Développement d'Applications Mobiles sous Android

Abdelhak-Djamel Seriai

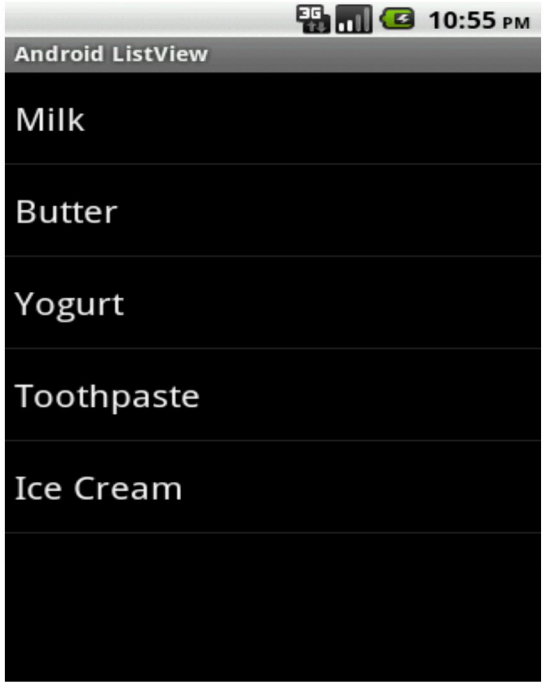
Gabarits avec un adaptateur

Gabarits avec un adaptateur

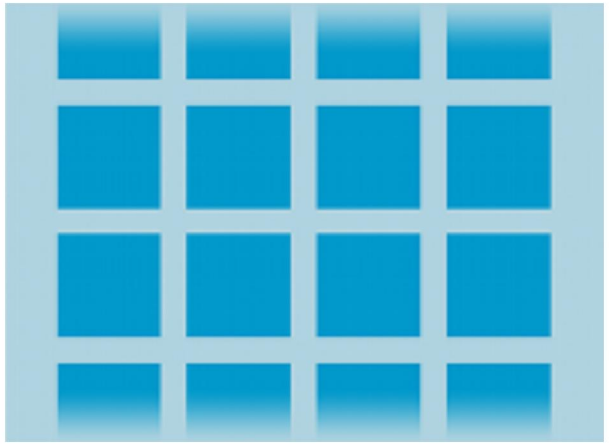
- Les éléments (vues) d'un gabarit (layout) peuvent être
  - statiques : Les éléments (les vues) qui composent le gabarit n'évoluent pas ni par rapport à leur identité ni par rapport à leur nombre
    - Exemple du **LinearLayout**, **RelativeLayout**
  - Dynamiques : Les éléments (les vues) qui composent le gabarit peuvent évoluer (changent) par rapport à leur identité comme par rapport à leur nombre
    - Exemple du **ListView** et du **GridView**

Layouts Dynamiques

List View



Grid View



Layouts Dynamiques

- Les identités de leurs éléments ainsi que leur nombre peuvent changer dynamiquement
- L'adaptation dynamique de leur contenu (quels éléments à afficher) est difficile à gérer par le programmeur
  - Gestion (Suppression/remplacement) des éléments (vues) en cas de défilement (scrolling)
  - Adaptation du nombre d'éléments par rapport à la taille de l'écran

Objet/class Adapter

- Les gabarits dynamiques dont les élément sont récupérés dynamiquement sont crées comme une sous classe de **AdapterView**

- Une sous classe **AdapterView** utilise un objet de la classe **Adapter** pour lier le gabarit correspondant à ses données (ses éléments/ses vues).

