Université de Nice-Sophia Antipolis
POLYTECH CiP1

Donné le 28 avril 2006
À rendre le 9 juin 2006

Projet de Java : dessin géométrique

Avant de commencer : la qualité des commentaires, avec notamment la présence des antécédents, des conséquents, des invariants de boucle et de classe, et des rôles de chaque méthode, ainsi que les noms donnés aux variables, l'emploi à bon escient des majuscules et la bonne indentation rentreront pour une part importante dans l'appréciation du travail. Vous trouverez en annexe quelques informations utiles.

Dans le cadre de ce projet, vous devez respecter précisément les consignes qui vous sont données concernant la structure du programme (et donc, notamment, le nom et le rôle de chacune des classes). En revanche, vous êtes libres d'apporter diverses améliorations à votre programme, qu'elles aient été suggérées ou non par ce sujet. Pour cette raison, vous devez rendre avec votre projet un court rapport expliquant éventuellement la façon dont vous avez enrichi votre projet.

Enfin, si plusieurs projets s'avèrent identiques, leurs auteurs seront sanctionnés par un zéro.

1 L'application de dessin

1.1 Objectifs

Dans le cadre de ce projet, vous réaliserez un programme de dessin géométrique avec une interface graphique. Ce programme permettra à l'utilisateur de dessiner des figures géométriques (rectangles, cercles, etc.), d'en supprimer, d'en sélectionner certaines pour leur appliquer des transformations géométriques (translation ou autre) ou de modifier leurs propriétés (couleur et remplissage, notamment).

L'interface doit comporter un menu et des boutons permettant à l'utilisateur de choisir l'une des fonctions suivantes :

- dessiner une figure prédéfinie : segment, triangle, carré, rectangle, etc. ;
- choisir la couleur de dessin par défaut ;
- supprimer une figure du dessin ;
- sélectionner des figures ;
- afficher les propriétés d'une figure pour pouvoir les modifier ;
- enregistrer un dessin dans un fichier ou en charger un depuis un ficher ;
- quitter le programme.

1.2 Exemple d'exécution

Au lancement, la zone réservée au dessin est vierge (fig. 1). La fenêtre comporte un menu permettant de sauvegarder ou charger un dessin et des boutons permettant d'accéder aux fonctions décrites ci-dessus. Enfin, une barre d'état en bas de la fenêtre permet de renseigner l'utilisateur sur les actions en cours.

Lorsque plusieurs figures ont été dessinées, le mode *sélection* (fig. 2) permet de *sélectionner* certaines d'entre elles pour leur appliquer une même transformation. Par ailleurs, on peut modifier les attributs d'une figure en cliquant sur le bouton *Propriétés* puis sur la figure en question : cela fait apparaître une boite de dialogue contenant les propriétés de la figure (fig. 3).

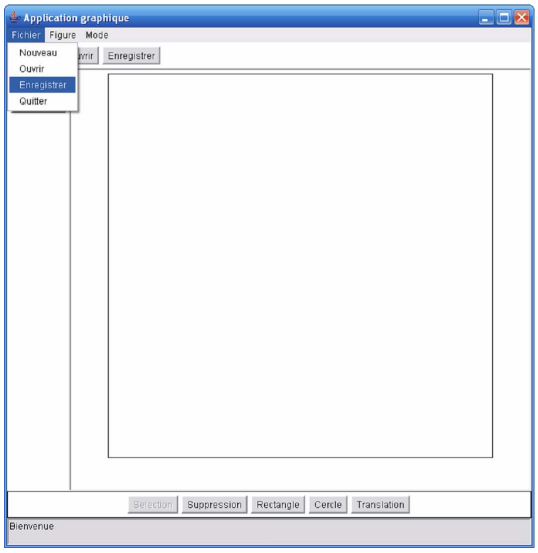

FIG. 1 – Fenêtre principale : un menu en haut, une zone de dessin au centre, des boutons pour le choix des figures et du mode courant, une barre d'état en bas pour informer sur le mode en cours.

