

Programarea Aplicațiilor Incorporate și Independente de Platformă

L.univ. Antohi Ionel
email: antohi.ionel@gmail.com

NOȚIUNI TEORETICE

- Structura unei aplicatii Android
- Contextul
- Activitati
- Servicii
- Intentia

Platforme și Medii



ANDROID(Java)



WINDOWS(C#)



iOS(Objective-C)

ANDROID



Body Areas

- Controled by hardware
- Non programatic [1]

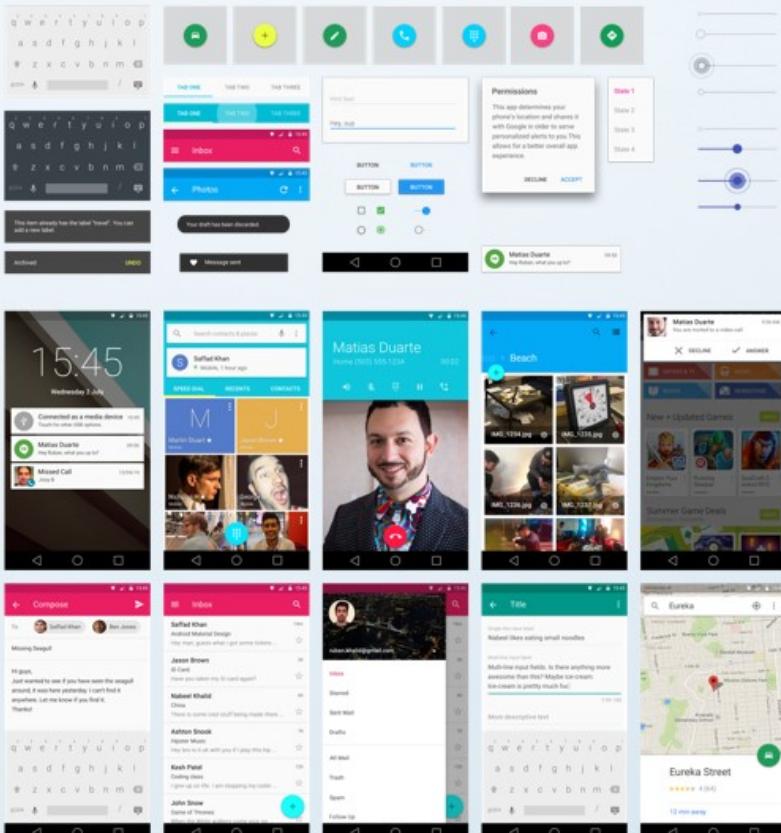
System Areas

- Interference with apps
- Managed by OS

User Areas

- Full control
- Creativity limitless

ANDROID UI Kit



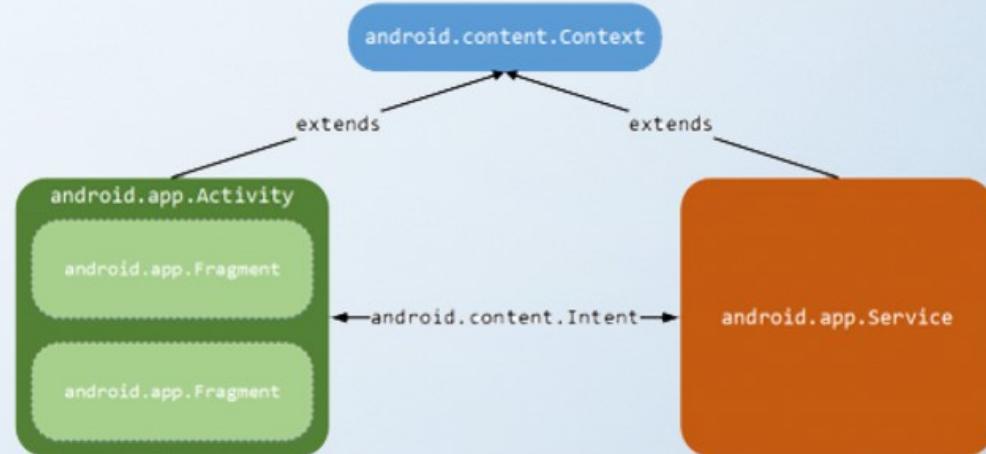
Structura unei aplicatii și modul ei de afișare strict depind de componentele dezvoltate în sistemul de operare Android.

Gama de versiuni și moduri de stilizare a sistemelor de operare dependează de la producător la producător. De aici și se pornește ideea de sinergie între design al dispozitivului și soft.

Android:
“Cel mai nemulțumit sistem de operare.”

Structura unei aplicații

- Contextul
- Activitatea / Activity
- Fragmentul
- Serviciul
- Intentia / Intent



CONTEXTUL

- **punctul central** al unei aplicatii
- ofera acces mai multor functionalitati
 - resursele dispozitivului mobil
 - serviciile sistemului de operare
 - diferite fisiere de configurare
- este instantiat sub forma unui obiect de tip **android.app.Application**

ACTIVITATEA

- sarcini a caror executie nu influenteaza **timpul de raspuns** al aplicatiei
- este asociata unei **ferestre** (interfeete grafice)
- o aplicatie Android e formata din **una sau mai multe activitati**

FRAGMENTUL

- interfata grafica si logica a aplicatiei
- corespunzatoare unei **parti** din cadrul unei activitati
- asigurarea **consistentei** si **flexibilitatii** aplicatiei Android pe mai multe tipuri de display (rezolutii diferite, aspect ratio diferit, etc)

SERVICIUL

- procese care ar trebui sa ruleze
in **background**

INTENTIA

- mecanismul de **comunicare**
intre elementele unei aplicatii
(activitati si servicii)
- sistem de mesagerie asincrona

Si cum creez o activitate?

pe langa codul efectiv al unei activitati,
aceasta trebuie definita si in Manifest



The image shows a screenshot of an AndroidManifest.xml file in an IDE. The file is an XML configuration for an Android application. It defines a single activity named LifecycleMonitorActivity, which is set to be the main launcher activity. The manifest includes standard XML tags like <manifest>, <application>, <activity>, <intent-filter>, <action>, and <category>. The code is color-coded, with tags in blue and attribute values in red.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <!-- ... -->
    <application ... >
        <activity
            android:name=".LifecycleMonitorActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Ciclul de viață al activității

Resumed (running)

