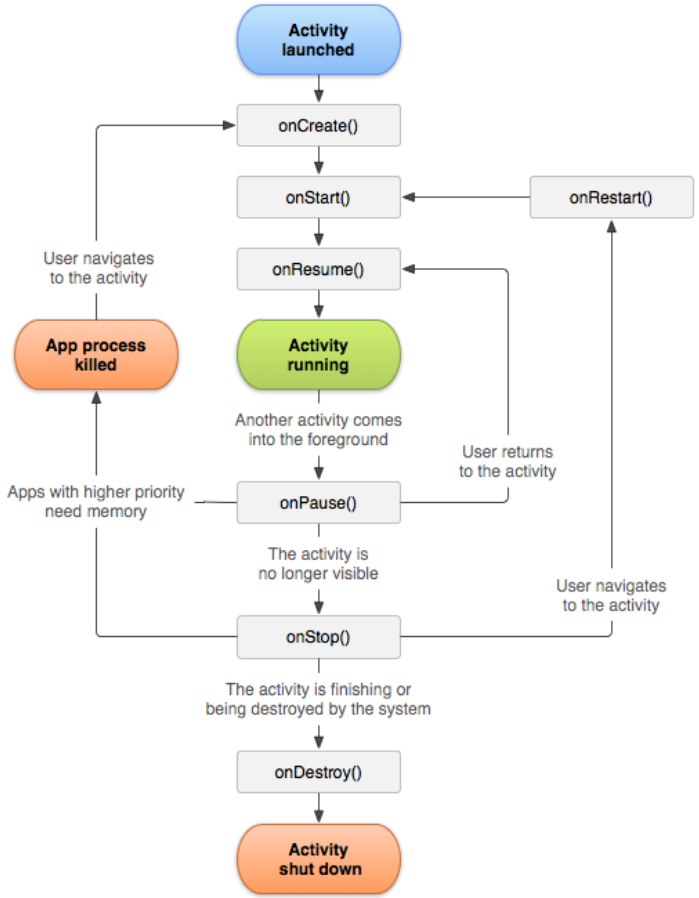
CAI-IP01 SMB116

Cycle de vie des **Activités** et **Intent**

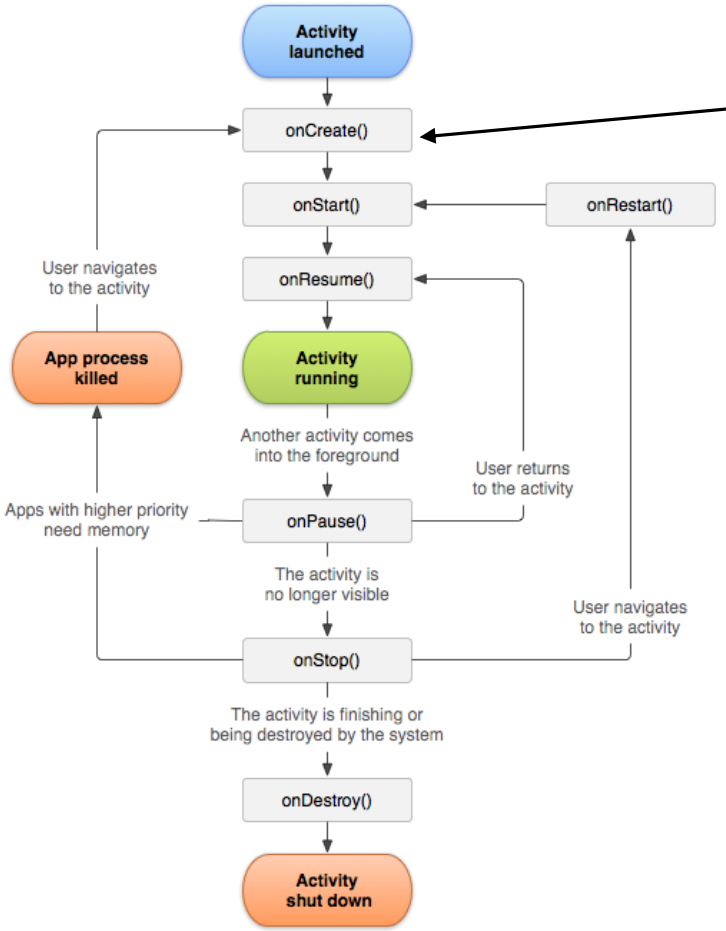
Activité

- Une application peu comporter plusieurs écrans
- A chacun de ces écrans correspond une Activité
- Une Activité comporte un certain nombre de méthodes
  onCreate(), onPause(), onStop(), onDestroy() …
- Pour créer une Activité il faut créer une classe héritant de la classe:
  android.app.Activity et la déclarer dans **Manifest.xml**
- En fonction des interactions de l'utilisateur Android appelle les méthodes de l'activité. (Design Pattern Inversion de contrôle)
- Android impose un Cycle de vie à l'Activité.

Cycle de vie d'une Activité



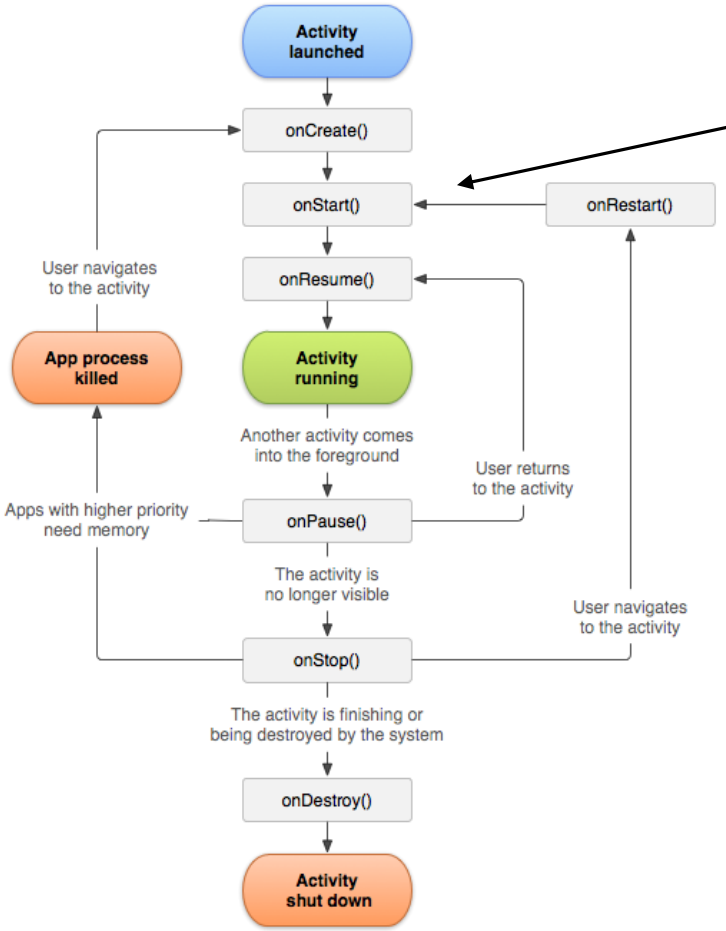
Cycle de vie d'une Activité – onCreate()



void onCreate (Bundle savedInstanceState)

- Méthode appelée à la création de l'activité
- Le bundle si il existe contient l'état de l'activité sauvé préalablement
- Prévue pour effectuer les initialisations (chargement de la vue)
- Si **onCreate()** se termine avec succès la méthode **onStart()** est appelée

Cycle de vie d'une Activité – onStart()

