D[i] | Département Informatique

UNIVERSITÉ TOULOUSE III PAUL SABATIER — Université de Toulouse

Faculté Sciences et Ingénierie

# Programmation Orientée Objet

*Mathieu RAYNAL*

*mathieu.raynal@irit.fr*

*http://www.irit.fr/~Mathieu.Raynal*

D[i] | Département Informatique

UNIVERSITÉ TOULOUSE III PAUL SABATIER
Université de Toulouse
Faculté Sciences et Ingénierie

# Programmation Orientée Objet

***Héritage***

*Classe abstraite*

*Interface*

# Héritage

- Une ellipse a les mêmes caractéristiques qu'un rectangle
  - Un centre, une longueur et une hauteur
- On retrouve également les mêmes fonctionnalités entre les 2
  - Calculer la surface
  - Déplacer
- Certaines méthodes fonctionnent de la même manière (déplacer) et d'autres ont un fonctionnement différent (calculer surface)

→ La classe Ellipse peut hériter de la classe Rectangle

# Héritage

- Une classe qui hérite d'une autre aura (*a minima*)
  - Les mêmes attributs
  - Les mêmes méthodes

  → Il n'est pas nécessaire de les définir dans la classe. Elle en **hérite** automatiquement

- Elle peut avoir des attributs et/ou méthodes supplémentaires qui lui seront propre

- Une classe peut redéfinir une méthode déjà présente dans la classe dont elle hérite → on parle de **masquage**

# Définir l'héritage

- Définir qu'une classe B hérite d'une classe A

```
public class B extends A
```

- **Attention :** Une classe ne peut hériter que d'une seule classe.

# Exemple

```
public class Rectangle{
  Point centre;
  double longueur, hauteur;

  public Rectangle() {
    centre = new Point();
    longueur = 10;
    hauteur = 20;
  }

  public double calculerSurface() {
    return longueur * hauteur;
  }

  public void deplacer(int dx, int dy) {
    centre.deplacer(dx, dy);
  }
}
```

```
public class Ellipse extends Rectangle{
  public double calculerSurface() {
    return Math.PI * longueur/2 * hauteur/2;
  }
}
```

- On définit dans Ellipse que ce qui diffère de Rectangle
- Le reste est utilisable directement par Ellipse

```
public static void main(String[] args){
  Ellipse e = new Ellipse();
  System.out.println(e.hauteur);
  System.out.println(e.calculerSurface());
}
```

# Le masquage de méthode

- Une classe peut redéfinir les attributs et méthodes de la classe dont elle hérite
- Un attribut peut être masqué
  - Même nom d'attribut, avec un type différent
- Une méthode qui en masque une autre doit avoir une définition au moins aussi précise que celle qu'elle masque
  - Même profil (i.e. même nombre de paramètre et mêmes types de paramètres) → sinon c'est de la surcharge
  - Même type de retour
  - Levée d'un sous-type des exceptions levées par celle qu'elle masque
  - **Attention :** Les méthodes static ou privées ne peuvent être masquées

# Le mot clé **super**

- L'accès aux membres hérités masqués peut se faire grâce au mot clé **super**

```
public class Rectangle{
   Point centre;
   double longueur, hauteur;

   …

   public void afficherLongueur(){
      System.out.println(longueur);
   }

   …
}
```

```
public class Ellipse extends Rectangle{
   double longueur;

   …

   public void afficherLongueur(){
      super.afficherLongueur();
      System.out.println(longueur);
      System.out.println(super.longueur);
   }

   …
}
```

# Utilisation de **super()**

- **super**() fait appel au constructeur de la classe dont on hérite

```
public class Rectangle{
  Point centre;
  double longueur, hauteur;

  public Rectangle() {
    centre = new Point();
    longueur = 10;
    hauteur = 20;
  }

  public Rectangle(int l, int h) {
    centre = new Point();
    longueur = l;
    hauteur = h;
  }
  …
```

```
public class Ellipse extends Rectangle{
  public Ellipse() {
    super();
  }

  public Ellipse(int lE, int hE) {
    super(lE, hE);
  }

  …
```

# Les références

- Une référence de type A peut contenir des instances de la classe A et de toute classe dérivant de A

