Chapitre 5

Les interfaces utilisateurs avec Android (fin)

Plan du chapitre 5

- La gestion des événements
- Enchaîner les écrans
- `Toast` et traces
- L'internationalisation

La gestion des événements

- Deux moyens :
  - 1°) créer un auditeur d'événements (classe qui implémente une interface connue) et l'enregistrer auprès du composant (`View`)
  - 2°) les `View` sont elles mêmes auditrices de certains événements : (touché de l'écran). Spécialiser la méthode adaptée et lancée lorsque l'événement survient
- 1°) est classique (Java SE, Java ME). Les interfaces sont des interfaces internes à la classe `View` et de nom `OnXXXListener` (donc des interfaces de nom `View.OnXXXListener`). Cela nécessite d'implémenter une méthode de nom `onXXX()`. On enregistre un auditeur par `setOnXXXListener(View.OnXXXListener l)`
- 2°) permet d'écrire directement la gestion de certains événements qui peuvent se produire dans la `View`

Créer un auditeur d'événements : exemple

- Le code peut être :

```
private OnClickListener lAuditeurDuBouton = new OnClickListener() {
    public void onClick(View v) {
      // code lancé lorsque le bouton est cliqué
    }
}

protected void onCreate(Bundle savedValues) {
    ...
    // Récupération du Button à partir de l'IHM en XML
    Button button = (Button)findViewById(R.id.leBeauBouton);
    // Enregistrer l'auditeur auprès du bouton
    button.setOnClickListener(lAuditeurDuBouton);
    ...
}
```

- Evidemment avec une classe anonyme, cela fonctionne aussi :

```
protected void onCreate(Bundle savedValues) { ...
     button = (Button)findViewById(R.id. leBeauBouton);
     button.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View view) {
               // code lancé lorsque le bouton est cliqué
          }
     });
}
```

Méthodes lancées par les auditeurs d'événements

- `onClick()` (de `View.OnClickListener`) est lancée lorsque l'utilisateur touche le composant graphique, ou après appui sur enter alors que le composant a le focus
- `onLongClick()` (de `View.OnLongClickListener`) : idem que si dessus mais après un appui de plus de 1 seconde
- `onKey()` (de `View.OnKeyListener`) est lancée après appui et relachement d'un touche clavier
- `onTouch()` (de `View.OnTouchListener`) est lancée pour toute action de toucher (appui, relachement, mouvement de l'utilisateur sur l'écran)
- `onCreateContextMenu()` (de `View.OnCreateContextMenuListener`) est lancée pour créer un menu contextuel

L'attribut `android:onClick`

- On peut indiquer dans le fichier `.xml` de description d'IHM, la méthode qui sera lancée sous une certaine action sur un composant graphique
- Par exemple, l'attribut `android:onClick` d'un composant graphique indique le nom de la méthode qui sera lancée si on clique sur cette `View`
- Par exemple, dans le fichier de description de l'IHM, on écrit

```
<Button
    android:id="@+id/push"
    ...
    android:onClick="onClickEmpiler">
</Button>
```

- Dans l'activité chargeant l'IHM contenant ce `Button`, on pourra écrire :

```
public void onClickEmpiler(View v){
    // traitement lancé lorsque l'utilisateur clique sur le Button d'id push
}
```

Plus sur les `Intents`

- Un `Intent` est une description abstraite d'une opération à faire
- Un `Intent` peut être utilisé avec :
  - `startActivity(Intent i)` pour demander à lancer une activité
  - `sendBroadcast(Intent i)` pour demander à contacter un `BroadcastReceiver`
  - `startService(Intent i)` ou `bindService(Intent i, ...)` pour communiquer avec un `Service`
- Essentiellement un `Intent` est formé :
  - d'une description d'action à effectuer
  - de données utiles pour cette action
- source : `https://developer.android.com/reference/android/content/Intent.html`

"Enchaîner" les écrans (1/2)

