

---

# IJA – TD 5

## L'héritage en Java

# L'héritage - Objectifs

---

- Organiser les classes dans une hiérarchie de fonctionnement
- Les classes présentent dans ces relations d'héritage un rapport parent / fils
- La relation d'héritage représente une relation sémantique non standard entre le père et le fils
- Il n'existe pas de relation d'héritage universelle entre les classes. C'est le rôle de l'architecte d'application de définir la relation qu'il sous-entend

# L'héritage - Syntaxe

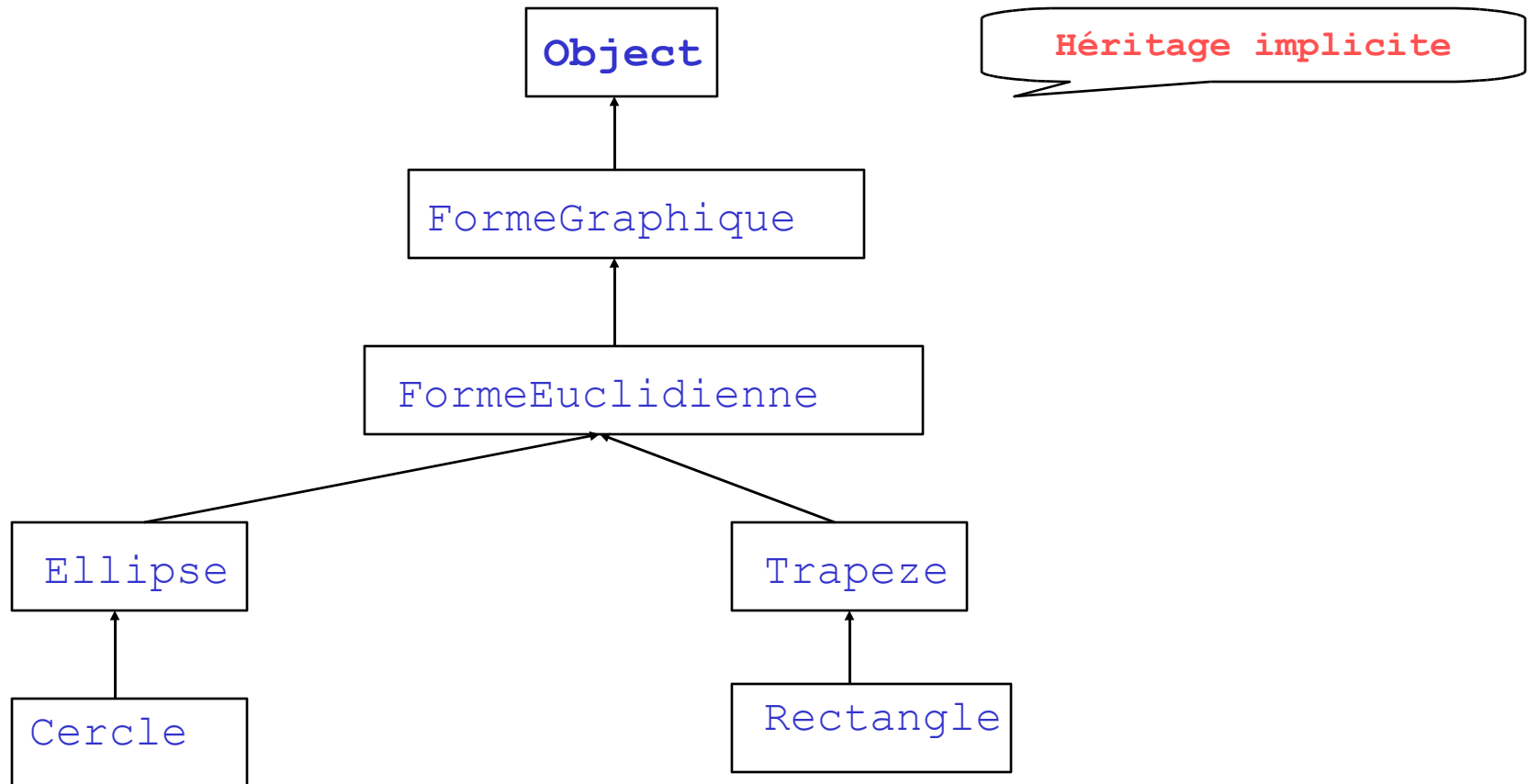
---

```
public class Cercle extends FormeGeometrique {  
...  
}
```

- La classe parente présente généralement soit des attributs et méthodes généraux à toutes les classes filles, soit des attributs et méthodes types qui doivent être (re)définie dans dans les classes filles

# L'héritage - Hiérarchie

- La relation d'héritage indique ce que l'objet **est**.



