fi stocate în sistemul de fișiere, baza de date sau în altă parte în întregime.
Un furnizor de conținut este implementat ca o subclasă a clasei Contentprovider și trebuie să implementeze un set standard API care să permită alte aplicații să efectueze tranzacții.

```
public class MyContentProvider extends ContentProvider {
   public void onCreate(){}
}
```

Vom trece prin aceste etichete în detaliu în timp ce acoperim componentele aplicației în capitole individuale.

Componente Aditionale

Există componente suplimentare care vor fi utilizate în construcția entităților menționate mai sus, logica lor și cablarea între ele. Aceste componente sunt:

No	Componente si Descriptii
1	**Fragments / Fragmente** Reprezintă o parte din interfața cu utilizatorul într-o activitate.

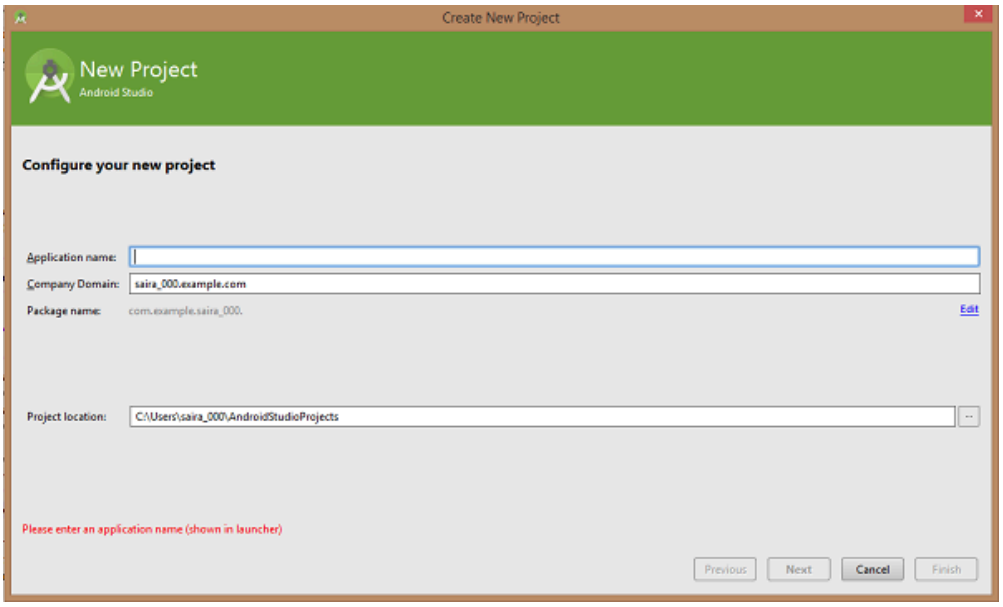
2	**Views / Vizualizari** Elemente UI care sunt plasate pe ecran, inclusiv butoane, lista de formulare etc.
3	**Layouts / Straturi** Vizualizeaza ierarhiile care controlează formatul ecranului și apariția elementelor de UI (view).
4	**Intents / Intenturi** Mesaje de conectare a componentelor împreună.
5	**Resources / Resurse** Elemente externe, cum ar fi șiruri, constante și imagini desenate.
6	**Manifest / Manifestare** Fișier de configurare pentru aplicație.

Înainte de a începe să scrieți primul exemplu folosind Android SDK, trebuie să vă asigurați că ați configurat în mod corespunzător mediul de dezvoltare Android si sa aveti conectata o masina virtuala cu Android sau un sistem real.

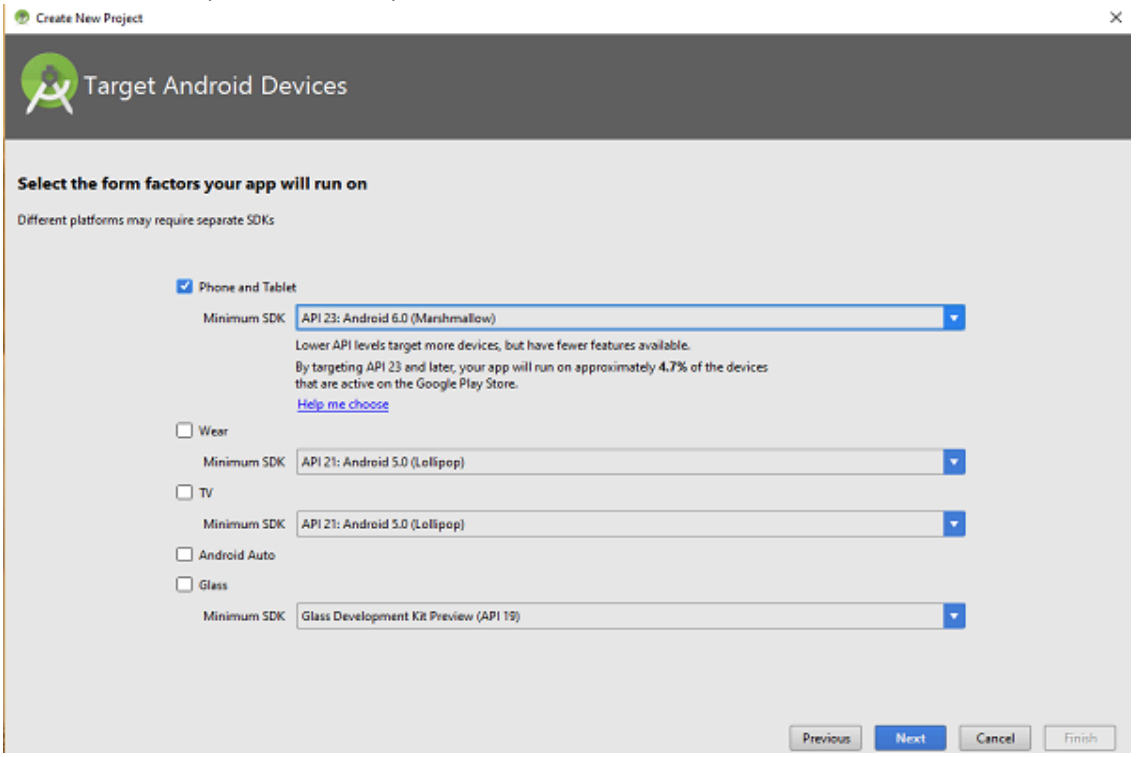
Creeaza o aplicatie Android

Primul pas este crearea unei aplicații simple Android utilizând Android Studio. Când faceți clic pe pictograma Android Studio, acesta va afișa ecranul după cum se arată mai jos.

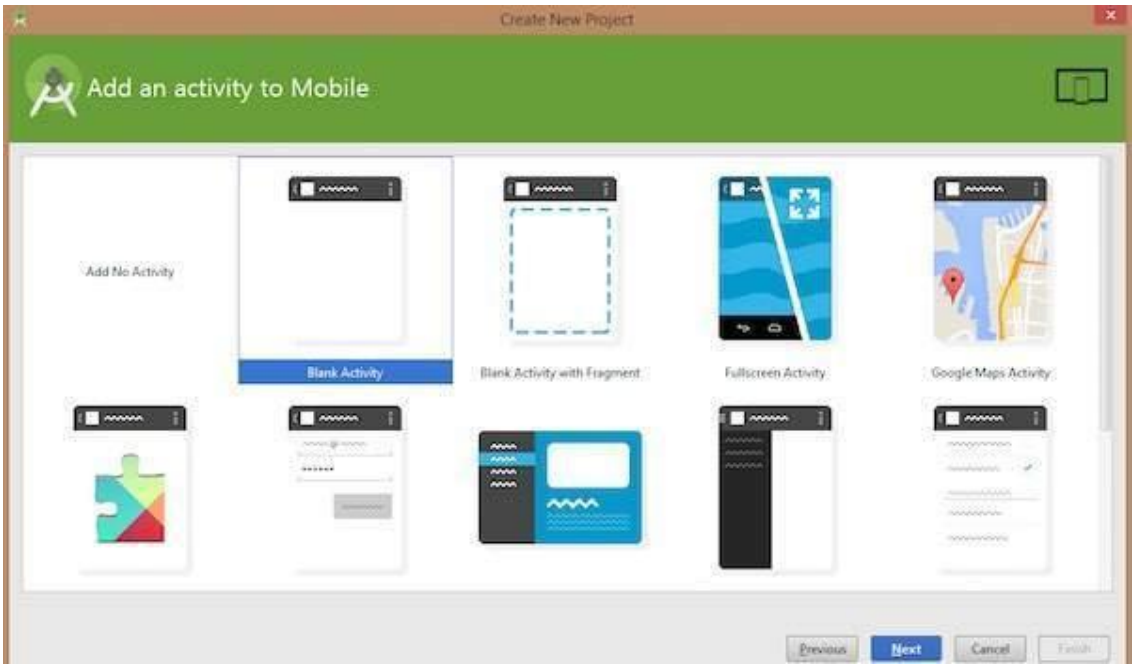
Puteți începe dezvoltarea aplicațiilor prin apelarea “Start a new Android Studio Project”. Într-un nou cadru de instalare ar trebui să ceară numele aplicației, informații despre pachete și locația proiectului.



