PROGRAMMATION CONCURRENTE ET CONNECTIVITE

CRÉATION D'APPLICATIONS: CRÉATION D'INTERFACES UTILISATEUR AVANCÉES

DÉVELOPPEMENT D'APPLICATION MOBILE (ANDROID)

E.I. Djebbar

Département de Génie des systèmes

Ecole Nationale Polytechnique d'Oran

CRÉATION D'UN SERVICE

- Les services sont conçus pour s'exécuter en arrière plan et nécessitent de pouvoir être démarrés, contrôlés et stoppés par d'autres applications.
- Les services partagent un certain nombre de comportements avec les activités.
- Pour créer un service, on utilise la classe de android.app.Service puis, à l'instar d'une activité, il faut redéfinir les méthodes du cycle de vie.

IMPLÉMENTATION D'UN SERVICE

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
public class MonService extends Service {
@Override
public void onCreate() {
// Placez ici le code qui sera exécuté lors de la création de ce service
}
Override
public void onStart(Intent intent, int startId) {
// Placez ici le code qui sera exécuté à chaque démarrage du service
}
@Override
public void onDestroy() {
// Placez ici le code qui sera exécuté lors de la destruction de ce service
}
@Override
public IBinder onBind(Intent arg0) {
return null;
}
}
```

DÉCLARER UN SERVICE DANS LE FICHIER DE CONFIGURATION DE L'APPLICATION

```
<service android:name=".MonService">
```

Nom de l'attribut	Description
process	Spécifie un nom de processus dans lequel le code sera exécuté.
enabled	Indique si ce service est actif ou non (peut-être instancié par le système).
exported	Booléen indiquant si le service est utilisable par d'autres applications.
permission	Spécifie une permission nécessaire pour exécuter ce service.

DÉMARRER ET ARRÊTER UN SERVICE

```
// Démarre le service explicitement.
startService(new Intent(this, MonService.class));

// Démarre le service en utilisant une action (enregistré
// dans Intent Filter).
startService(new Intent(MonService.MON_ACTION_SPECIALE));
```

IMPLÉMENTER UN SERVICE POUVANT S'ASSOCIER À UNE ACTIVITÉ

```
import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import android.widget.Toast;
public class MonService extends Service {
private final IBinder mBinder = new MonServiceBinder();
@Override
public IBinder onBind(Intent arg0) {
return mBinder;
}
public void AfficheStatut() {
Toast.makeText(MonService.this, "Voici le statut de mon service !",
Toast.LENGTH_SHORT).show();
}
```

```
public class MonServiceBinder extends Binder {
MonService getService() {
return MonService.this;
}
}
}
```

IMPLÉMENTER UNE ACTIVITÉ AVEC UNE CONNEXION VERS UN SERVICE

```
import android.app.Activity;
public class GestionServiceActivite extends Activity {
private MonService mMonService;
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
Intent intentAssociation = new Intent(this, MonService.class);
if (bindService(intentAssociation, mConnexion,
Context.BIND_AUTO_CREATE)) {
Button btnService = (Button) findViewById(R.id.BoutonService);
btnService.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
mMonService.AfficheStatut();
}
});
}
}
```

```
@Override
protected void onDestroy() {
super.onDestroy();
unbindService(mConnexion);
}
private ServiceConnection mConnexion = new ServiceConnection() {
public void onServiceConnected(ComponentName className, IBinder
service) {
mMonService = ((MonService.MonServiceBinder) service).getService();
}
public void onServiceDisconnected(ComponentName className) {
mMonService = null;
}
};
}
```

NOTIFIER L'UTILISATEUR PAR LE MÉCANISME DES « TOASTS »

```
// La chaîne représentant le message
String message = "Vous prendrez bien un toast ou deux ?";
// La durée d'affichage (LENGTH_SHORT ou LENGTH_LONG)
int duree= Toast.LENGTH_LONG;
// Création du Toast (le contexte est celui de l'activité ou du service)
Toast toast = Toast.makeText(this, message, duree);
// Affichage du Toast
toast.show();
```

AFFICHAGE D'UN TOAST S'EXÉCUTANT DEPUIS UN SERVICE :

- L'utilisateur est notifié sans entraver son utilisation.

POSITIONNER UN TOAST

