

Laborator 01. Introducere în Programarea Android

Sistemul de Operare Android

Android - Prezentare Generală

Android este un SO mobil bazat pe o versiune modificată de Linux (pentru gestiunea componentelor hardware, a proceselor și a memoriei) și biblioteci Java (pentru telefonie (audio/video), conectivitate, grafică, programarea interfețelor cu utilizatorul). Este un produs open-source (putând fi dezvoltat de producătorii de dispozitive mobile cu extensii proprietare pentru a-și particulariza platforma), dezvoltat în prezent de compania Google, conceput pe ideea transformării dispozitivelor mobile în adevărate mașini de calcul. Google încearcă totuși să realizeze tranzitia de la AOSP (Android Open Source Project) către GMS (Google Mobile Services), peste care sunt construite cele mai multe aplicații, în încercarea de a-și apăra acest proiect în fața concurenței. În acest sens, a fost dezvoltat **proiectul Google One**, prin care este oferit un set de specificații (écran de 4.5 inchi - 845×480 pixeli, procesor quad-core, memorie 1GB RAM, spațiu de stocare 4GB, suport pentru dual sim) pe care producătorii de dispozitive mai ieftine trebuie să le respecte astfel încât acestea să fie compatibile cu un sistem Android, fără a întâmpina probleme de performanță. Aceasta include toate aplicațiile și serviciile Google, la care se pot adăuga și altele, furnizate de producător sau operatorul de telefonie mobilă. În acest fel, se asigură calitatea (păstrând renumele Android) și controlul asupra veniturilor. Comunitatea Android este în creștere, mai multe companii renunțând la propriul sistem de operare în favoarea acestuia, pentru a putea face față fenomenului iPhone. În condițiile în care pe piață dispozitivelor mobile aplicațiile sunt cele care aduc avantajul competițional, beneficiul Android este reprezentat de abordarea unitară pentru dezvoltarea aplicațiilor. Cu alte cuvinte, o aplicație dezvoltată conform API-ului Android va putea rula pe mai multe dispozitive mobile pe care este instalat sistemul de operare respectiv.

Versiuni Android

Versiune Android	Nivel API	Data Lansării	Nume de Cod	Cota de Piață
7.1 - 7.1.1	25	04.10.2016	Nougat	0.3%
7.0	24	22.08.2016	Nougat	0.9%
6.0 - 6.0.1	23	05.10.2015	Marshmallow	30.7%
5.1 - 5.1.1	22	09.03.2015	Lollipop	23.1%
5.0 - 5.0.2	21	12.11.2014	Lollipop	9.8%
4.4W - 4.4W.2	20	25.06.2014	KitKat with wearable extensions	
4.4 - 4.4.4	19	31.10.2013	KitKat	21.9%
4.3	18	24.07.2013	Jelly Bean	1.6%
4.2.x	17	13.11.2012	Jelly Bean	5.7%
4.1.x	16	09.07.2012	Jelly Bean	4.0%
4.0.3 - 4.0.4	15	16.12.2011	Ice Cream Sandwich	1.0%
4.0 - 4.0.2	14	19.10.2011	Ice Cream Sandwich	
3.2	13	15.07.2011	Honeycomb	
3.1	12	10.05.2011	Honeycomb	

3.0	11	22.02.2011	Honeycomb	
2.3.3 - 2.3.7	10	09.02.2011	Gingerbread	1.0%
2.3 - 2.3.2	9	06.12.2010	Gingerbread	
2.2 - 2.2.3	8	20.05.2010	Froyo	
2.1	7	12.01.2010	Eclair	
2.0.1	6	03.12.2009	Eclair	
2.0	5	26.10.2009	Eclair	
1.6	4	15.09.2009	Donut	
1.5	3	30.04.2009	Cupcake	
1.1	2	09.02.2009		
1.0	1	23.09.2008		

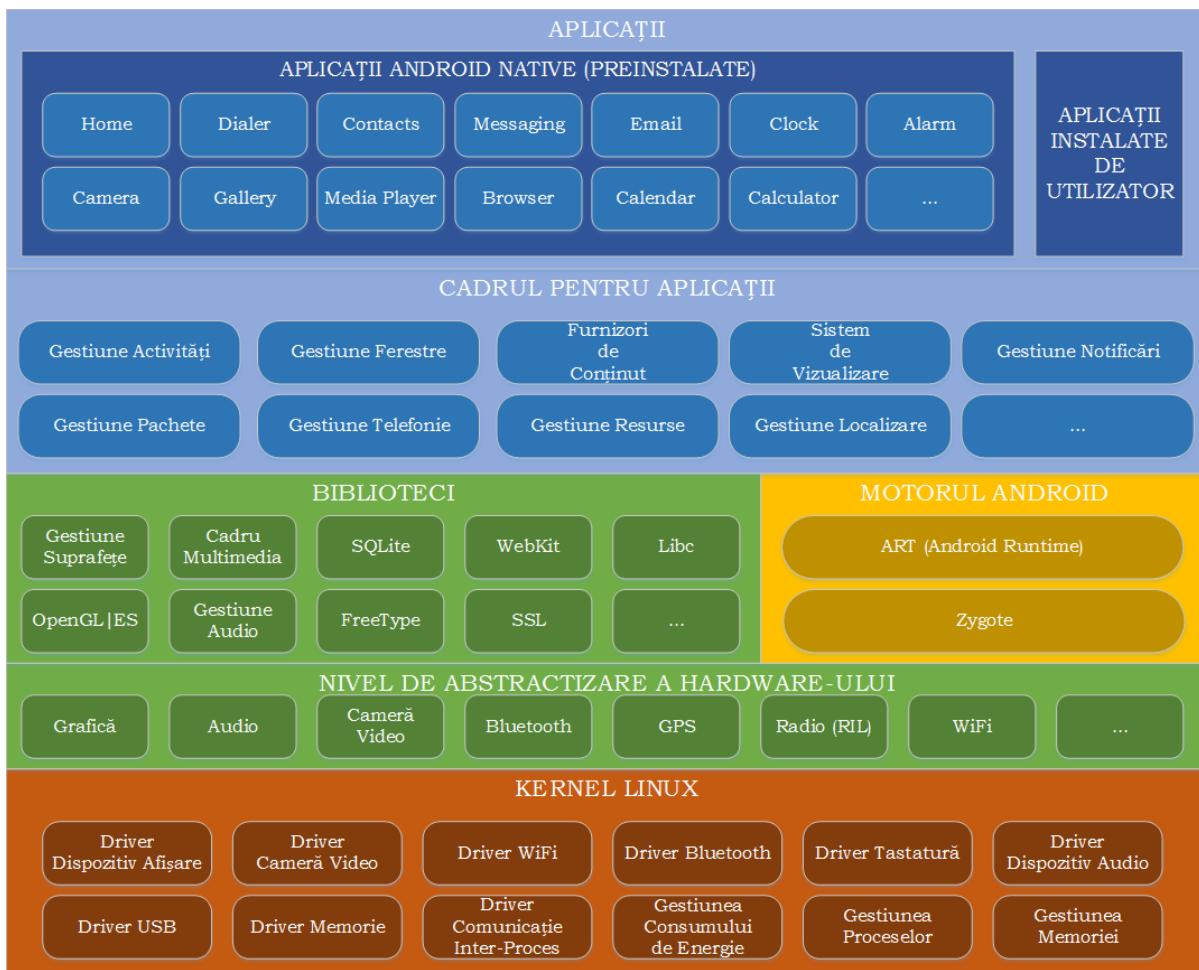
Pentru identificarea versiunilor se folosesc, de regulă, trei sisteme:

- un număr, ce respectă formatul major.minor[.build], desemnând dacă modificările aduse sunt substanțiale sau reprezintă ajustări ale unor probleme identificate anterior; versiunea curentă este 7.1.1, lansată la sfârșitul anului 2016;
- un nivel de API (același putând grupa un număr de mai multe versiuni), prin care se indică funcționalitățile expuse către programatori; versiunea curentă are nivelul de API 25;
- o denumire, având un nume de cod inspirat din lumea dulciurilor; termenii respectivi încep cu inițiale care respectă ordinea alfabetică; versiunea curentă este Nougat.

În momentul în care se ia decizia cu privire la versiunea pentru care se dezvoltă o aplicație Android, trebuie avute în vedere și cotele de piață ale dispozitivelor mobile. Dezvoltarea unei aplicații Android pentru cea mai nouă versiune are avantajul de a se putea utiliza cele mai noi funcționalități expuse prin API. Dezvoltarea unei aplicații Android pentru cea mai veche versiune are avantajul unei adresabilități pe scară largă. Un compromis în acest sens poate fi obținut prin intermediul **bibliotecilor de suport**, dezvoltate pentru fiecare versiune, prin intermediul cărora pot fi utilizate la niveluri de API mai mici funcționalități din niveluri de API mai mari (în limita capabilităților dispozitivului mobil respectiv). Utilizarea acestora reprezintă o practică recomandată în dezvoltarea aplicațiilor Android.

Arhitectura Android

Arhitectura sistemului de operare Android cuprinde cinci secțiuni grupate pe patru niveluri:



1. **Kernelul Linux** (cu unele modificări) conține driver-ele pentru diferitele componente hardware (ecran, cameră foto, tastatură, antenă WiFi, memorie flash, dispozitive audio), fiind responsabil cu gestiunea proceselor, memoriei, perifericelor (audio/video, GPS, WiFi), dispozitivelor de intrare/ieșire, rețelei și a consumului de energie; de asemenea, au fost implementate și unele îmbunătățiri:
 - a. **Binder**, sistemul de comunicație inter-proces, a fost adaptat, întrucât reprezintă mediul de comunicație principal dintre aplicații și sistemul de operare, inclusiv funcțiile (serviciile) dispozitivului mobil; expunerea sa este realizată prin intermediul AIDL (Android Interface Definition Language) prin care pot fi manipulate obiecte transformate în primitive utilizate la comunicația propriu-zisă dintre aplicații și sistemul de operare;
 - b. **Logger**, sistemul de jurnalizare, este esențial în cazul în care trebuie realizată depanarea aplicațiilor, în special pentru a detecta anumite situații particulare (informații cu privire la rețea, senzori); acesta este capabil să agere datele provenite atât de la aplicația propriu-zisă cât și de la sistemul de operare, datele fiind disponibile prin intermediul unor utilitare specializate;

- c. sistemul prin intermediul căruia se previne transferul sistemului de operare într-o stare de latență (**wake locks**), în care consumul de energie este redus, întrucât se blochează execuția oricărei aplicații; utilizarea unui astfel de mecanism trebuie realizată cu precauție, întrucât poate determina epuizarea bateriei;
 - d. sistemul de **alarme** oferă posibilitatea ca anumite sarcini să fie planificate la anumite momente de timp, putând fi executate, chiar dacă sistemul de operare se găsește într-o stare de latență;
 - e. **Viking Killer** este un mecanism prin care sistemul de operare revendică memoria utilizată, atunci când nivelul acesta atinge un anumit prag (aplicațiile Android care au fost rulate anterior sunt de regulă stocate în memorie pentru a se putea comuta rapid între ele, de vreme ce încărcarea în memorie este o operație costisitoare);
 - f. **YAFFS2** (Yet Another Flash File System) este un sistem de fișiere adecvat pentru cipuri flash bazate pe porți NAND; platforma Android este stocată pe mai multe partiții, ceea ce îi conferă flexibilitate la actualizări, împiedicând modificarea sa în timpul rulării (/boot - conține secvența de pornire, /system - stochează fișierele de sistem și aplicațiile încorporate, /recovery - deține o imagine din care se poate restaura sistemul de operare, /data - include aplicațiile instalate și datele aferente acestora, /cache - utilizată pentru fișiere temporare, folosind memoria RAM, pentru acces rapid).
2. **Bibliotecile** (user-space) conțin codul care oferă principalele funcționalități a sistemului de operare Android, făcând legătura între kernel și aplicații. Sunt incluse aici motorul open-source pentru navigare WebKit, biblioteca FreeType pentru suportul seturilor de caractere, baza de date SQLite utilizată atât ca spațiu de stocare cât și pentru partajarea datelor specifice aplicațiilor, biblioteca libc (Bionic), biblioteca de sistem C bazată pe BSD și optimizată pentru dispozitive mobile bazate pe Linux, biblioteci pentru redarea și înregistrarea de conținut audio/video (bazate pe OpenCORE de la PacketVideo), biblioteci SSL pentru asigurarea securității pe Internet și Surface Manager, bibliotecă pentru controlul accesului la sistemul de afișare care suportă 2D și 3D. Aceste biblioteci nu sunt expuse prin API, reprezentând detalii de implementare Android.
3. **Motorul Android** rulează serviciile de platformă precum și aplicațiile care le utilizează, fiind reprezentat de:
- a. **ART (Android Runtime)** este mașina virtuală Java care a fost implementată începând cu versiunea 5.0, folosind un tip de compilare AOH (Ahead of Time), în care bytecode-ul este transpus în cod mașină la momentul instalării, astfel încât acesta este executat direct de mediul dispozitivului mobil; compatibilitatea cu versiuni anterioare (care folosesc mașina virtuală Dalvik, ce se bazează pe un compilator JIT - Just in Time) este asigurată prin transformarea pachetelor în format .dex (Dalvik Executable) la momentul compilării, urmând

ca translatarea în format .oat să se realizeze la momentul instalării; fiecare aplicație Android rulează în procesul propriu, într-o instanță a mașinii virtuale ART, izolând astfel codul și datele sale prin intermediul unor permisiuni, care se aplică inclusiv la comunicația prin intermediul interfețelor de comunicare oferite de sistemul de operare Android;

- b. **Zygote** este procesul care gestionează toate aplicațiile, fiind lansat în execuție odată cu sistemul de operare:

- I. inițial, creează o instanță a mașinii virtuale Java pentru sistemul de operare Android, în contextul căreia plasează serviciile de bază: gestiunea energiei, telefonie, furnizori de conținut, gestiunea pachetelor, serviciul de localizare, serviciul de notificări;
 - II. atunci când este necesar să lanseze în execuție o anumită aplicație, se clonează, partajând astfel componentele sistemului de operare Android, astfel încât să se asigure performanța (timp de execuție) și eficiența (memorie folosită), de vreme ce fiecare aplicație trebuie rulată în propria sa instanță a mașinii virtuale Java;
4. **Cadrul pentru Aplicații** expune diferențele funcționalități ale sistemului de operare Android către programatori, astfel încât aceștia să le poată utiliza în aplicațiile lor.
5. La nivelul de **aplicații** se regăsesc atât produsele împreună cu care este livrat dispozitivul mobil (Browser, Calculator, Camera, Contacts, Clock, FM Radio, Launcher, Music Player, Phone, S Note, S Planner, Video Player, Voice Recorder), cât și produsele instalate de pe Play Store sau cele dezvoltate de programatori.

Funcționalități Android

De vreme ce Android este un produs open-source, producătorii având posibilitatea de a-l modifica în mod gratuit, nu există configurații hardware sau software standard. Totuși, Android implementează următoarele funcționalități:

- **stocare** - folosește SQLite, o bază de date relațională ce utilizează resurse puține
- **conectivitate** - suportă GSM/CDMA, GPRS, EDGE, 3G, IDEN, EV-DO, UMTS, Bluetooth (inclusiv A2DP și AVRCP), WiFi, LTE, WiMAX
- **WiFi Direct** - tehnologie care permite aplicațiilor să se descopere și să se interconecteze peste o conexiune punct-la-punct având lățime de bandă mare
- **Android Beam** - o tehnologie bazată pe NFC (Near Field Communication) care permite utilizatorilor să partajeze conținut instant, prin apropierea dispozitivelor mobile respective
- **mesagerie** - atât SMS cât și MMS
- **navigare pe Internet** - bazat pe motorul open source pentru navigare WebKit împreună cu motorul JavaScript de la Chrome V8 suportând HTML5 și CSS3

- **multimedia** - suportă formatele H.263, H.264 (într-un container 3GP sau MP4), MPEG-4 SP, AMR, AMR-WB (într-un container 3GP), AAC, HE-AAC (într-un container MP4 sau 3GP), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF și BMP
- **grafică** - 2D optimizată, 3D (OpenGL ES)
- **senzori** - accelerometru, cameră foto, busolă digitală (magnetometru), senzor de proximitate, GPS / AGPS
- **multi-touch** - suportă ecrane cu posibilitate de contact în mai multe puncte concomitent
- **multi-tasking** - permite rularea de aplicații cu mai multe fire de execuție
- **GCM (Google Cloud Messaging)** - serviciu ce permite dezvoltatorilor să trimită date de dimensiuni mici către utilizatori pe dispozitive Android, fără a avea nevoie de o soluție de sincronizare proprietară
- **multi-Language** - suport pentru text unidirecțional și bidirecțional
- **suport pentru aplicații Flash** (până în versiunea 4.3)
- **legătură la Internet** - suportă partajarea conexiunilor la Internet ca punct de distribuție cu fir / fără fir

Android vs. iPhone

Piața de telefoane inteligente este dominată de Android (care - potrivit estimărilor - își va menține supremația până în 2018), cu 82.8% (peste un miliard de dispozitive vândute) pe întreg anul 2015, în timp ce Apple deține doar 13.9%, la nivel mondial.

În 2015, numărul de aplicații disponibile pentru platforma Android le-a depășit pe cele destinate dispozitivelor Apple (1.6 milioane, comparativ cu 1.5 milioane). Și în privința numărului de descărcări Android se află în fața Apple cu aproximativ 25%. Totuși, politica de distribuție a aplicațiilor (faptul că aplicațiile iPhone pot fi instalate numai prin intermediul App Store, în timp ce Android pune la dispoziție mai multe posibilități - Play Store, Amazon App Store, prin conexiune USB de la calculator, prin email sau prin pagina Internet a organizației), la care se adaugă numărul mare de programe gratuite și posibilitatea de piratare mai ușoară se traduce prin încasările obținute, Apple generând un profit mult mai mare din aplicații decât Google (cifrele oficiale nu sunt disponibile încă). O altă explicație a acestei situații este dată și de puterea financiară a posesorilor de produse Apple (valoarea unui iPhone fiind de aproximativ 600 dolari) față de puterea de cumpărare a persoanelor ce dețin un telefon Android (al cărui preț mediu este de 200-300 dolari), existând o corespondență directă cu disponibilitatea de achiziționa aplicații. De asemenea, statisticile arată că utilizatorii Apple își folosesc mult mai intens dispozitivele pentru accesarea de conținut Internet decât cei cumpărătorii de produse echipate cu Android.

Aplicațiile destinate dispozitivelor mobile reprezintă un segment extrem de productiv al economiei, doar vânzările din AppStore depășind încasările obținute din producția de filme de la Hollywood. Ca tematică, cele mai multe aplicații aparțin domeniului social (jocuri, fotografie, muzică, aplicații culinare, stil de viață), urmate de cele destinate gestionării unor segmente ale unor afaceri.

În privința limbajului de programare utilizat pentru dezvoltarea de aplicații mobile, iPhone folosește **Objective-C**, similar cu C++, care nu se bucură însă de o răspândire prea largă (cu

excepția aplicațiilor pentru iPhone), în timp ce Android utilizează **Java** (limbajul de programare cel mai adoptat pe scară largă în cadrul corporațiilor) dar și **C/C++**, prin care se pot apela (cu o oarecare dificultate) aplicații native, prin suport JNI (numărul bibliotecilor disponibile în cazul C/C++ este mai redus decât în Java, însă viteza aplicațiilor este mai mare). Dezvoltarea aplicațiilor pentru iPhone se poate realiza numai folosind mașini Mac (echipate cu MacOS), în timp ce aplicațiile Android pot fi scrise în orice sistem de operare cu Java și Eclipse (atât PC-uri cât și Mac-uri echipate cu Windows, Linux sau MacOS). În această situație, se pune problema cotelor de piață ale sistemelor de operare în cadrul companiilor dezvoltatoare de software, care creează sau nu oportunități pentru dezvoltarea unui anumit tip de aplicații (la începutul lui 2016, sistemele de operare Windows dețineau 90,60% din piață, OS X 7,68%, sistemele de operare cu kernel Linux 1,71%, iar alte sisteme de operare 0,01%).

Așadar, dacă pentru utilizare personală nu se poate stabili un câștigător clar între Android și iPhone (deși produsul celor de la Apple pare să aibă un ușor avantaj într-un număr mai mare de aplicații și prin loialitatea clienților), în cazul aplicațiilor dezvoltate de corporații situația este inversă, date fiind posibilitățile de instalare și limbajul de programare folosit.

Comunitatea programatorilor Android

Ajunsă deja la a șaptea versiune, Android este o platformă care beneficiază de experiența a numeroși dezvoltatori ce poate fi exploataată:

- [Google Android Training](#) conține o serie de tutoriale și exemple de clase grupate în funcție de diferite subiecte, utile pentru deprinderea cunoștințelor de bază pentru dezvoltarea aplicațiilor Android.
- [Stack Overflow](#) este un forum pentru programatori editat în mod colaborativ, conținând întrebări și răspunsuri la acestea (cele mai bune putând fi identificate cu ușurință prin voturile primite de la participanți). Este destul de probabil ca o întrebare să își găsească deja răspunsul pe acestă resursă.
- [Android Discuss](#) este o listă de discuții monitorizată îndeaproape de echipa Android de la Google astfel încât reprezintă un loc unde pot fi clarificate numeroase nelămuriri putând fi însușite diferite sfaturi și trucuri.

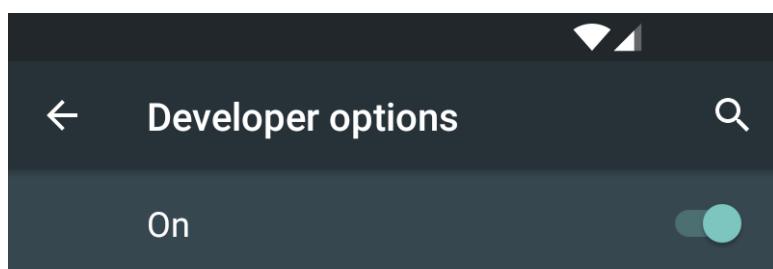
Cerințe pentru dezvoltarea unei aplicații Android (obligatoriu)

Pentru dezvoltarea unei aplicații Android sunt necesare:

1. [kit-ul de dezvoltare pentru limbajul de programare Java](#)
2. [SDK-ul de Android](#), pentru care se descarcă definițiile corespunzătoare unuia sau mai multor niveluri de API
3. un mediu integrat de dezvoltare (IDE)
 - a. [Eclipse](#), cu plugin-ul ADT (Android Developer Tools)
 - b. [Android Studio](#)
4. un dispozitiv pe care să se ruleze aplicațiile

- a. un emulator
 - I. [Genymotion](#)
 - II. [Android Virtual Device](#) (livrat împreună cu SDK-ul de Android)
- b. un telefon mobil cu sistemul de operare Android pentru care s-a dezvoltat aplicația

Pentru a se putea rula o aplicație pe un dispozitiv mobil fizic, trebuie să se activeze posibilitatea de depanare prin USB, din *Settings → System → Developer Options*. Această opțiune trebuie activată, ca de altfel și opțiunea *Debugging → Android Debugging* (pe unele sisteme poate apărea ca *USB Debugging*).



Debugging

Android debugging

Enable the Android Debug Bridge (adb) interface



ADB over network

Enable TCP/IP debugging over network interfaces (Wi-Fi, USB networks). This setting is reset on reboot



Debugging notify

Display a notification when USB or network debugging is enabled



Device hostname

android-70d59f81e71ab2b7

Local terminal

Enable terminal app that offers local shell access

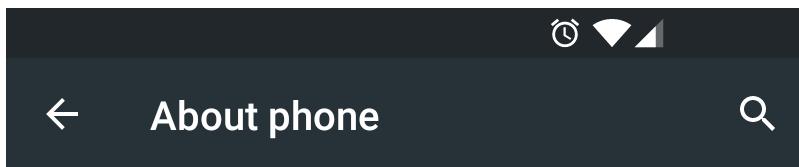


Bug report shortcut

Show a button in the power menu for taking a



În situația în care opțiunea *Developer Options* nu este disponibilă, aceasta poate fi vizualizată prin intermediul modului *Developer*, obținut prin apăsarea de mai multe ori asupra opțiunii *Build Number* din secțiunea *Settings → System → About Phone*.



CyanogenMod version
12.1-20150515-UNOFFICIAL-n7000

Android version
5.1.1

Baseband version
N7000XXLT3

Kernel version
3.0.64-Bauner-g7f37452-dirty
bauner@pc #1
Fri May 15 17:56:48 CEST 2015

Build date
Fr 15. Mai 17:07:57 CEST 2015

Build number
cm_n7000-userdebug 5.1.1 LMY47V 92f7b18bf0 test-keys

SELinux status
Permissive

Dacă telefonul nu este recunoscut la conectarea prin USB, trebuie instalate niște reguli pentru udev, conform instrucțiunilor de pe [stackexchange](#):

```
student@eim2017:~$ sudo bash
```

```
student@eim2017:~# lsusb
```

După ce s-a identificat dispozitivul mobil (prin intermediul comenzi `lsusb`), se precizează o regulă pentru acesta:

```
student@eim2017:~# gedit /etc/udev/rules.d/51-android.rules
```

/etc/udev/rules.d/51-android.rules

```
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", ATTR{idProduct}=="d002",  
MODE="0660", GROUP="plugdev", SYMLINK+="android%n"
```

Se reîncarcă dispozitivele conectate prin USB:

```
student@eim2017:~# /etc/init.d/udev restart
```

Mai multe informații sunt disponibile și la <http://blog.janosgyerik.com/adding-udev-rules-for-usb-debugging-android-devices/>.

Sisteme de control a versiunilor

Ce este un sistem de control al versiunilor?

Un sistem de control al versiunilor (eng. VCS - Version Control System) este un mecanism prin intermediul căruia sunt gestionate fișiere / proiecte în dinamică (pe măsură ce sunt modificate), în scopul de a se putea realiza revenirea la o anumită stare în caz de eroare (restaurarea unei versiuni stable) și pentru a permite colaborarea între mai multe echipe care lucrează concomitent la diferite funcționalități ale unui același proiect.

Deși în mod curent astfel de produse sunt folosite pentru dezvoltarea de aplicații (urmărindu-se o gestionare eficientă a codului sursă și a utilizatorilor care implementează anumite funcționalități sau corectează defecte), ele pot fi folosite și pentru alte tipuri de proiecte, ce implică lucru cu fișiere binare, pentru care diferențele între diferite versiuni se realizează mai dificil.

Clasificarea sistemelor de control al versiunilor

În prezent, sunt folosite trei tipuri de sisteme de control a versiunilor, fiecare dintre acestea fiind adecvate unei anumite situații:

1. **sisteme locale de control a versiunilor**, în cazul în care se dorește monitorizarea variantelor unui fișier exclusiv pe discul local; în baza de date, versiunile sunt reținute sub forma diferențelor (rezultatul comenzi `diff`) dintre versiuni succesive, astfel că se poate reveni oricând la o stare anterioară (exemplu: rcs);
2. **sisteme centralizate de control a versiunilor** implică stocarea diferențelor dintre versiuni într-o bază de date rezidentă pe un server dedicat, la care au acces toți utilizatorii implicați, astfel încât fiecare poate consulta versiunea curentă (exemplu: CVS, Subversion, Perforce);
 - avantaje: posibilitatea de colaborare între echipe care lucrează la același proiect, stabilirea de drepturi cu privire la fișierele ce pot fi încărcate pe server de fiecare utilizator în parte;

- dezavantaje: în cazul când serverul dedicat pe care găsește baza de date cu tot istoricul versiunilor se defectează, există riscul de a se pierde toate aceste informații (dacă nu există copii de siguranță), păstrându-se doar versiunile salvate pe mașinile utilizatorilor; mai mult, într-o astfel de situație utilizatorii se află în incapacitatea de a mai transmite propriile modificări și de a consulta modificările celorlalți;
3. **sisteme distribuite de control a versiunilor** în care consultarea stării actuale a unui fișier presupune și descărcarea, pe discul local, a întregului istoric de modificări astfel încât acesta să poată fi reconstituit în situații precum defectarea serverului; de asemenea, acestea au capacitatea de a gestiona mai multe depozite aflate la distanță, chiar în cazul în care acestea conțin același proiect, permitând stabilirea unor anumite fluxuri de transmitere a informației (exemple: Git, Mercurial, Bazaar, Darcs);

GIT

Istoric

Apariția Git în 2005 este strâns legată de dezvoltarea kernelului pentru Linux, proiect open-source la care lucra o comunitate destul de numeroasă de programatori. Dacă anterior actualizările erau distribuite sub forma unor arhive ce conțineau modificările (1991-2002), respectiv prin intermediul unui sistem distribuit de control al versiunilor denumit BitKeeper (2002-2005), pe măsură ce proiectul a devenit mai complex și din ce în ce mai multe persoane și-au exprimat disponibilitatea de a contribui la dezvoltarea acestuia, s-a pus problema conceperii unui produs care să satisfacă cerințele legate de viteză, arhitectură scalabilă, suport pentru dezvoltare non-liniară (numeroase ramificații la care se lucrează concomitent), distribuire totală, capacitate de a gestiona proiecte de dimensiuni mari.

Caracteristici (obligatoriu)

Git se diferențiază de alte sisteme de control al versiunilor prin câteva caracteristici:

1. actualizările aduse fișierelor din cadrul proiectului nu se rețin sub forma unui set de diferențe între versiuni succesive, ci ca **instantanee** ale acestora la momentul respectiv (pentru eficiență, în situația în care nu există modificări ale unui fișier între versiuni succesive, se va reține o legătură către original); în acest fel, Git se aseamănă foarte mult cu un sistem de fișiere, peste care sunt implementate mai multe utilizare.

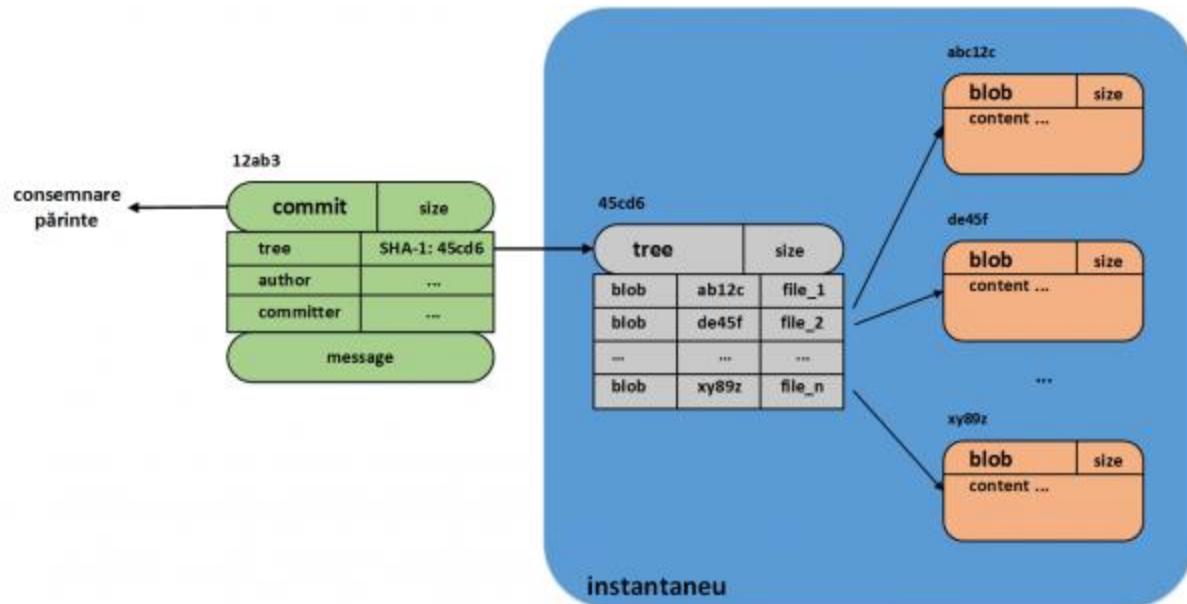
Pentru fiecare operație de consemnare, Git stochează un obiect de tip `commit` ce conține o referință către părintele sau părinții săi (consemnarea / consemnările anterioare, din care a fost obținut, prin modificarea fișierelor) și o referință către instantaneul propriu-zis. De asemenea, în cadrul acestui obiect se rețin și informații despre dimensiunea totală și suma de control SHA-1, autor și contributor (nume și adresă de poștă electronică), data la care a fost

realizată

consemnarea,

mesajul

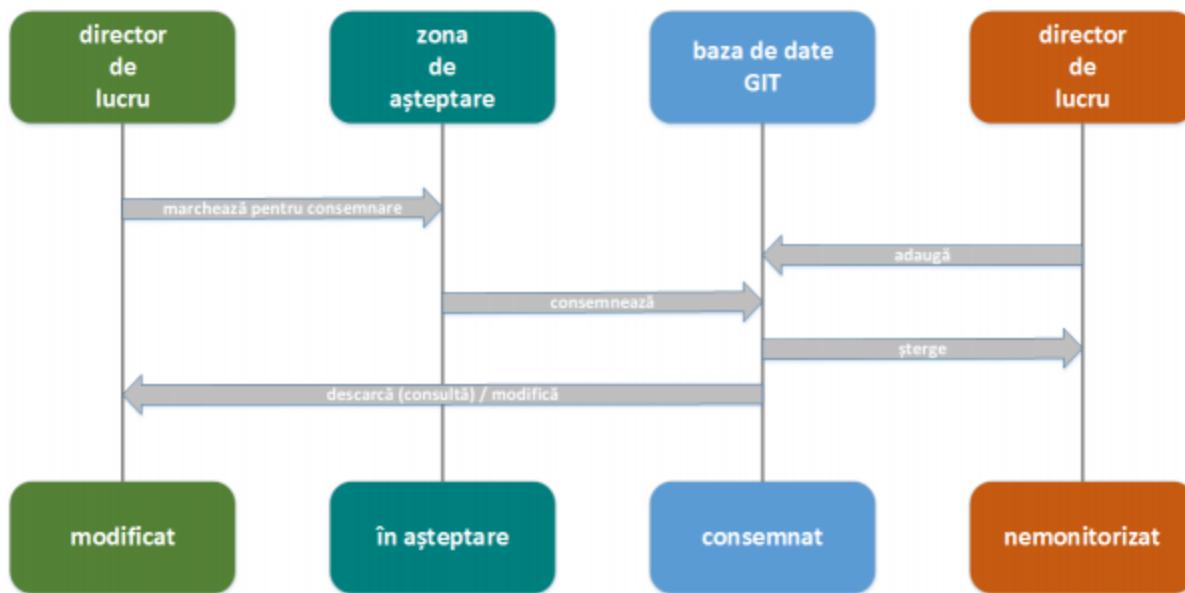
asociat.



Structura unui instantaneu este formată din:

- un obiect de tip `tree`, o structură de date ce conține referințe spre fiecare resursă ce au fost modificată, aceasta fiind identificată prin denumire și sumă de control SHA-1; și în cazul unui astfel de obiect se rețin dimensiunea totală și suma de control SHA-1;
 - mai multe obiecte de tip `blob`, corespunzătoare fiecărei resurse din cadrul consemnării respective; acestea sunt identificate prin denumire, dimensiune și suma de control SHA-1.
2. **majoritatea operațiilor se realizează local**, astfel încât nu este necesară o conexiune la Internet pentru a putea consemna anumite actualizări sau pentru a consulta istoricul modificărilor; acest fapt este posibil datorită stocării pe discul local a bazei de date care conține toate versiunile proiectului; se îmbunătățește astfel viteza, iar încărcarea/descărcarea informațiilor (de) pe server se realizează numai atunci când aceasta este posibilă;
 3. **integritatea** informațiilor este asigurată prin intermediul sumelor de control folosind algoritmul SHA-1; astfel, un fișier / director este reținut în baza de date printr-un sir de 40 de caractere hexazecimale calculat pe baza conținutului acestuia; în acest mod, este imposibil să nu se detecteze coruperea unui fișier / director, în situația în care aceasta se produce;
 4. majoritatea operațiilor implică în principiu adăugarea de informații în baza de date care conține toate versiunile proiectului, ceea ce face ca **probabilitatea de a pierde fișiere să fie extrem de redusă**, odată ce acestea au fost consemnate;
 5. **cele trei stări** în care se pot găsi datele monitorizate prin sistemul de control al versiunilor
- Git sunt:

- **consemnat** (eng. committed): modificările aduse au fost stocate în **directorul GIT** (eng. GIT directory/repository); în acesta sunt reținute toate metadatele și obiectele bazei de date locale conținând istoricul tuturor versiunilor, elemente preluate de fiecare dată când sunt descărcate actualizări ale proiectului;
- **modificat** (eng. modified): modificările aduse nu au fost stocate în baza de date locală; astfel de fișiere se regăsesc în **zona de lucru** (eng. working area), de pe discul local, în care a fost descărcată o anumită versiune a proiectului, spre a fi modificată;
- **în așteptare** (eng. staged): modificările aduse au fost consemnate spre a fi incluse într-un instantaneu ce va fi stocat în baza de date locală; acestea sunt reținute în **zona de așteptare** (eng. staging area), un fișier (denumit și index) din directorul GIT care conține toate modificările ce vor fi consemnate în următoarea versiune.



Utilizatorul poate alege să ignore anumite date generate (fișiere binare, executabile), care se regăsesc în zona de lucru, dar care nu vor fi monitorizate și nu vor fi consemnate în baza de date, chiar dacă se marchează întregul director din care fac parte în acest scop.
Un scenariu tipic de utilizare a sistemului de versiune Git implică:

1. descărcarea unei versiuni din directorul GIT în zona de lucru;
2. modificarea fișierelor corespunzătoare din zona de lucru;
3. marcarea informațiilor actualizate din zona de lucru ca fiind în așteptare, în vederea consemnării lor;
4. consemnarea propriu-zisă a datelor din zona de așteptare înapoi în directorul Git.

Instalare & Configurare (obligatoriu)

Instrucțiunile pentru instalarea și configurarea Git sunt disponibile [aici](#). Informații suplimentare cu privire la comenzi Git și sintaxa acestora pot fi obținute folosind paginile de manual:

```
student@eim2017:~$ git help <command>  
student@eim2017:~$ git <command> --help  
student@eim2017:~$ man git-<command>
```

Moduri de Lucru (obligatoriu)

Local

În cazul în care se dorește monitorizarea unui proiect nou / existent prin sistemul de control al versiunilor Git, directorul în care se găsește acesta va trebui inițializat folosind comanda:

```
student@eim2017:~$ git init
```

Astfel, se creează un director `.git` în care vor fi plasate toate versiunile fișierelor care sunt monitorizate. Inițial, acesta este vid.

Indicarea fișierelor care sunt monitorizate se face prin intermediul comenzi `git add <file>`, fiind permisă și folosirea de expresii regulate folosind măști pentru a indica conținutul unui întreg director. Prin intermediul acestei comenzi, fișierele sunt transferate din directorul de lucru în zona de așteptare.

Consemnarea propriu-zisă a fișierelor se face rulând comanda `git commit -m "<message>"`, mesajul care o însoțește trebuind să fie relevant pentru modificările care au fost realizate. Se recomandă ca această operație să fie realizată cât mai des pentru actualizări de dimensiuni relativ reduse ale codului sursă. În acest moment, fișierele trec din zona de așteptare în directorul Git.

Păși descriși vor fi repetați în situația în care aceleași fișiere care au fost consemnate într-o versiune anterioare sunt modificate din nou. Odată ce se realizează operația de consemnare, zona de așteptare devine goală și fișierele modificate nu vor fi preluate în directorul Git chiar dacă sunt marcate ca fiind monitorizate.

La Distanță

În situația în care utilizatorul vrea să lucreze pe un proiect găzduit pe un server la distanță, poate descărca întregul conținut în zona de lucru, inclusiv istoricul complet al versiunilor anterioare (care poate fi ulterior reconstituit după această copie, în cazul coruperii informațiilor stocate pe serverul la distanță), prin intermediul comenzi:

```
git clone <URL> [<local_directory>]
```

unde:

- URL - reprezintă adresa serverului la distanță care găzduiește proiectul, putând fi utilizate în acest sens mai multe protocoale pentru transferul de informație

- **git**

```
student@eim2017:~$ git clone git://github.com/eim2017/Laborator01.git
```

- **https**

```
student@eim2017:~$ git clone https://github.com/eim2017/Laborator01
```

- **ssh**

```
student@eim2017:~$ git clone git@github.com:eim2017/Laborator01.git
```

- **local_directory (optional)** - denumirea directorului local în care va fi stocată versiunea curentă a proiectului (precum și istoricul din directorul Git), în cazul în care se dorește schimbarea acestuia

Fișierele astfel descărcate, aflate atât în zona de lucru cât și în directorul Git pot fi modificate în funcție de necesități și transferate, succesiv, în zona de aşteptare (prin `git add`) și în baza de date locală (prin `git commit -m`).

Dacă este necesar ca fișierele modificate să fie încărcate pe serverul de unde au fost preluate, trebuie ca mai întâi să se actualizeze modificările care se vor fi produs pe acesta între timp (folosind comanda `git pull --rebase`) - rezolvând eventualele conflicte - și apoi să se transfere efectiv prin intermediul comenzii `git push origin master`.

Operații Git (obligatoriu)

Determinarea stării fișierelor din zona de lucru

Comanda `git status` furnizează informații cu privire la starea fișierelor aflate în zona de lucru, fie că este vorba de resurse deja monitorizate (care se găsesc în directorul Git) care au fost modificate între timp, fie că este vorba despre date care au fost adăugate (și care nu au fost marcate în mod explicit pentru a fi ignorate). De asemenea, comanda indică și ramificația (eng. branch) pe care se găsește utilizatorul în mod curent.

O astfel de comandă poate întoarce mai multe rezultate:

- toate fișierele din directorul de lucru se regăsesc încămai și în directorul Git (nu au fost realizate modificări)
- ```
student@eim2017:~$ git status
```
- ```
On branch master
```
- ```
Your branch is up-to-date with 'origin/master'.
```

```
nothing to commit, working directory clean
```

- există date în zona de lucru care nu sunt monitorizate (nu au făcut parte dintr-un instantaneu anterior), care vor fi consemnate numai în situația în care se specifică acest lucru în mod explicit; acest mecanism urmărește ca fișierele generate să nu fie incluse în mod eronat în directorul Git

```
student@eim2017:~$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
 (use "git add <file>..." to include in what will be committed)
MyFirstAndroidApplication/
```

```
nothing added to commit but untracked files present (use "git add" to track)
```

Monitorizarea acestor date se face prin intermediul comenzi `git add <files/directory>`, care suportă specificarea de expresii regulate desemnând măști pentru indicarea mai multor fișiere/directoare.

- există informații în zona de așteptare care vor fi incluse în următorul instantaneu în momentul în care se va realiza consemnarea (trecute în secțiunea `Changes to be committed`); eventualele fișiere care au existat în instantanee anterioare și au fost modificate în directorul de lucru nu vor fi marcate ca făcând parte din zona de așteptare (trecute în secțiunea `Changes not staged for commit`) dacă versiunile respective nu vor fi incluse în mod explicit prin comanda `git add <files/directory>`

```
student@eim2017:~$ git status
```

- On branch master
- Your branch is up-to-date with 'origin/master'.
- Changes to be committed:
- (use "git reset HEAD <file>..." to unstage)
- - new file: MyFirstAndroidApplication/.classpath
  - new file: MyFirstAndroidApplication/.project
  - new file: MyFirstAndroidApplication/.settings/org.eclipse.jdt.core.prefs
  - new file: MyFirstAndroidApplication/AndroidManifest.xml
  - new file: MyFirstAndroidApplication/ic\_launcher-web.png
  - new file: MyFirstAndroidApplication/libs/android-support-v4.jar
  - new file: MyFirstAndroidApplication/proguard-project.txt
  - new file: MyFirstAndroidApplication/project.properties
  - new file: MyFirstAndroidApplication/res/drawable-hdpi/ic\_launcher.png
  - new file: MyFirstAndroidApplication/res/drawable-mdpi/ic\_launcher.png
  - new file: MyFirstAndroidApplication/res/drawable-xhdpi/ic\_launcher.png
  - new file: MyFirstAndroidApplication/res/drawable-xxhdpi/ic\_launcher.png
  - new file: MyFirstAndroidApplication/res/layout/activity\_main.xml

```
■ new file: MyFirstAndroidApplication/res/menu/main.xml
■ new file: MyFirstAndroidApplication/res/values-v11/styles.xml
■ new file: MyFirstAndroidApplication/res/values-v14/styles.xml
■ new file: MyFirstAndroidApplication/res/values-w820dp/dimens.xml
■ new file: MyFirstAndroidApplication/res/values/dimens.xml
■ new file: MyFirstAndroidApplication/res/values/strings.xml
■ new file: MyFirstAndroidApplication/res/values/styles.xml
■ new file: MyFirstAndroidApplication/src/ro/pub/cs/systems/eim/lab01/MainActivity.java
■
■ Changes not staged for commit:
■ (use "git add <file>..." to update what will be committed)
■ (use "git checkout -- <file>..." to discard changes in working directory)
■
```

```
modified: README.md
```

Fișierele care nu au existat într-un instataneu anterior sunt marcate prin `new file`, iar cele care au fost modificate față de versiunile precedente sunt marcate prin `modified`. Un același fișier se poate găsi simultan în secțiunile `Changes to be committed`, respectiv `Changes not staged for commit` în situația în care au mai fost realizate modificări asupra acestuia după rularea comenzi `git add <files/directory>`.

## Transferul fișierelor din zona de lucru în zona de așteptare

Prin intermediul comenzi `git add <files/directory>` se specifică fișiere / directoare care vor fi transferate din zona de lucru în zona de așteptare, pentru a fi incluse în următorul instantaneu în momentul în care se va realiza consemnarea acestora.

La consemnare vor fi incluse fișierele / directoarele în starea în care acestea se aflau la momentul la care a fost emisă comanda `git add`. Dacă ele sunt modificate după precizarea

acestei comenzi, actualizările nu vor fi consemnate în directorul Git decât dacă se rulează din nou comanda `git add`.

Dacă argumentul este un director, conținutul acestuia va fi inclus în mod recursiv.

```
student@eim2017:~$ git add MyFirstAndroidApplication/*
```

În cazul în care un fișier este transferat din greșală din zona de lucru în zona de așteptare, parcursul invers poate fi realizat prin intermediul comenzi `git reset HEAD <file>`.

Dacă se dorește eliminarea modificărilor realizate asupra unui fișier din zona de lucru, se poate folosi comanda `git checkout -- <file>` (disponibilă începând cu versiunea 1.6.1). O astfel de operație este totuși periculoasă, în sensul că actualizările respective sunt pierdute fără posibilitatea de a mai putea fi recuperate (întrucât nu au fost consemnate niciodată în directorul Git).

Alte operații care pot fi realizate asupra fișierelor din zona de lucru / zona de așteptare sunt mutarea (redenumirea), respectiv ștergerea acestora.

Comanda `git mv <source> <target>` este folosită pentru operația de mutare (redenumire) a unui fișier. Necesitatea sa este dată de faptul că Git nu detectază în mod automat fișierele care sunt redenumite pe discul local. Dacă se verifică starea fișierelor din zona de lucru / zona de așteptare, fișierele redenumite apar în zona `Changes to be committed`, în secțiunea `renamed`.

Comanda `git mv <source> <target>` este echivalentă cu aceeași succesiune de operații:

```
student@eim2017:~$ mv <source> <target>
student@eim2017:~$ git rm <source>
student@eim2017:~$ git add <target>
student@eim2017:~$ git mv README.md README.txt
```

Prin intermediul comenzi `git rm <file/directory>`, resursa specificată este eliminată nu numai de pe discul local, ci și din zona de așteptare, astfel încât atunci când se realizează operația de consemnare, acesta este eliminat din directorul Git și din lista fișierelor care sunt monitorizate. În cazul în care resursa este eliminată manual, doar de pe discul local, aceasta va apărea ca modificată în zona de lucru, dar nemarcată pentru a fi consemnată în zona de așteptare. Dacă se verifică starea fișierelor din zona de lucru / zona de așteptare, fișierele șterse apar în zona `Changes to be committed`, în secțiunea `deleted`.

În cazul în care fișierul a fost modificat înainte de a se încerca ștergerea sa, comanda trebuie rulată cu opțiunea `-f`, pentru a preveni eliminarea accidentală a unor resurse. Dacă se dorește ștergerea unui director care nu este gol, comanda trebuie rulată cu opțiunea `-r` (recursiv).

Dacă se dorește ca resursa să fie eliminată doar din zona de așteptare (să nu mai fie monitorizată) dar păstrată pe discul local (dacă a fost inclusă într-o versiune din directorul Git, omițându-se specificarea sa în fișierul `.gitignore`), comanda trebuie rulată cu opțiunea `--cached`.

```
student@eim2017:~$ git rm LICENSE
rm 'LICENSE'
```

## Ignorarea unor (tipuri de) fișiere

Este recomandat ca fișierele generate să nu fie incluse în directorul Git (binare, jurnale), acestea fiind rezultatul procesului de compilare / rulare a proiectului. Mai mult, nu se dorește ca informații cu privire la modificarea lor să fie incluse în raportul rezultat ca rulare a comenzi `git status`.

Un astfel de comportament poate fi obținut prin specificarea acestui tip de fișier în `.gitignore`, respectându-se următoarele reguli:

- pe fiecare linie se poate specifica un anumit tip de fișier care va fi ignorat
- liniile vide sau precedate de caracterul # nu sunt luate în considerare
- pot fi folosite expresii regulate (standard) pentru a specifica un set de fișiere
- \* desemnează 0 sau mai multe caractere (trebuie precedat de \)
- \*\* (suportat din versiunea 1.8.2) referă conținutul unui director
- ? indică un singur caracter
- între { și } sunt trecute colecții de şablonane
- între [ și ] sunt trecute seturi de caractere (sau intervale între două caractere, separate prin -)
- folosirea caracterului ! înaintea unui şablon îl neagă

La crearea unui proiect nou, GitHub oferă posibilitatea de a include în mod automat un fișier `.gitignore` în funcție de limbajul de programare utilizat.

Pentru Android, conținutul fișierului `.gitignore` generat este:

### `.gitignore`

```
Built application files
```

```
*.apk
```

```
*.ap_
```

```
Files for the Dalvik VM
```

```
*.dex
```

```
Java class files
```

```
*.class
```

```
Generated files
```

```
bin/
```

```
gen/
```

```
Gradle files

.gradle/

build/

Local configuration file (sdk path, etc)

local.properties

Proguard folder generated by Eclipse

proguard/

Log Files

*.log
```

## Vizualizarea modificărilor

Prin intermediul comenții **git diff**, utilizatorul are posibilitatea de a vizualiza în ce constau actualizările pentru fiecare fișier în parte.

- rulată fără parametri, comanda indică diferențele pentru fișierele care au fost modificate, dar nu au fost marcate pentru a fi incluse în următorul instantaneu (cu alte cuvinte, sunt indicate diferențele dintre zona de lucru și zona de așteptare)

- student@eim2017:~\$ git diff
- 
- diff --git a/README.md b/README.md
- index dff203e..77d86c8 100644
- --- a/README.md
- +++ b/README.md
- @@ -1 +1,3 @@

- -# Laborator01
- +MyFirstAndroidApplication
- +=====

+A simple application displaying a dialog in which the user is asked to provide his/her name in order to be properly greeted to the EIM laboratory.

Dacă toate fișierele din zona de lucru au fost fie marcate pentru a fi incluse în următorul instantaneu, fie sunt ignorate, rezultatul comenzi git diff va fi vid.

- rulată cu parametrul --cached sau --staged (disponibil din versiunea 1.6.1, efectul este identic), comanda indică diferențele pentru fișierele care au fost marcate pentru a fi incluse în următorul instantaneu, față de situația existentă în directorul Git (cu alte cuvinte, sunt indicate diferențele dintre zona de așteptare și directorul Git)

- student@eim2017:~\$ git diff --staged
- 
- 
- diff --git a/MyFirstAndroidApplication/.classpath  
b/MyFirstAndroidApplication/.classpath
- new file mode 100644
- index 000000..5176974
- --- /dev/null
- +++ b/MyFirstAndroidApplication/.classpath
- @@ -0,0 +1,9 @@
- +<?xml version="1.0" encoding="UTF-8"?>
- +<classpath>
- + <classpathentry kind="con" path="com.android.ide.eclipse.adt.ANDROID\_FRAMEWORK"/>

```

■ + <classpathentry exported="true" kind="con"
path="com.android.ide.eclipse.adt.LIBRARIES"/>

■ + <classpathentry exported="true" kind="con"
path="com.android.ide.eclipse.adt.DEPENDENCIES"/>

■ + <classpathentry kind="src" path="src"/>

■ + <classpathentry kind="src" path="gen"/>

■ + <classpathentry kind="output" path="bin/classes"/>

■ +</classpath>

```

...

## Consemnarea modificărilor (transferul fișierelor din zona de aşteptare în directorul Git)

Dacă toate fișierele din zona de lucru au fost marcate spre a fi incluse în următorul instantaneu (rezultatul comenzi `git status` nu conține nici un fișier în secțiunea `Changes not staged for commit`), ele pot fi consemnate, adică trecute din zona de aşteptare în directorul Git, prin intermediul comenzi `git commit`.

Rulată fără parametru, comanda deschide editorul de text implicit (sau cel indicat de proprietatea `core.editor`) completat cu rezultat comenzi `git status`, care poate fi modificat pentru a constitui mesajul asociat fișierelor consemnate în directorul Git. Dacă se dorește ca în acest mesaj să se includă și rezultatul comenzi `git diff`, se poate utiliza parametrul `-v`.

Mesajul care însoțește consemnarea fișierelor poate fi inclusă direct în cadrul comenzi prin intermediul parametrului `-m`.

```

student@eim2017:~$ git commit -m "initial commit: graphical user interface,
some code to handle the button click"

[master 95fa0b2] initial commit: graphical user interface, some code to handle
the button click

21 files changed, 300 insertions(+)

create mode 100644 MyFirstAndroidApplication/.classpath
create mode 100644 MyFirstAndroidApplication/.project
create mode 100644 MyFirstAndroidApplication/.settings/org.eclipse.jdt.coreprefs
create mode 100644 MyFirstAndroidApplication/AndroidManifest.xml

```

```

create mode 100644 MyFirstAndroidApplication/ic_launcher-web.png

create mode 100644 MyFirstAndroidApplication/libs/android-support-v4.jar

create mode 100644 MyFirstAndroidApplication/proguard-project.txt

create mode 100644 MyFirstAndroidApplication/project.properties

create mode 100644 MyFirstAndroidApplication/res/drawable-
hdpi/ic_launcher.png

create mode 100644 MyFirstAndroidApplication/res/drawable-
mdpi/ic_launcher.png

create mode 100644 MyFirstAndroidApplication/res/drawable-
xhdpi/ic_launcher.png

create mode 100644 MyFirstAndroidApplication/res/drawable-
xxhdpi/ic_launcher.png

create mode 100644 MyFirstAndroidApplication/res/layout/activity_main.xml

create mode 100644 MyFirstAndroidApplication/res/menu/main.xml

create mode 100644 MyFirstAndroidApplication/res/values-v11/styles.xml

create mode 100644 MyFirstAndroidApplication/res/values-v14/styles.xml

create mode 100644 MyFirstAndroidApplication/res/values-w820dp/dimens.xml

create mode 100644 MyFirstAndroidApplication/res/values/dimens.xml

create mode 100644 MyFirstAndroidApplication/res/values/strings.xml

create mode 100644 MyFirstAndroidApplication/res/values/styles.xml

create mode 100644 MyFirstAndroidApplication/src/ro/pub/cs/systems/eim/lab01/MainActivity.java

```

Se observă că în rezultatul acestei comenzi sunt indicate ramificația pe care se face consemnarea (`master`), suma de control SHA-1 a consemnării (`95fa0b2`), numărul de fișiere modificate (21) precum și statistici cu privire la numărul de linii adăugate (300), respectiv eliminate (0).

Consemnarea va transfera în directorul Git numai acele fișiere care au fost marcate în acest sens, existând în zona de așteptare. Dacă se dorește să se realizeze transferul direct din zona de lucru în directorul Git, fără a mai trece în zona de așteptare (doar pentru fișierele monitorizate care au fost modificate), comanda `git commit -m "<message>"` trebuie rulată și cu parametrul `-a`.

```
student@eim2017:~$ git commit -a -m "initial commit: graphical user interface, some code to handle the button click"
```

Într-un astfel de caz, nu mai este necesar ca în prealabil să se ruleze comanda `git add`.

Dacă s-au omis resurse în cadrul celei mai recente versiuni transmise către directorul Git, se poate încerca consemnarea modificărilor respective prin intermediul comenzi `git commit -amend` (ulterior marcării fișierelor în cauză în zona de aşteptare) prin intermediul căreia se suprascrie varianta anterioară cu actualizările din zona de aşteptare curentă. Ca și în cazul precedent, este afișat un editor de text care conține mesajul corespunzător versiunii consemnate, acesta putând fi actualizat.

## Etichetarea versiunilor

Unei versiuni consemnate în directorul Git îi poate fi asociată o etichetă (eng. tag) prin care se desemnează, de regulă, o anumită funcționalitate.

Git oferă posibilitatea de a defini două tipuri de etichete, prin intermediul comenzi `git tag`:

- **etichete 'ușoare'** (eng. lightweight) pentru care nu se rețin informații suplimentare, cu excepția sumei de control asociate consemnării respective; de regulă, sunt folosite pentru resurse temporare sau în situația în care informațiile asociate nu sunt necesare

```
student@eim2017:~$ git tag v1.0
```

- **etichete adnotate** care sunt stocate ca obiecte de sine stătătoare în directorul Git, reținându-se numele și adresa de poștă electronică a utilizatorului care a realizat-o, un mesaj asociat (informații distincte de cele ale consemnării asociate etichetei) precum și suma de control

```
student@eim2017:~$ git tag -a v1.0 -m 'an annotated tag for the first release version'
```

Etichetele de acest tip pot fi **semnate** folosind GPG (GNU Privacy Guard), dacă utilizatorul dispune de o cheie privată, acestea având avantajul că pot fi verificate

```
student@eim2017:~$ git tag -s v1.0 -m 'a signed tag for the first release version'
```

Verificarea unei etichete semnate se face prin intermediul comenzi `git tag -v <tag_name>`, fiind necesară existența cheii publice a utilizatorului care a semnat-o. Așadar, opțiunile cu care se poate ruleaza comanda `git tag` în cazul unei etichete adnotate sunt:

- `-a`: specifică faptul că este vorba despre o etichetă nesemnată
- `-s`: specifică faptul că este vorba despre o etichetă semnată
- `-m`: indică mesajul asociat etichetei (dacă se rulează comanda fără acest parametru, va fi deschis editorul de text pentru ca acesta să fie introdus)
- `-v`: dacă se dorește verificarea unei etichete semnate

Lista tuturor etichetelor asociate unui proiect poate fi consultată dacă se rulează comanda `git tag` fără nici un parametru:

```
student@eim2017:~$ git tag
v1.0
```

Aceasta suportă opțiunea `-l <tag_mask>` pentru a se afișa doar lista etichetelor care respectă o anumită expresie regulată.

Dacă se dorește consultarea conținutului unei etichete, se poate utiliza comanda `git show <tag>`, prin care sunt listate toate informațiile asociate versiunii respective.

```
student@eim2017:~$ git show v1.0

commit 5e4b82a0783e5841e5c5241d44104e8b15e4270f

Author: eim2017 <informaticamobila2017@gmail.com>

Date: Sun Feb 22 18:00:00 2015 +0300
```

Initial commit

În situația în care o versiune a fost csemnată fără a-i se asocia o etichetă, o astfel de operație poate fi realizată și ulterior, adăugând la comanda `git tag` suma de control asociată respectivei versiuni (obținută ca rezultat al comenzi `git log`).

```
student@eim2017:~$ git tag -a v1.1 -m 'version 1.1'
e71c1c4b1b9818292b4cda084e47e25bfb573507
```

Acestei comenzi i se poate asocia și numai un fragment al sumei de control asociat versiunii, atât timp cât acesta o identifică în mod unic.

De regulă, etichetele nu sunt transferate în cadrul depozitelor găzduite de serverele la distanță, fiind necesar ca acest lucru să fie realizat manual:

- pentru o singură etichetă, se specifică denumirea acesteia `git push <remote_name> <tag>`;
- pentru toate etichetele, se folosește comanda `git push <remote_name> --tags`.

### Vizualizarea istoricului de versiuni

Istoricul versiunilor csemnate pentru un proiect poate fi consultat, în ordine invers cronologică (de la cele mai noi la cele mai vechi) rulând comanda `git log`. Pentru fiecare versiune a proiectului, vor fi afișate următoarele informații:

- suma de control SHA-1 asociată csemnării
  - numele și adresa de poștă electronică a utilizatorului care a realizat csemnarea
  - data și ora la care a fost realizată csemnarea
  - mesajul care însărcă csemnarea
- Cele mai folosite opțiuni ale acestei comenzi sunt:

| OPȚIUNE         | FUNCTIONALITATE                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -p              | afisează diferențele (rezultatul comenzi <code>git diff</code> ) realizate în consemnarea curentă față de consemnarea anterioară                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| --word_diff     | afisează diferențele între versiuni, la nivel de cuvânt (mai ales pentru fișiere text de dimensiuni mari, mai rar pentru cod sursă): cuvintele adăugate sunt cuprinse între {+}, iar cuvintele șterse între [-]; dacă se dorește omiterea contextului în care apare diferența (linia precedentă și linia care o succede), se poate folosi opțiunea <code>-U1</code> (se afisează doar linia curentă)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| --stat          | afisează statistici cu privire la fiecare consemnare în parte: numărul de fișiere modificate, lista acestora și numărul de modificări (adăugări / ștergeri) - la nivel de linie - pentru fiecare dintre acestea                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| --shortstat     | afisează doar statistici generale: numărul total de fișiere modificate, adăugări și ștergeri (la nivel de linie)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| --name-only     | afisează lista fișierelor modificate                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| --name-status   | afisează lista fișierelor modificate împreună cu natura actualizării (actualizat, adăugat, șters)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| --abbrev-commit | afisează numai câteva caractere (din cele 40) ale sumei de control SHA-1 asociată fiecărei consemnări                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| --relative-date | afisează momentul la care a fost realizată consemnarea relativ la data curentă                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| --graph         | afisează un graf (în format ASCII) al ramificațiilor, ilustrând momentul la care acestea au fost combinate                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| --pretty        | modifică formatul în care sunt afișate informațiile despre versiuni:<br>♦ <code>oneline</code> - informațiile despre fiecare consemnare sunt afișate pe o singură linie<br>♦ <code>short, full, fuller</code> - controlează cantitatea de informație<br>♦ <code>format</code> - permite personalizarea informațiilor, util pentru situația în care conținutul urmează să fie prelucrat în mod automat:<br>✓ %H - suma de control, %h - suma de control prescurtată<br>✓ %T - arborele sumei de control, %t - arborele sumei de control prescurtat<br>✓ %P - suma de control a părintelui, %p - suma de control prescurtată a părintelui<br>✓ %an - numele autorului, %ae - adresa de poștă electronică a autorului<br>✓ %ad - data autorului, %ar - data autorului (relativă)<br>✓ %cn - numele contributorului*), %ce - adresa de poștă electronică a contributorului<br>✓ %cd - data contributorului, %cr - data contributorului (relativă)<br>✓ %s - subiectul |
| --oneline       | afisează numai câteva caractere (din cele 40) ale sumei de control SHA-1 asociată fiecărei consemnări, pe o singură linie<br>prescurtare pentru <code>--pretty=oneline abbrev-commit</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

\*) Distincția dintre autor și contributor este următoarea:

- autorul este utilizatorul care a implementat o anumită funcționalitate
- contributorul este utilizatorul care a realizat cea mai recentă consemnare pentru funcționalitatea în cauză

În situația în care pentru un proiect au fost realizate foarte multe consemnări, există posibilitatea ca rezultatul comenzi `git log` să fie limitat doar la cele care sunt de interes

(oricum, în mod implicit, se realizează o paginare astfel încât este imposibil ca acestea să fie afișate toate dintr-o dată):

| OPȚIUNE           | FUNCȚIONALITATE                                                                                                                                                       |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -<n>              | afișează doar cele mai recente n consemnări                                                                                                                           |
| --since, --after  | afișează consemnările realizate după cu o anumită dată / oră absolută / relativă *)                                                                                   |
| --until, --before | afișează consemnările realizate înainte de o anumită dată/oră absolută / relativă                                                                                     |
| --author          | afișează doar consemnările având un anumit autor                                                                                                                      |
| --committer       | afișează doar consemnările având un anumit contributor                                                                                                                |
| --grep            | afișează doar consemnările având anumite cuvinte în mesajul asociat; dacă se dorește indicarea mai multor cuvinte, trebuie utilizată împreună cu opțiunea --all-match |
| --<path>          | afișează doar consemnările care au realizat modificări asupra fișierelor localizate în calea specificată; această opțiune trebuie inclusă întotdeauna ultima          |

\*) Formatul în care se afișează data este specificat de opțiunea --date, care poate lua valorile iso (ISO 8601), rfc (RFC 2822), raw (număr de secunde de la 01/01/1970 UTC), local (în conformitate cu zona de timp), relative (raportat la momentul curent de timp).

Dacă se utilizează mai multe opțiuni, se vor lua în considerare consemnările care îndeplinesc toate criteriile specificate.

```
student@eim2017:~$ git log --pretty=format:"%h - %an [%ae] - %ar -> %s" --graph
* df5829c - andreirosucojocaru [andrei.rosucojocaru@gmail.com] - 5 minutes
ago -> resolved conflicts among customized_message and fade_effect branches
| \
| * c857d27 - andreirosucojocaru [andrei.rosucojocaru@gmail.com] - 9 minutes
ago -> added fade effect
|
* | e1a9a2e - andreirosucojocaru [andrei.rosucojocaru@gmail.com] - 8 minutes
ago -> added customized message
| /
* 839f14e - andreirosucojocaru [andrei.rosucojocaru@gmail.com] - 42 minutes ago
-> initial commit: graphical user interface, some code to handle the button
click
* e2594f8 - eim2017 [informaticamobila2017@gmail.com] - 2 days ago -> Initial
commit
```

## Gestiunea depozitelor la distanță (obligatoriu)

Mai mulți utilizatori pot colabora în cadrul unui proiect Git aflat pe un server la distanță (eng. remote repository), pe care pot avea fie doar drepturi de citire fie atât drepturi de citire cât și de scriere. Operațiile pe care le pot realiza sunt descărcarea de cod sursă, respectiv încărcarea (în situația în care au drepturi suficiente).

## Vizualizarea referințelor către depozitele la distanță (git remote)

Prin intermediul comenții `git remote`, pot fi consultate depozitele la distanță cu care se lucrează în mod curent. În mod implicit, sunt afișate doar denumirile scurte asociate acestora. În cazul în care se dorește să se afișeze și URL-ul locației corespunzătoare fiecărui depozit la distanță, se va folosi opțiunea `-v`.

Dacă directorul pe care se lucrează în mod curent a fost obținut ca urmare a clonării unui depozit la distanță, acesta va fi afișat ca având denumirea `origin`.

```
student@eim2017:~$ git remote -v

Laborator01 andreirosucojocaru
git@github.com:andreirosucojocaru/Laborator01.git (fetch)

Laborator01_andreirosucojocaru
git@github.com:andreirosucojocaru/Laborator01.git (push)

origin https://github.com/eim2017/Laborator01 (fetch)

origin https://github.com/eim2017/Laborator01 (push)
```

Utilizatorul va avea drepturi de scriere numai pe depozitele la distanță pentru care dispune de cheile SSH corespunzătoare.

## Adăugarea unei referințe către un depozit la distanță

Pentru a putea referi un depozit la distanță prin intermediul unei denumiri (mai scurte) se va rula comanda `git remote add`:

```
git remote add <remote_name> <URL>
```

Astfel, nu va mai fi necesară introducerea întregului URL corespondent locației la care se găsește depozitul la distanță (pentru comenzi de încărcare/descărcare, respectiv consultare a acestuia), fiind suficientă precizarea denumirii asociate.

```
student@eim2017:~$ git remote add Laborator01_andreirosucojocaru
git@github.com:andreirosucojocaru/Laborator01.git
```

## Descărcarea de cod sursă de pe un depozit la distanță

Pentru a descărca cod sursă aflat într-un depozit găzduit de un server la distanță pe discul local există trei posibilități:

- comanda `git clone <URL>` realizează o copie a datelor aflate la locația indicată de URL, inclusiv a tuturor ramificațiilor (eng. branches), ramificația `master` (în situația în care există de pe server fiind monitorizată pentru modificări, astfel încât acestea să fie integrate automat (eng. merged) în codul sursă din directorul de lucru
- student@eim2017:~\$ git clone https://github.com/eim2017/Laborator01.git

- Cloning into 'Laborator01'...
- remote: Counting objects: 5, done.
- remote: Compressing objects: 100% (4/4), done.
- remote: Total 5 (delta 0), reused 0 (delta 0)
- Unpacking objects: 100% (5/5), done.

Checking connectivity... done.

- comanda `git fetch <remote_name>` descarcă toate informațiile din depozitul de pe serverul la distanță care nu se regăsesc pe discul local, creându-se referințe către ramificația de la locația respectivă, care pot fi consultate pentru ca ulterior să fie integrate
- student@eim2017:~\$ git fetch Laborator01\_andreirosuojocaru
- remote: Counting objects: 39, done.
- remote: Compressing objects: 100% (19/19), done.
- remote: Total 39 (delta 5), reused 39 (delta 5)
- Unpacking objects: 100% (39/39), done.
- From <https://github.com/andreirosuojocaru/Laborator01>

```
* [new branch] master -> Laborator01_andreirosuojocaru/master
```

Comanda `git fetch` nu integrează modificările existente în depozitul existent pe serverul la distanță în mod automat, fiind necesar ca această operație să fie realizată manual.

- comanda `git pull` descarcă modificările dintr-o ramificație monitorizată (de exemplu, ramificația `master` în situația în care depozitul este clonat) din depozitul de pe serverul la distanță, încercând să le integreze în mod automat în codul sursă din directorul de lucru.  
**Încărcarea de cod sursă pe un depozit la distanță**

Transmiterea modificărilor operate asupra unui cod sursă pe un depozit găzduit de un server la distanță se face prin intermediul comenzi `git push <remote_name> <branch>`, care primește ca parametrii denumirea referinței către depozitul la distanță (aceasta

este origin în situația în care proiectul a fost clonat) și ramificația pe care se găsește codul sursă care urmează a fi clonat.

```
student@eim2017:~$ git push origin master

Counting objects: 47, done.

Delta compression using up to 2 threads.

Compressing objects: 100% (29/29), done.

Writing objects: 100% (45/45), 891.37 KiB | 0 bytes/s, done.

Total 45 (delta 2), reused 0 (delta 0)

To https://github.com/eim2017/Laborator01

 e2594f8..839f14e master -> master
```

Operația `git push` va eșua în situația în care depozitul din directorul de lucru local nu este actualizat, adică dacă pe serverul la distanță se găsesc modificări (încărcate de alți utilizatori) care nu au fost descărcate încă. În această situație, înainte de a se transmite propriile modificări, va trebui să se ruleze comanda `git pull`.

## Consultarea conținutului unui depozit la distanță

Dacă se dorește obținerea de informații cu privire la conținutul unui proiect găzduit de un server la distanță, se poate rula comanda `git remote show <remote-name>`, care afișează:

- URL-ul depozitului la distanță
- ramificația curentă din depozitul la distanță
- ramificația care este încărcată pe server atunci când se rulează comanda `git push`
- ramificațiile descărcate de pe server în momentul în care se rulează comanda `git pull`
- ramificațiile care au fost adăugate pe server dar nu se găsesc și în directorul de lucru local, ramificațiile care au fost șterse de pe server dar se mai găsesc încă în directorul de lucru local

```
student@eim2017:~$ git remote show origin

* remote origin
 Fetch URL: https://github.com/eim2017/Laborator01
 Push URL: https://github.com/eim2017/Laborator01
 HEAD branch: master

 Remote branch:

 master tracked

 Local branch configured for 'git pull':
```

```
master merges with remote master

Local ref configured for 'git push':

 master pushes to master (up to date)
```

## Redenumirea unei referințe către un depozit la distanță

În cazul în care se dorește să se schimbe denumirea asociată unei referințe către un depozit la distanță, acest lucru poate fi realizat prin comanda `git remote rename <old_remote_name> <new_remote_name>`.

```
student@eim2017:~$ git remote rename Laborator01_andreirosuojocaru Lab01_arc
```

Comanda are impact și asupra denumirilor ramificațiilor asociate depozitului la distanță corespunzător referinței, care va fi prefixat de noua denumire.

## Ștergerea unei referințe către un depozit la distanță

O referință către un depozit la distanță poate fi ștearsă în situația în care codul sursă nu mai este disponibil la adresa respectivă sau dacă utilizatorul respectiv nu mai lucrează pe proiectul în cauză. Comanda utilizată într-o astfel de situație este `git remote rm <remote_name>`.

```
student@eim2017:~$ git remote rm Lab00_arc
```

## Gestiunea ramificațiilor (facultativ)

O ramificație (eng. branch) marchează un punct din care dezvoltarea unui proiect se realizează în zone diferite (corespunzătoare unor anumite funcționalități sau unor anumite echipe care le implementează), în scopul obținerii unui nivel de izolare cât mai mare până la obținerea rezultatului dorit, când mai multe ramificații pot fi integrate împreună.

De regulă, există o ramificație corespunzătoare unei versiuni stabile, aflată în exploatare și o ramificație pentru versiunea de lucru a proiectului în care sunt integrate, după ce au fost suficient testate, ramificațiile aferente diferitelor funcționalități. Acestea vor constitui o propunere de actualizare care va fi inclusă în următoarea versiune stabilă atunci când se consideră necesar.

Modul în care Git implementează gestiunea ramificațiilor îl evidențiază printre celelalte sisteme de versionare a codului sursă, operațiile care le vizează realizându-se foarte rapid.

O ramificație în Git nu este altceva decât o referință (un pointer) către cea mai recentă consemnare (obiect de tip `commit`) pe care o urmează pe măsură ce sunt dezvoltate noi versiuni. Astfel, o ramificație este un fișier ce conține cele 40 de caractere ale sumei de control SHA-1 corespunzătoarele consemnării curente, motiv pentru care crearea și distrugerea acesteia se realizează foarte rapid (comparativ cu alte sisteme de versionare a codului sursă care copiau întregul proiect). De asemenea, referințele către consemnările părinte face ca operația de integrare a modificărilor să fie foarte rapidă, întrucât se poate identifica ușor care este consemnarea de bază (comună tuturor ramificațiilor implicate) peste care se pot aplica actualizările.

Datorită modului eficient în care sunt implementate ramificațiile în Git, se recomandă ca pentru fiecare funcționalitate care va fi implementată să se creeze o ramificație separată care va fi integrată doar după ce se rezolvă toate problemele identificate.

