Programarea dispozitivelor mobile

Cursul 1 - 3

NAVIGAREA ÎN APLICAȚII

1. Panou de control (**Dashboard**)
2. **Listă**
3. **Bară de acțiune**
4. **Bară de instrumente**
5. **Meniu**
6. Selectori (**tab**)
7. **Meniu glisant**

MATERIAL DESIGN

- Teme noi
- Controale noi
- Animații
- Efecte de umbrire

CULORI

a) **colorPrimaryDark**
– bara de stare
b) **colorPrimary**
– bara aplicației
c) **colorAccent**
– controlul selectat
d) **textColorPrimary**
e) **windowBackground**
f) **navigationBarColor**

COMPONENTE SUPORT PENTRU MATERIAL DESIGN

A. **Toolbar**

- bara aplicatiei
- inlocuiește bara de acțiune (ActionBar)
- **android.support.v7.widget.Toolbar**
- **android.widget.Toolbar (API 21+)**

Macheta activitatii	<**android.support.v7.widget.Toolbar** android:id="@+id/toolbar" android:background="@color/culoare_background" android:layout_width="match_parent" android:layout_height="?attr/actionBarSize"/>
Tema activitatii	**Theme.AppCompat.*NoActionBar*** **sau** **Theme.AppCompat cu stilul:** <**style** name="AppTheme" parent="Theme.AppCompat"> <**item** name="**windowNoTitle**">true**</item>** **<item** name="**windowActionBar**">false**</item>** **</style>**

Toolbar - Java

- Clasa **derivată** din **AppCompatActivity,**
- Metoda **onCreate()**
  - – Se instanțiază obiectul de tip Toolbar
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
  - – se asociază obiectul de tip Toolbar:
    setSupportActionBar(toolbar)
- **Activare** buton **UP**

```
ActionBar actionBar = getSupportActionBar();
if (actionBar != null)
{
    actionBar.setDisplayHomeAsUpEnabled(true);
    actionBar.setHomeAsUpIndicator(pictograma);
}
```

B. NavigationView si **DrawerLayout**

NavigationView

- Include **opțiunile meniului glisant**
- 2 **Componente**
  - – Antet
  - – Meniul propriu-zis
- **Inclus** într-o componentă de tip **DrawerLayout**
- **android.support.design.widget**
- **Proprietăți**
  - – **Poziționare**
    android:**layout_gravity**
  - – **Antet**
    app:**headerLayout**
  - – **Meniu**
    app:**menu**

```
<android.support.design.widget.NavigationView
    android:id="@+id/navigation_view"
    android:layout_height="match_parent"
    android:layout_width="wrap_content"
    android:layout_gravity="start"
    app:headerLayout="@layout/antet"
    app:menu="@menu/meniu_drawer"/>
```

DrawerLayout

- **Container suport** pentru **meniul glisant**
- Două **componente**
  - – **Meniul**
  - – **Containerul** de conținut al activității
- Implicit **meniul** este **ascuns**
- **Activarea** meniului
  - – Glisare
  - – Butonul dedicat din bara de acțiune
- Uzual, meniul este poziționat în stînga ecranului, dar poate fi amplasat și în dreapta acestuia

openDrawer()
closeDrawers()

Tratarea evenimentelor

- **setNavigationItemSelectedListener()**
  OnNavigationItemSelectedListener
  onNavigationItemSelected(MenuItem menuItem)
- **ActionBarDrawerToggle**

C. **RecyclerView**

- **Înlocuitor** pentru controale de tip **listă** (ListView, GridView)
- Management eficient al memoriei
- **Sarcinile** sînt **distribuite**
  – Poziționare
  – Animație
- Tratarea evenimentele de selecție nu este la fel de facilă
- **Adapter** – asociere colecție de date și crearea de controale pentru fiecare element din colecție
  - Definit în pachetul **android.support.v7.widget**
  - Se bazează pe șablonul ViewHolder
  - Este **derivat** din **RecyclerView.Adapter<VH>**
  - include o **clasă privată, statică, derivată** din RecyclerView.ViewHolder (VH) responsabilă cu **inițializarea controalelor**
  - **Metode abstracte**
    – public int **getItemCount()**
    – public VH **onCreateViewHolder**(ViewGroup parent, int viewType)
    – public void **onBindViewHolder**(VH holder, int position)
- **ViewHolder** – container pentru controalele asociate datelor elementului curent
- **LayoutManager** – poziționează **elementele** în suprafața disponibilă
  1. LinearLayoutManager – **Listă** de elemente
  2. GridLayoutManager – Afișare **tabelară**
  3. StaggeredGridLayoutManager - Afișare **tabelară** cu adaptare la **conținut**
     – Afișare **verticală** (implicit) și **orizontală**
- **ItemDecoration** – decorează elementele afișate
  - Desenarea peste elemente
  - Utilizat pentru
    – Personalizarea elementelor
    – Evidențierea elementelor
    – Desenarea de separatori între elemente
  - Pot exista mai multe obiecte de tip ItemDecoration apelate în ordinea adăugării

public void **onDraw**(Canvas canvas, RecyclerView parent, RecyclerView.State state) – Desenare înainte de afișarea conținutului elementului
public void **onDrawOver**(Canvas canvas,RecyclerView parent, RecyclerView.State state) – Desenare peste conținutul elementului
public void **getItemOffsets**(Rect outRect, View view, RecyclerView parent, RecyclerView.State state) – Apelată pentru calcularea dimensiunii elementului

- **ItemAnimator** – animează elementele la adăugare, ștergere, reordonare etc.

Tratarea evenimentelor

- **Implementare în adaptor**
- **Implementare în** clasa de tip **activitate**
- Obiectul de tip **View** din constructorulViewHolder
  – **setOnClickListener()**

- În **onBindViewHolder()** se stochează în ViewHolder elementul corespunzător din sursa de date
- Se implementează **onClick(View)**
  – View – elementul apăsat
  – Se testează în prealabil dacă este inițializat un element din sursa de date asociată
- Implementare **RecyclerView.OnItemTouchListener**
- Definire **interfață** în cadrul clasei
  – Metode pentru apăsare sau apăsare prelungită pe element
- **onInterceptTouchEvent()**
  – Determinare element
  – Determinare poziție element
- Utilizare **GestureDetector** pentru determinarea acțiunilor apăsare/apăsare prelungită
- Asociere obiect de tratare a evenimentelor **addOnItemTouchListener()**

D. **CardView**
- **Control** utilizat pentru **afișarea grupată** a informației, sub forma unui card
- Permite
  – efecte de umbrire
  – colțuri rotunjite
- Utilizat deseori în cadrul listelor
- **android.support.v7.widget**
- **Proprietăți**
  - app:**elevation**
  - app:**cardCornerRadius**
  - app:**contentPadding**
- compile '**com.android.support:cardview-v7:23.2.0**'

E. **TextInputLayout**
- Afișează sugestiile cu privire la textul care trebuie introdus într-un control de editare
- Controlul de editare este inclus în containerul TextInputLayout
- Se bazează pe proprietatea **android:hint** a controlului de editare
- După introducerea textului, sugestiile vor fi afișate deasupra controlului de editare
  – Textul suport nu dispare după introducerea conținutului
- Posibilitatea de afișare de mesaje de eroare
- **android.support.design.widget**
- compile '**com.android.support:design:23.1.1**'

```
<android.support.design.widget.TextInputLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

<EditText
        android:id="@+id/editTextParola"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Parola"
        android:inputType="textPassword" />
</android.support.design.widget.TextInputLayout>
```

- Gestiunea mesajelor de eroare
  – **setErrorEnabled()**
    - apelul înainte de afișarea erorii – nu se va modifica dimensiunea controlului
    - XML: **android.support.design:errorEnabled**

– **isErrorEnabled()**
– **setError()** - null pentru ștergerea mesajului

– **getError()**

F. **CoordinatorLayout**

- Permitea definirea relațiilor dintre controalele incluse
  – Comportament
  – Nu toate controalele sînt compatibile
- Coordonează controalele incluse
- Animație
- Efecte de derulare
- **android.support.design.widget**
- **app:layout_behavior**
  – @string/**appbar_scrolling_view_behavior**
- Controalele pot extinde clasa
- **CoordinatorLayout.Behavior**
  – l**ayoutDependsOn()**
  – **onDependentViewChanged()**

G. **AppBarLayout**

- Control linear, vertical
- Implementează gesturi de derulare
- Controalele incluse pot preciza comportamentul la derulare
- Bara aplicației devine parte a conținutului și reacționează la derularea acestuia
- Compatibilă cu:
  – NestedScrollView
  – RecyclerView
- Funcționalitate **dependentă** de includerea directă într-un container de tip **CoordinatorLayout**
- **android.support.design.widget**
- **Parametrii de derulare** pentru controalele incluse
  – Proprietatea **app:layout_scrollFlags**
  1. **scroll**: controlul va fi derulat în afara ecranului
  2. **enterAlways**: controlul va deveni vizibil imediat ce are loc o derulare din afara ecranului (afișare rapidă)
  3. **enterAlwaysCollapsed:** controlul va deveni vizibil, la dimensiunea minimă specificată prin minHeight, imediat ce are loc o derulare din afara ecranului; la finalizarea derulării controlul va fi afișat la dimensiune completă
  4. **exitUntilCollapsed:** la derulare, controlul se va restrînge pînă la dimensiunea minimă, dată de minHeight, înainte să iasă din ecran
  5. **snap**: animația se realizează fără stări intermediare; controlul este ascuns sau afișat integral

H. **CollapsingToolbarLayout**

- Efecte de animație la derulare
- Pe lîngă bara aplicației pot fi incluse și alte controale
- Textul barei aplicației este expandat/restrîns automat
- **android.support.design.widget**
- Moduri de restrîngere (**layout_collapseMode)**
  – off
  – pin
  – parallax

I. **TabLayout si ViewPager**

ViewPager

- Derularea orizontală a conținutului

- **android.support.v4.view.ViewPager**
- Fragmente pentru conținut
  – Clase de tip **Fragment**
  – **Machete** asociate
- Adaptor
  – Extinde clasa
    • **FragmentStatePagerAdapter**
    • **FragmentPagerAdapter** (s.f.m.)
  – Constructor cu FragmentManager
  – Fragment **getItem(int pozitie)**
  – int **getCount()**

TabLayout

- Afișarea grupată a conținutului
- Conținutul accesibil prin intermediul selectorilor (tab-uri)
- Suport pentru ViewPager
- Selectorii pot fi ficși sau derulanți
- Selectorii pot ocupa spațiul în mod egal sau pot fi centrați
- **android.support.design.widget**
- **Proprietăți**
  – **tabMode** (modul de afișare a selectorilor: fixed sau scrollable)
  – **tabGravity** (modul de centrare a conținutului: fill sau centre)
- **Evenimente**
  – **OnTabSelectedListener**
  – **TabLayoutOnPageChangeListener**
  – **ViewPagerOnTabSelectedListener**
- Asociere cu ViewPager
  – setup**WithViewPager()**

J. **Floating Action Button** (FAB)

- Buton flotant
- Execuția rapidă a unei acțiuni frecvente
- **android.support.design.widget**
- **Proprietăți**
  – fabSize (normal, mini)
  – src (pictograma asociată)
  – backgroundTint (culoarea de fundal; implicit se alege colorAccent)

K. **Snackbar**

- Afișarea rapidă a unui mesaj scurt
- Posibilitatea de întreprindere a unei **acțiuni** prin intermediul unui **buton**
- Mesajul dispare
  – după un interval prestabilit
  – prin glisare
  – prin acționarea butonului (dacă există)
- **android.support.design.widget**
- **Creare**
  – metoda **make()** - Control (**View**), mesaj, durată
- **Adăugarea** unei **acțiuni**
  – Metoda **setAction()** -Text buton acțiune, obiect de tip **OnClickListener**
- **Afișare** – Metoda **show()**

J. NestedScrollView

- Necesitate:
  – Utilizarea unui control cu derulare în cadrul altui control cu derulare

- **android.support.v4.widget.NestedScrollView**

Cursul 4

Furnizori de continut

Partajarea datelor

- Pentru **partajarea datelor** între aplicaţii se utilizează și **furnizorii de conţinut** care pun la dispoziție un mecanism standardizat pentru **transferul** datelor între aplicații
- **Surse de date**:
  - fişiere
  - baze de date
  - alte surse (Internet)
- O **alternativă** la furnizorii de conţinut: **comunicarea între procese**

Furnizorii de continut

- **Acces** deseori **ierarhic**
- **Bază de date**
- Mai multe **tabele**
  - Coloane
  - Rînduri
- **Referire** prin **URI**
- **Conținutul** are asociat un tip **MIME**

Accesul la furnizorii de conținut

```
Uri uri = ...
        //Pachetul android.provider
ContentResolver cr = getContentResolver();
Cursor date = cr.query(uri, null, null, null, null);
if (date != null)
while(date.moveToNext())
        {
        //prelucrare linie curenta
        }
}
```

Accesul la furnizorii de conținut **predefiniți**

- Pachetul **android.provider**
- **ContentResolver**
  - Context#**getContentResolver()**

