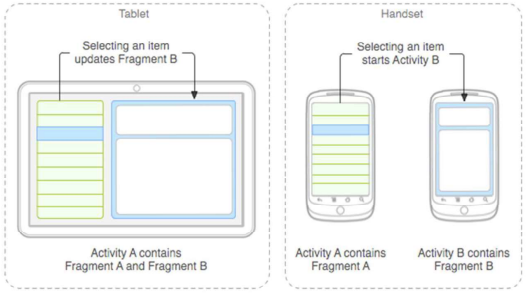
# Android – Fragments & Services

*Emmanuel Conchon*
*(emmanuel.conchon@unilim.fr)*

## Présentation générale

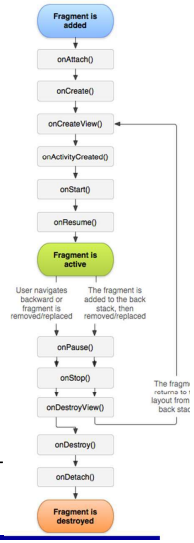
- Les fragments sont des composants Android permettant de s'adapter à une situation
  - Différents types d'équipement (tablette, téléphone, montre, TV …)
  - Différentes taille d'écran
  - Différentes orientation de ces écrans






## Présentation générale

- Pour créer un fragment il suffit d'hériter de la classe Fragment
  - Un fragment se comporte quasiment comme une activité
    - onAttach(): Permet de faire le lien avec son Activité
    - onCreate(): Ne crée pas l'interface graphique
    - onCreateView(): Crée et renvoie la vue du fragment
    - onActivityCreated(): Qui s'exécute lorsque l'activité est prête
    - onPause(): Lorsque l'on quitte le fragment
  - Le code que l'on retrouve dans le Fragment est similaire à celui de l'activité mais toutes les méthodes de celle-ci ne sont cependant pas présentes
    - Possibilité de les retrouver dans le Fragment avec la méhode getActivity()
  - De même, la gestion des handlers ne peut pas se faire dans le XML
    - Il faut donc les gérer dans le Java






## Présentation générale

- Le passage d'arguments vers un Fragment se fait à l'aide de l'activité qui l'embarque
  - Un Fragment ne discute jamais directement avec un autre Fragment
  - Un Fragment ne peut pas exister en dehors du cadre d'une activité
- Dans le même ordre d'idée, la création du fragment se déroule souvent après la création de l'activité à laquelle il est lié
  - Si le fragment a besoin de l'activité pour s'initialiser, il faudra privilégier la méthode onActivityCreated() à onCreate()
- Un Fragment peut être
  - statique
    - Son appartenance à une activité est définit dans un fichier Layout
  - dynamique

## Présentation générale

- Le patron de conception à utiliser pour l'utilisation des fragments repose sur quelques principes simple
  - Depuis l'API 22, l'activité qui doit manipuler des fragments doit hériter de FragmentActivity
  - L'activité connait les fragments qu'elle contient
    - Elle utilise le FragmentManager pour les retrouver et interagir avec eux
      - ★ Il est obtenu à partir de la méthode *getSupportFragmentManager()*
  - Le fragment peut appartenir à plusieurs activités il ne connait donc pas a priori l'activité qui le charge
    - C'est ce qui va permettre de gérer l'adaptation
    - Il peut la retrouver à l'aide de la méthode getActivity()



## Exemple

- Exemple d'activité contenant deux fragments définis dans un *Layout*

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">

    <fragment
        android:id="@+id/myListView"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:orientation="horizontal"
        android:name="fr.unilim.testfragment.MainActivityFragment">
    </fragment>

    <fragment
        android:id="@+id/Visualisation"
        android:name="fr.unilim.testfragment.ViewFragment"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent"
    />
</LinearLayout>
```



## Exemple

- Le layout du premier fragment

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp" tools:context=".ListFragment">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height=" fill_parent"
        android:orientation="vertical">

        <ListView
            android:id="@+id/myListView"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"/>

    </LinearLayout>

</RelativeLayout>
```



## Exemple

- Le layout du second fragment

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="fr.unilim.testfragment.ViewFragment">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Hello blank fragment" />

