Legond-Aubry Fabrice
fabrice.legond-aubry@u-paris10.fr

PPM(A)

Programmation sur Plateformes Mobiles (Android)

BACKEND

PLAN

- Communication & Partage
- Services, AsyncTask, Jobs
- Persistance & Gestion des données
- Sécurité
- Spécificité de la PM

Communication

&

Partage de données

Communications - Partage de données

- **Possibilité de partage de données entre activités**
  - Données dans un message (Intent)
    - EXTERNE et INTERNE
  - Utilisation d'une classe Singleton
    - ATTENTION aux initialisations !!!
    - INTERNE (entre classes d'une même application)
  - Utilisation du singleton "application"
    - Votre application Android étend la classe "Application"
    - INTERNE (entre classes d'une même application)
  - Champs publics statiques dans les classes
    - ATTENTION aux initilisations !!!
    - INTERNE (entre classes d'une même application)
  - Hashmap de références faibles (WeakReferences)
    - Peu utilisé (complexe) – pas abordé dans ces slides
- **Possibilité de persistance (lourd)**
  - Une forme particulière de partage
  - SQLite, fichier ...

Communication : Intent

- ELEMENT PRINCIPAL DE COMMUNICATIONS DANS ANDROID
- Bus de communication
  - Asynchrone
  - Qui fait circuler des objets « Intent »
- Permet d'envoyer des messages. Peut être vu comme une demande de démarrage du composant récepteur.
- Permet de démarrer une activité, des services auxiliaires
- Permet de transmettre des données dans le messages
- Ne pas confondre « Intent » et « Message » sous Android
- **Utilisé aussi pour les évènements Systèmes**
- Rappel :
  https://developer.android.com/reference/android/content/Intent.html
  https://developer.android.com/guide/components/intents-common.html

Communication : Intent

- Générateur d'Intent pour les activités (dans la classe Intent)
  - makeRestartActivityTask, makeMainSelectorActivity
- Ne pas confondre « Intent » et « IntentCompat »
  - IntentCompat nécessite la libraire externe android.support.v4.*
- Bonne pratique : déclaration dans une classe d'activité de méthode « public static » pour la démarrer l'activité ou générer un Intent.
- Bonne Pratique : Mettre les noms des champs extras en « public static final »

```
public static Intent newActivityIntent (Context context, String param1, String param2)  {
        Intent intent = new Intent(context, ExampleActivity.class); // extras
        intent.putExtra(EXTRA_PARAM1, param1);
        intent.putExtra(EXTRA_PARAM2, param2);
        return intent;
}

public static void startActivity (Context context, String param1, String param2)  {
        Intent intent = new Intent(context, ExampleActivity.class); // extras
        intent.putExtra(EXTRA_PARAM1, param1);
        intent.putExtra(EXTRA_PARAM2, param2);
        context.startActivity(intent);
}
```

Communication : Intents

- Ne pas oublier de déclarer les classes activable dans le AndroidManifest
- Description abstraite d'une opération à réaliser
  - C'est un message (« à la MDB » dans EJB, « oneway » dans CORBA)
- On utilise les intents pour démarrer une nouvelle activité
  - Sert de «glue» entre les différentes activités
  - startActivity: permet de lancer une nouvelle activité
- On utilise les intents pour démarrer un service
  - startService ou bindService: communication avec les Services
- On utilise des intents pour communiquer, ou envoyer/recevoir des évènements
  - broadcastIntent: Communication avec les broadcastReceiver (exemple Alarme, SMS, etc.)
- Toutes les activités activable via des Intent doivent être déclarées dans AndroidManifest.xml
  - Automatique dans les IDE

Communication : Intents

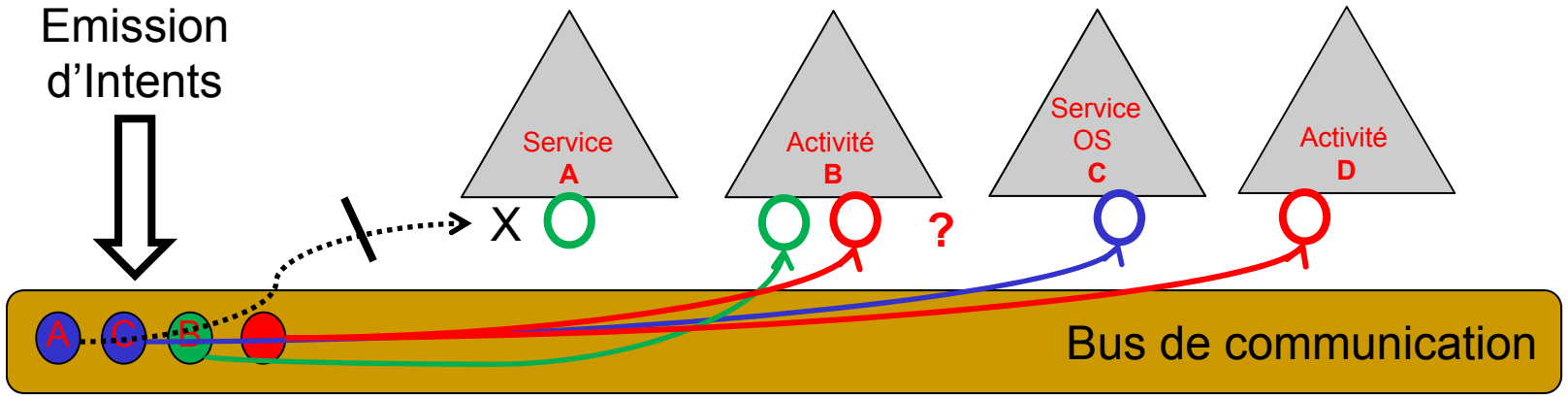
- Il existe des intents implicites et des intents explicites
  - Intent implicite : Intent sans destinataire
  - Intent explicite : Intent avec destinataire
- Il existe des « Intents OS » (ie non applicatifs)
  - Ils permettent de démarrer des activités standards
  - Ils offrent des mécanismes permettant de trouver l'activité la plus adéquate pour une action (intent implicite)
    - Pour ouvrir un fichier particulier
    - Pour ouvrir une page web, etc.
- Une application peut se déclarer auprès du système comme récepteur possible d'un intent. On utilise
  - un filtre (IntentFilter)
  - un broadcastReceiver : Récepteur / Consommateur de messages

Communication : Intent

- **Attention ! Un filtre dédié à chaque action**
- Déclaration des intents dans le AndroidManifest.xml
  - Souvent automatique dans les IDE
- Intent est un message asynchrone !!!
  - Attention aux délais de livraisons
  - Aux fausses assertions de causalité
  - PROGRAMMATION CONCURRENTE
- Tous les Intents partagent le même bus
  - Possibilité de «Broadcast» très utilisé pour les services.
  - Eviter les broadcasts non nécessaires → coûteux
  - Eviter les broadcast publics → coûteux et polluant

Communication : Intent

- Compléments sur les intents prédéfinis (OS)
  - Intent: « android.content.Intent.ACTION_* »
  - Paramètres : « android.content.Intent.EXTRA_* »
  - On les retrouve disséminés dans diverses classes du système (services systèmes, ...)
- On peut créer ses propres Intent



Communication : Structure d'un Intent

- Un intent est constitué **d'attributs primaires**
  - D'une action (chaîne de caractère)
    - Vous pouvez mettre n'importe quoi, mais ne le faîtes pas
    - Rappel pour le nom de l'action : utiliser des chaînes déclarées « public static final »
      - Utiliser MAJUSUCULES et _ (uniquement)
      - Ex: « ACTION_* »
    - Rappel : valeur chaîne « package.package.class.action.* »
      - * en MAJUSCULE
    - Méthode setAction()
  - D'une donnée
    - Sous forme d'un objet Uri
    - Passage URI → toUrl() / URL → toUri()
    - Méthode setData()
    - Peut être précisé par le champ type optionnel

Communication : Structure d'un Intent

- Un intent est constitué **d'attributs secondaires**
  - D'une ou plusieurs catégorie(s)
    - ✔ Tableau de chaînes de caractères
    - ✔ Peut être une catégorie définie par vous ou une catégorie prédéfini
      - Rappel : Même règle que pour les ACTION créées …
      - « package.package.class.category.* »
    - ✔ Méthode addCategory()
    - ✔ Pour faire simple : soit null soit « android.content.Intent.CATEGORY_DEFAULT »
    - ✔ Sinon il existe des catégories prédéfinies utilisable dans AndroidManifest.xml
      - CATEGORY_CAR_DOCK: activité à activer lors de l'activation du mode voiture
      - CATEGORY_LAUNCHER: l'activité peut être lancée du launcher

Partage de données

Communication : Structure d'un Intent

- Un intent est constitué **d'attributs secondaires**
  - D'extras (données optionnelles) – putExtras()
    - Les Extras sont des couples (clef,valeur)
    - Rappel : Pour les clefs applicatives, même règle que pour les ACTION créées …
    - « package.package.class.extras.* »
  - De drapeaux d'états (flags)
    - Méthode setFlags(), addFlags(), getFlags()
    - Influence le traitement de l'intent
      - FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS
      - FLAG_ACTIVITY_NO_ANIMATION
      - FLAG_ACTIVITY_NO_HISTORY

Communication : Structure d'un Intent

- Un intent est constitué **d'attributs secondaires**
  - D'un type – setDataAndType()
    - ✔ Type mime du « data » pour les intents implicites
    - ✔ Ne pas utiliser setData() puis setType(), ou inversement
      - L'un remet l'autre à zéro
    - ✔ Utiliser setDataAndType()
  - D'une cible du message (une classe)
    - ✔ setComponent() ou setClass()
    - ✔ **NULL → Intent IMPLICITE**
    - ✔ **Not NULL (Uneclasse.class ou un component) → Intent EXPLICITE**

Communication : Intent Explicite

- Intent Explicite
  - Permet d'envoyer un Intent à un composant précis
    - ✔ Essentiellement utilisé dans le cadre d'une application
    - ✔ Mais aussi pour exécuter une action sur un composant précis
- Création d'un Intent Explicite pour démarrer un service

```
// création avec une classe de destination non nulle (DownloadImageService)
Intent downloadIntent = new Intent(this, DownloadImageService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

Communication : Intent Implicite

- Intent Implicite
  - Sélection automatique suivant
    - ✔ Action, l'uri (data) et le type de données
  - Recherche au niveau des IntentFilters enregistrés
  - Utilisé quand une application ne possède pas un service ou quand on veut laisser le système gérer le traitement (URL)
  - les intents implicites (**cible null**) peuvent être dangereux
  - on ne sait pas quel service va le traiter !!

Communication : Intent Implicite

- En cas d'utilisation implicit des intents

```
Intent intent = new Intent ( Intent.ACTION_VIEW,
            Uri.parse ("http://www.lip6.fr") );
// Verify that the intent will resolve to an activity
if (intent.resolveActivity(getPackageManager()) != null) {
      startActivity(intent);
}
```

- Utilisation du PackageManager pour tester les capacités du téléphone
  - Permet de nombreux autres tests sur les senseurs, les détails de l'application
  - Permet aussi de tester les intents

```
PackageManager mgr = ctx.getPackageManager();
List<ResolveInfo> list = mgr.queryIntentActivities
        (intent, PackageManager.MATCH_DEFAULT_ONLY);
```

Communication : Intent Implicite

- Exemple d'utilisation implicite des intents
  - ATTENTION A LA GESTION DES DROITS

