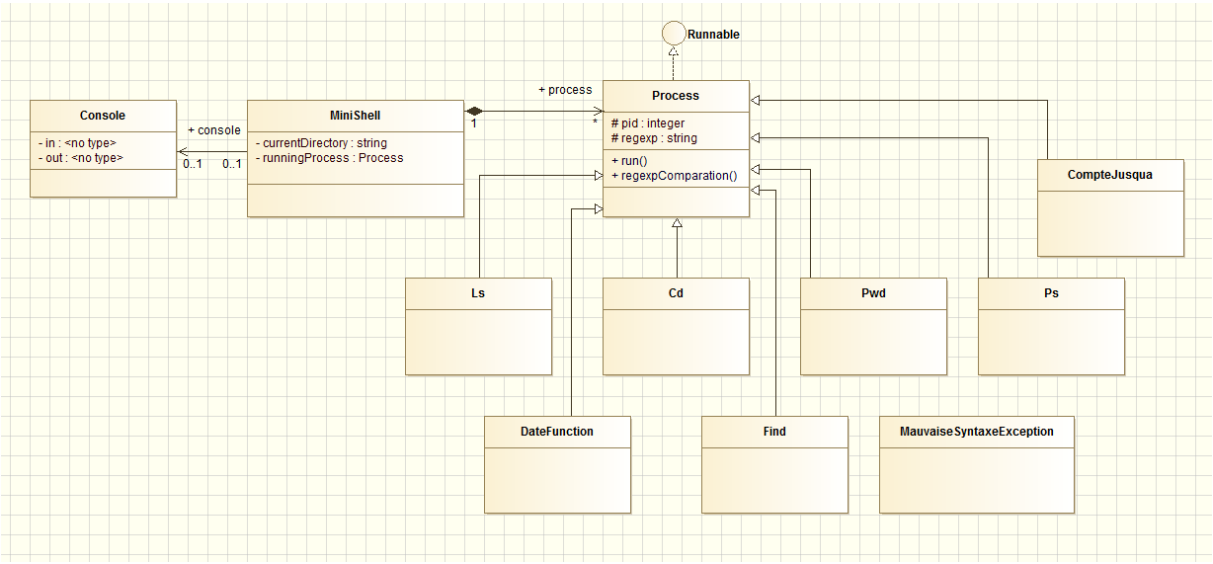
PROJET CPOO n°2: Petite invite de commandes (minishell)

Hafça TIRICHINE et Benjamin VIC

I. Diagramme des classes :



Comme le montre le diagramme, notre implémentation ce fait autour de 3 classes principales : **Process**, **Minishell** et **Console**.

Nous avons donc partager le code en deux packages : process et shell.
Dans process nous allons trouver toutes nos commandes, leur implémentation, mais aussi les classes tests et une exception;
Dans shell nous avons les 3 classes spécifiées ci-dessus.

II. Package shell:

La classe Process :

Celle-ci implémente l'interface Runnable et a pour attributs :

- un AtomicInteger **pid** (le choix de l' AtomicInteger permet d'avoir un pid distinct pour chaque Thread de commandes)
- un entier **currentProcessPid** (qui donne le pid du processus courant)
- une String **regexp** (qui donne l'expression régulière propre à chaque commande)
- une String **commande** (dans laquelle on stocke la commande renvoyer par l'utilisateur).

C'est cette commande que nous allons comparer avec l'expression régulière regexp dans la **fonction regexp().**

- une String **localDir** (dans laquelle on stocke l'emplacement actuel de l'utilisateur dans son arborescence).

On y a ajouté une **fonction purgeEmptyString** qui permet de récupérer la première cellule non-vide de la chaine de caractères tapée par l'utilisateur en ligne de commande.

Tous les autres processus de notre Minishell héritent de cette classe.

La classe Minishell :

Cette classe a pour principale vocation d'exécuter les commandes.
Elle prend comme attributs le répertoire actuel (**currentDir**) et une map de Process, Future<Void>.
L'idée est de stocké chaque processus crée avec un Future qu'on lui associe afin d'avoir accès aux threads plus rapidement.

Dans la **fonction processMatcher** on récupère le nom de la commande tapée ;
On crée un ExecutorService ce qui nous permet de gérer 10 threads en même temps (nombre choisi arbitrairement) ;
Puis avec un switch on génère le processus correspondant avec un Future que nous allons ajouter à notre map de process.

La classe Console :

III. Package process:

Les classes de ce package fonctionnent à peu près de la même manière; On prend la commande tapée par l'utilisateur;
Si nécessaire (pour **cd**, et **find** par exemple), on coupe la chaine afin de séparer les arguments, puis on traite la commande (pour **ls** on va directement faire un affichage des fichiers/répertoires du répertoire courant, pour **ps** on va prendre la map de process et pareil, faire un affichage des processus car elles ne prennent pas d'arguments).
Nous avons aussi implémenter **pwd**, **date** (dont la classe est appelée DateFunction pour ne pas empiéter sur la classe Date déjà présente dans l'API Java), **compteJusqua** et **kill** (seule commande qui n'hérite pas de process et que nous avons implémenter sous forme de fonction dans la classe Minishell car kill lui même n'est pas un processus, c'est une commande qui va agir sur les processus).

Comme elles héritent de Process, elles implémentent aussi Runnable et c'est donc dans la **fonction run** que nous allons faire tout le travail (cette fonction étant appelée par défaut lors de la création de nos processus).

(Dans ce package nous avons aussi les classes de tests RegexpTest et RegexParamTest et l'exception MauvaiseSyntaxeException.)