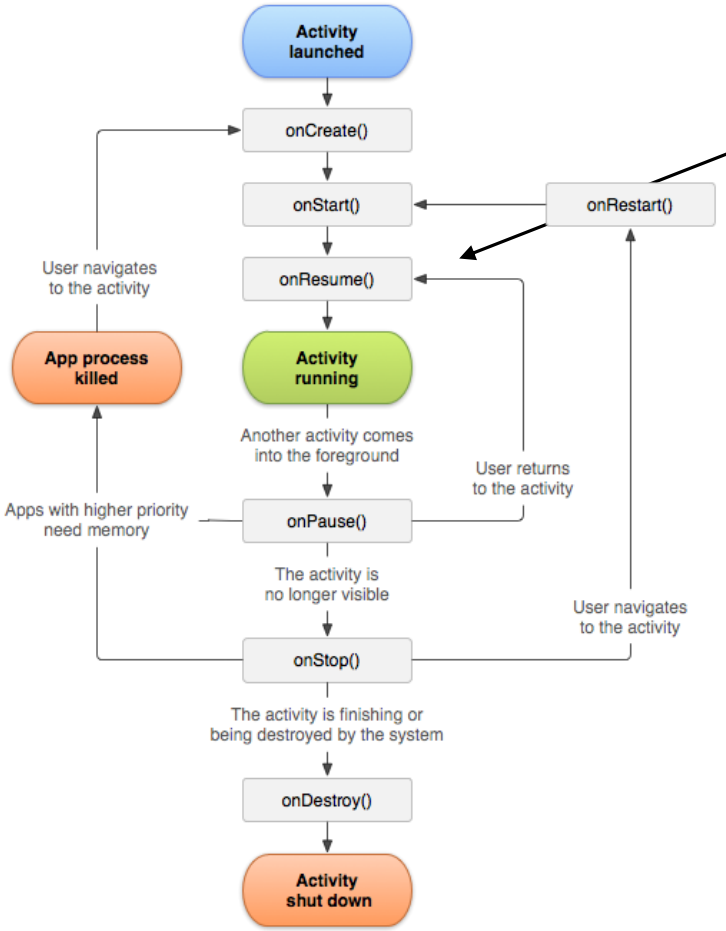


void onStart ()

- La méthode **onStart()** est appelée :
  - Soit après **onCreate()** si il s'agit de la création de l'activité
  - Soit après **onRestart()** si l'activité reprend après avoir été stoppée

- **onStart()** est appelée juste avant que l'activité soit visible à l'utilisateur.
- Après **onStart(),** la méthode **onRestoreInstanceState(bundle)** est apelée
- Immédiatement après, **onResume()** est appelée

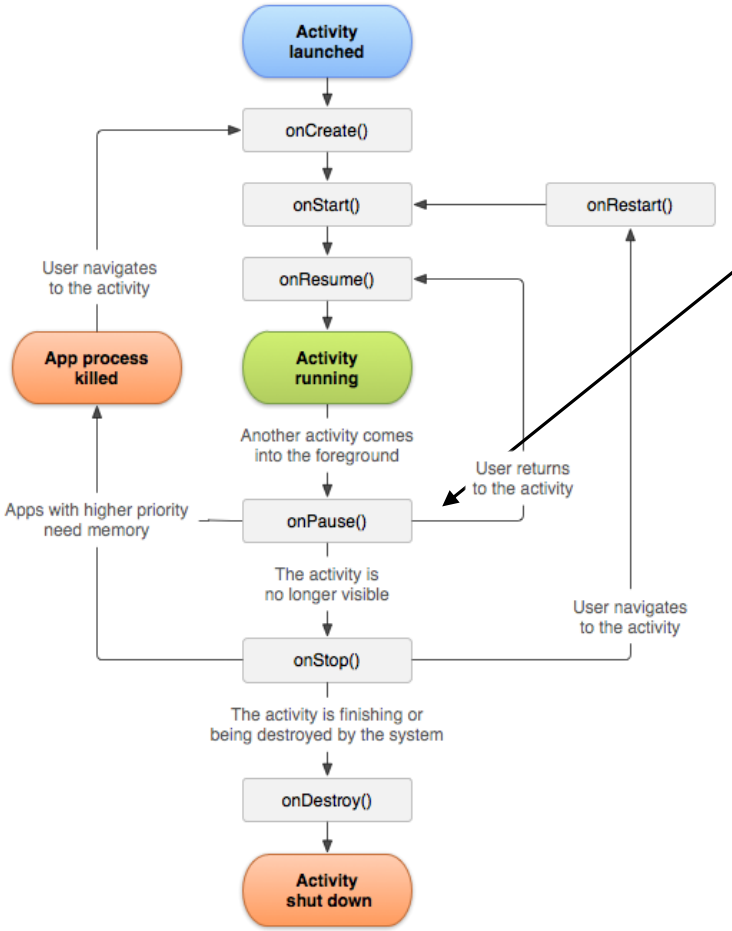
Cycle de vie d'une Activité – onResume()



void onResume()

- La méthode **onResume()** est appelée :
  - Soit après **onStart()** quand l'application est prête à interagir avec l'utilisateur.
  - Soit après lorsque l'activité reprend son exécution après avoir été mise en pause
- Si **onResume()** se termine avec succès l'activité se retrouve en phase d'exécution.

Cycle de vie d'une Activité – onPause()

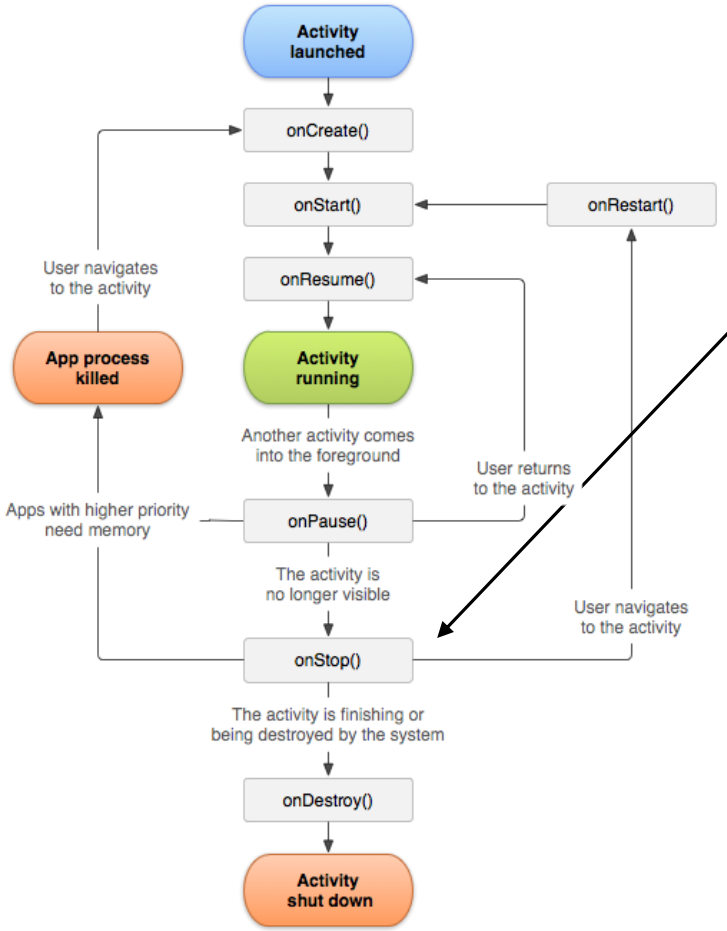


void onPause()

- La méthode **onPause()** est appelée si une autre activité passe au premier plan ou lorsque l'utilisateur quitte l'activité par la touche **Retour**
- Sur pause l'activité doit libérer toutes les ressources qu'elle utilise:
  - Capteur
  - BroadcoastReceiver

- Après **onPause()** l'activité est arrêtée
- Si le système a besoin de ressource (mémoire) Android peut décider de détruire l'activité mais **onSaveInstanceState(bundle)** sera invoquée avant pour permettre de sauvegarder l'état actuel de l'activité et de le restituer lorsque l'utilisateur réactivera l'activité.

