Android les Services, Receiver et processus

jean-michel Douin, douin au cnam point fr
version : 04 Avril 2017

Notes de cours

Sommaire

- Services
  - Cycle de vie
    - création, démarrage et arrêt
  - Service local
  - Service et processus
- IntentService
- Service et Notification
- Receveur et services
- Service global
  - AIDL

Note: A chaque thème un exemple complet et une démonstration

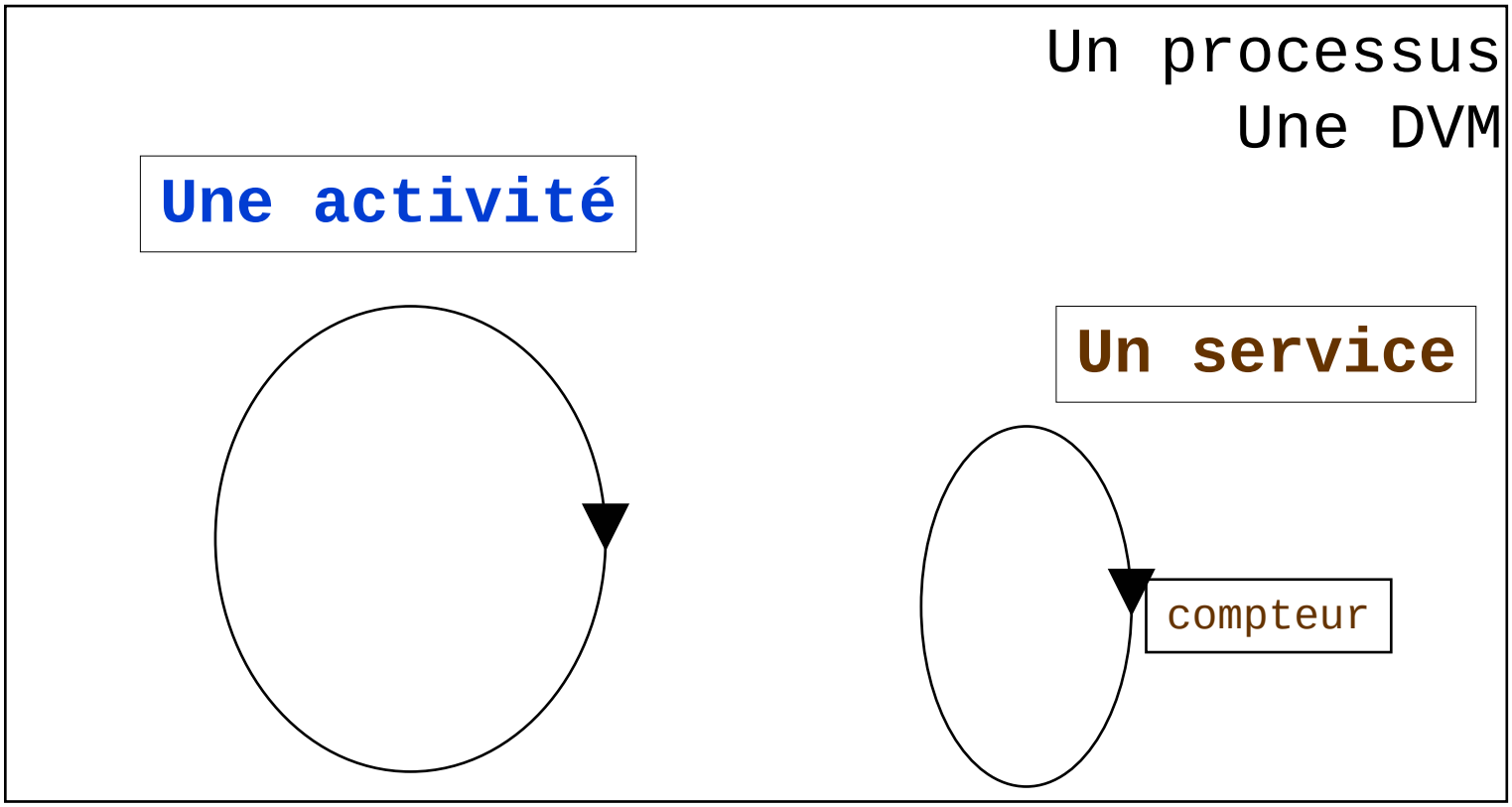
Intergiciel : fonctionnalités attendues

- **Sélection du service**
  - Comment ?
  - Recherche et découverte ?
- **Démarrage et arrêt du service**
  - Re-démarrage ?
- **Communication, clients/service**
  - Au sein de la même application ?
    - Même DVM ? Plusieurs ?
  - D'une application externe ?
    - Quel langage commun ?
    - Passage des paramètres ?
      - Sérialisation/dé-sérialisation ?

Architectures présentées

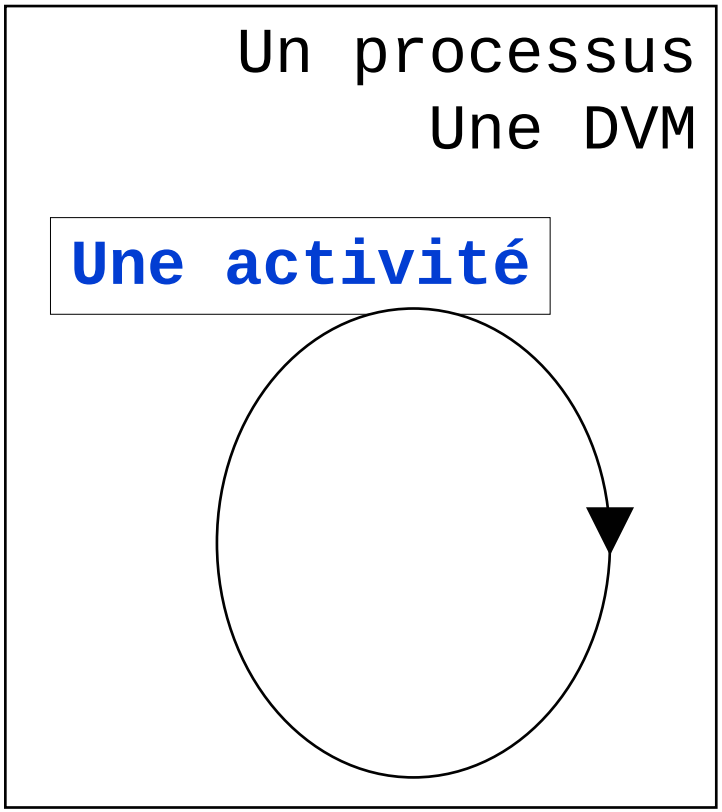
- **Simplifiées pour l'exemple…**
- **Une application, un processus/une DVM**
  - **Une activité et un service**
- **Une application, deux processus/deux DVM**
  - **Une activité,**
  - **Communication inter-processus**
- **Deux applications**
  - **Des clients**
  - **Un service distant**
  - **Une librairie commune aux clients et au service**

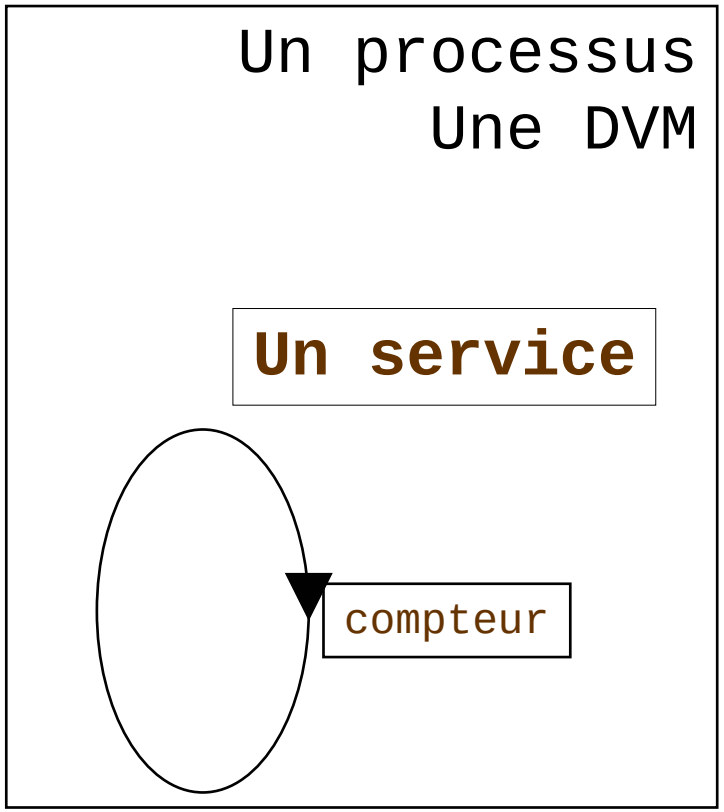
Sommaire : Architectures présentées 1/3



- **Service local: une activité, un service**
  - **Échange de données**
    - **Envoi des paramètres depuis l'activité par une intention**
    - **Réception, lecture des résultats**
      - **Par l'usage de variables globales, ici le *compteur* (même machine)**

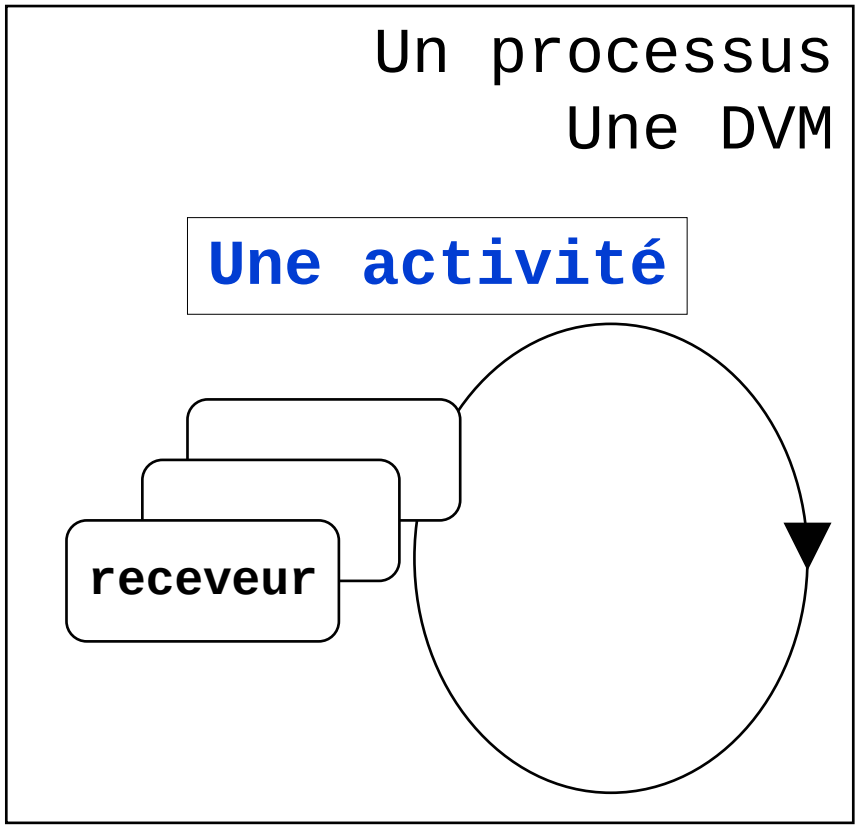
Sommaire : Architectures présentées 2/3

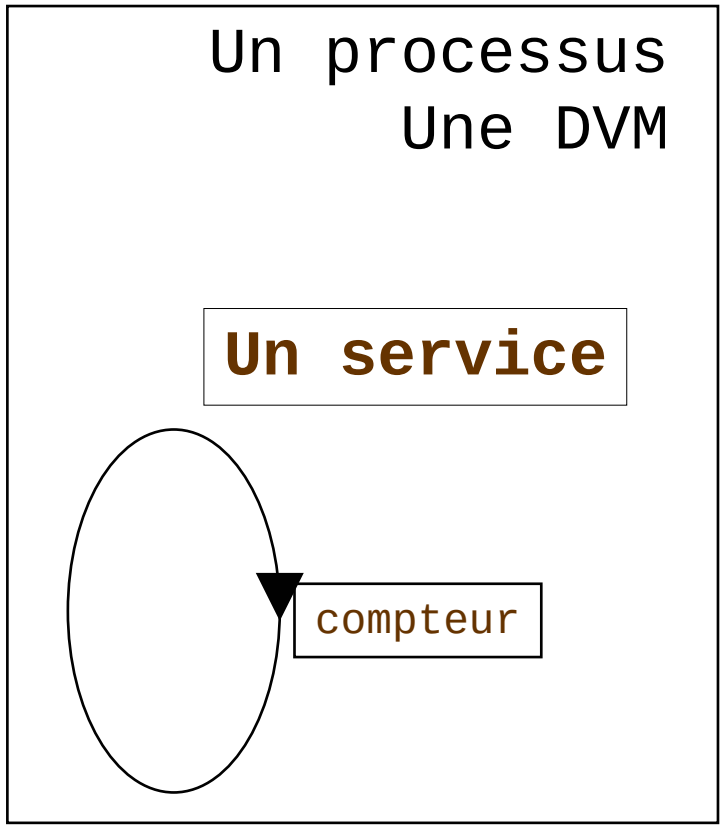




- **Un service global, une activité**
  - **Échange de données**
    - **Envoi depuis l'activité par une intention**
    - **Réception des résultats**
      - **Messenger et handler**
      - **ServiceConnection, accès via un mandataire AIDL**

Sommaire : Architectures présentées 3/3





- **Un service global, une activité, des receveurs**
  - **Échange de données**
    - **Envoi depuis l'activité par une intention**
    - **Réception des résultats**
      - **broadcastReceiver,**
        - » sendBroadcast, sendOrderedBroadcast

Bibliographie utilisée

- http://developer.android.com/reference/android/app/Service.html
- http://www.vogella.com/articles/AndroidServices/article.html
- http://grail.cba.csuohio.edu/~matos/notes/cis-493/lecture-notes/Android-Chapter22-Services.pdf
- http://davanum.wordpress.com/2007/12/15/android-listen-for-incoming-sms-messages/

Service

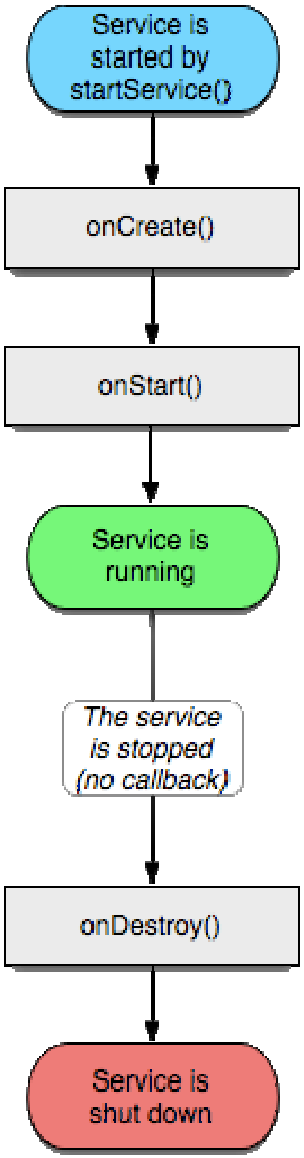
- **En tâche de fond**
  - **Pas d'IHM**
    - **J'écoute un fichier audio mp3, pendant que je navigue sur le web,**
    - **Dès que je m'empare du mobile, une sonnerie retentit,**
    - **Les coordonnées GPS sont envoyées régulièrement,**
    - **Dès que je m'approche d'une zone dangereuse mon mobile vibre…,**
    - **…**

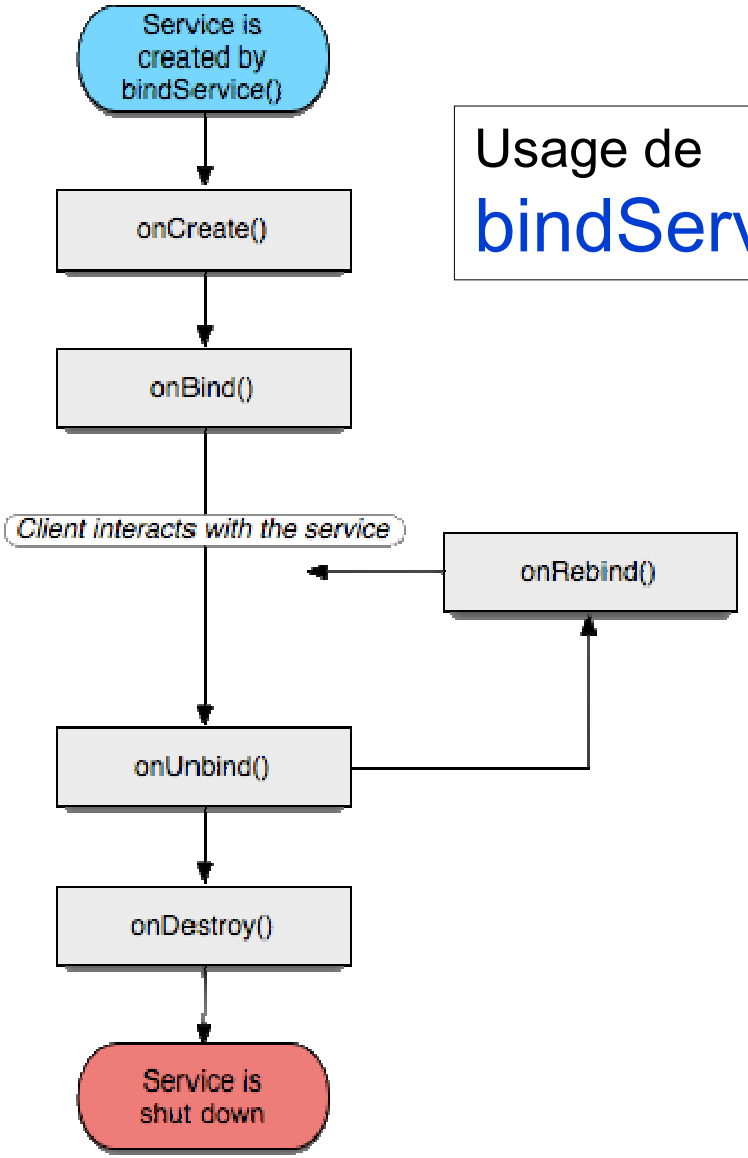
- **Toujours disponibles**
  - **Locaux**
    - **Service personnel, inaccessible pour les autres applications**
      - **Service, IntentService**
        - » En accès restreint
  - **« Distants »** *sur le même mobile*
    - **Accessible aux autres applications**
      - **Langage commun AIDL**
      - **Service, IBinder, ServiceConnection**

Services : cycle de vie, deux styles

Quelque soit l'architecture choisie

Usage de
startService





Usage de
bindService

Services, schéma des exemples

- **Un service local**
  - **Une horloge**
    - **Un compteur est incrémenté chaque seconde …tic…tic…tic…tic…tic…tic**
  - **Lecture synchrone du compteur de tics**
    - **À son initiative, le client questionne … (combien de tics ?)**
  - **Un message contenant le nombre de tics est envoyé par le service**
    - **Réception asynchrone par le client … (veuillez recevoir ce nombre de tics, svp)**
  - **A chaque incrémentation du compteur une notification a lieu**
    - **Dans la barre des notifications… (le compteur a changé de valeur !)**
  - **A chaque incrémentation un envoi aux receveurs agréés est effectué**
    - **Envois asynchrones aux multiples receveurs éventuels selon un ordre**

Services, exemples suite

- **Un service global**
  - **Une horloge**
    - **Le service est accessible depuis plusieurs activités**

