

Programmation Android

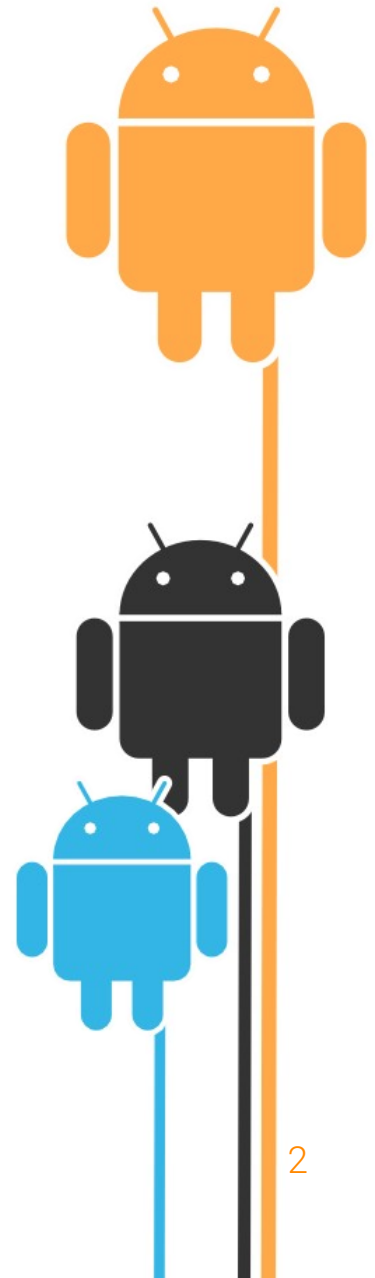
09 – WebServices

Yann Caron

ENSG
Géomatique

Sommaire - WebServices

- Généralités
- Interrogation
- Parcours (parsing)
- AsyncTask
- Mapping avancé : Jackson
- Côté serveurs



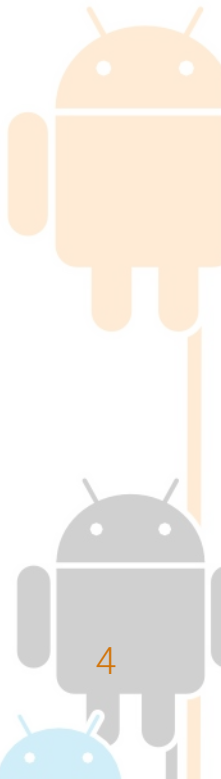
IN01 – Séance 09

Généralités



Généralités

- Assure la connexion et l'échange de données entre applications hétérogènes distribuées, généralement via le protocole HTTP
- Idée générale : utiliser une technologie existante, facilement interopérable et ouverte
- Deux grands types de WebServices :
 - ➔ WS-*
 - ➔ REST



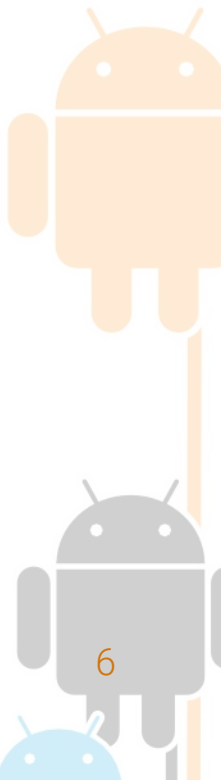
WS-* / REST

- WS-* ou RPC : Remote Procedure Call
 - ➔ Paradigme d'appel de méthodes à distance (accès aux objets via méthodes)
 - ➔ Transfert de données via accesseurs (Getter / Setter)
 - ➔ Bibliothèques serveur et cliente
- REST : Orienté données
 - ➔ Transfert des objets via attributs
 - ➔ Normalisation des URLs
 - ➔ Côté client, sa simplicité permet l'utilisation ou non d'un client



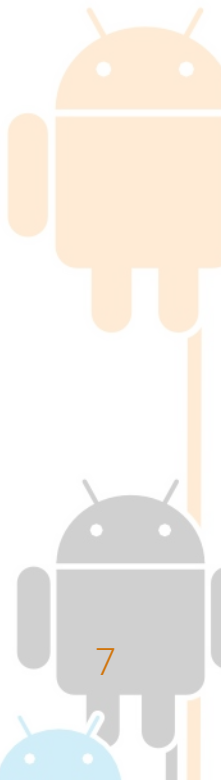
WS-*

- Ensemble de standards
 - ➔ SOAP (Simple Object Access Protocol) : échange de messages
 - ➔ WSDL (Web Service Description Language) : description du message
 - ➔ UDDI : L'annuaire
- Basé sur le modèle SOA (Service Oriented Architecture)

