2013 - **2014**

Tehnologii de Programare in Internet (TPI / RST)

Titulari curs: **Eduard-Cristian Popovici**, Mihnea Magheti

Suport curs: *http://discipline.elcom.pub.ro/tpi/*

Platforma Android

2. Programarea pe platforma Android

Dezvoltarea pe Eclipse IDE

Pregatirea dezvoltarii pe Eclipse IDE ***http://www.vogella.de/articles/Android/article.html***

- **instalarea** mediului **Eclipse IDE** ***http://www.vogella.de/articles/Eclipse/article.html***
  - **prin descarcarea si dezarhivarea** arhivei cu versiunea dorita de **Eclipse IDE**
    - de exemplu **3.5.2** ***http://www.eclipse.org/downloads/packages/release/galileo/sr2***
    - de exemplu in directorul **C:\eclipse**
  - **trebuie sa fie instalata si Java Runtime** (de preferinta **Java 1.6**)

- **instalarea Android SDK** ***http://dl.google.com/android/android-sdk_r07-windows.zip***
  - **prin descarcarea si dezarhivarea** arhivei oferita la adresa ***http://developer.android.com/sdk/index.html***
    - de exemplu in directorul **c:\android-sdk-windows**

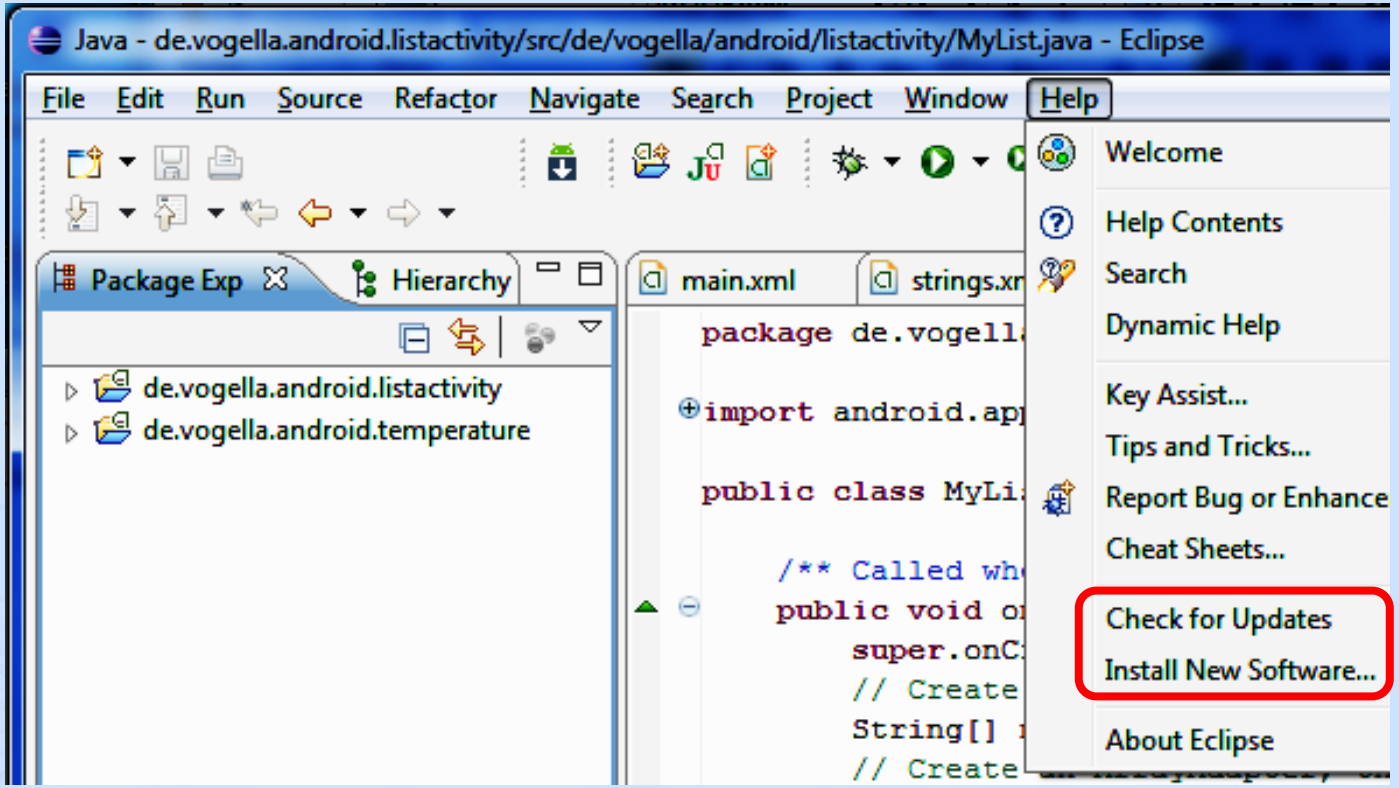
Dezvoltarea pe Eclipse IDE

Pregatirea dezvoltarii pe Eclipse IDE

http://www.vogella.de/articles/Android/article.html

– **instalarea** pluginurilor din **Android Development Tools (ADT)**

– **utilizand *Eclipse update manager*** pentru a descarca pluginurile de la adresa ***https://dl-ssl.google.com/android/eclipse/***



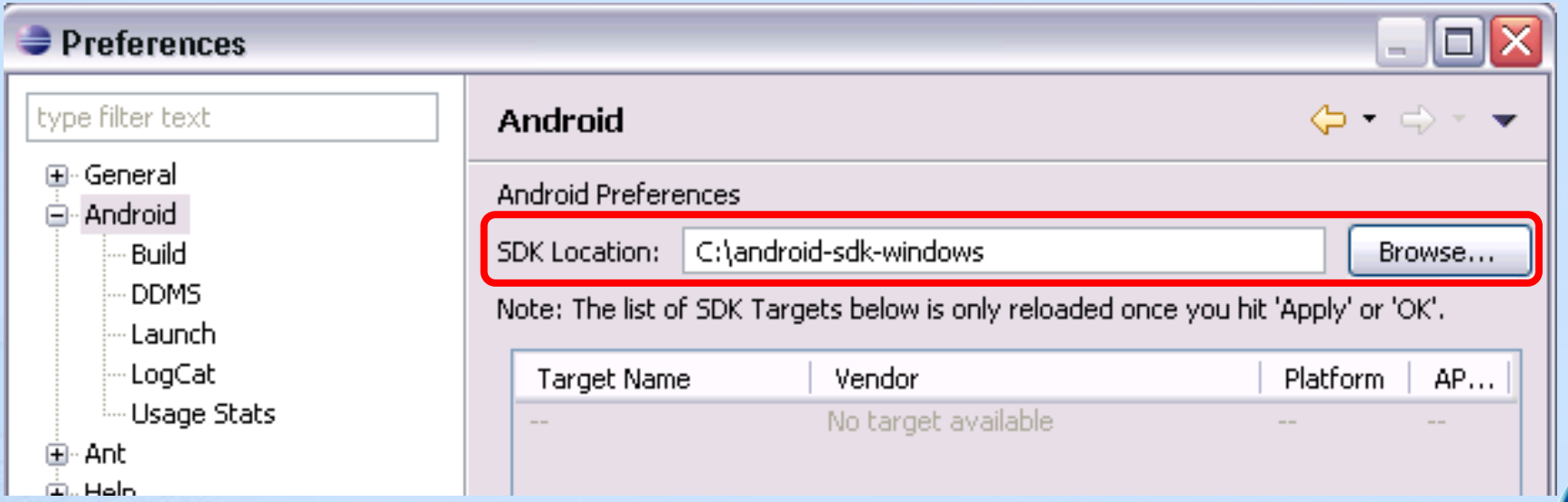
Dezvoltarea pe Eclipse IDE

Pregatirea dezvoltarii pe Eclipse IDE

http://www.vogella.de/articles/Android/article.html

- **configurarea Eclipse**
  - se selecteaza ***Windows -> Preferences***
  - apoi se selecteaza **calea instalarii Android SDK**

    c:\android-sdk-windows

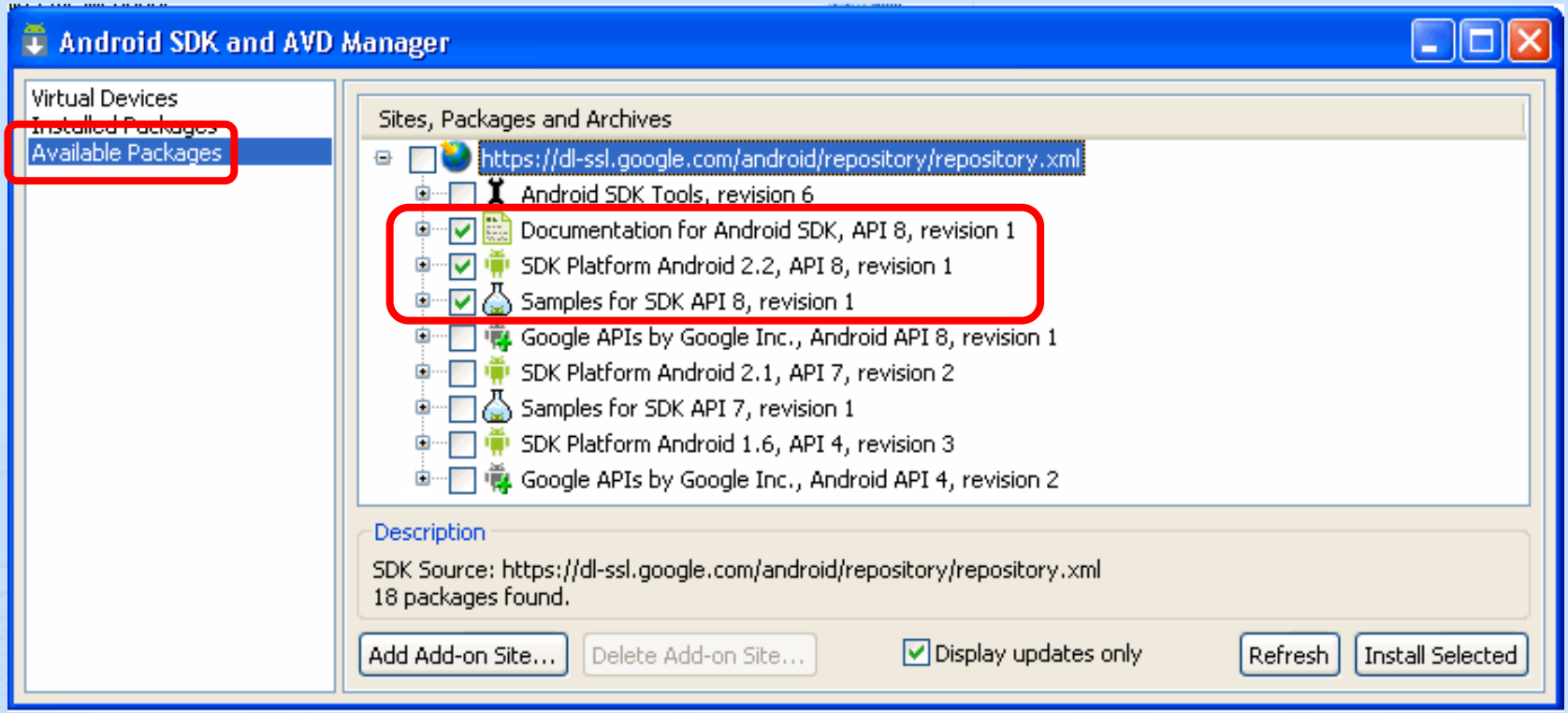


Dezvoltarea pe Eclipse IDE

Pregatirea dezvoltarii pe Eclipse IDE

http://www.vogella.de/articles/Android/article.html

- **configurarea Eclipse**
  - se selecteaza ***Window -> Android SDK and AVD Manager***
  - apoi se selecteaza ***Available Packages*** si **ultima versiune de SDK**



Dezvoltarea pe Eclipse IDE

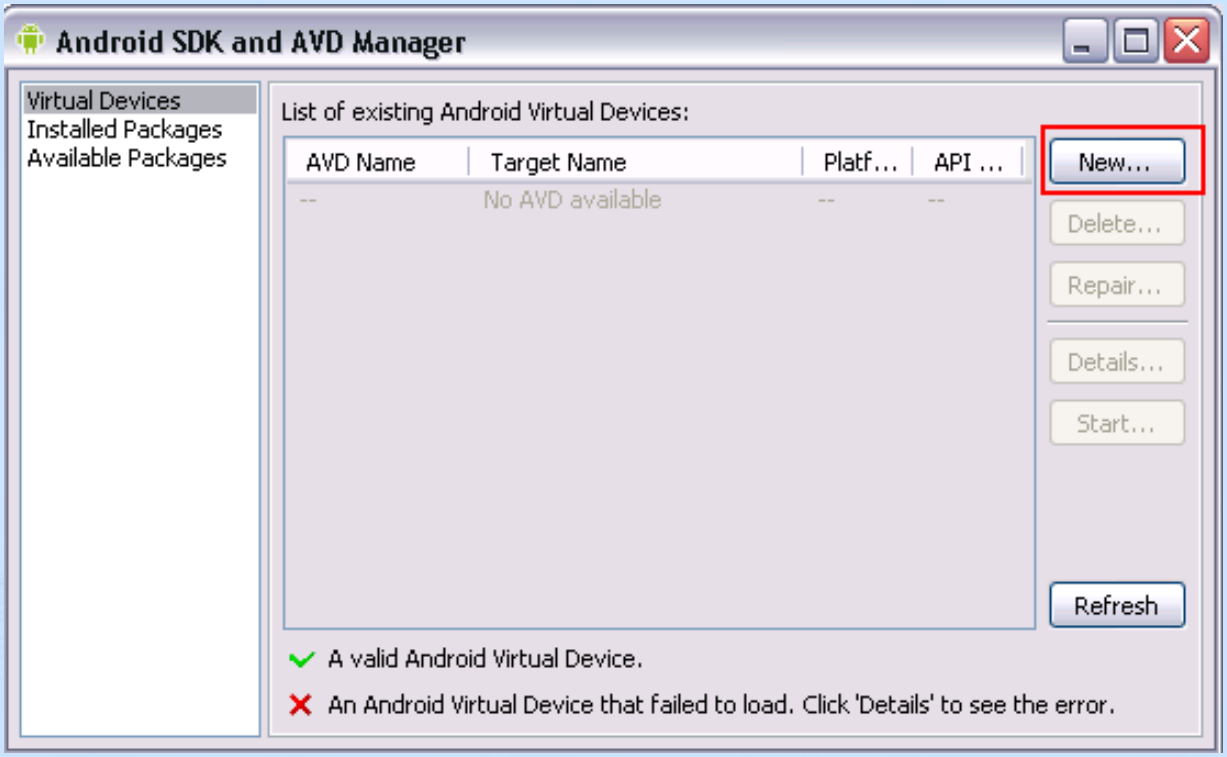
Pregatirea dezvoltarii pe Eclipse IDE

http://www.vogella.de/articles/Android/article.html

– **crearea unui terminal (*device*) pentru emulare**

– se apasa butonul ***device***

– apoi se selecteaza ***New***

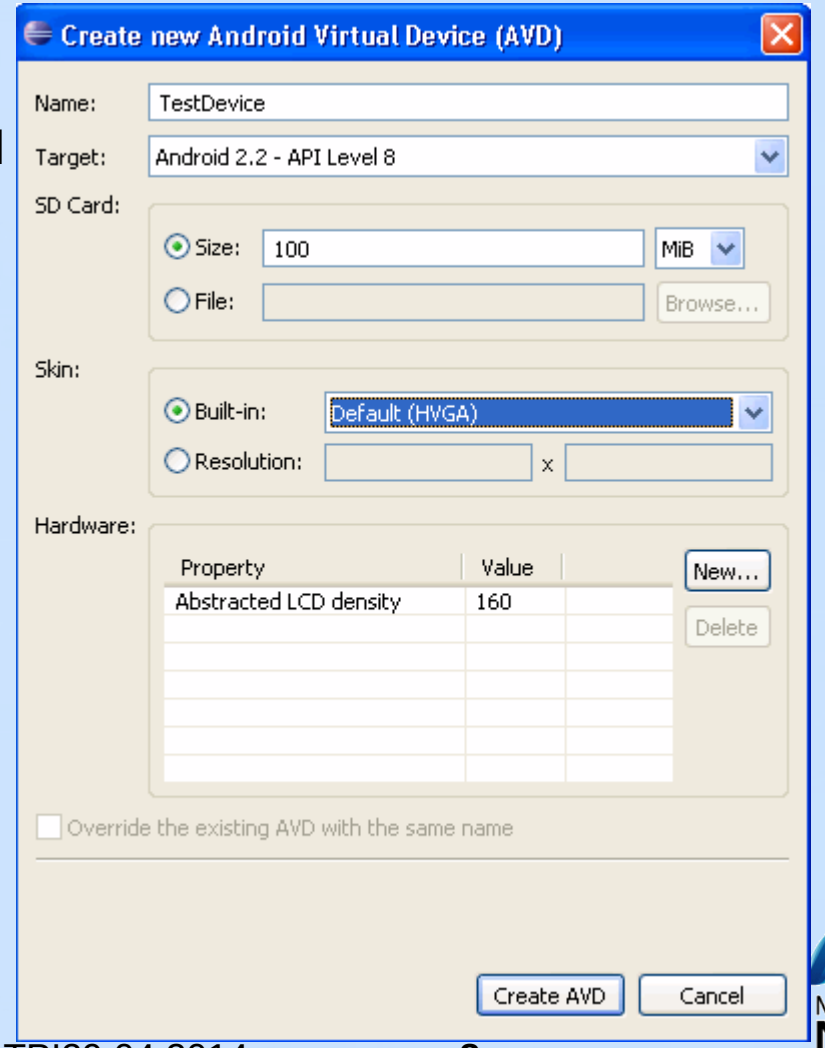


Dezvoltarea pe Eclipse IDE

Pregatirea dezvoltarii pe Eclipse IDE

http://www.vogella.de/articles/Android/article.html

- crearea unui terminal pentru emulare
  - se completeaza informatiile astfel
  - apoi **se apasa Create AVD**

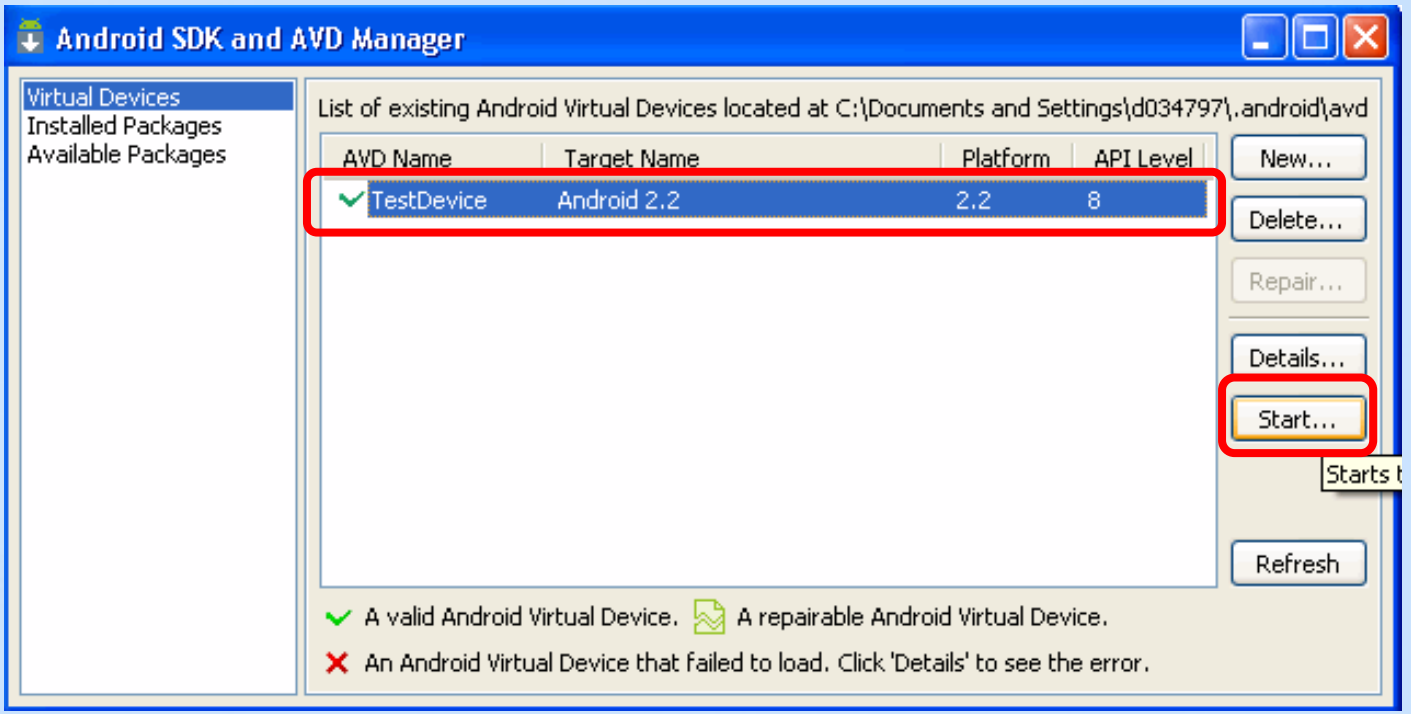


Dezvoltarea pe Eclipse IDE

Pregatirea dezvoltarii pe Eclipse IDE

http://www.vogella.de/articles/Android/article.html

- **crearea unui terminal pentru emulare**
  - in acest fel **este creat un *device***
  - **pentru a se testa daca e configurat corect** trebuie **selectat terminalul** si **apasat Start**