Implicit, ramificația curentă pe care se lucrează în Git poartă denumirea de `master`. Ramificația pe care se găsește utilizatorul în mod curent este indicată de o altă referință, denumită `HEAD`.

## Vizualizarea ramificațiilor

Pentru a se lista ramificațiile din cadrul proiectului curent, se rulează comanda `git branch`, fără nici un parametru:

```
student@eim2017:~$ git branch

customized_message

fade_effect

* master
```

În cadrul rezultatului, ramura pe care se găsește utilizatorul în mod curent este marcată prin caracterul `*`, al cărui conținut este reflectat de directorul de lucru și pe care urmează să se realizeze următoarea operație de consemnare.

Opțiunile cu care se poate rula această comandă sunt:

- `-v` pentru a se indica cea mai recentă consemnare corespunzătoare fiecărei ramificații (se afișează o parte din suma de control și mesajul asociat)
- `--merged` pentru a se afișa doar ramificațiile care au fost deja integrate în cadrul ramificației pe care utilizatorul se găsește în mod curent  
Ramificațiile care au fost deja integrate în ramificația pe care utilizatorul se găsește în mod curent pot fi șterse încărcăt conținutul lor există în mai multe locuri simultan.
- `--no-merged` pentru a se afișa ramificațiile al căror conținut nu a fost încă integrat în cadrul ramificației pe care utilizatorul se găsește în mod curent  
Sistemul Git nu va permite ștergerea ramificațiilor al căror conținut nu a fost încă integrat decât în situația în care se rulează comanda `git branch -D <branch_name>`, cu pierderea modificărilor realizate în cadrul acestora.

## Crearea unei ramificații

Comanda `git branch <branch_name>` are rolul de a crea o nouă ramificație în cadrul proiectului, având o denumire dată. Inițial, aceasta va fi o referință către aceeași versiune ca și `master`, urmând ca pe măsură ce sunt realizate noi consemnări, să indice obiectele de tip `commit` corespunzătoare acestora.

```
student@eim2017:~$ git branch customized_message

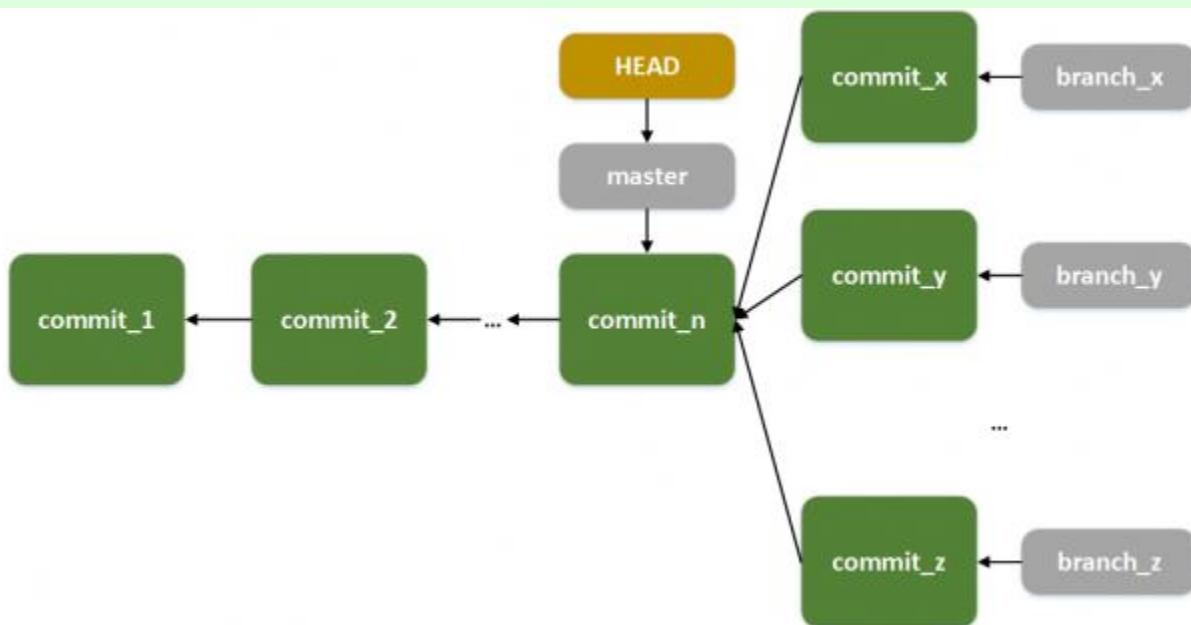
student@eim2017:~$ git branch fade_effect
```

## Transferul între ramificații

Prin intermediul comenzi `git checkout <branch_name>` se realizează mutarea din ramificația curentă în ramificația indicată ca parametru, prin modificarea pointerului `HEAD`. Totodată, se actualizează și conținutul directorului de lucru de pe discul local, corespunzător cu structura ramificației pe care s-a trecut.

```
student@eim2017:~$ git checkout customized_message
Switched to branch 'customized_message'
```

Este recomandat ca înainte de a se realiza mutarea pe o altă ramificație, să se consemneze toate modificările din directorul de lucru întrucât, în caz contrar, operația de transfer nu va fi permisă (restaurarea directorului de lucru la conținutul corespunzător ramificației respective va intra în conflict cu modificările realizate).



Operațiile de creare a unei noi ramificații și de mutare în cadrul acesteia (obținută prin rularea succesivă a comenziilor `git branch` și `git checkout`) pot fi realizate concomitent prin intermediul comenzi `git checkout -b <branch_name>`.

```
student@eim2017:~$ git checkout -b customized_message
Switched to a new branch 'customized_message'
```

## Integrarea conținutului a două ramificații și rezolvarea conflictelor

Integrarea modificărilor realizate pe o ramificație poate fi realizată în Git prin:

- merge
- rebase

Rezultatul celor două tipuri de operații este în totdeauna același, în sensul că versiunea obținută va avea același conținut, distincția constând în modul în care este consemnat istoricul: în cazul merge acesta reflectă exact ce s-a întâmplat (care este punctul din care

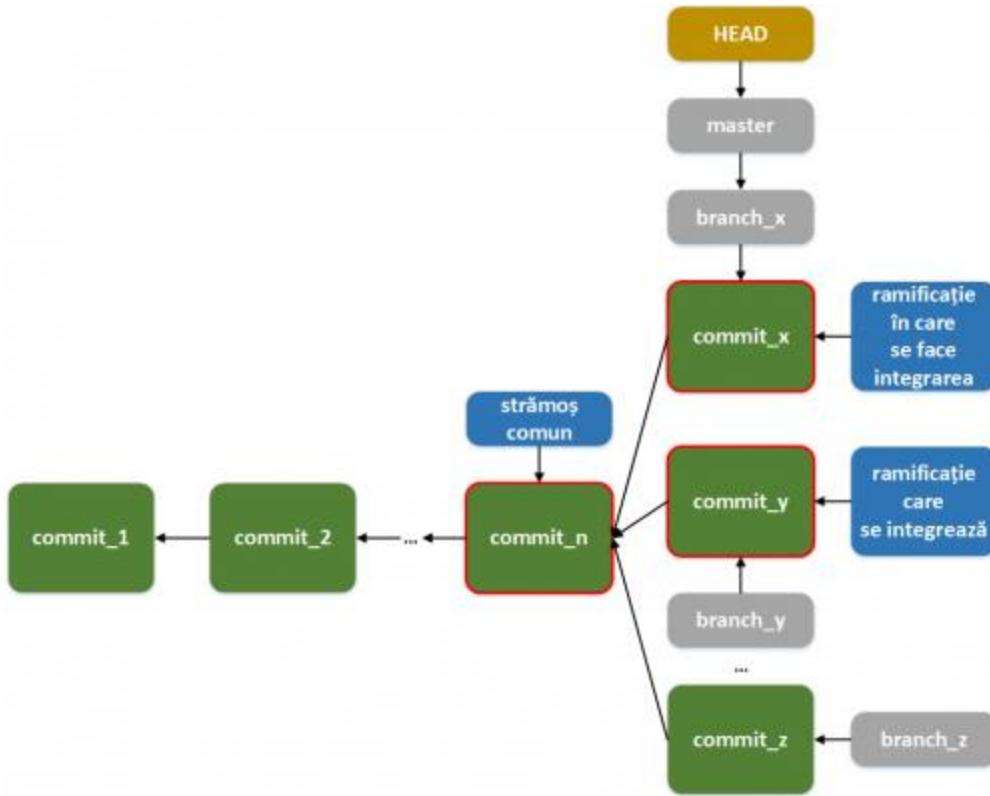
proiectul a fost dezvoltat în paralel și care este momentul de timp în care modificările au fost integrate), fiind însă mai dificil de gestionat, în timp ce în cazul rebase, dezvoltarea proiectului apare ca fiind liniară, ceea ce face ca referințele către stările proiectului să poată fi mutate mai ușor, putând fi introduse însă probleme serioase în cazul în care această operație este realizată pentru o consemnare existentă pe un server la distanță (aceasta dispare de pe server, în timp ce poate să existe în directoarele de lucru de pe discul local al utilizatorilor).

## Varianta MERGE

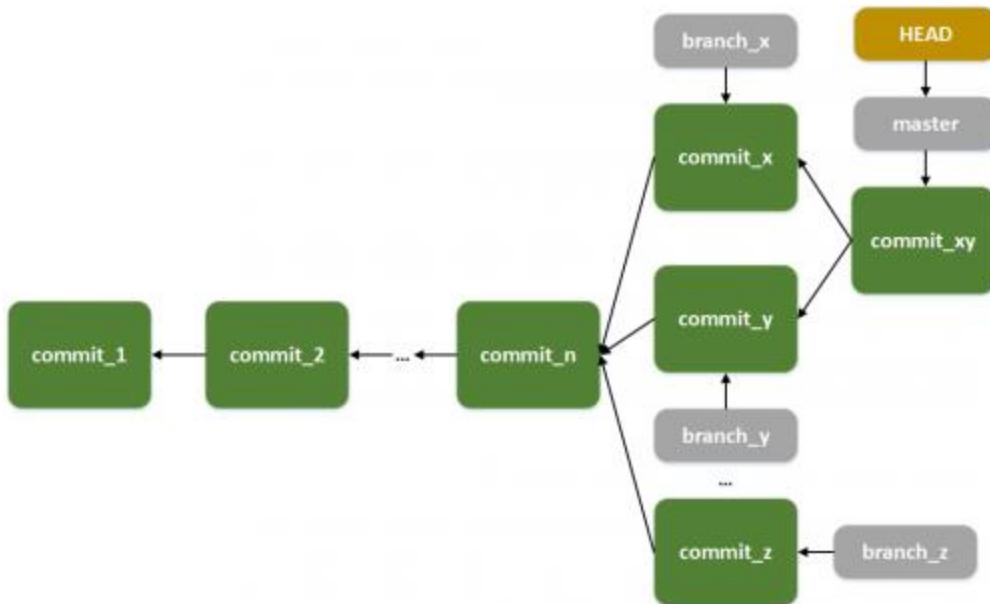
Integrarea modificărilor realizate pe o ramificație în varianta MERGE se face prin comanda `git merge <branch_name>`, după ce toate modificările din cadrul acesteia au fost consemnate și s-a trecut înapoi pe ramificația master.

```
student@eim2017:~$ git checkout master
Switched to branch 'master'
student@eim2017:~$ git merge customized_message
Updating 839f14e..e1a9a2e
Fast-forward
 .../src/ro/pub/cs/systems/eim/lab01/MainActivity.java | 1 +
 1 file changed, 1 insertion(+)
student@eim2017:~$ git merge fade_effect
Auto-merging
MyFirstAndroidApplication/src/ro/pub/cs/systems/eim/lab01/MainActivity.java
CONFLICT (content): Merge conflict in
MyFirstAndroidApplication/src/ro/pub/cs/systems/eim/lab01/MainActivity.java
Automatic merge failed; fix conflicts and then commit the result.
```

- operația poartă denumirea de fast-forward în situația în care obiectul `commit` corespunzător ramificației care este integrată referă ramificația cu care se integrează (urmărind pointerul părinte), întrucât în acest caz pointer-ii `master` și `HEAD` sunt pur și simplu mutați înainte;
- în situația în care dezvoltarea proiectului s-a realizat prin ramificații paralele dintr-un anumit punct înainte și obiectul `commit` corespunzător ramificației care este integrată nu referă ramificația cu care se integrează (urmărind pointerul părinte), se procedează la următorul algoritm:
- se identifică cel mai recent strămoș comun al celor două ramificații (urmărind pointerul părinte al celor două ramificații până se întâlnesc punctul din care acestea au urmat căi independente de dezvoltare)



- se creează o nouă ramificație în care sunt integrate modificările din cele două ramificații, pornind de la codul sursă comun al celui mai recent strămoș comun, pointerul părinte acestuia indicând spre ambele ramificații din care a provenit



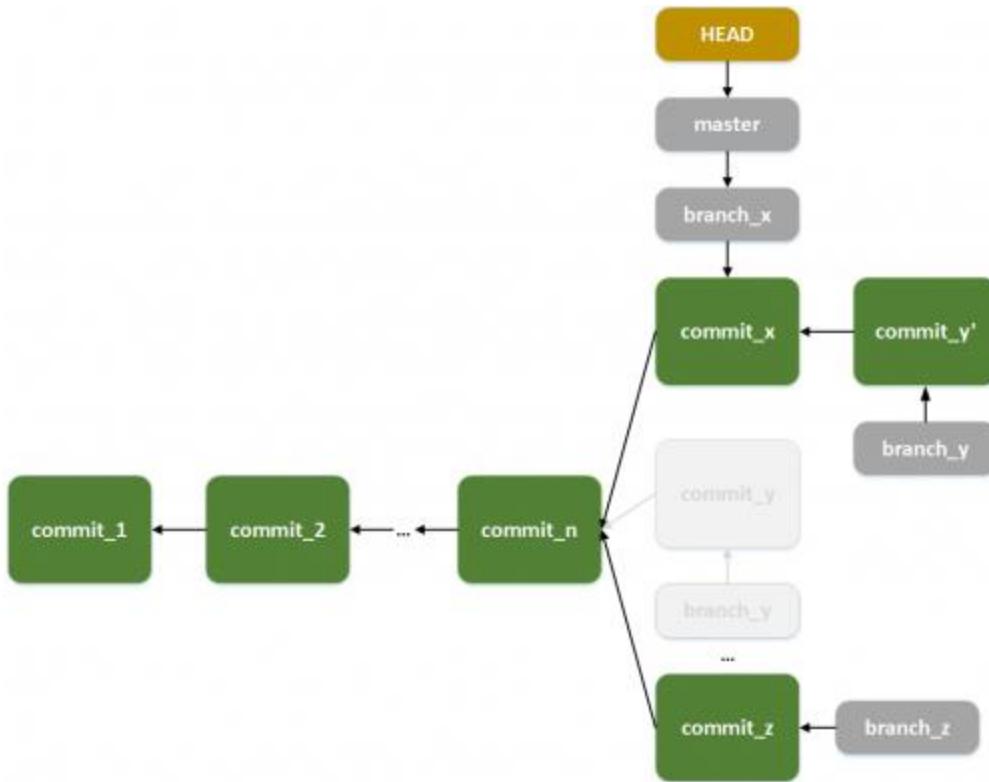
Există posibilitatea ca ambele ramificații să fi modificate aceeași resurse în același loc, ceea ce poate genera anumite conflicte de integrare. Într-o astfel de situație, procesul de creare al consemnării este opus, iar fișierele în care sunt detectate astfel de anomalii nu vor fi marcate în zona de aşteptare pentru a fi incluse în versiunea ce integrează cele două ramificații. Fișierele respective vor conține regiuni în care conflictele sunt delimitate prin conținutul versiunii master (între <<<<< HEAD și =====) și a versiunii care se dorește a fi integrată (între ===== și <<<<< <branch\_name>).

Rezolvarea conflictului poate fi realizată în două moduri, impunându-se însă realizarea consemnării propriu-zise după fiecare dintre acestea (cu un mesaj care explică în ce a constat rezolvarea conflictului, dacă nu este evident):

3. manual, înlocuind secțiunea de cod sursă marcată ca reprezentând sursa de conflict cu varianta care se dorește a fi regăsită în versiunea finală, urmată de transferul fișierului din directorul de lucru în zona de aşteptare prin `git add`;
4. folosind un utilitar vizual (indicat de variabila de configurare `merge.tool`)

### **Varianta REBASE**

Integrarea modificărilor realizate pe o ramificație în varianta REBASE se face prin comanda `git rebase master`, după ce toate modificările din cadrul acesteia au fost consemnate. În acest caz, se identifică modificările realizate în cadrul ramificației de la cel mai recent strămoș comun, acestea fiind aplicate versiunii indicate de ramificația `master` și introduse într-o consemnare care o va referi direct pe aceasta, ca părinte. După această operație, se poate trece înapoi pe ramificația `master`, integrându-se printr-o operație de tip fast-forward modificările din ramificația obținută (spre care va indica referința ramificației care a fost integrată).



```

student@eim2017:~$ git checkout customized_message
student@eim2017:~$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: added staged command
student@eim2017:~$ git checkout master
Switched to branch 'master'
student@eim2017:~$ git merge customized_message

```

Dacă nu se dorește ca inițial să se treacă pe ramificația care urmează a fi integrată, se poate rula comanda `git rebase <branch_to_integrate_in> <branch_to_be_integrated>`. Prin aceasta, se trece pe ramificația ce urmează a fi integrată, realizându-se apoi operația de integrare a acesteia în ramura de bază, în care se face integrarea.

Astfel,

```

student@eim2017:~$ git rebase master customized_message

```

este echivalentă cu

```

student@eim2017:~$ git checkout customized_message

```

```
student@eim2017:~$ git rebase master
```

În situația în care proiectul conține mai multe ramificații de dezvoltare în paralel, dorindu-se doar integrarea unor și omiterea (pe moment) a altora, se poate folosi comanda `git rebase --onto <branch_to_integrate_in> <branch_to_be_omitted> <branch_to_be_included>`. Prin intermediul acestei comenzi, se identifică cel mai recent strămoș comun între ramificația care trebuie omisă și cea care trebuie inclusă, aplicându-se modificările realizate din acest punct asupra ramurii pe care se realizează integrarea. Varianta REBASE se folosește de obicei atunci când se realizează contribuții pe depozite la distanță, astfel încât istoricul acestora să nu devină mai complex prin introducerea de ramificații suplimentare, existente doar în directorul de lucru local, integrarea constând într-o simplă operație de tip fast-forward.

Varianta REBASE nu trebuie folosită asupra versiunilor care se găsesc în depozite rezidente pe servere la distanță încât referințele către acestea nu vor mai exista, făcând dificilă integrarea modificărilor realizate de alți utilizatori care se bazau pe ea. De aceea, ea reprezintă o soluție doar pentru acele ramificații care nu au fost făcute publice niciodată.

## Distrugerea unei ramificații

Ștergerea unei ramificații este necesară în momentul în care aceasta referă o consemnare spre care mai există și alte referințe, ale unor ramificații în care au fost integrate (de regulă `master`). O astfel de operație poate fi realizată prin comanda `git branch -d <branch_name>`.

```
student@eim2017:~$ git branch -d customized_message
Deleted branch customized_message (was 477dcd6).

student@eim2017:~$ git branch -d fade_effect
Deleted branch fade_effect (was 30fe2f3).
```

## Lucrul cu ramificații la distanță

O ramificație la distanță este o referință către starea unei ramificații aflată într-un depozit găzduit de un server la distanță. Ele sunt stocate în directorul local, fiind descărcate în mod automat în momentul în care se realizează comunicații prin rețea. Nu se pot realiza nici un fel de operații asupra lor, cu excepția consultării, pentru a se realiza mai ușor comparația față de directorul de lucru de pe discul local.

Modul în care sunt referite ramificațiile la distanță sunt `<remote-name> / <branch-name>`. Spre exemplu, atunci când se clonează un proiect, se reține pe discul local o referință către `origin/master` în mod automat.

În momentul în care se rulează comanda `git fetch`, sunt descărcate toate consemnările din depozitul la distanță, actualizându-se în mod corespunzător referințele către ramificații, aşa cum se găsesc acolo.

Git nu realizează automat integrarea modificărilor din directorul Git local cu conținutul ramificațiilor la distanță, fiind necesar ca operația să fie realizată manual prin `git merge <remote-name>/<branch-name>` urmată de încărcarea versunii obținute pe server, după rezolvarea eventualelor conflicte.

Dacă se dorește crearea unei ramificații locale pe baza conținutului rezident pe serverul la distanță, se poate rula comanda `git checkout -b <local_branch>` cu forma `git checkout --track <remote_name>/<branch_name>`, dacă denumirile ramificațiilor local și la distanță corespund). Aceste obiecte poartă denumirea de **ramificații de monitorizare**, iar operațiile `git push` și `git pull` încarcă / descarcă informații în/din ramificația corespunzătoare de pe serverul la distanță în mod automat, fără a mai avea nevoie de alți parametri.

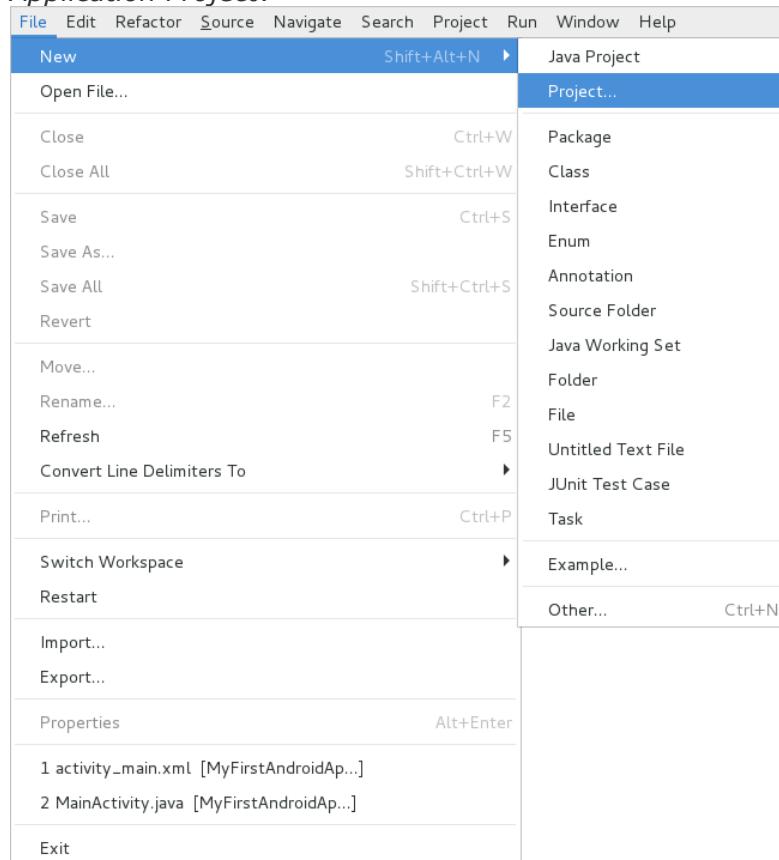
De regulă, ramificațiile create în directorul de lucru de pe discul local nu sunt vizibile la distanță. Dacă se dorește totuși să se partajeze o astfel de informație, se va rula comanda `git push <remote_name> <local_branch>[:<remote_branch>]`. În acest mod, modificările din ramificația `local_branch` din directorul de lucru de pe discul local vor fi încărcate pe ramificația `remote_branch` din depozitul de pe serverul la distanță (cu condiția să existe drepturi de acces pe server).

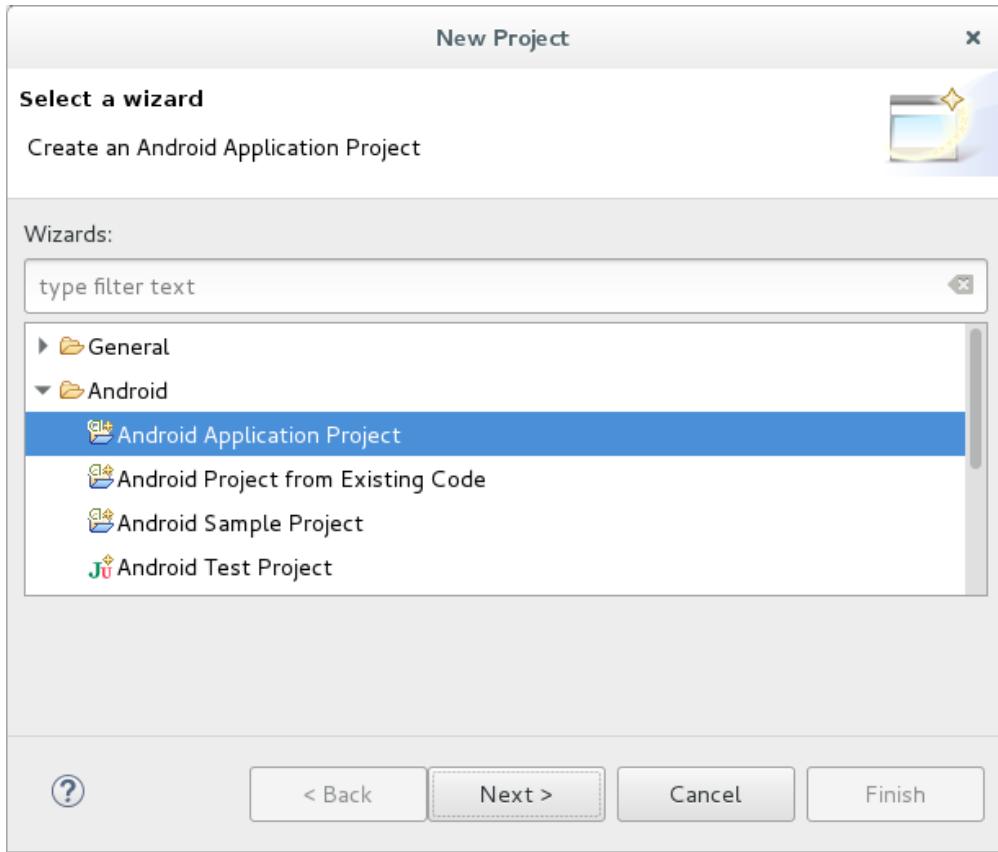
În situația în care o ramificație de pe serverul la distanță nu mai este necesară (modificările sale au fost integrate în altă ramificație de pe serverul la distanță, de exemplu `master`), se poate rula comanda `git push <remote_name> :<remote_branch>`.

În cazul în care denumirea ramificației din depozitul de pe serverul la distanță coincide cu ramificația din directorul de lucru de pe discul local, comanda ia forma `git push <remote_name> <branch_name>`.

## Crearea unei aplicații Android în Eclipse - obligatoriu

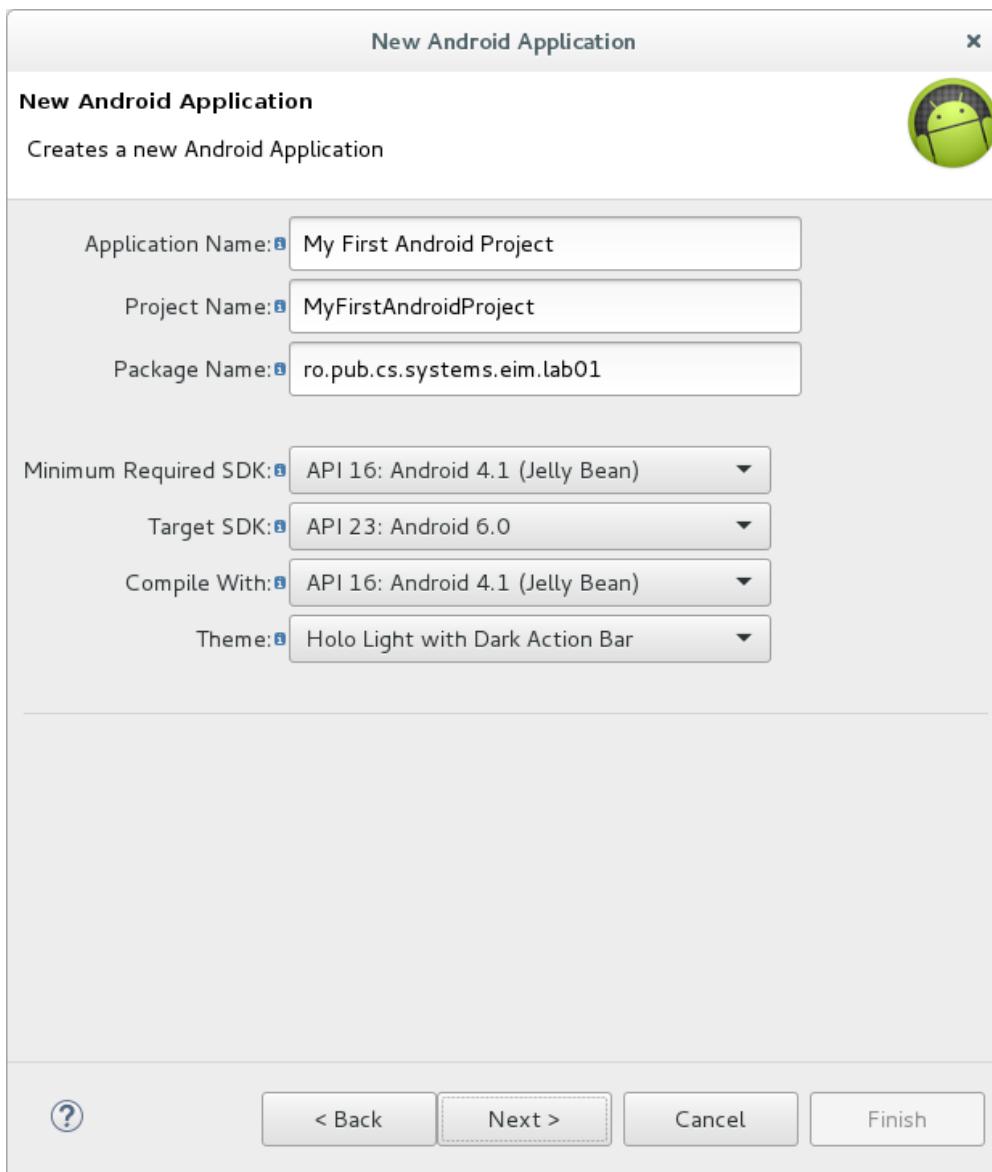
Pentru a crea o aplicație Android se selectează `File → New → Project`, iar apoi *Android Application Project*.





În cazul în care anterior au fost create alte aplicații Android, proiectele de acest tip vor putea fi create accesând scurtătura *File → New → Android Application Project*.

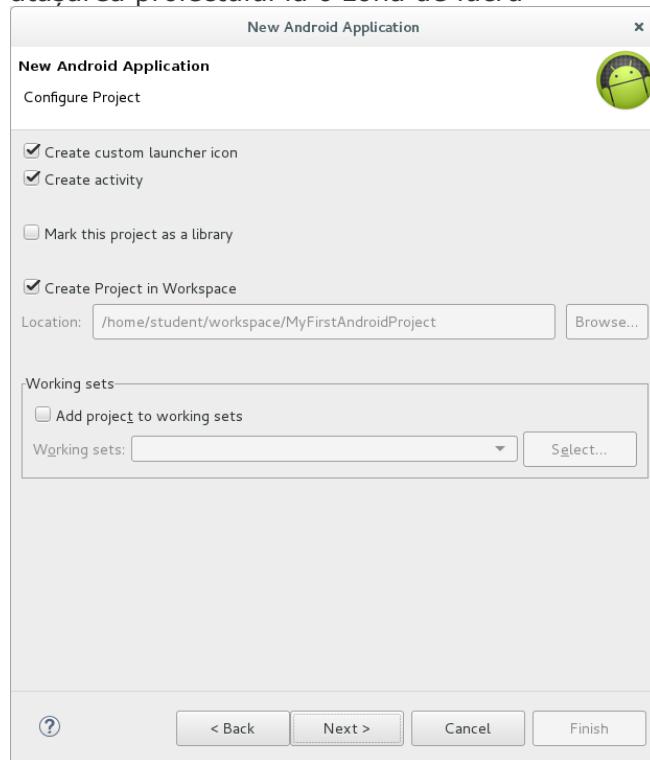
Ulterior, se vor specifica detaliile proiectului:



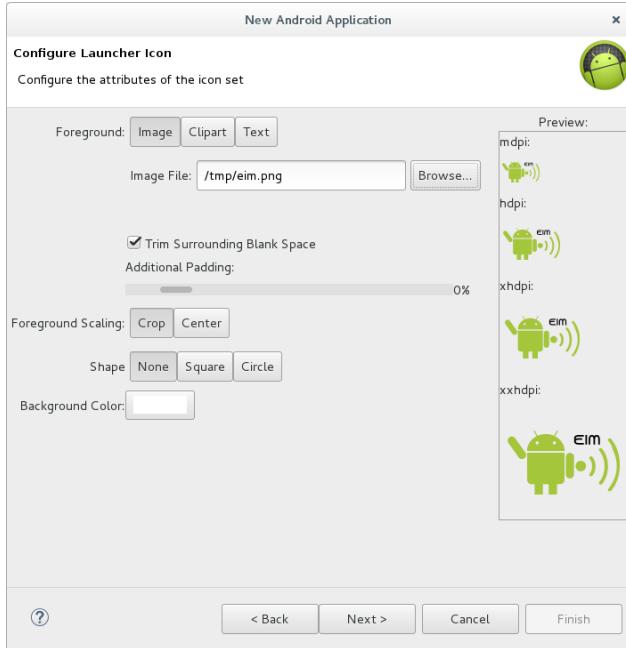
- **Application Name** - denumirea aplicației, aşa cum apare în Play Store și în *Settings → Application Manager* (de obicei este același ca Project Name)
- **Project Name** - denumirea proiectului Eclipse, folosind convențiile utilizate în cadrul acestui IDE (nu mai este folosită în altă parte, nu sunt permise spații)
- **Package Name** - denumirea pachetului care va conține aplicația, aceasta identificând-o în mod unic; utilizatorii nu o pot accesa, însă trebuie păstrată aceeași valoare pe întreaga durată de viață a programului, acesta fiind mecanismul prin care se stabilește că diferite versiuni aparțin aceleiași aplicații; drept convenție, se folosește denumirea domeniului organizației urmat de unul sau mai mulți identificatori ai aplicației, alcătuind o denumire validă pentru un pachet Java (trebuie să conțină cel puțin un caracter '.')

Este de preferat să se scrie domeniul organizației în ordine inversă.  
Exemplu: ro.pub.cs.systems.eim.lab01.

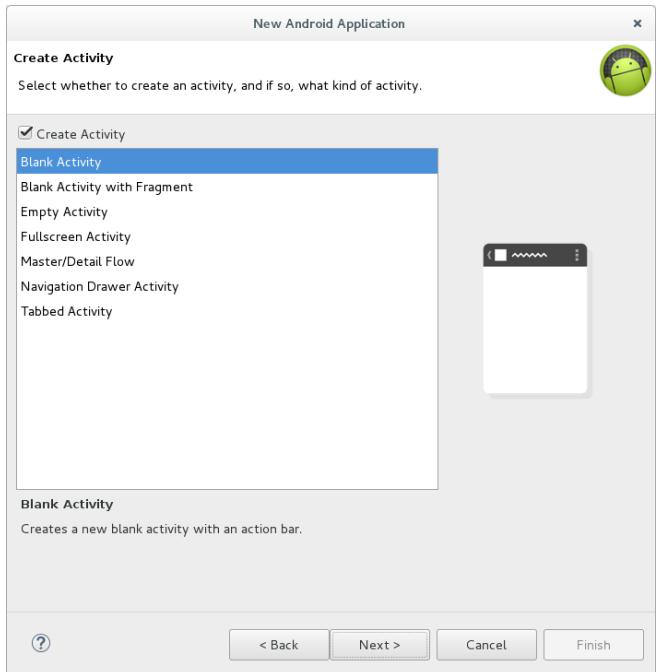
- **Minimum Required SDK** - versiunea pe care se dorește să se ruleze aplicația; cu cât va fi aleasă o valoare mai mică, vor fi cuprinse mai multe platforme, însă vor putea fi utilizate mai puține funcționalități; pentru a include cât mai multe dispozitive mobile, se va alege nivelul de API 8 (Android 2.2); în scop didactic, se folosește nivelul de API 16 (Android 4.1)
- **Target SDK** - nivelul de API cel mai mare pe care aplicația a fost testată, fără a fi întâmpinate nici un fel de probleme
- **Compile With** - nivelul de API folosit pentru compilarea codului sursă dintre SDK-urile instalate; de obicei, se utilizează cel mai recent API sau un API care suportă toate funcționalitățile care vor fi accesate în cadrul aplicației
- **Theme** - tema de bază care va fi utilizată în cadrul aplicației  
Procesul de configurare a proiectului implică și precizarea următoarelor informații:
  - necesitatea de a se crea o pictogramă implicită pentru aplicația aferentă proiectului
  - necesitatea de a se crea o activitate (corespunzătoare unei ferestre a aplicației aferente proiectului)
  - caracterul de bibliotecă al proiectului
  - locația la care va fi plasat proiectul (în spațiul de lucru sau nu)
  - atașarea proiectului la o zonă de lucru



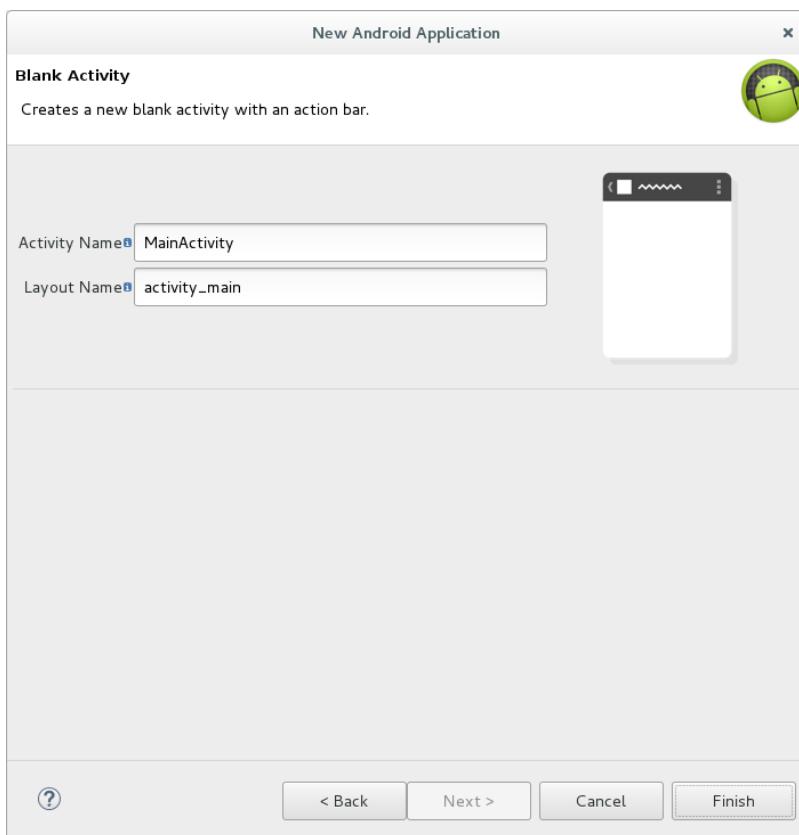
Se va indica și pictograma aplicației prin care aceasta va putea fi accesată din meniul dispozitivului mobil. Pentru aceasta se va specifica tipul (imagine, text, anumătie), locația de la care va fi încărcată, dacă se dorește adăugarea sau ștergerea de spațiu gol, modul în care se va realiza scalarea pentru toate rezoluțiile (mp1, hdpi, xhdpi, xxhdpi), forma (fără formă, pătrat sau cerc), culoarea de fundal.



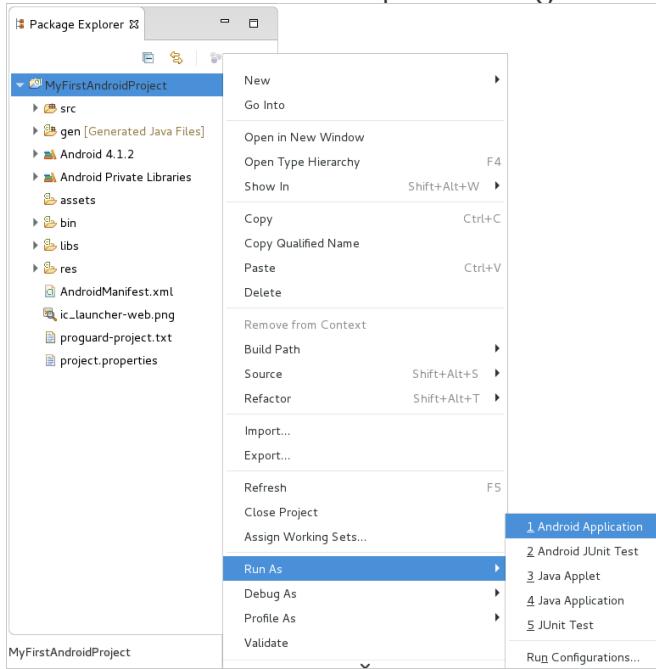
De asemenea, se specifică dacă se va crea o activitate (și ce tip va avea aceasta - *Blank Activity*, *Blank Activity with Fragment*, *Empty Activity*, *Full Screen Activity*, *Master/Detail Flow*, *Navigation Drawer Activity*, *Tabbed Activity*) sau nu. În Android, o activitate este o fereastră care conține interfața cu utilizatorul a aplicațiilor. O aplicație poate avea zero sau mai multe activități. De asemenea, o activitate poate fi împărțită în mai multe subunități denumite fragmente, utile pentru gestiunea spațiului, în special în situația în care aplicația este proiectată pentru dispozitive cu dimensiuni și rezoluții diferite ale ecranului.



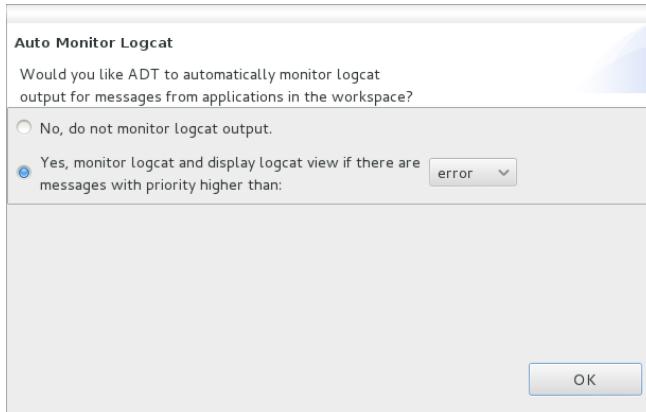
Activitatea va fi identificată printr-o denumire, precizându-se un nume și pentru fișierul xml care conține modul de dispunere al elementelor grafice (acesta putând fi editat atât în mod text cât și în mod vizual).



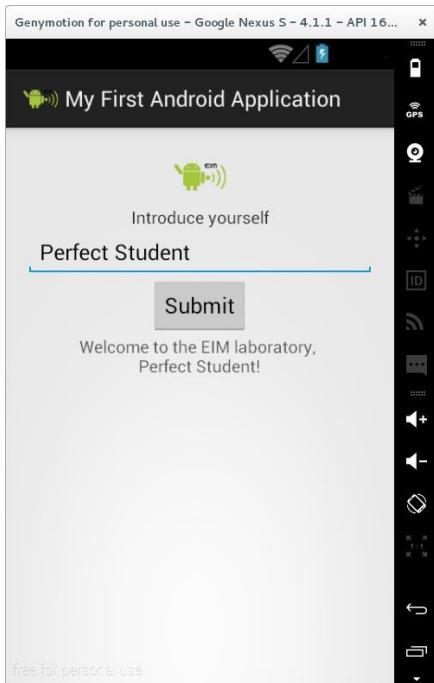
Rularea aplicației se face prin accesarea opțiunii *Run As → Android Application* din cadrul meniului contextual asociat proiectului (și accesibil prin click dreapta pe mouse).



Utilizatorul va putea specifica dacă dorește monitorizarea mesajelor transmise de aplicație prin intermediul logcat precum și nivelul de prioritate al acestora (informație, avertisment, eroare).



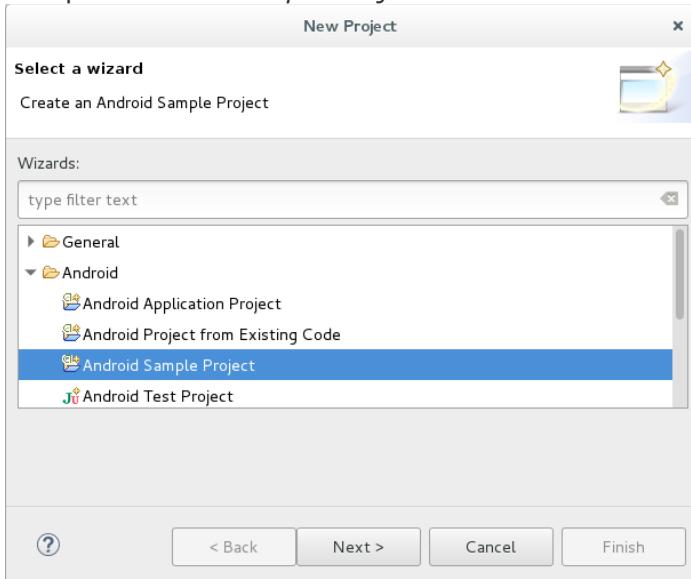
În prealabil, trebuie conectat un dispozitiv fizic prin cablu USB (sau prin wi-fi, în aceeași rețea cu calculatorul) sau se pornește un emulator pe care se va testa aplicația.



## Testarea exemplelor de aplicații Android în Eclipse - obligatoriu

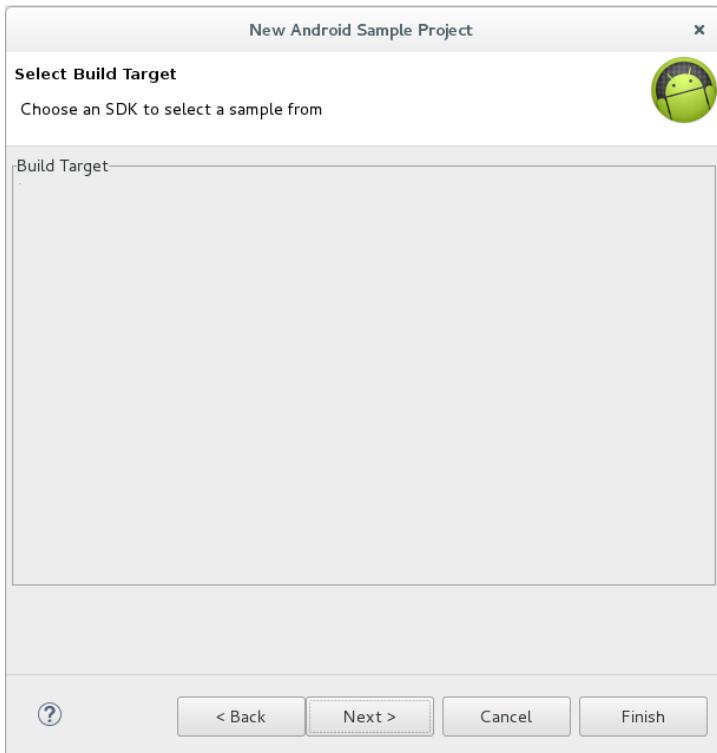
Fiecare versiune de Android este distribuită cu mai multe exemple de aplicații, ce ilustrează modul în care pot fi utilizate diferite funcționalități implementate de API.

Pentru a accesa proiectul corespunzător unui exemplu, se selectează *File → New → Project*, iar apoi *Android Sample Project*.

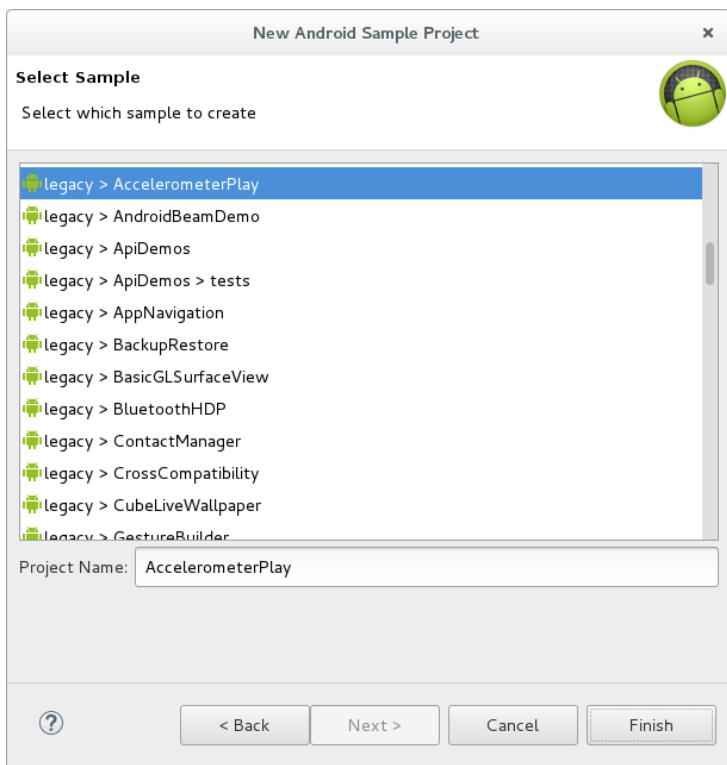


Va fi solicitat nivelul de API pentru care se dorește consultarea colecției de aplicații, precum și furnizorul acestora (parte a AOSP - Android Open Source Project sau Google) și platforma

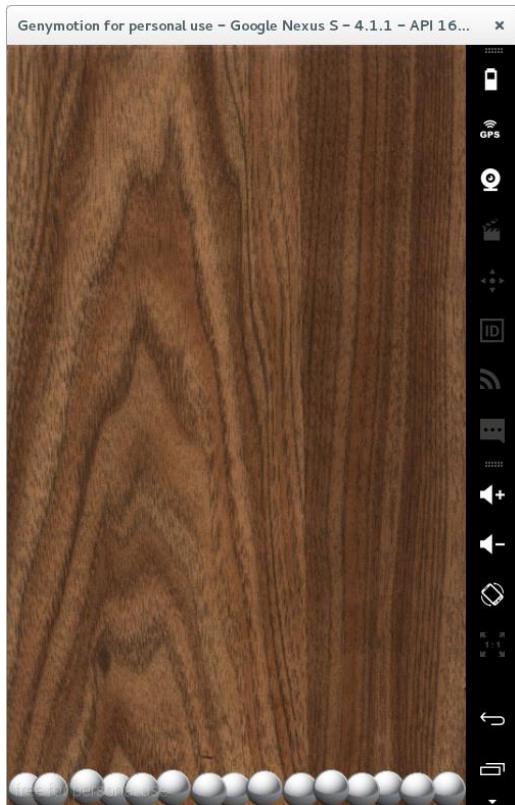
pe care va fi rulat. Pentru unele platforme, aceste informații nu mai sunt afișate, putând fi vizualizate exemplele de proiecte pentru toate nivelurile de API.



Din setul de aplicații disponibile va fi selectată una, pentru care va putea fi consultat și codul sursă, cu posibilitatea modificării acestuia.

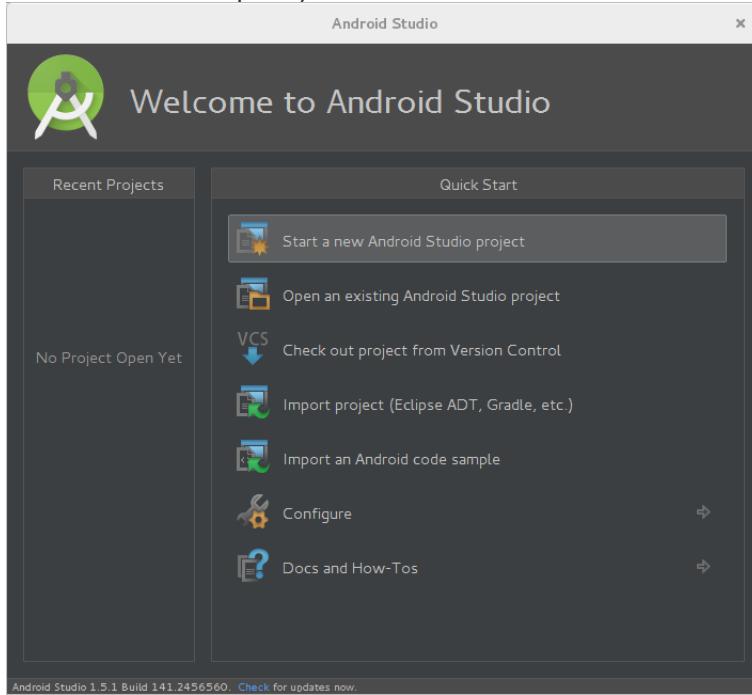


Rularea proiectului Android se face ca pentru orice resursă de acest tip.



# Crearea unei aplicații Android în Android Studio - obligatoriu

Pentru a crea o aplicație Android se selectează *Start a New Android Studio project*.

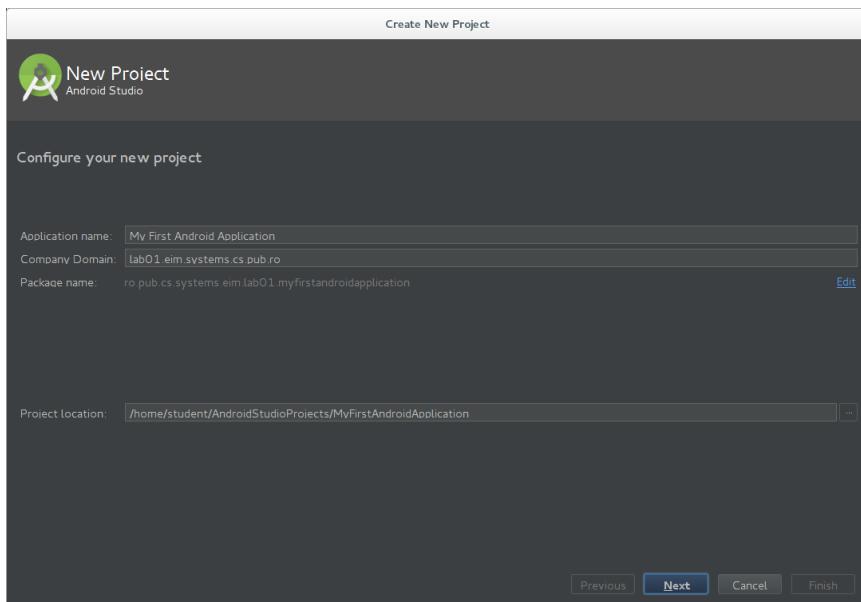


Portarea proiectelor dezvoltate folosind Eclipse ADT este foarte facilă, prin intermediul opțiunii *Import project (Eclipse ADT, Gradle, etc.)*.

Configurarea proiectului presupune specificarea unor parametri:

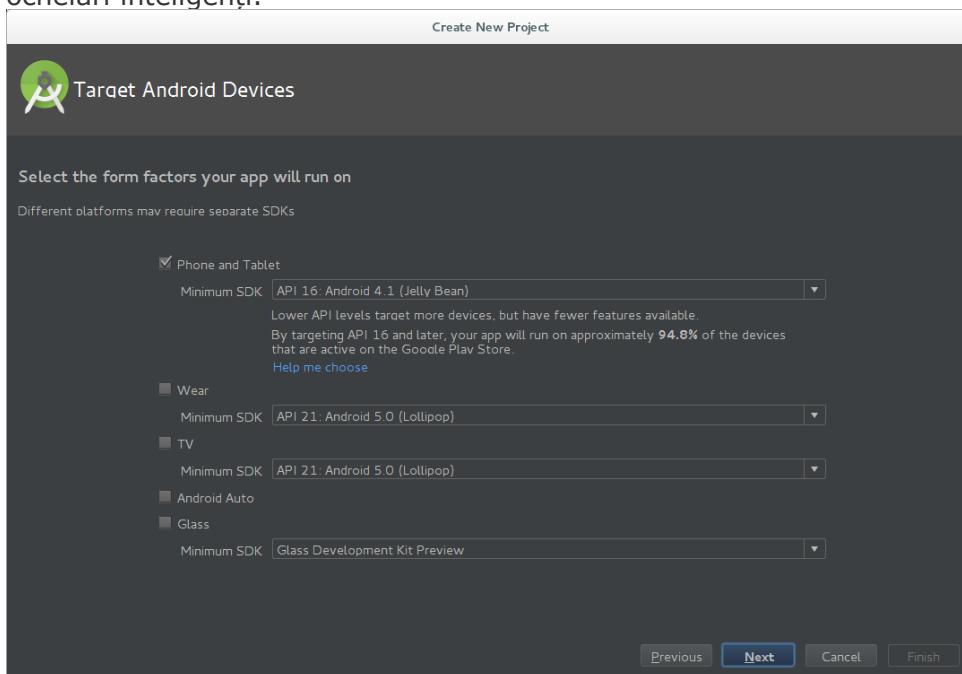
- denumirea aplicației;
  - domeniul companiei care dezvoltă aplicația respectivă.
- Pe baza valorilor introduse, se generează denumirea pachetului care va identifica în mod unic aplicația. Acesta este format din domeniu (scris în formă inversă) la care se adaugă denumirea aplicației (din care sunt eliminate caracterele albe).

De asemenea, este necesar să se indice locația la care va fi plasat proiectul respectiv.

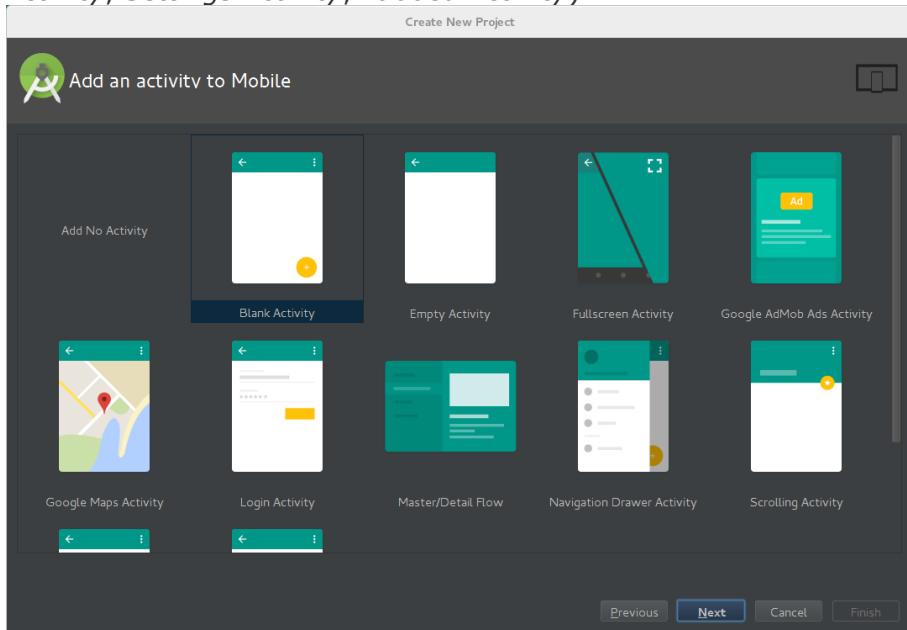


Se indică platforma căreia îi este destinată aplicația Android:

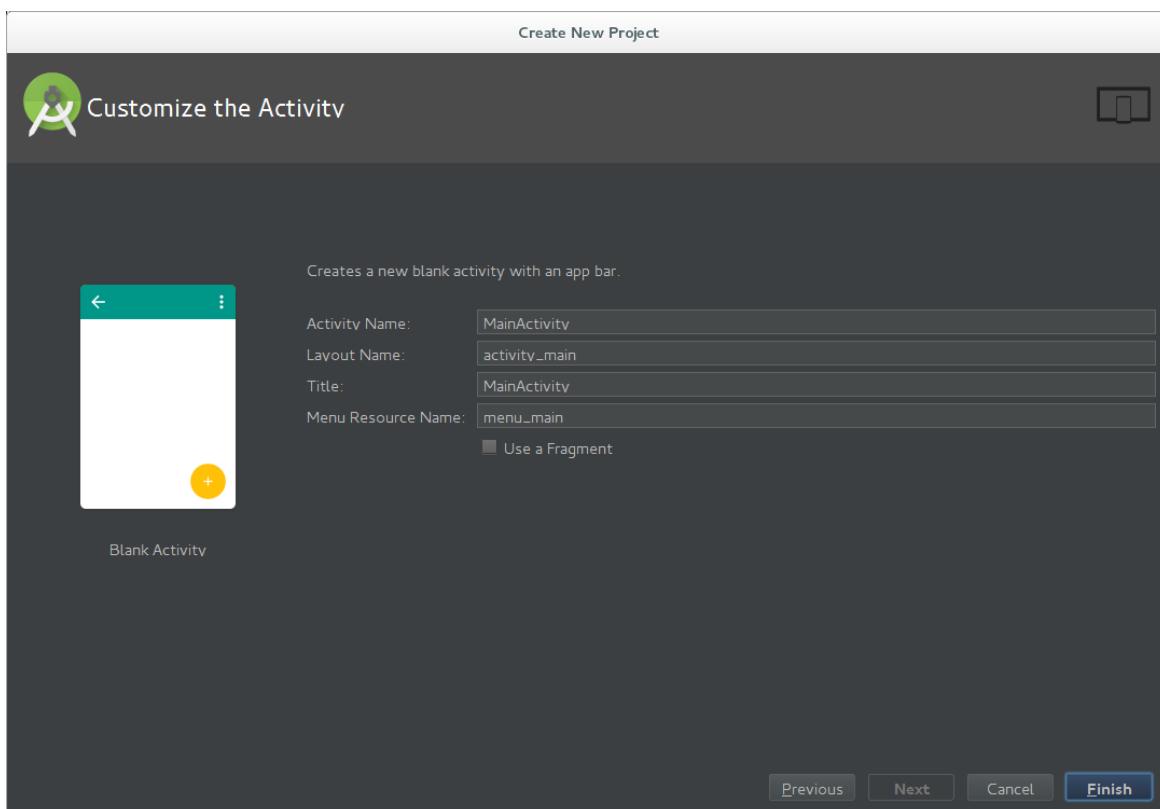
- dispozitiv mobil (telefon sau tabletă), caz în care trebuie să se precizeze valoarea minimă a nivelului de API pentru care se dezvoltă aplicația (cea mai scăzută valoare a platformei pe care poate rula aplicația);
- obiect vestimentar (ceas, brățără);
- televizor;
- dispozitiv pentru automobil;
- ochelari inteligenți.



Este selectat tipul de activitate care va fi vizualizată în momentul în care aplicația Android este pornită (*Blank Activity*, *Empty Activity*, *Fullscreen Activity*, *Google AdMob Activity*, *Google Maps Activity*, *Login Activity*, *Master/Detail Flow*, *Navigation Drawer Activity*, *Scrolling Activity*, *Settings Activity*, *Tabbed Activity*).

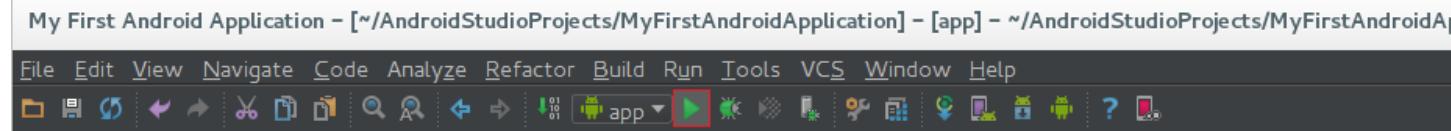


Se stabilește o denumire pentru activitatea principală, precum și denumirea fișierelor în care vor fi plasate structura interfeței grafice, respectiv a meniului asociat. Se poate preciza, de asemenea, folosirea de fragmente în cadrul activității.

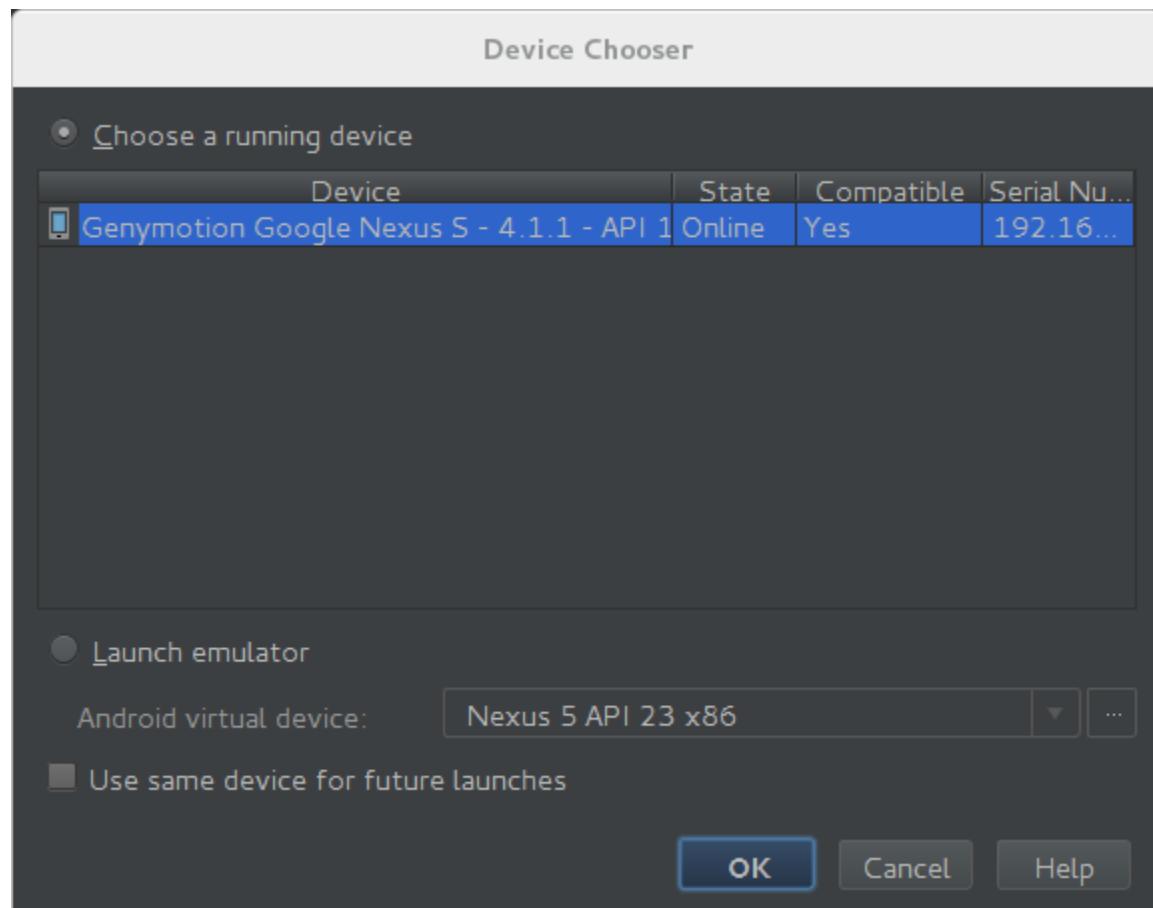


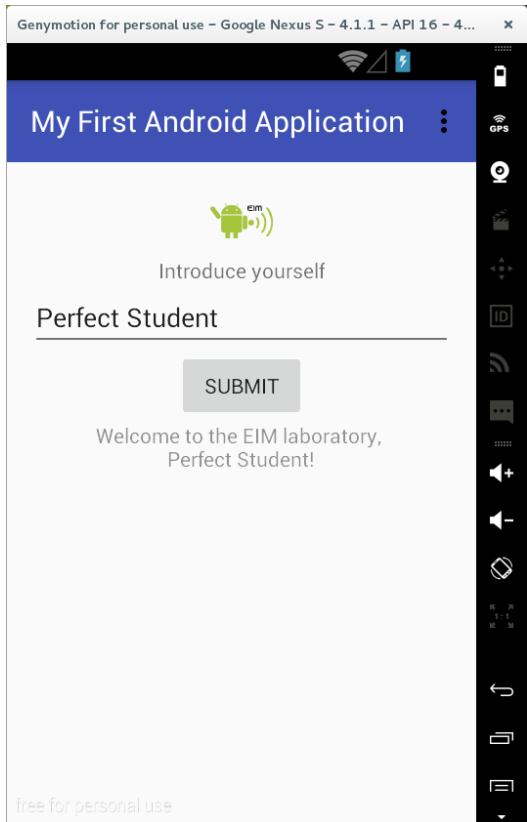
Se apasă butonul *Finish*.

Rularea unei aplicații Android se face prin intermediul unei pictograme asociate unei liste de selecție, în care sunt plasate toate aplicațiile disponibile în cadrul mediului integrat de dezvoltare. Se poate folosi și prescurtarea Shift + F10.



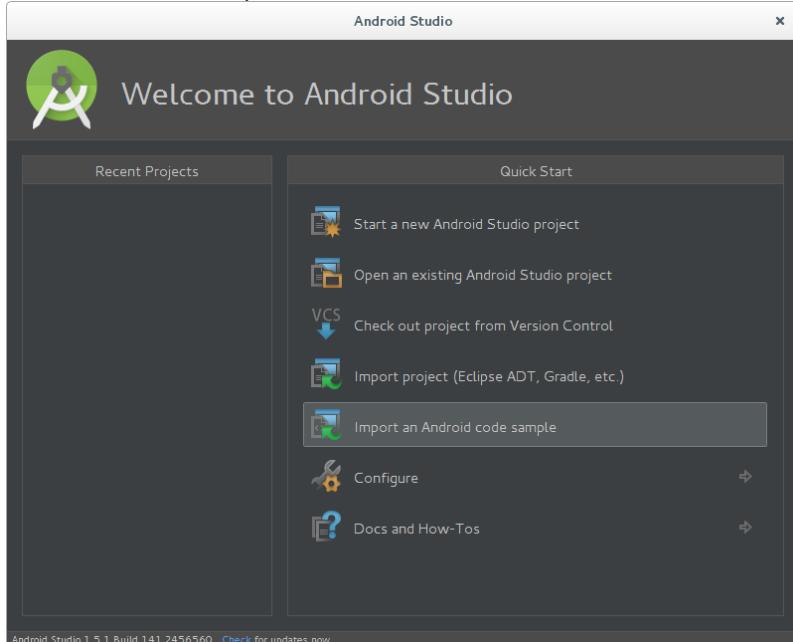
Utilizatorul are posibilitatea de a selecta dispozitivul pe care va fi rulată aplicația, dintre cele conectate la mașina pe care se rulează.



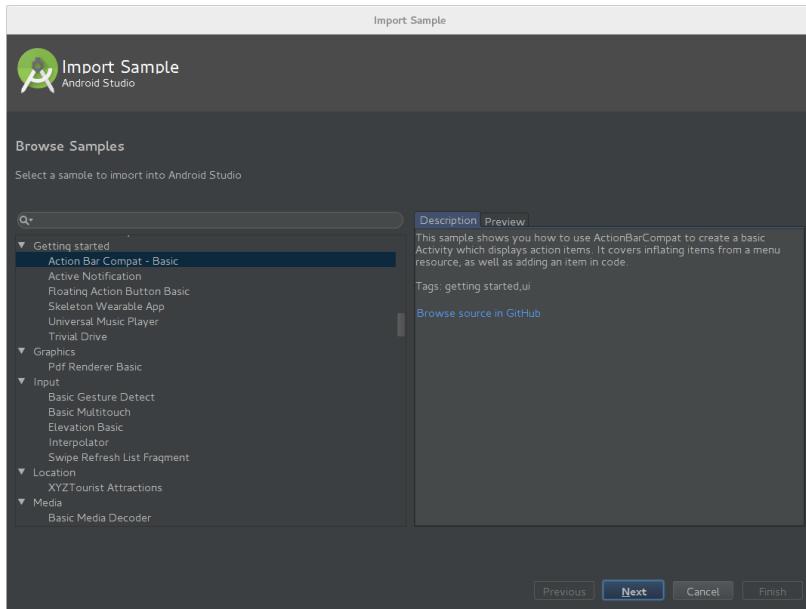


## Testarea exemplelor de aplicații Android în Android Studio - obligatoriu

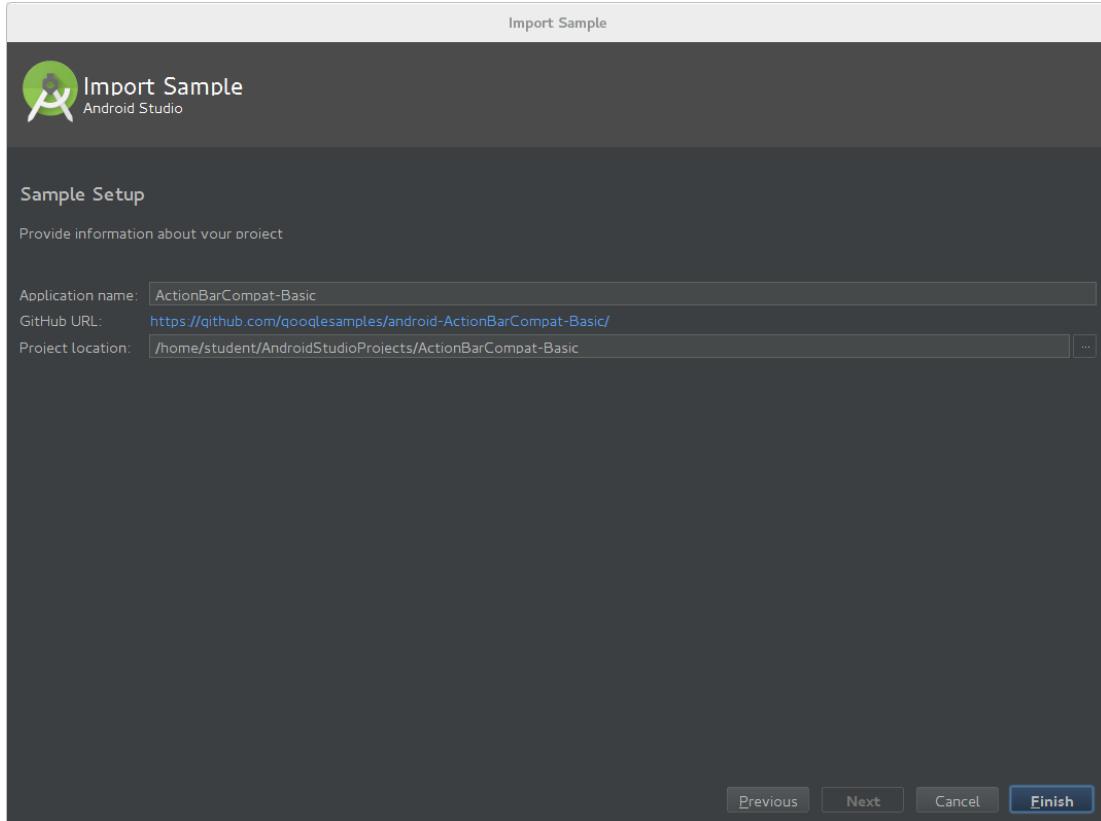
Accesarea unui exemplu de aplicație Android se face prin intermediul opțiunii *Import an Android code sample*.