- in foreground
- user focus

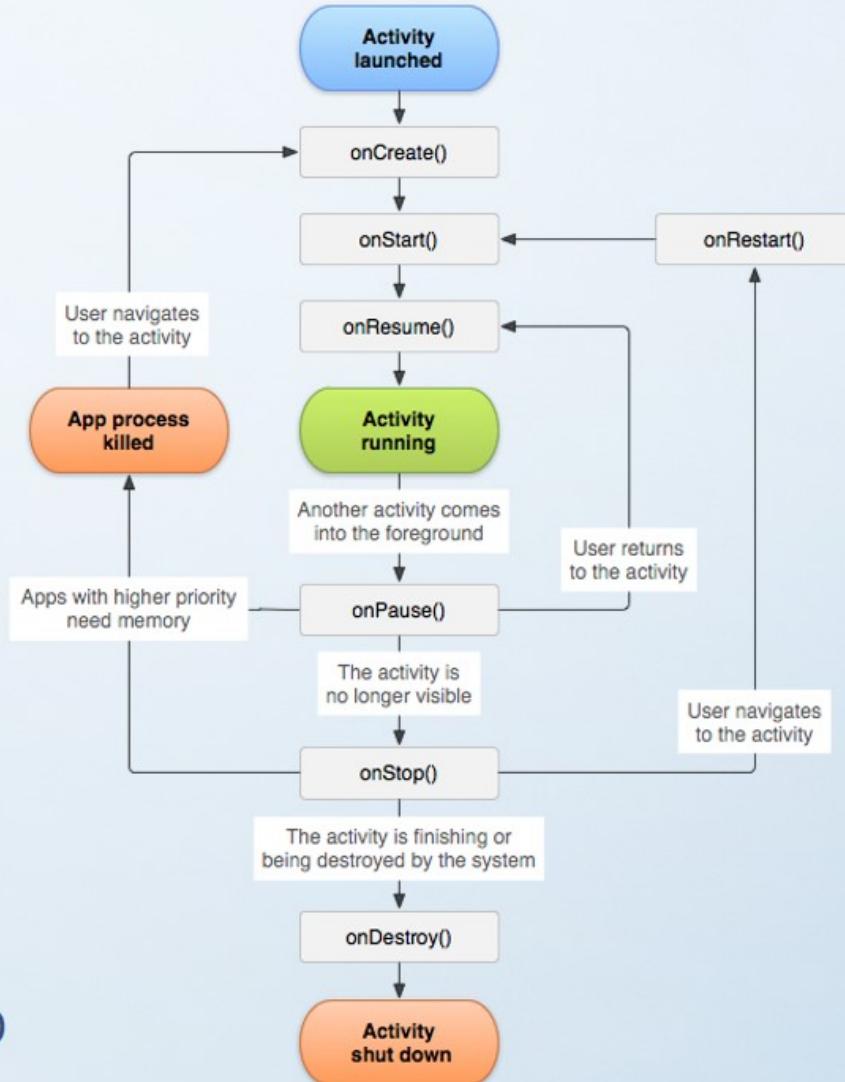
Paused

- alta activitate e in focus, dar inca e vizibila (sau cel putin parțial)
- stopata parțial de sistem

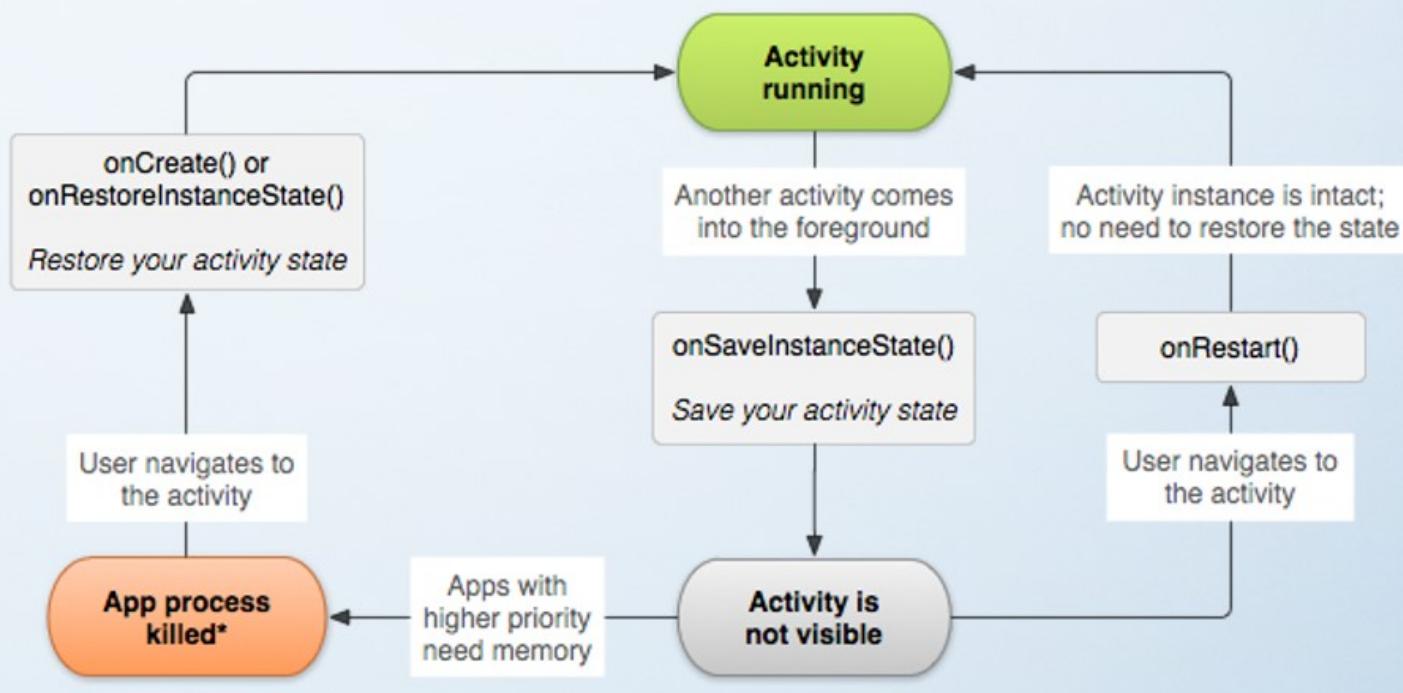
Stopped

- complet in background

```
public class ExampleActivity extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        // The activity is being created.  
    }  
    @Override  
    protected void onStart() {  
        super.onStart();  
        // The activity is about to become visible.  
    }  
    @Override  
    protected void onResume() {  
        super.onResume();  
        // The activity has become visible (it is now "resumed").  
    }  
    @Override  
    protected void onPause() {  
        super.onPause();  
        // Another activity is taking focus (this activity is about to be "paused").  
    }  
    @Override  
    protected void onStop() {  
        super.onStop();  
        // The activity is no longer visible (it is now "stopped")  
    }  
    @Override  
    protected void onDestroy() {  
        super.onDestroy();  
        // The activity is about to be destroyed.  
    }  
}
```