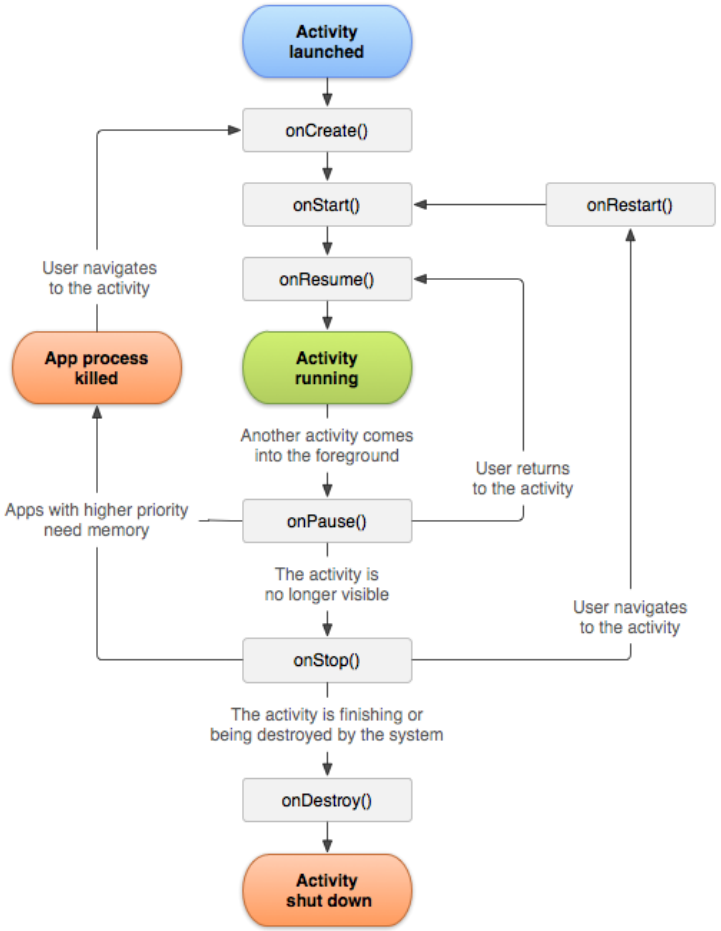
Cycle de vie d'une Activité – onStop()



void onStop()

- La méthode **onStop()** est appelée quand l'activité n'est plus visible à l'utilisateur.
  - Une autre application est passée au premier plan
  - L'activité va être détruite par Android ou suite à une interaction de l'utilisateur.
- Si le système a besoin de ressource (mémoire) Android peut décider de détruire l'activité mais **onSaveInstanceState(bundle)** sera invoquée avant pour permettre de sauvegarder l'état actuel de l'activité et de le restituer lorsque l'utilisateur réactivera l'activité.

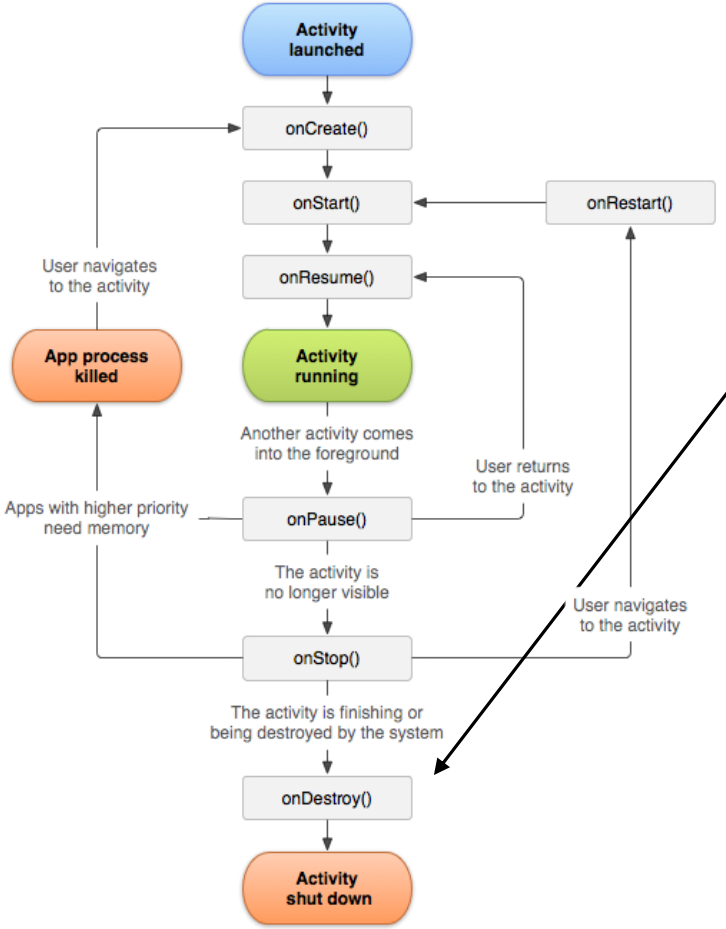
Cycle de vie d'une Activité – onRestart()



void onRestart()

- La méthode **onRestart ()** est appelée si l'activité est réactivée après avoir été stoppée

Cycle de vie d'une Activité – onDestroy()



void onDestroy()

- La méthode **onDestroy()** est appelée juste avant que l'activité soit détruite
- La destruction peut avoir été lancée par Android (récupération des ressources) ou par l'invocation de la méthode **finish()**

Cycle de vie d'une Activité – Démo (1)

- Toute activité hérite de la classe **Activity** on peut créer une **ActivityCycle** qui est une activité qui écrit dans la console ses méthodes invoquées par Android.

```
public class ActivityCycle extends AppCompatActivity {

    protected String activityName;

    public ActivityCycle() {}
    public ActivityCycle(String activityName) {this.activityName = activityName;}

    public String getActivityName() {
        return activityName != null ? activityName : this.getClass().getSimpleName().toUpperCase();
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.i(getActivityName(), "onCreate");
        setContentView(R.layout.activity_cycle);
        TextView labelCai = (TextView)findViewById(R.id.lblActivityName);
    }

    // Autres méthodes du cycle …
```

Cycle de vie d'une Activité – Démo (2)

- Pour créer des activités qui écrivent dans la console les méthodes invoquées par Android il suffit de les faire hériter de notre activité **ActivityCycle** et de les déclarer dans le **Manifest.xml**

```
public class MainActivity extends ActivityCycle {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
 // Suite du code de la MainActivity . . .
```