  → Dans ce cas, on ne pourra utiliser que les membres définis dans A

```
public class Rectangle{
  …

  public double calculerSurface() {
    return longueur * hauteur;
  }

  public void deplacer(int dx, int dy) {
    centre.deplacer(dx, dy);
  }
}
```

```
public class Ellipse extends Rectangle{
  …
  public double calculerSurface() {
    return Math.PI * longueur/2 * hauteur/2;
  }

  public double calculerPerimetre() {
    return 2*Math.PI
          *Math.sqrt((Math.pow(longueur/2, 2)
          +Math.pow(hauteur/2, 2))/2);
  }
}
```

# Les références

```
public static void main(String[] args){
  Ellipse e = new Ellipse();
  e.deplacer(50, 30);
  System.out.println("Surface de e : "+e.calculerSurface());
  System.out.println("Perimetre de e : "+e.calculerPerimetre());

  Rectangle e2 = new Ellipse();
  e2.deplacer(50, 30);
  System.out.println("Surface de e2 : "+e2.calculerSurface());
  System.out.println("Perimetre de e2 : "+e2.calculerPerimetre());

  Rectangle r = new Rectangle();
  r.deplacer(50, 30);
  System.out.println("Surface de r : "+r.calculerSurface());
}
```

calculerPerimetre() est une méthode de la classe **Ellipse**.
→ Donc pas utilisable sur une reference de type **Rectangle**

# La classe Object

- Toute classe hérite implicitement de la classe Object

- Cette classe a des méthodes par défaut que toute classe peut utiliser
  - Object clone()
  - boolean equals(Object)
  - String toString()

# Exercice 1

- Créer une classe Ellipse qui hérite de Rectangle
  - Masquer les méthodes qui ont besoin de l'être
- Ajouter la méthode toString() aux classes Point, Rectangle, Ellipse
- Rappels :
  - Surface d'une ellipse $= \pi \times longueur \times hauteur$
  - Périmètre d'une ellipse $= 2\pi \sqrt{\frac{longueur^2 + hauteur^2}{2}}$

D[i] | Département Informatique

UNIVERSITÉ TOULOUSE III PAUL SABATIER
Université de Toulouse
Faculté Sciences et Ingénierie

# Programmation Orientée Objet

*Héritage*

***Classe abstraite***

*Interface*

# Classes abstraites

- Une classe abstraite est une classe ayant au moins une méthode abstraite
- Une classe abstraite ne peut pas être instanciée
- Les constructeurs et les méthodes statiques ne peuvent pas être abstraits
- Une classe dérivée d'une classe abstraite ne masquant pas toutes les méthodes abstraites doit elle-même être abstraite.

# Exercice 2

- Créer une classe abstraite Forme
  - Attributs centreX, centreY, périmètre, nom
  - Méthodes
    - Getters / setters
    - calculerPerimetre
    - déplacer
- Rectangle hérite de Forme
- Créer la classe Triangle qui hérite également de Forme

D[i] | Département Informatique

UNIVERSITÉ TOULOUSE III PAUL SABATIER
Université de Toulouse
Faculté Sciences et Ingénierie

# Programmation Orientée Objet

*Héritage*

*Classe abstraite*

***Interface***

# Définition d'une interface

- La définition d'une interface correspond à une définition de classe complètement abstraite

```
public interface MonInterface{
   public void m();
}
```

- Définition d'une interface
  - On déclare seulement des méthodes
  - On peut définir des constantes (final static)

- Une interface peut hériter de plusieurs interfaces

- Par défaut une interface n'est visible que dans un même package. Elle peut être déclarée public.

- Une seule par fichier

# Implémenter des interfaces

```
public class MaClasse implements MonInterfaceA, MonInterfaceB …{
```

- Obligation : Avoir dans la classe toutes les méthodes définies dans les interfaces qu'implémente la classe
- Une référence de type interface peut contenir tout objet dont la classe implémente cette interface
- Une classe peut implémenter plusieurs interfaces

# Exercice 3

- Créer l'interface Dessinable qui contient les méthodes
  - afficher()
  - deplacer(int dX, int dY)

- Les classes Point et Forme implémentent Dessinable