- Pour passer d'un écran à un autre, il faut écrire le code

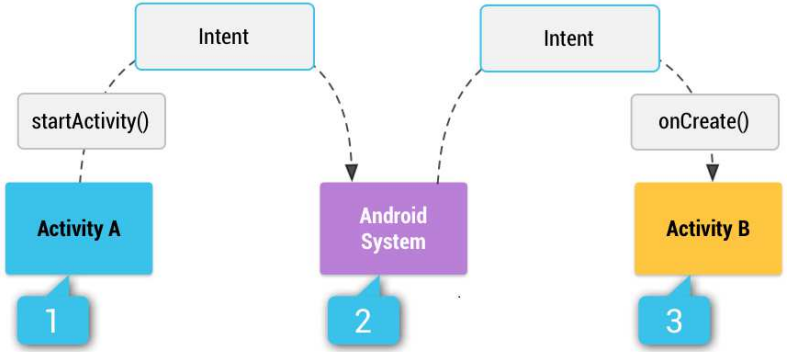
```
Intent i0 = new Intent(getApplicationContext(), NouvelleActivity.class); //1
startActivity(i0);
```

  et déclarer la nouvelle activité `NouvelleActivity.class` (le futur écran) dans `AndroidManifest.xml`

- `public void startActivity (Intent intent)` est une méthode de la classe `Activity` permettant de lancer une autre Activity (qui affichera un nouvel écran). `intent` est l'`Intent` (l'intention) qui prépare ce lancement

"Enchaîner" les écrans (2/2)

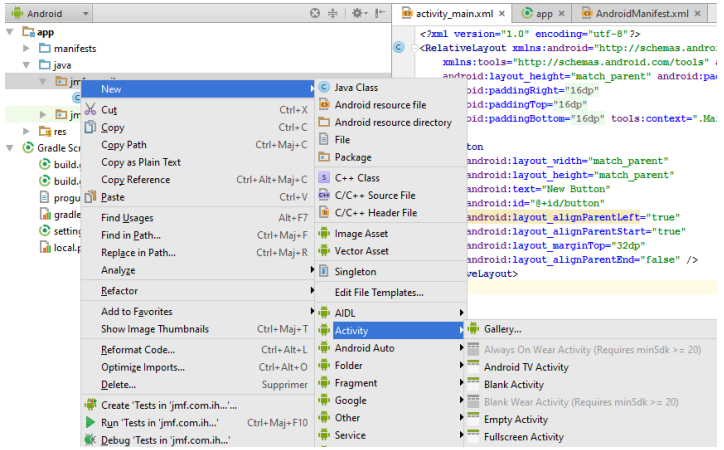
- En fait cet `Intent` est envoyé à l'environnement d'exécution. Celui-ci le redirige vers l'activité concernée



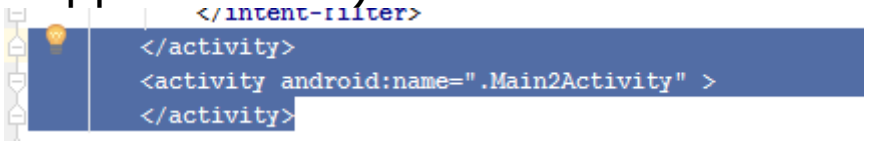
- Le premier argument du constructeur de l'`Intent` doit être le `Context`. Si l'appel est fait dans une activity `MonActivity`, `this` est souvent utilisé et convient car `Activity` dérive de `Context`
- On utilise souvent `MonActivity.this` quand on est dans un listener d'événement. Mais, la méthode `getApplicationContext()` est la plus pratique

Créer une nouvelle activité

- Dans un projet, on peut créer une nouvelle activité comme indiqué dans les diapos précédentes : écrire une classe dérivant de `Activity`, redéfinir les méthodes `onCreate()`, ... et déclarer cette activité dans l'`AndroidManifest.xml`
- Avec l'environnement de développement, si on utilise clic droit sur le répertoire java paquetage | New | Activity,



en complétant les écrans qui suivent, l'activité est en partie créée et sa déclaration correcte (fils de l'élément `application`) dans l'`AndroidManifest.xml` aussi !



