S2 Module M2103 - Programmation Orientée Objet (POO)

EX 5 : Classes Abstraites

Objectifs

- Comprendre l'utilité d'une classe abstraite
- Programmer des classes abstraites

1. Editeur graphique

POUR CET EXERCICE, METTEZ VOUS EN BINOME : un étudiant jouera le rôle de M. Pasquier, un étudiant jouera le rôle de M. Dupont.

M. Pasquier est chef de projet et développeur.
Le but du projet est de développer un éditeur de dessin sur SmartPhone en Java.
Il ne peut pas tout developper tout seul et doit collaborer avec M. Dupont.

M. Pasquier va prendre en charge la partie affichage graphique et stockage en mémoire de toutes les formes graphiques dans un tableau. Il devra notament écrire un programme affichant des statistiques avec toutes les formes crées classées par surface decroissante.
M. Dupont va prendre en charge la partie implémentation de différentes formes graphiques : des cercles, des rectangles ... Cette liste pourra évoluer au fur et à mesure du projet.

1) TRAVAIL DE M. PASQUIER
M. Pasquier a écrit une classe générale Editeur qui inclue une méthode afficherStats. Cette méthode affiche une fenetre graphique avec les informations sur une forme graphique passée en parametre à cette méthode (notamment surface). Cette classe vous est fournie ci-dessous. Elle comporte des instructions liées aux interfaces graphiques que vous verrez lors d'un prochain cours et que vous n'avez pas besoin de comprendre dès maintenant.

```
import java.awt.* ;
public class Editeur {
        public void afficherStats (FormeGeometrique f) {
           // Declarer une reference vers une fenetre independante
           Frame frame ;

           // Creer un objet de type fenetre independante
           frame = new Frame ("Statistiques sur une forme graphique");

           // Declarer une reference vers une etiquette
           Label labelNom ;

           // Creer une etiquette avec toutes les informations sur la forme
           String classe = f.getClass().getName();

           labelNom = new Label (classe + ", x = " + f.getX() + ", y = " + f.getY() + ", Surface = " + f.surface
() + ", Perimetre = " + f.perimetre ());

           // Ajouter l'etiquette dans la frame
           frame.add(labelNom);
```

```
            // Fixer la taille de la fenetre
            frame.setSize (300, 200);

            // Afficher la fenetre
            frame.setVisible (true);
      }
}
```

a) créez un projet et copiez y la classe Editeur.

b) Vous devez programmer la classe abstraite FormeGeometrique qui comporte
- deux attributs x, y
- deux méthodes implémentées : void déplacer () et void afficherPosition ()
- deux méthodes abstraites : perimetre () et surface ()

c) Est-ce que votre programme peut s'exécuter avec uniquement les classes Editeur et la classe abstraite FormeGeometrique ? .....
Que faut-il d'autre ? .....

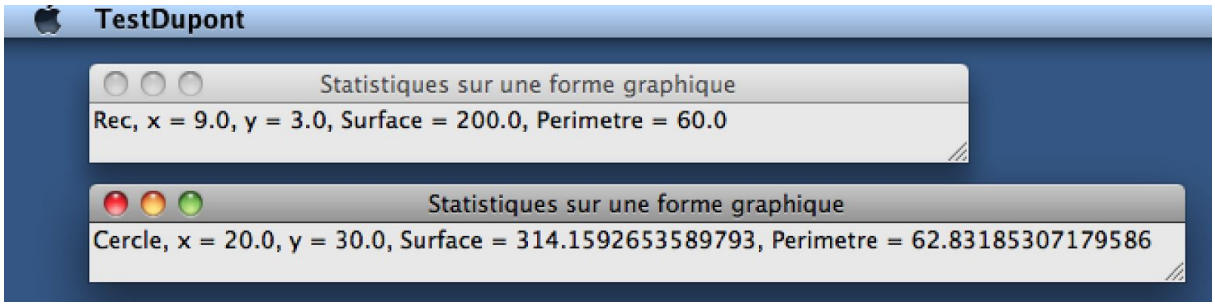
M. Pasquier ne s'est pas pré-occupé du type de FormeGeometrique *.*
M. Pasquier sait que pour utiliser ses classes, M. Dupont devra passer en paramètre à la méthode afficherStats() une instance d'une classe concrète qui implémente obligatoirement les méthodes spécifiées comme abstraites dans la classe FormeGeometrique *.*

2) TRAVAIL DE M. DUPONT
a) Faites le travail de M. Dupont : programmez une classe Rec et une classe Cercle qui implémentent les méthodes abstraites de la classe abstraite FormeGeometrique.
Rappel : Le périmètre d'un cercle de rayon R est P = 2 π R. Sa surface est S = π R^2.
La valeur de PI est fournie par la classe Math (cherchez dans la javadoc).
Programmez aussi d'autres classes, par exemple TriangleRectangle.

b) Complétez ensuite la classe suivante pour tester le programme

```
public class TestDupont{
  public static void main (String args[]) {

    // M. Dupont cree un rectangle
          …
    // M. Dupont appelle les methodes developpees par M. Pasquier
    // Il est *oblige* de passer a la methode afficherStats un objet instance d'une classe implementant la classe abstraite FormeGeometrique
    Editeur ed = new Editeur ();
          …

    // Changer les coordonnées de la forme en appelant la méthode déplacer

    // Demander de nouveau d'afficher des statistiques sur cette forme

    // Idem avec un cercle
  }
}
```

Exemple d'affichage :



c) Que se passe-t-il si M. Dupont n'implémente pas les méthodes abstraites de la classe abstraite FormeGeometrique ? Enlevez le « extends » FormeGeometrique dans une des classes.
Est-ce qu'il y a une erreur à la compilation ou à l'exécution ? .....