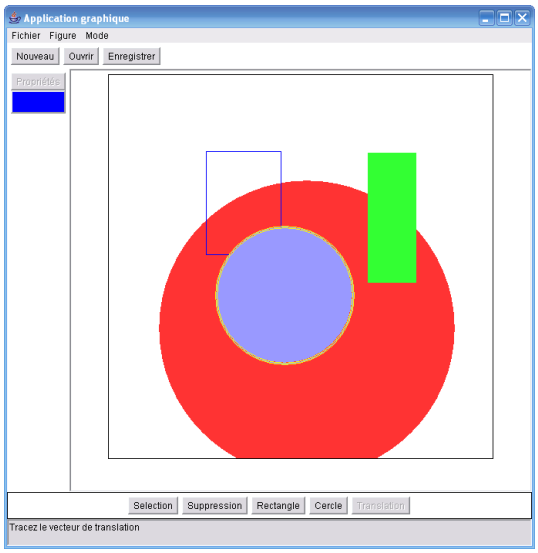

FIG. 2 – Fenêtre principale : plusieurs figures ont été dessinées.

La fenêtre *Propriétés* permet de modifier la figure choisie. On peut l'utiliser pour choisir de remplir ou non la figure, définir sa couleur et son ordre d'apparition (cette dernière fonction est utile en cas de chevauchement des figures).

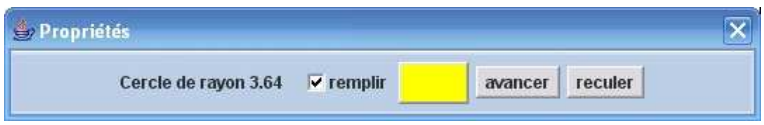

FIG. 3 – Fenêtre permettant de modifier les propriétés de la figure choisie. La couleur, le remplissage et l'ordre des figures peuvent être modifiés.

1.3 Structure du programme

Pour programmer cette application, vous utiliserez les classes suivantes :
- des classes d'interface graphique
  - une classe `FenêtrePrincipale` qui organise tous les éléments de l'interface ;
  - une classe `ZoneDessin` qui traite les événements souris liés au dessin et aux choix de mode ;
  - une classe `FenêtrePropriétés` qui présente les propriétés d'une figure ;
  - une classe `BarreBoutons` qui organise les différents boutons ;
  - une classe `BoutonCouleur` qui décrit le bouton utilisé pour sélectionner une couleur ;
- des classes géométriques
  - une classe abstraite `Figure` dont dérivent toutes les figures ;
  - une classe `Dessin` qui contient toutes les figures ;
  - des classes héritant de la classe `Figure`, telles que la classe `Rectangle` ou la classe `Cercle`.

2 Classes géométriques

2.1 Classe `Point`

Définissez une classe `Point` qui contient, sous forme de `double`, les coordonnées d'un point.

2.2 Classe abstraite `Figure` et ses héritières

La classe abstraite `Figure` contient les déclarations des méthodes et des variables communes à toutes les figures. Au fur et à mesure de la construction de l'application, il faudra y ajouter les éléments nécessaires.

Pour le moment, on a seulement besoin qu'une figure possède une méthode `dessine` qui prend en paramètre un contexte graphique et une échelle. Ajoutez à la classe `Figure` la méthode suivante :

```
public abstract void dessine(Graphics g, int échelle);
```

Le rôle de cette méthode est d'appeler les commandes de tracé du contexte graphique `g` . Contrairement à ce que vous avez fait en TP où vous avez travaillé avec des coordonnées entières sur une grille, nous désirons gérer des coordonnées décimales. Il faut donc choisir combien de pixels représentent une unité. Précisément, vous transmettrez à la fonction `dessine` un paramètre entier `échelle` qui indique ce nombre de pixels.

Pour dessiner une figure, il faut aussi connaître sa couleur. Ajoutez une variable `couleur` à la classe `Figure`. Les constructeurs de vos classes prendront comme paramètres une couleur et deux points. Pour le rectangle, ces deux points seront les deux extrémités d'une diagonale. Pour le cercle, il s'agira du centre du cercle et d'un point sur le cercle...

Pour sélectionner une figure, il est nécessaire de savoir mesurer la distance d'un point à celle-ci. Ajoutez une méthode

```
public abstract double distance(Point p);
```

à la classe `Figure`. Modifiez les classes héritant de `Figure` en conséquence.

Vous trouverez en annexe les formules mathématiques utiles.