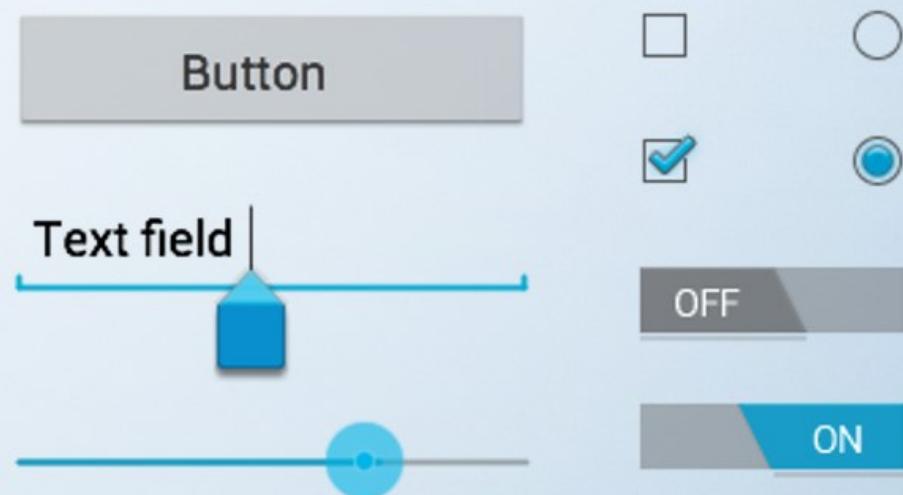
ACTIVITY ALL MOD



GESTIUNEA STARII UNEI ACTIVITATI

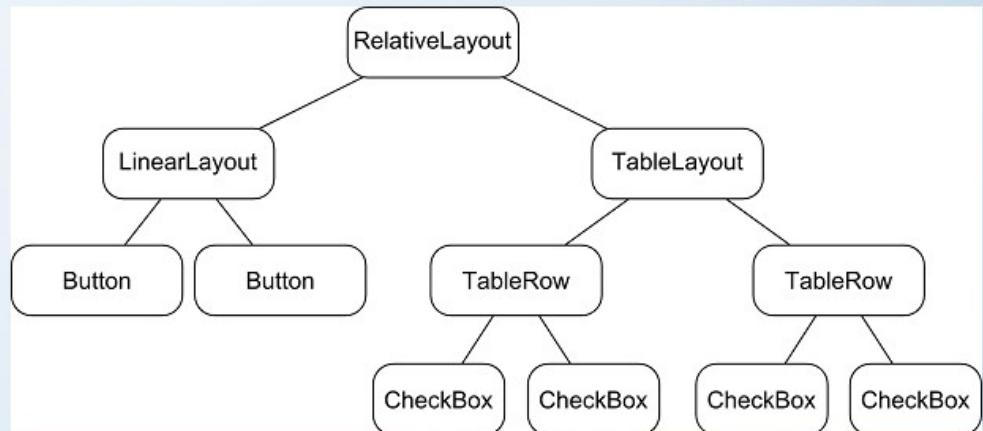
Ce poate contine o activitate?

- componente (Widgets)
- extind clasa View
- statice
 - TextView
 - ImageView
 - etc
- dinamice
 - Button
 - EditText
 - Checkbox
 - etc



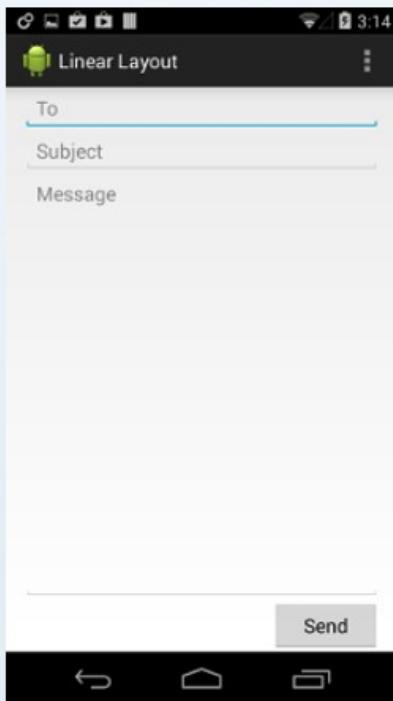
Container

- View
 - poate contine alte view-uri
 - Layout
- Tipuri de Layout
 - LinearLayout
 - RelativeLayout
- În practică folosim combinații

