

Cette application permet de calculer une suite et de l'enregistrer dans une structure de données de type liste chaînée.

Développement de l'application

Les classes

Nous avons fait le choix de séparer notre code en deux classes : `ListeChaine` et `Suite`. La première permet uniquement de créer et manipuler une liste chaînée et peut être réutilisée par d'autres programmes. La seconde représente une suite, enregistrée dans un objet `ListeChaine`, et permet de faire les calculs et gérer le fichier de sauvegarde.

La sauvegarde

Pour nos fichiers de sauvegarde, nous avons utilisé les fichiers `properties` qui sont nativement gérés par JAVA. Ceux-ci permettent de lire et d'enregistrer facilement des couples clé-valeur dans un fichier texte grâce à des fonctions incluses dans JAVA.

Lorsque le fichier existe déjà, il n'est pas écrasé et les calculs, s'ils ne sont pas terminés, reprennent. Cependant, si l'option `emptyState` est à `true`, les valeurs de sauvegarde sont effacées et les calculs reprennent du début.

Nous avons aussi fait le choix, dans le cas où les deux valeurs initiales passées en paramètres diffèrent des deux dernières valeurs contenues dans le fichier, de continuer les calculs avec les deux valeurs passées en paramètre mais de ne pas enregistrer ces deux valeurs dans la liste.

On notera qu'après chaque calcul, c'est-à-dire après chaque nouvelle valeur dans la liste, une sauvegarde dans le fichier `properties` associé est effectuée.

Les opérateurs

Quatre opérateurs différents sont disponibles : addition (`add`), soustraction (`sub`), multiplication (`mult`) et division (`div`). Afin de s'affranchir des opérateurs `+`, `-`, `*` et `/`, nous avons programmé chacune de ces fonctions.

- La fonction `add(a, b)` est celle donnée dans l'énoncé du TD et consiste à chaque tour de boucle à décrémenter la valeur `b` et à incrémenter la valeur de `a`.
- La soustraction est effectuée grâce à `add`, en passant l'opposé de la première valeur en paramètre, c'est-à-dire : `sub(a,b) = add(-a,b)`.
- La multiplication `mult(a, b)` utilise elle-aussi l'addition. Pour obtenir le résultat, on fait `b` tours de boucle qui ajoutent à chaque tour `a` au total.
- La division `div(a, b)` est une division euclidienne entière, `a/b`, et consiste à soustraire `a` à `b` tant que le total reste positif. Le nombre de soustractions correspond au quotient recherché.

Gestion des erreurs

Lorsqu'un comportement inhabituel est détecté, nous levons une exception avec un message d'information ce qui induit un arrêt des calculs et une terminaison du programme.

Cependant, puisque après chaque calcul, une sauvegarde est effectuée, il est relativement aisé de recommencer des calculs non terminés.

Conception des tests

Chaque cas de tests est représenté par une méthode JUnit.

Boite Noire

Nous allons préciser ci-après notre conception des tests de type boite noire que nous avons appliqué à notre projet.

Nous testons dans la classe `SuiteChaine` la méthode `void build(String path, String op, int val1, int val2, int taille, Boolean etatVide)` en nous basant sur le retour de la méthode `boolean isValid(String path)`.

Partitionnement en catégories

Nous avons partitionné les entrées en catégories :

- `p : path`
 - fichier existe [**don't care**]
 - fichier n'existe pas [**don't care**]
- `o : op`
 - `add` [**opNonDiv**]
 - `soust` [**opNonDiv**]
 - `mult` [**opNonDiv**]
 - `div` [**div**]
 - *autre valeur* [**erreur**]
- `v1 : val1`
 - integer sauf 0 [**don't care IF opNonDiv**]
 - 0 [**IF (opNonDiv OU (div AND (t3 OR to_2))), don't care IF opNonDiv**]
- `v2 : val2`
 - integer sauf 0 [**don't care if opNonDiv**]
 - 0 [**IF (opNonDiv OU (div AND to_2)), don't care IF opNonDiv**]
- `t : taille`
 - `t <= 0` [**erreur**]
 - `0 < t <= 2` [**to_2 don't care IF opNonDiv**]
 - `t = 3` [**t3 don't care IF opNonDiv**]
 - `3 < t <= 10` [**don't care IF opNonDiv**]
 - `t > 10` [**erreur**]
- `e : etatVide`
 - `true` [**don't care**]
 - `false` [**don't care**]

Justifications

- `p : path` : Peu importe si le nom de fichier existe ou non, si il n'existe pas on créera un nouveau. Sinon on utilisera l'ancien. Cela n'a pas d'influence sur la validité de la chaine finale ; d'où la propriété **don't care**.
- `o : op` : seul la division va demander de l'attention dans le sens où les autres valeurs seront aussi importantes : on devra faire attention quand `val1` et `val2` sont à 0 car à partir d'une certaine taille, on tombera sur une division par 0, c'est à dire un erreur. On a pas ce problème avec les autres opérations qui

ne seront pas impactées par les différentes valeurs de val1, val2 et taille ; d'où le don't care IF opNonDiv.

- v1 : val1 : confer remarque précédente pour o
- v2 : val2 : confer remarque précédente pour o
- t : taille : confer remarque précédente pour o
- e : etatVide : élément indépendant des autres.

