4. Fiecare aplicaţie trebuie să aibă un fişier AndroidManifest.xml în directorul rădăcină al aplicaţiei.
În acest fişier sunt descrise componentele aplicaţiei şi resursele de care aceasta are nevoie ca să ruleze.

- În primul rând include numele aplicaţiei şi al pachetul Java. Numele pachetului este considerat un identificator unic în sistem. Nu se pot instala două aplicaţii cu acelaşi nume de pachet. De exemplu, dacă avem o aplicaţie cu numele de pachet com.myapp instalată de telefon şi încercăm să instalăm o alta aplicaţie cu acelaşi nume de pachet, vom primi o eroare de instalare.
- De asemenea, descrie toate componentele aplicaţiei - activităţi, servicii, broadcast receivers, content providers. Pentru fiecare, se specifică clasa şi capabilităţile (ce Intent-uri pot primi).
- Aici este specificată şi activitatea principală, cea care va porni atunci când dăm click pe iconiţa din home screen.
- Sunt declarate permisiunile de care aplicaţia are nevoie pentru a apela anumite părţi protejate din API sau a interacţiona cu alte aplicaţii.
- De asemenea sunt specificate permisiunile de care are nevoie altă aplicaţie pentru a interacţiona cu aplicaţia noastră.
- Sunt specificate bibliotecile cu care va fi link-ată aplicaţia.
- Nu în ultimul rând, se specifică nivelul de API minim şi ţintă. De exemplu, dacă vrem ca aplicaţia noastră să fie compilată pentru Android 7, dar vrem să fie compatibilă şi cu Android 5, vom folosi target API 24 şi minimum API 21. Fiecare versiune de Android are asociat un anume nivel de API.

5.

- Fiecare aplicaţie are nevoie de un set de permisiuni pentru a efectua anumite operaţii/apeluri de API sau pentru a avea acces la anumite date/resurse.
- Acest mecanism de securitate oferă protecţie prin sandboxing. Asta înseamnă că aplicaţiile trebuie să declare capabilităţile de care au nevoie ca să funcţioneze. Astfel, o aplicaţie nu poate efectua anumite operaţii fără să aibă permisiunile necesare.
- Permisiunile sunt specificate în fişierul Manifest folosind tag-ul <uses-permission>. Aici aveţi un exemplu, dacă o aplicaţie vrea să acceseze Internetul atunci va avea nevoie de permisiunea Internet. Astfel aplicaţia va putea folosi WiFi sau LTE pentru a accesa resurse din Internet.
- De asemenea, putem controla cine accesează componentele aplicaţiei noastre, pentru a porni o activitate, a porni şi a face bind pe un serviciu, pentru a ne trimite un mesaj de broadcast, pentru a accesa datele din content provider.
- Aveţi aici un exemplu de permisiune necesară pentru a porni o activitate. Aceasta este o permisiune custom, creată de dezvoltator.
- Permisiunile standard nu sunt suficiente pentru lucrul cu content providerii. De aceea putem avea permisiuni specifice (read, write) per URI-uri.

6. În directorul res/ sunt organizate resursele de genul imagini, string-uri şi layout-uri.
Fiecare tip de resursă se află într-un subdirector cu nume specific, de exemplu drawable, layout, values, menu, xml, etc. În aceste subdirectoare avem resursele default.
Drawables sunt imagini, layouts sunt fişiere xml care descriu cum elementele UI sunt plasate pe ecran.
Values includ string-uri, menus includ descrieri ale meniurilor aplicaţiei, xml includ alte fişiere xml folosite de aplicaţie.

Diferite tipuri de dispozitive Android pot avea nevoie de resurse diferite.

- De exemplu, pe un dispozitiv cu un ecran mai mare (tabletă) vom face un layout diferit pentru a profita de spaţiul disponibil.
- Pe un dispozitiv cu o setare de limbă diferită, putem avea alte valori pentru string-uri.
- La rulare, aplicaţia va folosi resursele asociate cu o anumită configuraţie (de exemplu limba română şi ecran hdpi).
- Pentru a oferi resurse alternative, vom avea subdirectoare pentru fiecare configurare alternativă.
- Numele subdirectorului va fi nume resursa - nume configuraţie (de exemplu, drawable-hdpi pentru highdensity screens - 240dpi).

