Licence MASS – INF F3 – Examen de TP

17 janvier 2013 – Durée 2h

1 Un, deux, trois...

Le but de ce sujet de TP est de réaliser une calculette simple en se servant des capacités de programmation orientée-objet pour séparer l'aspect exécution de l'aspect interface.

2 Déroulement de la séance

Commencez par créer un nouveau projet avec votre nom. Toutes les classes que vous créerez au cours de cette scéance le seront dans le package calcul (que vous devrez créer) de ce projet. Une fois votre travail terminé, exportez le projet (menu fichier) sous forme d'archive et suivez les consignes du professeur pour rendre votre travail.

3 Travail demandé

3.1 Côté calculs

Écrivez une classe `Calculatrice` qui va permettre de réaliser des calculs. Cette classe doit pouvoir stocker les informations suivantes :

1. Un état, permettant de déterminer si la calculatrice marche ou est en erreur. Si elle marche il n'y a pas d'erreur (et inversement).
2. Un résultat (initialisé à 0) qui représente la mémoire interne de la calculatrice.
3. Un nombre (initialisé à 0) qui représente l'état courant de la calculatrice. Ce nombre pourra être modifié (voir méthode `chiffre`) par ajout successif de chiffres à sa droite.
4. Une opération (initialisée à `"rien"`) qui sera à appliquer quand l'utilisateur voudra le résultat du calcul.

Pour simplifier les choses, la calculatrice ne devra gérer que des nombres entiers.

3.1.1 Méthodes

La classe `Calculatrice` possède les méthodes suivantes :

1. `init` qui permet de remettre tous les attributs à leur état d'origine.
2. `chiffre` qui permet de construire un nombre en plusieurs étapes. À chaque appel de cette méthode, le chiffre passé en paramètre doit être rajouté à droite du nombre en cours de construction. Il représente donc ses unités à la fin de l'opération. Si le chiffre n'est pas valide, la calculatrice passe en état d'erreur.
3. `oppose` qui remplace le nombre en train d'être construit par son opposé.
4. `addition`, `soustraction`, `multiplication` et `division` (vous pouvez en ajouter d'autres s'il vous reste du temps) qui commencent par appliquer calcule (voir juste après) et stockent la nouvelle opération à effectuer.
5. `calcule` qui applique, au résultat et au nombre en cours de construction, l'opération stockée. Cette méthode stocke ensuite le résultat de ce calcul et réinitialise l'opération à effectuer.
   Attention, une division par 0 provoque une erreur.
6. `toString` qui permet de connaître l'état courant de la calculatrice sous forme de chaîne de caractères. Si la calculatrice est en erreur, la chaîne `"ERREUR"` doit être renvoyée.
   Pour savoir si vous devez représenter le résultat ou le nombre en train d'être construit, vous pouvez vous aider d'un attribut dont la valeur sera changée par les méthodes `chiffre` et `calcule`.

3.2 Côté interface

Créez une classe visuelle pour représenter votre calculatrice. Cette interface devra contenir une zone de texte pour afficher le résultat des calculs et une grille de boutons pour intéragir avec la classe que vous venez d'écrire.

1. Cette classe visuelle devra posséder un attribut de type calculatrice qui sera utilisé pour appeler les méthodes correspondantes à chaque appui de bouton.
2. À chaque clic sur un bouton, l'affichage doit être mis à jour en utilisant la méthode `setText` de la zone de texte.
3. Un exemple d'interface acceptable est donné ci-contre (la case à cocher fait partie de la question suivante).

4 Deuxième partie

En plus de pouvoir faire des calculs sur des entiers en base 10, la calculette va devoir supporter l'affichage et l'arithmétique binaire.

4.1 Préparation du terrain

1. Ajoutez un constructeur à la classe `Calculatrice` qui initialise la calculatrice avec un résultat déjà en mémoire.
2. Ajoutez une case à cocher dans la classe visuelle. Chaque clic sur cette case à cocher doit créer une nouvelle calculatrice ayant en mémoire le résultat affiché à l'écran. Cette nouvelle calculatrice remplace l'ancienne.

   Note : la case à cocher réagit aux clics de la même façon qu'un bouton.
3. Ajoutez une méthode `operationNonSupportee` à la classe `Calculatrice` qui fait passer la calculatrice en état d'erreur.

4.2 Héritage

La calculatrice binaire sera créée en tant que sous classe de la calculatrice de base. Pour celà :

1. Écrivez une classe qui dérive de `Calculatrice`. Exploitez autant que possible l'héritage en réutilisant les méthodes déjà écrites plutôt qu'en les recopiant.
2. Modifiez le comportement de `chiffre` pour n'accepter que des 0 ou des 1. Les autres chiffres font passer la calculatrice en mode d'erreur.

   Attention, les nombres binaires s'expriment en base 2.
3. Pour simplifier la représentation pour l'affichage, la méthode `oppose` fait passer la calculatrice dans un état d'erreur.
4. La méthode `calcule` garde son comportement d'origine mais (toujours pour simplifier) fait passer la calculatrice en état d'erreur si le résultat calculé est négatif.

   Un comportement supplémentaire doit aussi lui être associé qui va permettre d'appliquer une opération binaire avant le comportement mathématique normal.
5. Ajoutez les opération binaires `ou`, `et`, `non` et `ouExclusif` sur le même modèle que les opérations mathématiques.
6. Modifiez la méthode `toString` pour décomposer le nombre à afficher en suite de 0 et de 1. Gardez le même comportement en cas d'erreur.

4.3 Adaptation de l'interface

Commencez par modifier le comportement du clic sur la case à cocher pour, au lieu de remplacer une calculatrice classique par une autre, remplacer une instance de `Calculatrice` par une instance de la calculatrice binaire et inversement.

Ajoutez ensuite les boutons pour les opérations binaires qui vont correspondre à une opération non supportée pour une calculatrice classique et à l'opération adéquate pour une calculatrice binaire.