```
Intent makeCall = new Intent ( Intent.ACTION_CALL,
      Uri.parse (''0601020304'') ) ;
startActiviy (makeCall);
```

- Exemple d'utilisation explicite des intents

```
Intent i = new Intent(this, NewActivity.class);
i.putExtra("key", "value");
startActivity(i);
```

- Pour récupérer les données de l'intent dans l'activité
  - getIntent() sur l'intance de l'activité

Communication : Intent Extras « Simples »

- Pour ajouter des données (extras) simples, il existe
  - Différentes méthodes putExtra pour les types basiques et les tableaux de types basiques
  - Des méthodes pour manipuler les extras : removeExtra, replaceExtras
  - un putExtra pour un Bundle (voir plus loin)
- Pour les types complexes, il faut un traitement particulier
  - Il faut que la classe « implements » (AU CHOIX)
    - ✔ Soit Serializable
    - ✔ Soit Parcelable

Communication : Intent – Extras Parcelable

- Un objet Parcelable n'est pas une simple interface
  - Codage à la main du mécanisme de codage
  - Champ « public static final » CREATOR qui
    - référence une instance d'un objet « Parcelable.Creator »
    - est capable de créer une instance de votre classe à partir d'une Parcel
    - implémente une Méthode createFromParcel(Parcel source, ClassLoader loader) dans le creator
  - Implémenter une méthode writeToParcel(Parcel, int). Crée une Parcel
    à partir d'une instance
- Mécanisme très proche de Serializable
  - Ce n'est pas une sérialisation, cela n'a pas le même but
  - C'est plus léger dans le traitement, c'est plus lourd dans le codage
- Fonctionne pour les Intent et la variable Bundle

Communication : Intent – Extras Parcelable


```
public class Person implements Parcelable {
    // parcel keys
    private static final String KEY_NAME = "name";
    private static final String KEY_AGE = "age";
    private String name;
    private int age;
    …
    // getter / setter
    …
    // constructeur
    public Person(String name, int age) {
        this.name = name; this.age = age;
    }
    …
```

Communication : Intent – Extras Parcelable


```
…
public int describeContents()
    { return 0; }

// Sauvegarde
@Override
public void writeToParcel(Parcel dest, int flags) {
    // create a bundle for the key value pairs Bundle
    bundle = new Bundle();
    // insert the key value pairs to the bundle
    bundle.putString(KEY_NAME, name);
    bundle.putInt(KEY_AGE, age);
    // write the key value pairs to the parcel
    dest.writeBundle(bundle);
}
```

Communication : Intent – Extras Parcelable

Persistance

```
...

// * Creator required for class implementing the parcelable interface.
// Restauration
public static final Parcelable.Creator<Person> CREATOR
    = new Creator<Person>() {
        @Override
        public Person createFromParcel(Parcel source) {
            // read the bundle containing key value pairs from the parcel
            Bundle bundle = source.readBundle();
            // instantiate a person using values from the bundle
            return new Person(bundle.getString(KEY_NAME),
                bundle.getInt(KEY_AGE));
        }
        @Override
        public Person[] newArray(int size)
            { return new Person[size]; }
    };
}
```

Communication : Consommateur d'Intent

- Gestionnaire des récepteurs de l'activité / service:
  - LocalBroadCastManager
    - Se récupère via la méthode static getInstance()
  - Il permet d'enregistrer les BroadCastReceiver de l'application AVEC UN FILTRE d'intent
    - Le filter peut être définit dans le manifest (<intent-filter>)
    - registerReceiver, unregisterReceiver
    - Attention aux abonnements aux events Systèmes (COUTEUX)
  - Il permet d'envoyer des Intent
    - sendBroadcast, sendBroadcastSync
- Votre receiver doit étendre BroadCastReceiver
  - Doit implanter la méthode onReceive

Communication : startActivity avec retour

- Pour démarrer une activité renvoyant un résultat
- Il faut définir une activité générant un retour

```
public class ActivityWithResult extends Activity {
    @Override
    public void finish() {
        Intent intent = new Intent();
        intent.putExtra("result", string);
        // peut être RESULT_CANCELED, RESULT_OK
        setResult(RESULT_OK, intent);
        super.finish();
    }
}
```

Communication : startActivity avec retour

- Démarrer l'activité classiquement (vous pouvez utiliser putExtras)
  - startActivityForResult(intent, REQUEST_CODE);
  - REQUEST_CODE → code identifiant la requête
- Le résultat de l'activité « ActivityWithResult » se fait par l'exécution de la méthode « onActivityResult » dans l'activité invoquante

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode == RESULT_OK && requestCode == REQUEST_CODE) {
                if (data.hasExtra("result")) {
                        String result = data.getExtras().getString("result");
                        if (result != null && result.length() > 0) {
                                Toast.makeText(this, result, Toast.LENGTH_SHORT).show();
        } } }
}
```

Communication : PendingIntent

- Un « PendingIntent »
  - Un jeton (token) que l'on donne a une autre application
  - Il permet a l'autre application d'invoquer un morceau de son propre code à la fin d'un évènement. **(sécurité)**
  - Vous donnez l'autorisation à processus invoqué (service, ...) de vous répondre plus tard
  - Peut être vu comme un callBack
- Ex: armement d'un timer, exécution d'un code à la fin du timer par le renvoie (en retour) du PendingIntent du timer vers votre application.
- Nécessite un moyen de transport. En général, ils sont transmis dans des « Intents » pour une future réponse (~ une BAL).
- Utile par exemple pour associer un Intent de retour d'une notification (via setContentIntent), d'une alarme …

Communication : PendingIntent

- Pour obtenir une instance d'un PendingIntent
  - Pas de contructeur
  - Utilisation des méthodes
    - ✔ getActivity() → l'intent de retour servira a démarrer une activité
      Context.startActivity(Intent)
    - ✔ getActivities() → même chose que précédent, mais gère un tableau d'intent
    - ✔ getBroadcast() → l'intent de retour servira pour envoyer un message
      Context.sendBroadcast()
    - ✔ getService() → l'intent de retour servira pour démarrer un service
      Context.startService()
- Exemple de création d'un PendingIntent

```
Intent retour = new Intent(MainActivity.this, MonReceiver.class);
pendingIntent = PendingIntent.getBroadcast
        (MainActivity.this, 0, retour,0);
```

Communication : Classe Singleton

- Une classe de stockage et de partage des données
  - Avec ou sans getter/setter
  - Toutes les classes peuvent y accéder
  - Mieux vaut éviter le initInstance et initialiser lors de la déclaration
  - Attention en cas de multithread, utiliser synchronize

```
public class StaticSingletonAppData  {
      // Le singleton et les données
      private static StaticSingletonAppData  instance = null;
      private  ArrayList<String> stringList = null;

      // le constructeur est en private → caché
      private StaticSingletonAppData  () { }

      // initInstance & getInstance du singleton
      public static void initInstance () {
      if (instance==null)
            instance=new instance();
      }
      public static StaticSingletonAppData  getInstance()
            { return instance; }

      // les getter/setter sur les variables
      public ArrayList<String> getStringList()
            { return stringList; }
      public void setStringList(ArrayList<Host> _stringList)
            { stringList = _stringList; }
}
```

Communication : Classe Singleton « Application »

- Surdéfinition de la classe Application
  - Démarrage de l'application par Application
  - Possibilité d'ajouter des Singletons
  - Possibilité d'ajouter des variables membres classiques avec getter/setter
- Utilisation de la classe par la méthode getApplicationContext()

```
public class MyApplication extends Application
{
	private static StaticSingletonAppData singleton;
	@Override
	public void onCreate()	{
		super.onCreate();
		// Initialize the singletons so their instances
		// are bound to the application process.
		initSingletons();
	}

	protected void initSingletons()	{
		// Initialize the instance of StaticSingletonAppData
		singleton.initInstance();
	}
	...

}
```

Communication : Variables de classes statiques

- Même idée que le singleton
- Getter/setter statiques dans la classe

```
public class DataHolder {
    private static String data;
    public static String getData()
        { return data; }
    public static String setData(String data)
        { this.data = data; }
}
```

- Dans un des activités de l'application :

```
String data = DataHolder.getData();
```

Communication : Hashmap de références faibles


```
public class DataHolder {

    Map<String, WeakReference<Object>> data = new
                        HashMap<String, WeakReference<Object>>();

    void save(String id, Object object) {
        data.put(id, new WeakReference<Object>(object));
    }

    Object retrieve(String id) {
        WeakReference<Object> objectWeakReference = data.get(id);
        return objectWeakReference.get();
    }
}
```

Communication : Hashmap de références faibles

- Weak References. Elles existent en Java
  - Pour la culture aller voir aussi les SoftReference, PhantomReference

Communication : Hashmap de références faibles

- Utilisation en sauvegarde

```
DataHolder.getInstance()
    .save(someId, someObject);
```

Utilisation en restauration

```
DataHolder.getInstance()
    .retrieve(someId);
```

Les Services

Les tâches asynchrones

Les (Lollipop) Jobs

Threads

Services : Description

- Tâches de fond
  - Equivalent à la notion de services sous Linux
  - Pas d'interface utilisateur
  - Peut lancer des notifications, des activités (UTILISE AVEC PARCIMONIE)
  - Utilisé pour les mails, connexions
- Un cycle de vie simplifié par rapport à l'activité
- Un service peut être contrôlé par une activité ou un widget
- Une application peut déclarer plusieurs services
- Un service permet d'effectuer des tâches répétitives et de longues durées
- Un service est déclarés dans l'AndroidManifest.xml

Services : Description

- Un service n'est pas un processus séparé par défaut
  - Un service n'est pas un thread
- Un service peut être un processus séparé
  - Utilisé par ex. pour faire des opérations hors du thread principal de l'UI
- Comme souvent, le système se charge uniquement d'instancier l'objet et d'appeler les callbacks
- Le développeur doit gérer la création d'un thread dédié, allocation ressources etc.
  - Cas typique : création d'un thread / AsyncTask, à la fin du thread le service meurt

Services Systèmes : Description

- Il existe des services systèmes
  - Accessibles via getSystemService() et le context / activité
  - Object getApplicationContext(). getSystemService(String serviceName)
- Liste complètes des services systèmes (*_SERVICE):

  https://developer.android.com/reference/android/content/Context.html#constants

« serviceName »	Classe du service
WINDOWS_SERVICE	WindowsManager
LAYOUT_INFLATER_SERVICE	LayoutInflater
POWER_SERVICE	PowerManager
NOTIFICATION_SERVICE	NotificationManager
CONNECTIVITY_SERVICE	ConnectivityManager
WIFI_SERVICE	WifiManager
LOCATION_SERVICE	LocationManager
SENSOR_SERVICE	SensorManager
…	…