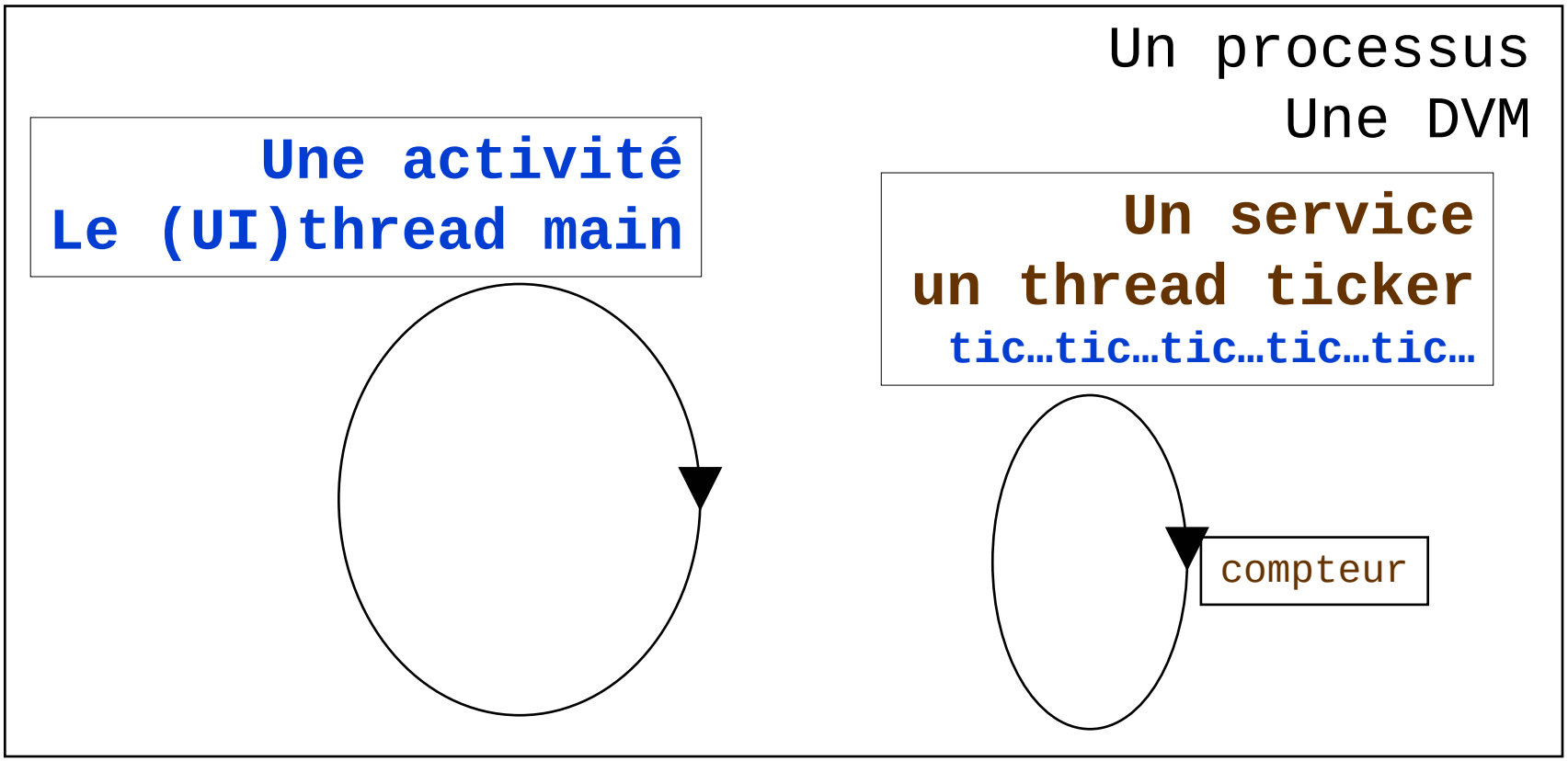
- **Un autre service global associé à un receveur**
  - **Un compteur de SMS entrants en trois étapes**

1. **AIDL, les services, le langage commun entre les clients et le service**
   - **Le squelette du programme**
     - » Types primitifs et Parcelable

2. **Mise en place du receveur de SMS**
   - **Liaison receveur et service**

3. **Filtrage du contenu des SMS, codage,décodage du contenu**

Architecture du premier exemple



- Depuis l'activité,
  - Démarrage du service par **startService**
  - accès au compteur de tics par l'appel d'une méthode de classe, lecture de la valeur
- *Activité : classe ServiceLocalActivity*
- *Service : classe TimeService*

Un service local c'est :

Une déclaration XML et un héritage de la classe Service

Une déclaration

- **Balise *service* dans la configuration** *AndroidManifest.xml*
  - *Par défaut le service se trouve dans le même processus que l'activité*

Une classe au sein de la même application

- **public class TimeService extends android.app.Service{**
  - **public void onCreate(){…}**
  - **public int onStartCommand(Intent intent, int flags, int startId) {…}**
    - *onStart appelle cette méthode*
  - **public onDestroy{…}**

Un Service Local, au sein d'une activity

- **Une horloge**
  - **Un compteur est incrémenté chaque seconde** …tic…tic…tic…tic…tic…tic

- Le fichier AndroidManifest.xml mentionne le service (***TimeService***)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="service.local"
    android:versionCode="1"
    android:versionName="1.0">
   <uses-sdk android:minSdkVersion="8" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".ServiceLocalActivity"  android:label="@string/app_name">
            <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>

        <service android:name= " .TimeService" ></service>

    </application>
</manifest>
```

Démarrage du service 1/2

- **Au sein de l'activité,**
  - **Déclaration de l'intention**

  - **Intent intent = new Intent(this, TimeService.class);**
    - **this : le contexte (l'activité initiatrice)**

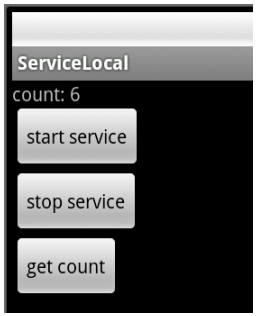
    - **TimeService.class : le service,**

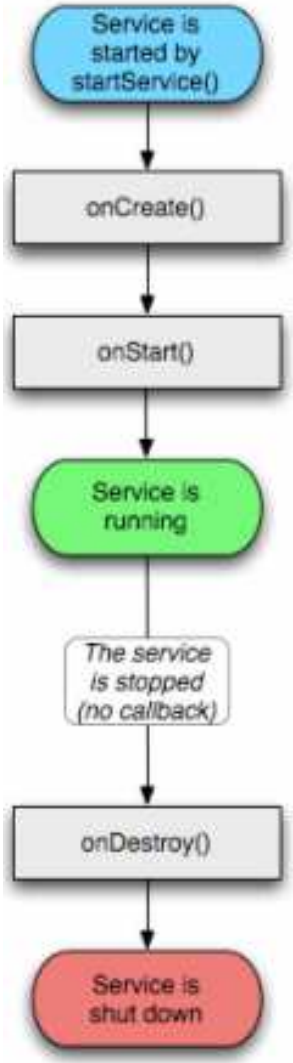
      » dans la même application (un package référencé par Android)

    - *L'intent peut contenir paramètres, exemple*
      - **intent.putExtra("pid", android.os.Process.*myPid*());**

Démarrage et arrêt du service 2/2

- **Au sein de l'activité,**
  - Démarrage et arrêt du service,
    - → Cycle de vie du service





startService(intent);

Démarrage effectif du service,

appel de onCreate, onStart->onStartCommand du service

stopService(intent);

Arrêt du service,

appel de onDestroy du service

Schéma de programme du service, 1/3

```
public class TimeService  extends Service {

// à la suite de l'appel de startService
public void onCreate(){  // est appelé
 }


// non renseigné dans cette version qui utilise startService
public IBinder onBind(Intent i) {
  return null;
}
```

Schéma de programme du service, 2/3

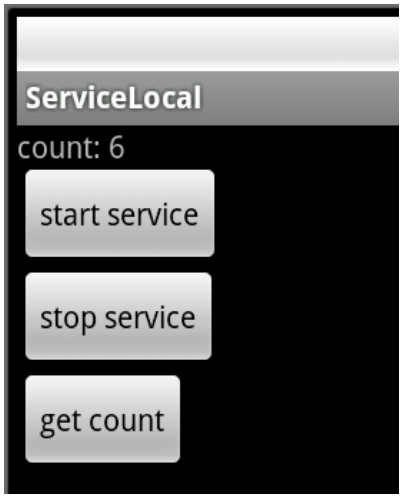
```
private static Thread ticker;
private static AtomicLong count;

// suite de l'appel de startService, cf. cycle de vie
public int onStartCommand(Intent intent, int flags, int startId) {

    if(ticker==null){
        count = new AtomicLong();              // accès concurrents
        ticker = new Thread(new Ticker());  // un thread local
        ticker.start();
      }

    return START_STICKY; // pour un re-démarrage automatique avec un intent==null
       //  START_NOT_STICKY , le redémarrage n'est pas souhaité
       // START_REDELIVER_INTENT // pour un re-démarrage avec l'intent initial

       // la constante retournée à destination du Context induit un comportement du service
       // http://developer.android.com/reference/android/app/Service.html
}
```

Méthode run (le ticker), du service 3/3

```
// l'horloge et son thread, « du classique »
private class Ticker implements Runnable{
 public void run(){
    while(!ticker.isInterrupted()){
    try{
      Thread.sleep(1000);
      count.set(count.longValue()+1L);
    }catch(Exception e){
      return;
    }
  }
 }

// accès en lecture, par l'activité (même DVM)
public static long getCount(){
  return count.get();
}
```

Le client, une activity, un affichage

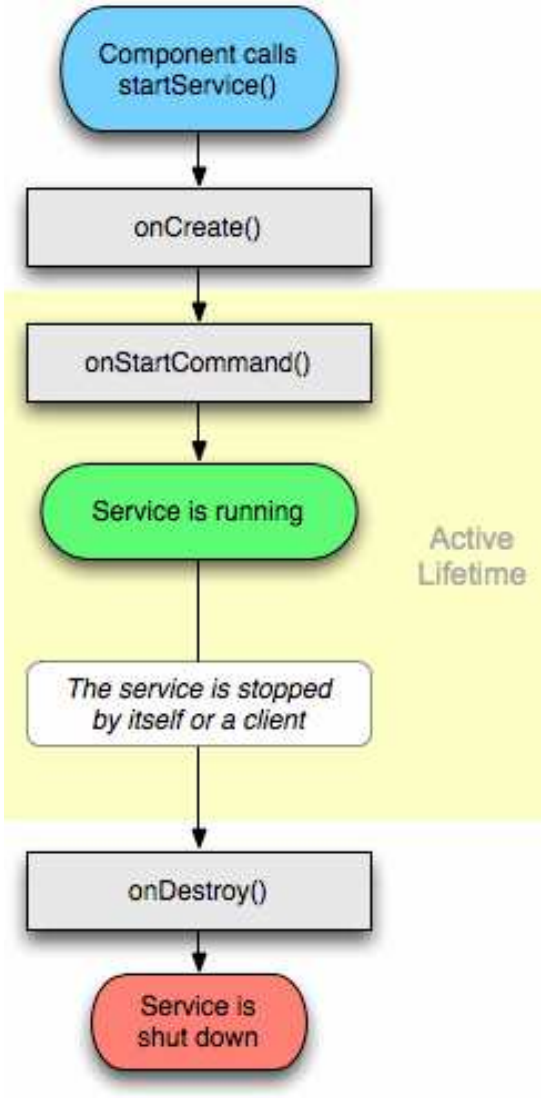
```
public class ServiceLocalActivity extends Activity

 public void onClickStart(View v){
    Intent intent = new Intent(this, TimeService.class);
    intent.putExtra("pid", android.os.Process.myPid());
    started = startService(intent) != null;
 }

  public void onClickStop(View v){
    started = !stopService(new Intent(this, TimeService.class));
  }

public void onClickGetCount(View v){
    if(started)
      texte.setText("count: " + TimeService.getCount());
    else
      texte.setText("service non démarré");
 }
```



Méthodes de classe
Même DVM (tout va bien)

Démonstration

Résumé: Service par startService

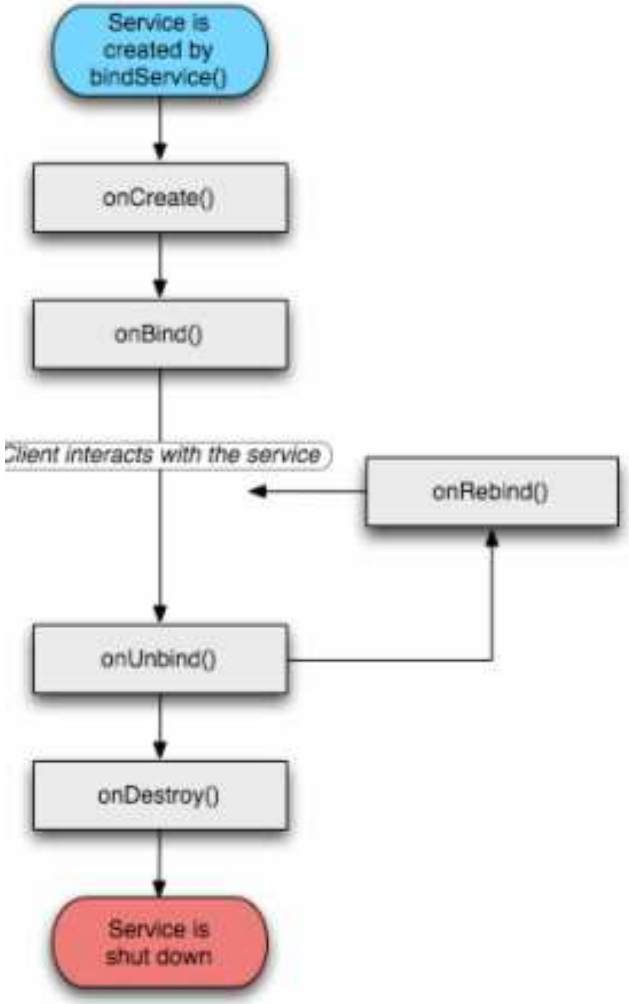


- **Le service a démarré et le reste indéfiniment…**
  - **même si l'application initiatrice n'existe plus …**
- **stopService ou stopSelf**
  - **permettent d'arrêter le service**

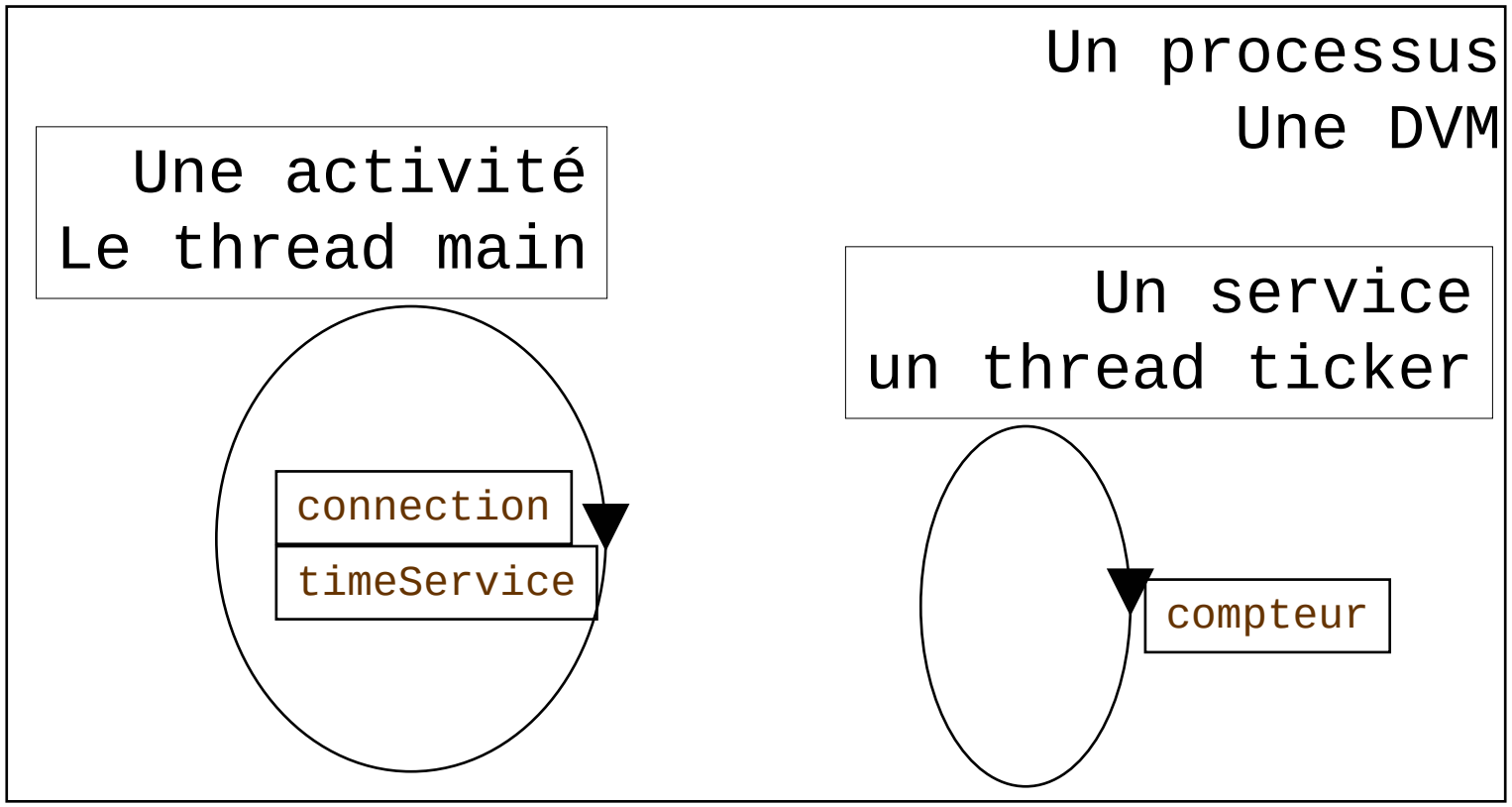
Le même service, avec bindService

- **bindService**
  - **Appel de**
    - **onCreate, onBind**

- **Le système android**
  - **Établit la connexion**



Le même exemple



- De l'activité,
  - Démarrage du service par **bindService**
  - connection est connue d'android, sera utilisé à chaque liaison (bindService)
  - accès au compteur de tics par une instance du service (timeService) fournie par le système

Cycle de vie

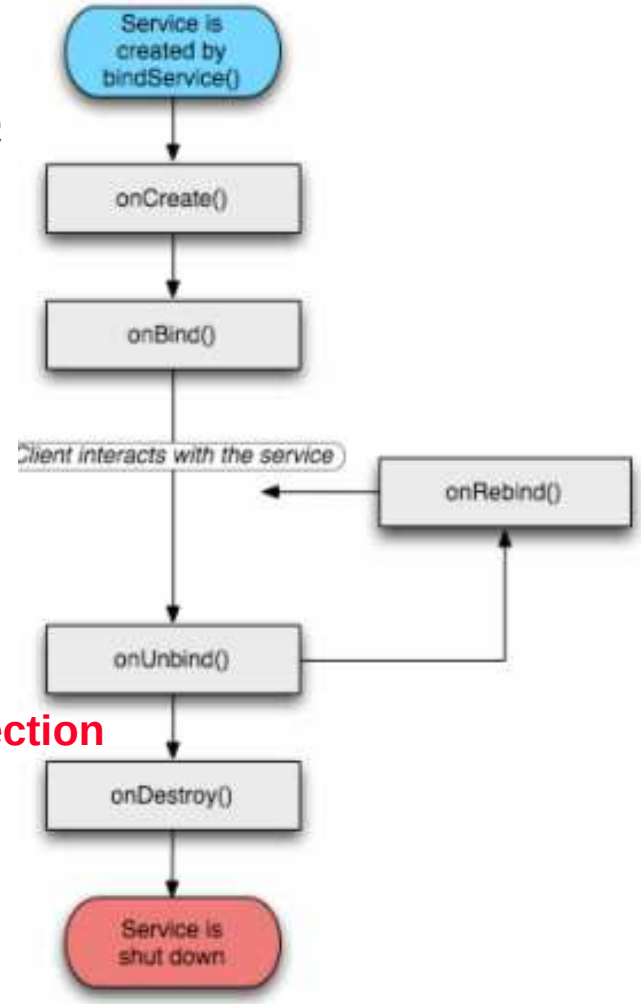
```
Intent intent = new Intent(this, TimeService.class);
ServiceConnection connection = ..
```

bindService(intent, connection, Context.*BIND_AUTO_CREATE*);

bindService **chez le client déclenche *onCreate* du service**

L'intergiciel Android déclenche la méthode connected() de connection

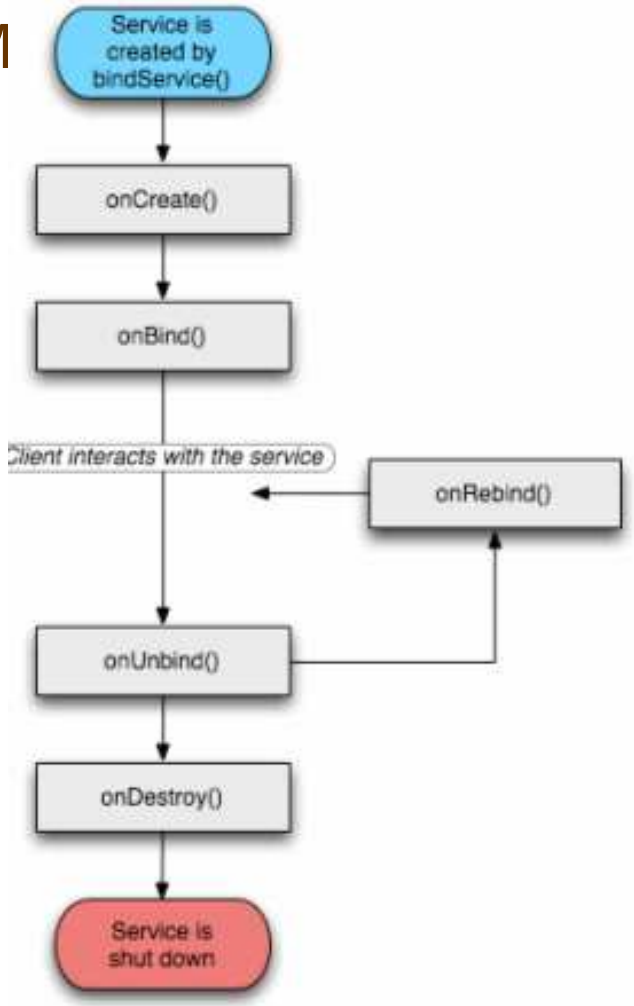
connection est une référence chez le client



TimeService, le même programme, onBind

Le service et l'activité sont sur la même DVM