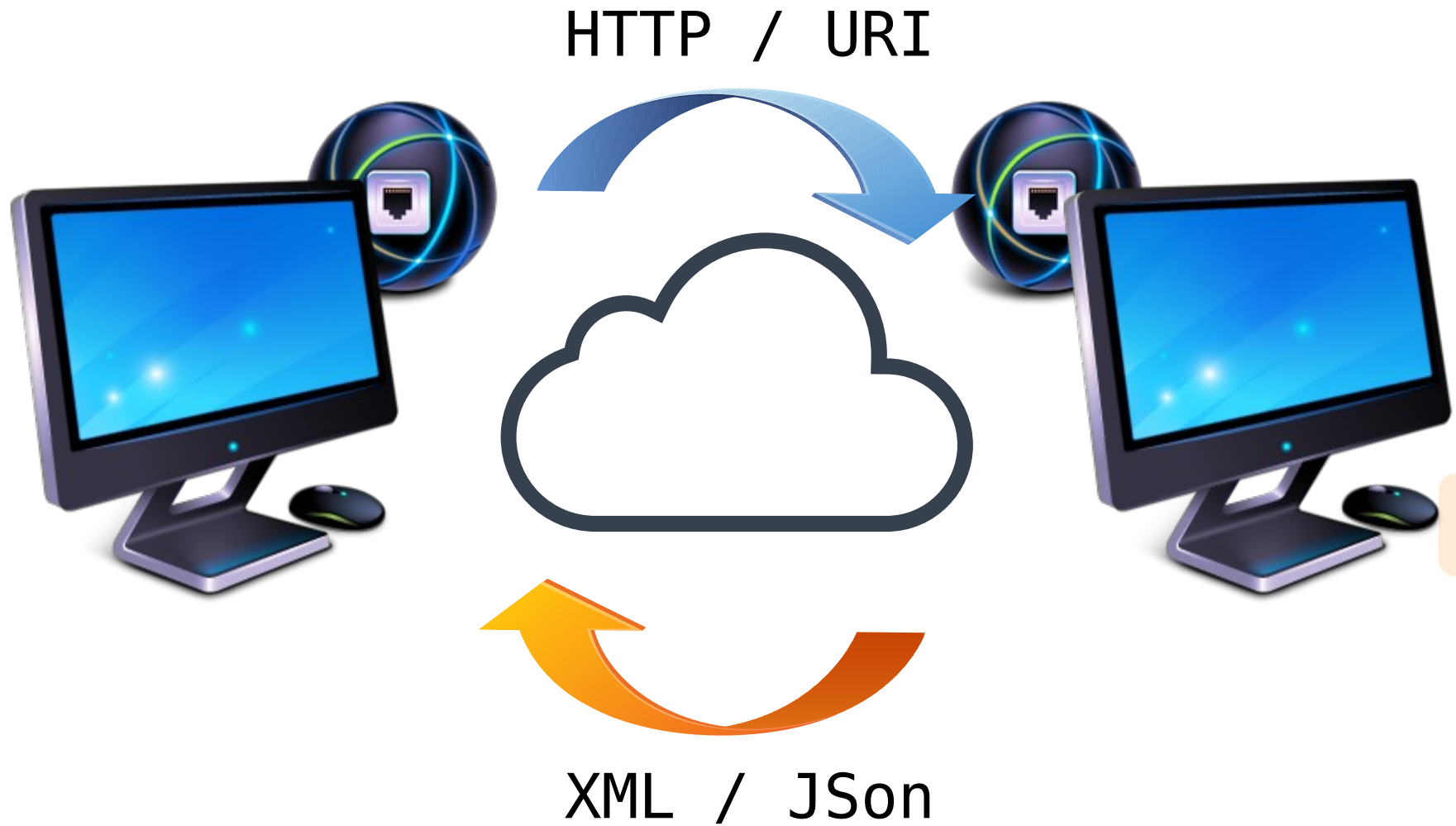


REST

- Representational State Transfer
- Formalisé par Roy Fielding en 2000 (lors de sa thèse de doctorat)
- Plutôt un style architectural qui respecte des règles
 - ➔ Client - Serveur
 - ➔ State less (n'entretient pas d'état client côté serveur)
 - ➔ Mise en cache
 - ➔ Interface uniforme (Une URI pour une ressource)
 - ➔ Architecture en couches
 - ➔ Code-on-demand (optionnel)



Architecture REST



XML

- Extensible Markup Language
- Langage à balises encadrés par des chevrons

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

JSon

- JavaScript Object Notation
- Sous-ensemble grammatical issu de JavaScript

```
{  
  "menu": {  
    "id": "file",  
    "value": "File",  
    "popup": {  
      "menuitem": [  
        { "value": "New", "onclick": "CreateNewDoc()" },  
        { "value": "Open", "onclick": "OpenDoc()" },  
        { "value": "Close", "onclick": "CloseDoc()" }  
      ]  
    }  
  }  
}
```