</RelativeLayout>
```

- Coté Activité

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    FragmentManager manager = getSupportFragmentManager();
    mainFragment=(ListFragment) manager.findFragmentById(R.id.Liste);
    viewFragment = (ViewFragment) manager.findFragmentById(R.id.Visualisation);
}
```

## Exemple

- Exemple de code minimal de fragment

```
import android.support.v4.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;


public class ListFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_main, container, false);
    }
}
```



## Chargement dynamique de fragments

- L'objectif est d'obtenir une application dynamique en fonction de l'orientation de l'écran et/ou de la taille


- Avec l'exemple précédent, les fragments sont définis dans des Layouts et sont fixes



## Chargement dynamique de fragments

- Pour le chargement dynamique deux classes vont être utilisées
  - Le FragmentManager qui permet de retrouver les fragments et que l'on obtient à l'aide de la méthode getSupportFragmentManager()
  - Les FragmentTransaction qui vont permettre à une activité de dialoguer avec les fragments
    - S'obtient à partir du FragmentManager avec la méthode beginTransaction()
    - La transaction permet de faire le lien entre le fragment et la vue dans laquelle il doit être affiché avec la méthode add()
    - La transaction doit finalement être validée à l'aide de la méthode commit()
- Le layout de l'activité doit également être repensé pour permettre l'adaptation
  - Utilisation d'un FrameLayout qui va servir ensuite de container de fragments

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```



## Chargement dynamique de fragments

- On spécifie ensuite dans la méthode onCreate de l'activité, les fragments à utiliser

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

    if (findViewById(R.id.fragment_container) != null) {
        // Si jamais une instance précédente existe on ne fait rien
        if (savedInstanceState != null) {
            return;
        }
        // On construit le fragment
        ListFragment firstFragment = new ListFragment();

        // Si jamais on doit passer des arguments au fragment
        firstFragment.setArguments(getIntent().getExtras());

        // On recupere le FragmentManager puis on ajoute
        // le fragment dans le FrameLayout
        getSupportFragmentManager().beginTransaction()
                .add(R.id.fragment_container, firstFragment).commit();
    }
}
```

## Remplacement dynamique de fragment

- Le remplacement et le passage d'information sont gérés au niveau de l'activité

```
// Create fragment and give it an argument specifying the article it should show
ArticleFragment newFragment = new ArticleFragment();
Bundle args = new Bundle();
args.putInt(ArticleFragment.ARG_POSITION, position);
newFragment.setArguments(args);

FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();

// Replace whatever is in the fragment_container view with this fragment,
// and add the transaction to the back stack so the user can navigate back
transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);

// Commit the transaction
transaction.commit();
```



## Echange d'informations entre fragments

- Les fragments ne doivent jamais discuter directement entre eux
  - Toute les communications doivent passer à travers l'activité qui les a chargés
- Pour la communication de l'activité vers le fragment le moyen le plus simple est de passer par le Bundle comme pour les Intents
- Pour la communication entre le fragment et l'activité, il va falloir construire le lien retour
  - On met en général en place une interface de communication que l'on définit au niveau du fragment



## Echange d'informations entre fragments

```
public class HeadlinesFragment extends ListFragment {
    OnHeadlineSelectedListener mCallback;

    // Container Activity must implement this interface
    public interface OnHeadlineSelectedListener {
        public void onArticleSelected(int position);
    }

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);

        // This makes sure that the container activity has implemented
        // the callback interface. If not, it throws an exception
        try {
            mCallback = (OnHeadlineSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString()
                    + " must implement OnHeadlineSelectedListener");
        }
    }