Passer de données entre activités grâce aux `Intent`

- Les `Intent` servent parfois d'enveloppes pour passer des informations d'une `Activity` à une autre. On utilise pour cela une des méthodes `public Intent putExtra(String nomDeLExtra, unType valeur)`
- Par exemple

```
Intent i = new Intent(leContexte, MapStationActivity.class);
i.putExtra("latitude", latitudeDuPointCourant);
i.putExtra("longitude", longitudeDuPointCourant);
startActivity(i);
```

- Dans une `Activity`, on récupére l'`Intent` qui a lancé l'`Activity` par `getIntent()`. On peut alors récupérer tous les extras de l'`Intent` par `getExtras()`, et, par la suite, un extra associé à une entrée par `get`*`TypeEntrée`*`(nomEntrée, valeurParDefaut)`, `valeurParDefaut` est la valeur retournée si il n'y a pas d'extra associé à `nomEntrée` dans l'`Intent`
- Par exemple :

```
double laLatitudeDeLaStation =
getIntent().getExtras().getDouble("latitude", 0);
```

Un simple avertissement : `Toast`

- Une fenêtre de dialogue qui affiche un message pendant 2 (`Toast.LENGTH_SHORT`) ou 3,5 (`Toast.LENGTH_LONG`) secondes est un composant graphique Android : le Toast
- On le construit et on l'affiche avec le code

```
Toast leToast = Toast.makeText(leContexte, "texteAAfficher", Toast.LENGTH_LONG);
leToast.show();
```

- Une des méthodes qui construit un `Toast` est la méthode statique :
  `public static Toast makeText (Context context, CharSequence text, int duree)`
  `context` est le contexte à utiliser. En général on passe l'activité courante
  `text` est la chaîne de caractères à afficher
  `duree` est la durée d'affichage `LENGTH_SHORT` ou `LENGTH_LONG`
- Attention construire le `Toast` ne l'affiche pas : il faut utiliser `show()` pour cela
- Et donc finalement on écrit souvent :

```
Toast.makeText(leContexte, "texteAAfficher", Toast.LENGTH_LONG).show();
```

Code de "trace" en Android

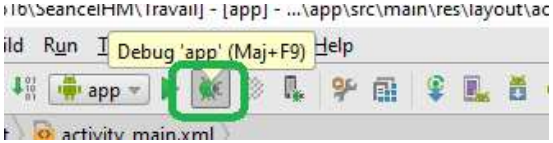
- La classe `android.util.Log` propose plusieurs méthodes de trace (de Log) hierarchisées. Ces méthodes ont pour nom une seule lettre. Ce sont, dans l'ordre, les méthodes `v()` (verbose), `d()` (debug), `i()` (information), `w()` (warning) et `e()` (erreur)
- Ces méthodes ont deux arguments : (`String tag, String msg`)
- Elles permettent de visualiser des traces lors de l'exécution en utilisant l'onglet LogCat. On peut filtrer ces traces en ne laissant afficher que les log de balise `tag`
- Par exemple le code :

