TP 3 Conception d'une interface graphique avec Swing

Environnement de travail

Pour ce TP un fichier build.xml vous est fourni. C'est l'équivalent d'un makefile en C. Il permet de définir les commandes de compilation et d'exécution. Dans un premier temps vous devez éditer ce fichier pour modifier la variable COLLIE qui prend l'emplacement, dans votre dossier personnel, de la racine de l'application Collie. Dans une console et à partir de la racine de Collie, vous pouvez exécuter les commandes suivantes :

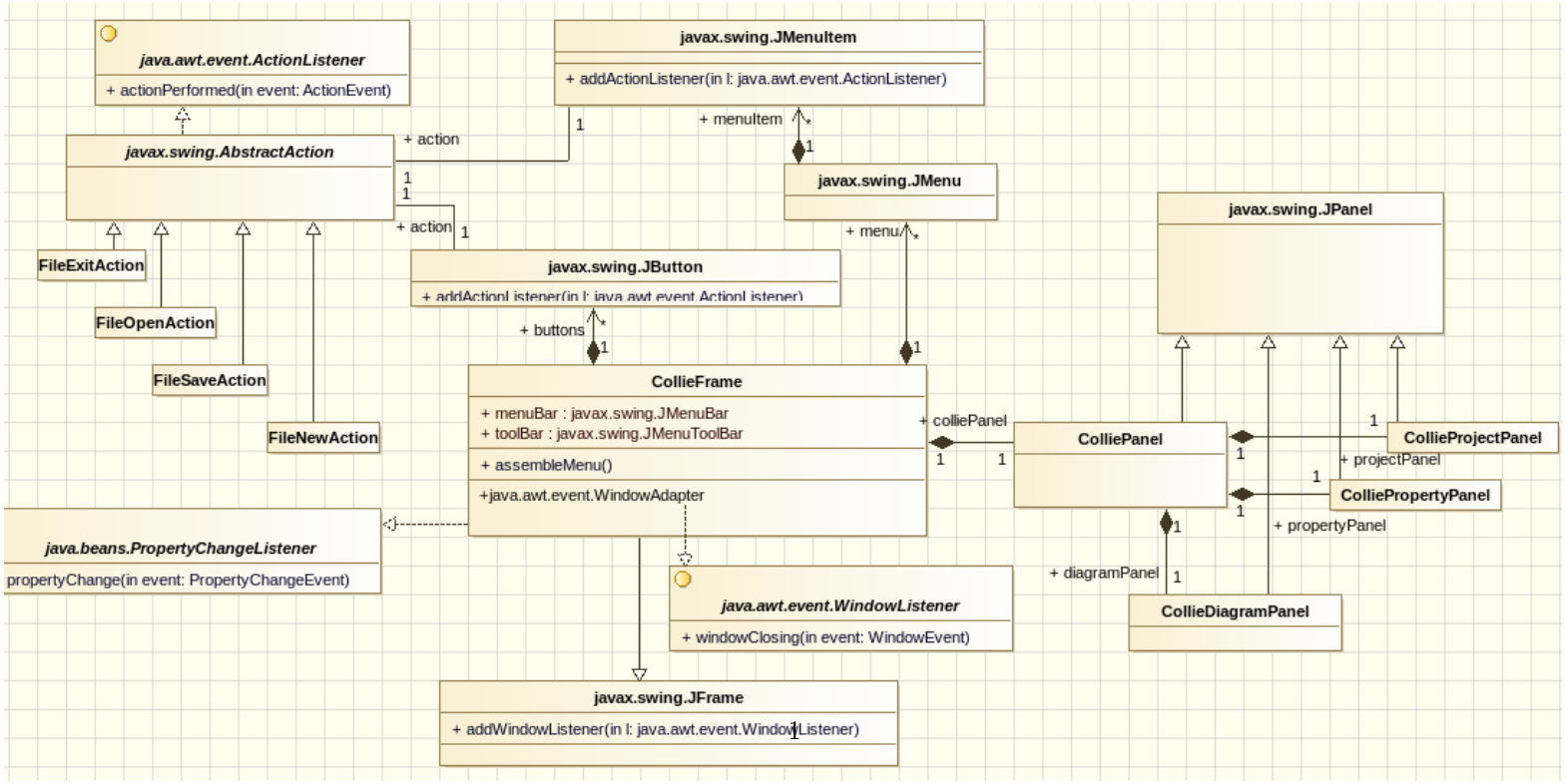
- ant build pour compiler les fichiers sources qui se trouvent dans le répertoire /src (un répertoire /classes est ainsi créé lors de la compilation)
- ant run pour démarrer l'application
- ant clean pour supprimer le répertoire /classes et tout ce qu'il s'y trouve.