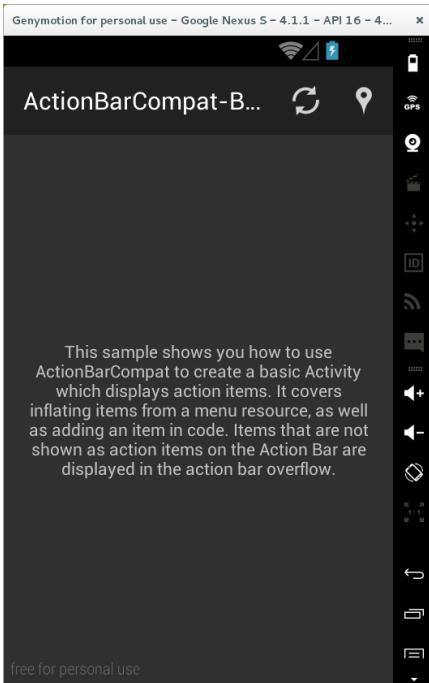


Pot fi consultate mai multe proiecte Android, pentru fiecare existând posibilitatea de a vizualiza codul sursă, disponibil în cadrul unui depozit GitHub. Gruparea aplicațiilor Android este realizată pe categorii tematice.



Un exemplu de aplicație Android, care se descarcă din contextul depozitului Github, poate fi redenumită, stabilindu-se și locația la care va fi stocată.





## Activitate de Laborator

---

1. Să se instaleze toate instrumentele necesare pentru a putea dezvolta o aplicație Android.
    - a. kit de dezvoltare pentru limbajul de programare Java;
    - b. SDK pentru Android;
    - c. mediu integrat de dezvoltare Eclipse sau Android Studio cu plugin-uri pentru Android, Genymotion;
    - d. emulator Genymotion în care se configurează un dispozitiv virtual Nexus S - 4.1.1 - API 16 - 400x800;
  2. Să se instaleze Git, în cazul în care nu există deja.
  3. Să se acceseze [GitHub](#) și să se creeze un cont.
  4. Să se realizeze configurațiile globale, specificând informații precum `user.name`, `user.email`, `core.editor`, `merge.tool`. Verificați faptul că informațiile au fost introduse corect, prin două metode diferite.
5. `student@eim2017:~$ git config --global user.name "Perfect Student"`
  6. `student@eim2017:~$ git config --global user.email "perfect_student@cti.pub.ro"`
  7. `student@eim2017:~$ git config --global core.editor gedit`

```
student@eim2017:~$ git config --global merge.tool diff
```

Verificarea valorii pe care o au variabilele de configurare poate fi realizată în mai multe moduri:

```
student@eim2017:~$ git config --list

student@eim2017:~$ cat .git/config

student@eim2017:~$ git config user.name

student@eim2017:~$ git config user.email

student@eim2017:~$ git config core.editor

student@eim2017:~$ git config merge.tool
```

8. Să se creeze un depozit pe contul Github creat, denumit 'Laborator01'. Inițial, acesta trebuie să fie gol (nu trebuie să bifăți nici adăugarea unui fișier README.md, nici a fișierului .gitignore sau a fișierului LICENSE).
9. Să se cloneze în directorul de pe discul local conținutul depozitului la distanță de la <https://www.github.com/eim2017/Laborator01>. În urma acestei operații, directorul Laborator01 va trebui să se conțină un director labtaks care conține proiectele Eclipse și AndroidStudio denumite MyFirstAndroidApplication, fișierele README.md și LICENSE și un fișier .gitignore care indică tipurile de fișiere (extensiile) ignorate.

```
student@eim2017:~$ git clone https://www.github.com/eim2017/Laborator01.git
```

10. Să se încarce conținutul descărcat în cadrul depozitului 'Laborator01' de pe contul Github personal.

```
11. student@eim2017:~$ cd Laborator01
```

```
12. student@eim2017:~/Laborator01$ git remote add Laborator01_perfectstudent https://github.com/perfectstudent/Laborator01
```

```
student@eim2017:~/Laborator01$ git push Laborator01_perfectstudent master
```

13. Să se ruleze aplicația schelet:

- în Eclipse: *File → New → Project → Android Project from existing code* SAU *File → Import → Android → Existing Android Code Into Workspace* și se indică directorul `Laborator01/labtasks/eclipse`;
- în Android Studio: *Open an existing Android Studio project* și se indică directorul `Laborator01/labtasks/androidstudio`;

14. În fișierul `MainActivity.java` din pachetul `ro.pub.cs.systems.eim.lab01` (directorul `src`), să se modifice metoda `onClick` a clasei interne `ButtonClickListener` astfel încât:

- mesajul afișat să includă numele utilizatorului, aşa cum apare în widget-ul de tip `EditBox`;

```
greetingTextView.setText(greetingTextView.getText().toString().replace("xxx",
"\n"+userNameEditText.getText()));
```

- să se aplice un efect de fade, astfel încât mesajul afișat să dispară treptat în curs de `TRANSPARENCY_EFFECT_DURATION` milisecunde.

```
c. AlphaAnimation fadeEffect = new AlphaAnimation(1.0f, 0.0f);
d. fadeEffect.setDuration(TRANSPARENCY_EFFECT_DURATION);
e. fadeEffect.setFillAfter(true);
```

```
greetingTextView.setAnimation(fadeEffect);
```

15. Să se încarce modificările realizate în cadrul depozitului 'Laborator01' de pe contul Github personal, folosind un mesaj sugestiv.

```
16. student@eim2017:~/Laborator01$ git add
MyFirstAndroidApplication/src/ro/pub/cs/systems/eim/lab01.MainActivity.java
```

```
17. student@eim2017:~/Laborator01$ git commit -m "implemented functionality
for customized message and fade effect"
```

```
student@eim2017:~/Laborator01$ git push Laborator01_perfectstudent master
```

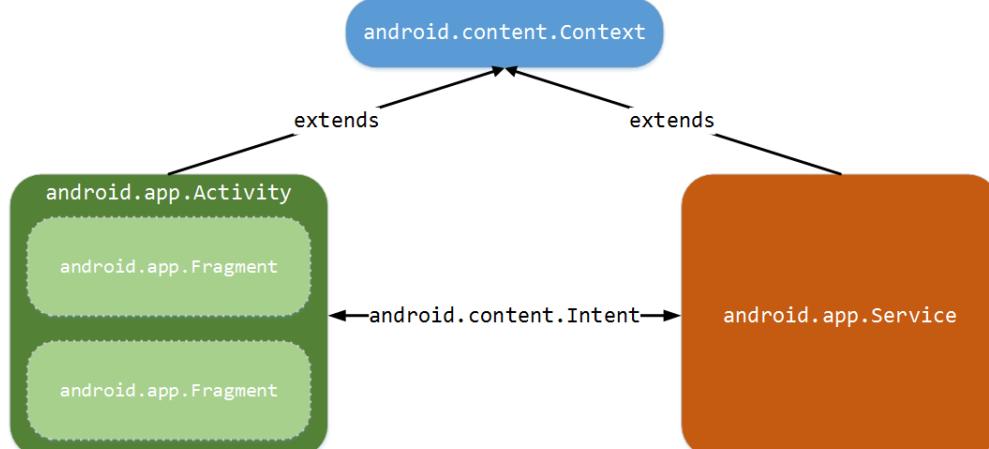
18. Să se ruleze un exemplu de proiect Android, dintre cele disponibile, folosind dispozitivul virtual instalat în cadrul emulatorului Genymotion. Să se simuleze un eveniment de tipul rotirea ecranului și să se observe modul în care se comportă aplicația.

- în Eclipse, `AccelerometerPlay` din cadrul exemplelor corespunzătoare nivelului de API 16 (legacy);
- în Android Studio, `Action Bar Compat - Basic`, din cadrul categoriei *Getting Started*.

## Laborator 02. Structura unei Aplicații (I)

O aplicație Android poate conține mai multe componente:

1. contextul reprezintă punctul central al unei aplicații Android, oferind acces către mai multe funcționalități ale acesteia (inclusiv la resursele dispozitivului mobil, serviciile sistemului de operare, diferite fișiere de configurație); este instantiat sub forma unui obiect de tip `android.app.Application`;
2. activitatea realizează sarcini a căror execuție nu influențează timpul de răspuns al aplicației Android, astfel încât să nu aibă un impact asupra experienței utilizatorului; de aceea, este asociată unei ferestre (interfețe grafice), o aplicație Android fiind formată din una sau mai multe activități;
3. fragmentul conține interfața grafică și logica aplicației corespunzătoare unei părți din cadrul unei activități; motivul pentru care se recurge la modularizarea unei activități prin intermediul a mai multor fragmente este asigurarea consistenței și flexibilității aplicației Android pe mai multe echipamente mobile, cu dispozitive de afișare de dimensiuni și rezoluții diferite;
4. serviciul încapsulează procese mai complexe, executate în fundal (și posibil la intervale de timp regulate) a căror rulare durează o perioadă de timp semnificativă, astfel încât să nu poată fi plasate în cadrul același fir de execuție ca și interfața grafică prin care se asigură interacțiunea cu utilizatorul;
5. intenția este mecanismul de comunicare între elementele unei aplicații Android (activități și servicii); prin intermediul unui sistem de mesagerie (asincronă), sistemul de operare Android mapează o solicitare (împachetată sub forma unei intenții) către componenta adecvată.



### Contextul

Contextul este utilizat pentru a implementa diferite funcționalități la nivelul întregii aplicații:

- obținerea de referințe la resursele aplicației (șiruri de caractere, elemente grafice, fișiere XML);
- accesarea preferințelor aplicației;
- gestiunea sistemului de fișiere corespunzător aplicației;
- lucrul cu resurse necompilate ale aplicației;
- utilizarea serviciilor de sistem;
- folosirea unei baze de date SQLite;
- administrarea permisiunilor aplicației.

În clasele de tip `Activity` sau `Service`, contextul aplicației poate fi obținut printr-un apel de forma:

```
Context context = getApplicationContext(); // definită în clasa Context
Context context = getApplication(); // definită în clasa Activity
```

Pentru a se evita utilizarea ineficientă a memoriei (eng. memory leak), în situația în care obiectele trebuie să existe pe toată durata aplicației, se recomandă ca acestea să refere contextul aplicației (nu al activității), astfel încât să nu fie condiționate de diverse evenimente din ciclul de viață al acestora.

## Activitatea

Activitatea reprezintă o componentă a aplicației Android ce oferă o interfață grafică cu care utilizatorul poate să interacționeze. Cele mai multe activități ocupă întreaga suprafață de afișare, însă acest lucru nu este obligatoriu.

O aplicație Android este formată din una sau mai multe activități (slab cuplate între ele). Există întotdeauna o activitate principală care este afișată atunci când aplicația Android este lansată în execuție inițial.

O activitate poate invoca o altă activitate pentru a realiza diferite sarcini, prin intermediul unei intenții. În acest moment, activitatea veche este opriță și salvată pe stivă (eng. back stack), după care este pornită activitatea nouă. Restaurarea și (re)începerea activității vechi este realizată în momentul în care activitatea nouă (currentă) este terminată. Un comportament similar are loc în momentul în care se produce un eveniment (primirea unui apel telefonic, apăsarea tastelor *Home* sau *Back*, lansarea altor aplicații).

O activitate trebuie să își gestioneze corespunzător comportamentul (inclusiv folosirea memoriei) în cazul producerii diferitelor evenimente, salvând și restaurând starea de fiecare dată când aplicația este opriță, respectiv (re)pornită.

## Componentele unei Activități

O activitate poate fi utilizată numai dacă este definită în fișierul `AndroidManifest.xml`, în cadrul elementului de tip `<application>`. Pentru fiecare activitate trebuie creată o intrare de tip `<activity>` pentru care se specifică diferite atribute, dintre care obligatoriu este numai denumirea activității (`android:name`). Din momentul în care aplicația Android este publicată, conținutul manifestului devine un contract față de utilizatori, iar denumirile componentelor nu mai pot fi modificate întrucât pot genera erori în cazul producerii unor actualizări.

Pentru o activitate se poate specifica și un filtru de intenții, în cadrul elementului `<intent-filter>`, spre a indica modul în care componentele aplicației o pot accesa. Aceasta este necesar pentru ca activitatea să poată fi rulată folosind intenții implicate (furnizate de alte aplicații). Așadar, o astfel de intrare va fi necesară pentru fiecare tip de intenție, precizând elementele `<action>` și optional `<category>` / `<data>`. În cazul activităților care nu vor fi accesibile din cadrul altor aplicații Android, nu este necesară definirea unui filtru de intenții. O activitate principală din cadrul unei aplicații Android este caracterizată prin următoarele proprietăți:

- acțiunea are valoarea `android.intent.action.MAIN`, întrucât reprezintă punctul de intrare al aplicației Android;
- categoria are valoarea `android.intent.category.LAUNCHER`, întrucât activitatea trebuie inclusă în meniu dispozitivului mobil pentru a putea fi lansată în execuție.

#### AndroidManifest.xml

```
<?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>
```

Ulterior, se va implementa și clasa identificată prin denumirea activității. Aceasta trebuie să fie derivată din `android.app.Activity` și să implementeze cel puțin metoda `onCreate()` în care sunt inițializate componente sale.

#### LifecycleMonitorActivity.java

```
public class LifecycleMonitorActivity extends Activity {

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

 // ...

}
```

Este obligatoriu ca metoda `onCreate()` să apeleze metoda părinte, în caz contrar generându-se o excepție.

Pentru fiecare activitate, este necesar să se descrie interfața grafică în cadrul unui fișier .xml (încărcat manual, în cadrul metodei `onCreate()`, printr-un apel al metodei `setContentView()`) în care elementul părinte este un mecanism de dispunere a conținutului (derivat din clasa `android.view.ViewGroup`). Acest fișier este plasat în directorul `res/layout` și conține referințe către toate obiectele care vor fi afișate în cadrul ferestrei. În urma compilării, vor fi generate niște referințe (adrese în cadrul pachetului de resurse) prin intermediul cărora resursele vor putea fi accesate.

#### activity\_lifecycle\_monitor.xml

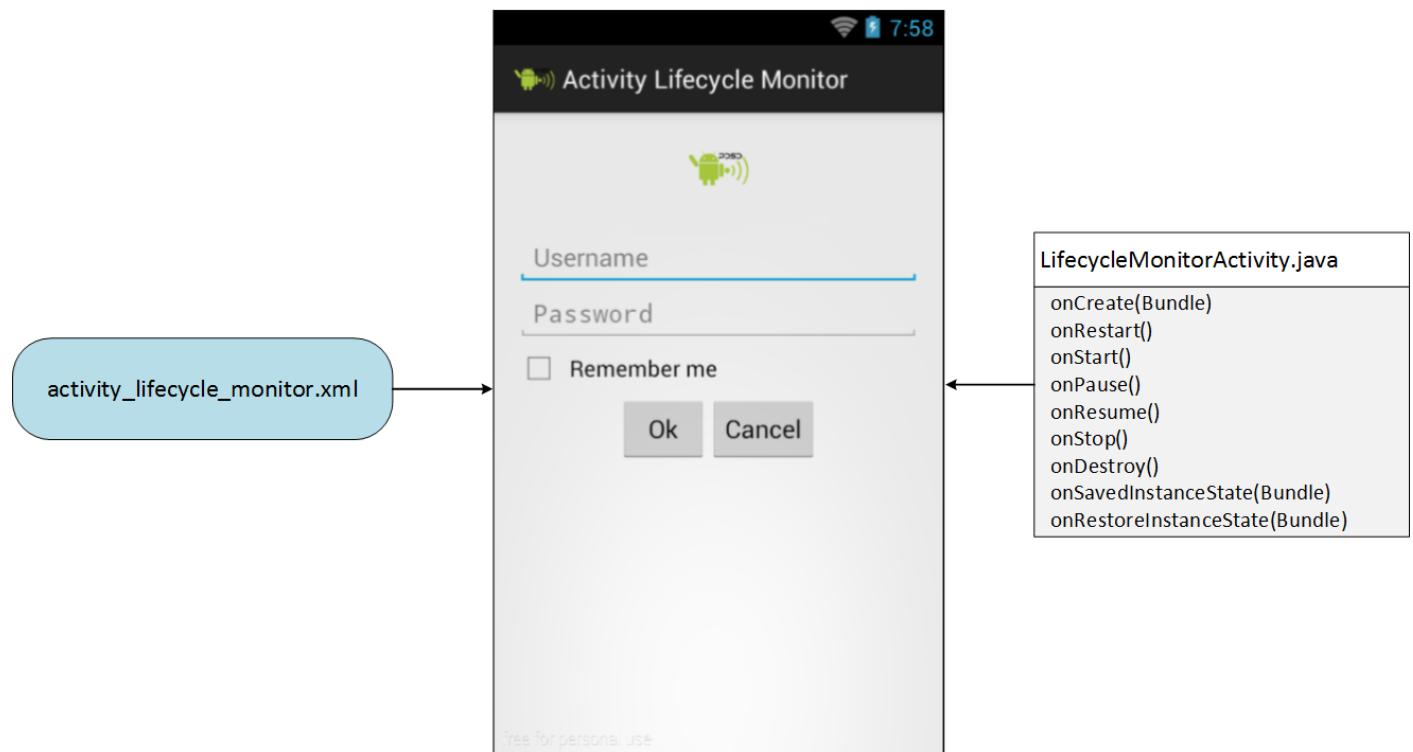
```
<LinearLayout
 xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:paddingBottom="@dimen/activity_vertical_margin"
 android:paddingLeft="@dimen/activity_horizontal_margin"
 android:paddingRight="@dimen/activity_horizontal_margin"
 android:paddingTop="@dimen/activity_vertical_margin"
 android:orientation="vertical"

 tools:context="ro.pub.cs.systems.eim.lab02.graphicuserinterface.Life
 cycleMonitorActivity" >

 <!-- ... -->

</LinearLayout>
```

Prin urmare, componentele definitorii ale unei activități sunt clasa în care este implementat comportamentul în urma interacțiunii cu utilizatorul și fișierul .xml care descrie modelul static al interfeței grafice.



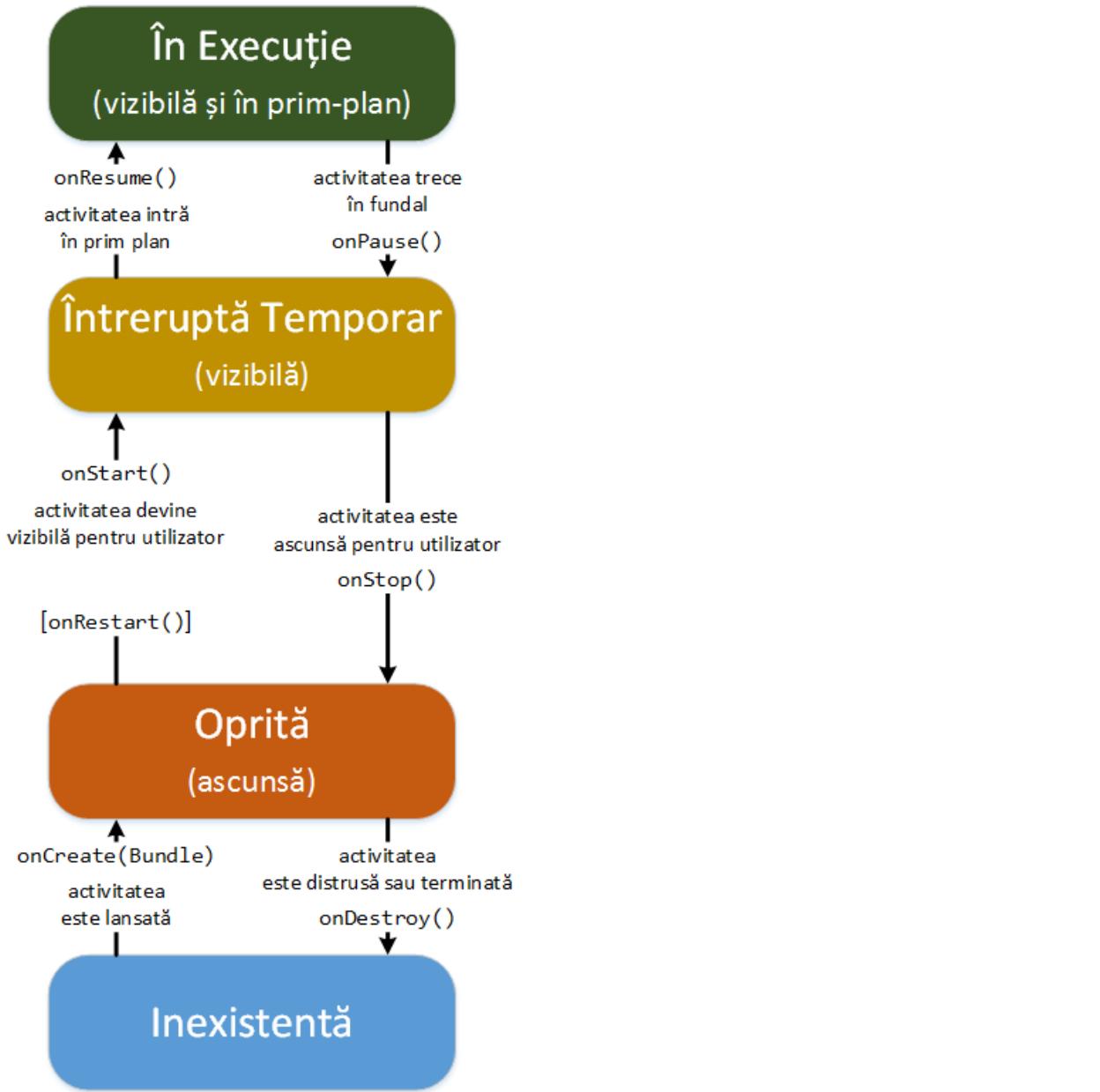
O activitate poate fi pornită în mod explicit prin intermediul metodei `startActivity()` care primește ca parametru un obiect de tip `Intent`ie (cărui i se pot atașa și niște date, transferate activității). În cazul în care se așteaptă ca activitatea să producă unele rezultate, se va apela metoda `startActivityForResult()` - furnizându-se și un cod de cerere -, fiind necesar ca activitatea astfel invocată să implementeze metoda `onActivityResult()`.

O activitate poate fi opriță în mod explicit prin intermediul metodei `finish()`. De regulă, folosirea unei astfel de metode este descurajată, datorită impactului pe care îl poate avea asupra experienței utilizatorului.

## Ciclul de Viață al unei Activități

Din momentul în care activitatea este creată și până la momentul în care este distrusă, ea trece printr-o serie de etape, cunoscute sub denumirea de **ciclul de viață al activității**:

- **în execuție** (eng. running) - activitatea se află în prim plan și este vizibilă, astfel încât utilizatorul poate interacționa cu aceasta prin intermediul interfeței grafice pe care o oferă;
- **întreruptă temporar** (eng. paused) - activitatea se află în fundal și este (parțial) vizibilă; o astfel de situație este întâlnită în momentul în care o altă activitate a fost pornită, însă interfața sa grafică este transparentă sau nu ocupă întreaga suprafață a dispozitivului de afișare; în acest caz, activitatea este încă activă în sensul că obiectul de tip `Activity` este stocat în memorie, fiind atașată în continuare procesului responsabil cu gestiunea ferestrelor și menținându-se starea tuturor componentelor sale; totuși, ea poate fi distrusă de sistemul de operare dacă necesarul de memorie disponibilă nu poate fi întrunit din cauza sa;
- **oprită** (eng. stopped) - activitatea se află în fundal și este complet ascunsă; o astfel de situație este întâlnită în momentul în care o altă activitate a fost pornită, iar interfața sa grafică ocupă întreaga suprafață a dispozitivului de afișare; și în acest caz, activitatea este activă în sensul că obiectul de tip `Activity` fiind stocat în memorie, menținându-se starea tuturor componentelor sale, dar detașându-se de procesul responsabil cu gestiunea ferestrelor; ea poate fi distrusă de sistemul de operare dacă necesarul de memorie disponibilă nu poate fi întrunit din cauza sa;
- **inexistentă** - activitatea a fost terminată sau distrusă de sistemul de operare, rularea sa impunând crearea tuturor componentelor sale ca și când ar fi accesată inițial.



Tranzitia unei activitati dintr-o stare in alta este notificata prin intermediul unor metode (eng. callbacks), care pot fi suprascrise pentru a realiza diferite operatii necesare pentru gestiunea memoriei, asigurarea persistentei informatiilor si a consistentei aplicatiei Android in situația producerii de diferite evenimente:

- `onCreate(Bundle)` - apelată în momentul în care activitatea este creată; această metodă va fi folosită pentru inițializări statice:
  - încărcarea interfeței grafice printr-un apel al metodei `setContentView(int)` (al cărei parametru reprezintă referința către resursa de tip `.xml` care descrie interfața grafică);

- obținerea de referințe către componentele interfeței grafice printr-un apel al metodei `findViewById(int)` (al cărei parametru reprezintă referința către componenta respectivă - eng. widget - aşa cum apare în resurse);
- indicarea unor obiecte de tip ascultător care să gestioneze evenimentele legate de interacțiunea cu utilizatorul;
- realizarea unor conexiuni către alte modele de date.

Crearea activității este diferită de apariția acesteia pe ecran.

Metoda `onCreate()` este întotdeauna urmată de metoda `onStart()`.

- `onRestart()` - apelată atunci când activitatea a fost oprită și ulterior repornită; este urmată întotdeauna de metoda `onStart()`;
- `onStart()` - apelată înainte ca activitatea să apară pe ecran; poate fi urmată de metoda `onResume()` dacă activitatea trece în prim-plan sau de metoda `onPause()` dacă activitatea trece în fundal;
- `onResume()` - apelată înainte ca activitatea să interacționeze cu utilizatorul; această metodă va fi folosită pentru a porni servicii sau cod care trebuie să ruleze atât timp cât aplicația este afișată pe ecran; este urmată întotdeauna de metoda `onPause()`;
- `onPause()` - apelată înainte ca activitatea să fie întreruptă temporar, iar o altă activitate să fie reluată; această metodă va fi utilizată pentru a opri servicii sau cod care nu trebuie să ruleze atât timp cât activitatea se află în fundal (întrucât consumă timp de procesor) și pentru a salva starea diferitelor componente în vederea asigurării persistenței și a consistenței aplicației înainte și după evenimentul care a produs suspendarea sa; poate fi urmată de metoda `onResume()` dacă activitatea trece în prim-plan sau de metoda `onStop()` dacă activitatea este ascunsă;

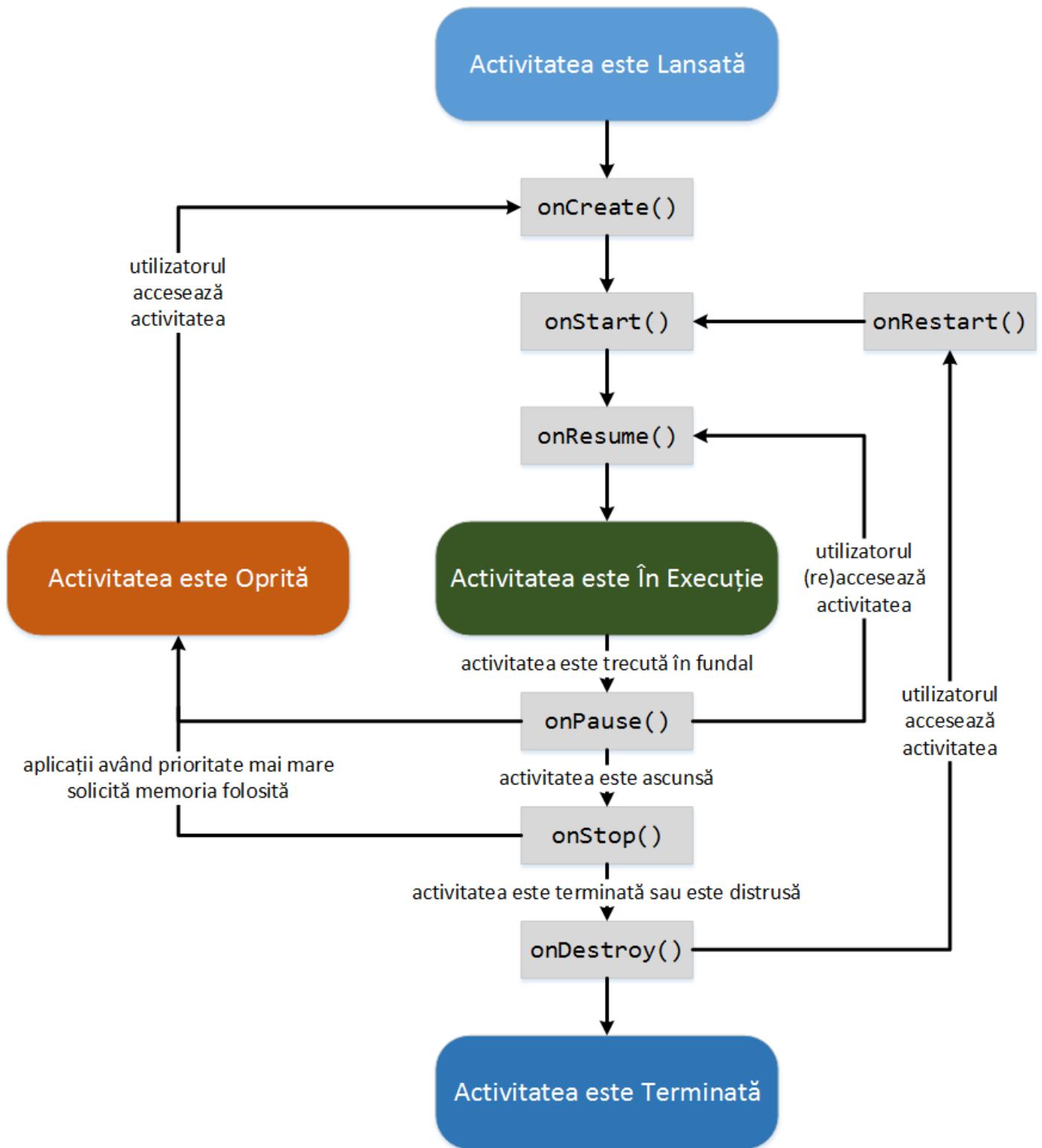
Procesările realizate în metoda `onPause()` nu trebuie să ocupe un interval de timp prea mare întrucât în caz contrar ar bloca noua activitate care urmează să fie lansată în execuție.

Procesul în contextul căruia rulează o activitate nu poate fi distrus de sistemul de operare până ce nu s-a terminat apelul metodei `onPause()`, aceasta fiind singura metodă care este apelată în mod garantat **înainte** de a se solicita memoria pe care activitatea o folosește.

- `onStop()` - apelată în momentul în care activitatea este ascunsă, fie din cauză că urmează să fie distrusă, fie din cauză că o altă activitate, a cărei interfață grafică ocupă întreaga suprafață a dispozitivului de afișare, urmează să devină vizibilă; poate fi urmată de metoda `onRestart()`, dacă activitatea urmează să interacționeze (din nou) cu utilizatorul, sau de metoda `onDestroy()` dacă activitatea urmează să fie terminată sau distrusă de sistemul de operare;

- `onDestroy()` - apelată înainte ca activitatea să se termine sau să fie distrusă de către sistemul de operare (fie manual, fie automat) din lipsă de memorie; această metodă va fi utilizată pentru a elibera resursele ocupate.

Distincția între cele două situații în care o activitate este distrusă poate fi realizată prin intermediul metodei `isFinishing()`.



Fiecare dintre metodele ce gestionează ciclul de viață al activității trebuie să apeleze metoda părintelui. În cazul metodei `onCreate()`, este necesar ca acest lucru să fie realizat la începutul metodei.

În cadrul unei activități, este interzisă apelarea manuală a metodelor legate de ciclul de viață, acestea fiind invocate în mod automat de sistemul de operare Android la producerea diferitelor evenimente.

#### LifecycleMonitorActivity.java

```
public class LifecycleMonitorActivity extends Activity {

 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 // ...
 }

 @Override
 protected void onStart() {
 super.onStart();
 // ...
 }

 @Override
 protected void onResume() {
 super.onResume();
 // ...
 }

 @Override
 protected void onPause() {
 super.onPause();
 // ...
 }

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

 @Override
 protected void onDestroy() {
 super.onDestroy();
 // ...
 }

 @Override
 protected void onRestart() {
 super.onRestart();
 // ...
 }

 // metode folosite pentru salvarea si restaurarea starii

 @Override
 protected void onSaveInstanceState(Bundle savedInstanceState) {
 // apelarea metodei din activitatea parinte este recomandata,
 // dar nu obligatorie
 super.onSaveInstanceState(savedInstanceState);
 }
}
```

```
// ...
}

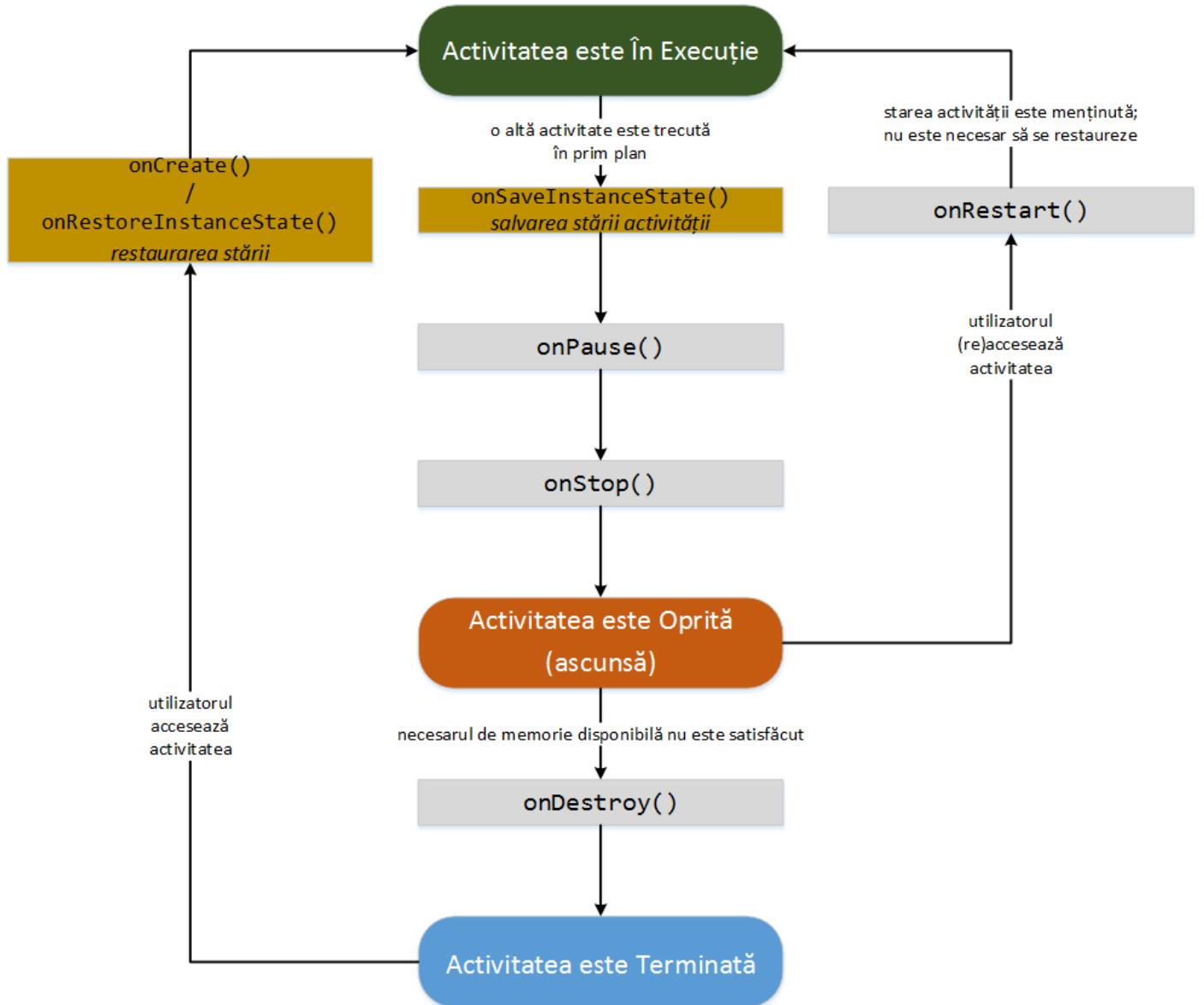
@Override
protected void onRestoreInstanceState(Bundle savedInstanceState)
{
 // apelarea metodei din activitatea parinte este recomandata,
 // dar nu obligatorie
 super.onRestoreInstanceState(savedInstanceState);
 // ...
}

}
```

## Gestiunea Stării unei Activități

Unele dintre metodele care gestionează ciclul de viață al unei activități primesc ca parametru un obiect de tip [Bundle](#), utilizat pentru gestiunea stării în cazul în care activitatea este distrusă din lipsă de memorie:

- `onCreate()` - parametrul `savedInstanceState` poate să fie `null`, dacă activitatea nu a mai fost rulată anterior, fie este o instanță a unui obiect de tip `Bundle` în care se găsește starea anterioară (așa cum a fost reținută de metoda `onSaveInstanceState()`);
- `onSaveInstanceState()` - este apelată când activitatea urmează să fie ascunsă și există posibilitatea ca procesul acesteia să fie terminat din lipsă de memorie, pentru a salva starea activității;
- `onRestoreInstanceState()` - este apelată doar dacă există o stare a activității care ar trebui să fie restaurată.



Obiectul de tip `Bundle` este o formă de hash map, în care cheia este întotdeauna de tip `String`, iar valorile au tipul `Parcelable` (asemănător cu `Serializable` din Java, deși acesta nu este un mecanism de serializare în sine). Astfel, tipurile de date care pot fi salvate și încărcate prin intermediu unui obiect de tip `Bundle` sunt:

boolean	boolean[]	
Bundle		
byte	byte[]	
char	char[]	
CharSequence	CharSequence[]	ArrayList<CharSequence>
double	double[]	
float	float[]	
int	int[]	ArrayList<Integer>

long	long[]	
Serializable		
short	short[]	
SparseArray		
String	String[]	ArrayList<String>

Starea unei activități este menținută atâtă vreme cât ea este activă (deci inclusiv când au fost apelate metodele `onPause()` și `onStop()`), aceasta putând fi restaurată corespunzător. Necesitatea asigurării consistenței activității de către programator apare în momentul în care activitatea este terminată/distrusă și apoi (re)pornită. O astfel de situație este frecventă în cazul în care se produce o schimbare de configurație (se modifică orientarea dispozitivului de afișare - portrait vs. landscape, limba, disponibilitatea tastaturii), întrucât de fiecare dată este necesar să se (re)încarce resursele specifice pentru valorile respective. Înținându-se cont de faptul că activitatea poate fi terminată/distrusă în orice moment după ce nu se mai găsește în prim-plan / este ascunsă, asigurarea consistenței trebuie realizată de fiecare dată.

## Salvarea Stării

Metoda `onSaveInstanceState()` este înainte de metoda `onStop()` și posibil înainte de metoda `onPause()`, deși acest lucru nu este garantat. Ea primește ca parametru un obiect de tip `Bundle` în care vor fi plasate datele din cadrul activității care se doresc a fi salvate, acestea putând fi identificate prin intermediul unei chei (de tip `String`).

Apelul metodei `onSaveInstanceState()` nu este garantat să fie realizat de fiecare dată întrucât pot fi situații în care nu este necesar ca starea activității să fie restaurată (utilizatorul a terminat activitatea prin apăsarea butonului *Back*).

În cazul în care se dorește salvarea explicită a conținutului unui obiect `EditText`, se poate proceda astfel:

```
@Override
protected void onSaveInstanceState(Bundle savedInstanceState) {
 super.onSaveInstanceState(savedInstanceState);
 EditText usernameEditText = (EditText) findViewById(R.id.username_edit_text);
 savedInstanceState.putString(Constants.USERNAME_EDIT_TEXT,
 usernameEditText.getText());
}
```

Valoarea introdusă de utilizator va fi salvată în obiectul de tip `Bundle` sub denumirea `userNameEditText`, acesta fiind menținut și prin urmare utilizat între mai multe instanțe ale acestei activități.

Apelarea metodei părinte este necesară întrucât API-ul Android furnizează o implementare implicită pentru salvarea stării unei activități, parcurgând ierarhia de componente grafice (obiecte de tip `View`) care au asociat un identificator (`android:id`), folosit drept cheie în obiectul `Bundle`. Astfel, de regulă, pentru elementele interfeței grafice, nu este necesar să se mențină starea, acest lucru fiind realizat în mod automat, cu respectarea condițiilor menționate. De aceea, în metoda `onSaveInstanceState`, va fi realizată salvarea unor informații (obiecte ale clasei) pentru care procesul de salvare a stării nu este apelat. Totuși, asigurarea persistenței datelor nu se va realiza niciodată aici (întrucât nu se garantează apelarea sa), ci în metoda `onPause()`.

În obiectul de tip `Bundle`, prin cheia `android:viewHierarchyState` se va reține un alt obiect de tip `Bundle` care reține starea tuturor obiectelor de tip `View` din cadrul activității (care sunt identificate prin intermediul unui câmp `android:id`). În cadrul acestuia, prin cheia `android:views` se reține un obiect de tip `SparseArray` (un tip de date specific Android care realizează mapări între întregi și obiecte, mai eficient decât un `hashmap`) care conține starea fiecărui obiect de tip `View` prin intermediul identificatorului său.

```
Bundle viewHierarchy
savedInstanceState.getBundle("android:viewHierarchyState");
if (viewHierarchy != null) {
 SparseArray<? extends Parcelable> views = viewHierarchy.getSparseParcelableArray("android:views");
 for (int k=0; k < views.size(); k++) {
 Log.d(Constants.TAG, views.get(k) + "->" + views.valueAt(k));
 }
}
else {
 Log.d(Constants.TAG, "No view information was saved!");
}
```

Prin urmare, dacă aplicația este opriță și apoi pornită din nou, obiectele de tip `View` vor avea conținutul existent în prealabil. Pentru a dezactiva această opțiune, trebuie specificată în fișierul `.xml` care descrie interfața grafică a activității, în elementul corespunzător obiectului de tip `View` proprietatea `android:saveEnabled="false"`.

Dezaktivarea se poate realiza și programatic, folosind metoda `setSaveEnabled(boolean)`.

## Restaurarea Stării

Încărcarea conținutului din obiectul de tip `Bundle` (în vederea restaurării stării) poate fi realizată:

1. În metoda `onCreate()`

2. @Override

```
3. protected void onCreate(Bundle savedInstanceState) {
4. super.onCreate(savedInstanceState);
5. setContentView(R.layout.activity_lifecycle_monitor);
6. EditText usernameEditText = (EditText) findViewById(R.id.username_edit_text);
7. if ((savedInstanceState != null) &&
8. (savedInstanceState.getString(Constants.USERNAME_EDIT_TEXT) != null)) {
9. usernameEditText.setText(bundle.getString(Constants.USERNAME_ED
```

```
 }
```

10. prin intermediul metodei `onRestoreInstanceState()`, apelată în mod automat între metodele `onStart()` și `onResume()`; această abordare permite separarea dintre codul folosit la crearea ferestrei și codul utilizat la restaurarea stării unei ferestre

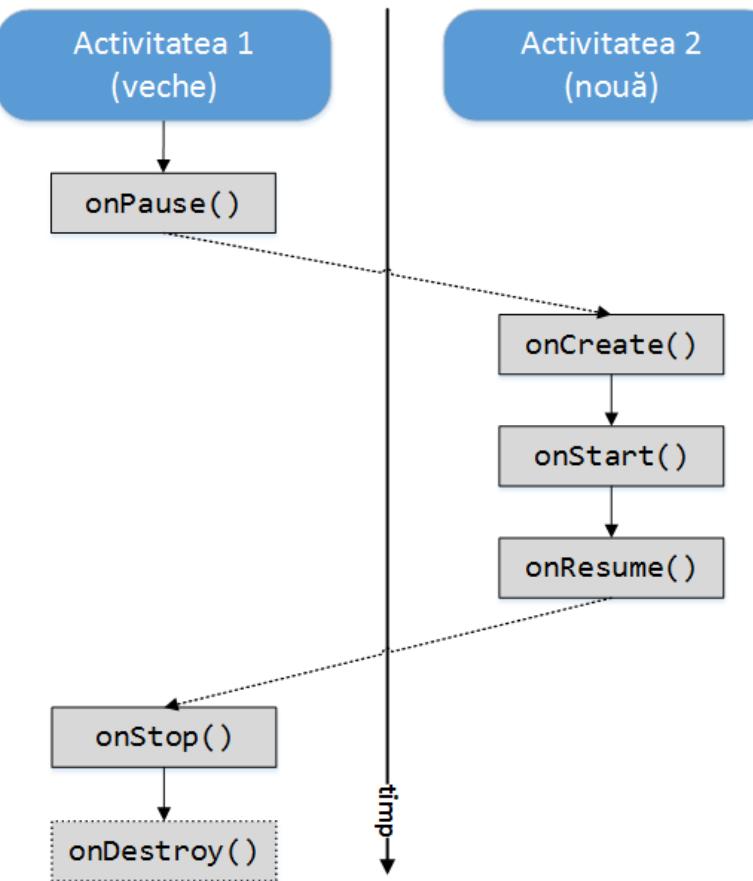
```
11. @Override

12. protected void onRestoreInstanceState(Bundle
 savedInstanceState) {
13. super.onRestoreInstanceState(bundle);
14. EditText usernameEditText=
 (EditText) findViewById(R.id.username_edit_text);
15. if
 (savedInstanceState.getString(Constants.USERNAME_EDIT_TEXT) !=
 null) {
16. usernameEditText.setText(savedInstanceState.getString(Constants
 .USERNAME_EDIT_TEXT));
17. }
}
```

## Coordonarea Activităților

În situația în care o activitate nouă este pornită în timp ce o activitate veche este în execuție, fiind necesară transmiterea de informații dintre acestea, este util să se cunoască ordinea în care sunt apelate metodele care gestionează ciclul de viață al celor două activități, astfel încât comunicarea dintre acestea să se realizeze în mod corect:

1. se apelează metoda `onPause()` a activității vechi - aici trebuie realizată partea de scriere a informațiilor care trebuie să fie trimise;
2. se apeleză metodele `onCreate()`, `onStart()` și `onResume()` ale activității noi - aici trebuie realizată partea de citire a informațiilor care trebuie să fie primite;
3. se apeleză metoda `onStop()` a activității vechi;
4. în situația în care necesarul de memorie nu este satisfăcut, este posibil să se apeleze metoda `onDestroy()` a activității vechi.

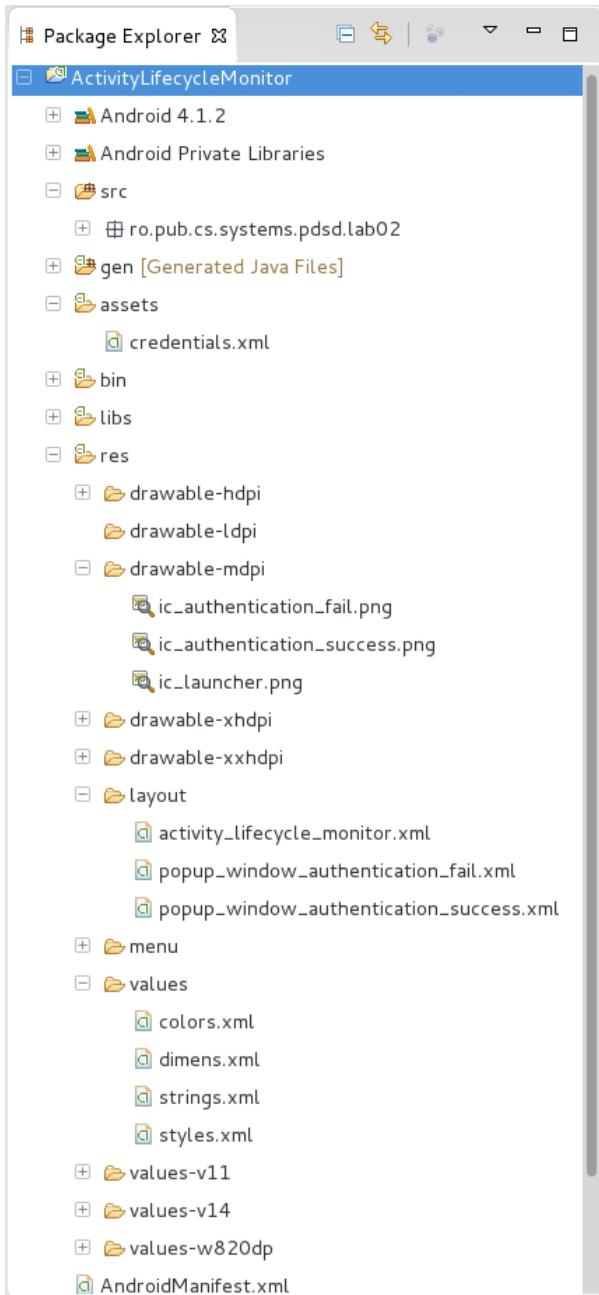


## Structura unui Proiect Android

---

### Eclipse

Structura unui proiect Android în mediul integrat de dezvoltare Eclipse poate fi observată în fereastra *Package Explorer*.



- biblioteca **Android 4.1.2** (Google APIs) - conține mai multe fișiere:
  - android.jar - cuprinde toate clasele necesare pentru aplicația Android;
  - usb.jar - încapsulează clase necesare instalării aplicației Android prin USB și comunicării cu dispozitivul fizic;
  - effects.jar - utilizată pentru aplicarea de diferite efecte;
  - maps.jar - folosită de aplicațiile Android care implementează funcționalități legate de localizare.

- **Android Private Libraries** conține un singur fișier `android-support-v4.jar`;
- **src** - conține sursele `.java` ale proiectului, organizate pe pachete. În exemplu, codul este organizat pe trei pachete, cea mai importantă clasa fiind `LifecycleMonitorActivity.java` (în pachetul `ro.pub.cs.systems.eim.lab02.graphicuserinterface`) continand activitatea principală a aplicației Android.
- **gen** - cuprinde fișiere generate:
  - `BuildConfig.java` - definește unele proprietăți ale aplicației (cum ar fi activarea sau nu a modului de depanare);
  - `R.java` - include identificatori către toate resursele din cadrul proiectului, prin intermediul cărora pot fi obținute referințe către acestea; conține subclasele:
    - `attr` - diferite attribute;
    - `color` - culorile utilizate ca fundal pentru diferite componente grafice afișate în cadrul activităților din cadrul aplicației;
    - `dimen` - dimensiunile dispozitivului de afișare, pentru fiecare configurație suportată de aplicație;
    - `drawable` - resurse de tip grafic, utilizate în cadrul aplicației;
    - `id` - identificatori pentru toate resursele care pot fi referite prin intermediul unui câmp de tipul `android:id`;
    - `layout` - mecanisme de dispunere a conținutului aferent activităților (interfețelor grafice), pentru fiecare configurație suportată de aplicație;
    - `menu` - diferitele meniuri atașate activităților (interfețelor grafice) din cadrul aplicației;
    - `string` - șirurile de caractere definite în cadrul aplicației;
    - `style` - stilurile (temele) folosite în cadrul aplicației.

Este interzisă modificarea fișierelor din directorul `gen`. În cazul în care fișierul `R.java` este șters manual, acesta va fi imediat generat de către mediul de dezvoltare, cu condiția ca proiectul să nu conțină erori (erori de sintaxă în cazul codului sursă sau documente XML care nu sunt bine formate).

- **assets** - cuprinde toate activele utilizate de aplicație (fișiere text, pagini HTML, baze de date), acestea fiind necompilate;
- **bin** - include fișierele construite de ADT în timpul procesului de generare, mai ales fișierul `.apk`(Android Package), un binar care conține toate resursele necesare pentru a rula o aplicație Android;

- **libs** - cuprinde toate bibliotecile necesare pentru compilarea și rularea proiectului;
- **res** - deține toate resursele folosite în cadrul aplicației
  - `drawable-<resolution>`: imaginile utilizate în aplicație, în funcție de rezoluție (ldpi, mdpi, hdpi, xhdpi, xxhdpi)
  - `layout`: fișierul `activity_lifecycle_monitor.xml` care descrie modul de disponere a elementelor grafice în cadrul interfaței cu utilizatorul, separată de codul programului (de asemenea, sunt conținute și fișierele care descriu mecanismele de disponere pentru ferestrele care indică rezultatul operației de autentificare: `popup_window_authentication_success.xml` și `popup_window_authentication_fail.xml`);

#### activity\_lifecycle\_monitor.xml

```

<LinearLayout
 xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:paddingBottom="@dimen/activity_vertical_margin"
 android:paddingLeft="@dimen/activity_horizontal_margin"
 android:paddingRight="@dimen/activity_horizontal_margin"
 android:paddingTop="@dimen/activity_vertical_margin"
 android:orientation="vertical"

 tools:context="ro.pub.cs.systems.eim.lab02.graphicuserinterface.Li
fecycleMonitorActivity" >

 <ImageView
 android:id="@+id/icon_image_view"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"

```

```
 android:layout_gravity="center"

 android:contentDescription="@string/icon_image_view_content_descri
ption"
 android:src="@drawable/ic_launcher" />

<EditText
 android:id="@+id/username_edit_text"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:layout_gravity="center"
 android:ems="10"
 android:hint="@string/username" >

 <requestFocus />
</EditText>

<EditText
 android:id="@+id/password_edit_text"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:layout_gravity="center"
 android:ems="10"
 android:inputType="textPassword"
 android:hint="@string/password" />

<CheckBox
 android:id="@+id/remember_me_checkbox"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@string/remember_me_checkbox_content" />

<LinearLayout
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_gravity="center"
```

```

 android:orientation="horizontal" >

<Button
 android:id="@+id/ok_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_gravity="center"
 android:text="@string/ok_button_content" />

<Space
 android:layout_width="0dp"
 android:layout_height="0dp"
 android:layout_weight="1" />

<Button
 android:id="@+id/cancel_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_gravity="center"
 android:text="@string/cancel_button_content" />

</LinearLayout>

</LinearLayout>

```

Astfel, interfața grafică este dispusă folosind un mecanism de aliniere verticală, fiind formată din:

- un element `ImageView` prin intermediul căruia poate fi vizualizată o imagine;
- două elemente de tip `EditText` prin intermediul cărora care se vor transmite un nume de utilizator, respectiv o parolă (numele de utilizator fiind afișat în clar, în timp ce parola este ascunsă);
- un element de tip `CheckBox` prin intermediul căruia utilizatorul va putea să indice dacă dorește ca informațiile sale de autentificare să fie reținute sau nu;

- două butoane, dispuse folosind un mecanism de aliniere orizontală, prin intermediul cărora utilizatorul poate să transmită (către un server) informațiile sale de autentificare, pentru a fi procesate, respectiv dorește să anuleze informațiile introduse;
- menu: fișierul `main.xml` conține un meniu care se va afișa la accesarea butonului `Options`, fiind folosit de metoda `onCreateOptionsMenu()`;
- `values`
- `colors.xml`: conține definițiile pentru culorile utilizate în cadrul interfeței grafice, ca fundal;

#### colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

 <color name="light_green">#bfd1bc</color>
 <color name="light_red">#efcad1</color>

</resources>
```

- `dimens.xml`: definește dimensiunile dispozitivului de afișare (`activity_horizontal_margin` și `activity_vertical_margin`);
- `strings.xml`: se recomandă ca toate sirurile de caractere constante definite în cadrul aplicației să fie definite aici, fiind referite prin intermediul unui identificator; astfel, în cazul în care este necesar să se localizeze aplicația în altă limbă, tot ce trebuie făcut este să se creeze o copie a directorului `values`, modificându-se corespunzător valorile din acest fișier pentru a conține texte în cauză;

#### strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<?xml version="1.0" encoding="utf-8"?>
```

```

<resources>

 <string name="app_name">Activity Lifecycle Monitor</string>
 <string name="action_settings">Settings</string>
 <string name="icon_image_view_content_description">This is the
application icon.</string>
 <string name="introduce_yourself_text_view_content">Introduce
yourself</string>
 <string name="username">Username</string>
 <string name="password">Password</string>
 <string name="remember_me_checkbox_content">Remember
me</string>
 <string name="ok_button_content">OK</string>
 <string name="cancel_button_content">Cancel</string>
 <string name="authentication_success">The authentication
process was successful!</string>
 <string name="authentication_fail">The authentication process
failed!</string>
 <string name="dismiss_button_content">Dismiss</string>
 <string name="empty"></string>

</resources>

```

- **styles.xml**: temele aplicației;
  
- **AndroidManifest.xml** - conține informații detaliate despre aplicație

### AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<?xml version="1.0" encoding="utf-8"?>
<manifest
 xmlns:android="http://schemas.android.com/apk/res/android"
 package="ro.pub.cs.systems.eim.lab02"
 android:versionCode="1"

```

```

 android:versionName="1.0" >

 <uses-sdk
 android:minSdkVersion="16"
 android:targetSdkVersion="16" />

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

 android:name="ro.pub.cs.systems.eim.lab02.graphicuserinterface.Lif
 ecycleMonitorActivity"
 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>

```

- denumirea pachetului aplicației este ro.pub.cs.systems.eim.lab02;
- versiunea codului aplicației este 1 (definit prin intermediul atributului android:versionCode); această valoare este utilizată pentru a identifica numărul de versiune al aplicației, putând fi folosit pentru a determina (programatic) dacă o aplicație trebuie să fie actualizată;
- denumirea versiunii aplicației este 1.0 (definată prin intermediul atributului android:versionName); această valoare este afișată utilizatorului, recomandându-se să se utilizeze formatul <major>.<minor>.<build>

- atributul `android:minSdkVersion` al elementului `<uses-sdk>` specifică versiunea minimă a sistemului de operare pe care va rula aplicația;
- aplicația folosește drept pictogramă imaginea `ic_launcher.png` din directorul `drawable`;
- denumirea aplicației este dată de sirul de caractere `app_name` definit în fișierul `strings.xml`;
- există o singură activitate în cadrul aplicației, definită în fișierul `LifecycleMonitorActivity.java`, eticheta afișată pentru aceasta fiind aceeași ca și denumirea aplicației; există și un element `<intent-filter>`:
  - acțiunea pentru filtrul de intenții este denumit `android.intent.action.MAIN` pentru a indica faptul că activitatea reprezintă punctul de intrare pentru aplicație;
  - categoria pentru filtrul de intenții este denumită `android.intent.category.LAUNCHER` pentru a indica faptul că aplicația poate fi lansată prin intermediul pictogramei afișată în contextul dispozitivului;

Pentru a specifica o acțiune asociată butoanelor *OK* respectiv *Cancel* (deschiderea unei ferestre prin care se indică rezultatul operației de autentificare, respectiv ștergerea datelor din câmpurile ce conțin credențialele), se modifică codul sursă (fișierul `LifecycleMonitorActivity.java` din directorul `src`).

În acest sens, se va defini o clasă ascultător (de tip `Button.OnClickListener`) internă, cu nume, a cărui instanță va fi atașată obiectului pe care se realizează evenimentul:

```
private class ButtonClickListener implements Button.OnClickListener {

 @Override
 @SuppressWarnings("all")
 public void onClick(View view) {
 EditText usernameEditText = (EditText) findViewById(R.id.username_edit_text);
 EditText passwordEditText = (EditText) findViewById(R.id.password_edit_text);
 if (getResources().getString(R.string.ok_button_content).equals(((Button) view).getText().toString())) {
 LayoutInflater layoutInflater = (LayoutInflater) getBaseContext().getSystemService(LAYOUT_INFLATER_SERVICE);

 if (Utilities.allowAccess(getApplicationContext(), usernameEditText.getText().toString(), passwordEditText.getText().toString())) {
 View popupContent = layoutInflater.inflate(R.layout.popup_window_authentication_success, null);
 }
 }
 }
}
```

```

 final PopupWindow popupWindow = new PopupWindow(popupContent,
LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);

 Button dismissButton
(Button)popupContent.findViewById(R.id.dismiss_button);
dismissButton.setOnClickListener(new Button.OnClickListener() {
 @Override
 public void onClick(View view) {
 popupWindow.dismiss();
 }
});
popupWindow.showAtLocation(view, Gravity.CENTER, 0, 0);
} else {
 View popupContent
layoutInflater.inflate(R.layout.popup_window_authentication_fail, null);
final PopupWindow popupWindow = new PopupWindow(popupContent,
LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);

 Button dismissButton
(Button)popupContent.findViewById(R.id.dismiss_button);
dismissButton.setOnClickListener(new Button.OnClickListener() {
 @Override
 public void onClick(View view) {
 popupWindow.dismiss();
 }
});
popupWindow.showAtLocation(view, Gravity.CENTER, 0,
0);
}
}

if
(getResources().getString(R.string.cancel_button_content).equals(((Button)view).getText().toString())) {
 usernameEditText.setText(getResources().getString(R.string.empty));
 passwordEditText.setText(getResources().getString(R.string.empty));
}
}
}

```

Se observă faptul că elementele din cadrul interfeței grafice sunt identificate prin intermediul valorilor generate în clasa `R`, transmise ca parametri metodei `findViewById()`. De asemenea, trebuie realizat cast-ul explicit la tipul respectiv.

Pentru sărurile de caractere se folosește metoda `getString()` aplicabilă unui obiect `Resources()`, obținut prin apelul metodei `getResources()` din clasa `Context`.

În cadrul metodei `onClick()` se preia textul corespunzător butonului care a fost apăsat, comparându-se cu valorile posibile (pentru ambele butoane se va folosi același obiect ascultător):

- în cazul în care s-a apăsat butonul *OK*, se parsează fișierul `.xml` din assets (`credentials.xml`), preluându-se de acolo toate combinațiile de nume de utilizator și parolă cu care se permite accesul în cadrul aplicației; în funcție de datele pe care le furnizează în câmpurile *Username* și *Password*,

utilizatorul este informat, prin intermediul unei ferestre, de rezultatul operației de autentificare (succes sau eșec);

- În cazul în care s-a apăsat butonul *Cancel*, se șterg datele din câmpurile ce conțin credențialele (nume de utilizator și parolă), precizându-se pentru acestea un text asociat vid.

Pentru ca metoda să fie apelată la apăsarea butonului, va trebui realizată legătura dintre metodă și buton. Acest lucru se poate realiza în două moduri:

1. programatic, în codul sursă:

```
2. private ButtonClickListener buttonClickListener = new
 ButtonClickListener();

3. Button okButton = (Button) findViewById(R.id.ok_button);
4. okButton.setOnClickListener(buttonClickListener);

5. Button cancelButton = (Button) findViewById(R.id.cancel_button);

cancelButton.setOnClickListener(buttonClickListener);
```

6. În fișierul `activity_lifecycle_monitor.xml`, prin specificarea proprietății `android:onClick` asociată elementului buton având ca valoare denumirea metodei în cauză:

android:onClick="onClick"

La compilare, este posibil ca unele clase să nu poată fi rezolvate. Se foloseste Ctrl+Shift+O pentru a adăuga automat importurile necesare.

## Android Studio

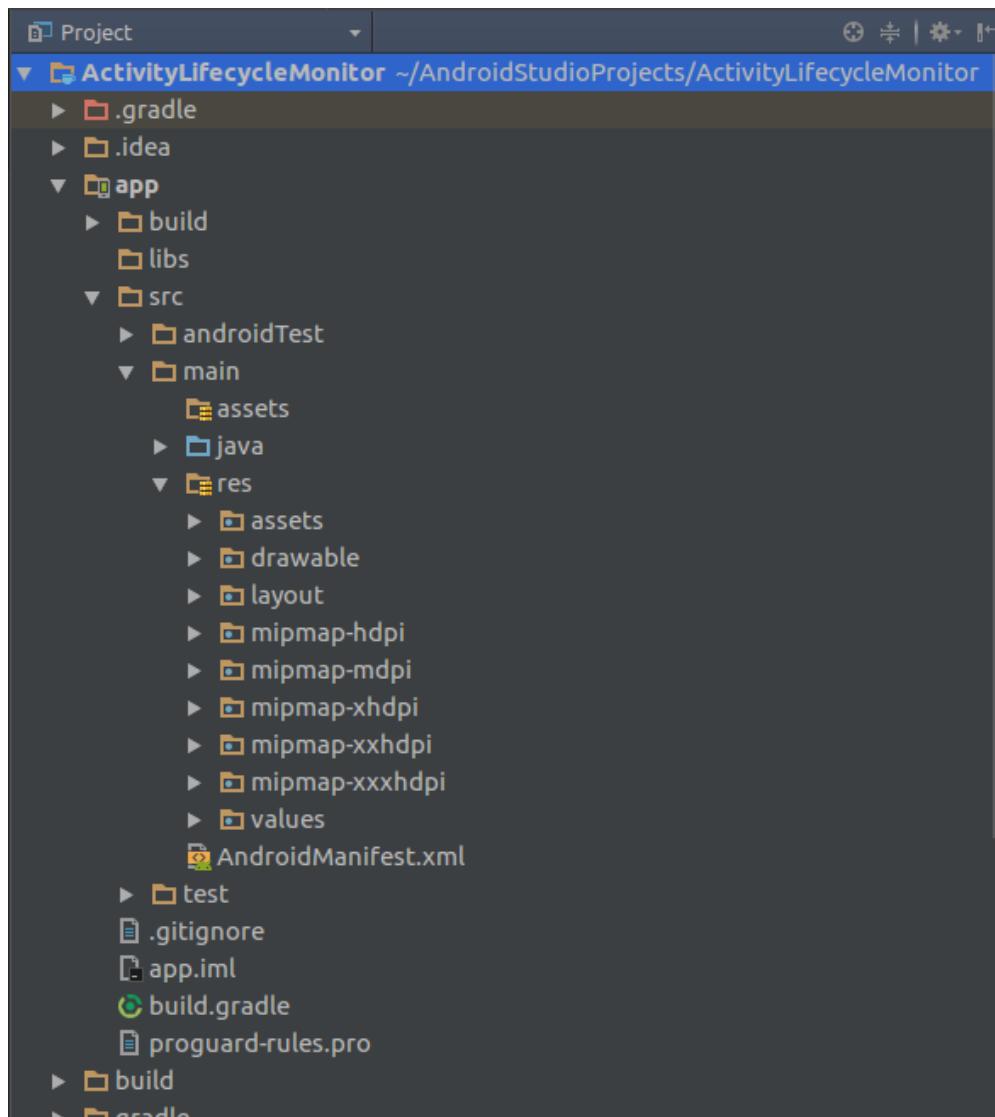
În Android Studio, există mai multe perspective sub care poate fi vizualizată structura unui proiect:

- Project (reprezintă structura de pe discul fizic)
  - Packages
  - Scratches
  - Android (reprezintă gruparea logică a modulelor aplicației Android)
  - Project Files
  - Problems
  - Production

- Tests
- Local Unit Tests
- Android Instrumented Tests

Cele mai frecvent utilizate perspective sunt *Project* și *Android*.

Organizarea unui proiect Android Studio conține următoarele directoare:



- .gradle - fișiere temporare Gradle (biblioteci descărcate, fișiere de configurare)
- .idea - fișiere temporare Android Studio
- app
  - build - fișiere generate conținând referințe (adrese) către resursele definite (spre exemplu, în directorul generated/source se găsesc clasele BuildConfig și R);
  - src

- androidTest - cazuri de test
- main
  - assets - resurse necompilate
  - java - codul sursă java
  - res
    - drawable - resurse grafice
    - layout - interfețe grafice definite static în fișiere XML
    - midmap-<resolution> - pictograma aplicației, pentru diferite rezoluții
    - values - constante care vor fi referite în text
      - colors.xml
      - dimens.xml
      - strings.xml
      - styles.xml
    - AndroidManifest.xml - doar componentele aplicației Android: activități, servicii, receptoare cu difuzare, filtre de intenții pentru fiecare componentă
      - test - surse pentru teste unitare
  - build - cache pentru fișiere .dex
  - gradle
  - External Libraries
  - alte fișiere (în rădăcină):
    - .gitignore
    - <project>.iml
    - build.gradle
    - gradle.properties
    - gradlew
    - local.properties
    - settings.gradle

## Gradle

Android Studio folosește un mecanism automat pentru construirea aplicației Android, denumit **Gradle**, responsabil cu aducerea bibliotecilor referite de pe un depozit la distanță, cu definirea proprietăților aplicației Android, cu compilarea și împachetarea tuturor resurselor folosite, pentru rularea și instalarea aplicației astfel rezultate.

Regulile pentru construirea aplicației Android sunt precizate în fișiere build.gradle, care se definesc pentru fiecare modul și proiect constituent.

Un fișier de configurare Gradle pentru o aplicație Android conține de regulă două secțiuni:

- android - conține proprietățile aplicației Android
  - compileSdkVersion - reprezintă versiunea de SDK care va fi utilizată pentru compilarea proiectului Android
  - buildToolsVersion - reprezintă versiunea de Android SDK Build Tools folosită pentru construirea fișierului care va fi instalat pe dispozitivul mobil
  - defaultConfig - conține diferite configurații
    - applicationId - pachetul care identifică **în mod unic** aplicația Android
    - minSdkVersion - platforma minimă pe care se garantează că aplicația Android va rula; astfel, se vor folosi numai funcționalități definite la acest nivel sau funcționalități definite în API-uri superioare, dar care sunt disponibile la nivelul bibliotecilor de suport;
    - targetSdkVersion - platforma maximă la care se garantează că aplicația Android va rula (de regulă, este versiunea cea mai recentă și este aceeași versiune folosită pentru compilarea codului sursă);
    - versionCode - versiunea curentă a aplicației (număr întreg)
    - versionName - versiunea curentă a aplicației, afișată către utilizator (format lizibil, de tip sir de caractere)
    - testInstrumentationRunner - biblioteca folosită pentru testele aplicației Android
  - dependencies - reprezintă bibliotecile de care depinde aplicația Android pentru a putea fi compilată / rulată, precum și reguli de compilare
    - compile - se precizează care fișiere sunt luate în considerare pentru classpath
      - directiva `include` se folosește pentru a indica tipuri de fișiere care conțin diverse biblioteci);
      - directiva `fileTree` este utilizată pentru a indica o structură de directoare
    - testCompile și androidTestCompile - indică pachete care conțin biblioteci pentru teste unitare.

### `build.gradle`

```
apply plugin: 'com.android.application'

android {
```

```
compileSdkVersion 25

buildToolsVersion "25.0.2"

defaultConfig {

 applicationId
"ro.pub.systems.eim.lab02.activitylifecyclemonitor"

 minSdkVersion 16

 targetSdkVersion 25

 versionCode 1

 versionName "1.0"

 testInstrumentationRunner
"android.support.test.runner.AndroidJUnitRunner"

}

buildTypes {

 release {

 minifyEnabled false

 proguardFiles getDefaultProguardFile('proguard-
android.txt'), 'proguard-rules.pro'

 }

}

dependencies {

 compile fileTree(dir: 'libs', include: ['*.jar'])

 compile 'com.android.support:appcompat-v7:25.2.0'