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
    case R.id.new_game:
        Log.v("JMF", "Une nouvelle partie ?");
        Log.d("JMF", "Une nouvelle partie ?");
        Log.i("JMF", "Une nouvelle partie ?");
        Log.w("JMF", "Une nouvelle partie ?");
        Log.e("JMF", "Une nouvelle partie ?");
    return true;
    // ...
    }
}
```

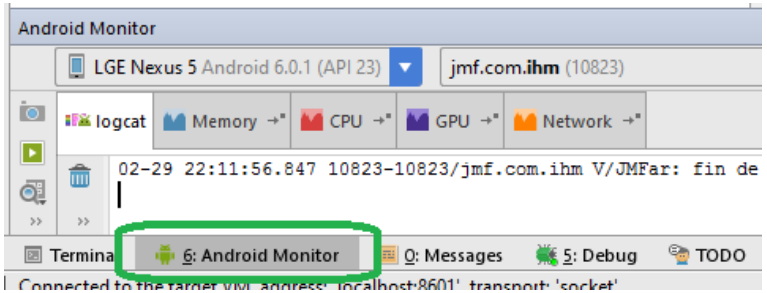
permet de voir les sorties dans l'onglet LogCat

L'onglet logcat

- Les traces de Log sont affichés dans l'onglet logcat
- Pour afficher cet onglet, il faut lancer l'exécution en mode Debug : bouton



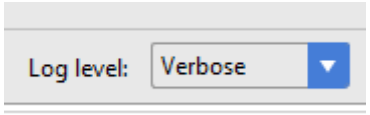
- Faire afficher l'onglet 6: Android Monitor (en bas de l'IDE)



- Il y a alors plusieurs onglets disponibles utiles lors de l'exécution de l'app dont l'onglet logcat
- On peut faire des filtres sur les sorties dans la zone de texte de recherché de cet onglet

Traces (log) en Android

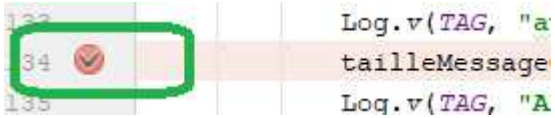
- On indique le niveau de trace dans la liste Log level :



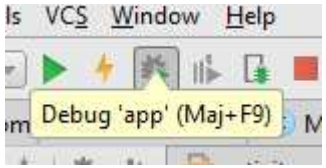
- Le niveau verbose affichera toutes les traces du filtre, le niveau info n'affichera que les traces info, warning, erreur et assert

Avec le débugger d'Android studio (1/2)

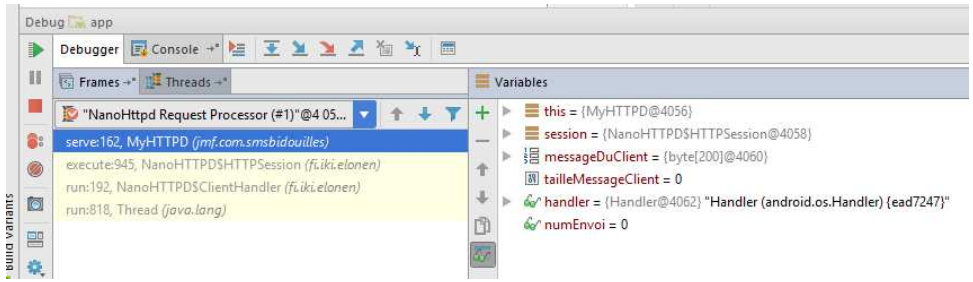
- On peut mettre des points d'arrêt et examiner les valeurs des variables, évaluer des expressions, continuer l'exécution ligne par ligne ou en entrant dans le code des méthodes appelées, etc.
- Mettre un point d'arrêt : cliquer sur la colonne de gauche (ou se placer sur la ligne et CTRL F8. Pour enlever de même



- Lancer l'exécution par en mode débug

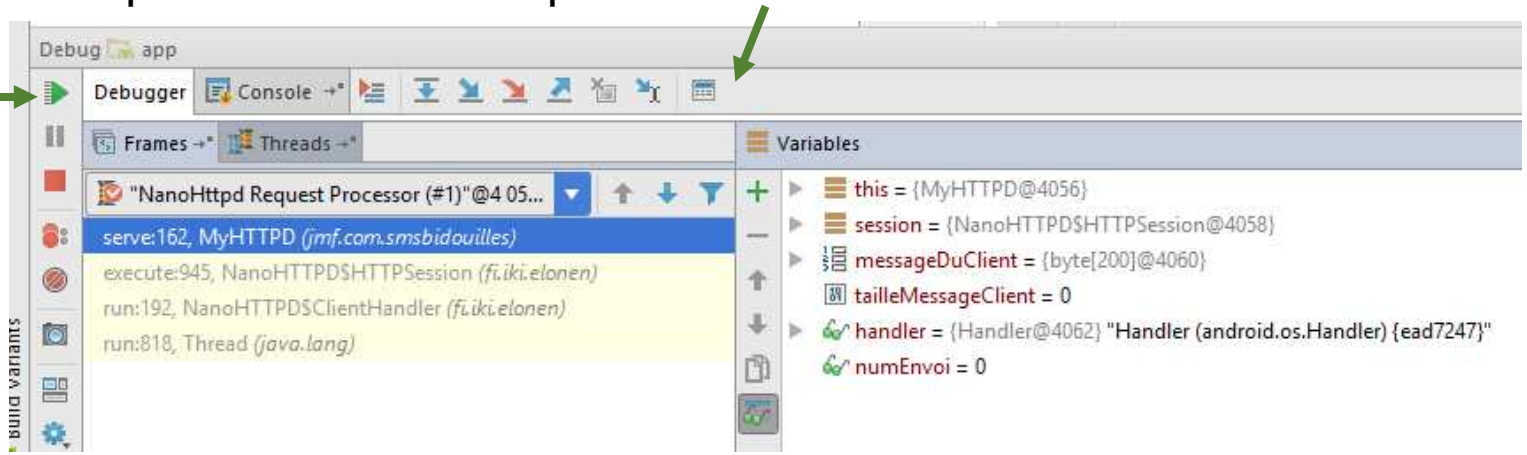


- A l'exécution, arrivé sur un point d'arrêt, l'exécution s'arrête et dans l'onglet débugger, on a les outils nécessaires :



Avec le débugger d'Android studio (2/2)

- On peut évaluer une expression avec



- Avancer à la ligne de code suivante par
- Avancer dans le code de la méthode appelée par
- Avancer sur la ligne qui suit l'appel de la méthode courante par
- Continuer le programme au prochain point d'arrêt par

- Bibliographie :
  `https://developer.android.com/studio/debug/index.html`

Exercice

- Gérer les actions des utilisateurs sur une IHM

Les Intents implicites et explicites

- On peut construire un `Intent` en indiquant explicitement la classe de l'objet qui gérera l'`Intent`. Par exemple, on écrit :

