Projet de Licence Java

Le but de ce projet est de réaliser une application permettant de calculer des métriques sur un code Java.

Modalité

Ce projet est à faire par binôme (toujours 2 personnes max). et est à rendre pour le lundi 12 mai.
L'archive ZIP contenant l'ensemble du projet devra être envoyé simultanément aux adresses : forax@univ-mlv.fr, dr@univ-mlv.fr et bedon@univ-mlv.fr.

Introduction

Une métrique est une mesure effectuée sur le code permettant de quantifier un aspect de celui-ci.
Par exemple, il existe des métriques permettant de savoir quelle est le coût de développement ou de maintenance d'un projet. Ces métriques sont des fonctions qui dépendent par exemple du nombre de champs et/ou de méthodes par classe, du nombre de types utilisés, du nombre de classes hérités, etc..
Le but de ce projet est de fournir une application permettant de calculer une métrique pour un code java et de rajouter facilement de nouvelles métriques. L'accent devra être mis sur une réalisation propre et objet de l'application.

Fonctionnalités

L'application manipule deux concepts différents :

1. Les métriques (`Metric`) qui évaluent chaque item (classes, paquetages, champs, etc ...) du code
2. Les parcours (`Traversal`) qui permettent d'indiquer le parcours et sur quel ensemble d'items une métrique devra travailler

Chaque métrique et chaque parcours possède un nom qui permet à un utilisateur d'indiquer sur la ligne de commande quelle métrique et optionnellement quel parcours il veut utiliser pour son code. L'application devra utiliser les classes et interfaces suivantes :

- La classe `MetricDoclet` permet d'évaluer une métrique à l'aide d'une des méthodes `eval()` et utilise l'outil `javadoc` pour récupérer l'ensemble des items du code.

  ```
  class MetricDoclet {
    void eval(Metric m,Traversal t);
    void eval(Metric m);
  }
  ```

  La méthode `eval()` qui ne prend pas de parcours en paramètre, utilisera le parcours par défaut de la métrique (voir la méthode `getDefaultTraversal()` de la classe `Metric`)
- L'interface `Metric` qui possède autant de méthodes `evaluate` qu'il existe d'items de code plus les méthodes `printResult()` et `getDefaultTraversal()` qui permettent respectivement d'afficher le résultat du calcul de la métrique et d'indiquer

quel est le parcours par défaut d'une métrique. Les méthodes `evaluate()` calculent la valeur de la métrique pour l'item correspondant.

```
interface Metric {
  void eval(RootDoc root);
  void eval(ClassDoc clazz);
  void eval(ConstructorDoc constructor);
  void eval(FieldDoc field);
  void eval(MethodDoc method);
  void eval(PackageDoc packaze);

  Traversal getDefaultTraversal();
  void printResult(Writer writer);
}
```

Les interfaces `RootDoc`, `ClassDoc`, `ConstructorDoc`, `FieldDoc`, `PackageDoc` et `MethodDoc` représente respectivement un projet, une classe, un constructeur, un champ, un paquetage et une méthode.
Le résultat d'une évaluation de la métrique est stockée dans l'objet métrique correspondant et affichable par l'intermédiaire de la méthode `printResult()`.

- L'interface `Traversal` spécifie pour chaque item de code quel est le parcours à effectuer et les appels à la métrique.

```
interface Traversal {
  void traverse(RootDoc root,Metric metric);
  void traverse(ClassDoc clazz,Metric metric);
  void traverse(ConstructorDoc constructor,Metric metric);
  void traverse(FieldDoc field,Metric metric);
  void traverse(MethodDoc method,Metric metric);
  void traverse(PackageDoc packaze,Metric metric);
}
```

Parcours

Voici une liste de parcours à implanter ainsi que leur nom

- **class**: parcours de l'ensemble des classes
- **method&constructor**: parcours de l'ensemble des méthodes et/ou des constructeurs
- **alltype**: parcours de tous les types (classe, type de retour et paramètres des méthodes, exceptions)
- **override**: parcours de l'ensemble des méthodes redéfinies.
- **class:noinner**: parcours de l'ensemble des classes qui n'ont pas de classes internes
- etc...

Vous devrez peut-être implanter d'autre parcours, pour pouvoir implanter correctement les métriques ci-dessous.

Métriques

Voici une liste des métriques à implanter ainsi que leur nom

- **privacy**: Rapport entre le nombre de classes publiques et le nombre de classes privées d'un paquetage.
- **concreteness**: Différence entre le rapport du nombre de classes concrètes sur le nombre de classes total et du nombre de classes abstraites (et d'interfaces) sur le nombre de classe

- **line**: Nombre de lignes de code par méthode, l'affichage sera triée par ordre décroissant du nombre de ligne (avec qui est l'auteur de chaque méthode :).
- **mutable**: Rapport entre le nombre de classes mutables et immutables (une classe immutable ne possède que des champs constants) par paquetage.
- **innerClass**: Nombre de classes internes par classe ainsi que la moyenne et l'écart-type par paquetage
- **type**: Nombre de types manipulés par classe et moyenne par paquetage.

En plus de ces métriques, vous devrez implanter une métrique complexe **maintenance** permettant de mesurer le coût de maintenance d'un projet.
Si vous voulez de plus amples informations sur les différentes métriques existantes, le Web est à vous.

JavaDoc

L'outil `javadoc` permet non seulement de générer la documentation d'un projet Java mais permet aussi par l'intermédiaire de **doclet** de travailler directement sur le source d'un projet.
Nous utiliserons ici cet outil pour obtenir une représentation des sources Java en mémoire.

En ligne de commande

L'application possède deux utilisations distinctes :

- La commande `-list` doit afficher l'ensemble des métriques et des parcours ansi que leurs noms.

```
run list
```

- La commande classique exécute une métrique avec optionnellement un parcours sur un ensemble de paquetages passés en paramètres.

```
run line fr.umlv.metric fr.umlv.metric.metrics
```

  Le parcours choisi est alors le parcours par défaut de la métrique **line**.
  La commande

```
run line override fr.umlv.metric fr.umlv.metric.metrics
```

  appliquera la métrique **line** uniquement sur les méthodes redéfinies.

Organisation du projet sur le disque

L'archive zip devra contenir les fichiers et répertoires suivants :

1. un fichier `build.xml`, script de compilation ANT
2. un répertoire `src` contenant les sources Java (organisées dans un ou plusieurs paquetages)
3. un répertoire `classes` contenant l'ensemble des classes du projet.
4. un répertoire `lib` contenant une l'archive jar correspondant à l'application
5. un répertoire `docs` contenant la documentation
   - Un manuel utilisateur au format PDF.
   - Un manuel de développeur aussi en PDF expliquant l'architecture du programme, la procédure permettant de rajouter de nouvelles métriques et de nouveaux

parcours ainsi que les élements retenus pour juger de la maintenabilité d'un programme.
    - Dans le sous-répertoire `api`, une documentation JavaDoc complète des sources du programme. Contrairement aux autres documentations, la documentation JavaDoc devra être écrite en anglais (anglais de base, environ 50 mots).

6 Un répertoire `bin` contenant deux fichiers `run.sh` et `run.bat` permettant de lancer l'application resepectivement sous linux et sous windows.

Références

- JavaDoc (`/usr/local/j2sdk1.4.1/docs/tooldocs/javadoc/index.html`)
- Ant 1.5 (`http://ant.apache.org/`)
- Métriques sur le Web (`http://www.google.fr/search?q=metrics`)