```
// Création du Toast
Toast toast = Toast.makeText(this, "Vous prendrez bien un toast ou deux ;",
Toast.LENGTH_LONG);
// Spécifie la dispostion du Toast sur l'écran
toast.setGravity(Gravity.TOP, 0, 40);
// Affichage du Toast
toast.show();
```

GESTION DES THREADS

- Faire le travail dans un thread en arrière-plan ;
- La plate-forme Android offre quelques raffinements pour pouvoir utiliser des threads d'arrière-plan et pouvoir échanger avec le thread de l'interface utilisateur.
- Ces mécanismes se trouvent implémentés au sein des classes Runnable et Handler.

EXÉCUTION ET COMMUNICATION ENTRE THREADS AVEC LA CLASSE HANDLER

- La façon la plus simple de gérer une tâche en arrière-plan est de créer un thread en implémentant une classe dérivant de Handler.
- Utiliser une même instance de cette classe dans l'activité en cours.
- La classe Handler après instanciation, s'enregistre d'elle-même dans la gestion des threads du système Android.

UTILISER LES MESSAGES

- La communication entre les threads d'un Handler utilise un système de messages. Ce système permet d'ajouter des messages dans une file et de les traiter dans leur ordre d'arrivée.
- Pour envoyer un message à l'instance de Handler, il faut d'abord récupérer une instance de Message en appelant la méthode obtainMessage.
- Une fois le message créé et configuré, placer ce message dans la file de messages avec les méthodes sendMessage, sendMessageAtFrontOfQueue, sendMessageDelayed et sendMessageAtTime.

INTERFACE DE L'ACTIVITÉ CONTENANT UNE BARRE DE PROGRESSION

- Un exemple typique de l'utilisation de la classe Handler pour gérer les threads est l'intégration d'une barre de progression dans l'interface.

INTERFACE DE L'ACTIVITÉ CONTENANT UNE BARRE DE PROGRESSION

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical" android:layout_width="fill_parent"
android:layout_height="fill_parent">
<TextView android:text="Barre de progression avec Handler :"
android:id="@+id/text_barre"
android:layout_width="wrap_content" android:layout_height="wrap_content"
android:gravity="center_horizontal" />
<ProgressBar android:id="@+id/barre_progression"
style="?android:attr/progressBarStyleHorizontal"
android:layout_width="fill_parent"
android:layout_height="wrap_content" />
</LinearLayout>
```

ACTIVITÉ UTILISANT UN HANDLER POUR GÉRER UNE BARRE DE PROGRESSION

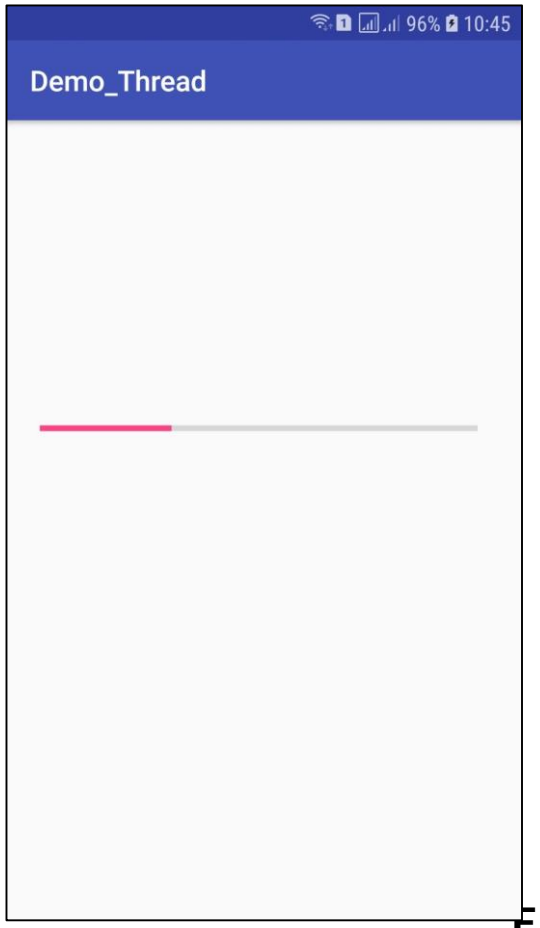
```
import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.widget.ProgressBar;
public class HandlerActivite extends Activity {
// La barre de progression à mettre à jour
ProgressBar barreProgression;
// Comme pour les threads classiques, nous gardons la trace de son activité.
boolean enExecution = false;
```

```
// Notre sous-classe Handler qui gère le traitement des messages.
final Handler handler = new Handler() {
@Override
public void handleMessage(Message msg) {
barreProgression.incrementProgressBy(10);
}
};
@Override
public void onCreate(Bundle icicle) {
super.onCreate(icicle);
setContentView(R.layout.main);
barreProgression = (ProgressBar)
findViewById(R.id.barre_progression);
}
```

ACTIVITÉ UTILISANT UN HANDLER POUR GÉRER UNE BARRE DE PROGRESSION

```
public void onStart() {
super.onStart();
barreProgression.setProgress(0);
Thread threadArrierePlan = new Thread(new Runnable() {
public void run() {
try {
for (int i = 0; enExecution && i < 10; ++i) {
// Mise en pause d'une seconde
Thread.sleep(1000);
// Création d'un message vide
Message msg = handler.obtainMessage();
// Envoi du message au handler
handler.sendMessage(msg);
}
} catch (Throwable t) { }
}
});
enExecution = true;
// Lancement du thread d'arrière-plan
threadArrierePlan.start();
}
public void onStop() {
super.onStop();
enExecution = false;
}
}
```