Implementarea furnizorilor de conținut

- Definire URI-ul asociat conținutului
  - Crearea definițiilor UriMatcher
- Implementarea clasei abstracte ContentProvider
  - onCreate()
  - metode CRUD
  - obținerea tipului rezultatului

- Declararea furnizorului de conținut în fișierul AndroidManifest.xml

Referirea conținutului

- **URI** de forma **content://furnizor[/cale][/id]**
- **furnizor** (authority)
  - identificatorul furnizorului de conținut
  - trebuie să fie unic pentru fiecare furnizor de conținut
- **cale**
  - asigură accesul la diferite colecții de date puse la dispoziție de furnizorul de conținut
  - un segment din cale poate fi asociat unei tabele dintr-o bază de date relațională
- **id**
  - Identificatorul unei anumite înregistrări

Constante definite în clasa furnizorului de conținut:

– **Calls.CONTENT_URI** = "**content://call_log/calls**"

- **Uri** – static Uri parse(String)
- **ContentUris**
  - static long parseId(Uri)
  - static Uri withAppendedId(Uri, long)
- **UriMatcher:**
  - **CALE** – Potrivire exactă
  - **CALE/#** – CALE urmată de o valoare numerică
  - **CALE/*** – CALE urmată de un șir de caractere
  - **CALE/*/SUBCALE/#**
  - Adăugare căi și asocierea de identificatori
    - Metoda **addURI()**
  - Pe baza căilor furnizate este returnat identificatorul asociat
    - Metoda **match()**

Implementarea clasei abstracte **ContentProvider**

- onCreate()
- getType()
- query()
- insert()
- update(
- delete()

Inițializarea furnizorului de conținut

- Rulează în firul principal
- Operații reduse ca durată
  - Evitarea deschiderii bazei de date, operații complexe etc.
- Returnează o valoare booleană
  - Rezultatul inițializării furnizorului de conținut

Tipul MIME asociat

- Pereche de forma t**ip/subtip**
  - text/xml
  - application/pdf
- Returnat de metoda **getType()**
  - **String**

Returnarea tipului asociat

- Valoarea **vnd.android.cursor.item**
  - conținutul de tip **Cursor** cu **o singură înregistrare**

– **ContentResolver.CURSOR_ITEM_BASE_TYPE**

- Valoarea **vnd.android.cursor.dir**
  – conținutul de tip **Cursor** cu **zero** sau mai **multe** înregistrări
  – **ContentResolver.CURSOR_DIR_BASE_TYPE**
- Este necesară includerea subtipului

Interogarea datelor
- Metoda **query()**
- **Parametri**
  – identificatorul furnizorului de conținut (**Uri**);
  – **coloanele** corespunzătoare datelor (proiecția) (String[])
  – **criteriul** de **selecție** (String)
  – **parametrii** de **selecție** (String[])
  – **ordinea** de **sortare** a acestora (String)
- **Returnează** un obiect de tip **Cursor**

Inserarea
- Metoda **insert()**
- **Parametri**
  – identificatorul furnizorului de conținut (**Uri)**
  – valorile asociate înregistrării (**ContentValues**)
- **Returnează** un **Uri** asociat înregistrării inserate
  – Include identificatorul înregistrării

Modificarea
- Metoda **update()**
- **Parametri**
  – identificatorul furnizorului de conținut (**Uri**)
    • dacă nu este inclus un identificator, modificarea se aplică tuturor înregistrărilor
  – valorile asociate cîmpurilor care se vor modifica (**ContentValues**)
  – **criteriul** de **selecție** (String)
  – **parametrii** de **selecție** (String[])
- **Returnează numărul** de înregistrări afectate

Ștergerea înregistrărilor
- Metoda **delete()**
- **Parametri**
  – identificatorul furnizorului de conținut (**Uri**)
    • dacă nu este inclus un identificator, vor fi șterse toate înregistrările
– criteriul de **selecție** (String)
– parametrii de **selecție** (String[])
- Returnează numărul de înregistrări afectate

Notificarea executării operațiilor
- **ContentObserver**
  – Clasă abstractă
  – Apeluri inverse la apariția modificărilor – **onChange()**
- Metode în clasa **ContentResolver**
  – **registerContentObserver()**/unregisterContentObserver()
  – **notifyChange()**
    - Inserare, modificare, ștergere

- Se transmite Uri-ul asociat înregistrării modificate

- Exemplu apel

```
getContext().getContentResolver().
notifyChange(uri, null);
```

Declararea în fișierul AndroidManifest.xml

```
<provider
    android:name="com.pdm.provider.FC"
    android:authorities="com.pdm.provider.FC"
    android:exported="true" >
</provider>
```

Cursul 5

RECEPTORI DE MESAJE

Transmiterea și **recepționarea** mesajelor globale

- La nivel global pot fi transmise mesaje
- Mesajele pot fi transmise din **aplicații**:
  - **de sistem**
  - **utilizator**
- Aplicațiile pot reacționa la apariția mesajelor transmise global, la nivelul sistemului
- Exemple:
  - Finalizarea descărcării unui fișier
  - Identificarea unui dispozitiv (NFC, Bluetooth etc.)
  - Nivelul bateriei etc.

Mesaje globale

- Intent.**ACTION_BATTERY_LOW**
- Intent.**ACTION_BOOT_COMPLETED**
- Intent.**ACTION_MEDIA_MOUNTED**
- Intent.**ACTION_SCREEN_OFF**
- Telephony.Sms.Intents.**SMS_RECEIVED**

Recepționarea mesajelor globale

- Se implementează clasa abstractă **BroadcastReceiver**
- Receptorii nu prezintă interfață grafică
- **O aplicație** poate avea mai **multe componente** de acest tip
- Includ **filtre de mesaje**

Înregistrarea receptorilor

- Receptorii sunt **înregistrați**:
  - **Static:** fișierul manifest **XML** (elementul receiver)
  - **Dinamic**: codul sursă
- Receptorul
  - Independent (XML)
  - Legat de componenta în care este definit (Java)

Înregistrarea receptorilor (**XML**)

- **Androidmanifest.xml**

```
<receiver android:name=".ClasaReceptor" >
    <intent-filter>
        <action android:name="ACTIUNE_SPECIFICA" />
    </intent-filter>
```

</receiver>

- În fișierul **Java**
  - Crearea clasei **ClasaReceptor** derivată din clasa abstractă **BroadcastReceiver**

Înregistrarea receptorilor (**Java**)

- Creare obiect de tip **IntentFilter**
- Creare **obiect** care **implementează** clasa abstractă **BroadcastReceiver**
- Înregistrarea unui receptor
  - **registerReceiver()**
- Deconectarea receptorului
  - **unregisterReceiver()**

Prelucrarea mesajelor

- Metoda onReceive(Context, Intent)
  - Mesajul este primit ca parametru
  - Prelucrări la recepționarea mesajului
  - Reprezintă durata de viață a unui receptor
- Oprirea retransmiterii mesajului în sistem (pentru mesajele cu priorități)
  - abortBroadcast()

Transmiterea mesajelor globale

- Mesajele sînt transmise
  - Fără prioritate
  - Cu priorități
- **sendBroadcast()**
- **sendOrderedBroadcast()**
- Parametrul comun: mesajul (**Intent)**
- O formă care **include**:
  - **Permisiunea** necesară (String)

Mesaje locale

- Clasa **LocalBroadcastManager**
  - **android.support.v4.content**
- Obținere instanță
  - **LocalBroadcastManager.getInstance(Context)**
- Transmitere mesaj
  - **sendBroadcast(Intent)**
- Asociere receptor
  - **registerReceiver(BroadcastReceiver, IntentFilter)**
- Terminare asociere
  - **unregisterReceiver(BroadcastReceiver)**

Servicii

- Rutine care **rulează** în **paralel** cu firul **principal**
- **Nu** prezintă **interfaţă grafică**
- Permit derularea unor acţiuni în fundal fără a bloca firul principal de execuţie şi interacţiunea cu aplicaţiile
- **Servicii**

1. predefinite (**sistem**)
   - Numeroase servicii
     - Notificare
     - Conectivitate
     - Descărcare fișiere
     - Locație etc.
   - Gestionate prin clase specializate
   - Identificate prin **constante** definite în clasa **Context**

Servciu	Constantă asociată (clasa Context)
Alarm	ALARM_SERVICE
Bluetooth	AUDIO_SERVICE
Location	LOCATION_SERVICE
Notification	NOTIFICATION_SERVICE
Sensor	SENSOR_SERVICE
Telephony	TELEPHONY_SERVICE
Wifi	WIFI_SERVICE

- **Context#getSystemService**(String serviciu)
  – **serviciu**: șir de caractere cu numele acestuia definit în clasa Context
  – returnează un obiect de tipul serviciului:
    - **LocationManager**
    - **AudioManager**
    - **DownloadManager**
    - **WiFiManager** etc.

  prin intermediul căruia acesta este utilizat.

Notificari:

```
private final int NOTIF_ID = 1;
String url = "url valid";
//initializare manager notificari
NotificationManager notifMgr = (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);
//intent pentru comportamenul la click pe notficare
Intent intentNotificare = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intentNotificare, 0);
//creare notificare
Notification notif = new Notification.Builder(this)
.setContentTitle("Notificare")
.setSmallIcon(R.drawable.icon).setContentText("Serviciu Info")
.setContentIntent(pendingIntent)
.build()
//transmiterea notificarii notif
notifMgr.notify(NOTIF_ID, notif);
//Pentru oprirea notificărilor:
notifMgr.cancel(NOTIF_ID);
```

Descărcare fișiere:

```
DownloadManager dManager = (DownloadManager)
getSystemService(Context.DOWNLOAD_SERVICE);
//cererea de descărcare
Request request = new Request(Uri.parse(urlCurs));
request.setDescription("PDM - Curs");
long dldId = dManager.enqueue(request);
registerReceiver(descarcat, new
IntentFilter( DownloadManager.ACTION_DOWNLOAD_COMPLETE));
//...
BroadcastReceiver descarcat = new BroadcastReceiver() {
        public void onReceive(Context ctxt, Intent intent)
```

{ /* */}
//...
unregisterReceiver(descarcat);

2. servicii **utilizator**

Senzori

- Componente hardware

• Sursă de date pentru
  – Măsurarea mișcării, orientării și a condițiilor de mediu
  – Determinarea poziției

• ***Exemple***:

- **Accelerometr**u – măsoară accelerația pe axele x, y și z (**TYPE_ACCELEROMETER**)
- **Ambiental** – determină nivelul luminii ambientale exprimat în lucși (**TYPE_LIGHT**)
- **Proximitate** – măsoară distanța în centimetri dintre senzor și un obiect (**TYPE_PROXIMITY**)
- **Busolă** – determină atracția magnetică pe axele x, y și z (**TYPE_MAGNETIC_FIELD**)
- **Giroscop** – determină viteza de rotație în jurul axelor x, y și z, măsurată în radiani/secundă (**TYPE_GYROSCOPE**)
- **Gravitație** – determină gravitația pe axele x, y și z (**TYPE_GRAVITY**)
- **Presiune** – măsoară presiunea atmosferică (**TYPE_PRESSURE**)
- **Temperatură** – măsoară temperatură (**TYPE_TEMPERATURE**)

• Posibilitatea de preluare a datelor
• Pachetul **android.hardware**
• **Servicii**
  – **SENSOR_SERVICE**
• Clasa **SensorManager**
• Clasa **Sensor**
• Interfața **SensorEventListener**

Utilizare senzori

• Inițializarea serviciului
• **Înregistrarea** unui **listener** care va fi informat cu privire la modificările apărute
  – Listener-ul este asociat unui senzor
  – Metoda corespunzătoare este apelată periodic cu valorile senzorului
• **Terminarea** asocierii **listener-ului**

Clasa Sensor

- **Denumire**
  – **getName()**
- **Puterea** utilizată (mA)
  – **getPower()**
- Informații despre senzor
  – **getVendor()**
  – **getVersion()**
- **Valoarea maximă**
  – **getMaximumRange()**
- **Tipul**
  – Constante de format **TYPE_** (**TYPE_LIGHT**)

Obținerea datelor de la senzori

• Interfața **SensorEventListener**
• Modificările senzorilor
  – void **onSensorChanged**(SensorEvent event)

• Valorile citite disponibile prin: **SensorEvent**:
  – float [] values

SensorManager
• Obținerea unui anumit senzor (Sensor)
  – **getDefaultSensor**(tip)
• **Inițializare** listă de senzori de un anumit tip (**List< Sensor>**)
  – **getSensorList**(tip)
    • Sensor.**TYPE_ALL** – pentru toți senzorii disponibili

• **Înregistrarea** unui **obiect** care va fi informat cu privire la **modificările** senzorului
  – **registerListener()**
    • Context, senzor, perioada informării
• **Eliminarea** asocierii
  – **unregisterListener()**

Perioada informării
• **SensorManager.**
  – SENSOR_**DELAY_NORMAL**
  – SENSOR_**DELAY_FASTEST**
  – SENSOR_**DELAY_UI**
  – SENSOR_**DELAY_GAME**