Remarque Le programme dans son état actuel ne compile pas. Il faut d'abord répondre à la 1ere question.

Sujet

Vous allez programmer les éléments manquants de Collie, une application avec interface graphique pour l'édition de diagrammes de collaboration UML 1.4. Cette application est découpée en 4 packages. Le package `colliec` contient un `main()` qui crée une fenêtre de base `JFrame`. Les élements qui constituent cette JFrame (panels, menus...) sont implémentés dans le package `window`. Le diagramme de classes sur la figure 1 montre comment opère la fenêtre de base `CollieFrame`. Cette fenêtre est complètement occupée par un panel `ColliePanel`, qui est lui-même décomposé en trois sous-panels. Sur la gauche le panel `ColliePorjectPanel` permet la gestion du projet en cours et pourra contenir par exemple son arborescence. Ce panel partage l'espace avec `CollieDiagramPanel` qui contient le diagramme de collaboration. Ces deux panels partagent la hauteur de la fenêtre avec `ColliePropertyPanel`, un panel situé en bas qui permet d'éditer les propriétés des éléments du diagramme de collaboration.

La fenêtre implémente l'interface `PropertyChangeListener`. Lorsque une modification de l'utilisateur survient, un évènement de changement (`ChangeEvent`) est déclenché permettant de mettre les boutons save/save as en clair.



Toutes les classes du package `window` implémentent le patron de modélisation singleton afin de restreindre l'instanciation des classes a un seul objet par classe. Pour cela chaque classe possède un attribut statique qui va permettre de stocker l'unique instance de la classe qui est créée au démarrage de l'application. Leur constructeur est protégé et une méthode statique `get_nomdelaclasse` permet de retourner cette unique instance.

1 Question.1

La fenêtre gère une barre de menu avec les menus File et Help. Les items de menus correspondants sont associés avec des objets listeners spécifiques. L'enregistrement des objets listeners aux items de menu sont déjà réalisés dans la méthode `assembleMenu`*. Développez les classes de ces objets listeners. Dans un premier temps vous ne développerez pas les effets réellement attendus des évènements, vous vous contenterez d'afficher l'évènement correspondant (New, Open, Save, SaveAs, Exit, About) sur la console. Vous utiliserez une méthode de la classe* `java.awt.event.ActionEvent` *pour récupérer la chaine de caractères associée à l'action correspondante. Compilez et démarrez l'application.*

2 Question.2

Ajoutez un `WindowListener` *à la fenêtre afin de gérer correctement sa procédure de fermeture. Pour cela vous dériverez une classe* `WindowAdapter` *et vous redéfinirez uniquement la méthode* `windowClosing()`*. Compléter la classe correspondant à l'action sur l'item de menu exit que vous avez créée dans la question précédente pour fermer l'application par le menu.*