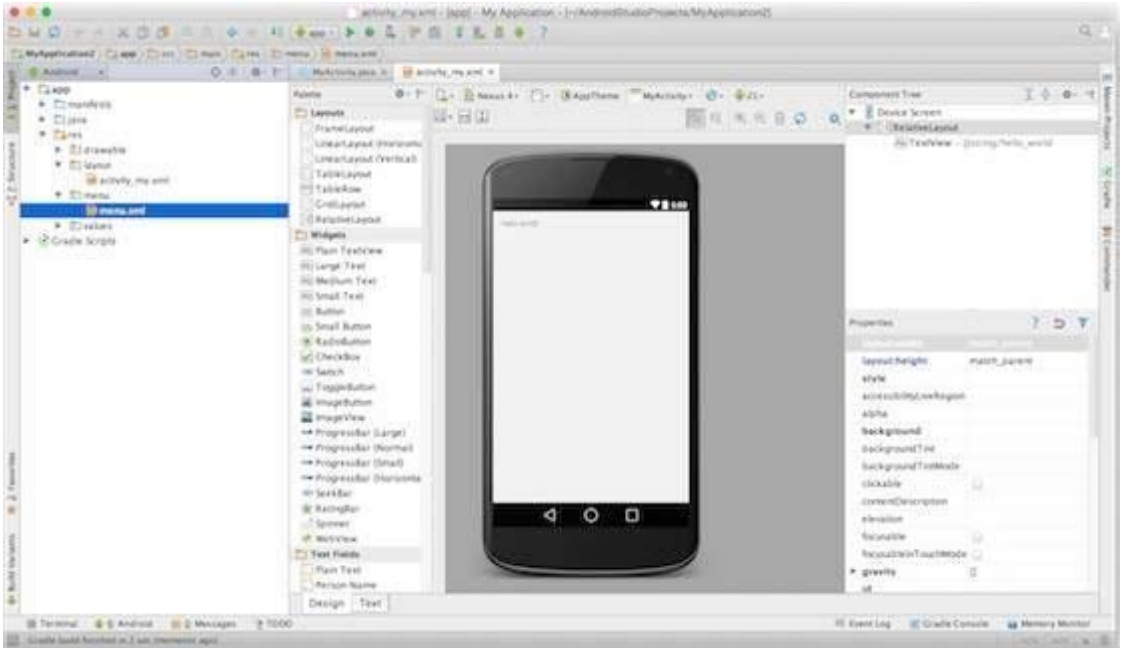
După ce ați introdus numele aplicației, va trebui sa selectati versiunea pe care rulează aplicația dvs., aici trebuie să specificați SDK-ul minim. În tutorialul nostru, am declarat ca API23: Android 6.0 (Mashmallow)



Următorul nivel de instalare trebuie să conțină selectarea activității pe mobil, specifică aspectul implicit pentru aplicații.

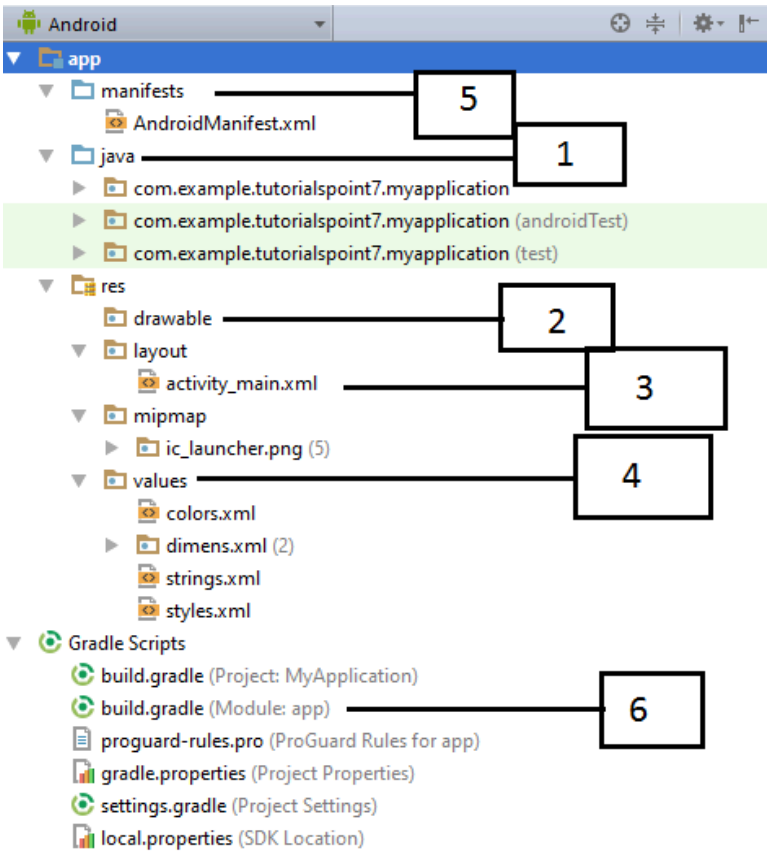


În etapa finală vom deschide instrumentul de dezvoltare pentru a scrie codul aplicației.



Anatomia aplicației android

Înainte de a rula aplicația, ar trebui să fiți conștienți de câteva directoare și fișiere din proiectul Android



No.	Folder, Fisier & Descriere
1	**Java** Acesta conține fișierele sursă .java pentru proiectul dvs. În mod implicit, include un fișier sursă MainActivity.java având o clasă de activitate care rulează atunci când aplicația dvs. este lansată utilizând pictograma aplicației.
2	**res/drawable-hdpi** Acesta este un director pentru obiecte desenabile care sunt proiectate pentru ecrane cu densitate mare.
3	**res/layout** Acesta este un director pentru fișierele care definesc interfața de utilizare a aplicației.
4	**res/values** Acesta este un director pentru alte fișiere XML care conțin o colecție de resurse, cum ar fi definițiile șirurilor și culorilor.
5	**AndroidManifest.xml** Acesta este fișierul manifest care descrie caracteristicile fundamentale ale aplicației și definește fiecare dintre componentele sale.
6	**Build.gradle** Acesta este un fișier generat automat care conține compileSdkVersion, buildToolsVersion, applicationId, minSdkVersion, targetSdkVersion, versionCode și versionName

Următoarea secțiune va oferi o scurtă prezentare generală a fișierelor importante ale aplicației.