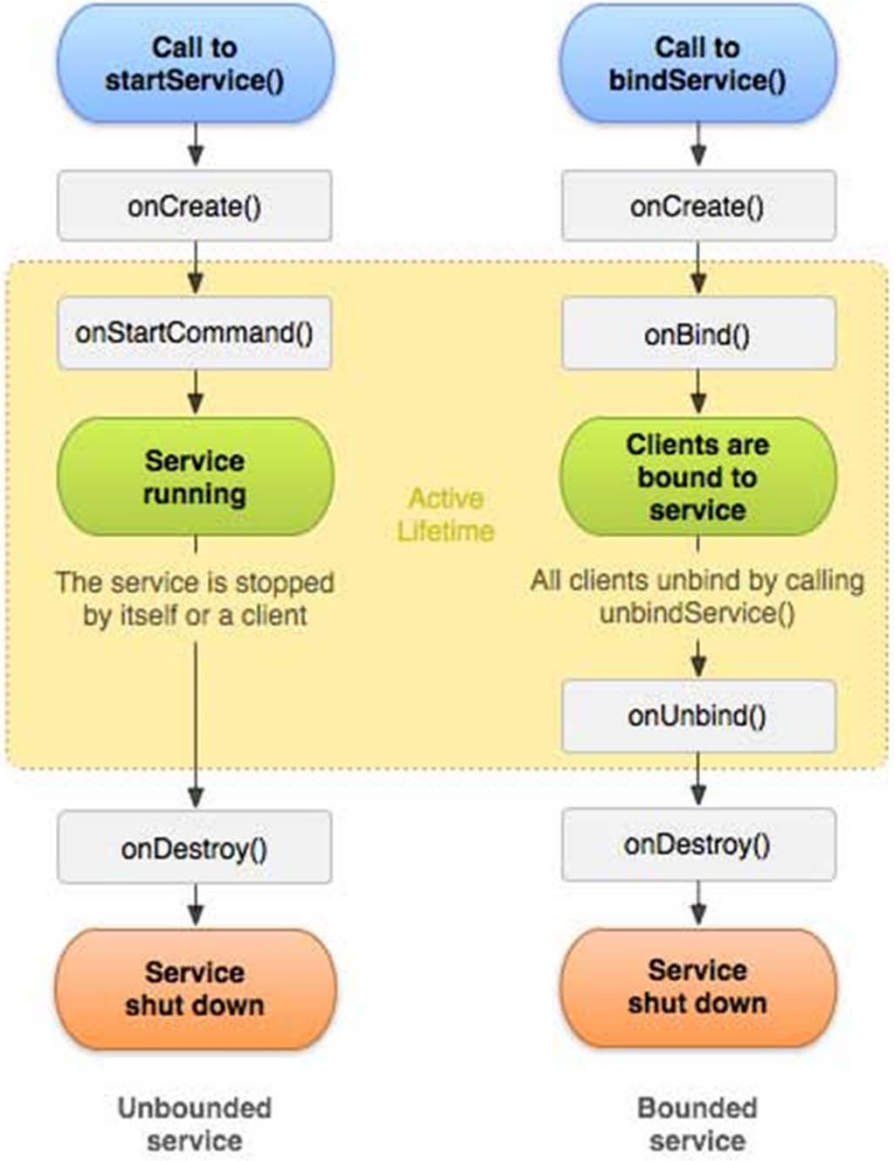
Services Applicatifs : Description

- La classe service peut être gérée de deux façons (classe Service) :
  - Sous forme de service lié, service en continu (bound service)
    - ✔ lié à une activité, exécution tant qu'il y a une connexion au service
    - ✔ Le client se connecte au service (callback, utilisation de la classe ServiceConnection)
    - ✔ Implémentation de « onBind » du côté 'service' ne doit pas renvoyer null
    - ✔ Implémentation des méthodes de callback onServiceConnected, onServiceDisconnected du côté 'client'
  - Sous forme de service non lié (unbound service)
    - ✔ onStartCommand() du 'service' est déclenchée par l'envoi d'un Intent du côté client
    - ✔ Service « oneshot » : téléchargement d'un fichier
    - ✔ Surdéfinition des méthodes « onCreate », « onDestroy », … de votre classe « Service »
- Un service peut être lié ET non lié
  - 2 aspects d'un même service en même temps

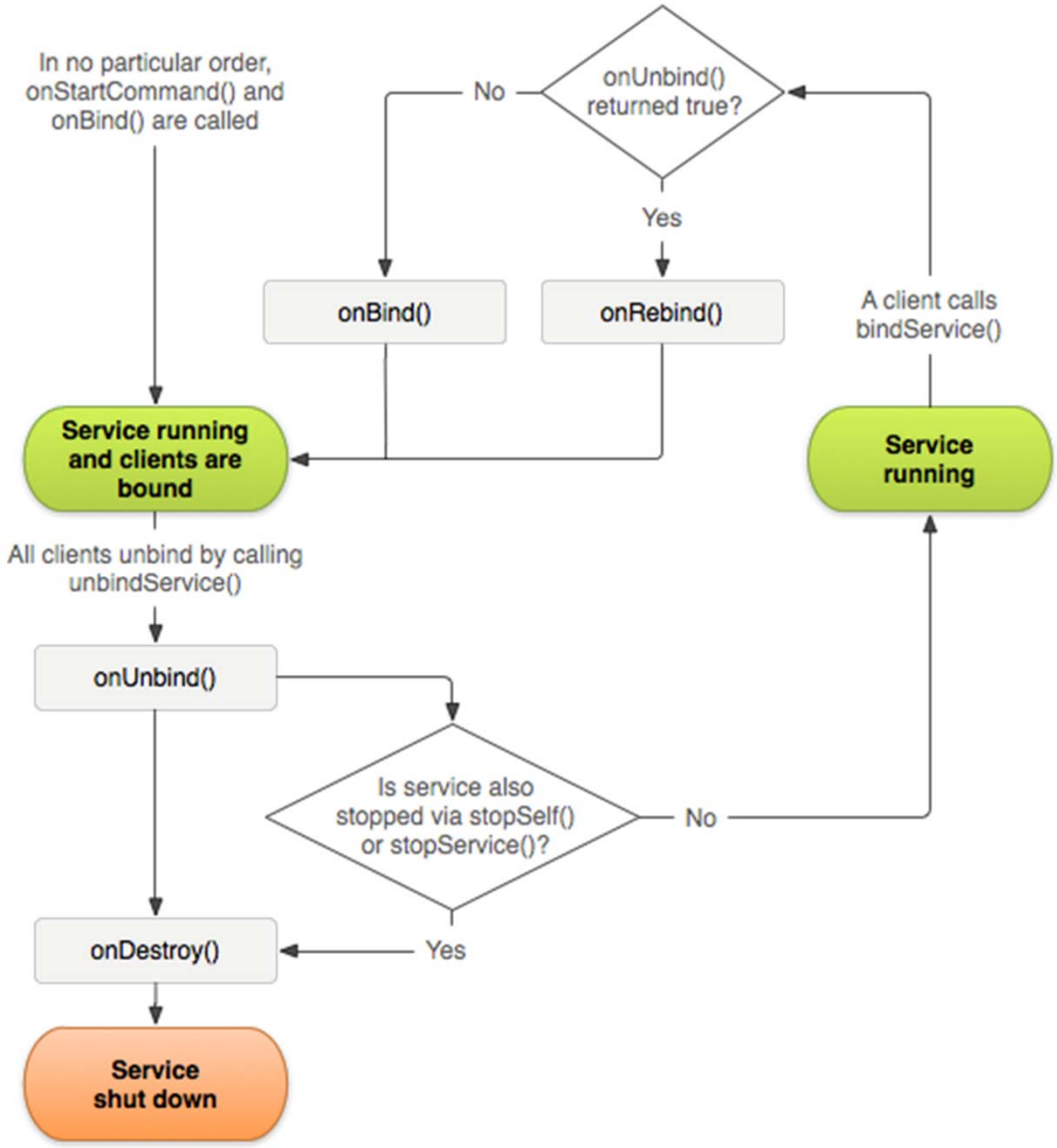
Services Applicatifs : Description

- Un service (orthogonalement à être lié ou non) peut être privé ou public
  - Privé: Seule l'application peut l'utiliser
    - ✔ Utilisation d'un nom COMMENCANT AVEC « : » dans l'attribut « android:process » du manifest
  - Public: Toute application installée sur le téléphone peut l'utiliser.
    - ✔ Utilisation d'un nom NE COMMANCANT PAS AVEC « : » dans l'attribut « android:process » du manifest
    - ✔ Communiquer nécessite un message handler (gestionnaire) pour gérer les **messages** IPC
    - ✔ Utilisation des IPC via un fichier **AIDL ou Messenger (2 méthodes)**
    - ✔ Les messages IPC ne sont pas des Intents !!!
- Lorsqu'un service est public, il est DANS UN PROCESSUS SEPARE !!
- En plus, quelque soit sa forme, un service peut gérer les intents
  - en étendant IntentService
  - Gestion des évènements asynchrones
  - En implémentant la méthode « onHandleEvent »

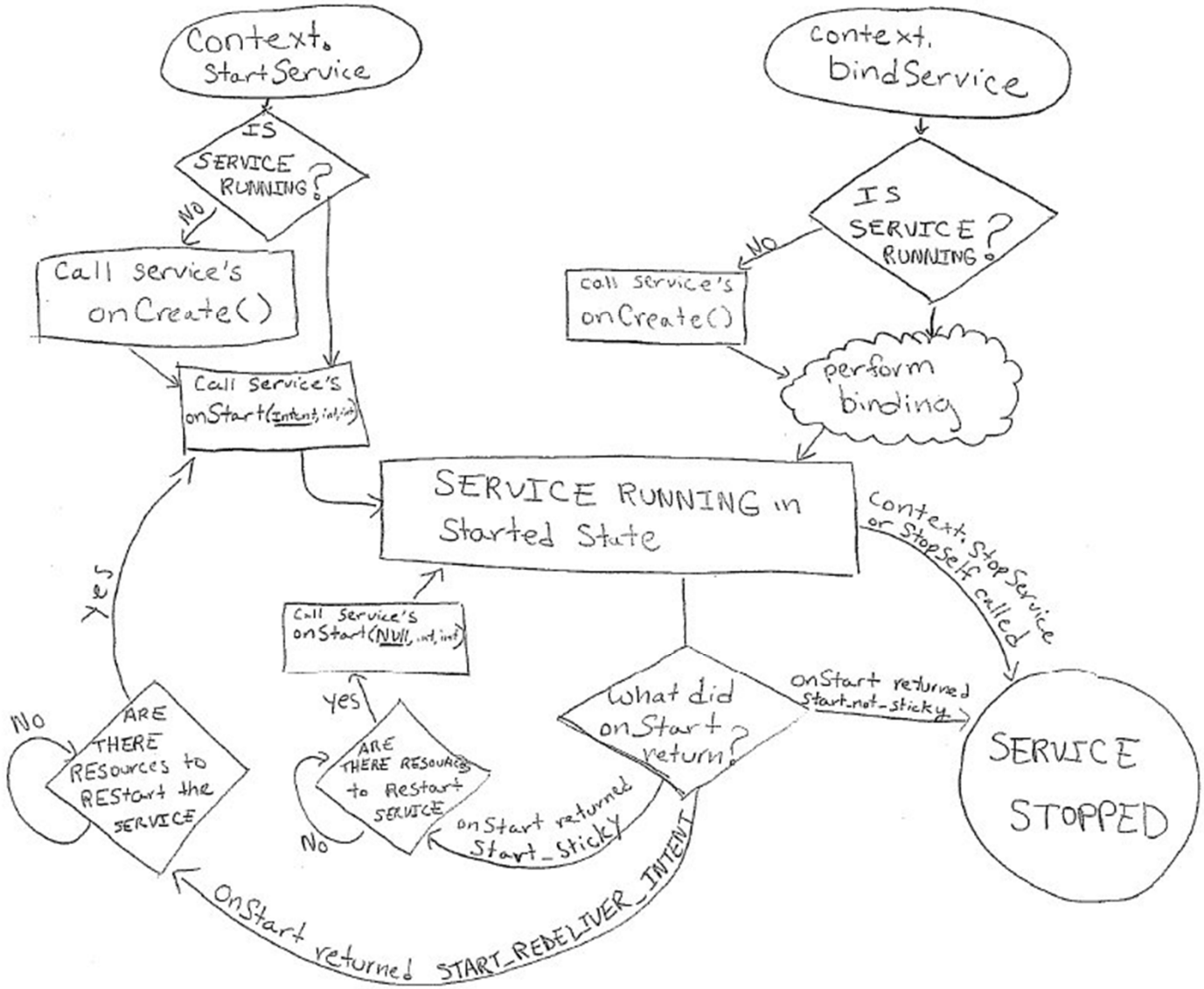
Services : Cycle de vie



Services : Cycle de vie



Services : Cycle de vie



Services privés non liés (private unbounded services)

