APPLICATIONS MOBILES

INTERFACE GRAPHIQUE
FRAGMENTS

- À partir de android 3.0
- Composant graphique similaire à une activity.
- Un fragment est un composant qui fonctionne dans le contexte d'une activité.
- Une sorte de sous activité qui possède son propre cycle de vie.
- On peut intégrer plusieurs fragments dans une activités, et changer un fragment par un autre dynamiquement, sans changer l'activité principale.
- → programmation graphique dynamique plus aisée.

INTERFACE GRAPHIQUE
FRAGMENTS

Pour déclarer un fragment dans le layout
dans le tag *name*, il faut préciser la classe à utiliser à l'instanciation

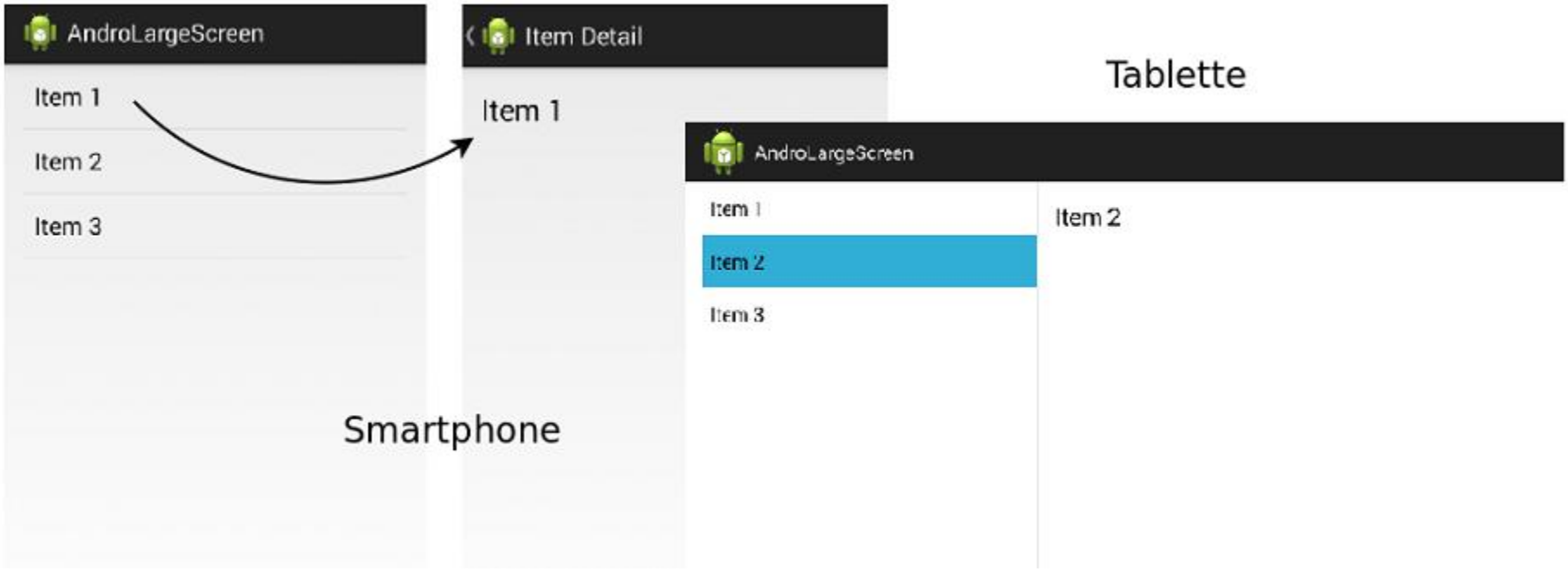
```
<LinearLayout ...>
  <fragment
      android:id="@+id/fragmentstatic"
      android:name="andro.jf.MonFragmentStatic" />
</LinearLayout>
```

