RAPPORT

Création de Bakefile, recréation de Makefile

Membres :

Basile LEGRELLE
Raphaël HOCHLAF
Thomas FOLLEA

Sommaire :

Introduction à Make

Le programme Make est un utilitaire de construction automatisée principalement utilisé pour la compilation de programmes, bien qu'il puisse servir à d'autres tâches d'automatisation. Il repose sur un fichier appelé Makefile, qui définit un ensemble de règles décrivant comment générer des fichiers cibles à partir de fichiers sources.

Il fonctionne en vérifiant les dépendances entre les fichiers et en exécutant uniquement les commandes nécessaires pour mettre à jour les fichiers obsolètes. Il compare les horodatages des fichiers pour déterminer s'ils doivent être reconstruits.

Introduction à Bake

Le programme Bake est une copie simplifiée de l'utilitaire Make. Il repose sur un fichier appelé Bakefile, qui définit un ensemble de règles décrivant comment générer des fichiers cibles à partir de fichiers sources.

À la manière de Make, Bake fonctionne en vérifiant les dépendances entre les fichiers et en exécutant uniquement les commandes nécessaires pour mettre à jour les fichiers obsolètes.

Descriptions des fonctionnalitées

Lancement du programme :

- Pour compiler le programme Bake :

```
make
```

- Pour lancer le programme Bake file:

```
java -jar ./Bake.jar
```

Gestion des variables :

Les variables sont gérées avec un dictionnaire de la classe HashMap()<String,String> et automatiquement substituée aux lignes de commandes les utilisant. Celles-ci sont renseignée dans le Bakefile comme suit :

```
nom = valeur
```

et sont utilisée comme suit :

```
$(nom)
```

Gestion des cibles :

Les cibles sont renseignées comme suit :

```
cible : dépendances
        commandes à exécuter
```

Chaque cible est composée de son nom et de dépendances qui sont exécutées avant la cible.

Gestion des dépendances :

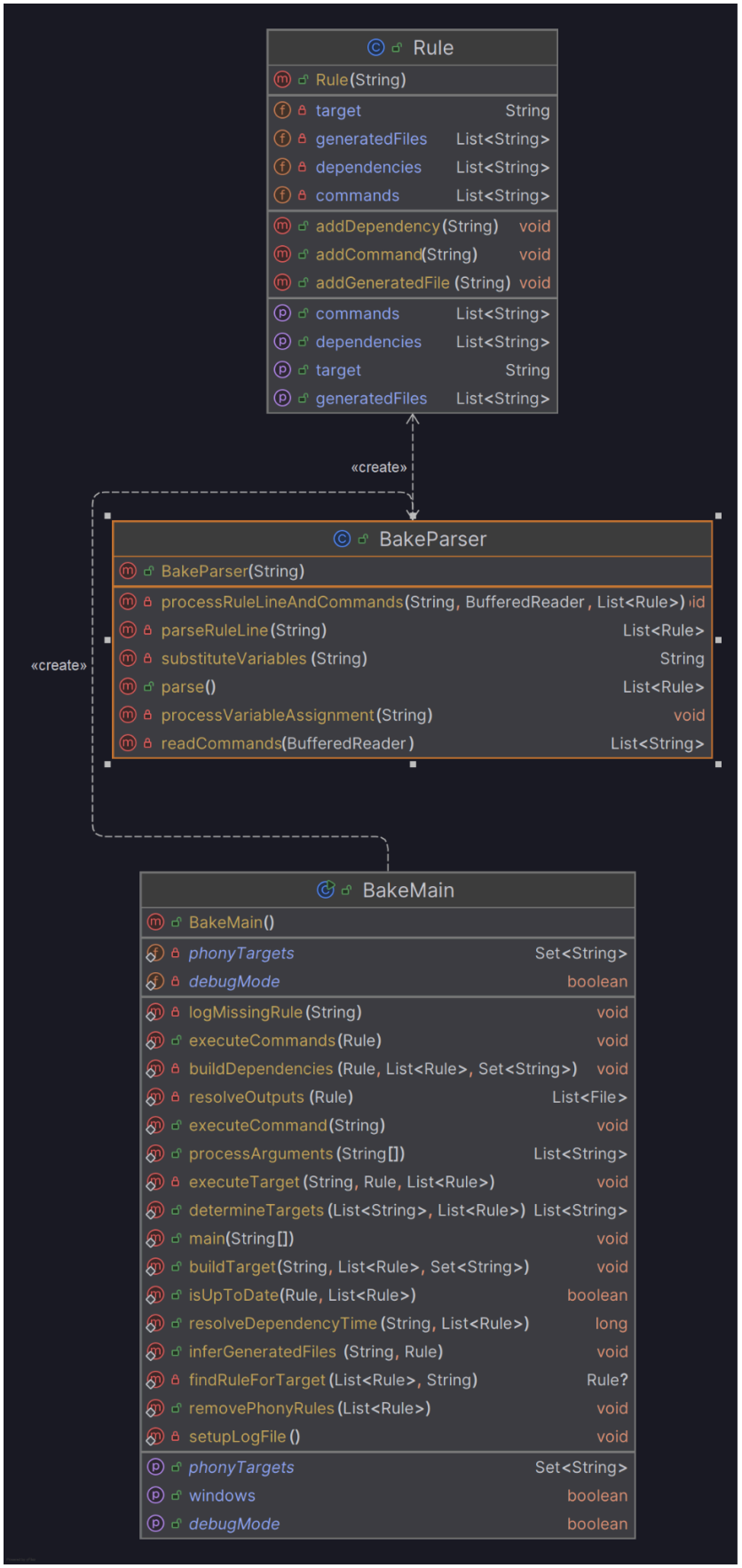
Les dépendances sont gérées et appeler si la cible principale dépend de celle si. Elles peuvent être des cibles ou des fichiers .