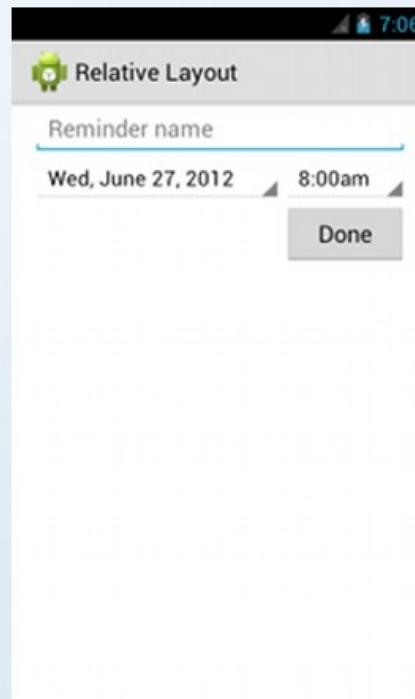


Tipuri de Layout și View-uri

LinearLayout



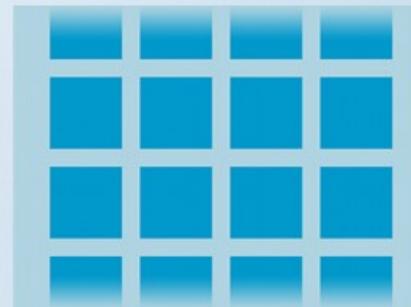
RelativeLayout



ListView

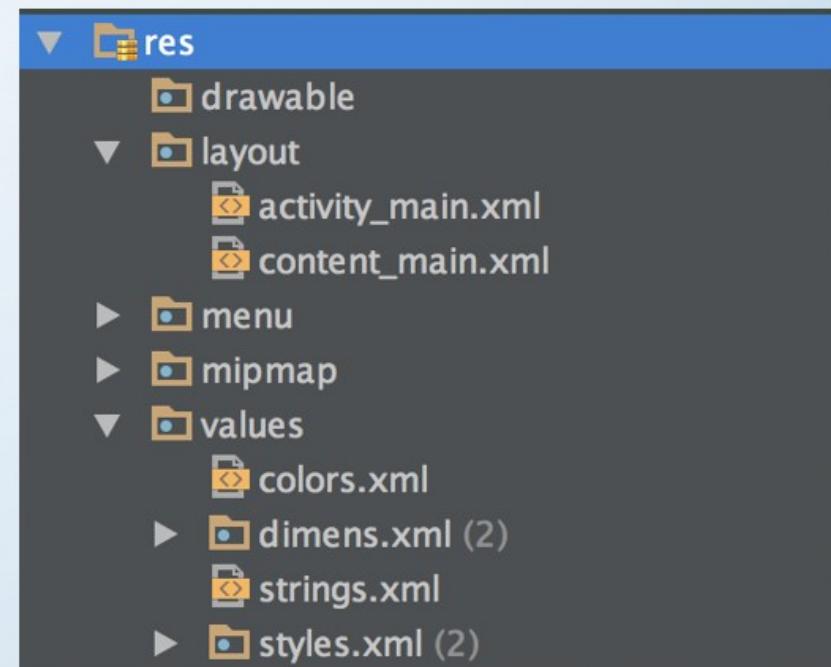


GridView



RESURSE

- directorul res (din sursa)
- imagini
 - drawable - _dpi
 - ldpi
 - mdpi
 - hdpi
 - xdpi
- interfete grafice
 - layout
- constante
 - values
 - strings.xml
- raw
 - resurse nemodificate



Resurse Automate și GUI

Drawable

- imagini
- adresabile in cod prin `R.drawable.nume`
- adresabile in XML prin `@drawable/nume`

Layout

- containere
- adresabile in cod prin `R.layout.nume`
- adresabile in XML prin `@layout/nume`

GUI

- se face prin fisierele XML
- se creeaza un arbore de View-uri
- se aplica in codul unei activitati prin:
 - `this.setContentView (R.layout.nume);`

activity_main.xml x content_main.xml x MainActivity.java x

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto" android:layout_width=
    android:layout_height="match_parent" android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    app:layout_behavior="android.support.design.widget.AppBarLayout$Scrolling
    tools:showIn="@layout/activity_main" tools:context=".MainActivity">

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

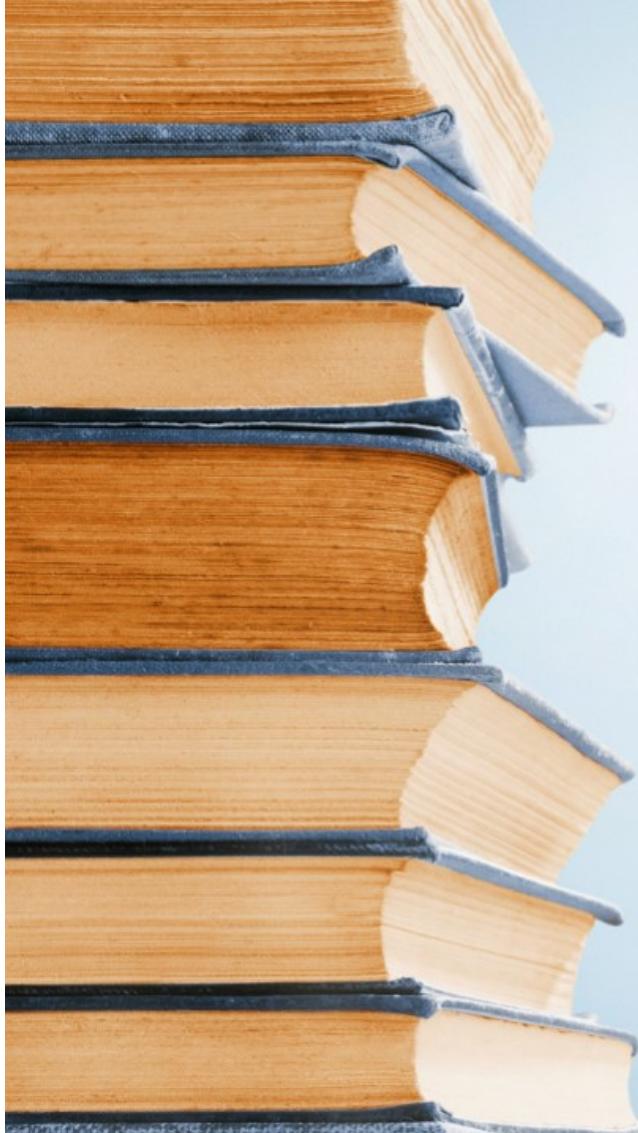
Preview

Nexus 4 NoActionBar MainActivity

The screenshot shows the Android Studio interface. On the left, there are three tabs: 'activity_main.xml' (selected), 'content_main.xml', and 'MainActivity.java'. The 'activity_main.xml' tab contains XML code for a RelativeLayout. The 'content_main.xml' tab is empty. The 'MainActivity.java' tab has a blue circular icon next to it. On the right, there is a 'Preview' window titled 'Nexus 4' showing a smartphone screen with a purple header bar and a white body containing the text 'Hello World!'. Below the screen are standard Android navigation icons. At the bottom of the preview window, there are buttons for 'Design' and 'Text'.