La classe MonFragmentStatique contient le code permettant de générer la View

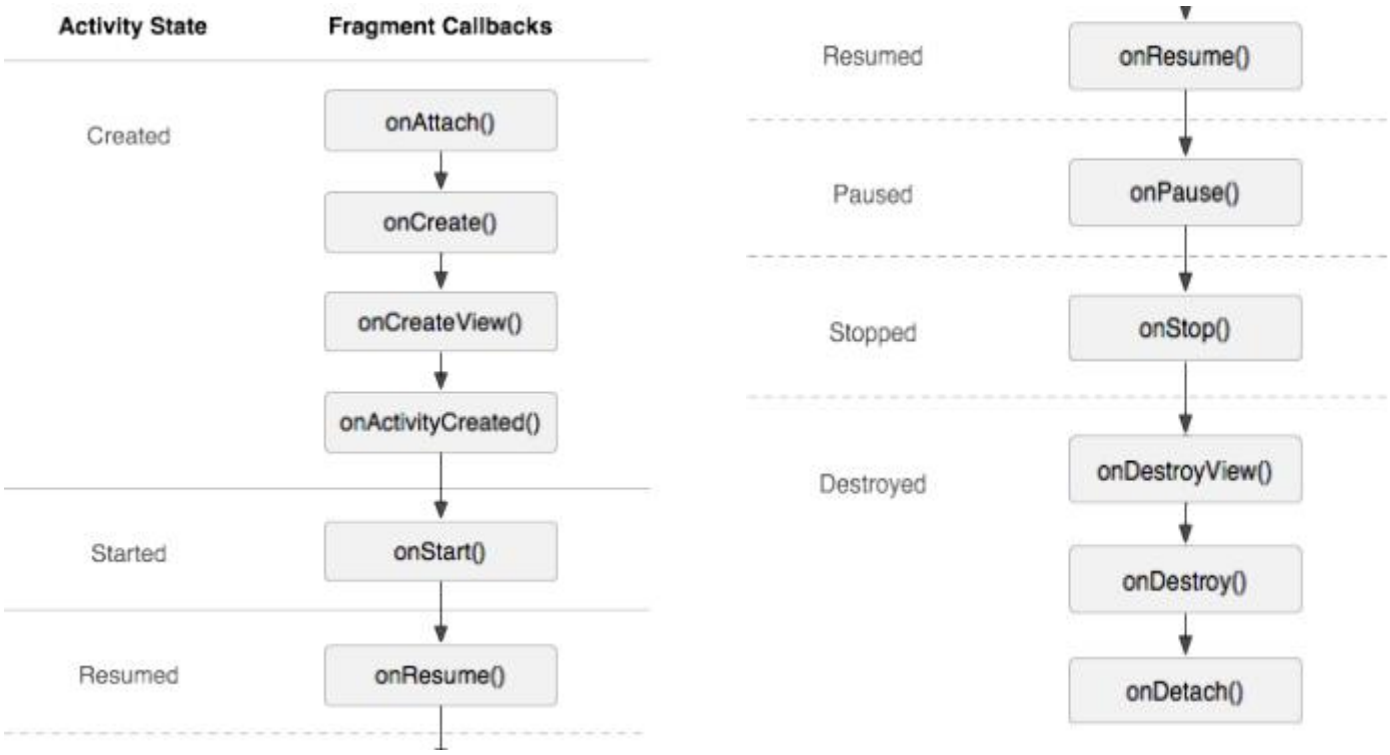
```
public class MonFragmentStatique extends Fragment {
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
View v = inflater.inflate(R.layout.fragment_main, container, false);
return v;
}}
```

INTERFACE GRAPHIQUE
FRAGMENTS

- Les fragments permettent aussi de gérer la diversité des appareils qui ne disposent pas des mêmes tailles d'écran



INTERFACE GRAPHIQUE
CYCLE DE VIE FRAGMENTS



- Ceci est une représentation du cycle de vie d'un fragment par rapport à celui d'une activity