Pentru fiecare nume de resursă se va genera un id unic în directorul gen/.

7. Acesta este un exemplu de folosire a layout-urilor diferite pentru dimensiuni de ecran diferite, pentru a folosi mai bine spaţiul disponibil.

8.

- Layout este o resursă inclusă în res/layouts care descrie interfaţa cu utilizatorul pentru o activitate sau o parte a interfeţei.
- Practic un layout va conţine elemente de UI: butoane, liste, câmpuri de text, etc.
- Un layout este inclus într-un fişier de tipul res/layout/filename.xml, iar filename va fi ID-ul resursei respective. Veţi putea să vă referiţi la acel layout în cod folosind R.layout.filename. R.java este generat la compilare, poate fi găsit în folderul gen/, şi include toate ID-urile resurselor. Nu este permis sa editaţi fişierul R de mână deoarece este generat automat pe baza resurselor.
- Fişierele xml conţinând layout-uri pot fi editate direct sau cu alte tool-uri, de exemplu Eclipse sau Android Studio.
- E uşor să faci configuraţii simple, să adaugi un buton, sau alt element de UI, dar e mai greu să fii expert în asta, deoarece sunt multe opţiuni şi configuraţii care pot fi făcute.

9. Acesta este un exemplu de layout pentru o activitate. Avem acest fişier XML: res/layout/main_activity.xml. Conţine un linear layout care include un câmp de text şi un buton. Jos aveţi codul care aplică acest layout activităţii.

10.

- Drawables sunt resurse din res/drawables, elemente ce pot fi afişate pe ecran. Acestea pot fi imagini sau fişiere xml.
- Fişierele xml sunt folosite pentru a descrie cum va arăta un element de UI atunci când se acţionează asupra lui (de exemplu când este apăsat). Aceste xml-uri se referă la imagini, de exemplu putem avea o imagine şi când este apăsată să fie înlocuită cu altă imagine.
- Acest lucru îmbunătăţeşte calitatea experienţei utilizatorului prin feedback-ul vizual atunci când utilizatorul interacţionează cu interfaţa grafică.

12.

- O activitate este o componentă a aplicaţiei care reprezintă o fereastră în care este descrisă interfaţa cu utilizatorul. Acesta va interacţiona direct doar cu activitatea.
- Pentru ca o activitate să fie afişată avem nevoie de un layout care să descrie elementele de UI şi aşezarea lor pe ecran.
- Interfaţa grafică poate fi schimbată doar din thread-ul looper al activităţii. De aceea este bine să nu facem procesări intensiv computaţionale sau blocante în acest thread. Este de preferat să se facă aceste operaţii în thread-uri diferite. O regulă importantă este să nu accesăm reţeaua în thread-ul de UI, trebuie creat un thread separat pentru operaţiile cu reţeaua.
- O aplicaţie poate include mai multe activităţi din care doar una este cea principală care va fi pornită atunci când dăm click pe iconiţa din launcher (home screen).
- Putem porni o activitate din altă activitate, cea veche va fi pusă pe stop iar cea nouă va fi pornită. Astfel avem o stivă de activităţi numită şi backstack.
- Atunci când apăsăm pe back, activitatea curentă va fi distrusă şi cea precedentă va fi resumed.

13.

- Pentru a folosi o nouă activitate în aplicaţie, trebuie sa o declaraţi în fişierul manifest. Mai exact, trebuie adăugat un element activity în cadrul elementului application.
- Elementul activity poate să includă multe atribute, dar android:name este singurul obligatoriu. Acesta este şi numele clasei care implementează funcţionalitatea activităţii.

14.