- Rappel : Un service non lié
  - onStartCommand() du ‘service’ est déclenchée par l’envoi d’un Intent du côté client
  - Invocation de startService du côté invocateur
  - Service « oneshot »

Services privés non liés (private unbounded services)

- Exemple (Dummy Test Service)

```
public class TestUnboundedService extends Service {
    private static final String TAG = TestUnboundedService.class.getCanonicalName();

    // doit être codé, mais doit retourner  NULL
    @Override public IBinder onBind(Intent arg0)  {
        Log.d(TAG, "onBind()");
        return null;
    }

    @Override public void onRebind(Intent intent) {
        Log.d(TAG, "onRebind()");
        super.onRebind(intent);
    }
```

Services privés non liés (private unbounded services)

- Exemple suite (le Binder et son Service)

```
    @Override public boolean onUnbind(Intent intent) {
            Log.d(TAG, "onUnbind()");
            return super.onUnbind(intent);
    }

    @Override public void onCreate() {
            Log.d(TAG, "onCreate()"); super.onCreate();
    }

    @Override public void onDestroy() {
            Log.d(TAG, "onDestroy()"); super.onDestroy();
    }

    @Override public int onStartCommand(Intent intent, int flags, int startId) {
            Log.d(TAG, "onStartCommand()");
            // DO SOMETHING ....
            return super.onStartCommand(intent, flags, startId);
    }
}
```

Services privées liés (private bounded services)

- Exemple (à partir du service non lié)
- Côté serveur (service)
  - Ajout du gestionnaire du binder
  - Comme classe interne de la classe du service

```
public class TestBoundedServiceBinder extends Binder {
    private final String TAGB = TestServiceBinder.class.getCanonicalName();

    public TestBoundedService getService() {
        Log.d(TAGB, "getService()");
        return TestBoundedService.this;
    }
}
```

Services privés liés (private bounded services)

- Exemple côté service (Dummy Test Service)
- Rappel: on a souvent l'habitude de mettre la classe de Binder en classe privée du service

```
public class TestBoundedService extends Service {
    private static final String TAG = TestBoundedService.class.getCanonicalName();
    private final IBinder binder = new TestBoundedServiceBinder();

    // ON PEUT METTRE ICI LA CLASSE TestBoundedServiceBinder

    // NE DOIT PAS RENVOYER NULL
    @Override public IBinder onBind(Intent arg0)  {
        Log.d(TAG, "onBind()");
        return binder;
    }

    @Override public void onRebind(Intent intent) {
        Log.d(TAG, "onRebind()");
        super.onRebind(intent);
    }
```

Services privés liés (private bounded services)

- Exemple côté service suite (le Binder et son Service)

```
    @Override public boolean onUnbind(Intent intent) {
            Log.d(TAG, "onUnbind()");
            return super.onUnbind(intent);
    }

    @Override public void onCreate() {
            Log.d(TAG, "onCreate()"); super.onCreate();
    }

    @Override public void onDestroy() {
            Log.d(TAG, "onDestroy()"); super.onDestroy();
    }

    @Override public int onStartCommand(Intent intent, int flags, int startId) {
            Log.d(TAG, "onStartCommand()");
            return super.onStartCommand(intent, flags, startId);
    }
}
```

Services privés liés (private bounded services)

- Exemple (à partir du service non lié)
- Côté client
  - Instanciation du gestionnaire de connexion
  - Classe anonyme de callback du côté client

```
ServiceConnection mConnection = new ServiceConnection() {
    public void onServiceDisconnected(ComponentName name) {
        Log.d(TAG, "Service disconnected");
        testService = null;
    }
    public void onServiceConnected(ComponentName name, IBinder service) {
        Log.d(TAG, "Service connected");
        testService = ((ServiceBinder)service).getService();
        // TestService methods are reachable via testService instance
    }
};
```

Services privés liés (private bounded services)

- Exemple (à partir du service non lié)
- Côté client
  - Ajout du code de connexion

```
public void startTestService(View v) {
    Log.d(TAG, "starting TestService");
    Intent serviceIntent =
        new Intent(MainActivity.this, TestBoundedService.class);
    bindService(serviceIntent, mConnection, Context.BIND_AUTO_CREATE);
}
```

Rendre un Service PUBLIC via Messenger

- Rappel: Ne pas mettre de « : » au début du nom
- Rappel: peut nécessiter l'utilisation des IPC
  - ne pas confondre avec les Intents
- La création d'un service public non lié
  - nécessite un code complexe pour onCreate, ...
  - onBind renvoie toujours null
  - Voir https://developer.android.com/guide/components/services.html#ExtendingService
- La création d'un service public lié est plus simple
- Communication unidirectionnelle via messenger
  - Soit service vers activité soit activité vers service

Rendre un Service PUBLIC via Messenger

- Méthode pour qu'un service envoie des données à une activité
  - Unidirectionnel, raisonnement « à l'envers »
  - Création d'un « handler » de message chez l'activité
    - Sous classe de « Handler » qui implémente « handleMessage »
  - Pour communiquer, création d'un Messenger qui sera envoyé au service
    - Constructeur « Messenger » qui a pour paramètre le « Handler »
    - En faire une variable membre de la classe
    - Lie définitivement le « Messenger » au « Handler »
  - Envoyer le Messenger au service via un Intent
    - Messenger parceable, utiliser « putExtra »
  - Utilisation de la méthode « send » du Messenger par service

Rendre un Service PUBLIC via Messenger

- Méthode pour qu'une activité envoie des données à un service
  - Unidirectionnel, raisonnement « à l'envers »
  - Création d'un « handler » de message chez le service
    - Sous classe de « Handler » qui implémente « handleMessage »
  - Pour communiquer, création d'un Messenger qui sera envoyé à l'activité
    - Constructeur « Messenger » qui a pour paramètre le « Handler »
    - En faire une variable membre de la classe
    - Lie définitivement le « Messenger » au « Handler »
  - Envoyer le Messenger à l'activité
    Via OnBind() pour un service lié (bounded)

```
@Override public IBinder onBind(Intent intent) {
        return myMessenger.getBinder();
}
```

  - Utilisation de la méthode « send » du Messenger par l'activité

Rendre un Service PUBLIC via Messenger

- Attention une communication bidirectionnelle via un Messenger implique un handle de chaque côté (un pour l'activité et un pour le service) !!
- Un exemple sur les différents cas possibles : Création du service public lié avec récepteur de message
- Rappel:
  - On a souvent l'habitude de mettre la classe de Binder en classe privée du service
  - On a souvent l'habitude de mettre la classe de handler en classe privée du service

```
public class MyMessengerService extends Service {

    // on peut mettre ici la classe MyMessageHandler


    final Messenger myMessenger =
                new Messenger(new MyMessageHandler());

    public IBinder onBind(Intent intent) {
            return myMessenger.getBinder();
      }
}
```

Rendre un Service PUBLIC via Messenger

- Création d'un Handler de réception des messages qui sera utilisé par le service

```
class MyMessageHandler extends Handler {
    public static final int MSG_SAY_HELLO = 1;

    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
            case MSG_SAY_HELLO:
                Toast.makeText(getApplicationContext(), "hello!",
                    Toast.LENGTH_SHORT).show();
                break;
            default:
                super.handleMessage(msg);
        }
    }

}
```

Rendre un Service PUBLIC via Messenger

- Intanciation d'un « Message »
  - Préférer l'utilisation de « Message.obtain() » que le constructeur
  - Injection des données via un bundle grâce à « setData() »
    - ↙ Voir plus loin la variable « Bundle »
- Utilisation de sendMessage(Message m) sur l'instance de Messenger

Rendre un Service PUBLIC via fichier AIDL

- AIDL est requis si plusieurs clients (ie différentes applications) accèdent à votre service (ie pour gérer le multithreading).
  - Si pas besoin d'appel concurrent entre application(s) et le service → binder
  - Si besoin d'IPC mais pas de multithread → Messenger
- Comprendre la notion de service lié pour utiliser AIDL

Rendre un Service PUBLIC via fichier AIDL

- AIDL ? Kesako ? Euh du RMI pour android …
- Méthode:
  - Définition de l'interface du service
  - Création du service & Implémentation de l'AIDL (encapsulé dans un stub)
  - Création de la connexion (ServiceConnection) pour le bind du service
  - Lier (bind) le service à l'activité
  - Invocation des méthodes
- ATTENTION LE DEMARRAGE EST ASYNCHRONE
- Définition de l'interface du service (à la RMI)

```
interface IRemoteItf {
    int getPid();
}
```

Rendre un Service PUBLIC via fichier AIDL

- Définition du service (suite) – classe privée souvent

```
public class AidlService extends Service {
    public AidlService() {}
    IRemoteItf.Stub mIRemoteItf = new IRemoteItf.Stub() {
        public int getPid()
            { return android.os.Process.myPid(); }
    };
    @Override
    public IBinder onBind(Intent intent) {
        Log.d(AidlService.this.getPackageName(), "onBind");
        return mIRemoteItf;
    }
}
```

Rendre un Service PUBLIC via fichier AIDL

- Création de la connexion (ServiceConnection) pour le bind du service – côté client

```
// Variable membre contenant la référence sur le service externe
IRemoteItf mService=null;

// communication asynchrone, création de callback
private ServiceConnection mConnection =
    new ServiceConnection() {

        // callback lors du démarrage du service ...
        public void onServiceConnected
            (ComponentName className, IBinder service) {
                // ON SAUVEGARDE l'ITF sur le service externe
                mService = IRemoteItf.Stub.asInterface(service);
            }

        // callback lors du CRASH du service (PAS A LA DECONNEXION !)
        public void onServiceDisconnected(ComponentName className) {
            ... do something ...
        }

};
```

Rendre un Service PUBLIC via fichier AIDL

- Lié le service à l'activité

```
Intent ibs = new Intent (MainActivity.this,AidlService.class);
bindService(ibs, mConnection, Context.BIND_AUTO_CREATE);
```

- La variable mService sera mise à jour par la méthode onServiceConnected ()
- Invocation des méthodes

```
mService.getPid()
```

Services utilisant des « intents »

- Sous classe de IntentService
- Pour une MAJ de la GUI
  - Il faut transmettre une référence sur Activity
  - Traitement séquentiel dans onHandleIntent
  - Ne peut être interrompu (Attention au boucles infinies / interblocages)
  - Peut utiliser thread / AsyncTask pour éviter les blocages

Services utilisant des « intents »

- Exemple

```
public class HelloIntentService extends IntentService {

    // Constructeur requis et doit appeler super IntentService(String)
    public HelloIntentService() {
        super("HelloIntentService");
    }

    // Méthode appelé par le thread de l'intent de l'appelant
    // A la fin de la méthode le service s'arête → utilisation d'une AsyncTask
    @Override
    protected void onHandleIntent(Intent intent) {
        // travaille …
        long endTime = System.currentTimeMillis() + 5*1000;
        while (System.currentTimeMillis() < endTime) {
            synchronized (this) {
                try {
                    wait(endTime - System.currentTimeMillis());
                } catch (Exception e) { }
            }
        }

    }
}
```