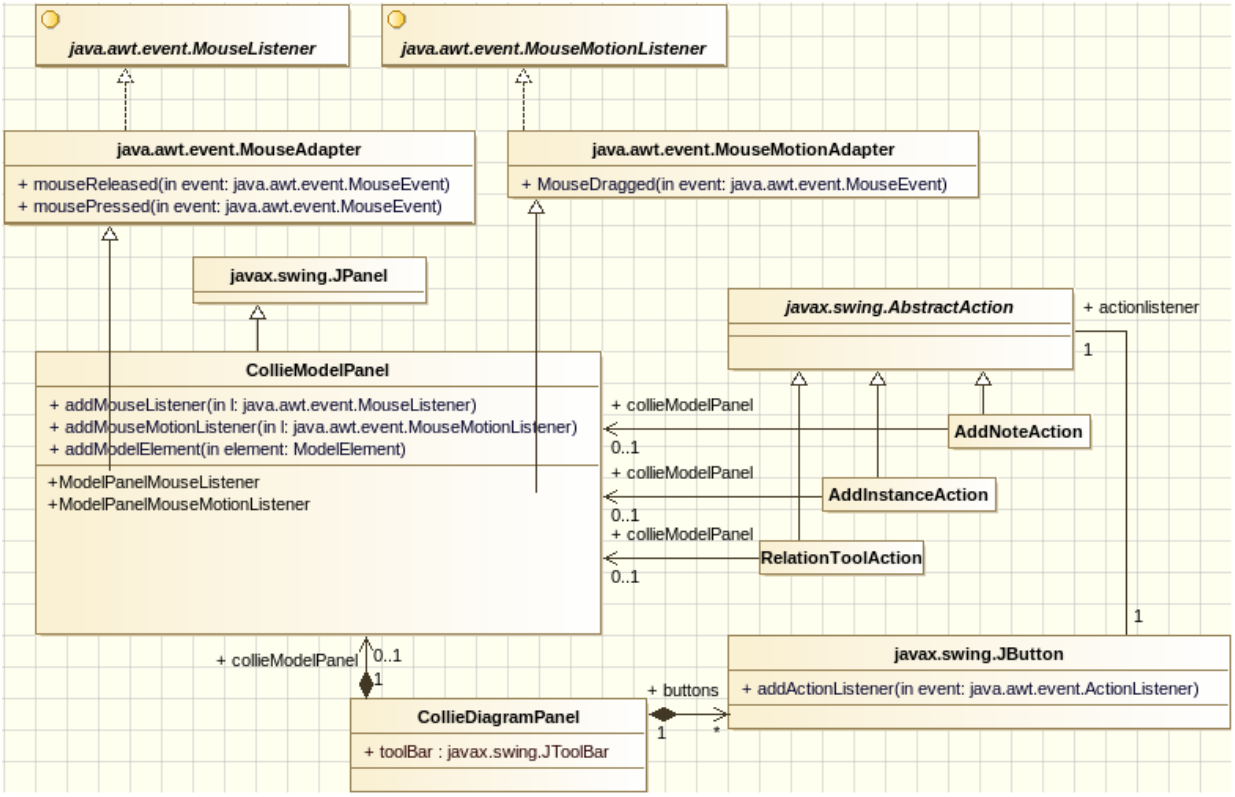


FIGURE 2 – `CollieDiagramPanel`

Le panel `CollieDiagramPanel` est constitué d'une barre d'outil permettant d'insérer des éléments du diagramme de collaboration et d'un sous-panel `CollieModelPanel` qui contient le modèle saisi par l'utilisateur. La figure 2 décrit comment opère `CollieDiagramPanel`. La barre d'outil est constituée de boutons auxquels sont associés des objets listeners pour créer et placer en haut à gauche du panel `CollieModelPanel` les éléments du diagramme de collaboration. `CollieModelPanel` enregistre deux types d'objet listener : `MouseListener` et `MouseMotionListener` qui permettent respectivement de gérer les évènements boutons de la souris et les évènements de mouvement de la souris sur ce panel. Pour cela `CollieModelPanel` utilise les adaptateurs `MouseAdapter` et `MouseMotionAdapter` afin de n'implémenter que les méthodes qui nous intéressent. Ici : `mouseReleased`, `MousePressed` et `MouseDragged`.