2.3 Classe `Dessin`

Définissez la classe `Dessin` qui contient un tableau de figures (l'ensemble des figures du dessin), qui permet d'ajouter ou de supprimer facilement une figure et qui peut renvoyer la figure la

plus proche d'un point donné. On définira les méthodes

```
public void ajoute(Figure f) {...}
public void supprime(Figure f) {...}
public Figure donnePlusProche(Point p) {...}
```

puis une méthode

```
public void dessineTout(Graphics g, double échelle) {...}
```

3 Classes d'interface de test

Pour pouvoir tester vos classes, il est maintenant nécessaire de programmer une interface de test rudimentaire. Celle-ci sera enrichie par la suite.

3.1 Classe `BarreBoutons`

Définissez la classe `BarreBoutons` qui hérite de la classe `Panel` et qui implémente l'interface `ActionListener` pour gérer les appuis sur les boutons qu'elle contient. En particulier, elle doit posséder une méthode publique `modeCourant()` qui indique quel est le dernier bouton sur lequel on a cliqué. Pour le moment, la barre doit comporter au moins les boutons permettant de créer un rectangle (en en traçant une diagonale), de créer un cercle (en en traçant un rayon) et de supprimer l'élément le plus proche du point où l'on clique.

3.2 Classe `ZoneDessin`

Définissez la classe `ZoneDessin` qui hérite de la classe `Panel`. Elle possède un champ

```
public static final int échelle;
```

(voir la méthode `dessine` dans la partie 2.2 pour son utilisation). Elle gère les événements souris et implémente notamment l'interface `MouseListener`. Son constructeur crée un objet de la classe `Dessin` et un objet de la classe `BarreBoutons`. Bien sûr, les réactions à un clic de la souris dépendront du mode renvoyé par la méthode `modeCourant()` de la barre de boutons (cf. section 4).

Enfin, il est nécessaire de fournir une méthode `paint(Graphics g)` qui trace l'ensemble des figures dans le contexte graphique `g`.

3.3 Classe `FenêtrePrincipale`

Définissez la classe `FenêtrePrincipale` qui hérite de la classe `Frame`. Cette fenêtre devra posséder au moins un menu en haut, une zone zone de dessin et sa barre de boutons, comme sur le modèle (fig.1). Il ne faudra surtout pas oublier de mettre une entrée *Quitter* dans le menu *Fichier*.

Ajoutez enfin une fonction d'aide qui affiche le mode d'emploi du programme. Vous êtes libres d'effectuer cet affichage de la façon qui vous paraît la meilleure.

Vérifiez que tout fonctionne comme prévu.

4 Les différents modes

Nous voulons, pendant le déroulement d'une session de dessin, tour-à-tour dessiner, sélectionner, modifier ou translater des figures. Toutes ces actions doivent être faisables à la souris. Le nombre de boutons étant limité à 3, il va falloir définir des modes pour qu'une même action n'ait pas le même effet suivant le mode courant de l'application. Les différents modes sont décrits dans les parties suivantes.

Vous ajouterez un menu qui permette de sélectionner le mode courant ainsi que des boutons donnant un accès plus rapide à certains de ces modes.

4.1 Mode *dessin*

Nous allons développer ce qui vous a été demandé en TP. En particulier, une figure ne sera plus positionnée aléatoirement dans la zone de dessin lors de sa création.

Création : Pour créer une figure, l'utilisateur doit cliquer pour définir l'origine de la figure, puis glisser pour définir un deuxième point de contrôle (ce sont ces deux points qui seront donnés comme paramètres au constructeur). Durant le mouvement de la souris, tant que le bouton est appuyé, on souhaite qu'une petite croix reste visible pour indiquer le point de départ du mouvement, *i.e.* l'endroit où le bouton a été enfoncé.

Affichage : Chaque figure peut s'afficher à l'aide de sa méthode `dessine(Graphics g, int échelle)`. On souhaite ajouter la possibilité de dessiner soit le contour de la figure, soit la figure remplie. On ajoutera par la suite à l'interface graphique les éléments nécessaires à l'utilisation de cette fonction.

Couleur des figures : Ajoutez à la barre de boutons un bouton permettant de choisir la couleur des prochaines figures créées.