Les Tâches asynchrones : Description

- Dans une application seul le thread UI peut mettre à jour l'interface graphique
  - Ce thread doit répondre rapidement sinon blocage
  - On ne l'utilise pas pour faire du calcul intensif
- Pour les traitements longs 2 possibilités
  - Création d'un nouveau thread qui va effectuer les calculs
    - Attention! Seul le thread UI peut faire les mises à jour de la vue…
    - …il faut trouver un moyen de le forcer à faire la mise à jour
  - Utilisation des tâches asynchrones
    - Mécanisme dédié conçu pour ce genre de problèmes
    - Proches des services « oneshot » (non liés) avec Intent
    - Lié à l'activité qui l'a créé
    - Plus simple qu'un service

Les Tâches asynchrones : Description

- Manipulation facile du thread UI
  - Effectue des opérations en arrière plan
  - Diffuse le résultat sur l'interface graphique sans avoir besoin de gérer des handlers
- Ne doit pas remplacer les threads
  - Il s'agit d'une facilité d'écriture
  - Ne doit être utilisée que pour les tâches de quelques secondes
- Défini via 3 paramètres génériques
  - Param: le type du paramètre envoyé à la tâche pour le calcul
  - Progress: l'unité d'avancement utilisée pour la notification
  - Result: le type du résultat de la tâche asynchrone

Les Tâches asynchrones : Description

- Déclaration de la classe

private class MyFirstAsyncTask extends AsyncTask
<Type-Params, Type-Progress, Type-Return>

- Définition des méthodes (protected)
  - onPreExecute(), invoquée par la thread UI avant l'éxecution de la tâche. On la configure ici.
  - doInBackground(Params...), invoquée sur la thread d'éxecution immédiatement après onPreExecute(). Le coeur de la tâche. Les paramètres sont récupérés ici (par ex. une liste d'URL). Cette méthode peut utilisé publishProgress(Progress...) pour publier la progression.
  - onProgressUpdate(Progress...), invoquée par la thread UI après un appel à publishProgress(Progress...). Le timing est indéfini. Permet de mettre à jour l'affichage de la progression (ProgressBar, %, ...)
  - onPostExecute(Result), invoquée par le thread UI après la fin de l'exécution doInBackground(Params...). Le résultat de doInBackground(Params...) est passé comme paramètre cette méthode

Les Tâches asynchrones : Description

- Ces méthodes ne doivent pas être invoquées.
  Seule la méthode execute(Params) doit être invoqué par le client
- Classe de définition de la tâche
  - Si vous devez intervenir sur l'UI, vous devez fabriquer un constructeur avec Context ou utiliser getApplicationContext()

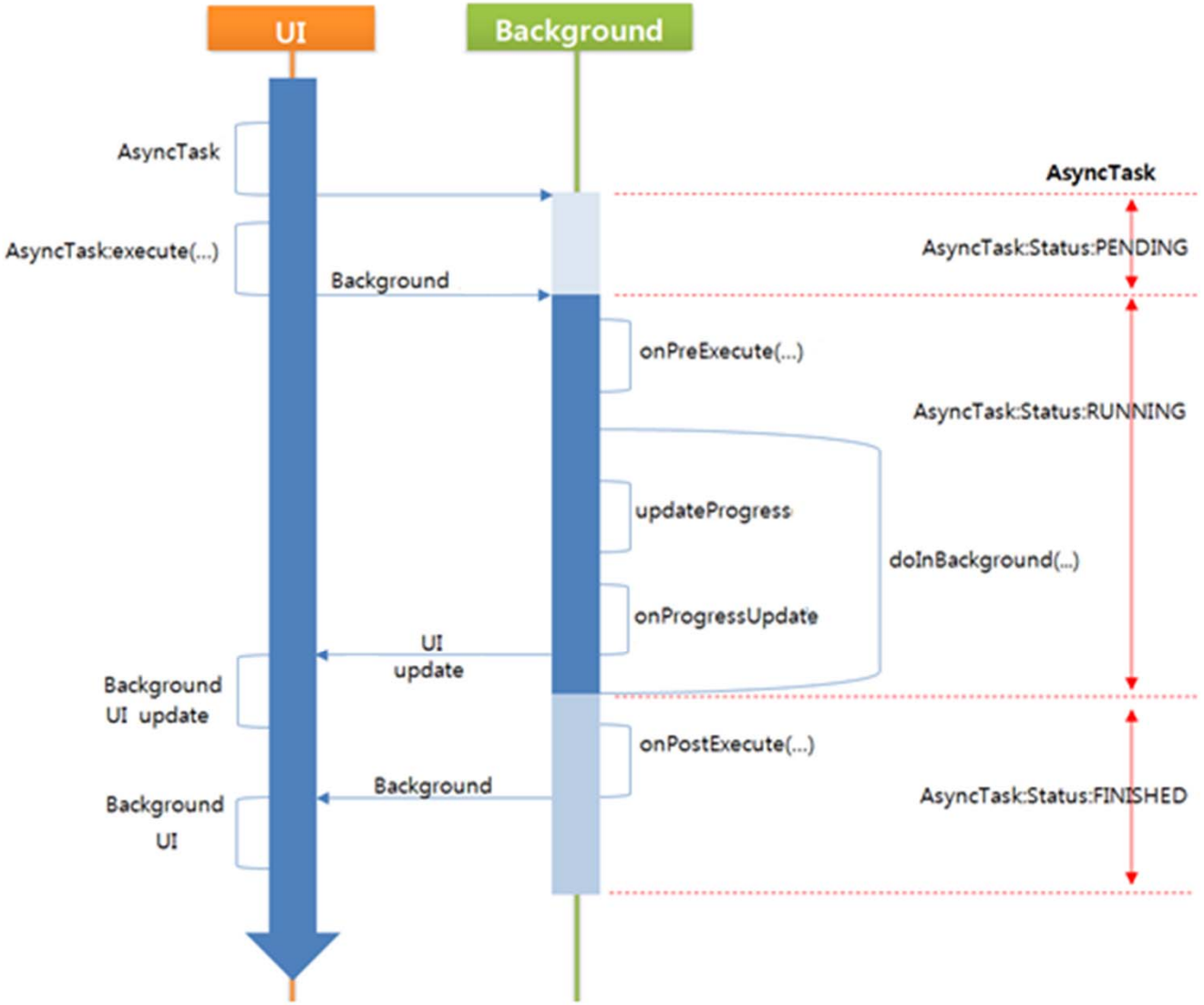
```
public class DummyAsyncTask extends AsyncTask<Integer, Short, Long> {
    // constructeur(s)
    // méthodes onPreExecute, doInBackground, ...
}
```

- Invocation dans l'activité

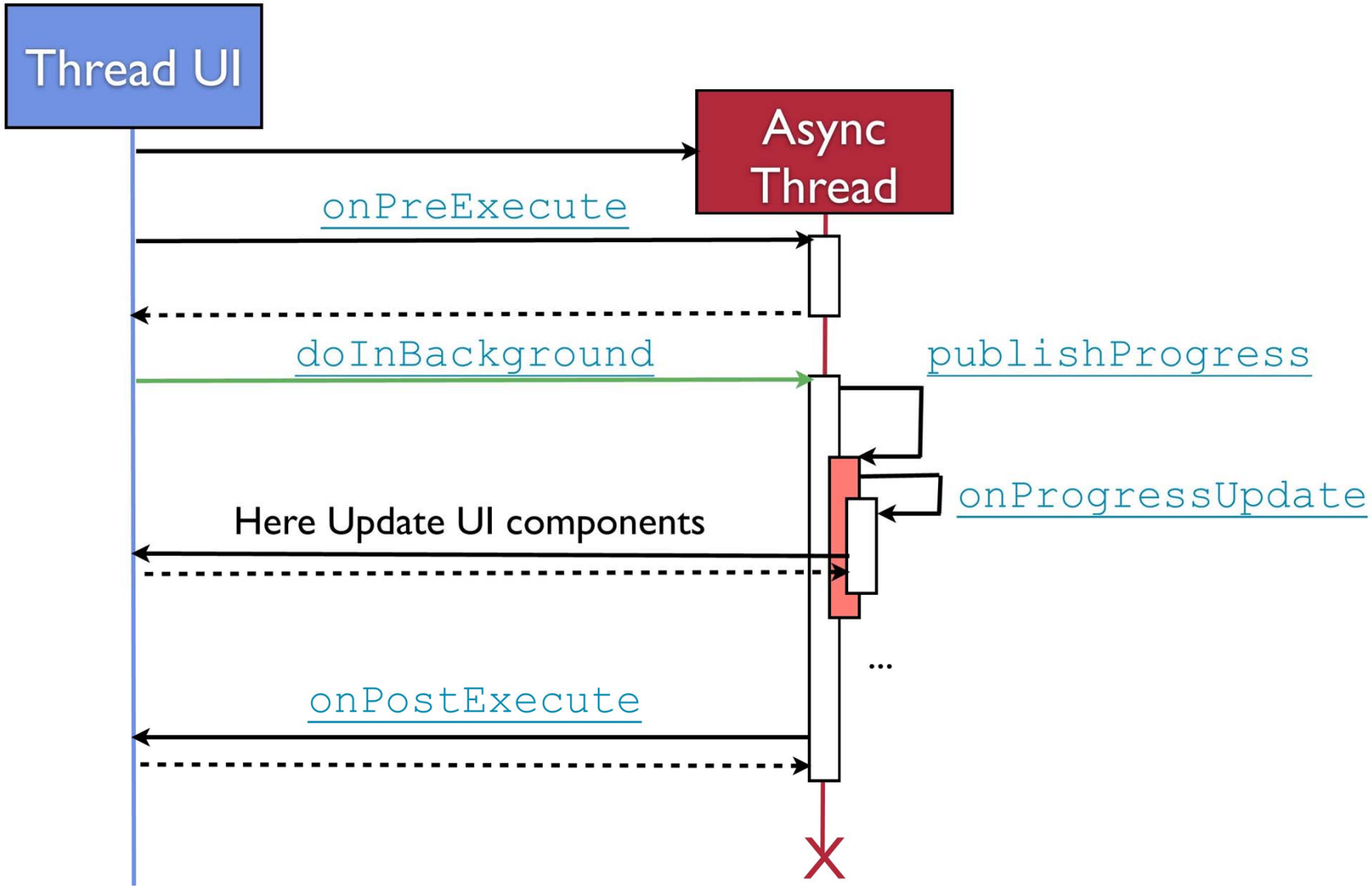
```
DummyAsyncTask dat = new DummyAsyncTask();
dat.execute (1,-4,3,5,20,10,-20,45,67,223,345,23,234,345,23);
```

Les Tâches asynchrones : Description

Programmation Concurrente



Les Tâches asynchrones : Description



Jobs : Description

- Package « android.app.job »
- API > 20 (RECENT), Lollipop !
- Différent de AsyncTask
  - « AsyncTask » gestion de l'exécution d'une tâche dans un thread différent
  - « JobScheduler » permet de programmer (schedule) une tâche (JobService) selon des contraintes
  - JobService s'exécute dans le MEME thread UI
  - JobService doit déléguer une tâche longue à un autre thread
  - On peut déléguer à / encapsuler une AsyncTask !
- Utilisation des Jobs:
  - Définition d'une classe puis instance de JobService
  - Définition d'une classe puis instance de JobInfo (JobService + contraintes)
  - Utilisation du JobScheduler avec le JobInfo