Fișierul Main Activity

Codul principal de activitate este un fișier Java MainActivity.java. Acesta este fișierul real al aplicației care în cele din urmă este convertit într-un executabil Dalvik și rulează aplicația dvs. Următorul este codul implicit generat automat pentru aplicatia Hello World!

```
package com.example.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

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

Aici, R.layout.activity_main se referă la fișierul activity_main.xml aflat în folderul res/layout. Metoda onCreate () este una dintre numeroasele metode care se calculează atunci când o activitate este încărcată.

Fișierul Manifest

Pentru orice componentă pe care o dezvoltați ca parte a aplicației dvs., trebuie să declarați toate componentele sale într-un manifest.xml care se află la rădăcina directorului proiectului aplicației. Acest fișier funcționează ca o interfață între sistemul de operare Android și aplicația dvs., deci dacă nu declarați componenta în acest fișier, atunci acesta nu va fi luata în considerare de sistemul de operare. De exemplu, un fișier manifest implicit va arăta astfel :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.t201.myapplication">

   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:supportsRtl="true"
      android:theme="@style/AppTheme">

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

```
    </application>
</manifest>
```

Aici etichetele <application> ... </application> au inclus componentele legate de aplicație. Atributul Android:icon va indica pictograma aplicației disponibilă in res/drawable-hdpi. Aplicația utilizează imaginea numită ic_launcher.png situată în folderul drawable.

Eticheta <activity> este utilizata pentru a specifica o activitate si atributul android:name specifica specifică numele clasei complet calificate a subclasei Activity și atributele android: label specifică un șir de utilizat ca etichetă pentru activitate. Puteți specifica mai multe activități utilizând etichete < activity >.

Acțiunea pentru filtrul de intenție este denumită android.intent.action.MAIN pentru a indica faptul că această activitate servește drept punct de intrare pentru aplicație. Categoria pentru filtrul de intenție este denumită android.intent.category.LAUNCHER pentru a indica faptul că aplicația poate fi lansată de pe pictograma de lansare a dispozitivului.

@String se referă la fișierul strings.xml explicat mai jos. Prin urmare, @ string / app_name se referă la șirul app_name definit în fișierul strings.xml, care este „HelloWorld”. În mod similar, alte șiruri sunt populate în aplicație.

Urmează lista etichetelor pe care le veți utiliza în fișierul dvs. manifest pentru a specifica diferite componente ale aplicației Android

- <activity>pentru activitati
- <service> pentru servicii
- <receiver> pentru receptoare
- <provider> pentru furnizorii de conținut

Fisierul Strings

Fișierul strings.xml se află în folderul res/values și conține tot textul pe care îl folosește aplicația dvs. De exemplu, numele butoanelor, etichetelor, textului implicit și tipurilor similare de șiruri intră în acest fișier. Acest fișier este responsabil pentru conținutul lor textual. De exemplu, un fișier șiruri implicit va arăta ca următorul fișier

```
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_main">MainActivity</string>
</resources>
```

Fișierul Layout

Activity_main.xml este un fișier de aspect disponibil în directorul res / layout, la care se referă aplicația dvs. atunci când construiți interfața sa. Veți modifica acest fișier foarte frecvent pentru a schimba aspectul aplicației dvs. Pentru aplicatia „Hello World!”, acest fișier va avea următorul conținut legat de aspectul implicit

```
<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" >

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:layout_centerVertical="true"
      android:padding="@dimen/padding_medium"
      android:text="@string/hello_world"
      tools:context=".MainActivity" />

</RelativeLayout>
```

Acesta este un exemplu de RelativeLayout simplu pe care îl vom studia într-un capitol separat. TextView este un control Android utilizat pentru a construi interfața grafică și are diverse atribute, cum ar fi android: layout_width, android: layout_height etc., care sunt folosite pentru a-și seta lățimea și înălțimea, etc. @String se referă la fișierul strings.xml aflat în folderul res / values. Prin urmare, @ string / hello_world se referă la șirul hello definit în fișierul strings.xml, care este „Hello World!”.

Rularea aplicației

Pentru a rula aplicația pe care tocmai am creat-o aveti nevoie de un AVD creat în timp ce realizați setarea mediului. Pentru a rula aplicația din Android Studio, deschideți unul dintre fișierele de activitate ale proiectului dvs. și faceți clic pe pictograma Executare din bara de instrumente. Android Studio instalează aplicația pe AVD și o pornește și dacă totul este în regulă cu configurarea și aplicația dvs., aceasta se va afișa după fereastra Emulator.

Există multe alte elemente pe care le folosiți pentru a crea o aplicație Android bună. În afară de codificarea aplicației, aveți grijă de diverse alte resurse, cum ar fi conținutul static pe care îl folosește codul dvs., cum ar fi bitmap-urile, culorile, definițiile aspectului, șirurile interfeței utilizator, instrucțiunile de animație și multe altele. Aceste resurse sunt întotdeauna întreținute separat în diferite subdirectoare din directorul res / al proiectului. Acest tutorial vă va explica cum puteți organiza resursele aplicației, specifica resurse alternative și le puteți accesa în aplicațiile dvs.

Organizare resurse in Android Studio

```
MyProject/
   app/
      manifest/
         AndroidManifest.xml
   java/
      MyActivity.java
      res/
         drawable/
            icon.png
         layout/
            activity_main.xml
            info.xml
         values/
            strings.xml
```

No	Director & tipul de resursa
1	**anim/** Fișiere XML care definesc animațiile de proprietate. Acestea sunt salvate în res / anim / folder și accesate din clasa R.anim.
2	**color/** Fișiere XML care definesc o listă de culori a stării. Acestea sunt salvate în res / color / și accesate din clasa R.color.
3	**drawable/** Fișiere imagine, cum ar fi .png, .jpg, .gif sau fișiere XML care sunt compilate în hărți de biți, liste de stări, forme, animații desenabile. Sunt salvate în res / drawable / și accesate din clasa R.drawable.
4	**layout/** Fișiere XML care definesc un aspect al interfeței cu utilizatorul. Acestea sunt salvate în res / layout / și accesate din clasa R.layout.
5	**menu/** Fișiere XML care definesc meniurile aplicației, cum ar fi un meniu Opțiuni, un meniu contextual sau un submeniu. Sunt salvate în res / meniu / și accesate din clasa R.menu.

6	**raw/** Fișiere arbitrare de salvat în forma lor brută. Trebuie să apelați Resources.openRawResource () cu ID-ul resursei, care este numele R.raw.filename pentru a deschide astfel de fișiere brute.
7	**values/** Fișiere XML care conțin valori simple, cum ar fi șiruri, numere întregi și culori. De exemplu, iată câteva convenții de nume de fișier pentru resursele pe care le puteți crea în acest director - • arrays.xml pentru matricele de resurse și accesate din clasa R.array. • integers.xml pentru resurse întregi și accesate din clasa R.integer. • bools.xml pentru resursa booleană și accesat din clasa R.bool. • colors.xml pentru valorile culorilor și accesat din clasa R.color. • dimens.xml pentru valorile dimensiunii și accesat din clasa R.dimen. • strings.xml pentru valorile șirurilor și accesate din clasa R.string. • styles.xml pentru stiluri și accesat din clasa R.style.
8	**xml/** Fișiere XML arbitrare care pot fi citite la runtime apelând Resources.getXML (). Aici puteți salva diverse fișiere de configurare care vor fi utilizate în timpul rulării.

Resurse alternative

Aplicația dezvoltata ar trebui să ofere resurse alternative pentru a accepta configurații specifice ale dispozitivului. De exemplu, ar trebui să includeți resurse alternative desenabile (adică imagini) pentru rezoluții diferite ale ecranului și resurse șir alternative pentru diferite limbi. În timpul rulării, Android detectează configurația curentă a dispozitivului și încarcă resursele adecvate pentru aplicația dvs.
Pentru a specifica alternative specifice configurației pentru un set de resurse, urmați pașii următori

• Creați un nou director în res / denumit în formatul <resources_name> - <config_qualifier>. Aici resources_name vor fi oricare dintre resursele menționate în tabelul de mai sus, cum ar fi aspectul, desenabil etc. Puteți verifica documentația oficială pentru o listă completă a calificărilor pentru diferite tipuri de resurse.

• Salvați resursele alternative respective în acest nou director. Fișierele de resurse trebuie să fie denumite exact la fel ca fișierele de resurse implicite, așa cum se arată în exemplul de mai jos, dar aceste fișiere vor avea conținut specific alternativei. De exemplu, deși numele fișierului imagine va fi același, dar pentru ecranul de înaltă rezoluție, rezoluția acestuia va fi mare.

Mai jos este un exemplu care specifică imagini pentru un ecran implicit și imagini alternative pentru ecranul de înaltă rezoluție.

```
MyProject/
   app/
      manifest/
         AndroidManifest.xml
   java/
      MyActivity.java
      res/
         drawable/
            icon.png
            background.png
```

```
        drawable-hdpi/
            icon.png
            background.png
        layout/
            activity_main.xml
            info.xml
        values/
            strings.xml
```

Mai jos este un alt exemplu care specifică aspectul pentru o limbă implicită și aspectul alternativ pentru limba arabă.