```
new Intent(getApplicationContext(), ActivityClassName.class);
```

- C'est un `Intent` explicite
- Mais on peut aussi indiquer au système l'intention à faire une action. L'environnement d'exécution (= le système), recherche alors les activités qui peuvent satisfaire cette action, les propose dans une liste si il y en a plusieurs, ou lance l'activité s'il est unique
- On utilise pour cela les `Intents` implicites
- "An implicit Intent does not specify the name of a particular target component; rather it declares a general action to perform, which allows a component from another app to handle it."

Les Intents implicites

- Le programmeur peut se construire des Intents implicites
- Si aucune activité ne peut gérer l'Intent, l'application émettrice est arrétée. Pour éviter cela, il est bon d'écrire :

```
Intent intent = new Intent(...)....
if (intent.resolveActivity(getPackageManager()) != null) {
     startActivity(intent);
}
```

Les Intents implicites courants

- Un smartphone Android possède des applications pré-installées
- Ces applications sont sensibles à des actions standards, pouvant être lancées par des Intents implicites. Par exemple :
  - Envoi d'un SMS :

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("sms:"));
intent.putExtra("sms_body", "hello world");
startActivity(intent);
```

  - Lancer une écoute de musique :

```
Intent intent = new Intent(Intent.ACTION_VIEW);
File hello = new File("/sdcard/hello.mp3");
intent.setDataAndType(Uri.fromFile(hello), "audio/mpeg");
startActivity(intent);
```

  - Appeler un numéro de téléphone :

```
String url = "tel:43556";
Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse(url));
startActivity(intent);
```

  - Lancer un navigateur :