Informații senzori – Exemplu
```
SensorManager sm = (SensorManager)getSystemService(SENSOR_SERVICE);
List<Sensor> senzori = sm.getSensorList(Sensor.TYPE_ALL);
String [] numeSenzori = new String[senzori.size()]; int i = 0;
for (Sensor senz : senzori) {
    numeSenzori[i++] = "Nume: " + senz.getName() + "; Tip: " + senz.getType() + "; Max: " +
        senz.getMaximumRange();

}
```

Preluare date accelerometru
```
//onResume()
sm.registerListener(this, senzor,SensorManager.SENSOR_DELAY_UI);
//onPause()
sm.unregisterListener(this);
//activitate
public void onSensorChanged(SensorEvent event) {
    switch(event.sensor.getType()) {
    case Sensor.TYPE_ACCELEROMETER:
        tvValori.setText("Valori [m/s^2]");
        tvValX.setText("x = " + event.values[0]);
        tvValY.setText("y = " + event.values[1]);
        tvValZ.setText("z = " + event.values[2]);
    break;
    }
}
```

Curs 6

Servicii utilizator

• **Derivate** din clasa **Service**
• Ciclu de viață bine determinat
• Trebuie **declarate** în fișierul **AndroidManifest.xml**

Servicii

1. Servicii **locale**
   – Uzual, **rulează** în același **proces** cu **aplicația** care a **pornit serviciul**
2. Servicii **la distanță**
   – **Rulează** în **propriul proces**
   – **Comunicare inter-proces**
   • RPC, AIDL (Android Interface Definition Language) etc.
3. Servicii **fără referință**
   – **nu** permit clienților **interacțiunea**
      • doar **pornire** și **terminare**
   – inițializate folosind metoda **startService()** din clasa Context
4. Serviciile **cu referință**
   – permit continuarea **comunicării** din partea clientului **după** ce acesta a fost **inițializat**
   – inițializate prin metoda **bindService()** din clasa Context

Tip serviciu	Inițializare serviciu	Oprire serviciu
Local	Context#**startService()**	Context#**stopService()** Service#**stopSelf()**
La distanță cu referință	Context#**bindService()**	Context#**unbindService()**

Evenimente din ciclul de viață
   • **Crearea** serviciului
      – **onCreate()**
   • **Pornirea** serviciului – client **startService()**
      – **onStartCommand()**
   • **Conectarea** la serviciu – client **bindService()**
      – **onBind()**
   • **Terminarea** serviciului
      – **onDestroy()**

Declarare în fișierul manifest **XML**

```
<application ... >
    <service android:name=".Serviciu" />
    ...
</application>
```

• **Atribute**
   – android:**icon**
   – android:**label**
   – android:**process**

Servicii locale

Evenimente din ciclul de viață

1. **Conectarea** la serviciu: **startService()**
   – Serviciul este creat: **onCreate()**
   – Se apelează metoda **onStartCommand()**
2. **Conectarea** la serviciu: **startService()**
   – Se apelează metoda **onStartCommand()**
3. ...
4. **Terminarea** serviciului: **stopService()**
   – Serviciul este terminat: **onDestroy()**
   • Se realizează la primul apel

Structura unui **serviciu local**

```
public class Serviciu extends Service {
//metoda trebuie implementată obligatoriu
@Override
        public IBinder onBind(Intent arg0) {
        return null;
        }

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

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
        //inițiere operații serviciu
}

@Override
public void onDestroy() {
        super.onDestroy();
        //eliberare resurse ; terminare serviciu
        }
}
```

Inițierea unui **serviciu local**

```
Intent intent = new Intent(this, Serviciu.class)
//Apel din client: pornirea serviciului
startService(intent);
// Apel din client: oprirea serviciului
stopService(intent);
```

Servicii de tip **IntentService**

- **Derivare** din clasa **IntentService**
- Pune la dispoziție o metodă executată într-un **fir de execuție distinct**
- Serviciul este **oprit după finalizarea prelucrărilor** din metoda invocată
- **Rezultatele** sînt trasmise printr-un mesaj (**Intent**)

– **Necesar** un **receptor** de evenimente

Structura unui serviciu de tip **IntentService**

```
public class Serviciu extends IntentService {
        public Serviciu () {
                super(Serviciu.class.getName());
        }

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

@Override
protected void onHandleIntent(Intent intent) {
        //prelucrări de durată
}

@Override
public void onDestroy() {
        super.onDestroy();
        //eliberare resurse
        }
}
```

Inițierea unui **serviciu local**

```
Intent intent = new Intent(this, Serviciu.class);
        //Apel din client: pornirea serviciului
        startService(intent);
```

SERVICII LA DISTANȚĂ

Comunicarea între **procese**

- Fișiere
- Socket
- Transmiterea de mesaje
- Partajarea memoriei
- Furnizori de conținut
- Intent
- Messenger
- Binder

Interfața IBinder

- Descrie **protocolul de comunicare** cu un obiect la **distanță**
- **Extinderea** clasei **Binder**
- **Interfețe AIDL**
- Clasa **Messenger**
  - Implementată prin intermediul **AIDL**
  - Utilizează obiecte **Handler** și **Message**

Comunicarea cu **serviciul**

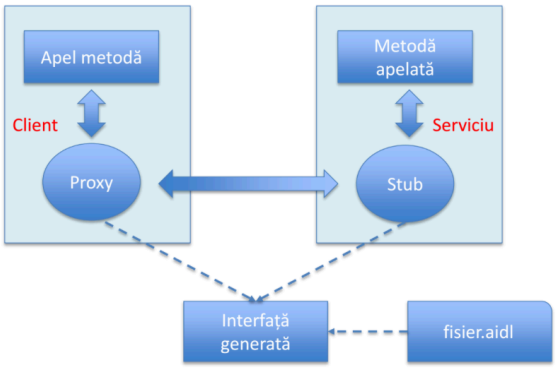
- Limbajul **AIDL** (Android Interface Definition Language)
  - asemănător Java
- Fișier **.aidl**
- Include interfața cu metodele puse la dispoziție de serviciu
- Compilatorul **generează** o **interfață Java**

Interfața generată

- Metodele puse la dispoziție de interfața serviciului
- Clasa abstractă **Stub**
  - **Extinde Binder**
  - **Implementează interfața generată**
  - Include clasa **Proxy**
    - Implementează apelul metodelor din interfață

Serviciul

- Extinde clasa abstractă **Stub** din interfața generată
  - Implementează metodele puse la dispoziție de serviciu
- Metoda **onBind()**
  - **Returnează** obiectul de tip **Stub**



Conectarea la serviciile la **distanță**

- Implementarea interfeței **ServiceConnection**
  - monitorizează **starea** conexiunii la serviciu
  - metode
    - **onServiceConnected()**
      - **Inițializare** obiect de tip interfață serviciu
    - **onServiceDisconnected()**
- Metoda **bindService()**
  - **Intent** asociat componentei serviciului
  - Obiectul de tip **ServiceConnection**
  - **Indicatori** (repornirea/terminarea serviciului)

Terminarea execuției serviciului

- Sînt **reținute referințe** la fiecare client conectat
- **Deconectarea** clienților
  - metoda **unbindService()** din clasa Context
- **Terminarea** serviciului
  - Apelul **realizat** de către **ultimul client** rămas **conectat** la serviciu

Structura unui **serviciu la distanță**

```
public class Serviciu extends Service {
@Override
public IBinder onBind(Intent arg0) {
        return new InterfataServiciu.Stub() {
        //metode serviciu
        }
}

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

@Override
public void onDestroy() {
        super.onDestroy();
        //eliberare resurse
        }
}
```

Inițierea unui **serviciu la distanță**

```
//...
        Serviciu serviciu;
        //clasă imbricată
        class ConexiuneServiciu implements ServiceConnection {
                public void onServiceConnected(ComponentName name, IBinder boundService) {
                serviciu = InterfataServiciu.Stub.asInterface((IBinder) boundService);
}

public void onServiceDisconnected(ComponentName name) {
        serviciu = null;
}}
```

Servicii

```
        Intent intent = new Intent();
        intent.setClassName("pachet", "pachet.Serviciu");
        ConexiuneServiciu conexiune = new ConexiuneServiciu ();
        //Apel din client: pornirea serviciului
        boolean stare = bindService(intent, conexiune,
                Context.BIND_AUTO_CREATE);
        // Apel din client: oprirea serviciului
        unbindService(conexiune);
```

Curs 7

Aplicații de tip Widget

- Includ **ferestre** care sînt **atașate** ecranului principal (**Home Screen**)
- Rulează în **alt proces**
  - Asociat aplicației gazdă
- Ocupă o **suprafață** reprezentată prin **celule**
- **Conținutul** este **actualizat periodic** la recepționarea notificărilor
- O aplicație poate gestiona mai multe componente de același tip
- Sunt **definite** pe **baza receptorilor** de mesaje
- Folosesc **obiecte speciale** pentru afișarea interfeței utilizator

Componentele aplicațiilor de tip **Widget**

1. Fișier de tip **machetă** asociat interfeței
2. Fișier **xml** cu proprietățile aplicației
3. **Receptor de mesaje**
   - Inițializarea interfeței utilizator
4. Opțional, **activitate de configurare**
5. Pentru **colecții**, serviciu de tip **adaptor**
6. **Intrări** pentru receptor, **activitatea de configurare** și **serviciu** în fișierul manifest

Interfața utilizator

- **Definită** prin fișiere de **resurse** de tip **machetă**
- Este **expusă** prin obiecte de tip **RemoteViews**
  - Obiectele sunt gestionate de sistem
- **Limitări** în ceea ce privește tipul **controalelor** ce pot fi utilizate

Containere

- FrameLayout
- GridLayout
- LinearLayout
- RelativeLayout

Controale

- AnalogClock
- Button
- Chronometer
- ImageButton
- ImageView
- ProgressBar
- TextView
- ViewFlipper

Colecții

- GridView
- ListView
- StackView

- AdapterViewFlipper

RemoteViews

- **Constructor** cu **inițializare layout**
  - Nume pachet, layout

Modificare stare controale

- setTextView*()
- setProgressBar()
- setImageView*()
- setChronometer()
- setUri() etc

Acțiuni controale

- **Transmiterea** de **mesaje** (obiecte de tip **Intent**) la apăsarea controalelor
- Metoda **setOnClickPendingIntent**(id_buton, pend_intent)
  - Nu este disponibilă pentru elementele colecțiilor!

Apeluri metode

- Atributul @RemotableViewMethod
- setLong()
- setInt() etc

Fișierul xml cu proprietăți

- Salvat în directorul **res/xml**
- Include **proprietățile** asociate unei aplicații de tip Widget
- **Referit în intrarea** pentru **receptor** din fișierul manifest
- Nodul **rădăcină**: **appwidget-provider**
- Asociat clasei de tip **AppWidgetProviderInfo**
- Posibilitatea de **redimensionare**
  - android:**resizeMode** (vertical/horizontal)
- **Categoria**
  - android:**widgetCategory** (home_screen/keyguard)
- **Macheta** pentru ecranul de **blocare**
  - android:**initialKeyguardLayout**
- Activitatea de **configurare**
  - android:**configure**

Proprietăți

- **Macheta** inițială
  - android:**initialLayout**
- **Dimensiuni**
  - android:**minWidth,** android:**minHeight**
  - Recomandat: (numărul_de_celule * 70) – 30 dp
- **Frecvența** de actualizare
  - android:**updatePeriodMillis**
- Imaginea afișată pentru aplicație în lista de aplicații disponibile (**resursă**)
  - android:**previewImage**

Receptorul

- Asociat aplicației de tip Widget
- Include **metode specifice** receptorilor de mesaje, dar și **metode proprii**
- Are ca **scop**
  - **Actualizarea interfeței**
  - **Definirea** mecanismelor de tratare evenimentelor generate de interacțiunea cu aplicația

AppWidgetProvider

- **Derivată** din **BroadcastReceiver**
- **Notificări** specifice aplicației
  - actualizare
  - activare
  - dezactivare
  - ștergere

Metode în ciclul de viață

- **onReceive()**
- **onUpdate**(Context, AppWidgetManager, int[] id_widgets)
  - Inițializarea interfeței utilizator
- **onEnabled()**
- **onDeleted()**
- **onDisabled()**

AppWidgetManager

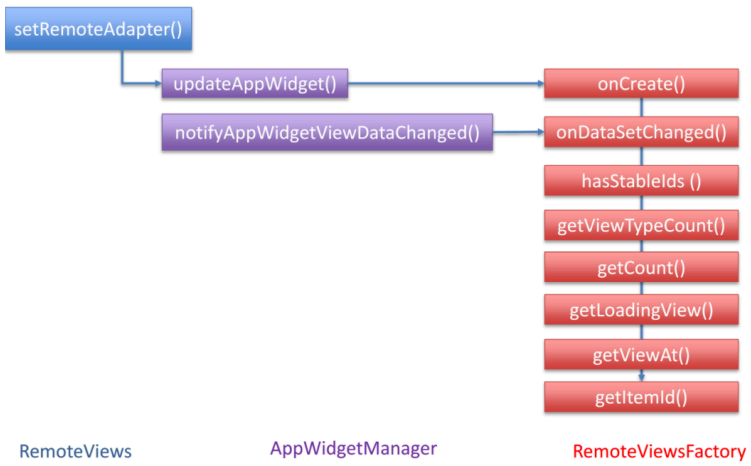
- **Actualizarea stării** aplicațiilor de tip Widget
- Obținerea de **informații** despre aplicațiile de tip Widget instalate
- Transmiterea structurii interfeței
  - **updateAppWidget**(id_widget, remote_views)
  - **updateAppWidget**(ComponentName, remote_views)