- Un objet **Adapter** agit comme un intermédiaire entre une source de données et un gabarit **AdapterView**

- L'objet **Adapter** récupère les données d'une source comme un tableau ou une requête d'une base de données et convertit chaque entrée en une vue qui peut être ajoutée dans un gabarit **AdapterView**

Remplissage d'un Adapter View

- Les deux **Adapter** les plus courants sont :
  - **ArrayAdapter** : utilisé quand la source de données est un tableau
  - **SimpleCursorAdapter :** Utilisé quand la source de donnée est un curseur (Cursor)

Exemple ArrayAdapter

- Affichage des éléments d'un tableau comme une ListView

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(
        this,
        android.R.layout.simple_list_item_1,
        myStringArray);
```

```
ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(adapter);
```

Personnaliser un Adapter

- Par défaut, **ArrayAdapter** crée une vue pour chaque élément(item)
- La méthode **toString()** est appelée sur chaque élément du tableau pour retourner le contenu d'une vue **TextView**
- Pour personnaliser l'apparence de chaque items (éléments) de la liste il faut surcharger la méthode **toString()** pour les objets du tableau
- Pour personnaliser les vues à intégrer au gabarit, par exemple afficher une image dans un **ImageView** au lieu de **TextView** : Spécialiser **ArrayAdapter** et surcharger la méthode **getView()** pour retourner le type de vue désirée.

Persistance des données

Persistance de l'état des applications/activités

Persistance de l'état des applications/activités

- Besoin de conserver l'état de l'interface utilisateur
  - Lorsqu'un utilisateur navigue dans une application
  - Pour préparer le retour sur les écrans précédents

- Android gère le cycle de vie d'une application
  - Une activité d'arrière plan peut être déchargée de la mémoire en fonction de la politique de gestion des ressources du système

  - Besoin de restaurer l'état d'une activité entre deux sessions → sauvegarde de l'état d'une activité
    - Utilisation des méthodes de cycle de vie de l'activité

Gestion de la persistance des activités

- La méthode **onSaveInstanceState()** d'une activité est appelée lorsque le système a besoin de libérer des ressources et de détruire l'activité
  - Utilisation d'un objet de **Bundle** pour stocker les données
    - Passé en paramètre aux méthodes
      - **OnCreate()** : rétablir l 'interface lors de la création
      - **OnRestoreInstanceState()** : rétablir l'interface lors de la restauration
  - Par défaut :
    - Les valeurs de toutes les vues possédant un attribut **id** renseigné sont enregistrées, puis restaurées
    - **OnSaveInstanceState()** enregistre l'état des vues identifiées dans un objet Bundle.
      - L'objet **Bundle** est ensuite passé aux **onCreate()** et **onRestoreInstanceState()** pour restaurer l'état de l'activité

Gestion de la persistance des activités

```
import android.R;
import android.app.Activity;
import android.os.Bundle;
import android.widget.Toast;

public class SauvegardeEtatActivite extends Activity {

	@Override
	public void onCreate(Bundle savedInstanceState) {
		...}

	@Override
	protected void onRestoreInstanceState(Bundle savedInstanceState) {
		...}

	@Override
	protected void onSaveInstanceState(Bundle savedInstanceState) {
		...}

	@Override
	protected void onDestroy() {
		...}
}
```

Gestion de la persistance des activités

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);      }
```

```
@Override
    protected void onSaveInstanceState(Bundle savedInstanceState) {
        super.onSaveInstanceState(savedInstanceState);
        Toast.makeText(this, "Etat de l'activité sauvegardé",
                Toast.LENGTH_SHORT).show();    }
```

```
@Override
    protected void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);
        Toast.makeText(this, "Etat de l'activité restauré",
                        Toast.LENGTH_SHORT).show();    }
```

```
@Override
    protected void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "L'activité est détruite", Toast.LENGTH_SHORT)
                .show();        }
```

Gestion de la persistance des activités

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/layotuPrincipal"
	android:layout_width="fill_parent" android:layout_height="fill_parent"
	android:orientation="vertical"
	xmlns:android="http://schemas.android.com/apk/res/android">

	<TextView android:id="@+id/nomDescription" android:layout_width="wrap_content"
		android:layout_height="wrap_content" android:text="Saisissez votre nom :">
	</TextView>

	<EditText android:id="@+id/nom" android:layout_width="fill_parent"
		android:layout_height="wrap_content" android:textSize="18sp">
	</EditText>