    ...
}
```



## Echange d'informations entre fragments

- L'activité qui contient le fragment va donc devoir implémenter l'interface de communication
  - Lorsque la méthode définit dans l'interface est appelée on peut utiliser les paramètres pour récupérer l'information du fragment
  - On peut ensuite appeler un autre fragment pour lui fournir de l'information

## Les sous classes de Fragment

- Pour simplifier un peu la conception plusieurs sous classes de Fragment sont disponibles
  - ListFragment : Définit une liste d'item similaire à ListActivity
    - Offre la méthode de callback onListItemClick() pour gérer les interactions
  - DialogFragment: Ouvre une fenêtre de dialogue par-dessus l'activité en cours
  - PreferenceFragment: Permet d'ouvrir une fenêtre pour changer les préférences de l'application
- Conseil: Regardez le tutoriel officiel Android et téléchargez l'exemple



# Les services



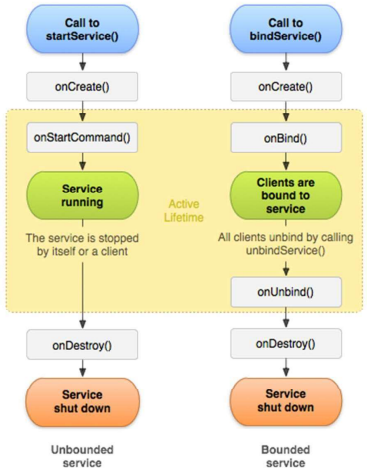
## Introduction

- Un service peut être vu comme une activité particulière ne disposant pas d'interface graphique
  - Téléchargement d'une pièce jointe dans les mails
  - Lecture d'un fichier de musique
  - Antivirus
  - …
- Pour cela, Android définit deux types de services
  - Les services standards
    - Pour les tâches longues qui doivent persister après la fermeture de l'application
  - Les services applicatifs
    - Pour des tâches courtes qui sont lancées par les activités via des Intent



## Le cycle de vie des services

- Un service est démarré par une activité et peut opérer dans deux modes
  - startService()
    - Fonctionne jusqu'à ce qu'on le stoppe manuellement
  - bindService()
    - Fonctionne jusqu'à ce que l'application soit stoppée
- Le cycle de vie d'un service se rapproche de celui d'une activité
  - onStartCommand()
  - onBind()



## Construction d'un service

- La création d'un nouveau service se fait en héritant de la classe Service ou de la classe IntentService
  - Service
    - classe de base qui fonctionne dans le Thread principal de l'application par défaut
      - ★ *Pour les requêtes multiples, il est nécessaire de créer un Thread dans lequel va se retrouver le code du service*
    - Se démarre en appelant la méthode startService()
    - Doit être arrêté manuellement
  - IntentService
    - Fonctionne dans un Thread séparé de celui de l'application
    - Méthode à privilégier s'il n'y a pas de requêtes multiples à gérer
    - Se démarre à l'aide d'un Intent
      - ★ *Déclenche la méthode onHandleIntent()*
    - S'arrête de manière automatique après avoir traité les demandes



## Exemple d'IntentService

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="fr.unilim.simplehelloservice" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="SimpleHelloService"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="SimpleHelloService" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name=".HelloService"
            android:exported="false" >
        </service>
    </application>

</manifest>
```



## Exemple d'IntentService

```
public class HelloService extends IntentService {

    private static final String TAG = "HelloService";

    public HelloService() {
        super("HelloService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        long endTime = System.currentTimeMillis() + 5*1000;
        while (System.currentTimeMillis() < endTime) {
            synchronized (this) {
                try {
                    wait(endTime - System.currentTimeMillis());
                    Log.d(TAG,"Hello");
                } catch (Exception e) {
                }
            }
        }
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.d(TAG, "service starting");
        return super.onStartCommand(intent,flags,startId);
    }
}
```



## Exemple d'IntentService

- L'activité déclenche un service à la manière d'une autre activité
  - Utilisation de startService() au lieu de startActivity()

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Intent intent = new Intent(this, HelloService.class);
    startService(intent);
}
```

- Dans le cas d'un fragment, il faut faire appel à l'activité dont il dépend pour lancer le service

```
Intent intent = new Intent(getActivity(),HelloService.class);
```

- L'utilisation d'un Intent permet de passer de l'information au service à l'aide des extras sur le même modèle que pour l'échange d'informations entre activités

## Les actions

- Lorsqu'un service peut réaliser plusieurs traitements on parle alors d'actions
  - Lecture, Pause, Arrêt pour un lecteur de musique par exemple

- Dans le système Android ces actions sont fournies à travers les Intents et prennent la forme d'une chaine de caractère

```
protected static final String HELLO="hello action";

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Intent intent = new Intent(this, HelloService.class);
    intent.setAction(HELLO);
    startService(intent);
}
```



## Les actions

- Du coté du service il est possible de connaître l'action ayant déclenché le service à travers la méthode getAction()