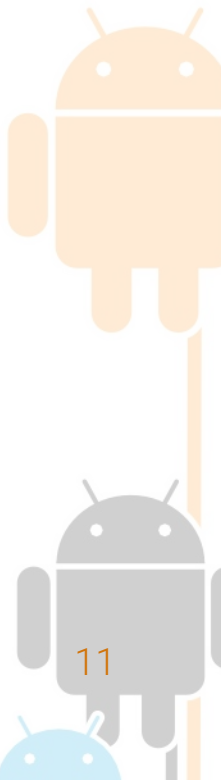
JSON vs XML

- Pour JSON :

- Moins verbeux, donc moins couteux en ressources
- Facile à apprendre
- Typage simple et connu
- Plus facile à parser

- Pour XML :

- Plus flexible (Syntax JSON rigide)
- Commentaires
- Validation (XML Schema, DTD)



Avantages / inconvénients

- Avantages :

- Interopérabilité et portabilité
- Technologie ouverte et connue
- Formats textes (XML / JSON) humainement lisibles
- Facilement transportable (port 80)

- Inconvénients

- Format volumineux et peu performant (cas du XML)



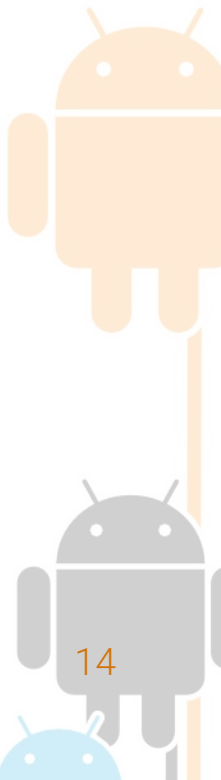
IN01 – Séance 09

Interrogation



Introduction

- Choix de l'architecture :: REST
- Choix du langage :: JSON
- Pourquoi ?
 - ➔ Natif, rien à installer
 - ➔ WebServices RPC nécessiterait des librairies, comme Ksoap2-android
 - ➔ RPC complexe à mettre en oeuvre



Android

- Utilisation des objets URL et HttpURLConnection


```
URL url = new URL("http://My URL");  
  
HttpURLConnection urlConnection = (HttpURLConnection)  
url.openConnection();  
  
InputStream in = new  
BufferedInputStream(urlConnection.getInputStream());
```



Lire le Stream

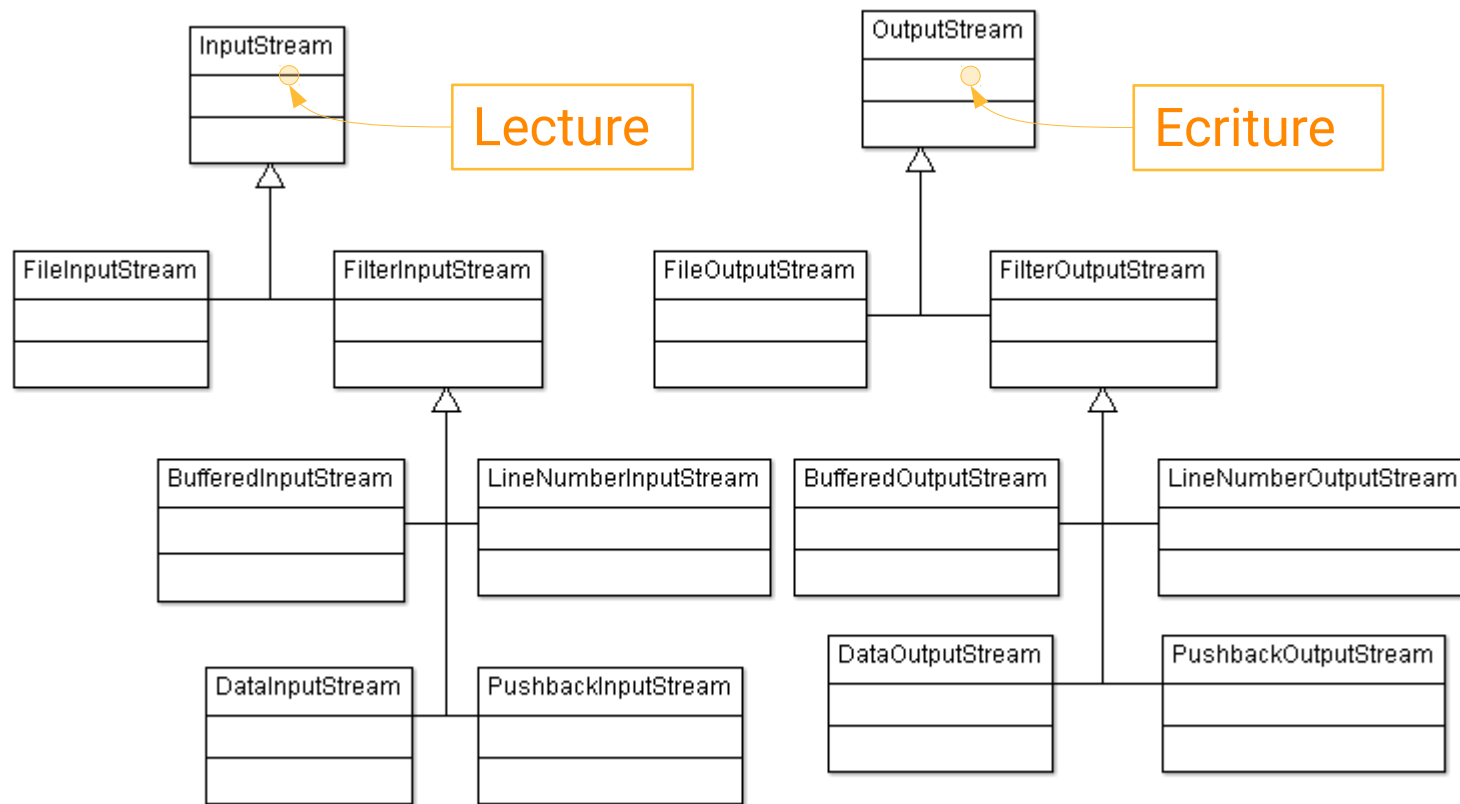
- Stream → String

```
BufferedReader reader = new BufferedReader(new  
InputStreamReader(in));  
StringBuilder sb = new StringBuilder();  
String line = null;  
while ((line = reader.readLine()) != null) {  
    sb.append(line + "\n");  
}
```



