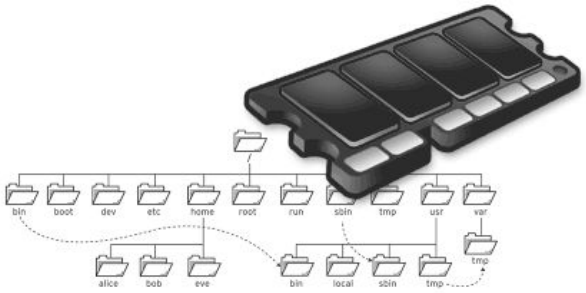

M1 GIL – Mini-projet d'architecture logicielle 2021–2022
Un système de fichiers Web en mémoire

F. Nicart

À rendre avant le **9 mai 2022**

1 Présentation générale du mini-projet

Un « système de fichiers en mémoire » est un système de fichiers dont le contenu est stocké uniquement dans la mémoire vive d'un ordinateur. Son contenu est perdu, s'il n'est pas explicitement sauvegardé, lors du redémarrage ou de l'arrêt de la machine. Ils sont en général utilisé pour du stockage temporaire (par exemple pour `/tmp`) pour éviter de fatiguer le dispositif de stockage principale de la machine (comme une carte SD sur un nano-ordinateur) ou bien pour augmenter significativement les performances de logiciels (par exemple pour les fichiers intermédiaires d'un compilateur sur des très gros projets).

On se propose ici de réaliser un tel système de fichiers, simplifié, en le dotant d'une interface de manipulation Web légère permettant de parcourir et de modifier le système de fichiers en mémoire.

2 Spécifications

2.1 Le système de fichiers

Le système de fichiers sera constitué des trois concepts suivants :

— Les fichiers réguliers qui sont caractérisés par : leur nom, leur taille (en octets) et leur contenu brute (bytes). Il ne sera pas fait de distinction sur la nature des fichiers.

— Les dossiers, caractérisés par leur nom, leur contenu (les éléments qu'ils contiennent) et leur taille (la somme des tailles de leurs éléments).

— Des liens symboliques, caractérisés par leur nom et une chaîne de caractères représentant le chemin de la cible du lien. Ils ont une taille nulle.

Le nom de ces éléments (fichiers réguliers, dossiers et liens symboliques) sera limité à la classe de caractères `[a-zA-Z0-9._]` avec une longueur comprise entre 1 et 20. Il n'y a pas de contrainte syntaxique sur les noms (par exemple la notion d'extension n'existe pas, un nom qui commencerait par un point n'a pas de sens particulier, *etc.*). Plus précisément, la syntaxe des noms est : `[a-zA-Z0-9._]+`. Un chemin sera une séquence de noms séparés par un symbole `/`.

Les fichiers réguliers peuvent contenir un nombre arbitraire d'octets. Leur contenu sera stocké en mémoire dans un simple tableau de `bytes`. La nature du contenu n'a aucune importance (texte, données binaires, format particulier, *etc.*).

Les dossiers peuvent contenir un nombre illimité d'éléments avec une profondeur (imbrication de dossiers) illimitée. Toutefois, un dossier ne peut contenir deux éléments portant le même nom.

Les liens symboliques contiennent uniquement un chemin pointant un élément du système de fichiers depuis la racine. Un lien symbolique peut ne pas être valide. C'est-à-dire que la cible, ou une partie du chemin vers la cible, peut ne pas exister. Dans ce cas, une erreur sera alors déclenchée au moment de son utilisation uniquement, mais il sera stocké et affiché normalement.

La bibliothèque permettant de gérer le système de fichiers en mémoire stockera les informations en mémoire uniquement, par simple allocation dynamique. Toutefois, en entrevoit qu'à l'avenir la bibliothèque pourrait proposer d'autres modes de représentation en mémoire des éléments de fichiers. On souhaite donc que la bibliothèque soit conçue de manière à pouvoir intégrer facilement ces variantes d'implémentation d'un système de fichiers en mémoire.

2.2 L'interface Web

Pour manipuler le système de fichiers en mémoire, on propose d'utiliser une interface web rudimentaire, à la manière d'un micrologiciel. Pour cela, afin d'éviter l'emploi d'une plateforme lourde comme `JEE`, on pourra faire appel au paquetage `com.sun.net.httpserver` [1] pour gérer les communications http.

