2IA - Approfondissement : conception orientée objet

Programmation orienté objet JAVA - Calculatrice à notation polonaise inverse

Etudiants :

Omar El Badaoui

Johanna Saoud

Enseignants :

Christelle Urtado

Sylvain Vauttier

14 novembre 2021

1 Réalisation

Le travail a été effectué avec la version 15.0.1 de JAVA et 17 de JAVAFX. Ce projet repose sur l'implémentation d'une calculatrice à notation polonaise inverse suivant le diagramme présenté en figure 1 et 2.

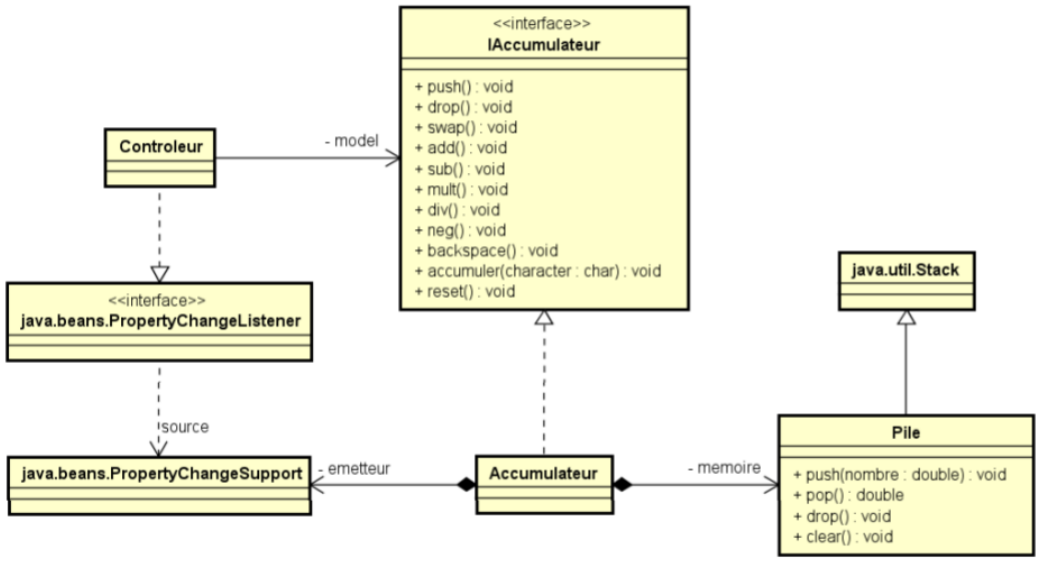

FIGURE 1 – Vue conceptuelle des principaux éléments du modèle pour la calculatrice (partie 1)

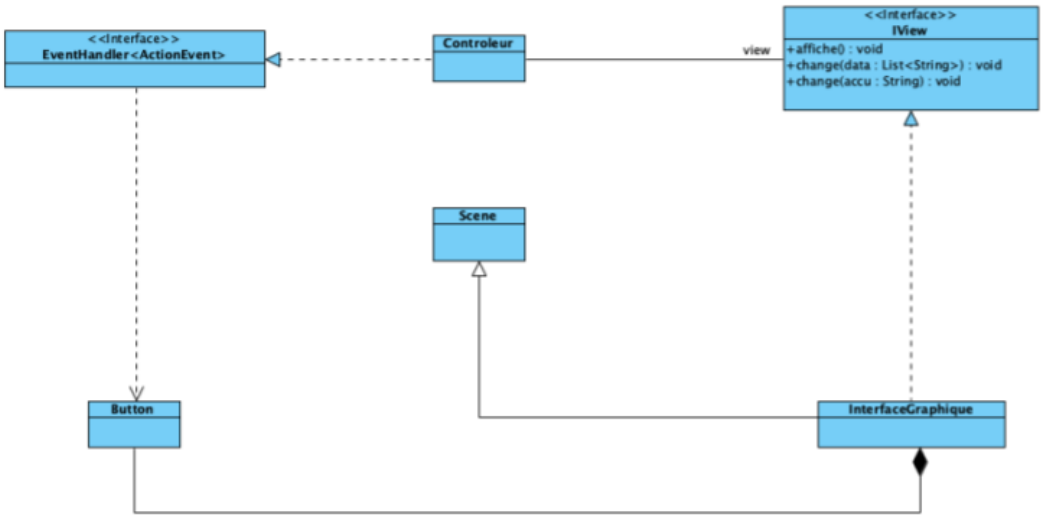

FIGURE 2 – Vue conceptuelle des principaux éléments du modèle pour la calculatrice (partie 2)

Les différentes parties du diagramme vont fonctionner de la façon suivante : en cliquant sur les boutons de la calculatrice, on passe un *Action Event* au contrôleur, on transit ainsi de l'interface graphique au contrôleur. On retrouve ensuite des *handlers* qui vont venir s'occuper de l'action à effectuer via une fonction associé dans le controleur. A chaque *event* on va avoir une fonction associée dans la déclaration qui va être cherché dans le contrôleur et ce dernier va ordonner à l'accumulateur d'activer les fonctions associées à l'événement. Dans la direction inverse, dans le cas où on retrouve une modification dans l'accumulateur, le contrôleur va détecter cette dernière et va ordonner à l'interface de se mettre à jour. La détection repose sur la mise en place de *listeners*.