Activitatea de configurare

- **Configurarea** și **actualizarea** aplicației
- Lansată în momentul creării aplicației
- Activitatea primește ca **parametru** prin mesaj identificatorul aplicației
  - AppWidgetManager.**EXTRA_APPWIDGET_ID**

Utilizarea colecțiilor

- Se bazează pe un **serviciu** pentru **popularea** colecției
  - **RemoteViewsService**
- **Necesită** un **adaptor** special care implementează interfața **RemoteViewsService.RemoteViewsFactory**
- Adaptorul este asociat cu metoda **setRemoteAdapter()** din clasa RemoteViews



RemoteViewsService

- Responsabil cu **returnarea** unui **obiect** care implementează interfața:
  - **RemoteViewsService.RemoteViewsFactory**
- Implementează metoda **onGetViewFactory()**

RemoteViewsFactory

- **Constructor**
  - **Context**
  - Obiect de tip **Intent**
- **onCreate()**
  - apelată la **crearea** obiectului de tip factory
- **onDataSetChanged()**
  - apelat la **notificarea** de **actualizare** a colecției
- **onDestroy()**
- **getViewAt()**
  - invocată pentru fiecare element al colecției în parte
  - **returnează** un obiect de tip **RemoteViews**
  - se completează mesajul de tip șablon cu valori concrete, legate de poziția curentă în colecție
- **getCount()**
  - returnează **numărul** de elemente ale colecției
- **getItemId()**
  - Returnează **identificatorul** elementului de la poziția dată
- **getLoadingView()**
  - Permite utilizarea unui **obiect personalizat** pentru inițializarea controlului
  - **null** – se utilizează cel **implicit**
- **hasStableIds()**
  - Specifică dacă un identificator **referă același obiect**
- **getViewTypeCount()**
  - **Numărul** de tipuri de **controale** returnate de clasa de tip factory

Metode specifice RemoteViews

- **setPendingIntentTemplate()**
  - **Șablonul** pentru evenimentul de **click** pe un element din colecție
  - Este transmis **identificatorul** colecției
  - Se utilizează în **AppWidgetProvider**
- **setOnClickFillInIntent()**
  - Pentru evenimentul de **click** pe un element din colecție

– Se utilizează în **RemoteViewsFactory**
– Este transmis **identificatorul** controlului

AndroidManifest.xml: **definirea** aplicației de tip **Widget**

- **Receptorul** asociat aplicației de tip Widget
  - Elementul **receiver**
  - **Filtrul** de mesaje cu acțiunea
    – android.appwidget.action.**APPWIDGET_UPDATE**
  - **Proprietăți**
    – **meta-data**
    • android-name="**android.appwidget.provider**"
    • android:**resource**="@xml/fisierul_xml_cu_proprietati"
- Activitatea de configurare
  - Elementul **activity**
  - **Filtrul** de mesaje cu acțiunea
    – android.appwidget.action.**APPWIDGET_CONFIGURE**
- **Serviciul** pt **adaptor**, pentru aplicații de tip Widget care includ **colecții**
  - Elementul **service**
  - **Permisiunea**
    – android.permission.**BIND_REMOTEVIEWS**

Cursul 8

Resurse de tip multimedia

- **Locale**
  – Incluse în pachetul aplicației (**resurse** și **conținut**)
  – Sistemul de **fișiere**
  – **Biblioteci globale media**
- Disponibile la distanță (**server**)
  – **Accesibile** prin diferite **protocoale**
    • HTTP, HTTPS
    • RSTP etc.

Operații

- **Redare**
  – **Prezentarea** conținutului **media** într-o formă specifică
- **Captare**
  – **Preluarea conținutului** media și **salvarea** sau **retransmiterea** acestuia
- **Prelucrare**
  – **Modificarea conținutului** media sau aplicarea de **transformări** specifice acestuia (rotiri, translatări, modificare frecvențe etc.)

- **Imagini**
  – Afișare (ecran, în memorie)
  – Captare imagini (camera, memorie, ecran)
  – Prelucrare imagini

- **Audio**
  - – Redare clipuri audio cu posibilități de control al derulării
  - – Înregistrare clipuri audio (Microfon)
  - – Prelucrare clipuri audio (frecvență, volum etc.)
- **Video**
  - – Redare clipuri video cu posibilități de control al derulării
  - – Înregistrare clipuri video (Camera și microfon)
  - – Prelucrare clipuri video
- **Redare**
  - – Sursa media
- **Înregistrare**
  - – **Sursa** (camera, microfon etc.)
  - – **Destinația**
    - fișier local
    - fișier la distanță
    - streaming etc.
  - – Formatul fișierului
  - – Codec-ul utilizat

Multimedia Android

- **Pachete**:
  - – android.**media**
  - – android.**hardware**
- Includerea de **capabilități hardware**
  - – camera, autofocus, flash, front
- Includerea de **permisiuni** de acces
  - – camera
  - – microfon
  - – memoria externă
  - – Internet etc.

Suport clipuri media

- Audio
  - – 3gp (AAC),
  - – flac (FLAC),
  - – mp3 (MP3),
  - – mid (MIDI),
  - – ogg (Ogg Vorbis)
  - – wav (PCM/WAVE)
- Video 3gp (3GPP) și mp4 (MPEG-4)
  - – codificate H.263 și H.264.

Referirea resurselor

- Referire prin **identificator**, **URI** sau **cale**
- **Incluse** în proiect (directorul **raw**)
  - – Identificator resursă: ***R.raw.id_resursă***
- **Fișiere locale**
  - – **Uri**: file:///sdcard/Audio3.mp3
  - – **Cale:** /sdcard/Audio3.mp3
- Fișiere **la distanță** (**protocoale** RTSP, HTTP/HTTPS)
  - – **Uri**: http://pdm.ase.ro/Audio.mp3

MediaStore

- **Furnizor de conținut**
- **Acces global** la clipurile media din sistem
- Clase specializate pe **conținut**:
  - MediaStore.**Audio**
  - MediaStore.**Video**
  - MediaStore.**Images**

- Informații despre clipuri
  - Artist, titlu, dimensiune, locație etc.

Exemplu MediaStore

```
String[] coloane = {
MediaStore.Images.ImageColumns._ID, MediaStore.Images.ImageColumns.DATA };
String sortare = MediaStore.Images.ImageColumns._ID + " DESC";
Cursor imagini = getContentResolver().query( MediaStore.Images.Media.EXTERNAL_CON-
TENT_URI, coloane, null, null, sortare);
try {
        if (imagini.moveToFirst()) {
        int indexData = imagini.getColumnIndex(
        MediaStore.Images.ImageColumns.DATA);
        String caleImagine = imagini.getString(indexData);
        Bitmap fotografie = BitmapFactory.decodeFile( Uri.parse(caleImagine).getPath());
        imagini.close();
}
} catch (Exception ex) { ex.printStackTrace();
}
```

Clase pentru **redare/afișare**

Resurse	Clase/Interfețe
Audio	**MediaPlayer** + (**MediaController** + MediaController.MediaPlayerControl) (opt.) **SoundPool** **AudioTrack** (generare secvență tonuri)
Video	**MediaPlayer** + **SurfaceView** **VideoView** (control) + **MediaController** (opt.)
Imagini	**ImageView** (control)

Clase pentru înregistrare

Resurse	Clase
Audio	**MediaRecorder**
Video	**MediaRecorder** **Camera**
Imagini	**Camera** **Bitmap**

Utilizare aplicații implicite

Aplicații dedicate	Clase
Audio	Intent • Intent.ACTION_VIEW • MediaStore.Audio.Media.RECORD_SOUND_ACTION
Video	Intent • Intent.ACTION_VIEW • MediaStore.ACTION_VIDEO_CAPTURE
Imagini	Intent • Intent.ACTION_VIEW • MediaStore.ACTION_IMAGE_CAPTURE
Biblioteca globală media	MediaStore

REDAREA CLIPURILOR
Clasa MediaPlayer

• Redă clipuri media **locale** și la **distanță**
• Se bazează pe **stări**:
- – **inițializat**
  - • **setDataSource()**
- – **pregătit**
  - • **prepare()**
- – în **redare**
  - • **start()**
- – **întrerupt**
  - • **pause()**
- – **oprit**
  - • **stop()**

• **Inițializare**:
- – create() – metodă statică
- – setDisplay(SurfaceHolder)
- – setDataSource(src)

• **Redare**:
- – start()
- – stop()
- – pause()
- – setVolume(stinga, dreapta)

• **Eliberare resurse**
- – release()

Clasa MediaController

• Control care include butoane pentru:
- – redare/întrerupere, oprire, derulare înainte/înapoi, control progres

• **Definire**: **dinamic** sau în **XML**
• Sincronizat cu stările unui MediaPlayer
• **Afișare** controale
- – show()

• **Ascundere** controale

– hide()

- **Asociere** MediaPlayer
  - **setMediaPlayer**(MediaController.MediaPlayerControl)

Redarea clipurilor audio

```
//Redarea unui clip audio inclus în pachetul aplicației
MediaPlayer player = MediaPlayer.create(this, R.raw.muzica);
// sau player = MediaPlayer.create(this, uri);

player.setLooping(true);
player.start();
//eliberarea resurselor ocupate
if (player != null) {
        player.stop();
        player.release();
        player = null;
}
```

Clasa SoundPool

- Gestionează o **colecție** de **clipuri audio** de scurtă durată preluate din resurse și încărcate în memorie
- **Redare rapidă**, **consum redus** de resurse
- Posibilitate de r**edare simultană**
- **Metode**
  - load()
  - play(), stop()
  - autoResume(), autoPlay()
  - **setOnLoadCompleteListener()**

```
//nr. max de streamuri, tip stream, calitate - 0 implicit
SoundPool sp = new SoundPool(5, AudioManager.STREAM_MUSIC, 0);
//prioritate; 1 - implicit
int idSound1 = sp.load(this, R.raw.muzica1, 1);
int idSound2 = sp.load(this, R.raw.muzica2, 1);
sp.setOnLoadCompleteListener(new OnLoadCompleteListener() {

@Override
public void onLoadComplete(SoundPool soundPool, int sampleId, int status) {
        float lVol = 1f; float rVol = 1f;
        //id, vol dr., vol. st., prioritate (0 - min), loop (-1 continuu), rate 1 - normal
        //soundPool.play(sampleId, rVol, lVol, 0, 0, 1);}
});
```

Clasa AudioTrack

- Nivel scăzut
- Control
  - Frecvență, format, tip sunet (mono/stereo), dimensiune zonă tampon
- Posibilitate modificare dinamică a conținutului audio
  - Secvențe de tsonuri

REDAREA CLIPURILOR VIDEO

Clasa **VideoView**

- Control care permite **redare** clipurilor **video**
- Metode **control** clip:
  - start(), pause(), stopPlayback()
- Metode **informare**
  - getDuration(), getCurrentPosition(), isPlaying()
- **Asociere** clipuri
  - **setVideoPath**(String)
  - **setVideoUri**(Uri)
- **Asociere controale** control
  - **setMediaController**(MediaController)

```
//redarea unui clip video de pe card
String cale = "/sdcard/DCIM/test video.3gp";
VideoView vv = new VideoView(this);
MediaController mc = new MediaController(this);
vv.setMediaController(mc);
vv.setVideoPath(cale);
vv.requestFocus();
setContentView(vv);
vv.start();
```

ÎNREGISTRAREA CLIPURILOR

Clasa **MediaRecoder**

- Necesită **parcurgerea** unor **etape** bine-stabilite
  1. Stabilirea **sursei** de intrare: microfon, camera etc.
     - **setAudioSource()**
     - **setVideoSource()**
  2. Stabilirea fișierului de **ieșire**
     - **setOutputFile()**
  3. Stabilirea **formatului** acestuia
     - **setOutputFormat()**
  4. Stabilirea tipului de **codificare** audio/audio
     - **setAudioEncoder()**
     - **setVideoEncoder()**
  5. **Dimensiunea** clipului
     - **setVideoSize()**
  6. **Numărul** de **cadre** pe secundă
     - **setVideoFrameRate()**
  7. **Suprafața** pe care se previzualizează conținutul video
     - **setPreviewDisplay()**

<u>Înregistrarea clipurilor audio</u>

- Utilizare **MediaRecorder**
- Permisiunea: android.permission.**RECORD_AUDIO**
- **Selecție** surse de **intrare**

```
MediaRecorder aRecorder;
//...
try {
aRecorder = new MediaRecorder();
aRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
```

```
aRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
aRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AAC);
aRecorder.setOutputFile(NUME_FISIER_AUDIO);
aRecorder.setMaxDuration(3000);
aRecorder.prepare();
aRecorder.start();
} catch(Exception ex)
{ }
//Eliberarea resurselor
if (aRecorder != null) {
        aRecorder.stop();
        aRecorder.release();
        aRecorder = null;

}
```