    Context.getSystemService(Context.JOB_SCHEDULER_SERVICE)

Jobs : Description

- Les infos sur le job sont contenu dans le paramètre JobParameters
  - Pour transmettre des paramètres « .setExtras » sur le « JobInfo »
  - Définition de contraintes d'exécution
    - Réseau, Mémoire, …
  - Association à une instance de JobService

```
JobInfo.Builder jobInfoBuilder = new JobInfo.Builder ( myJobServiceId , myJobService );
jobInfoBuilder.setMinimumLatency ( delay * 1000 );
jobInfoBuilder.setExtras ( parametresDuJob );
...
```

- Soumission du JobInfo au scheduler de Job

```
JobScheduler jobScheduler = (JobScheduler)
        getApplication().getSystemService(Context.JOB_SCHEDULER_SERVICE);
JobInfo myJobInfo = jobInfoBuilder.build();
jobScheduler.schedule(myJobInfo);
```

Java Threads : définitions

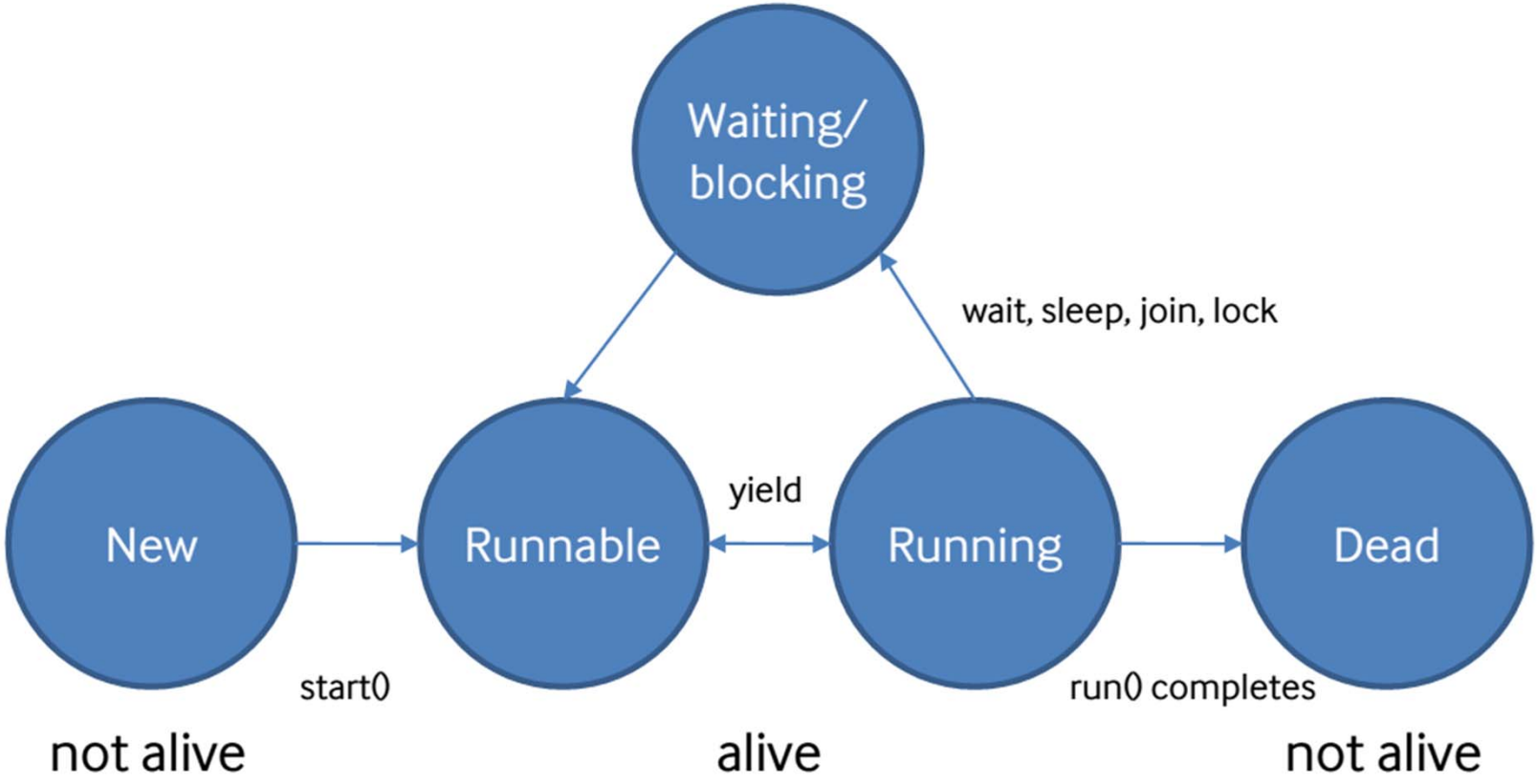
- Quasi obligatoire pour les applications Androids
  - Rappel: Ne pas utiliser le thread UI pour les calculs, MAJ des données, etc..
  - Même Modèle et propriétés que les threads Java
  - http://docs.oracle.com/javase/tutorial/essential/concurrency/
  - “Java Concurrency in Practice” par Brian Goetz et Tim Peierls
  - “Effective Java, 2nd Ed” par Josh Bloch
  - “The Art of Multicore programming” par M. Herlihy et N. Shavit

Java Threads : définitions

- Pour les threads :
  - Peu de garanties
  - ATTENTION: Lorsque plusieurs Thread peuvent travailler ou être réactivées, le choix ne dépendra jamais du programmeur ...
  - Création : étendre java.lang.Thread ou implémenter l'interface Runnable
  - Démarrage : Appel de la méthode run()
    → Doit être implémenter dans la classe bien évidemment ....
  - Mort : A la fin de la méthode run ...
  - Une fois le thread démarré, il ne pourra être redémarré
  - Utilisation des méthodes habituelles sleep(), yield(), join(), setPriority(), wait(), notify(), notifyAll()

Java Threads : cycle de vie

Spécificité PPM



Java Threads : la version simple

- Vous pouvez aussi utiliser AsyncTask (plus simple)
- Une façon simplifiée pour faire exécuter un code complexe par ex sur un click de bouton :

```
public void onClick(View v) {
        new Thread(new Runnable() {
                public void run() {
                        // ajouter le code ici
                }
        }).start();
}
```

- Créer des Threads “à la mano”
  - Pb de multiplication des threads
- Il existe des gestionnaires de Threads
  - **Ils permettent l’éxécution de tâches AVEC OU SANS UN RESULTAT**
  - **Avec : l’interface est “Callable”. Sans : l’interface est “Runnable”.**
  - Utile pour créer un pool de Thread, et soumettre les tâches via une file d’attente
  - La classe qui gère les Threads est un Executor
  - La classe de factory pour les Executor s’appelle “Executors”

Java Threads : avancé

Spécificité PPM

- Executors fournit des méthodes static pour construire des gestionnaire de pool de Threads
  - Il renvoie des instances de “ExecutorService” ou une de ses sous-classes

```
ExecutorService es = Executors.newScheduledThreadPool(3)
```

- ExecutorService offre les méthodes pour démarrer les threads et gérer le pool de threads actifs
  - Méthode execute()
    - Prends une classe qui implémente Runnable
  - Méthode submit()
    - Prends une classe qui implémente Callable ou Runnable
    - Submit renvoie un objet Future pour obtenir le résultat ou tester la terminaison du calcul.
- Quelques rappels
  - Callable doit implémenter la méthode call()
  - Runnable doit implémenter la méthode run()
  - Les classes Callable/Runnable peuvent être définies en “Inner Class”

Java Threads : Synchronisation

- Synchroniser l'accès entre toutes les instances
  - Eviter car très couteux
  - Utiliser un bloc synchronized(x) avec une classe comme paramètre pas avec une instance
- Utilisation de synchronized(x) sur une méthode
  - La synchronisation se fait séparement sur chaque instance de l'objet
  - Eviter car coûteux
- Utilisation de synchronized(x) sur un bloc de code
  - Utiliser un objet o comme mutex pour synchroniser un code sur une instance
  - Utiliser une classe comme mutex pour synchroniser un code sur toutes les instances d'une classe
  - Il faut obligatoirement utiliser la classe avec les méthodes "static"
  - Permet de minimiser au maximum le code synchronisé
- "volatile" impose la propagation d'un écriture sur une variable
  - Limiter aussi son utilisation. Préférer l'utilisation du syncrhonized()

Spécificité PPM

Bundle

Préférences

Fichiers

SQLite

Persistance : Les voies possibles

- Persistance
  - Utilisation de la variable Bundle
    - ✔semi-persistance ?
    - ✔persistance lors de crash de l'application ou d'une rotation
  - Utilisation des préférences
  - Par fichier interne/externe (dans « cache » / dans « data »)
  - Par Content Providers
    - ✔Parse de fichiers XML, JSON
    - ✔Réseau
  - Base de données (sqlite)

Persistance : Variable « Bundle »

- Persistance « temporaire » / « volatile »
  - **Lors d'un restart par l'OS**
  - **Lors d'une rotation**
  - **Lors d'un changement de configuration**
    - ✔ **public void onConfigurationChanged (Configuration newConfig)**
- Utilisation de la variable Bundle (lors d'une rotation)

  protected void onSaveInstanceState(Bundle state)

  public void onRestoreInstanceState(Bundle state)
- Exécutées dans le cycle de vie :
  - ✔ onResume / IDLE / onSave (par l'OS) / onPause / onStop /onDestroy
  - ✔ onCreate / onStart / onRestore / onResume / IDLE

Persistance : Variable « Bundle »

- Ne pas oublier lors de la création de l'activé principale (MainActivity) de reinstancier les variables du Bundle
- Bundle : savedInstanceState

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (savedInstanceState != null) {
        b = savedInstanceState.getStringArray("MyArray");
        // Do here for resetting your values which means state
        // before the changes occured.
    } else{
        // default..
    }
    Here you do general things.
}
```

Persistance : Variable « Bundle »

- Utilisation de l'objet Bundle
- Lors de la suspension ou avant une rotation

```
//Bundle outState is never null when the OS kill the app
protected void onSaveInstanceState(Bundle outState) {
      super.onSaveInstanceState(outState);
      outState.putString("hello", "message a sauvegarder");
}
```

- Lors de la réactivation ou après une rotation

```
protected void onRestoreInstanceState(Bundle savedInstanceState) {
super.onRestoreInstanceState(savedInstanceState);

....

if (savedInstanceState == null) return;
String seq = savedInstanceState.getString("hello");
}
```

- via un MAP (clé,valeur)
  - TYPES SIMPLES
    - Habituellement key est une chaîne sous forme « a.b.c.d »
    - De nombreuses méthodes get/put pour différents types simples, Vecteur, etc …
  - Types COMPLEXES: soit Serializable soit Pacelable

Persistance : Sauvegarde de Préférences

- Les préférences
  - Equivalent à fichier .INI/CFG ou à un morceau de base de registre sous Windows
  - Stocker des options, paramètres …
  - Certains s'en serve pour stocker de données
  - Une application peut en utiliser plusieurs
- Données primaires via MAP (clé,valeur)
- On peut surveiller les préférences par un listener
  - registerOnSharedPreferenceChangeListener()
- Mode d'accès possibles :
  - MODE_PRIVATE : seulement accessible par l'activité
  - MODE_WORLD_READABLE : en lecture pour toutes les activités (A EVTIER)
  - MODE_WORLD_WRITEABLE : en écriture pour toutes les activités (ENCORE PLUS A EVITER – A BANNIR)

Persistance : Sauvegarde de Préférences

- ## Ecriture / Création des préférences