}
```

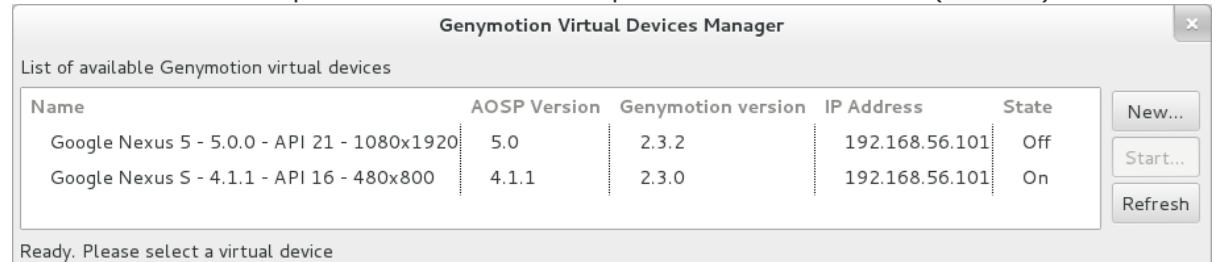
## Rularea și Depanarea unei aplicații Android

---

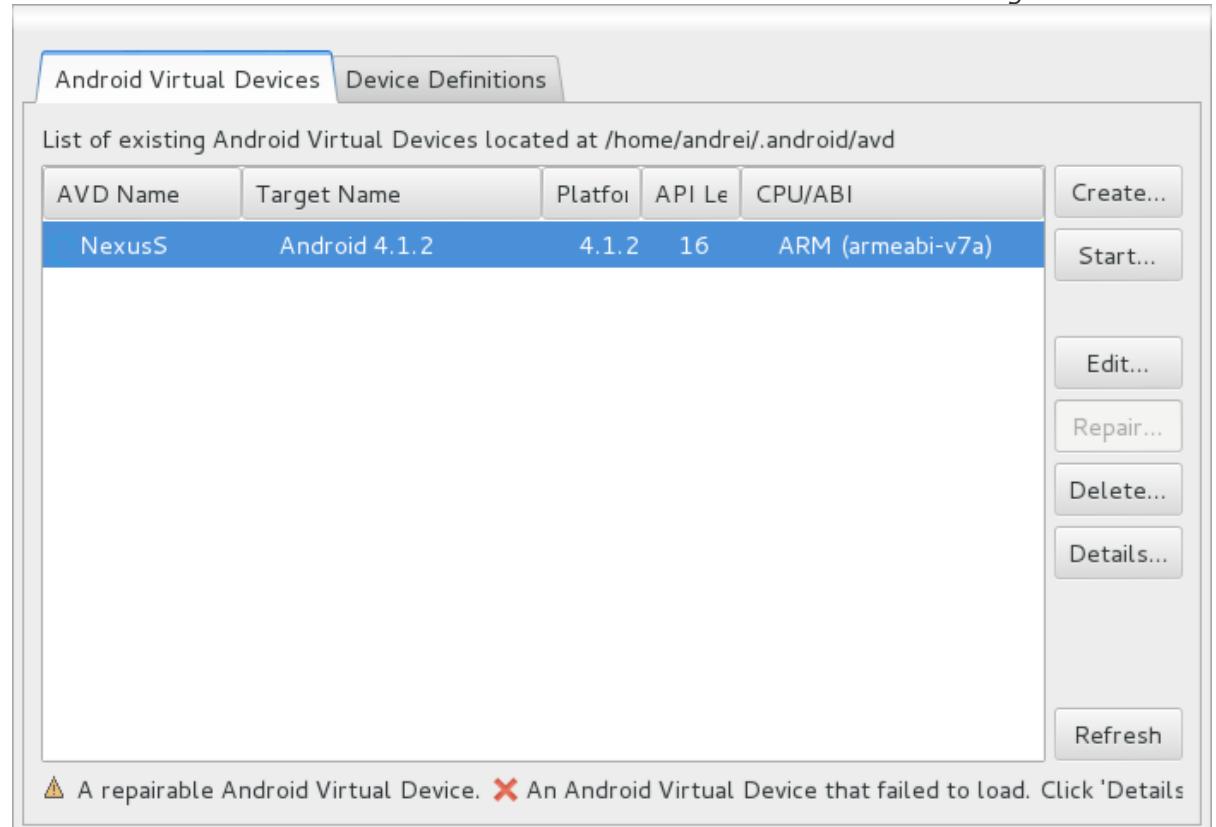
### Rularea unei aplicații Android

Pentru a rula aplicația, este necesar un dispozitiv mobil fizic sau un dispozitiv mobil virtual (emulator).

- În cazul în care se folosește **GenyMotion**, se pornește un dispozitiv virtual, starea acestuia putând fi monitorizată folosind Genymotion Virtual Devices Manager, fereastră ce se deschide în momentul în care se accesează plugin-ul de Eclipse corespunzător (Ctrl+6).

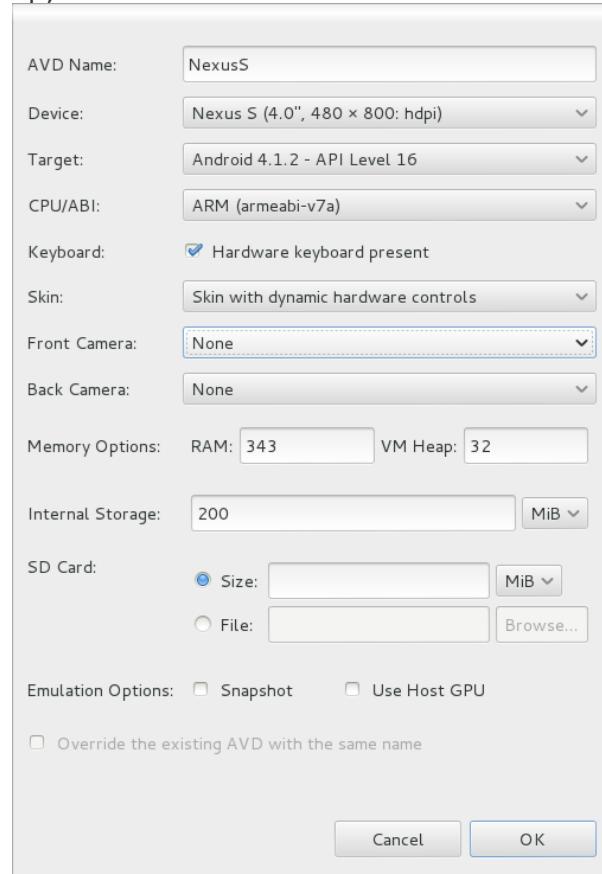


- În cazul în care se folosește **emulatorul AVD** livrat împreună cu SDK-ul de Android, se va crea un dispozitiv virtual accesând *Window → Android Virtual Device Manager*

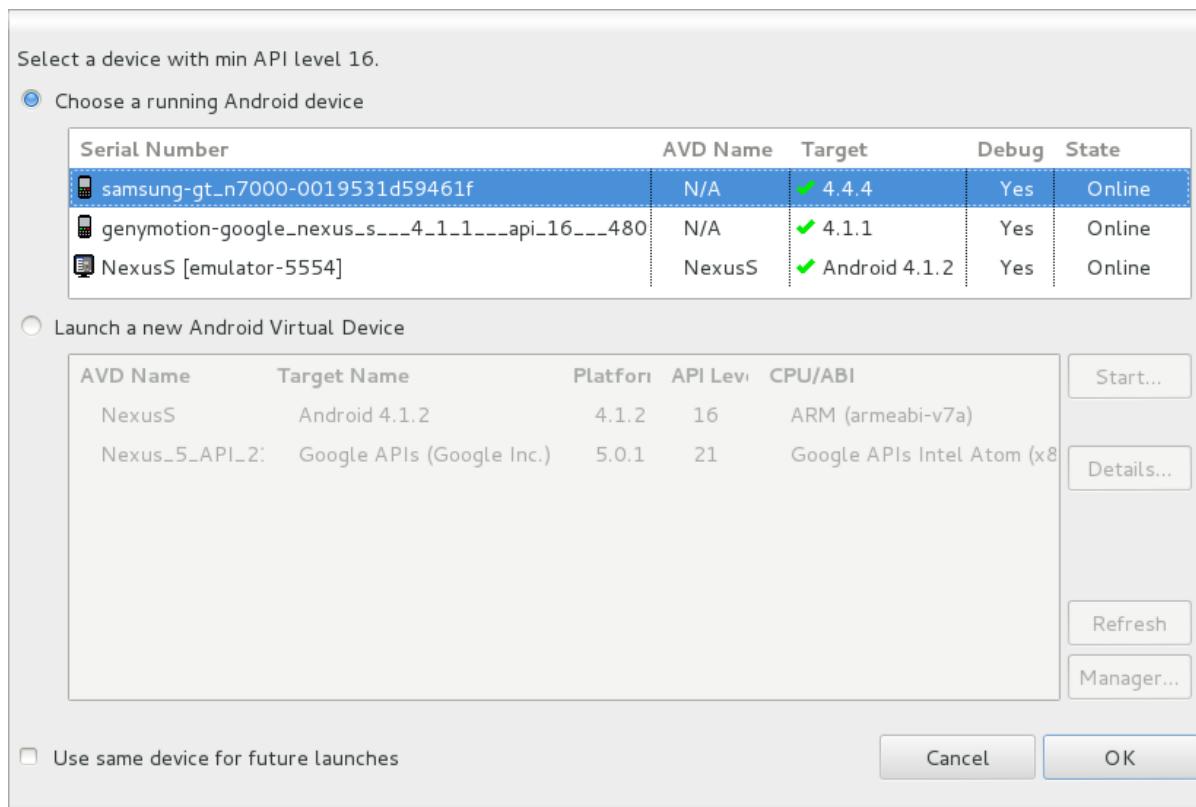


Acesta permite gestiunea dispozitivelor virtuale (New / Edit / Delete) prin specificarea unor parametrii precum dispozitivul (poate fi utilizat un dispozitiv predefinit sau un dispozitiv definit de utilizator, pornind de la configurațiile existente în *Device Definitions*), versiunea de Android pe care se va rula, caracteristicile referitoare la memorie (RAM, heap-ul pentru mașina virtuală), dimensiunea spațiului de stocare, emularea unui card de memorie

(dimensiune, fișier pe disc). Pornirea propriu-zisă a dispozitivului se face prin opțiunea *Start*.



Ulterior se poate rula aplicația, accesând din meniul contextual al proiectului (click dreapta) *Run As.. → Android Application* (sau *Ctrl+F11*). În acest moment, se vor identifica toate dispozitivele Android (fizice sau virtuale) care sunt disponibile (conectate) și care rulează un sistem de operare având cel puțin un nivel de API specificat de proprietatea `android:minSdkVersion` în cadrul fișierului `AndroidManifest.xml`, afișându-se lista acestora, astfel încât utilizatorul să își poată alege unul dintre ele.

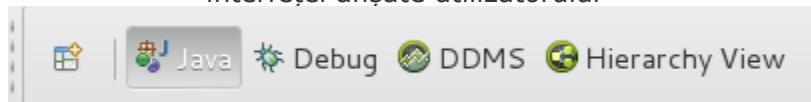


## Depanarea unei aplicații Android

Depanarea unei aplicații Android poate fi realizată în mod clasic, ca în cazul oricărui proiect (prin stabilirea unor puncte în care execuția codului sursă este întreruptă și rularea pas cu pas) sau prin intermediul unor utilitare specifice.

În cadrul mediului integrat de dezvoltare Eclipse, depanarea unei aplicații este facilitată prin împărțirea ecranului în mai multe perspective între care programatorul are posibilitatea de a comuta:

- Java - pentru realizarea de modificări la nivelul codului sursă
- Debug - pentru monitorizarea pas cu pas a execuției programului, cu posibilitatea de consultare a valorilor pe care le au la un moment dat diferite obiecte
- DDMS - pentru furnizarea de informații detaliate la nivel de procese, fire de execuție, sistem de fișiere, memorie ocupată, evenimente specifice dispozitivelor mobile
- Hierarchy View - pentru vizualizarea ierarhiei de componente grafice din cadrul interfeței afișate utilizatorului



## Android Debug Bridge (ADB)

Android Debug Bridge este un utilitar în linie de comandă care permite comunicarea cu un dispozitiv mobil fizic sau cu un emulator, prin intermediul unui program client-server ce include 3 componente:

- un client, apelat prin comanda `adb` (alți clienți sunt plugin-ul ADT, ADM-ul, Layout Inspector);
- un server (rulează ca proces de fundal), care gestionează comunicarea dintre client și daemonul ce rulează pe emulator sau dispozitivul mobil fizic;
- un daemon, care rulează ca un proces de fundal pentru fiecare emulator sau dispozitiv mobil fizic.

ADB este integrat în SDK-ul de Android, regăsindu-se în directorul `platform-tools`.

### Comenzi ADB

- pentru a folosi ADB shell, device-ul Android trebuie să fie root-at (imaginile genymotion sunt deja).
- comenzile ADB pot fi rulate din linia de comandă sau din script, având următorul format:

```
student@eim2017:/opt/android-sdk-linux/platform-tools$ adb [-d|-e|-s <serialNumber>] <command>
```

- Înainte de a utiliza comenzi `adb` este important să fie cunoscut identificatorul dispozitivului care este conectat la serverul `adb`, acesta putând fi identificat prin comanda `adb devices`:

- student@eim2017:/opt/android-sdk-linux/platform-tools\$ adb devices
  - List of devices attached
  - emulator-5556 device
  - 192.168.56.101:5555 device

```
0123456789ABCDEF device
```

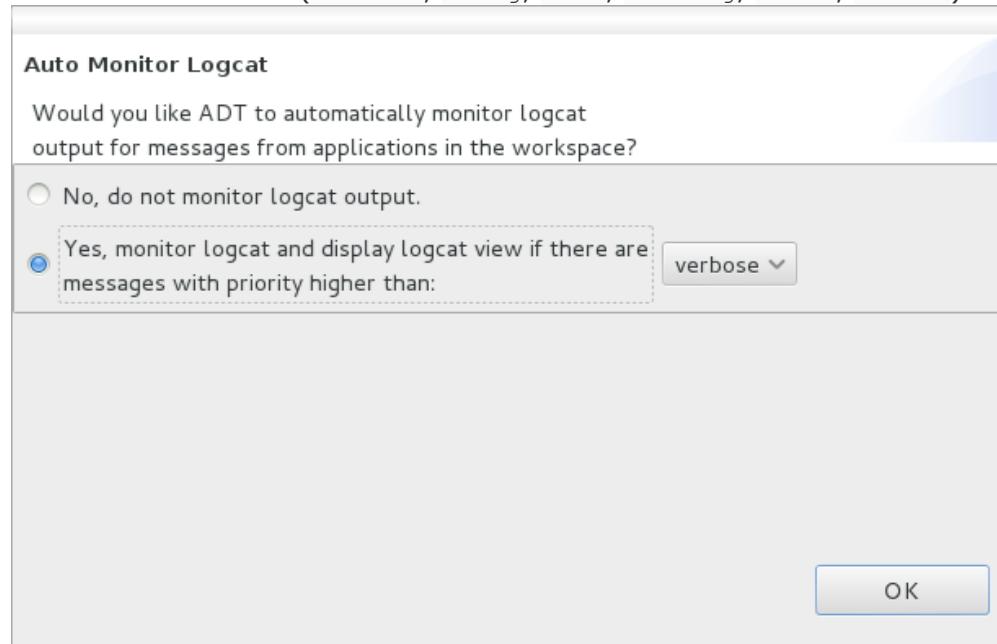
- conexiunea la emulator se realizează folosind comanda `adb -s <serialNumber> shell`

Depanarea este foarte importantă în procesul de realizare a aplicațiilor pentru dispozitive mobile. Există însă unele diferențe față de depanarea programelor pentru calculator, întrucât aplicațiile rulează pe un alt dispozitiv, fiind necesare programe specializate. De asemenea, fiind vorba de dispozitive mobile, apar și anumite evenimente specifice, cum ar fi apeluri telefonice, primirea unui mesaj, descărcarea bateriei, intreruperi ce trebuie tratate într-un fel sau altul.

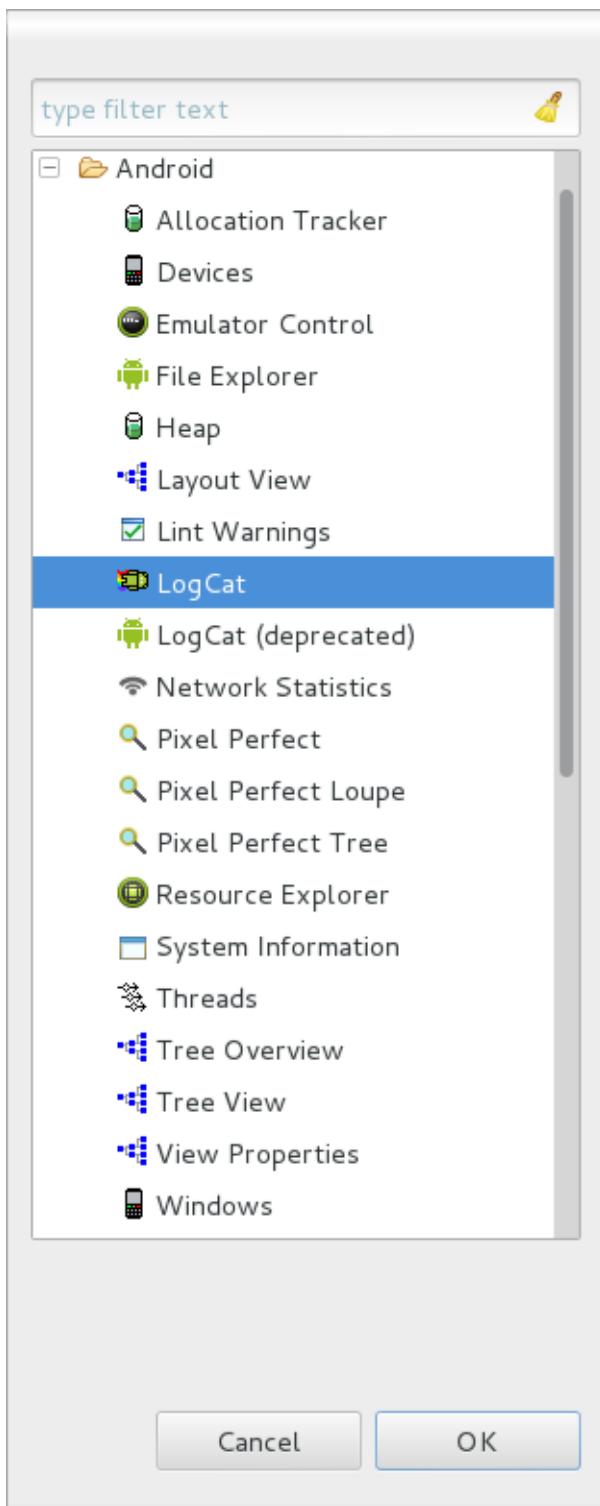
## LogCat

Jurnalele sistemului de operare conțin cele mai importante informații pentru programator. Acestea descriu toate acțiunile realizate de către dispozitivul mobil, exceptiile apărute precum și alte informații necesare depanării. Ele pot fi vizualizate în panoul denumit **LogCat**. LogCat nu trebuie confundat cu Console, în care sunt afișate mesajele provenind de la Eclipse.

De regulă, la pornirea unei aplicații, utilizatorul este întrebat dacă dorește să monitorizeze mesajele provenind de la dispozitivul mobil, indicându-se și nivelul de prioritate prin care acestea vor fi filtrate (verbose, debug, info, warning, error, assert).



Ulterior, acest *Window → Show View → Other... → Android → LogCat*

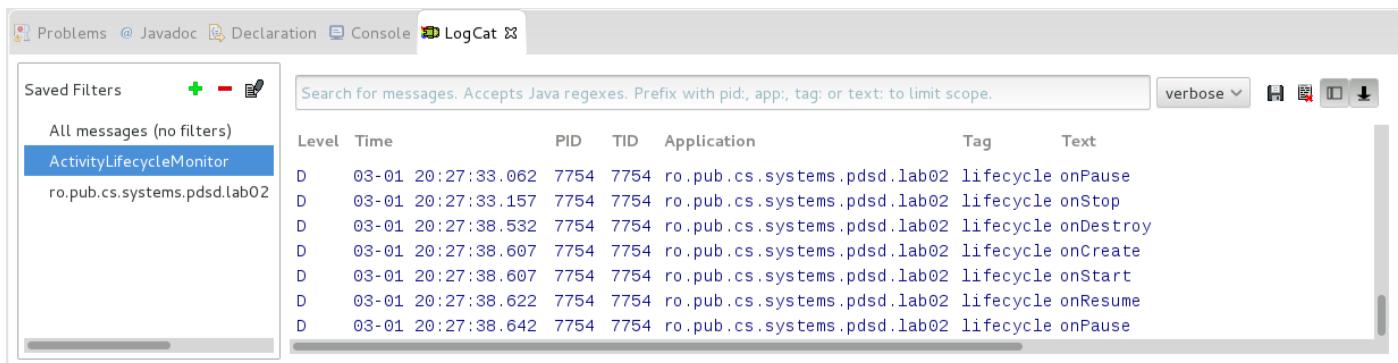


Fiecare mesaj din această listă este însoțit de următoarele informații (fiecare pe cale o coloană):

- *Level* (tipul mesajului)

- **V** - *Verbose* - informații suplimentare afișate de programe (de obicei la cerere, se folosesc la depistarea problemelor de funcționare rezultate din cauza configurației greșite a programelor)
- **D** - *Debug* - mesaj utile la depanare
- **I** - *Information* - mesaj informativ
- **W** - *Warning* - mesaj de avertizare (excepțiile ce nu sunt foarte importante, neavând un impact semnificativ asupra componentei)
- **E** - *Error* - mesaj de eroare (excepțiile ce întrerup o componentă)
- **A** - *Assert* - mesaj ce indică faptul că o condiție esențială pentru rularea aplicației nu a fost îndeplinită
- *Time* - data și ora la care a fost generat mesajul
- *PID* - identificatorul procesului ce a generat mesajul
- *TID* - identificatorul firului de execuție ce a generat mesajul
- *Application* - aplicația care a generat mesajul
- *Tag* - eticheta sau categoria mesajului (util pentru definirea de filtre)
- *Text* - conținutul propriu-zis al mesajului

Este de remarcat faptul că fiecare mesaj reprezintă câte o linie de text. În cazul excepțiilor, întrucât acestea conțin în general mai multe linii, cuprindând foarte multă informație, ele vor fi reprezentate de mai multe mesaje.



Există două mecanisme prin care utilizatorii pot genera astfel de mesaje:

- metodele statice ale clasei `Log`, care primesc ca parametru prioritatea mesajului, eticheta și mesajul propriu zis

```
Log.println (Log.DEBUG, "log sample", "this is a log message
using 'log sample' tag");
```

Pentru a se evita specificarea priorității mesajului de fiecare dată, se pot utiliza metode specifice, care primesc ca parametrii doar denumirea etichetei și mesajul ce se dorește a fi jurnalizat:

Nivel de Prioritate	Metodă	Observații
ERROR	Log.e (...)	erori
WARNING	Log.w (...)	avertismente
INFO	Log.i (...)	mesaje de informare
DEBUG	Log.d (...)	mesaje utilizate pentru depanare; pot fi filtrate (ignorate)
VERBOSE	Log.v (...)	utilizate doar de programatori, pentru dezvoltarea aplicațiilor

- *System.out.println* mesajele către consola standard (stdout)

```
System.out.println("this is a message to the standard console");
```

Mesajele transmise sub această formă vor fi de tipul `Information` și vor avea eticheta `System.out`.

O altă funcționalitate importantă este posibilitatea de filtrare a mesajelor de tip jurnal. Deoarece LogCat afișează toate mesajele de log din sistem, urmărirea unor anumite mesaje poate fi dificilă. Pentru a facilita această sarcină, se pot genera filtre în funcție de anumite valori ale:

- etichetei
- mesajului propriu-zis
- PID
- denumirii aplicației
- nivelului minim al tipului de mesaj (`verbose → assert`)

Un filtru se creează prin apăsarea butonului plus de culoare verde din bara panoului Log (respectiv LogCat în Eclipse).

## Logcat Message Filter Settings

Filter logcat messages by the source's tag, pid or minimum log level.

Empty fields will match all messages.

Filter Name: ActivityLifecycleMonitor

by Log Tag: lifecycleevents

by Log Message:

by PID:

by Application Name: ro.pub.cs.systems.pdsd.lab02

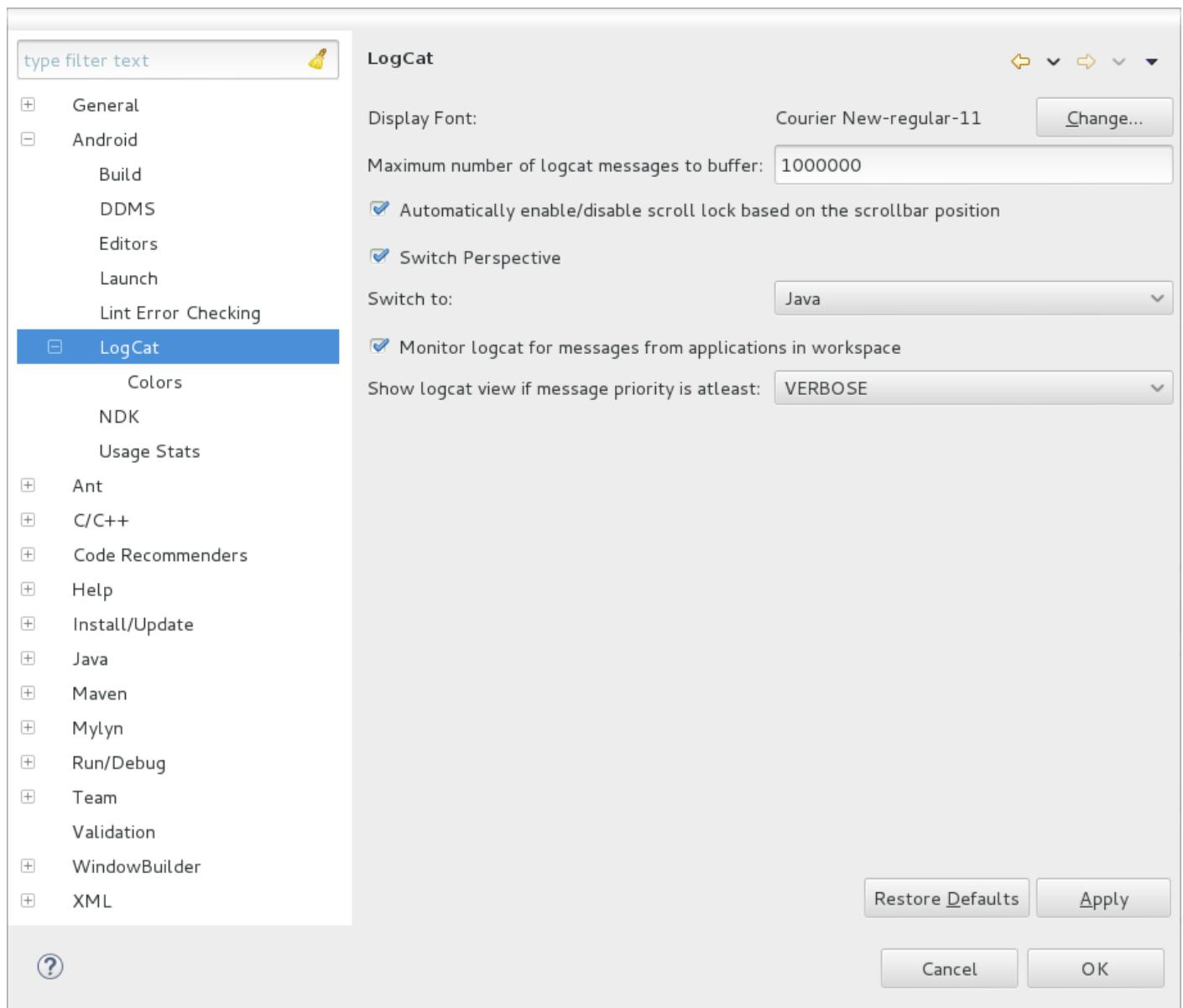
by Log Level: verbose ▾



Cancel

OK

Se recomandă să se configureze un număr cât mai mare de mesaje de jurnalizare care să fie stocate în memoria tampon (*Window → Preferences → Android → LogCat*) încrucit după depășirea valorii respective, monitorizarea nu va mai fi posibilă.



## Android Device Monitor

Utilitarul pentru depanarea aplicațiilor pentru Android se numește **Android Device Monitor** (anterior purta denumirea DDMS - Dalvik Debug Monitor System). Acesta folosește **ADB (Android Debug Bridge)**, pentru a se conecta dispozitive fizice sau la emulatoare. Prin intermediul ADM se pot vizualiza parametrii dispozitivului și a aplicațiilor ce rulează pe acesta. În Android Studio, Android Device Monitor este accesibil din *Tools → Android → Android Device Monitor*. În Eclipse, poate fi accesat sub forma unei perspective, care va fi deschisă prin *Window → Open Perspective → DDMS*.

Name		Online	4.1.1, debug
genymotion-google_nexus_s_4.1.1	[ ]	Online	4.1.1, debug
NexusS [emulator-5554]	[ ]	Online	NexusS [4.1.2,
samsung-gt_n7000-0019531d594	[ ]	Online	4.4.4, debug
com.android.deskclock	7873		8600
com.facebook.orca	3079		8602
<b>ro.pub.cs.systems.pdsd.lab02</b>	<b>7754</b>		<b>8603 / 8700</b>
ro.activesoft.virtualcard	3917		8604
com.google.android.apps.maps	7949		8605
com.google.android.apps.plus	7120		8606
org.omnirom.omnисwitch	2513		8607
com.facebook.katana	6610		8608
com.android.calendar	8018		8609
com.android.phone	2518		8610
com.dropbox.android	4119		8611
com.android.smspush	2650		8612
com.google.android.gms.wearabl	6363		8613
com.bel.android.dspmanager	2524		8614
android.process.media	2588		8615
org.telegram.messenger	4060		8616
com.facebook.katana:dash	7773		8617
com.android.vending	6174		8618
com.google.android.youtube	3615		8619
com.google.process.location	2721		8620
android.process.acore	7011		8622
system_process	2276		8623
com.whatsapp	2918		8624
eu.chainfire.opendelta	3302		8625
com.android.mms	7912		8626

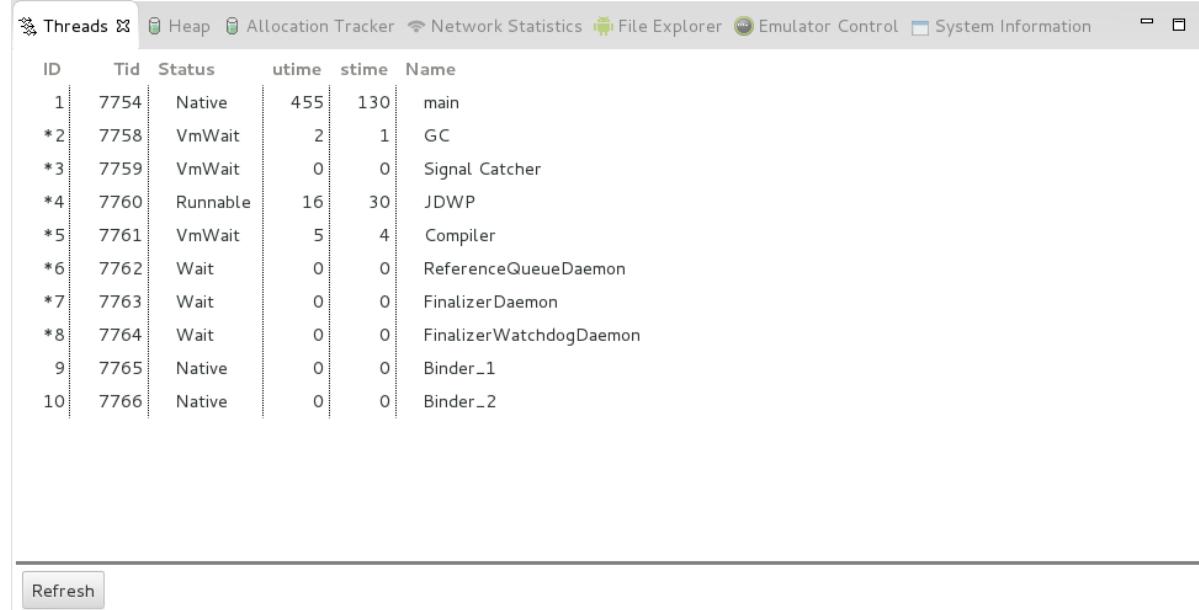
Android Device Monitor va afișa lista tuturor dispozitivelor mobile conectate (fizice sau virtuale), pentru fiecare indicându-se toate procesele care rulează (identificabile prin denumirea pachetului), numărul de proces și numărul firului de execuție. Alegerea dispozitivului curent se face prin selectarea lui din panoul Devices. Operațiile disponibile pentru fiecare proces în parte sunt: depanarea, actualizarea informațiilor cu privire la memoria utilizată, rularea (forțată) procesului de colectare a memoriei disponibile, afișarea firelor de execuție asociate, analiza metodelor, oprirea, realizarea unei capturi de ecran, vizualizarea ierarhiei de componente grafice.

Cele mai importante funcționalități pe care le oferă DDMS sunt:

- afișarea informațiilor despre procesele curente și firele lor de execuție
- afișarea informațiilor cu privire la utilizarea memoriei
- monitorizarea alocărilor de memorie
- vizualizarea unor statistici cu privire la traficul generat prin rețea
- consultarea sistemului de fișiere
- simularea unor evenimente de tip întrerupere în cadrul emulatoarelor
- furnizarea unor mesaje generale cu privire la sistemul de operare

## Procese și Fire de Execuție

În panoul **Threads** sunt afișate informațiile despre fiecare fir de execuție corespunzător unui proces. Întrucât aceste informații sunt preluate prin intermediul unei legături de rețea, astfel încât traficul generat este destul de mare, pentru nu a se încetini semnificativ funcționarea dispozitivului mobil (care este nevoie să transmită aceste date permanent), vizualizarea trebuie activată manual. Astfel, se selectează dispozitivul mobil dorit și procesul respectiv (din panoul *Devices*) și se apasă butonul *Refresh*.



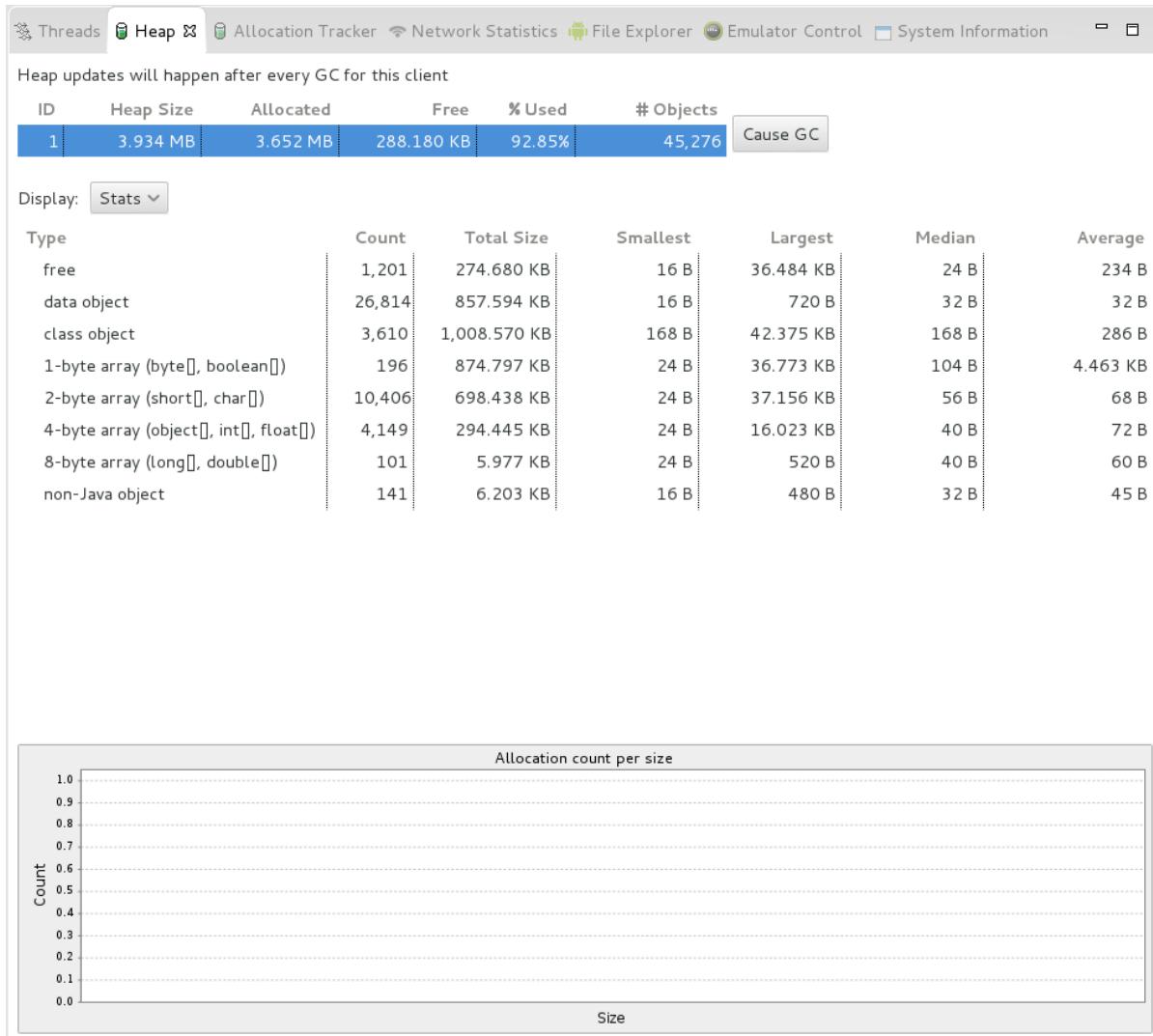
The screenshot shows the DDMS interface with the 'Threads' tab selected. The table displays the following data:

ID	Tid	Status	utime	stime	Name
1	7754	Native	455	130	main
*2	7758	VmWait	2	1	GC
*3	7759	VmWait	0	0	Signal Catcher
*4	7760	Runnable	16	30	JDWP
*5	7761	VmWait	5	4	Compiler
*6	7762	Wait	0	0	ReferenceQueueDaemon
*7	7763	Wait	0	0	FinalizerDaemon
*8	7764	Wait	0	0	FinalizerWatchdogDaemon
9	7765	Native	0	0	Binder_1
10	7766	Native	0	0	Binder_2

A 'Refresh' button is located at the bottom left of the table area.

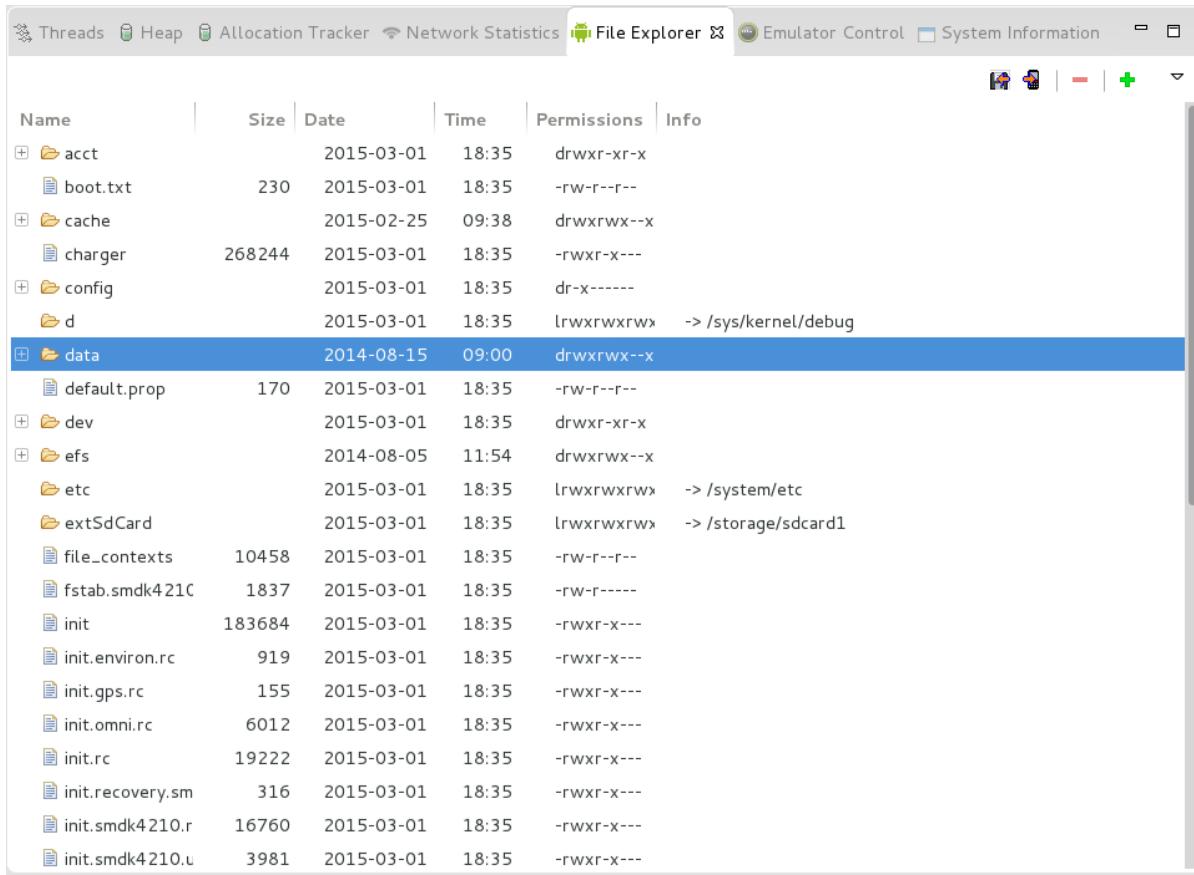
### Utilizarea Memoriei

DDMS oferă posibilitatea de a monitoriza utilizarea memoriei în cadrul panourilor **Heap** și **Allocation Tracker**, oferind informații cu privire la gradul de ocupare a memoriei, defalcată pe tipuri de obiecte (obiecte, clase, tablouri conținând elemente stocate pe 1, 2, 4, 8 octeți, date non-Java).



## Sistemul de Fișiere

Consultarea sistemului de fișiere al dispozitivului mobil poate fi realizat prin intermediul panoului **File Explorer**, putând fi vizualizate datele aplicației, stocate în `data/data/<package-name>`, unde `<package-name>` este denumirea pachetului corespunzător aplicației investigate.



The screenshot shows the DDMS File Explorer interface. The top navigation bar includes tabs for Threads, Heap, Allocation Tracker, Network Statistics, File Explorer (which is active), Emulator Control, System Information, and several icons for zooming and switching between panes.

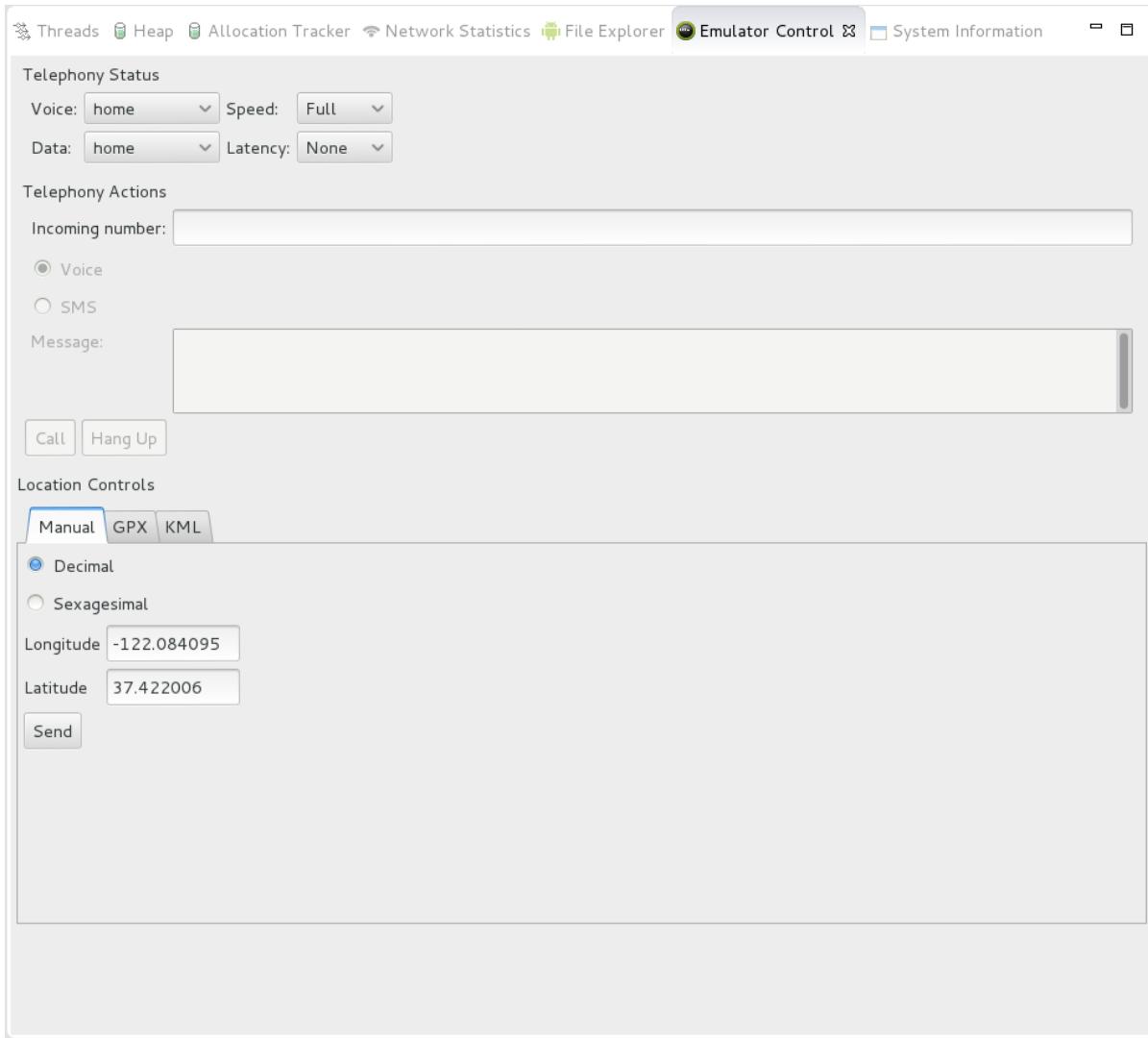
The main content area is a table listing system files and directories. The columns are Name, Size, Date, Time, Permissions, and Info. The 'data' directory is selected, highlighted with a blue background.

Name	Size	Date	Time	Permissions	Info
+ acct		2015-03-01	18:35	drwxr-xr-x	
boot.txt	230	2015-03-01	18:35	-rw-r----	
+ cache		2015-02-25	09:38	drwxrwx--x	
charger	268244	2015-03-01	18:35	-rwxr-x---	
+ config		2015-03-01	18:35	dr-----	
d		2015-03-01	18:35	lrwxrwxrw>	-> /sys/kernel/debug
+ data		2014-08-15	09:00	drwxrwx--x	
default.prop	170	2015-03-01	18:35	-rw-r--r--	
+ dev		2015-03-01	18:35	drwxr-xr-x	
+ efs		2014-08-05	11:54	drwxrwx--x	
etc		2015-03-01	18:35	lrwxrwxrw>	-> /system/etc
extSdCard		2015-03-01	18:35	lrwxrwxrw>	-> /storage/sdcard1
file_contexts	10458	2015-03-01	18:35	-rw-r--r--	
fstab.smdk421C	1837	2015-03-01	18:35	-rw-r-----	
init	183684	2015-03-01	18:35	-rwxr-x---	
init.environ.rc	919	2015-03-01	18:35	-rwxr-x---	
init.gps.rc	155	2015-03-01	18:35	-rwxr-x---	
init.omni.rc	6012	2015-03-01	18:35	-rwxr-x---	
init.rc	19222	2015-03-01	18:35	-rwxr-x---	
init.recovery.sm	316	2015-03-01	18:35	-rwxr-x---	
init.smdk4210.r	16760	2015-03-01	18:35	-rwxr-x---	
init.smdk4210.u	3981	2015-03-01	18:35	-rwxr-x---	

### Simularea unor evenimente de tip întrerupere pentru emulator

Întrucât emulatorul de Android nu poate simula toate funcțiile unui telefon real, pentru a se putea testa comportamentul aplicațiilor în cazul apariției unor evenimente de tip întrerupere, DDMS pune la dispoziție un panou **Emulator Control**, prin care pot fi controlate:

- starea conexiunii de voce / date
- primirea unui apel telefonic / SMS (apel, pierdere apel)
- date primite de la GPS



În cazul testării pe dispozitive mobile reale, aceste funcționalități nu vor fi disponibile prin intermediul ADM.

Emulatorul Genymotion nu permite simularea nici una dintre evenimentele de mai sus prin intermediul DDMS.

Din cadrul emulatoarelor (deci nu folosind DDMS) mai pot fi simulate evenimente de tip:

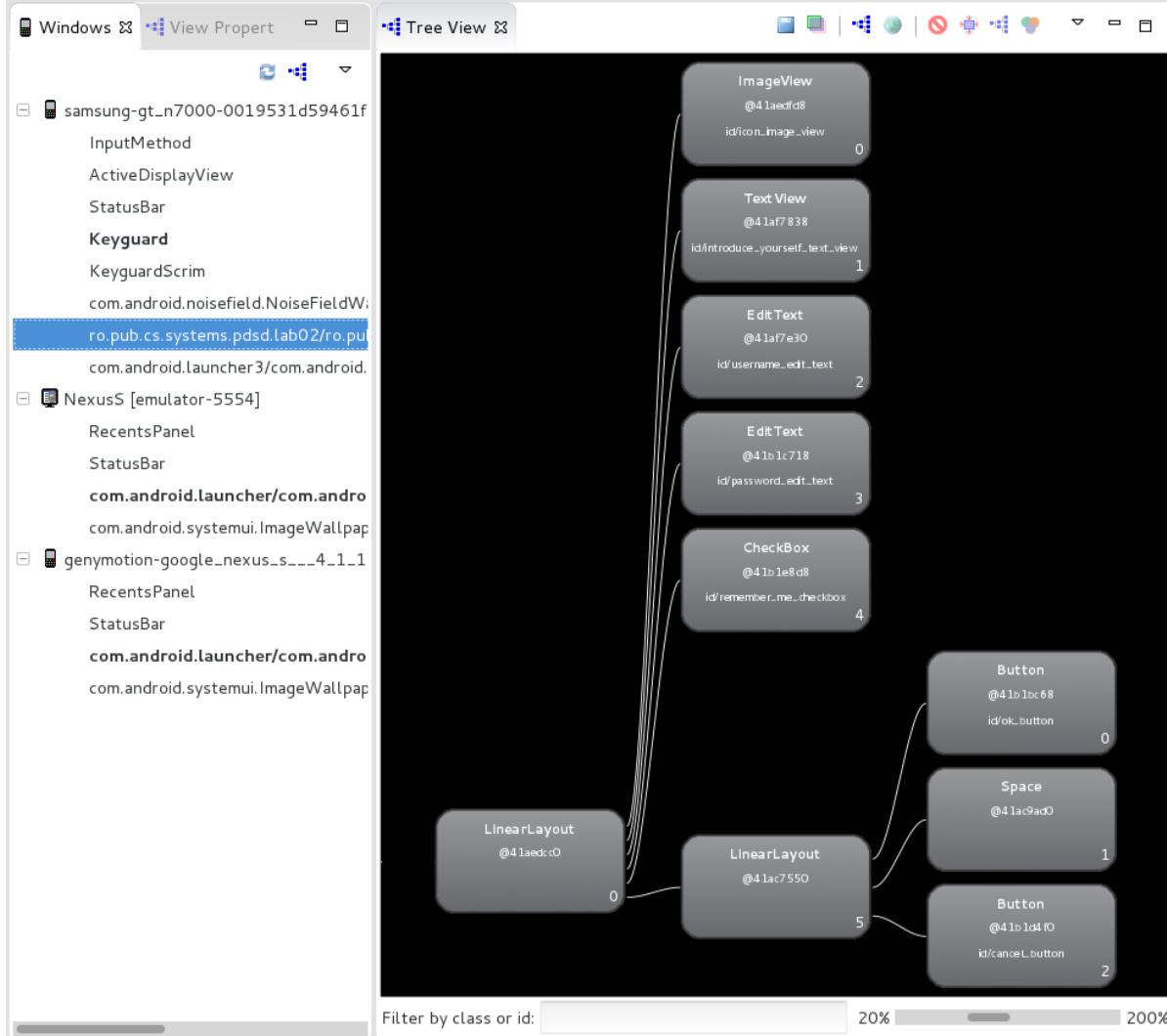
- cameră foto
- accelerometru

### Layout Inspector

Un alt program foarte util pentru depanare este **Layout Inspector** (anterior, purta denumirea Hierarchy View). Acesta permite vizualizarea arborescentă a structurii ferestrelor. De asemenea, afișează parametrii fiecărui View, inclusiv timpul de încărcare. O altă funcționalitate pusă la dispoziția programatorilor este posibilitatea capturării unei imagini a View-ului.

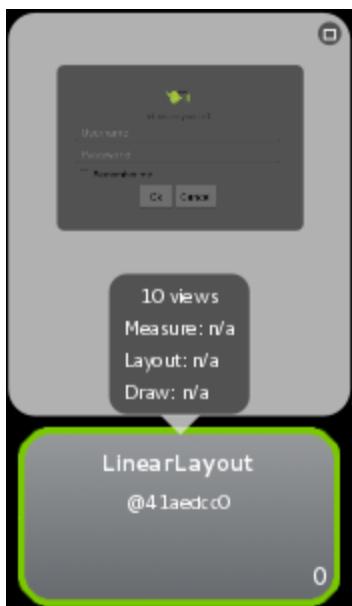
Întrucât prin acest utilitar se poate analiza orice aplicație (componentă) care rulează pe un dispozitiv mobil, el poate fi utilizat și ca sursă de inspirație asupra modului în care este construită interfața grafică.

Layout Inspector este accesibil în Android Studio în *Tools → Android → Android Device Monitor*. Hierarchy View poate fi accesat în Eclipse sub forma unei perspective, care va fi deschisă prin *Window → Open Perspective → Hierarchy View*.

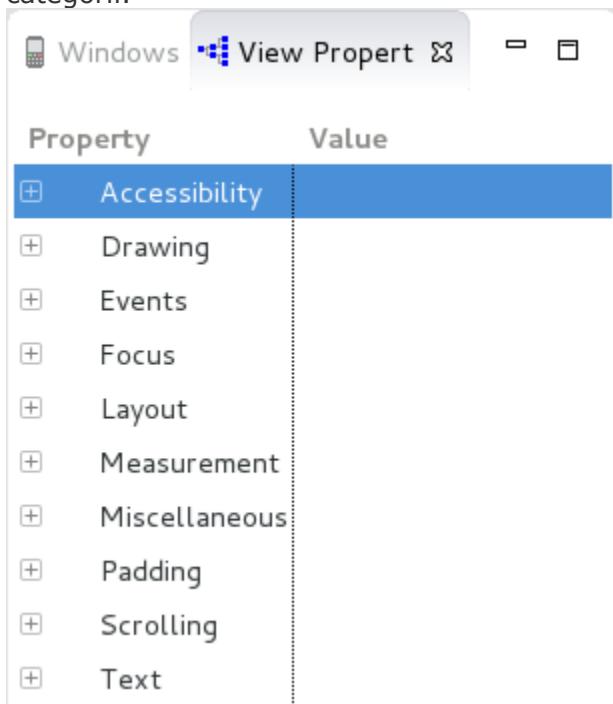


Utilitarul funcționează atât pentru emulatoare cât și cu dispozitive fizice.

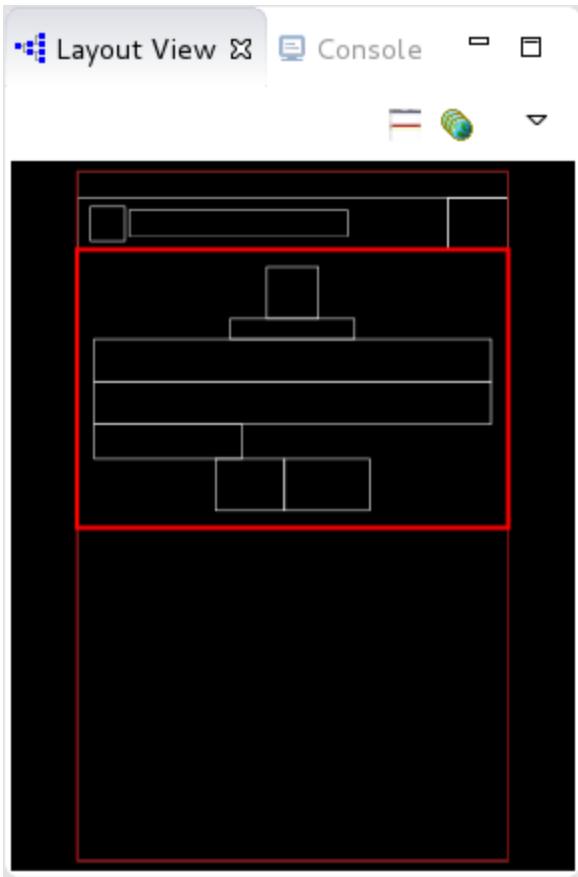
La pornire, în panoul *Windows* (stânga) este afișată lista tuturor dispozitivelor mobile conectate. Se alege un dispozitiv mobil și o componentă ce rulează pe acel dispozitiv (prin expandarea dispozitivului mobil) și se apasă butonul *Load the view hierarchy into the tree view*. Afișarea ierarhiei de View-uri se face în panoul *Tree View* (centru) sub forma unui arbore. Panoul *Tree Overview* (dreapta) va prezenta o imagine de ansamblu a ierarhiei, evidențiindu-se selecția vizibilă la momentul respectiv în fereastra principală. În momentul în care este selectat un element al arborelui, acesta va putea fi previzualizat (se prezintă imaginea sa), indicându-se componența sa, precum și unii parametrii ai acestuia (*measure*, *layout*, *draw*).



Fereastra *View Properties* va conține toate proprietățile unei componente grafice, grupate pe categorii.



Fereastra *Layout View* evidențiază componenta curentă în cadrul interfeței grafice.



## Activitate de Laborator

**1.** În contul Github personal, să se creeze un depozit denumit 'Laborator02'. Inițial, acesta trebuie să fie gol (nu trebuie să bifăți nici adăugarea unui fișier README.md, nici a fișierului .gitignore sau a fișierului LICENSE).

**2.** Să se cloneze în directorul de pe discul local conținutul depozitului la distanță de la <https://www.github.com/eim2017/Laborator02>. În urma acestei operații, directorul Laborator02 va trebui să se conțină un director labtasks ce va deține proiectul Android Studio denumit ActivityLifecycleMonitor, fișierul README.md și un fișier .gitignore care indică tipurile de fișiere (extensiile) ignorate.

```
student@eim2017:~$ git clone https://www.github.com/eim2017/Laborator02.git
```

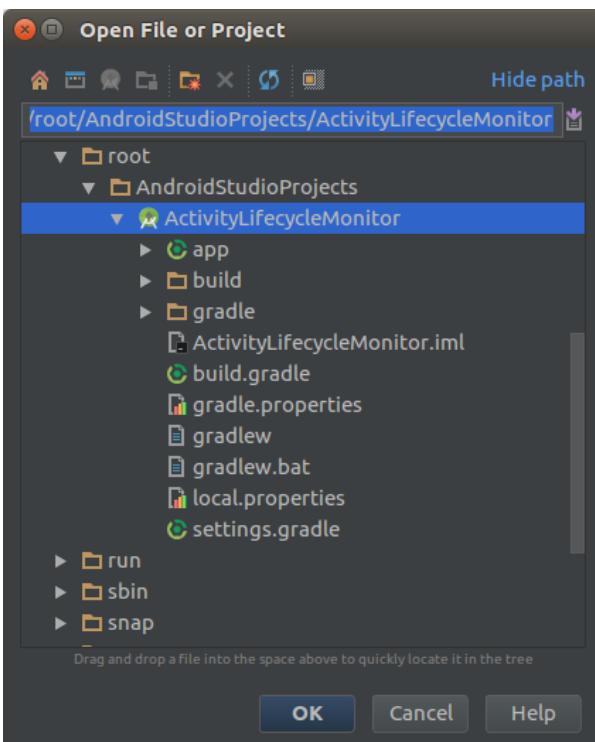
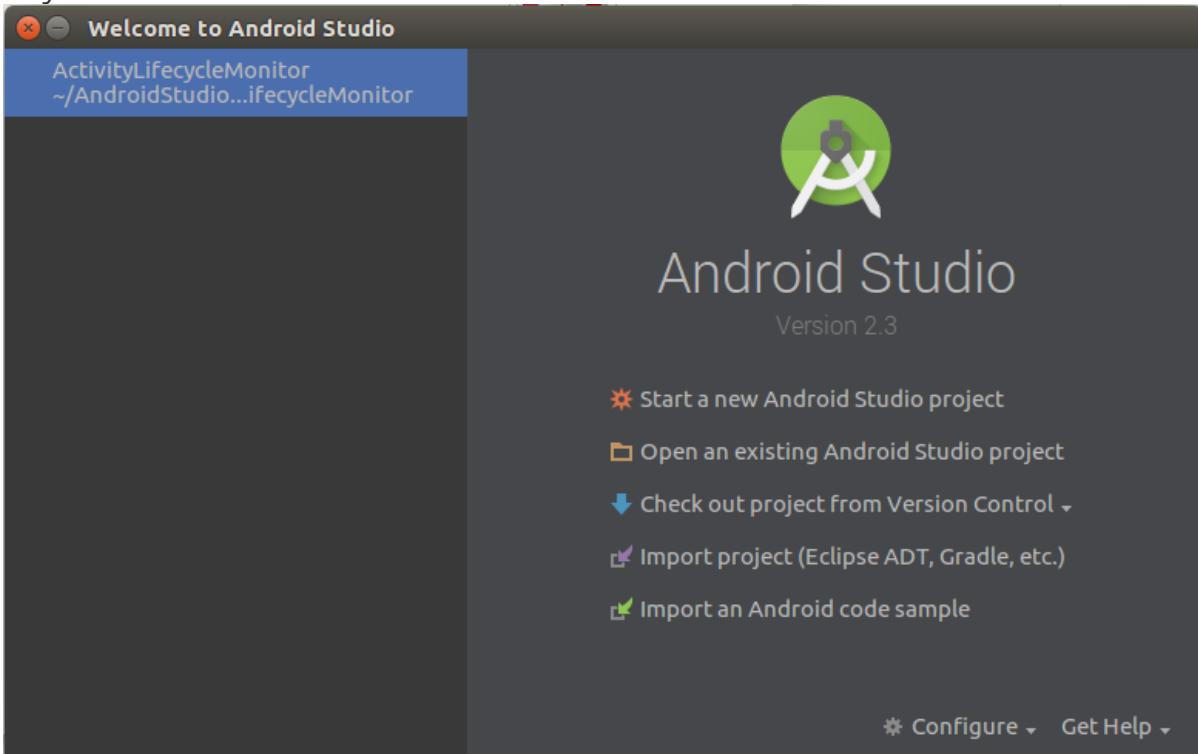
**3.** Să se încarce conținutul descărcat în cadrul depozitului 'Laborator02' de pe contul Github personal.

```
student@eim2017:~$ cd Laborator02
```

```
student@eim2017:~/Laborator02$ git remote add Laborator02_perfectstudent https://github.com/perfectstudent/Laborator02
```

```
student@eim2017:~/Laborator02$ git push Laborator02_perfectstudent master
```

**4.** Să se încarce în mediul integrat de dezvoltare Android Studio proiectul ActivityLifecycleMonitor, folosind opțiunea *Open an Existing Android Studio Project*.



**5.** În clasa LifecycleMonitorActivity din pachetul ro.pub.cs.systems.eim.lab02.activitylifecyclemonitor.graphicuserinterface, să se suprascrie metodele care monitorizează ciclul de viață al unei activități; fiecare dintre acestea va trebui să apeleze metoda **părinte** și să notifice apelarea sa prin **intermediul unui mesaj**, având prioritatea DEBUG și eticheta activitylifecyclemonitor:

```
Log.d(Constants.TAG, "??? method was invoked");
```

1. onRestart()
2. onStart()
3. onResume()
4. onPause()
5. onStop()
6. onDestroy()

**6.** Să se creeze un filtru, denumit ActivityLifecycleMonitor, astfel încât LogCat să afișeze doar mesajele care au eticheta activitylifecycle, generate de aplicația ro.pub.systems.eim.lab02.activitylifecyclemonitor și au cel puțin prioritatea debug.

**7.** Să se modifice mesajul din metoda onCreate(), astfel încât să se indice dacă activitatea a mai fost lansată în execuție anterior sau nu (dacă există o stare a activității care trebuie restaurată).

**8.** Să se inspecteze mesajele care sunt generate la producerea următoarelor evenimente:

1. se apasă butonul *Home*
2. se apasă butonul *Back*
3. se apasă butonul *OK* din cadrul aplicației (indiferent dacă datele de autentificare sunt corecte sau nu)
4. se ține apăsat butonul *Home* o perioadă de timp mai mare
5. se primește un apel telefonic
  - a) pentru AVD, se poate folosi ADM → Emulator Control
  - b) pentru Genymotion se poate simula doar formarea unui număr de telefon

```
6. student@eim2017:/opt/android-sdk-linux/platform-tools$ adb devices
```

```
7. List of devices attached
```

```
8. 192.168.56.101:5555 device
```

```
9. student@eim2017:/opt/android-sdk-linux/platform-tools$ adb -s 192.168.56.101:5555 shell
```

```
10. root@android:/ # am start -a android.intent.action.CALL tel:1122334455
```

```

Starting: Intent { act=android.intent.action.CALL dat=tel:xxxxxxxxxx
}

```

- a. se acceptă con vorbirea
- b. se respinge con vorbirea

11. se rotește ecranul (nu trebuie blocată opțiunea *Auto Rotate* din configurațiile dispozitivului mobil)

Să se observe ce metode sunt apelate în momentul în care se revine în aplicație.

Pe baza mesajelor, să se completeze tabelul de mai jos, indicând ordinea în care s-au apelat metodele respective:

	<code>onCreate (())</code>	<code>onRestart (())</code>	<code>onStart (())</code>	<code>onResume (())</code>	<code>onPause (())</code>	<code>onStop (())</code>	<code>onDestroy (())</code>
1) buton <i>Home</i>			3	4	1	2	
2) buton <i>Back</i>							
3) buton <i>OK</i> din aplicație							
4) buton <i>Home</i> perio adă de timp mai mare							
5) apel telefonic							
a) acceptare							
b) respingere							
6) rotire ecran							

#### 9. Să se dezactiveze opțiunea de salvare a stării.

În fișierul `activity_lifecycle_monitor.xml`, pentru fiecare dintre elementele grafice pentru care se dorește să se dezactiveze opțiunea de salvare a stării, se va completa proprietatea `android:saveEnabled="false"`.

Să se observe care este comportamentul în privința informațiilor reținute în elementele grafice de tip `EditText`, respectiv `CheckBox`, în condițiile în care activitatea este distrusă (se apasă butonul *Home*, astfel încât să se apeleze metodele `onPause()` și `onStop()`, apoi, în DDMS, în panoul *Devices*, se identifică procesul corespunzător aplicației și se oprește folosind butonul *Stop Process*). Să se repornească aplicația din meniul dispozitivului mobil.

10. Să se implementeze metoda `onSaveInstanceState()`, astfel încât, **în condițiile în care este bifat elementul grafic de tip `CheckBox`, să se salveze informațiile din interfață cu utilizatorul.**

Se vor folosi metodele `putString()` și `putBoolean()` ale clasei `Bundle`.

Cheile sub care vor fi identificate valorile salvate sunt definite în interfața `Constants` din pachetul `ro.pub.cs.systems.eim.lab02.activitylifecyclermonitor.general`.

Verificarea faptului că un element grafic de tip `CheckBox` este bifat se face prin intermediul metodei `isChecked()`.

Să se observe comportamentul aplicației în condițiile producerii evenimentului descris anterior.

**11.** Să se implementeze metoda `onRestoreInstanceState()` astfel încât să se restaureze starea elementelor grafice. Să se observe comportamentul aplicației în condițiile producerii evenimentului descris anterior.

Să se transfere comportamentul de restaurare a stării pe metoda `onCreate()` și să se identifice diferențele de implementare ([Hint](#)).

**12.** Să se încarce modificările realizate în cadrul depozitului 'Laborator02' de pe contul Github personal, folosind un mesaj sugestiv.

```
student@eim2017:~/Laborator02$ git add labtasks/*
student@eim2017:~/Laborator02$ git commit -m "implemented tasks for laboratory 02"
student@eim2017:~/Laborator02$ git push Laborator02_perfectstudent master
```

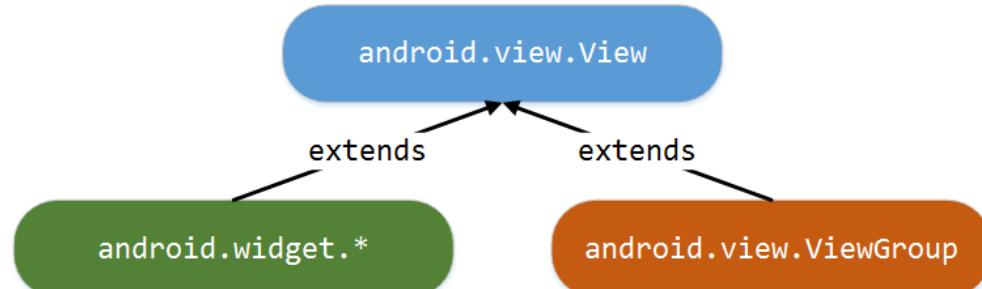
## Laborator 03. Proiectarea Interfețelor Grafice

### Clase Android utilizate pentru definirea unei interfețe grafice

În cadrul unei aplicații Android, o interfață grafică conține elemente care au capabilitatea de a afișa informații către utilizator, în diferite formate, respectiv de a interacționa cu acesta, preluând datele necesare realizării diverselor fluxuri operaționale din cadrul aplicației. Există și o categorie specială de controale grafice, responsabile numai cu gestiunea mecanismului de disponere a celorlalte componente, determinând modul în care vor fi plasate în cadrul ferestrei precum și coordonatele la care vor fi poziționate.

Structura unei interfețe grafice este arborescentă. Întotdeauna, elementul rădăcină va fi un control care gestionează modul în care sunt dispuse componente sale, în timp ce elementele frunză sunt controale grafice propriu-zise, vizibile pe ecran, cu o funcționalitate bine delimitată. Pe toate celelalte niveluri (intermediare) din această ierarhie se pot regăsi elemente de ambele tipuri (atât elemente grafice cât și mecanisme de disponere a conținutului - care controlează în acest fel o secțiune din cadrul interfeței cu utilizatorul).

Clasa `android.view.View` reprezintă baza pentru construirea oricărei interfețe grafice dintr-o aplicație Android. Ea definește o zonă rectangulară a dispozitivului de afișare (écran), majoritatea controalelor grafice și a mecanismelor de disponere a conținutului fiind derivate din aceasta.



**1.** Cele mai multe **elemente grafice** sunt definite în pachetul android.widget, fiind implementate controale care implementează cele mai multe dintre funcționalitățile uzuale (etichete, câmpuri text, controale pentru redarea de conținut multimediat - imagini, filme -, butoane, elemente pentru gestiunea datei calendaristice și a timpului).

Trebuie să se facă distincția între control, afișat în interfața grafică a unei activități, definit în pachetul android.widget și extensiile aplicațiilor, afișate în ecranul principal al dispozitivului mobil, cunoscute sub denumirea de widget-uri, dar care fac parte din clasa android.appwidget.AppWidget.

**2.** Controalele pentru gestiunea **mecanismului de disponere a conținutului** au rolul de a determina modul în care sunt afișate elementele conținute. Acestea sunt derivate din clasa android.view.ViewGroup, definind mai multe reguli prin care se determină poziția la care vor fi plasate componentele pe care le includ.

## Mecanisme pentru construirea unei interfețe grafice

O interfață grafică poate fi construită în două moduri:

1. prin definirea elementelor componente și a modului lor de disponere în cadrul unui fișier .xml, asociat fiecarii activități (sau fragment) în parte, situație adecvată cazurilor în care interfața grafică este statică;
2. programatic, prin instanțierea unor obiecte de tipul elementelor componente direct în codul sursă (cu stabilirea proprietăților respective) al activității (sau fragmentului), abordare potrivită pentru situațiile în care interfața grafică are o structură dinamică (este actualizată în funcție de unele condiții specifice identificate în momentul execuției).

De regulă, se preferă ca interfața grafică să fie definită în cadrul unui fișier .xml pentru fiecare fereastră din cadrul aplicației Android, întrucât acesta este mult mai ușor de întreținut, separând sarcinile ce țin de proiectare propriu-zisă de cele care țin de programare (putând fi realizate astfel de persoane diferite). Totuși, pentru situațiile în care interfața grafică nu este cunoscută la momentul compilării sau pentru cazul în care interfața grafică trebuie modificată în funcție de anumite condiții identificate în momentul compilării, pot fi utilizate metode programatice spre a obține o astfel de funcționalitate.

## Construirea unei interfețe grafice în XML

Pentru fiecare activitate se va construi un fișier .xml în directorul res/layout care va descrie conținutul interfeței grafice precum și modul de disponere al controalelor componente.

În cazul în care se dorește să se definească o altă interfață grafică, adaptată la schimbarea orientării ecranului, se va folosi directorul res/layout-land în care conținutul fișierului .xml (având aceeași denumire) va fi modificat corespunzător. Selecția interfeței grafice în funcție de dimensiunile suprafetei de afișare este realizată în mod automat de sistemul de operare Android.

Pentru fiecare resursă de acest tip, se va genera o referință în clasa layout din fișierul generat R.java, care va putea fi utilizată pentru încărcarea interfeței grafice în cadrul metodei onCreate(Bundle savedInstanceState).

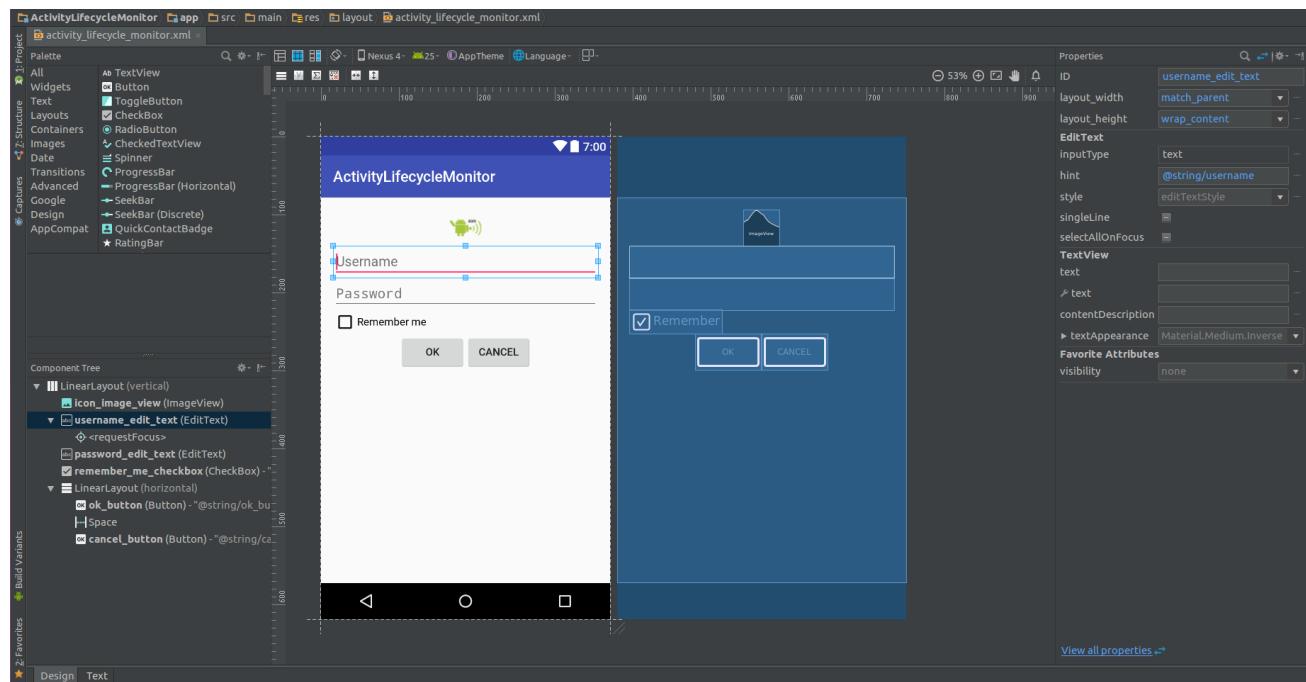
```
setContentView(R.layout.activity_layout_sample_1);
```

După încărcarea propriu-zisă a elementelor din cadrul interfeței grafice vor putea fi obținute referințe către ele prin intermediul metodei `findViewById()`, care:

- primește ca parametru un identificator (întreg) definit (automat) în clasa `id` din fișierul generat `R.java` pentru toate componentele din cadrul interfeței grafice care au definit atributul `android:id` (pe baza căruia pot fi referite);
- returnează un obiect de tip `android.view.View`, fiind necesar să se realizeze conversia explicită către tipul de control grafic dorit.