4.2 Mode *sélection*

Nous voulons maintenant sélectionner une ou plusieurs figures pour pouvoir leur appliquer une même transformation. Pour cela, on souhaite désormais qu'un clic de souris dans la zone de dessin sélectionne la figure la plus proche du pointeur. Lorsqu'un objet est sélectionné, il est mis en surbrillance, c'est-à-dire qu'on l'entoure de jaune. Pensez à traiter cela différemment, selon que l'objet est rempli ou non.

Plusieurs objets peuvent être sélectionnés à la fois. Cliquer près d'un objet déjà sélectionné doit le déselectionner.

4.3 Mode *translation*

Lorsqu'on est en mode translation, on souhaite déplacer tous les objets sélectionnés en même temps. Ajoutez à la classe `Figure` une méthode

```
public abstract void translate(Point a, Point b)
```

et implémentez-la dans chacune des classes dérivées : elle doit appliquer à la figure la translation de vecteur $\overrightarrow{u} = \overrightarrow{ab}$. Les points `Point a` et `Point b` sont les deux points définis par l'appui et le relâchement du bouton de la souris. Complétez l'interface comme il se doit. La translation (fig. 4) correspond à l'opération matricielle suivante :

$$t_{\vec{u}} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \vec{u}_x \\ \vec{u}_y \end{bmatrix} \tag{1}$$

4.4 Mode *propriétés*

Nous voulons maintenant modifier les propriétés d'une figure. Plus précisément, il faut qu'un clic de souris en mode *propriétés* entraîne l'ouverture d'une fenêtre de propriétés pour la figure la plus proche. On souhaite lire des informations sur la figure et pouvoir les modifier.

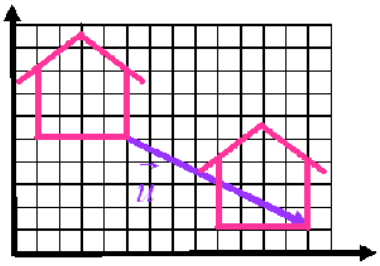

FIG. 4 – Translation

Propriétés de la figure : Créez une classe `FenêtrePropriétés` qui hérite de la classe `Dialog`. Ajoutez à la classe `Figure` un attribut de ce type. Cet attribut sera la fenêtre des propriétés de la figure considérée.

En plus des informations sur la figure, ajoutez à cette fenêtre une boîte à cocher permettant de remplir la figure ou non et un bouton permettant de choisir la couleur de la figure. La classe `Figure` devra implémenter les interfaces `ItemListener` et `ActionListener` pour prendre en compte les événements provenant de ces deux éléments.

Position de la figure dans le dessin : Il est intéressant, lorsque le dessin est composé de plusieurs figures, de pouvoir modifier leur ordre. On souhaite avoir la possibilité de faire avancer une figure vers l'avant-plan, ou de la faire reculer vers l'arrière-plan. Ainsi, quand les figures se chevauchent, le dessin changera selon l'ordre dans lequel on trace les figures. Modifiez la classe `Dessin` pour qu'elle permette de modifier cet ordre et ajoutez à la fenêtre des propriétés de chaque figure deux boutons *avancer* et *reculer* qui permettront de modifier la profondeur de la figure en échangeant sa position avec la figure qui la précède ou qui lui succède.

4.5 Mode *suppression*

Nous souhaitons pouvoir supprimer des figures. En mode suppression, un clic de souris dans la zone de dessin doit supprimer la figure la plus proche.

5 Sauvegarde et utilisation des fichiers

Implémentez dans la classe `Dessin` une méthode de sauvegarde qui écrit dans un fichier, dont le nom sera choisi à l'aide d'un `JFileChooser` (voir l'annexe pour plus de précisions), les figures du dessin. Attention, seule la classe `Figure` doit déclarer implémenter l'interface `Serializable`, mais pas la classe `Dessin` ni aucune autre classe. Implémentez aussi une méthode de lecture d'une liste de figures depuis un tel fichier de sauvegarde.

Vous ajouterez à l'interface graphique les éléments nécessaires pour donner accès à ces fonctions.

6 Bonus

Voici quelques possibilités d'amélioration de votre programme :

- ajouter une barre d'état qui guide l'utilisateur ;
- ajouter une function *undo* ;
- modifier la classe rectangle pour qu'un rectangle puisse ne pas avoir ces cotés parallèles aux axes ;

– ajouter d'autres transformations géométriques ;
– compléter la fonction d'aide pour qu'elle lise les informations qu'elle affiche depuis un fichier respectant un certain formatage ;
– ...

