Contenu du cours

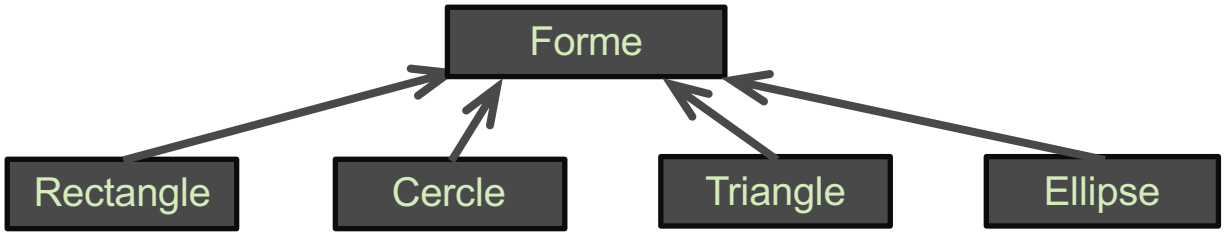
1. Introduction à la POO
2. Une **classe** : définition de nouveaux objets
3. **Instanciation** et utilisation d'objets
4. Création des objets : les **constructeurs**
5. **Références**, visibilité des variables
6. **Encapsulation** et masquage des données
7. **Statique**, ou d'instance ?
8. **Héritage**
9. **Polymorphisme**
10. **Classes abstraites** et **interfaces**
11. **Introduction aux types génériques**
12. **Exceptions** en java
13. *Compléments syntaxiques*

Classes Abstraites & Interfaces

Classes Abstraites

L'héritage pour typer des objets

- **Exemple : Formes géométriques**
  - **Déjà étudiées : Rectangle, Cercle**
  - **Autres possibilités : Triangle, Ellipse, etc.**
  - **Toutes sont des variantes du type Forme**



- **Regroupement de formes géométriques : typage par l'ancêtre commun**

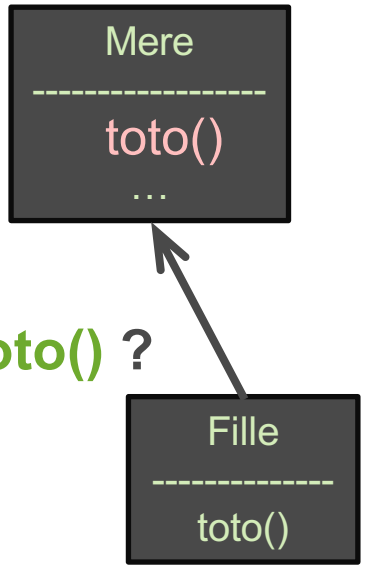
```
Forme[] tabFormes = new Formes[10];
tabFormes[0] = new Rectangle(…);
tabFormes[1] = new Cercle(…);
tabFormes[2] = new Triangle(…);
//  Etc.
```

OK grâce à l'héritage : transtypage ascendant vers la classe Forme

Polymorphisme : rappel

- **Transtypage ascendant**
  - **Une sous-classe (ex : classe Fille) est typée par sa super-classe (ex : classe Mere)**

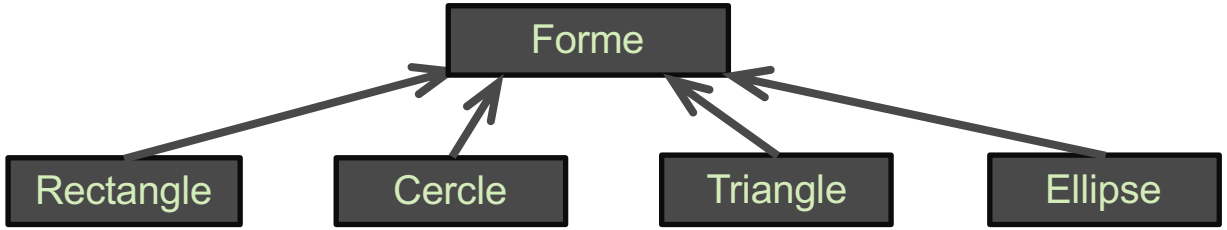
```
Mere m = new Fille();
m.toto(); // appel de la méthode toto() sur l'instance m
```



  - **Compilation: m est de type Mere**
  - **Si la classe Mere ne déclare pas de méthode toto() ?**
    - **Erreur compilation**
- **Exemple**
  - **l'appel à toString() compile toujours car toute classe hérite de Object**

Lors du transtypage ascendant, toutes les méthodes invoquées doivent être définies dès la super-classe

Exemple de méthodes abstraites




```
Forme[] tabFormes = new Formes[10];
tabFormes[0] = new Rectangle(…);
tabFormes[1] = new Cercle(…);
tabFormes[2] = new Triangle(…);
//  Etc.
```

- **Toutes les formes géométriques possèdent :**
  - **Une méthode aire()**
  - **Une méthode perimetre()**
- **Affichage de l'aire de chaque forme du tableau**

```
for (int i=0; i<tabFormes.length; i++)
      Ecran.afficher("aire ["+i+"] = " + tabFormes[i].aire();
```