Dezvoltarea pe Eclipse IDE

Pregatirea dezvoltarii pe Eclipse IDE

http://www.vogella.de/articles/Android/article.html

– dupa mult timp terminalul pentru emulare este gata

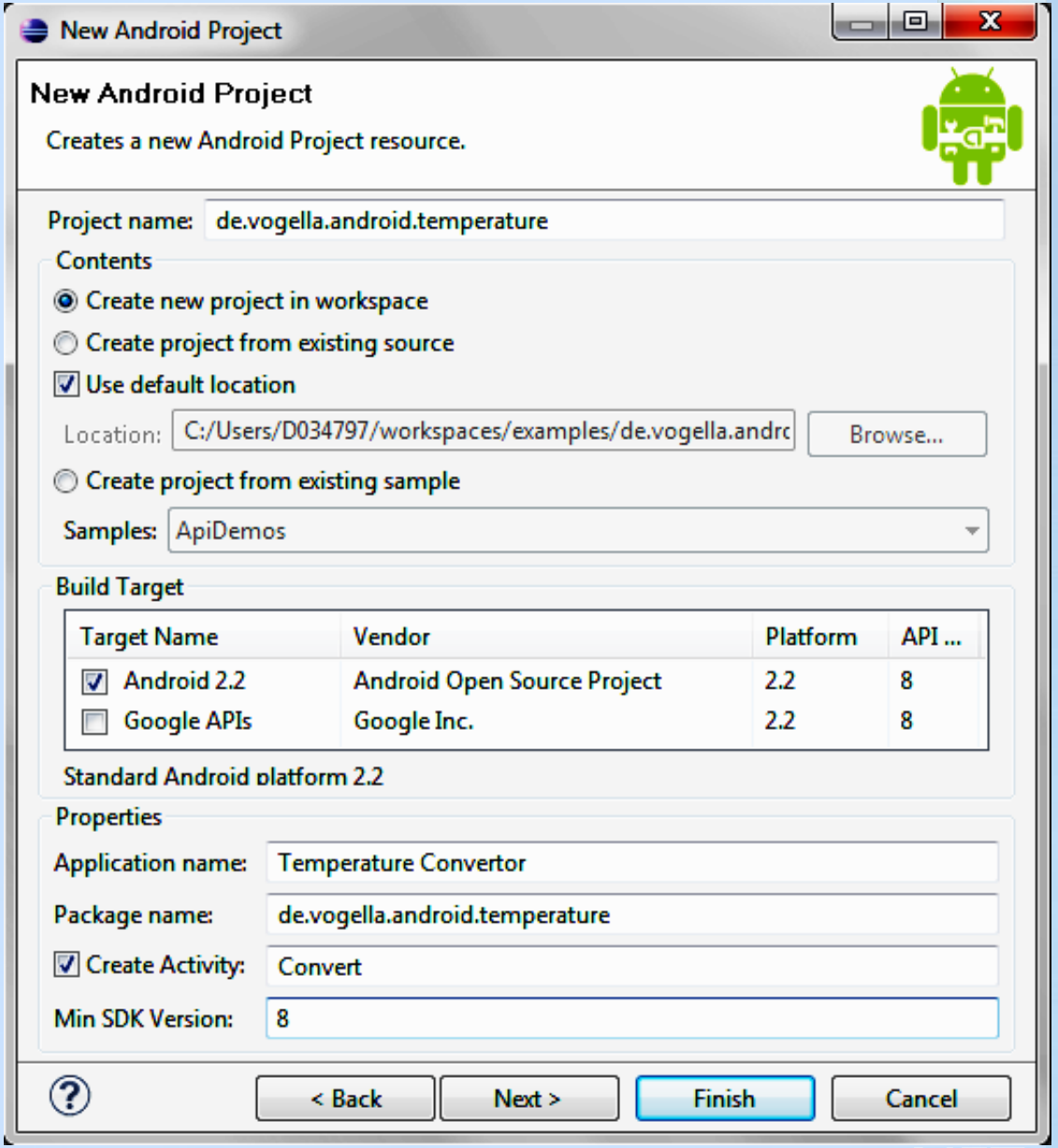
Dezvoltarea pe Eclipse IDE

Crearea unui proiect

– se selecteaza ***File -> New -> Other -> Android -> Android Project***

– se creeaza un proiect Android, **de.vogella.android.temperature** cu **urmatoarele valori**

– se apasa ***Finish***



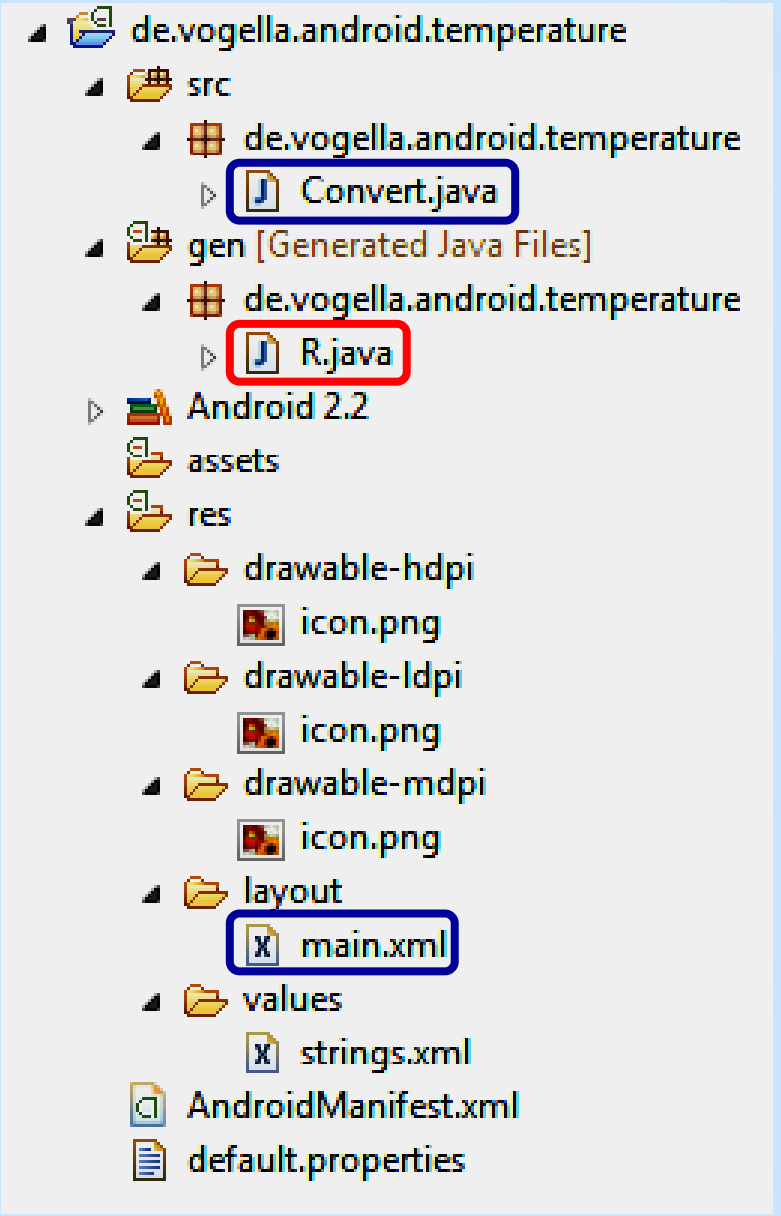
Dezvoltarea pe Eclipse IDE

Crearea unui proiect

– va fi creata urmatoarea structura de directoare

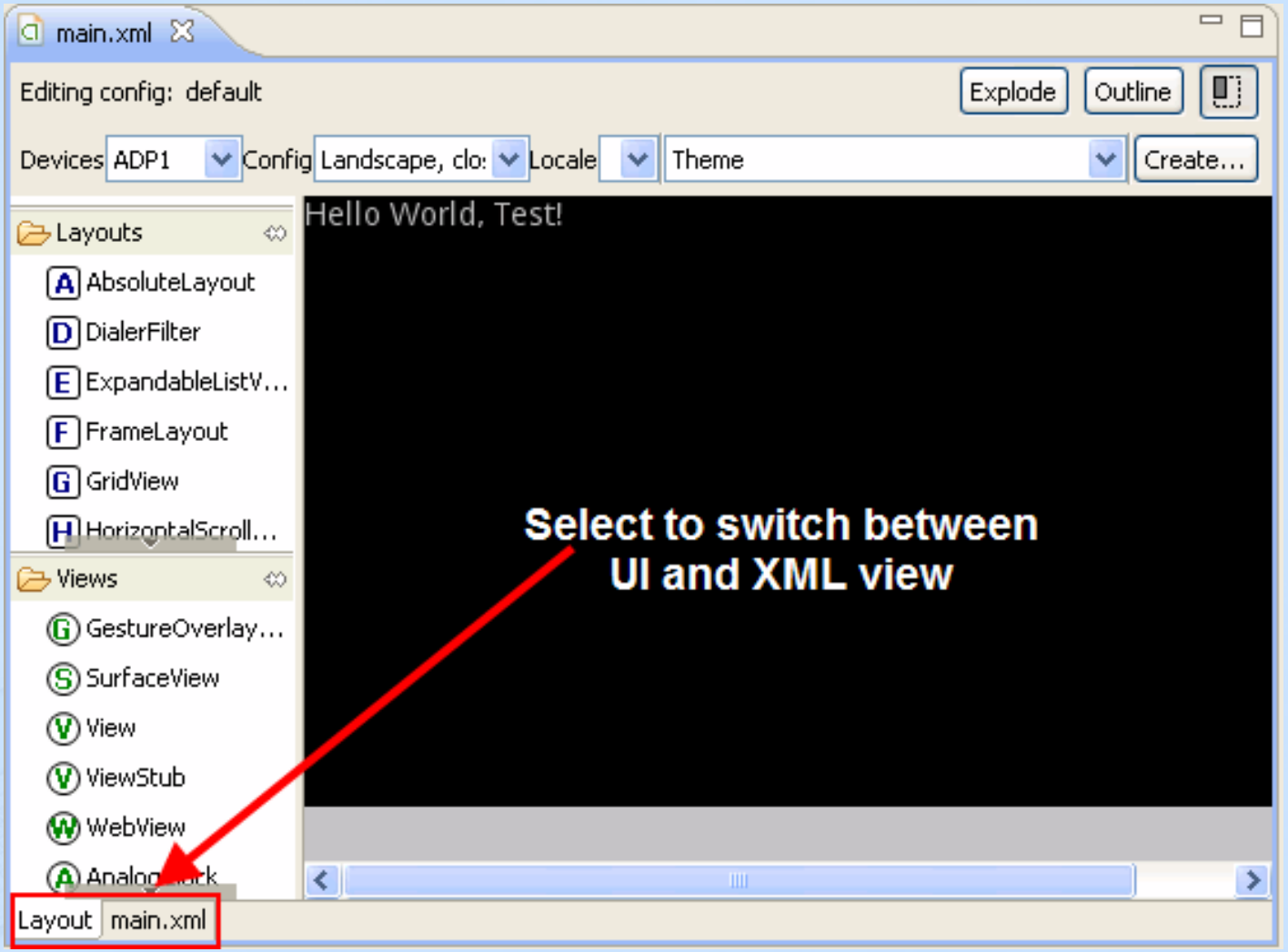
– "**R.java**" este clasa generata automat care contine textul si elementele de interfata UI elements

– **nu trebuie incercata modificarea manuala a acestei clase**



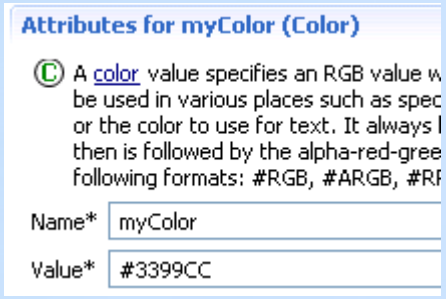
Dezvoltarea pe Eclipse IDE

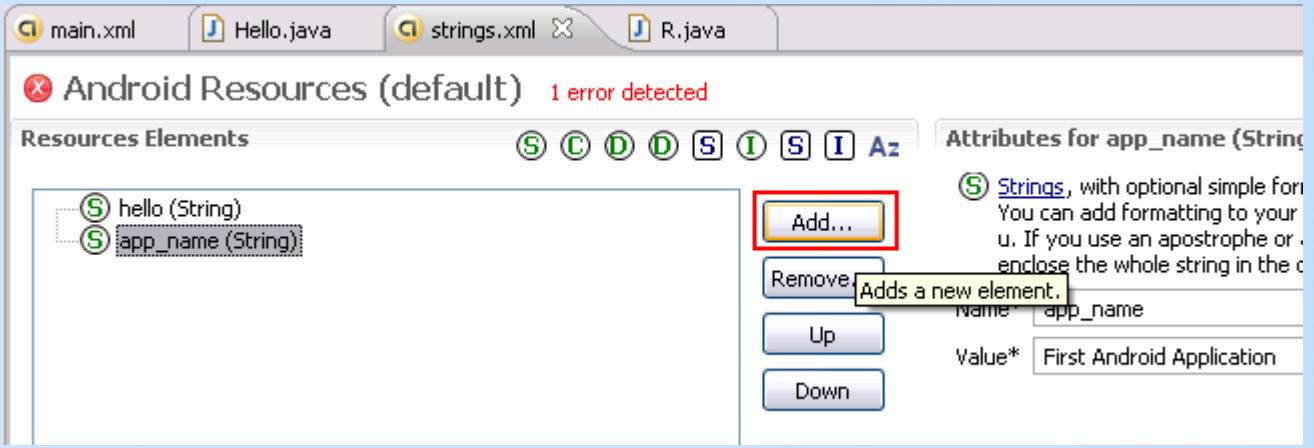
Cele doua perspective - *rich editor* sau direct **cod XML**



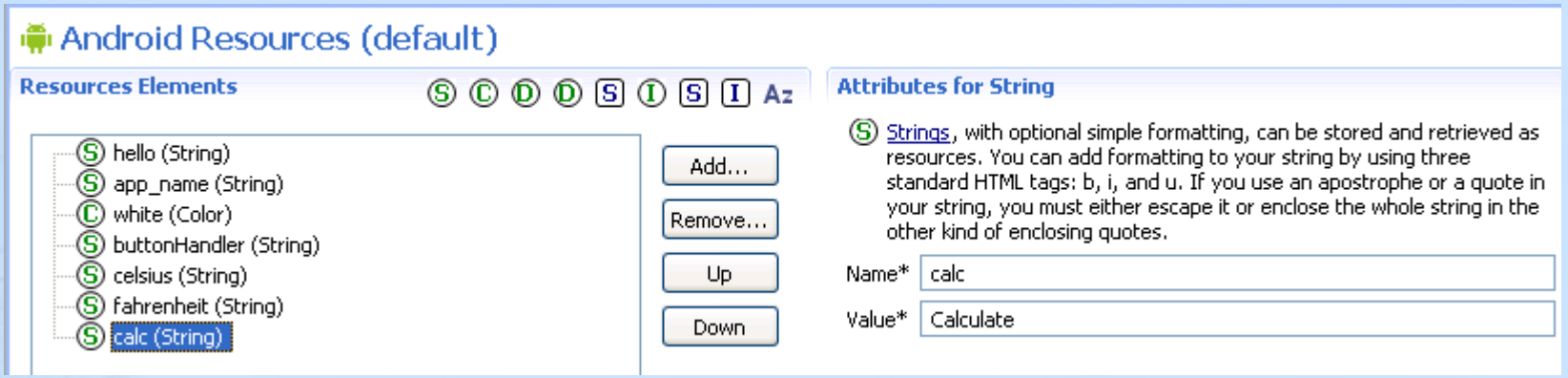
Dezvoltarea pe Eclipse IDE

Crearea unor atribute



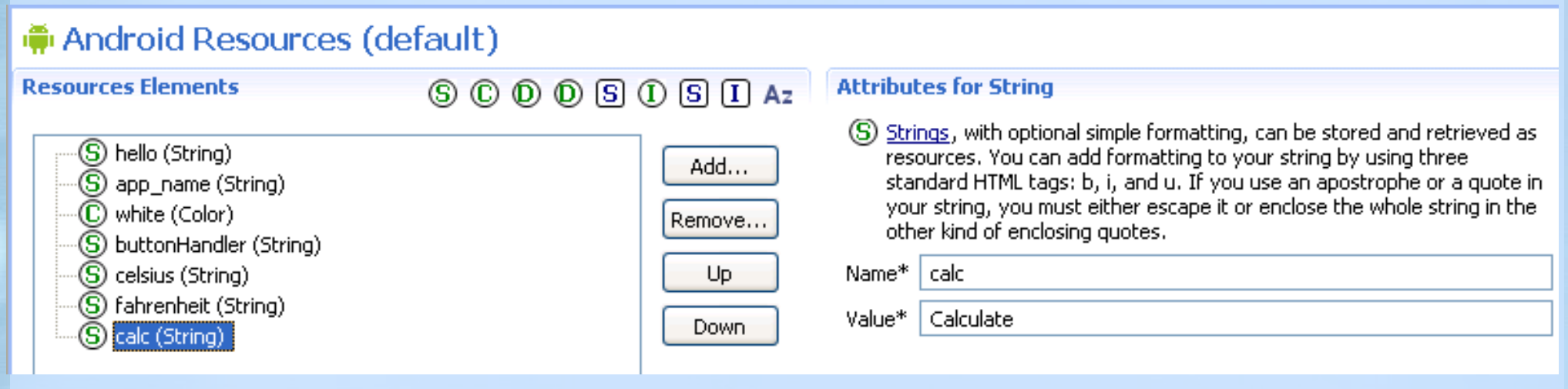


Name	Value
buttonHandler	myClickHandler
celsius	to Celsius
fahrenheit	to Fahrenheit



Dezvoltarea pe Eclipse IDE

Atributele – cod XML

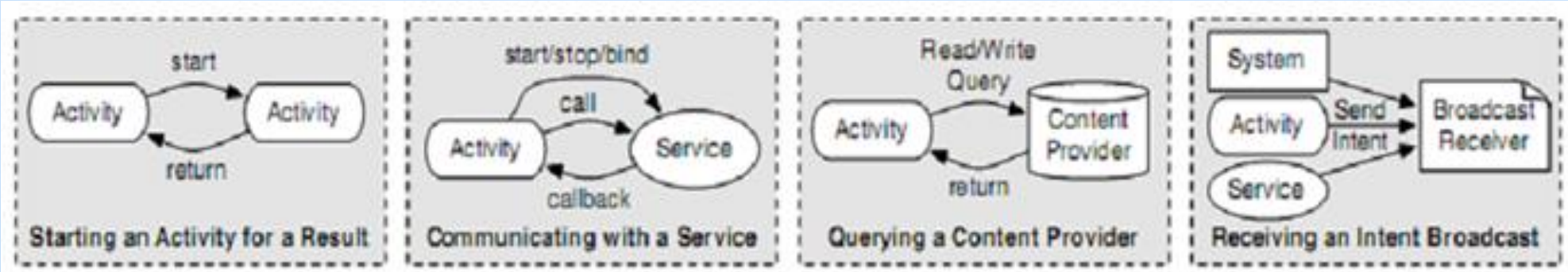



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
        <string name="hello">Hello World, Convert!</string>
        <string name="app_name">Temperature Converter</string>
        <color name="myColor">#3399CC</color>
        <string name="buttonHandler">myClickHandler</string>
        <string name="celsius">to Celsius</string>
        <string name="fahrenheit">to Fahrenheit</string>
        <string name="calc">Calculate</string>
</resources>
```

Componentele aplicatiilor Android

Exista 4 tipuri de componente

- **Activitati**
  - **interfete grafice** corespunzatoare actiunilor utilizatorului
- **Servicii**
  - **procese fara interfata** care sunt executate (ruleaza) in *background*
- ***Broadcast Receivers***
  - **receptioneaza notificari** si **reactioneaza** la ele
- **Furnizori continut**
  - **pun la dispozitie** altor aplicatii un set specific de **date**



Intent – descrierea abstracta a unei operatii

Trimiterea explicita a unui obiect **Intent**

– **nu** se foloseste **Intent-filter**

```
Intent intent = new Intent(Intent.ACTION_MAIN, null);
intent.addCategory(Intent.CATEGORY_LAUNCHER);
ComponentName cn = new ComponentName("com.android.settings",
                "com.android.settings.fuelgauge.PowerUsageSummary");
intent.setComponent(cn);
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
startActivity(intent);
```

Definirea unui Intent pentru trimitere implicita

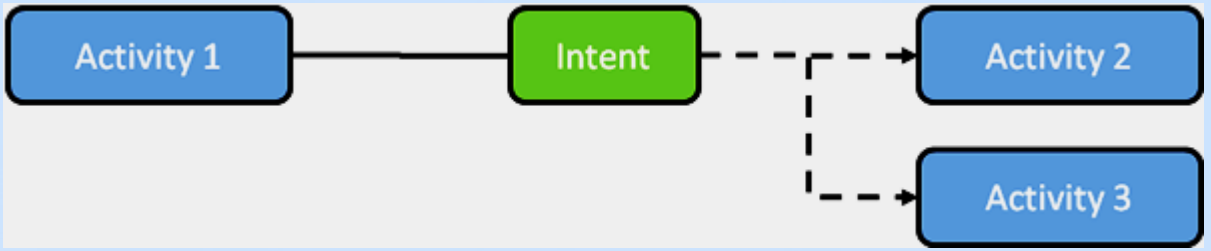
```
Intent intent = new Intent(Intent.ACTION_CALL);
intent.setData(Uri.parse("tel:555-1234"));
```

Intent – descrierea abstracta a unei operatii

Trimiterea implicita a unui obiect **Intent** catre entitatile ce **pot executa `ACTION_CALL`**

– se foloseste **Intent-filter**

```
startActivity(intent);
```



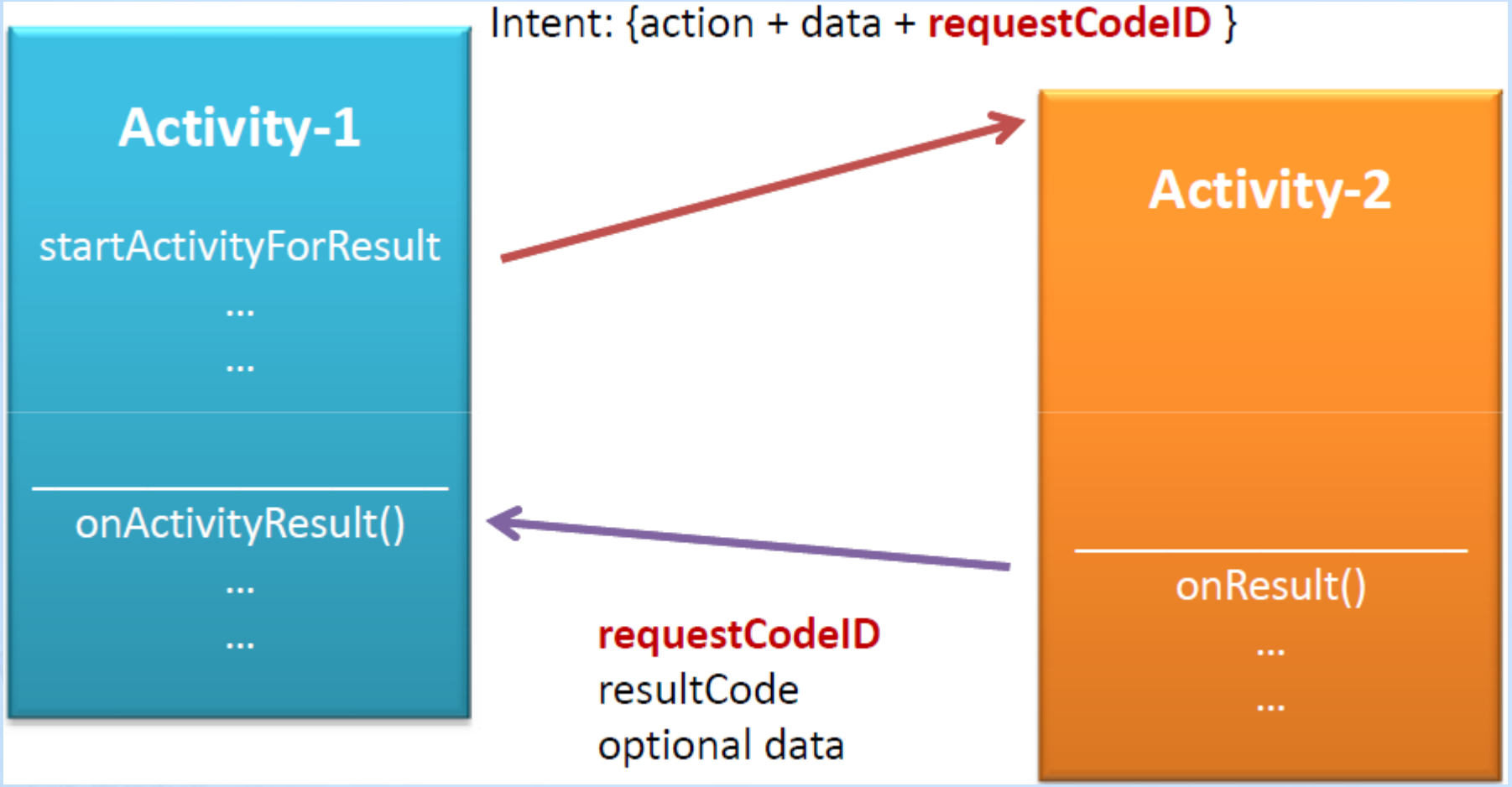
Pot executa **`ACTION_CALL`**

Definirea Intent-filter in fisierul **Androidmanifest.xml**