	<TextView android:id="@+id/messageDescription"
		android:layout_width="fill_parent" android:layout_height="wrap_content"
		android:text="Saisissez un message (qui ne sera pas enregistré) :">
	</TextView>

	<EditText android:layout_width="fill_parent"
		android:layout_height="wrap_content" android:textSize="18sp">
	</EditText>
</LinearLayout>
```

Configurer le mode de conservation des activités

- Mode de sauvegarde par défaut
  - Avantage : pas de code spécifique
  - Inconvénient : pas adapté à certaines situations
    - Ne pas sauvegarder les valeurs de certains champs qui possèdent un identifiant

- Personnalisation de l'enregistrement de l'état d'une activité
  - Redéfinitions des méthodes **onSaveInstanceState()**, **onCreate()** et **onRestoreInstanceState()**
  - Utilisation de l'objet **Bundle** pour lire et écrire des valeurs

Configurer le mode de conservation des activités

```
import projet.seriai.android.R;
import android.app.Activity;
import android.os.Bundle;
import android.widget.Toast;

public class SauvegardeEtatActivite extends Activity {

	private final static String MA_CLE = "MaCle";

	@Override
	public void onCreate(Bundle savedInstanceState) {
		...}
	@Override
	protected void onRestoreInstanceState(Bundle savedInstanceState) {
		...}
	@Override
	protected void onSaveInstanceState(Bundle savedInstanceState) {
		...}
	@Override
	protected void onDestroy() {
		...}
}
```

Configurer le mode de conservation des activités

```
@Override
    protected void onSaveInstanceState(Bundle savedInstanceState) {
        super.onSaveInstanceState(savedInstanceState);
        savedInstanceState.putString(MA_CLE, "Ma valeur !");
        Toast.makeText(this, "onSaveInstanceState",
                Toast.LENGTH_SHORT).show();
    }
```

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if ((savedInstanceState != null) &&(savedInstanceState.containsKey(MA_CLE))) {
            String val = savedInstanceState.getString(MA_CLE);
            Toast.makeText(this, "onCreate() : " + val,
                    Toast.LENGTH_SHORT).show();
        }
        setContentView(R.layout.persistance_etat_activite1);
    }
```

Stockage dans des fichiers

Stockage dans des fichiers

- Lire, écrire dans le système de fichiers
  - **OpenFileOutput()** : ouvrir un fichier en écriture ou de le créer s'il n'existe pas
    - Retourne un **FileOutputStream**
    - Par défaut : le fichier est écrasé s'il existe
    - Ecrire dans le fichier : **write()**
    - Fermer le fichier : **close()**
  - **DeleteFile()** : supprimer un fichier à partir de son nom
- Gestion de fichiers
  - **FileList()** : retourne tous les fichiers locaux de l'application
  - **GetFileDir()** : retourne le chemin absolu du répertoire où tous les fichiers créés par openFileOutput
  - **GetFileStreamStore** : retourne le chemin absolu du répertoire du fichier passé en paramètre

Stockage dans des fichiers

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

Partager un fichier avec d'autres applications

- Par défaut :
  - Les fichiers créés par la méthode **openFileOutput** sont restreints à l'application

- Spécification d'un mode d'ouverture
  - **MODE_PRIVATE** : mode par défaut, fichier accessible uniquement par l'application
  - **MODE_WORLD_READABLE** : lecture pour les autres application mais pas la modification
  - **MODE_WORLD_WRITABLE** : lecture/écritures aux autres applications
  - **MODE_APPEND** : ajouter des données en fin de fichier. Peut être combiné avec un autre mode

Base de données SQLite

SQLite

- SQLite : base de données relationnelle
  - Légère, gratuite et open Source
  - Www.sqlite.org
  - S'exécute sans nécessiter de serveur → exécution des requêtes dans le même processus de l'application
  - Chaque BD SQLite est réservée à son application créatrice
    - Utilisation d'un fournisseur de contenu pour partager une BD
  - Possibilité de créer plusieurs BDs par application