- Activity
- Fragment
- View
- Life cycle of Fragment
- Threads





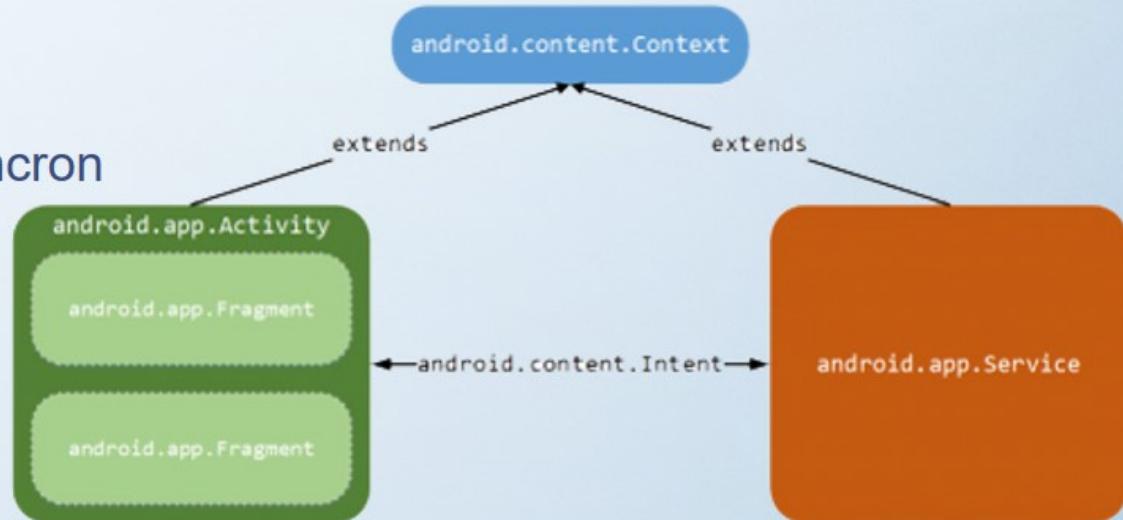
Intenții & Servicii

Agenda

- Intentii
 - structură
 - mecanisme de invocare
 - fluxuri informaționale între componente
 - mesaje cu difuzare
- Servicii
 - operații asupra serviciilor
 - relația serviciu
 - intenție
 - activitate
 - categorii de procesări
 - alarme

Ce este o intenție?

- mecanism de conexiune între componentele SO Android (cuplare slabă!!!)
 - flexibilitate
 - extensibilitate
- propagată ca mesaj asincron
- intenție = acțiune + date



Funcționalități

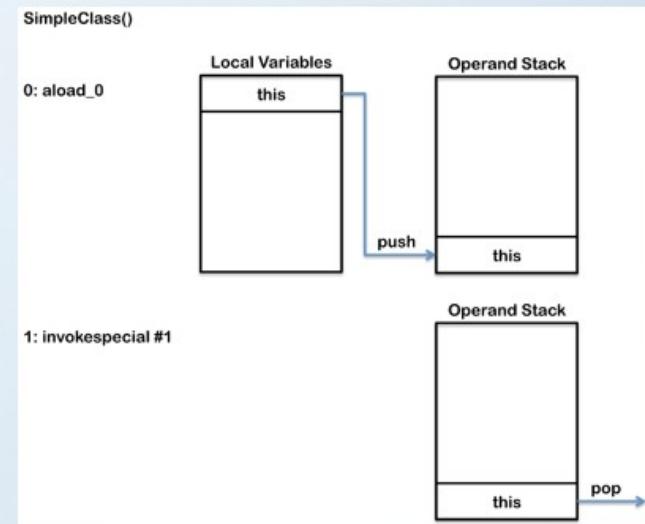
- invocă
 - activități / servicii
 - mesaje cu difuzare (eng. broadcast messages)
- unde se folosesc?
 - peste tot (transmiterea unui mesaj la nivelul SO: apel telefonic, SMS, status conectivitate, baterie)

Structura

- o componentă (activitate / serviciu)
 - definită în AndroidManifest.xml
 - eticheta <intent-filter> definește caracteristicile intenției prin care aceasta poate fi invocată
 1. **action (android:name)** – MAIN, VIEW
 2. category (android:name) – LAUNCHER, DEFAULT
 3. **data** – Uri.parse()
 - android:host, android:port
 - android:mimeType - tel:, content://contacts/people, geo:, http://
 - android:path
 - android:scheme
 4. type
 5. component
 6. extra: putExtras() / getExtras()
- clasificare: explicite, implicate

Mecanisme de Invocare

1. precizarea clasei încărcate
 - a) în contextul aceleiași aplicații Android
 - b) în componente diferite
2. precizarea acțiunii (și eventual, a datelor)
3. precizarea unui URI



Invokeare prin Precizarea Clasei Încărcate

- intenții explicate
- pentru ca o activitate să poată fi invocată
 - trebuie să definească minim un element de tip în AndroidManifest.xml (nu este generat automat de IDE!!!)
 - trebuie să definească
- o acțiune (predefinită sau definită de utilizator)
- categoria DEFAULT

```
<activity
    android:name=".RegisterActivity"
    android:label="@string/title_activity_register" >
    <intent-filter>
        <action android:name="org.rodedu.dandroid.messaging.intent.action.RegisterActivity" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Invokeare prin Precizarea Clasei Încărcate

- tipuri de constructori
 - invocarea unei activități în contextul aceleiași aplicații Android

```
public Intent (Context packageContext, Class<?> cls)
```

Added in API level 1

```
Intent registerActivityIntent = new Intent(this, RegisterActivity.class);
startActivity(registerActivityIntent);
```

- invocarea unei activități în componente diferite

```
public Intent (String action)
```

Added in API level 1

```
Intent registerActivityIntent = new Intent("org.roasedu.dandroid.messageme.intent.action.RegisterActivity");
startActivity(registerActivityIntent);
```

- `startActivity(intent)`
 - activitatea este lansată în execuție
 - stiva de componente permite întoarcerea la ecranele anterioare (butonul Back)

Invocare prin Precizarea Acțiunii (și, eventual, a datelor)

- intenții implicate (utilizatorul nu cunoaște componenta care va realiza acțiunea propriu-zisă)
- mecanism de rezoluție
 - sunt parcuse toate elementele ale componentelor care îl defines
 - se identifică acele activități ce pot deservi acțiunea, respectiv datele (precum și alți parametri)
 - să specifice minim o acțiune
 - să coincidă în fiecare parte din URI-ul datelor
 - dacă există mai multe activități este solicitată acțiunea utilizatorului
 - o preferință poate fi reținută



Tipuri de Acțiuni

- Intent.ACTION_VIEW – vizualizarea conținutului specificat în secțiunea data – poate invoca componente diferite: http, tel, geo, content
- Intent.ACTION_INSERT, Intent.ACTION_EDIT, Intent.ACTION_DELETE, Intent.ACTION_SEARCH
- Intent.ACTION_WEB_SEARCH – căutarea de informații folosind un motor de căutare – cheia SearchManager.QUERY conține interogarea pășată în câmpul extra
- Intent.ACTION_DIAL
- Intent.ACTION_CALL – permisiunea android.permission.CALL_PHONE
- Intent.ACTION_ALL_APPS
- Intent.ACTION_PICK
- Intent.ACTION_SEND , Intent.ACTION_SENDTO

Invokeare prin Precizarea Acțiunii (și, eventual, a datelor)

- Intent intent = new Intent(Intent.ACTION_...);
intent.setData(Uri.parse("..."));
- nu se garantează faptul că intenția indicată de utilizator va putea fi rezolvată!!!
 - se recomandă să se verifice înainte de lansarea în execuție
 - PackageManager – queryIntentActivities()

