TP8

Exercice – récursivité

1. Écrivez un programme récursif qui calcule la suite entière suivante (où la division est une division entière) :

$$\forall n > 1, \begin{cases} si\ n\ est\ pair, & u_n = u_{n-1}/u_{n-2} - u_{n-2}/u_{n-1} \\ si\ n\ est\ impair, & u_n = u_{n-2}/u_{n-1} - u_{n-1}/u_{n-2} \end{cases}$$

   Les valeurs n, u_0 et u_1 sont données en dur dans le code du programme.
2. Que se passe-t-il pour les valeurs suivantes :
   (a) $n = 10$, $u_0 = 100$, $u_1 = 2$?
   (b) $n = 10$, $u_0 = 100$, $u_1 = 1$?
3. Modifiez le programme pour intercepter correctement l'erreur et afficher le message suivant sur la sortie d'erreur : `[ERREUR]java.lang.RuntimeException: division par 0`.
4. Pour savoir à quel moment l'erreur s'est produite dans l'évaluation de la suite, il faut remonter la pile des appels de fonctions lorsqu'il y a une erreur. Pour cela, nous proposons de créer une classe d'exception dans laquelle seront ajoutées les informations nécessaires pendant la remontée dans la pile d'appel des fonctions.
   (a) Créez une classe `SuiteException` qui hérite de la classe `Exception`.
   (b) Ajoutez un attribut `trace` de type `String` qui stocke la trace des appels de fonctions.
   (c) Ajoutez une méthode `ajouterTrace(String s)` qui ajoute la chaîne de caractères `s` à l'attribut `trace`.
   (d) Redéfinissez la méthode `toString()` afin que la chaîne de caractères retournée corresponde à l'exception sous forme de chaîne suivie de la trace des appels de fonctions.
5. Modifiez le programme pour intercepter correctement l'erreur et afficher le message suivant sur la sortie d'erreur :

```
[ERREUR]SuiteException: division par 0
dans suite(5, 100, 1)
qui est appelée dans suite(6, 100, 1)
qui est appelée dans suite(7, 100, 1)
qui est appelée dans suite(8, 100, 1)
qui est appelée dans suite(9, 100, 1)
qui est appelée dans suite(10, 100, 1)
```

6. Nous souhaitons maintenant donner les valeurs de n, u_0 et u_1 en paramètres du programme, et ainsi les récupérer dans le paramètre de la méthode `main`.
   (a) Cliquez sur l'icône raccourcie *Run As...*, puis sélectionnez *Run Configurations...*. Dans la fenêtre qui s'ouvre, sélectionnez l'onglet *Argments* et ajoutez les différentes valeurs.
   (b) Modifiez le programme pour s'assurer qu'il y a exactement trois paramètres entrés par l'utilisateur, et capturer l'exception levée dans le cas où les valeurs des paramètres ne sont pas des nombres (utiliser la méthode statique `parseInt(String)` de la classe `Integer` pour convertir une chaîne de caractères en une valeur entière).

Exercice – pile de formes géométriques (suite)

Dans un TP précédent, nous avons écrit deux implémentations d'une pile de formes géométriques, utilisant des tableaux pour stocker leurs éléments.

1. Modifiez ces implémentations de sorte qu'une exception soit levée :
   (a) En cas d'appel des méthodes `depiler()` et `sommet()` sur une pile vide.
   (b) En cas d'appel de la méthode `empiler(Forme f)` sur une pile pleine.
2. Écrivez une méthode `clone()` qui réalise une copie profonde de la pile.
3. Créez une classe abstraite `PileFormesAbstraite` implémentant l'interface `PileFormes` et indépendante de la manière dont les éléments de la pile sont stockés (*e.g.*, tableau, liste...). Les deux implémentations de pile hériteront de cette classe.
   (a) Écrivez une méthode `equals(Object o)` qui retourne `true` si l'objet receveur et l'objet `o` passé en paramètre possèdent les mêmes éléments empilés dans le même ordre, sinon retourne `false`.
   (b) Écrivez une méthode `toString()` qui retourne une chaîne de caractères de la forme `[`$f_1, f_2, ...$`]` où f_1 correspond à la forme au sommet de la pile.
4. Complétez le programme de test.