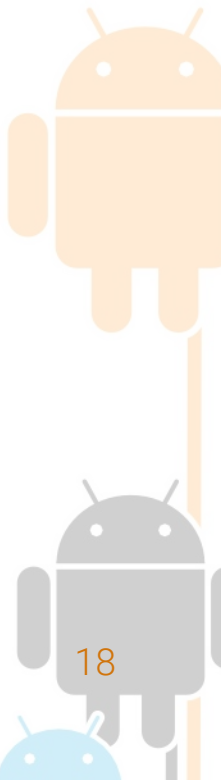
FileInputStream et FileOutputStream

- Héritent des classes abstraites InputStream et OutputStream du package java.io



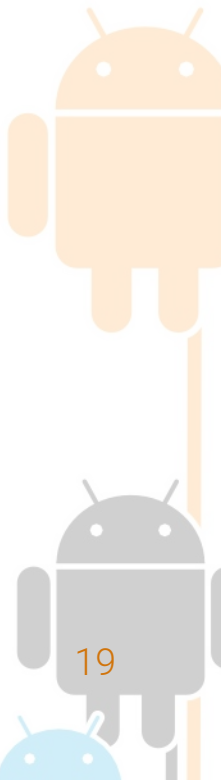
Java.io

- Les classes `InputStream` gèrent la lecture
- Les classes `OutputStream` gèrent l'écriture
- Il faudra gérer les exceptions (`IOException`)



Java.io

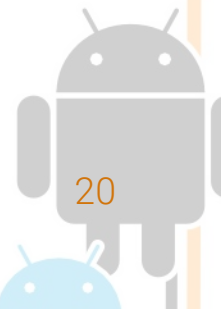
- Ajoute des fonctionnalités grâce au pattern Decorator
- `FileInputStream` : données brutes
- `DataInputStream` : données typées
- `BufferedInputStream` : ajoute un tampon (une ligne, le fichier etc...)
- `PushbackInputStream` : permet de remettre des données déjà lues dans le flux entrant
- `LineNumberInputStream` : indique le numéro de la ligne lue



Decorator

- On peut les cumuler (système de filtrage paramétrable)

```
BufferedInputStream bis = new BufferedInputStream(  
    new DataInputStream(  
        new FileInputStream(  
            new File("toto.txt"))));
```



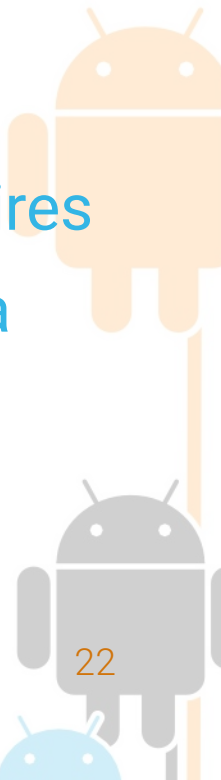
IN01 – Séance 09

Parcours (parsing)



Parcours (parsing)