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.google.com"));
startActivity(intent);
```

Action et catégorie d'un `Intent`

- Les `Intent` sont classés par catégorie (ensemble d'`Intent`) et d'action (un élément dans cette ensemble)
- Catégorie et action d'un `Intent` apparaissent comme sous élément de l'élément `intent-filter` d'un composant Android dans le fichier `AndroidManifest.xml`
- Exemple de catégories :

CATEGORY_TAB	Activité prévue pour fonctionner en tant qu'onglet d'une autre activité
CATEGORY_HOME	Affichage de l'écran d'accueil au démarrage du mobile ou après pression du bouton *Home*
CATEGORY_LAUNCHER	Activité initiale d'une tâche listée dans le gestionnaire d'applications
CATEGORY_DEFAULT	Obligatoire pour recevoir des intents implicites. Ainsi, elles pourront être trouvées par startActivity
CATEGORY_BROWSABLE	Activité qui peut être invoquée par un navigateur pour afficher les données référencées par un lien

Les Intents implicites (1/2)

- On peut se construire des Intent implicites. Par exemple :

```
<activity android:name=".HelloWorldActivity"
     android:label="@string/hello_world_name">
     <intent-filter>
          <action android:name="bonjour.ACTION" />
          <category android:name="android.intent.category.DEFAULT" />
     </intent-filter>
</activity>
```

- Cette activité HelloWorldActivity est sensible à l'action "bonjour.ACTION". Elle peut être lancée par :

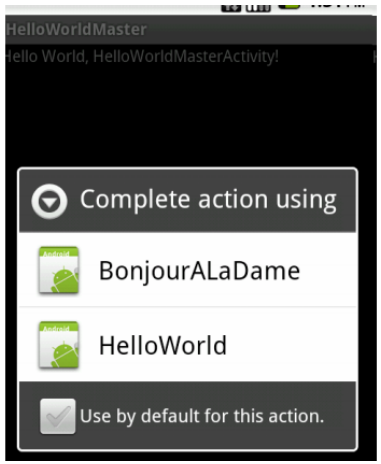
```
Intent intent = new Intent();
intent.setAction("bonjour.ACTION");
startActivity(intent);
```

Les `Intents` implicites (2/2)

- Si un autre composant peut être invoqué par la même action, par exemple :

```
<activity android:name=".DisBonjourALaDameActiviy"
     android:label="@string/dis_bonjour_a_la_dame_name">
     <intent-filter>
          <action android:name="bonjour.ACTION" />
          <category android:name="android.intent.category.DEFAULT" />
     </intent-filter>
</activity>
```

lorsque l'activité `HelloWorldMasterActivity` demande à traiter cette action, le système présente :



- ...y compris (et surtout ?!) si cet autre composant (= autre activité ici) appartient à une autre application

La méthode `startActivityForResult()`

- Permet de lancer une activité et pouvoir récupérer un résultat de cette nouvelle activité
- Le lancement est de la forme :

```
Intent lIntent = new Intent(getApplicationContext(), AutreActivity.class);
startActivityForResult (lIntent, CODE_DE_LA_REQUETE);
```

- `CODE_DE_LA_REQUETE` est un `int` identifiant la requête de lancement de la nouvelle activité
- La nouvelle activité va retourner des valeurs dans des extras de l'`Intent` retourné par :

```
Intent intentARetouner = new Intent();
intentARetouner.putExtra("valeurARetourner", resultatARetourner);
setResult(Activity.RESULT_OK, intentARetouner);
finish();
```

- Si tout se passe bien il faut utiliser `RESULT_OK`, sinon mettre `RESULT_CANCELED`
- biblio : `http://developer.android.com/training/basics/intents/result.html`

La méthode `onActivityResult()`

- C'est cette méthode qui, en fait, permet récupérer un résultat de d'une autre activité
- Son code est de la forme :

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if ((resultCode == RESULT_OK) && (requestCode == CODE_DE_LA_REQUETE)) {
        // traitement du résultat retourné
        ... data.getXXXExtra("valeurARetourner")...
    }
}
```

