Mini-projet Java (séance du 01/04/2022)

EMERY
Antoine
i1

Exercice 1

1.1) @Override est présent car nous voulons redéfinir une méthode déjà définie par défaut. Ce n'est pas obligatoire de l'avoir, mais c'est une bonne idée, car cela fait comprendre au compilateur que nous voulons bien redéfinir une méthode existante.

1.2) Les attributs ne doivent pas être déclarés publics car ils pourraient être modifier par d'autres classes sans contrôle préalable. Ils doivent être mis en private et accessible via des getters/setters qui permettent de réguler leur modification.

1.3) Oui c'est possible, cependant si nous ne redéfinissons pas la méthode equals dans la classe comme c'est le cas ici, equals se contentera de regarder si la référence vers les objets sont les même, c'est-à-dire c1 == c2. Donc même si la valeur des attributs sont correctes, il nous renverra faux si ce sont deux objets instanciés séparément. Pour obtenir le bon résultat, il faut redéfinir la méthode equals en comparant chaque attribut et en renvoyant le booléen correspondant.

Exercice 2

2.1) voir Git (CLIClassiqueTest.java)

2.2) Le résultat devrait être « alpha=0.85, epsilon=0.001, indice=100, mode=CREUSE »

2.3) On peut déduire que CLIClassique.configuration de gère pas la conversion des nombres flottants en integer

2.4) Oui on peut le déduire car on parseInt une string, donc le compilateur n'émet pas de réserve, c'est bien une string qui est attendue en paramètre de parseInt.

2.5) voir Git (CLIClassique.java)

2.6)

Exercice 3

Voir Git
(UML.png, CLI.java, ...)

Exercice 4

4.1) Pour produire la vue, il faut créer une classe qui étende JFrame. Dans cette classe on définit des layout (pour arranger la disposition des éléments que nous allons placer), puis ajoute des composants par briques (on crée un panneau, on lui associe un layout, puis on ajoute les différents élément tels que les boutons et les champs de texte avant d'ajouter ce panneau a une partie du Jframe).

Ici, nous allons utiliser un flowlayout pour la partie haute du Jframe (layout.NORTH) et y placer nos deux boutons creuse et pleine, un grid layout de 3x3 au centre (layout.CENTER) pour les différents label de textes, les textfields et les boutons, et enfin un flowlayout pour la partie du bas (layout.SOUTH) afin d'y placer notre champ de résultat.

4.2) Pour se faire, il faut utiliser des actionlistener, qui permettent de détecter quand un clic est effectué sur le bouton. J'ai choisi d'instancier l'actionlistener au moment de l'ajouter en redéfinissant directement la méthode actionperformed, qui redirige vers une méthode définie en fin de classe, juste avant le main. Ainsi, chaque bouton aura sa fonction associée, ce qui est plus pratique que de traiter chaque évènement avec le getsource pour vérifier d'ou provient l'action, mais qui est aussi moins lourd que de définir plusieurs classes qui étende actionlistener et redéfinir la methode actionperformed.

4.3) Voir Git (PageRankFenetre.java)

Exercice 5

5.1) Voir Git (CLIOutils.java)
5.2) Avec deux classes Action différentes, une qui gérerait des booléens et une qui gererait les options avec valeurs, on pourrait attribuer lors de la création de la CLI les actions correspondantes à chaque option (qui auraient pour effet de changer un compteur dans l'option associée ou alors de changer un booléen).

Exercice 6

6.1) Voir Git
6.2) Voir Git (XMLCLIGenerator.java, test avec fonction main)
6.3) On pourrait utiliser par exemple l'API SAX pour parser le document, en extraire les différentes options de la CLI et en instancier une. On devrait pour cela redéfinir les méthodes startdocument, enddocument, startelement, endelement et characters, et plutot que de les afficher dans la sortie standard, nous pourrions stocker les différentes informations dans des variables pour ensuite nous en servir pour la construction d'une CLI.