UNIVERSITÉ PARIS-EST CRÉTEIL VAL DE MARNE

UNIVERSITÉ PARIS-EST CRÉTEIL

SAE 31_2024 : Bake

Élèves :
Amir KABBOURI
Bamba TOP
Athiran NAGATHURAI

Enseignant :
Luc HERNANDEZ

Table des matières

1 Introduction

1.1 Présentation du projet

Dans le cadre de la SAÉ 32, nous avons développé une version simplifiée de l'utilitaire Make, appelée Bake. Cet outil permet de gérer les dépendances entre fichiers et d'automatiser leur mise à jour à l'aide d'un fichier de configuration spécifique, le Bakefile. Ce rapport présente une description des fonctionnalités principales, la structure du programme, les algorithmes utilisés, ainsi que les tests réalisés pour valider son fonctionnement. Pour plus de détails techniques, les sources et fichiers associés sont disponibles sur le dépôt Git du département.

1.2 Méthodologie de développement

Pour réaliser ce projet, nous avons commencé par analyser les spécifications fournies, en identifiant les fonctionnalités principales comme la gestion des dépendances et la mise à jour des fichiers. Une fois ces bases établies, nous avons réfléchi à l'architecture du programme, en définissant les classes nécessaires et leurs rôles respectifs. Cette organisation a été soutenue par un diagramme de classes simplifié pour mieux visualiser les interactions.

Le développement s'est fait étape par étape, avec des tests réguliers pour valider chaque fonctionnalité ajoutée, notamment la détection des cycles dans les dépendances. Les tests ont été pensés pour couvrir différents cas, comme une compilation depuis zéro ou la gestion d'une modification partielle. Enfin, nous avons utilisé un dépôt privé sur Gitea pour le partage et la gestion du code, ce qui a permis une collaboration fluide et un suivi rigoureux des versions.

2 Fonctionnalités du programme

Le programme Bake intègre plusieurs fonctionnalités essentielles qui permettent de gérer efficacement les dépendances et l'automatisation des mises à jour de fichiers.

2.1 Description détaillée des fonctionnalités

1. Mise à jour des cibles :
Le programme accepte en argument une ou plusieurs cibles à mettre à jour. Si aucune cible n'est spécifiée, la première cible du fichier de configuration Bakefile est utilisée par défaut.

2. Gestion des dépendances :
Chaque cible peut dépendre de fichiers ou d'autres cibles définies dans le fichier Bakefile. Le programme détermine quelles cibles doivent être mises à jour en comparant les dates des fichiers sources et des résultats.

3. Exécution des recettes :
Pour chaque cible à mettre à jour, le programme exécute les recettes associées via un ProcessBuilder. Cela permet de lancer des commandes système directement depuis le programme.

4. Option de débogage :

En utilisant l'option `-d` avant les arguments, le programme affiche des informations détaillées pendant son exécution :

- Les fichiers analysés pour la mise à jour.
- Les comparaisons de dates effectuées et leurs résultats.
- Les fichiers régénérés.

5. Lecture du fichier de configuration :

Le fichier Bakefile contient :

- Les règles de dépendance pour chaque cible.
- Les recettes associées pour les mettre à jour.
- Des commentaires commençant par `"#"` et des variables définies par des affectations avec le symbole `"="`. Ces variables sont substituées directement lors de la lecture du fichier.

6. Gestion des erreurs :

En cas d'échec lors de l'exécution d'une recette, le programme interrompt immédiatement son fonctionnement. Cela garantit l'intégrité des résultats.

7. Détection des dépendances circulaires :

Le programme inclut un algorithme permettant d'éviter les boucles infinies en détectant et en ignorant les dépendances circulaires.

8. Cible spéciale .PHONY :

Le programme supporte la cible spéciale .PHONY, qui permet de spécifier des cibles qui ne correspondent pas à des fichiers réels, mais doivent être exécutées systématiquement.

2.2 Limitations

Pour simplifier l'implémentation, certaines fonctionnalités avancées de Make ne sont pas prises en charge, notamment :

- Les variables d'environnement.
- Les règles implicites et les motifs de règles.
- Les fonctions et les caractères spéciaux pour les recettes.

3 Structure du programme

3.1 Détail de la structure

La structure du programme repose sur plusieurs classes interconnectées :

1. BakeReader : Lit et extrait les informations du Bakefile.

2. Bake : Coordonne le processus global (lecture, création du graphe, exécution des commandes).

3. GrapheDep : Représente un graphe de dépendances entre les cibles et fournit des méthodes pour détecter les cycles et générer l'ordre de construction.