Cependant, le controleur a besoin de connaitre l'interface sur la quelle il devra demander l'affichage des modifications, de fait, l'interface devrait etre un attribut du controleur. Mais l'interface graphique a aussi besoin d'un controleur où retrouver les *events handlers*... Pour créer l'un on aura alors besoin d'avoir l'autre.

Afin de remerdier à ce problème, nous avons, au niveau du main, créé un controleur sans valeur pour son attribut "interface graphique". On instancie ensuite une interface graphique, avec le controleur, que l'on met dans l'attribut "interface graphique" du controleur à l'aide d'un *Setter*.

2 Développement

Au cours de ce projet, nous avons été amenées à développer des méthodes et astuces afin de pallier à certaines problématiques rencontrées.

2.1 Les types $IntergerProperty$,$StringProperty$,$BooleanProperty$

Au niveau du contrôleur, on retrouve sur des éléments étant de *types* declarés dans la bibliothèque *(javafx.beans.property)* qui sont les attributs de l'accumulateur.ils integrent deja des *PropertyChangeSupport* L'avantage est que leurs changements sont détectés dynamiquement et qu'il n'est donc pas nécessaire de "*fire*" vers les *listeners*.

Cependant, ces *types* ne recouvrent pas toutes les classes dont on aurait besoin. En effet, on ne retrouve pas de type *pile* où l'on peut mettre des *listeners*. La solution que nous avons apportée a été de créer une variable *IntergerProperty* (nommée n) qui représente le nombre d'éléments présents sur la pile et sur lequel il est possible de mettre un *listener* dynamique. Ainsi, lorsque l'on *push*, *pop* ou qu'une opération vient modifier le nombre d'éléments présent dans la *pile*, on vient activer indirectement le *listener*.

2.2 La méthode $triggerNEvent()$

Néanmoins, il est possible que certaines opérations venant modifier la *pile* ne soient pas détectées par les *listeners* car le nombre d'éléments de la pile reste inchangé : c'est le cas pour le *swap* qui vient modifier la *pile*, mais pas le nombre d'éléments de la *pile*. Pour pallier à cela, nous avons choisi de développer une méthode *triggerNEvent* qui va changer la valeur de n en 0 puis lui redonner sa valeur. Ainsi, il est possible de détecter

et de signaler le changement car n aura été modifié : l'interface graphique pourra ainsi se mettre à jour.

2.3 La méthode $neg()$

L'accumulateur possède un attribut "neg" booléen, si sa valeur est à "$True$" la valeur accumulée sera intégrée à la *pile* négativement. La methode $neg()$ est en outre un setter qui changera la valeur de "neg" entre "$True$" et "$False$".

2.4 La méthode $handleCharacterButton()$

Déclarée au niveau du controleur, cette méthode prend un *ActionEvent* appelé préalablement par les boutons allant de 0 à 9 ainsi que le "." de notre interface graphique. On va ainsi pouvoir accumuler le nouveau caractère à la valeur déjà saisie dans l'accumulateur. On retrouve d'autres fonctions *handle* pour les fonctions traitant les opérations et méthodes des piles tel que *handleAddButton, handleSwapButton ....*

2.5 La méthode $setIg()$

Nous avons crée le controleur sans interface graphique, nous avons donc dues créer une méthode *set* afin de lui donner une interface graphique après qu'elle ait été instanciée dans le *main*.

2.6 L'interface graphique

Nous avons utilisé un fichier *FXML* et *SceneBuilder* afin de créer notre design. Ayant déjà de l'experience avec *JavaSwing* et étant déjà familiarisés avec les principes de *noeuds* et de *containers*, l'utilisation de *SceneBuilder* representait une méthode connue se présentant comme plus simple et intuitive, nous evitant ainsi les déclarations manuelles. Cela représentait pour nous un gain de temps car nous avons passé la major partie de notre temps à comprendre ce que l'on attendait de nous, le flux d'information souhaité (comment fonctionne l'application ? Qui communique avec quoi ? Comment ?...) .

2.7 Respect des consignes

Nous avons réussi à implémenter toutes les fonctionnalités demandées, mais le programme reste exécutable via le *main* à l'aide d'un éditeur (comme *eclipse* par exemple) et non exécutable en un clique. En effet, les *Jar* que l'on utilise se trouvent dans un dossier avec le projet et sont referencés dans le *Modulepath*, *eclipse* effectue le travail de compilation automatiquement lorsque l'on demande d'executer le main.

De plus, au niveau de l'accumulateur, au lieu d'implémenter les *ChangeListeners* et *ChangeSupport* de la bibliothèque *java.beans*, on utilise des classes définies dans la bibliothèque *javafx.beans*, en veillant à respecter les normes des beans en *Java* (tous les attributs sont *private*, et on utilise des *setters* et *getters*...)