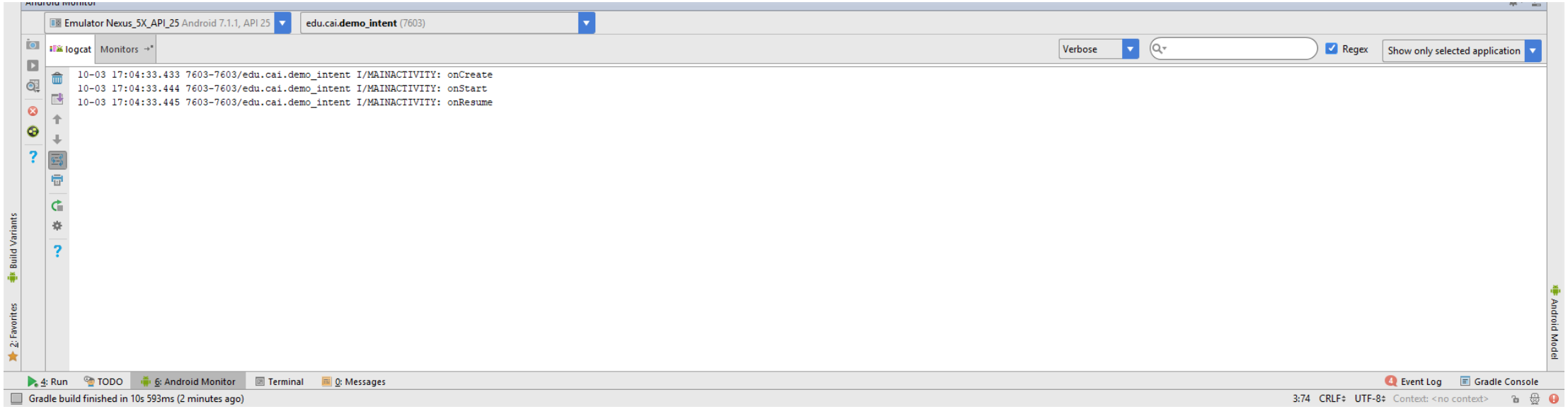
```
public class ActiviteCible extends ActivityCycle {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_activite_cible);
    }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=http://schemas.android.com/apk/res/android package="edu.cai.demo_intent">
    <application
       . . .
            <activity android:name=".MainActivity">
               <intent-filter>
                  <action android:name="android.intent.action.MAIN" />
                  <category android:name="android.intent.category.LAUNCHER" />
               </intent-filter>
            </activity>

            <activity android:name=".ActiviteCible">
            </activity>
    </application>
</manifest>
```

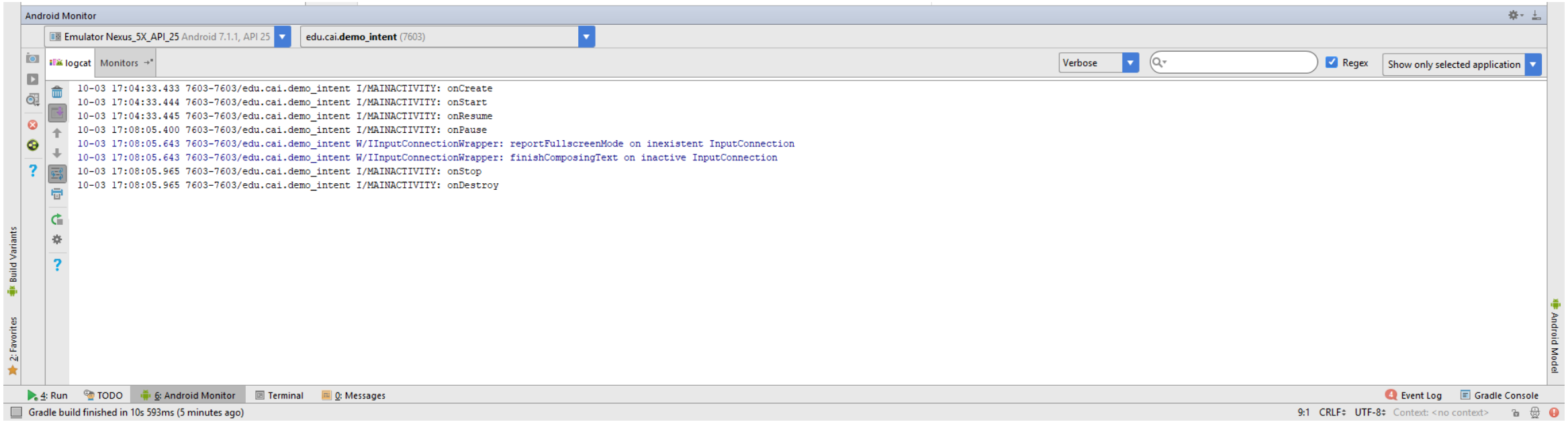
Cycle de vie d'une Activité – Démo (3)

Lancement de **MainActivity**: l'activité affiche des informations dans la console **logcat**



Cycle de vie d'une Activité – Démo (3)

Fermeture de **MainActivity**: l'activité affiche des informations dans la console **logcat**



Cycle de vie des Activités – Démo (4)

Lancement de **MainActivity** puis lancement **ActiviteCible** et enfin retour de l'**ActiviteCible** vers **MainActivity**

```
10-04 08:11:56.112 3290-3290/edu.cai.demo_intent I/MAINACTIVITY: onCreate
10-04 08:11:56.124 3290-3290/edu.cai.demo_intent I/MAINACTIVITY: onStart
10-04 08:11:56.126 3290-3290/edu.cai.demo_intent I/MAINACTIVITY: onResume
10-04 08:13:06.491 3290-3290/edu.cai.demo_intent I/MAINACTIVITY: onPause
10-04 08:13:06.507 3290-3290/edu.cai.demo_intent I/ACTIVITECIBLE: onCreate
10-04 08:13:06.512 3290-3290/edu.cai.demo_intent I/ACTIVITECIBLE: onStart
10-04 08:13:06.513 3290-3290/edu.cai.demo_intent I/ACTIVITECIBLE: onResume
10-04 08:20:13.076 3290-3290/edu.cai.demo_intent I/ACTIVITECIBLE: onPause
10-04 08:20:13.078 3290-3290/edu.cai.demo_intent I/MAINACTIVITY: onRestart
10-04 08:20:13.079 3290-3290/edu.cai.demo_intent I/MAINACTIVITY: onStart
10-04 08:20:13.079 3290-3290/edu.cai.demo_intent I/MAINACTIVITY: onResume
10-04 08:20:13.464 3290-3290/edu.cai.demo_intent I/ACTIVITECIBLE: onStop
10-04 08:20:13.464 3290-3290/edu.cai.demo_intent I/ACTIVITECIBLE: onDestroy
```

Intent et Intent Filters

- Un **Intent** est un message destiné à être envoyé pour demander une action à un autre composant d'application Android ou à permettre de communiquer entre composants. Un Intent permet:
  - **De démarrer une activité :** invocation de la méthode *startActivity(Intent)*
  - **De démarrer un service** (tâche en fond): invocation de *startService(Intent)*
  - **D'envoyer un message en Broadcoast** (aux abonnés): invocation de *sendBroadcoast(Intent)*
- **Intent explicite:** Spécifie le nom explicite du composant destinataire. Démarrage d'un composant par son nom.
- **Intent implicite:** A la place du nom du composant, le **nom d'une action à réaliser** est spécifié. Si un composant s'est déclaré capable de réaliser cette action **(IntentFilter)** il recevra l'Intent.