INTENTS

- Permettent de gérer l'envoi et la réception des MSG afin de faire coopérer les applications
- C'est principalement un moyen d'activer un autre composant
- Un objet Intent contient :
  - nom du composant ciblé
  - l'action à réaliser, sous forme de chaine de caractères
  - les données : MIME ou URI
  - des données supplémentaires sous forme de paires clé/valeur
  - une catégorie pour cibler un type d'application
  - des drapeaux (infos supplémentaires)
- Intent (cas d'utilisation) :
  - Lancer une activité
  - Lancer un service
  - Diffuser une information

INTENTS
POUR UNE NOUVELLE ACTIVITÉ

- Plusieurs façons pour créer l'objet de type *Intent*
- Exemple :
  - lancement d'une activité interne à l'application

```
Intent login = new Intent(this, GiveLogin.class);
startActivity(login);
```

  - S'il s'agit de passer la main à une autre application
    - on donne au constructeur de l'Intent l'action et l'URI cible:

```
Button b = (Button)findViewById(R.id.Button01);
b.setOnClickListener(new OnClickListener() {
public void onClick(View v) {
  Uri telnumber = Uri.parse("tel:0248484000");
  Intent call = new Intent(Intent.ACTION_DIAL, telnumber);
  startActivity(call);
}
});
```

INTENTS
RETOUR D'UNE ACTIVITÉ

- Une activité peut vouloir récupérer un code de retour de l'activité « enfant »
- startActivityForResult() qui envoie un code avec l'Intent
- onActivityResult() vérifier le code renvoyer par l'activité « enfant »
- Exemple :
  - retour d'un appel téléphonique vers l'activité qui l'a lancé

```
public void onCreate(Bundle savedInstanceState) {
  Intent login = new Intent(getApplicationContext(), GivePhoneNumber.class);
  startActivityForResult(login,48);
... }
```

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
  if (requestCode == 48)
    Toast.makeText(this, "Code de requête récupéré (je sais d'ou je viens)",
                   Toast.LENGTH_LONG).show();
}
```

INTENTS
RÉSULTAT D' UNE ACTIVITÉ

- Définir un résultat d' activité avant d' appeler finish() pour finir l' activité
- setResult(code) : permet d' enregistrer un code de retour qu' il sera possible de filtrer dans l' activité parente

Dans l' activité enfant, on met :

```
Button finish = (Button)findViewById(R.id.finish);
  finish.setOnClickListener(new OnClickListener() {

