Université Paul Sabatier – Toulouse III
IUT A - Toulouse Rangueil
Projet tuteuré #20

Antoine de Roquemaurel
Mathieu Soum
Geoffroy Subias
Marie-Ly Tang
Groupe B

Pour Monsieur Thierry Millan (Client)
Madame Caroline Kross (Tutrice)

Installation et exploitation

Bibliothèque d'objets graphiques UML

Toulouse, le 18 janvier 2012

Table des matières

1 Installation et téléchargement de la bibliothèque **4**
1.1 JGraphx 4
1.2 LibUML 4
1.3 Version projet Netbeans 4

2 Utilisation de libUML **5**
2.1 Le démonstrateur 5
2.1.1 La barre d'outils 5
2.1.2 Le graphe 6
2.1.3 Le panneau de droite 6
2.2 Exemple de création d'un diagramme simple 7
2.3 Documentation 8

3 Poursuite de développement de la bibliothèque **9**

4 Conclusion du projet **10**

A Diagramme de classes **11**

B Diagramme de paquetages **12**

C Glossaire **13**

LibUML est une **bibliothèque** d'objets graphiques représentant les différents éléments de modélisation de la norme **UML 2** [1]. Celle-ci à été développé dans le cadre des projets tuteurés à l'IUT [2] 'A' de Toulouse.

- Antoine de Roquemaurel
- Mathieu Soum
- Geoffroy Subias
- Marie-Ly Tag

Nous l'avons développée en **Java** et conçut comme une bibliothèque pouvant être utilisée dans des programmes Java comme composant. Vous pouvez vous en servir pour développer un outil complet de modélisation UML par exemple.

Cette bibliothèque permet de faire différentes choses dans le cadre de la conception UML, une fois la que vous aurez compris comment fonctionne la bibliothèque vous pourrez :

- Créer des éléments de modélisation
  - Acteur actif ou passif
  - Traitement
  - Cas d'utilisation
  - Classe
- Relier ses composants via différents types de flèches
  - Agrégation
  - Composition
  - Association binavigable ou mononavigable
  - Messages synchrones ou asynchrones
  - Généralisation
- Supprimer des éléments et flèches
- Modifier le contenu des éléments et flèches
- Redimensionner les éléments de modélisation

1. Unified Modelling Language
2. Institut Universitaire de Technologies

Chapitre 1

Installation et téléchargement de la bibliothèque

1.1 JGraphx

Notre bibliothèque UML dépend d'une bibliothèque externe, cette bibliothèque est JGraphx, vous devez donc posséder JGraphx 1.8 pour que notre bibliothèque puisse fonctionner, vous devez inclure celle-ci dans votre projet.
▷ `http://telechargements.joohoo.fr/projet_IUT/JGraphx-1.8.jar`

1.2 LibUML

Vous pouvez choisir de télécharger notre bibliothèque via un .jar ou si vous préférez les sources sont également disponibles.
De même, il est nécessaire d'inclure soit les sources de la bibliothèque, soit le .jar dans le projet pour pouvoir utiliser notre bibliothèque.
▷ `http://telechargements.joohoo.fr/projet_IUT/libUML.jar`
▷ `http://telechargements.joohoo.fr/projet_IUT/libUML-src.zip`

1.3 Version projet Netbeans

Également, une version existe sous forme de projet Netbeans dans le cas où vous utilisez cet EDI [1].
Ce projet contient les sources de la bibliothèque UML, la bibliothèque JGraphx, les Tests unitaires du projet et le code du démonstrateur.
▷ `http://telechargements.joohoo.fr/projet_IUT/libUML-projetNetbeans-1.8.zip`

1. Environnement de Développement Intégré

Chapitre 2

Utilisation de libUML

2.1 Le démonstrateur

Nous avons développé un démonstrateur afin que vous puissiez comprendre comment fonctionne la bibliothèque, et pour montrer ses possibilités. Ainsi, tout ce qu'il est possible de faire avec la bibliothèque sera présent dans le démonstrateur.
Dans cette partie nous allons passer brièvement ses fonctionnalités, cependant, pour une meilleure compréhension, nous avons commentés le code du démonstrateur pour qu'il soit le plus simple possible et pour que vous ayez le moins possible de vous référez à ce document.