```
TextView greetingTextView = (TextView) findViewById(R.id.greeting_text_view);
```

Mediul integrat de dezvoltare Android Studio pune la dispoziția dezvoltatorilor atât un utilitar conținând biblioteci de controale, în care interfața grafică poate fi definită prin operații de tip drag-and-drop, cât și un editor text în care proprietățile acestor elemente pot fi specificate manual. Schimbările realizate într-o dintre aceste ferestre sunt transpusă automat și în celalaltă, astfel încât modificările din mediul vizual generează etichetele XML corespunzătoare, în timp ce efectul unor proprietăți stabilite în fișierul XML poate fi vizualizat imediat în editorul grafic.



Astfel, utilitarul vizual poate fi accesat din panoul *Design*, putând fi specificate (prin selecția dintr-o listă) dispozitivul mobil pentru care se proiectează interfața grafică, orientarea ecranului (precum și alte moduri în care se poate găsi acesta), tema aplicației (stilul folosit), activitatea căreia îl este asociat, localizarea precum și nivelul de API.

În cadrul bibliotecii de controale, organizarea elementelor grafice se face prin împărțirea lor în mai multe categorii:

- forme de bază (*Widgets*);
- câmpuri text (*Texts*);

- mecanisme de dispunere a conținutului (*Layouts*);
- controale grafice ce pot conține alte componente(*Containers*);
- resurse multimedia (*Images*);
- elemente pentru gestiunea datei calendaristice și a timpului (*Date*);
- animații (*Transitions*);
- componente complexe de interacțiune cu utilizatorul (*Advanced*);
- componente Google (*Google*);
- elemente grafice definite în Material Design (*Design*);
- obiecte definite în bibliotecile de suport (*AppCompat*);

Structura interfeței grafice (ierarhia de controale grafice) poate fi (pre)vizualizată, pentru fiecare obiect în parte indicându-se identificatorul său (dacă este definit) și tipul (între paranteze rotunde), în secțiunea *Component Tree*.

Pentru fiecare control grafic, atributele sale pot fi gestionate în secțiunea *Properties*, fiind afișate toate proprietățile pe care le poate avea (proprii sau moștenite de la părinte), împreună cu valorile pe care le poate lua, utilizatorul putând alege dintre acestea.

#### activity layout sample\_1.xml

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

 tools:context="ro.pub.cs.systems.eim.lab03.examples.layoutsample1.
 LayoutSample1Activity" >

 <EditText
 android:id="@+id/introduce_yourself_edit_text"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:ems="10"
 android:hint="@string/introduce_yourself"
 android:inputType="text" >

 <requestFocus />

 </EditText>

 <CheckBox
 android:id="@+id/display_identity_check_box"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_alignLeft="@+id/introduce_yourself_edit_text"
 android:layout_below="@+id/introduce_yourself_edit_text"
 android:text="@string/display_identity" />

</Button>
```

```

 android:id="@+id/submit_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_alignLeft="@+id/display_identity_check_box"
 android:layout_below="@+id/display_identity_check_box"
 android:text="@string/submit" />

 <TextView
 android:id="@+id/greeting_text_view"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:layout_alignLeft="@+id/submit_button"
 android:layout_below="@+id/submit_button"
 android:text="@string/greeting"
 android:textSize="32sp"
 android:gravity="center"
 android:alpha="0" />

</RelativeLayout>

```

Fiecare control din cadrul interfeței grafice va fi reprezentat printr-un element corespunzător, denumirea acestuia fiind identică cu cea a clasei care îi implementează funcționalitatea (de regulă, din pachetul `android.widget`).

Un control care va fi referit ulterior (fie în codul sursă, fie în fișierul XML) trebuie să aibă asociat un identificator, indicat prin proprietatea `android:id`. Acesta are forma `@+id/identificator` (în momentul în care este definit), respectiv `@+id/identificator` pentru referirile ulterioare. Pentru fiecare componentă ce definește un element grafic, se generează o referință în clasa `id` din fișierul `R.java`.

Elementele interfeței grafice sunt caracterizate prin anumite proprietăți, cum ar fi poziționarea, dimensiunile, conținutul pe care îl afișează, tipurile de date acceptate de la utilizator, informațiile ajutătoare. Fiecare parametru va fi indicat prin sintaxa `android:proprietate="valoare"` unde proprietate și valoare trebuie să respecte restricțiile definite în clasa ce descrie controlul respectiv.

## Dezvoltarea programatică a unei interfețe grafice

O interfață grafică poate fi definită și în codul sursă, într-un mod similar. Se creează inițial un obiect container (de tip `Layout`, derivat din `android.view.ViewGroup`) care va cuprinde toate controalele, acesta fiind argumentul cu care va fi apelată metoda `setContentView()`.

Pentru fiecare control vor fi specificate (manual, prin apelul metodei corespunzătoare) diferitele caracteristici, asociindu-i-se și un identificator (uzual, acesta poate fi orice număr întreg). Pentru fiecare proprietate a unui control grafic, sunt definite programatic metodele de tip getter și setter corespunzătoare.

În situația în care unui control nu i se asociază un identificator prin apelul metodei `setId()`, valoarea acestui parametru va fi `NO_ID`, astfel încât acesta nu va mai putea fi referit în cod (spre exemplu, pentru poziționarea elementelor interfeței grafice relativ, unele față de altele).

Ulterior, controlul va trebui asociat containerului din care face parte, prin intermediul metodei `addView()`, specificându-se modul în care va fi poziționat (precum și dimensiunile sale) prin intermediul unui obiect de tip `LayoutParams`, specific mecanismului de dispunere respectiv.

[LayoutSample2Activity.java](#)

```
public class LayoutSample2Activity extends Activity {

 final private static long TRANSPARENCY_EFFECT_DURATION = 5000;

 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);

 RelativeLayout container = new RelativeLayout(this);
 container.setLayoutParams(new
 LayoutParams(LayoutParams.MATCH_PARENT,
 LayoutParams.MATCH_PARENT));

 final EditText introduceYourselfEditText = new EditText(this);
 introduceYourselfEditText.setId(1);
 introduceYourselfEditText.setEms(10);

 introduceYourselfEditText.setHint(getString(R.string.introduce_yourself));

 introduceYourselfEditText.setInputType(InputType.TYPE_CLASS_TEXT);
 introduceYourselfEditText.setFocusable(true);
 RelativeLayout.LayoutParams
 introduceYourselfEditTextLayoutParams = new
 RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.WRAP_CONTENT,
 RelativeLayout.LayoutParams.WRAP_CONTENT);
 container.addView(introduceYourselfEditText,
 introduceYourselfEditTextLayoutParams);

 final CheckBox displayIdentityCheckBox = new CheckBox(this);
 displayIdentityCheckBox.setId(2);

 displayIdentityCheckBox.setText(getString(R.string.display_identity));
 RelativeLayout.LayoutParams
 displayIdentityCheckBoxLayoutParams = new
 RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.WRAP_CONTENT,
 RelativeLayout.LayoutParams.WRAP_CONTENT);

 displayIdentityCheckBoxLayoutParams.addRule(RelativeLayout.ALIGN_LEFT,
 introduceYourselfEditText.getId());

 displayIdentityCheckBoxLayoutParams.addRule(RelativeLayout.BELOW,
 introduceYourselfEditText.getId());
 container.addView(displayIdentityCheckBox,
 displayIdentityCheckBoxLayoutParams);

 final Button submitButton = new Button(this);
 submitButton.setId(3);
 submitButton.setText(getString(R.string.submit));
 RelativeLayout.LayoutParams submitButtonLayoutParams = new
 RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.WRAP_CONTENT,
 RelativeLayout.LayoutParams.WRAP_CONTENT);
 submitButtonLayoutParams.addRule(RelativeLayout.ALIGN_LEFT,
 displayIdentityCheckBox.getId());
```

```

 submitButtonLayoutParams.addRule(RelativeLayout.BELOW,
displayIdentityCheckBox.getId());
 container.addView(submitButton, submitButtonLayoutParams);

 final TextView greetingTextView = new TextView(this);
 greetingTextView.setId(4);
 greetingTextView.setText(getString(R.string.greeting));
 greetingTextView.setTextSize(32);
 greetingTextView.setGravity(Gravity.CENTER);
 greetingTextView.setAlpha(0);
 RelativeLayout.LayoutParams greetingTextViewLayoutParams = new
RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.MATCH_PARENT,
RelativeLayout.LayoutParams.MATCH_PARENT);

 greetingTextViewLayoutParams.addRule(RelativeLayout.ALIGN_LEFT,
submitButton.getId());
 greetingTextViewLayoutParams.addRule(RelativeLayout.BELOW,
submitButton.getId());
 container.addView(greetingTextView,
greetingTextViewLayoutParams);

 submitButton.setOnClickListener(new View.OnClickListener() {

 @Override
 public void onClick(View view) {
 String identity =
introduceYourselfEditText.getText().toString();

 greetingTextView.setText(getResources().getString(R.string.greetin
g).replace("???",
(displayIdentityCheckBox.isChecked())?identity:getResources().getS
tring(R.string.anonymous)));
 greetingTextView.setAlpha(1);
 AlphaAnimation fadeEffect = new AlphaAnimation(1.0f, 0.0f);
 fadeEffect.setDuration(TRANSPARENCY_EFFECT_DURATION);
 fadeEffect.setFillAfter(true);
 greetingTextView.setAnimation(fadeEffect);
 }
 });
 }

 setContentView(container);
}

// ...
}

```

## Controle în Android (widget-uri)

În Android, un control (pentru care se utilizează și denumirea de widget) este de regulă derivat din clasa `android.view.View`, unde sunt definite câteva caracteristici de bază cu privire la dimensiuni și la modul de disponere:

ATRIBUT	TIP OBIECT	DESCRIERE
<code>layout_width</code>	<code>View / ViewGroup</code>	lățimea obiectului

layout_height	View / ViewGroup	înălțimea obiectului
layout_marginTop	View / ViewGroup	spațiu suplimentar ce trebuie alocat în partea de sus a obiectului
layout marginBottom	View / ViewGroup	spațiu suplimentar ce trebuie alocat în partea de jos a obiectului
layout_marginLeft	View / ViewGroup	spațiu suplimentar ce trebuie alocat în partea din stânga a obiectului
layout_marginRight	View / ViewGroup	spațiu suplimentar ce trebuie alocat în partea din dreapta a obiectului
layout_gravity	View	modul de poziționare a elementelor componente în cadrul unui container
layout_weight	View	proportia pe care o are controlul, raportată la întregul conținut al containerului
layout_x	View / ViewGroup	poziția pe coordonata x
layout_y	View / ViewGroup	poziția pe coordonata y

Unele dintre aceste proprietăți pot fi specificate și pentru containerele în care sunt cuprinse controalele, derivate din clasa `android.view.ViewGroup`.

Valorile pe care le pot lua atributele `layout_width` și `layout_height` sunt:

- `match_parent` - dacă se dorește ca obiectul să ocupe tot spațiul pe care îl pune la dispozitie containerul său;

Deși există și posibilitatea de a indica valoarea `fill_parent` pentru lățimea și înălțimea unui control, aceasta trebuie evitată, întrucât a fost înlocuită începând cu nivelul de API 8.

- `wrap_content` - dacă se dorește ca obiectul să fie restrâns la dimensiunile conținutului său.

În situația în care nu se specifică cel puțin valorile `layout_width` și `layout_height` pentru un anumit control, se va genera o excepție întrucât interfața grafică nu poate fi încărcată corespunzător.

De asemenea, pot fi specificate valori absolute, exprimate în una din unitățile de măsură:

- **dp - pixel independent de rezoluție**

Se recomandă să se utilizeze această unitate de măsură în momentul în care se specifică dimensiunea unui control în cadrul unui container. Se asigură astfel faptul că se utilizează o proporție adecvată pentru un control, indiferent de rezoluția ecranului, Android scalându-i dimensiunea automat.

- **sp - pixel independent de scală**, echivalent cu `dp`

Se recomandă să se utilizeze această unitate de măsură în momentul în care se specifică dimensiunea unui set de caractere, cu care va fi afișat un text.

- **pt - punct**, echivalent cu 1/72 inchi, bazat pe dimensiunile ecranului
- **px - pixel**, corespunzător unui pixel al dispozitivului mobil

Utilizarea acestei unități de măsură nu este recomandată, întrucât interfața grafică definită în acești termeni nu se va afișa corect pe dispozitive mobile cu altă rezoluție a ecranului.

Specificarea dimensiunii ecranului unui dispozitiv mobil se face prin indicarea numărului de pixeli pe orizontală și pe verticală. Pentru a obține densitatea (rezoluția) dispozitivului respectiv, se împarte dimensiunea exprimată în pixeli la dimensiunea exprimată în inchi. Se va compara valoarea obținută cu una dintre valorile standard definite pentru încadrarea dispozitivului mobil într-o anumită categorie de rezoluție:

- 120 dpi - ldpi (Low Density)
- 160 dpi - mdpi (Medium Density)
- 240 dpi - hdpi (High Density)
- 320 dpi - xhdpi (Extra High Density)
- 480 dpi - xxhdpi (Extra Extra High Density)

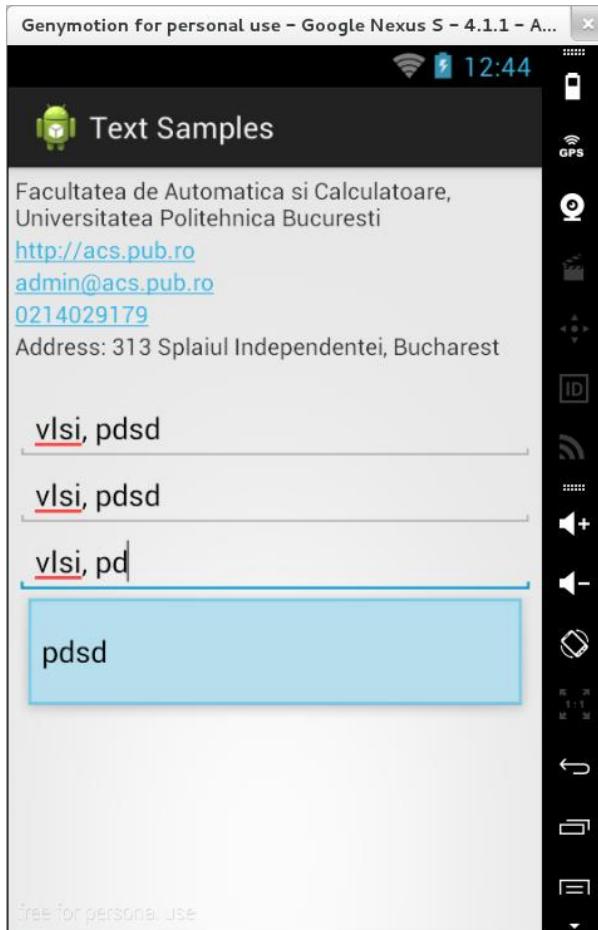
Formula de conversie între pixeli ( $\text{px}$ ) și pixeli independenți de rezoluție ( $\text{dp}$ ) este:  
 $\text{px} = \text{dp} * (\text{resolution\_category}) / 160$

Se observă că 1  $\text{px}$  este echivalent cu 1  $\text{dp}$  pe un ecran cu rezoluția 160 dpi, considerată drept referință în Android.

Dimensiunea (exprimată în pixeli) a unui control poate fi obținută apelând metodele `getWidth()`, respectiv `getHeight()`.

## Controale de tip text

Android pune la dispoziția programatorilor un set complet de controale de tip text, dintre care cele mai utilizate sunt `TextView`, `EditText`, `AutoCompleteTextView` și `MultiCompleteTextView`.



## TextView

Controlul de tip `TextView` este utilizat pentru afișarea unui text către utilizator, fără ca acesta să aibă posibilitatea de a-l modifica.

Conținutul pe care îl afișează un obiect `TextView` este indicat de proprietatea `text`. De regulă, acesta referă o constantă definită în resursa care conține sirurile de caractere utilizate în cadrul aplicației. Gestiona acestei atribut poate fi realizată prin metodele `getter` și `setter` respective.

Metoda `getText()` întoarce un parametru de tip `CharSequence`, astfel încât trebuie realizată conversia sa la o implementare a acestei interfețe (cel mai frecvent, `String`), înainte de a putea folosi valoarea respectivă.

Pentru un control de tip `TextView` pot fi specificate mai multe proprietăți, printre care `minLines`, `maxLines`, `minEms`, `maxEms` și `textcolor`.

Pentru a defini lățimea și înălțimea unui control de tip `TextView`, este de preferat să se utilizeze următoarele unități de măsură:

- **ems** (pentru lățime) - termen folosit în tipografie relativ la dimensiunea punctului unui set de caractere, oferind un control mai bun asupra modului în care este vizualizat textul, independent de dimensiunea propriu-zisă a semnelor respective;

- **număr de linii** (pentru înălțime) - garantează afișarea netrunchiată a textului, indiferent de dimensiunea caracterelor;

Pentru a avertiza utilizatorul de faptul că textul nu a fost afișat complet, se poate folosi parametrul `ellipsize`, care atașează sirul de caractere (...) în finalul secvenței vizibile. În cazul în care textul este trunchiat și totuși textul afișat are sens, fără a exista indicii că ar mai fi urmat și altceva, utilizarea acestui atribut este foarte utilă.

O componentă de tip `TextView` NU implementează doar funcționalitatea unei etichete, ea putând fi utilizată și pentru a realiza anumite acțiuni, în cazul în care conținutul pe care îl afișează are un format cunoscut:

- `android:autoLink="none"` - dezactivează orice tip de acțiune care poate fi atașată unui text;
- `android:autoLink="web"` - pentru pagini Internet care vor fi accesate prin intermediul unui navigator;
- `android:autoLink="email"` - pentru adrese de poștă electronică la care se vor trimite mesaje prin intermediul unei aplicații dedicate;
- `android:autoLink="phone"` - pentru numere de telefon care se doresc a fi formate;
- `android:autoLink="map"` - pentru coordonate GPS care se doresc a fi vizualizate pe hartă;
- `android:autoLink="all"` - activează orice tip de acțiune care poate fi atașată unui text.

Se pot folosi și combinații ale acestor tipuri de legături, agregate folosind operatorul | (pe biți).

Nu întotdeauna este detectat tipul corect de informație și în unele cazuri se poate realiza transferul către un alt tip de activitate sau cu informații eronate.

Programatic, comportamentul unui obiect de acest tip poate fi stabilit prin intermediul metodei `setAutoLinkMask()`, care va primi ca parametru o constantă a clasei `Linkify`.

Alternativ, poate fi utilizată metoda statică `addLinks()` din clasa `Linkify`, apelată cu un parametru de tip `TextView` și un parametru care indică tipul de conținut.

În situația în care se dorește ca informația să fie afișată într-un mod care să o evidențieze, însă fără a permite utilizatorului să lanseze în execuție o altă activitate prin intermediul ei, se poate utiliza atributul `linksClickable` (având valoarea `false`).

## EditText

`EditText` este o componentă utilizată pentru obținerea unui text de la utilizator. Implementarea sa pornește de la obiectul de tip `TextView`, astfel încât sunt moștenite toate proprietățile sale.

În mod obișnuit, valoarea introdusă va fi afișată doar pe o linie. Dacă se dorește redimensionarea controlului pe măsură ce este introdus un text de dimensiuni mai mari, va trebui specificată proprietatea `inputType="textMultiLine"`. De asemenea, se poate specifica explicit numărul de linii prin intermediul proprietății `lines` (în cazul în care aceasta nu este specificată, câmpul va fi redimensionat în mod automat pentru ca textul introdus să

poată fi afișat; totuși indicarea acestui atribut este util pentru că impune o dimensiune fixă, utilizatorul având posibilitatea de a naviga în cadrul textului prin operația de derulare).

De asemenea, există posibilitatea indicării unei sugestii, care va fi afișată pe fundal, semitransparentă, prin intermediul proprietății `hint`. Aceasta precizează utilizatorului ce fel de informații ar trebui specificate prin intermediul controlului. Ea va dispare în mod automat în momentul în care utilizatorul începe să introducă textul respectiv.

Printre funcționalitățile elementului de tip `EditText` se numără și posibilitatea de a accepta un text (proprietatea `inputType` are valoarea `textAutoCorrect`) sau de a accepta doar valori care respectă un anumit format (număr de telefon, adresă de poștă electronică, parolă, număr zecimal (cu sau fără semn), dată calendaristică, oră). Totodată, utilizatorul poate fi forțat să introducă valori care respectă doar un anumit format, prin intermediul metodei `setFilters()` care primește ca parametru un tablou de filtre (asigurându-se în acest mod îndeplinirea concomitentă a condițiilor specificate de fiecare în parte):

- predefinite în clasa [InputFilter](#)
  - `InputFilter.AllCaps` - toate caracterele trebuie scrise cu majusculă;
  - `InputFilter.LengthFilter` - sunt acceptate doar șiruri de caractere care au o dimensiune fixă (precizată ca parametru în constructor);
- definite de utilizator, prin implementarea metodei `filter()` din interfața `InputFilter`, indicându-se valoarea cu care se înlocuiește o anumită secvență.

Operația de tip apăsare lungă în cadrul unui control de tip `EditText` invocă meniul contextual, care oferă facilități pentru copierea unui text într-o / dintr-o zonă de memorie tampon (*eng. copy-cut-paste*), schimbarea mecanismului de introducere a textului, modificarea dicționarului ce conține cele mai frecvent folosite cuvinte. De asemenea, poate fi selectată (evidențiată) un fragment din textul respectiv (programatic, această funcționalitate poate fi obținută prin metodele `setSelection()` / `selectAll()`).

## AutoCompleteTextView

Controlul de tip `AutoCompleteTextView` indică utilizatorului anumite valori cu care poate fi completat, în funcție de conținutul pe care acesta îl are acesta la un moment dat, pe baza unei liste de sugestii. Momentul la care sistemul de operare va oferi o sugestie este controlat prin intermediul proprietății `completionThreshold`, care indică numărul (minim) de caractere pe baza căruia se face potrivirea între șirul de caractere introdus de utilizator și valorile existente în lista de propuneri. Se poate afișa și un text sugestiv în cadrul listei de sugestii, prin proprietatea `completionHint`.

```
AutoCompleteTextView coursesAutoCompleteTextView =
 (AutoCompleteTextView) findViewById(R.id.courses_auto_complete_text_view);

String[] suggestions = {"vlsi", "eim", "ts", "std", "so2", "idp", "ia", "scad",
 "pw", "ecom"};
ArrayAdapter<String> coursesArrayAdapter = new ArrayAdapter<String>(this,
 android.R.layout.simple_dropdown_item_1line, suggestions);

coursesAutoCompleteTextView.setAdapter(coursesArrayAdapter);
```

Sugestia este oferită pentru întregul conținut al componentei, astfel încât dacă se dorește specificarea mai multor cuvinte, nu se va lua în calcul fiecare dintre acestea.

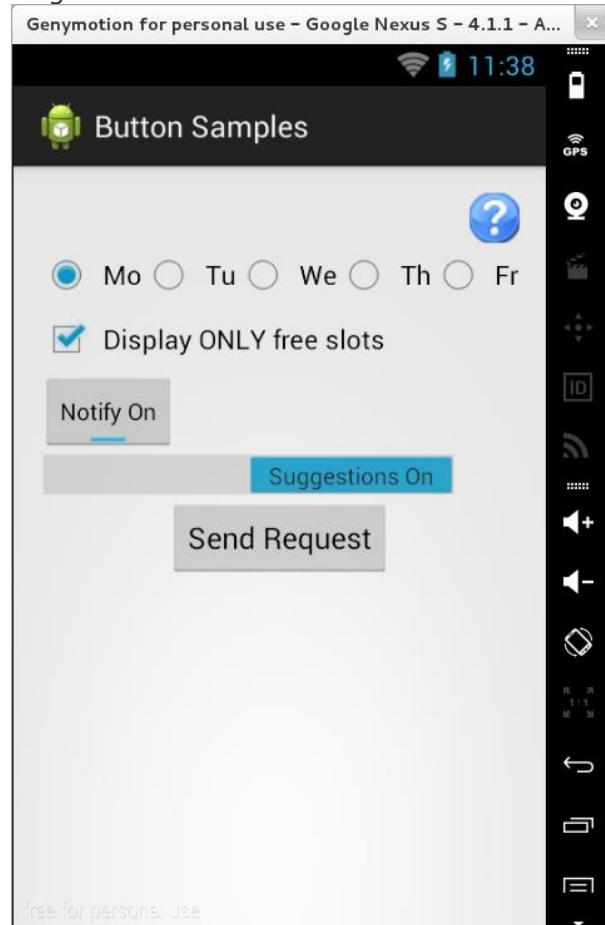
Un astfel de control nu constrânge utilizatorul să introducă doar valori din lista de sugestii, conținutul controlului grafic putând fi oarecare.

## MultiAutoCompleteTextView

Controlul de tip MultiAutoCompleteTextView permite ca sugestiile oferite să țină cont de fiecare componentă a textului introdus. Mai exact, va fi specificat un delimitator (prin intermediul metodei `setTokenizer()`) după apariția căruia vor fi luate în considerare caracterele pentru determinarea sugestiei. Android pune la dispoziție câteva clase standard pentru delimitatori (`CommaTokenizer` - separare prin virgulă, `Rfc822Tokenizer` - conținutul de tip adresă de poștă electronică) însă se pot utiliza și clase definite de utilizator, care implementează interfața `MultiAutoCompleteTextView.Tokenizer`.

## Controle de tip buton

În Android pot fi utilizate mai multe tipuri de butoane, între care `Button` (buton ce are atașat un text), `ImageButton` (buton ce are atașată o imagine), `ToggleButton`, `CheckBox` și `Switch` (controale ce pot avea două stări - selectat sau nu), `RadioButton` / `RadioGroup` (grupuri de butoane ce pot avea două stări - selectat sau nu, asigurându-se totodată exclusiunea mutuală între acestea).



Button

O componentă de tip buton ce are atașat un text este definită de clasa `android.widget.Button`, fiind caracterizată prin proprietatea `text`, ce conține mesajul pe care acesta îl va afișa.

În clasa `android.R.string` sunt definite mai multe constante care pot fi utilizate ca etichete pentru butoane (*OK, Cancel, Yes, No*).

Se obișnuiește ca butoanele să fie stilizate, în sensul că sunt afișate fără un chenar, prin precizarea atributului `style`, având valoarea "`?android:attr/borderlessButtonStyle`".

Întrucât un buton este un tip de control ce interacționează cu utilizatorul, pentru aceasta trebuie definită o clasă ascultător (ce implementează `View.OnClickListener`) pentru tratarea evenimentelor de tip apăsare. Aceasta definește o metodă `onClick()` ce primește ca parametru componenta (de tip `android.view.View`) care a generat evenimentul respectiv. Există două mecanisme de tratare a unui eveniment:

1. precizarea metodei care tratează evenimentul în fișierul XML corespunzător activității, având dezavantajul că nu se pot transmite parametrii clasei ascultător

```
 android:onClick="myButtonClickHandler"
```

2. definirea unei clase ascultător în codul sursă
  - a. clasă dedicată, având dezavantajul că nu poate accesa decât acele resurse ale activității care sunt publice (referința către obiectul de tip `Activity` trebuind să fie transmisă ca parametru);
  - b. folosirea unei clase interne cu nume în cadrul activității, având avantajul posibilității accesării tuturor resurselor din cadrul acesteia, fără necesitatea transmiterii unei referințe către ea; **aceasta este abordarea recomandată pentru tratarea evenimentelor pentru orice tip de control**;
  - c. folosirea unei clasei interne anonime în cadrul activității, având dezavantajul că trebuie (re)definită pentru fiecare control în parte, ceea ce se poate dovedi ineficient în cazul în care se poate elabora un cod comun pentru tratarea evenimentelor mai multor componente din cadrul interfeței grafice;
  - d. utilizarea clasei corespunzătoare activității ca ascultător, prin implementarea interfeței respective și a metodei aferente, având dezavantajul unei scalări ineficiente în cazul în care există mai multe controale pentru care tratarea evenimentului se realizează în mod diferit.

În cazul folosirii de clase interne, membrii din clasa părinte ce se doresc a fi accesăți trebuie să aibă imutabili (trebuie declarate cu atributul `final`).

## ImageButton

Un obiect de tip `ImageButton` (definit de clasa `android.widget.ImageButton`) este utilizat pentru afișarea unei imagini, specificată în fișierul XML prin proprietatea `android:src` sau programatic prin intermediul metodei `setImageResource()`.

Imaginea asociată unui buton de tip `ImageButton` este o resursă din directorul `/res/drawable/` (accesibilă în XML prin adnotarea `@drawable/` sau programatic prin referința sa din clasa `R.drawable`).

Formatele suportate sunt `png`, `jpeg`, `gif` și `bmp`.

Dacă se dorește să se omită afișarea propriu-zisă a butonului, reținându-se doar imaginea sa, se poate specifica un fundal vid (în XML, `android:background="@null"` sau în codul sursă se poate apela `setBackground(null)`).

Un buton de tip `ImageButton` poate avea și una din următoarele stări:

- `focus` - dacă este controlul ce captează toate evenimentele (în cazul în care selecția a fost mutată pe el)
- `pressed` - dacă este apăsat de utilizator (înainte ca evenimentul corespunzător să fie procesat)

În funcție de aceste stări, unui buton de tip `ImageButton` i se pot asocia diferite imagini, specificate într-un fișier XML din directorul `/res/drawable`, care va fi referit ulterior ca sursă a acestuia:

imagebuttonselector.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector
 xmlns:android="http://schemas.android.com/apk/res/android">
 <item android:state_pressed="true"
 android:drawable="@drawable/button_pressed" />
 <item android:state_focused="true"
 android:drawable="@drawable/button_focused" />
 <item android:drawable="@drawable/button" />
</selector>

<Button
 ...
 android:src="@drawable/imagebuttonselector"
 ... />
```

Ordinea elementelor în fișierul XML pentru realizarea selecției este important întrucât Android îl va verifica pe fiecare în parte pentru a vedea dacă acesta corespunde stării controlului. De aceea, imaginea pentru starea normală a controlului trebuie să se regăsească întotdeauna după toate celelalte cazuri particulare, în caz contrar aceasta fiind afișată întotdeauna de vreme ce nu impune nici o restricție asupra componentei.

## ToggleButton

Un obiect `ToggleButton` (definit de `android.widget.ToggleButton`) este un tip de buton care are asociată o stare cu două valori (selectat și neselectat), asemenea unui checkbox sau radiobutton. Implicit, atunci când este selectat, acesta afișează o bară de culoare albastră în partea sa inferioară, iar atunci când nu este selectat, o bară de culoare gri.

Textele afișate pentru valorile stăriilor pot fi controlate în fișierul XML prin intermediul proprietăților `android:textOn` respectiv `android:textOff`, iar programatic prin metodele `setTextOn()` și `setTextOff()`.

Valorile implicate pentru proprietățile `textOn` și `textOff` sunt *ON*, respectiv *OFF*.

Atributul `text`, moștenit de la obiectul de tip `Button`, nu este utilizat cu toate că poate fi definit.

## CheckBox

Controlul de tip `CheckBox` (din clasa `android.widget.CheckBox`) este tot un element de tip buton ce poate avea două stări (selectat și neselectat), accesibile prin intermediul proprietății `checked` (`android:checked` în fișierul XML și `setChecked()` sau `toggle()` în codul sursă).

Deși este definit un tip de eveniment asociat selecției sau deselecției acestei componente (specificat de interfața `CompoundButton.OnCheckedChangeListener` pentru care trebuie implementată metoda `onCheckedChange()`), de cele mai multe ori nu își se asociază o clasă ascultător, verificându-se starea sa prin intermediul metodei `isChecked()` la producerea unui alt eveniment.

## Switch

Controlul de tip `Switch` (din clasa `android.widget.Switch`), introdus începând cu nivelul de API 14, este tot un element de tip buton ce poate avea două stări (selectat și neselectat), având forma unei bare derulante prin care se poate realiza tranziția între cele două valori, aflate la capetele acesteia.

## RadioButton / RadioGroup

Un element de tip `android.widget.RadioButton` este tot un buton cu două stări (selectat / neselectat), de obicei utilizat pentru selecția unei singure opțiuni dintr-o listă, excluderea mutuală fiind realizată prin includerea mai multor obiecte de acest tip într-o componentă de tip `android.widget.RadioGroup`.

Un obiect de tip `RadioButton` se poate afla și în afara unui grup la fel cum un obiect de tip `RadioGroup` poate conține și alte controale decât butoane.

Înțial, toate butoanele din cadrul unui grup sunt deselectate, deși se poate preciza această proprietate atât prin fișierul XML cât și programatic. Odată ce un utilizator a selectat un buton radio din cadrul unui grup, acesta nu mai poate fi deselectat (decât prin schimbarea selecției pe un alt buton radio din cadrul grupului), anularea selecției putând fi realizată prin metoda `clearCheck()` apelată pe obiectul `RadioGroup` care conține obiectul de tip `RadioButton`.

Proprietățile care determină selecția unui `RadioButton` sunt aceleași ca și în cazul celorlalte tipuri de butoane cu două stări.

Evenimentele de selecție a unui `RadioButton` în cadrul unui `RadioGroup` pot fi realizate:

1. prin definirea unei clase ascultător pe fiecare buton radio din cadrul grupului, aceasta implementând interfața `CompoundButton.OnCheckedChangeListener` cu metoda `onCheckChanged(CompoundButton, boolean)` care transmite starea butonului radio (selectat, deselectat)

2. prin definirea unei clase ascultător pe grupul de butoane, aceasta implementând interfața `RadioGroup.OnCheckedChangeListener` cu metoda `onCheckChanged(RadioGroup, int)` care transmite identificatorul (din clasa `R.id`) butonului care a fost selectat

Se va genera un eveniment de modificare a selecției și atunci când aceasta este anulată (programatic).

În situația în care nu se dorește generarea unui eveniment de fiecare dată când se realizează o selecție, ci preluarea acesteia la un moment dat de timp ulterior, va fi utilizată metoda `getCheckedRadioButtonId()` din clasa `RadioGroup` care întoarce identificatorul butonului (din clasa `R.id`) care a fost selectat (sau -1 dacă nu este selectat nici un buton radio din cadrul grupului).

## Controale de tip multimedia -- Optional

Android pune la dispoziția utilizatorilor mai multe controale pentru redarea conținuturilor de tip multimedia, dintre care cele mai folosite sunt `ImageView` (pentru imagini) și `VideoView` (pentru conținut video).

### ImageView

Controlul de tip `ImageView` este utilizat pentru afișarea unei imagini, aceasta putând proveni:

- dintr-o resursă a aplicației Android, ce poate fi specificată
  - în fișierul XML prin proprietatea `android:src=@drawable/...` (se poate indica și codul unei culori în formatul #RRGGBB)
  - programatic, folosind una din metodele
    - `setImageResource()`, care primește ca parametru identificatorul resursei din clasa `R.drawable`
    - `setImageBitmap()`, ce primește un parametru de tip `Bitmap` încărcat din resursa respectivă, ulterior putând fi realizate și unele modificări

```
▪ ImageView someImageView =
 (ImageView) findViewById(R.id.some_image_view);

▪ Bitmap someBitmap =
 BitmapFactory.decodeResource(this.getResources(),
 R.drawable.some_image);

▪
▪ // realizeaza modificari asupra obiectului Bitmap
▪

someImageView.setImageBitmap(someBitmap);
```

Clasa `BitmapFactory` pune la dispoziția programatorilor și alte metode pentru crearea unui obiect de tip `Bitmap`, folosind un vector de octeți sau un obiect de tip `InputStream` (mai ales pentru încărcarea unor resurse de pe un server).

- dintr-un fișier local, apelând pentru obiectul de tip `ImageView` una dintre metodele:
  - `setImageDrawable()` ce primește ca parametru rezultatul întors de metoda statică `createFromPath()` a clasei `Drawable` pentru indicarea locației la care se găsește fișierul ce conține imaginea
  - `setImageURI()`, transmitându-se un URI care să indice locația la care se găsește fișierul ce conține imaginea; în acest caz, va putea fi utilizat doar protocolul `file`
  - de la un furnizor de conținut

Pentru un astfel de obiect pot fi precizate parametrii precum lățimea și înălțimea maximă (`maxWidth`, respectiv `maxHeight`), precum și mecanismul folosit pentru scalarea conținutului în situația în care suprafața pe care se realizează afișarea este diferită de dimensiunile reale ale resursei (`scaleType`).

## VideoView

Controlul de tip `VideoView` este utilizat pentru redarea de conținut video, într-unul din formatele H.263, H.264 AVC, MPEG-4 SP sau VP8.

Conținutul acestui control poate fi specificat prin una din metodele `setVideoPath(String)` sau `setVideoUri(Uri)`, prin care se indică locația unui fișier stocat pe dispozitiv sau aflat la distanță, pe un server.

Pentru a se putea accesa conținutul aflat la distanță, în fișierul `AndroidManifest.xml` trebuie să se specifice o permisiune explicită în acest sens:

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

Metodele pe care le pune la dispoziție un astfel de obiect pentru controlul procesului de redare sunt `start()`, `pause()`, respectiv `stopPlayback()`. De asemenea, se pot verifica anumiți parametri ai conținutului prin metodele `isPlaying()`, `getDuration()` și `getCurrentPosition()`.

De asemenea, pot fi monitorizate o serie de evenimente, cum ar fi începutul și sfârșitul redării conținutului video, generarea unor informații cu privire la redarea conținutului, producerea unei erori:

```
videoView.setOnPreparedListener(new MediaPlayer.OnPreparedListener() {
 @Override
 public void onPrepared(MediaPlayer mediaPlayer) {
 // TODO Auto-generated method stub
 }
});
videoView.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
 @Override
 public void onCompletion(MediaPlayer mediaPlayer) {
 // TODO Auto-generated method stub
 }
});
videoView.setOnInfoListener(new MediaPlayer.OnInfoListener() {
```

```
@Override
public boolean onInfo(MediaPlayer mediaPlayer, int what, int extra) {
 // TODO Auto-generated method stub
 return false;
}
});
videoView.setOnErrorListener(new MediaPlayer.OnErrorListener() {
 @Override
 public boolean onError(MediaPlayer mediaPlayer, int what, int extra) {
 // TODO Auto-generated method stub
 return false;
 }
});
```

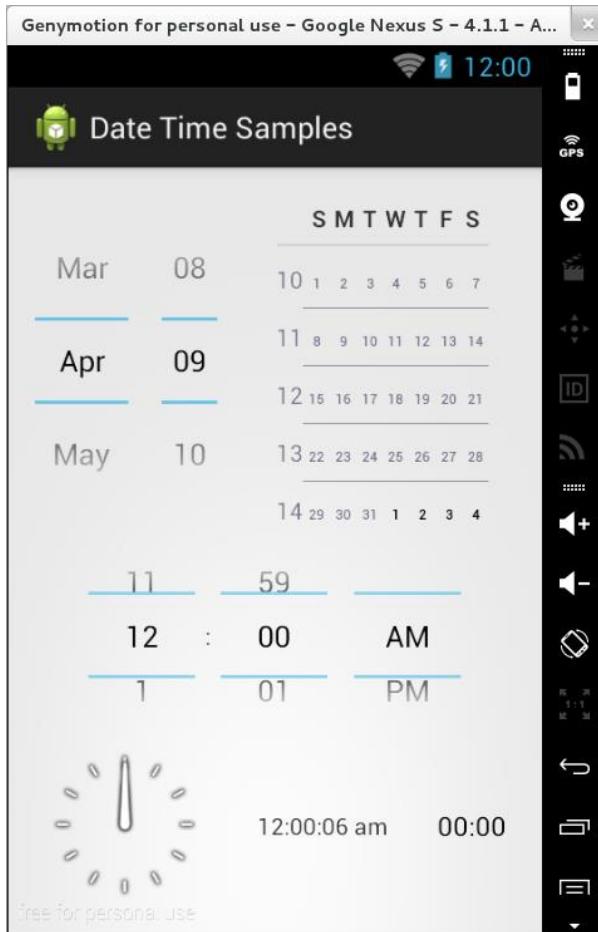
Pentru ca utilizatorul aplicației să poată controla redarea conținutului video, obiectului de tip VideoView trebuie să i se asocieze un obiect MediaController care va pune la dispoziție o serie de controale prin care se poate realiza pornirea și oprirea procesului.

```
MediaController mediaController = new MediaController(this);
mediaController.setAnchorView(videoView);
videoView.setMediaController(mediaController);
```

Controalele obiectului MediaController pot fi afișate sau nu prin metodele show(int timeout) respectiv hide().

## Controale pentru gestiunea datei calendaristice și a timpului -- Optional

Majoritatea bibliotecilor de controale pun la dispoziția dezvoltatorilor componente pentru gestiunea informațiilor legate de data calendaristică și timp. În Android, cele mai frecvent uzitate elemente de acest tip sunt DatePicker, TimePicker, AnalogClock și DigitalClock.



## DatePicker

Obiectul de tip `DatePicker` este folosit pentru gestiunea componentelor unei date calendaristice (zi, lună, an).

Inițializarea unei astfel de componente se realizează prin metoda `init()`, ce primește ca parametrii anul, luna, ziua și o clasă ascultător (ce implementează interfața `DatePicker.OnDateChangedListener` cu metoda `onDateChanged()`). De regulă, evenimentele pe acest control nu sunt monitorizate, preluându-se conținutul său la un moment de timp ulterior. În cazul în care nu se specifică nici o valoare pentru acest control, va fi preluată data calendaristică a sistemului Android.

Conținutul controlului se obține prin metodele `getDayOfMonth()`, `getMonth()`, respectiv `getYear()`.

Numerotarea unei luni se face pornind de la 0, astfel încât valoarea furnizată de acest control va trebui incrementată. Invers, la inițializarea conținutului unei componente de tip `DatePicker`, se va realiza operația de decrementare înainte de transmiterea unei date calendaristice.

Valorile pe care le poate lua acest element pot fi controlate prin intermediul proprietăților `minDate`(respectiv `minYear`) și `maxDate` (respectiv `maxYear`). De asemenea, dacă se dorește sau nu afișarea unui calendar va fi indicat prin intermediul proprietății `calendarViewShown`.

## TimePicker

Obiectul de tip `TimePicker` este folosit pentru gestiunea componentelor timpului (oră, minut). Inițializarea unei astfel de componente se realizează prin metodele `setCurrentHour()`, respectiv `setCurrentMinute()` care primesc ca parametrii obiecte de tip `Integer`. Clasa ascultător pentru evenimentele produse în legătură cu acest obiect implementează interfața `TimePicker.OnTimeChangedListener` și metoda `onTimeChanged()`. În cazul în care nu se specifică nici o valoare pentru acest control, va fi preluat timpul sistemului Android. Formatul utilizat pentru oră (12/24 ore) se precizează prin intermediul proprietății `is24HourView`.

Conținutul controlului se obține prin metodele `getCurrentHour()`, respectiv `getCurrentMinute()`.

### AnalogClock

Obiectul de tip `AnalogClock` este folosit pentru afișarea timpului curent, folosind un ceas analogic. Utilizatorul nu poate interacționa cu un astfel de control, actualizarea conținutului său fiind realizată în mod automat, la fiecare minut.

Un obiect de tip `AnalogClock` poate fi particularizat în sensul că se pot specifica anumite resurse grafice pentru afișarea cadranelui orar și a limbilor care indică ora și minutul.

### DigitalClock (nivel API < 17)

Obiectul de tip `DigitalClock` este folosit pentru afișarea timpului curent, folosind un ceas digital. Este un control de tip `TextView` de la care moștenește toate funcționalitățile. Utilizatorul nu poate interacționa cu un astfel de control, actualizarea conținutului său fiind realizată în mod automat, la fiecare secundă.

### TextClock (nivel API >= 17)

Obiectul de tip `TextClock` este folosit pentru afișarea datei calendaristice și/sau a timpului curent (folosind formatul 12/24 ore și permitând schimbarea zonei de timp). Implicit, acest control nu afișează și secundele.

### Chronometer

Un obiect de tip `Chronometer` este utilizat pentru monitorizarea timpului scurs începând de la un anumit moment, pentru activitățile în care un astfel de element este esențial.

Modul în care este afișat contorul ce indică trecerea timpului este controlat prin intermediul parametrului `android:format`:

- %h - număr de ore;
- %m - număr de minute;
- %s - număr de secunde.

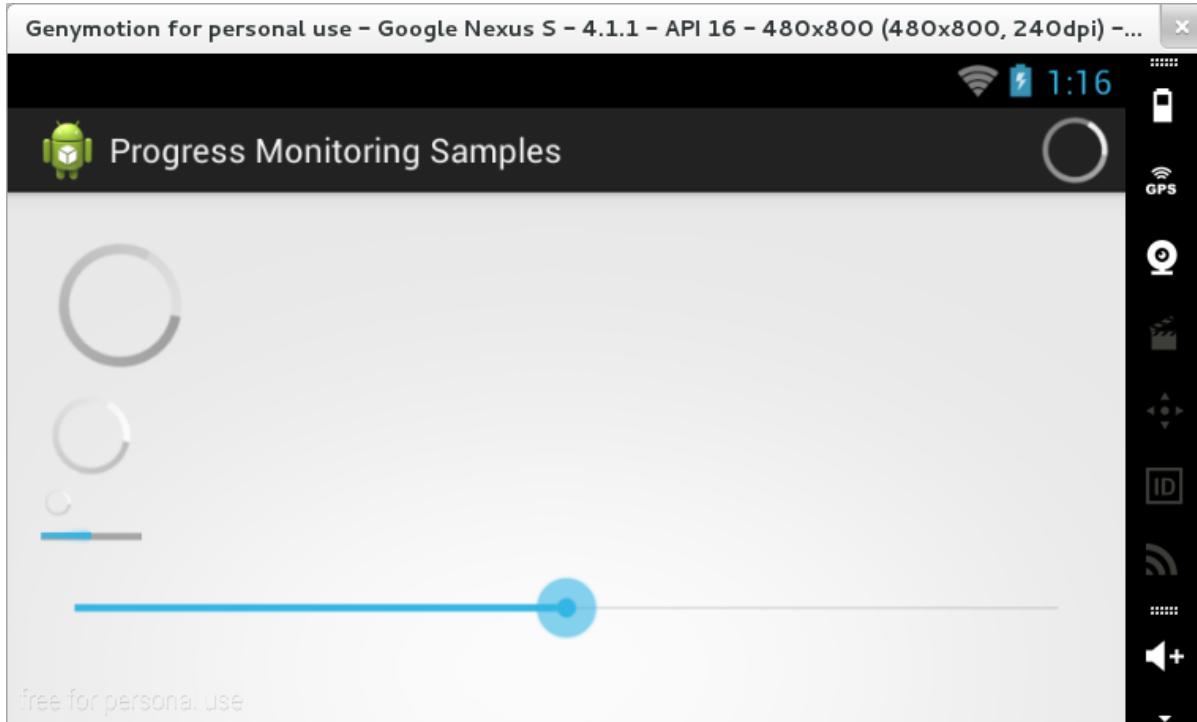
Cronometrarea propriu-zisă este realizată între apelurile metodelor `start()`, respectiv `stop()`. De asemenea, se poate indica momentul de timp care este considerat drept referință pentru numărătoare, prin intermediul metodei `setBase()`. Pornirea de la 0 se face transmițând metodei `setBase()` un parametru de tip `android.os.SystemClock.elapsedRealTime()`.

Monitorizarea evenimentelor legate de un obiect de tip `Chronometer` se face prin implementarea unui obiect ascultător de tip `Chronometer.OnChronometerTickListener` prin (re)definirea metodei `onChronometerTick()`.

## Controale pentru monitorizarea progresului -- Optional

În momentul în care sunt realizate acțiuni care durează o perioadă de timp mai mare, o practică uzuale este de a indica utilizatorului faptul că un proces este în desfășurare și că aplicația nu s-a blocat prin intermediu unei bare de progres. O altă situație în care se dorește să se afișeze progresul este indicarea evoluției pe care a realizat-o un utilizator în cadrul unei aplicații (rularea unui fișier multimedia - timp scurs vs. timp rămas, nivelul la care s-a ajuns în cadrul unui joc) sau chiar controlul acestei poziții.

SDK-ul de Android oferă mai multe controale care pot fi utile în monitorizarea progresului: bară de progres (`ProgressBar`) sau bară de căutare (`SeekBar`), precum și alte obiecte pentru a indica diferite forme de activitate.



## ProgressBar

Obiectul de tip `ProgressBar` are mai multe forme:

1. un indicator circular animat, care nu oferă nici un fel de informații cu privire la procentul de completitudine al procesului realizat în fundal (util mai ales pentru operații a căror lungime nu poate fi determinată); stilurile care pot fi folosite pentru acest tip de bară de progres sunt `progressBarStyleLarge` și `progressBarStyleSmall`, animația fiind realizată în mod automat;
2. o bară de progres orizontală care indică și măsura în care sarcina din fundal a fost realizată (poate fi însoțită și de o altă bară de progres orizontală care indică starea unui proces imbricat); stilul folosit în acest caz este `progressBarStyleHorizontal`, suportând proprietăți cum ar fi `progress` (valoarea curentă), respectiv `min / max`.

Se obișnuiește ca progresul să fie afișat direct în bara de titlu, ceea ce poate economisi din spațiul folosit de interfața grafică, acesta putând fi activat / dezactivat cu ușurință. Semnificația unui indicator de progres în acest caz este aceea că trebuie așteptată încărcarea mai multor resurse înainte ca utilizatorul să poată interacționa cu aplicația respectivă.

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
 requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
 setContentView(R.layout.progress_bar_activity);
 setProgressBarIndeterminateVisibility(true);
}
```

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
 requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
 setContentView(R.layout.progress_bar_activity);
 setProgressBarVisibility(true);
 setProgress(10000); // valoare
}
```

De remarcat faptul că metoda `requestWindowFeature()` trebuie apelată înainte de încărcarea propriu-zisă a interfeței grafice prin metoda `setContentView()`. Implicit, indicatorul de progres este vizibil, metodele `setProgressBarIndeterminateVisibility()` și `setProgressBarVisibility()` să vând rolul de a-l activa, respectiv dezactiva în funcție de parametrul care este transmis. În momentul în care progresul atinge valoarea maximă, starea obiectului asociat trece de la vizibil la non-vizibil, prin intermediul unei animații.

În situația în care nu se poate evalua progresul pe care îl înregistrează un proces, se folosesc **indicatori de activitate**, care pot avea forma unei bare de stare sau a unui cerc.

Fiecare operație care este realizată în fundal ar trebui reprezentată în interfața grafică prin intermediul unui indicator de activitate.

Se folosesc tot obiecte de tip `ProgressBar`, pentru care se precizează faptul că vor rula o perioadă de timp nedeterminată:

- prin precizarea proprietății `android:indeterminate`;
- prin invocarea metodei `setProgressBarIndeterminateVisibility()`.

## SeekBar

În situația în care se dorește să se ofere utilizatorului posibilitatea de a controla progresul unei anumite operații, se folosesc un obiect de tip `SeekBar`, care are în plus un selector a cărui locație poate fi modificată manual prin interfața grafică, la orice poziție cuprinsă între 0 și valoarea indicată de proprietatea `max`.

Resursa grafică folosită pentru redarea selectorului poate fi particularizată prin utilizarea oricărei imagini din directorul `drawable`.

Notificările cu privire la modificările operate pe un astfel de obiect sunt primite prin intermediul unui obiect ascultător de tipul `SeekBar.OnSeekBarChangeListener` pentru care se definește metoda `onProgressChanged()` (aceasta primește și un parametru de tip `boolean` care indică dacă actualizarea controlului a fost realizată în urma unei intervenții a utilizatorului sau programatic):

```
SeekBar moviePlayerSeekBar =
(SeekBar) findViewById(R.id.movie_player_seek_bar);
moviePlayerSeekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener())
{
```

```
 @Override
```

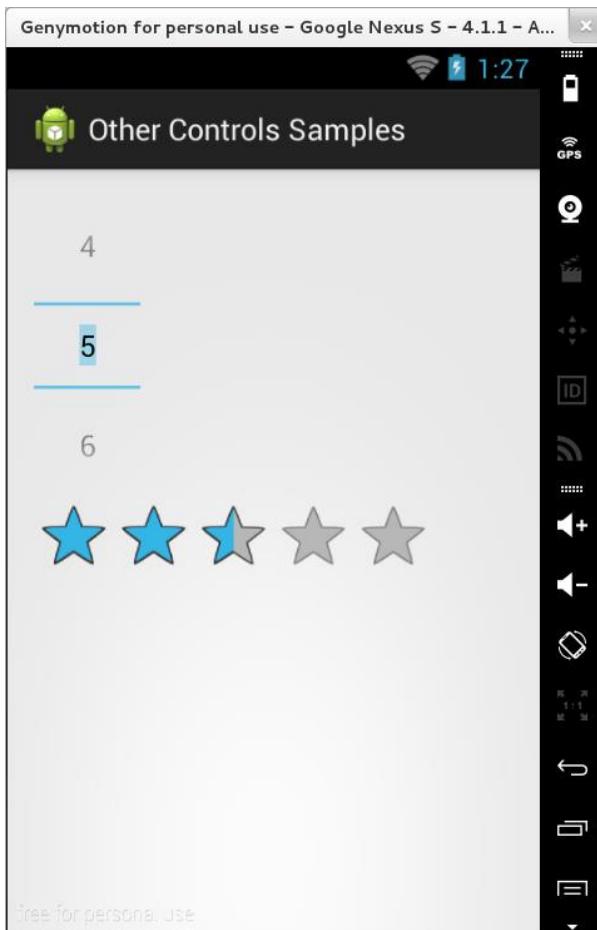
```

 public void onProgressChanged(SeekBar seekBar, int progress, boolean
fromTouch) {
 // ...
}

});

```

## Alte Tipuri de Controle Grafice -- Optional



### NumberPicker

Un obiect de tip `NumberPicker` este folosit pentru a alege o valoare dintr-un interval definit de proprietățile `min` și `max`. Forma în care este afișat depinde de tema pe care o folosește aplicația:

- dacă tema curentă este derivată din `Theme`, elementul este prezentat ca un câmp text editabil cu un buton de incrementare deasupra și un buton de decrementare dedesubt; operația de apăsare lungă a butoanelor determină schimbarea rapidă a valorii curente;
- dacă tema curentă este derivată din `Theme_Holo` sau `Theme_Holo_Light`, elementul este prezentat ca un câmp text editabil, având o valoare mai mică

deasupra și o valoare mai mare dedesubt; operația de apăsare lungă pe valorile mai mici / mai mari determină schimbarea rapidă a valorii curente; apăsarea pe valoarea curentă activează tastatura virtuală pentru a indica valoarea dorită; de asemenea, se poate folosi operația de derulare pentru a naviga prin întreaga gamă de valori.

Valoarea curentă este reprezentată de proprietatea `value`.

Identificarea momentului în care a fost modificată o valoare poate fi realizată prin implementarea unui obiect ascultător de tip `NumberPicker.OnValueChangeListener` și redelinirea valorii `onValueChange()`.

## RatingBar

Un obiect de tip `RatingBar` este utilizat pentru a gestiona calificative, exprimate sub formă de număr de stele. Definirea unui astfel de obiect implică:

- precizarea calificativului maxim, prin intermediul proprietății `numStars` (număr întreg);
- diferența utilizată între valori succesive, prin atributul `stepSize` (număr real, cu zecimale).

Evenimentele reprezentate de interacțiunea cu utilizatorul sunt gestionate prin intermediul unui obiect ascultător de tip `RatingBar.OnRatingChangeListener` și (re)definirea metodei `onRatingChanged()`, apelată în momentul în care a fost înregistrată o modificare a calificativului (în timpul selecției propriu-zise a valorii nu se invocă metoda).

## Mecanisme pentru disponerea controalelor (layout) -- Obligatoriu

---

Controalele Android fac parte din cadrul unui grup (obiect de tip `android.view.ViewGroup`) care definește și modul în care acestea sunt dispuse în cadrul interfeței grafice precum și dimensiunile pe care le pot lua, motiv pentru care o astfel de componentă este referită și sub denumirea de layout. Acest element nu vizează însă tratarea evenimentelor legate de interacțiunea cu utilizatorul.

În Android, denumirea de layout este utilizată și pentru fișierele de resurse care definesc interfața grafică corespunzătoare unei activități, a unui fragment sau a unui alt element din interfața grafică, plasate în `/res/layout` (respectiv în `/res/layout-land`). Acestea nu trebuie însă confundate cu tipurile de controale care gestionează mecanismul de disponere a diferitelor elemente grafice în cadrul interfeței.

Cele mai utilizate tipuri de grupuri de componente vizuale sunt `LinearLayout`, `AbsoluteLayout`, `RelativeLayout`, `FrameLayout`, `TableLayout` și `GridLayout`.

Există și alte tipuri de controale derive din pachetul `android.view.ViewGroup`, care au rolul de containere pentru anumite colecții de informații, cu care utilizatorul poate interacționa în mod direct: `ListView`, `GridView`, `ScrollView`, `ImageSwitcher`.

Elementele de tip layout pot fi imbricate (conținute) unele înaltele, astfel încât se pot proiecta interfețe grafice în care modul de disponere al controalelor să fie foarte complex, prin combinarea funcționalităților pe care le oferă fiecare dintre componente de tip  `ViewGroup`. Restricția care trebuie respectată în acest caz este ca spațiile de nume indicate prin proprietatea `xmlns` să fie precizate doar o singură dată, de obiectul layout rădăcină.

Dezvoltarea unei interfețe grafice se poate realiza atât în fișierul XML corespunzător activității cât și programatic. În ambele cazuri, va trebui precizat inițial obiectul de tip `ViewGroup` la care se va preciza componența, prin specificarea elementelor de tip `View` pe care le conțin. Fiecare clasă de tip layout are și o clasă internă `LayoutParams` în care proprietățile referitoare la dimensiuni și margini sunt reținute în obiecte de tip `layout_...`. Ele vor fi aplicate tuturor controalelor grafice conținute. Cele mai frecvent utilizate atrbute sunt:

- `layout_height`, `layout_width` - definesc lățimea și înălțimea componentei, putând avea valorile:
  - `match_parent` - va ocupa tot spațiul pus la dispoziție de componenta în care este conținută (fără padding);
  - `wrap_content` - va ocupa doar spațiul solicitat de componente pe care le conține (cu padding);
  - o valoare indicată explicit împreună cu unitatea de măsură.
- `layout_weight` - proporția pe care o ocupă în raport cu alte componente;

Presupunând cazul unei interfețe cu trei componente:

- dacă una are proprietatea `layout_weight` egală cu 1 (iar restul au specificată valoarea 0), aceasta va ocupa tot spațiul rămas liber
- dacă valorile asociate proprietății `layout_weight` au valori egale cu 1, 2, respectiv 3, acestea vor ocupa 16% (1/6), 33% (2/6), respectiv 50% (3/6) din spațiul pe care îl pune la dispoziție componenta în care sunt incluse
- `weightSum` - suma proporțiilor tuturor controalelor grafice conținute; valoarea implicită este 1;
- `layout_gravity` - modul în care componenta este aliniată în cadrul grupului din care face parte (valorile posibile pe care le poate lua această proprietate sunt: `top`, `botttom`, `left`, `right`, `center_vertical`, `center_horizontal`, `center` (centrare pe ambele direcții), `fill_vertical`, `fill_horizontal`, `fill` (ocuparea spațiului pe ambele direcții), `clip_vertical`, `clip_horizontal`; aceste valori pot fi combinate prin intermediul operatorului `|` (pe biți);

Această proprietate nu trebuie confundată cu `gravity`, care controlează modul în care sunt dispuse elementele conținute în cadrul componentei.

- marginile sunt precizate de proprietățile `layout_marginTop`, `layout_marginBottom`, `layout_marginLeft`, `layout_marginRight`, valorile precizate trebuind să aibă asociată și unitatea de măsură folosită; în situația în care se dorește să se folosească aceeași valoare pentru toate direcțiile, se va folosi atributul `layout_margin`;

Pentru această proprietate este permisă și precizarea de valori negative.

- spațiul de umplere (cu spații) este specificat prin paddingTop, paddingBottom, paddingLeft, paddingRight;
- background poate specifica o culoare sau o imagine care vor fi afișate pe fundal;
- textColor indică culoarea textului în format #rrggb (sau #aarrggbb), putând fi referită și o culoare definită anterior într-un fișier colors.xml (sau cu orice altă denumire) din directorul /res/values în care elementul rădăcină este <resources> (elementele de aici fiind referite prin @color/...).

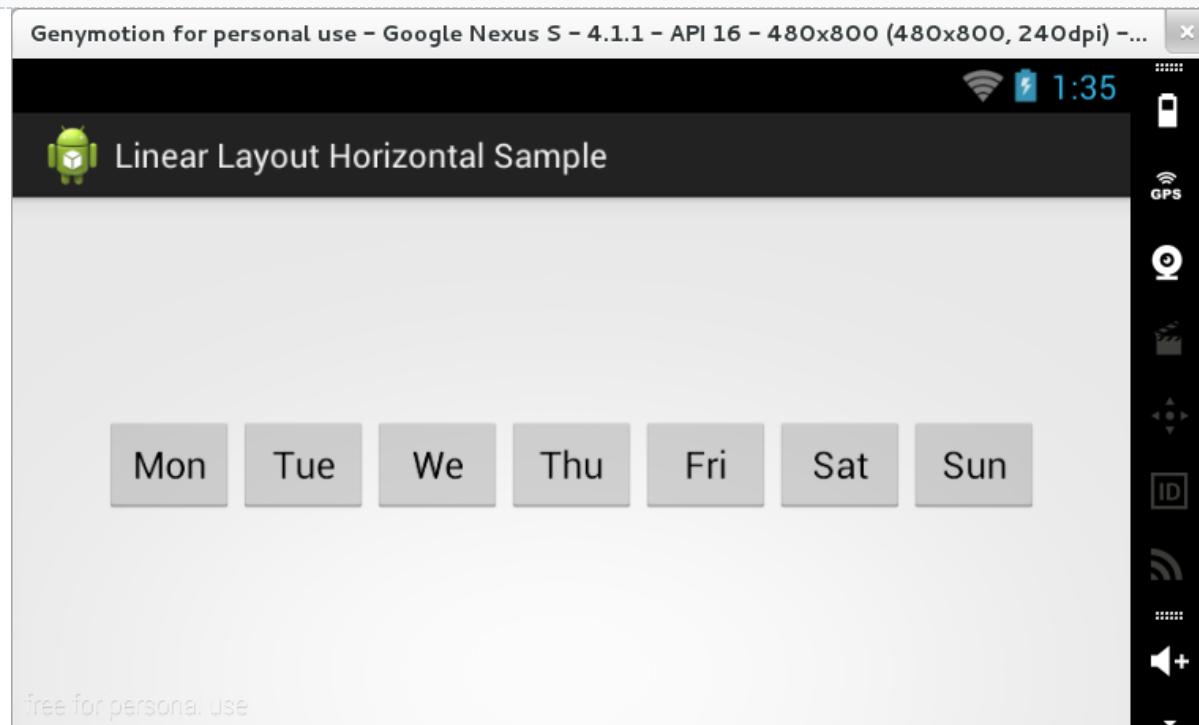
## LinearLayout

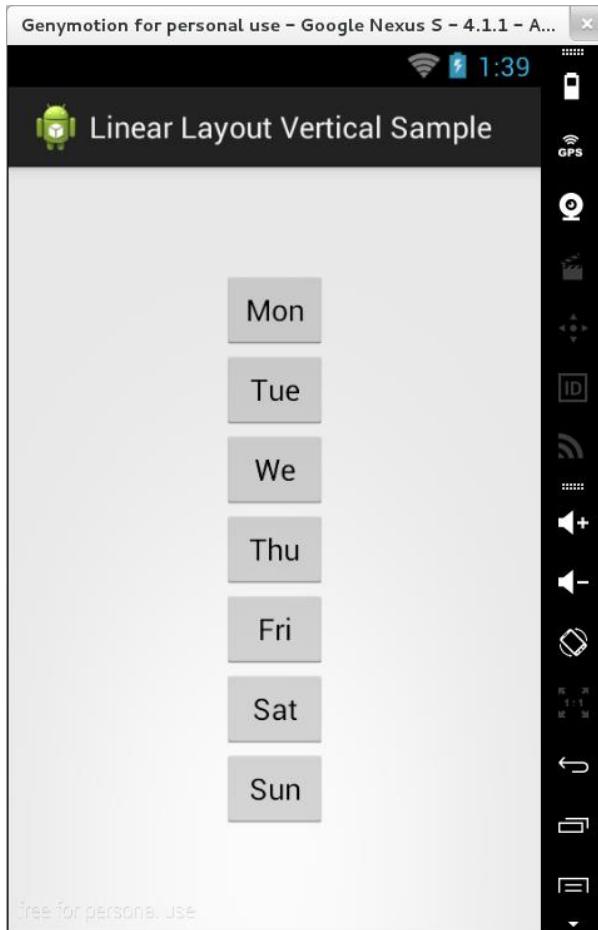
În cadrul unui grup de tip LinearLayout, componentele sunt dispuse fie pe orizontală, fie pe verticală, în funcție de proprietatea orientation (putând lua valorile horizontal - implicită sau vertical).

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

 <!-- other layouts or widgets -->

</LinearLayout>
```





## AbsoluteLayout

Într-o interfață grafică dezvoltată prin intermediul `AbsoluteLayout`, poziția fiecărei componente trebuie specificată prin coordonatele sale absolute, indicate prin intermediul proprietăților `layout_x` și `layout_y`. De asemenea, dimensiunile acestora vor fi indicate prin atributele `layout_width` și `layout_height`.

Întrucât acest mecanism de poziționare nu scalează corespunzător pentru dispozitive având alte rezoluții decât cea pentru care a fost proiectată aplicația, utilizarea sa nu mai este recomandată începând cu Android 1.5. Deși este încă suportat în versiunile curente, există posibilitatea de a fi eliminat, astfel încât aplicațiile care îl utilizează să nu mai ruleze pe dispozitivele implementând noile API. De aceea, este bine ca acest tip de layout să fie evitat pe cât posibil.

## RelativeLayout

În cadrul unui grup de tip `RelativeLayout`, componentele pot fi dispuse relativ la:

- containerul din care fac parte, prin intermediul unor proprietăți ce pot lua valorile `true` sau `false`:
  - **pe verticală:** `layout_alignParentTop`, `layout_alignParentBottom`;
  - **pe orizontală:** `layout_alignParentLeft`, `layout_alignParentRight`;

- `center`: `layout_centerInParent`, `layout_centerHorizontal`, `layout_centerVertical`.
- alte controale din cadrul aceluiași container, spre care există o referință prin intermediul unui identificator (`android:id`), accesat ca `@id/...`:
  - `layout_alignTop`, `layout_alignBottom`, `layout_alignLeft`, `layout_alignRight`;
  - `layout_toLeftOf`, `layout_toRightOf`, `layout_above`, `layout_below`.



## FrameLayout

FrameLayout reprezintă o strategie de poziționare folosită pentru afișarea unei singure componente la un moment dat. Totuși, aceasta poate fi populată cu mai multe elemente grafice, recomandându-se ca doar una singură să fie vizibilă. Vizibilitatea poate fi specificată în fișierul XML prin proprietatea `android:visibility` care poate lua valorile `visible` sau `gone`, respectiv programatic, folosind metoda `setVisibility()` ce primește ca parametru constantele `View.VISIBLE` sau `View.GONE`. De regulă, afișarea unei componente este realizată în mod dinamic, ca rezultat al unei metode de tratare a unui eveniment.

Dimensiunea obiectului de tip FrameLayout ar trebui să fie preluată de la controlul cel mai voluminos, riscându-se altfel ca pe măsură ce se afișează elemente de dimensiuni mai mari, acestea să nu fie vizibile în totalitate. În acest scop, se va folosi proprietatea XML `android:measureAllChildren="true"` sau se va apela metoda `setMeasureAllChildren(true)`, astfel încât să se realizeze redimensionarea container-ului la toate elementele conținute, nu doar la cele vizibile la un moment dat.

De regulă, acest tip de layout se folosește atunci când se dorește să se realizeze o animație folosind mai multe imagini, dintre care una singură este vizibilă la un moment dat.

În cazul în care mai multe controale grafice sunt vizibile simultan, ele vor fi afișate sub forma unei structuri de tip stivă, în funcție de ordinea în care sunt declarate, elementele din vârful stivei având vizibilitate maximă iar cele de la baza stivei vizibilitate minimă.

Alte atrbute utilizate de acest mecanism de dispunere a conținutului sunt:

- `foreground`, indicând o resursă de tip grafic (din `res/drawable`) care va fi afișată peste conținutul interfeței grafice;
- `foregroundGravity`, precizând poziționarea resursei de tip grafic care va fi afișată peste conținutul interfeței grafice (putând avea valorile `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, `fill`, `clip_vertical`, `clip_horizontal` sau combinații ale acestora, realizate prin intermediul operatorului `|`, pe biți).



## TableLayout

Într-un mecanism de poziționare de tip `TableLayout`, elementele componente (layout-uri sau widget-uri) sunt plasate în cadrul unui tabel, fără ca marginile să fie afișate și fără a se specifica în mod explicit numărul de rânduri sau de coloane, acestea fiind determinate în mod

automat (numărul de rânduri este determinat ca sumă a elementelor de tip `TableRow`, iar numărul de coloane ca maximul numărului de componente de pe o linie; de asemenea, dimensiunea unei coloane este dată de componenta cea mai voluminoasă afișată în cadrul ei).

Cele mai importante proprietăți ale unui layout de acest tip sunt:

- `stretchColumns` - specifică o listă de indecsări indicând coloanele care ar trebui să fie întărite în cazul în care tabelul este mai mic decât spațiul de afișare pe care îl pune la dispoziție;
- `shrinkColumns` - specifică o listă de indecsări indicând coloanele care ar trebui să fie reduse în cazul în care tabelul este mai mare decât spațiul de afișare pe care îl are la dispoziție;
- `collapseColumns` - specifică o listă de indecsări indicând coloanele care ar trebui să fie omise în totalitate.

În referirea indecsărilor, numerotarea se face de la 0. Separarea indecsărilor în cadrul listei se face prin virgulă.

De obicei, componentele sunt plasate într-un obiect de tip `TableRow`. De asemenea, este permisă definirea de componente și în afara acestui element, dacă se dorește ca obiectul respectiv să ocupe întregul spațiu al rândului respectiv.

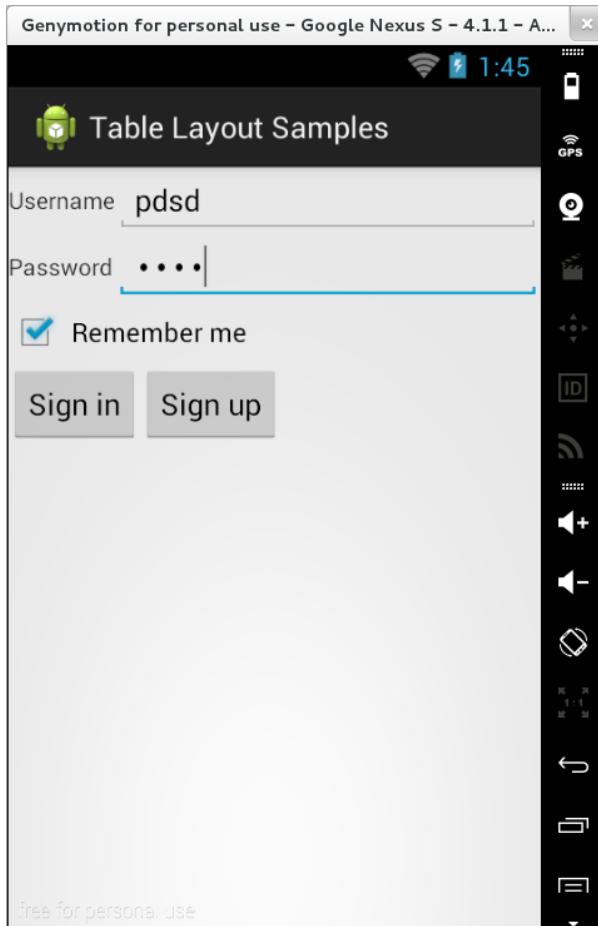
Dacă se dorește ca unele coloane să fie lăsate goale, pot fi utilizate și unele proprietăți:

- `layout_column` - se folosește pentru a specifica o anumită coloană, lăsând coloanele anterioare fără nici un conținut (în mod obișnuit, ordinea de procesare a coloanelor este de la stânga la dreapta);
- `layout_span` - este utilizat în situația în care se dorește afișarea unui anumit conținut în cadrul a mai multor coloane consecutive (`colspan`); numărul acestora este indicat prin intermediul acestei proprietăți.

Nu există o proprietate pentru afișarea unui conținut în cadrul a mai multe rânduri consecutive (`rowspan`). În acest sens, trebuie folosite tabele imbicate.

```
<TableLayout 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:stretchColumns="1"
 tools:context="..." >
 <TableRow>
 <TextView
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:gravity="center_vertical"
 android:text="@string/username" />
 <EditText
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:inputType="text" />
 </TableRow>
</TableLayout>
```

```
</TableRow>
<TableRow>
 <TextView
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:gravity="center_vertical"
 android:text="@string/password" />
 <EditText
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:inputType="textPassword" />
</TableRow>
<TableRow>
 <CheckBox
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:layout_span="2"
 android:text="@string/keepmesignedin" />
</TableRow>
<TableLayout
 android:layout_width="match_parent"
 android:layout_height="match_parent" >
 <TableRow>
 <Button
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@string/signin" />
 <Button
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@string/signup" />
 </TableRow>
</TableLayout>
</TableLayout>
```



## GridLayout

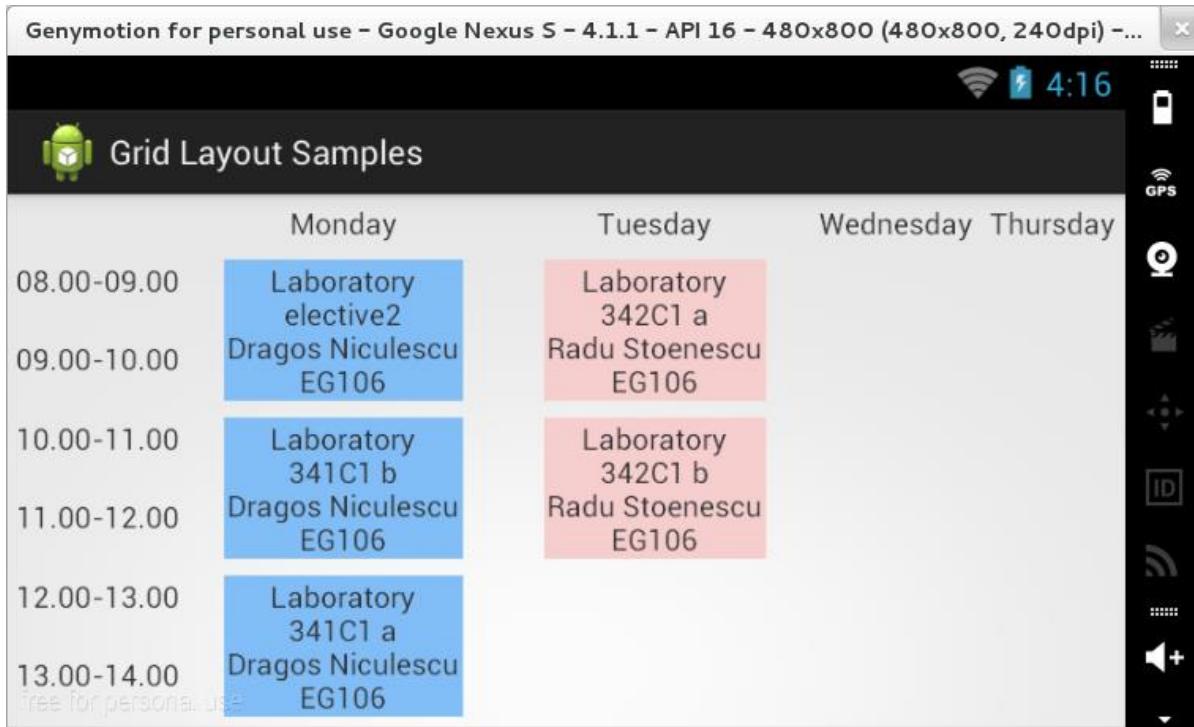
Layout-ul de tip `GridLayout` este utilizat tot pentru dispunerea componentelor într-un format tabelar, folosind însă o sintaxă mult mai flexibilă. Totodată, acest mecanism este și mult mai eficient din punctul de vedere al randării.

