TP 4: Client REST JSON

Introduction

Dans ce TP, vous allez écrire un **client** Http en Java en utilisant une bibliothèque dédiée, d'usage plus facile que le code utilisé au TP précédent. Vous l'utiliserez pour vous connecter à une API REST. Il faudra ensuite analyser les données JSON reçues et se connecter à d'autres points d'entrée à partir des informations reçues.

Aspects pratiques :

Vous utiliserez le *package Apache HttpComponents* : `http://hc.apache.org` . Il faudra donc préciser, à la fois lors de la compilation et l'exécution, l'usage de ces packages tiers.

- Copier tous les fichier **`.jar`** présents dans **`P:\RCPI01\jar`** dans le dossier **`C:\temp`**
- pour compiler : **`javac -classpath c:\temp\* Monfichier.java`**
- pour exécuter : **`java -classpath .;c:\temp\* Monfichier`**

Pour éviter les erreurs, il sera judicieux de mettre ces deux lignes dans deux scripts, appelés par exemple **`compile.bat`** et **`run.bat`**. Pour le deuxième, l'URL du serveur auquel on voudra se connecter doit pouvoir être transmise en argument, il faut donc passer celui-ci a votre programme Java (rappel :**`%1`** sur Windows).

1 Client HTTP

1. Ecrire un squelette de programme nommé **`Client1`**, comprenant une fonction main avec des arguments de ligne de commande.
2. Dans le main, ajouter le code pour vérifier que l'utilisateur a bien donné au moins 1 argument. Dans le cas contraire, afficher un message d'erreur et quitter le programme (appel de la fonction **`System.exit(1);`**) Vérifier la bonne compilation et exécution.
3. Ajouter ensuite la création du client, ainsi que la création de la requête "GET" :

```
CloseableHttpClient client = HttpClients.createDefault();
String url = "http://" + args[0];
HttpGet request = new HttpGet(url);
```

   Pour compiler, il faudra ajouter les imports suivants en tête du fichier :

```
import java.io.*;
import org.apache.http.HttpEntity;
import org.apache.http.client.*;
import org.apache.http.client.methods.*;
import org.apache.http.impl.client.*;
```

   Vérifier la bonne compilation.
4. Ajouter ensuite l'exécution de la requête HTTP, qui va également renvoyer la réponse du serveur :

```
System.out.println( "Executing request " + request.getRequestLine() );
CloseableHttpResponse resp = client.execute(request);
```

   On pourra afficher la réponse du serveur avec

```
System.out.println( "Response Line: " + resp.getStatusLine() );
System.out.println( "Response Code: " + resp.getStatusLine().getStatusCode() );
```

   Vérifier la bonne compilation, et vérifier que l'exécution est correcte avec (par exemple) :
   `run google.fr`
5. Ajouter ensuite le code pour récupérer la page envoyée par le serveur dans une chaîne, via un objet "lecteur" :

```
BufferedReader rd = new BufferedReader( new InputStreamReader( resp.getEntity().
    getContent()) );
```

Il faut ensuite lire le code HTML renvoyé en faisant une lecture ligne par ligne, avec une boucle **`while()`**. Comme le type Java **`String`** est immutable, on doit passer par un objet **`StringBuffer`** dans lequel on pourra ajouter les différentes lignes reçues du serveur, une à une, jusqu'à la fin :

```
StringBuffer result = new StringBuffer();
String line = "";
while ((line = rd.readLine()) != null)
{
  result.append(line);
  result.append("\n"); // pour avoir le saut de ligne
}
```

On peut ensuite afficher la page à l'écran, en convertissant en **String** :

```
String page = result.toString();
System.out.println( page );
```

6. Vérifier la bonne compilation, et vérifier que l'exécution est correcte avec (par exemple) :

   `run google.fr`

   Vérifier aussi que les sites qui ne répondaient pas avec la 1re version du client (TP2) répondent cette fois correctement.

7. Vérifier ensuite que l'on peut s'adresser à un WebService qui renvoie du JSON. Vous allez pour cela essayer avec l'API de la base de données cinématographique OMDb.

   Se connecter avec un navigateur sur `http://omdbapi.com` et analyser la façon dont on utilise l'API (section "usage").

   Faire un essai manuel via un navigateur avec une recherche sur un film quelconque, en saisissant dans la barre d'adresse une URL de requête sur un titre de film.

   Rappel : pour passer des paramètres dans une URL, la syntaxe est :

   `http://monsite.com?key1=value1&key2=value2&key3=value3`

   Pour utliser l'API, il faut fournir une clé (fournie par l'enseignant), en ajoutant un paramètre **`apikey=XXXXX`** dans l'URL.

   Essayez ensuite de reproduire cette requête via votre programme en redirigeant la sortie vers un fichier. (**Rappel : `a > b`** redirige la sortie du programme 'a' dans un fichier 'b'.) Ouvrez ensuite ce fichier et vérifiez que vous obtenez bien la même chose.