Each Choice (EC)

Cas de test

Test	Path(Valeur)	Opération (Valeur)	Val1 (Valeur)	Val2 (Valeur)	Taille (Valeur)	etatVide (Valeur)	Resultat Attendu	Résultat du test
EC1	p1("config.properties")	o1("add")	v1-1(3)	v1-2(5)	t1(0)	e1(True)	Error	Failed
EC2	p1("config.properties")	o1("add")	v1-1(3)	v1-2(5)	t2(2)	e1(True)	True	Ok
EC3	p2("test.properties")	o2("soust")	v1-2(0)	v2-2(0)	t3(3)	e2(False)	True	Failed
EC4	p1("config.properties")	o3("mult")	v1-1(3)	v2-1(5)	t4(10)	e1(True)	True	Ok
EC5	p1("config.properties")	o4("div")	v1-2(0)	v2-2(0)	t2(2)	e1(True)	True	Ok
EC6	p1("config.properties")	o1("add")	v1-1(3)	v1-2(5)	t5(15)	e1(True)	Error	Failed
EC7	p1("config.properties")	o5("abcd")	v1-1(3)	v1-2(5)	t1(0)	e1(True)	Error	Failed

Interprétation

Le test EC1 échoue car on attend qu'une exception soit levée en cas de taille incorrecte. Ce n'est pas le cas ici, il y a seulement un message d'avertissement qui ne termine pas le programme.

Le test EC3 échoue car une exception FileNotFoundException est levée. Or, le programme devrait créer automatiquement le fichier properties s'il n'existe pas.

Les tests EC6 et EC7 échouent car ils devraient lever une exception pour respectivement, une taille trop grande et un opérateur inconnu. En réalité, ils affichent seulement un message d'erreur sans arrêter le programme.

Le test EC4 réussit mais un message s'affiche à l'écran : "taille chaine doit etre inferieur/egal a 10". C'est une erreur non bloquante alors qu'elle devrait être bloquante, c'est-à-dire lever une exception. Ici, les calculs continuent et la construction de la chaîne est correcte ce qui conduit à un résultat positif de la fonction isValid. C'est une limite de ce type de tests.

Les autres tests ont réussi comme prévu. Cependant, la méthode EC ne considère pas beaucoup de cas de tests. Par exemple, il n'y a pas assez de test sur l'opérateur div qui est l'opérateur pouvant causer le plus de problèmes. C'est pour cela, que la méthode AC peut s'avérer ici très utile.

All Choices (AC)

Test	Path(valeur)	Opération (valeur)	Val1 (valeur)	Val2 (valeur)	Taille (valeur)	etatVide (valeur)	Resultat Attendu	Résultat du test
AC1	p1("config.properties")	o1("add")	v1-1(3)	v2-1(5)	t1(0)	e1(True)	Error	Failed
AC2	p1("config.properties")	o1("add")	v1-1(3)	v2-1(5)	t2(2)	e1(True)	True	Ok

Test	Path(valeur)	Opération (valeur)	Val1 (valeur)	Val2 (valeur)	Taille (valeur)	etatVide (valeur)	Resultat Attendu	Résultat du test
AC3	p1("config.properties")	o2 ("soust")	v1-1(3)	v2-1(5)	t3(3)	e1(True)	True	Ok
AC4	p2("test.properties")	o3 ("mult")	v1-1(3)	v2-1(5)	t4(10)	e1(True)	True	Failed
AC5	p1("config.properties")	o4 ("div")	v1-1(3)	v2-1(5)	t2(2)	e1(True)	True	Ok
AC6	p1("config.properties")	o4 ("div")	v1-1(3)	v2-1(5)	t3(3)	e2(False)	True	Ok
AC7	p1("config.properties")	o4 ("div")	v1-1(3)	v2-1(5)	t4(7)	e1(True)	True	Failed
AC8	p1("config.properties")	o4 ("div")	v1-1(3)	v2-2(0)	t2(2)	e1(True)	True	Ok
AC9	p1("config.properties")	o4 ("div")	v1-2(0)	v2-1(5)	t2(2)	e1(True)	True	Ok
AC10	p1("config.properties")	o4 ("div")	v1-2(0)	v2-1(5)	t3(3)	e1(True)	True	Ok
AC11	p1("config.properties")	o4 ("div")	v1-2(0)	v2-2(0)	t2(2)	e1(True)	True	Ok
AC12	p1("config.properties")	o1 ("add")	v1-1(3)	v2-1(5)	t5(15)	e1(True)	Error	Failed
AC13	p1("config.properties")	o5 ("abcd")	v1-1(3)	v2-1(5)	t2(2)	e1(True)	Error	Failed

Interprétation

On détecte ici les mêmes erreurs que pour EC : AC1 pour l'absence d'exception en cas de liste trop petite, AC4 pour le FileNotFoundException, AC12 pour l'absence d'exception en cas de taille de liste trop grande et AC13 pour l'absence d'exception en cas d'opérateur inconnu.

Cependant, un autre cas d'erreur est détecté, avec AC7. En effet, si après une division le résultat est de 0, au calcul suivant il y aura une division par 0 qui va causer une erreur. Il faudrait donc arrêter le calcul si un résultat de 0 après une division est détecté afin d'éviter la levée de cette exception. Dans ce cas, on aurait bien un isValid valant True.