UTILIZAREA CAMEREI FOTO/VIDEO

Camera

- **Acces** prin aplicații dedicate
  - **Intent**
- Utilizare clase specializate
  - **Camera**
- android.permission.**CAMERA**

Camera: aplicații **instalate**

- Acces prin intermediul mesajelor (**Intent)**
  - **Captare <u>imagini</u>**
    - Acțiune MediaStore.**ACTION_IMAGE_CAPTURE**
  - **Captare <u>video</u>**
    - Acțiune MediaStore.**ACTION_VIDEO_CAPTURE**

- **Lansare** aplicație
  - void **startActivityForResult**(Intent intent, int cod)
- **Preluare** rezultat
  - protected void **onActivityResult** (int **requestCode**, int **resultCode**, **Intent** intent)

1. **Fără** parametri
2. **Parametri**
   - Cheia: MediaStore.**EXTRA_OUTPUT**
   - Valoare: **Uri** (colecția de clipuri)
3. **Parametri**
   - Cheia: MediaStore.**EXTRA_OUTPUT**
   - Valoare: **Uri** (sistemul de fișiere)

Utilizarea aplicațiilor instalate

```
int CAPTEAZA_IMAGINE = 1;
private void fotoCameraIntent() {
Intent capteazaImagine = new
Intent(MediaStore.ACTION_IMAGE_CAPTURE);
startActivityForResult(capteazaImagine,
```

```
CAPTEAZA_IMAGINE);
}

private Bitmap fotografie = null;

...

@Override
protected void onActivityResult(int codCerere,int codRezultat, Intent data) {
if (resultCode == Activity.RESULT_OK && requestCode ==CAPTEAZA_IMAGINE) {
        Bundle rezultat = data.getExtras();
        fotografie = (Bitmap) rezultat.get("data");
        }
}
```

Utilizarea clasei **Camera**

- Pachetul **android.hardware**
- Clasa **Camera**
  - Depreciată începînd cu API 21
  - Înlocuită cu **android.hardware.camera2**
  - API **> 21** ~ 40% (aprilie 2016)
- **Inițializare**
  - Detectare camera
  - Acces la cameră
  - Activitate pentru vizualizare
  - Client pentru evenimente
- **Captură**
- **Salvare** fișiere
- **Eliberare** resurse

Clasa Camera

- **Inițializare**
  - **CameraParameters**
- Camera.**open**(idCamera)/open()
- Camera.**getNumberOfCameras**()
- **setPreviewDisplay**(SurfaceHolder)
- **lock()/unlock()**
- **takePicture()**

Captarea imaginilor

```
PictureCallback pcb = new PictureCallback() {
@Override
public void onPictureTaken(byte[] data, Camera camera) {
        try {
        /*Preluare șir de octeți imagine și prelucrare*/
        }
        catch (IOException ex) {
        /*tratare excepție */
        }
        }};
```

```
camera.takePicture(null, null, pcb);
```

Captarea conținutului video

```
Camera camera = Camera.open();
...
MediaRecorder vRecorder = new MediaRecorder();
//camera este eliberată pentru a putea fi accesată în aplicație
camera.stopPreview();
camera.unlock();
//asociere camera
vRecorder.setCamera(camera);
//stabilire sursa audio/video
vRecorder.setAudioSource(MediaRecorder.AudioSource.CAMCORDER);
vRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
vRecorder.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);

//stabilire codificare video/audio
vRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.DEFAULT);
vRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
vRecorder.setOutputFile(NUME_FISIER_VIDEO);
// sh este de tip SurfaceHolder
vRecorder.setPreviewDisplay(sh.getSurface());
vRecorder.prepare();
vRecorder.start();

//eliberare resurse
vRecorder.stop();
vRecorder.reset();
vRecorder.release();
vRecorder = null;
//reconectare serviciu camera
camera.reconnect();
```

Previzualizare

- Control special
- **Derivare** din clasa **SurfaceView**
- **Utilizează** un obiect de tip **SurfaceHolder**
- Utilizare interfața **SurfaceHolder.Callback** -notificări cu privire la modificare suprafeței:
  - **surfaceChanged()** - **inițializarea** camerei(parametri)
  - **surfaceCreated()** - **previzualizarea** imaginilor
  - **surfaceDestroyed()** - **eliberarea** resurselor

Utilizarea API camera2

- **android.hardware.camera2**
- CameraManager
- CameraDevice
- CameraCharacteristics
- CameraCaptureSession
- CaptureRequest

Cursul 9

Grafică <u>3D</u>

- Biblioteca **OpenGL ES**
  - Subset **OpenGL**
- Android, **suport** pentru versiunile
  - 1.x (API 1)
  - 2.0 (API 8)
  - 3.0 (API 18)
  - 3.1 (API 21)

<u>OpenGL ES</u>

- **Clasele** sînt definite în **pachetele**
  - android.**opengl**
  - android.**opengl.GLES10**
  - android.**opengl.GLES11**
  - android.**opengl.GLES20**
  - android.**opengl.GLES30**
  - android.**opengl.GLES31**
- **Interfețele OpenGL ES** sunt definite în **pachetele**
  - javax.microedition.khronos.**egl**
  - javax.microedition.khronos.**opengles**

Precizarea <u>versiunii</u> necesare

- **AndroidManifest.xml**
- Elementul **uses-feature**
- Atributul android:**glEsVersion**
- Valori
  - 0x000**2000**
  - 0x000**3000**
  - 0x000**3001**

<uses-feature
android:**glEsVersion**="0x000**20000**"
android:required="true" />

<u>Determinarea versiunii</u> OpenGL ES

ActivityManager am = (ActivityManager) **getSystemService**(Context.**ACTIVITY_SERVICE**);
ConfigurationInfo ci = am.**getDeviceConfigurationInfo**();
//ci.reqGlEsVersion: 0x20000, 0x30000, 0x30001

Matrice de transformare

- Dimensiunile **4 x 4**
- Matricea **model**
  - controlează transformările cu privire la **obiect** (model)
- Matricea **View**
  - controlează transformările cu privire la **cameră** (view)
- Matricea de **proiecție**
  - controlează transformările cu privire la **perspectivă**

Transformări

- Coordonatele **obiectului** (definit prin intermediul **vârfurilor**)
  - **transformate** în coordonatele **camerei**
- Coordonatele **camerei**
  - **transformate** în coordonatele **planului** de **decupare**
- Normalizarea **coordonatelor**
- Coordonatele **planului** de decupare
  - **transformate** în coordonatele **ecranului** (**fereastra** de vizionare).

Grafică 3D

- Componentă vizuală specializată
  - clasă care **implementează** interfața **GLSurfaceView**
- Operațiile de **desenare**
  - clasă care **implementează** interfața **GLSurfaceView.Renderer**

GLSurfaceView.Renderer

- Metode cu apel invers apelabile:
  - la **inițializarea** suprafeței
    - **onSurfaceCreated()**

      public void **onSurfaceCreated**(GL10 gl, EGLConfig config)
  - la **modificarea** suprafeței
    - **onSurfaceChanged()**

      public void **onSurfaceChanged**(GL10 gl, int width, int height)

      - Stabilire fereastră de vizionare
      - Stabilire perspectivă
  - la **desenare**
    - **onDrawFrame()**

      public void **onDrawFrame**(GL10 gl)

      • Pregătire suprafață

      • Stabilire poziție vizualizare

      - Desenare propriu-zisă

OPENGL ES 1.X

Grafică 3D

- Puncte **tridimensionale** și **matrice**
- Interfața **GL10**

Metode (**GL10**)

- **glVertexPointer**()
  - Definirea unui **vector** de **vîrfuri**
- **glClear**()
  - **ștergerea** suprafeței
- **glClearColor**()
  - stabilirea culorii de **fundal**
- **glDrawArrays**()
- **glDrawElements**()
  - **desenare** pe bază unui vector de puncte
- triunghiuri (**GL_TRIANGLE_STRIP**, **GL_TRIANGLE_FAN**),
- puncte (**GL_POINTS**),
- linii (**GL_LINES**, **GL_LINE_STRIPE**, **GL_LINE_LOOP**

OPENGL ES 2.0/3.X
OpenGL 2.0/3.x vs. 1.x

- Operațiile de **desenare** și **proiecție** sunt controlate de **programator**
- **Control** mai **bun** asupra prelucrărilor grafice **tridimensionale**
- Bazate pe programe (**shaders**) scrise în limbajul OpenGL Shader Language
- **Incompatibilitate**

Programe de tip shader

- **Compilate** și **rulate dinamic** de către unitatea de prelucrare grafică (**GPU**) a dispozitivului mobil
- Stabilesc **modul** în care sunt **redate** modelele **tridimensionale** de către procesorul grafic
- Definite pentru **redarea**
  - **vârfurilor** – vertex shader
  - **Fragmentelor** (pixelilor) – fragment shader

Etape din fluxul de **procesare tridimensionala**:

1. Varfuri
2. Rasterizare
3. Fragmente
4. Imagine

Încărcarea, compilarea și utilizarea programelor de tip shader

Exemplu Programe shader

vertexShaderCode	fragmentShaderCode

precision mediump float;
uniform mat4 u_mvpMatrix;
//matricea de transformare
attribute vec4 a_position; //pozitia
attribute vec4 a_color; //culoarea
varying vec4 v_color;
void main()
{
gl_Position = u_mvpMatrix
* a_position;
v_color = a_color;
}
	varying lowp vec4 v_color;
	void main()
	{
	gl_FragColor = v_color;
	}

Desenarea

- Clasele asociate controlului și suprafeței sînt aceleași
- Selectare **versiune**
  - **setEGLContextClientVersion**(vesiune)
- Metode statice **GL20**
  - Parametrul de tip **GL10 nu** este **utilizat**

Cursul 10

Bluetooth

- Tehnologia permite **schimbul** de **informații** fără fir între diferite **dispozitive**
- **Viteze** mult mai **mari** (spre deosebire de IR, NFC)
- Necesitatea **împerecherii dispozitivelor Bluetooth** în vederea comunicării
  - **pierde** la capitolul **ușurință** în **conectare**
- **Profile** –**specificații** pentru comunicații fără fir între dispozitive
- Radio frequency communication (**RFCOMM**)

Suport Android

- **Scanarea** pentru **descoperirea** de **dispozitive**
- Obținerea dispozitivelor **conectate** și **conectarea** la **alte dispozitive**
- **Transferul** de **date** între dispozitive
- Managementul **simultan** al **conexiunilor**

Etape în utilizarea Bluetooth în aplicații:

- **Activarea** Bluetooth
- **Identificarea** dispozitivelor **împerecheate** sau **disponibile**
- **Conectarea** dispozitivelor
  - Canale **RFCOMM**
- **Transferul** de **date** între dispozitive

Comunicarea prin **Bluetooth**

- **Descoperirea** reciprocă
- **Împerecherea**
  - dispozitivele **știu** fiecare de **existența** celuilalt
  - și-au **salvat** reciproc **adresele** pentru o viitoare comunicare
- **Comunicarea**
  - se realizează prin **socket**-uri folosind paradigma **client-server**
- Modul de comunicare poate fi descris și în cadrul unui profil (ex. "**mâini libere**")

Permisiuni

- android.permission.**BLUETOOTH**
  - **comunicare**
- android.permission.**BLUETOOTH_ADMIN**
  - **scanarea** dispozitivelor disponibile
  - **modificarea setărilor** Bluetooth

Clase Bluetooth API

- **BluetoothAdapter**
  - adaptorul local
- **BluetoothDevice**
  - dispozitiv Bluetooth la distanță
  - obținerea de informații: nume, adresa, clasa, starea etc.
- **BluetoothSocket**
  - interfața pentru un socket Bluetooth
- **BluetoothServerSocket**
  - **socket** de tip **server Bluetooth**, care permite conectarea clienților
  - așteaptă conexiuni noi
- **BluetoothClass**
  - descrie **caracteristicile** unui dispozitiv Bluetooth

BluetoothAdapter

- **Descoperirea** altor dispozitive **Bluetooth**
- Obținerea **listei dispozitivelor** deja **împerecheate**
- **Instanțierea** unui obiect de tip **BluetoothDevice** corespunzător unui alt dispozitiv
- Crearea obiecte de tip **BluetoothServerSocket** care așteaptă date prin socket de la alte dispozitive
- **getDefaultAdapter**()
  - inițializarea adaptorului Bluetooth disponibil
  - null – nu există adaptor pe dispozitiv
- **isEnabled**()
  - dacă serviciul este oprit, cerere de activare
- **ACTION_REQUEST_ENABLE**
  - acțiune pentru **activarea** serviciului
- **getBondedDevices**()
  - **lista** dispozitivelor **împerecheate**
    - Set<**BluetoothDevice**>
- **getRemoteDevice**(adresa)
- **startDiscovery**()
- **cancelDiscovery**()

BluetoothDevice

- **getName**()
  – numele dispozitivului
- **getAddress**()
  – adresa
- **getBluetoothClass**()