Le démonstrateur est volontairement simple, il n'a pas pour but d'être lourd, mais uniquement de montrer les possibilités de la bibliothèque. Il est disposé en trois parties :

- En haut, la barre d'outils, permettant de sélectionner l'élément graphique souhaité
- Au centre, ce que nous appelons le graphe, c'est ici qu'apparaîtrons les diagrammes UML
- À droite, un panneau contenant éventuellement un tableau avec les informations de la classe sélectionnée.

2.1.1 La barre d'outils

La barre d'outils contient tous les éléments graphiques que permet notre bibliothèque.

Element de modélisation

Il est actuellement possible de faire les éléments de modélisation suivants :

- Acteur actif
- Acteur passif
- Traitement
- Cas d'utilisation
- Classe

Le clic sur un bouton positionnera un élément de modélisation dans le graphe.
Chacun des éléments de modélisation correspondent à une classe(Traitement, CasUtilisation,...), qui hérite de la classe abstraite `ElementModelisation` elle même héritant de la classe `ElementGraphique`.
(Pour plus de détails, cf Annexe ? ? page ? ?)

Liens

La deuxième partie de la barre d'outils contiens tous les liens qu'il est possible de faire.

- Généralisation
- Association
- Dépendance
- ...

Pour relier deux éléments, vous devez cliquer sur la flèche voulut, puis cliquer sur les deux éléments à relier. Chacun des liens héritent de la classe abstraite `ElementGraphique`. (Pour plus de détails, cf Annexe ? ? page ? ?).Le clic sur les deux éléments aura pour effet de créer un lien, son constructeur accueillant la source et la destination du lien.

2.1.2 Le graphe

Le graphe est la partie central du démonstrateur, c'est dans celui-ci que tous les éléments graphiques s'affichent.

mxGraph

Le graphe est un objet de la classe `mxGraph`, pour pouvoir afficher les éléments graphiques de la bibliothèque celui-ci est indispensable, si vous souhaitez afficher un élément graphique vous devrez donc instancier un `mxGraph`, que vous devez ensuite faire passer à la création de chaque élément graphique via son constructeur, en effet chaque élément graphique possèdent un graphe.

Le diagramme

Le diagramme est un objet contenant la liste de tous les éléments graphiques présents dans le graphe, celui-ci peut vous permettre d'autoriser ou non la présence de certains éléments dans le graphe via la méthode `estAutorise`. Dans une approche de long terme, ce diagramme devrait pouvoir permettre d'exporter notre travail pour pouvoir le réutiliser plus tard, en effet toutes les informations des éléments graphiques sont stockés dans le diagrammes (nom, position, taille, liens, ...).

Actions sur le graphe – Événements

Pour pouvoir ajouter un `listener` sur le graphe, et ainsi pouvoir interagir avec des clics de souris, il faut utiliser la Classe `mxGraphControl` qui possèdent une méthode `addMousseListener`. Ainsi, sur le démonstrateur, le clic droit sur un élément permet de faire des choses différentes en fonction de l'élément.

- Sur tout élément graphique, un clic droit permettra de le supprimer.
  - Ce qui appellera à une super méthode `supprimer()`.
- Sur un acteur (Actif ou Passif), un clic droit permet d'afficher ou non la ligne de vie
  - Ce qui appellera une super méthode `afficherLigneDeVie(boolean)`.
- Sur un traitement, le menu contextuel permet d'afficher la flèche ou non de début de séquence
  - Ce qui appellera une méthode `setDebutSequence(boolean)`.

Lorsque vous sélectionnez un élément sur le graphe, si dans le code vous voulez récupérer l'élément graphique correspondant, et ainsi appeler différentes méthodes, vous devez utilisez deux méthodes, une développée par JGraphx, une présente dans libUML.

- `mxGraph.getSelectedCell()` qui va vous renvoyer une cellule, vous passez ensuite cette cellule en paramètre de la méthode suivante.
- `Diagramme.getElementGraphiqueViaCellule(mxCell)` qui va vous renvoyer un élément graphique correspondant à la cellule.

2.1.3 Le panneau de droite

Le panneau de droite est un panneau servant à afficher des informations. Ainsi, lorsque vous cliquez sur une classe, un tableau s'affiche avec la liste des méthodes et des attributs de la classe concernée. Ce panneau à pour but de vous montrer la possibilité de récupérer les informations stockés

et de les interpréter. Certaines informations sont présente dans une classe même si elle ne s'affiche pas dans le graphe, celle-ci on pour but de pouvoir être exploités si besoin, notamment avec de la génération de code (`static`, `final`, visibilité de la classe, ...).

