RAPPORT

UTILITAIRE DE COMPILATION BAKE

IZANIC Hugo

GALLEGO Julian

AISSI Jude-Christ

SOMMAIRE

IZANIC Hugo

GALLEGO Julian

AISSI Jude-Christ

INTRODUCTION

Dans le cadre de ce projet, nous avons développé Bake, un utilitaire de compilation inspiré de la commande make. L'objectif est de permettre la gestion automatisée de la compilation de fichiers à partir d'un fichier de configuration nommé Bakefile, en suivant un format simplifié par rapport au Makefile classique.

Le programme, entièrement développé en Java, respecte la contrainte de n'utiliser que l'API officielle, sans bibliothèques externes. Il a été conçu pour lire un fichier Bakefile, identifier les cibles à compiler, leurs dépendances, et exécuter les commandes associées uniquement lorsque les fichiers sources ont été modifiés. Un mode débogage optionnel permet d'afficher des informations détaillées sur le processus de compilation, et le programme intègre également un mécanisme de détection des dépendances circulaires pour éviter les boucles infinies.

Le développement a été réalisé en équipe sur le serveur Gitea du département, où nous avons suivi l'avancement du projet et la répartition des contributions à l'aide de commits réguliers.

Dans ce rapport, nous présenterons d'abord les principales fonctionnalités du programme et son mode de fonctionnement. Nous détaillerons ensuite l'organisation du code à travers un diagramme de classes simplifié et expliquerons l'algorithme utilisé pour détecter les dépendances circulaires. Enfin, nous aborderons les tests réalisés avant de conclure sur les résultats obtenus et les enseignements tirés du projet.

IZANIC Hugo
GALLEGO Julian
AISSI Jude-Christ

FONCTIONNALITÉS DU PROGRAMME

LECTURE DU BAKEFILE

Le **programme Bake** est capable de lire et d'interpréter un fichier de configuration nommé **Bakefile**. Ce fichier décrit les **cibles** à **compiler**, leurs dépendances et les commandes (recettes) à exécuter pour générer les fichiers correspondants. Il supporte :

Les cibles : fichiers ou actions à produire.

- **Les dépendances :** fichiers requis pour générer une cible.
- **Les recettes :** commandes à exécuter pour compiler la cible.
- **Les commentaires :** lignes commençant par # sont ignorées.
- **Les variables simples :** déclarées avec = et remplacées dès leur lecture.

MISE À JOUR DU DES FICHIERS

Le programme ne **recompile** un fichier que si nécessaire, en **comparant** les **dates** de modification des fichiers cibles et sources. Lorsqu'une cible est plus ancienne que l'un de ses fichiers sources, elle est automatiquement mise à jour.

Exemple de comportement :

- **main.java** modifié après la dernière compilation.
- **Bake** détecte que main.class est obsolète et le recompile.

Ce mécanisme évite des compilations inutiles et optimise les performances du programme.

MODE DÉBOGAGE

L'option -d (débogage) permet **d'afficher** des informations détaillées pendant l'exécution du programme pour aider à **comprendre son comportement :**

- Quel cibles est analysé.
- Comparaison des dates des fichiers et cibles.
- Liste des fichier recompilé.

IZANIC Hugo
GALLEGO Julian
AISSI Jude-Christ

FONCTIONNALITÉS DU PROGRAMME

DÉTECTION DES DÉPENDANCES CIRCULAIRES

Bake intègre un **mécanisme de prévention** des dépendances circulaires, c'est-à-dire **lorsqu'une cible dépend indirectement d'elle-même**, ce qui pourrait entraîner une boucle infinie.
Pour **ajouter un nœud en tant que fils**, le programme s'assure que le nœud fils n'a pas, directement ou indirectement, le nœud parent comme descendant

```
public Noeud ajouter(Noeud n) {
    if (!n.getAllFils().contains(this) && !this.getFils().contains(n)) {
        fils.add(n);
    }
    return n;
}
```

GESTION DES ERREURS

En cas d'échec d'une commande dans une recette, **le programme arrête immédiatement l'exécution et signale l'erreur**. Cette approche garantit la fiabilité du processus de compilation.

IZANIC Hugo
GALLEGO Julian
AISSI Jude-Christ

STRUCTURE DU PROGRAMME

Le programme Bake est organisé en plusieurs étapes logiques, chacune prise en charge par une **composante dédiée**. Cela garantit une **séparation claire des responsabilités** et une meilleure maintenabilité.

PRÉSENTATION DES PRINCIPALES CLASSES :

BakefileParser : lecture du fichier configuration Bakefile.

Cette classe est chargée de lire et **d'interpréter le fichier de**

- Elle **extrait les cibles,** leurs dépendances, et les commandes associées (recettes).
- Elle **gère les variables** à l'aide de la classe Variable simples, les commentaires et valide le format du fichier.