4. BakeExecute : Exécute les commandes des cibles dans l'ordre défini.

5. BakeComparator : Compare les dates de modification des fichiers sources et compilés pour déterminer si une recompilation est nécessaire.

3.2 Diagramme de Classes Simplifié

Voici le diagramme de classes simplifié de notre outil Bake, mettant en évidence les principales relations entre les composants :

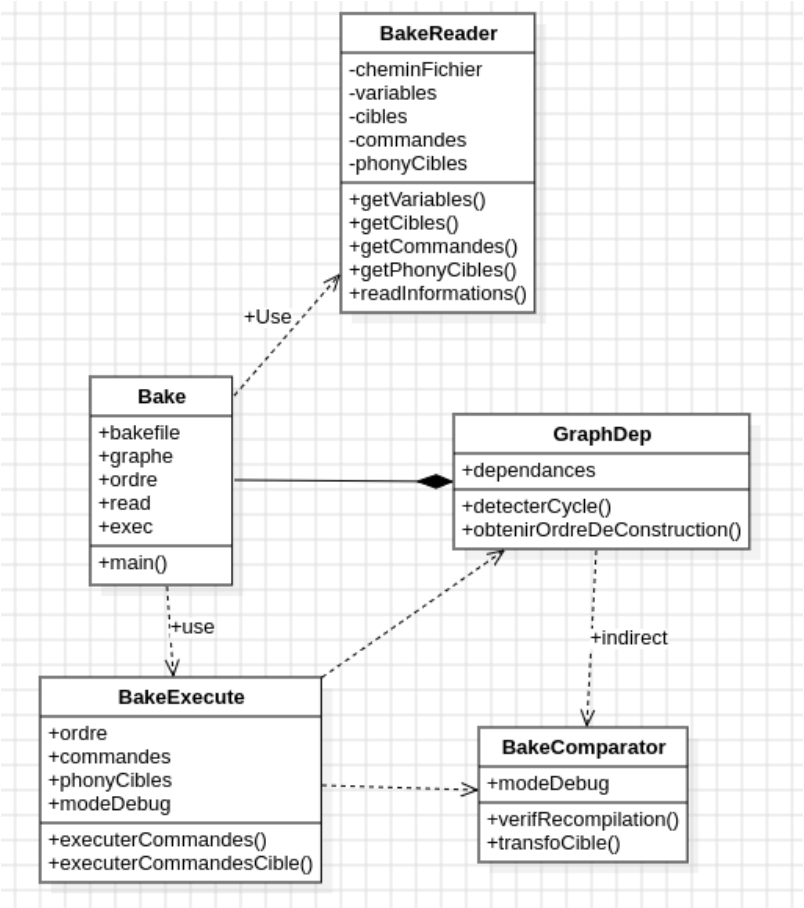

Figure 1 – Diagramme de Classes

4 Explication des classes

1. GrapheDep :

Cette classe est au cœur du programme en ce qui concerne le graphe des dépendances. Elle prend en entrée un ensemble de cibles et leurs dépendances sous forme de HashMap<String, String[]>.

- **Méthodes principales :**
  - detecterCycle() : Vérifie s'il existe un cycle dans les dépendancesentre les cibles. Un cycle dans un graphe dirigé indique unedépendance circulaire (par exemple, la cible A dépend de la cible B, et B dépend de A).
  - obtenirOrdreDeConstruction() : Calcule l'ordre de construction des cibles en utilisant un algorithme de tri topologique. Cela garantit que les cibles seront construites dans un ordre qui respecte toutes les dépendances.

2. BakeReader :

- Cette classe lit le fichier Bakefile et en extrait les informations sur les cibles et leurs dépendances. Elle stocke ces informations sous forme de HashMap<String, String[]> (pour les cibles et leurs dépendances).
- Elle remplace également les variables dans les commandes par leurs valeurs correspondantes.

3. Bake :

- La classe Bake coordonne le processus global. Elle crée un graphe de dépendances à partir des informations lues par BakeReader, détecte les cycles et génère un ordre de construction des cibles.
- Elle exécute ensuite les commandes en respectant cet ordre, en appelant la classe BakeExecute.

5 Exposition de l'algorithme

L'objectif de cet algorithme est de pouvoir identifier s' il y a un cycle dans les dépendances des cibles. Un cycle veut dire qu'une cycle dépend directement ou indirectement d'elle-même, ce qui fait que ce n'est pas possible. Par exemple : A $\rightarrow$ B $\rightarrow$ A Pour compiler A il faut compiler B mais pour compiler B il faut compiler A, on remarque bien que ici il y a un problème.

