TD 7 : interfaces

1 Présentation

Une *interface* permet de spécifier que les classes qui l'implémentent (`implements`) possèdent des fonctionnalités communes définies par cette interface. Par exemple, si une interface `A` déclare une méthode `toto`, , alors toutes les classes qui implémentent `A` devront posséder une méthode `toto` ou bien être abstraites :

```
interface A
{
    public void toto();
}

class B implements A
{
    public void toto(){...}
}

class C implements A
{
}// erreur à la compilation: pas de méthode toto!
```

Une fois l'interface définie on peut référencer des objets qui implémentent cette interface de la manière suivante :

```
...
A ref=new B();
A tab[]=new A[10];
tab[0]=ref;
...
```

2 Exercices

Supposons que l'on dispose de classes `Cercle`, `Triangle`, `Carre`, `Forme` et `Curseur`. Pour maintenir un conception objet simple, on ne souhaite pas que la classe `Curseur` fasse partie de la même hiérarchie de classes que `Cercle`, `Triangle`, `Carre`, et `Forme` (ces dernières pourront, elles, faire partie d'une même hiérarchie).

On voudrait cependant que toutes ces classes (à part `Forme`) possèdent une méthode `dessiner` et que l'on puisse gérer dans un même tableau des instances de toutes ces classes.

- Réflechissez à une solution en utilisant les interfaces.
- Écrivez l'interface.
- Écrivez les classes `Cercle`, `Triangle`, `Carre`, `Forme` et `Curseur`. Pour simplifier, vous pouvez laisser le corps des fonctions `dessiner` vides.
- Écrivez une classe principale `InterfaceTest` qui crée des `Cercle`, `Triangle`, `Carre`, `Curseur`. Ces objets seront stockés dans un même tableau. Les méthodes dessiner des objets seront appelés dans une boucle qui parcourra les éléments du tableau.

3 `ActionListener`

L'API java propose l'interface suivante :

```
interface ActionListener
{
    public void actionPerformed(ActionEvent e);
}
```

Celle-ci est notamment utilisée par les boutons d'une interface graphique (classe `JButton`). En effet, lorsqu'on crée un button, on doit lui dire quelle fonction doit etre appelée quand l'utilisateur clique dessus. Pour cela on peut utiliser la méthode `addActionListener(ActionListener l)` de la classe `JButton`.

Voici un programme qui crée deux buttons (en plus il vous montre une nouveauté : comment on peu créer un appelet dans le cadre d'une application) :

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

// Deux  nouveautés ici:
// 1: Création d'une fenêtre SANS appletviewer (application)
// 2: Créations de boutons

public class ExempleBoutons extends JApplet
{
    // comme d'habitude, la fonction init ...
    // (mais cette fois c'est nous qui allons l'appeler directement)
    public void init()
    {
        // demander a l'applet ou est-ce qu'on peut ajouter des
        // éléments graphiques
        Container cp = getContentPane();
        // type rangement pour les boutons qui vont être crées
        cp.setLayout(new FlowLayout());

        // creation de 2 bouttons
        JButton b1 = new JButton("Bouton 1");
        JButton b2 = new JButton("Bouton 2");

        // ajout des deux bouttons a la fenetre
        cp.add(b1);
        cp.add(b2);
    }

    // Cette fois il y a un "main" dans un Applet!!
    public static void main(String[] args)
    {
        // ici on fait une partie des choses qui se
        // passaient automatiquement lorsqu'on lance appletviewer

        // on crée l'objet de type JApplet
        JApplet applet = new ExempleBoutons();
        // on crée une fenêtre principale
        JFrame  frame  = new JFrame("Titre de la fenetre");
        // Pour fermer l'application
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // on ajoute notre affichage dans la fenêtre principale
        frame.getContentPane().add(applet);
        // taille de la fenêtre principale
        frame.setSize(200,200);
```

```
        // appel de notre méthode d'initialisation
        applet.init();
        // démarrage de notre application
        applet.start();
        // apparition de la fenêtre a l'écran
        frame.setVisible(true);
    }
}
```

On va modifier ce programme pour afficher un message :

- A l'aide des informations ci-dessus créez une classe qui contiendra une seule méthode qui affichera sur la console (`System.out.println`) le message *bouton appuyé !* lorsqu'on appuiera sur un des boutons. On appellera `addActionListener` (qui fait partie de la classe `JButton`) à la fin de la méthode `init`.
- Faites la même chose, mais cette fois, sans créer de classe supplémentaire (tout sera dans la classe `ExempleBouttons`).