- Pe acest slide avem o reprezentare detaliată a ciclului de viată al unei activităţi. Metodele pe care le vedem aici sunt callback-uri şi sunt apelate de către sistem (de ActivityManager), nu de către aplicaţia în sine.
- Când o activitate este pornită, se apelează întâi onCreate(), care este punctul de intrare în activitate, apoi activitatea intră în starea Created. În acest moment, activitatea îşi obţine layout-ul, dar nu îl desenează pe ecran.
- Apoi onStart() este apelat, care va desena layout-ul pe ecran şi se activitatea va intra în starea Started, în care este vizibilă pe ecran.
- Apoi onResume() este apelat, şi activitatea trece în starea Resumed, în care este vizibilă şi acceptă input-ul utilizatorului.
- Atunci când primim o notificare sau apare un dialog box, on Pause() este apelat, iar activitatea va trece în starea Paused, în care este parţial vizibilă şi utilizatorul nu poate interacţiona cu ea. Dacă închidem notificarea/dialogul, onResume() este apelat şi activitatea intră în starea Resumed. Layout-ul nu este redesenat, activitatea are focus din nou şi primeşte input-ul utilizatorului.
- Atunci când din activitatea A pornim activitatea B, se va apela onPause() şi onStop() din A, iar activitatea A va rămâne în starea Stopped în backstack în timp ce activitatea B îşi începe ciclul de viaţă. Atunci când activitatea este în foreground şi apăsăm back, această activitate va fi distrusă şi apoi se vor apela onRestart(), onStart() şi onResume() pentru activitatea A. De ce trebuie să trecem din nou prin starea Started? Deoarece este posibil să fie nevoie re-desenarea activităţii.
- Atunci când suntem în activitatea A şi se apasă back, se vor apela onPause(), onStop() şi onDestroy(), pentru a distruge acea activitate şi a reveni la cea anterioară.

- În starea Stopped/Paused activitatea poate fi omorâtă de Low Memory Killer pentru că altă aplicaţie cu prioritate mai mare are nevoie de memorie. În acest caz, atunci când utilizatorul va intra din nou în activitate, se va apela onCreate(), onStart(), onResume(), şi activitatea este re-creată de la zero.

15-16. Acesta este scheletul de cod al unei activităţi care include toate metodele ciclului de viaţă.

17.

- Activităţile pot fi omorâte de către sistem dacă este nevoie de memorie pentru alte aplicaţii mai prioritare.
- În acel moment se pierde starea activităţii. Avem posibilitatea de a salva această stare, incluzând primitive, obiecte, UI, prin callback-ul onSaveInstanceState(). În acest callback trebuie să salvăm toate informaţiile într-un singur Bundle. Starea activităţii va putea fi restaurată în onCreate sau în onRestoreInstanceState() care primesc ca argument Bundle-ul respectiv.
- Totuşi, este recomandat să salvam starea activităţii doar atunci când este foarte important să facem asta, pentru a nu ocupa memoria inutil.
- Atunci când se omoară procesul se vor omorî şi thread-urile din activitate. E bine ca acestea să fie oprite înainte ca procesul să fie omorât mai ales dacă se fac operaţii critice de genul scris în fişiere. De aceea este recomandat ca în onPause() să semnalăm thread-urilor să se oprească.

18. Aici avem procesul de salvare a stării şi restaurare. Observăm ca nu trebuie să facem această restaurare dacă procesul nu este omorât pentru că starea activităţii se păstrează în Paused şi Stopped.

19.

- Fragmentele sunt elemente de UI conţinute într-o activitate. Sunt ca un fel de activităţi mai mici în cadrul unei activităţi, şi au propriul ciclu de viaţă.
- Ele pot fi combinate în diferite moduri pentru a construi UI-ul în funcţie de layout. De exemplu, pe un telefon şi o tabletă, pot fi re-aranjate în moduri diferite, dar codul fragmentului rămâne acelaşi. De exemplu, dacă suntem în mod landscape pe o tabletă vom avea o aranjare diferită a fragmentelor decât în mod portret pe un telefon.
- În plus, fragmentele pot fi refolosite şi în alte activităţi.

20. Aveţi aici un exemplu cu două fragmente afişate pe dispozitive diferite. Prin selectarea unui element din fragmentul A, se actualizează fragmentul B. Pe o tabletă putem combina cele două fragmente în activitatea A, iar pe un telefon nu va fi suficient spaţiu pentru ambele fragmente, deci activitatea A va conține fragmentul A și activitatea B va conține fragmentul B.

21.

- Ciclul de viaţă al unui fragment este direct afectat de ciclul de viaţă al activităţii părinte. Când activitatea este pe pauza, fragmentele vor fi pe pauza. Când activitatea este distrusă, fragmentele vor fi distruse.