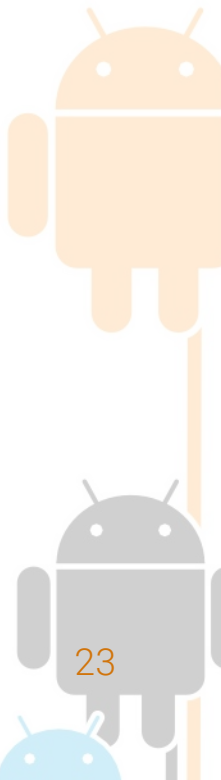
- Le SDK Android fournit une bibliothèque de parsing JSON simple d'utilisation : `org.json.JSONObject`
- Et les méthode de parcours du DOM :
 - ➔ `getJSONObject (name)` : lit un objet imbriqué
 - ➔ `getJSONArray (name)` : lit une collection
 - ➔ `getBool (name)`, `getDouble (name)`, `getInt (name)`, `getString (name)`, lisent des valeurs scalaires
 - ➔ `put(name, value)` ajoute une paire clé / valeur dans la structure JSON



REST Webservice JSON

- <http://api.geonames.org/findNearestIntersectionJSON?lat=37.451&lng=-122.18&username=demo>

```
{  
  "credits": "1.0",  
  "intersection":  
  {  
    "lng": "-122.180842",  
    "lat": "37.450649",  
    "street1": "Roble Ave",  
    "street2": "Curtis St"  
  }  
}
```

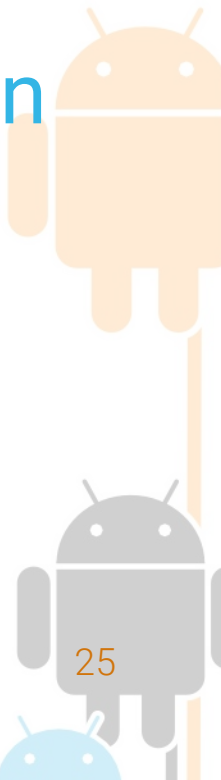


DOM & JSONObject

```
JSONObject jsonObject = new JSONObject(data);  
double credits = jsonObject.getDouble("credits");  
  
JSONObject intersection =  
jsonObject.getJSONObject("intersection");  
  
double lng = intersection.getDouble("lng");  
double lat = intersection.getDouble("lat");  
String street1 = intersection.getDouble("street1");  
String street2 = intersection.getDouble("street2");
```


Data model


- Le paradigme objet, par essence, tente de représenter toute entité sous forme d'un objet
- Les données de l'application seront regroupées dans un module appelé data model
- Better, faster lighter Java (Bruce Tate et Justin Gehtland)



Data model

```
class Intersection {  
    final double lng, lat;  
    final String street1, street2;  
    // constructor  
    public Intersection(double lng, double lat, String  
street1, String street2) { ... }  
  
    // getters  
    public double getLng() { return lng; }  
    public double getLat() { return lat; }  
    public String getStreet1() { return street1; }  
    public String getStreet2() { return street2; }  
}
```

Final / immutabilité

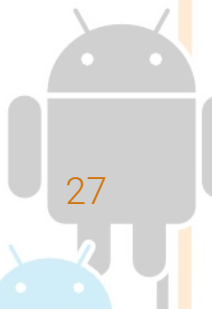
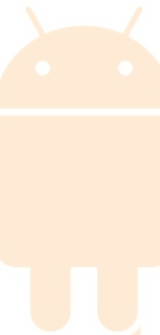


Data model

- Bonne pratique : immutabilité, évite les effets de bord
- Mapping (“maison”) JSON / Object

```
Intersection dataIntersection = new Intersection(lng,  
lat, street1, street2);
```

- Problématique récurrente et fastidieuse



XML ?

- L'ADK fournis des bibliothèques de parcours JSON et XML
- Pour XML on peut utiliser 3 types de “parseurs” :
 - ➔ DOM : Document Object Model, orienté objet
 - ➔ SAX : Simple API for XML, mécanisme à base de `callback` (`startDocument`, `startElement`, `endElement`, `endDocument`)
 - ➔ `XmlPullParser` : approche impérative (`while` / `if`, `elseif`), recommandé par Google

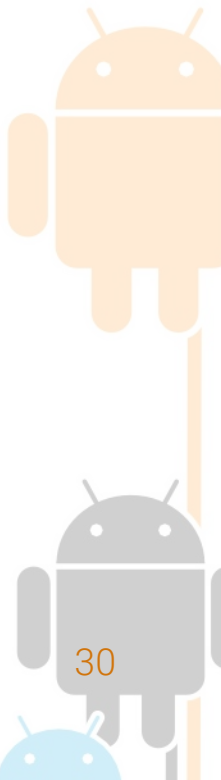
IN01 – Séance 09

AsyncTask



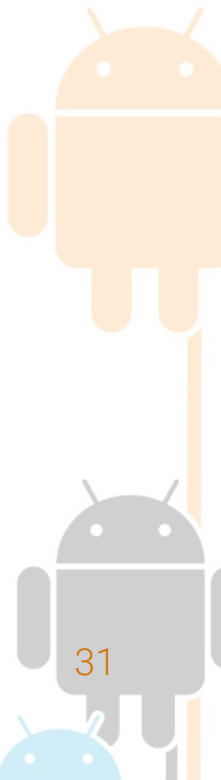
AsyncTask

- L'appel d'une ressource distante peut avoir un coût (temps)
- Bloque l'interface utilisateur
- L'ADK offre un outil pour gérer les tâches couteuses : la classe abstraite **AsyncTask**