```
public class TimeService  extends Service {


// interface avec le client, liaison
@Override
public IBinder onBind(Intent arg0) {
  return binder;
}

final IBinder binder = new LocalBinder();
public class LocalBinder extends Binder {
  TimeService getService() {
    return TimeService.this;
  }
}
```



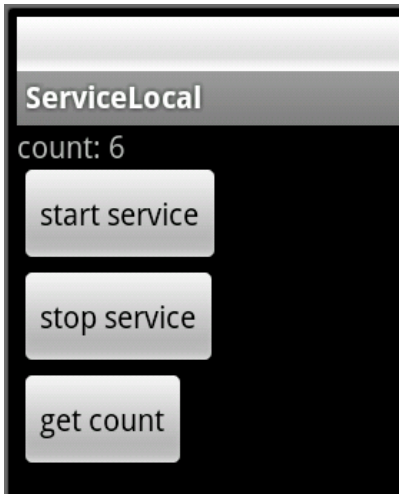
Le service : redéfinition de onCreate

- **Idem au Service précédent, un thread Horloge est créé**
  - …tic…tic…tic…tic…tic…tac…tic

```
// à la création du service, appel de bind
public void onCreate(){
// idem
    if(ticker==null){
        count = new AtomicLong();
        ticker = new Thread(new Ticker());
        ticker.start();
      }
```

Le client, une activity, un affichage

```
public class ServiceLocalActivity extends Activity

 public void onClickStart(View v){
   Intent intent = new Intent(this, TimeService.class);
   intent.putExtra("pid", android.os.Process.myPid());
    bindService(intent, connection, Context.BIND_AUTO_CREATE);
    started = true;      }


  public void onClickStop(View v){
     if(started){
          unbindService(connection);
          started = false;
    }
  }

public void onClickGetCount(View v){
    if(started)
      texte.setText("count: " + timeService.getCount());
    else
      texte.setText("service non démarré");
 }
```



Variable d'instance
Même DVM (simple)

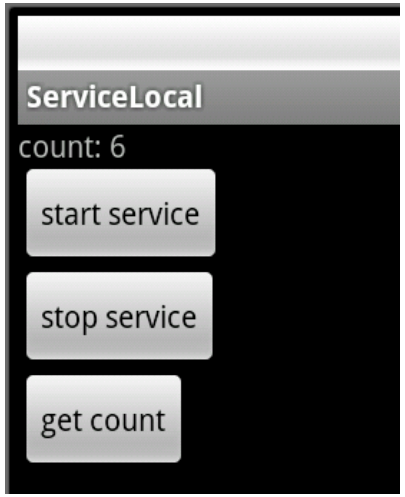
Cette connexion est asynchrone…

- **Liaison composant <-> service**

- **Au sein de l'activité cette instance est utilisée par le système**

```
private ServiceConnection connection = new ServiceConnection() {

  public void onServiceConnected(ComponentName className,
                                 IBinder service) {

   timeService = ((TimeService.LocalBinder)service).getService();
  }

  public void onServiceDisconnected(ComponentName className){
   // déclenchée si le service est global et stoppé
  }
};
```

Démonstration



- **Le service a démarré et le reste tant qu'un client est lié**
  - **Appel de bindService(…, connection,…);**
- **Le service est arrêté**
  - **Si tous les clients liés ont appelé unbindService(connection);**

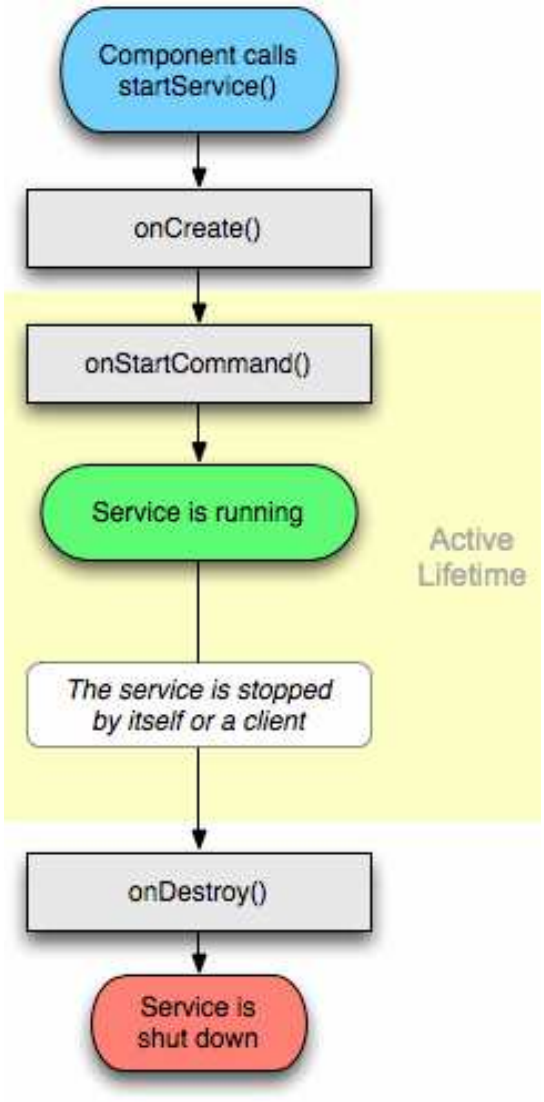
Discussion …

- **Avec l'exemple du cours**
  - **Le thread/Compteur est créé dans la méthode onCreate, onBind**
    - **suite à l'appel de bindService**

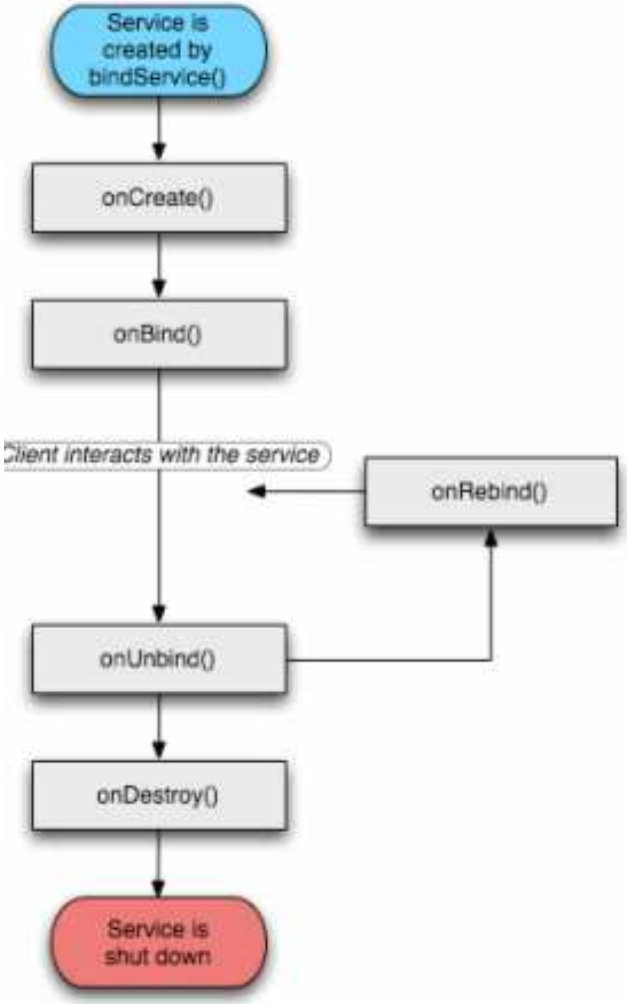
Pratique standard :

1. **Appel de startService**
   - **Le service démarre: appel de onCreate puis onStartCommand**
2. **Appel de bindService**
   - **Une connexion est établie avec le service : onCreate puis onBind**

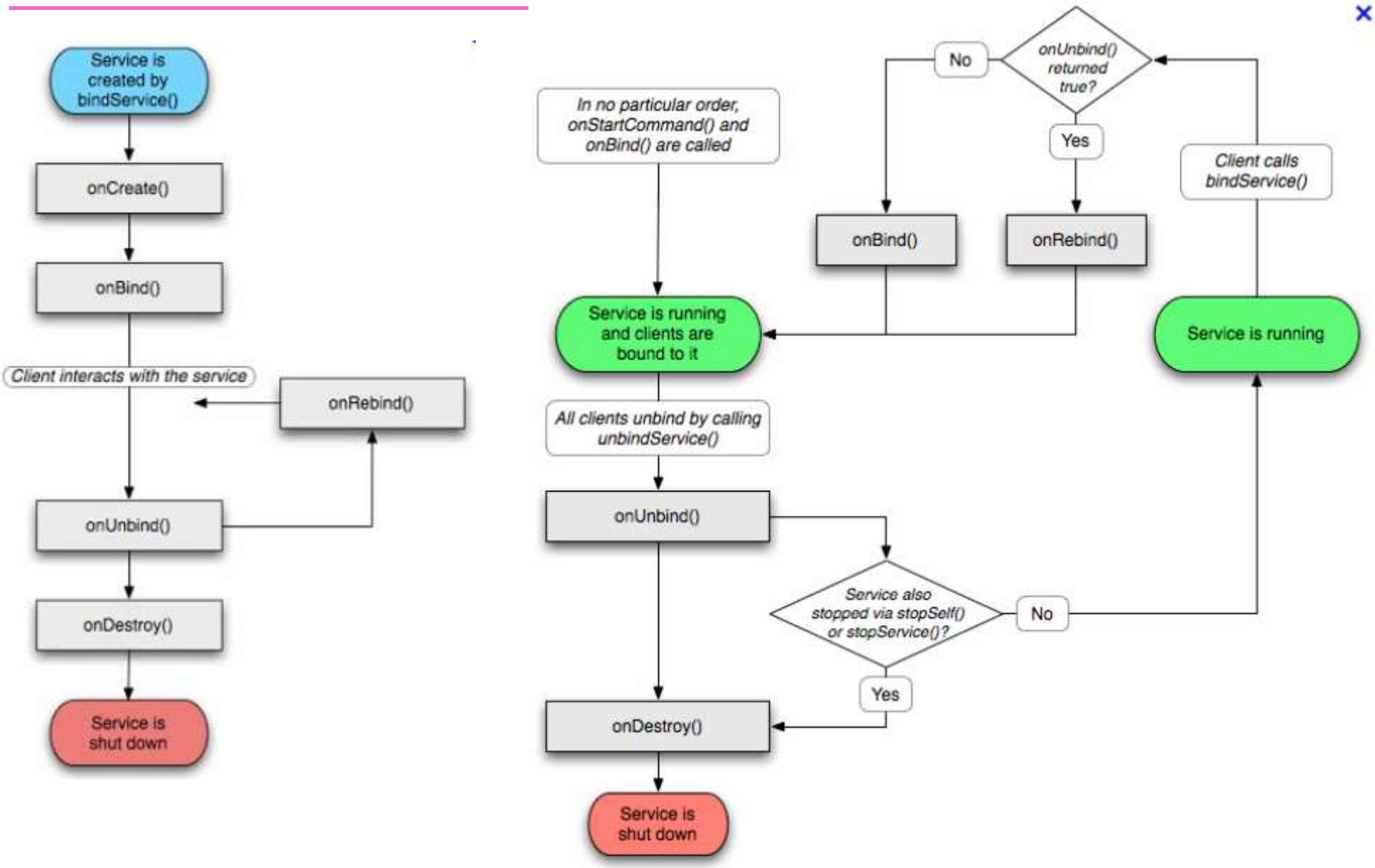
Pratique standard



1. **startService**
   1. onCreate
   2. onStartCommand
2. **bindService**
   1. onCreate
   2. onBind



Résumé: Service par bindService



Un résumé

- **Un service local**
  - **Hérite de Service**
  - **Stratégies possibles**
    - **startService**

    ou
    - **bindService**

    ou les deux combinées en pratique
  - **Les exemples présentés utilisent la même DVM**
    - **Accès « simplifié » aux services rendus**
      - **Accès par une méthode de classe,** *(usage de startService)*
      - **ou une instance de service.** *(usage de bindService)*

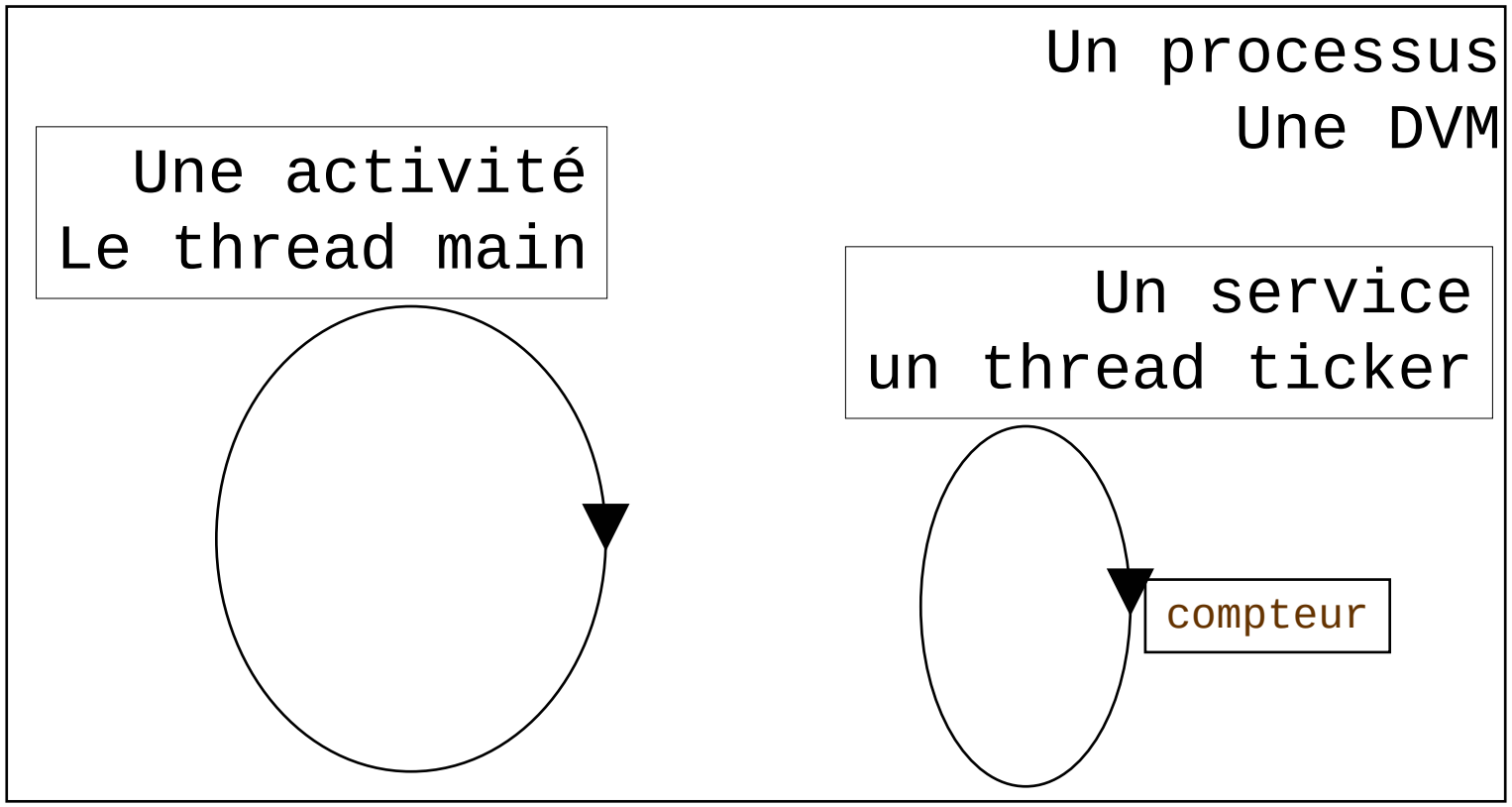
Sommaire suite

- **IntentService**
  - **Un service créé pour l'occasion**
- **Un service global**

IntentService

- **Une autre écriture pour un service prêt à l'emploi**
  - **Le système Android s'occupe de tout**
    - **Appels en interne de onCreate, OnStart et onDestroy**
  - **Un service est créé, déclenche la méthode onHandleIntent puis se termine**
  - **La méthode `void onHandleIntent(Intent intent)` est à implémenter**

Le premier exemple revu par IntentService



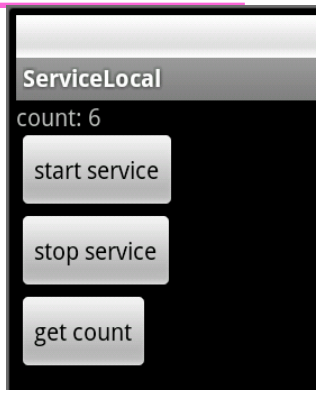
- De l'activité,
  - Démarrage du service par startService

- *Activité : classe ServiceLocalActivity*
- *IntentService : classe TimeService*

IntentService : tout en un

```
// code identique … ticker est une variable de classe

protected void onHandleIntent(Intent intent) {
  Bundle extras = intent.getExtras();

  if (extras != null) {

// Ici l'intent contient les commandes destinées au service

    boolean interrupted  = extras.getBoolean("interrupted");

    if(interrupted){
      interrupt();
    }else{
      if(ticker==null){
        ticker = new Thread(this);
        ticker.start();
    }
  }
}
}

// rappel: une instance est créée, onHandleIntent est appelée à chaque startService
```

IntentService : le client s'est adapté




```
public void onClickStart(View v){
    Intent intent = new Intent(this, TimeService.class);

    started = startService(intent) != null;

    }

  public void onClickStop(View v){
    Intent intent = new Intent(this, TimeService.class));