Intent Explicite (1)

- Lancement d'une activité par sa classe:
  - Création d'un Intent avec ce constructeur: `public Intent(Context packageContext, Class<?> cls)`
  - Lancement de l'activité avec *startActivity(intent)*

```
/**
 * Traitement bouton pour démarrer l'activité par sa classe
 * @param v
 */
public void onClickBtn_ParClasse(View v) {
    Intent intent = new Intent(this, ActiviteCible.class);
    startActivity(intent);
}
```

Limitation: La classe doit faire partie de la même application

Intent Explicite (2)

- Lancement d'une activité par le nom de sa classe:
  - Création d'un Intent
  - Affectation du nom de la classe de l'activité à démarrer
  - Lancement de l'activité avec *startActivity(intent)*

```
/**
 * Traitement bouton pour démarrer l'activité par le nom de sa classe
 * @param v
 */
public void onClickBtn_ParNomClasse(View v) {
    Intent intent = new Intent();
    intent.setClassName (getApplicationContext(), "edu.cai.demo_intent.ActiviteCible");
    startActivity(intent);
}
```

Intent Explicite (3)

- Lancement d'une activité par son nom de composant :
  - Création d'un Intent
  - Affectation du nom de composant créé avec ce constructeur: `public ComponentName(String pkg, String cls)`
  - Lancement de l'activité avec *startActivity(intent)*

```
/**
 * Traitement bouton pour démarrer l'activité par le nom du composant
 * @param v
 */
public void onClickBtn_ParNomComposant(View v) {
    Intent intent = new Intent();
    intent.setComponent(new ComponentName("edu.cai.demo_intent", "edu.cai.demo_intent.ActiviteCible"));
    startActivity(intent);
}
```

Intent Explicite (4)

- Lancement d'une activité d'une autre application par le nom de sa classe:
  - Création d'un Intent
  - Affectation du nom du package et du nom de la classe de l'activité à démarrer
  - Lancement de l'activité avec *startActivity(intent)*

```
/**
 * Traitement bouton pour démarrer l'activité d'une autre application par le nom de sa classe
 * @param v
 */
public void onClickBtn_ParNomClasseAutreApplication(View v) {
    Intent intent = new Intent();
    intent.setClassName ("edu.cai.applicationcible", "edu.cai.applicationcible.ActivityCibleExterne");
    startActivity(intent);
}
```

Intent Implicite (1)

- Lancement d'une activité par un intent Implicite: (Pattern Publish/ Subscribe)
- L'Intent est créé avec une **Action** et une **Category**, le système Android doit trouver l'activité correspondante : **celle déclarée avec un filtre correspondant**.
  - Une activité est déclarée dans le Manifest avec son filtre (IntentFilter) :

```
<!-- Activite Bonjour avec son IntentFilter -->
<activity android:name=".ActivityActionBonjour">
    <intent-filter>
        <action android:name="bonjour.ACTION" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

  - Création d'un Intent implicite
  - Affectation à l'Intent de l'Action, de la Category, de l'Uri que doit traiter l'activité à démarrer
  - Lancement de l'activité avec *startActivity(intent)*

Intent Implicite (2)

- Lancement de l'activité par un intent Implicite

- L'Intent est créé avec une **Action** et une **Category**, le système Android doit trouver l'activité correspondante : celle déclarée avec un filtre correspondant.
  - Création d'un Intent implicite
  - Affectation du nom de l'action demandée (à résoudre par Android)
  - Lancement de l'activité avec *startActivity(intent)*

```
/**
 * Traitement bouton pour démarrer l'activité par
 * une intent implicite basée sur le nom d'une action
 * @param v
 */
public void onClickBtn_ImpliciteParNomAction(View v) {
    Intent intent = new Intent();
    intent.setAction("bonjour.ACTION");
    startActivity(intent);
}
```

Si Android ne parvient pas à trouver une activité correspondant à l'action une Exception est levée

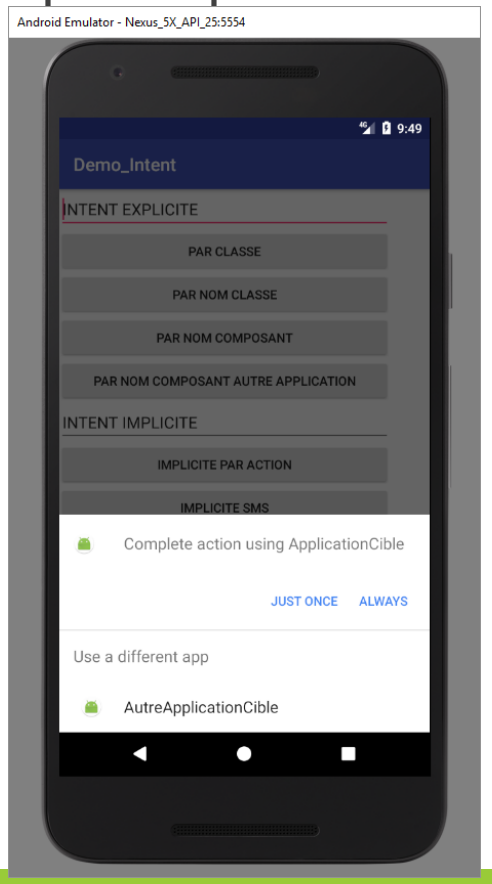
Intent Implicite - Plusieurs Activités candidates (3)

- Si plusieurs activités sont enregistrées pour la même Action que se passe-t-il ?

```
<!-- Activite Bonjour avec son IntentFilter -->
<activity android:name=".ActivityActionBonjour">
    <intent-filter>
        <action android:name="bonjour.ACTION" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>

<!--Une autre activité qui repond à l'action Bonjour -->
<activity android:name=".ActivityBonjourCoucou">
    <intent-filter>
        <action android:name="bonjour.ACTION" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

```
/**
 * Traitement bouton pour démarrer l'activité par
 * une intent implicite basée sur le nom d'une action
 * @param v
 */
public void onClickBtn_ImpliciteParNomAction(View v) {
    Intent intent = new Intent();
    intent.setAction("bonjour.ACTION");
    startActivity(intent);
}
```



Intent Implicite (4)

- Avant de lancer une Activité par un Intent Implicite on peut tester si il existe une Activité candidate

```
PackageManager pm = getPackageManager();
Intent intent = new Intent();
intent.setAction("bonjour.ACTION");
// queryIntentActivities(Intent intent, int resolveInfoFlags);
List<ResolveInfo> l = pm.queryIntentActivities(intent, 0);
if(l.size() > 0) {
    startActivity(intent);
}
```

Intent Implicite : Activités principales

- Toutes les activités avec comme Intent-Filter:
  - **Action =** `android.intent.action.MAIN`
  - **Catégorie =** `android.intent.category.LAUNCHER`

Sont sélectionnées par le Launcher d'Android comme point d'entrée de leur application. Leur Icone est affichée dans la liste des applications qui peuvent être lancées par l'utilisateur.

```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Intent Implicite : action + arguments

- Demande de l'activité pour afficher une page Web

```
/**
 * Traitement bouton pour démarrer l'activité par
 * une intent implicite basée :
 *  - sur le nom d'une action
 *  - des paramètres: url + texte  ==> Navigation Web
 * @param v
 */
public void onClickBtn_ImpliciteActionViewWeb(View v) {
    Intent intent = new Intent();
    // Action générique VIEW
    intent.setAction(Intent.ACTION_VIEW);
    // Url de l'élément à montrer
    intent.setData(Uri.parse("http://www.sysord.com"));
    startActivity(intent);
}
```

L'activité à sélectionner pour réaliser l'action **View** dépend de l'URI de la ressource à afficher

Activité pour afficher une URI personnalisée (1)

- Enregistrement de l'activité chargé d'afficher les URI de type **« cai »** (dans le Manifest)

```
<!--Une activité pour visualiser les url CAI -->
<activity android:name=".ActivityCAI">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="cai" />
    </intent-filter>
</activity>
```

Activité pour afficher une URI personnalisée (2)

- L' activité chargée d'afficher les URI de type « cai » (sur create: récupération valeur de l'extra **"info_cai")**