```
<activity android:name=".activity.Dashboard" android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.CALL" />
    </intent-filter>
</activity>
```

Intent – descrierea abstracta a unei operatii

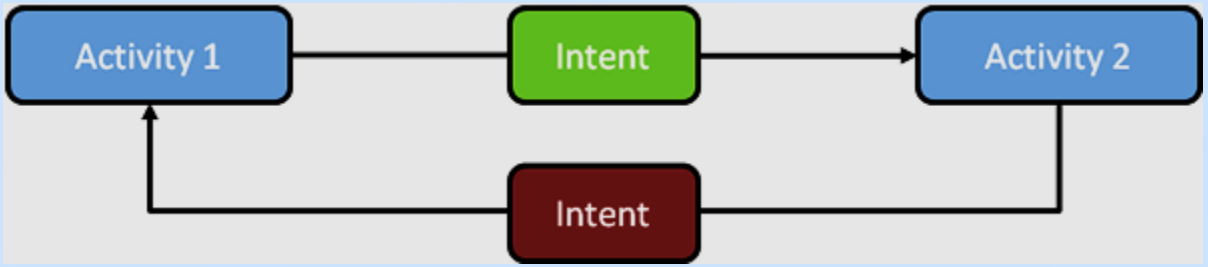
Trimiterea explicita a unui obiect **Intent** cu `startActivityForResult()` de catre `Activity1`



Intent – descrierea abstracta a unei operatii

Trimiterea explicita a unui obiect **Intent** cu `startActivityForResult()` de catre `Activity1`

```
Intent intent = new Intent(Intent.ACTION_PICK, ContactsContract.Contacts.CONTENT_URI);
startActivityForResult(intent, 9999);
```



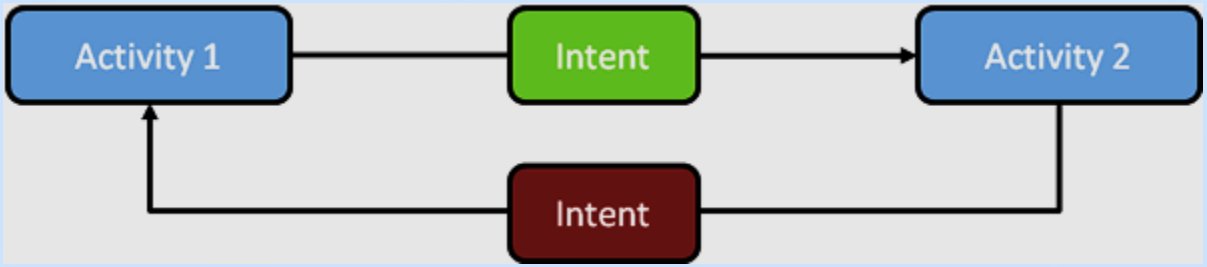
`Activity2` corespunde cu `intent` si e afisata si executata, iar inainte de a returna, apeleaza `setResult(data)`

```
Intent data = new Intent();
Uri uri = Uri.parse("content://com.android.contacts/contacts/lookup/318c5d/2098");
data.setData(uri);
setResult(Activity.RESULT_OK, data);
finish();
```

Intent – descrierea abstracta a unei operatii

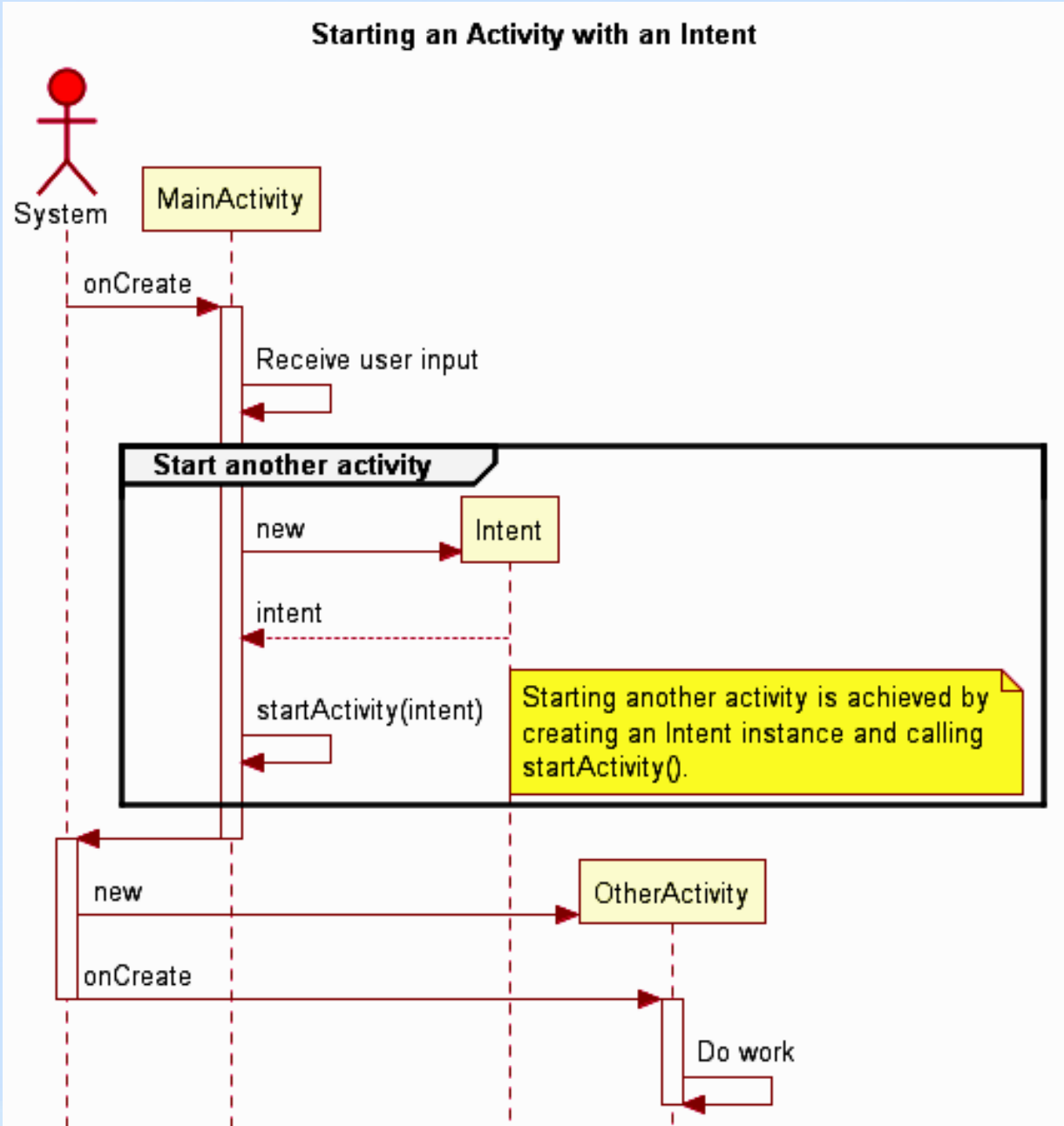
Atunci cand e apelata metoda `onActivityResult()`

– `Activity1` **primeste** un `Intent` **rezultat**

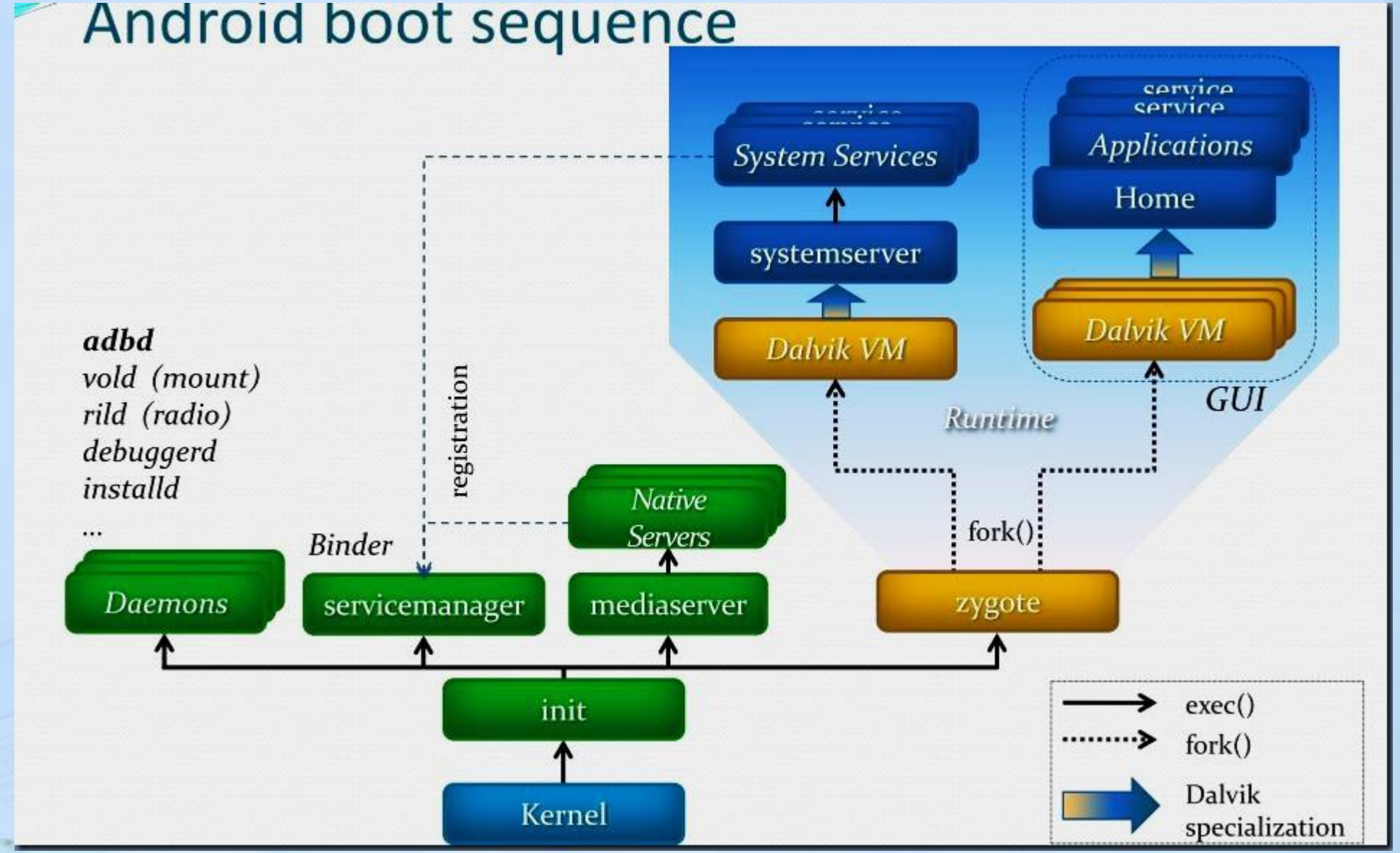