Astfel, pentru specificarea numărului de rânduri și de coloane se vor utiliza proprietățile `rowCount` și `columnCount`, indicându-se pentru fiecare element grafic în parte poziția la care va fi plasat, prin atributele `layout_row` și `layout_column`. În cazul în care pentru o componentă grafică nu se specifică linia sau coloana din care face parte, atributul `orientation` (având valori posibile `horizontal` sau `vertical` indică dacă elementul următor va fi plasat pe linia sau pe coloana succesiivă).

Întrucât modul de dispunere `GridLayout` permite plasarea mai multor componente la aceeași poziție (rând și coloană), nu se garantează faptul că acestea nu se vor suprapune.

În cazul în care se dorește extinderea unui element grafic pe mai multe rânduri sau pe mai multe coloane, se vor utiliza atributele `layout_rowSpan` și `layout_columnSpan`. Pentru controlul modului de dispunere se va folosi proprietatea `layout_gravity`. Precizarea `layout_width` și `layout_height` nu este neapărat necesară, valoarea lor implicită în acest caz fiind `wrap_content`.

Deși a fost introdus începând cu nivelul de API 14, mecanismul de dispunere a conținutului `GridLayout` poate fi utilizat și în aplicațiile Android folosind un SDK corespunzător nivelului de API 7, folosind biblioteca de suport v7 (revizie ulterioară versiunii 13).



## ScrollView

ScrollView este un mecanism de disponere a elementelor din interfața grafică utilizat pentru cazul în care dimensiunile acesteia depășesc posibilitățile de afișare ale dispozitivului mobil. Acesta poate avea o singură componentă care este de regulă un obiect de tip layout.

## Adaptarea interfeței grafice în funcție de orientarea ecranului

Dispozitivele Android suportă două moduri de orientare a ecranului: portrait și landscape. În momentul în care se produce o modificare a orientării dispozitivului de afișare, activitatea care rulează în mod curent este distrusă și apoi recreată. Aplicațiile trebuie să gestioneze astfel de situații, adaptând interfața grafică în funcție de suprafața de care dispun. În acest scop, poate fi adoptată una din următoarele abordări:

1. **ancorarea** elementelor grafice (de regulă, de cele patru colțuri ale ecranului), folosind o disponere de tip `RelativeLayout` împreună cu o combinație a proprietăților `layout_alignParentTop`, `layout_alignParentBottom`, `layout_alignParentLeft`, `layout_alignParentRight`, `layout_centerHorizontal`, `layout_centerVertical`
2. **redimensionarea și repoziționarea**
  - a. creând un fișier XML corespunzător fiecărei activități pentru fiecare orientare a ecranului, plasate în `/res/layout`, respectiv în `/res/layout-land`
  - b. programatic, detectând modificarea dimensiunilor dispozitivului de afișare în metoda `onCreate()`

```

c. @Override

d. public void onCreate(Bundle state) {
e. super.onCreate(state);
f. WindowManager windowManager = getWindowManager();
g. Display display = windowManager.getDefaultDisplay();
h. if (display.getWidth() <= display.getHeight()) {
i. // create graphic user interface for portrait mode
j. }
k. else {
l. // create graphic user interface for landscape mode
m. }
}

```

De asemenea, o activitate poate fi forțată să afișeze conținutul folosind un singur tip de orientare, indiferent de poziția în care se află dispozitivul mobil:

1. În fișierul `AndroidManifest.xml`, în elementul `<activity>` corespunzător, se specifică proprietatea `android:screenOrientation` cu valorile `portrait`, respectiv `landscape`;
2. programatic, în metoda `onCreate()`, se apelează `setRequestedOrientation()` cu unul din parametrii `ActivityInfo.SCREEN_ORIENTATION_PORTRAIT` sau `ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE`.

Deși în cadrul unui ecran pot fi utilizate combinații între mai multe mecanisme de dispunere a conținutului, se recomandă ca o astfel de abordare să fie evitată întrucât se poate ajunge la situația în care interfața grafică nu scalează pe mai multe dispozitive de afișare cu rezoluții diferite sau prezintă probleme de performanță.

În situația în care se dorește spațierea controalelor grafice, poate fi utilizat un obiect de tip `android.widget.Space`.

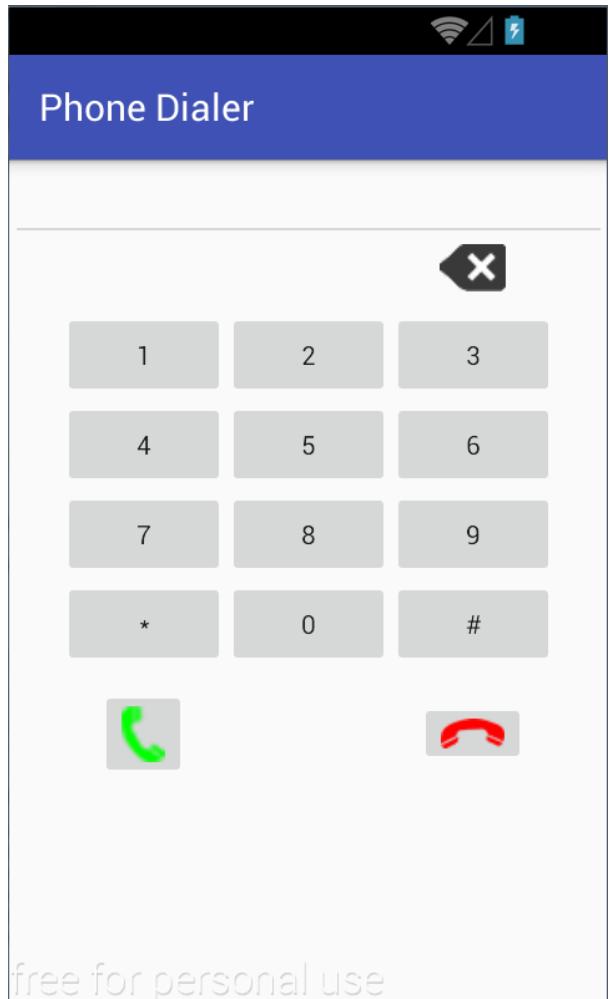
Dacă se folosesc elemente grafice definite de utilizator, se poate recurge la etichetele `<merge>` sau `<include>` pentru a le reutiliza (în loc de a le duplica). De asemenea, un obiect de tip [ViewStub](#) poate fi încărcat în mod dinamic la rulare (numai atunci când este nevoie), în loc de a fi definit static în cadrul interfeței XML.

Pentru optimizarea unei interfețe grafice pot fi utilizate și alte utilitare cum ar fi *Hierarchy View* sau [Lint](#).

## Activitate de Laborator

---

Se dorește implementarea unei aplicații Android, conținând o singură activitate, care să ofere utilizatorilor funcționalitatea necesară pentru formarea unui număr de telefon (PhoneDialer).



**1.** În contul Github personal, să se creeze un depozit denumit 'Laborator03'. Acesta trebuie să conțină unui fișier README.md, un fișier .gitignore specific unei aplicații Android și un fișier LICENSE care să descrie condițiile pentru utilizarea aplicației.

The screenshot shows a GitHub profile page for user **eim2017**. At the top, there's a search bar and navigation links for Pull requests, Issues, and Gist. Below the header, there's a profile picture placeholder and a **ProTip!** message encouraging users to update their profile. The main section displays two repositories: **Laborator01** and **Laborator02**. Repository **Laborator01** was updated 2 minutes ago and contains Java code. Repository **Laborator02** was updated 2 days ago and also contains Java code. There are tabs for Overview, Repositories (2), Stars (0), Followers (0), and Following (0). A search bar and filters for Type (All) and Language (All) are also present.

## Create a new repository

A repository contains all the files for your project, including the revision history.

**Owner**

**eim2017** / **Laborator03**

Great repository names are short and memorable. Need inspiration? How about [musical-octo-barnacle](#).

**Description (optional)**

**Public**

Anyone can see this repository. You choose who can commit.

**Private**

You choose who can see and commit to this repository.

**Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **Android**

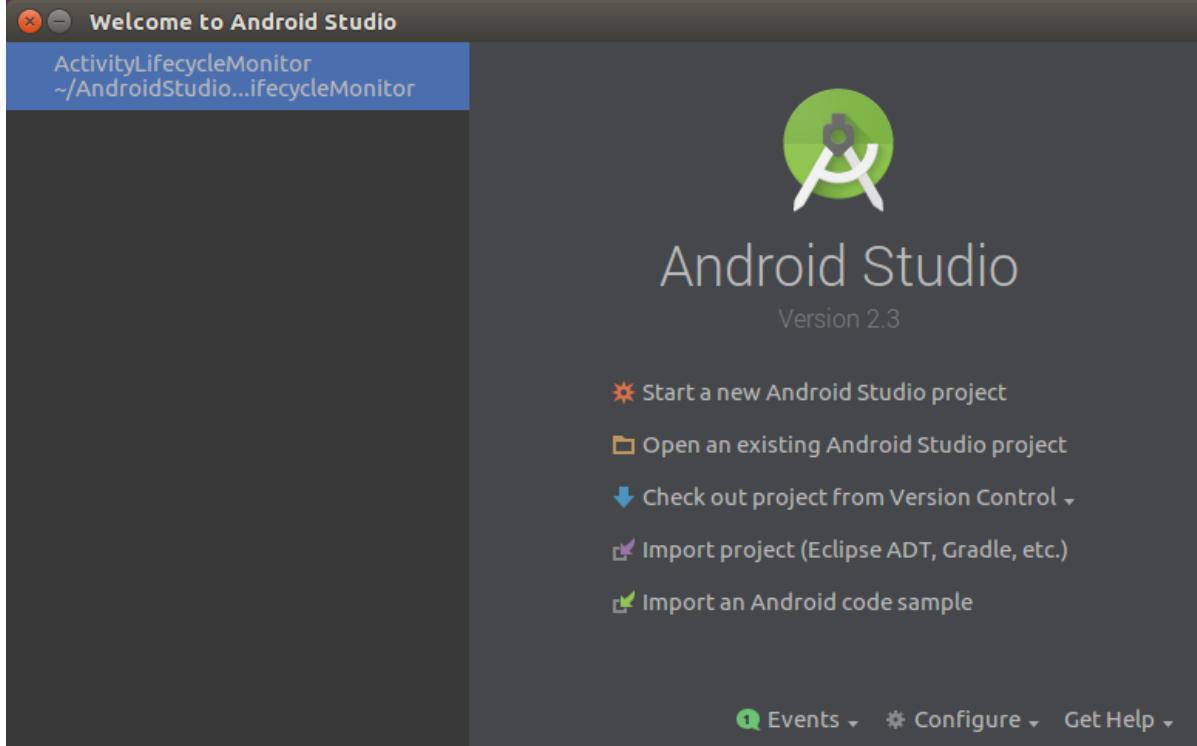
Add a license: **Apache License 2.0**

**Create repository**

**2. Să se cloneze într-un director de pe discul local conținutul depozitului la distanță astfel creat. În urma acestei operații, directorul **Laborator03** va trebui să se conțină fișierele **README.md**, **.gitignore** care indică tipurile de fișiere (extensiile) ignorate și **LICENSE**.**

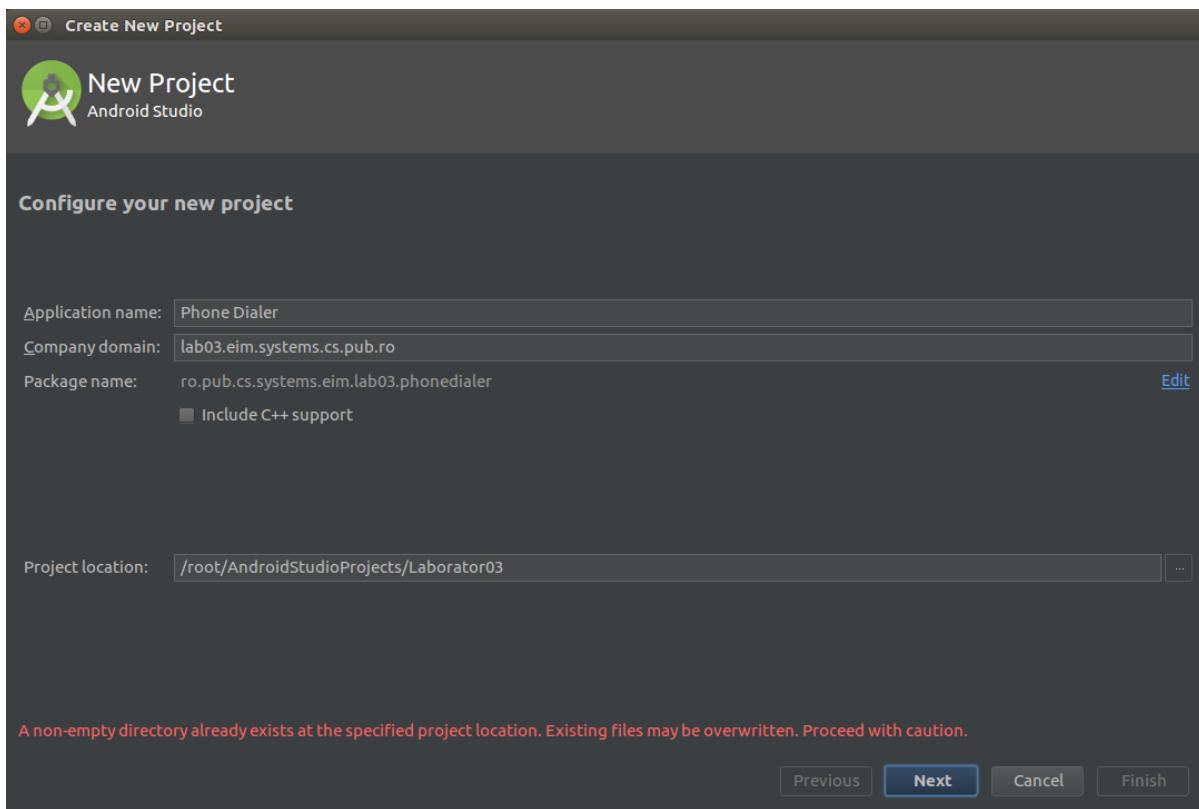
```
student@eim2017:~$ git clone https://www.github.com/perfectstudent/Laborator03.git
```

**3.** În directorul Laborator03 de pe discul local, să se creeze un proiect Android Studio denumit *PhoneDialer* (se selectează *Start a new Android Studio project*).

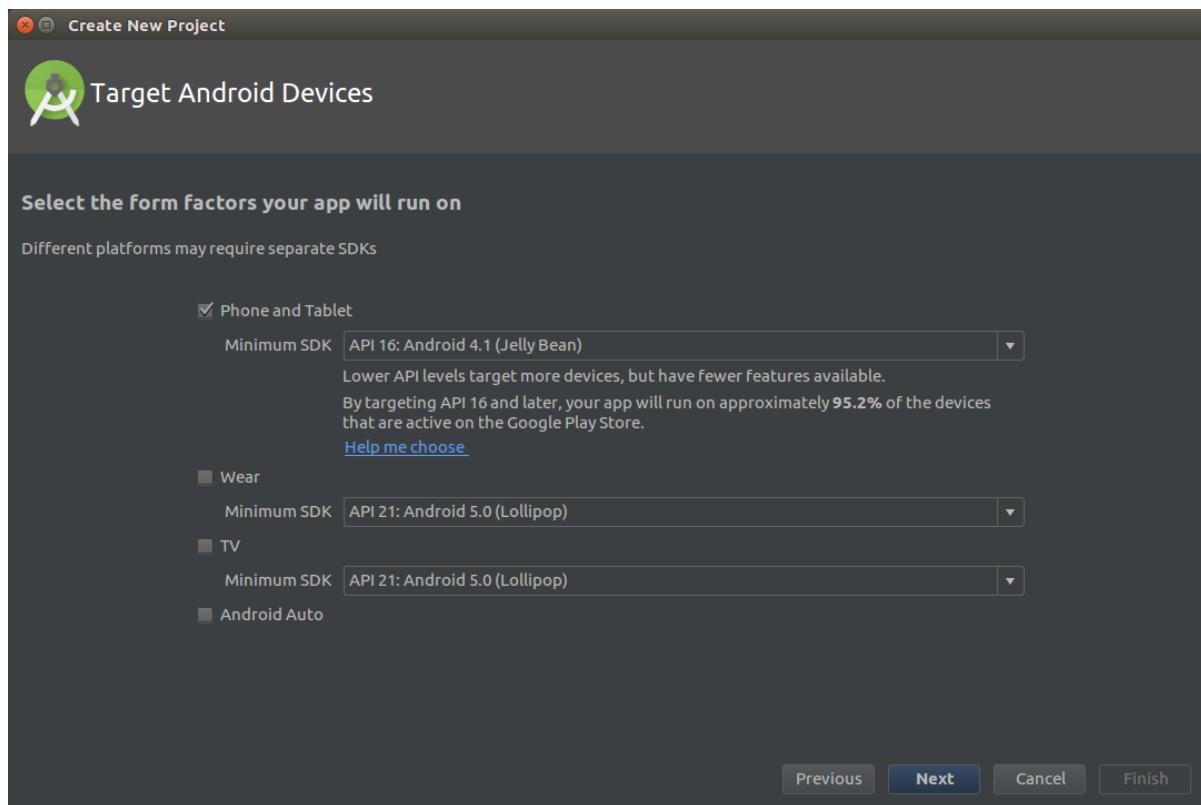


Se indică detaliile proiectului:

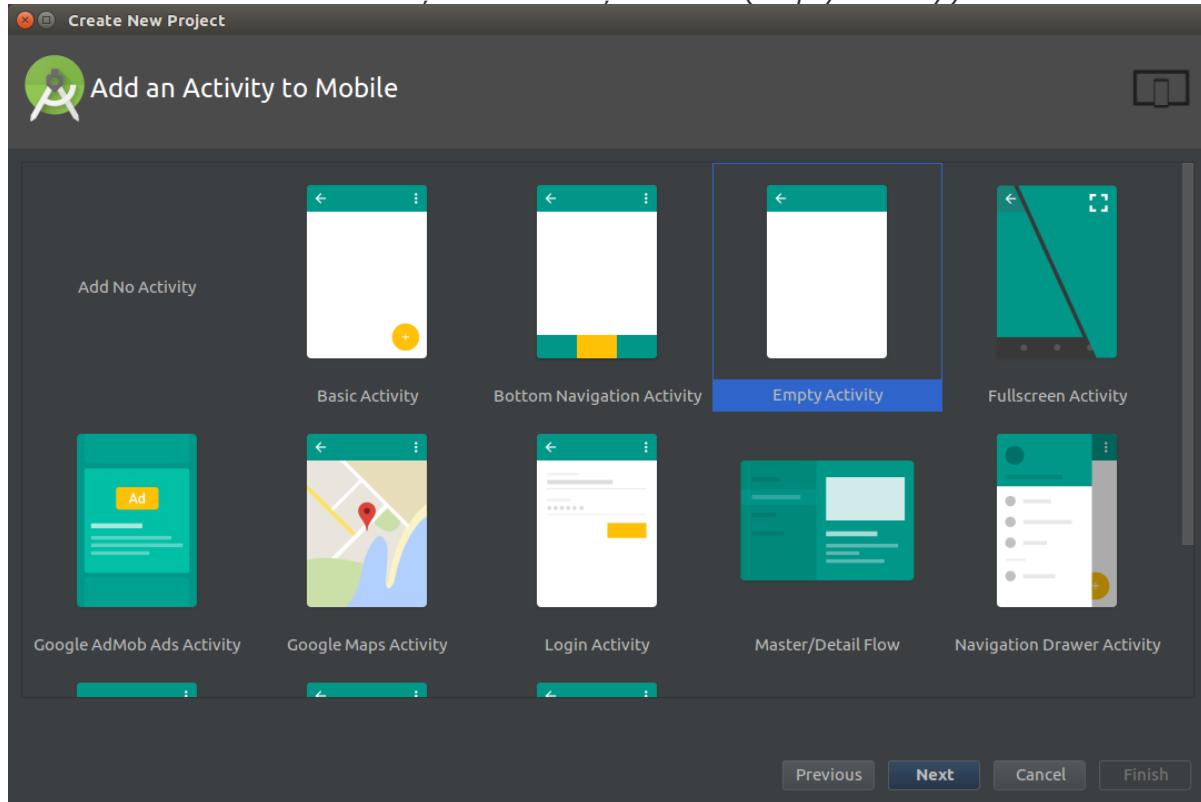
- **Application name** - *Phone Dialer*
- **Company domain** - *lab03.eim.systems.cs.pub.ro* (se va genera în mod automat **Package name** cu valoarea *ro.pub.cs.systems.eim.lab03.phonedialer*)
- **Project location** - locația directorului de pe discul local unde a fost descărcat depozitul la distanță *Laborator03*



Se indică platforma pentru care se dezvoltă aplicația Android (se bifează doar *Phone and Tablet*), iar SDK-ul Android (minim) pentru care se garantează funcționarea este API 16 (Jelly Bean, 4.1).



Se creează o activitate care inițial nu va conține nimic (*Empty Activity*):

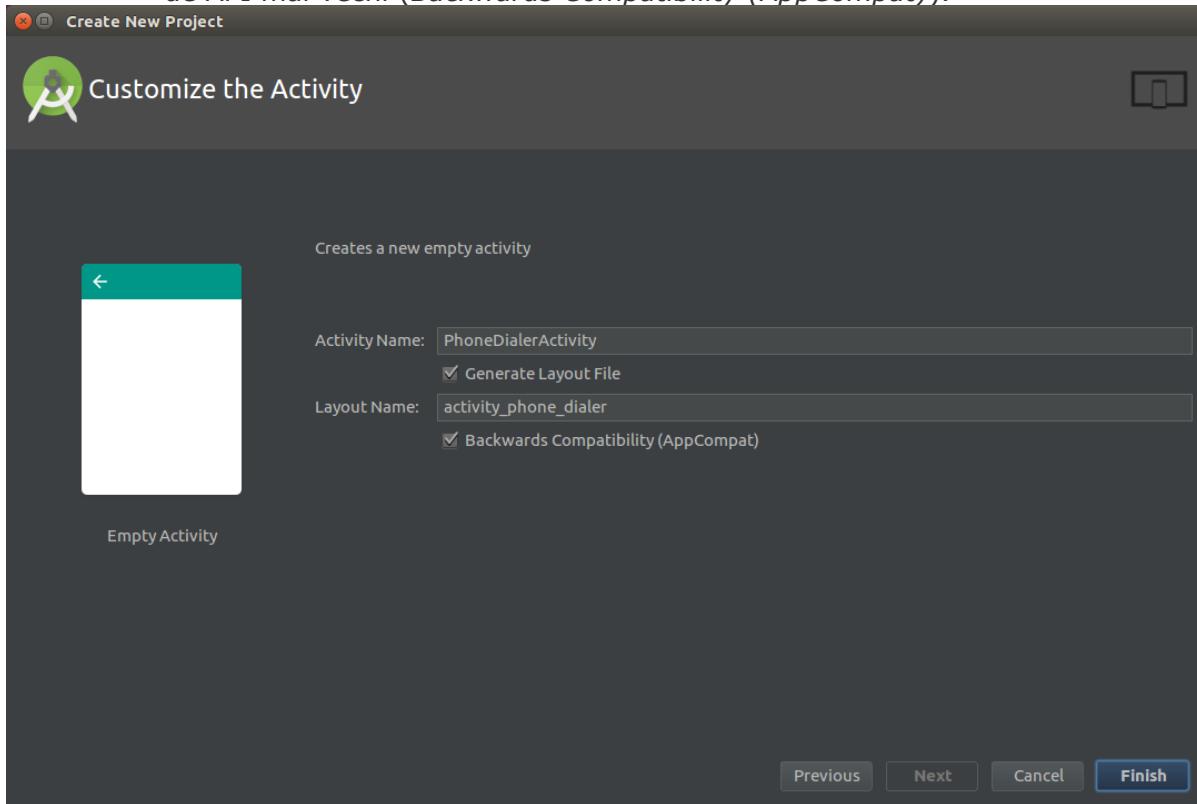


pentru care se precizează:

- **Activity Name** (denumirea activității) - PhoneDialerActivity;
- **Layout Name** (denumirea fișierului XML din res/layout în care va fi construită interfața grafică) - activity\_phone\_dialer.xml.

De asemenea:

- se bifează opțiunea de a se genera în mod automat fișierul XML care conține descrierea interfeței grafice (*Generate Layout File*);
- folosirea claselor din bibliotecile de suport care asigură posibilitatea de folosire a unor funcționalități din SDK-uri mai recente pe dispozitive mobile cu nivele de API mai vechi (*Backwards Compatibility (AppCompat)*).



4. În fișierul activity\_phone\_dialer din directorul res/layout se construiește interfața grafică folosind:

- editorul vizual (*Design*)
- editorul XML (*Text*)

Aceasta va conține:

- un obiect de tip `EditText`, care nu poate fi editat manual de utilizator, în care se va afișa numărul de telefon;
- 10 obiecte de tip `Button` corespunzătoare celor 10 cifre (0..9);

- 2 obiecte de tip `Button` corespunzătoare caracterelor speciale \* și #;
- un obiect de tip `ImageButton` pentru operația de editare a numărului de telefon, prin revenirea la caracterul anterior, în cazul în care s-a greșit;
- 2 obiecte de tip `ImageButton` pentru operațiile de formare a numărului de telefon, respectiv de închidere.

Se pot folosi următoarea arhivă conținând resursele grafice necesare: [graphical resources.zip](#).

Pentru dispunerea controalelor în cadrul interfeței grafice se va folosi un mecanism de tip `LinearLayout` cu orientare verticală, iar tastatura virtuală va fi realizată printr-un obiect de tip `GridLayout` cu 6 linii și 3 coloane.

**5. În** `clasa PhoneDialerActivity din pachetul ro.pub.cs.systems.eim.lab03.phonedialer, să se implementeze o clasă ascultător pentru tratarea evenimentelor de tip apăsare de buton.`

- pentru butoanele ce conțin cifre sau caracterele \* / #, se va adăuga simbolul corespunzător la numărul de telefon care se dorește format;
- pentru butonul de corecție, se va șterge ultimul caracter (în cazul în care numărul de telefon nu este vid);
- pentru butonul de apel, se va invoca intenția care realizează legătura telefonică; încrât se compilează proiectul Android folosind o versiune mai mare decât Marshmallow (6.0), este necesar să fie solicitată permisiunea de efectuare a apelului telefonic la momentul rulării:

```

▪ if (ContextCompat.checkSelfPermission(PhoneDialerActivity.this,
 Manifest.permission.CALL_PHONE) != PackageManager.PERMISSION_GRANTED) {
 ActivityCompat.requestPermissions(
 PhoneDialerActivity.this,
 new String[] {Manifest.permission.CALL_PHONE},
 Constants.PERMISSION_REQUEST_CALL_PHONE);
} else {
 Intent intent = new Intent(Intent.ACTION_CALL);
 intent.setData(Uri.parse("tel:" + phoneNumberEditText.getText().toString()));
 startActivity(intent);
}

```

Pentru a putea apela, în fișierul `AndroidManifest.xml` trebuie să se specifice o permisiune explicită în acest sens:  
`<uses-permission android:name="android.permission.CALL_PHONE" />`

- pentru butonul de oprire, se va închide activitatea (se va apela metoda `finish()`);

Se va defini o clasă internă cu nume, ce implementează interfața `View.OnClickListener` (implementează metoda public void `onClick(View view)`). Instanța acesteia va fi utilizată pentru **toate** obiectele de tip buton din cadrul interfeței grafice.

## 6. Să se gestioneze corespunzător evenimentul de tip rotire a ecranului;

- să se blocheze tranziția între modurile portrait și landscape:

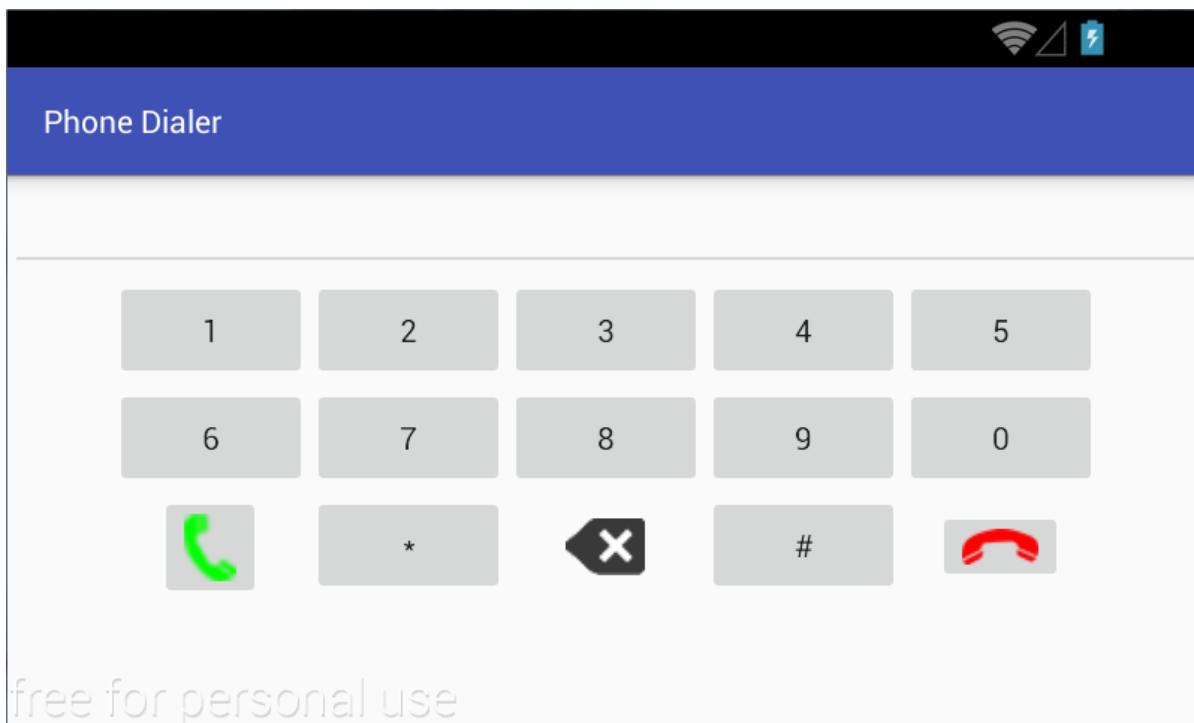
a. în fisierul `AndroidManifest.xml`

```
b. <manifest ...>
c. <application ... >
d. <activity ...
e. android:screenOrientation="portrait"
f. ... />
g.
h. <!-- ... -->
i. </application>
</manifest>
```

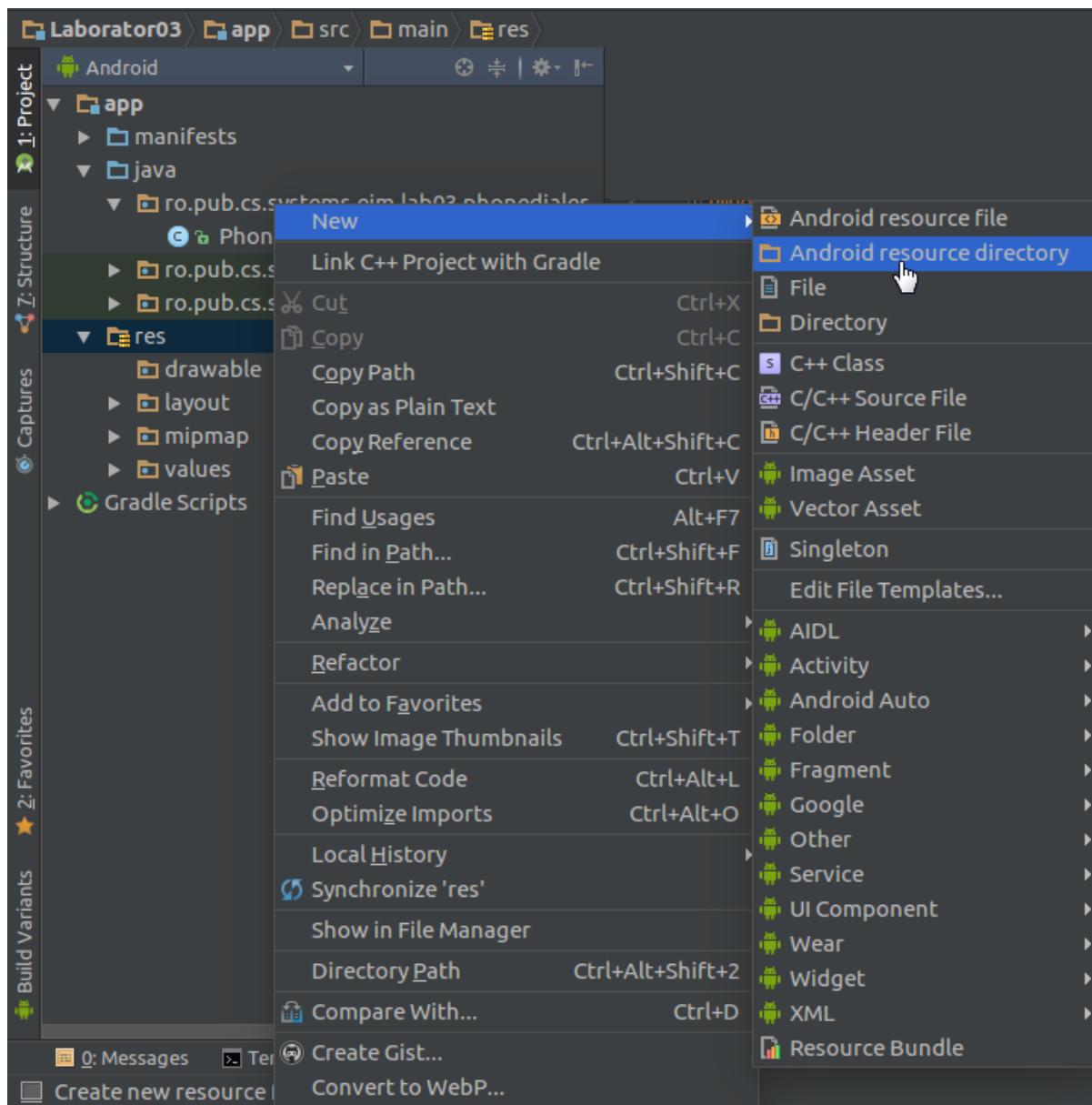
j. programatic

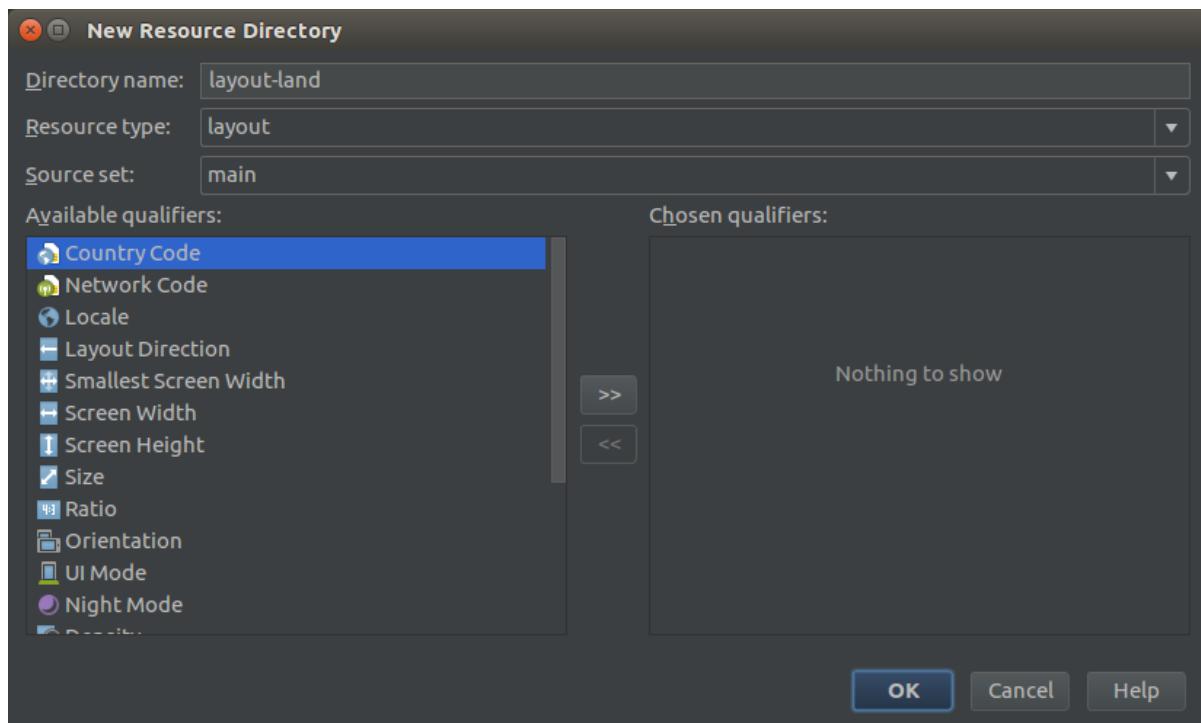
```
k. @Override
l. protected void onCreate(Bundle savedInstanceState) {
m. super.onCreate(savedInstanceState);
n.
o. setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT)
;
o. // ..
}
```

- să se definească, în directorul `res/layout-land` o interfață grafică adekvată acestei configurații a dispozitivului de afișare:

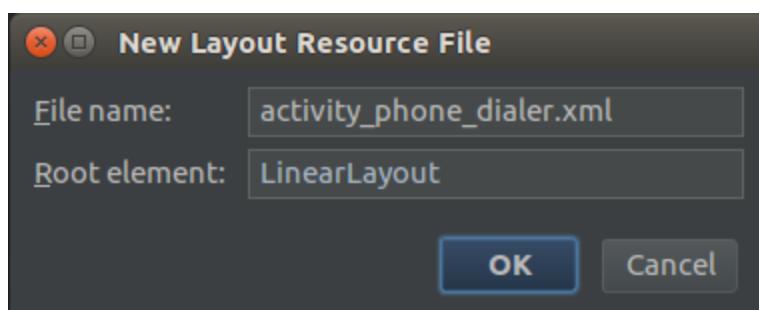
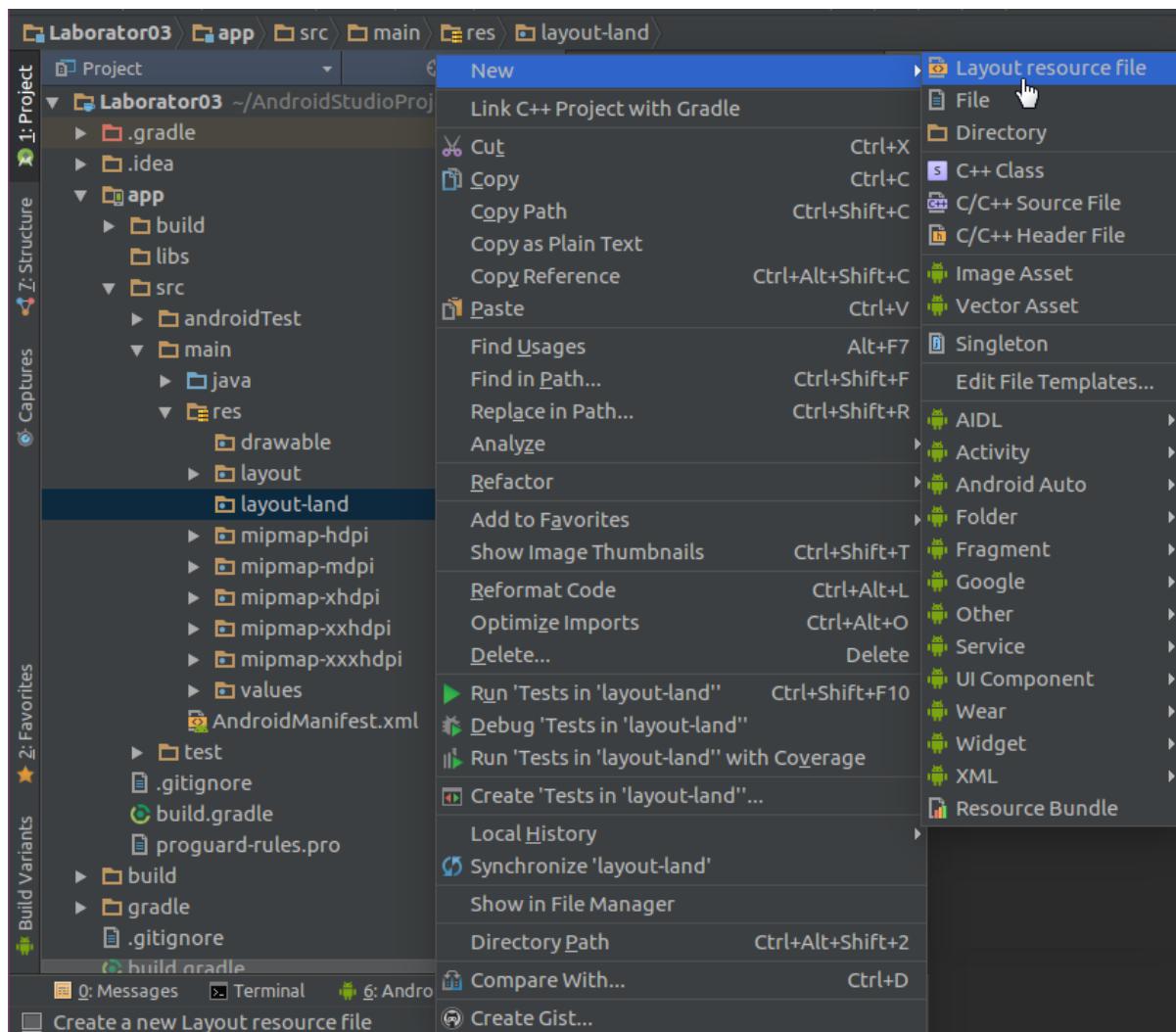


În Android Studio, se creează un nou subdirector `layout-land` în directorul `res` (din meniul contextual afişat cu right click, se selectează *New → Android resource directory*, indicându-se denumirea sa (`layout-land`), tipul de resursă (`layout`) și setul codului sursă pentru care se definește resursa respectivă (`main`)):





Se trece în modul de vizualizare *Project* și în directorul res/layout-land se copiază conținutul fișierului activity\_phone.dialer.xml prin operații de tip copy-paste și se realizează modificările necesare SAU se poate preciza un fișier nou cu aceeași denumire:



7. Să se încarce modificările realizate în cadrul depozitului 'Laborator03' de pe contul Github personal, folosind un mesaj sugestiv.

```
student@eim2017:~/Laborator03$ git add PhoneDialer/*
```

```
student@eim2017:~/Laborator03$ git commit -m "implemented tasks for laboratory 03"
```

```
student@eim2017:~/Laborator03$ git push origin master
```

# Laborator 04. Structura unei Aplicații (II)

## Intenții (obligatoriu)

Conceptul de intenție în Android este destul de complex (și unic), putând fi definit ca o acțiune având asociată o serie de informații, transmisă sistemului de operare Android pentru a fi executată sub forma unui mesaj asincron. În acest fel, intenția asigură interacțiunea între toate aplicațiile instalate pe dispozitivul mobil, chiar dacă fiecare în parte are o existență autonomă. Din această perspectivă, sistemul de operare Android poate fi privit ca o colecție de componente funcționale, independente și interconectate.

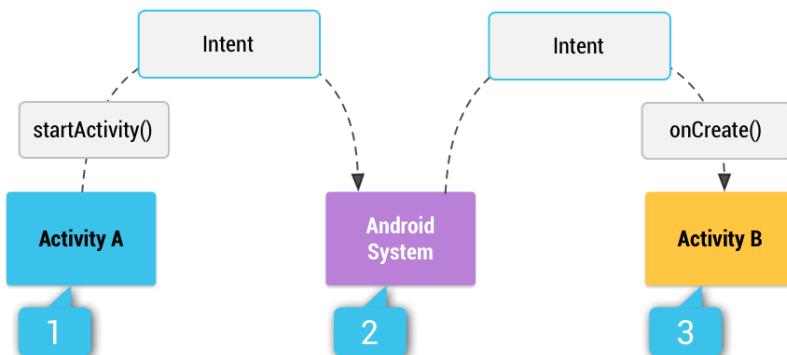
De regulă, o intenție poate fi utilizată pentru:

- a invoca activități din cadrul aceleiași aplicații Android;
- a invoca alte activități, existente în contextul altor aplicații Android;
- a transmite mesaje cu difuzare (*eng. broadcast messages*), care sunt propagate la nivelul întregului sistem de operare Android și pe care unele aplicații Android le pot prelucra, prin definirea unor clase ascultător specifice; un astfel de comportament este util pentru a implementa aplicații bazate pe evenimente.

În sistemul de operare Android, evenimente de tipul primirea unui apel telefonic / mesaj, modificarea nivelului de încărcare al bateriei, schimbarea stării legate de conectivitatea la Internet sunt transmise prin intermediul unor intenții prelucrate de aplicațiile specifice. Un utilizator poate înlocui astfel aplicațiile native cu propriile sale aplicații pentru a realiza operațiile specifice acestor tipuri de evenimente.

O intenție reprezintă o instanță a clasei `android.content.Intent`. Aceasta este transmisă ca parametru unor metode (de tipul `startActivity()` sau `startService()`, definite în clasa abstractă `android.content.Context`), pentru a invoca anumite componente (activități sau servicii). Un astfel de obiect poate încapsula anumite date (împachetate sub formă unui `android.os.Bundle`), care pot fi utilizate de componenta ce se dorește a fi executată prin intermediul intenției.

În programarea Android, un principiu de bază este de folosi intenții pentru a propaga acțiuni, chiar și în cadrul aceleiași aplicații, în detrimentul încărcării clasei corespunzătoare. În acest fel, se asigură cuplarea slabă a componentelor, oferind flexibilitate în cazul înlocuirii acestora, permitând totodată extinderea funcționalității cu ușurință.



Structura unei Intenții

În structura unei intenții pot fi identificate mai multe elemente, precizate în cadrul secțiunii `<intent-filter>`, prin intermediul cărora se declară faptul că o componentă a unei aplicații poate realiza o anumită acțiune pentru un anumit set de date (sau pe care un ascultător al unei intenții cu difuzare îl poate procesa):

- **acțiunea** (`action`) care trebuie executată este indicată prin proprietatea `android:name`, cele mai frecvent utilizate valori fiind: `MAIN`, `VIEW`, `DIAL`, `CALL`, `ANSWER`, `SEND`, `SENDTO`, `INSERT`, `EDIT`, `DELETE`, `SEARCH`, `WEB_SEARCH`; fiecare filtru de intenție trebuie să indice cel puțin o acțiune; este recomandat să se utilizeze convențiile folosite în Java pentru identificarea pachetelor pentru denumirile acțiunilor;

Între o acțiune și componenta pe care aceasta o invocă nu există o relație de tip 1:1, întrucât o acțiune poate determina execuția unor componente diferite, în funcție de tipul de date care sunt atașate acesteia. Astfel, fiecare activitate va defini ca atribute ale elementului `data` din `<intent-filter>` informații legate de categoria MIME ale datelor procesate (`mimeType`), de locația la care se găsesc (`path`, `host`, `port`), de schema utilizată (`scheme`).

- **categoria** (`category`), indicată tot prin proprietatea `android:name` oferă informații suplimentare cu privire la acțiunea care trebuie realizată; fiecare filtru de intenție poate specifica mai multe categorii, putând fi utilizate valori definite de utilizator sau valori preexistente în sistemul de operare Android:
  - `ALTERNATIVE` - acțiunea ar trebui să fie disponibilă ca o alternativă la activitatea implicită pentru tipul de date respectiv;
  - `SELECTED_ALTERNATIVE` - acțiunea poate fi selectată, dintr-o listă, ca o alternativă la activitatea implicită pentru tipul de date respectiv;
  - `BROWSABLE` - indică o acțiune disponibilă din cadrul navigatorului; pentru ca o activitate sau un serviciu să poată fi invocate din cadrul navigatorului trebuie să specifică în mod obligatoriu această categorie;
  - `DEFAULT` - utilizat pentru ca activitatea sau serviciul să fie utilizate în mod implicit pentru tipul de date specificat în filtrul de intenții; de asemenea, este necesar pentru activitățile sau serviciile care se doresc să fie lansate în execuție prin intermediul unor intenții explicate;
  - `HOME` - este folosit pentru a indica o alternativă la ecranul principal, dacă nu este indicată nici o acțiune;
  - `LAUNCHER` - atunci când este specificat, face ca activitatea să fie inclusă în meniul de aplicații care pot fi lansate în execuție direct de către utilizator, prin accesarea lor;
  - alte

**valori:** `INFO`, `PREFERENCE`, `CAR_DOCK`, `DESK_DOCK`, `CAR_MODE`, `APP_MARKET`.

- **datele** (`data`) reprezintă informațiile care vor fi procesate, fiind exprimate de obicei sub forma unui URI, fie că este vorba despre un număr de telefon (prefixat de `tel:`), despre datele unei persoane din lista de contacte (prefixate de `content://contacts/people`), despre coordonate geografice (prefixate de `geo:`) sau o adresă Internet (prefixată de `http://www.`); pot fi specificate o serie de proprietăți (în orice combinație) pentru a indica datele suportate de componentă respectivă:
  - `android:host` - o adresă (accesibilă prin rețea - poate fi indicată denumirea sau adresa IP a gazdei) la care se găsesc datele ce pot fi procesate de componentă;
  - `android:mimeType` - tipul de date;
  - `android:path` - locația la care se găsesc datele;
  - `android:port` - portul pe care se permite conexiunea pentru accesarea datelor;
  - `android:scheme` - un protocol prin intermediu căruia pot fi accesate datele (spre exemplu, `file`, `http`, `mailto`, `content`, `tel`).
- **tipul** (`type`) referă în mod explicit clasificarea MIME a datelor (deși aceasta poate fi dedus în mod automat din conținutul propriu-zis al datelor respective);
- **componenta** (`component`) specifică în mod explicit denumirea unei clase care va fi invocată de intenție (deși aceasta putea fi dedusă în mod automat din denumirea acțiunii și categoria ei, datele și tipul lor);

În situația în care se specifică un nume de componentă, intenția se numește explicită, iar în situația în care aceasta este determinată în funcție de acțiune și de date, intenția se numește implicită.

- **extra** (`extra`) este un obiect de tip `Bundle` care conține informații suplimentare cu privire la componentă respectivă; informațiile conținute într-o intenție pot fi obținute folosind `intent.getExtras()`, în timp ce specificarea unor informații care vor fi atașate la o intenție va fi realizată printr-un apel al metodei `intent.putExtras(Bundle)`.

Dintre aceste componente, esențiale sunt acțiunea și datele:

- acțiunea este transmisă ca parametru al constructorului clasei `Intent`;
- datele sunt transmise prin intermediu metodei `setData()` (pentru acțiunile care necesită date); fiind vorba despre un URI, acesta va fi construit ca rezultat al `Uri.parse(...)`.

După ce a fost construit obiectul de tip `Intent` prin specificarea acestor parametri, acțiunea poate fi executată prin transmiterea acestuia ca parametru al

metodei `startActivity()` sau `startService()`,  
clasa `android.content.Context`.

disponibile

în

```
startActivity(intent); startService(intent);
```

Terminarea unei activități (moment în care se realizează revenirea la activitatea părinte) este realizată prin apelul metodei `finish()`.

## Controlul fluxului de activități prin intenții

### Intenții construite prin precizarea clasei încărcate

În fișierul `AndroidManifest.xml`, orice activitate definește în cadrul elementului `<intent-filter>`, denumirea unei acțiuni care va putea fi folosită de o intenție pentru a o invoca.

```
<activity
 android:name="ro.pub.cs.systems.eim.lab04.MainActivity"
 android:label="@string/app_name" >
 <intent-filter>
 <action
 android:name="ro.pub.cs.systems.eim.lab04.intent.action.MainActivity" />
 <category android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
</activity>
```

Se obișnuiește ca denumirea unei acțiuni să respecte convenția `<pachet-aplicație>.intent.action.ACȚIUNE`.

Pentru fiecare activitate trebuie să existe un element de tip `<activity>` în fișierul `AndroidManifest.xml`.

Pentru ca o activitate să poată fi invocată, aceasta trebuie să specifice la elementul `category` din `<intent-filter>` valoarea `android.intent.category.DEFAULT`.

Pentru ca o funcționalitatea expusă de o activitate să poată fi invocată (în mod anonim) și din contextul altor componente ale sistemului de operare Android, pentru tipul de acțiune și pentru tipurile de date precizate, în cadrul secțiunii `<intent-filter>` trebuie precizat atributul `android:label` (șir de caractere care conține o descriere a funcționalității implementate), indicându-se ca tip de categorie valorile `ALTERNATIVE`, `SELECTED_ALTERNATIVE` sau ambele.

O activitate este în principiu invocată de o intenție care poate fi identificată prin apelul metodei `getIntent()`. Rezultatul acestei metode poate fi inclusiv `null`, în cazul în care activitatea nu a fost pornită prin intermediul unei intenții.

Prin intermediul unei intenții, o aplicație poate invoca atât o activitate din cadrul său, cât și o activitate aparținând altrei aplicații.

- În situația în care se apelează o activitate din cadrul aceleiași aplicații, se poate folosi metoda

```
startActivity(new Intent(this, AnotherActivity.class));
```

- dacă se dorește rularea unei activități existente în cadrul altrei aplicații, aceasta va trebui referită prin numele său complet, inclusiv denumirea pachetului care o identifică

```
startActivity(new
Intent("ro.pub.cs.systems.eim.lab04.AnotherActivity"));
```

De remarcat faptul că în situația în care este pornită o activitate din cadrul aceluiși aplicații Android, obiectul de tip `Intent` primește ca parametru și contextul curent (`this`), în timp ce în cazul în care este lansată în execuție o activitate din cadrul altor aplicații Android acest parametru este omis.

În momentul în care este invocată metoda `startActivity()`, activitatea respectivă este lansată în execuție (prin apelul metodelor `onCreate()`, `onStart()`, `onResume()`) și plasată în vârful stivei care conține toate componente care au rulate anterior, fără a fi fost terminate. În momentul în care se apelează metoda `finish()` (sau se apasă butonul *Back*), activitatea este încheiată (prin apelul metodelor `onPause()`, `onStop()`, `onDestroy()`), fiind scoasă din stivă, restaurându-se activitatea anterioară.

### Intenții construite prin precizarea acțiunii

O intenție poate fi definită și prin intermediul unei acțiuni care se dorește a fi realizată, pentru care pot fi atașate optional și anumite date. Utilizatorul care folosește un astfel de mecanism nu cunoaște activitatea (sau aplicația Android) care va fi lansată în execuție pentru realizarea acțiunii respective. Pentru a putea îndeplini o astfel de solicitare, sistemul de operare Android trebuie să identifice, la momentul rulării, activitatea care este cea mai adecvată pentru a rezolva acțiunea dorită. În acest fel, pot fi utilizate funcționalități deja implementate în cadrul sistemului de operare Android, fără a cunoaște în prealabil aplicația responsabilă de aceasta.

În cazul în care există mai multe activități care au specificat la elementul `action` din `intent-filter` aceeași valoare care este transmisă ca parametru constructorului clasei `Intent`, la execuția intenției în cauză utilizatorului i se va prezenta o listă de opțiuni dintre care poate alege. Dacă la realizarea selecției va fi precizată și opțiunea *Use by default for this action*, preferințele vor fi salvate astfel încât în continuare vor fi utilizate fără a se mai solicita intervenția utilizatorului în acest sens.

Procesul de rezolvare a unei intenții (*eng. intent resolution*) se face prin intermediul analizei tuturor ascultătorilor înregistrați.

Cele mai frecvent utilizate acțiuni implicate ale unui obiect de tip `Intent` sunt:

- vizualizarea conținutului specificat în secțiunea `data` asociată intenției, sub forma unui URI, de către aplicații Android diferite, în funcție de schema (protocolul) utilizat (`http` - navigator, `tel` - aplicația pentru formarea unui număr de telefon, `geo` - Google Maps, `content` - aplicația pentru gestiunea contactelor din agenda telefonică):

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://ocw.cs.pub.ro/eim"));
```

- căutarea unor informații pe Internet folosind un motor de căutare, termenul căutat fiind indicat în secțiunea `extra` asociată intenției, fiind identificată prin cheia `SearchManager.QUERY`:

```
▪ Intent intent = new Intent(Intent.ACTION_WEB_SEARCH);

intent.setData(Uri.parse("http://www.google.ro"));
```

- invocarea aplicației Android pentru formarea unui număr de telefon (interfața grafică a aplicației Android poate fi populată deja cu numărul de telefon furnizat în secțiunea de date a intenției asociate, în cadrul unui URI); aplicația Android nativă poate normaliza majoritatea schemelor de numere de telefon (cod de țară, prefix, număr de telefon propriu-zis):

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

- invocarea aplicației pentru formarea unui număr de telefon și realizarea propriu-zisă a apelului telefonic, folosind valoarea furnizată în secțiunea de date a intenției asociate (în cadrul unui URI); se recomandă să fie folosită pentru activitățile care înlocuiesc aplicația Android nativă pentru formarea unui număr de telefon:

```
▪ Intent intent = new Intent(Intent.ACTION_CALL);

intent.setData(Uri.parse("tel:0214029466"));
```

Pentru a fi posibil ca aplicația să realizeze un apel telefonic, în fișierul `AndroidManifest.xml` trebuie specificată explicit permisiunea pentru o astfel de acțiune

```
<uses-permission android:name="android.permission.CALL_PHONE">
```

- vizualizarea unei locații pe hartă pentru care s-au specificat coordonatele GPS

```
▪ Intent intent = new Intent(Intent.ACTION_VIEW);

intent.setData(Uri.parse("geo:44.436877,26.048029?z=100&q=Educa
tion"));
```

- selectarea unei valori din cadrul furnizorului de conținut indicat în cadrul secțiunii de date asociate intenției, sub forma unui URI; de regulă, este lansată în execuție ca subacvitate, fiind necesar să furnizeze un URI către valoarea care a fost accesată, atunci când este terminată

```
▪ Intent intent = new Intent(Intent.ACTION_PICK);

intent.setData(Uri.parse("content://contacts/people"));
```

Alte acțiuni implicate utilizate sunt:

- ACTION\_ALL\_APPS - lansează în execuție o activitate care afișează toate aplicațiile Android instalate pe dispozitivul mobil; implicit, această acțiune este tratată de aplicația nativă care listează aplicațiile Android în meniul de aplicații din care pot fi rulate de utilizator prin accesarea pictogramei asociate;
- ACTION\_ANSWER - lansează în execuție o activitate care gestionează apelurile primite;
- ACTION\_BUG\_REPORT - lansează în execuție o activitate prin intermediul căruia poate fi raportată funcționarea anormală a unei aplicații Android;
- ACTION\_CALL\_BUTTON - lansează în execuție o activitate responsabilă cu formarea numărului de telefon; de regulă, o astfel de acțiune este generată în momentul în care utilizatorul accesează un buton dedicat;
- ACTION\_DELETE - lansează în execuție o activitate care permite ștergerea informațiilor specificate în secțiunea `data` asociată intenției, sub forma unui URI;
- ACTION\_EDIT - lansează în execuție o activitate care permite modificarea informațiilor specificate în secțiunea `data` asociată intenției, sub forma unui URI;
- ACTION\_INSERT - lansează în execuție o activitate care permite adăugarea unor informații în cursorul specificat în secțiunea de secțiunea `data` asociată intenției, sub forma unui URI (în cazul în care este rulată ca subactivitate, trebuie să furnizeze URI-ul informațiilor adăugate);
- ACTION\_SEARCH - lansează în execuție o activitate care implementează o activitate de căutare; termenul care este căutat trebuie să fie specificat în secțiunea `extra` a activității, fiind identificat prin cheia `SearchManager.QUERY`;
- ACTION\_SEARCH\_LONG\_PRESS - lansează în execuție o activitate care implementează o activitate de căutare, fiind generată în momentul în care este detectat un eveniment de tip apăsare prelungită a unui buton dedicat (implicit, lansează în execuție o activitate pentru a realiza o căutare vocală);
- ACTION\_SENDTO - lansează în execuție o activitate pentru a transmite anumite informații către un contact indicat în secțiunea `data` asociată intenției;

- ACTION\_SEND - lansează în execuție o activitate care transmite informațiile conținute în cadrul intenției către un contact care va fi selectat ulterior (aflat pe un alt dispozitiv mobil):
  - tipul MIME trebuie indicat prin intermediul metodei `setType()`;
  - informațiile propriu-zise trebuie conținute în secțiunea `extra` asociată intenției, fiind identificate prin cheile `EXTRA_TEXT` sau `EXTRA_STREAM`, în funcție de tipul respectiv (pentru aplicațiile de poștă electronică sunt suportate și cheile `EXTRA_EMAIL`, `EXTRA_CC`, `EXTRA_BCC`, `EXTRA_SUBJECT`).

Totuși, un utilizator nu poate avea garanția că acțiunea pe care o transmite ca parametru al unei intenții va putea fi rezolvată, întrucât există posibilitatea să nu existe nici o activitate asociată acesteia sau ca aplicația ce ar fi putut să o execute să nu fie instalată în contextul sistemului de operare Android. Din acest motiv, o practică curentă este de a verifica dacă o acțiune poate fi rezolvată **înainte** de a apela metoda `startActivity()`. Astfel, procesul de gestiune a pachetelor poate fi interogat (prin intermediul metodei `resolveActivity()`) dacă există o activitate ce poate executa o acțiune și în caz afirmativ, care este aceasta.

```
Intent applicationIntent = new Intent(...);
PackageManager packageManager = new PackageManager();
ComponentName componentName = null;
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(intent);
}
```

În situația în care nu este identificată nici o activitate asociată acțiunii respective, utilizatorul poate dezactiva componenta grafică asociată până în momentul în care aceasta devine disponibilă, prin descărcarea aplicației Android corespunzătoare din Google Play Store.

Prin intermediul clasei `PackageManager` poate fi obținută lista tuturor acțiunilor care pot fi realizate pentru un set de date, atașat unei intenții, invocându-se metoda `queryIntentActivities()`:

```
Intent applicationIntent = new Intent();
intent.setData(...);
intent.addCategory(Intent.CATEGORY_SELECTED_ALTERNATIVE);

PackageManager packageManager = new PackageManager();
int flags = PackageManager.MATCH_DEFAULT_ONLY;

List<ResolveInfo> availableActions = packageManager.queryIntentActivities(applicationIntent, flags);
```

```
for (ResolveInfo availableAction: availableActions) {
 Log.d(Constants.TAG, "An available action for the data is
 "+getResources().getString(availableAction.labelRes));
}
```

Procesul de rezolvare a unei intenții pe baza unei acțiuni implică următoarele etape:

1. se construiește o listă cu toate filtrele de intenții asociate componentelor din aplicațiile Android existente;
2. sunt eliminate toate filtrele de intenții care nu corespund acțiunii sau categoriei intenției care se dorește a fi rezolvată:
  - a. verificările în privința acțiunii sunt realizate numai în situația în care filtrul de intenție specifică o acțiune; sunt eliminate acele filtre de intenții pentru care **nici una** dintre acțiunile pe care le include nu corespund acțiunii intenției care se dorește a fi rezolvată;
  - b. verificările în privința categorie sunt realizate numai în situația în care filtrul de intenție specifică o categorie sau în cazul în care nu specifică nici o categorie, dacă nici intenția care se dorește a fi rezolvată nu include nici o categorie; sunt eliminate acele filtre de intenții care nu includ **toate** categoriile pe care le conține și intenția care se dorește a fi rezolvată (putând conține totuși și categorii suplimentare);
3. fiecare parte a URI-ului datelor corespunzătoare intenției care se dorește a fi rezolvată este comparată cu secțiunea `data` din filtrul de intenție; gazda, autoritatea, tipul MIME, calea, portul, schema), orice neconcordanță conduce la eliminarea acestuia din listă (în situația în care filtrul de intenții nu specifică proprietăți în secțiunea `data`, acesta va fi considerat compatibil cu intenția care se dorește a fi rezolvată);
4. în situația în care, ca urmare a acestui proces, există mai multe componente rămase în listă, utilizatorul va trebui să aleagă dintre toate aceste posibilități.

Aplicațiile Android native sunt supuse aceluiași proces în momentul în care se realizează rezolvarea unei intenții ca și aplicațiile Android instalate din alte surse, având aceeași prioritate și putând fi chiar înlocuite de acestea, dacă definesc filtre de intenții cu aceleași acțiuni / categorii.

În cazul în care o componentă a unei activități este lansată în execuție prin intermediul unei intenții, aceasta trebuie să identifice acțiunea pe care trebuie să o realizeze și datele pe care trebuie să le proceseze. În acest sens, clasa Intent pune la dispoziție metodele `getAction()`, `getData()` și `getExtras()`.

```
@Override
```

```
protected void onCreate(Bundle state) {
```

```

 super.onCreate(state);
 setContentView(R.layout.activity_main);
 Intent intent = getIntent();
 if (intent != null) {
 String action = intent.getAction();
 Uri data = intent.getData();
 Bundle extras = intent.getExtras();
 }
 }
}

```

În anumite situații, o componentă poate primi și alte intenții după ce a fost creată. De fiecare dată, va fi apelată în mod automat metoda `onNewIntent()`:

```

@Override

public void onNewIntent(Intent newIntent) {
 super.onNewIntent(newIntent);
 // ...
}

```

O componentă are de asemenea posibilitatea de a transfera responsabilitatea cu privire la gestiunea unei intenții către altă componentă care corespunde criteriilor legate de acțiune și categorie, prin intermediul metodei `startNextMatchingActivity()`:

```

Intent intent = getIntent();
if (intent != null) {
 startNextMatchingActivity(intent);
}

```

În acest mod, o componentă poate indica condiții suplimentare cu privire la tratarea unei anumite acțiuni, în situația în care acestea nu pot fi exprimate în cadrul filtrului de intenții, pentru a putea fi luate în considerare în cadrul procesului automat de identificare a componentei care deservește o intenție.

## Intenții construite prin intermediul unui URI

De asemenea, un obiect de tip `Intent` poate fi creat și prin intermediul unui URI care identifică în mod unic o anumită activitate:

```

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

```

Pentru a putea fi apelată folosind acest mecanism, activitatea va trebui să definească elementul `data` în cadrul `<intent-filter>`:

```

<activity
 android:name="ro.pub.cs.systems.eim.lab04.AnotherActivity"
 android:label="@string/app_name" >
 <intent-filter>
 <action
 android:name="ro.pub.cs.systems.eim.lab04.intent.action.AnotherActivity" />
 <category android:name="android.intent.category.DEFAULT" />
 <data
 android:scheme="myprotocol"
 android:host="mynamespace" />
 </intent-filter>

```

```
</activity>
```

De remarcat este faptul că în structura URI-ului, partea de după schemă://protocol/ poate conține orice sir de caractere, rolul său fiind strict acela de a respecta forma unui astfel de obiect (estetic), fără a influența în vreo formă funcționalitatea acestuia.

## Transmiterea de informații între componente prin intermediul intențiilor

Intențiile pot încapsula anumite informații care pot fi partajate de componentele între care fac legătura (însă unidirecțional, de la componenta care invocă spre componenta invocată!) prin intermediul secțiunii extra care conține un obiect de tip Bundle. Obținerea valorii secțiunii extra corespunzătoare unei intenții poate fi obținute folosind metoda getExtras(), în timp ce specificarea unor informații care vor fi asociate unei intenții poate fi realizată printr-un apel al metodei putExtras().

În cazul în care o intenție are deja atașat un obiect de tip Bundle în momentul în care se apelează metoda putExtras(), perechile (cheie, valoare) vor fi transferate din cadrul parametrului metodei în obiectul deja existent.

Construirea unui obiect de tip Bundle care să fie transmis ca parametru al metodei putExtras() poate fi evitată prin utilizarea metodei putExtra() apelată pe obiectul Intent, primind ca parametrii denumirea cheii prin care datele vor fi identificate și o valoare având un tip compatibil cu android.os.Parcelable. Obținerea datelor se realizează apelând metoda pereche getExtra() căreia î se transmite denumirea cheii ce identifică în mod unic informațiile respective. De asemenea, sunt implementate și metode specifice pentru fiecare tip de dată (put<type>Extra(), respectiv get<type>Extra()).

Se recomandă ca pentru cheie să se utilizeze o denumire prefixată de pachetul aplicației.

O activitate copil, lansată în execuție prin intermediul metodei startActivity(), este independentă de activitatea părinte, astfel încât aceasta nu va fi notificată cu privire la terminarea sa. În situațiile în care un astfel de comportament este necesar, activitatea copil va fi pornită de activitatea părinte ca subactivitate care transmite înapoi un rezultat. Acest lucru se realizează prin lansarea în execuție a activității copil prin intermediul metodei startActivityForResult(). În momentul în care este finalizată, va fi invocată automat metoda onActivityResult() de la nivelul activității părinte.

La nivelul **activității părinte**, vor trebui implementate:

- metoda startActivityForResult() va primi ca parametrii obiectul de tip Intent precum și un cod de cerere (de tip întreg), utilizat pentru a identifica în mod unic activitatea copil care a transmis un rezultat;
- metoda onActivityResult() care va fi apelată în mod automat în momentul în care activitatea copil a fost terminată; parametrii pe care îi furnizează aceasta sunt:
  - codul de cerere (prin care se distinge între instanțe diferite ale activității copil);
  - codul de rezultat transmis activității părinte (poate avea valorile RESULT\_OK sau RESULT\_CANCELED);
  - un obiect Intent prin intermediul căruia pot fi furnizate date suplimentare.

```

final private static int ANOTHER_ACTIVITY_REQUEST_CODE = 2017;

@Override
protected void onCreate(Bundle state) {
 super.onCreate(state);
 setContentView(R.layout.activity_main);
 Intent intent = new Intent("ro.pub.cs.systems.eim.lab04.AnotherActivity");
 intent.putExtra("ro.pub.cs.systems.eim.eim.someKey", someValue);
 startActivityForResult(intent, ANOTHER_ACTIVITY_REQUEST_CODE);
 // start another activities with their own request codes
}

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

 // process other request codes
 }
}

```

În **activitatea copil**, înainte de apelul metodei `finish()`, va trebui transmis activității părinte codul de rezultat (`Activity.RESULT_OK`, `Activity.RESULT_CANCELED` sau orice fel de rezultat de tip întreg) și obiectul de tip intenție care conține datele (optional, în situația în care trebuie întoarse rezultate explicit), ca parametrii ai metodei  `setResult()`.

```

@Override

protected void onCreate(Bundle state) {
 super.onCreate(state);
 setContentView(R.layout.activity_another);

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

 // intent to parent
 Intent intentToParent = new Intent();
 intent.putExtra("ro.pub.cs.systems.eim.lab04.anotherKey", anotherValue);
 setResult(RESULT_OK, intentToParent);
 finish();
}

```

În cazul folosirii unor intenții în care activitățile sunt invocate prin intermediul unor URI-uri, datele vor putea fi concatenate direct în cadrul acestuia (fără a utiliza un obiect de tip `Bundle`), restricția constând în faptul că pot fi utilizate numai siruri de caractere:

- În activitatea părinte

```

 ▪ Uri uri = Uri.parse("myprotocol://mynamespace/myactivity?someKey=someValue&
 ...");
 ▪ Intent intent = new Intent(Intent.ACTION_VIEW, uri);
 ▪ startActivity(intent);

 ▪ în activitatea copil
 ▪ Uri uri = getIntent().getData();
 ▪ String someValue = uri.getQueryParameter("someKey");

```

## Gestiunea evenimentelor cu difuzare prin intermediul intențiilor

Intentiile reprezintă și un mecanism de comunicație inter-proces, asigurând transferul unor mesaje structurate. Astfel, intențiile pot fi distribuite către toate componentele de la nivelul sistemului de operare Android, pentru a notifica producerea unui eveniment (legat de starea dispozitivului mobil sau a unor aplicații), fiind procesate în cadrul unor obiecte ascultător dedicate tipului de mesaj respectiv.

Și sistemul de operare Android folosește acest mecanism pentru a notifica producerea unor modificări la nivelul stării curente (primirea unui apel telefonic / mesaj, schimbarea nivelului de încărcare al bateriei sau a conectivității).

Trebuie realizată distincția între intențiile cu difuzare transmise la nivelul întregului sistem de operare Android și a celor transmise doar la nivelul aplicației, prin intermediul unui obiect de tipul LocalBroadcastManager (a cărui instanță se obține prin intermediul metodei statice getInstance() ce primește ca parametru contextul aplicației curente). Aceasta operează într-un mod similar, implementând metodele sendBroadcast() și registerReceiver(). În plus, dispune de o metodă ce permite trimiterea sincronă a notificărilor, apelul acesteia fiind blocant până la momentul în care toți ascultătorii le-au primit.

Pentru o aplicație Android, în momentul rulării, pot fi activate / dezactivate oricare dintre componente (deci inclusiv ascultătorii pentru intențiile cu difuzare) prin intermediul metodei setComponentEnabledSetting() din cadrul clasei PackageManager. Un astfel de comportament este util pentru a optimiza performanțele aplicației atunci când o anumită funcționalitate nu este necesară.

```

PackageManager packageManager = getPackageManager();

ComponentName someEventBroadcastReceiver = new ComponentName(this,
SomeEventBroadcastReceiver.class);

packageManager.setComponentEnabledSetting(someEventBroadcastReceiver ,
PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
PackageManager.DONT_KILL_APP);

```

```
packageManager.setComponentEnabledSetting(someEventBroadcastReceiver ,
PackageManager.COMPONENT_ENABLED_STATE_DISABLED,
PackageManager.DONT_KILL_APP);
```

## Trimitera unei intenții cu difuzare

Construirea unei intenții care urmează să fie difuzată la nivelul sistemului de operare Android poate fi realizată prin definirea unui obiect de tipul Intent, pentru care se vor specifica acțiunea, datele și categoria, astfel încât obiectele de tip ascultător să îl poată identifica cât mai exact. Ulterior, acesta va fi trimis tuturor proceselor aferente aplicațiilor instalate pe dispozitivul mobil prin intermediul metodei sendBroadcast(), căreia îi este atașat ca parametru.

Pot fi utilizate atât acțiuni predefinite (care vor fi procesate atât de aplicațiile Android native cât și de eventuale aplicații instalate din alte surse) cât și acțiuni definite de utilizator, pentru care trebuie implementate aplicații dedicate, responsabile cu procesarea acestora.