Noeud: gestion des cibles et dépendances.

- **Une fois les données extraites, elles sont structurées dans un graphe orienté.**
  - **Chaque nœud** représente **une cible.**
  - **Les arêtes représentent les relations** entre les cibles et leurs dépendances.
  - Ce graphe permet de :
    - Déterminer l'ordre de compilation (tri topologique).

Variables: stocke et gère les variables dans un dictionnaire

IZANIC Hugo
GALLEGO Julian
AISSI Jude-Christ

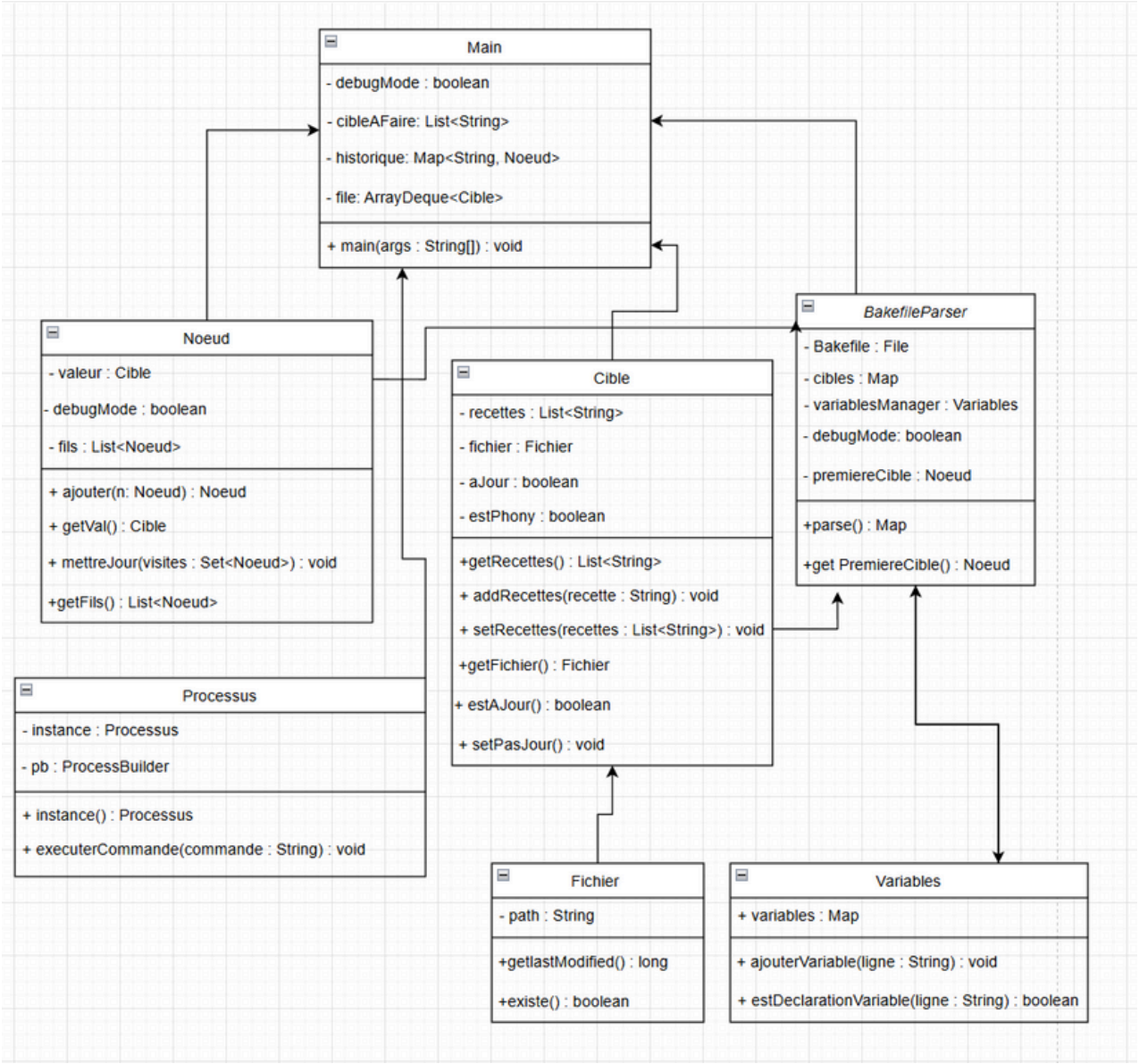
STRUCTURE DU PROGRAMME

Processus exécution des commandes.

Une fois les **cibles identifiées** pour **recompilation**, les commandes associées (recettes) sont exécutées.