– clasa Bluetooth (tipul dispozitivului **PHONE_SMART, COMPUTER_WEARABLE**)

- **getBondState**()
  – starea împerecherii (**BOND_NONE, BOND_BONDING, BOND_BONDED**)
- **getType**()

– tipul dizpozitivului (**DEVICE_TYPE_CLASSIC, DEVICE_TYPE_LE DEVICE_TYPE_DUAL, DEVICE_TYPE_UNKNOWN**)

- **getUuids**()
  – identificatorii

Transferul datelor

- Server
- Client
- **Operații** care **blochează**
- Necesitatea utilizării **firelor de execuție**

Implementare server

- **Obținerea** unui obiect de tip **BluetoothServerSocket**
  – **listenUsingRfcommWithServiceRecord**(nume, uuid) din BluetoothAdapter
- Așteptarea **cererilor de conectare** și efectuarea **conexiunii**
  – **accept**()
- **Transfer** prin **socket**
- **Închiderea** conexiunii
  – **close**()

```
BluetoothServerSocket serverSocket = adaptor.listenUsingRfcommWithServiceRecord(nume,uuid);
BluetoothSocket socket = serverSocket.accept();
//Transfer prin socket
serverSocket.close()
```

Implementare client

- **Obținerea** unui obiect de tip **BluetoothDevice** asociat dispozitivului de la distanță
- **Obținerea** unui obiect de tip **BluetoothSocket**
  – **createRfcommSocketToServiceRecord**(uuid)
- **Inițierea** conexiunii
  – **connect**()
  – modul de căutare trebuie dezactivat în prealabil
- **Transfer** prin **socket**

```
Bluetooth socket = device.createRfcommSocketToServiceRecord(uuid);
socket.connect();
//Transfer prin socket
```

Transferul datelor

- Obținerea **fluxurilor** de **intrare/ieșire** asociate socket-urilor
  - **InputStream/OutputStream**
- **Citirea/scrierea** din/în **fluxuri**
  - **read**()/**write**()

InputStream in = socket.**getInputStream**()
OutputStream out = socket.**getOutputStream**()
in.**read**(byte[])
out.**write**(byte[])

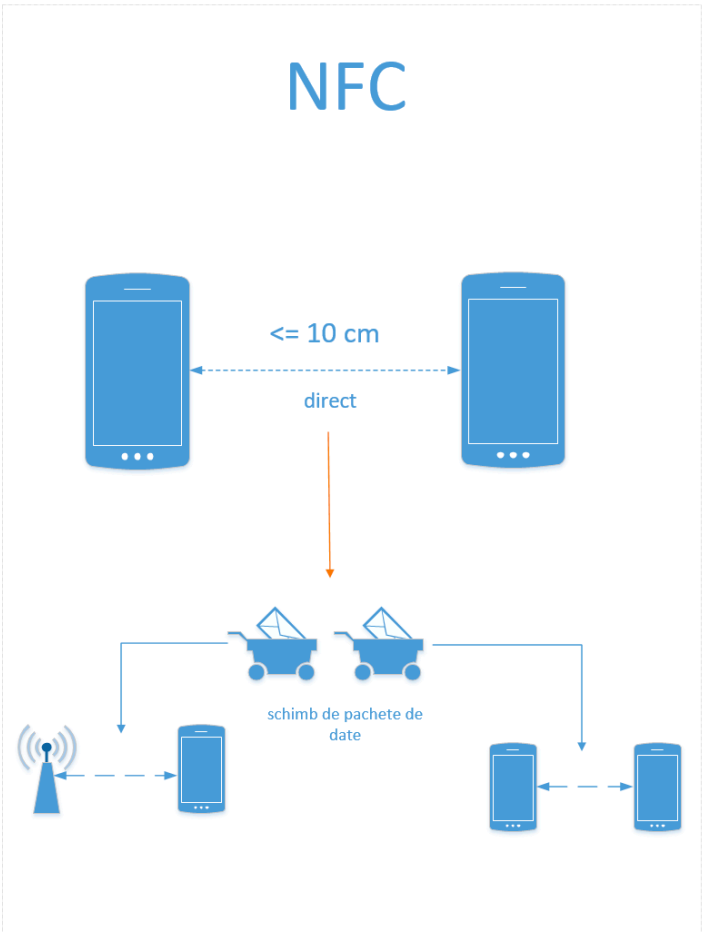
Alte clase Bluetooth API

- **BluetoothHeadset**
  - oferă **suport** pentru lucrul cu dispozitive de tip "**mîini libere**" sau căști bazate pe tehnologia Bluetooth
- **BluetoothA2dp**
  - definește modul în care poate fi transmis **sunet** de **înaltă calitate** prin intermediul unei conexiuni Bluetooth pe baza **A2DP** (Advanced Audio Distribution Profile)
- **BluetoothHealth**
  - **profilul HDP** (Health Device Profile) corespunzător **dispozitivelor medicale** care se pot conecta prin Bluetooth la un terminal Android

Cursul 11

NFC

- Consorțiul NFC Forum
- **Conectarea fără fir** a două dispozitive aflate la o **distanță** foarte mică (de maxim 10 cm)
- **Conectare** în mod **direct** (fără împerechere sau descoperire reciprocă)
- **Viteze** reduse (sute de Kbps)
- **Frecvența** 13.56 MHz
- Asigură **schimbul** de **pachete** de **date** între
  - **etichete** NFC și **dispozitive**
  - **dispozitive**



Moduri de <u>lucru</u>

- **Citire/scriere** (**RFID**)

  – dispozitivul poate **citi** sau **scrie** informații de pe sau pe etichete NFC pasive

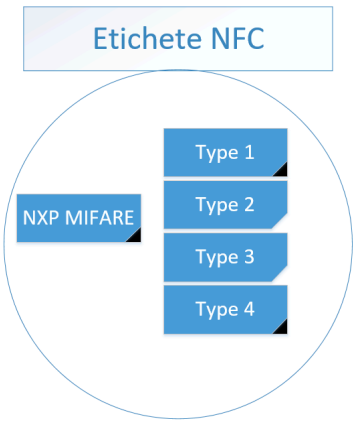
- **Peer-to-peer** (P2P)
  – dispozitivele NFC **schimbă date** între ele
  – mod folosit de **Android Beam**

- **Emulare card**
  – dispozitivul NFC se comportă ca un simplu card
  – folosit în special pentru **plăți electronice**

<u>Etichete</u> NFC

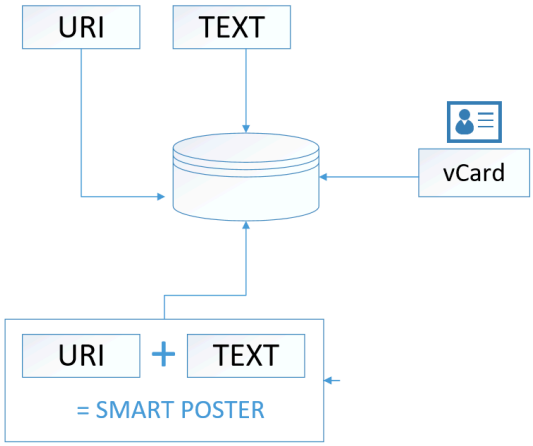
- **Type 1**
  – Bazat pe <u>standardul</u> **ISO14443 A**
  – Capacitatea <u>memoriei</u>: **96 B**, expandabilă la 2 KB
  – <u>Viteza</u> de transfer: **106 kbit/s**
- **Type 2**
  – Bazat pe standardul **ISO14443** A
  – Capacitatea **memoriei**: **48 B**, expandabilă la 2 KB

– **Viteza** de transfer: **106 kbit/s**

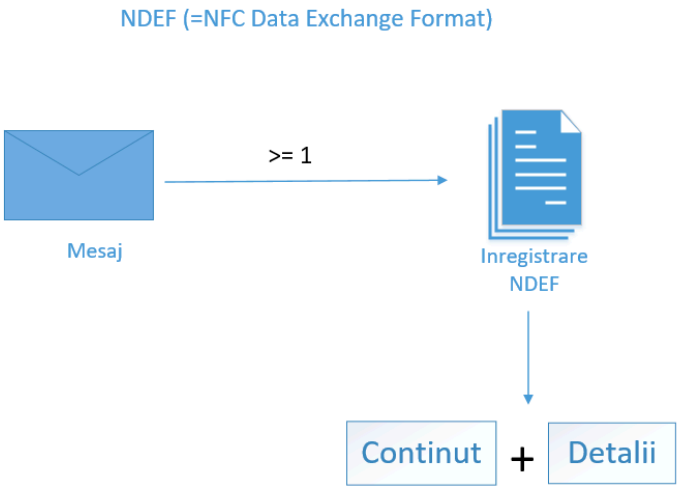


• **Type 3**
  – Bazate pe sistemul **Sony FeliCa**
  – **Configurate** de producător
  – Capacitatea **memoriei**: 2 KB
  – **Viteza** de transfer: 212 kbit/s
• **Type 4**
  – **Compatibil** cu standardele **ISO14443 A** și **B**
  – **Configurate** de producător
  – Capacitatea **memoriei**: pînă la **32 KB**
  – **Viteza** de transfer: între 106 kbit/s și 424 kbit/s
• **NXP MIFARE**
  – **Implementări** proprii ale standardului **ISO14443 A**
  • **Tipuri**
    – MIFARE Classic
    – MIFARE DESFire
    – MIFARE Plus
    – MIFARE Ultralight

NFC – Informatii **STOCATE**



NDEF (NFC Data Exchange Format)
• Formatul datelor transferate între dispozitive conforme NFC Forum
• **Independent** de **etichetă**
• Mesaj **NDEF**
  – **Conține** una sau mai multe înregistrări **NDEF**
• Înregistrări **NDEF**
  – Detalii despre conținut (tip, lungime etc.)
  – Conținutul propriu-zis



Suport NFC Android

- Clase dedicate
  - android.**nfc**
  - android.**nfc.tech**
  - android.**nfc.cardemulation**
- API bazat pe standardul **NDEF**
- Tehnologii suportate
  - NfcA, NfcB, NfcF, NfcV, IsoDep, Ndef, NdefFormatable
  - MifareClassic
  - MifareUltralight

Cerințe

- **Permisiune**
  - android.permission.**NFC**
- **uses-feature**
  - android.hardware.nfc
  - android.hardware.nfc.hce

Acțiuni sistem pentru NFC

- **ACTION_NDEF_DISCOVERED**
  - pornește o activitate atunci **când** o etichetă ce conține date NDEF este **scanată** și **recunoscută**
- **ACTION_TECH_DISCOVERED**
  - este pornit direct și în **cazul** în care
    - eticheta **scanată** conține informații din care **nu** se pot **extrage** date de tipul **MIME** sau **URI**
    - informația din etichetă **nu** este **formatată NDEF**, dar este corespunzătoare unei alte tehnologii cunoscute

– dacă **nu există activități** ce declară că pot manipula mesaje de tipul **ACTION_NDEF_DISCOVERED**, atunci sistemul de expediere al etichetelor încearcă să **pornească** o **activitate** cu acest **Intent**

• **ACTION_TAG_DISCOVERED**

– pornește **dacă nu** există activități care pot **manipula mesajele precedente**

Sistemul de expediere a etichetelor

• Se încearcă invocarea unei activitati cu un mesaj de tip Intent identificat la scanarea etichetei NFC

– ACTION_NDEF_DISCOVERED, ACTION_TECH_DISCOVERED

• Dacă **nu există** nicio activitate care să **manipuleze** aceste **tipuri** de **mesaje** atunci

– încearcă să **pornească** o **activitate** pentru **următorul** tip de **mesaj** declarat

• până când **găsește** o astfel de **activitate**

• sau până când **epuizează** toate tipurile de **mesaje**.

• Dacă **nu există** nicio activitate care să **manipuleze** niciunul dintre **mesaje**, atunci sistemul se **oprește**

Clase NFC

• **NfcAdapter**

- Asociată adaptorului NFC al dispozitivului
- Obținerea adaptorului
  – **getDefaultAdapter**(Context)
- Starea adaptorului
  – **isEnabled**()
- Starea Android Beam
  – **isNdefPushEnabled**()

• **Ndef**

Formatul NDEF

- **Datele** sînt **încapsulate** în interiorul unui obiect de tip **NdefMessage**
- Obiectul de tip **NdefMessage** conține unul sau mai multe obiecte de tip **NdefRecord**
- **Obiectele** de tip **NdefRecord** conțin

– **datele** și **tipul** informației conținute de înregistrare

• **NdefMessage**

• **NdefRecord**

- **tnf** – modul de **interpretare** (Type Name Format) al câmpului **type**; constante
  – **TNF_ABSOLUTE_URI** (pentru adrese absolute)
  – **TNF_MIME_MEDIA** (pentru conținut de tip media) etc.

• **type** –**tipul** înregistrării; constantele utilizate sunt

– **RTD_TEXT** (pentru conținut de tip text)

– **RTD_URI** (pentru conținut de tip Uri) etc.