Création et mise à jour de BD SQLite

- Utilisation de la classe SQLiteOpenHelper

```
private class MaBaseOpenHelper extends SQLiteOpenHelper {

    public MaBaseOpenHelper(Context context, String nom, CursorFactory
                                        cursorfactory, int version) {
        super(context, nom, cursorfactory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
            //code de création}

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            // code de mise à jour
        }
    }
```

Création et mise à jour de BD SQLite

```
class MaBaseOpenHelper extends SQLiteOpenHelper {

        public MaBaseOpenHelper(Context context, String nom, CursorFactory
                                                    cursorfactory, int version) {
            super(context, nom, cursorfactory, version);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {
            db.execSQL(REQUETE_CREATION_BD);
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            //Dans notre cas, nous supprimons la base et les données pour en
            // créer une nouvelle ensuite.

            db.execSQL("drop table" + TABLE_PLANETES + ";");
            // Création de la nouvelle structure.

            onCreate(db);
        }
    }
```

Création et mise à jour de BD SQLite

```
private static final int BASE_VERSION = 1;
	private static final String BASE_NOM = "planetes.db";

	private static final String TABLE_PLANETES = "table_planetes";

	public static final String COLONNE_ID = "id";
	public static final int COLONNE_ID_ID = 0;
	public static final String COLONNE_NOM = "nom";
	public static final int COLONNE_NOM_ID = 1;
	public static final String COLONNE_RAYON = "rayon";
	public static final int COLONNE_RAYON_ID = 2;

	/**
	 * La requête de création de la structure de la base de données.
	 */
	private static final String REQUETE_CREATION_BD = "create table "
			+ TABLE_PLANETES + " (" + COLONNE_ID
			+ " integer primary key autoincrement, " + COLONNE_NOM
			+ " text not null, " + COLONNE_RAYON + " text not null);";

	/**
	 * L'instance de la base qui sera manipulée au travers de cette classe
	 */
	private SQLiteDatabase maBaseDonnees;
```

Création et mise à jour de BD SQLite

- **Accéder à une base de données**
  - Utilisation des méthodes **getReadableDatabase()** et **getWritableDatabase**

```
private MaBaseOpenHelper baseHelper;

public PlanetesDBAdaptateur(Context ctx) {
        baseHelper = new MaBaseOpenHelper(ctx, BASE_NOM, null, BASE_VERSION);
    }
```

```
public SQLiteDatabase open() {
        maBaseDonnees = baseHelper.getWritableDatabase();
        return maBaseDonnees;
    }
```

Accéder à une base de données

- Besoin de rendre les traitements indépendants par rapport au type de la source des données
  - Utilisation des adaptateurs : encapsuler toutes les actions sur la BD dans une classe dédiée
  - Voir code associé

Effectuer une requête

- Utilisation de la méthode query()
  - Retourne un curseur permettant de naviguer dans les résultats
    - Objet Cursor

```
public Planete getPlanete(String nom) {
        Cursor c = maBaseDonnees.query(TABLE_PLANETES,
                new String[] {
                COLONNE_ID, COLONNE_NOM, COLONNE_RAYON },
                null, null, null,
                COLONNE_NOM + " LIKE " + nom, null);

        return cursorToPlanete(c);
    }
```

```
public Cursor getAllPlanetesCurseur() {
        return maBaseDonnees.query(TABLE_PLANETES, new String[] {
                COLONNE_ID,
                COLONNE_NOM, COLONNE_RAYON },
                null, null, null, null, null);
    }
```

Effectuer une requête

- **Opération sur objet Cursor**

abstract boolean moveToFirst()
Move the cursor to the first row.

abstract boolean moveToLast()
Move the cursor to the last row.

abstract boolean moveToNext()
Move the cursor to the next row.

abstract boolean moveToPosition(int position)
Move the cursor to an absolute position.

abstract boolean moveToPrevious()
Move the cursor to the previous row
é

Utilisation d'un CursorAdapter

- Exemple : création une liste de personnes avec leurs noms et leurs numéros de téléphones
  - Exécuter une requête qui retourne un objet **Cursor** qui contient une ligne pour chaque personne et deux colonnes pour le nom et le numéro de téléphone respectivement.
  - Créer un **« string array »** qui spécifie quelle colonne dans les lignes du Curseur à insérer dans le gabarit et un **« integer array »** qui spécifie la vue qui correspond à chaque colonne.