```
@Override
protected void onHandleIntent(Intent intent) {
    String action = intent.getAction();
    if (action==MainActivity.HELLO) {
        long endTime = System.currentTimeMillis() + 5 * 1000;
        while (System.currentTimeMillis() < endTime) {
            synchronized (this) {
                try {
                    wait(endTime - System.currentTimeMillis());
                    Log.d(TAG, "Hello");
                } catch (Exception e) {
                }
            }
        }
    }
}
```



## Les broadcasts

- Pour retourner un résultat, le service va utiliser un mécanisme de broadcast
  - Toutes les applications/activités vont devoir « s'abonner » à ce message pour recevoir l'information
  - On utilise encore la technique des actions pour différencier les résultats renvoyés par un service

```
Intent broadcastedIntent = new Intent();
broadcastedIntent.setAction("HELLO_BACK");
broadcastedIntent.putExtra("msg", "Hello From Service");
sendBroadcast(broadcastedIntent);
```

  - L'activité principale doit de son coté
    - créer un récepteur pour effectuer le traitement à la réception du broadcast
      - ★ *Définition d'une classe privée chargée de gérer cet évènement*
    - Enregistrer un filtre permettant de déclencher le récepteur
      - ★ *Utilisation de la méthode registerReceiver() et d'un IntentFilter()*



## Les broadcasts

- Extrait de l'activité principale déclenchant le service et attendant sa réponse

```
public class MainActivity extends AppCompatActivity {

    private class HelloReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            (Toast.makeText(context, intent.getStringExtra("msg"),Toast.LENGTH_SHORT)).show();
        }
    }

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

        IntentFilter filter = new IntentFilter();
        filter.addAction("HELLO_BACK");
        registerReceiver(new HelloReceiver(), filter);

        Intent intent = new Intent(this, HelloServiceBroadcast.class);
        intent.setAction(HELLO);
        startService(intent);
    }
```

## Les services liés (Bind Services)

- Les Bind Services sont des services permettant de fonctionner suivant un mode client serveur avec les activités
  - Les activités interrogent le service et reçoivent des réponses et peuvent également effectuer des communications inter processus (IPC)
  - Ce type de service permet souvent de dialoguer avec une autre application par exemple

- Plusieurs classes sont fournies par le système pour gérer ces services liés
  - onBind() qui renvoie un objet IBinder contenant l'interface que le client va pouvoir utiliser
  - Le client se connecte au service à l'aide de la méthode bindService()
  - Lorsque la connexion est établie la méthode onServiceConnected() est appelée coté client qui reçoit alors le IBinder



## Les services liés (Bind Services)

- Le service doit donc définir le IBinder qu'il va fournir aux clients

- Trois solutions sont possibles
  - Hériter de la classe Binder



## Les services liés (Bind Services)

```
public class LocalService extends Service {
    // Binder given to clients
    private final IBinder mBinder = new LocalBinder();
    // Random number generator
    private final Random mGenerator = new Random();

    /**
     * Class used for the client Binder.  Because we know this service always
     * runs in the same process as its clients, we don't need to deal with IPC.
     */
    public class LocalBinder extends Binder {
        LocalService getService() {
            // Return this instance of LocalService so clients can call public methods
            return LocalService.this;
        }
    }

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

    /** method for clients */
    public int getRandomNumber() {
      return mGenerator.nextInt(100);
    }
}
```



## Les services liés (Bind Services)

- Coté client il faut définir en interne une connexion au service

```
/** Defines callbacks for service binding, passed to bindService() */
private ServiceConnection mConnection = new ServiceConnection() {

    @Override
    public void onServiceConnected(ComponentName className,
            IBinder service) {
        // We've bound to LocalService, cast the IBinder and get LocalService instance
        LocalBinder binder = (LocalBinder) service;
        mService = binder.getService();
        mBound = true;
    }

    @Override
    public void onServiceDisconnected(ComponentName arg0) {
        mBound = false;
    }
};
```

## Les services liés (Bind Services)

- Il faut ensuite se connecter au service à la création de l'activité et se déconnecter à sa terminaison

```
@Override
protected void onStart() {
    super.onStart();
    // Bind to LocalService
    Intent intent = new Intent(this, LocalService.class);
    bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
}

@Override
protected void onStop() {
    super.onStop();
    // Unbind from the service
    if (mBound) {
        unbindService(mConnection);
        mBound = false;
    }
}
```



## Les services liés (Bind Services)

- A partir de là, le service est utilisable