- **Les commandes sont lancées via ProcessBuilder.**
- **En cas d'échec, l'exécution s'arrête immédiatement.**

DIAGRAMME DE CLASSE



IZANIC Hugo
GALLEGO Julian
AISSI Jude-Christ

Énumération des structures de données

Graphe(arbre): ranger les cibles sous forme de noeud

Dictionnaire: stockes les **variables** et les **noeuds**

Liste: pour stocker les fils d'un noeud

File: enfiler les cibles a exécuter afin garder le bon ordre

IZANIC Hugo

GALLEGO Julian

AISSI Jude-Christ

CAS DE TEST

Pour **garantir la robustesse** et la **fiabilité** de notre programme Bake, **nous avons conçu une série de tests** couvrant les cas d'utilisation les plus courants et les situations exceptionnelles.

COMPILATION COMPLÈTE DEPUIS ZÉRO

Pour ce cas : Voici notre démarche

- Objectif : Vérifier que tous les fichiers cibles sont générés correctement lorsque rien n'existe.
- Procédure : Fournir un Bakefile avec plusieurs cibles et lancer Bake.
- Résultat attendu : Toutes les recettes s'exécutent, et les cibles sont générées.

FICHIERS DÉJÀ À JOUR

Pour ce cas : Voici notre démarche

- L'Objectif : S'assurer que le programme ne recompile pas inutilement des cibles déjà à jour.
- Procédure : Lancer Bake alors que tous les fichiers cibles sont plus récents que leurs dépendances.
- Résultat attendu : Aucune commande n'est exécutée.

SOURCE MODIFIÉE DEPUIS LA DERNIÈRE COMPILATION.

Pour ce cas : Voici notre démarche

- Objectif : Tester la recompilation sélective des cibles affectées par une modification.
- Procédure : Modifier une dépendance d'une cible spécifique et relancer Bake.
- Résultat attendu : Seule la cible dépendante est recompilée.

IZANIC Hugo
GALLEGO Julian
AISSI Jude-Christ

CAS DE TEST

Rappelons qu'il est important d'exécuter le fichier refresh.sh pour remettre les cas de test en situation
donc la ligne de commande ./refresh dans le dossier de test

README

le script refresh.sh permet de remettre les cas de test en situation

./refresh.sh pour executer

1 --> Compilation depuis rien. Aucun .class n'existe

2 --> une compilation où le résultat existe déjà

3 --> une compilation où un résultat existe déjà, mais une source a été modifiée

4 --> une compilation où une dépendance circulaire est présente.

extrait du bakefile pour le test4:

${nom}.class: ${nom}.java ${nom}2.class

${nom}2.class:${nom}2.java ${nom}3.class ${nom}.class

${nom}.class dépend de ${nom}2.class qui lui meme depend de ${nom}.class

5 --> même exemple que le 4. mais avec un probleme de recette

6 --> Test compilation du projet en lui même

ERREUR DANS UNE RECETTE

Pour ce cas : Voici notre démarche:

- Objectif : S'assurer que le programme s'arrête immédiatement en cas d'erreur dans une recette.
- Procédure : Inclure une commande incorrecte dans une recette et lancer Bake.
- Résultat attendu : Le programme signale l'erreur et interrompt son exécution.

CONCLUSION

Izanic Hugo

Ce projet a été une opportunité unique de concevoir un outil fonctionnel en partant de zéro, tout en respectant des contraintes claires. J'ai particulièrement apprécié le fait de travailler sur des fonctionnalités concrètes, où chaque ligne de code avait une utilité directe et visible dans le résultat final. Cela m'a permis de mieux comprendre l'importance de concevoir des solutions simples, efficaces et faciles à maintenir.
De plus, ce projet m'a donné l'occasion de m'immerger dans des concepts essentiels comme la gestion des dépendances, la détection des cycles et l'automatisation des processus. C'était aussi l'opportunité de voir à quel point une bonne organisation et une réflexion préalable sont essentielles pour éviter les écueils dans des projets plus complexes.

GALLEGO Julian

Ce nouveau projet a permis de mettre en oeuvre les nouvelles notions vue en cours comme les arbres, les files ou bien les listes ou biens les dictionnaires. J'ai eu un peu de mal à comprendre comment conceptionner le programme pour répondre aux consignes mais des que j'ai compris je n'ai pas eu de difficulté particulière.

AISSI Jude-Christ

Ce projet m'a permis de renforcer mes compétences en programmation, notamment sur la gestion des dépendances et la détection des cycles dans un graphe. J'ai également appris à collaborer efficacement en équipe grâce à l'utilisation de Gitea.
Ce projet m'a fait apprécier la rigueur nécessaire à la création d'un programme robuste, capable de gérer des cas d'erreurs variés tout en restant fonctionnel et intuitif. Ce type de défi me motive à continuer d'explorer des problématiques techniques similaires dans mes futurs projets.