ISI 3

tp4. rEFACTORING de code & MVC

*Enseignant : Matignon Laëtitia*

POLYTECH LYON

Domaine Scientifique de La Doua – 15, Boulevard Latarjet

69622 VILLEURBANNE CEDEX

Tél. (33) 04.72.43.12.24 - Fax. (33) 04.72.43.12.25

Web: http://polytech.univ-lyon1.fr/

# Patrons de conceptions utilisés.

Notre application traite de la gestion de plusieurs tortues et de leurs déplacements sur une feuille de dessin. Pour cela, nous utilisons le patron d’architecture MVC afin de séparer tous les traitements (Contrôleur), les données(Métier) et la présentation (Vue).

Afin de faire communiquer notre modèle avec d’autres objets, nous allons utiliser le pattern observer. Notre fenêtre principale (la vue) contient donc une liste d’observateurs. Elle hérite de notre modèle Observable. Notre classe FeuilleDessin s’occupe de l’affichage des tortues et notre classe SimpleLogo s’occupe

De manière concrète, dès que nous voulons ajouter une nouvelle Tortue normale non autonome), nous avons ces classes internes qui réceptionnent le clic de la souris à l’aide de ***listeners***, ordres qui sont transférés au contrôleur qui va ensuite demander au modèle de ce mettre à jour et ainsi modifier la vue.

*//Implémentation du pattern observer*

*public void addTortue(Tortue t) {*

*tortues.add(t);*

*t.setParent(this);*

*notifyChanges();*

*}*

A chaque fois que l’on voudra faire une mise à jour, nous utiliserons un notifyObserver() dans notre méthode notifyChanges.

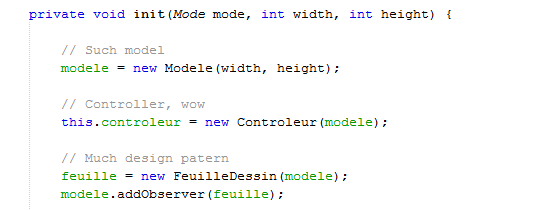
*public void notifyChanges() {*

*setChanged();*

*notifyObservers();*

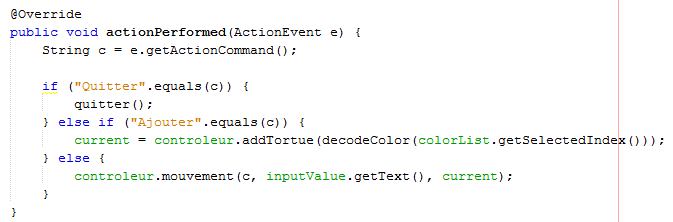
*}*

Dans notre vue, nous avons donc une méthode d’initialisation avec le modèle à observer.

**

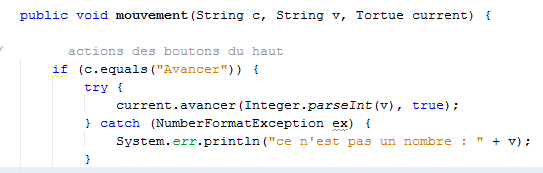
*Figure 1 : Méthode init de notre vue*

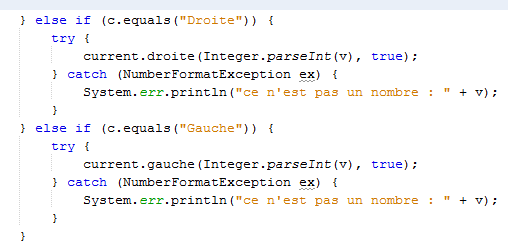
Notre conception doit aussi permettre d’ajouter facilement d’autres formes pour les tortues (cercles, polygones..). De ce fait nous auront nos différentes formes de tortues qui hériteront de notre classe abstraite GraphicTurtles.