```
/** Called when a button is clicked (the button in the layout file attaches to
  * this method with the android:onClick attribute) */
public void onButtonClick(View v) {
    if (mBound) {
        // Call a method from the LocalService.
        // However, if this call were something that might hang, then this request should
        // occur in a separate thread to avoid slowing down the activity performance.
        int num = mService.getRandomNumber();
        Toast.makeText(this, "number: " + num, Toast.LENGTH_SHORT).show();
    }
}
```



## Les services liés (Bind Services)

- Le service doit donc définir le IBinder qu'il va fournir aux clients

- Trois solutions sont possibles
  - Hériter de la classe Binder
  - Passer par un objet de type Messenger qui va s'appuyer sur un Handler
    - Permet la communication entre processus
    - Il va se charger de la diffusion d'objets Message entre les services et les clients



## Les services liés (Bind Services)

- Le service défit et implémente un Handler qui va être chargé de recevoir les requêtes des clients

```
public class MessengerService extends Service {
    /** Command to the service to display a message */
    static final int MSG_SAY_HELLO = 1;

    /**
     * Handler of incoming messages from clients.
     */
    class IncomingHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case MSG_SAY_HELLO:
                    Toast.makeText(getApplicationContext(), "hello!", Toast.LENGTH_SHORT).show()
                    break;
                default:
                    super.handleMessage(msg);
            }
        }
    }

    /**
     * Target we publish for clients to send messages to IncomingHandler.
     */
    final Messenger mMessenger = new Messenger(new IncomingHandler());
```

## Les services liés (Bind Services)

- Comme dans la solution précédente, le service fournit également une méthode onBind() renvoyant un IBinder au client
  - La différence tient au fait que le IBinder est obtenu à partir de l'objet messenger précédemment instancié

```
/**
 * When binding to the service, we return an interface to our messenger
 * for sending messages to the service.
 */
@Override
public IBinder onBind(Intent intent) {
    Toast.makeText(getApplicationContext(), "binding", Toast.LENGTH_SHORT).show();
    return mMessenger.getBinder();
}
```



## Les services liés (Bind Services)

- Coté client, il faut de nouveau définir une connexion qui va cette fois-ci devoir utiliser le Messenger

```
private ServiceConnection mConnection = new ServiceConnection() {
    public void onServiceConnected(ComponentName className, IBinder service) {
        // This is called when the connection with the service has been
        // established, giving us the object we can use to
        // interact with the service.  We are communicating with the
        // service using a Messenger, so here we get a client-side
        // representation of that from the raw IBinder object.
        mService = new Messenger(service);
        mBound = true;
    }

    public void onServiceDisconnected(ComponentName className) {
        // This is called when the connection with the service has been
        // unexpectedly disconnected -- that is, its process crashed.
        mService = null;
        mBound = false;
    }
};
```



## Les services liés (Bind Services)

- Toujours coté client, on retrouve l'établissement et la clôture de connexion

```
@Override
protected void onStart() {
    super.onStart();
    // Bind to the service
    bindService(new Intent(this, MessengerService.class), mConnection,
        Context.BIND_AUTO_CREATE);
}

@Override
protected void onStop() {
    super.onStop();
    // Unbind from the service
    if (mBound) {
        unbindService(mConnection);
        mBound = false;
    }
}
```



## Les services liés (Bind Services)

- Et enfin la phase de requête proprement dite au cours de laquelle le Message est construit puis envoyé au service

```
public void sayHello(View v) {
    if (!mBound) return;
    // Create and send a message to the service, using a supported 'what' value
    Message msg = Message.obtain(null, MessengerService.MSG_SAY_HELLO, 0, 0);
    try {
        mService.send(msg);
    } catch (RemoteException e) {
        e.printStackTrace();
    }
}
```

## Les services liés (Bind Services)

- Le service doit donc définir le IBinder qu'il va fournir aux clients

- Trois solutions sont possibles
  - Hériter de la classe Binder
  - Passer par un objet de type Messenger qui va s'appuyer sur un Handler
    - Permet la communication entre processus
    - Il va se charger de la diffusion d'objets Message entre les services et les clients
  - S'appuyer sur les AIDL (Android Interface Definition Language)
    - Solution complexe réservée à des cas très spécifique (voir documentation)



## Cycle de vie d'un Bind Service