```
public class ActivityCAI extends ActivityCycle {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_cai);
        // Obtention de l'intent ayant demandé le démarrage de l'activité
        Intent intent = getIntent();
        // obtention de l'argument info_cai
        String infoCai = intent.getStringExtra("info_cai");
        // affectation au label texte
        EditText labelCai = (EditText)findViewById(R.id.labelCAI);
        labelCai.setText(infoCai);
    }
}
```

Activité pour afficher une URI personnalisée (2)

- Création et propagation de l'Intent pour afficher l'URI **"cai://cai.2isa"**

```
/**
 * Traitement bouton pour démarrer l'activité par
 * une intent implicite basée :
 *  - sur le nom d'une action
 *  - des paramètres: url + texte ==> visualiser CAI
 * @param v
 */
public void onClickBtn_ImpliciteActionViewPersonnalise(View v) {
    Intent intent = new Intent();
    // Action générique VIEW
    intent.setAction(Intent.ACTION_VIEW);
    // Url de l'élément à montrer
    intent.setData(Uri.parse("cai://cai.2isa"));
    // autre paramètre:
    intent.putExtra("info_cai", "Cours de SMB116");
    startActivity(intent);
}
```

Passer des arguments au lancement d'une Activité (1)

- Pour passer des arguments à une Activité Il y a les **« Extras »** qui peuvent être affectés à un Intent. L'activité recevant l'Intent lit les **« Extras »** qu'il contient

- Les **Extras** sont stockés dans un Bundle (une Map: Clé, Valeur)

Intent	putExtra (String name, Bundle value) Add extended data to the intent.
Intent	putExtra (String name, Parcelable[] value) Add extended data to the intent.
Intent	putExtra (String name, Serializable value) Add extended data to the intent.
Intent	putExtra (String name, int[] value) Add extended data to the intent.
Intent	putExtra (String name, float value) Add extended data to the intent.

Bundle	getExtras () Retrieves a map of extended data from the intent.
int	getFlags () Retrieve any special flags associated with this intent.
float[]	getFloatArrayExtra (String name) Retrieve extended data from the intent.
float	getFloatExtra (String name, float defaultValue) Retrieve extended data from the intent.
int[]	getIntArrayExtra (String name) Retrieve extended data from the intent.
int	getIntExtra (String name, int defaultValue) Retrieve extended data from the intent.

La classe Intent met à disposition des méthodes pour déposer et obtenir des valeurs d'**Extra** primitives (String, int, float, double etc.) **Mais comment passer des objets métiers ?**

Passer des Objets au lancement d'une Activité

- Pour passer des arguments de type objet à une Activité Il y a les méthodes:

Intent	putExtra (String name, Parcelable[] value) Add extended data to the intent.
Intent	putExtra (String name, Serializable value) Add extended data to the intent.

- **Serializable :** Objet qui peut être transformé en un flux, une série de bits pour être enregistré ou transporté. L'Objet original pourra être reconstitué à partir de sa sérialisation, la série de bits obtenue. (processus automatique)

- **Parcelable :** Equivalent à Serializable mis à part que la transformation et la restitution est à la charge du programmeur.

Le processus de Serialization est puissant mais lent à cause de l'introspection, la « Parcelisation » est plus rapide car le code de parcelisation / déparcelisation est écrit en spécifique par le programmeur: Pas besoin d'instropection.

Pour passer un argument de type Objet par les Extra il doit être **Serializable** ou bien **Parcelable**

Les Objets Parcelables

- https://developer.android.com/reference/android/os/Parcelable.html

```
public class EtudiantCAI implements Parcelable {

    protected Long id;
    protected String nom;
    protected String prenom;
    //-----------------------
    // Parcelable
    //-----------------------

    /**
     * Retourne un indicateur du type de contenu du Parcelable
     * @return
     */
    @Override
    public int describeContents() {
        return 0;
    }

    /**
     * Parcelisation
     */
    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeLong(id);
        dest.writeString(nom);
        dest.writeString(prenom);
    }
```

```
    /**
     * Constructeur privé pour reconstruire le
     * Parcelable à partir du Parcel
     * @param in
     */
    private EtudiantCAI(Parcel in) {
        this.id = in.readLong();
        this.nom = in.readString();
        this.prenom = in.readString();
    }

    /**
     * Créateur d'objet à partir du Parcel
     */
    public static final Creator<EtudiantCAI> CREATOR = new
Creator<EtudiantCAI>() {
        @Override
        public EtudiantCAI createFromParcel(Parcel in) {
            return new EtudiantCAI(in);
        }
        @Override
        public EtudiantCAI[] newArray(int size) {
            return new EtudiantCAI[size];
        }
    };

}
```

Activité qui retourne un résultat (2)

- Une activité est lancé par un Intent mais avec une autre méthode:

Utilisation de **startActivityForResult** à la place de **startActivity** :

```
public void startActivityForResult(Intent intent, int requestCode)
```

- **Intent** pour lancer l'application
- **requestCode** pour identifier la requête **startActivityForResult** ayant lancée l'**Activité**

- L'activité retourne le résultat par l'invocation de la méthode **setResult().**

```
public final void setResult(int resultCode)
public final void setResult(int resultCode, Intent data)
```

`resultCode` : code de retour

`data` : Intent pour transmettre résultats en extras

- Le Résultat est obtenu par la méthode **onActivityResult** redéfinie par l'activité appelante

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
```

`requestCode` : code passé par l'activité appelante sur **`startActivityForResult`**

`resultCode` : code de retour

`data` : Intent contenant les valeurs résultats en extras ou en uri contenue dans data

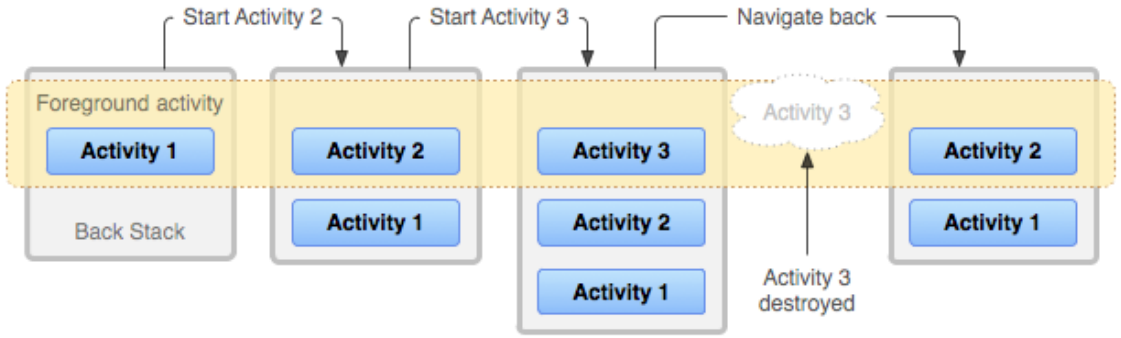
Quelques Intents utiles

- https://developer.android.com/guide/components/intents-common.html

Alarm Clock, Calendar, Camera, Contacts, Email, File Storage, Maps, Music et Video, Note, Phone, Sms etc.

Pile d'activités (1)

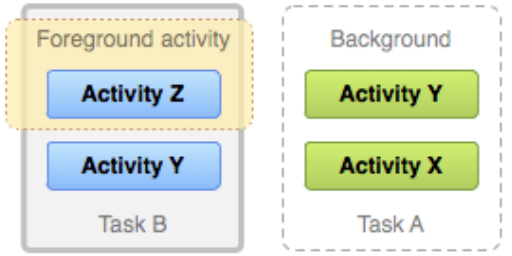
- L'utilisateur d'une application navigue entre activités :
  - Lorsqu'une Application est lancée l'Activité (a1) principale est affiché (Foreground)
  - Lorsque l'Activité (a1) courante lance une autre Activité (a2) : l'Activité courante (a1) est mise en pause et la nouvelle activité (a2) passe au premier plan
  - Lorsque l'Activité (a2) courante lance une autre Activité (a3) : l'Activité courante (a2) est mise en pause et la nouvelle activité (a3) passe au premier plan
  - Lorsque l'Activité (a3) se termine ou que l'utilisateur appuie sur **Retour** l'activité (a3) est détruite et l'Activité précédente (a2) retourne au premier plan.