```
final public static String SOME_ACTION =
"ro.pub.cs.systems.eim.lab04.SomeAction.SOME_ACTION";

Intent intent = new Intent(SOME_ACTION);
intent.putExtra("ro.pub.cs.systems.eim.lab04.someKey", someValue);
sendBroadcast(intent);
```

## Primirea unui intenții cu difuzare

Pentru a putea primi o intenție cu difuzare, o componentă trebuie să fie înregistrată în acest sens, definind un filtru de intenții pentru a specifica ce tipuri de acțiuni și ce tipuri de date asociate intenției poate procesa.

Acesta poate fi precizat:

- în fișierul AndroidManifest.xml (caz în care nu este necesar ca aplicația să ruleze la momentul în care se produce evenimentul cu difuzare pentru a-l putea procesa); elementul <receiver> trebuie să conțină în mod obligatoriu filtrul de intenții prin care se indică acțiunea care poate fi procesată:

```
AndroidManifest.xml

<manifest ... >
 <application ... >
 <receiver
 android:name=".SomeEventBroadcastReceiver">
 <intent-filter>
```

```
 <action
 android:name="ro.pub.cs.systems.eim.lab04.SomeAction.SOME_ACTION
" />
 </intent-filter>
 </receiver>
</application>
</manifest>
```

- programatic, în codul sursă (caz în care aplicația trebuie să fie în execuție la momentul în care se produce evenimentul cu difuzare pentru a-l putea procesa); o astfel de abordare este utilă când procesarea intenției cu difuzare implică actualizarea unor componente din cadrul interfeței grafice asociate activității:

```
▪ private SomeEventBroadcastReceiver someEventBroadcastReceiver =
 new SomeEventBroadcastReceiver();
▪ private IntentFilter intentFilter = new
 IntentFilter(SOME_ACTION);
▪
▪ @Override
▪ protected void onResume() {
 super.onResume();
▪
▪ registerReceiver(someEventBroadcastReceiver, intentFilter);
}
▪
▪ @Override
▪ protected void onPause() {
 super.onPause();
▪
▪ unregisterReceiver(someEventBroadcastReceiver);
}
```

O regulă este de a înregistra obiectul ascultător pe metoda `onResume()` și de a-l deînregistra pe metoda `onPause()`, astfel încât acesta să nu reacționeze decât atunci când activitatea este vizibilă.

O clasă capabilă să proceseze intenții cu difuzare este derivată din `android.content.BroadcastReceiver`, implementând metoda `onReceive()` pe care realizează rutina de tratare propriu-zisă:

#### SomeEventBroadcastReceiver.java

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

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

Metoda `onReceive()` va fi invocată în mod automat în momentul în care este primită o intenție cu difuzare, fiind executată pe firul de execuție principal al aplicației. De regulă, în cadrul acestei metode utilizatorul este anunțat asupra producerii evenimentului prin intermediul serviciului de notificare (*Notification Manager*), este lansat în execuție un serviciu sau sunt actualizate componente din cadrul interfeței grafice.

Este necesar ca metoda `onReceive()` să se termine în maximum 5 secunde, în caz contrar fiind afișată o fereastră de dialog pentru a determina oprirea sa forțată.

## Tipuri particulare de intenții cu difuzare

Există și tipuri particulare de intenții cu difuzare:

1. intenții cu difuzare **ordonate**, utile în situația în care o intenție cu difuzare trebuie să fie procesată secvențial de mai mulți ascultători, fiecare dintre aceștia având posibilitatea de a modifica intenția respectivă;
2. intenții cu difuzare **persistente**, care mențin valoarea care a fost transmisă cel mai recent.

### Intenții cu difuzare ordonate

Optional

### Intenții cu difuzare persistente

Optional

## Gestiunea intențiilor cu difuzare native

Cele mai multe servicii de sistem transmit intenții cu difuzare pentru a semnala faptul că s-au produs anumite modificări la nivelul stării dispozitivului mobil sau al aplicațiilor (primirea unui apel telefonic / mesaj, schimbarea nivelului de încărcare al bateriei, conectivitatea la Internet).

ACTIUNE	DESCRIERE
ACTION_BATTERY_CHANGED	acțiune transmisă în momentul în care se modifică nivelul de încărcare al bateriei; starea bateriei este

	disponibilă în secțiunea <code>extra</code> , prin intermediul cheii <code>EXTRA_STATUS</code> , putând avea valorile: ♦ <code>BatteryManager.BATTERY_STATUS_CHARGING</code> ♦ <code>BatteryManager.BATTERY_STATUS_FULL</code>
<code>ACTION_BATTERY_LOW</code>	acțiune transmisă în momentul în care nivelul de încărcare al bateriei este scăzut, impunându-se încărcarea acesteia
<code>ACTION_BATTERY_OKAY</code>	acțiune transmisă în momentul în care nivelul de încărcare al bateriei este acceptabil
<code>ACTION_BATTERY_CONNECTED</code>	acțiune transmisă în momentul în care bateria este conectată la o sursă de energie externă
<code>ACTION_BATTERY_DISCONNECTED</code>	acțiune transmisă în momentul în care bateria este deconectată de la o sursă de energie externă
<code>ACTION_BOOT_COMPLETED</code>	acțiune transmisă în momentul în care a fost realizată complet secvența de pornire a dispozitivului mobil (aplicația poate primi o astfel de intenție cu difuzare dacă deține permisiunea <code>RECEIVE_BOOT_COMPLETED</code> )
<code>ACTION_CAMERA_BUTTON</code>	acțiune transmisă în momentul în care este accesat butonul pentru pornirea camerei foto
<code>ACTION_DATE_CHANGED / ACTION_TIME_CHANGED</code>	acțiuni transmise în momentul în care data calendaristică sau timpul sunt modificate manual (nu datorită progresului său natural)
<code>ACTION_DOCK_EVENT</code>	acțiune transmisă în momentul în care dispozitivul mobil este ancorat, printr-un dispozitiv de birou sau de mașină, stare plasată în secțiunea <code>extra</code> prin intermediul cheii <code>ETRA_DOCK_STATE</code>
<code>ACTION_MEDIA_EJECT</code>	acțiune transmisă în momentul în care este îndepărtat un mediu de stocare extern (util în situația în care aplicația scrie / citește de pe acesta, pentru a salva conținutul și pentru a le închide)
<code>ACTION_MEDIA_MOUNTED / ACTION_MEDIA_UNMOUNTED</code>	acțiuni transmise de fiecare dată când dispozitive de stocare externe sunt adăugate sau îndepărtate cu succes
<code>ACTION_NEW_OUTGOING_CALL</code>	acțiune transmisă în momentul în care urmează să fie format un număr de telefon, a cărui valoare este plasată în secțiunea <code>extra</code> , prin intermediul cheii <code>EXTRA_PHONE_NUMBER</code> (aplicația poate primi o astfel de intenție cu difuzare dacă deține permisiunea <code>PROCESS_OUTGOING_CALLS</code> )
<code>ACTION_SCREEN_OFF / ACTION_SCREEN_ON</code>	acțiuni transmise în momentul în care ecranul este închis, respectiv este deschis
<code>ACTION_TIMEZONE_CHANGED</code>	acțiune transmisă în momentul în care zona de timp a telefonului este modificată, a cărui valoare (identificator) este plasată în secțiunea <code>extra</code> prin intermediul cheii <code>time-zone</code>

Pentru aceste tipuri de intenții cu difuzare, înregistrarea și deînregistrarea unor obiecte de tip ascultător poate fi realizată numai programatic, în codul sursă.

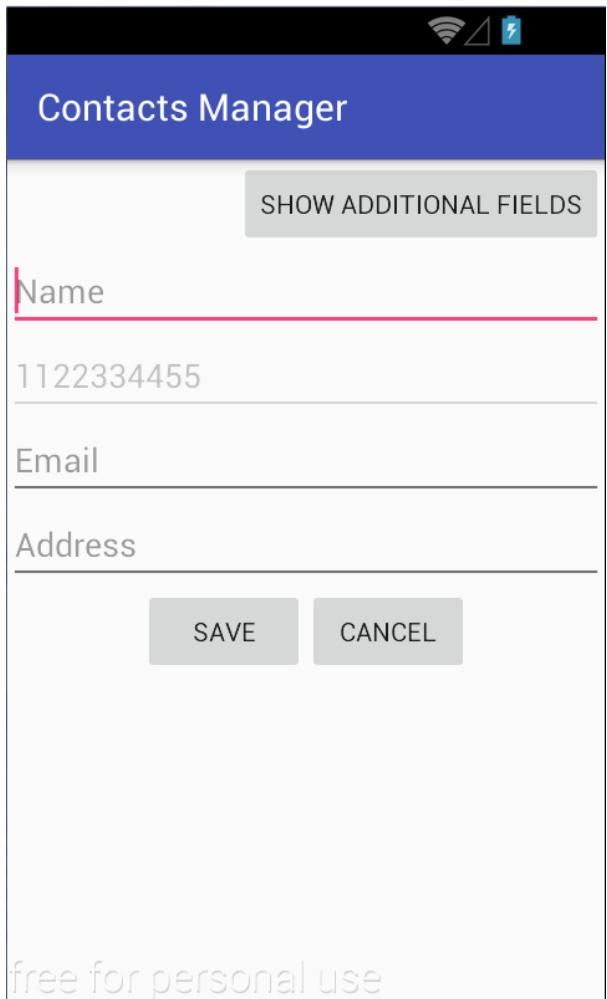
În cazul unei aplicații Android, foarte importante sunt și modificările în privința conectivității la Internet (inclusiv parametrii precum lățimea de bandă, latența) întrucât acestea pot fi semnificative în privința luării unor decizii legate de realizarea unui actualizări sau de descărcarea unor fișiere având dimensiuni mari. O astfel de funcționalitate poate fi definită prin implementarea unui obiect ascultător, care procesează acțiunea `android.net.conn.CONNECTIVITY_CHANGE` (`ConnectivityManager.CONNECTIVITY_ACTION`). Se transmise o intenție cu difuzie nepersistență care nu conține informații suplimentare cu privire la schimbarea stării.

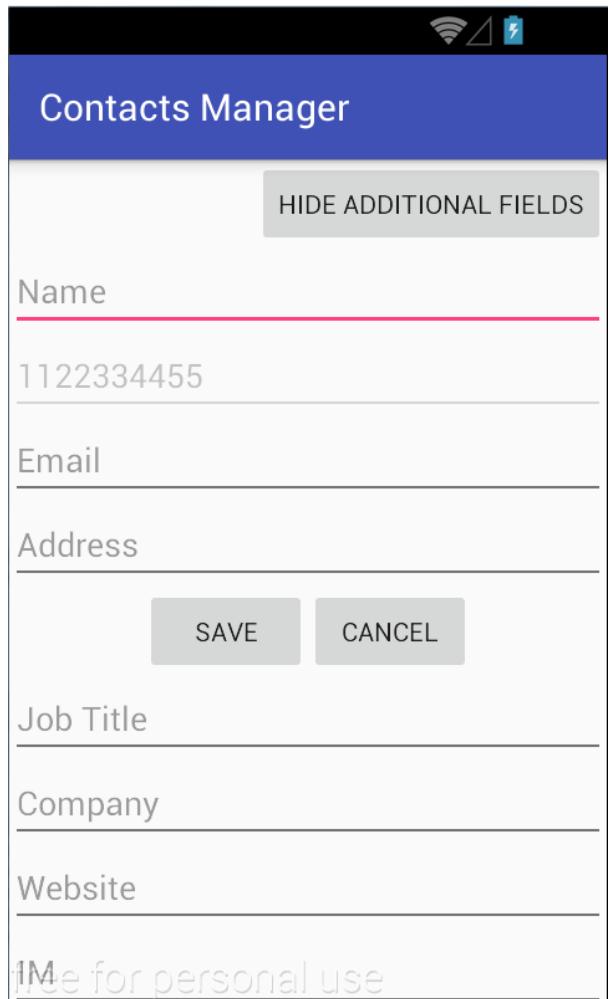
```
ConnectivityManager connectivityManager =
 (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo networkInfo = connectivityManager.getActiveNetworkInfo();
boolean isConnected = networkInfo.isConnectedOrConnecting();
boolean isMobile = (networkInfo.getType() == ConnectivityManager.TYPE_MOBILE);
```

## Activitate de Laborator

---

Se dorește implementarea unei aplicații Android, conținând o activitate care să ofere utilizatorilor funcționalitatea necesară pentru a stoca un număr de telefon în agenda de contacte, specificând pentru acesta mai multe informații.





**1.** În contul Github personal, să se creeze un depozit denumit 'Laborator04'. Acesta trebuie să conțină unui fișier README.md, un fișier .gitignore specific unei aplicații Android și un fișier LICENSE care să descrie condițiile pentru utilizarea aplicației.

The screenshot shows the GitHub profile of user `eim2017`. The profile picture is a red and white checkered pattern. The user has 4 repositories, 0 stars, 0 followers, and 0 following. There are three repository cards:

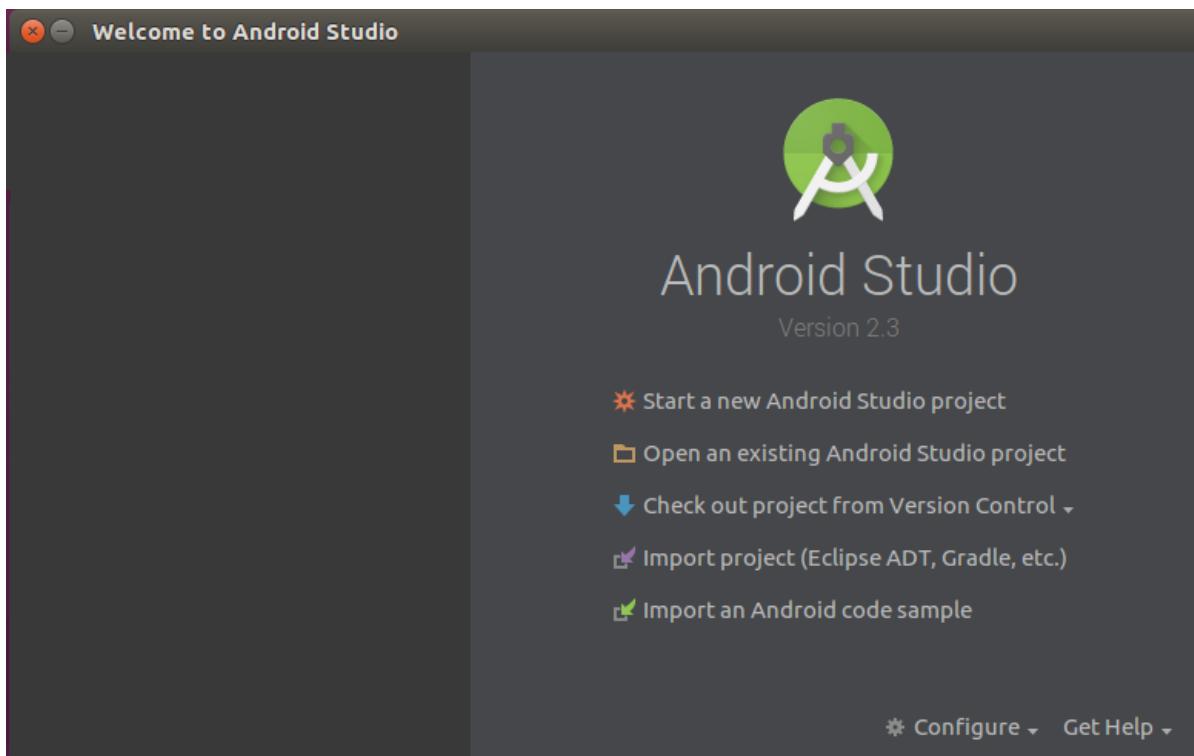
- Laborator01**: Updated 7 days ago, Java, 2 issues.
- Laborator02**: Updated 9 days ago, Java.
- Laborator03**: Updated 23 minutes ago, Java.

The screenshot shows the GitHub interface for creating a new repository. The owner is set to `eim2017` and the repository name is `Laborator04`. The repository is set to be public. The 'Initialize this repository with a README' checkbox is checked. The 'Create repository' button is at the bottom.

**2.** Să se cloneze într-un director de pe discul local conținutul depozitului la distanță astfel creat. În urma acestei operații, directorul `Laborator04` va trebui să se conțină fișierele `README.md`, `.gitignore` care indică tipurile de fișiere (extensiile) ignorate și `LICENSE`.

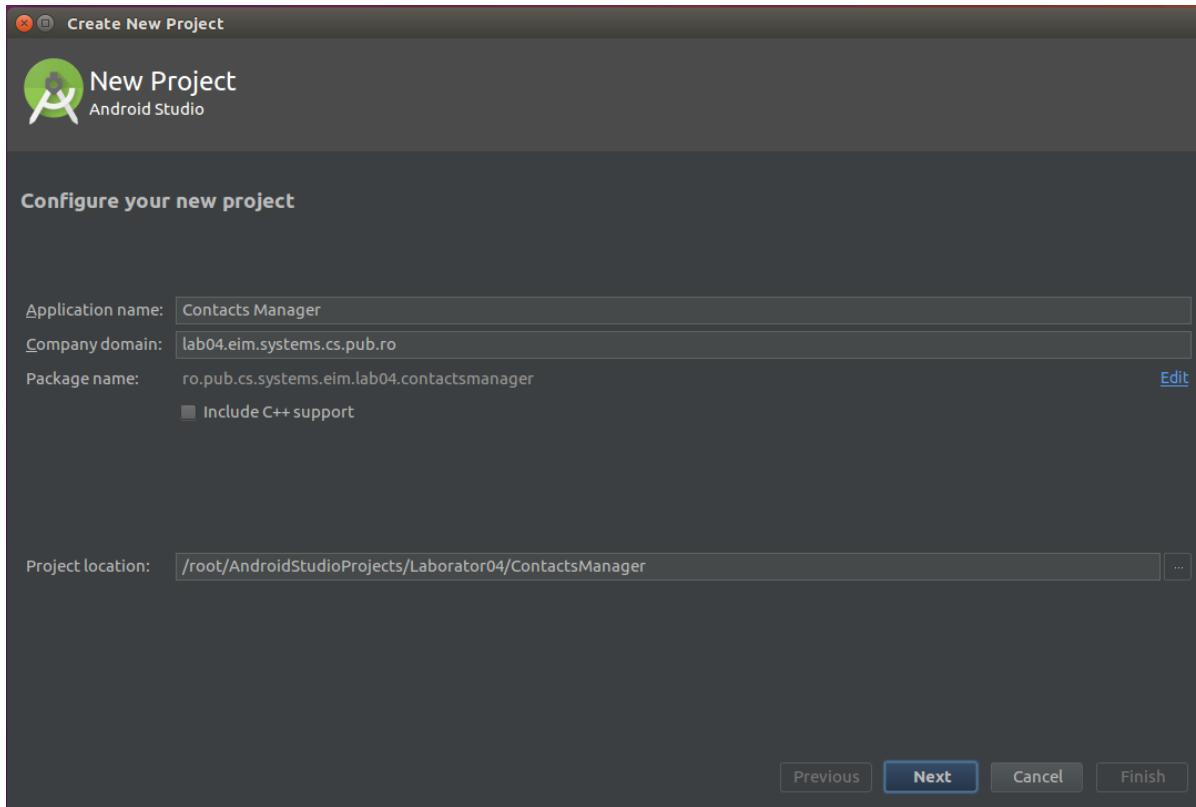
```
student@eim2017:~$ git clone https://www.github.com/perfectstudent/Laborator04
```

**3.** În directorul `Laborator04` de pe discul local, să se creeze un proiect Android Studio denumit `ContactsManager` (se selectează *Start a new Android Studio project*).

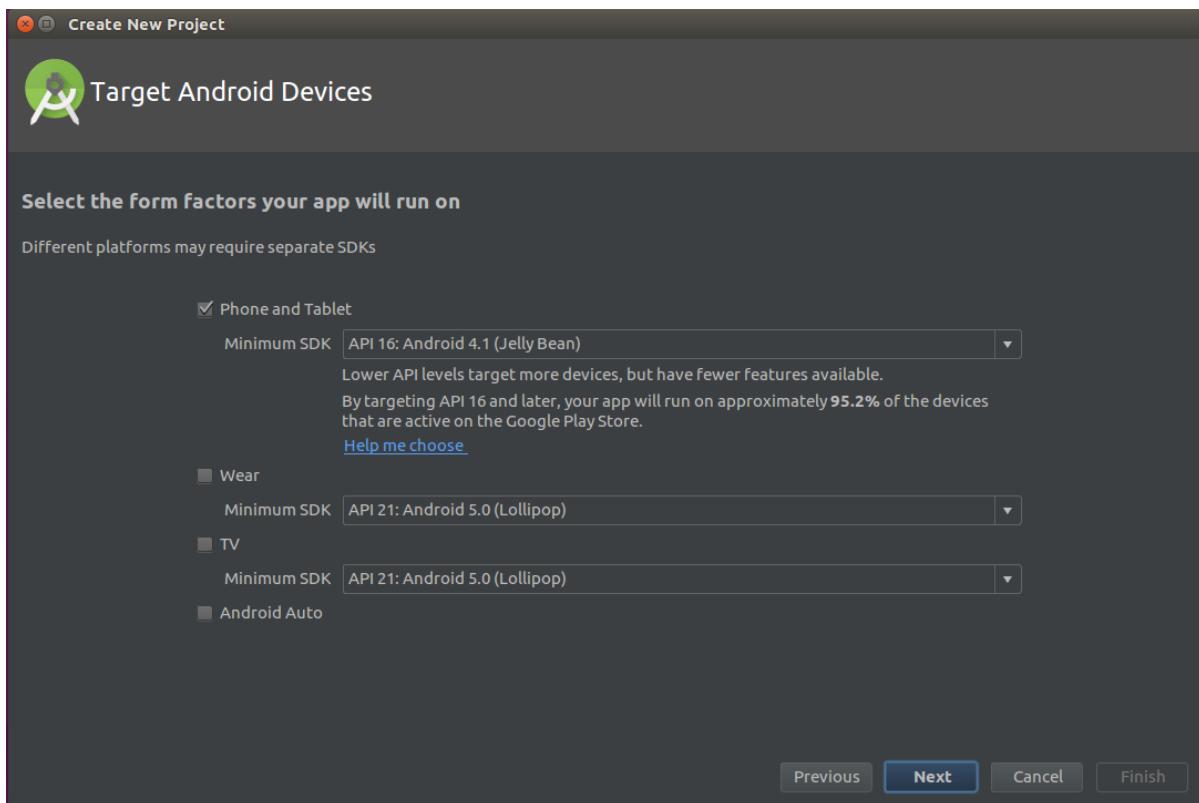


Se indică detaliile proiectului:

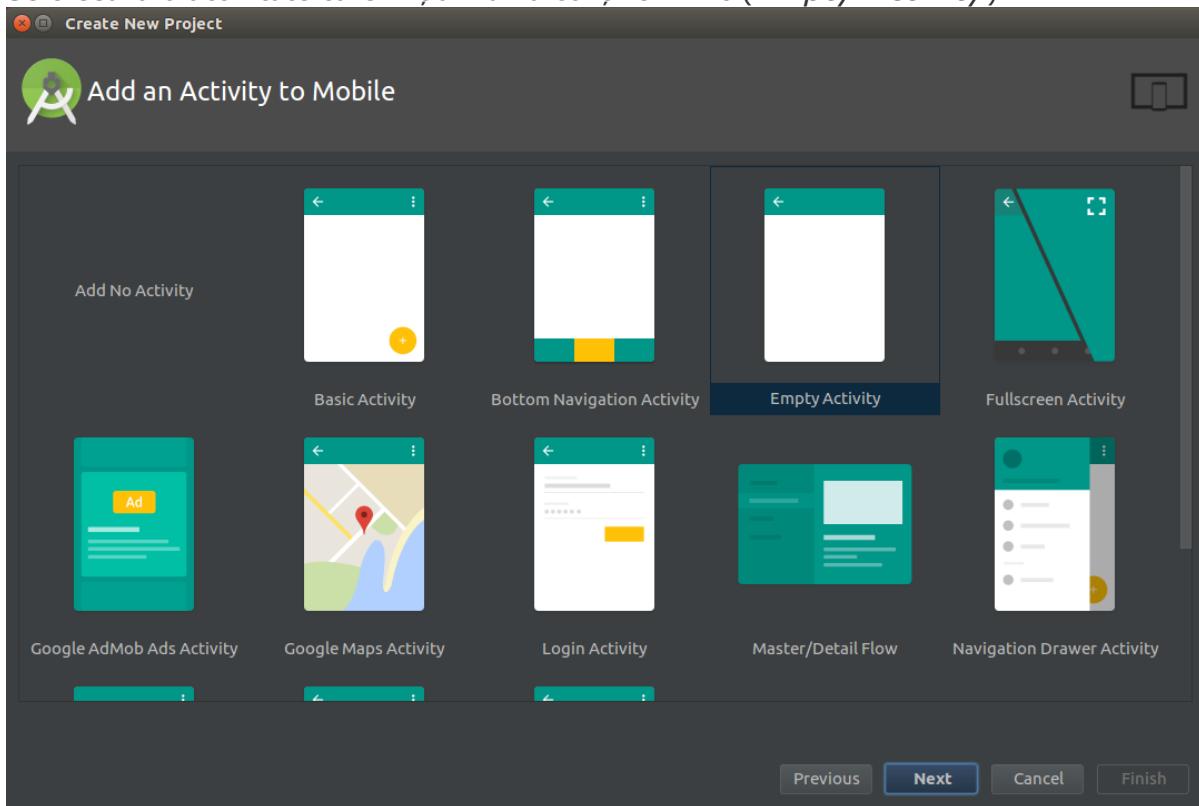
- **Application name** - *Contacts Manager*
- **Company domain** - *lab04.eim.systems.cs.pub.ro* (se va genera în mod automat **Package** **name** cu valoarea *ro.pub.cs.systems.eim.lab04.contactsmanager*)
- **Project location** - locația directorului de pe discul local unde a fost descărcat depozitul la distanță *Laborator04*



Se indică platforma pentru care se dezvoltă aplicația Android (se bifează doar *Phone and Tablet*), iar SDK-ul Android (minim) pentru care se garantează funcționarea este API 16 (Jelly Bean, 4.1).



Se creează o activitate care inițial nu va conține nimic (*Empty Activity*):

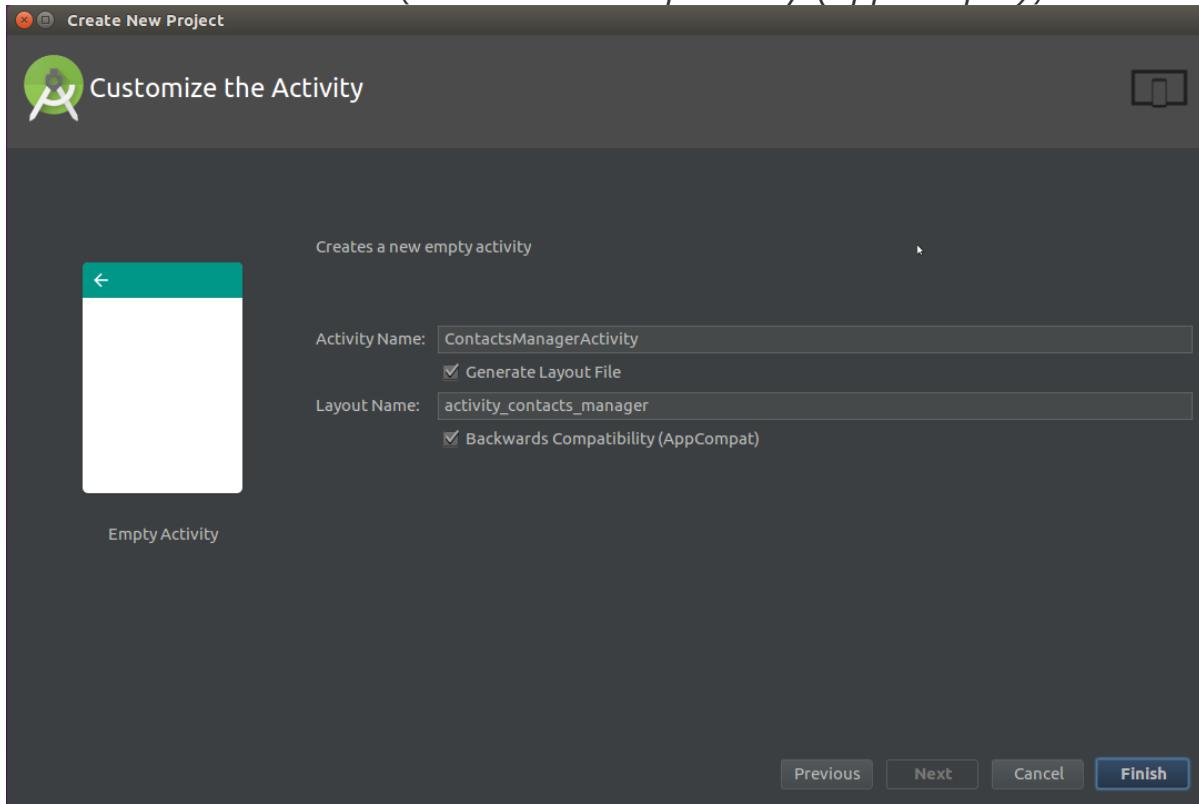


pentru care se precizează:

- **Activity Name** (denumirea activității) - ContactsManagerActivity;
- **Layout Name** (denumirea fișierului XML din res/layout în care va fi construită interfața grafică) - activity\_contacts\_manager.xml.

De asemenea:

- se bifează opțiunea de a se genera în mod automat fișierul XML care conține descrierea interfeței grafice (*Generate Layout File*);
- folosirea claselor din bibliotecile de suport care asigură posibilitatea de folosire a unor funcționalități din SDK-uri mai recente pe dispozitive mobile cu nivele de API mai vechi (*Backwards Compatibility (AppCompat)*).



4. În fișierul activity\_contacts\_manager din directorul res/layout să se construiască interfața grafică folosind:

- editorul vizual (*Graphical Layout*)
- editorul XML (*Text*)

Acesta va fi format din două containere după cum urmează:

- primul conține mai multe elemente dispuse vertical și ocupând pe lățime întregul spațiu avut la dispoziție;

- un buton (`Button`) având mesajul *Show Additional Fields* în cazul în care celălalt container nu este afișat, respectiv mesajul *Hide Additional Fields* în cazul în care celălalt container este afișat, determinând atașarea / detașarea acestuia la activitate;
- patru controale de tip câmpuri text (`EditText`) prin care se introduc:
  - numele;
  - numărul de telefon - acest câmp este dezactivat (are proprietatea `android:enabled="false"`), urmând ca valoarea să fie preluată din câmpul `extra` al unei intenții;
  - adresa electronică;
  - adresa poștală.
- cel de-al doilea container (care nu este vizibil inițial) conține patru controale de tip câmpuri text dispuse vertical și ocupând pe lățime întregul spațiu avut la dispoziție, prin care se introduc:
  - poziția ocupată;
  - denumirea companiei;
  - site-ul web;
  - identificatorul pentru mesagerie instantanee.

Fiecare container poate fi inclus într-un mecanism de dispunere a conținutului de tip `LinearLayout` cu dispunere verticală. Acestea vor fi incluse într-un container de tip `ScrollView`, pentru a preîntâmpina situația în care interfața grafică nu poate fi afișată pe ecranul dispozitivului mobil.

Să se implementeaze interacțiunea cu utilizatorul a aplicației.

- în metoda `onCreate()` a activității se obțin referințe către butoanele *Show Additional Details / Hide Additional Details*, respectiv *Save* și *Cancel* prin intermediul metodei `findViewById(R.id....)`;
- se implementează o clasă ascultător pentru butoane, care implementează `View.OnClickListener` și implementează metoda `onClick(View v)`; în funcție de id-ul butonului care este transmis argument metodei, sunt realizate următoarele acțiuni:
  - butonul *Show Additional Details / Hide Additional Details* - afișează / ascunde al doilea container în funcție de starea curentă , modificând corespunzător textul afișat pe buton. (Hint: atributul `visibility` al containerului, respectiv metoda `setVisibility()` a clasei Java împreună cu constantele `View.VISIBLE`, `View.GONE`).

- butonul **Save** - lansează în execuție aplicația Android nativă pentru stocarea unui contact în agenda telefonică, după ce în prealabil au fost preluate informațiile din controalele grafice:

```

■ Intent intent = new Intent (ContactsContract.Intents.Insert.ACTION);
■ intent.setType (ContactsContract.RawContacts.CONTENT_TYPE);
■ if (name != null) {
■ intent.putExtra (ContactsContract.Intents.Insert.NAME, name);
■ }
■ if (phone != null) {
■ intent.putExtra (ContactsContract.Intents.Insert.PHONE, phone);
■ }
■ if (email != null) {
■ intent.putExtra (ContactsContract.Intents.Insert.EMAIL, email);
■ }
■ if (address != null) {
■ intent.putExtra (ContactsContract.Intents.Insert.POSTAL,
■ address);
■ }
■ if (jobTitle != null) {
■ intent.putExtra (ContactsContract.Intents.Insert.JOB_TITLE,
■ jobTitle);
■ }
■ if (company != null) {
■ intent.putExtra (ContactsContract.Intents.Insert.COMPANY,
■ company);
■ }
■ ArrayList<ContentValues> contactData = new
■ ArrayList<ContentValues> ();
■ if (website != null) {
■ ContentValues websiteRow = new ContentValues ();
■ websiteRow.put (ContactsContract.Data.MIMETYPE,
■ ContactsContract.CommonDataKinds.Website.CONTENT_ITEM_TYPE);
■ websiteRow.put (ContactsContract.CommonDataKinds.Website.URL,
■ website);
■ }
```

```

 contactData.add(websiteRow);
}

if (im != null) {
 ContentValues imRow = new ContentValues();
 imRow.put(ContactsContract.Data.MIMETYPE,
 ContactsContract.CommonDataKinds.Im.CONTENT_ITEM_TYPE);
 imRow.put(ContactsContract.CommonDataKinds.Im.DATA, im);
 contactData.add(imRow);
}

intent.putExtraParcelableArrayListExtra(ContactsContract.Intents.Insert.DATA, contactData);

startActivity(intent);

```

Intenția pentru realizarea acestei operații are asociată acțiunea `ContactsContract.Intents.Insert.ACTION`

tipul `ContactsContract.RawContacts.CONTENT_TYPE`. Informațiile care se doresc să fie completate sunt atașate în câmpul extra al acesteia, având cheile:

- ✓ `ContactsContract.Intents.Insert.NAME`;
- ✓ `ContactsContract.Intents.Insert.PHONE`;
- ✓ `ContactsContract.Intents.Insert.EMAIL`;
- ✓ `ContactsContract.Intents.Insert.POSTAL`;
- ✓ `ContactsContract.Intents.Insert.JOB_TITLE`;
- ✓ `ContactsContract.Intents.Insert.COMPANY`;

Pentru site-ul web și identificatorul de mesagerie instantanee, se folosește un tablou de elemente `ContentValues` în care se specifică înregistrările de tipul `CommonDataKinds.Website.URL`, respectiv `CommonDataKinds.Im.DATA`;

Pentru a putea gestiona agenda telefonică, este necesar ca în fișierul `AndroidManifest.xml` să fie specificate următoarele permisiuni:

```

<uses-permission
 android:name="android.permission.READ_CONTACTS" />
<uses-permission
 android:name="android.permission.WRITE_CONTACTS" />

```

- butonul *Cancel* - termină aplicația Android:

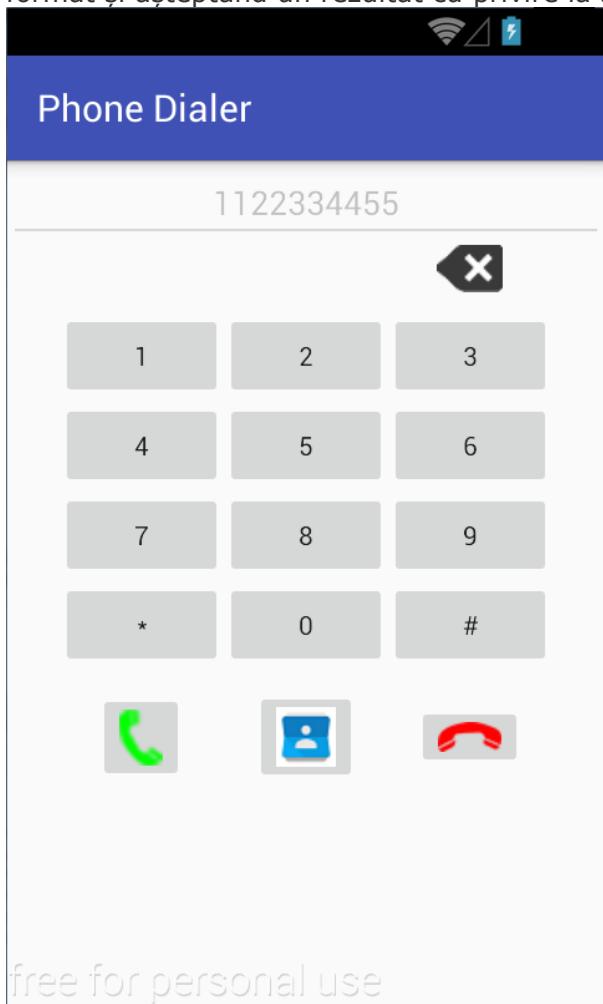
```

finish();

```

- se înregistrează o instanță a clasei ascultător ca mecanism de tratare a evenimentelor de tip accesare a butoanelor din cadrul interfeței grafice, prin apelul metodei `setOnClickListener()`.

**5.** Să se modifice aplicația Android Phone Dialer astfel încât să conțină un buton suplimentar prin care este invocată aplicația *Contacts Manager* căreia îi transmite numărul de telefon format și așteptând un rezultat cu privire la stocarea contactului în agenda telefonică.





Ca imagine pentru butonul care invocă aplicația *Contacts Manager* se poate folosi [această resursă](#).

Metoda de tratare a evenimentului de tip accesare a butonului de stocare a numărului de telefon în agenda telefonică invocă o intenție asociată aplicației *Contacts Manager*, transmițând și numărul de telefon în câmpul extra asociat acesteia, identificabil prin intermediul unei chei.

```
String phoneNumber = phoneNumberEditText.getText().toString();
if (phoneNumber.length() > 0) {
 Intent intent = new Intent("ro.pub.cs.systems.eim.lab04.contactsmanager.intent.action.ContactsManagerActivity");

 intent.putExtra("ro.pub.cs.systems.eim.lab04.contactsmanager.PHONE_NUMBER_KEY",
 phoneNumber);
 startActivityForResult(intent, Constants.CONTRACTS_MANAGER_REQUEST_CODE);
} else {
 Toast.makeText(getApplicationContext(),
 getResources().getString(R.string.phone_error), Toast.LENGTH_LONG).show();
}
```

Definiți în prealabil constanta `CONTACTS_MANAGER_REQUEST_CODE` și valoarea string `phone_error` în fișierele corespunzătoare din folderul de resurse statice `res`.

**6.** Să se modifice aplicația Android *Contacts Manager* astfel încât să poată fi lansată în execuție doar din contextul altei activități, prin intermediul unei intenții care conține în câmpul extra un număr de telefon, identificabil prin cheia `ro.pub.cs.systems.eim.lab04.contactsmanager.PHONE_NUMBER_KEY`, acesta fiind plasat în câmpul text needitabil corespunzător. Totodată, va transmite înapoi rezultatul operației de stocare (`Activity.RESULT_OK` sau `Activity.RESULT_CANCELED`).

- în fișierul `AndroidManifest.xml` se modifică filtrul de intenții (acțiunea și categoria), astfel încât activitatea să poată fi rulată doar prin intermediul unei intenții

```
AndroidManifest.xml

<manifest ...>
 <application ...>
 <activity
 android:name=".graphicuserinterface.ContactsManagerActivity"
 android:label="@string/app_name" >
 <intent-filter>
 <action
 android:name="ro.pub.cs.systems.eim.lab04.contactsmanager.intent.action.ContactsManagerActivity" />
 <category android:name="android.intent.category.DEFAULT" />
 </intent-filter>
 </activity>
 </application>
</manifest>
```

- în metoda `onCreate()` a activității aplicației `ContactsManager` este verificată intenția cu care este pornită, și în cazul în care aceasta nu este nulă, este preluată informația din secțiunea extra, identificată prin cheia `ro.pub.cs.systems.eim.lab04.contactsmanager.PHONE_NUMBER_KEY`, conținutul său fiind plasat în cadrul câmpului text corespunzător:

```
 Intent intent = getIntent();
 if (intent != null) {
 String phone = intent.getStringExtra("ro.pub.cs.systems.eim.lab04.contactsmanager.PHONE_NUMBER_KEY");
 if (phone != null) {
 phoneEditText.setText(phone);
 } else {
```

```

 □ Toast.makeText(this,
 getResources().getString(R.string.phone_error),
 Toast.LENGTH_LONG).show();
 □ }
}

```

- pe metodele de tratare a evenimentelor de accesare a butoanelor:
  - *Save* - este lansată în execuție aplicația nativă pentru gestiunea agendei telefonice, folosind un cod de cerere prin intermediul căruia se va verifica rezultatul furnizat:

```

startActivityForResult(intent,
Constants.CONTACTS_MANAGER_REQUEST_CODE);

```

- *Cancel* - se transmite înapoi rezultatul

```

setResult(Activity.RESULT_CANCELED, new Intent());

```

- În metoda `onActivityResult()` asociată activității aplicației *ContactsManager*, în momentul în care s-a părăsit aplicația nativă pentru gestiunea agendei telefonice, se verifică codul de cerere și se transmite înapoi un rezultat:

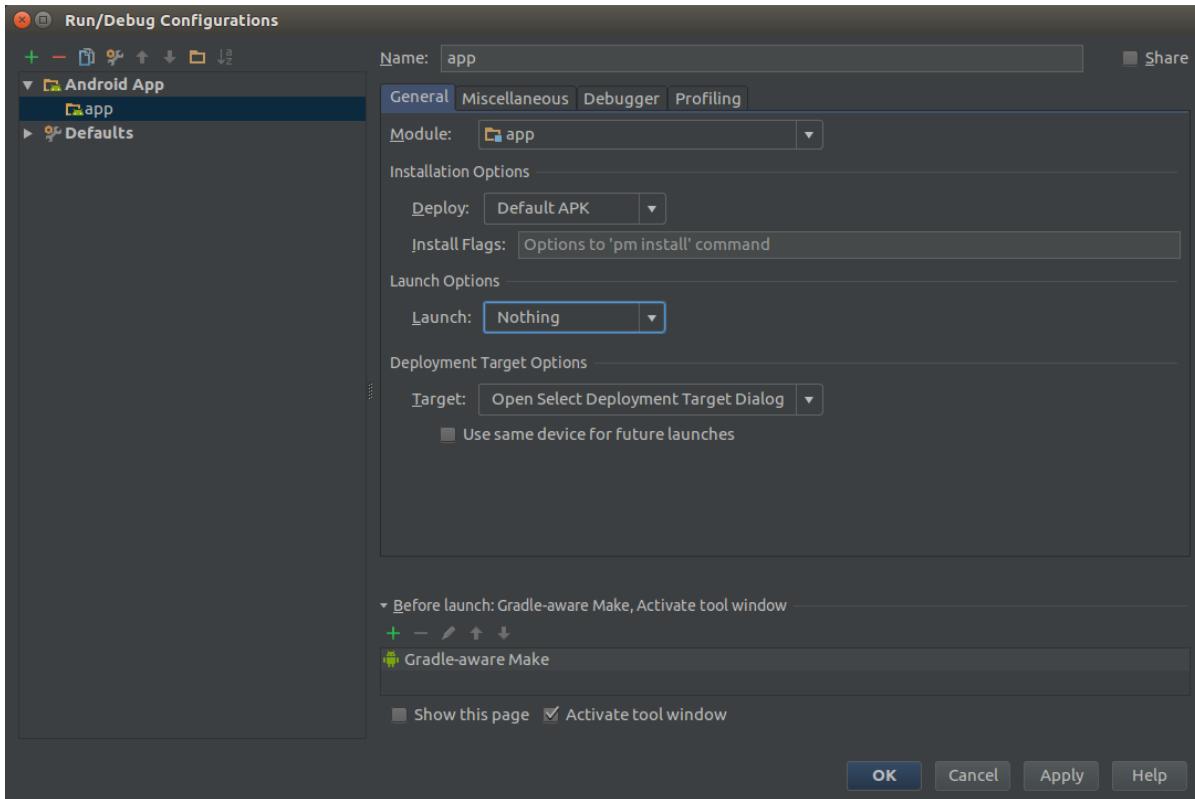
```

public void onActivityResult(int requestCode, int resultCode, Intent intent) {
 switch(requestCode) {
 case Constants.CONTACTS_MANAGER_REQUEST_CODE:
 setResult(resultCode, new Intent());
 finish();
 break;
 }
}

```

Datorită faptului că aplicația Android *Contacts Manager* nu dispune de o activitate principală (implicită), aceasta nu va mai putea fi lansată în execuție folosind mediul integrat de dezvoltare Android Studio. Pentru ca aceasta să fie doar instalată pe dispozitivul mobil, se accesează meniul *Run → Edit Configurations...*, iar în secțiunea *Launch Options* de pe panoul *General*, se selectează opțiunea *Launch: Nothing*.

▶ Run 'app'	Shift+F10
⚡ Apply Changes: No active 'app' launch	
🛠 Debug 'app'	Shift+F9
📊 Run 'app' with Coverage	
▶ Run...	Alt+Shift+F10
🛠 Debug...	Alt+Shift+F9
🔴 Record Espresso Test	
📝 Edit Configurations...	
📄 Import Test Results	▶
⚡ Apply Changes: No active 'app' launch	Ctrl+F10
⏹ Stop	Ctrl+F2
⌚ Show Running List	
⌚ Restart Activity	
⬇ Step Over	F8
⬇ Force Step Over	Alt+Shift+F8
➤ Step Into	F7
➤ Force Step Into	Alt+Shift+F7
➤ Smart Step Into	Shift+F7
➤ Step Out	Shift+F8
➤ Run to Cursor	Alt+F9
➤ Force Run to Cursor	Ctrl+Alt+9
✖ Drop Frame	
⏸ Pause Program	
▶ Resume Program	F9
⏹ Evaluate Expression...	Alt+F8
Quick Evaluate Expression	Ctrl+Alt+8
⠇ Show Execution Point	Alt+F10
Toggle Line Breakpoint	Ctrl+F8
Toggle Method Breakpoint	
Toggle Temporary Line Breakpoint	Ctrl+Alt+Shift+F8
Toggle Breakpoint Enabled	
⌚ View Breakpoints...	Ctrl+Shift+F8
⌚ Get thread dump	



- Să se verifice în emulator/telefon faptul că ContactsManager nu este o aplicație care poate fi lansată din launcher. Dacă aveți utilitarul MyAndroidTools, acesta va lista aplicația cu Activitatea definită.

**7. Să se încarce modificările realizate în cadrul depozitului 'Laborator04' de pe contul Github personal, folosind un mesaj sugestiv.**

```
student@eim2017:~/Laborator04$ git add *

student@eim2017:~/Laborator04$ git commit -m "implemented tasks for laboratory
04"

student@eim2017:~/Laborator04$ git push origin master
```

Laborator 5

## Structura unei Aplicații (III)

### Servicii

În Android, clasa [android.app.Service](#) este utilizată pentru componente a căror funcționalitate implică procesări complexe, de lungă durată, necesitând acces la anumite resurse, fără a fi necesar să pună la dispoziție o interfață grafică sau un mecanism de interacțiune cu

utilizatorul. Prin intermediul unui serviciu, se asigură faptul că aplicația Android continuă să se găsească în execuție, chiar și atunci când interfața grafică a acesteia nu este vizibilă. Întrucât prioritatea unui serviciu este mai mare decât a unei activități inactive, este mai puțin probabil ca acestea să fie distruse atunci când trebuie eliberate resursele sistemului de operare. De altfel, un serviciu poate fi configurat să fie repornit imediat ce este posibil sau chiar pentru a-i se asocia o prioritate echivalentă cu a unei activități active (dacă distrugerea serviciului are un impact la nivelul interfeței grafice).

Astfel, un serviciu nu trece prin evenimentele ce fac parte din ciclul de viață al unei activități. Totuși, un serviciu poate fi controlat (pornit, opus) din contextul altor componente ale unei aplicații Android (activități, ascultători de intenții cu difuzare, alte servicii).

**Serviciul este rulat pe firul de execuție principal al aplicației Android.** De aceea, în situația în care operațiile pe care le realizează poate influența experiența utilizatorului, acestea trebuie transferate pe alte fire de execuție din fundal (folosind clasele HandlerThread sau AsyncTask).

Un serviciu continuă să se ruleze chiar și în situația în care componenta aplicației Android care l-a invocat prin intermediul unei intenții devine inactivă (nu mai este vizibilă) sau chiar este distrusă. Acest comportament este adekvat în situația în care serviciul realizează operații de intrare/ieșire intensive, interacționează cu diverse servicii accesibile prin intermediul rețelei sau cu furnizori de conținut.

Opțiunea de a utiliza un serviciu în Android trebuie luată în considerare numai în situația în care este necesar ca o anumită funcționalitate să fie realizată independent de starea componentei care a invocat-o. Dacă este necesar ca o operație să fie executată în tandem cu o anumită componentă (doar în perioada în care aceasta este vizibilă, spre exemplu), dar fără a avea un impact asupra experienței utilizatorului (fără a influența timpul de răspuns al aplicației Android), se va utiliza un fir de execuție dedicat, a cărui stare va fi gestionată pe metodele de callback ce guvernează ciclul de viață al componentei respective.

## Tipuri de Servicii

În programarea Android, există două tipuri de servicii:

1. servicii **pornite explicit** (eng. started), lansate în execuție prin intermediul metodei [startService\(\)](#), invocată din contextul unei componente (activitate, serviciu, ascultător al unor mesaje cu difuzare); astfel de servicii nu furnizează un rezultat (către componentă care l-a apelat), fiind utilizate pentru a realiza o anumită operație la finalizarea căreia, de regulă, sunt terminate (apelându-se, în rutina lor, metoda [stopSelf\(\)](#)); există posibilitatea ca o altă componentă să îl opreasă explicit prin metoda [stopService\(\)](#); **un serviciu de tip started continuă să ruleze chiar și în situația în care componenta care l-a invocat nu mai există**;
2. servicii **atașate unei (unor) componente** (eng. bounded) sunt lansate în execuție prin intermediul metodei [bindService\(\)](#), invocată din contextul unei componente (activitate, serviciu, ascultător al unor mesaje cu difuzare) care interacționează cu acesta prin funcționalități pe care le expune; la un moment

dat, pot exista mai multe componente atașate aceluiași serviciu; detașarea unei componente la un serviciu se realizează prin intermediul metodei `unbindService()`; **atunci când nu mai există nici o componentă asociată serviciului, acesta se consideră încheiat.**

Un serviciu poate avea, în același timp, ambele tipuri (started și bounded). Acesta va putea rula o perioadă de timp nedefinită (până în momentul în care procesarea pe care o realizează este încheiată), permitând totodată componentelor unei aplicații Android să interacționeze cu el prin intermediul unor metode pe care le expune.

Indiferent de modul în care este folosit un serviciu, invocarea sa este realizată, ca pentru orice componentă Android, prin intermediul unei intenții. În cazul serviciilor, se preferă să se utilizeze intenții explicite, în detrimentul intențiilor implicate, din motive de securitate.

Categoria în care se încadrează un anumit serviciu este determinată de metodele pe care acesta le implementează:

1. pentru un serviciu de tip started, va fi implementată metoda `onStartCommand()`, apelată în mod automat în momentul în care serviciul este pornit prin intermediul metodei `startService()`;
2. pentru un serviciu de tip bounded, vor fi implementate metodele:
  - `onBind()`, apelată în mod automat în momentul în care o componentă este atașată serviciului, prin intermediul metodei `bindService()`;
  - `onUnbind()`, apelată în mod automat în momentul în care o componentă este detașată de la serviciu, prin intermediul metodei `unbindService()`.

## Gestiunea unui Serviciu

Pentru a putea fi utilizat, orice serviciu trebuie să fie declarat în cadrul fișierului `AndroidManifest.xml`, prin intermediul etichetei `<service>` în cadrul elementului `<application>`. Eventual, se poate indica o permisiune necesară pentru pornirea și oprirea serviciului, astfel încât aceste operații să poată fi realizate numai de anumite aplicații Android.

```
AndroidManifest.xml
<manifest ...>
 <application ...>
 <service
 android:name="ro.pub.cs.systems.eim.lab05.SomeService"
 android:enabled="true"
 android:exported="true"

 android:permission="ro.pub.cs.systems.eim.lab05.SOME_SERVICE_PERMISSION" />
 </application>
 </manifest>
```

Atributul `android:name` este singurul **obligatoriu** în cadrul elementului `<service>`, desemnând clasa care gestionează operațiile specifice serviciului respectiv. Din momentul în

care aplicația Android este publicată, această valoare nu mai poate fi modificată, întrucât poate avea un efect asupra componentelor care utilizează acest serviciu prin intermediul unei intenții explicite folosită la pornirea serviciului sau la asocierea componentei cu serviciul respectiv.

Pentru a asigura securitatea aplicației Android, se recomandă să se folosească numai **intenții explicite** pentru pornirea unui serviciu. Nu este recomandat să se utilizeze filtre de intenții, astfel încât acestea nu ar trebui să se regăsească în fișierul `AndroidManifest.xml`. Totuși, în situația în care acest lucru este absolut necesar, ar trebui să se indice intenției măcar pachetul în care se regăsește serviciul respectiv.

Atributul `android:enabled` indică dacă serviciul poate fi instantiat de către sistemul de operare.

Atributul `android:exported` specifică posibilitatea ca alte componente (apartenând altor aplicații) să poată interacționa cu serviciul. În situația în care serviciul nu conține filtre de intenții, acesta poate fi invocat numai prin precizarea explicită a numelui clasei care îl gestionează (calificată complet), valoarea sa fiind `false`, fiind destinat invocării din contextul unor componente aparținând aceleiași aplicații Android ca și el. În cazul în care serviciul definește cel puțin un filtru de intenții, valoarea sa este `true`, fiind destinat invocării din contextul altor aplicații Android.

Atributul `android:permission` precizează denumirea unei permișuni pe care entitatea trebuie să o dețină pentru a putea lansa în execuție un serviciu sau pentru a-i se putea asocia. Aceasta trebuie indicată în cadrul intențiilor transmise ca argumente metodelor utilizate pentru a invoca serviciul respectiv (`startService()`, respectiv `bindService()`), altfel intenția respectivă nu va fi livrată către serviciu.

Alte atrbute ale elementului `<service>` sunt: `android:icon`, `android:isolatedProcess`, `android:label`, `android:process`.

Un serviciu este o clasă derivată din `android.app.Service` (sau din subclasele sale), implementând o serie de metode din ciclul de viață al serviciului:

- `onCreate()` - realizând operațiile (unice) asociate construirii serviciului respectiv (legate de configurarea sa); această metodă este invocată doar atunci când este realizată o nouă instanță a serviciului; în situația în care serviciul este invocat, însă acesta există deja în memorie, metoda nu va mai fi apelată;
- `onStartCommand()` - apelată în mod automat, **numai pentru serviciile de tip started**, în momentul în care serviciul este invocat printr-un apel la metodei `startService()`; serviciul va fi executat imediat după această metodă; **este responsabilitatea programatorului să opreasă serviciul printr-un apel al uneia dintre metodele `stopSelf()`, respectiv `stopService()`**, altfel serviciul va rula pentru o perioadă de timp nedefinită; nu este necesar ca metoda să fie implementată, dacă serviciul este de tip bounded;

- onBind() - apelată în mod automat, **numai pentru serviciile de tip bounded**, în momentul în care o componentă a fost atașată unui serviciu printr-un apel al metodei `bindService()`; implementarea acestei metode trebuie să furnizeze un obiect ce implementează interfața `IBinder`, prin intermediul căruia serviciul să poată interacționa cu componenta care l-a invocat, punând la dispoziție o anumită funcționalitate, descrisă de metode publice; **toate tipurile de serviciu trebuie să implementeze această metodă**, însă pentru serviciile de tip started, valoarea întoarsă va fi `null`;
- onUnbind() - apelată în mod automat, **numai pentru serviciile de tip bounded**, în momentul în care toate componentele au fost detașate unui serviciu;
- onRebind() - apelată în mod automat, **numai pentru serviciile de tip bounded**, în momentul în care o componentă a fost atașată unui serviciu după ce acesta a fost notificat anterior că toate componentele care îi erau asociate au fost detașate (s-a apelat metoda `onUnbind()`); o astfel de metodă va fi invocată numai dacă rezultatul întors de metoda `onUnbind()` este `true`;
- onDestroy() - realizând operațiile asociate distrugerii serviciului respectiv, atunci când acesta nu mai este utilizat (a fost oprit sau sistemul de operare Android solicită memoria pe care o folosește); în cadrul acestei metode sunt eliberate resursele utilizate de serviciu (fire de execuție, obiecte, fluxuri de intrare / ieșire).

SomeStartedService.java

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

public class SomeStartedService
extends Service {

 private int startMode;

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

 @Override
 public int onStartCommand(Intent intent,
 int flags,
 int startId) {
```

SomeBoundedService.java

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

public class SomeBoundedService extends
Service {

 private IBinder iBinder;
 private boolean allowRebind;

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

 @Override
 public IBinder onBind(Intent intent) {
 // ...
 return iBinder;
 }
```

```
// ...
 return startMode;
}

@Override
public IBinder onBind(Intent intent)
{
 // ...
 return null;
}

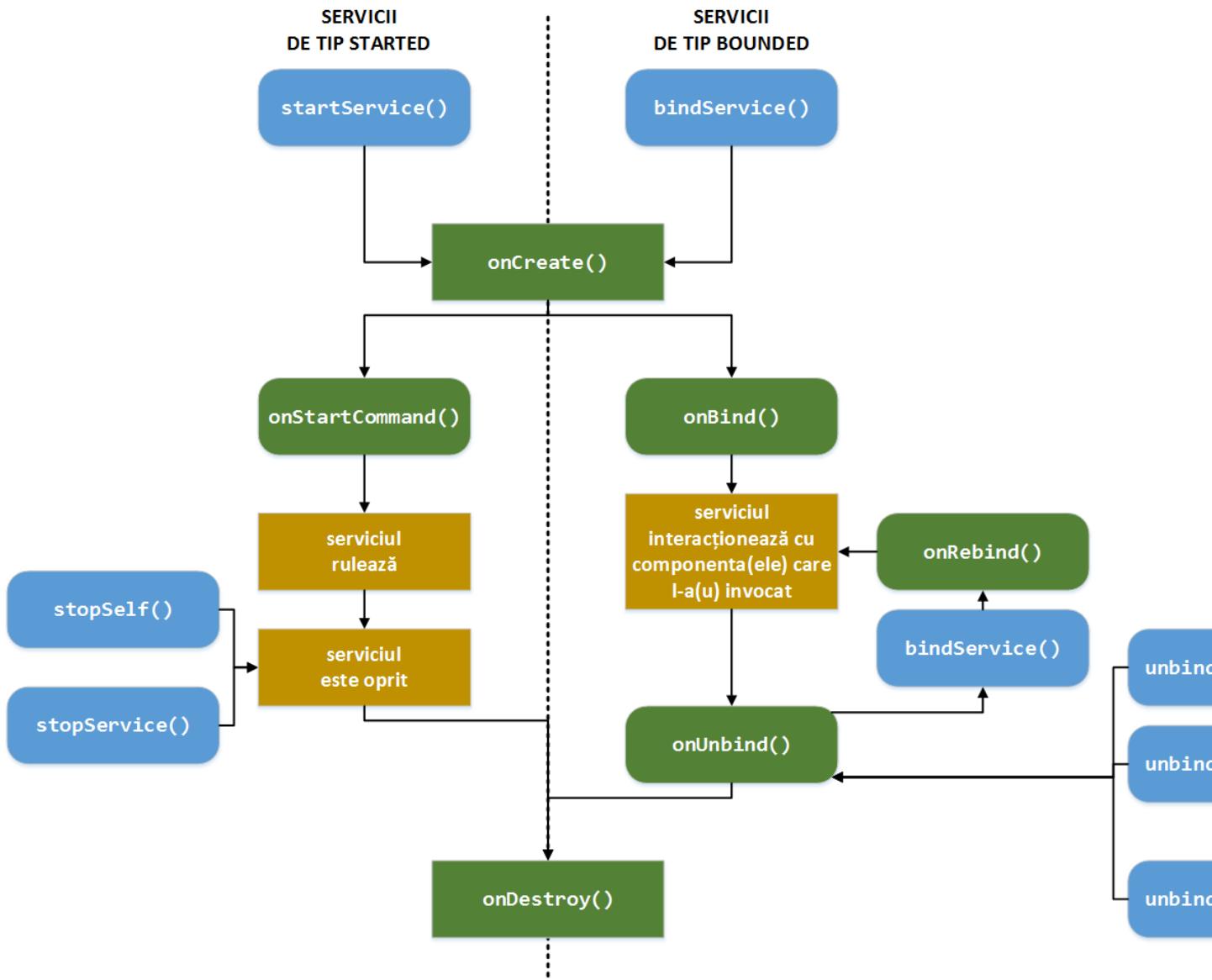
@Override
public void onDestroy() {
 super.onDestroy();
 // ...
}
```

```
@Override
public boolean onUnbind(Intent intent)
{
 // ...
 return allowRebind;
}

@Override
public void onRebind(Intent intent)
{
 // ...
}

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

În cazul metodelor care guvernează ciclul de viață al unui serviciu, NU este obligatoriu să se apeleze metodele din clasa părinte (nu se va genera nici o excepție în această situație).



Se observă faptul că ciclul de viață al unui serviciu are loc între metodele `onCreate()` și `onDestroy()`. De cele mai multe ori, este necesar ca un serviciu să folosească unul sau mai multe fire de execuție (dedicate) astfel încât să nu influențeze responsivitatea aplicației Android. Într-o astfel de situație, firul de execuție va fi pornit pe metoda `onCreate()` și va fi oprit pe metoda `onDestroy()`. De asemenea, în cadrul metodei `onCreate()` au loc diferite operații de configurare (initializări), în timp ce în cadrul metodei `onDestroy()` realizează eliberarea resurselor folosite. Metodele `onCreate()` și `onDestroy()` sunt invocate pentru toate tipurile de servicii, atât pentru cele de tip started, cât și pentru cele de tip bounded.

În cazul unui serviciu de tip started, perioada activă din ciclul de viață este cuprinsă între apelul metodei `onStartCommand()` (apelată în mod automat atunci când o componentă apelează metoda `startService()`, primind ca argument obiectul de tip `Intent` care a fost folosit la invocarea sa) și apelul metodei `onDestroy()` (apelat atunci când serviciul este oprit, prin intermediul uneia dintre metodele `stopSelf()` sau `stopService()`).

În cazul unui serviciu de tip bounded, perioada activă din ciclul de viață este cuprinsă între apelul metodei `onBind()` (apelată în mod automat atunci când o componentă apelează metoda `bindService()`, primind ca argument obiectul de tip `Intent` care a fost folosit la invocarea sa) și apelul metodei `onUnbind()` (apelată în mod automat atunci când toate componentele asociate serviciului au apelat metoda `unbindService()`).

Întrucât un serviciu poate fi simultan atât de tip started cât și de tip bounded, nu este exclus ca ulterior metodei `onStartCommand()` să fie apelată și metoda `onBind()`. În această situație serviciul va fi activ până când nu va mai deține nici o componentă asociată. Apelurile metodelor `stopSelf()` sau `stopService()` nu vor avea aşadar efect atâtă timp cât mai există componente conectate la serviciu.

## Servicii de tip Started

În momentul în care este pornit (prin un apel al metodei `startService()` din cadrul altelor componente), un serviciu apelează în mod automat metoda `onStartCommand()`. În cadrul acestei metode trebuie realizată procesarea pe care o presupune serviciul respectiv. Pentru serviciile de tip started, este necesar ca **serviciul să fie operat explicit**, fie din propriul său context, prin un apel al metodei `stopSelf()`, fie de către o componentă (aceeași sau alta decât cea care l-a invocat), prin intermediul metodei `stopService()`.

Metoda `onStartCommand()` este apelată în mod automat în momentul în care serviciul este pornit prin intermediul metodei `startService()` și primește ca parametrii:

- intenția care a invocat serviciul (argumentul transmis la apelul metodei `startService()`); **acesta poate fi null** în situația în care serviciul este repornit după ce procesul din care făcea parte a fost distrus;
- anumite valori prin care poate fi semnalat modul în care a fost pornit:
  - START FLAG REDELIVERY - serviciul a fost repornit ca urmare a distrugerii sale de către sistemul de operare înainte de se fi terminat corespunzător;
  - START FLAG RETRY - serviciul a fost repornit după o execuție anormală;
- un identificator unic prin care se poate face distincția între apeluri diferite ale aceluiași serviciu.

Având în vedere faptul că această metodă poate fi apelată de mai multe ori pe parcursul ciclului de viață al unui serviciu, tot aici trebuie implementat și comportamentul în cazul în care acesta este repornit.

Comportamentul serviciului în situația în care este repornit poate fi controlată prin intermediul valorii întregi care este furnizată ca valoare întoarsă:

- Service.START\_STICKY - mecanism standard, folosit de serviciile care își gestionează propriile stări și care sunt pornite și opuse în funcție de necesități (prin intermediul metodelor `startService()` și `stopService()`); prin aceasta, se indică faptul că metoda `onStartCommand()` va fi invocată

de fiecare dată când serviciul este (re)pornit după ce a fost distrus de sistemul de operare Android (situație în care parametrul de tip `Intent` va avea valoarea `null`, dacă între timp serviciul nu a mai fost invocat);

- `Service.START NOT STICKY` - mecanism folosit de serviciile utilizate pentru a procesa anumite comenzi, care se opresc singure (printr-un apel al metodei `stopSelf()`) atunci când operațiile pe care trebuiau să le realizeze s-au terminat; serviciul este (re)pornit după ce a fost distrus de sistemul de operare Android numai în situația în care între timp au mai fost realizateapeluri ale metodei `startService()`, altfel nu este recreat; un astfel de comportament este adekvat pentru procese care sunt realizate periodic;
- `Service.START REDELIVER INTENT` - mecanism utilizat atunci când se dorește să se asigure faptul că procesările asociate serviciului sunt terminate; în situația în care serviciul a fost distrus de sistemul de operare Android, este (re)pornit apelând metoda `onStartCommand()` folosind ca parametru intenția originală, a cărei procesare nu a fost terminată corespunzător.

```
@Override

public int onStartCommand(Intent intent, int flags, int startId) {
 processInBackground(intent, startId);
 return startMode;
}
```

În toate aceste cazuri, oprirea serviciului trebuie realizată explicit, prin apelul metodelor:

- `stopService()` din contextul componentei care l-a pornit;
- `stopSelf()` din contextul serviciului.

## Pornirea unui Serviciu

Un serviciu este pornit printr-un apel al metodei `startService()`. Aceasta primește ca parametru un obiect de tip `Intent` care poate fi creat:

- explicit, pe baza denumirii clasei care implementează serviciul respectiv;

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

 startService(intent);
```

- implicit, indicând componenta care gestionează serviciul respectiv (se va indica atât denumirea pachetului cât și denumirea clasei ca argument al metodei `setComponent()` asociat intenției respective):

```

 Intent intent = new Intent();
 intent.setComponent(new ComponentName("SomePackage",
 "SomeService"));

 startService(intent);

```

Transmiterea de informații suplimentare către serviciu poate fi realizată prin intermediul metodelor `putExtras(Bundle)`, `putExtra(String, Parcelable)` sau `put<type>Extra(String, <type>)`.

Metoda `startService()` presupune livarea unui mesaj asincron la nivelul sistemului de operare Android, astfel încât aceasta se execută instantaneu. Fiecare apel al acestei metode invocarea, în mod automat, al metodei `onStartCommand()`.

De regulă, comunicația dintre o componentă și un serviciu este unidirecțională, dintre componentă care invocă către serviciul invocat, prin intermediul datelor încapsulate în intenție. În situația în care se dorește ca serviciul să transmită date către componentă, se utilizează un obiect de tip [PendingIntent](#), prin intermediul căruia pot fi transmise mesaje cu difuzare.

## Oprirea unui Serviciu

Un serviciu poate fi opriți:

- de către o componentă (aceeași care l-a pornit sau alta), printr-un apel al metodei `stopService()`, ce primește ca parametru un obiect de tip `Intent` care poate fi creat explicit sau implicit;

Întrucât apelurile metodei `startService()` nu sunt imbicate, invocarea metodei `stopService()` oprește numai serviciul corespunzător (dacă se află în execuție).

- chiar de el însuși, în momentul în care procesările pe care trebuie să le realizeze s-au terminat, printr-un apel al metodei `stopSelf()`, eliberând resursele pe care sistemul de operare le-ar fi folosit pentru a-l menține în execuție; metoda poate fi apelată:
  - fără parametri, pentru a forța oprirea imediată;
  - transmînd un parametru de tip întreg, reprezentând identificatorul instanței care rulează, pentru a se asigura faptul că procesarea a fost realizată pentru fiecare apel care a fost realizat; dacă între timp serviciul a mai fost invocat, identificatorul folosit nu va corespunde celui mai recent apel (transmis ca argument al metodei `onStartCommand()`) și metoda nu va avea nici un efect.

Pentru oprirea unui serviciu este suficient un singur apel al uneia dintre metodele `stopSelf()` sau `stopService()`.

## Utilizarea clasei IntentService

Recursul la clasa `IntentService` este adekvată în situația în care se dorește implementarea unor servicii care rulează în fundal, **pe un fir de execuție dedicat** realizând un set de operații la un moment dat de timp, atunci când sunt solicitate. În situația în care serviciul este accesat simultan de mai multe componente ale unei (unor) aplicații Android, procesarea acestora va fi realizată secvențial.

Un obiect de acest tip este lansat în execuție prin pornirea unui serviciu și transmiterea unui obiect de tip `Intent` care conține toți parametrii necesari pentru realizarea sarcinii respective. Toate operațiile solicitate sunt înregistrate într-o coadă de așteptare și executate succesiv, prin invocarea automată a metodei `onHandleIntent()`. După ce procesarea a fost finalizată, procesul se oprește singur (nu este necesar să se apeleze metodele `stopSelf()` sau `stopService()` explicit).

Clasa `IntentService` oferă o implementare implicită a metodelor `onStartCommand()` care plasează intenția către coada de așteptare și invocă metoda `onHandleIntent()` precum și a metodei `onBind()` care întoarce valoarea `null`.

Prin urmare, o implementare presupune definirea unei clase derivată din `IntentService` care suprascrie metoda `onHandleIntent()`, ce primește ca parametru intenția conținând parametrii ce se doresc a fi procesați, execuția sa fiind realizată (în mod transparent) pe un fir de execuție ce rulează în fundal. De asemenea, trebuie furnizat și un constructor implicit.

```
import android.app.IntentService;
import android.content.Intent;

public class SomeStartedService extends IntentService {

 private final String name = "SomeStartedService";

 public SomeIntentService() {
 super(name);
 // ...
 }

 @Override
 protected void onHandleIntent(Intent intent) {
 // ...
 }
}
```

În situația în care este necesar să se suprascrie metodele `onCreate()`, `onDestroy()` sau `onStartCommand()`, este recomandat să se apeleze și metodele din clasa părinte, astfel încât ciclul de viață al firului de execuție pe care sunt realizate procesările să fie actualizat corespunzător.

```
@Override

public int onStartCommand(Intent intent, int flags, int startId) {
 // ...
 return super.onStartCommand(intent, flags, startId);
}
```

## Utilizarea clasei Service

În situația în care este necesar ca serviciul să gestioneze mai multe solicitări concomitent, se implementează o clasă derivată din `Service`, definindu-se (intern) o clasă de tip fir de

execuție (uzual, se folosește o instanță a clasei `Handler`, care permite transmiterea de mesaje) pe care vor fi realizate toate procesările.

#### SomeStartedService.java

```
public class SomeStartedService extends Service {
 private Looper looper;
 private ServiceHandler serviceHandler;
 private final static String handlerThreadName = "HandlerThread";

 private final class ServiceHandler extends Handler {
 public ServiceHandler(Looper looper) {
 super(looper);
 }

 @Override
 public void handleMessage(Message message) {
 // ...
 stopSelf(message.arg1);
 }
 }

 @Override
 public void onCreate() {
 HandlerThread handlerThread = new HandlerThread(handlerThreadName,
 Process.THREAD_PRIORITY_BACKGROUND);
 handlerThread.start();
 looper = handlerThread.getLooper();
 serviceHandler = new ServiceHandler(looper);
 }

 @Override
 public int onStartCommand(Intent intent, int flags, int startId) {
 Message message = serviceHandler.obtainMessage();
 message.arg1 = startId;
 serviceHandler.sendMessage(message);
 return START_STICKY;
 }

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

Procesarea este realizată în cadrul metodei `handleMessage()` a clasei de tip `Handler`. O instanță a acesteia este creată de fiecare dată când este invocată metoda `sendMessage()`, care are drept argument un obiect de tip `Message`. Prin intermediul mesajului (obținut ca rezultat al metodei `obtainMessage()` care furnizează o instanță dintr-o colecție globală), trebuie să se transmită și identificatorul cu care a fost invocat serviciul, astfel încât acesta să fie transmis către metoda `stopSelf()`, necesară pentru oprirea serviciului. Astfel, se asigură faptul că serviciul nu se termină până ce nu este tratată și cea mai recentă invocare a sa.

## Servicii de tip Bounded

Un serviciu de tip bounded implică o interacțiune permanentă cu una sau mai multe componente ale unei (unor) aplicații Android. Astfel, serviciul poate expune o anumită funcționalitate către componentele aplicației Android (prin intermediul unei interfețe) sau poate fi utilizat ca mecanism de comunicație inter-proces, pentru transmiterea de informații între acestea.

Arhitectura utilizată într-o astfel de situație este client-server, unde clientul este reprezentat de componenta aplicației Android, iar serverul de către serviciul de tip bounded. Existența unui serviciu de tip bounded este condiționată de existența unor componente atașate la el.

Implementarea unui serviciu de tip bounded presupune suprascrierea metodei `onBind()`, apelată în mod automat în momentul în care o componentă a unei aplicații Android este atașată acestuia, printr-un apel al metodei `bindService()`. Aceasta furnizează ca rezultat un obiect de tip `IBinder`.

De regulă, se definește o clasă internă de tip `Binder`, utilizată pentru apelul de metode la distanță în Android. Aceasta furnizează o metodă pentru obținerea unei referințe către serviciu, prin intermediul căreia pot fi accesate funcționalitățile pe care acesta le pune la dispoziție. O astfel de abordare este utilă mai ales în situația în care serviciul este folosit de o singură aplicație Android, **în cadrul aceluiași proces** (nefiind deci necesară comunicația inter-proces).