Principe

- Créer une classe concrète qui hérite d'AsyncTask
- Une classe (interne), locale et anonyme
- Surcharger les méthodes :
 - ➔ `onPreExecute` : exécuté avant l'exécution
 - ➔ `doInBackground` : l'appel à la ressource bloquante
 - ➔ `onPostExecute` : s'exécute après l'opération
- Peut aussi gérer l'annulation de l'utilisateur ou la progression



Code

```
// Classe locale et anonyme
new AsyncTask<Location, Void, String>() {
    ProgressDialog dialog;

    @Override
    protected void onPreExecute() {
        // UI thread
        dialog = ProgressDialog.show(MainActivity.this,
            "[titre]", "[message]", true, true);
    }

    @Override
    protected String doInBackground(Location... params) {
        // Autre thread
        return null;
    }

    @Override
    protected void onPostExecute(String res) {
        // UI thread
        dialog.dismiss();
    }

}.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR, currentLocation);
```

<Paramètre, Progression, Résultat>

Gestion de la boîte de dialogue

Ressource coûteuse

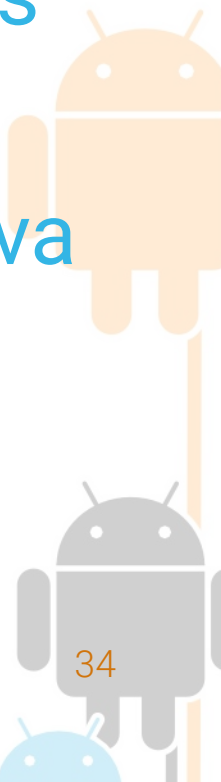
IN01 – Séance 09

Mapping avancé : Jackson

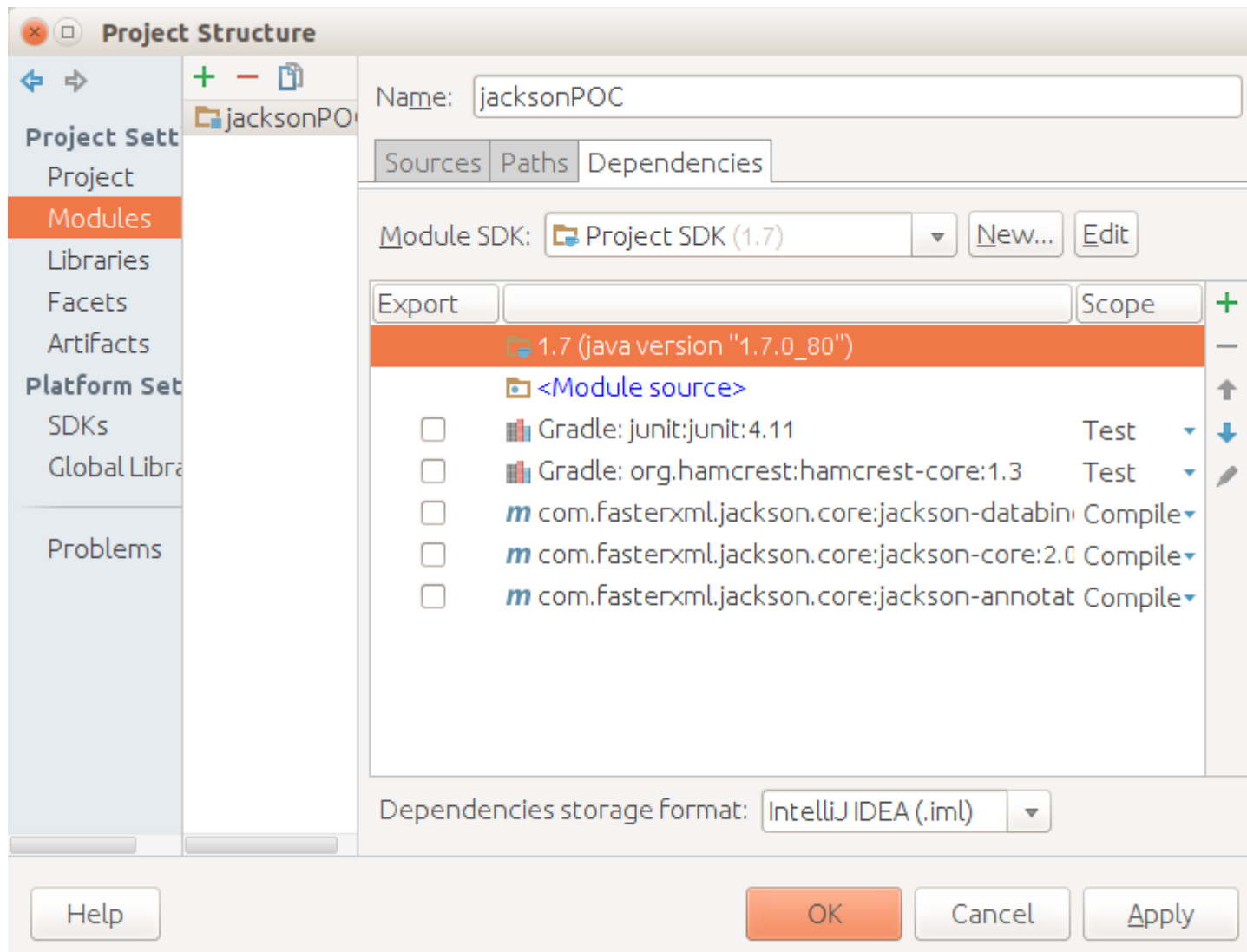


Jackson

- Jackson est une bibliothèque de serialization / Deserialisation au format JSON
- Une extension permet de faire du XML
- Utilise des POJOs (Plain Old Java Object); des objets Java sans artifice
- “Under the hood”, utilise l'introspection de Java
- Configuration du matching par annotation



Project setting : Gradle



Rappel : le fichier JSON

```
{  
  "credits": "1.0",  
  "intersection": {  
    "lng": "-122.180842",  
    "lat": "37.450649",  
    "street1": "Roble Ave",  
    "street2": "Curtis St"  