    intent.putExtra("interrupted", true);
     startService(intent); started=false;
  }
```

Attention: Deux nouvelles instances du service ont été créées

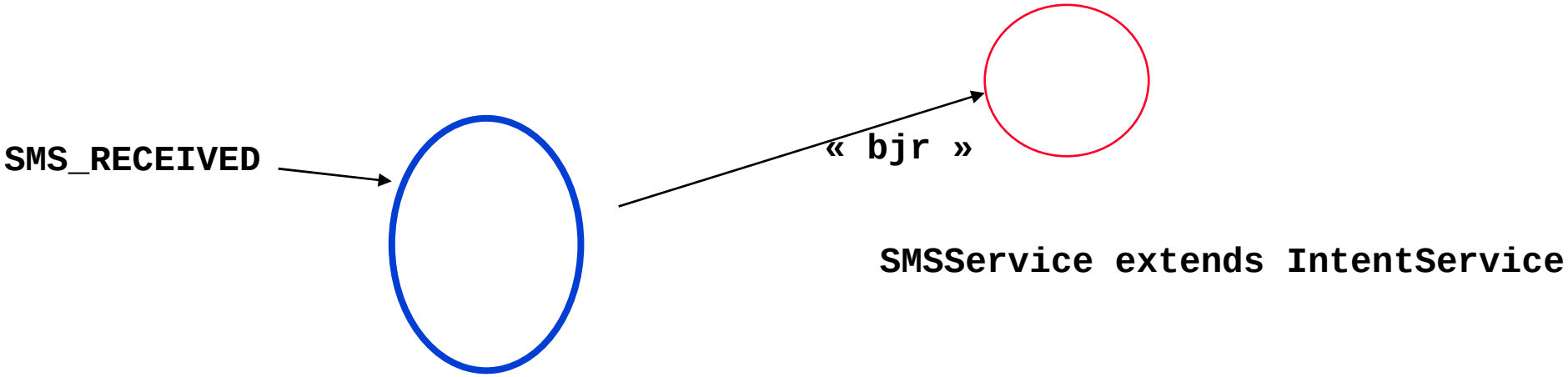
IntentService : un constat

- **Un service est créé en interne**
  - **Démarre et s'arrête aussitôt**
    - **Peu adapté à notre exemple de compteur qui s'incrémente toutes les secondes**

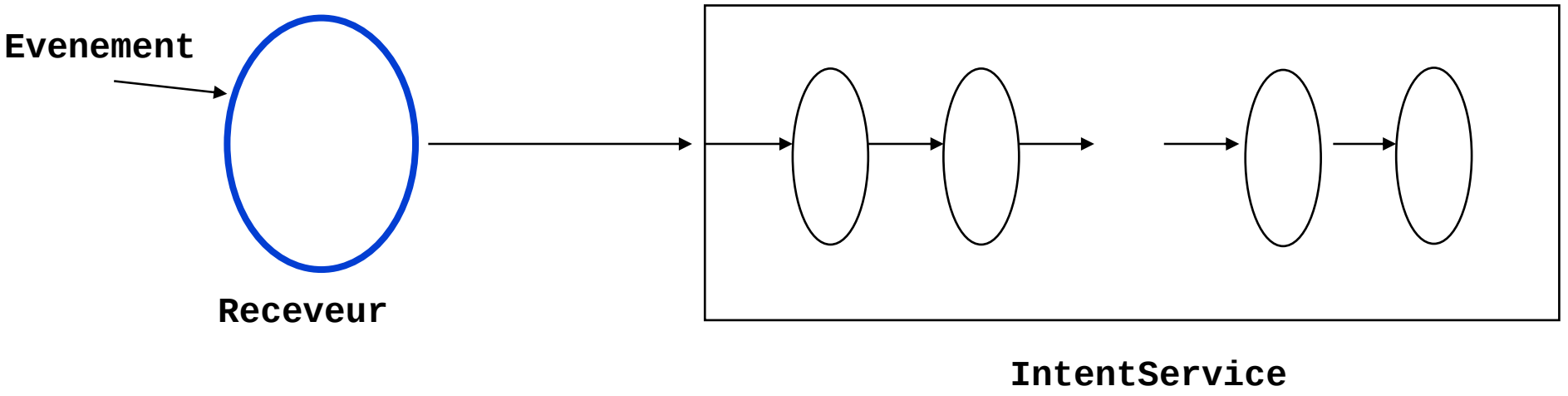
- **Pratique Standard**
  - **IntentService est un service créé à la suite d'un d'évènement**
    - **Par définition éphémère**

IntentService

- **A la suite d'un évènement**
- **En pratique**
  - **IntentService et BroadcastReceiver sont associées**
  - **A la suite d'un évènement, le souscripteur déclenche un intentService**



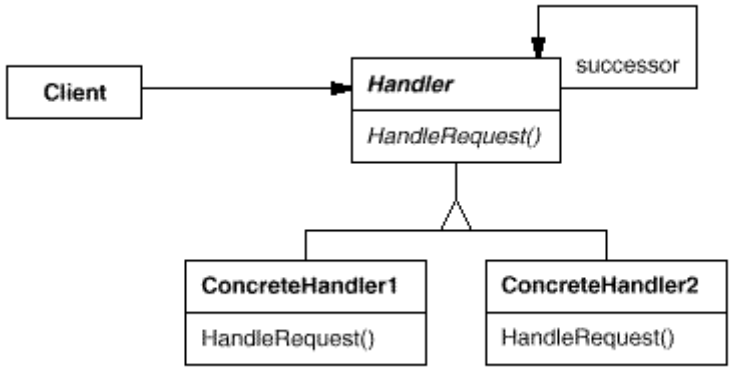
Architecture pour discussion

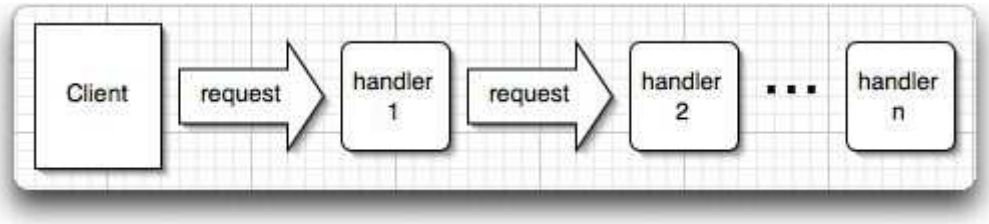


Acquisition **Traitement**

- **Acquisition : un receveur**
- **Traitement : un *Intent*service qui s'adresse à une chaîne de responsabilités**

Le patron chaîne de responsabilités





- https://en.wikipedia.org/wiki/Chain-of-responsibility_pattern
- https://dzone.com/articles/design-patterns-uncovered-chain-of-responsibility

Sommaire suite

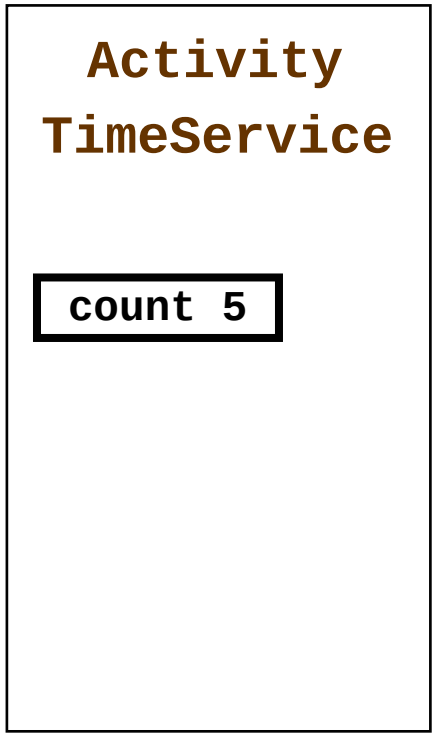
- **Même DVM : les limites**
- **Nécessité d'une communication sans mémoire partagée**
  - **Service et processus**
- **Service global**
  - **Par définition sur un autre processus**
  - **Nécessité d'un langage commun aux clients du service: AIDL**

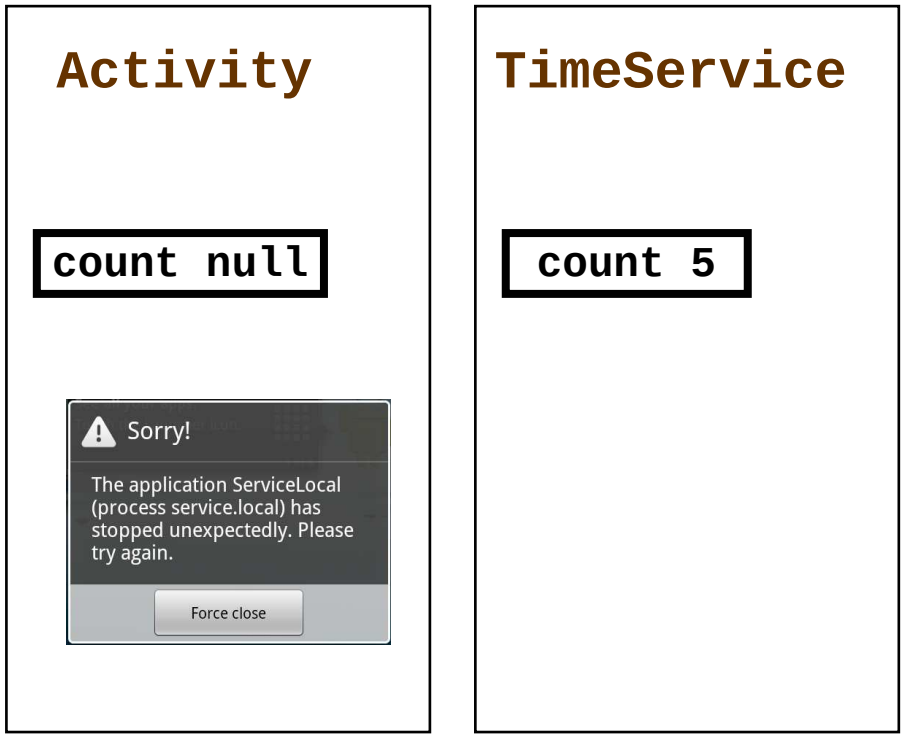
Discussion sur la même DVM

- **Ici le service partage le même processus que l'activité**
- **Attention**
- **Si nous choisissons, un processus dédié au service**
  - **<service android:name="*TimeService*"**
    android:process= "*:timeService*">
  - **</service>**

  - **Cela ne fonctionne plus…**
  - **explication … cf. schéma page suivante**

Schéma : processus et DVM





- **Un seul processus**
- **Activity + service**

Tout va bien en apparence

- **Deux processus, deux DVM**
- **Exception !!!!**

(Le code pour la lecture de count est le même, nullPointerException est appelée)

- **L'accès par des variables de classes est donc peu satisfaisant,** ou bien contraint fortement l'architecture à une seule DVM

help -> recherche sur le web

- http://blog.developpez.com/android23/p8571/android/creation_de_service
- **Utilisation de listener,**
  - idem …
- **Utilisation de la méthode IBinder (bindService)**
  - Même constat, ne fonctionne pas sur deux processus …
- **En conséquence pour un service global un IPC est nécessaire**
  - IPC comme Inter Process Communication
  - Plusieurs façons de faire
    - Messenger + Handler
    - BroadcastReceiver
    - AIDL

Communication entre processus

- **Solutions présentées**
  - **Messenger + Handler**
    - **Messages inter processus**
    - **Autorise une communication asynchrone**
    - **Un message est envoyé au client, celui-ci est prévenu à l'aide d'un Handler**
  - **Un receveur ou plusieurs receveurs**
    - **Un ou des receveurs et le bon filtre sont enregistrés**
    - **Le service envoie un « intent » et son filtre,**
      - **A destination des receveurs agréés, filtrés via le middleware Android**
  - **Le service devient global**
    - **Nécessité d'utiliser AIDL**
      - **in, out, inout, et certains types pris en compte,**
      - **implements Parcelable pour de la sérialisation d'instances**

Messenger, le messager s'installe

- http://developer.android.com/reference/android/app/Service.html

- **Messenger :**
  - **Une classe toute prête pour la communication entre processus**
  - **Contient un Handler installé dans le contexte de l'appelant**

1) Installation du messager:

- **Une instance de Messenger est transmise au service depuis le client,**
  - **via une intent (paramètre de startService)**

- **Ce messager contient une instance de Handler, installée côté client**
  - **la méthode du handler handleMessage(Message msg) est en place**

Messenger, le messager arrive

2) Le service voit arriver un messager

- Transmis par l'activité via une intention.

- **Le service demande et obtient un message**
  - Méthode de classe ***Message.obtain,***
  - Un ***bundle*** contient les données à transmettre,
  - Ce ***bundle*** correspond aux données du message
    - ***message.setData(bundle)***

Messenger, le messager repart

3) Le messager repart avec son message,

- **méthode *send(message)***

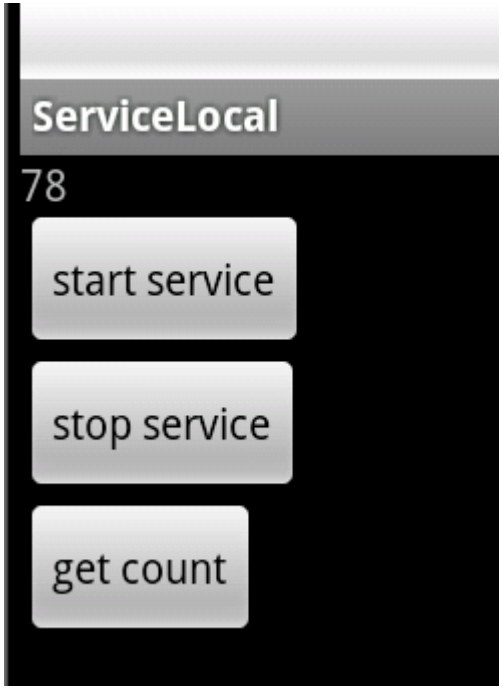
- **Le client est « soudain » *notifié* du retour du messager**
  - **La méthode *handleMessage* est déclenchée,**
  - **Le message transmis peut être lu,**
    - **bundle = message.getData()**

Le messager côté service

```
public int onStartCommand(Intent intent,int flags,int startId){
// Le message arrive via une intent
    Bundle extras = intent.getExtras();
    messager = (Messenger) extras.get("messager");

// Le service demande et obtient un message
    Message msg = Message.obtain();

// Un Bundle contient les données à transmettre,
    Bundle bundle = new Bundle();
    bundle.putLong("count", count);
    msg.setData(bundle);

// Le messager repart avec son message, méthode send.
    messager.send(msg);
```

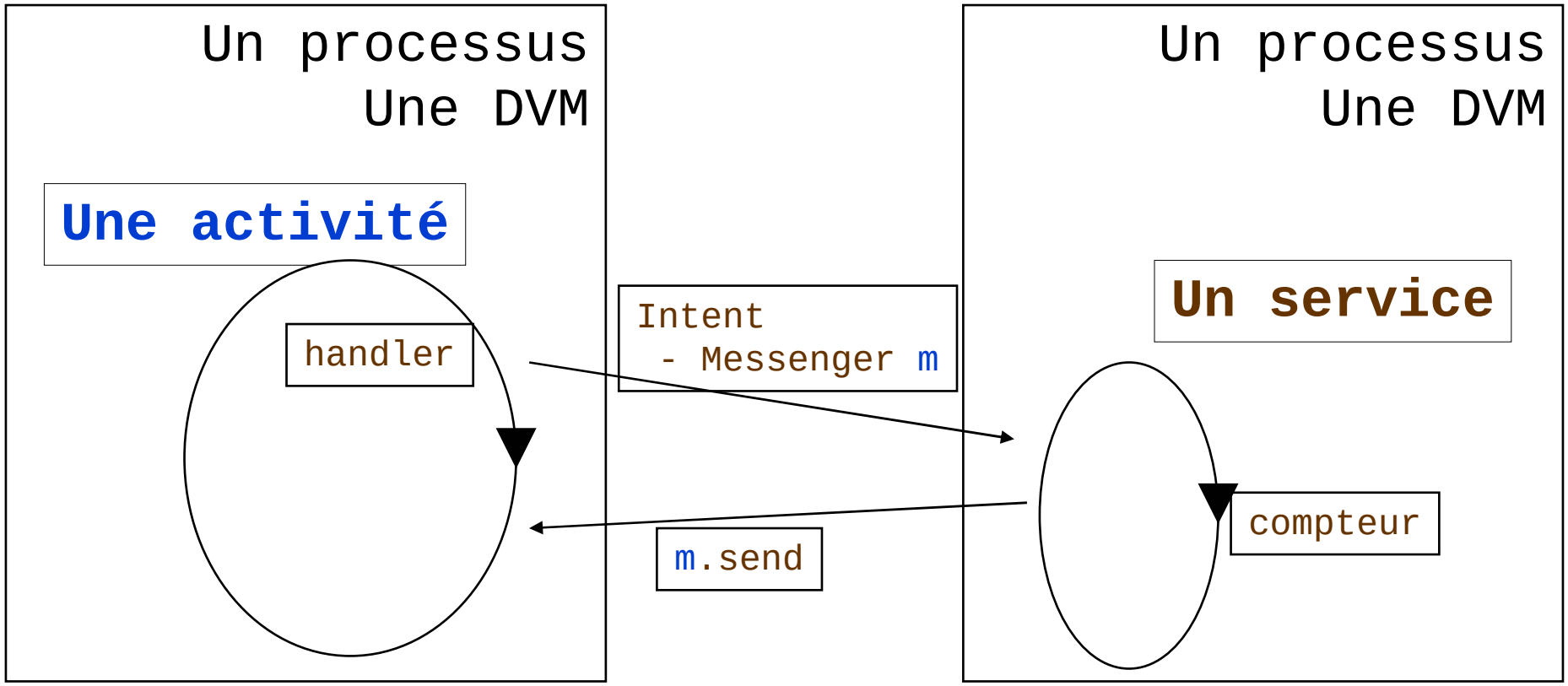
Le messager côté client et son handler

```
private Handler handler = new Handler() {

  public void handleMessage(Message message) {
    Bundle extras = message.getData();

    if (extras != null) {
      count = extras.getLong("count");
    } else {
      Toast(…).show();
    }
  };
};
```



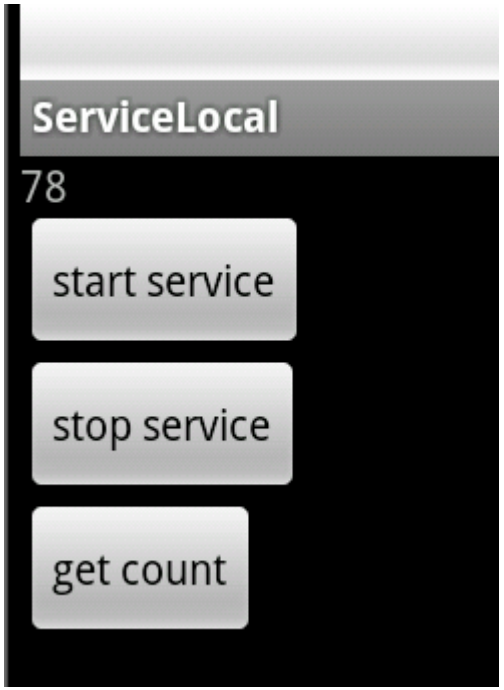

```
  public void onClickStart(View v){
    Intent intent = new Intent(this, TimeService.class);
    intent.putExtra("pid", android.os.Process.myPid());

    Messenger messager = new Messenger(handler);
    intent.putExtra("messager", messager);
    startService(intent);
    }
```

Architecture à deux DVM



- **Une application, un service, une activité, deux DVM**
  - **Échange de données**
    - **Envoi depuis l'activité par une intention**
    - **Réception des résultats**
      - **Messenger et Handler**

Démonstration



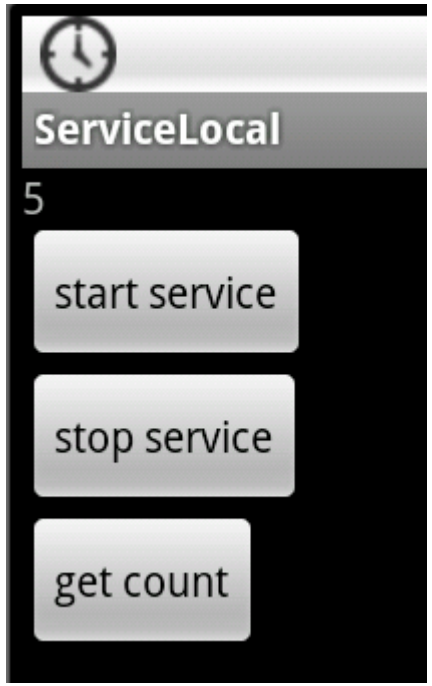
- **Communication entre deux processus, DVM possible**

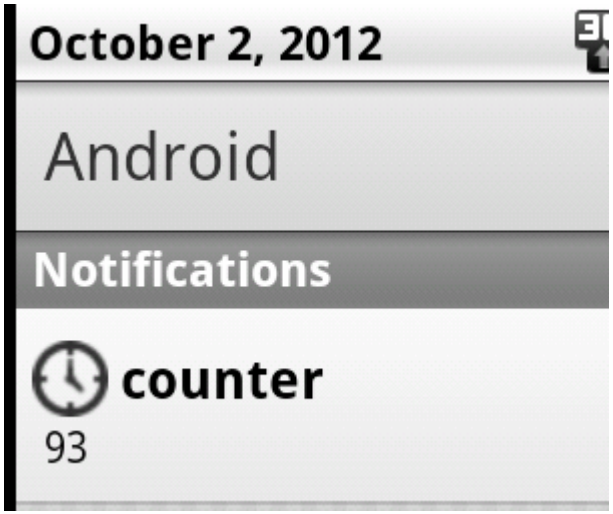
Notification effectuée par un service

- **Extra :**
  - **ajoutons maintenant une notification dans la barre de menus**

- **Notification est un service du système Android**
  - **Cf; NotificationManager**

Notification et service





- **Notification ? : un service « système »**

nm = (NotificationManager)getSystemService(Context.*NOTIFICATION_SERVICE*);

Notification, le code « standard »

- **Le service se charge de la notification**
  - **À chaque incrémentation, une notification**

```
// http://saigeethamn.blogspot.fr/2009/09/android-developer-tutorial-for_22.html
   private void showNotification(CharSequence title,
                                 CharSequence text){

   Notification notification =
     new Notification(R.drawable.watch, "counter",System.currentTimeMillis());
   PendingIntent intent = // un pending intent
     PendingIntent.getActivity(getApplicationContext(),0, null, 0);

   notification.setLatestEventInfo(this, title, text, intent);

   NotificationManager nm =
      (NotificationManager)getSystemService(Context.NOTIFICATION_SERVICE);

   nm.notify(R.string.started, notification);
}
```

  - *PendingIntent ? En annexe*

Notification à lire

- http://www.vogella.com/tutorials/AndroidNotifications/article.html

Résumé et la suite

- **Un service sur un processus**
- **Nécessité d'un « canal » de communication**
  - **Messenger: une classe toute prête et dédiée**
  - **Ou bien: utiliser les intent et les receveurs**
  - **Ou faire appel à l'IPC**
    - **Inter-Process Communication**

Un service et son (ses) receveur(s)

- **Un « Broadcast receiver » est à l'écoute des « Intents »**
- **Le système délivre les « Intents » à tous les receveurs candidats, lesquels s'exécutent séquentiellement**
  - **Les receveurs candidats avec le même filtre**
- **L'enregistrement du receveur se fait**
  - **par programme**
    - **getApplicationContext().registerReceiver(receiver, filter)**
  - **Ou bien avec la balise**
    - **<receiver> du fichier AndroidManifest.xml**
      - **<intent-filter>**
- **Plusieurs receveurs sont possibles**
- **Deux types d'envoi par Android**

Traitement de l'intent par le receveur

- La méthode onReceive est exécutée, l'intent est transmis

void onReceive(Context ctxt, Intent msg);

Abonnement auprès d'Android
via AndroidManifest.xml