• **id** – **identificator** unic pentru înregistrare (byte[])

• **payload** – **datele** efective care se doresc a fi citite sau scrise (byte[]);

– conținutul integral al mesajului poate fi inclus în mai multe înregistrări

SCRIEREA/CITIREA

Generarea obiectelor de tip NdefRecord

• Metode **statice** în clasa **NdefRecord**

• **createApplicationRecord**()

– aplicație definită prin pachetul complet (API 14)

• **createUri**()

– adresă referită printr-un obiect de tip Uri (API 14)

- **createMime**()
  – date precizate prin MIME (API 16)
- **createExternal**()
  – date externe aplicației (API 16)
- **createTextRecord**,
  – text codificat UTF8 (API 21)

Exemplu <u>scriere</u> text NFC:

```
byte[] payload;//va fi inițializat cu textul
// creare inregistrare noua
NdefRecord ndefRecord = new NdefRecord(
                NdefRecord.TNF_WELL_KNOWN,
                NdefRecord.RTD_TEXT,
                new byte[0],
                payload);
NdefRecord[] ndefRecords = { ndefRecord };
NdefMessage mesajNdef = new NdefMessage(ndefRecords);

//scrierea propriu-zisa
Ndef ndef = Ndef.get(tag);
ndef.connect();
ndef.writeNdefMessage(mesajNdef);
ndef.close();
```

<u>Citirea</u> etichetelor

- Atunci când un dispozitiv Android scanează o etichetă NFC încearcă **identificarea**
  – tipului **MIME** corespunzător datelor
  – unui obiect de tip **Uri**

Exemplu <u>citire</u> text NFC

```
tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
Ndef ndef = Ndef.get(tag);
if (ndef == null) { /*eroare, tag-ul nu suporta
ndef*/ }



else {
        NdefMessage mesajNdef = ndef.getCachedNdefMessage();
        NdefRecord[] ndefRecords = mesajNdef.getRecords();
        for (NdefRecord ndefRecord : ndefRecords) {
        // doar daca este de tip text
        if (ndefRecord.getTnf() == NdefRecord.TNF_WELL_KNOWN
        &&Arrays.equals(ndefRecord.getType(),
        NdefRecord.RTD_TEXT)) {
                        // obtinere continut
                        byte[] payload = ndefRecord.getPayload();
                        //preluare text din payload și utilizare
                        }
                }
```

}

MODUL PEER-TO-PEER

Transmiterea de mesaje peer-to-peer

- Trimiterea de mesaje între două dispozitive
- **Android Beam**
- Se creează un obiect de tip **NdefMessage**
  - **conține** cel puțin o înregistrare de tipul **NdefRecord** cu datele care vor fi transmise
- **Metode**
  - **setNdefPushMessage**()
    - **parametru**: obiectul de tip **NdefMessage** care va fi transmis
  - **setNdefPushMessageCallback**()
    - **asincronă**
    - apelată doar atunci **când** cele două **dispozitive** intră în raza de **comunicare**
    - obiectul de tip **NdefMessage** este creat doar atunci **când** acest lucru este **necesar**
    - **prioritatea** mai **mare** față de **setNdefPushMessage**()

***Exemplu trimitere* mesaj**

```
NfcAdapter nfcAdapter = NfcAdapter.getDefaultAdapter(this);
String mesaj; //se compune mesajul
NdefMessage mesajNdef = new NdefMessage(
        new NdefRecord[] { NdefRecord.createMime( “application/ro.ase.pdm.nfc",
        mesaj.getBytes()) });
nfcAdapter.setNdefPushMessage(mesajNdef, this);
class CNMCImpl implements CreateNdefMessageCallback {

@Override
public NdefMessage createNdefMessage(NfcEvent event) {
String mesaj;//mesajul va fi inițializat
    NdefMessage mesajNdef = new NdefMessage( new NdefRecord[]
        {
            NdefRecord.createMime( "application/ro.ase.pdm.nfc", mesaj.getBytes())});

            return mesajNdef;

        }
}

//In metoda onCreate() din activitate se inregistreaza apelul asincron
nfcAdapter.setNdefPushMessageCallback( new CNMCImpl (), this);
```

Recepționarea mesajelor NFC

- **Filtrul** de **mesaje**
  - **acțiunea**
    - android.nfc.action.**NDEF_DISCOVERED**
  - tipul **MIME** asociat aplicației
    - android:**mimeType**="application/com.pdm.nfc"
- Mesajul **NDEF** inclus în Intent
- Cheia NfcAdapter.**EXTRA_NDEF_MESSAGES**

Exemplu recepționare mesaj

```
String actiune = getIntent().getAction();
        if (actiune.equals(NfcAdapter.ACTION_NDEF_DISCOVERED)) {
                Parcelable[] mesajeNdef = intent.getParcelableArrayExtra(
                                        NfcAdapter.EXTRA_NDEF_MESSAGES);
// preluarea mesajului
NdefMessage mesajNdef = (NdefMessage) mesajeNdef[0];
String mesaj = new String(mesajNdef.getRecords()[0].getPayload());
//utilizarea mesaj
}
```

Cursul 12

Java Native Interface (JNI)

- **Platformă** care permite
  - apelul **funcțiilor native din** aplicațiile **Java**
  - **accesarea obiectelor** Java **din** funcțiile **native**
- Codul nativ are acces la funcționalitățile mașinii virtuale Java

JNIEnv

- **Interfața nativ – Java**
- **Pointer** la tabela de funcții cu funcțiile JNI disponibile
- Accesul la funcții care asigură interoperabilitatea între Java și codul nativ
- Toate metodele native au acces la această interfață

Metode interfață nativă

- **Creare/Conversii** șiruri
- Tratare **excepții**
- **Creare/copiere** vectori
- **Accesare** cîmpuri
- Apelare **metode**
- **Acces** la mașina virtuală Java
- **Sincronizare** execuție

Tipuri de date

- **Simple**
- **Referințe**
  - **Șiruri**
    - **Nu** pot fi utilizate **direct**
    - Necesită **conversie**
    - **Creare** șir nou (**nativ – Java**)
      - NewString()/NewStringUTF()
    - **Conversie** (Java – nativ)
      - GetStringChars()/GetStringUTFChars ()
      - ReleaseStringChars()/ReleaseStringUTFChars()
  - **Vectori**
    - **Creare** vector
      - **NewTipArray**()
    - **Copiere** vector (Java – nativ)

– **GetTipArrayRegion**()
- **Copiere vector** (nativ - Java)
– **SetTipArrayRegion**()
- Accesul direct
– **GetTipArrayElements**()
– **ReleaseTipArrayElements**()
– **Cîmpuri**
– **Metode**

Fișiere în cadrul proiectului

- Fișier **sursă Java**
  - Include **metodele** implementate în mod **nativ**
- Fișier **antent C/C++**
  - Include **semnăturile** metodelor **native**
  - Poate fi **generat** pe baza fișierului **Java** care le **expune**
- Fișier **sursă C/C++**
  - Include **implementarea** metodelor **native**

Fișierul Java cu metode native

- Încarcă **biblioteca** cu funcțiile **native**
- Expune funcțiile native aplicațiilor Java
- Declararea metodelor
  - **<vizibilitate>** native <tip returnat> nume_metoda(<lista parametri>);
- Metodele **statice** și de **instanță**

Încărcarea bibliotecilor

- System.**loadLibrary**()
  - Calea **curentă**
- System.**load**()
  - Calea trebuie **specificată** complet

Exemplu fișier Java cu metode native

```
public class TestNDK {
        static {
                System.loadLibrary("biblioteca");
}

public native String invText(String string);
public native int[] sortAsc(int[] vector);
}
```

Fișierele native

- Antetul metodelor native
- Include implementarea nativă a codului
- Toate metodele au acces la pointer la tabela de funcții cu funcțiile JNI disponibile (JNIEnv)
- Implementare diferită C/C++
  - .C, CPP

Metode native

- De **instanță**
  - **primesc contextul** în care are loc apelul

- echivalentul pointerului **this**

- **Statice**
  - **Primesc** un **parametru** asociat **referinței** clasei din care face parte metoda

Declararea metodelor în fișierul sursă nativ

JNIEXPORT <tip compatibil returnat> JNICALL
Java_<cale clasa>_nume_metoda(
JNIEnv * , jobject, <lista parametri>);

JNIEXPORT <tip compatibil returnat> JNICALL
Java_<cale clasa>_nume_metoda(
JNIEnv *, jclass, <lista parametri>);

Declararea metodelor native

- **JNIEXPORT** și **JNICALL**
  - destinate asigurării **compatibilității** cu mediul **Windows**
- **Tip compatibil returnat** de metodă
- **Numele metodei** are o structură impusă
  - prefixul **Java_**
  - **calea** completă separată cu "_" către clasa Java ce utilizează metoda nativă
  - **numele metodei** din Java
- **Parametrii** metodei native
  - **Pointer** la interfața **Java-Nativ** (JNINativeInterface), **JNIEnv**
  - **Referință** la obiectul cu tipul generic **jobject** sau **jclass**
  - Lista de **parametri** ai metodei Java

Accesare componentelor Java

- Apeluri **metode**
  - **GetMethodID**()
  - **CallTipMethod**()
- **Referire** cîmpuri
  - **GetFieldID**()
  - **SetTipField**()
  - **GetTipField**()

Instrumente utile

- **javah**
  - generează un **fișier antet** C/CPP (.h) corespunzător metodelor native
- **javap**
  - generează **semnăturile metodelor**

Android NDK (Native Development Kit)

- **Componentă** a **platformei** de dezvoltare **Android**
- Permite implementarea de **funcționalități** în cod **nativ**, în limbajele **C** sau **C++**
- **Funcționalități** cu **cerințe** de **performanță ridicate**
  - grafică **3D**
  - **prelucrarea** în timp real a **sunetelor** și **imaginilor**
  - **prelucrare video**
  - efectuarea de **calcule complexe**

- Criterii de alegere
  - **– cerințele aplicației**
  - – complexitatea și efortul necesar implementării native în raport cu sporul de performanță adusa

Etape

- **Instalare NDK/Verificare** instalare **NDK**
  - **Sursele** native sunt **stocate** în directorul **jni** din proiect
  - Se creează fișierul **Android.mk**
    - **– Descrie fișierele** sursă **native**
- **Compilarea** surselor **native**
  - – Scriptul **ndk-build**
  - => **Rezultă** o bibliotecă dinamică nativă (**.so/.dll)**
- **Compilarea** și **crearea** fișierului **binar** Android
  - – **Biblioteca nativă** este **inclusă** în fișierul binar (**apk**)

Componenta NDK

- Instrumente de compilare
- Fișiere antet
- Biblioteci
- Exemple
- Scripturi dedicate dezvoltării de module native

Variabile de mediu

- **NDK_ROOT**

Scripturi

- **ndk-build**
  - **– Compilarea** și **editarea** de legături
- **ndk-build clean**
  - **– Ștergerea** tuturor fișierelor rezultate

Android.mk

- **LOCAL_PATH**
  - **– Calea** către fișierele **sursă**
  - – Macrodefiniția **my-dir** întoarce **calea** către directorul care conține fișierul **Android.mk**
- directiva **include $(CLEAR_VARS)**
  - **– include** în fișierul curent conținutul unui fișier **Makefile** predefinit identificat de variabila **CLEAR_VARS**;
  - **– resetează** valorile variabilelor de tipul **LOCAL_*** folosite pentru a configura compilarea
- **LOCAL_MODULE**
  - – descrie numele modulului ce trebuie compilat;
  - – modulul rezultat va purta numele de **lib<nume_bibliotecă>.so**
- **LOCAL_SRC_FILES**
  - – indică **lista** de **fișiere** C sau C++ ce trebuie compilate
- directiva **include $(BUILD_SHARED_LIBRARY)**
  - **– include** în fișierul curent conținutul unui **script** predefinit care gestionează procesul de compilare și editarea de legături a modulului dinamic descris de variabilele LOCAL_*

Exemplu fișier Android.mk
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := <nume_bibliotecă>
LOCAL_SRC_FILES := <nume_ bibliotecă>.cpp
include $(BUILD_SHARED_LIBRARY)

Application.mk
• APP_OPTIM
• APP_ABI
  – APP_ABI := all
• APP_STL
• APP_CPPFLAGS
• APP_PLATFORM
• APP_BUILD_SCRIPT

Accesul la componente Android

jobject bitmap;
jclass clasa = env->FindClass("android/graphics/Bitmap");
jmethodID metLat = env->GetMethodID(clasa,"getWidth", "()I");
jmethodID metInalt = env->GetMethodID(clasa,
"getHeight", "()I");
jint lat = env->CallIntMethod(bitmap, metLat);
jint inalt = env->CallIntMethod(bitmap, metInalt);

Application Binary Interface (ABI)
• Stabilește **modul** în care aplicația **interacționează** cu sistemul la **execuție**
• Similar cu arhitectura setului de instrucțiuni
• Informații **furnizate**
  – Setul de **instrucțiuni** utilizat
  – **Formatul** fișierelor **executabile binare**
  – **Convenții** de apel
  – **Dimensiunile** și modalitatea de **aliniere** a datelor în **memorie**