Le package `diagram` regroupe les classes liés au diagramme de collaboration. Le modèle du domaine est décrit par le diagramme de classe sur la figure 3. Ces éléments sont des objets graphiques et ils peuvent donc être dessinés grâce à la méthode `draw`. Ils utilisent des classes de la librairies `java.awt` (`Rectangle`, `Point`) pour leur définir une forme graphique. Un diagramme de collaboration est constitué d'un ensemble d'élément `modelElements`. La classe `ModelElement` est abstraite, elle définit un ensemble d'opérations qui doivent être redéfinies par les classes dérivées. Un élément du modèle peut soit etre un noeud `NodeElement` soit une relation `LinkageElement`. Un noeud peut être associé à plusieurs relations, et une relation à un noeud source et un noeud destination. Un noeud est soit une instance, une multi-instance ou un acteur. Une relation est soit une relation d'agrégation soit une relation d'association. Un lien est constitué de un ou plusieurs segments. Un noeud a une localisation qui est définie par le point à l'extrême haut-gauche.

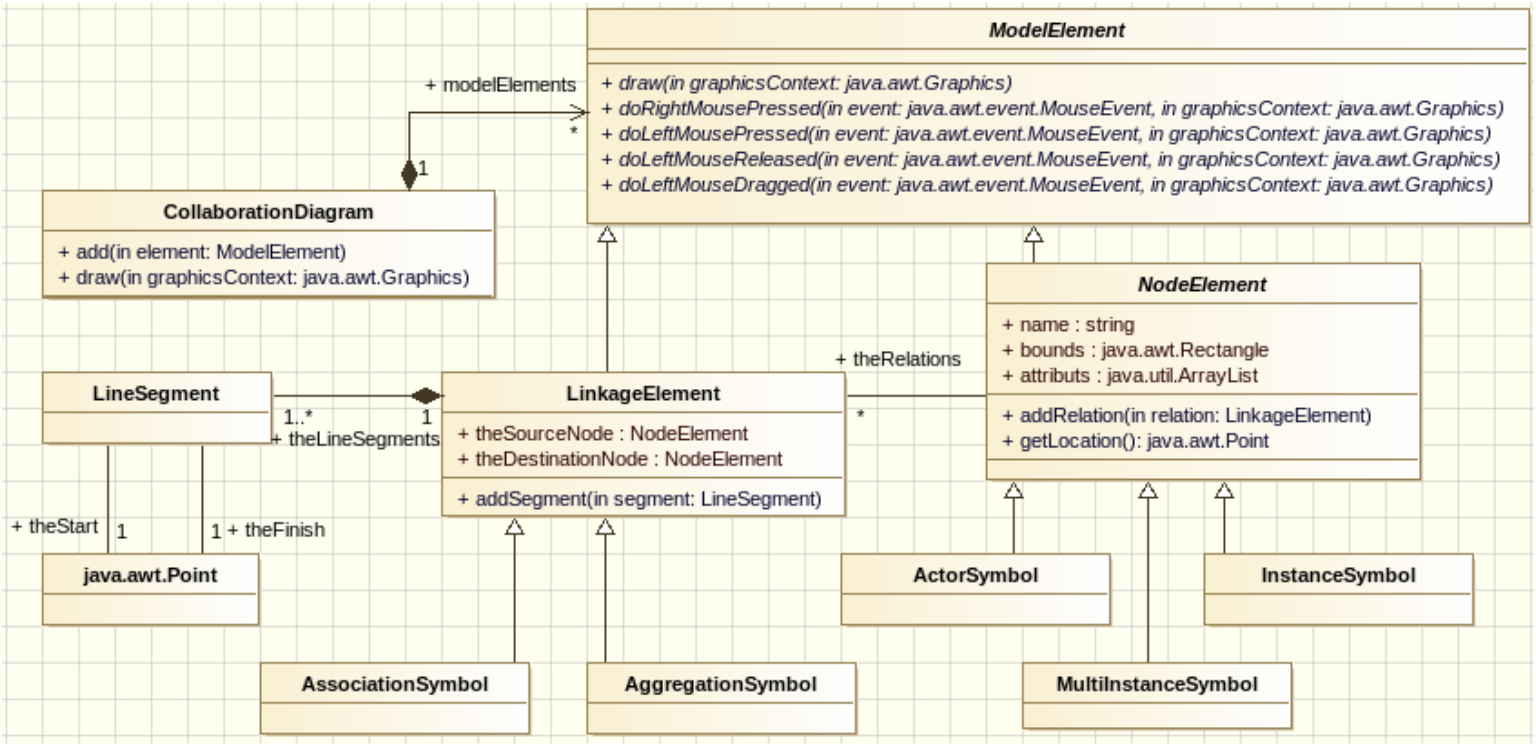


FIGURE 3 – Diagramme de classe UML du package `diagram`

Lorsqu'un évènement souris survient sur le panel `CollieModelPanel`, celui-ci détecte de quel évènement il s'agit (pression clic droit, pression clic gauche, relâchement clic droit, drag) et sur quel noeud cet évènement à eu lieu s'il en existe un. `CollieModelPanel` appelle alors une des méthodes du noeud en question et correspondant au type d'évènement (`doLeftMousePressed`, `doLeftMouseReleased`, `doLeftMouseDragged`, `doRightMousePressed`).

3 Question.3

Vous devez mettre en place la gestion des noeuds du diagramme de collaboration (la gestion des relations et de l'outil de sélection sont déjà mise en oeuvre dans le package `tool`*). Vous devez pour cela surcharger les méthodes* `doLeftMousePressed`*,* `doLeftMouseReleased`*,* `doLeftMouseDragged` *de la classe* `NodeElement` *pour permettre de sélectionner un élément noeud, de le déplacer et de le relâcher à une nouvel endroit sur le panel. Consultez l'API des classes* `java.awt.event.MouseEvent` *et* `java.awt.Point` *pour trouver les méthodes qui vous permettront cette réalisation.*