```
Intent applicationIntent = new Intent(...);
PackageManager packageManager = getPackageManager();
ComponentName componentName = applicationIntent.resolveActivity(packageManager);
if (componentName == null) {
    Intent marketIntent = new Intent(Intent.ACTION_VIEW, Uri.parse("market://search?q=pname:..."));
    if (marketIntent.resolveActivity(packageManager) != null) {
        startActivity(marketIntent);
    } else {
        Toast.makeText(getApplicationContext(), "Google Play Store is not available", Toast.LENGTH_LONG).show();
    }
} else {
    startActivity(applicationIntent);
}
```

Invokeare prin Precizarea unui URI

- se utilizează
 - acțiunea Intent.ACTION_VIEW
 - un URI care indică o schemă și o gazdă (restul spațiului de nume fiind ignorat)

```
public Intent (String action, Uri uri)
```

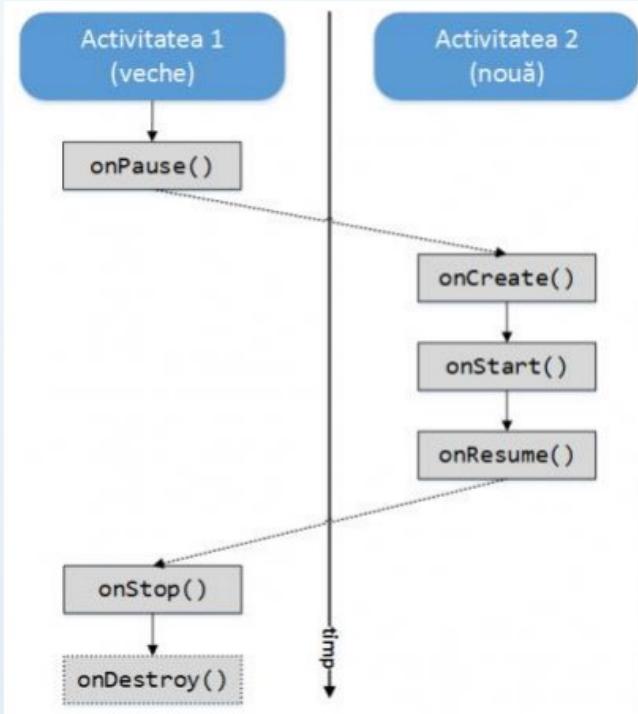
Added in API level 1

```
Uri uri = Uri.parse("myprotocol://mynamespace/myactivity");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);
```

- În secțiunea <intent-filter> se precizează parametrii android:scheme și android:host pentru elementul <data>

```
<activity
    android:name=".RegisterActivity"
    android:label="@string/title_activity_register" >
    <intent-filter>
        <action android:name="org.rosedu.dandroid.messaging.intent.action.RegisterActivity" />
        <category android:name="android.intent.category.DEFAULT" />
        <data
            android:scheme="myprotocol"
            android:host="mynamespace" />
    </intent-filter>
</activity>
```

Transferul Contextului Între Activități



- onPause()
 - gestiunea aspectelor critice
 - starea componentelor
 - cod sursă în execuție
 - nu trebuie să consume prea mult timp !!!
- onStop() – apelat numai după ce activitatea invocată este vizibilă
- onDestroy() – dacă necesarul de resurse impune acest lucru

Procesarea unei Intenții în cadrul unei Componente

- orice componentă are acces la:
 - intenția care a invocat-o
 - atributele intenției
 - acțiunea pe care trebuie să o realizeze
 - datele pe care trebuie să le proceseze
- o componentă poate primi mai multe intenții după ce este creată
 - public void onNewIntent(Intent newIntent);
- o componentă poate transfera responsabilitatea cu privire la gestiunea unei intenții către altă componentă
 - startNextMatchingActivity(intent);

```
Intent intent = getIntent();
if (intent != null) {
    String action = intent.getAction();
    Uri data = intent.getData();
    Bundle extras = intent.getExtras();
}
```

Fluxuri Informaționale Între Componente

- transmiterea de informații (unidirecțional !!!)
Între componente poate fi realizată prin intermediul unui obiect android.os.Bundle
 - secțiunea extra: `getExtras()` / `putExtras()` – cu transferul parametrilor în cazul unui obiect deja existent sau `get<type>Extra()` / `put<type>Extra()`
 - recomandare: denumirea cheii să includă pachetul aplicației!!!
- componentele care comunică sunt independente între ele!!!

Fluxuri Informaționale Între Componente Subactivități

ACTIVITATEA PĂRINTE

- **startActivityForResult(intent, code)**
 - codul identifică în mod unic activitatea copil atunci când aceasta se termină
 - în câmpul extra pot fi plasate informații suplimentare
- **onActivityResult(code, result, intent)**
 - rezultatul: Activity.RESULT_OK sau Activity.RESULT_CANCELED
 - intenția are rolul de a transmite informații suplimentare

ACTIVITATEA COPIIL

- **setResult(result, intent)**
 - rezultatul: Activity.RESULT_OK sau Activity.RESULT_CANCELED
 - intenția are rolul de a transmite informații suplimentare
- **finish()** – indică faptul că activitatea a fost terminată

Fluxuri Informaționale Între Componente Subactivități

ACTIVITATE PARINTE

```
final private static int REGISTER_ACTIVITY_REQUEST_CODE = 2015;