```
MyProject/
    app/
        manifest/
            AndroidManifest.xml
    java/
        MyActivity.java
        res/
            drawable/
                icon.png
                background.png
            drawable-hdpi/
                icon.png
                background.png
            layout/
                activity_main.xml
                info.xml
            layout-ar/
                main.xml
            values/
                strings.xml
```

Accesarea resurselor

În timpul dezvoltării aplicației va trebui să accesați resursele definite fie în codul dvs., fie în fișierele XML de aspect. Următoarea secțiune explică cum să accesați resursele în ambele scenarii

Accesarea Resurselor in Cod

Când aplicația dvs. Android este compilată, se generează o clasă R, care conține ID-uri de resurse pentru toate resursele disponibile în res / directorul dvs. Puteți utiliza clasa R pentru a accesa resursa utilizând subdirectorul și numele resursei sau direct ID-ul resursei.

Exemplu

Pentru a accesa *res/drawable/myimage.png* si a utiliza ImageView trebuie sa folosesti urmatorul cod.

```
ImageView imageView = (ImageView) findViewById(R.id.myimageview);
imageView.setImageResource(R.drawable.myimage);
```

Aici prima linie a codului folosește R.id.myimageview pentru a obține ImageView definit cu id myimageview într-un fișier Layout. A doua linie de cod folosește R.drawable.myimage pentru a obține o imagine cu numele myimage disponibilă în subdirectorul desenabil sub / res.

Exemplu

Luați în considerare următorul exemplu în care res / values / strings.xml are următoarea definiție –

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string  name="hello">Hello, World!</string>
</resources>
```

Acum puteți seta textul pe un obiect TextView cu msg ID utilizând un ID resursă după cum urmează:

```
TextView msgTextView = (TextView) findViewById(R.id.msg);
msgTextView.setText(R.string.hello);
```

Exemplu

Luați în considerare un layout res / layout / activity_main.xml cu următoarea definiție –

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

   <TextView android:id="@+id/text"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello, I am a TextView" />

   <Button android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello, I am a Button" />

</LinearLayout>
```

Acest cod de aplicație va încărca acest aspect pentru o activitate, în metoda onCreate () după cum urmează –

```
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
}
```

Accesarea resurselor în XML

Luați în considerare următorul fișier res resource XML / valori / strings.xml care include o resursă de culoare și o resursă string –

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="opaque_red">#f00</color>
   <string name="hello">Hello!</string>
</resources>
```

Acum puteți utiliza aceste resurse în următorul fișier de aspect pentru a seta culoarea textului și șirul de text după cum urmează –

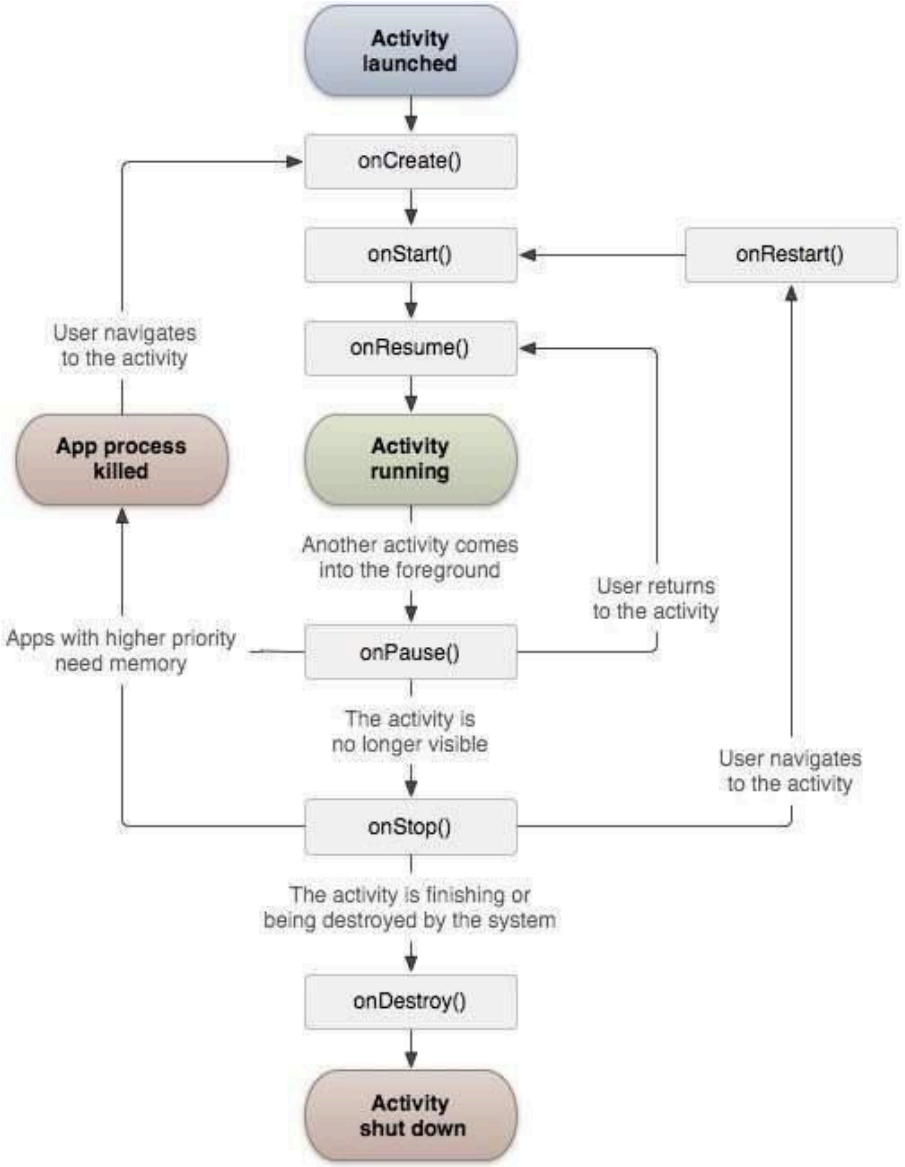
```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:textColor="@color/opaque_red"
   android:text="@string/hello" />
```

Acum, dacă veți parcurge încă o dată capitolul anterior, unde am explicat **Hello World!** exemplu și sunt sigur că veți avea o mai bună înțelegere a tuturor conceptelor explicate în acest capitol. Așadar, recomand cu tărie să verific capitolul anterior pentru un exemplu de lucru și să verific cum am folosit diverse resurse la nivel foarte de bază.

Activități în Android

O activitate reprezintă un singur ecran cu o interfață cu utilizatorul la fel ca fereastra sau cadrul Java. Activitatea Android este subclasa clasei ContextThemeWrapper.

Dacă ați lucrat cu limbaj de programare C, C ++ sau Java, atunci trebuie să fi văzut că programul dvs. pornește de la funcția main (). În mod foarte similar, sistemul Android își inițiază programul cu o activitate începând cu un apel pe metoda de apel invers onCreate (). Există o secvență de metode de apel invers care pornesc o activitate și o secvență de metode de apel invers care distrug o activitate așa cum se arată în diagrama ciclului de viață al activității de mai jos:



Clasa Activity definește următoarele apeluri de apel, adică evenimente. Nu este nevoie să implementați toate metodele de apelare inversă. Cu toate acestea, este important să le înțelegeți pe fiecare și să le implementați pe cele care vă asigură că aplicația dvs. se comportă așa cum se așteaptă utilizatorii.

No	Functie apelata & Descriere
1	**onCreate()** Acesta este primul apel invers și apelat atunci când activitatea este creată pentru prima dată.
2	**onStart()** Acest apel invers este apelat atunci când activitatea devine vizibilă pentru utilizator.
3	**onResume()** Aceasta este apelata atunci când utilizatorul începe să interacționeze cu aplicația.
4	**onPause()** Activitatea întreruptă nu primește introducerea utilizatorului și nu poate executa niciun cod și este apelată atunci când activitatea curentă este întreruptă și activitatea anterioară este reluată.
5	**onStop()** Acest apel invers este apelat atunci când activitatea nu mai este vizibilă.
6	**onDestroy()** Acest apel invers este apelat înainte ca activitatea să fie distrusă de sistem.
7	**onRestart()** Acest apel invers este apelat la repornirea activității după oprirea acestuia.

Exemplu

Acest exemplu vă va conduce prin pași simpli pentru a afișa ciclul de viață al activității aplicației Android. Urmați pașii următori pentru a modifica aplicația Android pe care am creat-o în capitolul Exemplu Hello World –

Pas	Descriere
1	Veți utiliza Android Studio pentru a crea o aplicație Android și o veți numi HelloWorld sub un pachet com.example.helloworld așa cum este explicat în capitolul Exemplu Hello World.
2	Modificați fișierul de activitate principal MainActivity.java așa cum este explicat mai jos. Păstrați restul fișierelor neschimbate.
3	Rulați aplicația pentru a lansa emulatorul Android și verificați rezultatul modificărilor efectuate în aplicație.

