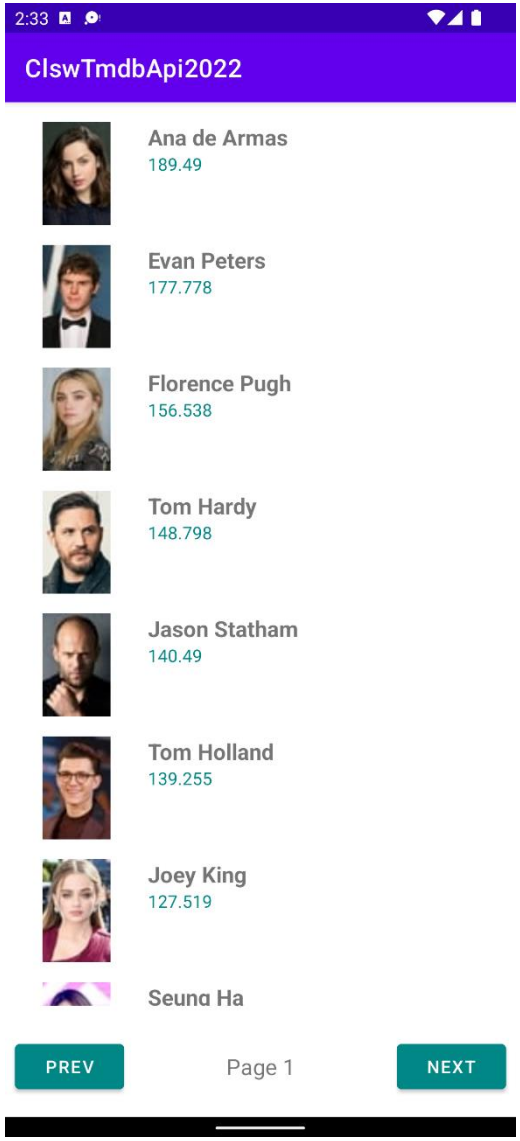
Séance 2
Consommer une API REST

Laurent PASTORELLI

laurent@bluebeacon.fr

Travaux Pratiques

- **Application TMDB API** (« The Movie Data Base »)
  - Créer une activité qui affiche la liste des « acteurs les plus populaires du moment »
  - Indiquer le numéro de la page visible à l'écran
  - Prévoir des contrôles pour passer d'une page à l'autre

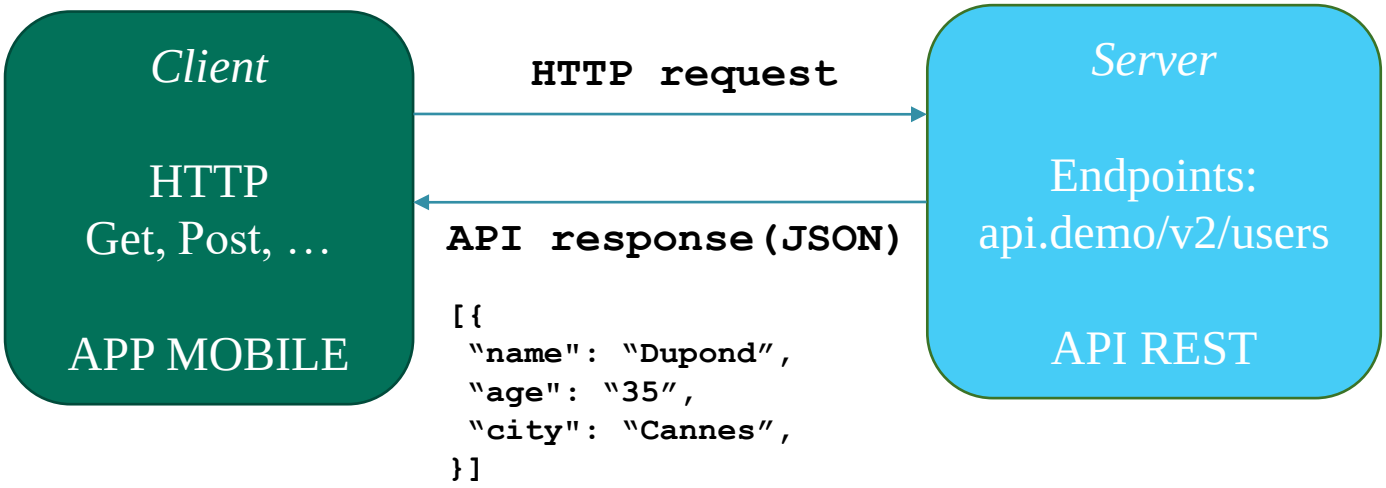


Plan d'attaque

- **Approche pour consommer l'API « TMDB » depuis une application Android native** :
  1. Déterminer les routes à utiliser pour obtenir les données à afficher.
  2. Modéliser les données reçues (JSON) par des classes JAVA
  3. Mettre en œuvre une librairie Android « sur étagère » pour consommer l'API REST (Retrofit, Volley, ...)
  4. A l'aide de traces `Log.d()` vérifier que l'on obtient bien les données attendues
  5. Créer et mettre à jour les vues avec les données reçues