```
public void onActivityResult(int requestCode, int resultCode, final Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode==9999 && resultCode==Activity.RESULT_OK) {
        Uri contactData = data.getData();
        // contactData = content://com.android.contacts/contacts/lookup/318c5d/2098
    }
}
```

Componentele aplicatiilor Android

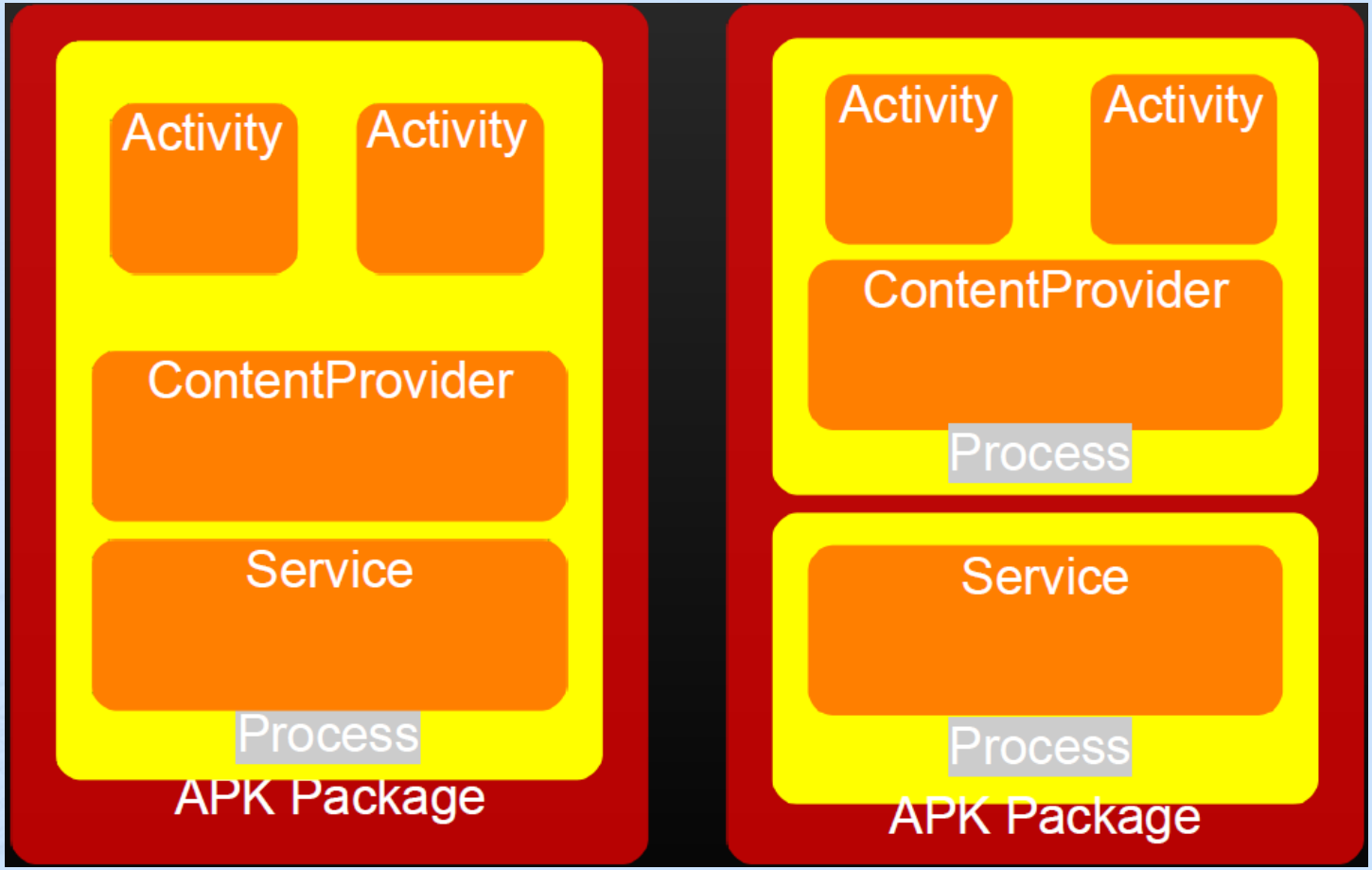


Componentele aplicatiilor Android



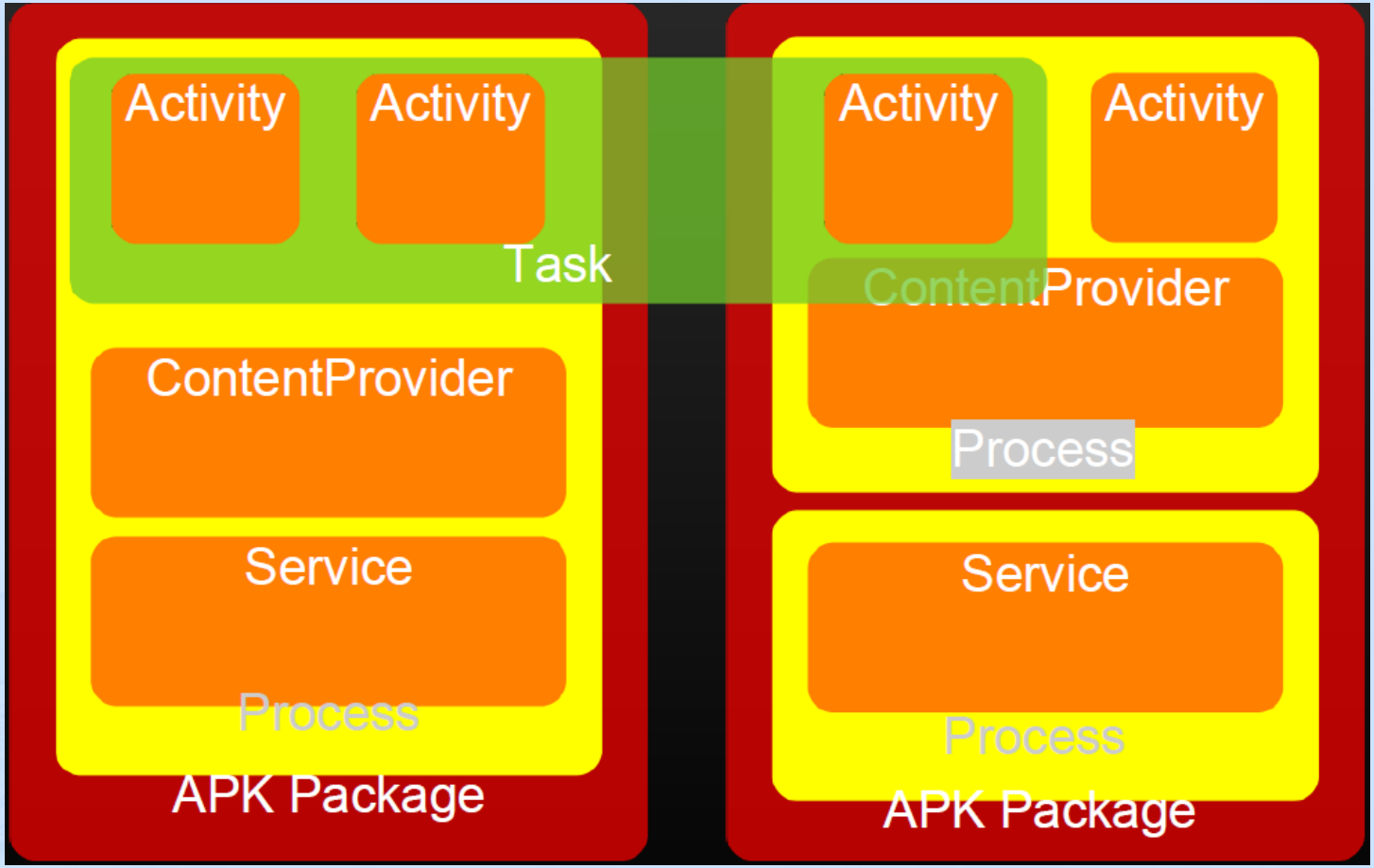
Activitati, task-uri, procese

Activitatea este ca o “molecula” – o portiune distincta de functionalitate
Un ***task*** este – o colectie de Activitati
Un **proces** este – un proces Linux standard



Activitati, task-uri, procese

Activitatea este ca o “molecula” – o portiune distincta de functionalitate
Un ***task*** este – o colectie de Activitati
Un **proces** este – un proces Linux standard



Activitati, task-uri, procese

Activitate

– **clasa concreta** in API

– **incapsulare** a unei **operatii**

– ruleaza **in procesul** .APK-ului **care a instalat-o**

– optional asociata cu o “**fereastra**” (UI)

Task

– o **notiune** (nu o entitate concreta)

– o **colectie de Activitati** legate intre ele

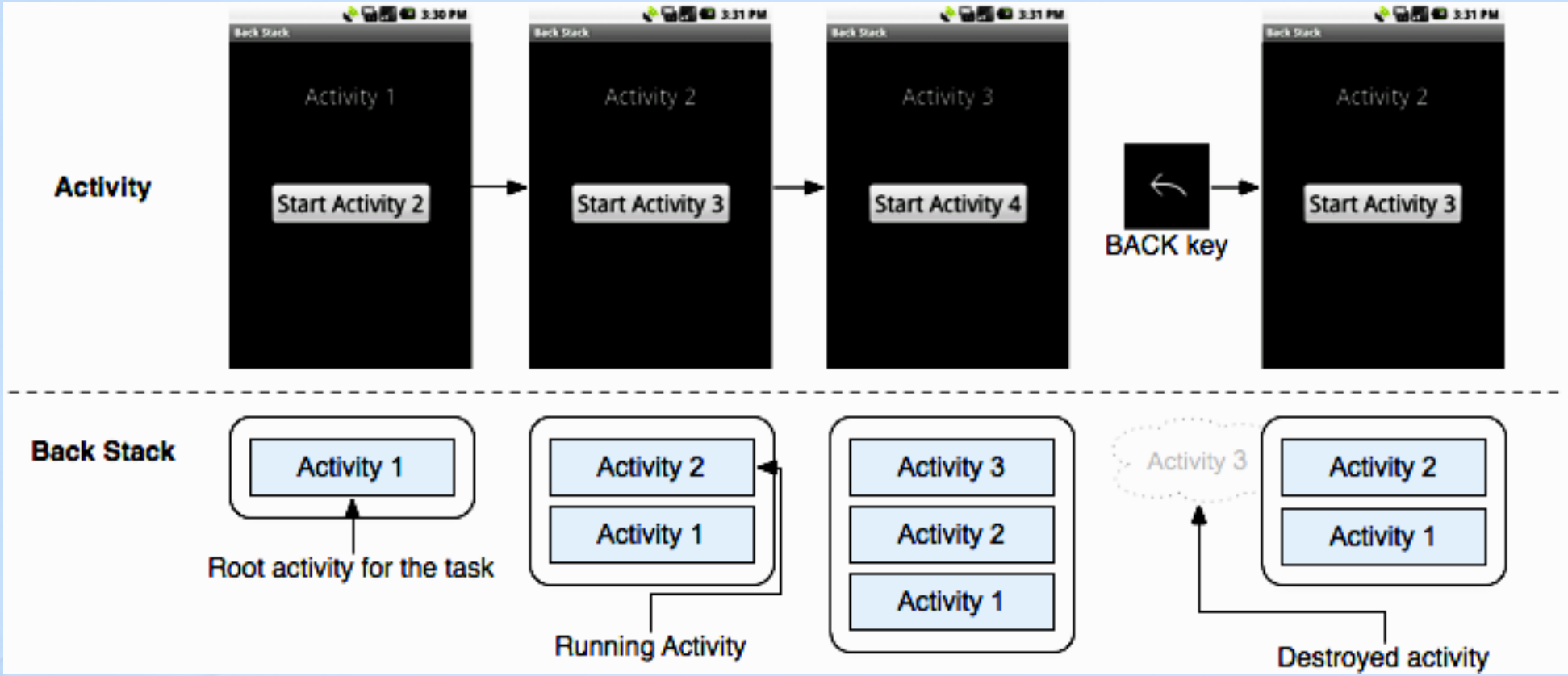
– capabil sa se intinda **peste mai multe procese**

– asociat cu **propria stiva a istoricului UI** (“fereastrelor”)

– similar **aplicatiilor** de pe alte platforme

Activitati, task-uri, procese

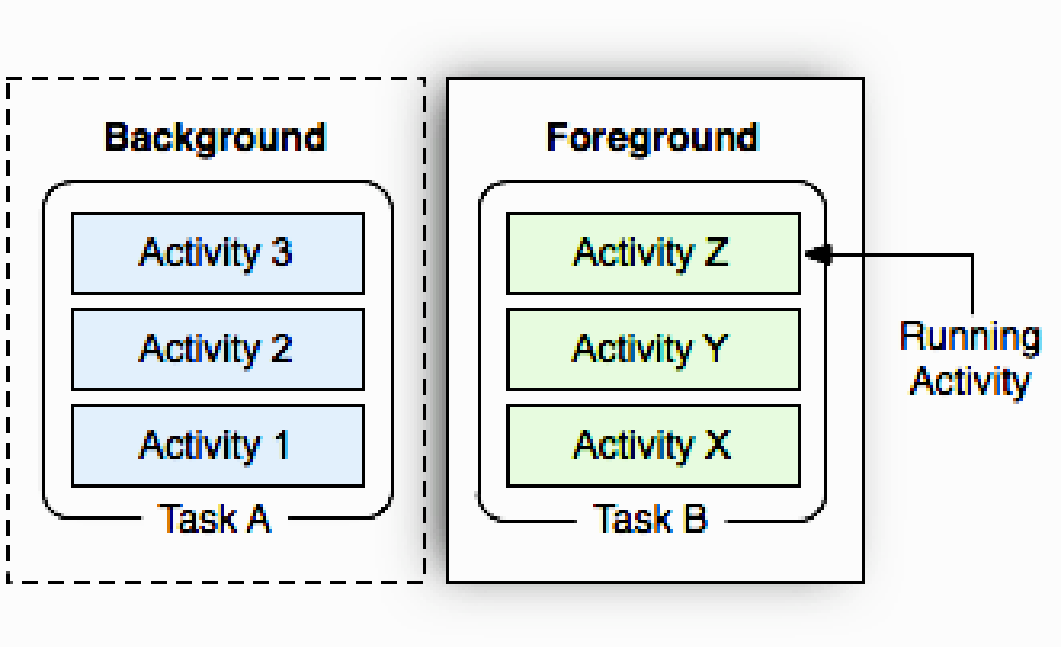
Task – o **colectie de Activitati** legate intre ele, are **propria stiva a istoricului UI**

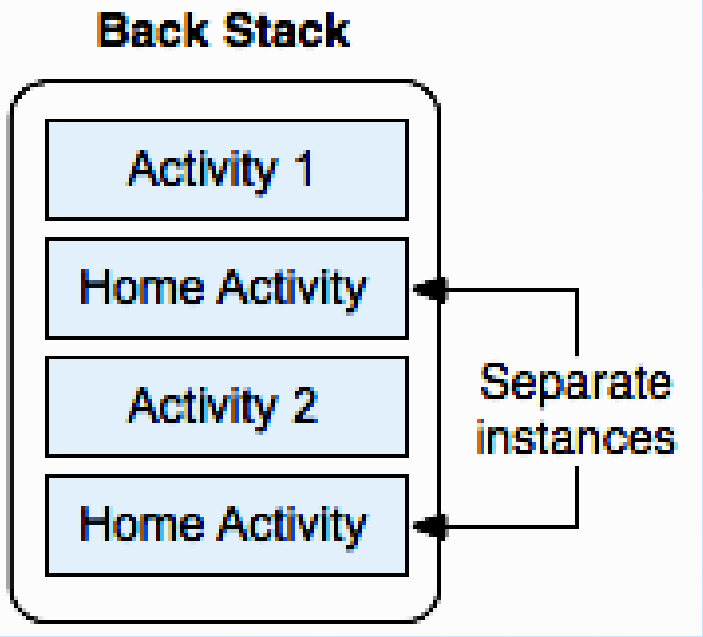


Fiecare noua activitate intr-un *task* adauga un **element** in ***back stack***. *Cand* ***utilizatorul apasa*** *tasta* ***BACK***, ***activitatea curenta*** *e* ***distrusa*** *si cea* ***anterioara reluata***

Activitati, task-uri, procese

Task – o **colectie de Activitati** legate intre ele, are **propria stiva a istoricului UI**



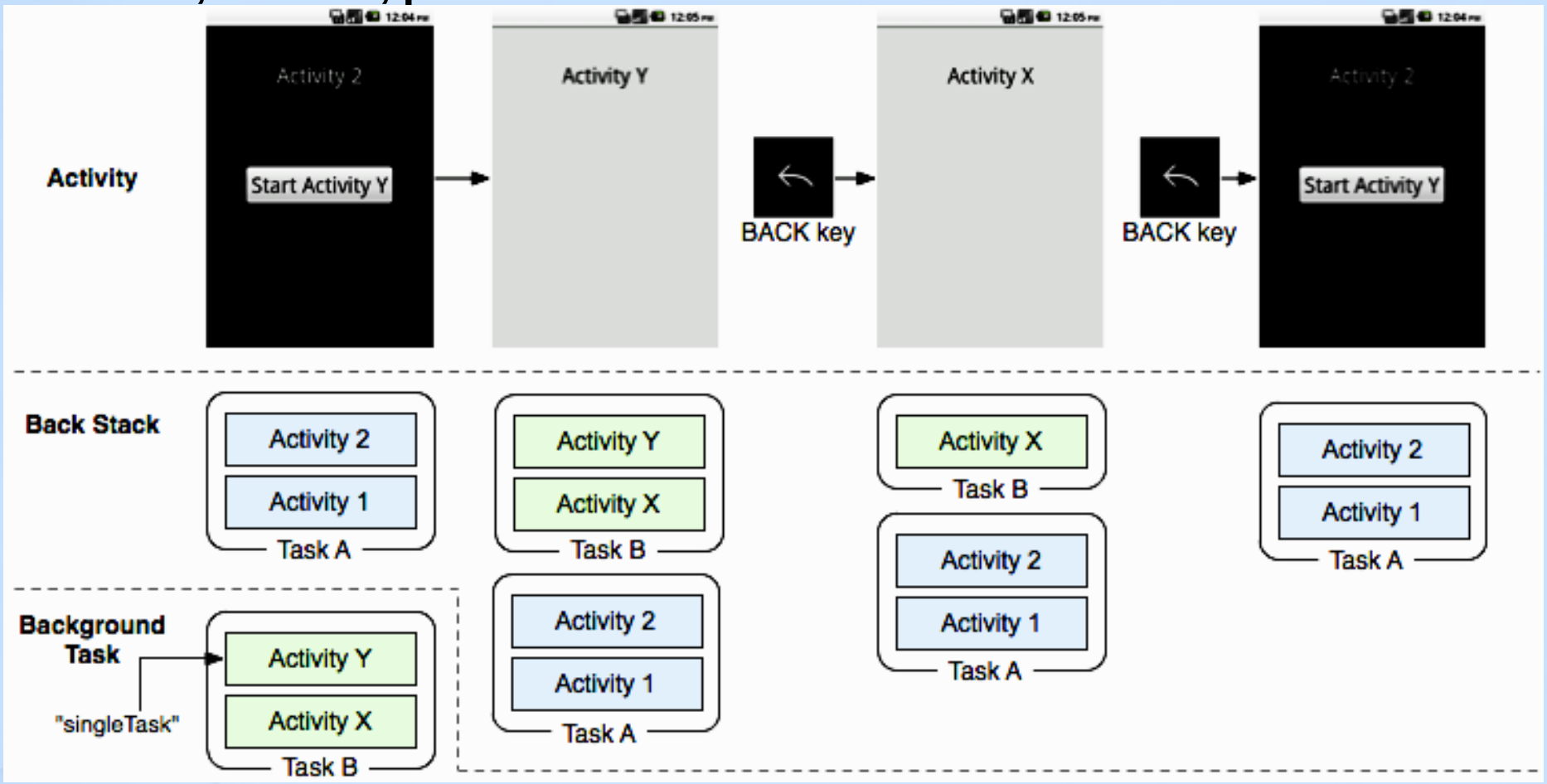


Doua *task*-uri*:* *Task*-ul ***A*** este in ***background***, asteptand sa fie reluat, in timp ce *Task*-ul ***B*** interactioneaza cu utilizatorul in ***foreground***.

O **activitate** poate fi instantiata **de mai multe ori**

*http://**developer.android.com**/guide/**topics**/fundamentals/**tasks-and-back-stack.html***

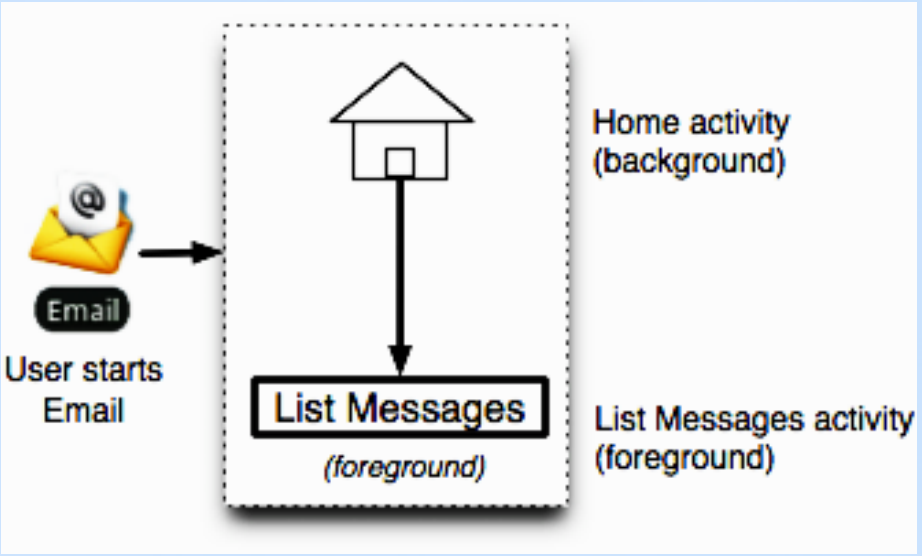
Activitati, task-uri, procese



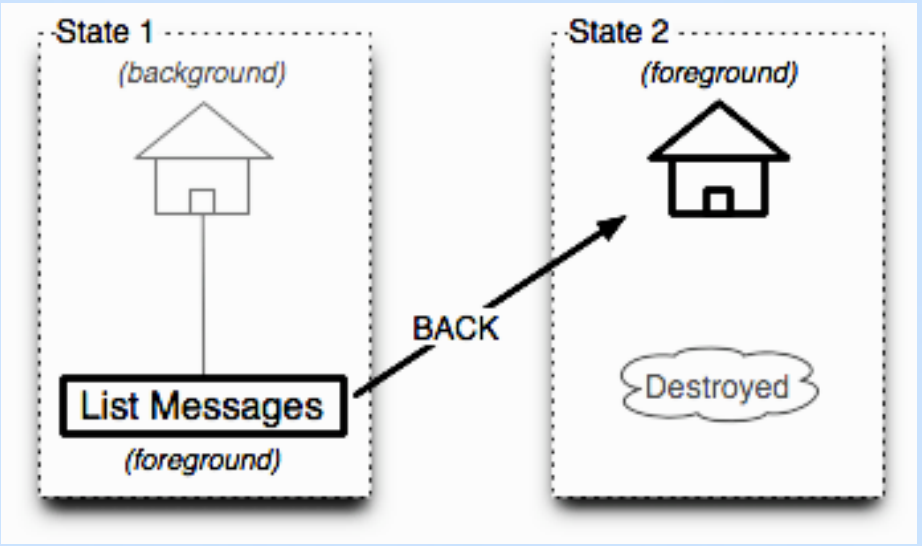
Daca o **activitate** este deja **parte a unui *task*** din ***background*** cu propria *back stack* ***(B)***, atunci **intreaga *back stack* revine in prim plan**, in varful *task*-ului curent ***(A)***.

Activitati, task-uri, procese

Lansarea unei *Activity* din *Home*

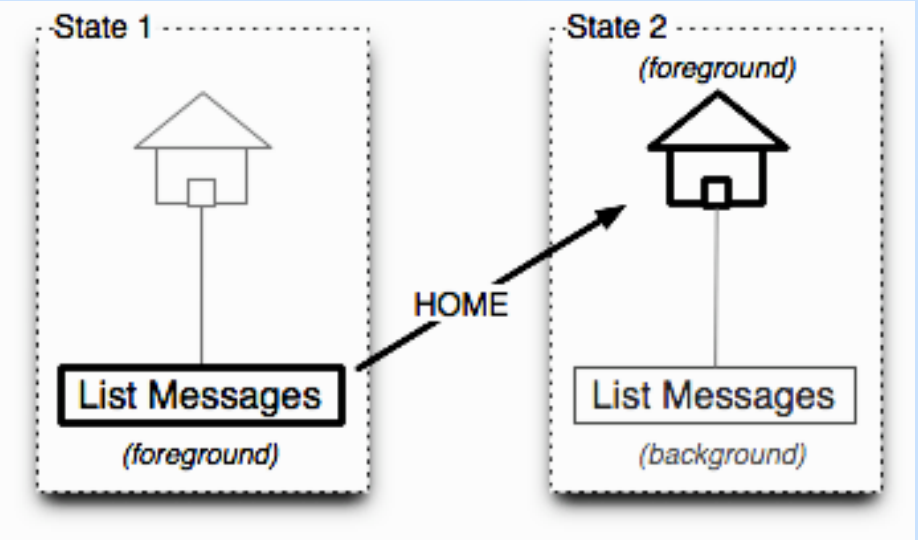


Iesirea dintr-o *Activity* cu tasta *BACK*



Activitati, task-uri, procese

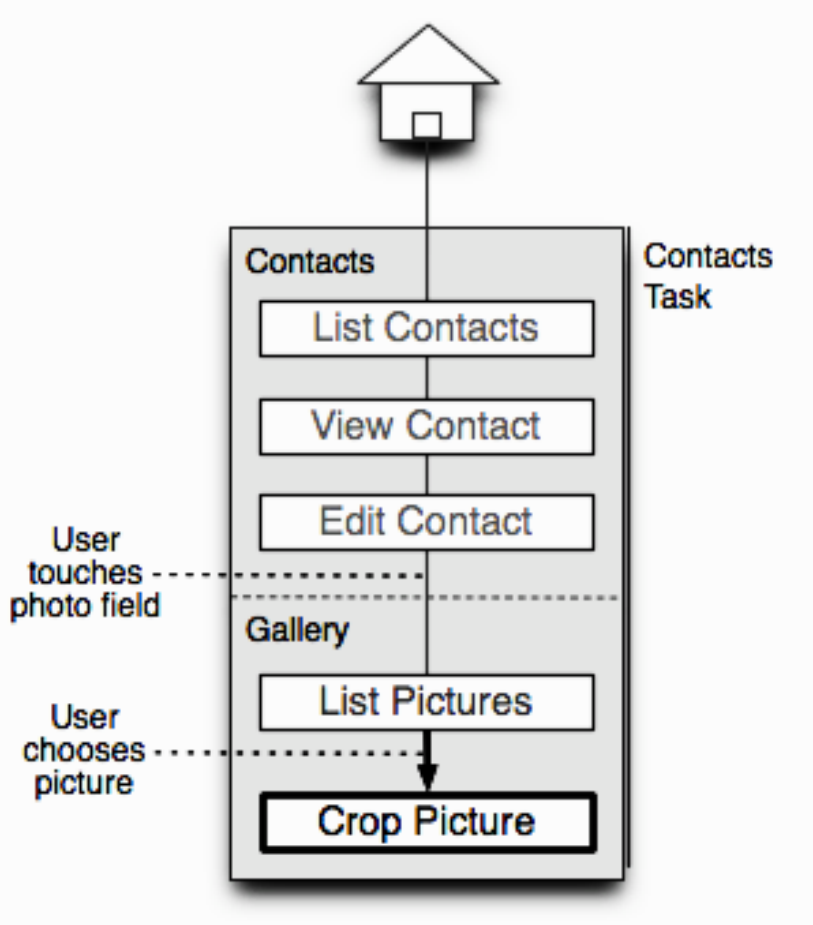
Iesirea dintr-o *Activity* cu tasta *HOME*



*http://**developer.android.com/**guide**/practices/**ui_guidelines**/activity_task_design.html***

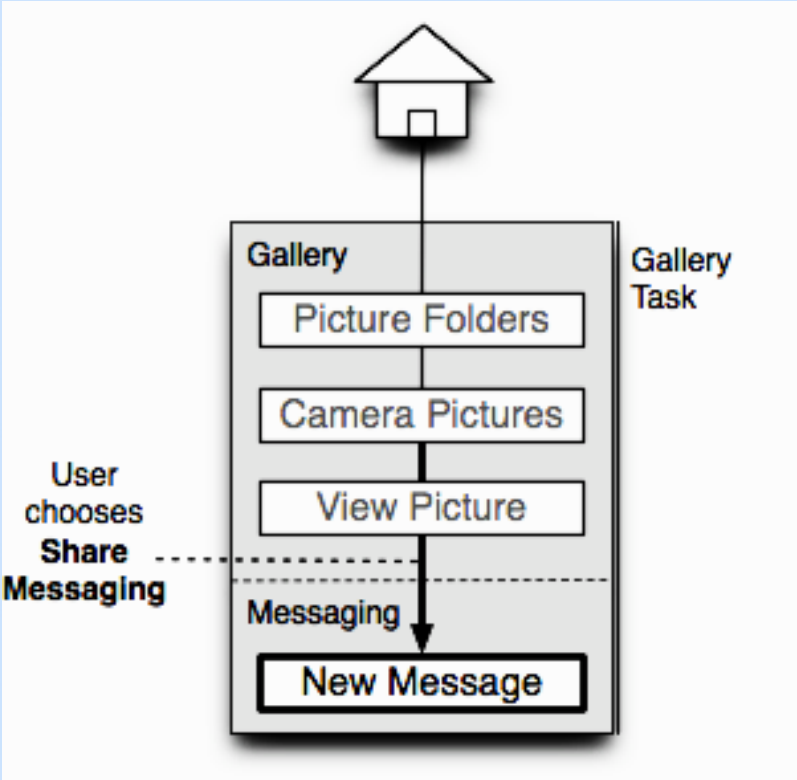
Activitati, task-uri, procese

Reutilizarea unei *Activity* – *Contacts* reutilizeaza *Gallery* pentru a *Obtine o imagine*



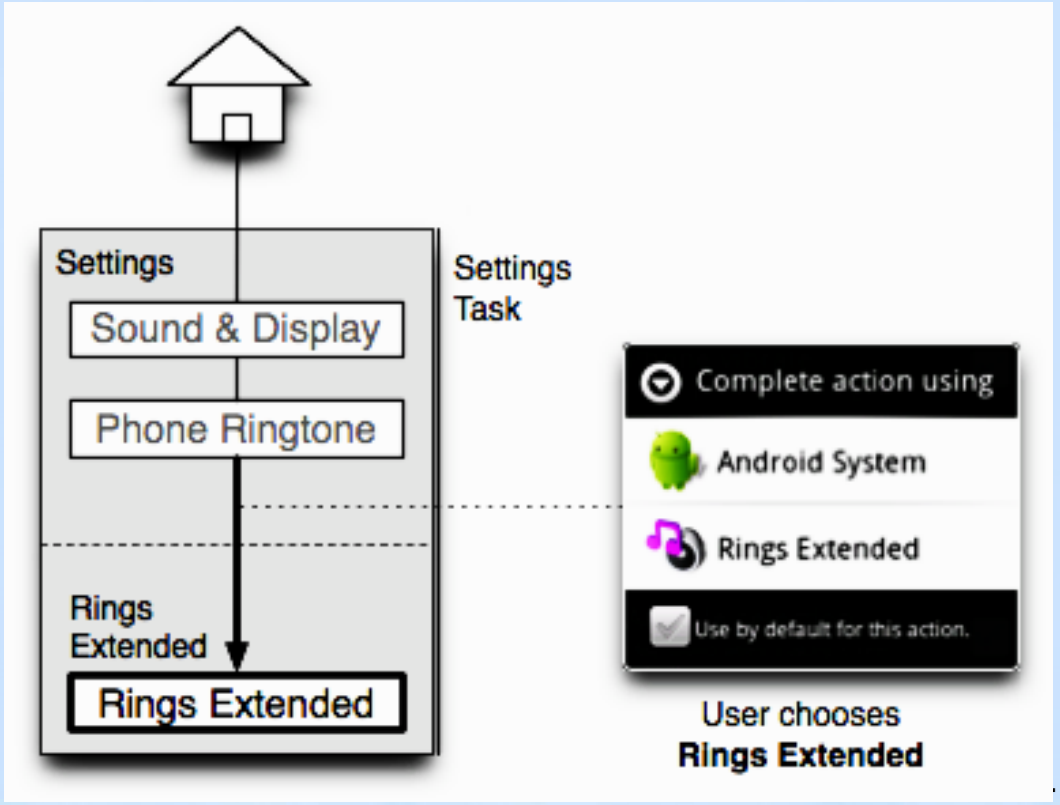
Activitati, task-uri, procese

Reutilizarea unei *Activity* – *Gallery* reutilizeaza *Messaging* pentru a *Partaja o imagine*



Activitati, task-uri, procese

Replacing an Activity
– *the* ***user downloads a replacement for Phone Ringtone*** *activity,* ***Rings Extended***
– when user *goes to* ***Settings > Sound & Display > Phone Ringtone****, the* ***system presents a choice*** *between the Android* ***System's ringtone activity*** *and* ***the new one***
– *the* ***dialog box*** *has an* ***option*** *to* ***remember choice "Use by default for this action"***
– *when choose "****Rings Extended****", that activity loads,* ***replacing*** *the original Android ringtone activity*



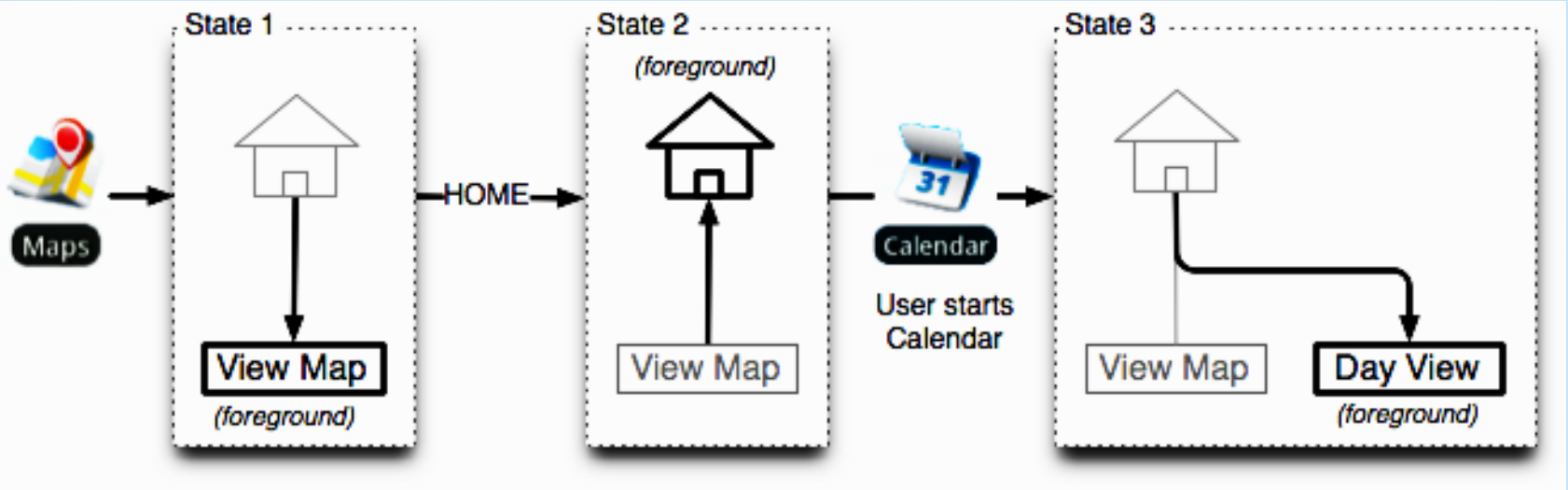
Activitati, task-uri, procese

Multitasking

– ***State 1*** *- The* ***user*** ***launches*** *the* ***View Map*** *activity and* ***searches*** *for a map* ***location****. Let's say the map is* ***taking an unusually long time to draw***

– ***State 2*** *- The* ***user wants to do something else*** *while they're waiting, so* ***press HOME****, which* ***does not interrupt*** *the* ***map's*** *network connection and* **allows** *the map to* ***continue loading*** *in the* ***background***

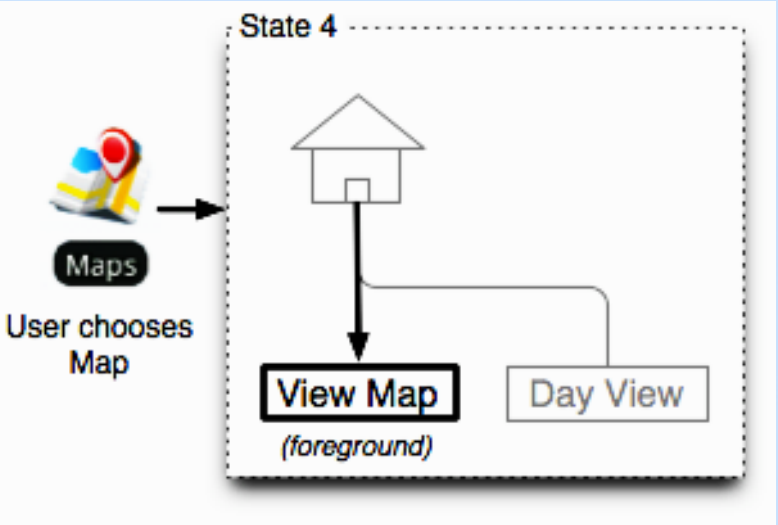
– ***State 3*** *- The* ***map*** *activity is* ***now running in the background****, with* ***Home*** *in the* ***foreground****. The* ***user*** *then* ***launches*** *the* ***Calendar*** *activity, which launches into the foreground, taking user focus, where they view today's calendar*

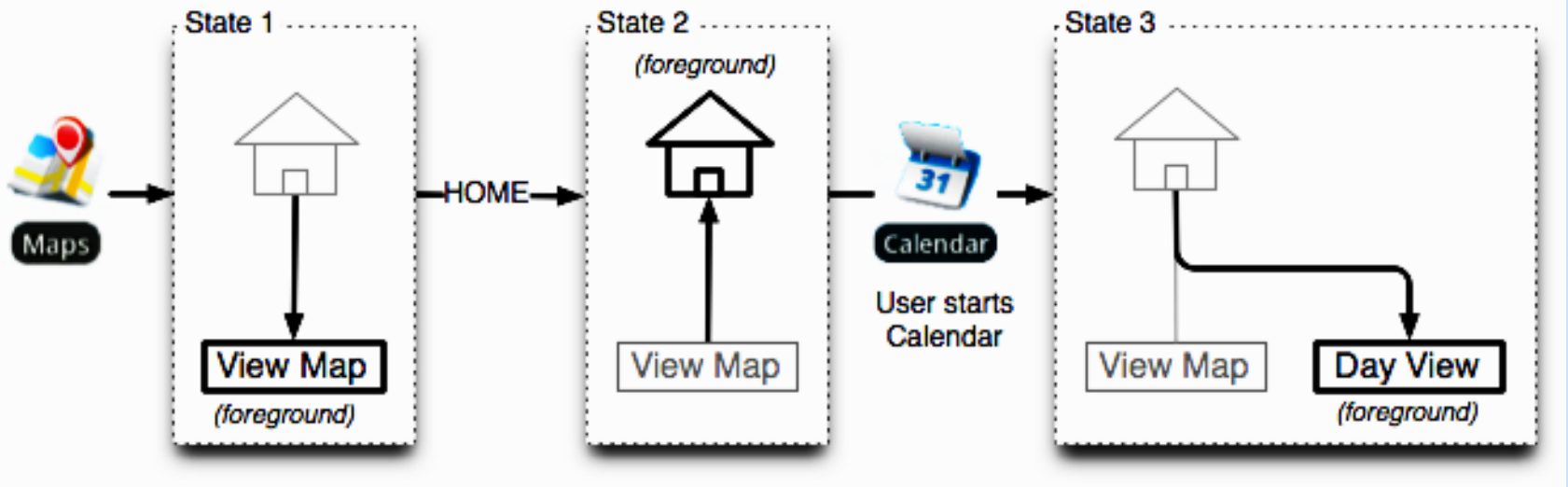


Activitati, task-uri, procese

Multitasking

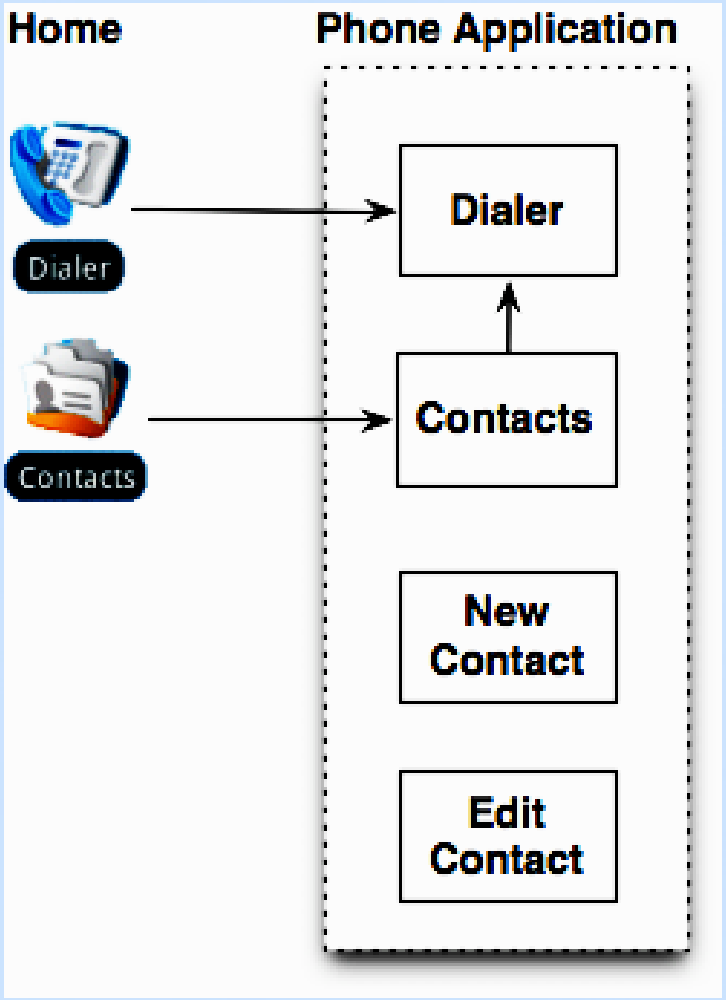
***– State 4** - The user **presses** **Home**, then **Maps** to **return** to the **map**, which by now has fully loaded*





Activitati, task-uri, procese