```
<receiver
  android:name="service.receiver.TimeReceiver">
    <intent-filter>
       <action android:name="time.action.TIC" />
    </intent-filter>
  </receiver>
```

Envoi de l'intent par l'émetteur

- L'envoi de l'intent est effectué par Android (asynchrone) à tous les receveurs candidats selon un ordre indéfini :

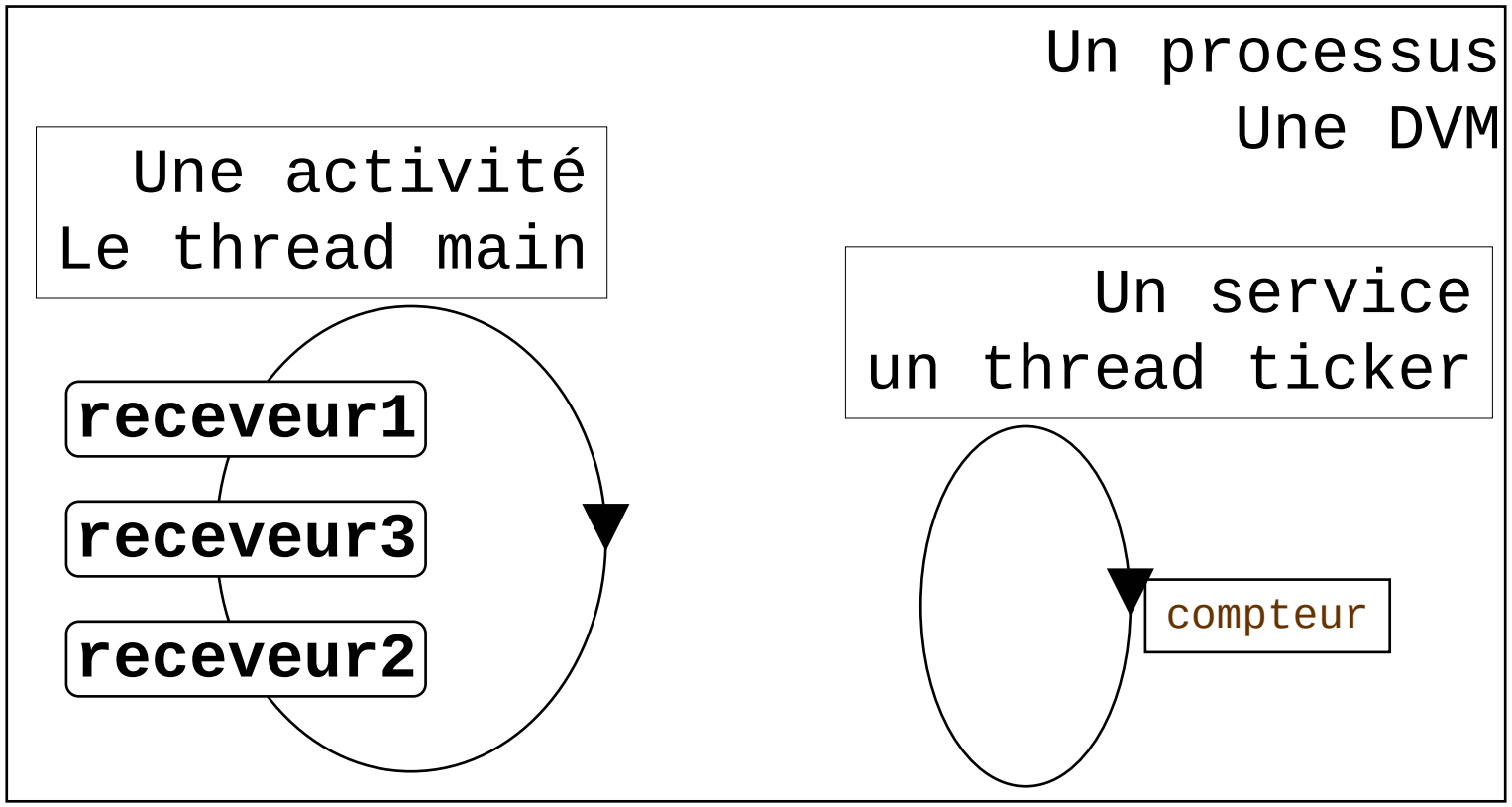
Context.sendBroadcast

- L'envoi de l'intent est délivré selon une relation d'ordre définie entre les receveurs, (*android:priority attribut du filtre*) un receveur peut interrompre la propagation (*abortBroadcast*) :

Context.sendOrderedBroadcast

cf. le Pattern Chaîne de responsabilités dans lequel la chaîne est une liste ordonnée, mise en œuvre uniquement par l'appel de sendOrderedBroadcast

Un exemple



- De l'activité,
  - Démarrage du service par startService
  - accès au compteur de tics par une méthode de classe (même DVM)
- *Activité : classe ServiceLocalActivity*
- *Service : classe TimeService*

Exemple simplifié

- **Une activité, un service, un receveur**
- **Le fichier AndroidManifest.xml**

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="service.local" android:versionCode="1" android:versionName="1.0">
  <uses-sdk android:minSdkVersion="8" />
- <application android:icon="@drawable/icon" android:label="@string/app_name">
- <activity android:name="service.receiver.ServiceReceiverActivity"

  <service android:name="service.receiver.TimeService" />

  <receiver android:name="service.receiver.TimeReceiver">
    <intent-filter>
       <action android:name="time.action.TIC" />
    </intent-filter>
  </receiver>
  </application>
  </manifest>
```

Exemple suite, l'activité suivie du receveur

- **Au sein de l'activité**

  - **Un démarrage du service (habituel)**

```
public void onClickStart(View v){
     Intent intent = new Intent(this, TimeService.class);
     started = startService(intent) != null;
}
```

  - **Une association du filtre et du receveur dans le programme**
    - **Une autre écriture à la place de la balise <receiver**

```
IntentFilter filter= new IntentFilter("time.action.TIC");
// filter.setPriority(X); X >= 0 pour un envoi ordonné
receiver  = new TimeReceiver();
getApplicationContext().registerReceiver(receiver, filter);
```

Exemple suite : le receveur

```
public class TimeReceiver extends BroadcastReceiver {
    private String name;
    public TimeReceiver(final String name){
      this.name = name;
    }
    public TimeReceiver(){
      this("anonymous");
    }

  @Override
  public void onReceive(Context ctxt, Intent intent) {
     final Long count = intent.getLongExtra("count", -1L);
     Log.i("TimeReceiver_" + name,"count: " + count);
  }

}
```

Exemple suite : Le service

- **Le service demandé est effectif, le résultat est prêt**
  - Le résultat est envoyé en ordre dispersé ou non

1. Création de l'intent avec le bon filtre pour les receveurs
2. Le résultat du service est placé dans l'intent
3. Propagation à tous les receveurs candidats

```
public void run(){
   while(!ticker.isInterrupted()){
     try{
      Thread.sleep(1000);
      count.set(count.longValue()+1L);

1.          Intent response= new Intent("time.action.TIC");

2.          response.putExtra("count", count.get());
3.          sendBroadcast(response);
```

  - Ou bien `sendOrderedBroadcast(response, null);`

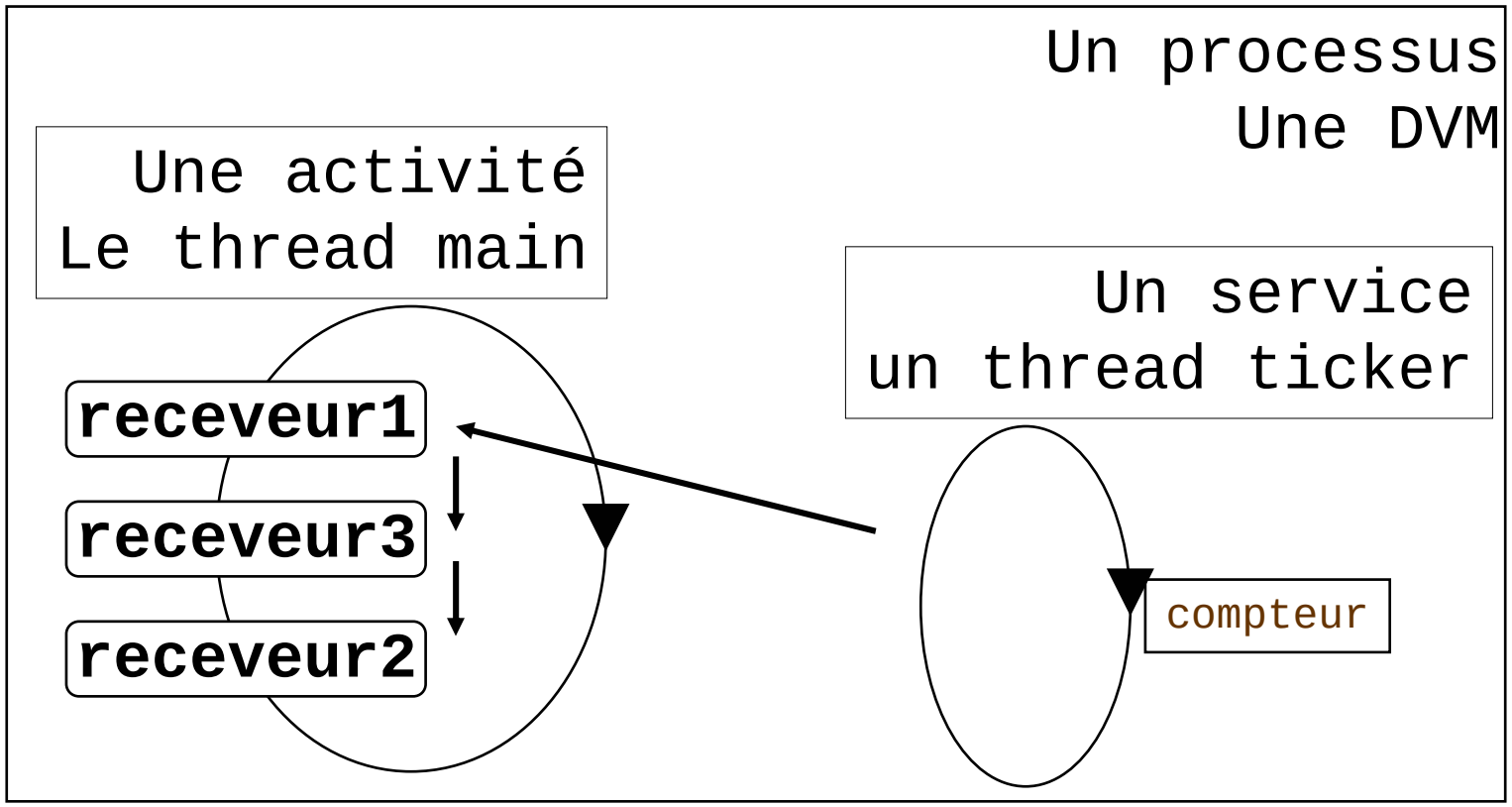
```
     }catch(Exception e){}
   }
}
```

Exemple suite et fin

- **Fin du service**
  - **stopService**
    - **unregisterReceiver**

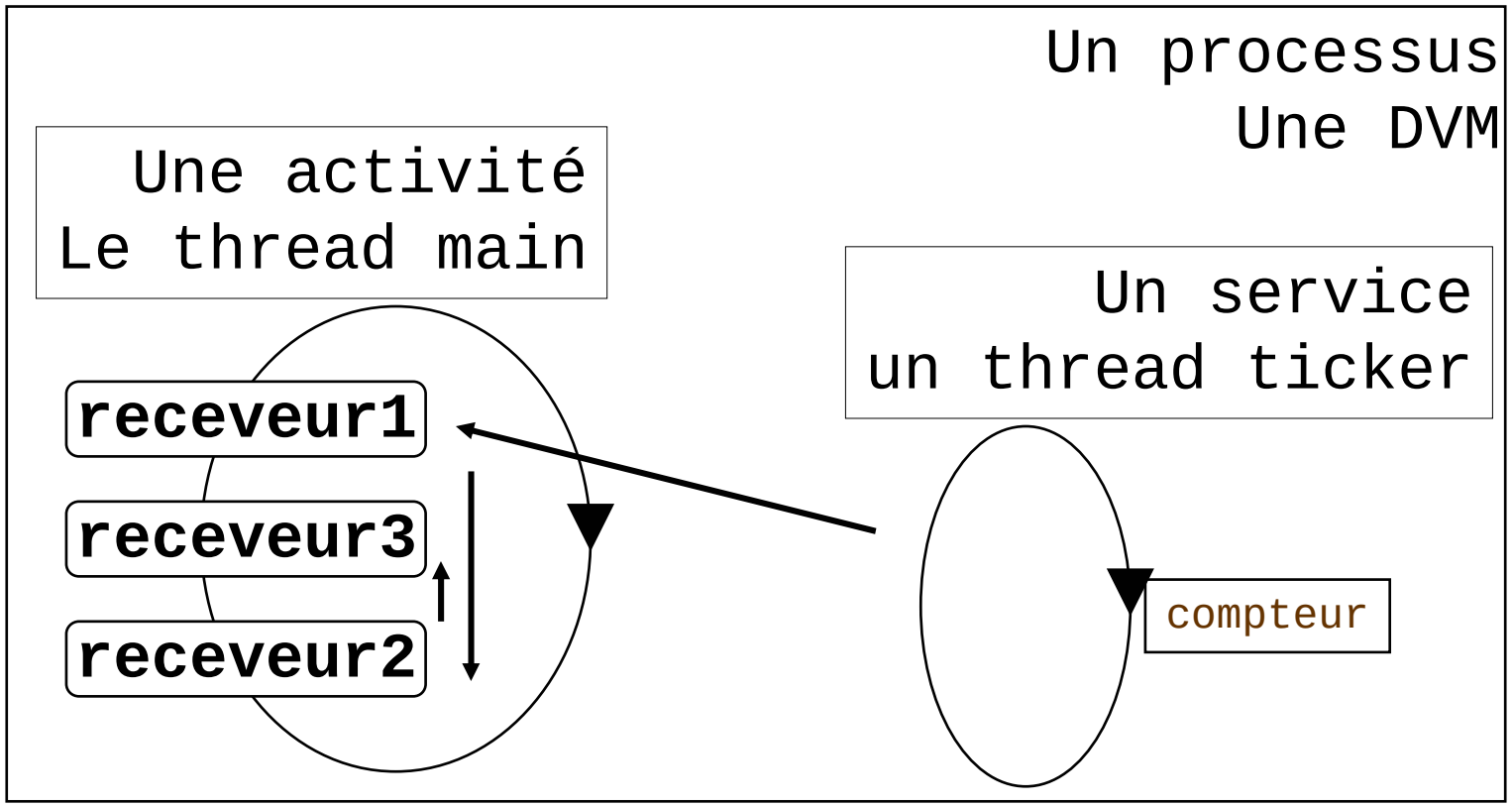
```
public void onDestroy(){
     super.onDestroy();
     stopService(new Intent(this, TimeService.class));
      getApplicationContext().unregisterReceiver(receiver);
   }
```

Un exemple, sendBroadcast



- De l'activité,
  - Démarrage du service par startService
  - Envoi du nombre de tics
- *Activité : classe ServiceLocalActivity*
- *Service : classe TimeService*

Un exemple, sendOrderedBroadcast



- De l'activité,
  - Démarrage du service par startService
  - Envoi du nombre de tics
- *Activité : classe ServiceLocalActivity*
- *Service : classe TimeService*

Démonstration

- **Une activité, un service, un receveur**
  - **Tout fonctionne …**

La démonstration évolue

1. **Le service est placé sur un autre processus**

   `<service android:name="service.receiver.TimeService" android:process=":timeService" />`
2. **Le receveur est également placé sur un autre processus**

   `<receiver android:name="service.receiver.TimeReceiver" android:process=":timeReceiver" />`

   Si usage de la balise <receiver

- **Ajout de nouveaux receveurs**
- **La stratégie d'envoi aux receveurs est modifiée**

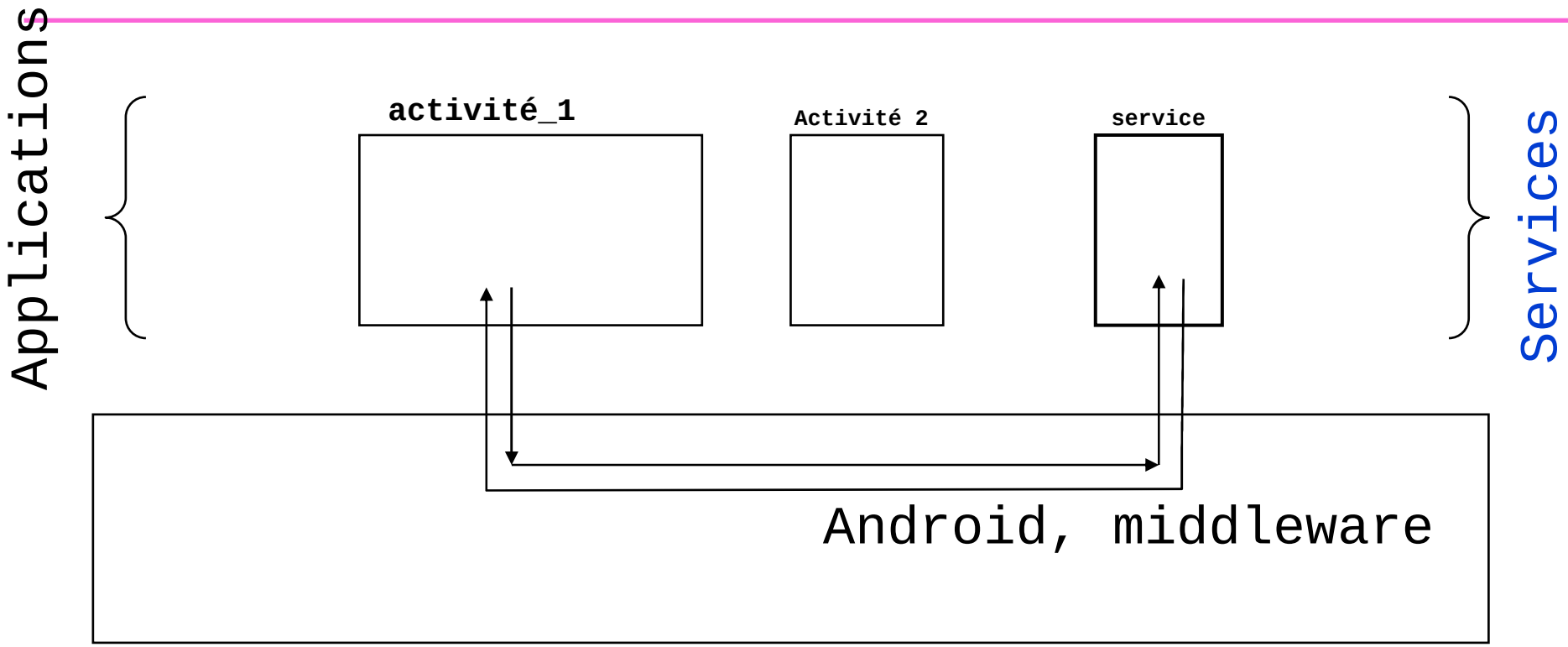
Résumé, pause intermédiaire

- **Nous avons vu,**
  - **Service local, sur la même DVM**
  - **Service local**
  - **Messenger et Handler**
  - **SendBroadcast, sendOrderedBroadcast aux receveurs**
- **Passons maintenant aux services globaux**
  - **AIDL et plus**

Service global : création, démarrage et arrêt

- **En tache de fond**
- **Services globaux**
  - **« Distants »**
    - **Rmi en plus simple … même machine …**
    - **Accessible à tous**
      - **Attribut android:exported="true", balise service**

Un Service à disposition de tous



Service distant,

- Découverte du service par une intention …
- Sérialisation/désérialisation lors du passage de paramètre
  - Implicite si types primitifs, List et Map,
  - Explicite par l'implémentation de Parcelable

Un simple service global

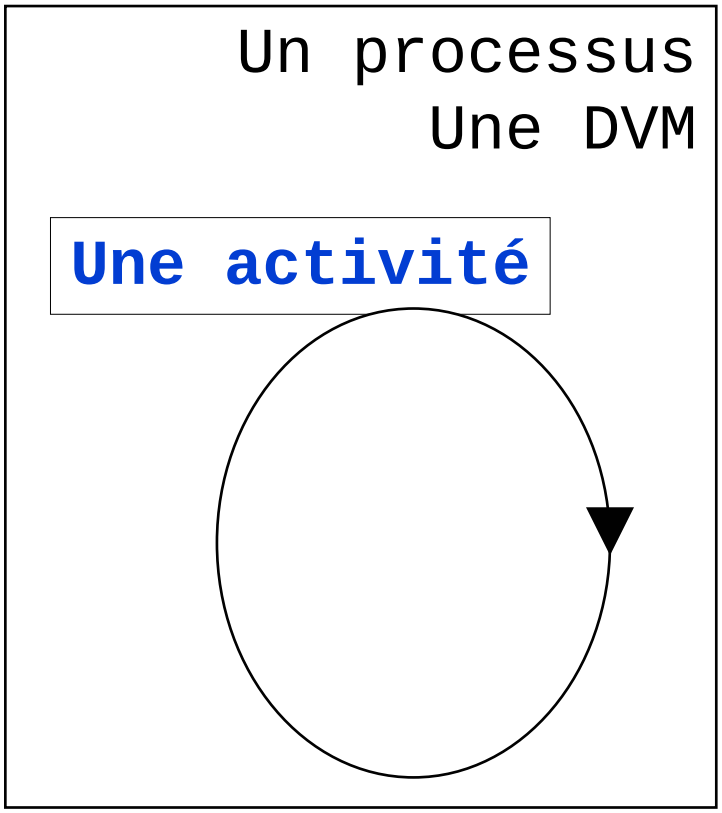
- **Un service accessible par toute application**
  - **Avec quelques contraintes**