Type statique (à la compilation) = Forme

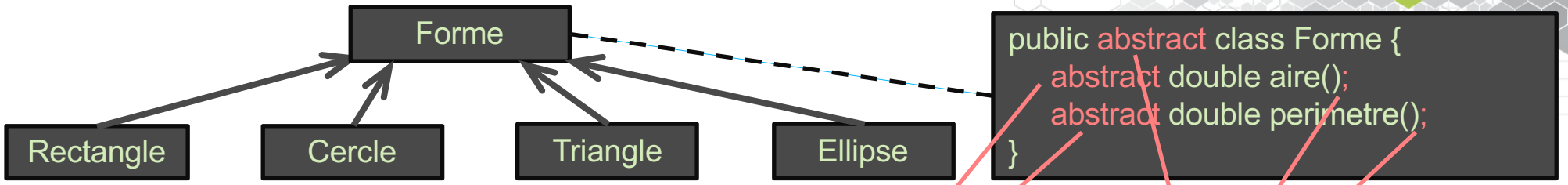
Pour compiler : aire() doit être une méthode de Forme

Question : Quel code donner à aire() dans la classe Forme ?

Réponse : Aucun ! On ne sait pas calculer une aire (ni d'ailleurs un périmètre) de manière générale : le calcul dépend de la forme.

Dans la classe Forme, aire() et perimetre() seront des **méthodes abstraites**

Classes et méthodes abstraites



- **Méthode abstraite**
  - **Déclarée avec le mot clé abstract**
  - **Pas de code : remplacé par un simple point-virgule**
- **Une classe contenant une méthode abstraite doit obligatoirement être elle-même déclarée abstract**

A retenir absolument.
- Les classes abstraites ne peuvent pas être instanciées
- Hériter (concrètement) d'une classe abstraite oblige à implanter toutes ses méthodes abstraites
- Une classe a le droit d'être déclarée abstraite même si toutes ses méthodes sont concrètes

Utile pour fournir une implantation partielle, à finaliser par la **sous-classe concrète**

Classes et méthodes abstraites : règles

Extrait de « Java in a Nutshell » par David Flanagan, 4ème édition en français (O'Reilly)

Les règles suivantes s'appliquent :

— Toute classe contenant une méthode abstraite est elle-même obligatoirement abstraite, et doit être déclarée abstraite.

— Une classe abstraite ne peut pas être instanciée.

— Une sous-classe d'une classe abstraite ne peut être instanciée que si elle redéfinit chaque méthode abstraite de sa super-classe en fournissant une implantation. Une telle classe est souvent appelée une sous-classe *concrète*.

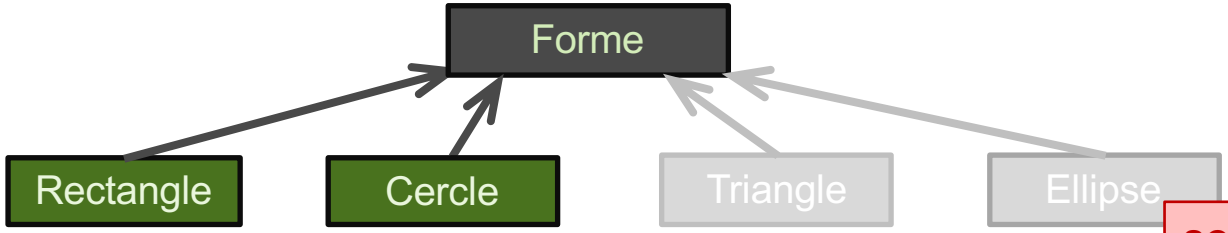
— Si une sous-classe d'une classe abstraite n'implante pas *toutes* les méthodes abstraites dont elle hérite, alors cette sous-classe est elle-même abstraite.

— Les méthodes déclarées *statiques*, *privées* ou *finales* n'ont pas le droit d'être abstraites.

— Une classe déclarée *finale* n'a pas le droit de contenir de méthode abstraite.

— Une classe peut être déclarée abstraite même si elle ne possède pas de méthode déclarée abstraite. C'est le cas quand les méthodes ne fournissent qu'une implantation partielle qui sera complétée dans chaque sous-classe.

Classe abstraite et sous-classes concrètes : un exemple



Montrer

classe abstraite

```
public abstract class Forme {
    abstract double aire();
    abstract double perimetre();
}
```

implantation

implantation

implantation

implantation

sous-classe concrète

```
public class Rectangle extends Forme {
    private double length, height;
    … // Constructeurs, etc.
    public double aire() {
        return this.length * this.height;
    }
    public double perimetre() {
        return 2*(this.length + this.height);
    }
}
```

sous-classe concrète

```
public class Cercle extends Forme {
    private double radius;
    … // Constructeurs, etc.
    public double aire() {
        return Math.PI*(this.radius*this.radius);
    }
    public double perimetre() {
        return 2*Math.PI*this.radius;
    }
}
```

programme principal

```
public static void main(String[] args) {
    Forme[] tabForme = new Forme[10];
    tabForme[0] = new Cercle(5);
    tabForme[1] = new Rectangle(3, 4);
    Ecran.aficher(tabForme[0].aire());
    Ecran.aficher(tabForme[1].aire());
}
```

78.54

12

Restrictif : une seule classe abstraite héritable

- On suppose qu'on veut des : **Formes positionnées**
  - Déjà étudiées : **Rectangle**, **Cercle** avec position dans le plan
  - La position est un **Point2D**
- Une modélisation avec héritage et classe abstraite