2. Quelle est l'instruction provoquant une erreur ? .....

2. Banque et classes abstraites

Dans votre projet banqueHeritee est-ce logique d'avoir une classe concrète Compte ? NON
Est-ce que vous aurez à un moment donné besoin de créer un objet de type Compte qui ne soit ni un compte courant ni un compte d'épargne ? NON (on n'aura jamais besoin de créer un objet de type Compte : c'est un niveau trop abstrait / général)

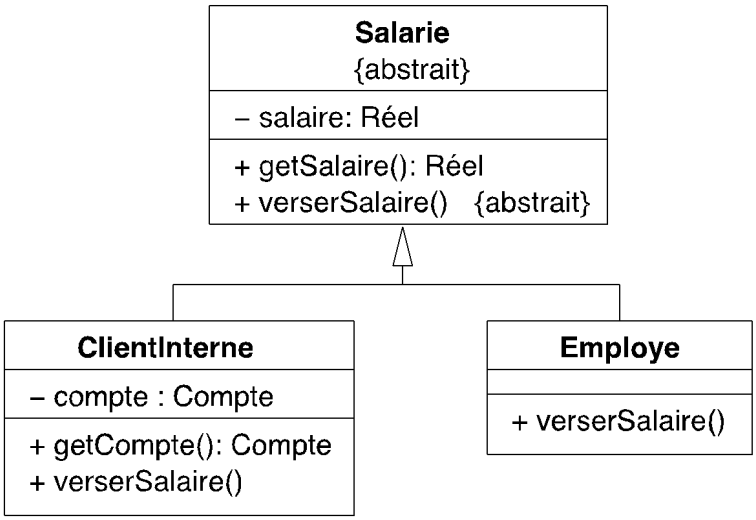
Dupliquer votre projet banqueHeritee en un projet banqueAbstraite.
Modifier les différentes classes de manière à gérer de manière plus générale une classe abstraite CompteAbstrait et ses sous-classes.

3. Salariés de la banque

On distingue deux catégories de salariés dans une banque :
1) ceux qui ont un compte dans la banque (les « clients internes »)
2) ceux qui sont clients d'une autre banque (les « employés »)

Suivant le cas, la méthode verserSalaire doit effectuer un traitement différent :
1) un virement sur le compte interne
2) l'envoi d'un chèque que l'on simulera par un affichage à l'écran de la phrase « Versement de salaire sur compte externe » accompagné du nom du salarié, et du montant de son salaire).

Pour représenter les attributs et méthodes communs à ces deux classes, on les fait hériter d'une classe abstraite Salarie. Cette classe est abstraite car elle n'implémente pas la méthode verserSalaire () mais la déclare comme méthode abstraite pour obliger toute classe dérivée à la redéfinir.



Définissez ces 3 classes.

Dans le main de la classe Banque, ajouter des instructions créant plusieurs salariés et appelant leur méthodes verserSalaire.

4. Exercice sur l'héritage : « Les limites de A ''est un'' B... »

(Martin Morterol)

Instructions : lire l'énoncé en entier.

Si chaque étudiant a un PC, travaillez en binôme.

Définissez à deux la signature (l'entête) des méthodes de la classe rectangle ainsi que le comportement attendu.

Ensuite un étudiant fait la question 1), pendant que l'autre fait la question 2.

(dans un monde parfait, les tests et le code sont faits par deux personnes différentes)

1) définir une classe Rectangle ayant une longueur, une largeur ainsi que des getteurs / setteurs et une méthode de calcul de l'aire du rectangle.

2) Définir une classe TestRectangle qui vérifie que votre rectangle a bien un comportement cohérent. La classe aura autant de méthodes que vous jugerez nécessaire.

Un exemple (incomplet) de test peut être le suivant :

```
private boolean testAire(Rectangle rect)
{
        rect.setHauteur(5);
        rect.setLargeur(10);
        return (rect.calculAire()==50) ;
}

public boolean isRectangleValide()
{
        return testAire() && testQch() && .... && ... ;
}
```

3) Implémenter une classe Carré qui hérite de Rectangle. Surcharger ses méthodes afin d'assurer un comportement correct d'un carré.

4) Puisqu'un Carré est-un rectangle, vous devez pouvoir tester votre carré grâce à la classe « TestRectangle ». Que constater vous ?

5) Proposer une architecture permettant de ne plus avoir de comportement incohérent.

Commentaire : Un objet B peut/doit hériter de A si « B se comporte comme un A » ou encore si « toutes méthodes qui utilisent un objet d'une classe mère doivent pouvoir utiliser toute instance d'une classe dérivée sans avoir à le savoir ». Cela s'appelle le « principe de substitution de liskov ». C'est un principe fondamental de la programmation orientée objet. Pour plus d'information, rechercher sur Google « S.O.L.I.D »