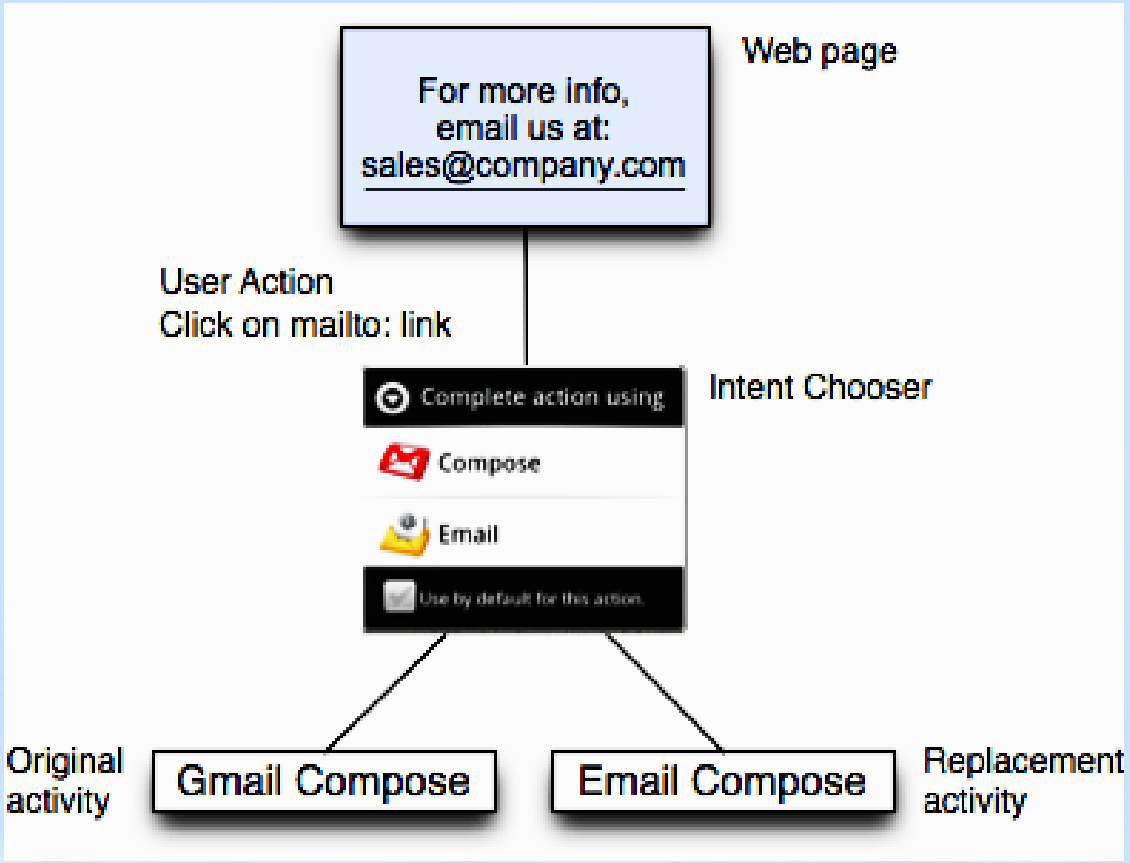
Lansarea din 2 puncte de intrare



Activitati, task-uri, procese

Intents

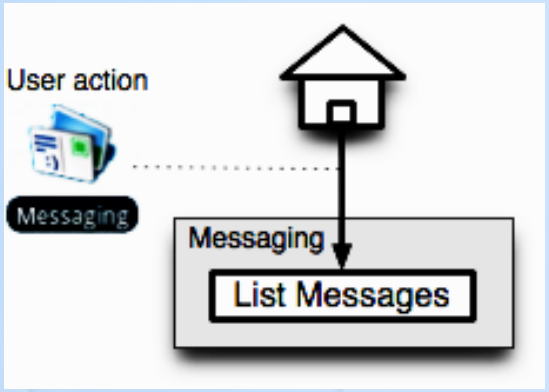
– *This example of touching the **mailto:link** is shown in the following figure. If the device has **two email applications set up**, when a user touches a mailto: email address on a web page, the result is an **Intent** object which displays a dialog box with a **choice** between the **two activities** to compose an email (**Gmail** and **Email**)*

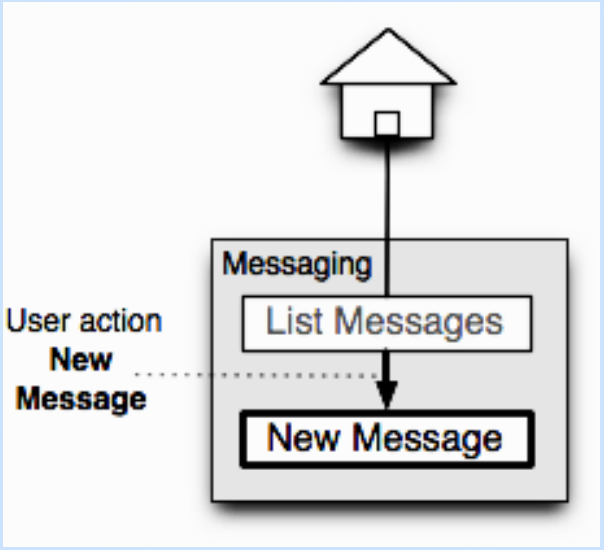


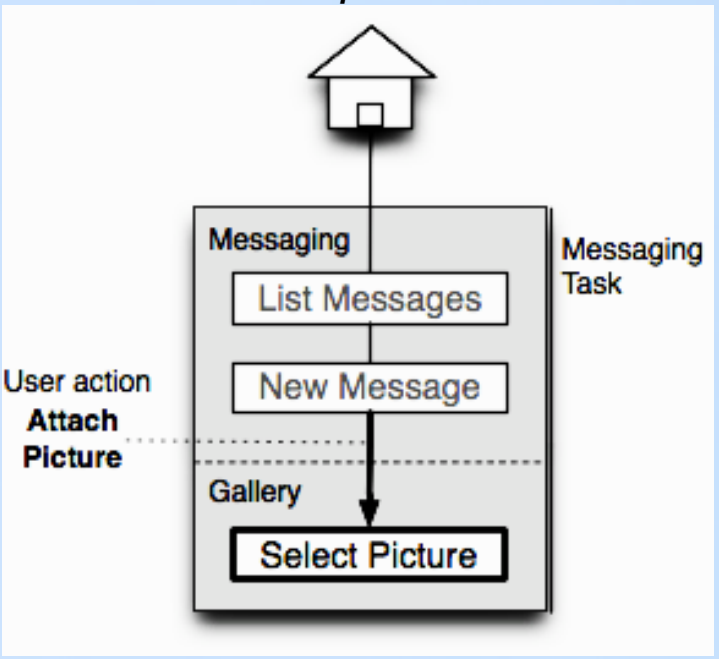
Activitati, task-uri, procese

Switching Between Tasks

*– **Start first task.** You want to send a text message and attach a photo. You would choose: **Home > Messaging > New message > MENU > Attach > Pictures**. This last step launches the picture **gallery** for picking a photo. Notice that picture gallery is an activity in a separate application*





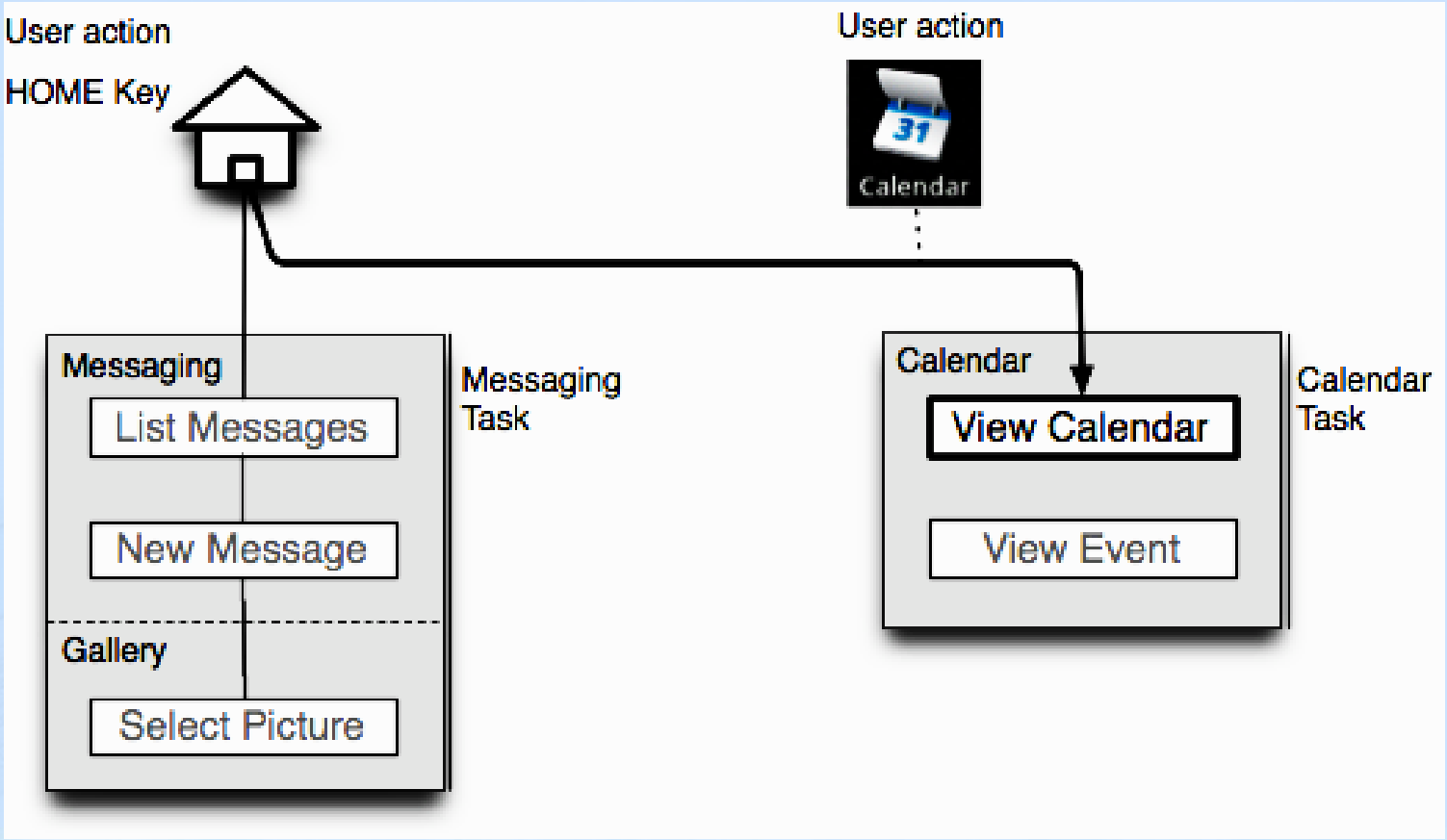


*At this point, before you have picked a picture, you decide to stop and glance at your calendar, which is a separate task. Because the current activity has no button to go directly to the **Calendar**, you need to start from Home*

Activitati, task-uri, procese

Switching Between Tasks

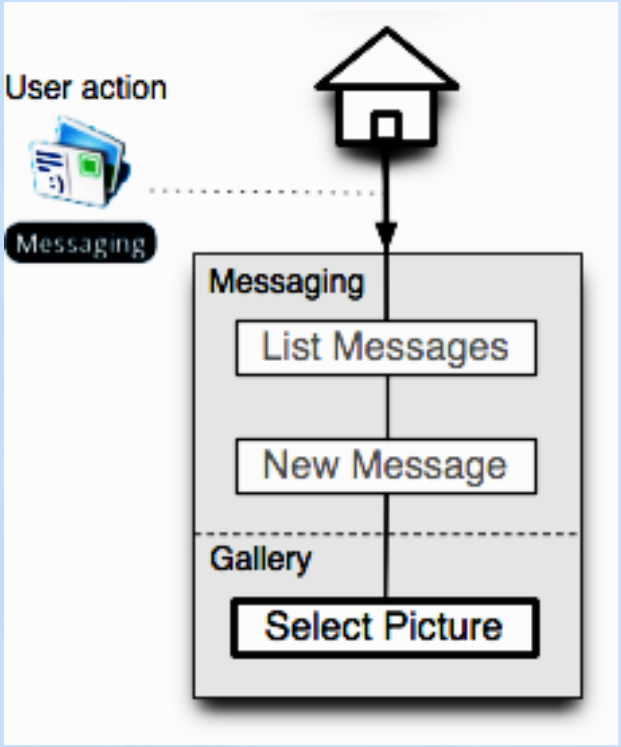
– Start second task. *You choose* ***Home >*** ***Calendar*** *to look at a calendar event.* ***Calendar*** *launches from Home as a new task because the application launcher creates a new task for each application it launches*

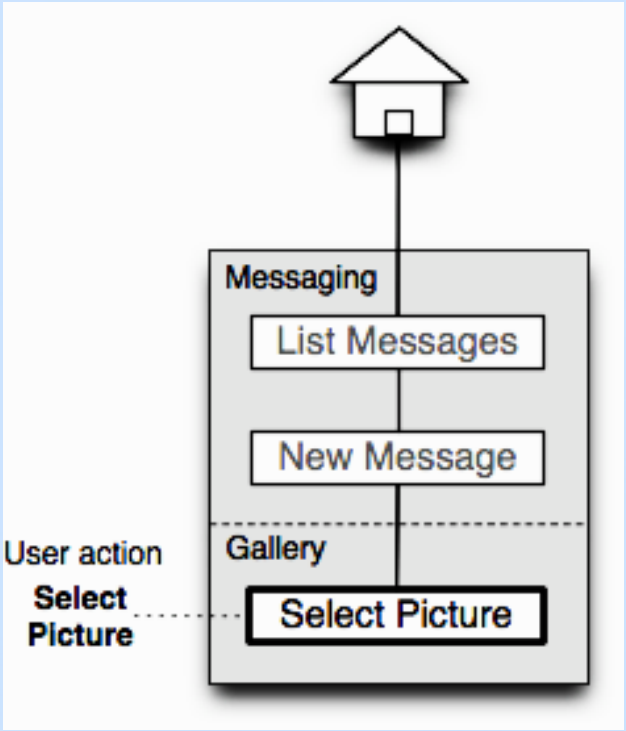


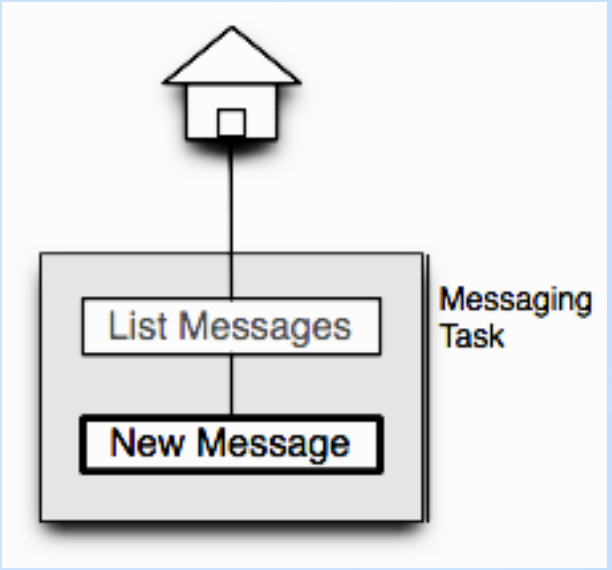
Activitati, task-uri, procese

Switching Between Tasks

– Switch to first task and complete it. *When done looking at the* ***Calendar****, you can return to attaching the picture by starting the root activity again for that task: choose* ***Home > Messaging****, which takes you* ***not to Messaging****, but directly* ***to*** *the* ***Picture gallery****, where you left off. You can then pick a photo, which is added to the message, you send the message and you're done with the first task*



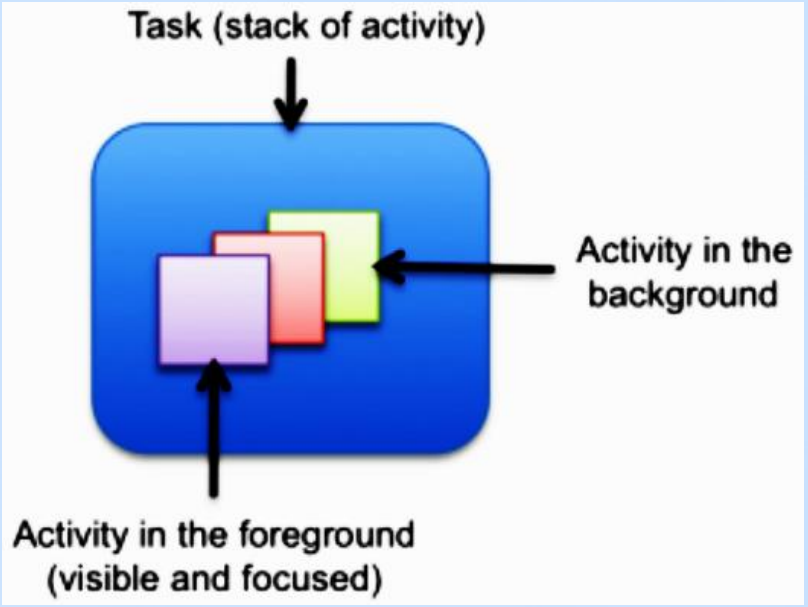




Fazele aplicatiilor Android

Visible** and **focused

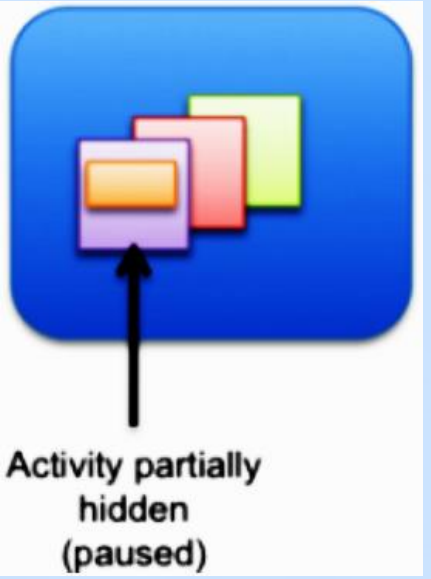
*– activity is **visible** (on the **top** of the **task**), the user can interact with the activity*



Visible** but **without the focus

*– activity is still **visible**, but the **user cannot interact with the activity** (because of a view, like a dialog, that has the focus on the top of the activity)*

*– activity is **paused**, but **maintains the state** (i.e. member values). It may be killed by the system in extreme low memory situation*



Fazele aplicatiilor Android

Hidden

*– activity is completely **hidden** by another activity (full-screen)*

*– activity is **stopped**, it still retains all state and members **BUT** is **often killed** by the system when memory is needed elsewhere*

Finished

*– an activity that is **paused** or **stopped can be killed** by the system*

*– the system can either call the **onDestroy()** method **or** simply **kill** the **process** (without any notification)*

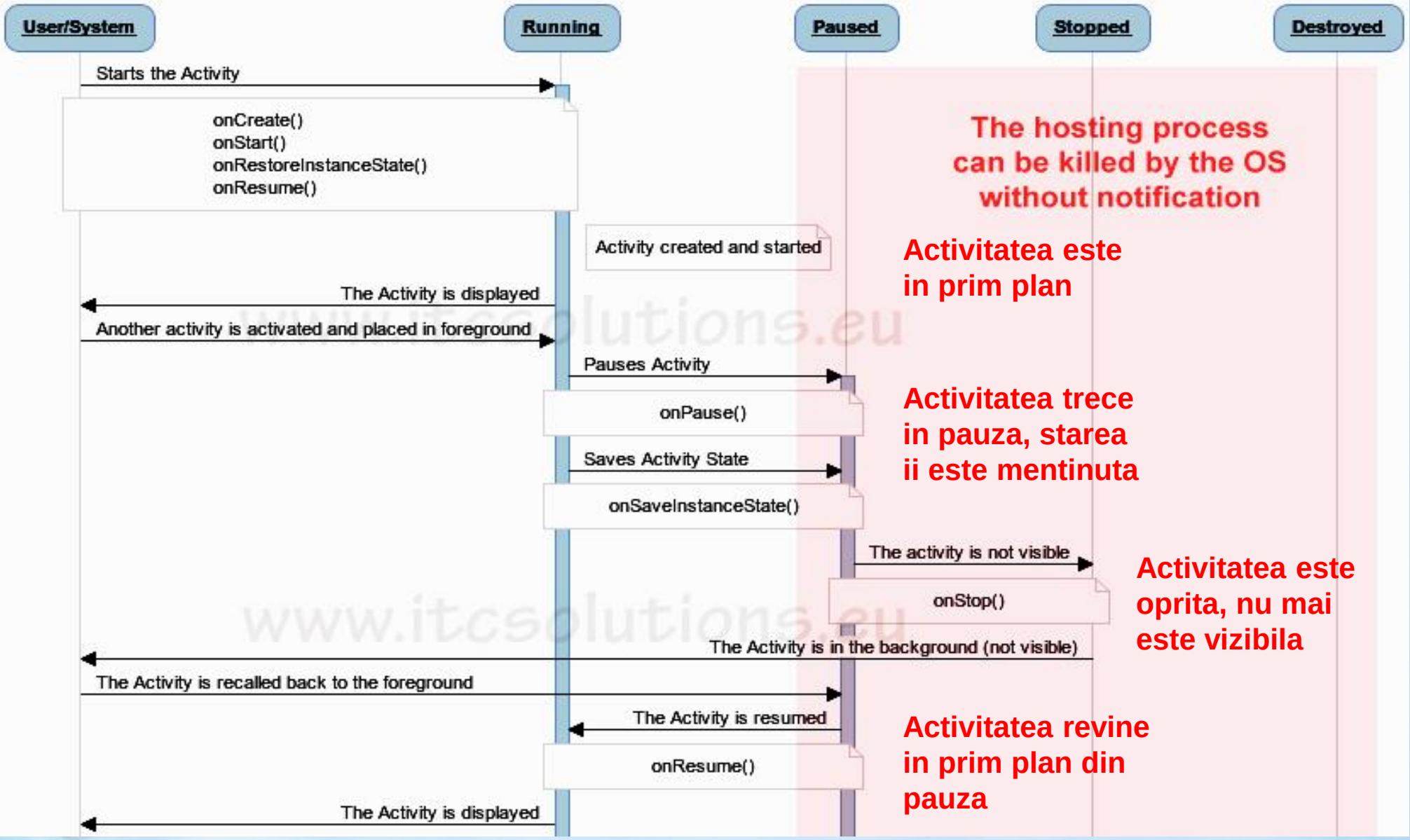
*– the **state** is **lost** in that case*

*– if the activity is **displayed again** to the user, it must be completely **restarted** and **restore its previous state itself** (if at all)*

Programarea pe platforma Android

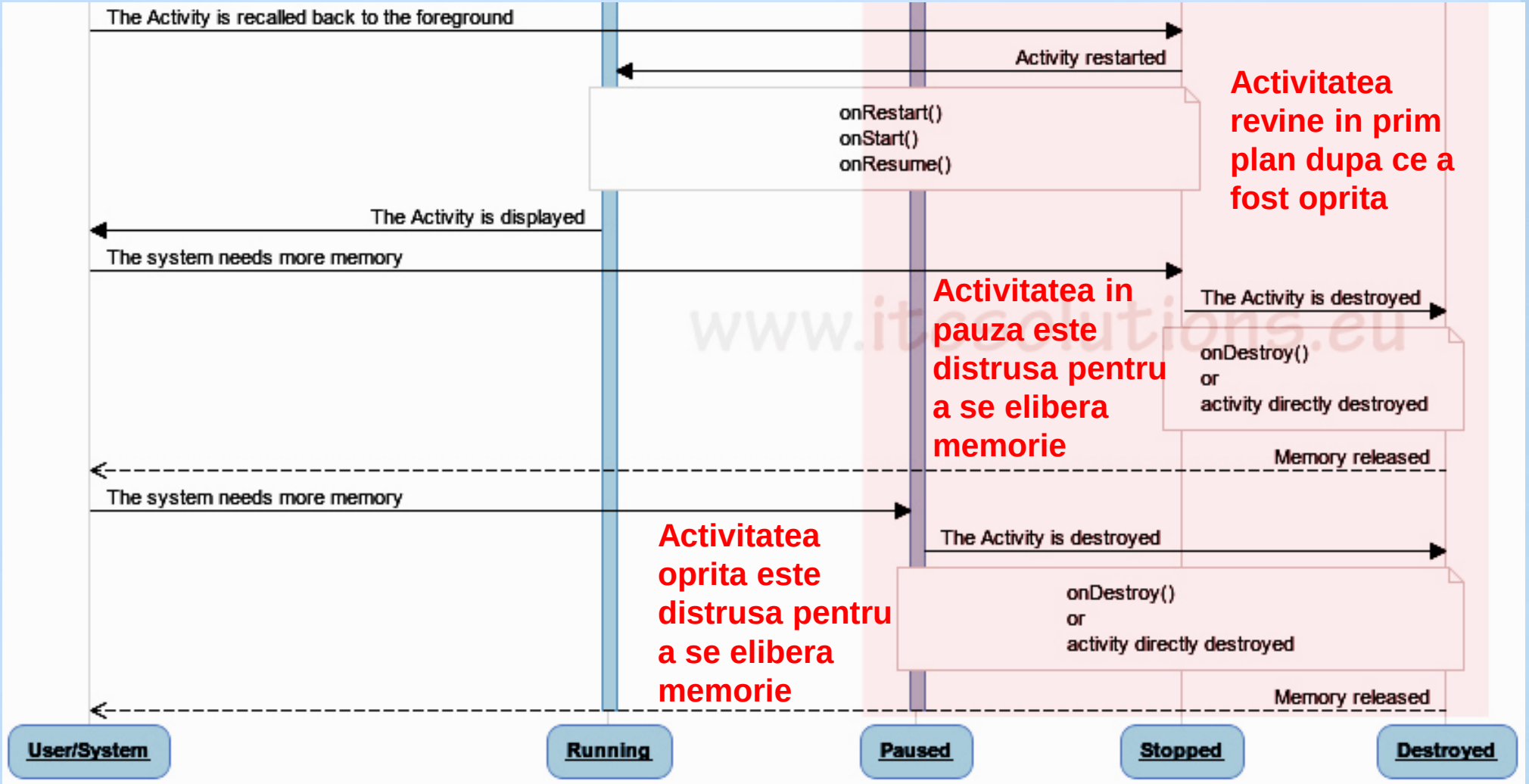
Fazele aplicatiilor Android

Diagrama MSC – schimbul de mesaje



Fazele aplicatiilor Android

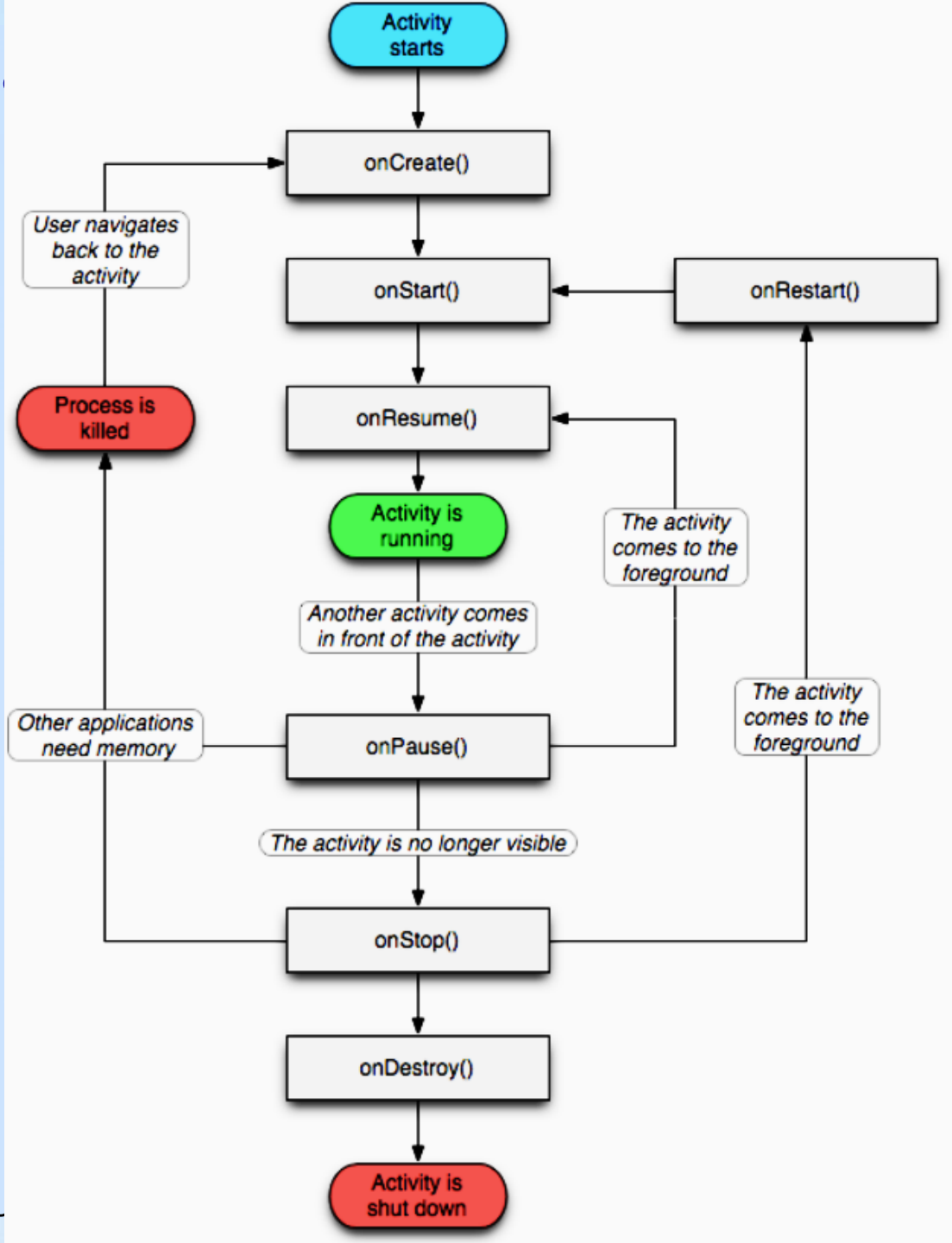
Diagrama MSC – schimbul de mesaje



Program

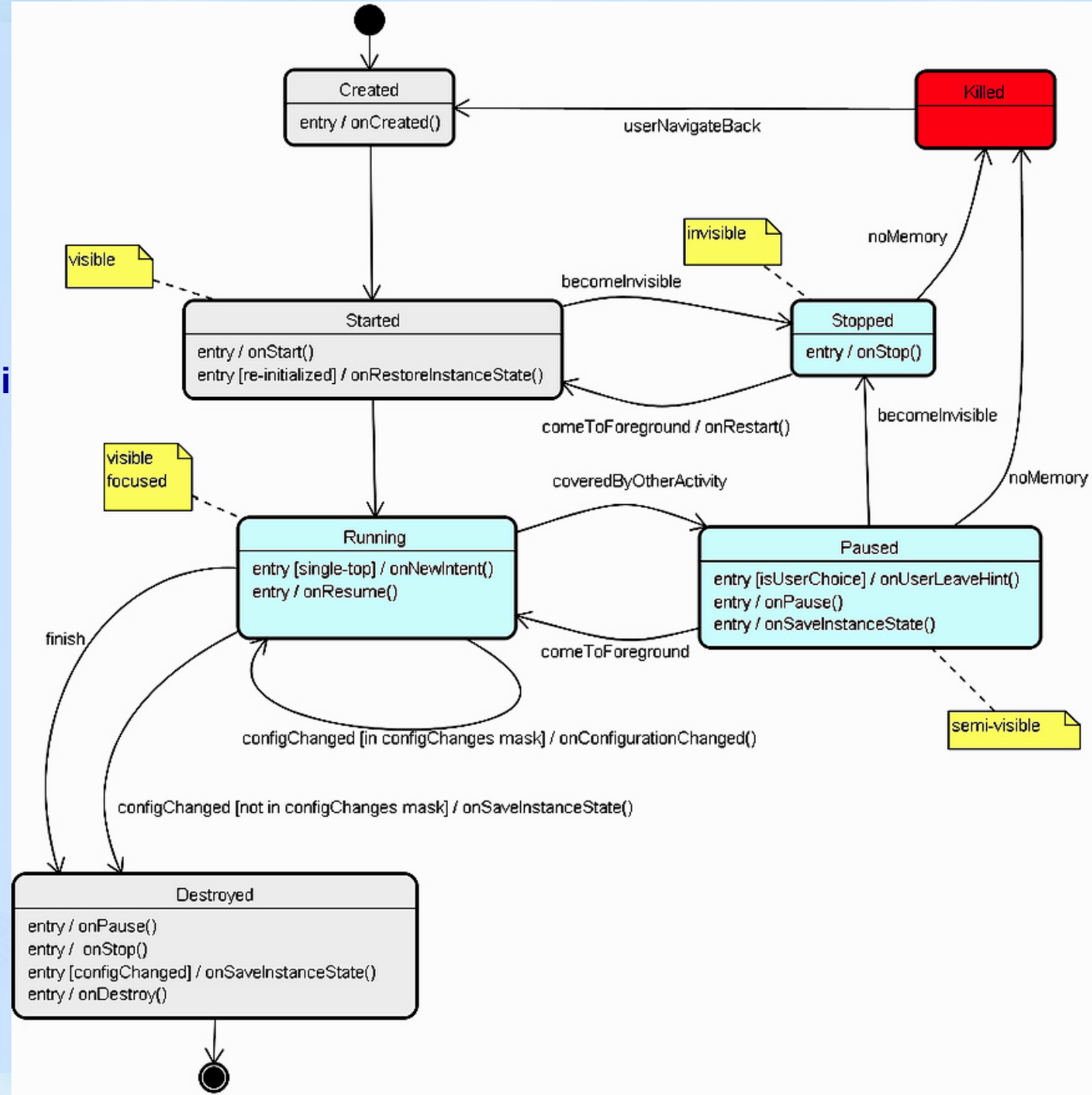
Fazele aplicatiilor Android

Diagrama de activitati



Fazele aplicatiilor Android

Diagrama de stari si tranzitii



Fazele aplicatiilor Android

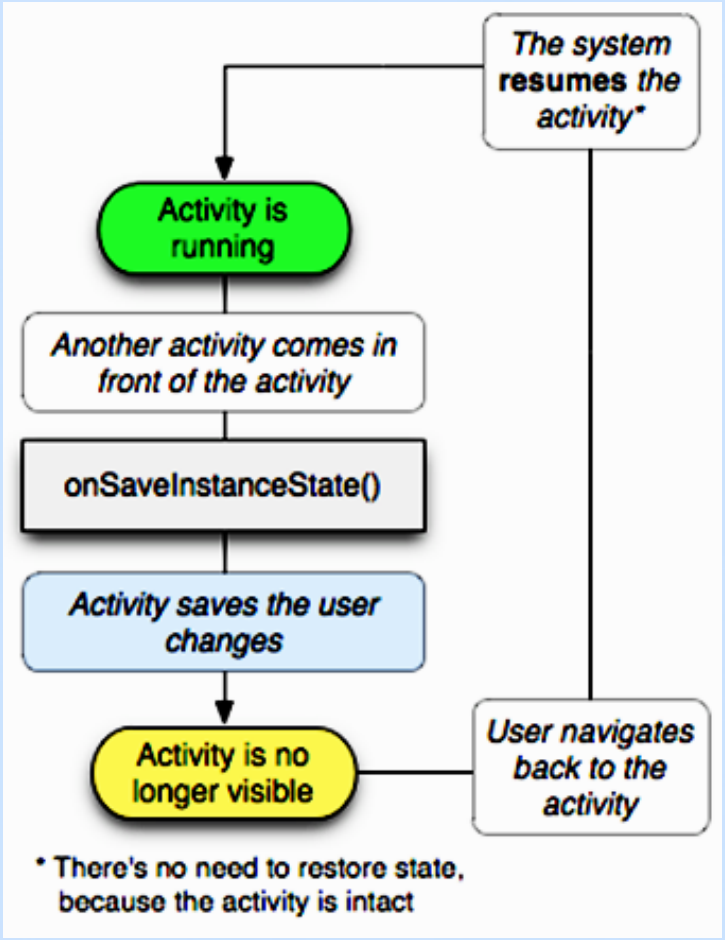
Detalii privind metodele asociate

Method	Description	Killable after?	Next
onCreate()	Called when the activity is first created. This is where you should do all of your normal static set up — create views, bind data to lists, and so on. This method is passed a Bundle object containing the activity's previous state, if that state was captured (see Saving Activity State, later). Always followed by onStart().	No	onStart()
onRestart()	Called after the activity has been stopped, just prior to it being started again. Always followed by onStart()	No	onStart()
onStart()	Called just before the activity becomes visible to the user. Followed by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden.	No	onResume() or onStop()
onResume()	Called just before the activity starts interacting with the user. At this point the activity is at the top of the activity stack, with user input going to it. Always followed by onPause().	No	onPause()
