**Université de Nice - Sophia Antipolis**
**Faculté des Sciences**

**DEUG MIAS MP1**

**Programmation 2000-01**

# 5. PROGRAMMATION OBJET & POLYMORPHISME

## A. Héritage et surcharge

Nous allons utiliser ce que nous avons vu jusqu'à maintenant sur les objets pour modéliser une famille d'objets mathématiques: les côniques. Le digramme des classes ci-contre montre comment l'héritage peut être utilisé pour décrire les liens entre ces types d'objets mathématiques. La catégorisation en classes basée sur leurs similarités (taxonomie) remonte à Aristote qui, il y a plus de 2300 ans, catégorisait les espèces biologiques sous la forme d'un arbre en partant du plus général (ex: végétal, minéral, animal) et en raffinant progressivement (ex: mammifère, carnivore) pour arriver au plus précis (ex: être humain, labrador, rouge-gorge). De la même façon on peut dire que la parabole, l'hyperbole, l'ellipse sont des côniques, et que le cercle est une ellipse ayant ses deux axes de même longueur. Par défaut, c.-à-d. sans rien spécifier, une classe hérite de la classe `Object`. Cette classe `Object` contient en particulier une méthode `public String toString()` qui est donc héritée par toutes les classes décrites et permet de récupérer quelle que soit la classe de l'objet une chaîne de caractères le décrivant.

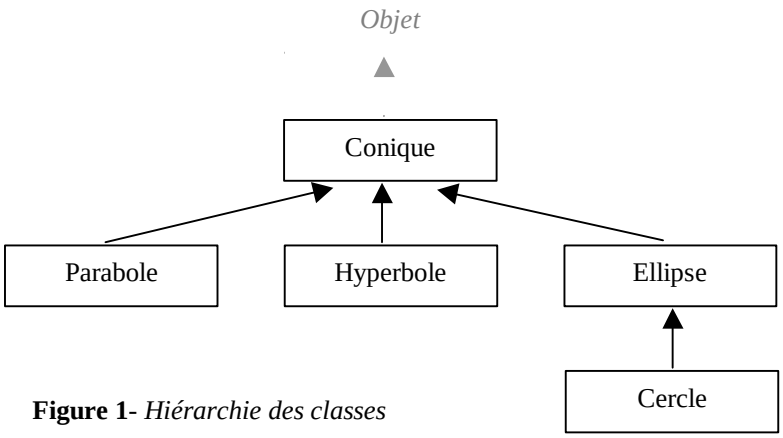


**Figure 1**- *Hiérarchie des classes*

**Exercice 5.1** Programmez les différentes classes permettant de décrire la hiérarchie de la Figure 1. Pour chaque classe, vous programmerez pour l'instant uniquement la méthode `public String toString()` qui renverra une chaîne donnant le type précis de l'objet ex : "Je suis une parabole." Dans le programme principal de test, vous créerez un objet de chaque classe et l'afficherez avec un `System.out.println(...)`

Chacune des classes précédentes hérite et surcharge la méthode `public String toString()` pour la personnaliser. On peut cependant appeler la méthode telle qu'elle était décrite dans la classe mère en utilisant le mot clef `super` qui représente la classe dont on hérite. Ainsi `super.toString()` représente la méthode de la classe-mère et renvoie la chaîne générée par celle-ci.

**Exercice 5.2** Modifiez les méthodes `public String toString()` pour qu'elles ajoutent à la fin de leur chaîne, la chaîne de caractères renvoyée par la classe-mère. Relancez l'exécution et constatez la réalité de l'héritage.

## B. Constructeurs et méthodes

**Exercice 5.3** Modifiez la classe `Ellipse` pour lui ajouter les constructeurs et les méthodes suivants :

```
Ellipse(double petitAxe, double grandAxe) {...}
Ellipse() {...}
double getPetitAxe() { ... }
double getGrandAxe() { ... }
void setPetitAxe(double petitAxe) { ... }
void setGrandAxe(double grandAxe) { ... }
```

Ajoutez la longueur des axes dans la méthodes `public String toString()` et relancez l'exécution en utilisant le constructeur avec paramètres pour créer l'ellipse.