7 Récapitulatif

Vous devez rendre une archive `jar` avant le 10 juin 2006 en l'envoyant à l'adresse `sujetmastermind@yahoo.fr`. Cette archive sera semblable à celle que vous aviez rendue pour le premier projet. Son nom sera de la forme `ELEVE1_ELEVE2.jar` et elle comportera :

– le rapport (au format HTML ou RTF) ;
– les fichiers Java : au minimum les cinq classes de l'interface graphique ainsi que les quatre classes géométriques ;
– la Javadoc.

Dans chacun des fichiers de l'archive, n'oubliez pas d'indiquer les noms des élèves du binôme.

Toutes les fonctionnalités mentionnées dans le sujet sont **obligatoires** (à l'exception de celles proposées dans la partie Bonus). Vous préciserez dans votre rapport les choix particuliers d'implémentation que vous aurez faits, les difficultés que vous aurez pu rencontrées et la façon dont vous les aurez résolues ainsi que les éventuelles améliorations que vous aurez ajoutées. Ces dernières seront prises en compte dans la notation.

8 Annexes

8.1 Distance d'un point à une figure

Distance d'un point à un rectangle : La distance $d(P,[A,B])$ d'un point P à un segment $[A,B]$ est égale à la distance du point à son projeté orthogonal P' sur la droite (AB) si $P' \in [A,B]$ et est égale à $\min(AP,BP)$ sinon (fig. 5). Pour calculer la distance d'un point à un rectangle, il suffit d'utiliser à bon escient la distance d'un point à un segment.

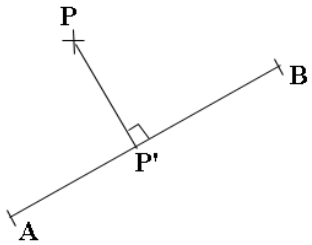

FIG. 5 – Calcul de la distance d'un point P au segment $[A,B]$, lorsque le projeté orthogonal P' de P appartient au segment $[A,B]$.

Distance d'un point à un cercle : La distance $d(P,C)$ d'un point P à un cercle C est la valeur absolue de la différence entre la distance du point P au centre c du cercle C et et rayon r du cercle (fig. 6).

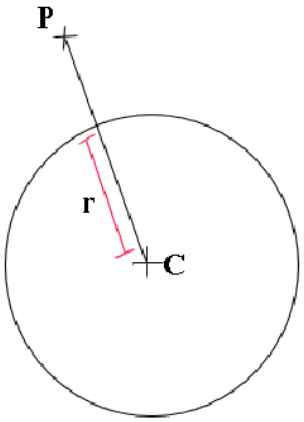

FIG. 6 – Calcul de la distance d'un point à un cercle.

8.2 Exemple d'utilisation des classes `swing`

Vous utiliserez principalement les bibliothèques `java.awt` et `java.io` qui ont été vues en cours. Vous trouverez leur documentation complète à l'addresse
`http ://java.sun.com/j2se/1.5.0/docs/api/`

Cependant, vous aurez aussi besoin des quelques composants suivants, tirés de la bibliothèque `javax.swing` :

– la boîte de dialogue `JFileChooser` permettant de sélectionner un fichier et qui peut être utilisée, par exemple, de la façon suivante :

```
import javax.swing.JFileChooser;
import java.io.File;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
...
JFileChooser filechooser = new JFileChooser();
...
filechooser.showOpenDialog(this);
File f = filechooser.getSelectedFile();
...
ObjectInputStream ois = new ObjectInputStream(new FileInputStream(f));
```

Vous n'oublierez pas de traiter correctement le cas où `f` contient `null` après l'appel à `getSelectedFile` et les exceptions éventuelles.

– la boîte de dialogue `JColorChooser` permettant de sélectionner une couleur et qui peut être utilisée, par exemple, de la façon suivante :

```
import javax.swing.JColorChooser;
...
Color fColor; // la couleur actuelle
...
fColor = JColorChooser.showDialog(this, "Choisissez une couleur", fColor);
```

Vous n'oublierez pas de traiter correctement le cas où `fColor` contient `null` après l'appel à `showDialog`.