4 Question.4

Surchargez la méthode `doRightMousePressed` *afin d'ouvrir un menu de type* `javax.swing.JPopMenu` *qui permettra :*

- *d'éditer l'instance pour modifier son nom par exemple,*
- *d'éditer les attributs de l'instance,*
- *afficher les attributs (qui par défaut ne sont pas affichés showflag = false).*
- *supprimer l'instance.*

Développez les classes des objets listeners associés aux items de menus. Dans un premier temps vous ne développerez pas les effets réellement attendus des évènements, vous vous contenterez d'afficher l'évènement correspondant (Edit instance, Edit attributes, Show attributes, Delete instance) sur la console.

5 Question.5

Vous devez mettre en oeuvre la saisie des propriétés d'un noeud. Dans le panel `ColliePropertyPanel`*, vous devez faire apparaitre, à la sélection (clic gauche) d'un noeud, la possibilité de saisir la référence de l'objet et la classe dont il est issu et la liste des couples attribut/valeur de l'attribut. Vous devez aussi permettre d'ajouter des attributs et de saisir leur valeur. Vous êtes libre d'utiliser les composants graphiques de la librairie* `javax.swing` *de votre choix.*

6 Question.6

Mettez en oeuvre l'ensemble des fonctionnalités restantes :

- *New, Open, Save, SaveAs et About vues en question 1.*
- *Edit instance, Edit attributes, Show attributes, Delete instance, vues en question 4, via des fenêtres de dialogue. Vous ferez un nouveau package* `dialog` *pour toutes les classes concernant des fenêtres de dialogue.*

7 Documents à rendre

La livraison doit comprendre l'ensemble de votre programme ainsi qu'un document qui contiendra :

- Les diagrammes de classe qui modéliseront rigoureusement l'architecture de votre implémentation.
- Les diagrammes de séquence qui décrivent les interactions entre vos classes lors du lancement de l'application et lors des scénarios d'utilisation que vous aurez identifiés.
- L'explication et la justification de vos mise en oeuvre des questions 3, 4, 5 et 6.