Urmează conținutul fișierului de activitate principal modificat src / com.example.helloworld / MainActivity.java. Acest fișier include fiecare dintre metodele fundamentale ale ciclului de viață. Metoda Log.d () a fost utilizată pentru a genera mesaje jurnal –

```
package com.example.helloworld;

import android.os.Bundle;
import android.app.Activity;
import android.util.Log;

public class MainActivity extends Activity {
   String msg = "Android : ";

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
```

```
        Log.d(msg, "The onCreate() event");
    }

    /** Called when the activity is about to become visible. */
    @Override
    protected void onStart() {
        super.onStart();
        Log.d(msg, "The onStart() event");
    }

    /** Called when the activity has become visible. */
    @Override
    protected void onResume() {
        super.onResume();
        Log.d(msg, "The onResume() event");
    }

    /** Called when another activity is taking focus. */
    @Override
    protected void onPause() {
        super.onPause();
        Log.d(msg, "The onPause() event");
    }

    /** Called when the activity is no longer visible. */
    @Override
    protected void onStop() {
        super.onStop();
        Log.d(msg, "The onStop() event");
    }

    /** Called just before the activity is destroyed. */
    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(msg, "The onDestroy() event");
    }
}
```

O clasă de activitate încarcă toată componenta UI utilizând fișierul XML disponibil în folderul res / layout al proiectului. Instrucțiunea următoare încarcă componentele UI din fișierul res / layout / activity_main.xml:

```
setContentView(R.layout.activity_main);
```

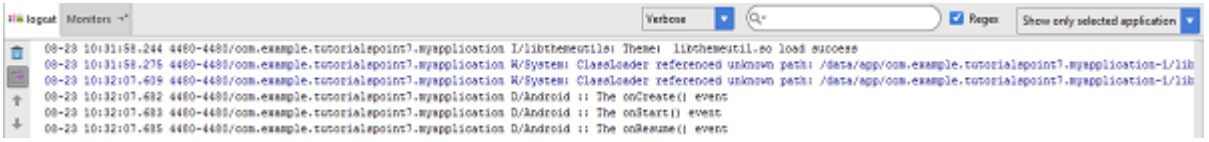
O aplicație poate avea una sau mai multe activități fără restricții. Fiecare activitate pe care o definiți pentru aplicația dvs. trebuie declarată în fișierul AndroidManifest.xml și activitatea principală pentru aplicația dvs. trebuie declarată în manifest cu un <intent-filter> care include acțiunea PRINCIPALĂ și categoria LAUNCHER după cum urmează:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.t201.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
```

```
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

Dacă acțiunea PRINCIPALĂ sau categoria LAUNCHER nu sunt declarate pentru una dintre activitățile dvs., pictograma aplicației dvs. nu va apărea în lista de aplicații din ecranul de întâmpinare.
Să încercăm să rulăm aplicatia in AVD-ul creat în timp ce efectuați configurarea mediului. Pentru a rula aplicația din Android Studio, deschideți unul dintre fișierele de activitate ale proiectului dvs. și faceți clic pe pictograma RUN din bara de instrumente. Android Studio instalează aplicația pe AVD și o pornește și, dacă totul este în regulă cu configurarea și aplicația dvs., va afișa fereastra Emulator și ar trebui să vedeți următoarele mesaje de jurnal în fereastra LogCat din Android Studio –

```
08-23 10:32:07.682 4480-4480/com.example.helloworld D/Android :: The onCreate() event
08-23 10:32:07.683 4480-4480/com.example.helloworld D/Android :: The onStart() event
08-23 10:32:07.685 4480-4480/com.example.helloworld D/Android :: The onResume() event
```



Să încercăm să facem clic pe butonul de blocare a ecranului de pe emulatorul Android și va genera următoarele mesaje de evenimente în fereastra LogCat din Android Studio:

```
08-23 10:32:53.230 4480-4480/com.example.helloworld D/Android :: The onPause() event
08-23 10:32:53.294 4480-4480/com.example.helloworld D/Android :: The onStop() event
```

Permiteți-ne să încercăm din nou să vă deblocăm ecranul pe emulatorul Android și va genera următoarele mesaje de evenimente în fereastra LogCat din studioul Android:

```
08-23 10:34:41.390 4480-4480/com.example.helloworld D/Android :: The onStart() event
08-23 10:34:41.392 4480-4480/com.example.helloworld D/Android :: The onResume() event
```

Apoi, haideți să încercăm din nou să faceți clic pe butonul BACK de pe emulatorul Android și va genera următoarele mesaje de evenimente în fereastra LogCat din studioul Android și aceasta completează Ciclul de viață al activității pentru o aplicație Android.

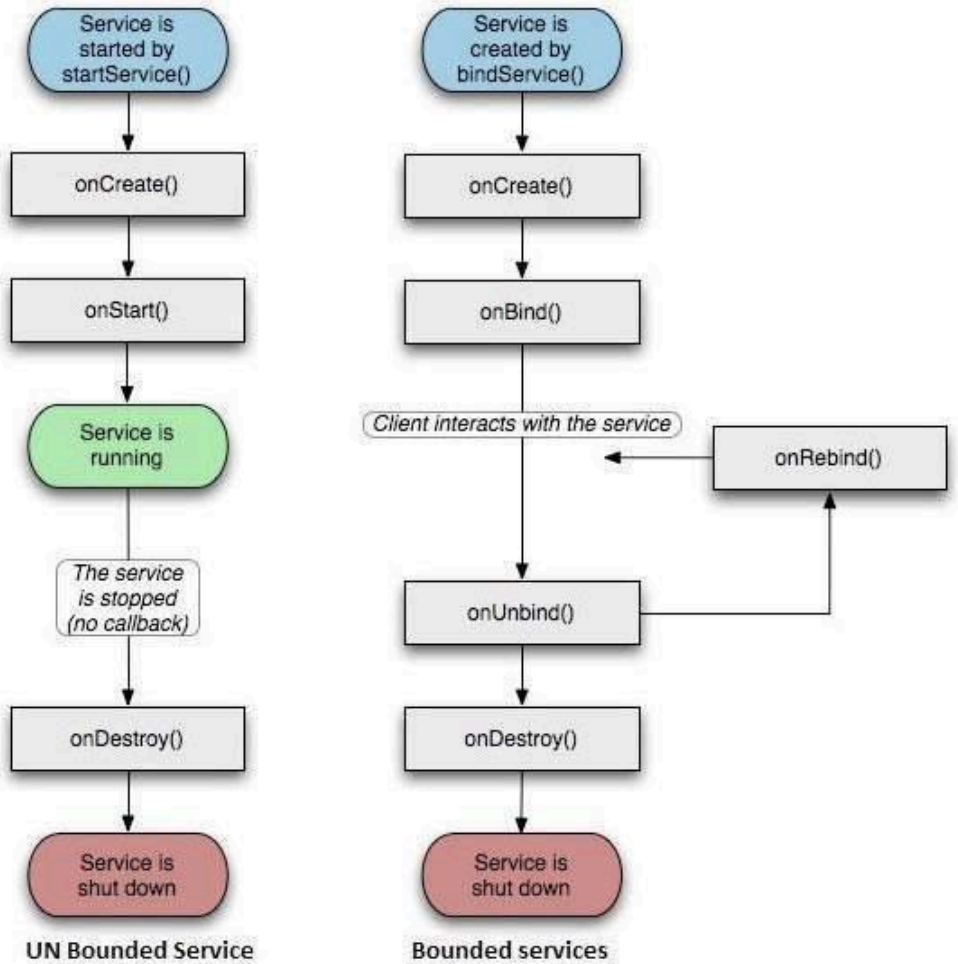
```
08-23 10:37:24.806 4480-4480/com.example.helloworld D/Android :: The onPause() event
08-23 10:37:25.668 4480-4480/com.example.helloworld D/Android :: The onStop() event
08-23 10:37:25.669 4480-4480/com.example.helloworld D/Android :: The onDestroy() event
```

Servicii Android

Un serviciu este o componentă care rulează în fundal pentru a efectua operațiuni de lungă durată fără a fi nevoie să interacționați cu utilizatorul și funcționează chiar dacă aplicația este distrusă. Un serviciu poate lua în esență două stări:

Sr.No.	Stare & Descriere
1	**Start** Un serviciu este pornit atunci când o componentă a aplicației, cum ar fi o activitate, îl pornește apelând startService(). Odată pornit, un serviciu poate rula în fundal pe termen nelimitat, chiar dacă componenta care a pornit este distrusă.
2	**Bind** Un serviciu este legat atunci când o componentă a aplicației se leagă de el apelând bindService(). Un serviciu conectat la o aplicatie oferă o interfață client-server care permite componentelor să interacționeze cu serviciul, să trimită cereri, să obțină rezultate și chiar să facă acest lucru în toate procesele cu comunicare interproces (IPC).