Les activités sont stockées dans une file LIFO, une Pile

Pile d'activités (2)

- Chaque Application est une « **Task** » et a sa propre pile d'activité
- Lorsque une application est active l'activité au sommet de sa pile est active et au premier plan, les autres applications sont inactives. Une seule **Task** est active à la fois.
- Lorsqu'une application est active, si l'utilisateur quitte l'activité courante
  - Par « Retour » : l'activité courante est retirée de la pile et est détruite.
    - Si la pile n'est pas vide l'activité au sommet de la pile devient active
    - Sinon l'application se termine, sa pile d'activités est détruite et l'écran principal (Launcher) est affiché
  - Par « Home » : la **« Task »** de l'application courante est suspendue et l'écran « Launcher » est affiché
    - Lorsque l'utilisateur demande à partir du « launcher » à lancer la même application, la « Task » suspendue redevient active.
    - A un instant donné, Il peut y avoir plusieurs « Task » suspendues , si Android manque de mémoire il pourra détruire l'une d'elle.
- Commande pour afficher la pile d'activité avec adb: `adb shell dumpsys activity`



Agir sur la pile d'activités (1)

- Passer au premier plan une Activité déjà présente dans la pile :

```
Intent intent = new Intent(this, MainActivity.class);
// Si il existe une activité du type "MainActivity" dans la pile
// cette activité et toutes les activités au-dessus d'elle dans la
// pile sont supprimées.
// Une nouvelle instance d'un activité du type "MainActivity«  est créée et
placé en haut de la pile

// Ex: La pile contient A, B, C, D. D est l'activité courante. Si D Lance B
// avec le flag CLEAR_TOP
// alors la pile contiendra : A, B

intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(intent);
```

Agir sur la pile d'activités (2)

- Passer au premier plan une Activité déjà présente dans la pile :

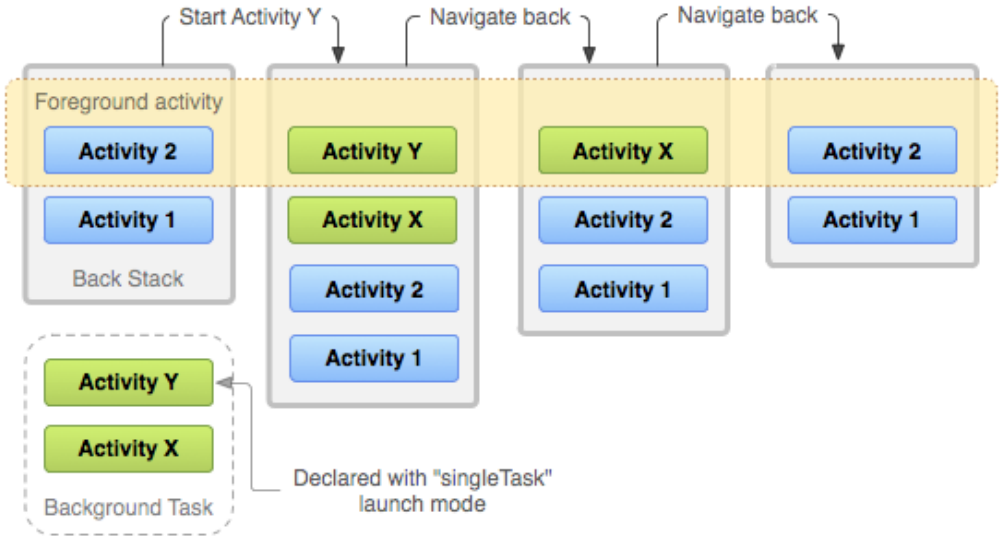
https://developer.android.com/reference/android/content/Intent.html#FLAG_ACTIVITY_REORDER_TO_FRONT

```
Intent intent = new Intent(this, MainActivity.class);
// L'activité "MainActivity" existante passe au premier plan,
// les autres activités de la pile ne sont pas supprimées

// Ex: La pile contient: A, B, C, D. D est l'activité courante.
// Si D Lance B avec le flag FLAG_ACTIVITY_REOARDER_TO_FRONT
// alors la pile contiendra : A, C, D, B
intent.addFlags(Intent.FLAG_ACTIVITY_REORDER_TO_FRONT);
startActivity(intent);
```

Agir sur la pile d'activités (3)

- Configuration de la stratégie de démarrage d'une activité à sa déclaration dans le Manifest :
  - Attribut **android:launchMode** sur l'élément **<activity>** https://.../activity-element#lmode



Une activité peut être lancée dans une nouvelle **Task**. Même si existe plusieurs piles les activités sont enchainées comme si il n'existait qu'une une seule pile.

Passage de paramètres entre Activités – Via l'Application (1)

- Créer une Classe héritant de **android.app.Application** :

```
public class MonApplication extends android.app.Application {
    /**
     * Structure pour stockage des paramètres
     */
    protected Map<String, Object> parametres = new HashMap<>();
    /**
     * Affectation d'une valeur
     */
    public void putValue(String key, Object value) {
        parametres.put(key, value);
    }
    /**
     * Obtention d'une valeur
     */
    public <T> T getValue(String key) {
        return (T) parametres.get(key);
    }
}
```

```
// Dépose d'une valeur de paramètre dans une activité
((MonApplication)getApplication()).putValue("PARAM1", "ValeurParam1");
Intent intent = new Intent(this, ActivityB.class);
startActivity(intent);
```

```
// Lecture de la valeur par l'activité cliente
super.onResume();
AlertDialog.Builder dlgAlert  = new AlertDialog.Builder(this);
dlgAlert.setMessage("Le parametre" + ((MonApplication)getApplication()).getValue("PARAM1"));
```

- La déclarer dans le Manifest en tant qu'Application:

```
<application android:name=".MonApplication"  ...
```

Passage de paramètres entre Activités – Avec Persistance

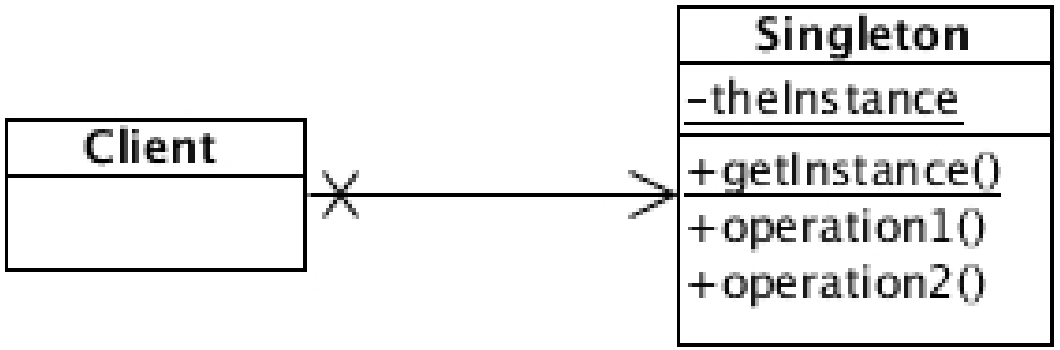
- Utilisation des **SharedPreferences** :

```
// Dépose d'une valeur de paramètre dans une activité
SharedPreferences preferences = getSharedPreferences("preferences", MODE_PRIVATE);
SharedPreferences.Editor editor = preferences.edit();
editor.putString("PARAM2", "Valeur Parametre 2 ");
editor.commit();
Intent intent = new Intent(this, ActivityC.class);
startActivity(intent);
```

```
// Lecture de la valeur par l'activité client
super.onResume();
AlertDialog.Builder dlgAlert  = new AlertDialog.Builder(this);
SharedPreferences preferences = getSharedPreferences("preferences", MODE_PRIVATE);
dlgAlert.setMessage("Le parametre" + preferences.getString("PARAM2", ""));
```