```
SharedPreferences settings = getSharedPreferences
                        ("preferences_file", Context.MODE_PRIVATE);
SharedPreferences.Editor editor = settings.edit();
editor.putInt("counter", current);
editor.commit();
```

- ## Lecture des préférences

  - **Conseil pour les valeurs par défaut : penser aux « public static final »**

```
SharedPreferences settings = getSharedPreferences
                        ("preferences_file", Context.MODE_PRIVATE);
// -42 Valeur par défaut si la clef « counter » n'est pas déf.
current = settings.getInt("counter", -42);
```

- ## Utilisation possible de

```
PreferenceManager.getDefaultSharedPreferences();
```

Persistance : Fichiers

- Le stockage interne / externe
  - Fichier de sauvegarde
- interne, externe ?
  - interne : la mémoire flash où s'exécute Android (ie NON AMOVIBLE)
  - externe : la mémoire flash sur un média externe (sdcard) – ie AMOVIBLE
- Différence :
  - L'une est toujours présente (interne)
  - L'autre peut être retirée (externe)

Persistance : Fichiers sur mémoire **INTERNE**

- **Rappel : Utilisation de répertoires prédéfinis !**
  N'écrivez pas n'importe où !!
- Méthode getFilesDir() de Context
- Constantes android.os.Environment.DIRECTORY_*
- Eviter des chemins absolus.
- Utilisation de la mémoire **interne**
  - Eviter les abus (pas de cache)
  - Stockage de préférence, données critiques
  - Clefs de cryptage
  - Pas de vérification de présence à faire

```
String toSave = "Hello the world";
FileOutputStream fos = openFileOutput("hello.txt", Context. MODE_PRIVATE);
fos.write(toSave.getBytes());
fos.close();
```

Persistance : Fichiers sur mémoire **EXTERNE**

- **Rappel : Utilisation des répertoires prédéfinis !**
  N'écrivez pas n'importe où !!
- Permissions nécessaires:
  - android.permission.READ_EXTERNAL_STORAGE
  - android.permission.WRITE_EXTERNAL_STORAGE
- Utilisation de la mémoire **externe (API 19).**
  - Méthode getExternalCacheDirs() d'une instance de « Context »
  - Méthode getExternalFilesDir() d'une instance de « Context »
  - Méthode getExternalMediaDir() d'une instance de « Context »
  - Méthodes comme Environment.getExternalStorageDirectory()

```
// Remarquez les tests inverses .. Equals sur les constantes
String state = Environment.getExternalStorageState();
if (Environment.MEDIA_MOUNTED.equals(state)) {
        // READ - WRITE on file system
} else if (Environment. MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        // READ only on file system
} else {
        // No access to MEDIA
}
```

Persistance : ContentProvider

Persistance

Persistance : Base de données

- Utilisation d'une classe qui étend SQLiteOpenHelper
  - Définit la méthode "onCreate()" qui sert à créer la DB si elle n'existe pas
  - Définit la méthode "onUpgrade()" qui sert à mettre à jour une ancienne version de la DB
    - ✔ Souvent la méthode upgrade contient un drop des tables SQL ce qui est peu professionel
  - Définit la version actuelle de la DB
  - Penser à appeler la méthode super dans le constructeur

Persistance : Base de données

Persistance

```
public class MyAppDbHelper extends SQLiteOpenHelper {

    // La version de  la DB doit changer si les schémas changent.
    public static final int DATABASE_VERSION = 1;
    public static final String DATABASE_NAME = "monfichiersqlite.db";

    // requete SQL pour la creation de la DB
    private static final String SQL_CREATE_ENTRIES = " …  ";
    private static final String SQL_DELETE_ENTRIES = " …  ";

    public MyAppDbHelper(Context context) {
        // Appel du super pour créer le fichier sur disque
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    public void onCreate(SQLiteDatabase db) {
        db.execSQL(SQL_CREATE_ENTRIES);
        …
    }

    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Le plus souvent, par facilité, on jette l'ancienne db, pour la recréer
        db.execSQL(SQL_DELETE_ENTRIES);
        onCreate(db);
    }

    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        onUpgrade(db, oldVersion, newVersion);
    }
}
```

Persistance : Base de données

- Ensuite on utilise le helper pour obtenir un objet représentant la base de données

```
MyAppDbHelper mDbHelper = new MyAppDbHelper(getContext();
SQLiteDatabase db = mDbHelper.getWritableDatabase();
```

- Définition d'un contrat pour la base
  - Contiendra des « inner class » pour chaque table de la base !!!

```
public final class MyAppDbContract  {
        // Pour éviter une instanciation non souhaitée de la classe (constructeur privé)
        private MyAppDbContract() {}

        // On peut mettre la version de la database et son nom dans le contrat
        …

        // Inner classes pour définir le contenu de chaque table
        public static abstract class MaPremiereTableEntry implements BaseColumns {
                public static final String TABLE_NAME = “premieretable";
                public static final String COLUMN_NAME_ENTRY_ID = "entryid";
                public static final String COLUMN_NAME_TITLE = "title";
                public static final String COLUMN_NAME_SUBTITLE = "subtitle";
                ...
                // on peut aussi mettre les requêtes SQL de suppression / création
        }

}
```

Persistance : Base de données

- Possibilité d'utiliser
  - query(), rawquery() de la classe SQLiteQueryBuilder
  - execSQL de l'objet SQLiteDatabase
- Insertion

```
SQLiteDatabase db = mDbHelper.getWritableDatabase();
ContentValues values = new ContentValues();
values.put (MyAppDbContract.MaPremiereTableEntry.COLUMN_NAME_ENTRY_ID, id);
values.put (MyAppDbContract.MaPremiereTableEntry.COLUMN_NAME_TITLE, title);
values.put (MyAppDbContract.MaPremiereTableEntry.COLUMN_NAME_CONTENT,
content);

// Insert the new row, returning the primary key value of the new row
long newRowId;
newRowId = db.insert(
        MyAppDbContract.MaPremiereTableEntry.TABLE_NAME,
        MyAppDbContract.MaPremiereTableEntry.COLUMN_NAME_NULLABLE,
        values);
```

Persistance : Base de données

- Sélection

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();
// Définir les colonnes utilisées dans la requête
String[] projection = {
      MyAppDbContract.MaPremiereTableEntry._ID,
      MyAppDbContract.MaPremiereTableEntry.COLUMN_NAME_TITLE
}
String sortOrder =
      MyAppDbContract.MaPremiereTableEntry.COLUMN_NAME_TITLE + " DESC";

Cursor c = db.query(
      MyAppDbContract.MaPremiereTableEntry.TABLE_NAME,  // La table sur laquelle faire la requete
      projection,                       // Les colonnes à extraire (SELECT '…')
      selection,                        // les colonnes concernées par la clause WHERE
      selectionArgs,                    // les valeurs pour la clause WHERE
      null,                             // ne pas grouper par colonne
      null,                             // ne pas filtrer par groupe de colonnes
      sortOrder                         //  chaine de tri (ORDER BY …)
);
cursor.moveToFirst();

long itemId = cursor.getLong(

   cursor.getColumnIndexOrThrow(MyAppDbContract.MaPremiereTableEntry._ID)

);
```

Persistance : Base de données

- Effacement

```
SQLiteDatabase db = mDbHelper.getWritableDatabase();
// Definir la partie 'where' de la requête, le « ? » sera remplace par les
// éléments de selectionArgs
String selection =

    MyAppDbContract.MaPremiereTableEntry.COLUMN_NAME_ENTRY_ID

    + " LIKE ?";
// les valeurs qui remplaceront le « ? »dans la chaîne « selection »
String[] selectionArgs = { String.valueOf(rowId) };
// exécuter la requête
db.delete(table_name, selection, selectionArgs);
```

- Pour faire du batch, utiliser sur SQLiteDatabase :
  - beginTransaction / endTransaction / setTransactionSuccessful / execSQL

Persistance : Base de données, API persistance

- Pas d'équivalent JPA par défaut officiel google
- Outils ORM
  - **OrmLite (Open source license ISC)**
    http://ormlite.com/
  - **SugarORM**
    https://satyan.github.io/sugar/
  - **GreenDAO (license Apache Version 2.0)**
    http://greendao-orm.com/
  - **Active Android (license Apache Version 2.0)**
    http://www.activeandroid.com/
- Choisissez le votre

JSON

XML

JSON - structure

- **Idée**
  - Représentation textuelle des objets Javascript
  - Objects reçus sous forme de strings
    - **Simple et direct en JavaScript**
      var someObject =
      { property1: value1, property2: value2, ... };
  - Largement utilisé dans d'autres langages
    - Le préférer au XML (plus léger)
    - Android a une classe interne nommée JSONObject pour gérer (parser, créer, écrire) le JSON
- type MIME pour le JSON sur un serveur web
  - RFC 4627 dit que le type JSON est "application/json"
  - Aucune implementation n'impose cela

Service Web utilisant JSON

Internet

- **Google APIs** - Recherche, traduction automatique, …
  - http://code.google.com/apis/customsearch/v1/overview.html
- **Yahoo APIs** - Recherche, voyages, yahoo answers
  - http://developer.yahoo.com/
- **Twitter APIs**
  - https://dev.twitter.com/
- **GeoNames APIs**
  - http://www.geonames.org/export/web-services.html
- **Flickr APIs**
  - http://www.flickr.com/services/api/
- **Facebook APIs**
  - https://developers.facebook.com/docs/unity/reference/current/Json
- **Des milliers d'autres**
  - http://www.programmableweb.com/apis/directory/1?format=JSON

Parser du JSON

Internet

- **Lecture du JSON**
  - **Utilise une chaine (String)**
  - **Instanciation d'un objet JSONObject**

http://developer.android.com/reference/org/json/JSONObject.html

```
# Just for testing, allow network access in the main thread
# NEVER use this is productive code
StrictMode.ThreadPolicy policy = new StrictMode.
            ThreadPolicy.Builder().permitAll().build();
StrictMode.setThreadPolicy(policy);

// Méthode à coder … Lecture dans un fichier, sur http, …
String input = getJSONString();
try {
      JSONObject json = new JSONObject(input);
      Log.i(ParseBugzillaActivity.class.getName(), jsonObject.toString())
} catch (Exception e) { … }
```

Parser du JSON

Internet

- **Lecture du JSON**
  - **Utilisation des objets HttpGet / HttpClient**

```
public String getJSONString() {
    StringBuilder builder = new StringBuilder();
    HttpClient client = new DefaultHttpClient();
    HttpGet httpGet = new
                HttpGet("https://bugzilla.mozilla.org/rest/bug?assigned_to=lhenry@mozilla.com");
    try {
        HttpResponse response = client.execute(httpGet);
        StatusLine statusLine = response.getStatusLine();
        int statusCode = statusLine.getStatusCode();
        if (statusCode == 200) {
            HttpEntity entity = response.getEntity();
            InputStream content = entity.getContent();
            BufferedReader reader = new BufferedReader(new InputStreamReader(content));
            String line;
            while ((line = reader.readLine()) != null) {
                builder.append(line); }
        } else {
            Log.e(ParseJSON.class.toString(), "Failed to download file");
        }
    } catch (Exception e) { … }
    return builder.toString();
}
```