onPause()	Called when the system is about to start resuming another activity. This method is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, and so on. It should do whatever it does very quickly, because the next activity will not be resumed until it returns. Followed either by onResume() if the activity returns back to the front, or by onStop() if it becomes invisible to the user.	**Yes**	onResume() or onStop()
onStop()	Called when the activity is no longer visible to the user. This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it. Followed either by onRestart() if the activity is coming back to interact with the user, or by onDestroy() if this activity is going away.	**Yes**	onRestart() or onDestroy()
onDestroy()	Called before the activity is destroyed. This is the final call that the activity will receive. It could be called either because the activity is finishing (someone called finish() on it), or because the system is temporarily destroying this instance of the activity to save space. You can distinguish between these two scenarios with the isFinishing() method.	**Yes**	*nothing*

Fazele aplicatiilor Android

Cai prin care **o activitate returneaza focusul** catre utilizator cu **starea** sa **intacta**

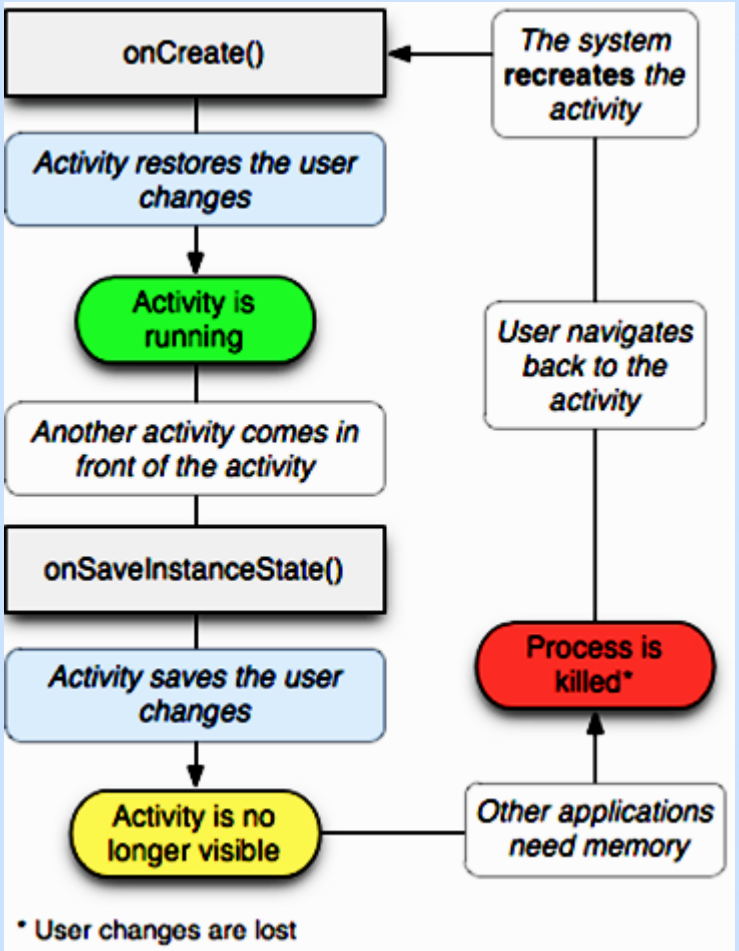
– fie e **oprita** apoi **repornita** (*resumed*)



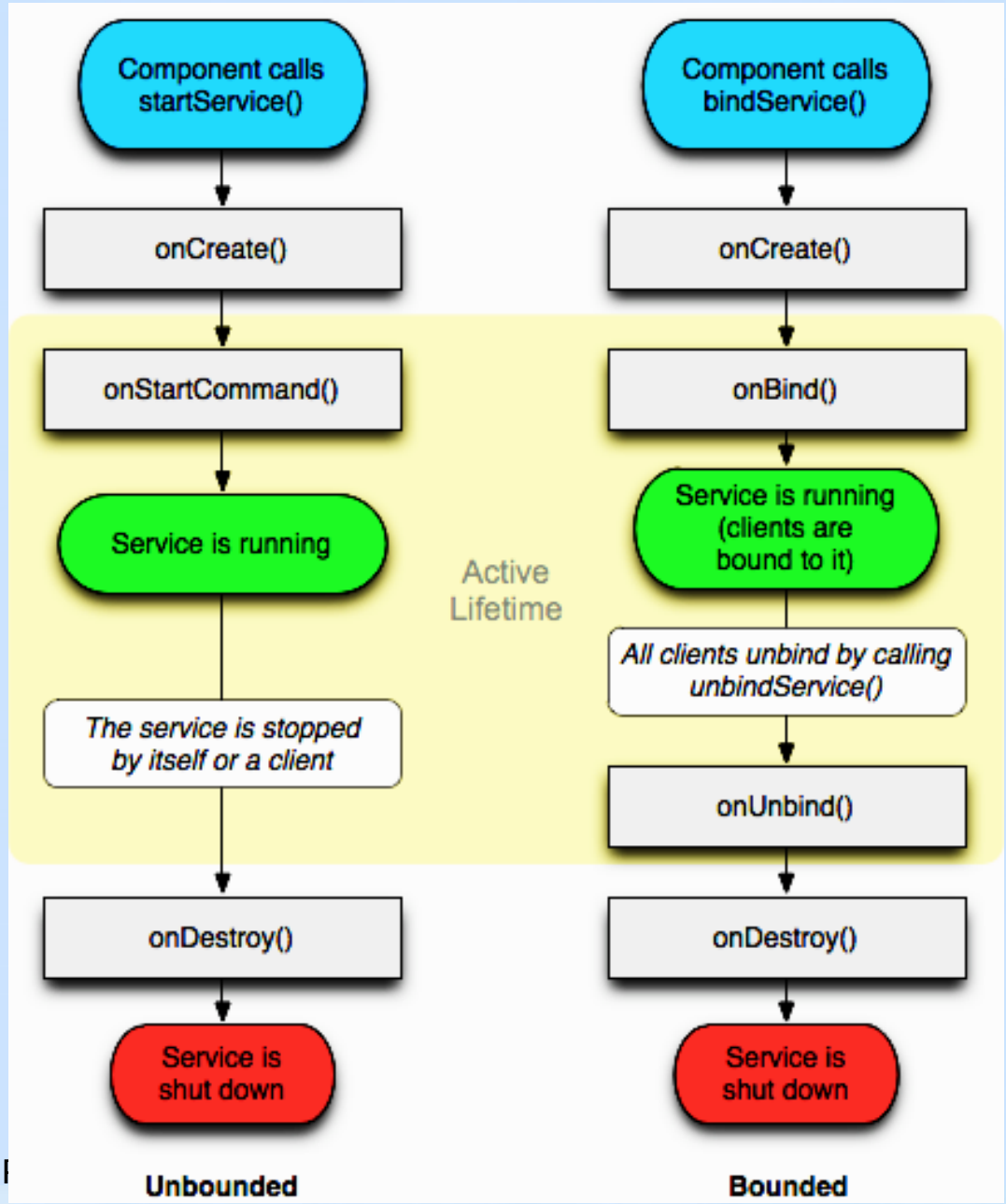
Fazele aplicatiilor Android

Cai prin care **o activitate returneaza focusul** catre utilizator cu **starea** sa **intacta**

– fie e **distrusa** si apoi **recreata**, si trebuie sa **isi reia starea anterioara**

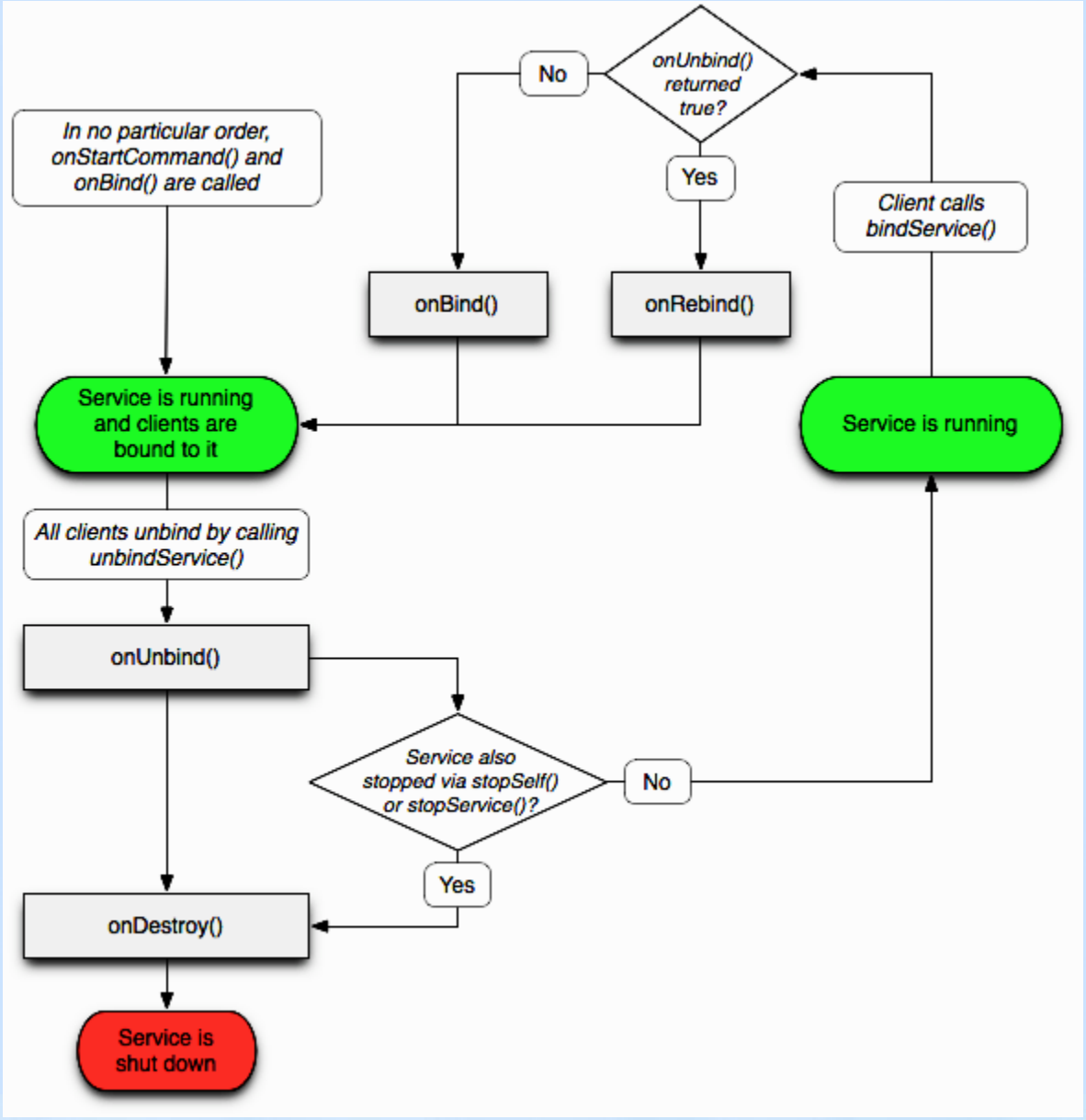


Fazele serviciilor Android



Fazele serviciilor Android

A service** that is **started** and also **allows binding



Exemplu de proiectare a unei aplicatii Android

Teach Yourself Android App Dev in 24 hrs 2010

Presupunem ca vrem sa **proiectam** joc numit ***Chippy's Revenge***, cu **5 ecrane**

Splash – ecran startup, cu logo-ul si versiunea jocului, eventual o melodie

Menu – ecran de selectie a unor optiuni (*play game, view scores, read help*)

Play – ecranul in care se desfasoara jocul

Scores – ecran pentru afisarea celor mai mari scoruri ale jocurilor (incluzandu-le pe cele obtinute de alti jucatori)

Help – ecran de instructiuni cu regulile jocului, modul de joc, scor, sfaturi, etc.

Exemplu de proiectare a unei aplicatii Android

In acest caz trebuie sa **implementam 5 clase activitate** (derivate din **Activity**)

SplashActivity – Activitate implicita la lansare, care afiseaza un *layout*, reda muzica pentru cateva secunde, apoi lanseaza ***MenuActivity***

MenuActivity – Activitate care afiseaza butoane corespondente caracteristicilor aplicatiei. Metodele *handler* **onClick()** pentru fiecare buton **lanseaza activitatile asociate**

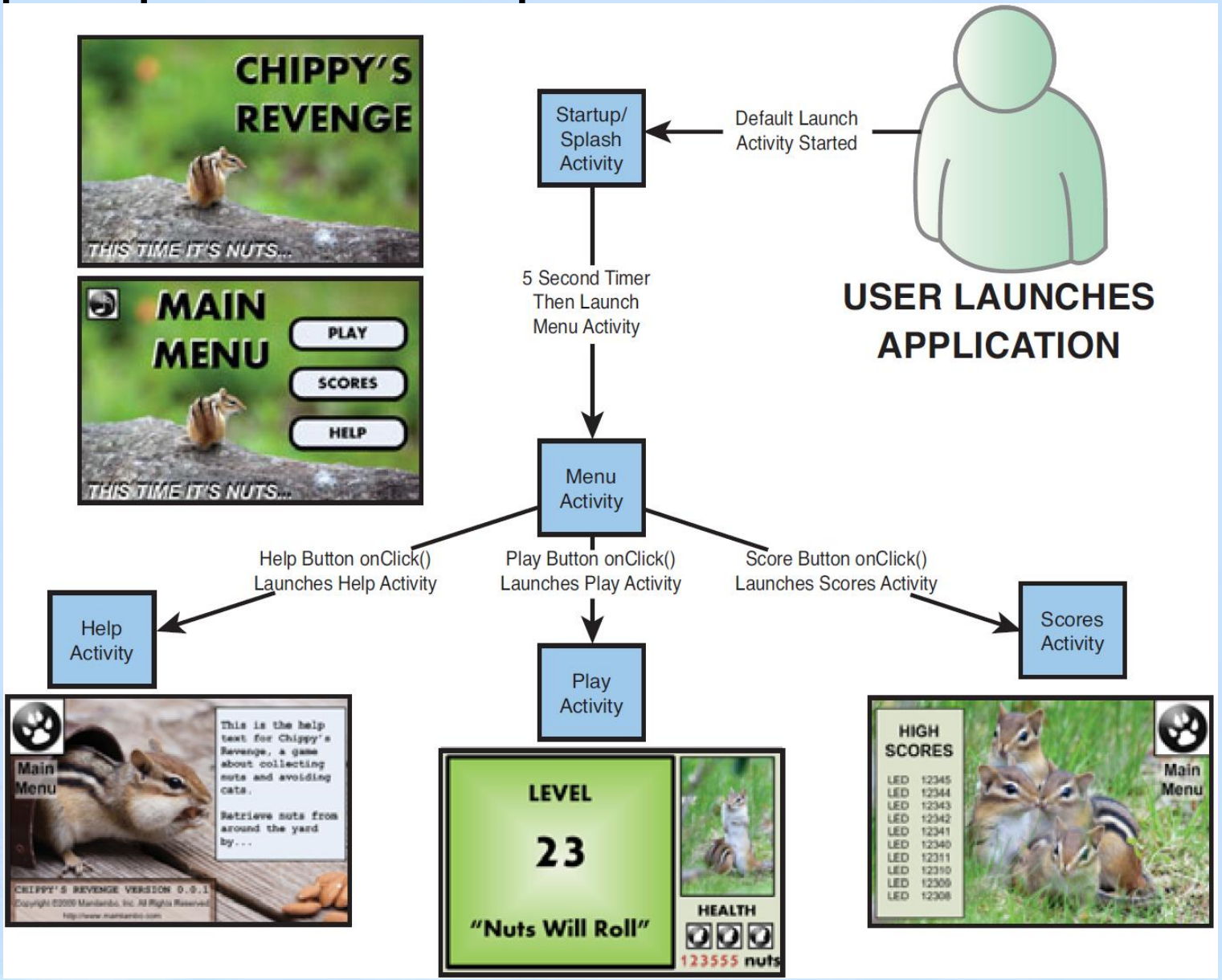
PlayActivity – Activitate care implementeaza aplicatia reala, desenand pe ecran, raspunzand la interactiunea cu utilizatorul, pastrand scorul, oferind in general suport pentru tot ce e necesar in desfasurarea jocului

ScoresActivity – Activitate simpla ca si ***SplashActivity***, care afiseaza informatiile privind scorul intr-un ***TextView***

HelpActivity – Activitate aproape identica cu ***ScoresActivity***, care afiseaza textul *help*, intr-un ***TextView*** eventual cu *scroll*

Programarea pe platforma Android

Exemplu de proiectare a unei aplicatii Android



The general process for a typical build

– The ***Android Asset Packaging Tool*** (**aapt**) takes your application resource files, such as the **AndroidManifest.xml** file and the XML files for your Activities, and compiles them. An **R.java** is also produced so you can reference your resources from your Java code.

– The **aidl** tool converts any **.aidl** interfaces that you have into Java interfaces.

– All of your Java code, including the **R.java** and **.aidl** files, are compiled by the Java **compiler** and **.class** files are output.

– The **dex** tool converts the **.class** files to ***Dalvik byte code***. Any 3rd party libraries and **.class** files that you have included in your project are also converted into **.dex** files so that they can be packaged into the final **.apk** file.

– All non-compiled resources (such as images), compiled resources, and the **.dex** files are sent to the **apkbuilder** tool to be packaged into an **.apk** file.

– Once the **.apk** is built, it must be signed (**Jarsigner**) with either a debug or release key before it can be installed to a device.

– Finally, if the application is being signed in release mode, you must align the **.apk** with the **zipalign** tool. Aligning the final **.apk** decreases memory usage when the application is running on a device.

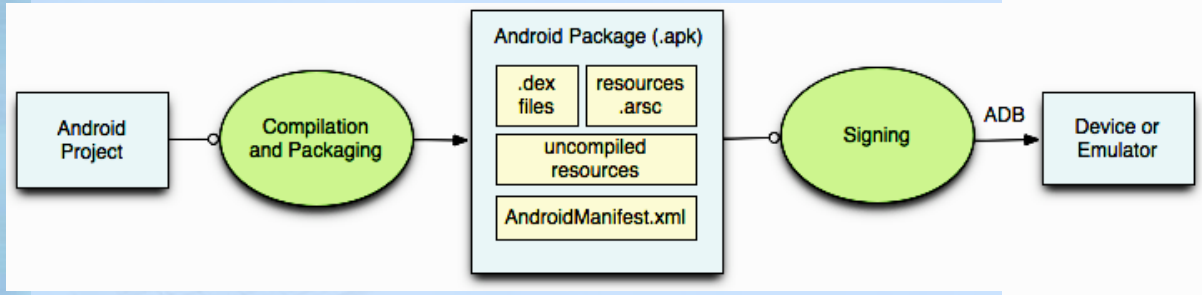
http://developer.android.com/guide/developing/building/index.html

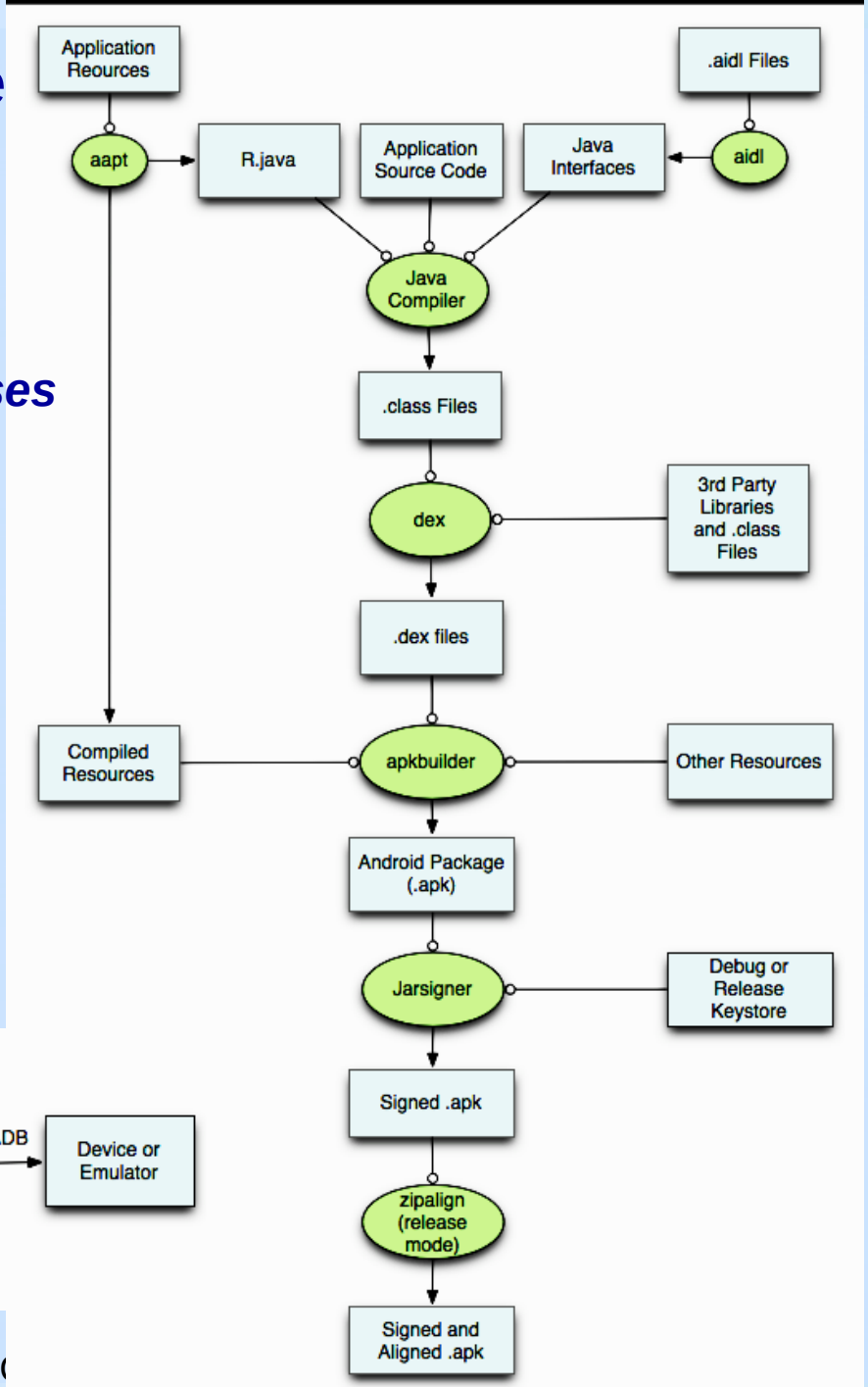
Programarea pe

The general process for a typical build

*different **tools** and **processes** that are involved in a **build***

***components** involved in **building** and **running** an application*





Microsoft® .NET

Tratarea evenimentelor – crearea inline a "ascultatoarelor"