   Attention, le caractère **`&`** est un caractère spécial pour le shell. Pour pouvoir utiliser une URL avec des paramètres comme argument de script, il faut "quoter" la chaîne :
   `run "monsite.com?key1=value1&key2=value2"`

2 Extraction de l'information : Client Rest

En l'état, nous n'obtenons qu'une chaîne de caractère peu exploitable, et il faut *parser* cette chaîne JSON pour obtenir l'information souhaitée sous forme structurée. En Java, de multiples moyens existent, nous utiliserons ici l'implémentation de référence : *Java API for JSON Processing (JSR 353)*, détaillée ici : `https://jsonp.java.net`

Des exemples plus concrets sont accessibles ici : `http://www.journaldev.com/2315/java-json-example`

1. Copier le fichier **`Client1.java`** en **`Client2.java`** et l'éditer. Supprimer les lignes relatives à la construction de la chaîne finale et remplacer la ligne créant le "BufferedReader" par la ligne suivante, qui va permettre au *parser* JSON de lire le flux directement :

   ```
   InputStreamReader fis = new InputStreamReader( resp.getEntity().getContent() );
   ```

   Vérifier la bonne compilation.

2. Créer le "lecteur JSON", puis l'objet JSON contenant les champs à partir du *reader* :

   ```
   JsonReader jsonReader = Json.createReader(fis);
   JsonObject jsonObject = jsonReader.readObject();
   ```

   Comme ces objets sont désormais inutiles, on peut fermer le flux et le lecteur JSON :

   ```
   jsonReader.close();
   ```

```
fis.close();
```

Vérifier la bonne compilation, après avoir ajouté l'import suivant : `import javax.json.*;`

3. On peut ensuite accéder aux valeurs, via les méthodes de la classe **`JsonObject`**. Par exemple, dans le cas de l'API "OMDb", on pourra accéder au champ "Runtime", qui indique la durée du film, avec :

```
System.out.println( "duree=" + jsonObject.getString("Runtime") );
```

On pourra de façon similaire récupérer des entiers avec la méthode `getInt(String name)`.
Vérifier le bon fonctionnement en affichant la valeur de la clé "Runtime" avec l'exemple de requête précédent.

Rappel (cf. cours) : un objet JSON (**`JsonObject`**) peut contenir :

- Une chaîne ou un nombre : `{"aaa": "bbb"}` ou `{"aaa": 123.45}`
- Un sous-objet JSON : `{"aaa": { ... } }`
  On y accèdera avec la méthode **`getJsonObject( "aaa" )`**, qui renvoie un objet de type **`JsonObject`**.
- Un tableau JSON : `{ "aaa": [ ... ] }`
  On y accèdera avec la méthode **`getJsonArray( "aaa" )`**, qui renvoie un objet de type **`JsonArray`**.

Pour les détails, voir `http://docs.oracle.com/javaee/7/api/javax/json/JsonObject.html`

3 Utilisation de l'API

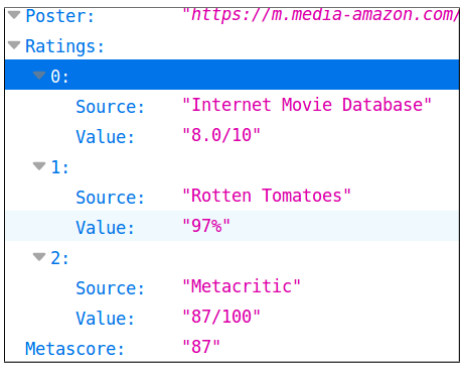
Copier le programme précedent en **`Client3.java`**. Le modifier pour qu'il tourne dans une boucle infinie : saisie d'un titre et affichage des infos essentielles : date de sortie et acteurs principaux. Vous ajouterez une mention suivant le score des critiques du site "Rotten Tomatoes" :

- "Nul" s'il est inférieur à 20 %
- "Bof" s'il est entre 20 et 50 %
- "Bien" s'il est entre 50 et 70%
- "Très bien" au dessus

Le score du film est accessible dans un tableau JSON (type : **`JsonArray`**) dont la clé est "ratings". Il faut donc récupérer avec **`getJsonArray()`** ce tableau (voire ci-dessus), puis itérer dessus avec une boucle "for" jusqu'à trouver l'objet JSON dont la valeur de la clé "Source" est "Rotten Tomatoes" :

```
JSonArray tab = getJsonArray( ...
for( int i=0; i<tab.size(); i++ )
{
  JsonObject ji = tab[i];
  ...
}
```

Il ne reste alors qu'à afficher la valeur de la clé "Value".



Annexe

Lecture d'une chaîne au clavier :

```
Scanner sc = new Scanner(System.in);
System.out.println("Veuillez saisir un mot :");
String str = sc.nextLine();
System.out.println("Vous avez saisi : " + str);
```

En ajoutant l'import suivant :
`import java.util.Scanner;`