Parser du JSON

- **Ecriture du JSON**
  - **Objet JSONArray pour les tableaux**
  - **Il existe une méthode put pour les objets JSON**

```
JSONObject object = new JSONObject();
try {
    object.put("name", "Jack Hack");
    object.put("score", new Integer(200));
    object.put("current", new Double(152.32));
    object.put("nickname", "Hacker");

    } catch (JSONException e)
        { … }
    System.out.println(object);
}
```

- **Combinable avec l'écriture des fichiers par ex. (voir persistance)**

Internet

Parser du JSON - Gson

- **Rappel de JSON**
  - Si le noeud JSON commence par *[*
    - ↙ Utilisation de *getJSONArray()*
  - Si le noeud JSON commence par *{*
    - ↙ Utilisation de *getJSONObject()*
- **Rien n'interdit l'utilisation de Gson**
  - Nécessite l'import / intégration de la librairie google au projet
  - Ajout au gradle

    dependencies { compile 'com.google.code.gson:gson:2.2.+' }

Parsers XML

- Les librairies xml disponibles sous Android
  - Elles sont **NATIVES**
  - Pour les détails des différentes approches
    - ✔ http://www.xmlpull.org/history/index.html
  - Document Object Model
    - ✔ Représentation des documents XML sous forme d'arbre
  - SAX (Simple API for XML)
    - ✔ Algorithme de parsing automatique de document XML
    - ✔ Association d'un code Java à chaque type de nœud
    - ✔ méthode ou réaction aux events
    - ✔ Push → La structure du document active du code java
  - XmlPull v1
    - ✔ Pull → Streaming API for XML
    - ✔ La structure des objets Java vont décider de la structure du XML

Internet

L10N – I18N pour android

Capteurs

Style de Programmation

Sécurité

L10N – I18N

- L10N ? I18N ? WTF ?
  - LocalizatioN (L10N), InternationalizatioN (I18N)
- Quelques liens
  - **Developer's Guide: Localization**
    http://developer.android.com/guide/topics/resources/localization.html
  - **Developer's Guide: Application Resources**
    http://developer.android.com/guide/topics/resources/
  - **Tutorial: Hello L10N**
    http://developer.android.com/resources/tutorials/localization/index.html
  - **Chapter: Localization**
    "*Android in Action*" by Ableson et al

L10N – I18N

- **Langage**
  - Utiliser les références de chaînes (strings.xml)
  - Fournir différentes version du fichier de chaînes
  - Utilisable aussi pour les sons, les couleurs, les images
    - Drapeaux, hymnes, …
  - Le fichier par défaut doit contenir toutes les chaînes
- Rappel :
  – Le répertoire “res” doit contenir toutes les version des images (xdpi, hdpi, xhdpi, …)
- **Best practices**
  - Tester toutes les combinaisons, spécialement les langages inattendus
  - Utiliser le layout editor des IDE pour les premiers tests (tailles, ...)

L10N – I18N

- Les répertoire “**res**” peut être ré-organisé pour gérer les langues
- Ajout d’un postfix aux sous répertoires de res
  - Un code pays spécifié par ISO 639-1
  - ISO 639-2 pour les codes pays a 3 lettres
  - http://en.wikipedia.org/wiki/ISO_639-1
- Possibilité de faire des traductions régionales
  - Répertoire “values- xx-rYY”
    xx code langage et YY code pays (sans casse)
    Note “r” c’est pour “region”.
  - Codes are specified by ISO 3166-1
    http://en.wikipedia.org/wiki/ISO_3166-1_alpha-3

L10N – I18N

- Exemple, le répertoire values (défaut)
  - “values-fr/strings.xml” pour le français
  - “layout-es/main.xml” pour l’espagnol
- Les répertoires sans les codes pays doivent contenir toutes les références
  - Chargement de toutes les valeurs des fichiers des répertoires sans code pays
  - Android remplace ensuite toutes les références définies dans les répertoires avec le code pays
    - ↙ C’est fonction de la configuration du système

L10N – I18N

- Si le fichier de langage est pas défini, android utilise donc le fichier par défaut
- Utilisation de la méthode getString()
  - Références des chaînes R.strings.xxxxx
- Forcer le changement de langue dans l'application

```
// code Langage iso
Locale locale = new Locale("es");
Locale.setDefault(locale);
Configuration config = new Configuration();
config.locale = locale;
context.getResources().updateConfiguration(config,null);
```

Capteurs (hors localisation)

- Utilisation du SensorManager

```
SensorManager sm = (SensorManager) context.getSystemService(Service.SENSOR_SERVICE);
```

- Android supporte plusieurs types de capteurs
  - Certains capteurs ne sont pas émulés (par ex. acceleromètre)

```
Sensor s = sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
List<Sensor> ls = sm.getSensorList(Sensor.TYPE_ACCELEROMETER);
```

- Quelques capteurs :
  - Sensor.TYPE_* ou (alias) Sensor.STRING_TYPE_*
  - TYPE_ACCELEROMETER
    TYPE_AMBIENT_TEMPERATURE
    TYPE_GRAVITY
    TYPE_GYROSCOPE
    TYPE_HEART_RATE
    TYPE_MAGNETIC_FIELD ....
- **Tout sauf la localisation. Pour la localisation utiliser le service:**
  - Service.LOCATION

Spécificité PPM

Capteurs (hors localisation)

- Possibilité d'ajouter un listener pour les capteurs
  - Deux types de listener
    - SensorEventListener, méthodes onSensorChanged et onAccuracyChanged
    - SensorEventListener2 depuis KitKat, ajoute une méthode onFlushCompleted
    - SensorListener deprecated …
- Utilisation de registerListener, unregisterListener
  - Utiliser une « classe anonyme » ou une « inner classes »

```
sm.registerListener(
    new SensorEventListener() {
        …
    },
    s, SensorManager.SENSOR_DELAY_NORMAL);
```

- Lorsque l'on enregistre (register) un listener sur un capteur, suivant la précision requise cela peut consommer beaucoup

Capteurs (hors localisation)

- onSensorChanged a pour paramètre un « SensorEvent »
- Description du contenu du SensorEvent en fonction du type de capteur l'ayant généré

http://developer.android.com/reference/android/hardware/SensorEvent.html

- Les champs importants :
  - Sensor: le champ pointant sur l'instance de l'objet représentant le capteur ayant généré l'événement (utile pour des listeners gérant de multiple capteurs)
  - Timestamp: date de génération de l'évènement (en nanosecondes)
  - Accuracy: Qualité de l'information
  - Values: tableau de float contenant les valeurs du capteurs (variable suivant le capteur)
- **Attention ce sont des données brutes !!!**
  - **Des calculs et des filtrages (passe haut, …) sont à prévoir.**

Capteurs - Localisation

- Pour la localisation, utiliser le LocationManager
- Il est impossible (officiellement) d'activer ou désactiver ce capteur programmatiquement. Seul l'utilisateur le peut.
  - Sauf si vous êtes « root »
- Il existe différents fournisseurs de localisation (manipulable via le manager):
  - GPS_PROVIDER, NETWORK_PROVIDER, PASSIVE_PROVIDER, FUSED_PROVIDER
- C'est le capteur le plus utilisé

```
LocationManager lm = (LocationManager)
                Context.getSystemService(Context.LOCATION_SERVICE);
boolean enabled = lm.isProviderEnabled(LocationManager.GPS_PROVIDER);

Criteria criteria = new Criteria();
provider = locationManager.getBestProvider(criteria, false);
Location location = locationManager.getLastKnownLocation(provider);
if (location != null) {
      Log.d (this.getClass().getName(),
                "Provider:"+provider+", Location:"+location);
}
```

Capteurs - Localisation

- En fait, l'activation de la géolocalisation se fait suivant des critères spécifiés par l'application
  - Création d'une instance de « Criteria »
  - Puis utilisation des méthodes set*
    - ✔ setAccuracy(int accuracy)
    - ✔ setAltitudeRequired(boolean altitudeRequired)
    - ✔ setPowerRequirement(int level)
- La récupération du meilleur provider par
    - ✔ getBestProvider(Criteria criteria, boolean enabledOnly)
    - ✔ getProvider(String name)

Capteurs - Localisation

- Comme pour les autres capteurs, il est possible d'installer un listener sur la géolocalisation.
  - On peut activer et désactiver les MAJ
  - Possibilité aussi d'utiliser des PendingIntent
  - requestLocationUpdates, removeUpdates
  - Ne porte pas les noms habituels de un/register
- Il est ensuite possible d'utiliser une classe classes « Geocoder » et « Address »
  - Permet l'association coordonnées-adresses
  - Attention : passe par google maps ...

Spécificité PPM

Code

- Références :
- Strong reference: Object obj = new Object();
  • SoftReference – soft references will be cleared before the JVM reports an out-of-memory condition (memory-sensitive cache)
  • WeakReference – gc frees weakly reachable object when it founds it, avoid memory leaks (e.g. http://en.wikipedia.org/wiki/Flyweight_pattern)
  • PhantomReference – is useful only to track the impending collection of the referring object. Must be used only with the ReferenceQueue class. Most often used for scheduling premortem cleanup actions in a more flexible way than is possible
  with the Java finalization mechanism

Syntaxe de codage

- Synatxe
  - Imports :
    - Trier les imports par ordre alphabétique
    - Regroupé par imports Android, puis librairies (com, junit, net, org), java/javax, puis imports applicatifs
  - Indentation
    - 4 espaces / 1 tabulation pour l'indentation des blocs
    - 8 espaces / 2 tabulations pour l'indentation de retour à la la ligne (wrap)
  - Nom de champs
    - Champs Non-public, non-static field commencent par "m"
    - Champs static commencent par "s"
    - Autres champs commencent avec un lettre minuscule
    - Champs membres static final sont en MAJUSCULE_AVEC_DES_SOULIGNES.
  - Commentaires : TODO, FIXME, XXX, Copyrights

- Programmation
  - Annotations, Logging (voir le début du module)
    http://source.android.com/submit-patches/code-style-guide
  - **Eviter les classes non statiques dans une activité si l'instance doit survivre à l'activité**
    → cela empêche le gc d'agir
  - **Privilégier les classes internes (inner class) sur les classes non statique**
    → cela permet au gc de garbage collecter les "inner classes" a partir de la classe encapsulante
  - **C'est à vous de faire le ménage pour les threads !!!**
  - **Bien séparer le code. Considérer les ressources et les objectifs avant d'utiliser un thread.**
    http://www.androiddesignpatterns.com/2013/04/activitys-threads-memory-leaks.html
  - Faire une classe outils (soit static soit singleton)

Programmation

- Performances :
  - Limiter au maximum les allocations d'objets et de mmoire
  - Le GC est lent (~x00 ms sur Android device)
    - Réduire le nombre d'objets pour que le GC travaille plus vite
    - i.e. StringBuffer vs String
  - Utiliser des types primitifs (int[] à la place de Integer[])
  - Eviter le boxing (float/Float)
  - Utiliser les méthodes prédéfinies
    - String.indexOf(), substring(), etc.
  - Manipuler les classes au lieu des interface lorsque c'est possible
    - Ex: HashMap au lieu de Map
    - Economise une indirection
  - Utiliser des variables / méthodes static (plus rapide à l'invocation)
  - Eviter les getter/setter (faire des champs publics sauf impossibilité)
  - Utiliser les versions de boucle de Java (for elt : inCollection) ...

Sécurité

Sécurité

- Application Sandboxée
  - Chaque application a un « user » spécifique
    - ↙ Possibilité de filtrer au niveau firewall par user
  - Chaque application a son propre processus
  - Chaque application a sa propre VM
  - Définition des droits associer à l'application
    - ↙ Utilisation du manifest

Gestion des applications