Classe, objet, interface, héritage, polymorphisme

Ce TD s'appuie sur l'exemple objet très classique de la visualisation de formes (cercles, rectangles, etc). Vous devez dans un premier temps récupérer les fichiers situés dans le répertoire

`Bibliotheque/STAGE_PROG_SIL/Formes`

Exécutez l'application (sous Eclipse, sélectionnez la classe `VisualiseurDeFormes` puis cliquez sur Run/Run as/Application). Cliquez sur le bouton, agrandissez ou réduisez la fenêtre.

Exercice 1 : Classes, hiérarchie

Parcourez les différents fichiers, repérez les classes. Pour chaque classe, indiquez si elle hérite d'une autre et si oui, laquelle. Pourquoi `Forme` est-elle une interface et non une classe ?

Quelle est la classe qui représente la fenêtre de l'application ? Changez donc le titre affiché sur la fenêtre.

Quelle est la classe qui se charge de dessiner les cercles ? Précisez la méthode où l'affichage des formes/cercles est réalisé. Cette méthode est-elle appelée explicitement quelque part dans l'application. Regardez alors la documentation de cette méthode dans `JComponent`.

Exercice 2 : Objets

Listez tous les objets instanciés (de façon visible) dans le constructeur de `VisualiseurDeFormes`. A quoi sert chacun des objets ? Quelles sont les relations entre ces objets ?

Exercice 3 : Programme principal

Recherchez une méthode `main` dans les classes. Où se trouve-t-elle ? Comment crée-t-elle la fenêtre graphique ? La variable objet `visu` référence donc une instance de la classe `VisualiseurDeFormes`.

Modifiez-la pour qu'elle crée deux applications "Visualiseur de formes" en même temps. Changez la classe `VisualiseurDeFormes` pour que les titres des fenêtres des deux applications soient différents (mettons `prems` et `deuze`).

Exercice 4 : Polymorphisme : autres formes

Sur le même principe que la classe `Cercle`, rajoutez une classe `Rectangle` dans un fichier `Rectangle.java`. Quelles vont être les données membres nécessaires ? Quelle(s) est (sont) la (les) méthode(s) à définir ?

Pour vérifier que votre classe `Rectangle` fonctionne correctement, modifiez le constructeur de `VisualiseurDeFormes` de façon que celui-ci affiche au moins un rectangle. Pour ce faire, rajoutez les lignes suivantes dans le constructeur de `VisualiseurDeFormes` :

```
Rectangle r = new Rectangle (10, 10, 100, 50 );
m_formes.add( r );
```

NB : Cela ajoute au vecteur de formes `m_formes` une instance de la classe `Rectangle`.

Vérifiez que votre `Rectangle` s'affiche immédiatement sans aucune autre modification du code. Ainsi, les cercles et rectangles sont stockés dans un même objet (un vecteur d'`Objet`) et pourtant leur affichage est bien différencié. Comment cela est-il possible ?

Que se passe-t-il si vous enlevez dans la méthode `ZoneDeDessin.paintComponent` le terme `(Forme)` devant `m_formes.get( i )` ? Ce terme permet le *transtypage* d'une référence à un objet vers un type plus spécialisé. Une fois le problème constaté, remettez le code dans l'état précédent.

Maintenant, rajoutez les lignes suivantes dans le constructeur de `VisualiseurDeFormes` :

```
m_formes.add( new String( "Texte" ) );
```

Le programme JAVA compile-t-il ? Si oui, que se passe-t-il à l'exécution ? Constatez que la machine virtuelle a déclenché une `Exception`. Comment s'appelle-t-elle ? En fait, le transtypage ne peut réussir que si l'objet avait le bon type au moment de son *instanciation* (ie le type `Forme` ou tout type dérivé).

Exercice 5 : Bouton de création de rectangles

Rajoutez maintenant un nouveau bouton à votre application, de nom "Rectangle", qui rajoute un rectangle de taille (entre 1 et 100) et position aléatoires (entre 1 et 300) à la liste des formes à afficher à chaque fois que l'on appuie dessus. Suivez le modèle du bouton "Cercle".

Exercice 6 : Bouton "Reset"

Faites un bouton qui efface toutes les formes lorsqu'on appuie dessus.

Exercice 7 : Bouton "Forme aléatoire"

Faites un bouton qui crée aléatoirement un cercle ou un rectangle supplémentaire. Comment faire pour ne pas avoir trop de répétitions de code ?

Exercice 8 : Classe abstraite : forme colorée

On veut maintenant avoir des formes de couleurs différentes (cercles ou rectangles). Il s'agit donc d'associer à toute forme colorée une couleur (classe `Color` en JAVA). Peut-on rajouter une donnée membre à l'interface `Forme` ? Si non, que pensez-vous de la technique consistant à rajouter une donnée membre couleur à chaque sous-classe de `Forme` ?

Modifiez la hiérarchie en rajoutant une classe `FormeColoree`. Pourquoi cette classe doit-elle être abstraite ?

Rajoutez ensuite des boutons qui permettent de changer la couleur de dessin des nouvelles formes que l'on rajoute.

Exercice 9 : Classes imbriquées

En regardant le code, vous constaterez que la classe `BoutonCercleAction` est incluse dans la classe `VisualiseurDeFormes`. Ce n'est pas juste une indication de portée. Cela indique que tout objet `BoutonCercleAction` est associé à *une* instance de `VisualiseurDeFormes`. Vérifiez cette assertion en modifiant les lignes suivantes :

```
Random r = m_visualiseur.getRandom();
Cercle c = new Cercle(
r.nextInt( 200 ),
r.nextInt( 200 ),
r.nextInt( 40 )+ 20 );
m_visualiseur.getFormes().add( c );
m_visualiseur.repaint();
```

par

```
Random r = getRandom();
Cercle c = new Cercle(
r.nextInt( 200 ),
r.nextInt( 200 ),
r.nextInt( 40 )+ 20 );
getFormes().add( c );
repaint();
```

Le programme compile toujours et s'exécute de la même façon. L'objet de la classe incluse accède directement aux méthodes de sa classe englobante en donnant le nom de la méthode à appeler.

Vous pouvez éliminer maintenant toute référence à `m_visualiseur` dans la classe `BoutonCercleAction`.

Exercice 10 : IHM, création de classe dérivée : rectangles de position et taille spécifiées à la souris

Regardez la classe `MouseAdapter`. Elle permet de spécifier des réactions à des événements de la souris. Pour que l'utilisateur puisse tracer un rectangle en cliquant sur un coin puis en maintenant le bouton appuyé pour étendre son rectangle, il va falloir associer une classe dérivant de `MouseAdapter` à la zone de dessin. Vous allez donc créer une nouvelle classe `RectangleParDragSouris` qui dérive de `MouseAdapter` et qui implémente un ensemble de méthodes de cette superclasse :

— `mousePressed` : pour mémoriser le premier coin du rectangle.
— `mouseReleased` : pour mémoriser le deuxième coin du rectangle, créer le rectangle et l'ajouter à la liste des formes.

Que faut-il faire en plus si on veut afficher le rectangle pendant la définition de sa taille ?