  }  
}
```

Objet imbriqué

Les Pojos correspondants

```
public class IntersectionContainer {  
    @JsonProperty("credits")  
    public double version;  
    public Intersection intersection;  
}
```

Annotation

Ou des accesseurs

```
public class Intersection {  
    public double lng, lat;  
    public String street1, street2;  
}
```

Attention ! Ce ne sont pas des BEANS

Le parsing

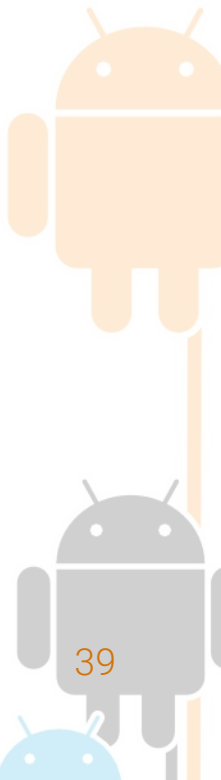
- Extrêmement simple !
- Dédduit le nom ou se sert des annotations:
 - `@JsonProperty (name)` : spécifie le nom Json
 - `@JsonIgnore ()` : ignore l'attribut

```
ObjectMapper mapper = new ObjectMapper();
```

```
IntersectionContainer ic = mapper.readValue(JSON_DATA,  
IntersectionContainer.class);
```

Annotation : intrusif

- Dans le cas ou l'objet à “mapper” existe déjà dans une autre librairie
- Solution : Utilisation des Mix-In annotations
- Une interface qui contient les attributs et qui leur associe les annotations nécessaires



Mix-In

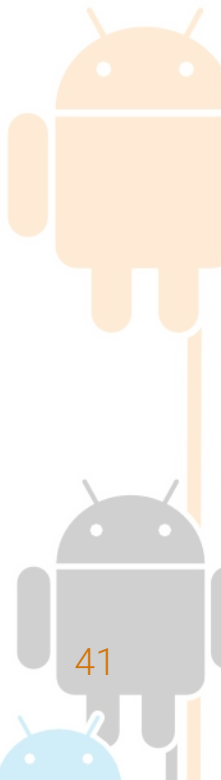
```
abstract class Mixin {  
    @JsonProperty("credits") double version;  
}  
  
public static void main(String[] args) throws IOException {  
    ObjectMapper mapper = new ObjectMapper();  
    mapper.addMixInAnnotations(  
        IntersectionContainer.class, Mixin.class);  
    IntersectionContainer ic = mapper.readValue(  
        JSON_DATA, IntersectionContainer.class);  
}
```

Specification

Association

Conclusion sur Jackson

- Ce n'est pas l'unique bibliothèque de “mapping” XML (XStream)
- Avantage : gère à la fois JSon et XML
- Performances Android
- Le mapping objet devient de plus en plus présent dans les applications. Standard de facto.