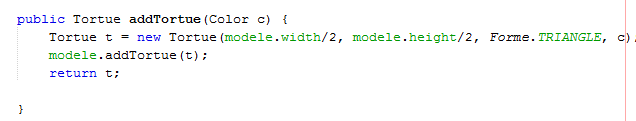
Les différents listeners : 

*Figure 2 : Les listeners (actionPerformed) de notre vue*

Notre contrôleur transforme ces ordres (mouvements, ajouts de tortues normales) et va dire au modèle quelles méthodes utiliser :

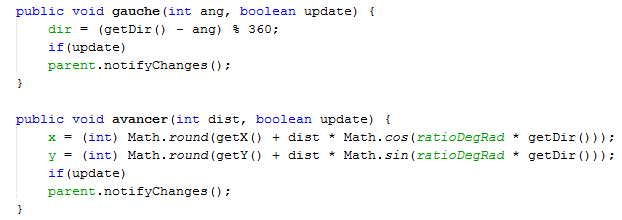




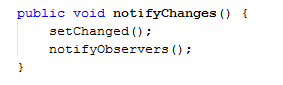


*Figure 3 : Récupération des commandes utilisateurs par le contrôleur*

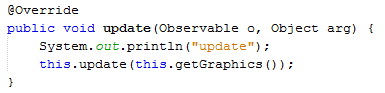
Puis le modèle effectue les différentes actions et va demander à la vue de se mettre à jour.



*Figure 4 : Le modèle effectue les différentes actions*

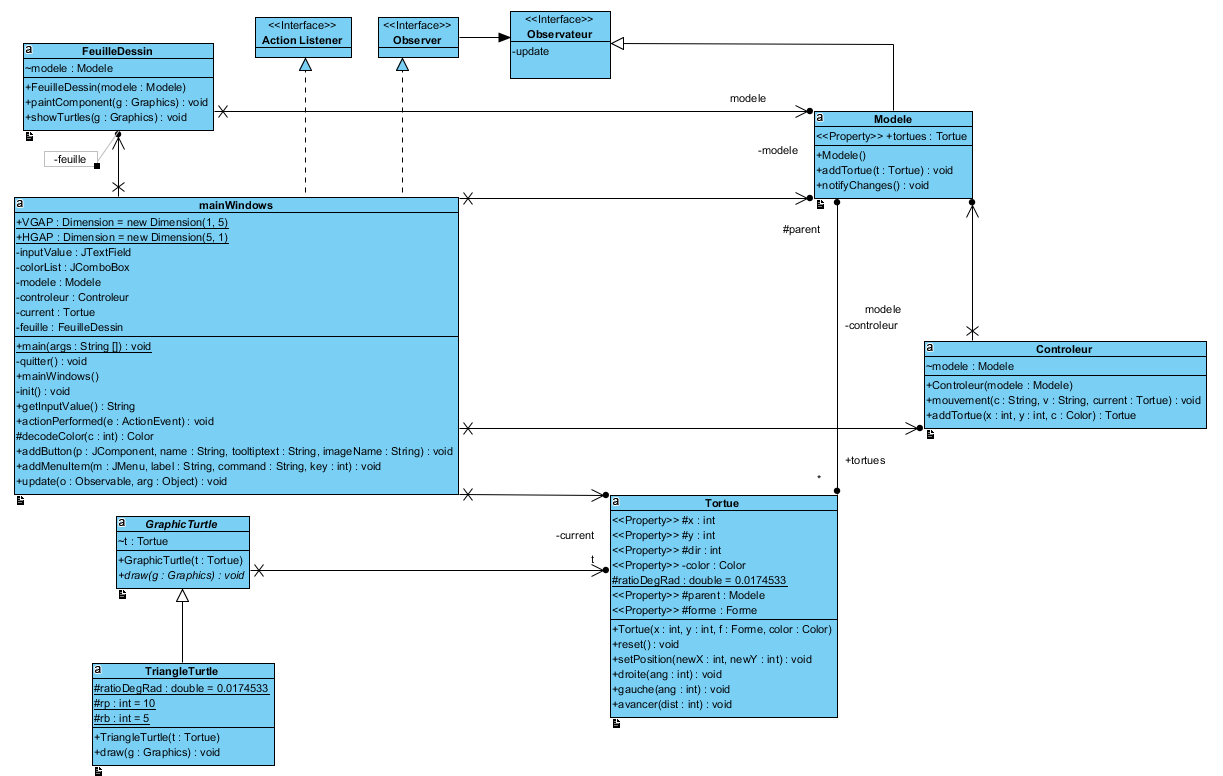


*Figure 5 : Le modèle rafraîchit la vue*



*Figure 6 : Le vue se rafraîchit via la méthode update.*

Nous obtenons donc, avant le refactoring, le diagramme UML suivant :



# Extension de l’application.