```
public class LoginExample extends Activity {

  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Set Click Listeners

    btnLogin.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        // Check Login
        String username = etUsername.getText().toString();
        String password = etPassword.getText().toString();
        if(username.equals("guest") && password.equals("guest")){
          lblResult.setText("Login successful.");
        } else {
          lblResult.setText("Login failed. Username and/or password doesn't match.");
        }
      }
    });

    btnCancel.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        // Close the application
        finish();
      }
    });
  }
```

Ascultator inline

Ascultator inline

Tratarea evenimentelor – crearea inline a “ascultatoarelor”

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class EventListenerForButtonActivity extends Activity {

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //1-Capture your button from View
        Button clickMe =(Button)findViewById(R.id.button);

        //2-Register your OnClickListener with your implementation
        clickMe.setOnClickListener( new OnClickListener() {

          public void onClick(View v) {
            //3- do some thing use full in here
            TextView clickMeText = (TextView)findViewById(R.id.textView);
            clickMeText.setText("You Clicked Me");
          }
        });
    }
}
```

Ascultator inline

http://sanathnandasiri.blogspot.com/2011/07/add-event-listener-for-button.html

Tratarea evenimentelor – crearea inline a “ascultatoarelor”

Advantaje

- **cod redus** si **compact**
- **usor** de **implementat**

Dezavantaje

- cod **inflexibil**
- **nu** poate fi **reutilizat**
- poate fi **destul de greu de mentinut** (adaptat)

Utilizare

- pentru **metode scurte** care sunt **utilizate o singura data**
  - cum sunt cele de tratare a unor **butoane care produc inchiderea / afisarea unei ferestre**

