Projet Programmation Orientée Objet

2019 - 2020

L'objectif de ce projet est de réaliser un agrégateur de flux météo en ligne de commande. Il permet de récupérer automatiquement depuis le web les prévisions météo d'une zone spécifiée et de les afficher dans la console.
Pour faciliter le travail des groupes, qui seront des trinômes, le projet est découpé en quatre itérations successives. Chaque itération fera l'objet d'une démo et d'une rapide revue de code durant le début de chaque séance de suivi de projet (sauf la première, dédiée à la clarification du sujet et au lancement du projet).

Étape 1: Un premier bulletin météo

L'objectif de cette première livraison est de développer un premier client qui récupère la météo provenant de l'API MetaWeather.
Une API (Application Programming Interface) permet d'avoir accès aux fonctionnalités proposées par une application. Une API REST n'est rien d'autre qu'une API accessible depuis le web via les commandes standards du protocole HTTP.
Par exemple une requête HTTP de type GET sur l'URL: https://www.metaweather.com/api/location/search/?query=bordeaux, nous permet de récupérer les coordonnées géographique de la ville de Bordeaux. Les résultats sont généralement retournés au format JSON ou XML.
En Java, consommer une API revient donc à simplement faire une requête HTTP et analyser sa réponse JSON. Pour ce faire, il faut utiliser la classe java.net.HttpUrlConnection de la librairie standard de Java pour effectuer la requête et utiliser un parseur JSON pour analyser la réponse. La plupart des APIs retournent leurs réponses par défaut au format JSON (JavaScript Object Notation). Afin de pouvoir extraire facilement les informations depuis la chaîne de caractères qui vous est retournée, il vous sera nécessaire de parser cette dernière afin de pouvoir aisément la parcourir. Pour ce faire, il existe plusieurs bibliothèques, l'une des plus populaires est org.json, vous pouvez cependant utiliser celle que vous préférez. Vous pouvez retrouver sa documentation sur le site suivant: https://stleary.github.io/JSON-java.

Le premier client devra fonctionner de cette manière :

```
shell$ java -jar weather.jar -l Bordeaux -j 3
              +-----+-----+-----+-----+
              | J+0 | J+1 | J+2 | J+3 |
+-------------+-----+-----+-----+-----+
|             | 10° | 12° | 8°  | 15° |
+-------------+-----+-----+-----+-----+
```

Le programme, packagé sous forme de jar devra récupérer la météo pour une ville passé en argument (`-l`) et pour un nombre de jours passés en argument (`-j`), et l'afficher sur la console. Voici un exemple :

Étape 2: Gestion de plusieurs flux météo

Pour la réalisation de notre aggrégateur, nous avons sélectionné une liste d'APIs gratuites. La liste est la suivante:

- MetaWeather
- prevision-meteo.ch
- Openweathermap

Chaque API possède sa propre documentation à laquelle vous pouvez avoir accès sur le site correspondant. L'objectif de la deuxième livraison est de rajouter ces API dans l'agrégateur, de la manière suivante :

```
shell$ java -jar weather.jar -l Bordeaux -j 3
              +-----+-----+-----+-----+
              | J+0 | J+1 | J+2 | J+3 |
+-------------+-----+-----+-----+-----+
| MetaWeather | 10° | 12° | 8°  | 15° |
+-------------+-----+-----+-----+-----+
| P-Meteo     | 11° | 11° | 9°  | 18° |
+-------------+-----+-----+-----+-----+
| OW Weather  | 10° | 11° | 10° | 17° |
+-------------+-----+-----+-----+-----+
```

Lors de cette livraison vous devez faire en sorte d'avoir un design qui rend l'ajout d'un nouveau flux météo le plus simple possible.

Étape 3: Un bulletin plus détaillé

Pour la troisième livraison nous allons fournir un bulletin météo plus détaillé de la manière suivante.

```
shell$ java -jar weather.jar -l Bordeaux -j 0
              +----------------+
              | J+0            |
+-------------+----------------+
| MetaWeather | 10° 69% 3km/h  |
+-------------+----------------+
| P-Meteo     | 11° 70% 24km/h |
+-------------+----------------+
| OW Weather  | 10° 71% 10km/h |
+-------------+----------------+
```

Les informations supplémentaires par rapport à la première livraison doivent être :

- la valeur de l'humidité
- la vitesse du vent.

Si une valeur n'est pas disponible pour un flux provenant d'une API donnée, "-" sera affiché à la place.

Étape 4: Historique des requêtes et erreurs

Pour la quatrième livraison nous allons améliorer la gestion des erreurs. Pour cela l'application devra maintenir un historique de toutes les requêtes réalisées dans un fichier `requetes.log` à la racine du dossier dans lequel est le fichier jar de l'application. Ce log doit contenir des informations relatives à la requête (Date, Code HTTP de la réponse, URL) sous le format suivant:

```
01-12-2017 10:00:01 - [200 OK]
https://www.metaweather.com/api/location/search/?query=bordeaux
01-12-2017 10:01:00 - [503 ERROR]
https://www.metaweather.com/api/location/search/?query=bordeaux
```

Vous devrez également gérer les erreurs qui peuvent arriver lors de l'utilisation de l'application. Voici la liste des erreurs à gérer:

- Erreur de ligne de commande
- Problème de connexion à une API
- Localisation inconnue

Quand une erreur survient, un message d'erreur doit être affiché sur la sortie standard. Si l'erreur n'est pas fatale (cas du problème de connexion à une API avec une autre API qui fonctionne), l'application doit continuer à s'exécuter. Dans le cas contraire, l'application doit stopper.

Fonctionnalités supplémentaires

Des points bonus pourront être obtenus pour chaque implémentation d'une fonctionnalitée proposée dans cette liste:

- Ajouter de nouvelles APIs à la liste des APIs disponibles (https://developer.accuweather.com/, https://darksky.net/dev/docs,, ...), ces APIs nécessitent une authentification pour leur utilisation.
- Enrichir l'interface de l'application en utilisant par exemple des ASCII Art pour présenter l'état de la météo à la manière du site (http://wttr.in).
- Réaliser les requêtes à travers des threads, ce qui permettra d'avoir un temps de traitement nettement plus rapide en parallélisant les requêtes.

Contraintes sur le code

Vous devez absolument RESPECTER ces contraintes. Une pénalité de 2 points sera appliquée sur chaque projet qui ne les respectent pas.

- Votre projet contient plusieurs paquetages.
- Les noms des paquetages sont intégralement en minuscules.
- Les noms des classes et interfaces commencent par une majuscule.
- Les noms des méthodes, des attributs et des variables commencent par une minuscule.
- Utilisez le *camel case* pour les noms des classes, interfaces, méthodes et attributs.
- Les attributs ont une visibilité `private` ou `protected`. Des accesseurs permettent si nécessaire d'accéder en lecture et/ou écriture aux attributs.
- La visibilité des méthodes dépend de leur utilisation.
- Les constructeurs sont définis uniquement s'ils sont nécessaires.
- Le code des méthodes compliquées doit être commenté.