Un serviciu are metode de returnare a ciclului de viață pe care le puteți implementa pentru a monitoriza modificările în starea serviciului și puteți efectua lucrări în etapa corespunzătoare. Următoarea diagramă din stânga arată ciclul de viață când serviciul este creat cu startService () și diagrama din dreapta arată ciclul de viață când serviciul este creat cu bindService (): (imagine, prin amabilitatea: android.com)



Pentru a crea un serviciu, creați o clasă Java care extinde clasa de bază Service sau una dintre subclasele sale existente. Clasa de bază Service definește diverse metode de apelare, iar cele mai importante sunt prezentate mai jos. Nu este nevoie să implementați toate metodele de apelare inversă. Cu toate acestea, este important să le înțelegeți pe fiecare și să le implementați pe cele care vă asigură că aplicația dvs. se comportă așa cum se așteaptă utilizatorii.

No.	Functie de tratare & Descriere
1	**onStartCommand()** Sistemul apelează această metodă atunci când o altă componentă, cum ar fi o activitate, solicită pornirea serviciului, apelând startService (). Dacă implementați această metodă, este responsabilitatea dvs. să opriți serviciul atunci când se termină activitatea, apelând metodele stopSelf () sau stopService ().
2	**onBind()** Sistemul apelează această metodă atunci când o altă componentă dorește să se lege cu serviciul apelând bindService (). Dacă implementați această metodă, trebuie să furnizați o interfață pe care clienții o utilizează pentru a comunica cu serviciul, prin returnarea unui obiect IBinder. Trebuie să implementați întotdeauna această metodă, dar dacă nu doriți să permiteți legarea, atunci ar trebui să returnați nul.
3	**onUnbind()** Sistemul apelează această metodă atunci când toți clienții s-au deconectat de la o anumită interfață publicată de serviciu.
4	**onRebind()** Sistemul apelează această metodă atunci când clienții noi s-au conectat la serviciu, după ce a fost notificat anterior că toți s-au deconectat în onUnbind (Intent).
5	**onCreate()** Sistemul apelează această metodă atunci când serviciul este creat pentru prima dată utilizând onStartCommand () sau onBind (). Acest apel este necesar pentru a efectua o singură configurare.
6	**onDestroy()** Sistemul apelează această metodă atunci când serviciul nu mai este utilizat și este distrus. Serviciul dvs. ar trebui să pună în aplicare acest lucru pentru a curăța orice resurse, cum ar fi thread-uri, ascultători înregistrați, receptoare etc.

Structura urmatoare demonstreaza fiecare dintre metodele ciclului de viata.

```
package com.t201;

import android.app.Service;
import android.os.IBinder;
import android.content.Intent;
import android.os.Bundle;

public class HelloService extends Service {

   /** indicates how to behave if the service is killed */
   int mStartMode;

   /** interface for clients that bind */
   IBinder mBinder;
```

```
   /** indicates whether onRebind should be used */
   boolean mAllowRebind;

   /** Called when the service is being created. */
   @Override
   public void onCreate() {

   }

   /** The service is starting, due to a call to startService() */
   @Override
   public int onStartCommand(Intent intent, int flags, int startId) {
      return mStartMode;
   }

   /** A client is binding to the service with bindService() */
   @Override
   public IBinder onBind(Intent intent) {
      return mBinder;
   }

   /** Called when all clients have unbound with unbindService() */
   @Override
   public boolean onUnbind(Intent intent) {
      return mAllowRebind;
   }

   /** Called when a client is binding to the service with
bindService()*/
   @Override
   public void onRebind(Intent intent) {

   }

   /** Called when The service is no longer used and is being destroyed
*/
   @Override
   public void onDestroy() {

   }
}
```

Exemplu

Acest exemplu vă va conduce prin pași simpli pentru a arăta cum să creați propriul serviciu Android. Urmați pașii următori pentru a modifica aplicația Android pe care am creat-o în capitolul Exemplu Hello World –

Step	Descriere
1	Veți utiliza Android StudioIDE pentru a crea o aplicație Android și a o denumi ca aplicația mea sub un pachet com.t201.myapplication, așa cum este explicat în capitolul Exemplu Hello World.
2	Modificați fișierul de activitate principal MainActivity.java pentru a adăuga metode startService () și stopService ().
3	Creați un nou fișier java MyService.java sub pachetul com.example.Aplicarea mea. Acest fișier va avea implementarea metodelor legate de serviciile Android.

4	Definiți-vă serviciul în fișierul AndroidManifest.xml utilizând eticheta <service ... />. O aplicație poate avea unul sau mai multe servicii fără restricții.
5	Modificați conținutul implicit al fișierului res / layout / activity_main.xml pentru a include două butoane în aspect liniar.
6	Nu este nevoie să modificați constantele din fișierul res / values / strings.xml. Android Studio are grijă de valorile șirurilor
7	Rulați aplicația pentru a lansa emulatorul Android și verificați rezultatul modificărilor efectuate în aplicație.

Urmează conținutul fișierului de activitate principal modificat MainActivity.java. Acest fișier poate include fiecare dintre metodele fundamentale ale ciclului de viață. Am adăugat startService () și stopService () metode pentru a porni și opri serviciul.

```
package com.t201.myapplication;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

import android.os.Bundle;
import android.app.Activity;
import android.util.Log;
import android.view.View;

public class MainActivity extends Activity {
   String msg = "Android :";

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Log.d(msg, "The onCreate() event");
   }

   public void startService(View view) {
      startService(new Intent(getBaseContext(), MyService.class));
   }

   // Method to stop the service
   public void stopService(View view) {
      stopService(new Intent(getBaseContext(), MyService.class));
   }
}
```

Următorul este conținutul MyService.java. Acest fișier poate avea implementarea uneia sau mai multor metode asociate Serviciului pe baza cerințelor. Deocamdată vom implementa doar două metode onStartCommand () și onDestroy ()−

```
package com.t201.myapplication;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
```

```
import android.support.annotation.Nullable;
import android.widget.Toast;

public class MyService extends Service {
   @Nullable
   @Override
   public IBinder onBind(Intent intent) {
      return null;
   }

   @Override
   public int onStartCommand(Intent intent, int flags, int startId) {
      // Let it continue running until it is stopped.
      Toast.makeText(this, "Service Started",
Toast.LENGTH_LONG).show();
      return START_STICKY;
   }

   @Override
   public void onDestroy() {
      super.onDestroy();
      Toast.makeText(this, "Service Destroyed",
Toast.LENGTH_LONG).show();
   }
}
```

Urmează conținutul modificat al fișierului AndroidManifest.xml. Aici am adăugat eticheta <service ... /> pentru a include serviciul nostru −

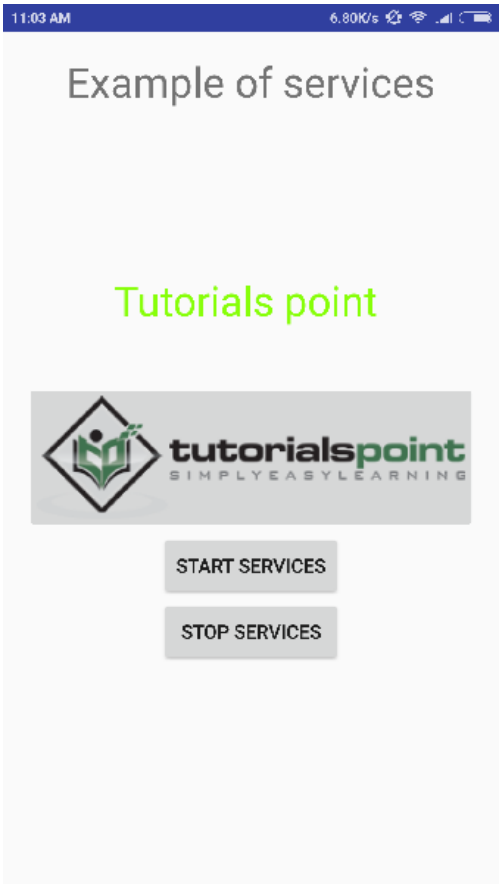
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.t201.myapplication">