Gestion des dépendances circulaires :

Les dépendances circulaires sont des dépendances particulières qui font que deux cibles sont dépendantes entre elles.
Ici, elles sont gérées de sorte à ce que si une cible est appelée une nouvelle fois, celle-ci ne soit pas reprise en compte.

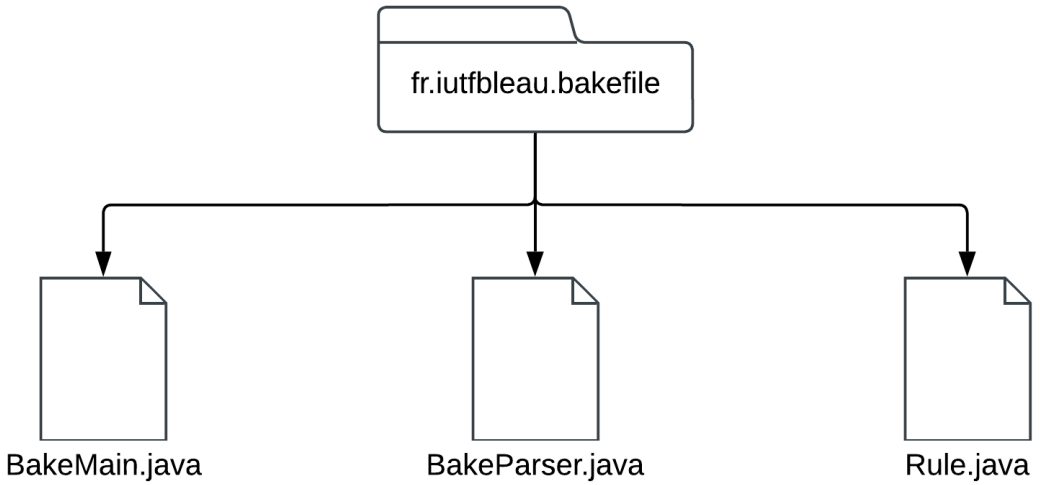
Diagrammes

Diagramme de classe :



Découpage des fichiers

Diagramme du découpage :



- **BakeMain.java** : est la classe principale, elle permet de déterminer les informations d'environnements et d'exécuter les lignes de commandes de chaque cible.
- **BakeParser.java** : est la classe permettant de récupérer les informations relatives aux variables, cibles et dépendances.
- **Rule.java** : permet de stocker les informations d'une cible pour accéder plus facilement aux informations de celle-ci.

Explications du développement

Gestion des dépendances circulaires :

Les dépendances circulaires sont gérées de sorte à ce que si une cible est appelée une nouvelle fois, celle-ci ne soit pas reprise en compte.

Cette détection se fait via un “Set<String>” stockant le nom des cibles qui n’ont pas pu être réalisées à cause de leurs dépendances. et avant de commencer le traitement des dépendances de la cible, on regarde si celle-ci est dans le Set<String> des cibles à réalisées.

Abstractions utilisée :

Nous avons utilisée des classes abstraite :

- List<String>
- List<Rule>
- Set<String>

Avis personnel

Raphaël :

L'utilitaire Make est un programme très utile et complet, mais aussi très floue pour moi, alors pouvoir le recréer m'à permis de mieux comprendre comment il fonctionne.

Basile :

Pour ma part, j'ai trouvé ce projet particulièrement stimulant car, contrairement aux projets précédents qui se limitaient souvent à des interfaces clic dromes, celui-ci m'a réellement permis de plonger dans les tests unitaires. J'ai ainsi pu mettre en pratique les nouvelles méthodes que nous avons découvertes en cours et en entreprise, en concevant et exécutant des tests automatisés qui garantissent la qualité du code. En plus, la répartition des tâches en équipe a permis à chacun de contribuer selon ses expertises, rendant l'expérience collective à la fois enrichissante et motivante.

Thomas :

J'ai trouvé ce projet plus intéressant que le premier projet inspiré de Dorfromantik. Contrairement au précédent, celui-ci ne comportait pas d'aspects liés au Front-end, ce qui signifiait que nous n'avions pas à gérer l'interface visuelle, la disposition des éléments ou l'ergonomie. Cela nous a permis de nous concentrer pleinement sur le développement fonctionnel et de mieux répartir les tâches au sein de l'équipe.

Une fois le développement terminé, nous avons également dû rédiger les tests, ce qui m'a particulièrement plu. En effet, même après avoir finalisé le code, l'écriture des tests permet de s'assurer que tout fonctionne correctement, d'identifier d'éventuelles erreurs et d'apporter les ajustements nécessaires. Cette approche a renforcé la rigueur et la qualité du projet.