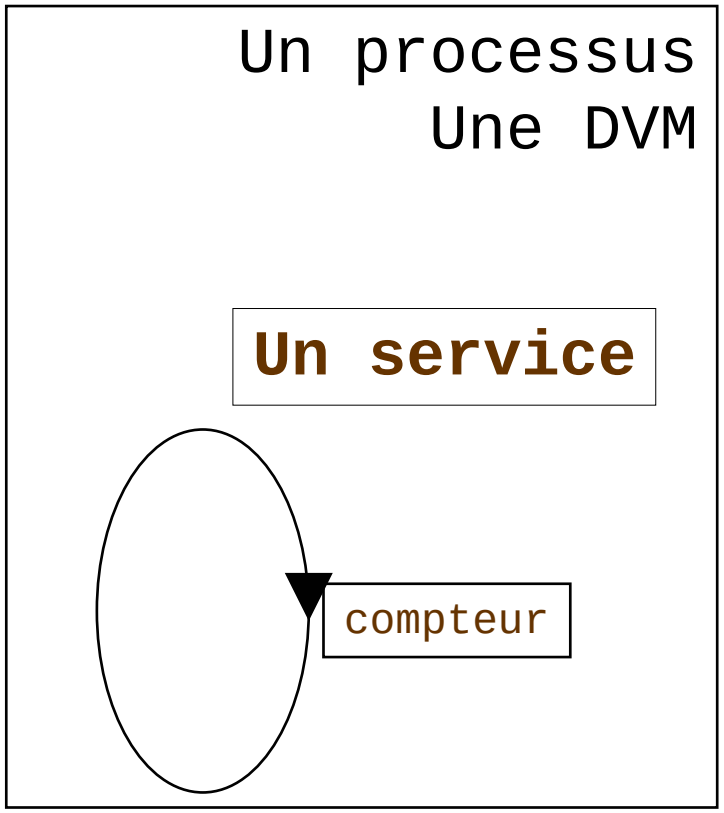
- **Un langage commun AIDL**
  - **AIDL, Android Interface Description Language**
  - **Types primitifs de java**
  - **String, CharSequence, List** (ArrayList coté serveur), **Map** (HashMap)
    - **Indication du type de passage de paramètre in, out, inout**
    - **Sérialisation/désérialisation**
      - **Pas de type générique**
  - **Autres classes, elles devront : implements Parcelable**
    - **Analogue à Serializable, mais rien n'est implicite**

Le service devient accessible de partout …

- **Le service devient global**
  - **Attribut android:exported du service**
- **Nécessité d'un langage commun**
  - **Un fichier décrit l'interface en AIDL**
    - **(Android Interface Description Language)**
- **Le service et les clients**
  - **Partagent cette interface**

L'exemple initial revisité

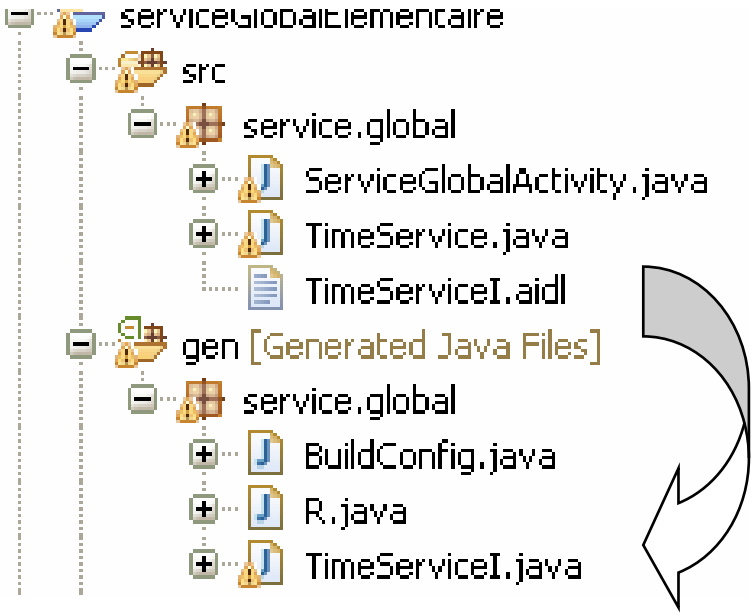




- **Le fichier AIDL:**

```
package service.global;
interface TimeServiceI{

  void setCounter(in long value);
  long getCounter();

}
```

Génération automatique du mandataire



- **À partir du fichier AIDL**
  TimeServiceI.aidl
- **Les sources du mandataire sont générées automatiquement…**
  TimeServiceI.java

Il ne nous reste plus qu'à implémenter le mandataire dont l'interface java est issue de TimeServiceI.java elle-même engendrée via TimeServiceI.aidl

Le mandataire

- **Au sein de TimeService**

```
@Override
public IBinder onBind(Intent intent){
  return new TimeServiceStub();
}

private static class TimeServiceStub extends TimeServiceI.Stub{

  public long getCounter()  throws RemoteException{
    return count.get();
  }

  @Override
  public void setCounter(long value) throws RemoteException {
    count.set(value);
  }
}
```

Note: TimeServiceI.Stub le mandataire a été généré automatiquement

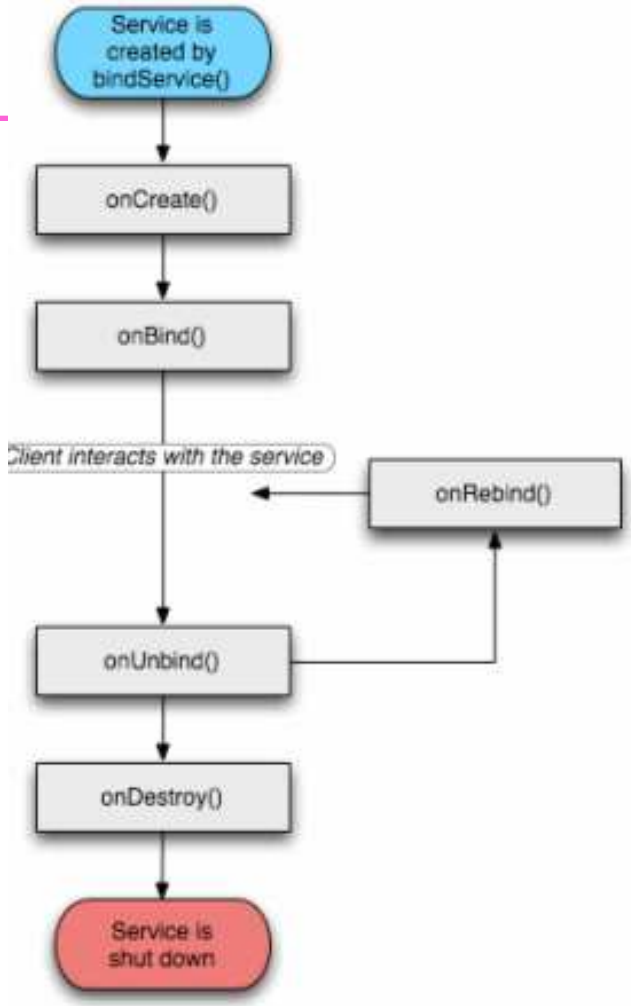
Le client attend un *IBinder* et se connecte

```
private TimeServiceI service;

private ServiceConnection connection = new ServiceConnection(){

public void onServiceConnected(ComponentName name, IBinder binder) {

 service = TimeServiceI.Stub.asInterface(binder);

   …
}

public void onServiceDisconnected(ComponentName name) {

    …
}
};
```

Le client se lie

bindService

- Android démarre le service
  - onCreate, onBind

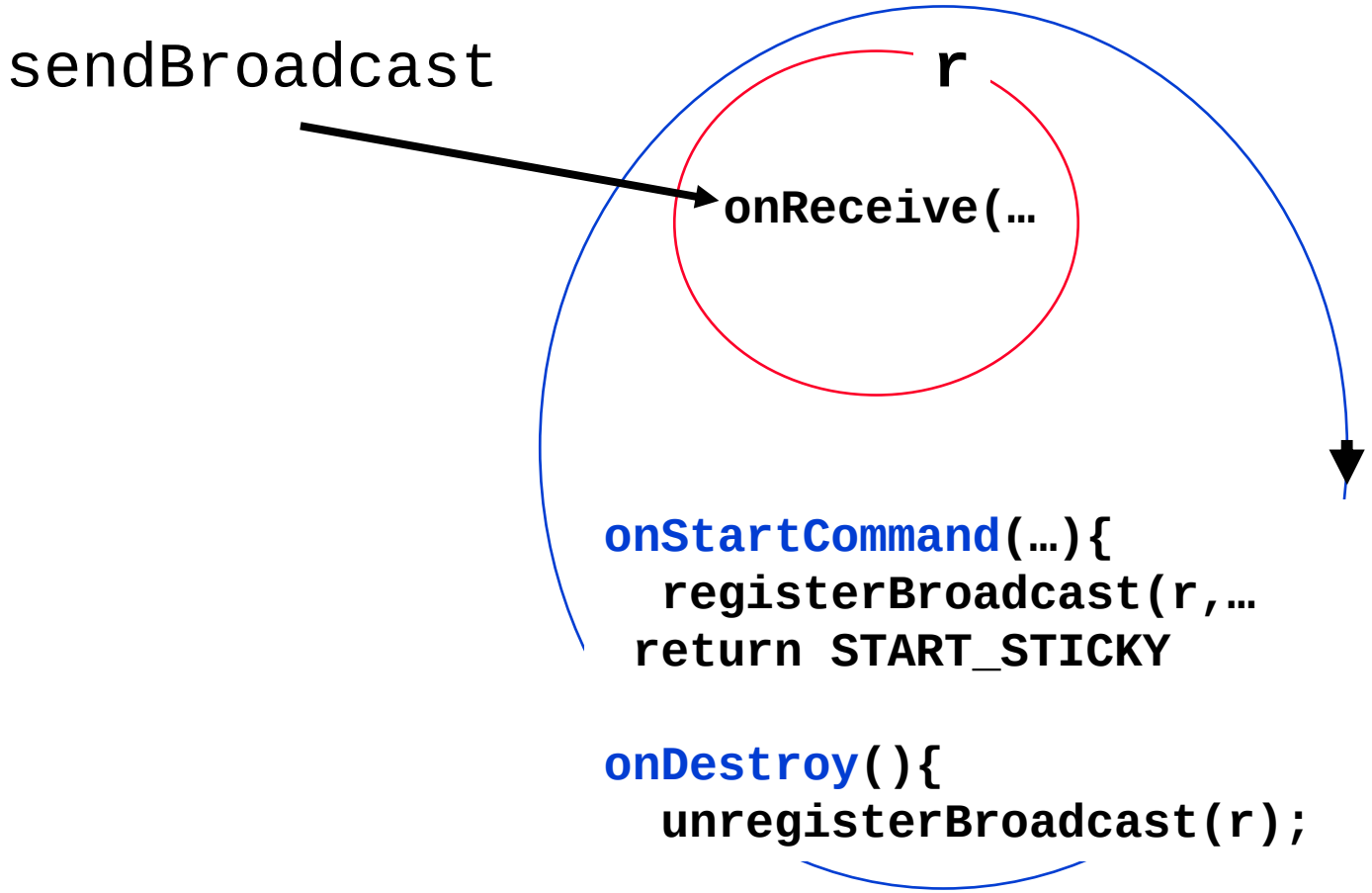


- bindService(intent, connection, Context.*BIND_AUTO_CREATE*);

Service et Receiver

- **Réception d'évènements globaux ?**
- **Un service global en tache de fond**
  - **Installe un receveur, à la création du service**
  - **Le service sur un processus linux séparé**

Un service + un receveur




```
onStartCommand(…){
  registerBroadcast(r,…
 return START_STICKY

onDestroy(){
  unregisterBroadcast(r);
```

```java
public class BroadcastService extends Service {

    private static class ReceiverSMB116 extends BroadcastReceiver{
        @Override
        public void onReceive(Context context, Intent intent) {
            Log.i("ReceiverSMB116", "onReceive: " + intent.getAction());
            // analyse de l'intent
        }
    }
    private ReceiverSMB116 receiver;
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        IntentFilter filter = new IntentFilter();
        filter.addAction("SMB116_ACTION_SERVICE");
        this.receiver = new ReceiverSMB116();
        getApplicationContext().registerReceiver(receiver,filter);
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        getApplicationContext().unregisterReceiver(receiver);
        Log.i("ReceiverSMB116", "onDestroy");
        killProcess( myPid() );
    }
```

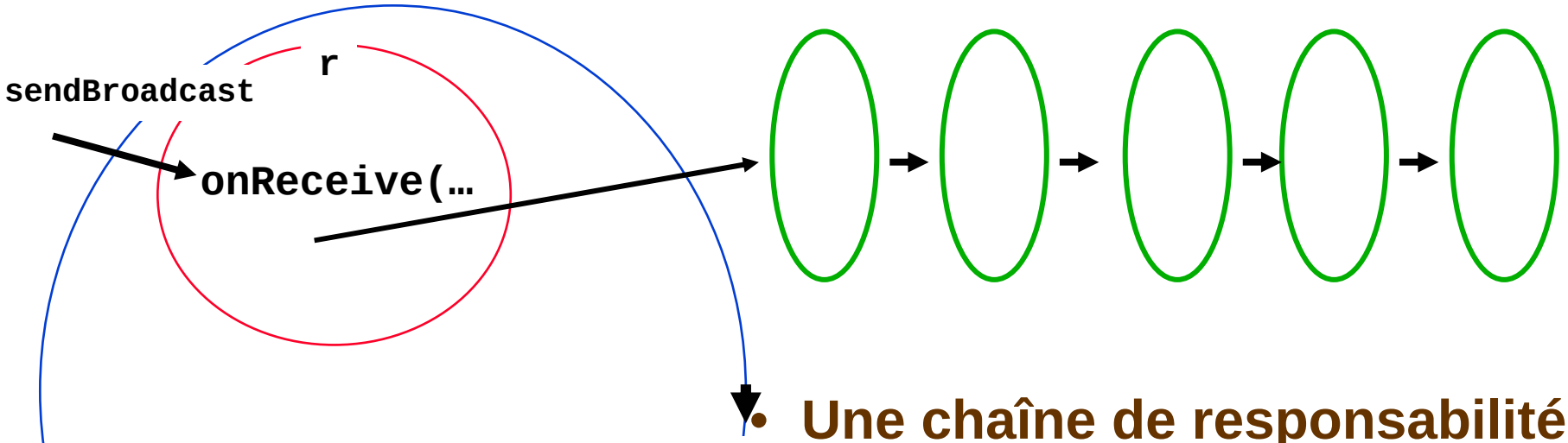
Depuis une autre application

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

        public void onClickSendBroadcast(View v){
            Intent intent = new Intent();
            intent.setAction("SMB116_ACTION_SERVICE");
            Log.i("MainActivity", "onClickSendBroadcast");
            getApplicationContext().sendBroadcast(intent);
        }
```

Le service déclaration

```
<application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
    ...

    <service
            android:name=".BroadcastService"
            android:enabled="true"
            android:exported="true"
            android:process=":BroadcastService">
        </service>

</application>
```

Réception et envoi vers une chaîne de responsabilités




```
onStartCommand(…){
  registerBroadcast(r,…
  messager = …
 return START_STICKY

onDestroy(){
  unregisterBroadcast(r);
```

- **Une chaîne de responsabilités**
  - Pour le traitement des données reçues

Un messager se charge de déclencher la chaîne de responsabilités

Le service, le messager est de retour

```
public class BroadcastService extends Service {

    private ReceiverSMB116 receiver;
    private Messenger messager;
    private class ReceiverSMB116 extends BroadcastReceiver{
        @Override
        public void onReceive(Context context, Intent intent) {
            Message msg = Message.obtain();
            Bundle bundle = new Bundle();
            bundle.putString("message",intent.getStringExtra("message"));
            msg.setData(bundle);
            try{
                messager.send(msg);
            }catch(Exception e){
            }
        }
    }

    public int onStartCommand(Intent intent, int flags, int startId) {
        IntentFilter filter = new IntentFilter();
        filter.addAction("SMB116_ACTION_SERVICE");
        this.receiver = new ReceiverSMB116();
        getApplicationContext().registerReceiver(receiver,filter);
        Bundle extras = intent.getExtras();
        messager = (Messenger) extras.get("messager");
        return START_STICKY; // à voir START_REDELIVER_INTENT;
    }
```

UIThread et son Handler

```
public class MainActivity extends AppCompatActivity {
    private ChainHandler<String> chaine;

    private Handler handler = new Handler() {
        public void handleMessage(Message message) {
            Bundle extras = message.getData();
            if (extras != null) {
                chaine.handleRequest(extras.getString("message"));
            }
        };
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        this.chaine = new ToastChainHandler(null);
    }

    public void onClickStartService(View v){
        Intent intent = new Intent(this,BroadcastService.class);
        Messenger messager = new Messenger(handler);
        intent.putExtra("messager", messager);
        startService(intent);
    }
```

Chaîne de responsabilités, un maillon

```
public class ToastChainHandler extends ChainHandler<String> {
    private final static boolean I = true;

    public ToastChainHandler(ChainHandler<String> successor){
        super(successor);
    }

    public boolean handleRequest(String value){
        Calendar c = Calendar.getInstance();
        DateFormat df = DateFormat.getDateInstance(DateFormat.SHORT, Locale.FRANCE);
        DateFormat dt = DateFormat.getTimeInstance(DateFormat.SHORT,Locale.FRANCE);
        String date = df.format(c.getTime()) + "-" + dt.format(c.getTime());
        //Toast.makeText()
        if(I) Log.i("ToastChainHandler","handleRequest: " + value );
        return super.handleRequest(value);
    }
}
```

- **ToastHandler**

Notification …

```
public void onClickSendBroadcast(View v){
    Intent intent = new Intent();
    intent.setAction("SMB116_ACTION_SERVICE");
    intent.putExtra("message","un message...");
    getApplicationContext().sendBroadcast(intent);
}
```

Autre exemple : un receveur de SMS

- **Un service global de réception/filtrage de SMS**
  - Éventuellement avec accès publics via AIDL
  - Installe un receveur/souscripteur de SMS
  - A chaque SMS entrant le souscripteur est notifié
    - Le contenu du SMS est analysé
      - https://goo.gl/Vwa8d9
    - Et en fonction du contenu il peut ne pas être délivré,
      - setPriority(Integer.MAX_VALUE);
      - abortBroadcast();
  - Application diverses et variées :
    - Contrôle/commandes distantes
    - Par exemple ce traceur :
      - http://medias.pearl.fr/technique/notice/PX3490.pdf

Service et Receiver de SMS interne: un schéma

```
public class SMSService extends Service {
    private static boolean serviceRunning;
    private SMSReceiver receiver;
     private class SMSReceiver extends BroadcastReceiver{
         public void onReceive(Context context, Intent intent) {…}
     }

    public SMSService() {}

    public void onCreate() {
        if(!serviceRunning){this.serviceRunning = true;
            this.receiver = new SMSReceiver();
            IntentFilter filter = new IntentFilter();
            filter.addAction("android.provider.Telephony.SMS_RECEIVED");
            getApplicationContext().registerReceiver(receiver, filter);
        }
    }

    public void onDestroy() {
        if(serviceRunning){
            if(receiver!=null) {
                getApplicationContext().unregisterReceiver(receiver);
            }
            serviceRunning=false;
            receiver = null;
        }
    }

    public int onStartCommand(Intent intent, int flags, int startId) {
       return START_STICKY;
    }
```

Démonstration

Annexes

- **Un compteur de SMS entrants**
- **Installation d'une librairie**
  - **Liée à l'usage de Parcelable**
  - **Nécessaire aux clients d'un service global**
- **Liste des services**
- **PendingIntent**
- **Divers**

Exemple : compteur de SMS …

- **Approche incrémentale à l'aide d'un exemple**

1. **Un service global accessible par tous (Exemple 1)**
   Limité à 3 fonctionnalités élémentaires
   Démarrer, arrêter, combien ?

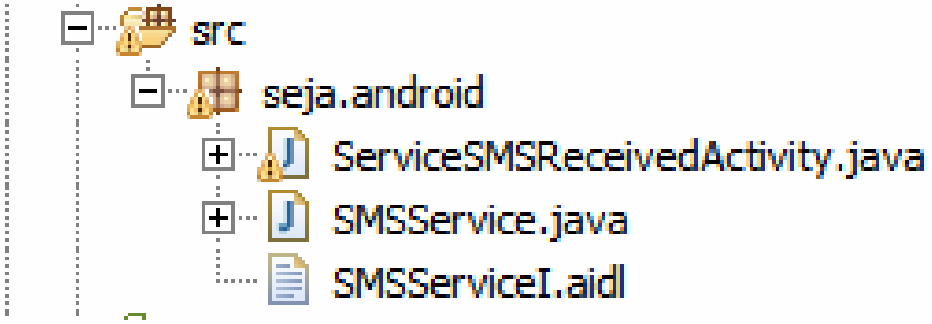
2. **Ajout effectif du receveur de messages (Exemple 2)**
   – **Mise en œuvre du receveur d'intention standard**
   Démarrer le receveur, arrêter la réception, combien de SMS reçus ?

3. **Ajout d'une nouvelle fonctionnalité (une idée de TP)**
   – **Le texte d'un sms contient une commande (secrète) qui déclenche l'action correspondante, (envoi de la position…)**

Exemple 1

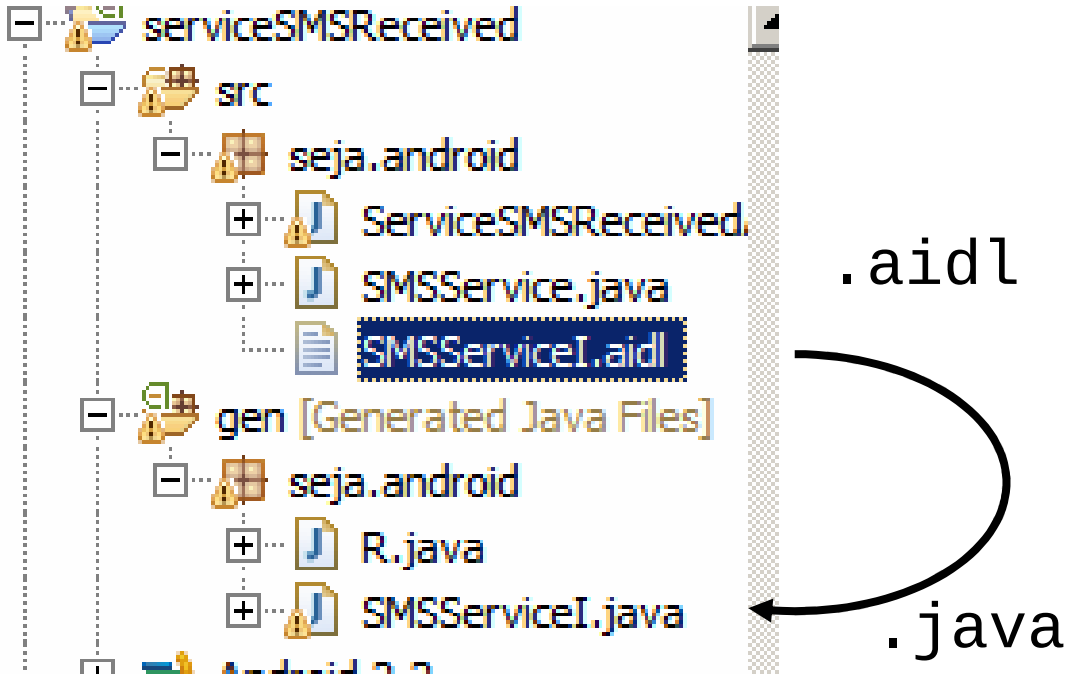
3 fichiers

1. **Une interface en AIDL des fonctionnalités distantes**
   - Sous ensemble de Java, par défaut types primitifs, List, Map
   - Parcelable sinon (cf en annexe)
2. **Une implémentation du service**
   - La souche, le mandataire à destination des clients distants, est générée
3. **Une activity de mise en oeuvre**
   - Recherche du service
   - Connexion avec celui-ci



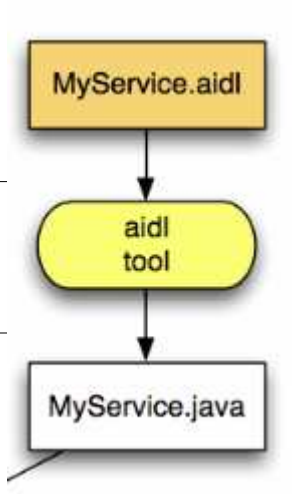
Exemple un préambule compteur de SMS