   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:supportsRtl="true"
      android:theme="@style/AppTheme">

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

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

      <service android:name=".MyService" />
   </application>

</manifest>
```

Următorul va fi conținutul fișierului res / layout / activity_main.xml pentru a include două butoane −

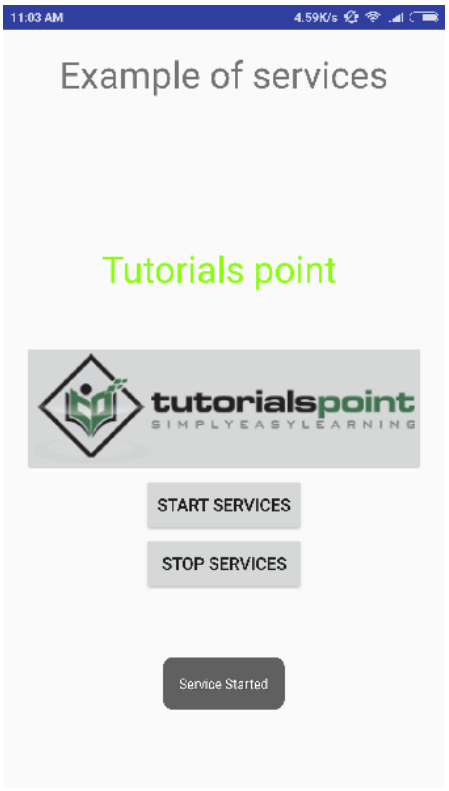
```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
   android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"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

   <TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Example of services"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:textSize="30dp" />

   <TextView
      android:id="@+id/textView2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="T201"
      android:textColor="#ff87ff09"
      android:textSize="30dp"
      android:layout_above="@+id/imageButton"
      android:layout_centerHorizontal="true"
      android:layout_marginBottom="40dp" />

   <ImageButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/imageButton"
      android:src="@drawable/abc"
      android:layout_centerVertical="true"
      android:layout_centerHorizontal="true" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/button2"
      android:text="Start Services"
      android:onClick="startService"
      android:layout_below="@+id/imageButton"
      android:layout_centerHorizontal="true" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Stop Services"
      android:id="@+id/button"
      android:onClick="stopService"
      android:layout_below="@+id/button2"
```

```
      android:layout_alignLeft="@+id/button2"
      android:layout_alignStart="@+id/button2"
      android:layout_alignRight="@+id/button2"
      android:layout_alignEnd="@+id/button2" />

</RelativeLayout>
```

Să încercăm să rulăm aplicatia in AVD-ul creat în timp ce efectuați configurarea mediului. Pentru a rula aplicația din Android Studio, deschideți unul dintre fișierele de activitate ale proiectului dvs. și faceți clic pe pictograma Executare din bara de instrumente. Android Studio instalează aplicația pe AVD și o pornește și dacă totul este în regulă cu configurarea și aplicația dvs., aceasta se va afișa după fereastra Emulator−



Acum, pentru a începe serviciul, să facem clic pe butonul Start Service, acesta va porni serviciul și, conform programării noastre în metoda onStartCommand(), un mesaj Service Started va apărea în partea de jos a simulatorului după cum urmează:



Pentru a opri serviciul, puteți face clic pe butonul Stop Service.

Broadcast Receivers

Broadcast Receivers răspund pur și simplu la mesajele difuzate de la alte aplicații sau de la sistemul de operare. Aceste mesaje se numesc uneori evenimente sau intent. De exemplu, aplicațiile pot iniția, de asemenea, transmisii pentru a informa alte aplicații că unele date au fost descărcate pe dispozitiv și că sunt disponibile pentru a le utiliza, deci acesta este un receptor care va intercepta această comunicare și va iniția acțiunea adecvată.

Urmează doi pași importanți pentru ca BroadcastReceiver să funcționeze pentru intențiile difuzate de sistem −

- Crearea Broadcast Receiver.
- Inregistrarea Broadcast Receiver

Există un pas suplimentar în cazul în care urmează să vă implementați intențiile personalizate, atunci va trebui să creați și să difuzați aceste intenții.

Crearea Broadcast Receiver

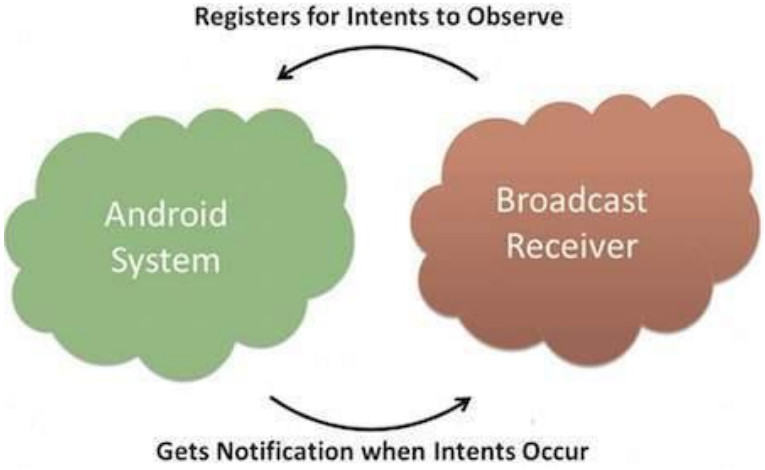
Un broadcast receiver este implementat ca o subclasă a clasei **BroadcastReceiver** și suprascrie metoda onReceive() în care fiecare mesaj este primit ca parametru de obiect **Intent**.

```
public class MyReceiver extends BroadcastReceiver {
   @Override
   public void onReceive(Context context, Intent intent) {
      Toast.makeText(context, "Intent Detected.",
Toast.LENGTH_LONG).show();
```

```
    }
}
```

Inregistrarea Broadcast Receiver

O aplicație ascultă anumite intenții de difuzare prin înregistrarea unui receptor de difuzare în fișierul AndroidManifest.xml. Luați în considerare că vom înregistra MyReceiver pentru evenimentul generat de sistem ACTION_BOOT_COMPLETED care este declanșat de sistem odată ce sistemul Android a finalizat procesul de boot.



Broadcast-Receiver

```
<application
   android:icon="@drawable/ic_launcher"
   android:label="@string/app_name"
   android:theme="@style/AppTheme" >
   <receiver android:name="MyReceiver">

      <intent-filter>
         <action android:name="android.intent.action.BOOT_COMPLETED">
         </action>
      </intent-filter>

   </receiver>
</application>
```

Acum, de fiecare dată când dispozitivul dvs. Android este pornit, acesta va fi interceptat de BroadcastReceiver MyReceiver și logica implementată în interiorul onReceive () va fi executată.

Există mai multe evenimente generate de sistem definite ca câmpuri statice finale în clasa Intent. Următorul tabel listează câteva evenimente importante ale sistemului.

No	Constante și descriere eveniment
1	**android.intent.action.BATTERY_CHANGED** Transmisie lipicioasă care conține starea de încărcare, nivelul și alte Informații despre baterie.
2	**android.intent.action.BATTERY_LOW** Indică starea bateriei descărcate pe dispozitiv.

3	**android.intent.action.BATTERY_OKAY** Indică că acum bateria este în regulă după ce a fost descărcată.
4	**android.intent.action.BOOT_COMPLETED** Aceasta este transmisă o singură dată, după ce sistemul a terminat de pornire.
5	**android.intent.action.BUG_REPORT** Afișați activitatea pentru raportarea unei erori.
6	**android.intent.action.CALL** Efectuați un apel către cineva specificat de date.
7	**android.intent.action.CALL_BUTTON** Utilizatorul a apăsat butonul „apel” pentru a accesa apelatorul sau altă interfață de utilizare adecvată pentru efectuarea unui apel.
8	**android.intent.action.DATE_CHANGED** Data s-a schimbat.
9	**android.intent.action.REBOOT** Solicitați repornirea dispozitivului.

Broadcasting Custom Intents

Dacă doriți ca aplicația dvs. să genereze și să trimită intenții personalizate, va trebui să creați și să trimiteți aceste intenții utilizând metoda sendBroadcast () din clasa dvs. de activitate. Dacă utilizați metoda sendStickyBroadcast (Intent), intenția este **lipicioasă (sticky)**, ceea ce înseamnă că intenția pe care o trimiteți rămâne după finalizarea difuzării.