Tipuri ABI
• **armeabi** – asociat procesoarelor **ARM** care **suportă** cel puțin setul de **instrucțiuni ARMv5TE**
• **armeabi-v7a** – asociat procesoarelor **ARM** cu s**uport extins** (Thumb-2, virgulă mobilă etc
• **arm64-v8a –**pentru procesoarea **ARM** care suportă **arhitectura AArch64**
• **x86 și x86_64** - asociate procesoarele bazate pe setul de **instrucțiuni x86** sau **IA-32,** respectiv **x86** pe **64 biți**
• **mips și mips64** – asociate procesoarelor cu arhitectura **MIPS32**, repectiv **MIPS64**

Cursul 13

Dispozitive portabile
• Procesor
  – Qualcomm SnapDragon 400 1.2 GHz
• Bluetooth 4.x

- 512 MB RAM
- 4GB memorie nevolatilă
- Baterie 400 mAh
- 1.3 inch, 320 x 320
- Android Wear instalată pe dispozitivul mobil
- Cele două dispozitive conectate folosind tehnologia Bluetooth
- Conectivitatea bazată pe dispozitivul mobil

Tipuri de aplicații pentru Android Wear

- Aplicații dedicate dispozitivelor portabile
- Carduri contextuale în fluxul de carduri
  - Notificări trimise de aplicațiile Android de pe dispozitivul mobil
  - Notificări contextuale de pe dispozitivul portabil (ceas)

Aplicații pentru Android Wear

- Necesită o aplicație companion
  - Aplicație Android standard
  - Numele pachetului trebuie să fie identic
- Aceleași permisiuni pentru ambele aplicații
- În Google Play, aplicația pentru dispozitivul portabil este inclusă în pachetul aplicației mobile
  - Instalare automată pentru
- Biblioteca Wearable UI
- Notificații
- Sincronizarea aplicațiilor
- Proiectarea ecranelor pentru ceas

Activități (Android Studio)

- Fără activitate
- Always On
- Blank Wear
- Display Notification
- Google Maps Wear
- Watch Face

API-uri nesuportate

- android.webkit
- android.print
- android.app.backup
- android.appwidget
- android.hardware.usb

ELEMENTE DE INTERFAȚĂ SPECIFICE

Activități și fragmente

- WearableActivity
  - Clasă de bază pentru activitățile specifice dispozitivelor portabile
- ConfirmationActivity
  - Activitate care afișează animații de confirmare după ce utilizatorul efectuează o anumită acțiune
- CardFragment
  - Fragment pentru prezentarea conținutului într-un card
  - Cardul este expandabil și se derulează pe verticală

Containere

- BoxInsetLayout
  - – Derivat din FrameLayout
  - – Adaptează conținutul după forma dispozitivului (rotundă sau dreptunghiulară)
- WatchViewStub
  - – Deserializează un container în funcție de tipul (forma) ecranului dispozitivului portabil (ceas)

Adaptoare și liste

- GridViewPager
  - – Permite navigarea verticală sau orizontală în cadrul paginilor
  - – Conținutul este furnizat de un obiect de tip GridPagerAdapter
  - – Indicarea pagini curente, față de celelalte pagini, este realizată printr-un control de tip DotsPageIndicator
- FragmentGridPagerAdapter
  - – Implementarea unui GridPagerAdapter, în care paginile sînt de tip fragment
- WearableListView
  - – Control de tip listă de elemente, adaptat pentru dispozitive portabile

Controale

- DismissOverlayView
  - – Control care implementează apăsarea prelungită pentru renunțare
- DelayedConfirmationView
  - – Afișează un cronometru descrescător pentru a permite confirmarea unei acțiuni

Controale imagini

- CircledImageView
  - – Control de tip ImageView înconjurat de un cerc
- CrossFadeDrawable
  - – Obiect de tip Drawable care conține două elemente Drawable și permite combinarea acestora

Notificări

- Crearea notificărilor
  - – Clasele
    - Notification și Notification.Builder sau
    - NotificationCompat și NotificationCompat.Builder
- Stiluri
  - – NotificationCompat.Style
    - BigPictureStyle, BigTextStyle, InboxStyle, MediaStyle
- Trimiterea notificărilor
  - – NotificationManager sau NotificationManagerCompat

Extensii notificări

- NotificationCompat.WearableExtender
  - – addPage()/addPages()
  - – addAction()/addActions()
  - – setBackground()
  - – setHintHideIcon()

• Notification.Builder
  – extend()

Transferul de informații

- Sincronizare
- Mesaje
- Fișiere
- Bazată pe GoogleApiClient
- Crearea clientului
  - Wearable.API
- Sincronizarea datelor
  - Wearable.DataItem
  - PutDataRequest
  - DataItem.DataListener
- Transmiterea mesajelor
  - Wearable.MessageApi

Programarea multiplatforma

Xamarin

- Limbajul C#
- Suport Windows, Mac
- Android SDK
- Android NDK

Platforme de dezvoltare

- Raspberry PI
- Arduino
- Intel Galileo

Cursul 14

Determinarea poziției geografice

- Serviciul de localizare Android
- Serviciul de localizare Google Location Services API

Serviciul de localizare Android

- Se utilizează serviciul de localizare identificat prin:
  - nume: Context.LOCATION_SERVICE
  - Se obține un obiect de tip LocationManager
- Localizare prin diferite surse:
  - Receptorul GPS
  - Rețele WiFi, celule mobile
- Permisiuni
  - android.permission.ACCESS_FINE_LOCATION
  - android.permission.ACCESS_COARSE_LOCATION
- Inițializare serviciu de localizare

- Asocierea unui obiect de tip listener cu precizarea:
  - Sursei utilizată la localizare
  - Intervalul de timp la care se face actualizarea datelor
- Notificări în momentul apariției de modificări legate de:
  - Poziția geografică
  - Starea sursei de localizare
- Întreruperea asocierii cu obiectul de tip listener
- Frecvența de actualizare!

Clasa LocationManager

- Mai multe supraîncărcări ale metodei requestLocationUpdates()
  - Asocierea unui obiect de tip LocationListener sau PendingIntent pentru informări cu privire la modificările poziției
  - Parametri comuni:
    - Intervalul minim de actualizare
    - Distanța minimă de actualizare (corelată cu intervalul de actualizare)
  - Alți parametri
    - Sursa de localizare:
      - GPS_PROVIDER sau NETWORK_PROVIDER din clasa LocationManager)
      - unui criteriu de selecție (clasa Criteria)
    - Obiectul care implementează interfața LocationListener sau obiectul de tip PendingIntent
- removeLocationUpdates(LocationListener/PendingIntent)
- Obținerea ultimei poziții cunoscute
  - Location getLastKnownLocation(String sursa)
- Informații despre starea receptorului GPS
  - GpsStatus getGpsStatus(GpsStatus status)

Interfața LocationListener

- Recepționarea de notificări la modificarea poziției geografice
  - Pe baza cerințelor de actualizare (interval și distanță)
- Metoda principală
  - void onLocationChanged(Location poz)
    - Coordonatele accesibile prin clasa Location

Clasa Location

- Coordonate
  - getLatitude(), getLongitude(), getAltitude()
- Viteza
  - getSpeed()
- Timp
  - getTime()
- Acuratețea localizării
  - getAccuracy()
- Alte informații

Poziționare geografică - *Exemplu*

```
//clasa va recepționa noua poziție
class GPSListener implements LocationListener {
        @Override
        public void onLocationChanged(Location poz) {
```

```
        /*poz.getLatitude(), poz.getLongitude(), poz.getAltitude(); */
}
//...
}
//crearea obiect utilizat pentru recepționarea actualizărilor poziției
GPSListener gl = new GPSListener();
//inițializare serviciu localizare
LocationManager lm = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
//asociere obiect recepționare actualizări
lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, gl);
//terminare asociere
lm.removeUpdates(gl);
```

SERVICIUL DE LOCALIZARE GOOGLE LOCATION SERVICES API

Poziționare geografică

- Google Play Services
- Fused Location Provider
  - Combină furnizorii disponibili în funcție de cerințele aplicației
- Precizie
- Consum de energie

– Selectarea celui mai potrivit serviciu

1. Determinare disponibilitate Google Play Services
2. Creare și inițializare client Google API
   a) GoogleApiClient
3. Implementare interfețe apeluri inverse pentru conectarea la servicii
   a) GoogleApiClient.ConnectionCallbacks
   b) GoogleApiClient.OnConnectionFailedListener
4. Implementare interfață pentru recepționarea poziției curente
   a) com.google.android.location.LocationListener

5. Asociere obiect pentru recepționarea poziției curente
   a) LocationRequest

Inițializare client Google Play Services

- GoogleApiClient
  - Integrarea serviciilor Google Play
- Inițializare
  - GoogleApiClient.Builder

GoogleApiClient.Builder

- addConnectionCallbacks()
  - GoogleApiClient.ConnectionCallbacks
- addOnConnectionFailedListener()
  - GoogleApiClient.OnConnectionFailedListener
- addApi()
  - LocationServices.API
- addScope()
  - OAuth 2.0

- build()
  - Creare obiect GoogleApiClient

Conectare GoogleApiClient

- Conectare
  - connect()
  - void onConnected(Bundle bundle)
- Deconectare
  - disconnect()

Inițializare LocationRequest

- setInterval()
  - Intervalul în care aplicația recepționează actualizările (ms)
- setFastestInterval()
  - Intervalul în care aplicația poate prelucra actualizările (ms)
- setPriority()
  - PRIORITY_HIGH_ACCURACY
    - Precizie maximă
  - PRIORITY_BALANCED_POWER_ACCURACY
    - Nivel de stradă (100 m)
  - PRIORITY_LOW_POWER
    - Nivel de oraș (10 km)
  - PRIORITY_NO_POWER
    - Poziția curentă este recepționată de la alte aplicații

Actualizare poziție geografică

- Interfața LocationListener
  - pachetul com.google.android.location.

– Recepționarea poziției curente

- public void onLocationChanged(Location location)
- LocationServices.FusedLocationApi.requestLocationUpdates(googleApiClient, locationRequest,this); //LocationListener

- LocationServices.FusedLocationApi.removeLocationUpdates(googleApiClient, this); //LocationLister

Google Maps

- Google Maps API v2
- Se bazează pe serviciul Google Maps
- Biblioteca opțională
- Inclusă în Google Play services
  - Categoria Extras în SDK Manager
- Google Play services APK trebuie să fie instalat și pe dispozitivul mobil
- Facilități
  - hărți 3D, interior, satelit, teren, trafic, marcaje, plane suprapuse, trasare linii etc.

Inițializare

- Înregistrare la Google Maps Service pentru obținerea unei chei pentru Maps API
  - https://code.google.com/apis/console
  - Cheia va fi inclusă în fișierul AndroidManifest.xml
- Instalare Google Play services SDK
- Adăugarea proiectului bibliotecii Google Play services la spațiul de lucru

- Referirea bibliotecii Google Play services în proiectul de lucru

Clase Google Maps API v2

- Pachetul com.google.android.gms.maps
- MapView
  - Control care încapsulează o hartă
- MapFragment
  - Fragment care încapsulează o hartă
- UiSettings
  - Control setări interfața hartă
- Marker
  - Pictograme pe hartă
  - Proprietăți
    - Titlu
    - Poziție (coordonate) – LatLng (latitudine și longitudine)
    - Pictogramă etc.
  - GoogleMaps#addMarker(MarkerOptions)
- MarkerOptions
  - Stabilire opțiuni obiect de tip Marker

Clasa GoogleMap

- Controlul hărții
- Inițializare
  - Metoda MapFragment#getMap() sau MapView#getMap()
- Control cameră
  - moveCamera(CameraUpdate)
  - CameraUpdateFactory
- Tip hartă
  - setMapType(tip_harta)
    - MAP_TYPE_NORMAL, MAP_TYPE_SATELLITE, MAP_TYPE_TERRAIN
- Control setări
  - getUiSettings()

Clasa GoogleMap - Adăgarea de figuri geometrice

- Cerc
  - Circle addCircle(CircleOptions)
- Linii
  - Polyline addPolyline(PolylineOptions)
- Imagine
  - GroundOverlay addGroundOverlay(GroundOverlayOptions)
- Poligon
  - Polygon addPolygon(PolygonOptions)

Fișierul xml asociat (res/layout)

- fragment
  - android:id="@+id/harta"
  - android:name="com.google.android.gms.maps.MapFragment"

Fișierul java asociat

```
((MapFragment) getFragmentManager().findFragmentById(R.id.harta))
.getMapAsync(new OnMapReadyCallback() {
```

```
@Override
public void onMapReady(GoogleMap map) {
//utilizare harta
});
```

Google Maps

• Fișierul AndroidManifest.xml

– Permisiuni (minim):

• INTERNET, WRITE_EXTERNAL_STORAGE,
com.google.android.providers.gsf.permission.READ_GSERVICES"

– Cheia Google Maps API

```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="cheia generata" />
```

Exemplu

```
if (map != null) {
    LatLng latLong = new LatLng(lat, longit);
    map.setMyLocationEnabled(false);
    map.moveCamera( CameraUpdateFactory.
    newLatLngZoom(latLong, 18));
Marker marker = map.addMarker(new MarkerOptions().position(latLong));
marker.setTitle(locatie);
marker.showInfoWindow();
}
```