POLYTECH ANGERS

TP 4A - Génie Logiciel
Modélisation UML à partir d'un code source

Nicolas Delanoue

Objectif : Se familiariser avec le langage de modélisation UML d'un programme, dans le cas de diagrammes de classes.

Dans un premier temps, vous devez vous familiariser avec le code suivant :

```
1  using System;
2  namespace FormesGeometriquesCSSujet
3  {
4   public class Init
5   {
6      static void Main (string[] args)
7      {
8        Dessin monDessin= new Dessin();
9        monDessin.print_informations();
10     }
11  }
12 }
```

Listing 1: Classe Init

```
using System;
namespace FormesGeometriquesCSSujet
{
public class Dessin
{
        private Rectangle m_rectangle;
        private Cercle m_cercle;

        public Dessin()
        {
                m_rectangle=new Rectangle(2,5);
                m_cercle=new Cercle(10,1,2);
        }
        public void print_informations()
        {
                this.print_surfaces();
                this.print_distances();
        }
        protected void print_surfaces ()
        {
                Console.WriteLine("Surfaces");
                float s1=m_rectangle.surface();
                Console.WriteLine("--> {0}",s1);
                float s2=m_cercle.aire();
                Console.WriteLine("--> {0}",s2);
        }
        protected void print_distances()
        {
                float x_r=m_rectangle.x();
                float y_r=m_rectangle.y();
                Console.WriteLine("Distances");
                double d1=Math.Sqrt((double)(x_r*x_r+y_r*y_r));
                Console.WriteLine("--> {0}",d1);
                float x_c=m_cercle.x();
                float y_c=m_cercle.y();
                double d2=Math.Sqrt((double)(x_c*x_c+y_c*y_c));
                Console.WriteLine("--> {0}",d2);
        }
}
}
```

Listing 2: Classe Dessin

```
using System;
namespace FormesGeometriquesCSSujet
{
public class Cercle
{   private float m_rayon;
    private float m_x;
    private float m_y;

    public Cercle (float rayon, float x=0,float y=0)
        { m_rayon=rayon;
          m_x=x; m_y=y;
        }
    public float aire () {return (float)( 3.14) *m_rayon*m_rayon;}
    public float x() {return m_x;}
    public float y() {return m_y;}
}}
```

Listing 3: Classe Cercle

```
using System;
namespace FormesGeometriquesCSSujet
{
public class Rectangle
{
  private float m_largeur;
  private float m_hauteur;
  private float m_x;
  private float m_y;

  public Rectangle (float largeur,float hauteur, float x=0,float y=0)
  {
   m_largeur=largeur; m_hauteur=hauteur;
   m_x=x; m_y=y;
  }

  public float surface () {return m_largeur*m_hauteur;}
  public float x() {return m_x;}
  public float y() {return m_y;}
}}
```

Listing 4: Classe Rectangle

Exercice 1

1. Proposer un diagramme de classes représentant les entités considérées dans le programme. On supposera qu'un cercle ou un rectangle ne peut pas être instancié autrement que par un dessin (i.e. cercle/rectangle sera systématiquement associé à un dessin).

2. Intégrer les deux packages associées (celui du projet et celui associé à la ligne de code `using System`), et le lien de dépendance approprié.
3. Modélisez, au moyen d'un diagramme d'objets, l'état des objets instanciés et leurs relations, en fin d'exécution du `Init::Main`.
4. Le code et le diagramme de classe font apparaître quelques redondances (e.g. attribut `m_x`) dont on souhaite s'affranchir en exploitant le mécanisme d'héritage. Modifier le diagramme de classes afin de bénéficier de ce mécanisme : on introduira la classe `mere` appropriée et les relations d'héritage adéquates. Sachant que seules `cercle` et `rectangle` sont instanciables, on prendra garde à interdire l'instanciation de la classe `mere` introduite. Ajuster le code en considérant que les deux objets (i.e. cercle et rectangle) sont vus comme étant du type de la classe mère, stockés dans un tableau de taille 2 au niveau de la classe `Dessin`. Lors du déroulement du programme, on privilégiera l'utilisation de boucle (parcours du tableau).

Exercice 2 (Diagramme de séquences)

1. Code avant *refactorying* : Proposer un diagramme de séquence modélisant le déroulement du `main` (le `main` étant modélisé par une classe `Main`) : création des objets, récupération et affichage des surfaces, récupération des coordonnées et affichage des distances au centre.
2. Code après *refactorying* : Proposer un nouveau diagramme de séquence, tirant partie de la modification du code après intégration du mécanisme d'héritage.