API REST*

- **Définition générale**
  - « Un moyen d'accéder à des ressources ou services distants via des requêtes HTTP »
  - Ne conserve pas d'état interne



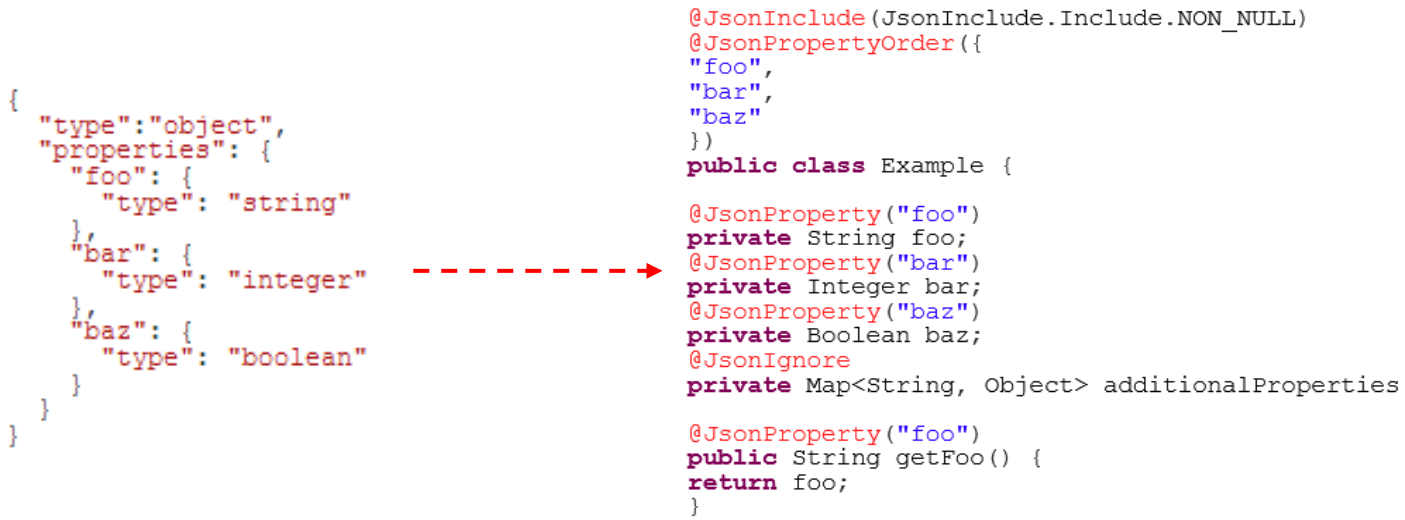
Representational State Transfert

Etude de l'API TMDB

- **API gratuite** (accès par clé privée)
  - Lire la documentation de la version 3 de l'API
  - Déterminer les **routes** (*endpoints*) à utiliser pour obtenir
    - la liste des « personnes populaires »
    - les photos de ces personalités
- Construire les requêtes HTTP correspondantes et les tester avec l'outil de votre choix:
  - Navigateur (Firefox, Chrome, Safari, ...)
  - Postman
- Repérer dans le JSON renvoyé par l'API, les informations pertinentes à afficher dans l'Application TMDB

Modélisation des données

- **Représenter le JSON renvoyé par l'API par une classe JAVA**
  - Créer manuellement la classe correspondante,
  - ou utiliser un « POJO* generator »,



```
{
  "type":"object",
  "properties": {
    "foo": {
      "type": "string"
    },
    "bar": {
      "type": "integer"
    },
    "baz": {
      "type": "boolean"
    }
  }
}
```