#### SomeBoundedService.java

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

public class SomeBoundedService extends Service {

 final private IBinder binder = new SomeBinder();

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

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

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

 public void doWork() {
 // ...
 }
}
```

Legătura dintre serviciu și o componentă a unei aplicații Android (locale) este reprezentată sub forma unui obiect de tipul `ServiceConnection`, responsabil cu monitorizarea conexiunii

dintre acestea. Implementarea sa presupune definirea metodelor `onServiceConnected()`, respectiv `onServiceDisconnected()`. Acestea sunt apelate automat în momentul în care componenta este conectată la serviciu (ca urmare a apelului `bindService()`), respectiv este deconectată (drept rezultat al apelului `unbindService()`).

Metoda `onServiceConnected()` primește ca argumente, un parametru de tip `ComponentName` care descrie pachetul și clasa coresunțătoare serviciului, respectiv un parametru de tip `IBinder` ce reprezintă rezultatul furnizat de metoda de callback `onBind()` de la nivelul serviciului. Aceasta este utilizată pentru a accesa funcționalitatea pusă la dispoziție de către serviciu. În situația în care conexiunea dintre componentă și serviciu este coruptă, va fi aruncată excepția `DeadObjectException`, care ar trebui să fie tratată pentru a asigura un comportament adecvat al aplicației Android.

Metoda `onServiceDisconnected()` primește ca argument un parametru de tip `ComponentName` care descrie pachetul și clasa coresunțătoare serviciului. De regulă, în cadrul acestei metode sunt eliberate resursele utilizate pentru comunicația dintre componentă și serviciu.

```
public class SomeActivity extends Activity {
 // ...

 private SomeBoundedService someBoundedService = null;
 private int status;

 private ServiceConnection serviceConnection = new ServiceConnection() {
 @Override
 public void onServiceConnected(ComponentName className, IBinder service) {
 someBoundedService =
 ((SomeBoundedService.SomeBinder) service).getService();
 status = BOUNDED;
 }

 @Override
 public void onServiceDisconnected(ComponentName className) {
 someBoundedService = null;
 status = UNBOUNDED;
 }
 };
}
```

Atașarea propriu-zisă a unei componente a unei aplicații Android la un serviciu este realizată de metoda `bindService()` care primește ca parametri:

- intenția reprezentând serviciul care se asociază activității;
- un obiect de tip `ServiceConnection`;
- anumite valori prin care se controlează modul în care este făcută asocierea între cele două componente ale aplicației Android:
  - `Context.BIND_AUTO_CREATE` - serviciul trebuie creat în momentul în care se realizează asocierea;

- Context.BIND\_ADJUST\_WITH\_ACTIVITY - modifică prioritatea serviciului în funcție de importanța activității de care este legat (mai mare atunci când este activă, mai mică atunci când este inactivă);
- Context.BIND\_ABOVE\_CLIENT / Context.BIND\_IMPORTANT - specifică faptul că prioritatea serviciului este mai mare decât a activității asociate;
- Context.BIND\_NOT\_FOREGROUND - asigură faptul că serviciul atașat activității nu va avea niciodată prioritatea necesară pentru a rula în prim-plan;
- Context.BIND\_WAIVE\_PRIORITY - face ca prioritatea serviciului asociat să nu fie modificată (implicit, prioritatea relativă a serviciului este mărită).

Asocierea este realizată, de obicei, în cadrul metodei `onStart()` a unei activități. Detașarea unei componente a unei aplicații Android la un serviciu este realizată de metoda `unbindService()` care primește ca argument un obiect de tip `ServiceConnection` (care monitorizează starea conectivității dintre componentă și serviciu). Decuplarea este realizată, de obicei, în cadrul metodei `onStop()` a unei activități.

```
public class SomeActivity extends Activity {

 // ...

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

 @Override
 protected void onStop() {
 super.onStop();
 if (status == BOUNDED) {
 unbindService(serviceConnection);
 }
 }
}
```

În situația în care este necesar ca serviciul să fie utilizat și atunci când activitatea se află în fundal, metodele `bindService()` și `unbindService()` pot fi apelate pe metodele de callback `onCreate()` respectiv `onDestroy()`. Totuși, aceasta implică o probabilitate mai mare ca memoria utilizată de serviciu să fie solicitată de sistemul de operare Android, dacă acesta nu rulează în contextul aceluiasi proces ca și componenta cu care interacționează.

Nu se recomandă ca atașarea, respectiv detașarea unei componente la / de un serviciu să fie realizată pe metodele `onResume()` și `onPause()` încărcăt acestea sunt apelate frecvent, astfel încărt procesările realizate în aceste momente să poată determina deprecieri ale performanței sistemului de operare Android. Mai mult, în cazul în care mai multe activități interacționează cu același serviciu, atunci când se realizează transferul între ele, serviciul ar putea fi distrus

(pe metoda `onPause()` se apelează `unbindService()`, deci toate componentele au fost detașate serviciului), respectiv recreat (pe metoda `onResume()` se apelează `bindService()`, deci serviciul are o componentă atașată).

În situația în care este necesar ca serviciul să poată fi atașat la mai multe componente simultan, aparținând unor procese diferite (realizându-se astfel comunicația inter-proces), realizând procesări pe fire de execuție dedicate, se folosește un obiect de tip [Messenger](#). Obiectul de tip `Binder` asociat acestuia va fi transmis ca rezultat al metodei  `onBind()`. De asemenea, el va încapsula un obiect de tip `Handler` pe căruia metoda `handleMessage()` va fi realizată procesarea corespunzătoare serviciului, acesta putând gestiona tipuri de mesaje diferite, corespunzătoare fiecărui tip specific de componentă. Obiectul `Messenger` plasează toateapelurile către serviciu într-o coadă de așteptare, toate fiind procesate, secvențial, în contextul aceluiasi fir de execuție.

#### SomeBoundedService.java

```
public class SomeBoundedService extends Service {

 class SomeHandler extends Handler {
 @Override
 public void handleMessage(Message message) {
 switch (msg.arg1) {
 // ...
 default:
 super.handleMessage(message);
 }
 }
 }

 final Messenger messenger = new Messenger(new SomeHandler());
}

@Override
public IBinder onBind(Intent intent) {
 return messenger.getBinder();
}
}
```

Dacă este necesar ca procesarea să fie realizată simultan, poate fi utilizat AIDL (Android Interface Definition Language) care realizează transformarea dintre obiecte și primitive la nivelul sistemului de operare prin intermediul cărora se realizează comunicația inter-proces. În această situație, trebuie avut în vedere faptul că sincronizarea dintre diferitele fire de execuție trebuie realizată manual, operație destul de complexă. Pe baza fișierului `.aidl` care descrie interfața de programare, se va genera o clasă abstractă responsabilă cu comunicația inter-proces, din care va fi derivat serviciul respectiv.

## Prioritatea unui Serviciu

În situația în care este necesar, sistemul de operare Android poate solicita memoria utilizată de un anumit serviciu, în funcție de prioritatea pe care acesta o are.

Un serviciu de tip `started` are o prioritate mai mare decât o activitate inactivă, dar mai mică decât o activitate activă. Din acest motiv, este necesar ca programatorul să trateze cazurile în care serviciul este repornit, ca urmare a distrugerii sale de către sistemul de operare Android, în vederea satisfacerii necesarului de memorie.

Un serviciu de tip bounded are de obicei o prioritate similară cu activității cu care interacționează, deși aceasta poate fi controlată prin intermediul unor parametri transmiși în momentul în care este realizată legătura. Este puțin probabil ca un serviciu atașat unei activități active să fie distrus la fel cum este destul de probabil ca un serviciu pentru care toate activitățile atașate sunt inactive să fie distrus.

Un serviciu care rulează în prim-plan nu este aproape niciodată distrus.

## Categorii de Procesări

În funcție de firul de execuție pe care rulează precum și de prioritatea care le este atribuită (în funcție de care sistemul de operare Android poate reclama resursele pe care acestea le utilizează), operațiile complexe pot fi realizate prin intermediul unor:

- servicii care rulează în prim-plan;
- procesări realizate în fundal.

### Servicii ce Rulează în Prim-Plan

În situația în care un serviciu interacționează cu utilizatorul (sau utilizatorul este conștient de existența sa), prioritatea acestuia trebuie să crească, astfel încât sistemul de operare Android să nu poată reclama resursele pe care le utilizează. Un astfel de serviciu trebuie să poată oferi o notificare în bara de stare, care nu poate fi ascunsă decât atunci când serviciul este distrus sau nu mai rulează în prim-plan.

Un astfel de comportament poate fi obținut prin marcarea serviciului ca rulând în prim-plan, prin invocarea metodei [startForeground\(\)](#) care primește ca parametri:

- identificatorul unei notificări, care **trebuie să fie diferit de zero**;
- un obiect de tip [Notification](#), care va fi afișată cât timp serviciul se află în prim-plan, întrucât se presupune că serviciul trebuie să aibă o reprezentare vizuală cu care utilizatorul să poată interacționa.

```
NotificationCompat.Builder notificationBuilder =
 new NotificationCompat.Builder(this)
 .setSmallIcon(R.drawable.notification_icon)
 .setContentTitle("...")

 .setContentText("...");

Intent intent = new Intent(this, SomeActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent,
PendingIntent.FLAG_UPDATE_CURRENT);
notificationBuilder.setContentIntent(pendingIntent);
Notification notification = notificationBuilder.build();
startForeground(NOTIFICATION_ID, notification);
```

Este recomandat ca utilizatorii să poată dezactiva ei însăși rularea serviciului în prim-plan, de regulă prin intermediul obiectului de tip [Notification](#). Același lucru trebuie realizat și în momentul în care rularea serviciului în prim-plan nu mai este necesară. În acest sens, trebuie apelată metoda [stopForeground\(true\)](#) (argumentul metodei indică dacă se dorește și terminarea notificării), aceasta anulând în mod automat notificarea asociată.

Nu este indicat să existe mai multe servicii care rulează în prim-plan simultan întrucât acestea nu pot fi distruse de sistemul de operare Android și în situația în care necesarul de memorie devine o problemă critică, performanțele dispozitivului mobil se vor deprecia considerabil.

## Procesări Realizate În Fundal

Întrucât responsivitatea aplicației Android reprezintă un criteriu foarte important, asigurarea unei experiențe corespunzătoare a utilizatorului poate fi obținută prin transferul operațiilor complexe de procesare și a operațiilor de intrare / ieșire în fundal, pe un alt fir de execuție. Acest lucru este necesar datorită faptului că pe firul de execuție principal sunt rulate mai multe componente ale aplicației Android (activități, servicii, ascultători ai intențiilor cu difuzare), astfel încât interacțiunea cu utilizatorul poate fi blocată în situația în care procesarea realizată de servicii nu este transferată în fundal.

În Android, o activitate care nu răsunde la un eveniment în decurs de 5 secunde sau un ascultător al unei intenții cu difuzare a cărei metodă `onReceive()` nu se termină în 5 secunde sunt considerate blocate.

De regulă, sunt plasate pe fire de execuție dedicate operațiilor cu fișiere (scrieri, citiri), căutări în rețea, tranzacții cu baza de date, calcule complexe.

Android oferă mai multe mecanisme pentru gestiunea serviciilor care rulează în fundal:

1. utilizarea clasei `AsyncTask` permite definirea unei operații care va fi realizată pe un fir de execuție separat, oferind metode care oferă informații cu privire la progres, ce pot fi consultate de alte fire de execuție;
2. definirea unei clase derivate din `Loader`;
3. utilizarea clasei `Handler`;
4. implementarea unui fir de execuție definit de utilizator, transmiterea valorilor către interfața grafică fiind realizat prin intermediul clasei `Handler`.

### Clasa `AsyncTask`

Prin intermediul clasei `AsyncTask` se permite mutarea operațiilor costisitoare din punct de vedere al utilizării procesorului și al memoriei pe fire de execuție rulate în fundal, oferind **sincronizarea** cu firul de execuție ce randează interfața grafică, acesta fiind notificat cu privire la progresul realizat cât și cu privire la momentul în care procesarea a fost terminată.

Întrucât clasa `AsyncTask` nu este persistentă în cazul (re)pornirii unei activități (situație în care firul de execuție asociat este distrus), se recomandă utilizarea sa pentru procese de fundal care nu durează o perioadă de timp prea mare.

O implementare a clasei `AsyncTask` poate fi parametrizată cu tipurile de date care sunt folosite pentru intrare, pentru raportarea progresului și pentru ieșire (rezultate).

În situația în care nu se dorește transmiterea unor parametrii de intrare / ieșire sau nu este necesară raportarea progresului, se poate utiliza valoarea `Void` pentru toate aceste tipuri.

Este necesară suprascrierea următoarelor metode:

- `doInBackground()` - metoda care va fi executată pe firul de execuție dedicat, care primește parametrii de intrare de tipul specificat; acesta trebuie să conțină procesările complexe care se doresc a fi realizate fără a interacționa însă cu firul de execuție principal; notificările sunt realizate:
  - prin intermediul metodei `publishProgress()`, care primește ca parametru progresul realizat, valorile fiind transmise metodei `onProgressUpdate()`;
  - prin întoarcerea unui rezultat, atunci când procesarea este terminată, valoarea fiind transmisă metodei `onPostExecute()`;
  - `onProgressUpdate()` - folosită pentru actualizarea interfeței grafice, cu valorile primite de la metoda `publishProgress()`; metoda este sincronizată cu firul de execuție principal (care randează interfața grafică), astfel încât diferitele controale pot fi accesate de aici;
  - `onPreExecute()` - apelată în contextul firului de execuție al interfeței grafice, înainte de metoda `doInBackground()`;
  - `onPostExecute()` - apelată în momentul în care procesarea este terminată, primind ca parametru rezultatul întors de metoda `doInBackground()`; metoda este sincronizată cu firul de execuție principal (care randează interfața grafică), astfel încât diferitele controale pot fi accesate de aici.

```

private class SomeAsyncTask extends AsyncTask<String, Integer, String> {
 @Override
 protected String doInBackground(String... parameter) {
 String result = new String();
 int progress = 0;
 for (int step = 1; step <= parameter[0].length(); step++) {
 progress = step;
 result += parameter[0].charAt(parameter[0].length() - step);
 try {
 Thread.sleep(100);
 } catch (InterruptedException interruptedException) {
 Log.e(Constants.TAG, "An exception has occurred: " +
interruptedException.getMessage());
 }
 publishProgress(progress);
 }
 return result;
 }

 @Override
 protected void onProgressUpdate(Integer... progress) {
 progressTextView.setText(progress[0].toString());
 }
}

```

```

@Override
protected void onPreExecute() {
 // ...
}

@Override
protected void onPostExecute(String result) {
 progressTextView.setText(result);
}
}

```

Rularea unui obiect de tip `AsyncTask` se face prin invocarea metodei `execute()` care primește ca parametru informațiile care se doresc a fi procesate.

Fiecare obiect de tip `AsyncTask` poate fi rulat o singură dată. Apelurile ulterioare ale metodei `execute()` vor genera excepții.

## Clasa Loader

Clasa abstractă `Loader` implementează practicile recomandate pentru încărcarea asincronă a informațiilor în cadrul unor controale grafice din interfața cu utilizatorul (afișate în activități sau fragmente).

O astfel de abordare este utilizată pentru:

- a încărca date asincron;
- a monitoriza sursele din care sunt încărcate datele, oferind informații cu privire la rezultatele obținute.

De regulă, se folosește o implementare a clasei `AsyncTaskLoader`.

## Clasa Handler

Clasa `Handler` gestionează transmiterea de mesaje sau execuția unor rutine asociate unor fir de execuție prin intermediul unei cozi de așteptare (de tipul `MessageQueue`). Un obiect de acest tip este asociat unui singur fir de execuție. Acesta este creat în momentul în care este realizată instanță, către cesta fiind transmise toate mesajele, **secvențial**.

Prin intermediul acestui obiect, se permite:

- planificarea mesajelor spre a fi procesate la un moment fix din viitor (metodele `send...`) de tip `send...(): sendEmptyMessage(), sendMessage(), sendMessageAtTime(), sendMessageDelayed()`;
- încapsularea unei acțiuni care va fi executată pe un alt fir de execuție decât firul de execuție al interfeței grafice (metodele de tip `post...(): post(), postAtTime(), postDelayed()`).

## Utilizarea firelor de execuție definite de utilizator

Folosirea unor fir de execuție definite de utilizator și sincronizarea manuală cu interfața grafică poate fi necesară în situația în care trebuie realizată o gestiune mai complexă decât cea oferită de clasele `AsyncTask`, `Handler` sau `Loader`.

În acest sens, este folosită o implementare a clasei `Thread`, procesarea pe un fir de execuție separat fiind realizată în cadrul metodei `run()`.

```
private void executeOnDedicatedThread() {
 Thread dedicatedThread = new Thread(new Runnable() {
 @Override
 public void run() {
 // ...
 }
 });
 dedicatedThread.start();
}
```

Dacă se dorește actualizarea controalelor din cadrul interfeței grafice, este necesar ca firile de execuție care rulează în fundal să fie sincronizate cu firul de execuție principal anterior acestei operații. Acest lucru poate fi realizat:

1. folosind metoda `Activity.runOnUiThread()` care forțează codul transmis să fie executat pe același fir de execuție care redă interfața grafică:

```
2. runOnUiThread(new Runnable() {
 3. @Override
 4. public void run() {
 5. // ...
 6. }
}) ;
```

7. utilizând un obiect de tip `Handler` pentru a realiza actualizări în contextul firului de execuție în care acesta a fost creat, utilizând metoda `post()`:

```
8. private Handler handler = new Handler(); // created on the
main thread
9.
10. private void executeOnSeparateThread() {
11. Thread separateThread = new Thread(new Runnable() {
12. @Override
13. public void run() {
14. // do some background processing here
15. handler.post(new Runnable() {
16. @Override
17. public void run() {
```

```

18. // access the graphical user interface here
19. }
20. })
21. }
22. });
23. separateThread.start();
}

```

Clasa Handler pune la dispoziție și metode pentru a executa anumite metode la un moment dat de timp:

- [postDelayed\(\)](#) - realizează o modificare la nivelul interfeței grafice cu o întârziere specificată ca parametru (exprimat în milisecunde);
- [postAtTime\(\)](#) - realizează o modificare la nivelul interfeței grafice la un moment de timp specificat ca parametru (exprimat ca număr de milisecunde ce au trecut de la 1 ianuarie 1970).

## Realizarea de Procesări prin Intermediul Alarmelor

(Optional)

### Utilitare

---

- Listare servicii din acest laborator

```
adb shell dumpsys activity services | grep lab05
```

- Oprire forțată a unui serviciu

```
am force-stop ro.pub.cs.systems.eim.lab05.startedservice
```

- Utilitarul MyAndroidTools (preinstalat în genymotion) listează activitățile și serviciile unei aplicații. Pentru servicii sunt două coloane: servicii pornite (albastru) și servicii disponibile(negru).
- În Android Studio/Tool/Android/Android Device Monitor - se deschide perspectiva DDMS care oferă control asupra proceselor și thread-urilor care rulează în emulator/telefon.

## Activitate de Laborator

---

**1.** În contul Github personal, să se creeze un depozit denumit 'Laborator05'. Initial, acesta trebuie să fie gol (nu trebuie să bifați nici adăugarea unui fișier README.md, nici a fișierului .gitignore sau a fișierului LICENSE).

**2.** Să se cloneze în directorul de pe discul local conținutul depozitului la distanță de la <https://www.github.com/eim2017/Laborator05>. În urma acestei operații, directorul Laborator05 va trebui să se conțină un director labtasks ce va deține proiectele AndroidStudio, fișierul README.md și un fișier .gitignore care indică tipurile de fișiere (extensiile) ignorate.

```
student@eim2017:~$ git clone https://www.github.com/eim2017/Laborator05.git
```

**3.** Să se încarce conținutul descărcat în cadrul depozitului 'Laborator05' de pe contul Github personal.

```
student@eim2017:~$ cd Laborator05
```

```
student@eim2017:~/Laborator05$ git remote add Laborator05_perfectstudent https://github.com/perfectstudent/Laborator05
```

```
student@eim2017:~/Laborator05$ git push Laborator05_perfectstudent master
```

**4.** Să se încarce în mediul integrat de dezvoltare Android Studio proiectele *StartedService* respectiv *StartedServiceActivity* din directorul labtasks/StartedService.

- Proiectul *StartedService* conține codul sursă pentru un serviciu de tip started care transmite mai multe valori, de diferite tipuri (șir de caractere, întreg, vector), temporizate la un anumit interval (dată de valoarea SLEEP\_TIME din interfața Constants). Aceste valori sunt transmise prin intermediul unor intenții cu difuzare (eng. broadcast intents), la nivelul întregului sistem de operare Android.
- Proiectul *StartedServiceActivity* conține codul sursă pentru o aplicație Android care utilizează un ascultător pentru intenții cu difuzare (eng. BroadcastReceiver), pentru tipurile de mesaje propagate la nivelul sistemului de operare de către serviciu, pe care le afișează în interfață grafică, prin intermediul unui câmp text.



5. În proiectul *StartedService*, în clasa `StartedService` din pachetul `ro.pub.cs.systems.eim.lab05.startedservice.service`, să se completeze metoda `onStartCommand()` astfel încât aceasta să pornească un fir de execuție în cadrul căruia să fie propagate 3 intenții cu difuzare la nivelul sistemului de operare Android. Pentru fiecare intenție, se vor specifica:

- **acțiunea**, care va avea valorile definite în interfața `Constants` (`Constants.ACTION_STRING`, `Constants.ACTION_INTEGER`, `Constants.ACTION_ARRAY_LIST`); se va utiliza metoda `setAction()`;

- **informațiile transmise**, plasate în câmpul extra (având cheia Constants.DATA și valoarea dată de Constants.STRING\_DATA, Constants.INTEGER\_DATA, Constants.ARRAY\_LIST\_DATA); se va utiliza metoda [putExtra\(\)](#) care primește ca argumente cheia și valoarea.

Transmiterea propriu-zisă a intenției se face prin intermediul metodei [sendBroadcast\(\)](#). Cele trei mesaje vor fi temporizate la intervalul indicat de valoarea Constants.SLEEP\_TIME (propagarea mesajelor va fi intercalată de apeluri [Thread.sleep\(\)](#)).

- De ce este necesar ca serviciul să realizeze operațiile pe un fir de execuție dedicat?
- Ce alternativă s-ar fi putut folosi pentru a se evita o astfel de abordare? Ce avantaj și ce dezvantaj prezintă această alternativă?

#### Indicații de Rezolvare

Monitorizați ciclurile din Thread.run() în logcat:

```
Log.d(Constants.TAG, "Thread.run() was invoked, PID: " + android.os.Process.myPid() + " TID: " + android.os.Process.myTid());
```

- În proiectul *StartedServiceActivity*, să se pornească serviciul, printr-un apel al metodei [startService\(\)](#); intenția care va fi transmisă ca argument metodei startService() trebuie să refere **explicit** serviciul care urmează a fi pornit, din motive de securitate (se folosește metoda [setComponent\(\)](#), care indică atât pachetul corespunzător aplicației Android care conține serviciul, cât și clasa corespunzătoare acestuia - calificată complet).

#### Indicații de Rezolvare

- Să se ruleze aplicațiile. Se va rula aplicația *StartedService* care instalează serviciul pe dispozitivul mobil. Ulterior se va rula aplicația *StartedServiceActivity*. Verificați faptul că serviciul a fost pornit și opus corespunzător prin mesajele afișate în consolă.
- Monitorizați în DDMS procesele asociate activității și serviciului. Ce se întâmplă dacă activitatea este eliminată ([onDestroy\(\)](#))?
- Explicați ce se întâmplă dacă repornim activitatea (monitorizați în DDMS și logcat).

- În proiectul *StartedServiceActivity*, să se implementeze un ascultător pentru intenții cu difuzare, în clasa *StartedServiceBroadcastReceiver* din pachetul *ro.pub.cs.systems.eim.lab05.startedserviceactivity.view*. Aceasta extinde clasa [BroadcastReceiver](#) și implementează metoda [onReceive\(\)](#), având ca argumente **contextul** din care a fost invocată și **intenția** prin intermediul căreia a fost transmis mesajul respectiv. Astfel, datele extrase din intenție (având cheia indicată de Constants.DATA) vor fi afișate într-un câmp text (*messageTextView*) din cadrul interfeței grafice.

#### Indicații de Rezolvare

- În proiectul *StartedServiceActivity*, în cadrul metodei [onCreate\(\)](#) a activității *StartedServiceActivity* (din pachetul *ro.pub.cs.systems.eim.lab05.startedserviceactivity*), să se realizeze următoarele operații:

- să se creeze o instanță a ascultătorului pentru intenții cu difuzare;

#### Indicații de Rezolvare

**b)** să se creeze o instanță a unui obiect de tipul [IntentFilter](#), la care să se adauge toate acțiunile corespunzătoare intențiilor cu difuzare propagate de serviciu; se va folosi metoda [addAction\(\)](#);

#### Indicații de Rezolvare

**c)** să se atașeze, respectiv să se detașeze ascultătorul de intenții cu difuzare, astfel încât acesta să proceseze mesajele primite de la serviciu doar în situația în care activitatea este vizibilă pe suprafața de afișare; în acest sens, vor fi utilizate metodele [registerReceiver\(\)](#), respectiv [unregisterReceiver\(\)](#), apelate pe metodele de callback ale activității corespunzătoare stării în care aceasta este vizibilă pe suprafața de afișare ([onResume\(\)](#), respectiv [onPause\(\)](#)).

#### Indicații de Rezolvare

**d)** Să se opreasă serviciul printr-un apel al metodei [stopService\(\)](#). Unde ar putea fi plasată aceasta? Care sunt avantajele și dezavantajele unei astfel de abordări?

**9.** Rulați din nou aplicația, întrerupând temporar activitatea (printr-o apăsare a tastei *Home*) în timp ce sunt procesate intențiile cu difuzare transmise de serviciu. Ce observați la revenirea în activitate?

Modificați modul de implementare al ascultătorului de intenții cu difuzare astfel încât în momentul în care se primește un mesaj, să repornească activitatea (dacă este cazul), asigurându-se astfel faptul că nu se mai pierde nici o informație transmisă de serviciu dacă aceasta nu este vizibilă pe suprafața de afișare.

#### Indicații de Rezolvare

**10.** Să se încarce în mediul integrat de dezvoltare Android Studio proiectul *BoundedServiceActivity* din directorul *labtasks/BoundedService*. Se dorește să se implementeze un serviciu de tip bounded care poate interacționa cu o activitate prin intermediul unei metode care furnizează un sir de caractere ales aleator dintr-o listă de valori.

**a)** În [clasa BoundedService](#) din pachetul [ro.pub.cs.systems.eim.lab05.boundedserviceactivity.service](#), să se implementeze o clasă internă **publică**, derivată din interfața [IBinder](#), care oferă o referință către serviciu prin intermediul metodei **public** [getService\(\)](#). Instanța acestei clase interne va fi furnizată ca rezultat al metodei [onBind\(\)](#).

#### Indicații de Rezolvare

**b)** În [clasa BoundedService](#) din pachetul [ro.pub.cs.systems.eim.lab05.boundedserviceactivity.service](#), să se implementeze o metodă care întoarce o valoare aleatoare din tabloul de constante [Constants.MESSAGES](#). Aceasta va avea identificatorul de acces **public**, astfel încât să poată fi accesată din cadrul activității.

**c)** În [clasa BoundedServiceActivity](#) din pachetul [ro.pub.cs.systems.eim.lab05.boundedserviceactivity.view](#), să se instantțieze un obiect de tipul [ServiceConnection](#), care va fi utilizat pentru asocierea serviciului la activitate.

Vor fi suprascrise metodele:

- onServiceConnected() - se creează referință la serviciu prin intermediul argumentului de tipul `IBinder`(prin intermediul metodei `getService()`) și se actualizează variabila care reține starea legăturii dintre serviciu și activitate;
- onServiceDisconnected() - se distrugе referință la serviciu și se actualizează variabila care reține starea legăturii dintre serviciu și activitate.

#### Indicații de Rezolvare

**d)** Să se atașeze, respectiv să se detașeze activitatea la / de serviciu, în cadrul metodelor de callback corespunzătoare stării în care aceasta poate interacționa cu utilizatorul.

- pe metoda `onStart()` se apelează metoda `bindService()`, utilizându-se un obiect de tip `Intent explicit` (care primește ca argument clasa care urmează să fie invocată);
- pe metoda `onStop()` se apelează metoda `unbindService()`; se actualizează variabila care reține starea legăturii dintre serviciu și activitate.

#### Indicații de Rezolvare

**e)** Să se implementeze o clasă ascultător pentru evenimentul de tip apăsare corespunzător controlului grafic de tip buton. În cadrul metodei de tratare a evenimentului, să se apeleze metoda din cadrul serviciului care furnizează un mesaj aleator și să se afișeze acest mesaj în cadrul câmpului text, împreună cu data și ora la care a fost furnizat.



#### Indicații de Rezolvare

**f)** Să se afișeze mesaje sugestive pe metodele de callback ale activității și ale serviciului și să se observe care este momentul în care acesta este creat, respectiv este distrus. Ce se întâmplă în momentul în care se apasă tasta *Home*?

**11.** Să se încarce modificările realizate în cadrul depozitului 'Laborator05' de pe contul Github personal, folosind un mesaj sugestiv.

```
student@eim2017:~/Laborator05$ git add labtasks/*
```

```
student@eim2017:~/Laborator05$ git commit -m "implemented tasks for laboratory 05"
```

```
student@eim2017:~/Laborator05$ git push Laborator05_perfectstudent master
```

# Colocviu 1

## Model de Subiect

### SUBIECTE

**A.1.** [5%] În contul Github personal, să se creeze un depozit denumit *PracticalTest01*.

Înțial, acesta trebuie să conțină:

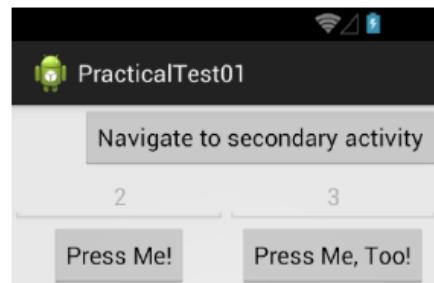
- ❑ un fișier `README.md`, în care veți preciza următoarele informații: numele și prenumele, grupa din care faceți parte;
- ❑ un fișier `.gitignore` prin intermediu căruia resursele generate ale aplicației Android (clase, fișiere `.ap_`, `.apk`, `.dex`) nu vor fi consemnate în cadrul sistemului de versionare a codului sursă;
- ❑ un fișier `LICENSE` care să descrie condițiile de utilizare a aplicației Android (la alegere).

**A.2.** [5%] Să se descarce pe discul local conținutul depozitului (la distanță) *PracticalTest01*.

**A.3.** [5%] În mediul integrat de dezvoltare preferat (Eclipse / Android Studio), să se creeze un proiect corespunzător unei aplicații Android, având următoarea specificație:

- ❑ denumirea aplicației Android și a proiectului este *PracticalTest01*;
- ❑ denumirea pachetului corespunzător aplicației Android este `ro.pub.cs.systems.eim.practicaltest01`;
- ❑ nivelul minim de API este cel corespunzător versiunii Jelly Bean (4.1);
- ❑ denumirea activității principale a aplicației Android este `PracticalTest01MainActivity`;
- ❑ denumirea fișierului ce descrie interfața grafică a activității principale a aplicației Android este `activity_practical_test01_main.xml`.

**A.4.** [15%] Să se creeze o interfață grafică care conține două câmpuri text, needitabile, dispuse pe o linie, unul lângă celălalt ocupând jumătate din suprafața containerului din care fac parte, având dedesubt câte un buton centrat relativ la ele. În cadrul câmpurilor text, se va afișa o cifră reprezentând numărul de apăsări al butonului (înțial, valoarea va fi 0).



**B.1.** [10%] Să se asocieze butoanelor un ascultător, astfel încât pentru fiecare eveniment de tip apăsare a butonului, valoarea din câmpul text corespunzător să fie incrementată.

**B.2.** [15%] **a)** Forțați sistemul de operare Android să nu salveze / restaureze în mod automat starea câmpurilor text în momentul în care aceasta este distrusă din cauza asigurării necesarului de resurse.

**b)** Implementați un mecanism pentru salvarea / restaurarea valorii din câmpurile text, dacă sistemul de operare Android distrug activitatea spre a asigura necesarul de resurse.

**c)** Simulați comportamentul de mai sus.

**d)** Ce se întâmplă în situația în care utilizatorul apasă butonul *Back*? Cum ar fi putut fi rezolvată problema de mai sus în aceste condiții?

---

---

**C.1.** [10%] Să se implementeze o nouă activitate (`PracticalTest01SecondaryActivity`, având interfață grafică în fișierul `activity_practical_test01_secondary.xml`) în contextul aceleiași aplicații Android, care să poată fi invocată doar prin intermediul unei intenții. Aceasta va conține un câmp text, care va afișa de câte ori au fost accesate cele două butoane (din activitatea principală) în total. Totodată, aceasta va avea și două butoane, *Ok* și *Cancel*, prin care se va reveni la activitatea din care a fost invocată, însă cu rezultate diferite.

**C.2.** [5%] În activitatea principală, să se creeze un buton prin care se poate invoca cealaltă activitate din aplicația Android, afișând la revenirea din ea (într-un control grafic de tip `Toast`) rezultatul care a fost primit.

**D.1.** [15%] **a)** Să se implementeze un serviciu de tip started denumit `PracticalTest01Service` care primește prin intenția cu care e invocat numerele din câmpurile text și propagă la nivelul sistemului de operare Android un mesaj cu difuzare, la fiecare 10 secunde ce conține data și ora la care a fost transmis, media aritmetică și media geometrică numerelor. Definiți trei acțiuni diferite, care vor fi asociate aleator intenției folosite pentru propagarea mesajului cu difuzare.

**b)** Să se pornească serviciul atunci când suma valorilor din cele două câmpuri text depășește un prag, definit la nivelul codului sursă sub formă de constantă. Serviciul este oprit (nu mai transmite mesaje cu difuzare) atunci când aplicația Android este distrusă.

**D.2.** [10%] Să se implementeze un ascultător pentru mesajele de difuzare. Procesarea unui mesaj cu difuzare implică jurnalizarea sa în consolă, folosind o anumită etichetă.

**D.3.** [5%] Să se încarce rezolvările în depozitul *PracticalTest01* de pe contul **Github personal**, folosind un mesaj sugestiv.

## Observații Generale

---

Pentru rezolvarea subiectelor propuse în cadrul colocviului 1, sunt necesare:

- un cont Github personal, pe care să existe drepturi de citire și de scriere;
- SDK-ul de Android (cu o imagine pentru nivelul de API 16 - Jelly Bean 4.1);
- mediul de dezvoltare integrat Android Studio;
- un dispozitiv mobil:
  - fizic (bring your own device);
  - virtual: Genymotion, AVD.

## Rezolvări

---

Proiectul Android Studio corespunzător aplicației Android ce conține rezolvările complete ale cerințelor colocviului sunt disponibile pe [contul de Github al disciplinei](#).

**A.1. a)** Se accesează [Github](#) și se realizează autentificarea în contul personal, prin intermediul butonului *Sign in*.



Features Business Explore Pricing

Search GitHub

Sign in or Sign up

Se creează o zonă de lucru corespunzătoare unui proiect prin intermediul butonului *New Repository*.

The screenshot shows a user interface for managing repositories. At the top, there's a header with "Your repositories 5" and a prominent green "New repository" button. Below the header is a search bar containing the placeholder "Find a repository...". Underneath the search bar are five filter tabs: "All" (which is underlined), "Public", "Private", "Sources", and "Forks". A list of five repositories is displayed below these filters:

- Laborator05
- Laborator01
- Laborator04
- Laborator03
- Laborator02

Configurarea depozitului la distanță presupune specificarea:

- unei denumiri;
- unei descrieri (optional);
- tipului de director (public sau privat - doar contra cost!!!);
- modului de inițializare:
  - local, prin `git init`;
  - la distanță, prin `git clone` - depozitul la distanță nu trebuie să fie vid, în această situație
- unui fișier `README` (optional);
- extensiilor ignore (corespunzătoare limbajului de programare folosit, în cazul de față - Android) - incluse în fișierul `.gitignore`;
- tipului de licență sub care este publicat codul sursă.

## Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

eim2017 ▾

Repository name

PracticalTest01



Great repository names are short and memorable. Need inspiration? How about [bookish-octo-broccoli](#).

Description (optional)

Public

Anyone can see this repository. You choose who can commit.

Private

You choose who can see and commit to this repository.

Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: [Android](#) ▾

Add a license: [Apache License 2.0](#) ▾



**Create repository**

**b)** Prin intermediul comenzi `git clone` se poate descărca întregul conținut în directorul curent (de pe discul local), inclusiv istoricul complet al versiunilor anterioare (care poate fi ulterior reconstituit după această copie, în cazul coruperii informațiilor stocate pe serverul la distanță).

În situația în care se dorește clonarea conținutului din depozitul la distanță în alt director decât cel curent, acesta poate fi transmis ca parametru al comenzi, după URL-ul la care poate fi accesat proiectul în Github.

```
student@eim2017:~$ git clone https://www.github.com/perfectstudent/PracticalTest01
```

**c)** Se urmăresc indicațiile disponibile în secțiunea [Crearea unei aplicații Android în Android Studio](#).

**A.2.** Implementarea interfeței grafice va fi realizată prin intermediul unui fișier `.xml` care va fi plasat în directorul `/res/layout` al proiectului.

De cele mai multe ori, interfața grafică poate fi realizată în două moduri:

- prin folosirea unui singur container de tip `RelativeLayout` în care vor fi plasate toate controalele, având poziția definită relativ unele la celelalte sau la marginile părintelui în care sunt incluse;
- prin folosirea mai multor containere de tip `LinearLayout`, imbricate, având orientări diferite (orizontal - implicit, sau vertical).
- Varianta 1 - un singur container de tip `RelativeLayout`

Dacă se dorește ca un anumit control să fie aliniat cu un control care se găsește deasupra sa (aliniere pe orizontală), acesta va fi indicat ca valoare pentru proprietățile `android:layout_alignLeft` și `android:layout_alignRight` (proprietațea `android:layout_width` fiind ignorată în acest caz). În mod similar, dacă se dorește alinierarea cu un control aflat la stânga (aliniere pe verticală), acesta va fi precizat ca valoare pentru proprietățile `android:layout_alignTop` și `android:layout_alignBottom` (proprietațea `android:layout_height` fiind ignorată în acest caz).

#### activity\_practical\_test01\_main.xml

```
<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="wrap_content"

 tools:context="ro.pub.cs.systems.eim.practicaltest01.PracticalTest01MainActivity" >

 <Button
 android:id="@+id/navigate_to_secondary_activity_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_alignParentTop="true"
 android:layout_alignParentRight="true"
 android:text="@string/navigate_to_secondary_activity" />

 <EditText
 android:id="@+id/left_edit_text"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:saveEnabled="false"
 android:enabled="false"
 android:inputType="number"
 android:ems="7"
 android:layout_alignParentLeft="true"
 android:layout_below="@+id/navigate_to_secondary_activity_button"
 android:gravity="center" />

 <EditText
 android:id="@+id/right_edit_text"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:saveEnabled="false"
 android:enabled="false"
```

```

 android:inputType="number"
 android:ems="7"
 android:layout_alignParentRight="true"
 android:layout_below="@+id/navigate_to_secondary_activity_button"
 android:gravity="center" />

 <Button
 android:id="@+id/left_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_alignLeft="@+id/left_edit_text"
 android:layout_below="@+id/left_edit_text"
 android:layout_alignRight="@+id/left_edit_text"
 android:text="@string/press_me" />

 <Button
 android:id="@+id/right_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_alignLeft="@+id/right_edit_text"
 android:layout_alignRight="@+id/right_edit_text"
 android:layout_below="@+id/right_edit_text"
 android:text="@string/press_me_too" />

</RelativeLayout>

```

- Varianta 2 - mai multe containere de tip `LinearLayout` imbricate

Dacă se dorește ca un anumit control să ocupe o anumită proporție în cadrul containerului, se va utiliza proprietatea `android:layout_weight`, însă pentru a fi luată în considerare, atributul `android:layout_width`, respectiv `android:layout_height` (în funcție de coordonata pe care se dorește obținerea unei astfel de funcționalități) trebuie să aibă valoarea `0dp`.

#### activity\_practical\_test01\_main.xml

```

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

 tools:context="ro.pub.cs.systems.eim.practicaltest01.PracticalTest01MainActivity" >

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

 <EditText
 android:id="@+id/left_edit_text"
 android:layout_width="0dp"
 android:layout_height="wrap_content"
 android:layout_weight="1"
 android:enabled="false"
 android:inputType="number"

```

```

 android:gravity="center" />

<EditText
 android:id="@+id/right_edit_text"
 android:layout_width="0dp"
 android:layout_height="wrap_content"
 android:layout_weight="1"
 android:enabled="false"
 android:inputType="number"
 android:gravity="center" />

</LinearLayout>

<LinearLayout
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:baselineAligned="false" >

 <ScrollView
 android:layout_width="0dp"
 android:layout_height="wrap_content"
 android:layout_weight="1">

 <Button
 android:id="@+id/left_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_gravity="center"
 android:text="@string/press_me" />

 </ScrollView>

 <ScrollView
 android:layout_width="0dp"
 android:layout_height="wrap_content"
 android:layout_weight="1">

 <Button
 android:id="@+id/right_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_gravity="center"
 android:text="@string/press_me_too" />

 </ScrollView>

</LinearLayout>
</LinearLayout>

```

- Varianta 3 - un container de tip `LinearLayout vertical` care include un control grafic de tip `Button` și un container `GridLayout`, cu 2 linii și 2 coloane; pe linia 0 se vor găsi cele 2 câmpuri text editabile, a căror lungime trebuie să fie specificată explicit

iar pe linia 1 cele două butoane, a căror aliniere (centrată pe orizontală raportat la celula din care fac parte) trebuie să fie precizată

### activity\_practical\_test01\_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:paddingBottom="@dimen/activity_vertical_margin"
 android:paddingLeft="@dimen/activity_horizontal_margin"
 android:paddingRight="@dimen/activity_horizontal_margin"
 android:paddingTop="@dimen/activity_vertical_margin"
 android:orientation="vertical"

 tools:context="ro.pub.cs.systems.eim.practicaltest01.PracticalTest01MainActivity" >

 <Button
 android:id="@+id/navigate_to_secondary_activity_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@string/navigate_to_secondary_activity"/>

 <GridLayout
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:rowCount="2"
 android:columnCount="2">

 <EditText
 android:id="@+id/left_edit_text"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:saveEnabled="false"
 android:enabled="false"
 android:inputType="number"
 android:ems="6"
 android:layout_row="0"
 android:layout_column="0"
 android:layout_gravity="center"
 android:gravity="center"/>

 <EditText
 android:id="@+id/right_edit_text"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:saveEnabled="false"
 android:enabled="false"
 android:inputType="number"
 android:ems="6"
 android:layout_row="0"
 android:layout_column="1"
 android:layout_gravity="center"
 android:gravity="center" />

```

```

 <Button
 android:id="@+id/left_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_row="1"
 android:layout_column="0"
 android:layout_gravity="center"
 android:text="@string/press_me" />

 <Button
 android:id="@+id/right_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_row="1"
 android:layout_column="1"
 android:layout_gravity="center"
 android:text="@string/press_me_too" />

 </GridLayout>

</LinearLayout>

```

În activitatea corespunzătoare, interfața grafică definită în fișierul .xml trebuie încărcată pe metoda `onCreate()` prin intermediul `setContentView()`, căreia i se transmite ca parametru identificatorul (referința) către această resursă (aşa cum a fost generată în clasa abstractă `R.layout`). În situația în care este necesar să se realizeze anumite operații pe controalele grafice componente (spre exemplu, să se încarce conținutul acestora), trebuie inițial să se obțină o referință către aceasta printr-un apel al metodei `findViewById()`, care primește de asemenea ca parametru identificatorul (referința) către această resursă (aşa cum a fost generată în clasa abstractă `R.id`).

Nu este necesar să se implementeze metodele `onCreateOptionsMenu()`, respectiv `onOptionsItemSelected()`.

### PracticalTest01Activity.java

```

package ro.pub.cs.systems.eim.practicaltest01;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class PracticalTest01MainActivity extends Activity {

 private EditText leftEditText = null;
 private EditText rightEditText = null;
 private Button leftButton = null;
 private Button rightButton = null;

 @Override

```

```

protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_practical_test01_main);

 leftEditText = (EditText) findViewById(R.id.left_edit_text);
 rightEditText = (EditText) findViewById(R.id.right_edit_text);
 leftEditText.setText(String.valueOf(0));
 rightEditText.setText(String.valueOf(0));

 leftButton = (Button) findViewById(R.id.left_button);
 rightButton = (Button) findViewById(R.id.right_button);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
 getMenuInflater().inflate(R.menu.practical_test01, menu);
 return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
 int id = item.getItemId();
 if (id == R.id.action_settings) {
 return true;
 }
 return super.onOptionsItemSelected(item);
}
}

```

În mediul integrat de dezvoltare Android Studio, completarea automată a import-urilor necesare este realizată prin *Alt+Enter*.

**B.1.** Pentru procesarea evenimentelor legate de interacțiunea de utilizator este necesară:

- implementarea unei clase ascultător pentru tipul de eveniment respectiv (se recomandă să se folosească o clasă internă - în cadrul activității -, cu nume);
- înregistrarea unui obiect - instanță a clasei - ca ascultător al controlului grafic, pentru tipul de eveniment respectiv.

Se recomandă să se utilizeze aceeași clasă / același obiect pentru tratarea evenimentelor de același tip, chiar dacă acestea vor fi asociate unor controale grafice diferite. În cazul în care este necesar să se realizeze procesări specifice pentru obiecte de același tip, se va realiza distincția dintre acestea prin intermediul identificatorului controlului grafic care le este asociat.

Pe metoda `onCreate()` a activității se obțin referințe ale controlului grafic (prin metoda `findViewById()`) și se înregistrează obiectul ascultător respectiv.

În situația în care în interfața grafică există mai multe obiecte de același tip, având în vedere că operațiile ce trebuie realizate pentru fiecare în parte sunt aceleași, acestea se pot grupa prin procesarea iterativă unui tablou de identificatori (referințe) ale controalelor grafice respective.

```

public class PracticalTest01MainActivity extends Activity {

 private ButtonClickListener buttonClickListener = new ButtonClickListener();
 private class ButtonClickListener implements View.OnClickListener {
 @Override

```

```

 public void onClick(View view) {
 switch(view.getId()) {
 case R.id.left_button:
 int leftNumberOfClicks = Integer.parseInt(leftEditText.getText().toString());
 leftNumberOfClicks++;
 leftEditText.setText(String.valueOf(leftNumberOfClicks));
 break;
 case R.id.right_button:
 int rightNumberOfClicks = Integer.parseInt(rightEditText.getText().toString());
 rightNumberOfClicks++;
 rightEditText.setText(String.valueOf(rightNumberOfClicks));
 break;
 }
 }

 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);

 // ...

 leftButton.setOnClickListener(buttonClickListener);
 rightButton.setOnClickListener(buttonClickListener);
 }

 // ...
}

```

## B.2.

**a)** Pentru majoritatea controalelor grafice, sistemul de operare Android realizează în mod automat salvarea / restaurarea stării în situația în care o activitate este distrusă pentru asigurarea necesarului de memorie. Dacă se dorește ca această operație să fie tratată în alt mod, dezactivarea acestui mecanism se realizează prin specificarea atributului `android:saveEnabled` corespunzător controlului grafic respectiv având valoarea false.

```

<LinearLayout ... >
 <EditText
 android:id="@+id/left_edit_text"
 android:saveEnabled="false"
 ... />

 <EditText
 android:id="@+id/right_edit_text"
 android:saveEnabled="false"
 ... />

 <!-- other nested layouts or widgets -->
</LinearLayout>

```

**b)** În situația în care sistemul de operare Android distrugе activitatea pentru asigurarea resurselor necesare, este necesară asigurarea unui comportament consistent / persistenței aplicației:

- salvarea stării se realizează pe metoda `onSaveInstanceState()` care primește un parametru de tip `Bundle` în care vor fi stocate informațiile respective, identificarea lor realizându-se prin intermediul unei chei de tip sir de caractere; metoda este invocată înainte de `onStop()` deși pot exista situații în care aceasta este apelată chiar înainte de `onPause()`;
- restaurarea stării se realizează:
  - pe metoda `onCreate()` pentru care parametrul de tip `Bundle` este nul în situația în care există o stare anterioară;
  - pe metoda `onRestoreInstanceState()` care este invocată în mod automat - primind un parametru de tip `Bundle` numai în situația în care există o stare anterioară; metoda se apelează între metodele `onStart()` și `onRestore()`.

În parametrul de tip `Bundle` nu pot fi stocate decât obiecte care implementează `android.os.Parcelable`.

Salvarea stării implică plasarea în obiectul de tip `Bundle` a unei valori, prin metode de tip `put<type>()`, folosind un sir de caractere definit de utilizator (o convenție de nume). Restaurarea stării implică preluarea în obiectul de tip `Bundle` a valorii, prin metode de tip `get<type>()`, folosind sirul de caractere corespunzător (convenția de nume). Se recomandă să se verifice faptul că valoarea respectivă este nenulă (respectiv nu a fost furnizată valoarea implicită).

Se recomandă ca restaurarea stării să fie realizată pe metoda `onRestoreInstanceState()` în detrimentul metodei `onCreate()` pentru că în acest fel nu trebuie să se verifice starea obiectului de tip `Bundle`, iar procesările în urma cărora este creată o instanță a activității sunt mai rapide.

```
public class PracticalTest01MainActivity extends Activity {

 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);

 // ...

 if (savedInstanceState != null) {
 if (savedInstanceState.containsKey("leftCount")) {
 leftEditText.setText(savedInstanceState.getString("leftCount"));
 } else {
 leftEditText.setText(String.valueOf(0));
 }
 if (savedInstanceState.containsKey("rightCount")) {
 rightEditText.setText(savedInstanceState.getString("rightCount"));
 } else {
 rightEditText.setText(String.valueOf(0));
 }
 } else {

```

```

 leftEditText.setText(String.valueOf(0));
 rightEditText.setText(String.valueOf(0));
 }
}

@Override
protected void onSaveInstanceState(Bundle savedInstanceState) {
 savedInstanceState.putString("leftCount",
 leftEditText.getText().toString());
 savedInstanceState.putString("rightCount",
 rightEditText.getText().toString());
}

@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
 if (savedInstanceState.containsKey("leftCount")) {
 leftEditText.setText(savedInstanceState.getString("leftCount"));
 } else {
 leftEditText.setText(String.valueOf(0));
 }
 if (savedInstanceState.containsKey("rightCount")) {
 rightEditText.setText(savedInstanceState.getString("rightCount"));
 } else {
 rightEditText.setText(String.valueOf(0));
 }
}
// ...
}

```

c) Pentru a simula faptul că sistemul de operare Android distrugе activitatea pentru asigurarea necesarului de resurse, se poate proceda astfel:

1. se oprește activitatea prin accesarea butonului *Menu*, astfel încât activitatea să rămână în memorie, fără a fi însă vizibilă, asigurându-se totodată apelarea metodelor `onPause()` și `onStop()` și implicit a metodei `onSaveInstanceState()`;
2. în Android Studio, în secțiunea *Android Monitor*, se oprește aplicația Android prin intermediul butonului *Terminate*



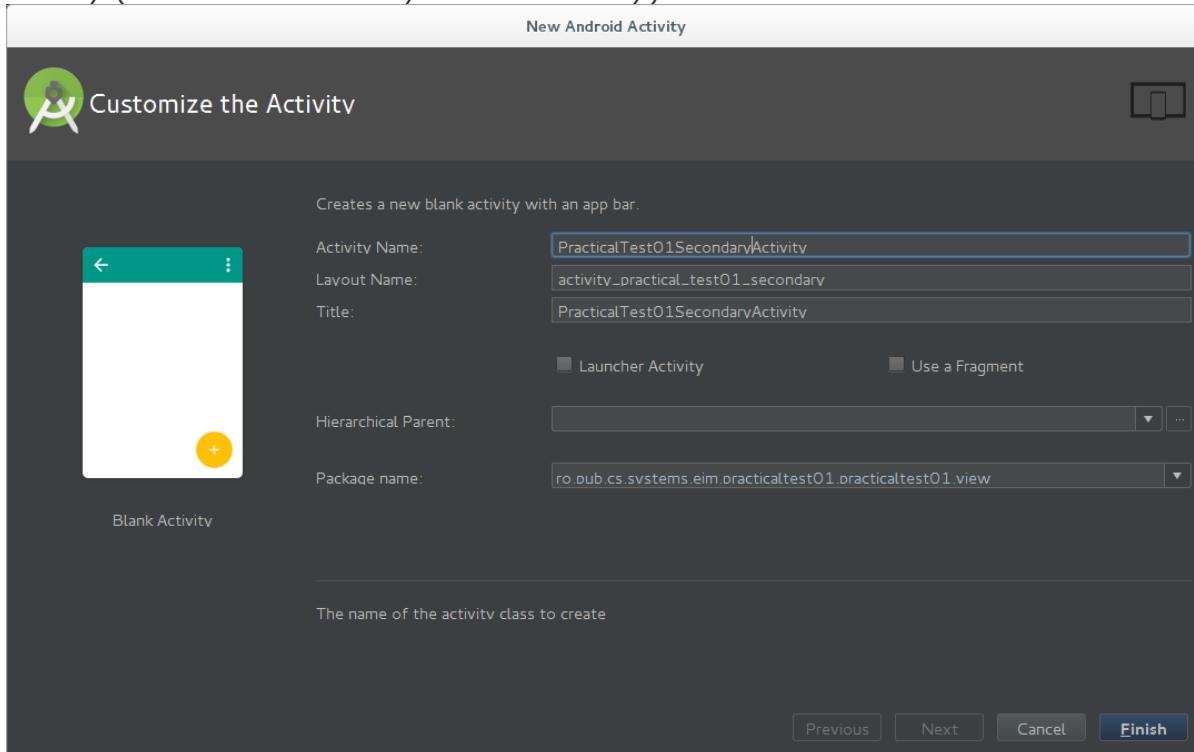
3. se (re)pornește activitatea din meniul dispozitivului mobil, astfel încât să se apeleze metodele `onCreate()`, respectiv `onStart()` și `onResume()` (și, implicit, `onRestoreInstanceState()`).

În mediul integrat de dezvoltare Android Studio, AMS (Android Device Monitor) este pornit din *Tools → Android → Android Device Monitor*.

**C.1.** Într-o aplicație Android, o activitate trebuie să fie precizată prin următoarele elemente:

- o etichetă de tip `<activity>` în fișierul `AndroidManifest.xml`, având asociată și un filtru de intenții (etichetă de tip `<intent-filter>`);
- o interfață grafică, definită în cadrul unui fișier `.xml` în directorul `/res/layout`;
- o clasă derivată din `Activity`, în care să se implementeze cel puțin metoda `onCreate()`, pe care să se încarce interfața grafică definită anterior.

În mediul integrat de dezvoltare Android Studio, crearea acestor resurse este realizată în mod automat în momentul în care se solicită definirea unei resurse de tip *Android Activity* (`File → New → Activity → Blank Activity`).



**a)** În fișierul `AndroidManifest.xml` se specifică activitatea printr-o element de tip `<activity>` în cadrul etichetei `<application>` pentru care se definesc:

- atributul `android:name` în care se indică denumirea clasei de tip `Activity` care va gestiona activitatea (afișarea interfeței grafice și tratarea interacțiunii cu utilizatorul);
- atributul `android:label` precizează o etichetă care va fi asociată activității;
- un filtru de intenții, prin care se controlează modul în care va putea fi lansată în execuție prin intermediul unei alte intenții:
  - eticheta `<action>` stabilește acțiunea pe care o poate realiza activitatea; va avea o valoare definită de utilizator, reprezentând o convenție;
  - eticheta `<category>` clasifică activitatea, impunându-se să fie folosită valoarea `android.intent.category.DEFAULT` pentru ca activitatea să poată fi invocată prin intermediul unei intenții.

În fișierul `AndroidManifest.xml` nu se completează în mod automat și filtrul de intenții, fiind necesar ca acesta să fie menționat de utilizator, manual.

### AndroidManifest.xml

```
<manifest ... >
 <application ... >
 <activity
 android:name=".PracticalTest01SecondaryActivity"
 android:label="@string/title_activity_practical_test01_secondary"
 >
 <intent-filter>
 <action
 android:name="ro.pub.cs.systems.eim.intent.action.PracticalTest01SecondaryActivity" />
 <category android:name="android.intent.category.DEFAULT" />
 </intent-filter>
 </activity>
 </application>
</manifest>
```

b) O interfață grafică este definită în mod obișnuit, prin intermediul unui fișier `.xml` plasat în directorul `/res/layout` în care sunt precizate controalele grafice cu atributele și valorile lor, sub formă de componente în cadrul unor mecanisme de dispunere.

### activity\_practical\_test01\_secondary.xml

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

 tools:context="ro.pub.cs.systems.eim.practicaltest01.PracticalTest01SecondaryActivity" >

 <TextView
 android:id="@+id/number_of_clicks_text_view"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:gravity="center" />

 <LinearLayout
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_gravity="center" >

 <Button
 android:id="@+id/ok_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@string/ok" />

 <Button
 android:id="@+id/cancel_button"
 android:layout_width="wrap_content"
```

```

 android:layout_height="wrap_content"
 android:text="@string/cancel" />

 </LinearLayout>

</LinearLayout>

```

c) În clasa care gestionează comportamentul activității, pe metoda `onCreate()`, în mod obișnuit, se încarcă interfața grafică (prin metoda `setContentView()`), se obțin referințe către controalele grafice (printr-un apel al metodei `findViewById()`) și se stabilește conținutul acestora, respectiv se înregistrează obiecte ascultător pentru anumite tipuri de evenimente (după ce în prealabil a fost implementată și clasa corespunzătoare).

Pentru activitățile care sunt invocate prin intermediul unei intenții:

- se poate obține o referință către intenție, din contextul căruia pot fi obținute informațiile atașate în secțiunea extra prin intermediul metodelor `getExtras()` sau `get<type>Extra()` care primesc ca parametru cheia sub care au fost stocate datele respective;
- se poate transmite un rezultat înapoi către activitatea din care au fost invocate prin intermediul metodei  `setResult()` care primește ca parametri:
  - un cod de rezultat întreg (frecvent, sunt folosite valorile `RESULT_OK` și `RESULT_CANCELED`, dar poate fi folosită și o valoare definită de utilizator);
  - o intenție, în care pot fi plasate informații suplimentare.

Metoda `finish()` poate fi folosită pentru a finaliza activitatea în mod forțat (semnalând faptul că procesările corespunzătoare activității au fost terminate).

### PracticalTest01SecondaryActivity.java

```

package ro.pub.cs.systems.eim.practicaltest01;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class PracticalTest01SecondaryActivity extends Activity {

 private TextView numberOfClicksTextView = null;
 private Button okButton = null;
 private Button cancelButton = null;

 private ButtonClickListener buttonClickListener = new
 ButtonClickListener();
 private class ButtonClickListener implements View.OnClickListener {
 @Override
 public void onClick(View view) {
 switch(view.getId()) {

```

```

 case R.id.ok_button:
 setResult(RESULT_OK, null);
 break;
 case R.id.cancel_button:
 setResult(RESULT_CANCELED, null);
 break;
 }
 finish();
}

@Override
protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_practical_test01_secondary);

 numberOfClicksTextView
 (TextView) findViewById(R.id.number_of_clicks_text_view);
 Intent intent = getIntent();
 if (intent != null && intent.getExtras().containsKey("numberOfClicks")) {
 int numberOfClicks = intent.getIntExtra("numberOfClicks", -1);
 numberOfClicksTextView.setText(String.valueOf(numberOfClicks));
 }

 okButton = (Button) findViewById(R.id.ok_button);
 okButton.setOnClickListener(buttonClickListener);
 cancelButton = (Button) findViewById(R.id.cancel_button);
 cancelButton.setOnClickListener(buttonClickListener);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
 getMenuInflater().inflate(R.menu.practical_test01_secondary, menu);
 return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
 int id = item.getItemId();
 if (id == R.id.action_settings) {
 return true;
 }
 return super.onOptionsItemSelected(item);
}
}

```

**C.2.** Pentru a invoca o altă activitate prin intermediul unei intenții, o activitate trebuie:

- să definească în interfața grafică un control prin intermediul căruia va fi lansată în execuție activitatea respectivă (în fișierul .xml din directorul /res/layout);

```

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

```

```

tools:context="ro.pub.cs.systems.eim.practicaltest01.PracticalTest01MainActivity" >

<Button
 android:id="@+id/navigate_to_secondary_activity_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_gravity="right"
 android:text="@string/navigate_to_secondary_activity" />

<!-- other nested layouts or widgets -->

</LinearLayout>

```

- în codul sursă corespunzător clasei care gestionează activitatea respectivă:
  1. pe metoda `onCreate()`:
    - a. să obțină o referință către controlul grafic prin intermediul căruia va fi lansată în execuție o altă activitate;
    - b. să înregistreze un obiect ascultător care va trata evenimentul legat de controlul grafic respectiv;
  2. să definească o clasă ascultător (internă, cu nume) pentru tipul de eveniment respectiv, iar pe metoda de tratare se vor realiza următoarele operații:
    - a. se va instanția un obiect de tip `Intent`, care poate primi ca parametru:
      - I. contextul curent (obiectul `this`) și clasa care deservește activitatea (obiect de tip `.class`) - adekvată situației în care activitatea invocată se găsește în aceeași aplicație ca și activitatea care invocă;
      - II. acțiunea asociată activității în filtrul de intenții - adekvată situației în care activitatea invocată se găsește în altă aplicație decât activitatea care o invocă;
    - b. se asociază un set de informații care vor fi transmise celeilalte activități (în secțiunea `extra` a intenției) prin intermediul metodei `putExtra()`, fiecare valoare (de tip `android.os.Parcelable`) având asociat un atribut prin care va fi identificat ulterior, de tip sir de caractere;
    - c. se lansează în execuție activitatea prin intermediul metodei `startActivityForResult()` care primește ca parametru intenția și un cod de cerere, având rolul de a identifica ulterior instanța activității din care se revine.
  3. să implementeze metoda `onActivityResult()`, apelată în mod automat la revenirea din activitatea invocată, aceasta primind ca parametrii:

- a. codul de cerere trimis în momentul în care activitatea a fost lansată în execuție;
- b. rezultatul furnizat de activitatea invocată;
- c. intenția generată de activitatea invocată.

```

public class PracticalTest01MainActivity extends Activity {

 private final static int SECONDARY_ACTIVITY_REQUEST_CODE = 1;

 private Button navigateToSecondaryActivityButton = null;

 private ButtonClickListener buttonClickListener = new ButtonClickListener();
 private class ButtonClickListener implements Button.OnClickListener {

 @Override
 public void onClick(View view) {
 switch(view.getId()) {
 case R.id.navigate_to_secondary_activity_button:
 Intent intent = new Intent(getApplicationContext(),
PracticalTest01SecondaryActivity.class);
 int numberOfClicks = Integer.parseInt(leftEditText.getText().toString()) +
Integer.parseInt(rightEditText.getText().toString());
 intent.putExtra("numberOfClicks", numberOfClicks);
 startActivityForResult(intent, SECONDARY_ACTIVITY_REQUEST_CODE);
 break;
 // ...
 }
 }
 }

 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 // ...

 navigateToSecondaryActivityButton =
(Button)findViewById(R.id.navigate_to_secondary_activity_button);

 navigateToSecondaryActivityButton.setOnClickListener(buttonClickListener);
 }

 @Override
 protected void onActivityResult(int requestCode, int resultCode, Intent intent) {
 if(requestCode == SECONDARY_ACTIVITY_REQUEST_CODE) {
 Toast.makeText(this, "The activity returned with result " + resultCode,
Toast.LENGTH_LONG).show();
 }
 }
}
// ...

```

```
}
```

**D.1. a)** Pentru a putea fi utilizat, serviciul trebuie să fie declarat în fișierul `AndroidManifest.xml`. Este obligatoriu să se definească proprietatea `android:name` care desemnează clasa care gestionează serviciul. Opțional, pot fi precizate și atributele:

- `android:enabled`, care indică posibilitatea ca sistemul de operare Android să instanțieze serviciul;
- `android:exported`, prin care se oferă posibilitatea ca serviciul să poată fi invocat și din contextul altei aplicații Android decât cea din care face parte acesta;
- `android:permission`, care limitează posibilitatea ca serviciul respectiv să fie lansat în execuție exclusiv de componentele care dețin permisiunea specificată.

#### AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
 <application ...>
 <!-- other components -->
 <service
 android:name="ro.pub.cs.systems.eim.PracticalTest01Service"
 android:enabled="true"
 android:exported="false" />
 </application>
</manifest>
```

Clasa Java care gestionează serviciul trebuie să fie derivată din clasa `android.app.Service`. Fiind un serviciu de tip `started`, trebuie suprăîncărcate metodele de callback:

- `onStartCommand(intent, flags, id)`, apelat în mod automat în momentul în care componenta Android lansează serviciul în execuție printr-o invocare a metodei `startService()`;
- `onBind(intent)` care va furniza un rezultat `null`, întrucât nu este necesară o interfață prin care componenta Android să interacționeze cu serviciul.

#### PracticalTest01Service.java

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

public class PracticalTest01Service extends Service {

 private ProcessingThread processingThread = null;

 @Override
 public int onStartCommand(Intent intent, int flags, int startId) {
 int firstNumber = intent.getIntExtra("firstNumber", -1);
 int secondNumber = intent.getIntExtra("secondNumber", -1);
```

```

 processingThread = new ProcessingThread(this, firstNumber,
secondNumber);
 processingThread.start();
 return Service.START_REDELIVER_INTENT;
 }

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

 @Override
 public void onDestroy() {
 processingThread.stopThread();
 }

}

```

Întrucât procesarea realizată în cadrul serviciului ar ocupa timp de procesor, putând avea un impact asupra responsivității aplicației Android în privința interacțiunii cu utilizatorul, operațiile respective vor fi realizate pe un fir de execuție dedicat. Argumentele care vor fi transmise firului de execuție dedicat sunt **contextul**, întrucât pe baza acestuia se vor transmite intențiile cu difuzare, precum și cele două numere din cadrul interfeței grafice, transmise prin intermediul intenției, ca valori asociate cheilor `firstNumber`, respectiv `secondNumber`. Valoarea pe care o furnizează metoda `onStartCommand()` este `Service.START_REDELIVER_INTENT`, ceea ce indică sistemului de operare Android faptul că serviciul trebuie repornit în situația în care este distrus pentru a se recupera resursele pe care acesta le utilizează, în caz de nevoie. De asemenea, va fi retransmisă și intenția utilizată pentru a invoca serviciul respectiv.

### ProcessingThread.java

```

import java.util.Date;
import java.util.Random;

import android.content.Context;
import android.content.Intent;
import android.util.Log;

public class ProcessingThread extends Thread {

 private Context context = null;
 private boolean isRunning = true;

 private Random random = new Random();

 private double arithmeticMean;
 private double geometricMean;

 public ProcessingThread(Context context, int firstNumber, int
secondNumber) {
 this.context = context;

 arithmeticMean = (firstNumber + secondNumber) / 2;
 geometricMean = Math.sqrt(firstNumber * secondNumber);
 }
}

```

```

@Override
public void run() {
 Log.d("[ProcessingThread]", "Thread has started!");
 while (isRunning) {
 sendMessage();
 sleep();
 }
 Log.d("[ProcessingThread]", "Thread has stopped!");
}

private void sendMessage() {
 Intent intent = new Intent();

 intent.setAction(Constants.actionTypes[random.nextInt(Constants.actionTypes.length)]);
 intent.putExtra("message", new Date(System.currentTimeMillis()) + " " +
 arithmeticMean + " " + geometricMean);
 context.sendBroadcast(intent);
}

private void sleep() {
 try {
 Thread.sleep(10000);
 } catch (InterruptedException interruptedException) {
 interruptedException.printStackTrace();
 }
}

public void stopThread() {
 isRunning = false;
}
}

```

În contextul firului de execuție dedicat, se calculează inițial valorile necesare (media aritmetică și media geometrică).

Rutina asociată firului de execuție implică următoarele operații:

1. propagarea unei intenții cu difuzare la nivelul sistemului de operare Android;
  - a. se creează un obiect de tip `Intent` care reprezintă intenția cu difuzare care va fi propagată;
    - I. acțiunea asociată intenției este aleasă în mod aleator dintr-o colecție de tipuri de intenții definite de utilizator;
    - II. informația conținută de intenția cu difuzare este plasată în câmpul extra, valoarea (șirul de caractere obținut prin concatenarea datei și orei curente cu media aritmetică și cu media geometrică) fiind asociată cheii `message`;
  - b. se transmite intenția cu difuzare la nivelul sistemului de operare Android prin invocarea metodei `sendBroadcast()` (definită în clasa `Context`);

2. așteptarea unui interval de timp (10.000 ms) până la următoarea iterație, prin intermediul unui apel `Thread.sleep()`.

Trebuie implementată și o metodă pentru oprirea rulării firului de execuție în situația în care serviciul asociat este distrus.

**b)** În clasa `PracticalTest01Activity`, sunt plasate operațiile de pornire și de oprire a serviciului.

Verificarea momentului în care trebuie pornit serviciul este realizată pe metoda ascultător a evenimentului de apăsare a unui buton. Se realizează suma dintre valorile conținute în cele două câmpuri text editabile și, în situația în care aceasta depășește pragul minim stabilit și dacă serviciul se găsește în starea oprit, atunci poate fi invocată metoda `startService()`. Aceasta primește ca argument un obiect de tip `Intent` (explicit, instantiat cu contextul și cu clasa aferentă serviciului) în care sunt plasate, în câmpul `extra`, valorile din câmpurile text editabile sub cheile `firstNumber` și `secondNumber`.

```
private class ButtonClickListener implements View.OnClickListener {
 @Override
 public void onClick(View view) {
 int leftNumberOfClicks = Integer.parseInt(leftEditText.getText().toString());
 int rightNumberOfClicks = Integer.parseInt(rightEditText.getText().toString());
 // ...
 if (leftNumberOfClicks + rightNumberOfClicks > Constants.NUMBER_OF_CLICKS_THRESHOLD
 && serviceStatus == Constants.SERVICE_STOPPED) {
 Intent intent = new Intent(getApplicationContext(),
 PracticalTest01Service.class);
 intent.putExtra("firstNumber", leftNumberOfClicks);
 intent.putExtra("secondNumber", rightNumberOfClicks);
 getApplicationContext().startService(intent);
 serviceStatus = Constants.SERVICE_STARTED;
 }
 }
}
```

Momentul la care este oprit serviciul coincide cu distrugerea activității, așadar pe metoda de callback `onDestroy()`. Similar, această operație poate fi realizată și pe metoda de callback `onStop()`. Metoda `stopService()` primește ca argument tot un obiect de tip `Intent` construit explicit, cu numele clasei care deservește serviciul.

```
@Override
protected void onDestroy() {
 Intent intent = new Intent(this, PracticalTest01Service.class);
 stopService(intent);
 super.onDestroy();
}
```

Dacă serviciul este oprit, el va fi distrus de sistemul de operare Android (deși `stopService()` nu implică invocarea unei metode de callback la nivelul serviciului, totuși acesta va fi oprit și ulterior distrus). Pe metoda `onDestroy()` a serviciului, trebuie să se aibă grija ca rutina din cadrul firului de execuție asociat să se termine.

**D.2.** Un ascultător pentru intenții cu difuzare este o clasă derivată din `android.content.BroadcastReceiver`. Acesta poate fi declarat în fișierul `AndroidManifest.xml`, dacă se dorește ca acesta să fie invocat în mod automat chiar dacă aplicația Android din care face parte nu este instantiată sau poate fi înregistrat / deînregistrat pe metodele de callback `onResume()` / `onPause()` dacă se dorește ca funcționarea să fie limitată la perioada în care activitatea este disponibilă pe suprafața de afișare a dispozitivului mobil.

În momentul în care o intenție cu difuzare are aceeași acțiune ca cea indicată în filtrul de intenții asociat ascultătorului respectiv, este apelată în mod automat metoda de callback `onReceive(context, intent)`. Aceasta va fi responsabilă exclusiv cu jurnalizarea valorii asociate cheii `message` din câmpul `extra` asociat intenției cu difuzare respective.

```
private MessageBroadcastReceiver messageBroadcastReceiver = new MessageBroadcastReceiver();
private class MessageBroadcastReceiver extends BroadcastReceiver {
 @Override
 public void onReceive(Context context, Intent intent) {
 Log.d("[Message]", intent.getStringExtra("message"));
 }
}
```

Un ascultător pentru intenții cu difuzare trebuie să aibă asociat și un filtru de intenții. Acesta conține acțiunile asociate intenților cu difuzare, pe care ascultătorul respectiv trebuie să le proceseze.

```
public class PracticalTest01MainActivity extends Activity {
 // ...

 private IntentFilter intentFilter = new IntentFilter();

 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_practical_test01_main);

 for (int index = 0; index < Constants.actionTypes.length; index++) {
 intentFilter.addAction(Constants.actionTypes[index]);
 }
 }
}
```

Operațiile de activare, respectiv de dezactivare a ascultătorului pentru intenții cu difuzare, sunt realizate pe metodele de callback ale activității `onResume()`, respectiv `onPause()`. În acest fel, se asigură faptul că ascultătorul pentru intenții cu difuzare nu va procesa decât mesajele transmise în perioada în care activitatea este afișată pe suprafața de afișare a dispozitivului mobil.

```
public class PracticalTest01MainActivity extends Activity {
 // ...

 @Override
 protected void onResume() {
 super.onResume();
 registerReceiver(messageBroadcastReceiver, intentFilter);
 }
}
```

```

 }

 @Override
 protected void onPause() {
 unregisterReceiver(messageBroadcastReceiver);
 super.onPause();
 }
}

```

**D.3.** Pentru încărcarea codului în contextul depozitului din cadrul contului Github personal:

1. se transferă modificările din zona de lucru în zona de lucru în zona de aşteptare prin intermediul comenzi git add, indicându-se și fișierele respective (pot fi folosite şablonane pentru a desemna mai multe fișiere);
2. se consemnează modificările din zona de aşteptare în directorul Git prin intermediul comenzi git commit -m, precizându-se și un mesaj sugestiv;
3. se încarcă modificările la distanță, prin intermediul comenzi git push, care primește ca parametrii:
  - a. sursa (prin eticheta origin se indică depozitul de unde au fost descărcat conținutul);
  - b. destinația: ramificația curentă (implicit, aceasta poartă denumirea master).

```

student@eim2017:~/PracticalTest01$ git add *

student@eim2017:~/PracticalTest01$ git commit -m "finished tasks for PracticalTest01"

student@eim2017:~/PracticalTest01$ git push origin master

```

În cazul în care este necesar, vor fi configurați parametrii necesari operației de consemnare (numele de utilizator și adresa de poștă electronică):

```

student@eim2017:~/PracticalTest01$ git config --global user.name "Perfect Student"

student@eim2017:~/PracticalTest01$ git config --global user.email perfectstudent@cs.pub.ro

```