5.1 Principe de l'algorithme

L'algorithme que nous avons utilisé est l'algorithme du parcours en profondeur (DFS) parce qu' il est facile à appliquer pour ce cas précis et qu'il évite d'explorer des chemins inutilement, chaque cible est décrite comme :

- **Visité** : Une cible qui a déjà été complètement explorée et validée.
- **En cours d'exploration** : Une cible qui est en cours d'exploration. Si on revient sur une cible encore "en cours", ça veut dire qu'il y a un cycle.

5.2 Fonctionnement pas à pas

Étape 1 : Initialisation

La méthode detecterCycle initialise deux ensembles :

- Visites : Une cible qui a déjà été complètement explorée et validée.
- enCours : Une cible qui est en cours d'exploration. Si on revient sur une cible encore "en cours", ça veut dire qu'il y a un cycle.

Elle parcourt toutes les cibles du graphe pour vérifier si un cycle existe, en appelant la méthode récursive graphVerifCycle pour chaque cible.

Étape 2 : Exploration récursive

La méthode graphVerifCycle explore chacune des cibles et leurs dépendances. Voici les différentes étapes de cette exploration :

(a) Détection d'un cycle .

- Si la cible est dans enCours, ca signifie qu'elle est déjà en cours d'exploration. Un cycle est détecté.

(b) Vérification des cibles déjà visitées.

- Si la cible est dans visites, cela signifie qu'elle a déjà été explorée et validée (pas de cycle pour cette cible). On peut l'ignorer.

(c) Marquage de la cible.

- La cible est marquée comme en cours d'exploration et ajoutée aux cibles visitées.

(d) Exploration des dépendances.

- Pour chaque dépendance de la cible :
  - L'algorithme appelle récursivement graphVerifCycle pour vérifier s'il existe un cycle dans cette branche du graphe.

(e) Nettoyage après exploration.

- Une fois toutes les dépendances explorées, la cible est retirée de enCours, car sa branche est terminée.
- Si aucun cycle n'a été détecté, la méthode retourne false.

6 Énumération des structures de données

Dans ce projet Bake, nous avons recours à plusieurs structures de données abstraites couramment étudiées.

6.1 Dictionnaires (HashMap)

- **HashMap<String, String[]> cibles** : Utilisé pour stocker les cibles et leurs dépendances.
- **HashMap<String, String> variables** : Utilisé pour stocker les variables et leurs valeurs dans le fichier Bakefile.
- **HashMap<String, String> commandes** : Utilisé pour stocker les commandes associées à chaque cible.
- **HashMap<String, String[]> dependances** : Utilisé dans la classe GrapheDep pour représenter le graphe des dépendances (les clés sont les cibles, les valeurs sont les cibles dont elles dépendent).

6.2 Listes (ArrayList)

- **ArrayList ordre** : Utilisé pour stocker l'ordre de construction des cibles, une fois que toutes les dépendances ont été résolues.
- **List cibles** : Utilisé pour gérer les cibles à traiter dans la classe Bake.

6.3 Arbres (Graphes Orientés)

- Bien que ce ne soit pas un arbre traditionnel avec une hiérarchie strictement binaire, nous utilisons un graphe orienté où chaque cible peut avoir plusieurs dépendances. Ce graphe est représenté par une HashMap<String, String[]> qui associe chaque cible à ses dépendances.

6.4 Récursivité

La récursivité est utilisée dans les méthodes suivantes :

- graphVerifCycle (pour détecter les cycles dans le graphe des dépendances).
- graphOrdre (pour générer l'ordre de construction des cibles, en parcourant les dépendances de manière récursive).

7 Conclusion personnelle

7.1 Amir Kabbouri

7.2 Bamba Top

J'ai trouvé ce projet trés amusant à réaliser. En effet, c'est un projet qui n'a pas été aussi dur que les précédents mais qui nécessite une réflexion commune sur la structuration de donnée. Ce projet m'a permis de combler quelques lacunes concernant le développement en java. J'espère en apprendre autant sur les projets à venir. C'était top !

7.3 Athiran Nagathurai

Cette SAE m'a permis de mieux comprendre comment fonctionnent les systèmes de compilation, un sujet que je trouvais assez abstrait avant de commencer. Travailler sur "Bake" a été parfois compliqué, surtout pour gérer les dépendances et détecter les cycles, mais cela m'a appris à structurer mon code et à mieux organiser mes idées.

En bref, cette SAE m'a aidé à mieux m'organiser, à collaborer efficacement et à prendre confiance en mes compétences techniques, tout en me montrant qu'il reste encore beaucoup à apprendre.