2.2 Exemple de création d'un diagramme simple

Nous allons vous montrer comment fonctionne notre bibliothèque à l'aide d'un exemple, cet exemple est relativement simple et vient en complément du démonstrateur pour que vous puissiez comprendre rapidement la logique de la bibliothèque sans rentrer dans les détails du démonstrateur. Ainsi, nous allons réaliser le diagramme ci-dessous, en vous expliquant pas à pas la démarche qui est relativement simple.

```
1  package com.mxgraph.examples.swing;
2
3  import javax.swing.JFrame;
4
5  import com.mxgraph.swing.mxGraphComponent;
6  import com.mxgraph.view.mxGraph;
7
8  public class Exemple extends JFrame {
9    public HelloWorld() {
10     super("Hello, World!");
11     mxGraph graph = new mxGraph(); // On commence par créer un graphe
           ou apparaîtrons les éléments
12
13     /* on instancie différents éléments de modélisation */
14     ElementGraphique maClasse1 = new Classe();
15     ElementGraphique maClasse2 = new Classe();
16     Acteur monActeur = new ActeurActif();
17
18     /* L'appel à la super méthode créer aura pour effet d'afficher
          nos éléments de modélisation sur le graphe */
19     maClasse1.creer();
20     maClasse2.creer();
21     monActeur.creer();
22
23     /* on instancie des liens en mettant en paramètre la source et la
           destination du lien */
24     Lien monLien1 = new lien();
25     Lien monLien2 = new Lien();
26
27     /* de la même manière, on peut ensuite créer le lien pour l'
          afficher */
28     monLien1.creer();
29     monLien2.creer();
30
31     /* Finalement, on ne voulait pas de l'acteur, alors on le
          supprimer, automatiquement la flèche pointant sur Acteur sera
          supprimée*/
32     monActeur.supprimer();
33     monActeur = null; //on met l'acteur à null pour ne pas perdre de
```

```
        mémoire
34    }
35
36    public static void main(String[] args)
37    {
38      Exemple fenetre = new Exemple();
39      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
40      frame.setSize(400, 320);
41      frame.setVisible(true);
42    }
43  }
```

Listing 2.1 – Exemple de création d'un diagramme simple

2.3 Documentation

L'exemple ci-dessus est succins pour apprendre à se servir de la bibliothèque, cependant il vous donne la logique, pour voir toutes les possibilités de la bibliothèque, vous pouvez voir le démonstrateur.

Également, la documentation sous forme de Javadoc est accessible ici :
▷ `http://documentation.joohoo.fr/projet_IUT/libUML/index.html`
Grâce à celle-ci, vous pouvez savoir comment fonctionne et à quoi sert chaque méthode.

Chapitre 3

Poursuite de développement de la bibliothèque

La bibliothèque utilise entièrement JGraphx pour dessiner les composants. Nous avons fait en sorte qu'un utilisateur voulant se servir de libUML n'ai nullement besoin de la connaissance de JGraphx, ainsi, il faut restreindre l'utilisation de JGraphx au strict minimum (`mxGraph`...) quitte à redéfinir des fonctions appellant la fonction parente, ça à l'avantage d'intégrer la Javadoc de cette méthode à la documentation du projet.

La bibliothèque JGraphx à le grand défaut de n'avoir qu'une toute petite communauté, ainsi, vous trouverez rarement des réponses à votre problème sur un forum, nous avons donc beaucoup utilisé la documentation (Javadoc et manuel), cette documentation est accessible ici :

Chapitre 4

Conclusion du projet

Annexe A

Diagramme de classes

Annexe B

Diagramme de paquetages

Annexe C

Glossaire

Bibliothèque (p 3) – Composant programmé dans un langage donné fournissant des méthodes permettant d'effectuer des tâches voulut

Java (p 3) – Langage de programmation orienté objet moderne, il compile le programme pour ensuite l'exécuter sur une machine Java, ainsi le programme une fois compilé peut être exécuté sur différentes plateformes (Windows, Linux, Mac OS X, ...).

UML (p 3) – (Unified Modeling Language) Langage de modélisation graphique à base de pictogramme. Il est apparu dans le monde du génie logiciel dans le cadre de la conception orientée objet. Ce langage est composé de différents diagrammes, allant du développement à la simple analyse des besoins.