```
package seja.android;
interface SMSServiceI{
  void start();     // démarrage de la réception sms
  void stop();      // arrêt de celle-ci
  long received(); // combien de SMS reçus ?
}
```



Service, AIDL (android interface description language)

```
package cnam.android;

interface SMSServiceI{
  void start();
  void stop();
  long received();
}
```



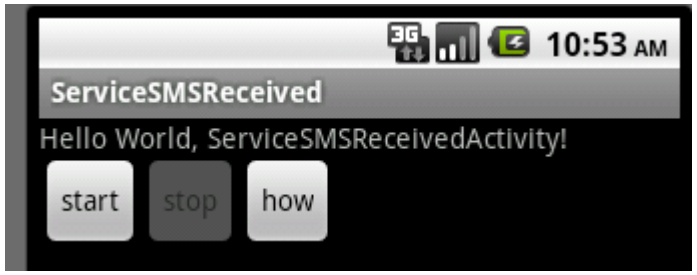

```
/*
This file is auto-generated.  DO NOT MODIFY.
*/
package cnam.android;
import …;
public interface SMSServiceI extends android.os.IInterface{


/** Local-side IPC implementation stub class. */
public static abstract class Stub extends android.os.Binder
implements cnam.android.SMSServiceI{ …
```

Implémentation du service, SMSService.java

```
public class SMSService extends Service{

  // appelée par les clients
  public IBinder onBind(Intent intent) {
    return new SMSServiceImplStub();
  }

  private static class SMSServiceImplStub extends SMSServiceI.Stub{
    private static long counter; // compteur de sms provisoire

    public void start() throws RemoteException {
      Log.i("SMSServiceImplStub","start");
    }

    public void stop() throws RemoteException {
      Log.i("SMSServiceImplStub","stop");
    }

    public long received() throws RemoteException {
      synchronized(SMSServiceImplStub.class){
        counter++;
      }
      return counter;
}}}
```

L'activité cliente du service

```
public class ServiceSMSReceivedActivity extends Activity {
private SMSServiceI service;
private ConnexionAuServiceSMS connexion;
```



-
```
public void onClickStart(View v){
 if(!connecte){
   // identification du service
   Intent serviceIntent = new Intent();
   serviceIntent.setClassName("seja.android", "seja.android.SMSService");


   // connexion avec le service
  this.connexion = new ConnexionAuServiceSMS();
  // connexion sera déclenchée par l'intergiciel (asynchrone)
  bindService(serviceIntent, connexion, Context.BIND_AUTO_CREATE);


   connecte = true;
   start.setEnabled(false);
   stop.setEnabled(true);
 }}
```

Connexion au service, par android, asynchrone

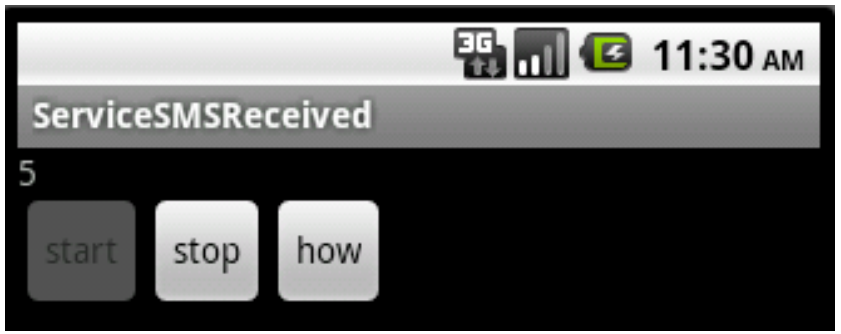
Cf. transparent précédent

```
    // connexion avec le service
    this.connexion = new ConnexionAuServiceSMS();
    // connexion sera déclenchée par l'intergiciel (asynchrone)
    bindService(serviceIntent, connexion, Context.BIND_AUTO_CREATE);
```

```
 private
  class ConnexionAuServiceSMS implements ServiceConnection{

  public void onServiceConnected(ComponentName name,
                                 IBinder binder) {
     // méthode appelée par Android,
     // service reçoit un mandataire, un client-proxy
     service = SMSServiceI.Stub.asInterface(binder);
  }

  public void onServiceDisconnected(ComponentName name) {
  }
}
```

L'activité cliente connectée

```
 public class ServiceSMSReceivedActivity extends Activity {
private SMSServiceI service;
private ConnexionAuServiceSMS connexion;
```




```
   public void onClickHow(View v){
    if(connecte){
      try {
       texte.setText(Long.toString( service.received()));
    } catch (RemoteException e) {
    }
    }
```

Appel de service.received(), Affichage de la valeur 5 …

Pause avant l'accès par tous

- **Rien de nouveau, sauf cette interface**

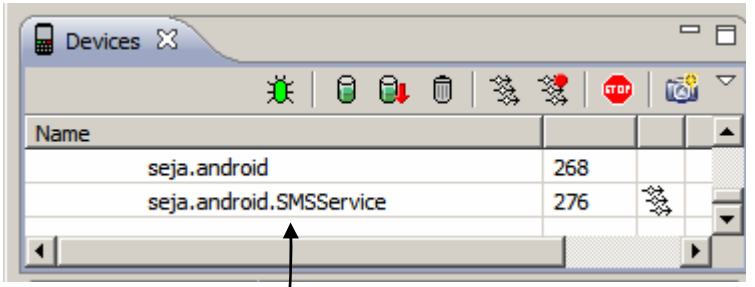
```
package cnam.android;

interface SMSServiceI{
  void start();
  void stop();
  long received();
}
```

  - **Architecture classique**
    - **Un service**
    - **Une liaison à l'aide d'une instance de la classe ServiceConnection**

Le manifest, installation du service

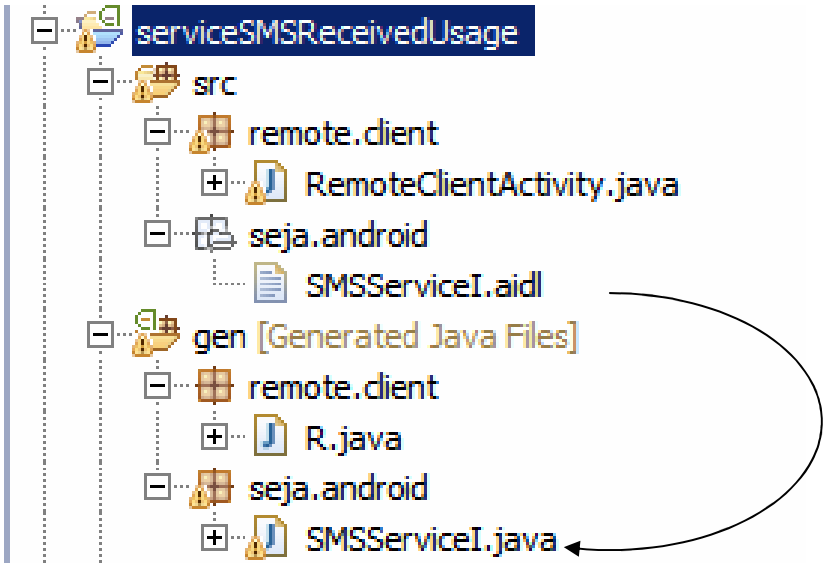
Pour tous, et ici dans un processus séparé

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="seja.android"
    android:versionCode="1"
    android:versionName="1.0">
  <uses-sdk android:minSdkVersion="8" />

  <application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".ServiceSMSReceivedActivity"
          android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

    <service android:name=".SMSService"
             android:exported="true"
             android:process="seja.android.SMSService" />
  </application>
</manifest>
```



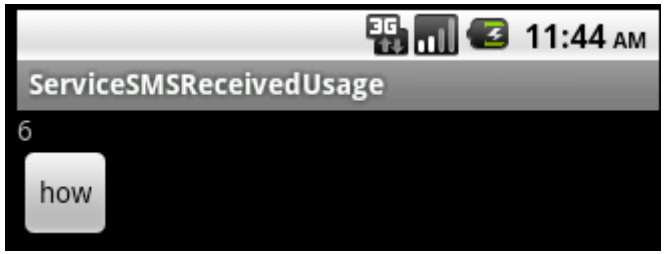
Un autre client depuis une autre application

- **Recopie du fichier .aidl, (langage commun)**
  - **Noms de paquetage identiques**



- **Même séquence qu'un client du même contexte**
  - **Recherche du service**
  - **Connexion au service**

Un client ordinaire d'une autre application




```
public void onClickHow(View v){
  if(!connecte){

   Intent intent = new Intent();
   intent.setClassName("seja.android", "seja.android.SMSService");
   this.connexion = new ConnexionAuServiceSMS();
   try{

     bindService( intent, connexion, Context.BIND_AUTO_CREATE);
   }catch(Exception e){}
   connecte = true;

  }else{
    try {
      texte.setText(Long.toString(service.received()));
    } catch (Exception e) {}
  }}

    private
       class ConnexionAuServiceSMS implements ServiceConnection{
```

à l'identique d'un client présent dans la même application

Exemple2 : SMSService enrichi

- **Un compteur de SMS reçus**
  - **A chaque sms reçu un compteur est incrémenté**
  - **Service accessible par toute application**
- **Le service a maintenant l'intention de recevoir des SMS**
  - **IntentFilter filter = new IntentFilter(*SMS_RECEIVED*);**
  - **registerReceiver(new SMSReceiver(), filter);**
- **Un client recherchera ce service via le middleware**
- **Le service : SMSService**

Le service 1/3, réveillé à chaque réception de SMS

```
public class SMSServiceImpl extends Service {

private static final String SMS_RECEIVED =
  "android.provider.Telephony.SMS_RECEIVED";

    private boolean active;
    private int     countSMSReceived;

  public void onCreate() {
    super.onCreate();
    IntentFilter filter = new IntentFilter(SMS_RECEIVED);
    registerReceiver(new SMSReceiver(), filter);
}
```

Le service 2/3, réveillé à chaque SMS

```
private class SMSReceiver extends BroadcastReceiver{
  public SMSReceiver(){
    Log.v("SMSReceiver","SMSReceiver()");
  }


@Override
public void onReceive(Context context, Intent intent) {
  Log.v("SMSReceiver","onReceive()");
  if(active) SMSServiceImpl.this.countSMSReceived++;
}
}
```

Le service 3/3, La souche le stub fournie à la demande

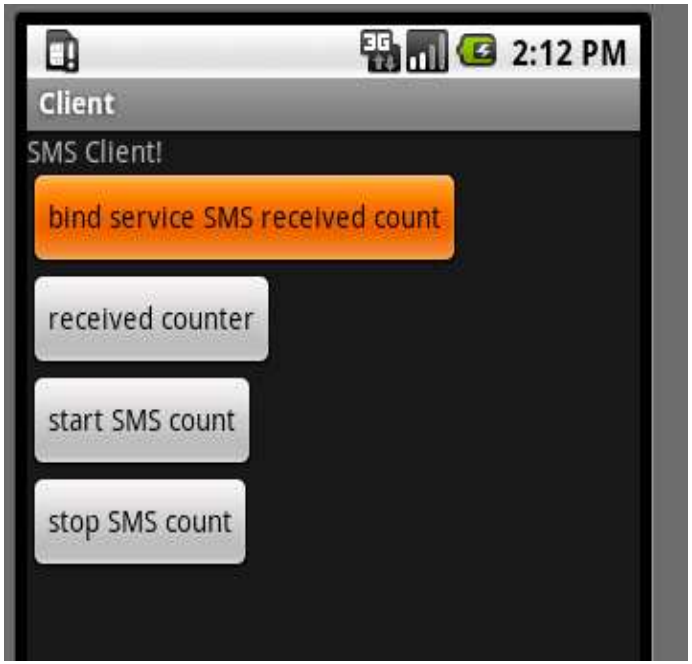
```
public IBinder onBind(Intent arg0) {
  return new SMSServiceStubImpl();
}


public class SMSServiceStubImpl extends SMSService.Stub{

  public long received() throws android.os.RemoteException{
    return SMSServiceImpl.this.countSMSReceived;
  }

  public void start() throws RemoteException {
    active = true;
  }
  public void stop() throws RemoteException {
    active = false;
  }
}
```

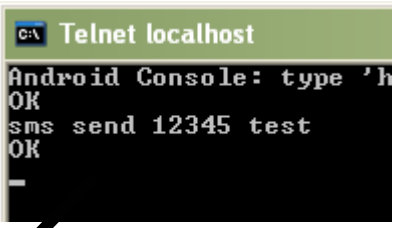
```
<service android:name=".SMSService"
                android:label= "mon service de SMS" />
```

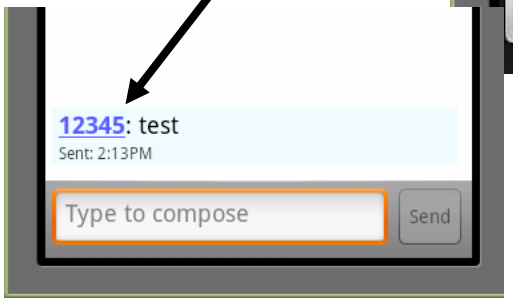
Le Client, une activity

1. bind service
2. start SMS count

telnet localhost 5554
3. sms send 12345 test







Conclusion

- **Une idée de tp …**
  - **Filtre et alerter en fonction des contenus des SMS**
    - **Un censeur ?**
    - **Big brother embarqué ?**
    - **Une commande distante**
- **Discussion**
  - **Choix d'une architecture**

Annexe suite

- **Parcelable**
- **PendingIntent**
- **Liste des services**
- **Divers**
- **MVC revisité !**

L'intent peut contenir des paramètres

- **Les extras, un *Bundle*, une *Map* !**
  - **Une table de couples <clé, valeur>, la clé est de type String**

```
Intent i = new Intent();
// i.setAction…
i.putExtra("fichier","hello.mp3");
i.putExtra("compteur",2);
```

Intent	putExtra (String name, double[] value) Add extended data to the intent.
Intent	putExtra (String name, int value) Add extended data to the intent.
Intent	putExtra (String name, CharSequence value) Add extended data to the intent.
Intent	putExtra (String name, char value) Add extended data to the intent.
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.

Des paramètres à l'intention de

L'activité lit les paramètres transmis

- **Les extras, un *Bundle*, une *Map* !**
  - **Une table de couples <clé, valeur>, la clé est de type String**

```
Intent i = getIntent();
String f = i.getStringExtra("fichier");
```

double	getDoubleExtra (String name, double defaultValue) Retrieve extended data from 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.

Des paramètres reçus par l'activité sélectionnée

putExtra, getExtras

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.
Intent	putExtra (String name, byte[] value) Add extended data to the intent.
Intent	putExtra (String name, long[] value) Add extended data to the intent.
Intent	putExtra (String name, Parcelable 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.
ArrayList<Integer>	getIntegerArrayListExtra (String name) Retrieve extended data from the intent.

http://developer.android.com/reference/android/content/Intent.html

Types simples : c'est prévu

Objets métiers : ils ***implement***eront Parcelable

L'activité transmet une instance complète : Parcelable

- **La classe des instances transmises implémente Parcelable**
- **Parcelable comme Serializable**
  - **–Sauf que tout est à la charge du programmeur**
  - **–Chaque champ doit être écrit et restitué**
- **Exemple :**
  - **–Auditeur est une classe qui implémente Parcelable**
    - **•Les noms de méthodes sont imposées via l'interface**
- **Envoi depuis l'activité a1**

```
Auditeur unAuditeur = new Auditeur("alfred");
intent.putExtra("auditeur", unAuditeur);
startActivity(intent);
```

- **Appel**
  - **– du constructeur**
  - **–Lors de la transmission**
    - **• writeToParcel**

```
public class Auditeur implements Parcelable {
   private String nom;
   private long   id;

   public Auditeur(String nom){
       this.nom = nom;
       this.id = globalId++;
    }

    @Override
    public int describeContents() {
        return 0;
    }

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

L'activité reçoit une instance Parcelable

- **Restitution :**
  - **Classe Auditeur suite**

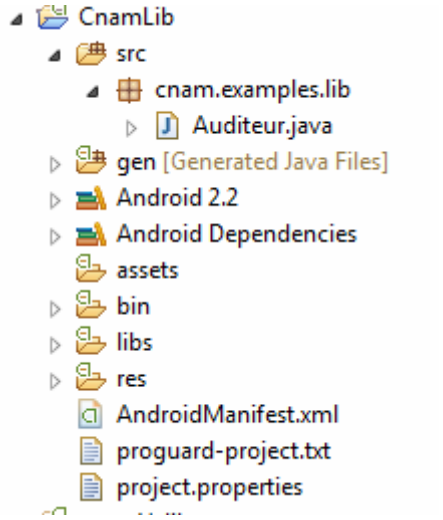
```
public static final Parcelable.Creator<Auditeur> CREATOR
        = new Parcelable.Creator<Auditeur>() {
    public Auditeur createFromParcel(Parcel in) {
        return new Auditeur(in);
    }

    public Auditeur[] newArray(int size) {
        return new Auditeur[size];
    }
};

private Auditeur(Parcel in) {
    this.nom = in.readString();
    this.id = in.readLong();
}
```

- **Les noms de méthodes sont imposées via l'interface**
- **Un constructeur avec en paramètre l'instance Parcel reçue**
  - **Déclenché par le mécanisme de restitution**

- **La classe *Auditeur* se doit d'être dans une librairie**
  - **(afin d'éviter la recopie multiple des sources)**

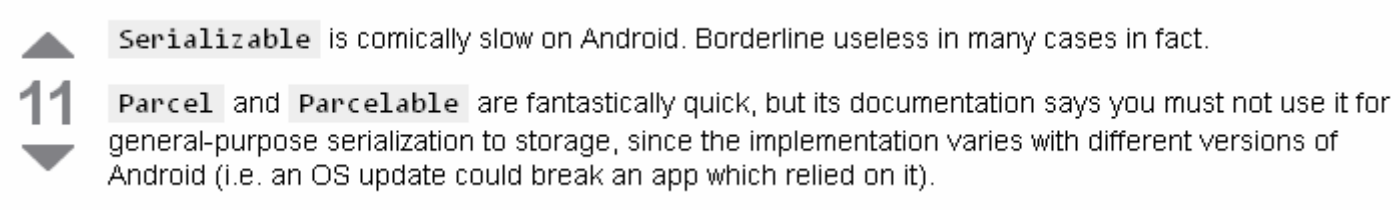
Une librairie, un .jar

- **Connaissance des classes par les deux activités ?**
  - **Toutes ces classes sont dans un projet eclipse : case isLibrary à cocher**
  - **Une librairie, un .jar à installer dans chaque application cliente**
  - **CnamLib contient la classe Auditeur ci-dessous**



Serializable / Parcelable, discussions

- **Serializable**
  - **Puissant mais lent, dû à l'usage de l'introspection**
  - ***Comically slow …***
- **Parcelable**
  - **Pas d'introspection, tout est défini par l'utilisateur**
  - ***Fanstastically quick …***
- **http://stackoverflow.com/questions/5550670/benefit-of-using-parcelable-instead-of-serializing-object**



- **Benchmark intéressant …**
  - **http://code.google.com/p/thrift-protobuf-compare/wiki/Benchmarking**
    - **Java Serializable, Externalizable, JSON …**

Autre façon de souscrire: PendingIntent

- Intent à effet immédiat
- PendingIntent à effet retardé

- A destination d'un service existant
  - AlarmManager, NotificationManager …

- Souscription auprès du service
  - **Exécution de l'intent passé en paramètre, à l'aide d'un PendingIntent**

Souscription auprès d'un service existant

- `Intent intent = new Intent(this, ReceiverTemplate.class);`
- `PendingIntent appIntent = PendingIntent.getBroadcast(this, 0, intent, 0);`