@Override
protected void onCreate(Bundle state) {
    super.onCreate(state);
    setContentView(R.layout.activity_login);
    Intent intent = new Intent("org.rosedu.dandroid.messageme.intent.action.RegisterActivity");
    intent.putExtra("org.rosedu.dandroid.messageme.someKey", someValue);
    startActivityForResult(intent, REGISTER_ACTIVITY_REQUEST_CODE);
    // start another activities with their own request codes
}

public void onActivityResult(int requestCode, int resultCode, Intent intent) {
    switch(requestCode) {
        case REGISTER_ACTIVITY_REQUEST_CODE:
            if (resultCode == Activity.RESULT_OK) {
                Bundle data = intent.getExtras();
                // process information from data ...
            }
            break;

        // process other request codes
    }
}
```

ACTIVITATE COPIL

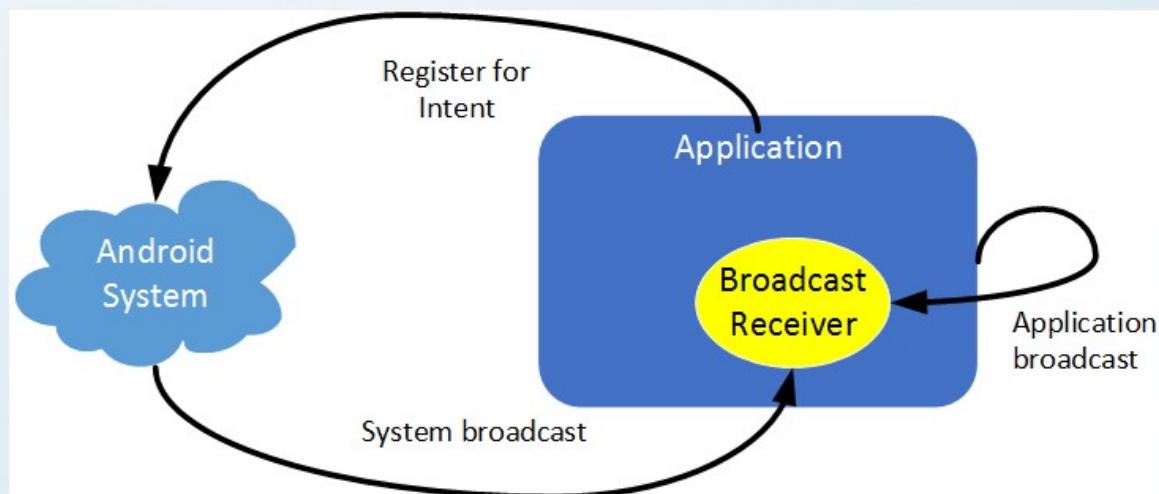
```
@Override
protected void onCreate(Bundle state) {
    super.onCreate(state);
    setContentView(R.layout.activity_register);

    // intent from parent
    Intent intentFromParent = getIntent();
    Bundle data = intentFromParent.getExtras();
    // process information from data ...

    // intent to parent
    Intent intentToParent = new Intent();
    intentToParent.putExtra("org.rosedu.dandroid.messageme.anotherKey", anotherValue);
    setResult(RESULT_OK, intentToParent);
    finish();
}
```

Intentii cu Difuzare (eng. Broadcast Intents)

- intenții transmise către toate componentele de la nivelul dispozitivului mobil, pentru a notifica producerea unui eveniment (HW/SW): apel telefonic, mesaj, nivel încărcare baterie, stare conectivitate
 - LocalBroadcastManager (la nivel local): apele sincrone
 - folosite nativ, la nivelul sistemului de operare



Intentii cu Difuzare (eng. Broadcast Intents)

- trimiterea
 - obiect **Intent** obișnuit, calificat prin cât mai multe atribute
 - acțiuni predefinite (aplicații native Android + alte aplicații)
 - acțiuni definite de utilizator
 - sendBroadcast(intent)
- primirea – de către componente specialize (broadcast receivers), care precizează un astfel de comportament
 - PackageManager.setComponentEnabledSetting(): activarea / dezactivarea unor astfel de componente

Ascultători pentru Intenții cu Difuzare (eng. Broadcast Receivers)

- clasa android.content.BroadcastReceiver
- metode de callback apelate automat – onReceive(Context, Intent)
 - executat pe firul de execuție principal
 - actualizare componente din cadrul interfeței grafice (notificare utilizator), lansare în execuție servicii

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class SomeBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        // ...
    }
}
```

- în fișierul AndroidManifest.xml
 - nu este necesar ca aplicația să ruleze în momentul în care mesajul este transmis
 - elementul trebuie să definească filtrul de intenții

Tipuri Particulare de Intenții cu Difuzare

- ordonate
 - intenție procesată secvențial de mai mulți ascultători, într-o anumită ordine
 - `sendOrderedBroadcast(intent, permission)`: sir de caractere indicând permisiunea de a prelucra intenția cu difuzare
 - atributul `android:permission` din dictează prioritatea de procesare
 - utilizate atunci când se așteaptă transmiterea înapoi de rezultate
- persistente
 - obținerea celei mai recente valori (nivelul de încărcare al bateriei, conectivitatea la Internet)
 - **`sendOrderedBroadcast(intent)`**
 - specificarea de permisiuni în `AndroidManifest.xml`:
`android.permission.BROADCAST_STICKY`
 - metoda `registerReceiver()` furnizează cea mai recentă valoare, fără indicarea obiectului ascultător (se precizează doar filtrul de intenții)

Intenții cu Difuzare Native

- ACTION_BATTERY_CHANGED (EXTRA_STATUS - BATTERY_STATUS: CHARGING, FULL), LOW, OKAY, CONNECTED, DISCONNECTED
- ACTION_BOOT_COMPLETED
- ACTION_CAMERA_BUTTON
- ACTION_DATE_CHANGED, ACTION_TIME_CHANGED
- ACTION_DOCK_EVENT
- ACTION_MEDIA_EJECT
- ACTION_MEDIA_MOUNTED, ACTION_MEDIA_UNMOUNTED
- ACTION_NEW_OUTGOING_CALL
- ACTION_SCREEN_OFF, ACTION_SCREEN_ON

Servicii



Ce este un serviciu?

