Développement informatique avancé orienté application - TP

Interfaces graphiques et collections

Virginie Van den Schrieck Arnaud Dewulf

Dans le cadre de ce TP, nous allons aborder deux notions : La conception d'interfaces graphiques d'une part, et les collections d'autre part.

Nous allons d'abord illustrer chacun de ces deux concepts dans des exercices simples, puis nous terminerons par un exercice de synthèse.

La théorie relative à ces notions se trouve sur Inginious.

1 Conception d'interfaces graphiques

1.1 Analyse d'une vue MVC

Reprenez l'interface graphique de l'application du TP6 (classe `BibliothequeVueGUI`), et analysez-là :

1. Faites un schéma représentant l'interface graphique, en mettant en avant la hiérarchie des containers (Frame, Panel, ...)
2. Pour chaque conteneur, identifiez la manière dont les composants sont positionnés les uns par rapport aux autres (Layout)
3. Identifiez les éléments associés à une interaction utilisateur/programme. Comment sont-ils modifiés ? Sont-ils associés à des événéments, et dans ce cas, quel est la méthode appelée pour y réagir ? (handler/-callback)

1.2 Conception d'une interface graphique simple

Concevez une interface graphique composée de :
— deux champs texte
— un bouton appelé `SWAP`
— un bouton appelé `FERMER`
Associez ensuite le bouton `SWAP` à un handler qui inverse le contenu des deux champs texte lorsque le bouton est déclenché.

Faites également en sorte que l'application se termine lorsqu'on appuie sur le bouton `FERMER`.

1.2.1 WindowBuilder

Installez à présent le plugin `WindowBuilder` dans Eclipse. Ce plugin vous offre une interface type *WYSIWYG* pour la conception d'interface graphique.

Recommencez à présent l'exercice précédent avec l'échange du contenu des deux boutons, cette fois en utilisant le plugin `WindowBuilder`.

Attention : soyez attentifs lorsque vous utilisez ce genre de logiciel générant automatiquement du code : Vous devez bien comprendre chaque instruction générée, et ne pas hésiter à refactorer le code générer pour obtenir du code propre et de qualité.

2 Les collections

2.1 Exploration du framework Collection

Sur base de l'API Java, identifiez trois structures de données différentes autre qu'`ArrayList`. Pour chaque structure de données, identifiez :

- Sa position dans la hiérarchie de classe Java : De quelle classe dérive-elle ? Quelle(s) interfaces implémente-t-elle ?
- Ses spécificités au niveau performances (ex : accès par indexation, rapiditié d'insertion/suppression, ...).
- Un cas d'utilisation typique

2.2 Utilisation d'une ArrayList

Récupérer une classe `Etudiant` et une classe `Professeur` fonctionnelles d'un TP précédent. Implémentez une nouvelle classe `ClassGroup`, qui possède un professeur titulaire et une collection d'étudiants. Utilisez pour cela une `ArrayList`. Implémentez les méthodes `addStudent()`, `removeStudent()`, `getNumStudents()` et `toString()`.

2.3 Utilisation d'une HashTable

Créez à présent une classe `School`, qui contient une série de classes identifiées par leurs noms (ex : 1TM1, 3TL2, ...). Implémentez les méthodes `getClassGroup(String name)`, `addClassGroup(String name, ClassGroup group)` et getClassesNames().

3 Exercice de synthèse

Comme exercice de synthèse, nous allons implémenter une petite application permettant de calculer les codes couleur des résistances.

3.1 Création du code couleur

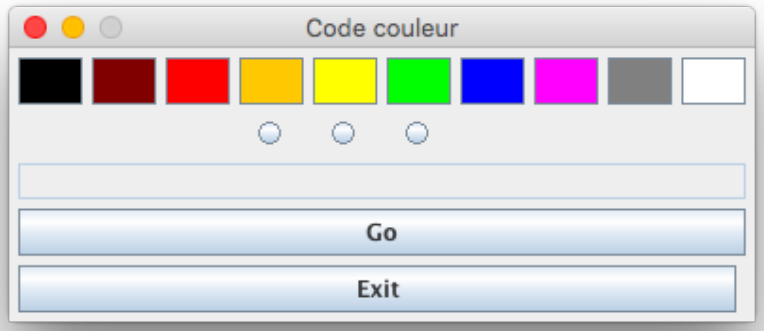
Créez une classe `CodeCouleur`, qui permet d'associer des entiers à des couleurs.

Cette classe devra posséder une méthode `addCouleur(Color c, int value)` qui permet d'ajouter une nouvelle couleur au code, un accesseur `getValue(Color c)`, qui renvoie la valeur associée à une couleur, et un autre accesseur `getColors()`, qui renvoie la liste des couleurs utilisées dans le code couleur.

Elle devra également avoir une méthode `calcule(Color dizaine, Color unite, Color puissance)`, qui, sur base de trois couleurs (le code d'une résistance), renvoie la valeur correspondante.

3.2 Création de la GUI

Concevez une GUI similaire à l'image ci-dessous :



Cette GUI possède une liste de couleurs, chaque couleur étant cliquable, et trois boutons radio représentant chacun une ligne du code couleur d'une résistance. Lorsqu'on sélectionne un des trois boutons radio puis qu'on clique sur une couleur, on attribue une couleur au fond du bouton correspondant. Lorsque les trois boutons ont une couleur, on peut alors calculer la valeur de la résistance en cliquant sur le bouton ad-hoc.

Bonus : Lorsqu'on passe la souris sur une couleur, faites en sorte qu'elle affiche le nombre associé à cette couleur.

Lors de la création de la GUI, celle-ci reçoit une instance de la classe `CodeCouleur`, instance qui aura été configurée avec un code couleur valide et sur laquelle la GUI se basera pour ses opérations.

Remarque : Nous n'utilisons pas ici l'architecture MVC. L'application est en effet purement statique au niveau des données (i.e. le code-couleur est défini une fois pour toute au lancement de l'app) et ne justifie donc pas la mise en oeuvre d'un modèle modifiable et d'un contrôleur