Pour faire simple, on propose le comportement suivant :

- chaque élément du système de fichiers est associé à une URL permettant d'y faire référence.
- Un requête `GET` (sans paramètre) sur une telle url permet d'obtenir la ressource associée : affichage du contenu si c'est un dossier, téléchargement du contenu si c'est un fichier, l'un des deux si c'est un lien symbolique en fonction de la nature de la cible si elle existe.
- À partir du listing d'un dossier, cliquer sur le nom d'un des éléments contenus doit déclencher une de ces opérations `GET`. On rappelle que pour forcer un navigateur à sauvegarder la réponse quelque soit sa nature, il suffit d'utiliser le type MIME `application/octet-stream` dans la réponse.
- Toujours à partir du listing d'un dossier, ou pourra y faire trois opérations :
  - y déposer un fichier,
  - y créer un sous-dossier,
  - y créer un lien symbolique (par saisie d'une simple chaîne de caractères).
- Enfin, pour faciliter l'ajout de contenu dans le système de fichiers en mémoire, on se propose d'ajouter une option en ligne de commande au programme qui permettra de charger (récursivement) le contenu d'un dossier spécifié au démarrage du serveur.

3 Modalités

- Le travail est à réaliser en binôme maximum [2].
- L'implémentation et le compte-rendu sont à rendre avant le **9 mai 2022**.

4 Travail à rendre

Vous fournirez une archive nommée `projet-al-nom1-nom2.zip` comportant un dossier éponyme. Celui-ci contiendra le code source de votre implémentation, votre rapport au format `PDF`.

Il est important de noter que le rapport aura autant d'importance que l'implémentation. Celui-ci devra contenir :

- Une explication de l'architecture globale illustrée par un ou plusieurs diagrammes UML.
- Pour chaque patron mis en oeuvre : la motivation du choix du patron, le diagramme UML de la partie de votre application correspondant au patron instancié (en prenant soin d'y indiquer les acteurs du patron).
- Éventuellement, une liste des patrons que vous avez hésité à utiliser et les raisons pour lesquelles vous les avez écartés.
- Le moins de fautes d'orthographe et de français possible. (Relisez-le !)

Une attention particulière sera portée à la précision sémantique des noms de vos entités (à vos dictionnaires d'anglais !), ainsi qu'à l'organisation du code en paquetages.

Un autre point **important** à garder à l'esprit est que ce mini projet est un prétexte à l'élaboration d'une architecture de taille raisonnable mais respectant les bons principes vus en cours. Par conséquent, le logiciel n'a pas vocation à être abouti ni ergonomique.

On pourra aussi utiliser des bouchons pour les fonctions pour lesquelles le temps a manqué en le mentionnant dans le rapport et en décrivant le travail à réaliser pour compléter la fonctionnalité, en particulier pour une fonctionnalité pour laquelle une autre fonctionnalité a été développée utilisant le même mécanisme/patron.

Par exemple, si la partie Web venait à vous poser trop de difficultés, un programme équivalent prenant ses instructions en ligne de commande pourra être accepté, la partie système de fichiers en mémoire étant suffisamment riche. Si la gestion du `multipart` pour le téléversement vous pose trop de difficultés, des alternatives seront acceptées (ce n'est pas un projet de Web).

1. `https://docs.oracle.com/javase/8/docs/jre/api/net/httpserver/spec/com/sun/net/httpserver/package-summary.html`
2. Les singletons sont acceptés ;-). Ceci dit le travail en binôme est recommandé pour faire du *pair-designing*.

5 Quelques indications

— Vous soignerez la modularité de votre application et justifierez dans votre rapport la répartition en paquetages.
— (Comme d'habitude,) vous soignerez la rédaction de vos interfaces. Vous détaillerez leur conception dans votre rapport.
— Vous fournirez, en annexe du rapport, la liste complète, par paquetage, des entités de votre application. Vous donnerez en face du nom de chaque entité sa responsabilité dans l'application en une phrase.
— Vous produirez la javadoc de l'ensemble de votre application que vous fournirez, après l'avoir lue, dans un sous-dossier intitulé `javadoc` de votre archive.