- XXX et le type de la valeur retournée
- Remarque : C'est la technique pour passer des informations d'une activité appelée à une activité appelante

Exercice

- Traiter les `Intents` implicites

L'internationalisation

- = i18n
- L'application doit être "localisée" = suivre la culture de l'utilisateur
- On parle de localisation
- En pratique, il faut tenir compte des différences de langues pour les textes, l'audio et les différences de présentation des nombres, des monnaies, etc.
- Une bonne solution : mettre les caractéristiques de localisation dans des fichiers de ressources plutôt que dans le code Java
- bibliographie : `http://developer.android.com/guide/topics/resources/localization.html`

La technique

- Généralement, les applications ont des fichiers de ressources par défaut. On leur ajoute des fichiers spécifiques de localisation
- A l'exécution, l'environnement choisit les fichiers adaptés à la culture (au "Locale") de l'utilisateur
- Toutes ces ressources se trouvent sous le répertoire `res` (et ses sous répertoires)

Les fichiers de ressources par défaut

- Pour les chaînes de caractères : `res/values/strings.xml`
- Pour les images : dans `res/drawable/`
- Pour les écrans d'IHM : dans `res/layout/`
- Ces fichiers doivent toujours être présents et toujours contenir toutes les ressources nécessaires à l'application
- Eventuellement on peut avoir :
  - dans `res/anim/` des animations
  - dans `res/xml/` des fichiers xml
  - dans `res/raw/` toutes sortes d'autres fichiers

Les fichiers de ressources localisés

- Ils se trouvent sous `res/`*`Repertoire`*`-`*`qualificateur`*
- *`Repertoire`* est le nom du répertoire où se trouve les ressources par défaut. Ce peut être `values`, `layout`, `drawable`, `menu`, `color`, etc.
- *`qualificateur`* est un nom spécifique de localisation. Il peut être formé de plusieurs abrévations séparées par `-`
- Plus précisément, *`qualificateur`* est défini :
  - par 2 lettres suivant le standard ISO 639-1 des codes de langues,
  - suivies éventuellement de 2 lettres de code de région suivant le standard ISO 3166-1-alpha-2 region code, précédé de la lettre `r`

Exemple de qualificateur

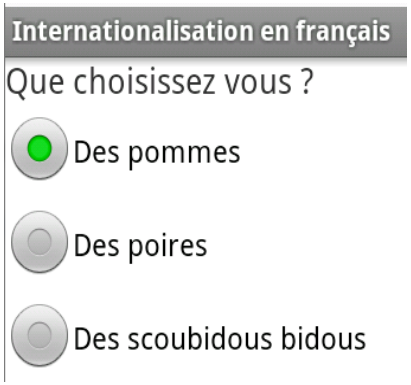
- exemple de *`qualificateur`* : `en`, `fr`, `en-rUS` (= anglais région united states = américain), `fr-rFR` (français de métropole), `fr-rCA` (français canadien), etc.
- Exemples : `res/values/strings.xml` (chaînes par défaut), `res/values-fr/strings.xml` (pour le français), `res/values-ja/strings.xml` (pour le japonais)
- bibliographie : `http://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources`

Localisation : un exemple (1/2)

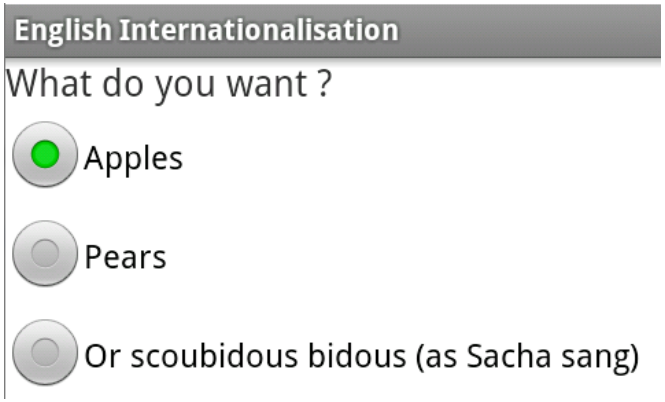
- Le fichier `res/layout/activity_main.xml`

```
<LinearLayout ... android:orientation="vertical" ...>
    <TextView ... android:text="@string/invite" />
    <RadioGroup ...>
        <RadioButton ...
            android:text="@string/choix1" />
        <RadioButton ...
            android:text="@string/choix2" />
        <RadioButton ...
            android:text="@string/choix3" />
    </RadioGroup>
</LinearLayout>
```