Utilisation d'un CursorAdapter

```
String[] fromColumns = {ContactsContract.Data.DISPLAY_NAME,
                        ContactsContract.CommonDataKinds.Phone.NUMBER};

int[] toViews = {R.id.display_name, R.id.phone_number};
```

```
SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
      R.layout.person_name_and_number, cursor, fromColumns, toViews, 0);
ListView listView = getListView();
listView.setAdapter(adapter);
```

Les services

Présentation

- Un service est un composant qui peut effectuer des opérations en arrière plan et ne fournit pas d'interface utilisateur
  - Par exemple: gérer les transactions réseau, lire de la musique

- Déclaration

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

Modes d'activation d'un service

- Deux modes pour lancer un service :
  - Démarré par un autre composant (mode Started)
    - Quand un composant d'application (par exemple, une activité) démarre ce service en appelant **startService()**
  - Lié à d'autres composants (mode Bound)
    - Quand un composant d'application se lie à ce service en appelant **bindService()**

Modes d'activation d'un service

- Démarré (Started)
  - Une fois démarré, un service peut s'exécuter en arrière-plan indéfiniment, même si le composant qui a commencé est détruit
  - Un service démarré effectue une opération unique et ne retourne pas de résultat à l'appelant
    - Par exemple,
      - télécharger un fichier sur le réseau
        - Lorsque l'opération est terminée, le service doit s'arrêter

Lancer un service

- Démarrer un service en passant un Intent à startService ()
  - Android appelle la méthode **onStartCommand ()** du service et lui transmet l'intention