# L'héritage - Sous-type

---

- Une sous-classe étend les capacités de sa super classe. Elle hérite des capacités de sa parente et y ajoute les siennes
- De plus, une sous-classe est une spécialisation de sa super-classe. Toute instance de la sous-classe est une instance de la super-classe (pas nécessairement l'inverse).

# L'héritage – Sous-type

---

- En Java, une classe ne peut hériter (**extends**) que d'une seule classe
- Les classes dérivent, par défaut, de **java.lang.Object**
- l'héritage est transitif (« transmis » ) (si B hérite de A et C hérite de B => C hérite de A via B)
- Une référence sur une classe C peut contenir des instances de C ou des classes dérivées de C.

# L'héritage – Sous-type

---

- L'opérateur **instanceOf** permet de déterminer la classe « réelle » d'une instance.
- Les classes **final** (et respectivement les attributs et les méthodes) ne peuvent pas être redéfinies dans des sous-classes.
- Les classes **abstract** (et respectivement les méthodes) doivent être définies dans des sous-classes.
- **super** et **this** pour accéder aux membres d'une classe

# L'héritage - Exemple

---

```
public abstract class FormeEuclidienne {
    public abstract double area();
}

public class Ellipse extends FormeEuclidienne {
    public double r1, r2;
    public Ellipse(double r1, double r2) { this.r1 =
        r1; this.r2 = r2; }
    public double area(){...}
}

final class Cercle extends Ellipse {
    public Cercle(double r) { super(r, r); }
    public double getRadius() { return r1; }
}
```



# Exo

---

- Implanter la hiérarchie de la diapo 4
  - Surcharger la méthode `toString()` pour afficher les attributs propres à l'objet courant et les attributs de la classe père (utilisation de `super()`)
  - Ex : `Cercle c.toString()`  
(FormeGraphique) (FormeEuclidienne)  
(Eclipse x:3 y:3) (Cercle r:3)

# Exo

---

- Implanter un client qui instancie plusieurs classes de `FormeGraphique` et qui les vérifie avec `instanceOf`
- Essayer d'assigner une référence de classe `FormeGraphique` avec un objet n'appartenant pas à la hiérarchie :
  - `FormeGraphique fg = new String (« test »)`
  - `Object o = new Cercle();`  
`System.out.println((String)o);`

Que se passe-t-il ?

---

# IJA – TD 5

## Les interfaces Java

# Les interfaces - Objectifs

---

- une interface décrit ce que **sait faire** une classe
- Spécification des comportements possibles pour une classe, en plus de son comportement de base
- Spécification formelle de classe. Elle indique les services rendus par la classe qui implante l'interface
- !!! Technique / Conceptuel

# Les interfaces - Syntaxe

---

```
interface Dessinable {  
  
    public void draw();  
  
    ...  
}
```

- L'interface définit l'ensemble des méthodes (par leur signature) devant être implémentées dans une classe réalisant cette interface

# Les interfaces - Syntaxe

---

- Une interface correspond à une classe où toutes les méthodes sont abstraites
- Une classe peut implémenter (`implements`) une ou plusieurs interfaces tout en héritant (`extends`) d'une classe
- Une interface peut hériter (`extends`) de plusieurs interfaces

# Les interfaces - Exemple

```
abstract class Forme { public abstract double perimeter(); }

interface Dessinable { public void draw(); }

class Cercle extends Forme implements Dessinable, Serializable {
    public double perimeter() { return 2 * Math.PI * r; }
    public void draw() {...}
    private void writeObject(ObjectOutputStream out) throws
        IOException {...}
}

class Rectangle extends Forme implements Dessinable, Serializable
{
    public double perimeter() { return 2 * (height + width); }
    public void draw() {...}
    private void writeObject(ObjectOutputStream out) throws
        IOException {...}
}

class Etudiant implements Serializable {
    private void writeObject(ObjectOutputStream out) throws
        IOException {...}
}
```

# Exo

---

- Intégrer l'interface `Dessinable` (avec la méthode `dessiner`) dans la hiérarchie développée dans l'exercice précédent. Que faut-il modifier dans chaque classe ?
- Taper le code suivant :

```
Dessinable d = new Cercle();  
d.dessiner();
```

Que permet la manière de programmer de type  
`Interface i = new Object()` ?