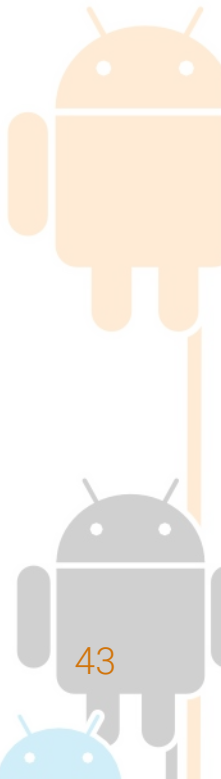
IN01 – Séance 09

Coté serveurs



Sur internet : Geonames

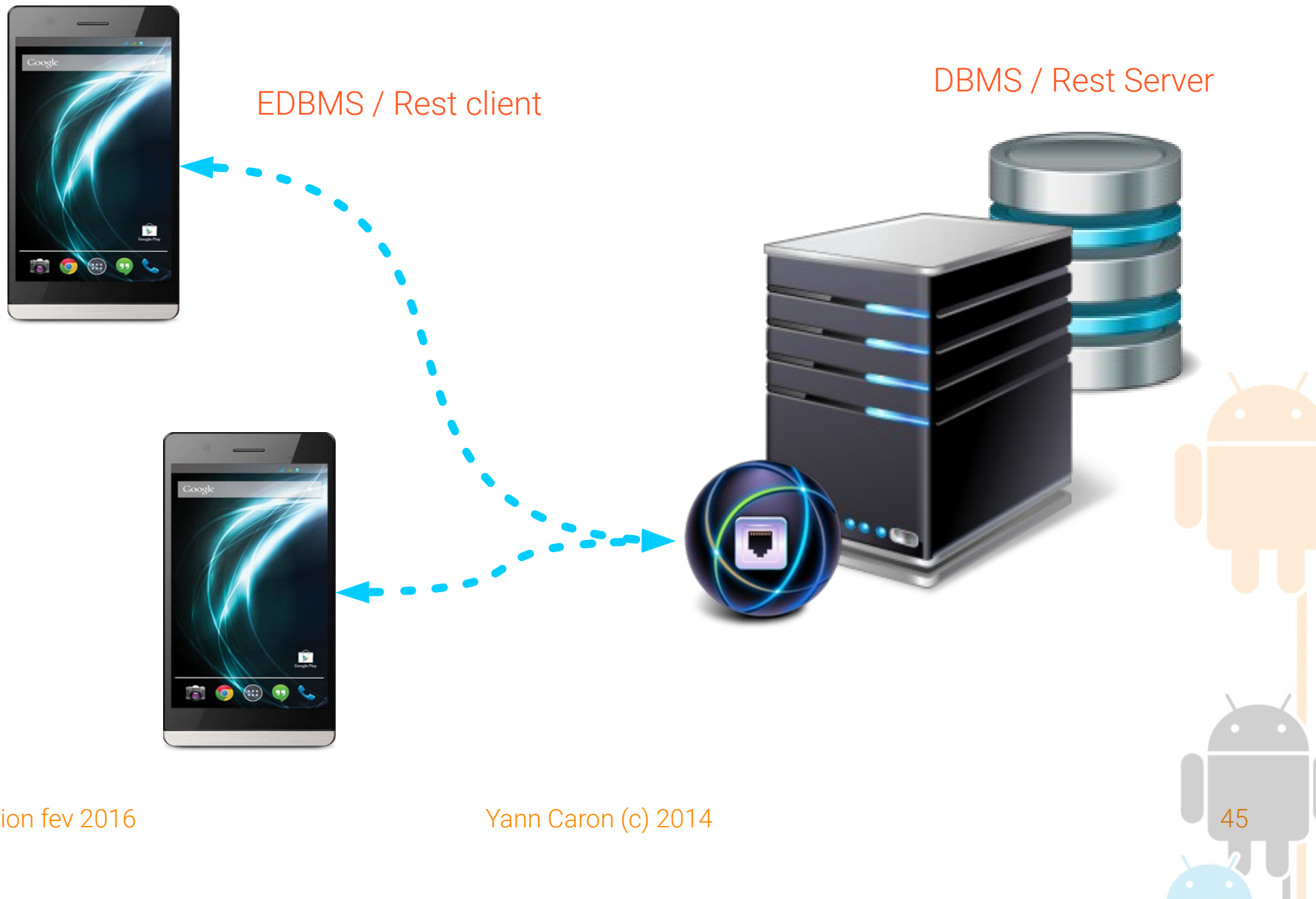
- Base de données géographique
- Service Météo METAR
- Offre une interface complète de WebServices REST au format XML / JSON, mais aussi, CSV, RSS, KML
- Exemples : Trouver l'intersection routière la plus proche (US) d'une position. Donner les données météo METAR de l'aéroport le plus proche de la position etc....



Serveur local : RESTFull

- Serveur REST en java (JAX-RS / GlassFish / JEE)
- But : Synchroniser les données de l'appareil sur un serveur mutualisé
- Principe, réaliser un “mapping” des tables de la base de données
- <http://mbaron.developpez.com/tutoriels/soa/developpement-services-web-rest-jaxrs-netbeans/>

Architecture



Fin

- Merci de votre attention
- Des questions ?