**Exercice 5.4** a) Modifiez la classe `Cercle` pour lui ajouter les constructeurs suivants :

```
Cercle(double diametre) {...}
Cercle() {...}
```

b) Surcharger les méthodes suivantes pour vous assurer que les axes sont toujours de même longueur:

```
void setPetitAxe(double diametre) {...}
void setGrandAxe(double diametre)  {...}
```

c) Ajoutez les méthodes suvantes:

```
double getDiametre() {...}
void setDiametre(double diametre) {...}
```

d) Ajoutez la longueur du diamètre dans la méthodes `public String toString()` et relancez l'exécution en utilisant le constructeur avec un paramètre pour créer le cercle.

## C. Variables de classe

• Nous savons que le mot-clef `static` permet de déclarer une méthode ou une variable de classe. Si le mot-clef n'apparaît pas, la variable ou la méthode sont dites *d'instance* c'est à dire qu'il y a une variable par objet et que la méthode s'appelle et s'applique à un objet. Par contre si le mot-clef `static` est utilisé alors la variable est la même et est *partagée par tous les objets* (si sa valeur change, elle change pour tous les objets). Une méthode statique quant à elle s'appelle via la classe comme nous l'avons fait dans la classe `Numerik`.

**Exercice 5.5** Modifiez la classe `Conique` pour comptabiliser le nombre de côniques crées et pour que chaque cônique connaisse son numéro. Ajoutez une méthode permet d'accéder au nombre de coniques créées et une autre permettant d'obtenir le numéro d'une conique. La méthode `public String toString()` affichera en plus le numéro de la cônique. N'oubliez pas de faire appel au constructeur de la classe supérieure dans chacune de vos classes.

## D. Le polymorphisme

• Le polymorphisme, c'est lorsqu'un même objet peut prendre plusieurs formes. Dans notre cas, par héritage, les paraboles, les hyperboles, les ellipses et les cercles sont tous des objets de la classe `Conique`. On peut par conséquent les manipuler en tant que tel et appeler les méthodes qu'ils ont en commun du fait de l'héritage. En Java il existe des objets permettant de manipuler des ensembles d'objets, en particulier, nous allons importer les deux classes suivantes [voir la doc dans l'API] :

```
import java.util.HashSet;   // classe pour manipuler un ensemble d'objets
import java.util.Iterator;  // classe pour parcourir un ensemble d'objets
```

Nous pouvons alors écrire dans la méthode principale:

```
HashSet l_EnsembleDeConiques = new HashSet();
l_EnsembleDeConiques.add(new Conique());
l_EnsembleDeConiques.add(new Conique());
l_EnsembleDeConiques.add(new Hyperbole());
l_EnsembleDeConiques.add(new Parabole());
l_EnsembleDeConiques.add(new Ellipse(5.1,26));
l_EnsembleDeConiques.add(new Cercle(6));
l_EnsembleDeConiques.add(new Cercle(8));

Iterator l_IterateurSurLesConiques = l_EnsembleDeConiques.iterator();
while(l_IterateurSurLesConiques.hasNext())
  {
    System.out.println(l_IterateurSurLesConiques.next());
  }
```

La méthode `public String toString()` étant héritée de la classe `Object` elle est commune à tous les objets et nous pouvons l'appeler ici directement: Le `System.out.println(...)` attend une `String` et appelle donc la méthode `toString()`. Remarquez à l'exécution que le système appelle bien la méthode `toString()` correspondant à la classe de l'objet. L'objet représentant l'ensemble [`HashSet`] ne respecte pas l'ordre donné, c'est un ensemble non ordonné. L'objet `Iterator` permet de parcourir l'ensemble des éléments, notez la condition d'arrêt `hasNest()` et la façon d'accéder à l'objet suivant de l'ensemble `next()`.

**Exercice 5.6** En vous inspirant du code ci-dessus et en n'oubliant pas d'utiliser l'opérateur de conversion de type, parcourez l'ensemble pour n'afficher que le numéro des coniques.

• Enfin il existe en Java un opérateur permettant de comparer la classe d'un objet : `instanceof`. On peut ainsi écrire un test de la forme:

```
if (un_object instanceof  une_classe) {...}
else {...}
```

Cela permet en particulier de parcourir un ensemble polymorphe pour n'appeler une méthode que sur les objets pour lesquels cette méthode est définie.

**Exercice 5.7** En vous inspirant des deux exercices précédents, parcourez l'ensemble pour n'afficher que les cercles avec leur numéro de cônique et leur diamètre.