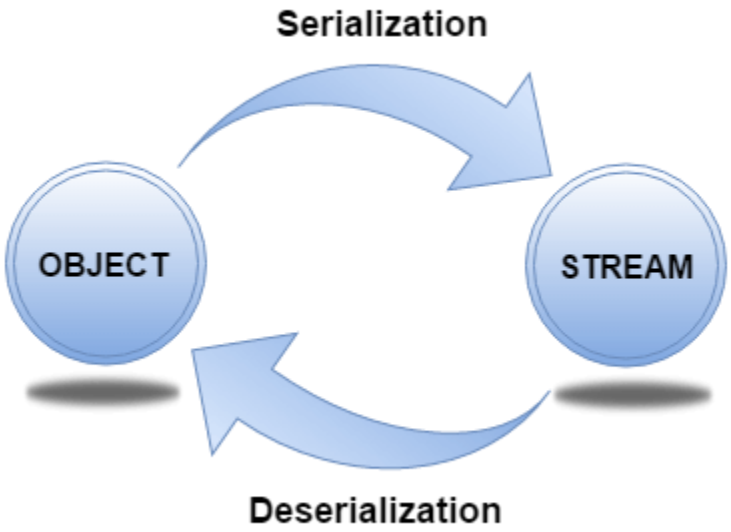
```
@JsonInclude(JsonInclude.Include.NON_NULL)
@JsonPropertyOrder({
"foo",
"bar",
"baz"
})
public class Example {

@JsonProperty("foo")
private String foo;
@JsonProperty("bar")
private Integer bar;
@JsonProperty("baz")
private Boolean baz;
@JsonIgnore
private Map<String, Object> additionalProperties

@JsonProperty("foo")
public String getFoo() {
return foo;
}
```

  - ou si possible, utiliser une méta description de l'API
    - ex: Open Api Specification (aka swagger)

(*)Plain **O**ld **J**ava **O**ject

(Dé)Sérialisation

- **Procédé consistant à convertir un objet en flux d'octets et inversement**
  - L'intérêt étant de pouvoir échanger des données entre deux machines aux architectures matérielles différentes



Retrofit 2 : Présentation

- **Librairie JAVA** open source développée par « **SQUARE** » (Jake Wharton)
  - Permet de faire des requêtes HTTP depuis une application Android via une interface JAVA simple
  - Utilisation des annotations JAVA pour définir les paramètres des requêtes HTTP
  - Conversion automatique de la réponse HTTP en objet JAVA
  - Choix du (dé)sérialiseur en paramètre (GSON, ...)
  - Requêtes synchrones et asynchrones

Retrofit 2 : Mise en œuvre

- Consulter le tutoriel en ligne de votre choix
  - Écrire le **code d'initialisation** du client RETROFIT
    `ApiClient.java`
  - Écrire l'**interface JAVA** de description des routes
    `TmdbApi.java`
- Créer un appel **asynchrone** pour la requête de type GET avec l'interface précédemment crée
- Dans un premier temps, **restituer** les données renvoyées par l'API TMDB sous forme de trace `Log.d()`
- Dans un second temps, **mettre à jour** les vues concernées avec les données renvoyées par l'API TMDB

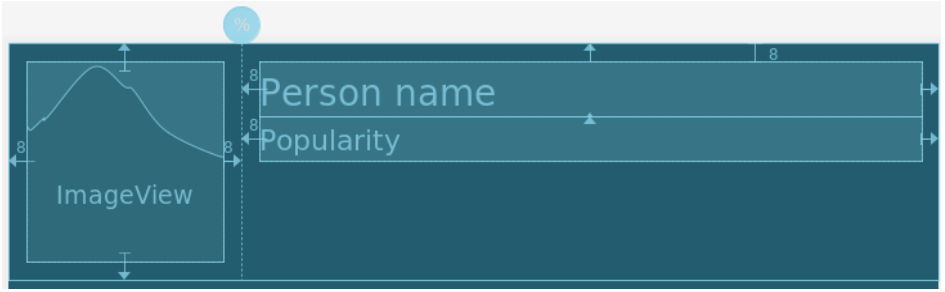
Création des layouts (1/2)

- **Activité principale**
  - RecyclerView
  - Ligne guide
  - ButtonView (x2)
  - TextView

. Progress bar (en visibilité "gone") si possible

Création des layouts (2/2)

- **Eléments de la lise**
  - ImageView
  - Ligne guide
  - TextView



Picasso : Présentation

- Une **librairie JAVA** open source développé par « **SQUARE** »
  - Permet de télécharger de manière asynchrone des images distantes dans une « ImageView » Android
  - Mise en cache en mémoire et sur disque
  - Possibilité de retailler les images « à la volée »
  - Gestion des erreurs