- componentă Android adecvată pentru implementarea de funcționalități complexe, ce implică un volum de procesare mare, de lungă durată, ce nu necesită interacțiune cu utilizatorul
 - menține o aplicație Android în execuție, datorită sistemului de priorități activitate activă > serviciu > activitate inactivă
 - gestionat din contextul altor componente Android
 - implicit, rulat pe firul de execuție principal al aplicației Android
 - poate avea un impact asupra interfeței grafice

Structura unui Serviciu

- clasa **android.app.Service**
 - `onCreate()`
 - `onBind(Intent)` – asociază serviciul la o altă componentă Android
- definit în **AndroidManifest.xml** – eticheta `<service>`
 - `android:name` – clasa care implementează serviciul
 - `android:permission` – restricționează operațiile de pornire / oprire la componente care o dețin

```
<service
    android:name=".SomeService"
    android:enabled="true"
    android:permission="org.roasedu.dandroid.messaging.SOME_SERVICE_PERMISSION" />
```

Operații asupra Serviciilor

- **pornire:** `startService()` – pe baza unei intenții
 - explicate: `new Intent(this, SomeService.class);`
 - implicate: `new Intent("org.rosedu.dandroid.messaging.SOME_SERVICE");`
- **oprire:**
 - `stopService()` – de către altă componentă, pe baza unei intenții
 - `stopSelf()` – de către el însuși
 - fără parametri
 - cu parametru – identificatorul instanței curente

Operații asupra Serviciilor

- **int onStartCommand(Intent intent, int flags, int id)**
 - realizează procesarea propriu-zisă
 - parametri
 - intenția care l-a invocat
 - modul în care a fost (re)pornit
 - START_FLAG_REDELIVERY
 - START_FLAG_RETRY
 - identificatorul instanței curente
 - valoare întoarsă: comportamentul serviciului în situația în care este repornit
 - Service.START_STICKY (standard) – repornit cu intenția null
 - Service.START_NOT_STICKY – repornit numai la apel explicit
 - Service.START_REDELIVER_INTENT – repornit cu intenția originală

```
@Override  
public int onStartCommand(Intent intent, int flags, int startId) {  
    processInBackground(intent, startId);  
    return Service.START_STICKY;  
}
```

Relația Serviciu – Înțenție - Activitate

- Între un serviciu și o activitate poate fi realizată o asociere
- **bindService()**
 - înțenția
 - ServiceConnection
 - mecanismul de asociere
 - Context.BIND_AUTO_CREATE
 - Context.BIND_ADJUST_WITH_ACTIVITY
 - Context.BIND_ABOVE_CLIENT / Context.BIND_IMPORTANT
 - Context.BIND_NOT_FOREGROUND
 - Context.BIND_WAIVE_PRIORITY
- **unbindService()**

Relația Serviciu – Intentie - Activitate

```
public class SomeService extends Service {

    private final IBinder someBinder = new SomeBinder();

    public class SomeBinder extends Binder {
        SomeService getService() {
            return SomeService.this;
        }
    }

    @Override
    public void onCreate() {
        super.onCreate();
        // ...
    }

    @Override
    public IBinder onBind(Intent intent) {
        return someBinder;
    }
}
```

Relația Serviciu – Intentie - Activitate

```
public class SomeActivity extends Activity {

    protected SomeService someService;
    protected boolean bound = false;

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

    @Override
    protected void onStart() {
        super.onStart();
        Intent intent = new Intent(this, SomeService.class);
        bindService(intent, serviceConnection, Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
        super.onStop();
        unbindService(serviceConnection);
    }
}

private ServiceConnection serviceConnection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName componentName, IBinder binder) {
        someService = ((SomeService.SomeBinder) binder).getService();
        bound = true;
    }

    @Override
    public void onServiceDisconnected(ComponentName componentName) {
        someService = null;
        bound = false;
    }
};
```

Categorii de Procesări

- servicii în prim-plan
- procesări realizate în fundal: operații I/O, căutări în rețea, accesul la baza de date, calcule complexe
 - AsyncTask
 - IntentService
 - Loader
 - fire de execuție definite de utilizator

Servicii care Rulează În Prim-Plan

- folosite cu precădere în situația în care un serviciu interacționează cu utilizatorul
- **startForeground()**
 - identificatorul notificării
 - un obiect de tip Notification (reprezentare vizuală) afișat cât timp serviciul este în prim-plan
- **stopForeground()** – acțiune accesibilă utilizatorilor
- nu mai mult de o instanță la un moment dat !!!

Clasa AsyncTask <Params, Progress, Result>

- rularea unor operații complexe pe un fir de execuție dedicat
- sincronizare cu interfața grafică
- nepersistentă raportată la ciclul de viață al unei activități
- parametrizată cu tipuri de date generice / Void
 - parametri de intrare
 - progres
 - parametru de ieșire
- metode
 - onPreExecute()
 - doInBackground(Params...) → publishProgress(Progress...)
 - onProgressUpdate(Progress...)
 - onPostExecute(Result)
- rularea se face prin invocarea execute(Params...) - !!!poate fi rulat o singură dată

Clasele IntentService & Loader

- un serviciu lansat în execuție printr-o intenție
- onHandleIntent(Intent) – fir de execuție dedicat, pentru fiecare invocare în parte
- Clasa IntentService
 - un serviciu lansat în execuție printr-o intenție
 - onHandleIntent(Intent)
 - fir de execuție dedicat, pentru fiecare invocare în parte
- Clasa Abstractă Loader
 - procesări asincrone ce implică elemente ale interfeței grafice
 - monitorizarea surselor de date

Fire de Execuție definite de Utilizator

- implementări ale clasei Thread
 - **runOnUiThread(Runnable)** – codul sursă executat în contextul firului de execuție al interfeței grafice
 - **Handler** – acces la firul de execuție în care a fost creat prin metodele `post()`, `postDelayed()`, `postAtTime()`

```
handler = new Handler();

final Runnable r = new Runnable() {
    public void run() {
        tv.append("Hello World");
        handler.postDelayed(this, 1000);
    }
};

handler.postDelayed(r, 1000);
```

Alarme

- Mecanisme de transmitere a unor intenții la momente de timp prestabilite sau periodic
- Independente de o aplicație Android
- Adequate pentru operații consumatoare de resurse la momente oportune
- Consumatoare de energie
- AlarmManager ↔ getSystemService(Context.ALARM_SERVICE)
- Tipuri
 - RTC / RTC_WAKEUP – moment de timp absolut
 - ELAPSED_REALTIME / ELAPSED_REALTIME_WAKEUP – moment de timp relativ