  @Override
  public void onClick(View v) {
    setResult(50);
    finish();
  }});
```

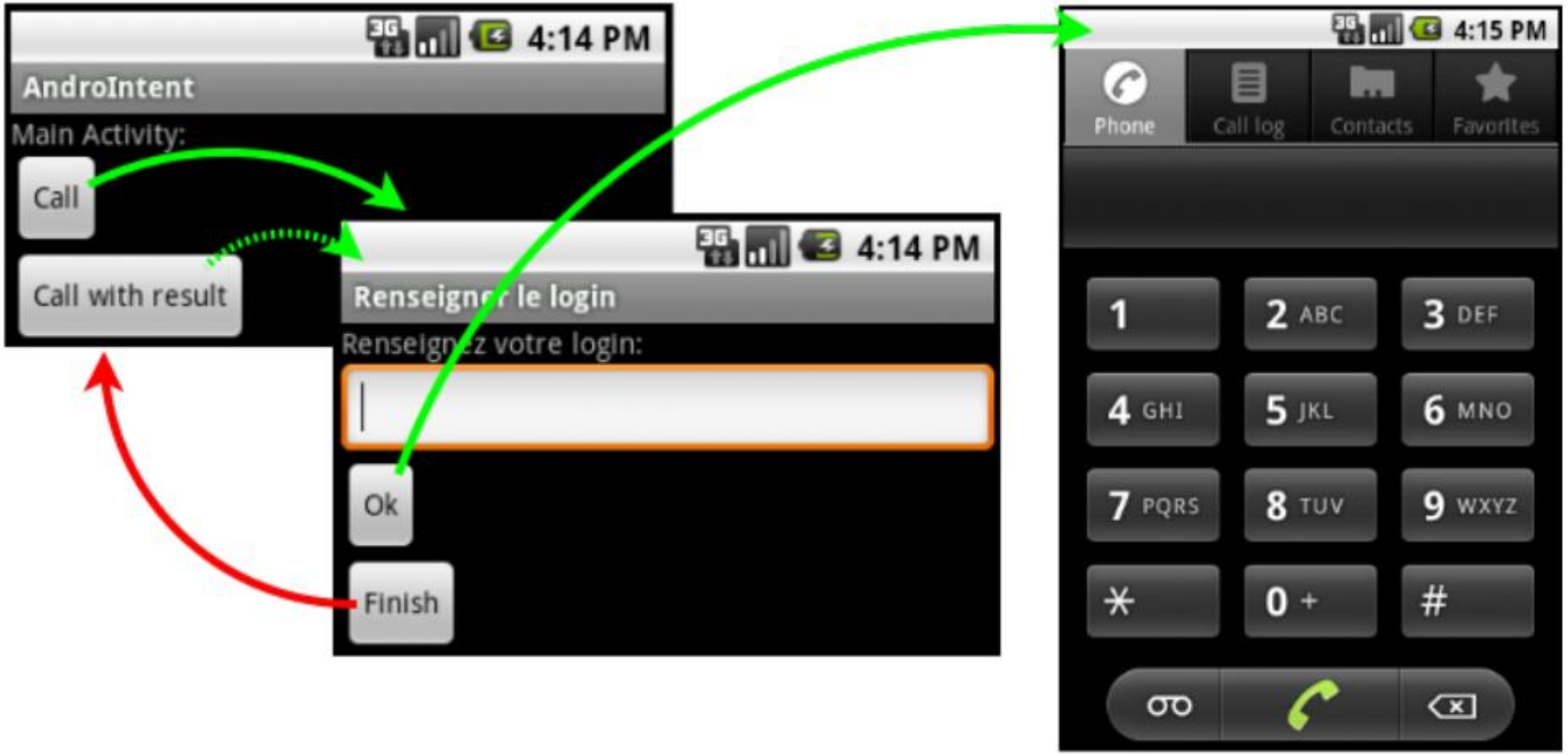
La classe parente peut filtrer ainsi :

```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    if (resultCode == 20) {
        Toast.makeText( context: this,  text: "je suis de retour (vérifié)", Toast.LENGTH_LONG).show();
    }
    else  {
        Toast.makeText( context: this,  text: "no thing!(non vérifié)", Toast.LENGTH_LONG).show();
    }

}
```

PRINCIPE DE LA DEMONSTRATION

Principe de la Démonstration



INTENTS
RÉSULTAT D'UNE ACTIVITÉ EXEMPLE

- Exemple :
  - Prendre une photo à partir d'une activité (lancer l'appareil photo)
  - Revenir avec un code de retour et la photo prise
- Démonstration

INTENTS
AJOUT D'INFORMATION

- Les Intents permettent de transporter des informations
- Ces informations sont appelées *Extra*
- Pour les manipuler : *putExtra()* et *getExtras()*
- Exemple : ajouter une info de type (clé / valeur)

```
Intent callactivity2 = new Intent(getApplicationContext(), Activity2.class);
callactivity2.putExtra("login", "jfl");
startActivity(callactivity2);
```

Pour récupérer l'information (coté de l'activité recevant l'Intent)

```
Bundle extras = getIntent().getExtras();
String s = new String(extras.getString("login"));
```

INTENT
ACTIONS

- Un des paramètres fondamentaux d'une Intent
  - L'action que l'Intent demande de faire (action qui s'est passée dans le cas d'un broadcast)
- Types d'action :
  - Actions prédéfinies par l'OS (actions natives)
  - Actions personnelles (définies par le développeur)
- Exemples:
  - ACTION_VIEW : pour visualiser une information
  - ACTION_SEND : pour envoyer une information
  - ACTION_PICK : pour choisir un élément parmi plusieurs

INTENT
ACTIONS (EXEMPLE)

- Exemple : envoi d'email