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

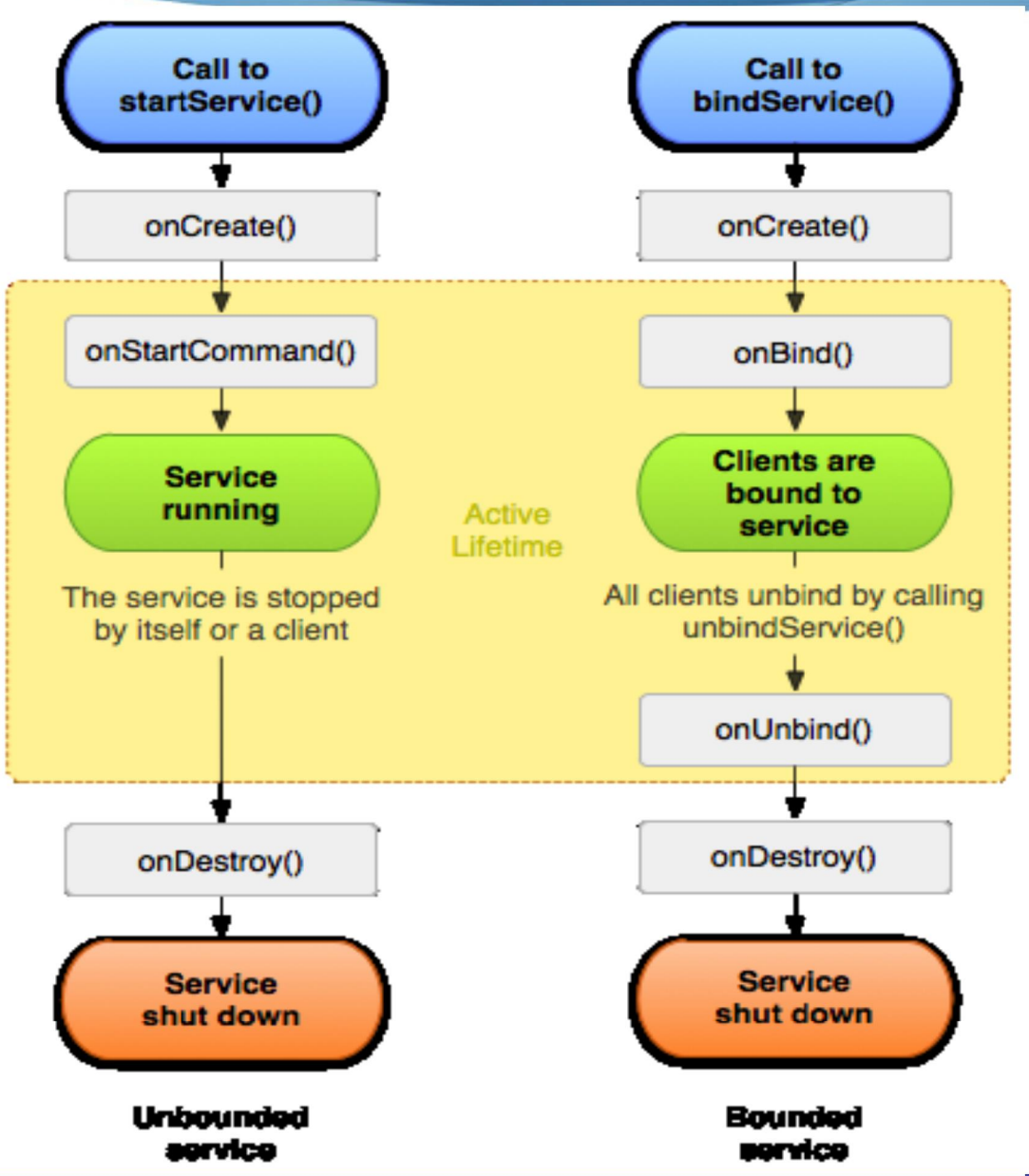
    - Par exemple, une activité doit enregistrer des données dans une base de données en ligne. L'activité peut démarrer un service et lui transmettre les données à sauvegarder en lui passant une intention via **startService ()**

  - Un service peut définir des filtres qui permettent de capter les appels avec intentions (intents) implicites

Modes d'activation d'un service

- Lié (Bound)
  - Un service lié propose une interface client-serveur qui permet aux composants d'interagir avec le service
    - Envoyer des demandes
    - Obtenir des résultats
    - Faire à travers des processus de communication inter-processus (IPC).
  - Un service lié ne fonctionne que tant qu'un ou plusieurs composants d'autres applications sont liés à celui-ci
    - Quand tous les liaisons sont détruites, le service est détruit

Cycle de vie d'un service



Unbounded service

Bounded service

Création de service

- Deux classes peuvent être étendues pour créer un service démarré:
  - Service
    - Il s'agit de la classe de base pour tous les services
  - IntentService
    - Il s'agit d'une sous-classe de Service qui utilise un même thread pour traiter toutes les demandes, une à la fois

Création de service

- **onStartCommand ()**
  - Le système appelle cette méthode lorsqu'un autre composant tel une activité, active un service en appelant **startService()**
  - Une fois que cette méthode lancée, le service est démarré et peut fonctionner en tâche de fond indéfiniment
    - C'est au développeur d'arrêter le service lorsque son travail est fait, en appelant **stopSelf ()** ou **StopService ()**

Création de service

- onBind ()
  - Le système appelle cette méthode quand un autre composant veut se lier avec le service
    - Par exemple, pour effectuer RPC en appelant **bindService()**
  - Utilisation d'une interface par les clients pour communiquer avec le service, en renvoyant un IBinder.
  - Si un service est appelé par **bindService ()**, une fois dissocié de tous ses clients, le système le détruit