ACTIVITÉ UTILISANT UN HANDLER POUR GÉRER UNE BARRE DE PROGRESSION



INFORMATIONS SUR L'ÉTAT DE L'APPAREIL

- La classe centrale pour la récupération d'informations concernant la téléphonie est TelephonyManager qui se trouve dans le paquetage android.telephony.
- Cette classe va permettre à la fois de récupérer des informations sur l'appareil et la carte SIM, mais également d'obtenir, voire de se tenir informer du statut et des changements d'états de la fonction téléphone (en conversation, appel en pause, etc.).

PERMISSION DE LECTURE DES INFORMATIONS DE TÉLÉPHONIE

```
<manifest>
...
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
...
</manifest>
```

INFORMATIONS SUR L'APPAREIL ET LE RÉSEAU

```
TelephonyManager telephonyManager =
(TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
// Infos réseau.
String nomOperateurReseau = telephonyManager.getNetworkOperatorName();
Log.d("Nom de l'operateur réseau ", nomOperateurReseau);
// Récupération de l'état de la fonction téléphone
int etat = telephonyManager.getCallState();
switch(etat) {
// Pas d'activité
case TelephonyManager.CALL_STATE_IDLE:
Log.d("Etat", "Inactif");
break;
// Appel venant d'arriver
case TelephonyManager.CALL_STATE_RINGING:
Log.d("Etat", "Sonne");
break;
```

INFORMATIONS SUR L'APPAREIL ET LE RÉSEAU

```
case TelephonyManager.CALL_STATE_OFFHOOK:
Log.d("Etat", "Appel en cours");
break;
default:
Log.d("Etat", "Autre : " + etat);
break;
}
String idAppareil = telephonyManager.getDeviceId();
Log.d("Identifiant de l'appareil", idAppareil);
String versionSoftwareAppareil =
telephonyManager.getDeviceSoftwareVersion();
if (versionSoftwareAppareil != null)
Log.d("Version logicielle", versionSoftwareAppareil);
String numeroSerieSIM = telephonyManager.getSimSerialNumber();
Log.d("SIM", numeroSerieSIM);
```

PASSER DES APPELS

Pour passer des appels, on peut envisager deux approches différentes :

- préremplir le numéro de téléphone dans le composeur de numéro ;
- préremplir le numéro et déclencher l'appel.

DÉMARRAGE DE L'APPLICATION DE COMPOSITION DE NUMÉRO DE TÉLÉPHONE

```
String telURI = "tel:" + "0612345678";
Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse(telURI));
startActivity(intent);
```

DÉCLENCHER L'APPEL

- **Démarrage de l'application d'appels téléphoniques**

```
String telURI = "tel:" + "0612345678";
Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse(telURI));
startActivity(intent);
```

- **Ajout de la permission nécessaire pour appeler un correspondant**

```
<manifest>
...
<uses-permission android:name="android.permission.CALL_PHONE" />
...
</manifest>
```

GÉRER LES SMS

- Plus les années passent, plus le nombre de SMS échangés augmente.
- Ce service de messages courts (160 caractères), très prisé du jeune public, est un incontournable si l'application doit envoyer ou gérer la réception de certains SMS.

AJOUT DE LA PERMISSION POUR ENVOYER DES SMS

```
<manifest>
...
<uses-permission android:name="android.permission.SEND_SMS"/>
...
</manifest>
```

APPLICATION D'ENVOI DE SMS

```
public class EnvoiSMS extends Activity {
protected void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
Button sendBtn = (Button)
findViewById(R.id.envoiBouton);
sendBtn.setOnClickListener(new OnClickListener() {
public void onClick(View view) {
envoiSMS();
}
});
}
```

```
private void envoiSMS() {
// Récupération des champs texte pour le numéro et le message.
EditText numeroEditText =
(EditText)findViewById(R.id.numeroEditText);
EditText messageEditText =
(EditText) findViewById(R.id.messageEditText);
String numero = numeroEditText.getText().toString();
String message = messageEditText.getText().toString();
SmsManager smsManager = SmsManager.getDefault();
try {
smsManager.sendTextMessage(numero, null, message, null, null);
Toast toast = Toast.makeText(EnvoiSMS.this, "SMS envoyé",
Toast.LENGTH_LONG);
toast.show();
} catch (Exception e) {
Toast toast = Toast.makeText(EnvoiSMS.this, "Erreur !",
Toast.LENGTH_LONG);
toast.show();
}
}
```