- Atunci când o activitate este în starea Resumed, se pot adăuga şi scoate fragmente. Aceste operaţii se numesc tranzacţii cu fragmente.
- Există un backstack pentru fragmente asociat cu activitatea. Fiecare intrare în backstack conţine câte o tranzacţie efectuată. Atunci când apăsăm back, se va anula ultima tranzacţie.
- Un fragment nu va primi layout-ul în onCreate(), ci în callback-ul onCreateView(). Această metodă este apelată atunci când fragmentul ştie la ce activitate este ataşată şi cât ocupă în cadrul acelei activităţi. Iar distrugerea va avea loc în onDestroyView(), aici vor fi oprite thread-urile.

22.

- Interfaţa cu utilizatorul este realizată printr-o ierarhie de view-uri (derivate din clasa View: Button, TextView, Checkbox, etc).
- Fiecare View controlează un spaţiu dreptunghiular în activitate şi oferă interacţiune cu utilizatorul.
- Exemple de View-uri: butoane, liste, imagini, căsuţe de text, etc.
- Pentru interacţiunea cu aceste obiecte există callback-uri în care putem specifica ce acţiuni să se execute, de exemplu dacă este apăsat un buton (se apelează callback-ul onClick()).
- Un ViewGroup va conţine mai multe obiecte View şi alte ViewGroup-uri. Pentru a crea View-uri mai complexe se pot extinde clasele actuale.
- De asemenea, se pot folosi adaptori pentru a afişa tipuri de date complexe. Exemple de adaptori: ArrayAdapter, ListAdapter, CursorAdapter.

24.

- Un serviciu este o componentă a unei aplicaţii care poate executa operaţii care durează mult, în background, şi care nu oferă interfaţă cu utilizatorul.
- El va continua să ruleze chiar dacă utilizatorul a deschis o altă aplicaţie care apare în foreground. De exemplu, o aplicaţie de muzică ar trebui să poată reda muzică chiar şi atunci când aplicaţia este în background.
- Un serviciu poate face operaţii cu reţeaua, operaţii I/O pe fişiere, poate interacţiona cu un content provider, etc.
- Un serviciu va rula în thread-ul principal al procesului - nu va crea un nou thread şi nu va rula în alt proces decât dacă specificăm în mod explicit.
- Prin urmare, dacă vrem să facem operaţii CPU intensive sau operaţii blocante, e bine să creăm un nou thread pentru serviciu.
- Un serviciu poate fi pornit folosind un Intent din aplicaţia curentă sau din altă aplicaţie.
- Dacă vrem să blocam accesul altor aplicaţii la serviciul nostru, trebuie sa-l declarăm privat în fişierul Manifest.

25.

- Serviciile trebuie declarate în fişierul Manifest. Aici avem un exemplu: se adaugă un element service în cadrul elementului application.
- Se pot folosi multe atribute, de exemplu permisiunea de care are nevoie alta aplicaţie pentru a porni acest serviciu. De asemenea se poate configura serviciul să ruleze într-un proces separat.

Atributul android:name este singurul obligatoriu. Putem face serviciul privat folosind atributul android:exported false.

26.

- Există două tipuri de servicii: Started şi Bound.
- Un serviciu este Started atunci când o componentă a unei aplicaţii apelează startService(). El va rula indefinit, chiar dacă componenta care l-a pornit este distrusă.
    - Un astfel de serviciu face o anumită operaţie apoi se opreşte, fără a returna vreun rezultat spre componenta care l-a pornit.
- Un serviciu este Bound atunci când o componentă se leagă la el folosind funcţia bindService().
    - Un astfel de serviciu va oferi o interfaţă client-server în care se vor putea trimite cereri şi primi răspunsuri.
    - AIDL va putea fi folosit pentru a genera interfaţa folosită între client şi server.
    - Acest serviciu va rula doar până când componenta care l-a pornit face unbind.
    - Avem aici un corner case: dacă faci bind la un serviciu dintr-o activitate, şi activitatea este pusa în Stopped, atunci când te întorci la acea activitate serviciul poate fi null pentru că sistemul a avut nevoie de memorie şi a omorât serviciul. De aceea este recomandat să verificăm dacă serviciul este null pentru a evita NullPointerException.
    - Putem avea mai multe componente care fac bind pe un serviciu. În acest caz, serviciul va fi distrus abia după ce toate fac unbind.
- Un serviciu poate fi Started şi Bound în acelaşi timp.