```
Calendar calendar = Calendar.getInstance();
calendar.setTimeInMillis(System.currentTimeMillis());
calendar.add(Calendar.SECOND, 3);

AlarmManager am = (AlarmManager)getSystemService(ALARM_SERVICE);

am.set(AlarmManager.RTC,
       calendar.getTimeInMillis(),
       appIntent);
```

// sendBroadcast(**intent)** par le service d'alarme

Variante de l'écriture précédente

```
PendingIntent appIntent =
  PendingIntent.getBroadcast
  (this, 0, new Intent(), 0);

Calendar calendar = Calendar.getInstance();
// idem diapositive précédente
am.set(AlarmManager.RTC,
       calendar.getTimeInMillis(),
       appIntent);
```

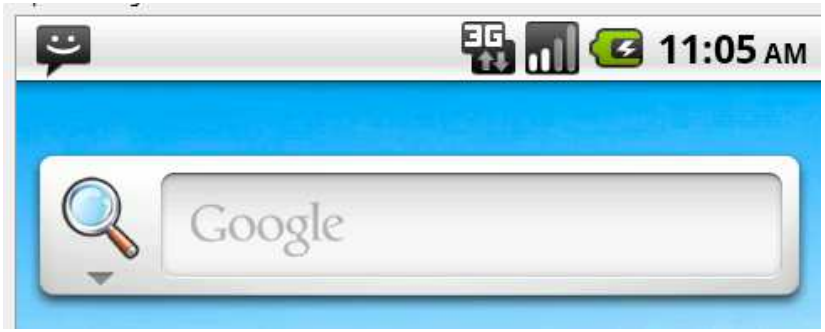
// Ce sont les receveurs déclarés dans AndroidManifest.xml

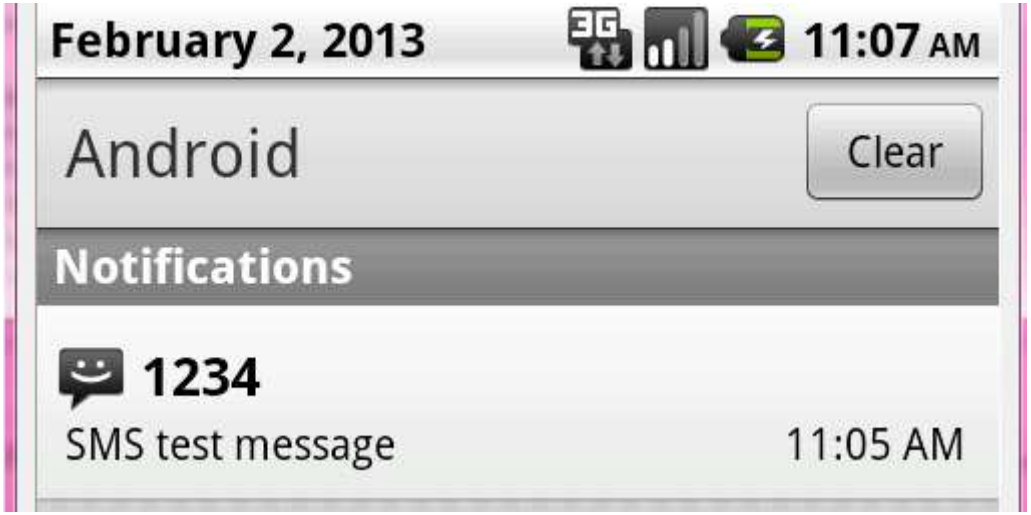
// qui seront déclenchés, (onReceive)

*// puis sendBroadcast(**intent)** par le service d'alarme*

Notification

- **Souvent associée à la réception d'un évènement**
  - **Déclenchée par le souscripteur**
    - **Un « Receiver », un service, …**





  - **Exemple:**
    - **Une notification à la réception d'un message sms**

» telnet localhost 5554

» send 1234 SMS test message

Notification : affichage

```
private static void generateNotification(Context context, String message) {
  long when = System.currentTimeMillis();


NotificationManager notificationManager = null;
notificationManager(NotificationManager)
   context.getSystemService(Context.NOTIFICATION_SERVICE);
Notification notification = new Notification(R.drawable.ic_launcher,message, when);
String title = context.getString(R.string.app_name);


Intent intent = // page suivante


notification.setLatestEventInfo(context, title, message, intent);
notification.flags |= Notification.FLAG_AUTO_CANCEL;
notificationManager.notify(0, notification);
}
```

Autre exemple : une notification

- Au clic sur la notification une application est déclenchée

```
private static void generateNotification(Context context, String message) {
  long when = System.currentTimeMillis();

NotificationManager notificationManager = null;
notificationManager(NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);
Notification notification = new Notification(R.drawable.ic_launcher,message, when);
String title = context.getString(R.string.app_name);


Intent notificationIntent = new
  Intent(context,GCMClientActivity.class);
// afin que l'intent retardée démarre une nouvelle activité
notificationIntent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
                            Intent.FLAG_ACTIVITY_SINGLE_TOP);


PendingIntent intent =
  PendingIntent.getActivity(context, 0, notificationIntent, 0);


notification.setLatestEventInfo(context, title, message, intent);
notification.flags |= Notification.FLAG_AUTO_CANCEL;
notificationManager.notify(0, notification);
}
```

- Source: http://android.amolgupta.in/2012/07/google-cloud-messaging-gcm-tutorial.html

Autre usage : une notification, sans effet au clic

- Au clic sur la notification rien ne se passe

```
private static void generateNotification(Context context, String message) {
  long when = System.currentTimeMillis();

NotificationManager notificationManager = null;
notificationManager(NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);
Notification notification = new Notification(R.drawable.ic_launcher,message, when);

String title = context.getString(R.string.app_name);

PendingIntent intent = PendingIntent.getActivity(context, 0, null, 0);

notification.setLatestEventInfo(context, title, message, intent);
notification.flags |= Notification.FLAG_AUTO_CANCEL;
notificationManager.notify(0, notification);
}
```

- Source: http://android.amolgupta.in/2012/07/google-cloud-messaging-gcm-tutorial.html
-

Quelques éléments pour le filtrage

Filtrage du contenu

```
public void onReceive(Context context, Intent intent) {

    Bundle bundle = intent.getExtras();
    Object[] pdus = (Object[]) bundle.get("pdus");
    SmsMessage messages =SmsMessage.createFromPdu((byte[]) pdus[0]);
    Log.i(TAG,  messages.getMessageBody());
```

Et la censure censure : sms n'est pas propagé …

```
if(contenuProhibé) abortBroadcast();
```

- **Si le contenu du SMS est inapproprié**
  - **Comment intercepter sa réception par l'application standard ?**
  - **À tester …**

```
<receiver android:name=".EmbeddedBigBrother">
  <intent-filter android:priority="1000">
      <action android:name="android.provider.Telephony.SMS_RECEIVED">
       </action>
  </intent-filter>
</receiver>
```

Divers : Service es-tu là ?

```
private boolean isMyServiceRunning(String nomCompletDuService) {
       ActivityManager manager = (ActivityManager) getSystemService(Context.ACTIVITY_SERVICE);
       for (RunningServiceInfo service : manager.getRunningServices(Integer.MAX_VALUE)) {
           if (nomCompletDuService equals(service.service.getClassName())) {
               return true;
           }
       }
       return false;
   }
```

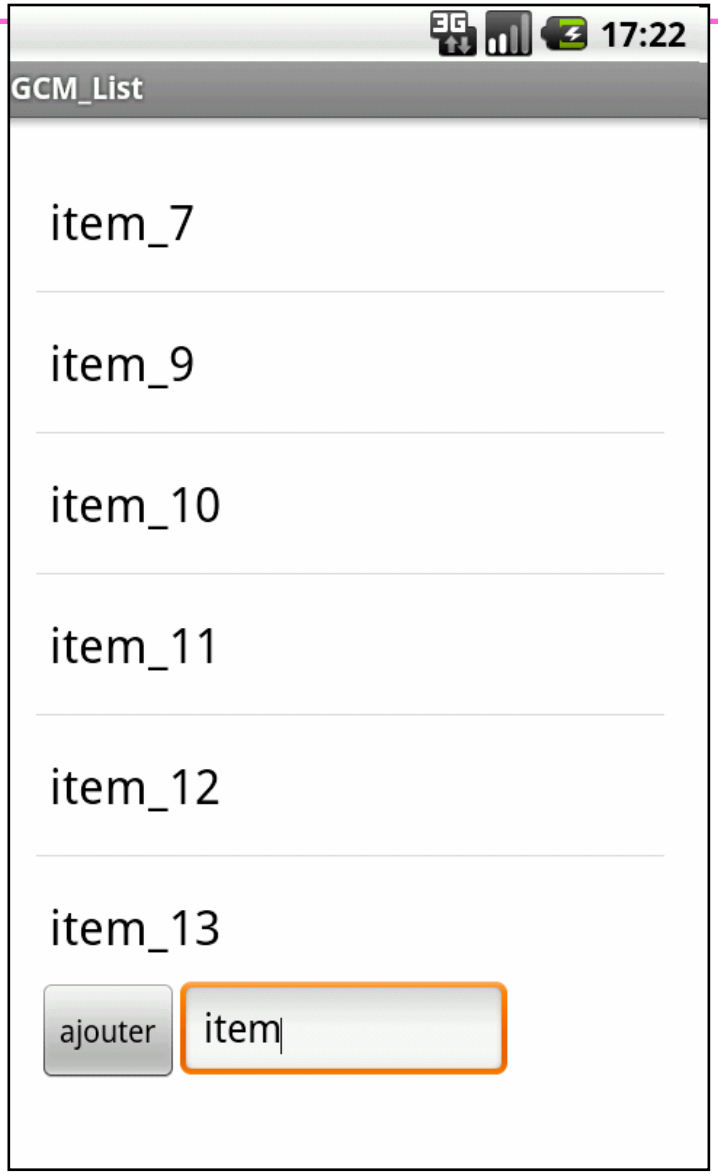
nomCompletDuService paquetage inclus

Un essai d'architecture: reprise de MVC en Android

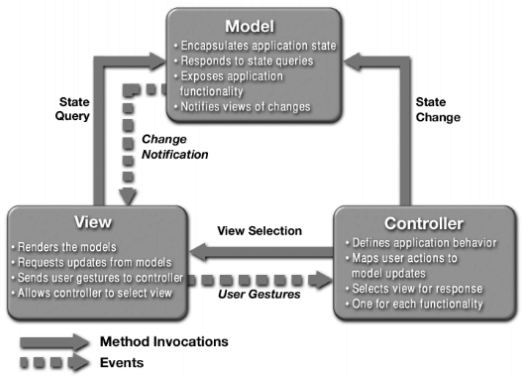
- **Application:**
  - **Une liste d'item : le modèle, du java standard, portable**
  - **ListView + ListActivity : la Vue**
  - **Un Receiver : le Contrôleur**

La vue

- **Une liste d'item …**
  - **Ajout et suppression**



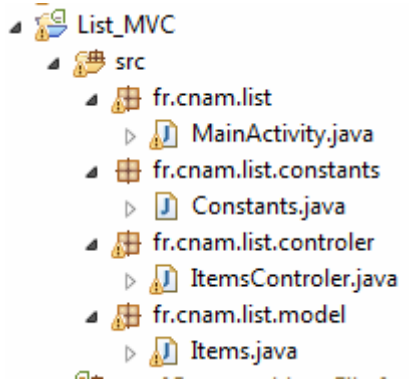
La liste d'items



Items
extends Observable

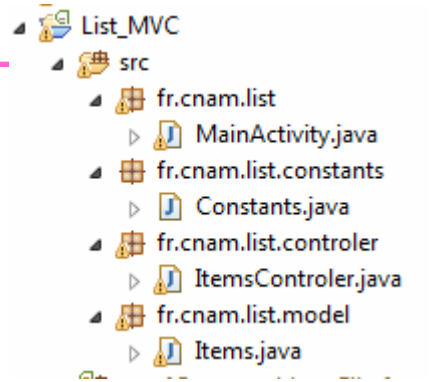
MainActivity
extends ListActivity
implements Observer

ItemsController
extends BroadcastReceiver



- Items : le modèle
- MainActivity : la vue
- ItemsController : le contrôleur

La classe Items : le modèle




```
public class Items extends Observable {

    private String name;
    private LinkedList<String> items;


    public Items(String name, Observer obs){
        this.name = name;
        this.items = new LinkedList<String>();
        this.addObserver(obs);
    }


    public void retirer(int position){
        synchronized(this){
            String str = this.items.remove(position);
            setChanged();
            notifyObservers(str);
        }
    }

    public void ajouterAuDebut(String item){
        synchronized(this){
            this.items.addFirst(item);
            setChanged();
            notifyObservers(item);
        }
    }
```

- **J2SE portable**

ItemsController

```java
public class ItemsControler extends BroadcastReceiver {

    public static final String ACTION = "fr.cnam.gcm.list.model.ITEMS";

    public static final String ADD        = "add";       // valeurs
    public static final String ADD_TOP    = "addtop";
    public static final String REMOVE     = "remove";

    private Items items;                // le modèle
    private Context context;            // Android framework

    public ItemsControler(Context context, final Items items){
        this.context = context;
        this.items = items;  // le modèle
    }

    @Override
    public void onReceive(final Context context, final Intent intent) {

        String operation = intent.getStringExtra(OPERATION_KEY);
        if(operation.equals(ADD_TOP))
            items.ajouterAuDebut(intent.getStringExtra(DATA_KEY));
        else if(operation.equals(REMOVE))
            items.retirer(Integer.parseInt(intent.getStringExtra(DATA_KEY)));
        else if(operation.equals(ADD))
            items.ajouter(intent.getStringExtra(DATA_KEY));

    }
```

- **Le contrôleur**

La Vue 1/2

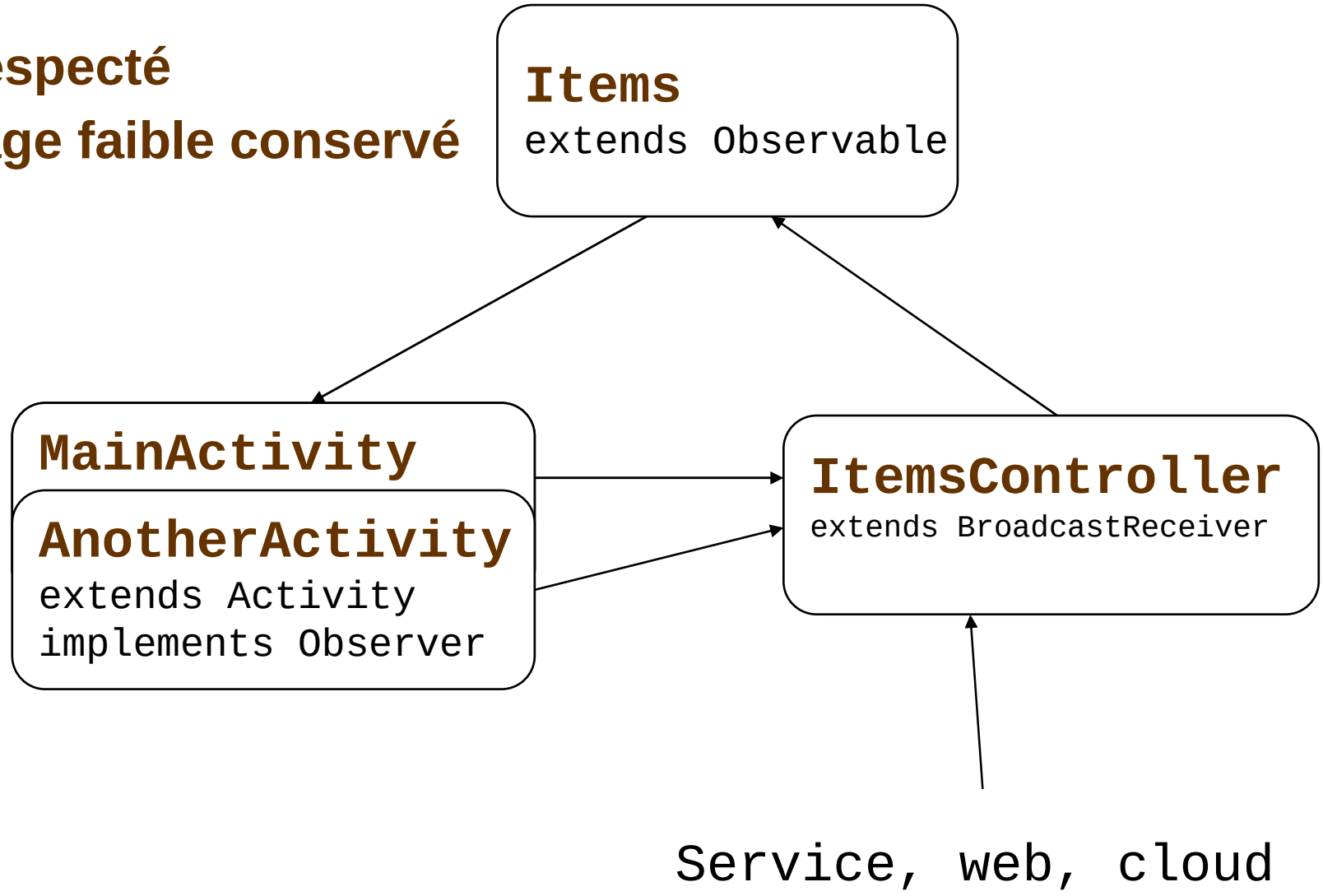
```
public class MainActivity extends ListActivity implements Observer{
    // pour l'appel de Log.i
    private final String TAG = getClass().getSimpleName();

    // la Vue : ListView, cf. le fichier XML

    private Items items; // le Modèle

    private ItemsControler itemsControler; // le Controleur


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.activity_main);

        this.items = new Items("items", this); // le modèle avec this son observateur
        this.itemsControler = new ItemsControler(this, items); // le contrôleur du modèle items

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, android.R.id.text1,items.getList());
        setListAdapter(adapter);

        for(int i=0;i<20;i++) items.ajouter("item_" + i);  // une liste avec quelques items

    }

    @Override
    public void onResume(){
        super.onResume();
        IntentFilter intentFilter = new IntentFilter();
        intentFilter.addAction(ItemsControler.ACTION);
        registerReceiver(itemsControler, intentFilter);
    }
```

La Vue 2/2

```
// Operation de retrait d'un item,
@Override
protected void onListItemClick(ListView lv, View v, int position, long id){
    // items.retirer(position);     // l'opération est transmise au Controleur

    Intent intentToItemsControler = new Intent();
    intentToItemsControler.setAction(ItemsControler.ACTION);
    intentToItemsControler.putExtra(OPERATION_KEY,ItemsControler.REMOVE);
    intentToItemsControler.putExtra(DATA_KEY,Integer.toString(position));
    sendBroadcast(intentToItemsControler);

}

// Operation d'ajout d'un item,
public void onClickAjouter(View v){
    EditText et = (EditText)findViewById(R.id.itemTexteId);
    String item = et.getText().toString();
    //items.ajouterAuDebut(item); // l'opération est transmise au Controleur


    Intent intentToItemsControler = new Intent();
    intentToItemsControler.setAction(ItemsControler.ACTION);
    intentToItemsControler.putExtra(OPERATION_KEY,ItemsControler.ADD_TOP);
    intentToItemsControler.putExtra(DATA_KEY,item);
    sendBroadcast(intentToItemsControler);

}


@Override
public void update(Observable arg0, Object arg1) {
    ((ArrayAdapter<String>)this.getListAdapter()).notifyDataSetChanged(); // mise à jour de la vue
}
```

MVC, discussions

- **MVC respecté**
- **Couplage faible conservé**



Intercepteur de SMS …

- **Cette application supprime tous les SMS entrants …**

```
public class SMSInterceptor extends BroadcastReceiver {
      @Override
      public void onReceive(Context context, Intent intent) {
          Log.i("onReceive","SMSInterceptor");
          Toast.makeText(context,"SMS received ...",Toast.LENGTH_LONG).show();
          abortBroadcast();
      }
   }
```

<- Le receveur

```
public class MainActivity extends AppCompatActivity {

      @Override
      protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          finish();
      }
}
```

<- L'activité

<- L'application

```
<activity android:name=".MainActivity"
  android:theme="@android:style/Theme.Translucent.NoTitleBar">
      <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
  </activity>

  <receiver android:name=".SMSInterceptor" android:exported="true"
  android:enabled="true">
      <intent-filter android:priority="2147483647">
         <action android:name="android.provider.Telephony.SMS_RECEIVED" />
      </intent-filter>
  </receiver>
```

Intercepteur de sms

Soit :

- **Une application « transparente » ***
- **Un receveur de SMS entrants**
  - **La bonne permission**
  - **Note : depuis la version 3.1,**
    - **les receveurs installés dans le fichier AndroidManifest.xml ne sont plus abonnés l'installation de l'application sur le mobile**
    - *** Ici l'application s'exécute en mode transparent et installe implicitement le receveur…**