Projet Programmation Orientée Objet

2017 - 2018

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.

Étape 1: Consommer une API REST

Une API (Application Programming Interface) permet d'avoir accès aux fonctionnalitées 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 (voir ci-après) pour analyser la réponse.

Étape 2: Liste des APIs à implémenter

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

- MetaWeather
- prevision-meteo.ch
- Yahoo! Weather

Chaque API possède sa propre documentation à laquelle vous pouvez avoir accès sur le site correspondant.

Étape 3: Parseur JSON

Toutes les APIs proposées 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.

Étape 4: Création du Wrapper

Comme vous avez surement dû le remarquer en utilisant les différentes APIs, chacune retourne un résultat différent avec une structure différente. L'idée d'un Wrapper c'est de tout simplement abstraire toutes les différences entre ces APIs pour faire en sorte que lorsque le développeur souhaite utiliser une API implémentée dans notre programme, les spécificités de chaque APIs sont abstraites et qu'il se retrouve avec une interface unifiée peu importe la source qu'il utilise. De la même manière si il souhaite en rajouter une nouvelle, il n'aura qu'à se contraindre aux contraintes déjà définie à travers notre Wrapper et son API sera très facilement ajoutée au programme sans qu'il ait à toucher quoi que ce soit, à part pour la rajouter à la liste des APIs disponibles.

Étape 5: Utilisation de l'agrégateur

L'application devra pouvoir être exécutée directement depuis le terminal à travers des lignes de commandes (Command-line Interface), elle devra avoir le comportement qui suit. Elle doit récupérer les arguments qui lui sont passés depuis la ligne de commande (à travers des flags):

```
shell$ java -jar weather.jar Bordeaux -j 3 -h

+-------------+-----+-----+-----+-----+
|             | J+0 | J+1 | J+2 | J+3 |
+-------------+-----+-----+-----+-----+
| MetaWeather | 10° | 12° | 8°  | 15° |
|             | 69% | 62% | 61% | 56% |
+-------------+-----+-----+-----+-----+
| P-Meteo     | 11° | 11° | 9°  | 18° |
|             | 63% | 68% | 51% | 52% |
+-------------+-----+-----+-----+-----+
| Y! Weather  | 10° | 11° | 10° | 17° |
|             | 69% | 61% | 61% | 63% |
+-------------+-----+-----+-----+-----+
```

L'application devra en plus d'être capable d'afficher la météo pour un nombre de jour déterminé à travers le flag -j qui peut aller jusqu'à 5 jours (-j 5), les autres flags sont:

- -o [nom_fichier_output]: Permet en plus d'afficher le résultat dans la console, de le printer dans un fichier. Si un fichier est déjà présent avec ce nom ils est vidé.
- -a [nom_fichier_output]: Permet en plus d'afficher le résultat dans la console, de le printer dans un fichier, et cela, sans effacer le contenu déjà présent dans le fichier.
- -h: Enrichit le résultat avec les valeurs de l'humidité.

- -m [F/C]: permet de spécifier si on veut avoir les résultats en Fahrenheit ou en Celsius. L'unité de mesure par défaut pour la température sera le Celsius.
- -w: permet d'afficher en plus des informations déjà affichées, la vitesse du vent.

Les valeurs dépendantes de l'API seront seulements affichées pour les APIs qui les proposent, si la valeur n'est pas disponible un "-" sera affiché à la place.

Étape 6: Historique des requêtes

L'application devra également 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
```

Étape 7: Gestion des erreurs

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 (Apixu, Openweathermap, ...), ces APIs nécessitent une authentification pour leur utilisation.
- Enrichir l'interface de l'application en utilisant 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.

Livrables

Il sera demandé aux étudiants de fournir une archive contenant le code source du programme. À la racine de l'archive, il doit y avoir un fichier **README** spécifiant les étapes qui ont été complétées et celles qui ne l'ont pas été, ainsi que les fonctionnalités supplémentaires qui ont été implémentées. L'archive doit aussi contenir un fichier java archive `weather.jar` qui permet de lancer le jeu à l'aide de la commande `java -jar weather.jar`.
Le schéma de nommage de l'archive doit être le suivant (pénalité de 3 points en cas de non respect) : **pg220-2018-LOGINS.zip**, où LOGINS est la liste des logins des membres du groupe, classée par ordre alphabétique et séparés par le symbole « _ ».
L'archive est à déposer, avant le **X XXXX 2017** à 23h, via la plateforme http://moodle.ipb.fr.