27.

- Pe acest slide avem reprezentat ciclul de viaţă al unui serviciu în cele două cazuri.
- În primul caz, când o componentă apelează startService(), se rulează onCreate(), apoi onStartCommand(), apoi serviciul rulează.
- După aceea, serviciul se poate opri cu stopSelf() sau poate fi oprit de către altă componentă cu stopService(). Înainte să fie omorât serviciul se va apela onDestroy() - aici ar trebui să aibă loc clean-up-ul.
- În al doilea caz, un serviciu este pornit cu bindService(), se rulează onCreate(), onBind(), iar apoi clientul este legat la serviciu şi poate comunica cu el. După ce toţi clienţii fac unbind, se apelează onUnbind() şi apoi onDestroy().
- Să nu uităm că un serviciu poate fi Started şi Bound în acelaşi timp. Chiar dacă serviciul a fost pornit folosind startService(), poate primi apeluri la onBind() (atunci când clienţii apelează bindService()).

28-29. Acesta este scheletul de cod pentru un serviciu, incluzând toate callback-urile.

31.

- Un intent este folosit pentru a trimite un mesaj sau a determina execuţia unei acţiuni.
- Un intent are trei componente principale: numele componentei destinaţie, acţiunea care trebuie efectuată şi datele folosite în acţiune (de exemplu, sună la un anumit număr de telefon).

- Pentru a specifica ce intenturi acceptă o anumită componentă, se va declara un intent filter în Manifest, iar acţiunea şi tipul de date se vor specifica în tag-urile <action> şi <data> incluse în intent filter.

32.

- Un intent poate fi folosit în următoarele cazuri: pentru a porni o activitate, pentru a porni sau a face bind pe un serviciu, şi pentru livrarea unui mesaj de broadcast.
- O activitate poate fi pornită prin folosirea unui intent la apelarea metodei startActivity(). Intentul include informaţii despre activitatea ce trebuie pornită şi datele necesare.
- De asemenea, o activitate poate fi pornită prin pasarea unui intent la apelarea metodei startActivityForResult. Rezultatul va fi primit printr-un intent separat.
- Un serviciu poate fi pornit prin pasarea unui intent la apelarea metodei startService. Intentul include informaţiile necesare despre serviciul ţintă şi date.
- De asemenea, se poate face bind la un serviciu prin pasarea unui intent la metoda bindService().
- Un mesaj de broadcast poate fi trimis prin folosirea unui intent la apelarea uneia dintre metodele: sendBroadcast(), sendOrderedBroadcast(), sau sendStickyBroadcast().

33.

- Există două tipuri de intenturi: explicite şi implicite.
- La cele explicite se specifică exact componenta destinaţie, mai exact numele clasei. De obicei se foloseşte pentru a porni o activitate sau un serviciu în cadrul aceleiaşi aplicaţii.
- Nu este nevoie să specificăm un intent filter daca vrem să folosim intenturi explicite, acestea vor fi livrate chiar dacă nu este declarat un intent filter.
- Intenturile implicite nu specifică numele componentei ţintă, ci doar acţiunea de executat.
- Sistemul Android va căuta cea mai potrivită componentă care poate executa acea acţiune. Va face asta prin căutarea unei potriviri între intent filter-ele din fişierele manifest şi intent-ul respectiv.
- Dacă se găsesc mai mule, utilizatorul va fi întrebat ce vrea să folosească (de exemplu pentru citirea unui fişier pdf).
- Deci în cazul acesta avem nevoie clară de intent filters declarate în Manifest.

34.

- Pe acest slide vedem cum sunt livrate intent-urile implicite.
- Activitatea A crează un obiect Intent cu acţiunea asociată şi îl dă ca argument metodei startActivity. Sistemul caută toate intent filters care se potrivesc cu acţiunea respectivă. Atunci când se găseşte activitatea potrivită, se apelează onCreate() şi apoi se trimite obiectul Intent activităţii respective.