Tratarea evenimentelor – transformarea activitatii in “ascultator”

Activitatea e ascultator

```
public class LoginExampleImplements extends Activity implements OnClickListener {

  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Set Click Listeners
    btnLogin.setOnClickListener(this);
    btnCancel.setOnClickListener(this);
  }

  public void onClick(View v) {

    if(v==btnLogin) {
      // Check Login
      String username = etUsername.getText().toString();
      String password = etPassword.getText().toString();
      if(username.equals("guest") && password.equals("guest")){
        lblResult.setText("Login successful.");
      } else {
        lblResult.setText("Login failed. Username and/or password doesn't match.");
      }

    } else if(v==btnCancel) {
      // Close the application
      finish();
    }
  }
}
```

Tratarea evenimentelor – transformarea activitatii in “ascultator”

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

//3- you should implement Listener interface in your Activity class
public class EventListenerForButtonActivity extends Activity implements OnClickListener{

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //1-Capture your button from View
        Button clickMe =(Button)findViewById(R.id.button);

        //2-Register your OnClickListener with your Activity class itself
        clickMe.setOnClickListener(this);
    }

    //4-Override unimplemented class
    public void onClick(View v) {
          //5- do some thing use full in here
          TextView clickMeText = (TextView)findViewById(R.id.textView);
          clickMeText.setText("You Clicked Me");
    }
}
```

Activitatea e ascultator

Tratarea evenimentelor – transformarea activitatii in “ascultator”

Advantaje

– metodele si “**ascultatorii**” pot fi **reutilizati in** mai multe ***widget*-uri diferite**

– codul mai multor “ascultatori” este plasat in **aceeasi sectiune** de cod

– se poate crea **o metoda pentru “ascultatori” similari**

Dezavantaje

– poate contine **mult cod nenecesar si dispersat**, **daca actiunile** executate sunt **foarte diferite** si **trebuie adaugate blocuri if / elseif / else**, facand codul greu de citit

– poate exista **o singura implementare** a unui astfel de “**ascultator**” **pe clasa**

Utilizare

– pentru cazul in care sunt **multiple widget-uri / elemente** care **utilizeaza “ascultatori” similari** sau **acelasi** “**ascultator**”

– cum ar fi un calculator cu **zeci de butoane**, ca in **exemplul urmator**:

Programarea pe platforma Android

Tratarea evenimentelor – transformarea activitatii in “ascultator”

```
public class CalculatorExample extends Activity implements OnClickListener {

    public void onClick(View v) {

        if(v==btnCalculate) {

            // Parse and calculate formula
            String formula = etFormula.getText().toString();
            Double result = performCalculation(formula);

            // Update the result TextView
            tvResult.setText(Strint.valueOf(result));

            // End it as we don't need or want to update the Formula field
            return;
        }

        // Get the button
        Button button = (Button)v;

        // Get the String/Button description
        String strToAppend = button.getText().toString();

        // Update Formula
        etFormula.append(strToAppend);
    }
}
```


Tratarea evenimentelor – utilizarea unor obiecte “ascultator”

```
public class LoginExampleVariableImplementation extends Activity {

  OnClickListener myClickListener = new OnClickListener() {

    public void onClick(View v) {
      if(v==btnLogin) {
        // Check Login
        String username = etUsername.getText().toString();
        String password = etPassword.getText().toString();
        if(username.equals("guest") && password.equals("guest")){
          lblResult.setText("Login successful.");
        } else {
          lblResult.setText("Login failed. Username and/or password doesn't match.");
        }
      } else if(v==btnCancel) {
         // Close the application
         finish();
      }
    }
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Set Click Listeners
    btnLogin.setOnClickListener(myClickListener);
    btnCancel.setOnClickListener(myClickListener);
  }
}
```

Obiect ascultator

Tratarea evenimentelor – utilizarea unor obiecte “ascultator”

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class EventListenerForButtonActivity extends Activity {

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //1-Capture your button from View
        Button clickMe =(Button)findViewById(R.id.button);

        //3-Register your OnClickListener with your implementation
        clickMe.setOnClickListener(clickMeListener);
    }

    //2-Implementation of your button's OnClickListener as local variable
    private OnClickListener clickMeListener = new OnClickListener() {

       public void onClick(View v) {
          //4- do some thing use full in here
          TextView clickMeText = (TextView)findViewById(R.id.textView);
          clickMeText.setText("You Clicked Me");
       }
    };
}
```

Obiect ascultator

Tratarea evenimentelor – utilizarea unor obiecte “ascultator”

Advantaje

- **– cod (obiect) reutilizabil**
- **– codul mai multor “ascultatori”** este plasat in **aceeasi sectiune** de cod
- **–** se pot crea **mai multi “ascultatori” de acelasi fel**

Dezavantaje

- – **foarte multi ascultatori** pot duce la un **cod greu de citit**

Utilizare

- – pentru cazul in care sunt necesari **“ascultatori” diferiti pentru aceeasi actiune**
  - – cum ar fi doua `OnClickListener` care executa 2 sarcini complet diferite
- – pentru cazul in care se implementeaza **proprii “ascultatori” pentru propriile widget-uri**

Tratarea evenimentelor – utilizarea unor obiecte “ascultator”

```
public class MyWidget extends View {

  OnClickListener myClickListener = null;

  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
  }

  public void setOnClickListener(OnClickListener listener) {
    myClickListener = listener;
  }

  public void onClick(View v) {
    // Check if Listener was set and call the onClick Method
    if(myClickListener!=null)
      myClickListener.onClick(v);
  }

  private void handleEventsMethod() {
    ...
    // handle clicks
    onClick(this);
  }
}
```

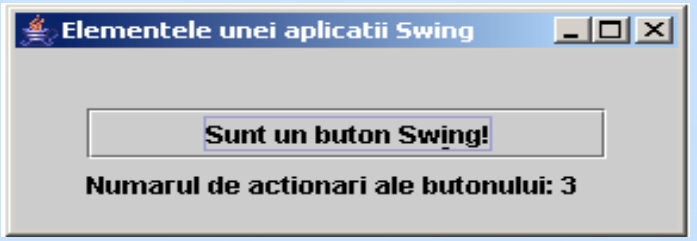
http://tseng-blog.nge-web.net/blog/2009/02/14/implementing-listeners-in-your-android-java-application/
http://tseng-blog.nge-web.net/blog/2009/02/17/how-implement-your-own-listener-android-java/

Tratarea evenimentelor

Comparatie cu tratarea evenimentelor in Java Swing (Java SE)

Tratarea evenimentelor in Java Swing

Elementele unei aplicatii grafice Swing



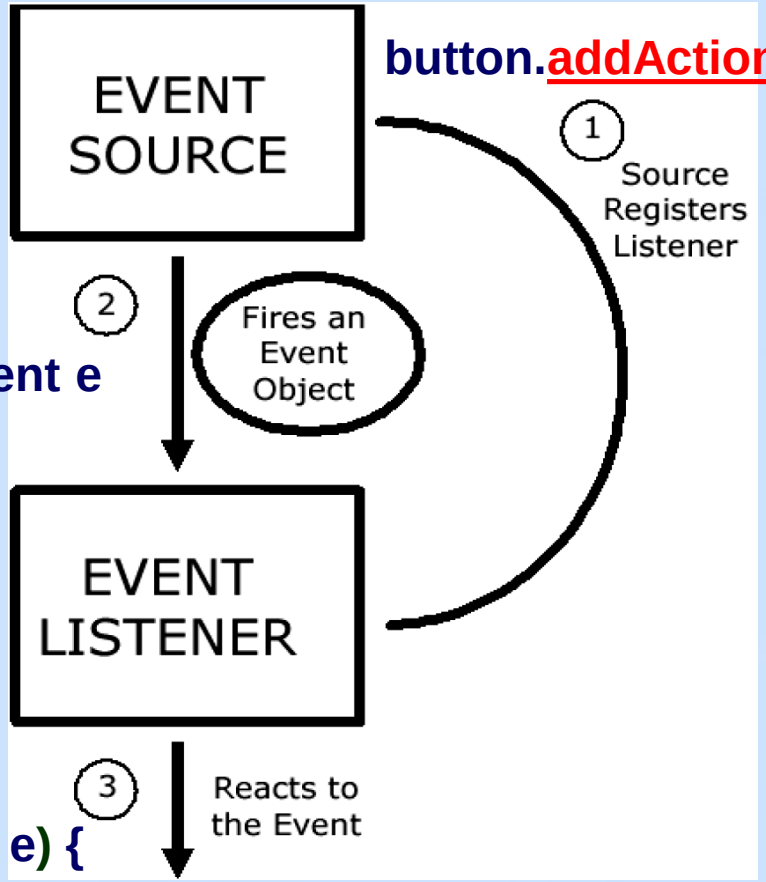

```
// 8. Crearea codului pentru tratarea evenimentelor
// (interactivitatii)

// 8.1. Atasarea unui obiect al unei clase anonime care
//      implementeaza interfata ActionListener, a carei metoda
//      actionPerformed() trateaza actionarea butonului

buton.addActionListener( new ActionListener() {
   int numActionari = 0;
   public void actionPerformed(ActionEvent e) {
      numActionari++;
      eticheta.setText(textEticheta + numActionari);
   }
});
```

Tratarea evenimentelor in Java Swing

Modelul folosit in tratarea evenimentelor (crearea interactivitatii) in interfetele grafice Swing

JButton button

button.addActionListener(this)



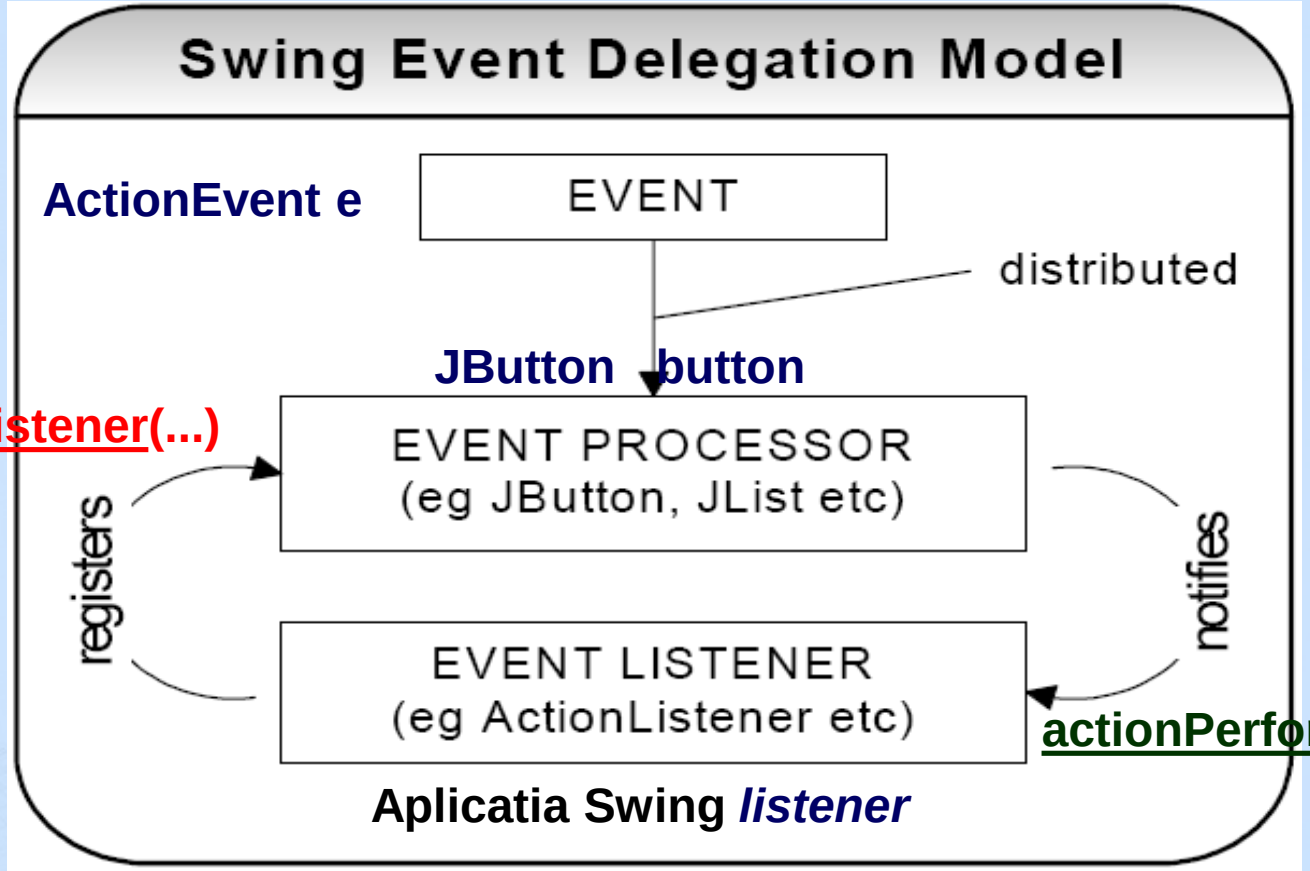
ActionEvent e

```
// Aplicatia Swing listener
public class MyClass
implements ActionListener {

actionPerformed(ActionEvent e) {
    // Codul prin care aplicatia reactioneaza la apasarea butonului
```

Tratarea evenimentelor in Java Swing

Modelul folosit in tratarea evenimentelor (crearea interactivitatii)

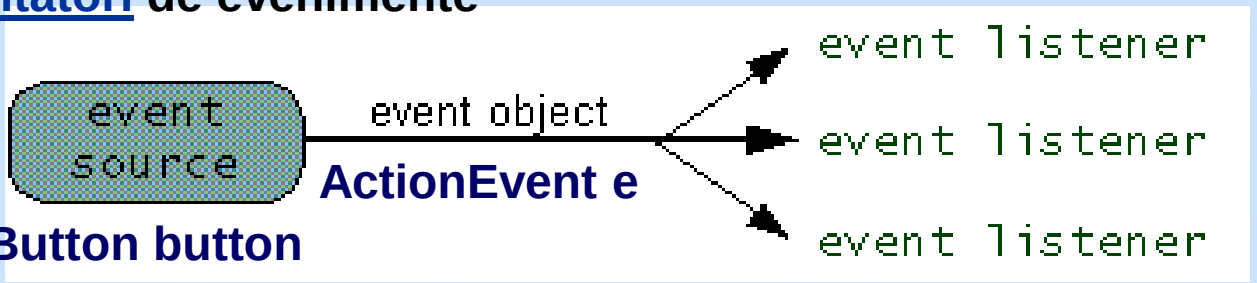
- modelul delegarii evenimentelor Swing



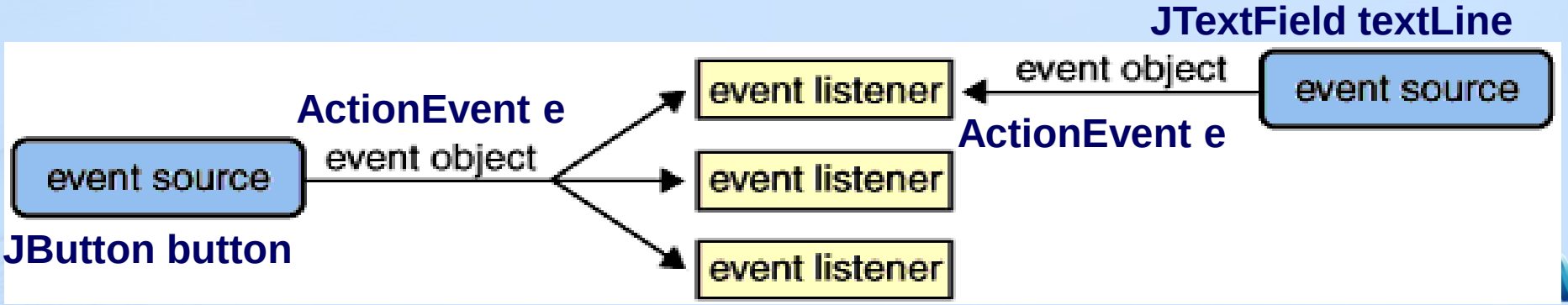
Tratarea evenimentelor in Java Swing

Modelul folosit in tratarea evenimentelor (crearea interactivitatii)

- o **sursa de evenimente** poate **trimite obiecte eveniment catre mai multi ascultatori de evenimente**



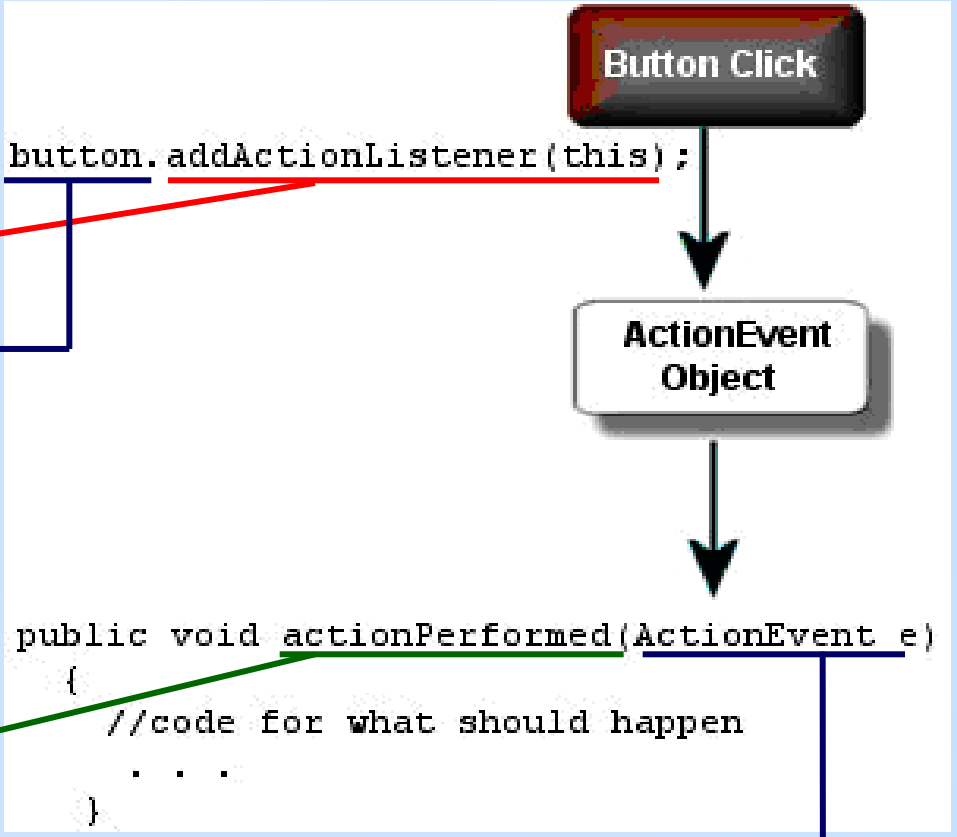
- in plus, **mai multe surse de evenimente** pot **trimite obiecte eveniment catre acelasi ascultator de evenimente**



Tratarea evenimentelor in Java Swing

Ascultatorul de evenimente (`ActionListener`) trebuie:

- **sa se INREGISTREZE (`addActionListener(this)`) la sursa de evenimente (`button`)**
- **si sa ofere cel putin o metoda prin care**
  - **sursa de evenimente sa il poata NOTIFICA pe ascultatorul evenimentului (`actionPerformed(...)`)**
  - **pasandu-i o referinta catre obiectul eveniment (`ActionEvent e`)**

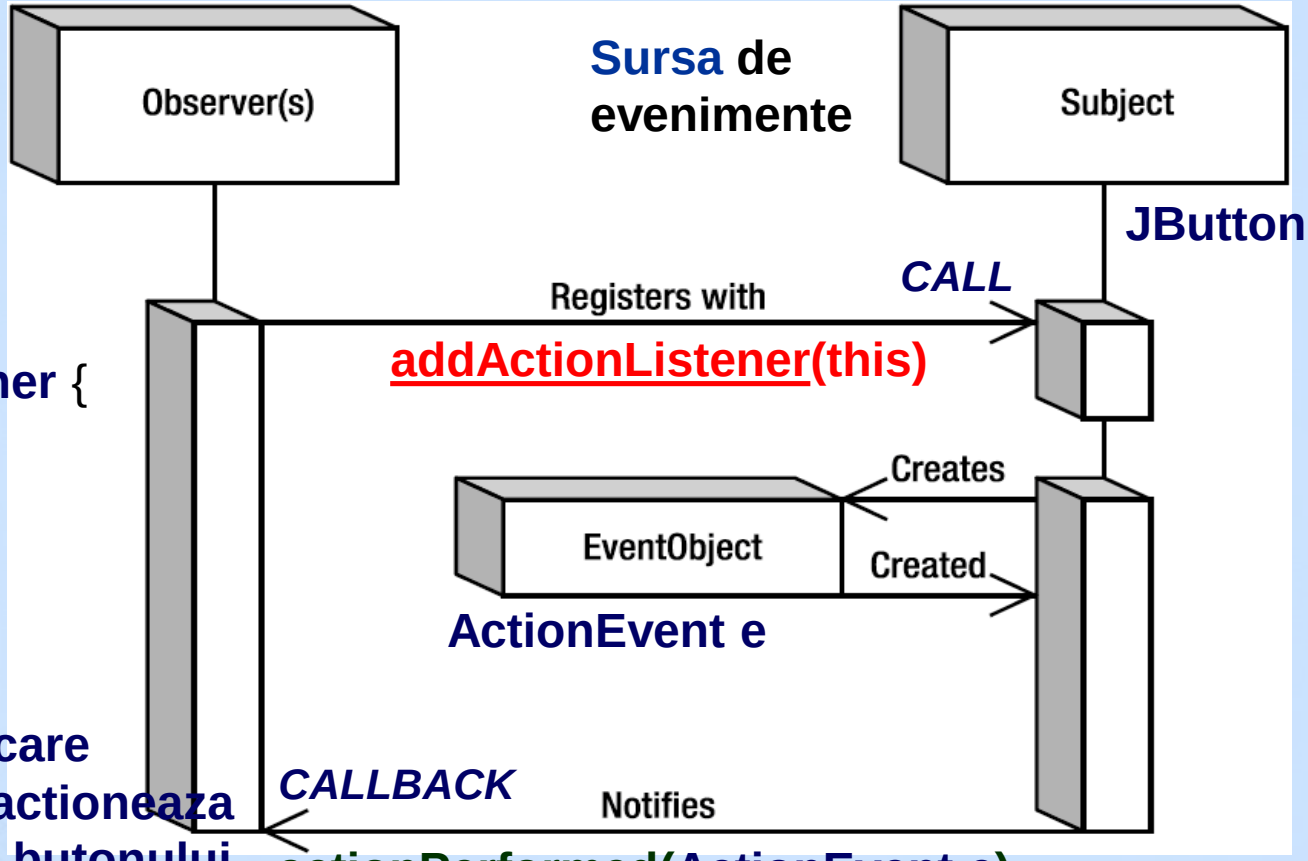


Tratarea evenimentelor in Java Swing

Acesta este un exemplu de pattern de proiectare *Observer* (cunoscut si ca ***Publisher-Subscriber*****, REGISTER–NOTIFY, *callback*****, etc.):**

Aplicatia Swing *listener*

public class **MyClass implements ActionListener** {



Codul prin care aplicatia reactioneaza la apasarea butonului