Création de service

- **Étendre la classe IntentService**
  - Permet de traiter les demandes du service dans l'ordre de leur arrivée
  - Le IntentService effectue les opérations suivantes:
    - Crée un thread par défaut qui exécute toutes les intentions passées à **onStartCommand()**
    - Crée une file d'attente des demandes vers le service
    - Arrête le service après que toutes les demandes ont été traitées
    - Fournit une implémentation par défaut de **onBind ()** qui renvoie la valeur null.
    - Fournit une implémentation par défaut de onStartCommand ()
  - Le développeur doit implémenter **onHandleIntent()**

```
public class HelloIntentService extends IntentService {

  /**
   * A constructor is required, and must call the super IntentService(String)
   * constructor with a name for the worker thread.
   */
  public HelloIntentService() {
      super("HelloIntentService");
  }
 /**
   * The IntentService calls this method from the default worker thread with the intent that *started
the service. When this method returns, IntentService stops the service, as appropriate.
   */
  @Override
  protected void onHandleIntent(Intent intent) {
      // Normally we would do some work here, like download a file.
      // For our sample, we just sleep for 5 seconds.

       long endTime = System.currentTimeMillis() + 5*1000;
      while (System.currentTimeMillis() < endTime) {
          synchronized (this) {
            try {
                wait(endTime - System.currentTimeMillis());
            } catch (Exception e) {
            }
          }
      }}}
```

Création de service

- **Étendre la classe de service**
  - Permet au service d'effectuer un multi-threading des tâches
  - Possibilité de traiter plusieurs demandes en même temps
    - Créer un nouveau thread pour chaque requête et l'exécuter immédiatement, au lieu d'attendre la la fin de la requête précédente
      - Chaque appel à **onStartCommand()** peut être traité à part

```
public class HelloService extends Service {
  private Looper mServiceLooper;
  private ServiceHandler mServiceHandler;

  // Handler that receives messages from the thread
  private final class ServiceHandler extends Handler {
      public ServiceHandler(Looper looper) {
          super(looper);
      }
   @Override
    public void handleMessage(Message msg) {//.… }
  }

  @Override
  public void onCreate() {//...}

   @Override
  public int onStartCommand(Intent intent, int flags, int startId)
   {//...}

   @Override
  public IBinder onBind(Intent intent) {//...}

   @Override
  public void onDestroy() {//...}
}
```

Création d'un service

```
@Override
      public void handleMessage(Message msg) {
     // Normally we would do some work here, like download a file.
    // For our sample, we just sleep for 5 seconds.

       long endTime = System.currentTimeMillis() + 5*1000;
          while (System.currentTimeMillis() < endTime) {
              synchronized (this) {
                  try {
                      wait(endTime - System.currentTimeMillis());
                  } catch (Exception e) {
                  }
              }
          }

   // Stop the service using the startId, so that we don't stop
          // the service in the middle of handling another job

       stopSelf(msg.arg1);
      }
```

Création d'un service

```
@Override
  public void onCreate() {
 // Start up the thread running the service.  Note that we create a
// separate thread because the service normally runs in the
//process's main thread, which we don't want to block.
//We also make it background priority so CPU-intensive
//work will not disrupt our UI.

    HandlerThread thread = new
                HandlerThread("ServiceStartArguments",
                              Process.THREAD_PRIORITY_BACKGROUND);
     thread.start();

     // Get the HandlerThread's Looper and use it for our Handler


    mServiceLooper = thread.getLooper();
    mServiceHandler = new ServiceHandler(mServiceLooper);
   }
```

Création d'un service

```
@Override

public int onStartCommand(Intent intent, int flags, int startId) {
Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show();

// For each start request, send a message to start a job and
//deliver the start ID so we know which request we're stopping
//when we finish the job

       Message msg = mServiceHandler.obtainMessage();
      msg.arg1 = startId;
      mServiceHandler.sendMessage(msg);

      // If we get killed, after returning from here, restart
      return START_STICKY;
  }
```

Création d'un service

```
@Override
  public IBinder onBind(Intent intent) {
      // We don't provide binding, so return null
      return null;
  }

  @Override
  public void onDestroy() {
    Toast.makeText(this, "service done",
    Toast.LENGTH_SHORT).show();
  }
```