35.

- Acesta este un exemplu de intent implicit. Presupunem că avem de partajat anumite date. Creăm un intent, configurăm acţiunea ACTION_SEND, adăugam mesajul text (tipul de date este text) şi apelăm metoda startActivity() cu acel intent.
- Dacă nu există o activitate care va primi acest tip de intent, aplicaţia poate crăpa. De aceea este recomandat să verificăm că există o aplicaţie care poate primi acel intent înainte să-l trimitem,

prin metoda resolveActivity(). Dacă rezultatul este diferit de null, atunci există cel puţin o activitate care poate primi acest timp de intent.

36. Acesta este un exemplu de declaraţie a unei activităţi în fişierul Manifest, care include un intent filter cu acţiunea SEND şi tipul de date text. Acest lucru specifică faptul că activitatea poate primi intentul trimis pe slide-ul anterior.

38.

- Un broadcast receiver este o componentă care tratează mesajele de broadcast care sunt trimise în sistem.
- Mesajele de broadcast sunt notificări sau anunţuri.
- Majoritatea mesajelor de broadcast sunt generate chiar de către sistem: de exemplu când bateria este pe terminate, când s-a stins ecranul, a fost primit un SMS, etc.
- Dar aplicaţiile por genera şi ele mesaje de broadcast pentru a face diverse anunţuri către aplicaţii.
- Deşi aceşti receiveri nu au interfaţă grafică, ei pot genera notificări în status bar pentru a anunţa utilizatorul.
- Un receiver de obicei nu face procesări elaborate, deoarece întotdeauna rulează pe thread-ul de UI.
- El va lăsa alte componente, de exemplu un serviciu, să facă operaţiile mai complexe., care sunt determinate de către eveniment.

39.

- Orice mesaj de broadcast este trimis folosind un Intent (cu sendBroadcast() sau sendOrderedBroadcast()).
- Dacă vrem să folosim mesaje de broadcast doar în aplicaţia noastră atunci este mai sigur să folosim LocalBroadcastManager, deoarece este mai eficient (nu se duce prin sistem), datele nu ies din aplicaţie, iar o altă aplicaţie nu poate să trimită acele mesaje (no security holes).
- Un receiver se poate declara static folosind tagul <receiver> în Manifest sau dinamic folosind metoda Context.registerReceiver(). Este recomandată folosirea tag-ului <receiver> pentru a păstra codul curat.

40. Putem avea două tipuri de mesaje de broadcast:

1) normale - care sunt complet asincrone. Receiver-ii pentru acel mesaj rulează într-o ordine nedefinită, posibil chiar în acelaşi timp (se da un intent funcţiei sendBroadcast())
2) ordonate - care sunt livrate la câte un receiver odată (receiveri ordonaţi). Receiver-ul se execută şi apoi poate alege să propage rezultatele la următorul receiver sau să dea abort broadcast şi să nu trimită broadcastul mai departe la următorul receiver. Ordinea receiver-ilor este determinată folosind android: priority în intent filter-ul din codul receiver-ului. (se da un intent funcţiei sendOrderedBroadcast()).

41. Acesta este un exemplu de declarare a unui broadcast receiver în fişierul Manifest. Include un intent filter pentru evenimentul BOOT_COMPLETED. Pentru acest lucru, trebuie cerută permisiunea RECEIVE_BOOT_COMPLETED.

42. Şi acesta este un exemplu de implementare al receiver-ului. Callback-ul onReceive() va fi apelat atunci când evenimentul BOOT_COMPLETED are loc. În callback este pornit serviciul.

44.

- Un content provider este o componentă care oferă acces la un set de date. De obicei o aplicaţie oferă acces altei aplicaţii la datele sale folosind un provider.
- Există content provideri de sistem (cei default), de exemplu: contacts, dictionary, calendar, settings, etc.
- Pentru a accesa un provider, aplicaţia noastră are nevoie de permisiuni specifice (care sunt declarate în fişierul Manifest) de read sau write pe datele respective. De exemplu, dacă vrem să citim date din dicţionar, ne trebuie permisiunea READ_USER_DICTIONARY.
- Există 2 metode de a stoca datele:
    - Folosind fişiere, de exemplu audio, video, poze
    - Folosind date structurate sub forma de tabele cu linii şi coloane (de exemplu baza de date sau array-uri). De obicei se foloseşte SQLite pentru stocarea datelor.