amène



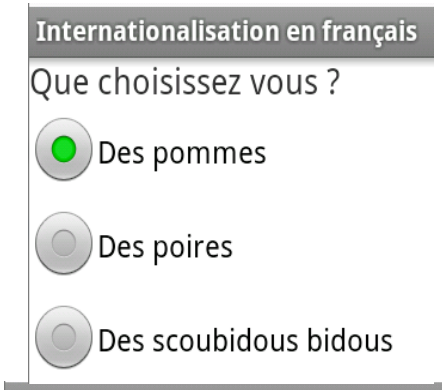
ou



car ...

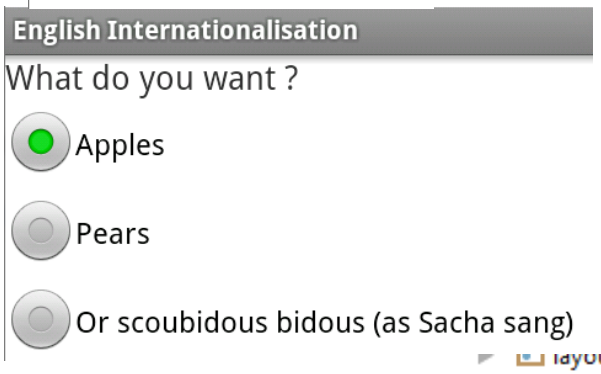
Localisation : un exemple (2/2)

utilise `res/values/strings.xml` (fichier de configuration par défaut) :



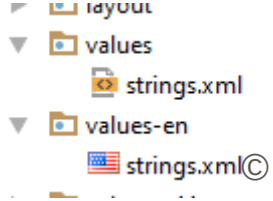

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Internationalisation en français</string>
    <string name="invite">Que choisissez vous ?</string>
    <string name="choix1">Des pommes</string>
    <string name="choix2">Des poires</string>
    <string name="choix3">Des scoubidous bidous</string>
</resources>
```

et utilise `res/values-en/strings.xml`




```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">English Internationalisation</string>
    <string name="invite">What do you want ?</string>
    <string name="choix1">Apples</string>
    <string name="choix2">Pears</string>
    <string name="choix3">Or scoubidous bidous (as Sacha sang)</string>
</resources>
```

- On a donc :



(utiliser l'onglet Project pas Android)

Configuration de la localisation

- Voir démo projet `Ch3InternationalisationProjet`
- Sur un AVD, choisir sur le bureau de l'AVD :
  Paramètres | Custom Locale

Trouver `fr` ou, éventuellement, l'ajouter avec le bouton "ajouter une nouvelle locale"

- Sur un Nexus S, Menu | Paramètre système | Langue et saisie, puis choisir la langue (cf. `http://www.youtube.com/watch?v=qE3B34I7tXo`)

Résumé du chapitre 5

- Il faut prévoir les actions de l'utilisateur sur une IHM : c'est la gestion des événements
- Dans l'utilisation d'une application Android, les écrans s'enchaînent les uns à la suite des autres. On utilise, pour cela les `Intents`
- les `Intents` peuvent être explicites ou implicites
- On peut utiliser les `Toast` et les traces (`Log`) pour l'aide au développement
- L'internationalisation est la particularité d'adapter les applications à la culture de l'utilisateur

Fin