```
public void broadcastIntent(View view) {
   Intent intent = new Intent();
   intent.setAction("com.t201.CUSTOM_INTENT");
   sendBroadcast(intent);
}
```

Această intenție com.t201.CUSTOM_INTENT poate fi, de asemenea, înregistrată în mod similar, deoarece am regsiterat intenția generată de sistem.

```
<application
   android:icon="@drawable/ic_launcher"
   android:label="@string/app_name"
   android:theme="@style/AppTheme" >
   <receiver android:name="MyReceiver">

      <intent-filter>
         <action android:name="com.t201.CUSTOM_INTENT">
         </action>
      </intent-filter>

   </receiver>
</application>
```

Exemplu

Acest exemplu vă va explica cum să creați BroadcastReceiver pentru a intercepta intenția personalizată. Odată ce sunteți familiarizat cu intenția personalizată, atunci puteți programa aplicația pentru a intercepta intențiile generate de sistem. Deci, să urmăm pașii următori pentru a modifica aplicația Android pe care am creat-o în capitolul Exemplu Hello World –

Pas	Descriere
1	Veți utiliza Android Studio pentru a crea o aplicație Android și a o denumi ca aplicația mea sub un pachet com.t201.myapplication, așa cum este explicat în capitolul Exemplu Hello World.
2	Modificați fișierul de activitate principal MainActivity.java pentru a adăuga metoda broadcastIntent ().
3	Creați un nou fișier java numit MyReceiver.java sub pachetul com.t201.myapplication pentru a defini un BroadcastReceiver.
4	O aplicație poate gestiona unul sau mai multe scopuri personalizate și de sistem fără restricții. Fiecare intenție pe care doriți să o interceptați trebuie să fie înregistrată în fișierul dvs. AndroidManifest.xml utilizând eticheta <receptor ... />.
5	Modificați conținutul implicit al fișierului res / layout / activity_main.xml pentru a include un buton pentru a difuza intenția.
6	Nu este nevoie să modificați fișierul șir, Android Studio se ocupă de fișierul string.xml.
7	Rulați aplicația pentru a lansa emulatorul Android și verificați rezultatul modificărilor efectuate în aplicație.

Urmează conținutul fișierului de activitate principal modificat **MainActivity.java**. Acest fișier poate include fiecare dintre metodele fundamentale ale ciclului de viață. Am adăugat metoda broadcastIntent () pentru a trimite un mesaj broadcast.

```
package com.t201.myapplication;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {

   /** Called when the activity is first created. */
   @Override

   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
```

```
   // broadcast a custom intent.

   public void broadcastIntent(View view){
      Intent intent = new Intent();
      intent.setAction("com.t201.CUSTOM_INTENT");
sendBroadcast(intent);
   }
}
```

Următorul este conținutul **MyReceiver.java**:

```
package com.t201.myapplication;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class MyReceiver extends BroadcastReceiver{
   @Override
   public void onReceive(Context context, Intent intent) {
      Toast.makeText(context, "Intent Detected.",
Toast.LENGTH_LONG).show();
   }
}
```

Urmează conținutul modificat al fișierului AndroidManifest.xml. Aici am adăugat eticheta <receptor ... /> pentru a include serviciul nostru:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.t201.myapplication">

   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:supportsRtl="true"
      android:theme="@style/AppTheme">

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

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

      <receiver android:name="MyReceiver">
         <intent-filter>
            <action android:name="com.t201.CUSTOM_INTENT">
            </action>
         </intent-filter>
```

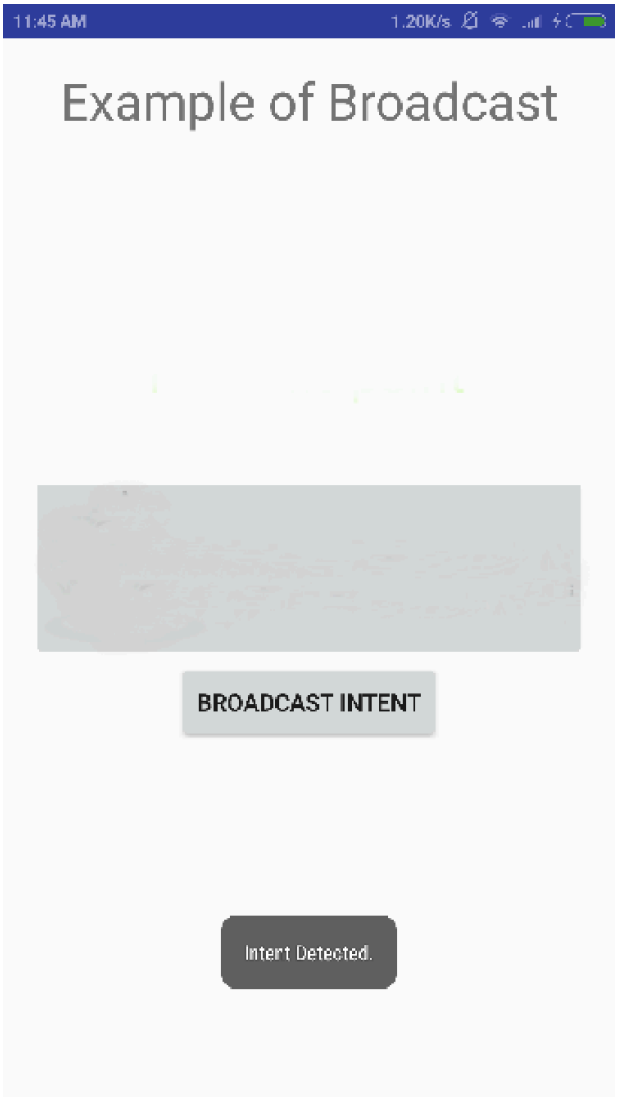
```
      </receiver>
   </application>

</manifest>
```

În continuare va fi conținutul fișierului **res / layout / activity_main.xml** pentru a include un buton pentru a difuza intenția noastră personalizată−

```
<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:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
   tools:context=".MainActivity">

   <TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Example of Broadcast"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:textSize="30dp" />

   <TextView
      android:id="@+id/textView2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="T201"
      android:textColor="#ff87ff09"
      android:textSize="30dp"
      android:layout_above="@+id/imageButton"
      android:layout_centerHorizontal="true"
      android:layout_marginBottom="40dp" />

   <ImageButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/imageButton"
      android:src="@drawable/abc"
      android:layout_centerVertical="true"
      android:layout_centerHorizontal="true" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/button2"
      android:text="Broadcast Intent"
      android:onClick="broadcastIntent"
      android:layout_below="@+id/imageButton"
      android:layout_centerHorizontal="true" />
```

```
</RelativeLayout>
```

Să încercăm să rulăm aplicatia in AVD-ul creat în timp ce efectuați configurarea mediului. Pentru a rula aplicația din Android Studio, deschideți unul dintre fișierele de activitate ale proiectului dvs. și faceți clic pe pictograma Executare din bara de instrumente. Android Studio instalează aplicația pe AVD și o pornește și dacă totul este în regulă cu configurarea și aplicația dvs., aceasta se va afișa după fereastra Emulator:



Acum, pentru a difuza intenția noastră personalizată, să facem clic pe butonul **Broadcast Intent**, aceasta va transmite intenția noastră personalizată „com.t201.CUSTOM_INTENT” care va fi interceptată de BroadcastReceiver înregistrat, adică MyReceiver și, conform logicii noastre implementate, un toast va apărea în partea de jos a simulatorului după cum urmează −

Puteți încerca să implementați alt BroadcastReceiver pentru a intercepta intențiile generate de sistem, cum ar fi pornirea sistemului, data schimbării, bateria descărcată etc.

Bibliografie:

https://developer.android.com/studio/command-line/adb