Passage de paramètres entre Activités – La Pattern Singleton

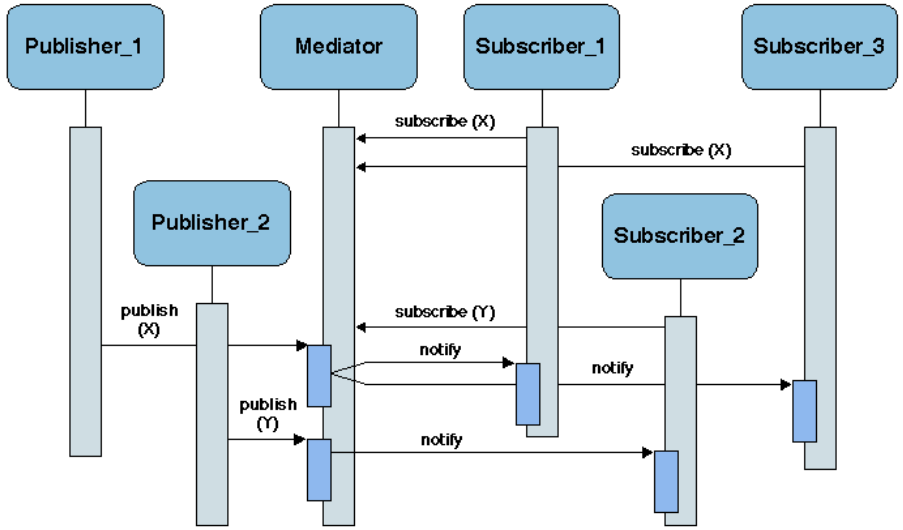
- Utiliser le design Pattern Singleton :



- **Avantages :**
  - Simple à mettre en place
- **Inconvénients :**
  - Méthode limitée à une même application
  - Besoin de gérer les accès concurrents

Passage de paramètres par Broadcoast Le Pattern Publish Subscribe

- **Rappel le pattern publish-Subscribe:**
  - Des **Subscriber** se sont abonnés à un type de message
  - Un **Mediator** enregistre les abonnements et se charge de **notifier** les abonnés lorsque un message du type attendu est envoyé
  - Des **Publisher** envoient un message par l'intermédiaire du **Mediator**
  - La **Notification** aux **Subscriber** peur être effectuée selon un ordre particulier, la propagation de cette notification peut être stoppée par un **Subscriber**.



Passage de paramètres par Broadcoast

- La classe **android.content.Context** (les activités héritent de cette classe):

```
// Enregistrement d'un Souscripteur
public abstract Intent registerReceiver(BroadcastReceiver receiver, IntentFilter filter);

// Retrait d'un souscripteur
public abstract void unregisterReceiver(BroadcastReceiver receiver);

// Publication d'un message sur un thème (thème défini dans l'intent)
public abstract void sendBroadcast(Intent intent);

// Publication ordonnée d'un message sur un thème
// Les souscripteur sont notifiés dans l'ordre de leur priorité
// Un souscripteur peut interrompre la notification par l'appel de la méthode abortBroadcoast()
public abstract void sendOrderedBroadcast(Intent intent, String receiverPermission);
```

Passage de paramètres par Broadcast

- Le BroadcastReceiver :

```
// Méthode appelé quand le BroadcoastReceiver reçoit un Intent
public abstract void onReceive(Context context, Intent intent);


// Méthode à appeler dans onReceive() pour déterminer
// si le BroadCoast en cours de traitement est un OrderedBroasCoast
public final boolean isOrderedBroadcast()


// Méthode à appeler dans onReceive() pour stopper la propagation de l'Intent
// quand il s'agit d'un OrderedBroasCoast
public final void abortBroadcast();
```

Exemple publish subscribe (1)

- Le BroadcastReceiver :

```
public class Constants {
    public static final String TICK___ACTION = "time.action";
    public static final String TICK_COUNT__EXTRA = "count";
}
```

```
public class TimeReceiver extends BroadcastReceiver {

    protected String receiverId;

    public TimeReceiver(String receiverId) {
        this.receiverId = receiverId;
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        // réception tick
        Log.i(receiverId, String.format("%d", intent.getLongExtra(Constants.TICK_COUNT__EXTRA, 0)));
        // Consommation des OrderedBroadcast pairs
        if(intent.getLongExtra(Constants.TICK_COUNT__EXTRA, 0) % 2 == 0) {
            if(isOrderedBroadcast()) {
                abortBroadcast();
            }
        }
    }
}
```

Exemple publish subscribe (2)

- L'activité : Enregistrement / Désenregistrement des Receiver

```
public class MainActivity extends AppCompatActivity {

    TimeReceiver receiver1 = new TimeReceiver("RECEIVER_1");
    TimeReceiver receiver2 = new TimeReceiver("RECEIVER_2");
    TimeReceiver receiver3 = new TimeReceiver("RECEIVER_3");

    //...

    @Override
    protected void onResume() {
        super.onResume();

        IntentFilter filter = new IntentFilter(Constants.TICK___ACTION);
        filter.setPriority(100);
        getApplicationContext().registerReceiver(receiver1, filter);

        filter = new IntentFilter(Constants.TICK___ACTION);
        filter.setPriority(50);
        getApplicationContext().registerReceiver(receiver2, filter);

        filter = new IntentFilter(Constants.TICK___ACTION);
        filter.setPriority(1000);
        getApplicationContext().registerReceiver(receiver3, filter);
    }
```

```
    @Override
    protected void onPause() {
        super.onPause();
        getApplicationContext().unregisterReceiver(receiver1);
        getApplicationContext().unregisterReceiver(receiver2);
        getApplicationContext().unregisterReceiver(receiver3);
    }
```

Exemple publish subscribe (3)

- L'activité : Génération et envoi de messages en Broadcoast

```
Thread thread;
AtomicLong count = new AtomicLong(0);

public void onBtnStartClicked(View v) {
    Log.i("BTN", "START");
    startSendBroadcoast(false);
}

public void onBtnStartOrderedClicked(View v) {
    Log.i("BTN", "START ORDERED");
    startSendBroadcoast(true);
}

public void onBtnStopClicked(View v) {
    Log.i("BTN", "STOP");
    if(thread != null) {
        thread.interrupt();
    }
    thread = null;
}
```

```
protected void startSendBroadcoast(final boolean isOrdered) {
    thread = new Thread() {
        public void run() {
            while(!isInterrupted()) {
                SystemClock.sleep(1000);
                count.set(count.longValue() + 1);
                Intent intentTick = new Intent(Constants.TICK___ACTION);
                intentTick.putExtra(Constants.TICK_COUNT__EXTRA, count.get());
                Log.i("MAIN_ACTIVITY", "TICK:" + count.get());
                if(isOrdered) {
                    sendOrderedBroadcast(intentTick, null);
                } else {
                    sendBroadcast(intentTick);
                }
            }
        }
    };
    thread.start();
}
```