45.

- Se foloseşte un provider dacă vrem să accesăm datele unei aplicaţii din altă aplicaţie, îi vom numi provider şi client.
- Aplicaţia care deţine datele va include provider-ul iar aplicaţia care va accesa datele va include clientul.
- Este nevoie de un obiect client numit ContentResolver. Prin metodele sale vom putea crea, accesa, actualiza şi şterge date.
- Atunci când apelăm metodele sale, se vor apela metodele cu acelaşi nume din ContentProvider.

46.

- Pentru a identifica datele se vor folosi Content URI-uri.
- Un URI are două componente: authority este numele provider-ului, şi path este numele tabelei.
- Un exemplu de URI este content://user_dictionary/words -> providerul este user_dictionary iar tabela este words.
- Astfel, ContentResolver-ul va citi acest URI şi va identifica provider-ul. Va căuta într-o tabelă de sistem cu toţi provider-ii şi va obţine acces la acest provider.
- Resolver-ul va realiza un query către provider pentru acel path.
- Provider-ul va folosi path-ul pentru a identifica tabela şi a face operaţia cerută pe aceea tabelă.

47. Acestea sunt câteva exemple de operaţii care pot fi efectuate asupra content provider-ului numit user dictionary. Putem vedea aici 3 operaţii: query, insert şi update. Pentru fiecare operaţie dăm Content URI-ul tabelei Words ca argument. Projection specifică coloanele care vor fi incluse pentru fiecare rând selectat. Putem specifica criteriile de selecţie pentru rânduri. În final putem specifica ordinea de sortare a rândurilor returnate.

49.

- În continuare vom vedea câteva utilitare, pe care le veţi folosi în laborator.
- Android Studio este IDE-ul oficial pentru dezvoltarea aplicaţiilor Android.
- Include un sistem de build bazat pe Gradle.

50.

- Android SDK Manager este folosit pentru download-area şi gestionarea pachetelor SDK, a exemplelor, a imaginilor de sistem pentru emulator şi a tool-urilor (platform şi build tools).
- Aici putem vedea un screensot al SDK Manager. Putem alege să instalăm sau să dezinstalăm diferite pachete.

51.

- Prin Android Virtual Device (AVD) Manager putem crea şi gestiona dispozitive virtuale care vor fi folosite de către emulator.
- Aici puteţi vedea un screenshot al AVD Manager.
- Emulatorul va rula dispozitive virtuale şi astfel se pot testa aplicaţiile pe calculator, fără să fie nevoie de un dispozitiv fizic.

52.

- Emulatorul de Android se bazează pe QEMU. Acesta oferă posibilitatea de lucru cu mai multe componente: display, tastatură, reţea, gps, audio, radio, etc.
- Se poate creşte viteza de virtualizare, pentru asta este nevoie de o imagine de Android de x86 şi folosind KVM pe Linux şi HAXM pe Windows (pentru ca nu se poate modifica kernelul). Astfel nu este nevoie ca instrucţiunile sa fie translatate, ci pot rula direct pe procesorul PC-ului.
- În plus, în locul folosirii unui GPU virtual, se poate folosi GPU-ul PC-ului, astfel vom obţine o performanţă mai bună.

53.

- Android Debug Bridge (adb) este folosit pentru comunicarea cu dispozitivul real sau virtual.
- Adb are 3 componente principale: un client care rulează pe PC, un server care rulează pe PC şi un daemon care rulează pe dispozitivul real sau virtual.
- Atunci când dăm în linia de comandă adb şi o comandă, aceasta va fi trimisă serverului care o va executa comunicând cu daemonul de pe dispozitiv.
- Se pot face următoarele acţiuni: copia fişiere de pe PC pe telefon şi invers (adb push, pull), instala aplicaţii pe telefon (adb install), afişa mesaje de debug (adb logcat), obţine un shell pe dispozitiv (adb shell).