```
Intent emailIntent = new Intent(android.content.Intent.ACTION_SEND);
String[] recipients = new String[]{"my@email.com", "",};
emailIntent.putExtra(android.content.Intent.EXTRA_EMAIL, recipients);
emailIntent.putExtra(android.content.Intent.EXTRA_SUBJECT, "Test");
emailIntent.putExtra(android.content.Intent.EXTRA_TEXT, "Message");
emailIntent.setType("text/plain");
startActivity(Intent.createChooser(emailIntent, "Send mail..."));
finish();
```

Exemple d'action personnelle

```
Intent monIntent = new Intent("andro.jf.nom_du_message");
```

INTENT
CATÉGORIE

- Les catégories d'*Intent* permettent de grouper les applications par grands types de fonctionnalités (clients emails, navigateurs, players de musique, etc...)
- Exemple :
  - DEFAULT : catégorie par défaut
  - BROWSABLE: une activité qui peut être invoquée depuis un clic sur un navigateur web
  - APP_MARKET: une activité qui permet de parcourir le market et de télécharger des applications
  - APP_MUSIC: une activité qui permet de parcourir et jouer de la musique
  - … etc.

INTENTS
BROADCAST

- Utiliser un objet Intent pour broadcaster un message à but informatif
  - But : toutes les applications pourront capturer l'info diffusée
- Exemple:

```
Intent broadcast = new Intent("andro.jf.broadcast");
broadcast.putExtra("extra", "test");
sendBroadcast(broadcast);
```

INTENTS
RECEVOIR ET FILTRER LES INTENTS

- Vu la multitude de message véhiculés par les Intents
- Android dispose d'un système de filtres déclaratifs
  - Pour faciliter aux applications d'écouter que les Intents dont elles ont besoin
- Un filtre peut préciser :
  - **Action** : filtre sur les actions que l'application peut faire
  - **Catégorie** : permet de filtrer une catégorie d'action (DEFAULT, BROWSABLE, ...)
  - **Data** : filtre sur les données de l'Intent

INTENTS
FILTRER SUR L'ACTION

En déclarant un filtre au niveau du tag **activity**, l'application déclare les types de messsage qu'elle sait gérer et qui l'invoquent.

```
Button autoinvoc = (Button)findViewById(R.id.autoinvoc);
autoinvoc.setOnClickListener(new OnClickListener() {
@Override
public void onClick(View v) {
  Intent intent = new Intent("andro.jf.nom_du_message");
  startActivity(intent);
}});
```

Ainsi, l'application répond à la sollicitation des *Intents* envoyés par:

```
<activity android:name=".Main" android:label="@string/app_name">
  <intent-filter>
    <action android:name="andro.jf.nom_du_message" />
    <category android:name="android.intent.category.DEFAULT" />
  </intent-filter>
</activity>
```

INTENTS
RECEPTION PAR BROADCAST RECEIVER

- Les messages broadcastés par les applications sont réceptionnées par une classe héritant de **BroadcastReceiver.**
- Cette classe doit surcharger la méthode **onReceive.**
- Dans le Manifest, un filtre doit être inséré dans un tag **receiver** qui pointe vers la classe se chargeant des messages.

```
<application android:icon="@drawable/icon" android:label="@string/app_name">
  <receiver android:name="MyBroadcastReceiver">
    <intent-filter>
              <action android:name="andro.jf.broadcast" />
              <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
  </receiver>
</application>
```

INTENTS
BROADCAST RECEIVER

La classe héritant de **BroadcastReceiver**:

```
public final class MyBroadcastReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    Bundle extra = intent.getExtras();
    if (extra != null)
    {
      String val = extra.getString("extra");
      Toast.makeText(context, "Broadcast message received: " + val,
                     Toast.LENGTH_SHORT).show();
}}}
```

BIBLIOGRAPHIE

- Applications Mobile; cours de R.meghatria et M.Khaled ; UDBKM 2018
- **Développement Android - Jean-Francois Lalande** - March 2019
- developer.android.com 2021.