



# Concepts de base de la programmation orientée objet

# Plan

## 1 Qu'est-ce qu'un objet ?

- Une définition
- Principe d'encapsulation des données et des traitements
- Exemples

## 2 Qu'est-ce qu'une classe ?

- Une définition
- Exemple de classe en Java

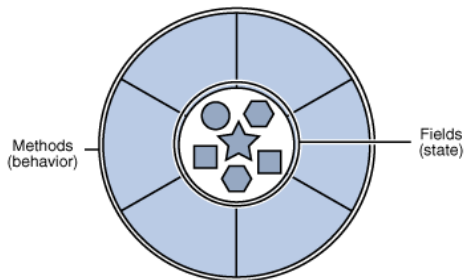
## 3 Concepts clés de la POO

- Héritage
- Spécialisation
- Redéfinition
- Interface

# Qu'est-ce qu'un objet ?

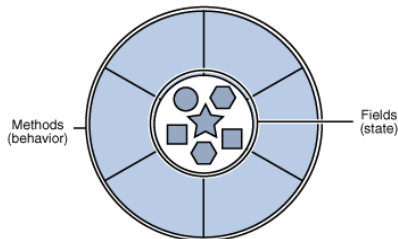
- Une entité logicielle cohérente définie par son **état** et son **comportement**.
- Utilisé pour modéliser informatiquement des "objets" de la vie courante (voiture, personne, etc.) ou des concepts (date, couleur, etc.).

## Qu'est-ce qu'un objet ?



- Son **état** est défini à l'aide d'**attributs** (Fields) : des variables associées à l'objet.
- Son **comportement** est défini par des **méthodes** (Methods) : une fonction rattachée à l'objet qui permet de déclencher un des comportements associés à l'objet.

# Principe d'encapsulation

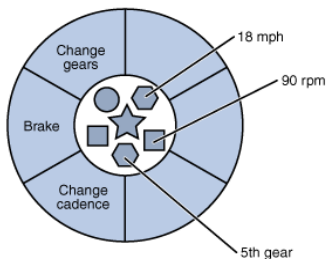


## Encapsulation des données et des traitements

- Les attributs sont généralement inaccessibles depuis l'extérieur de l'objet : seules certaines méthodes sont accessibles.
- Cacher le détail de la structure interne d'un objet correspond à appliquer le principe d'**encapsulation** : principe de base de la programmation objet.

## Un exemple

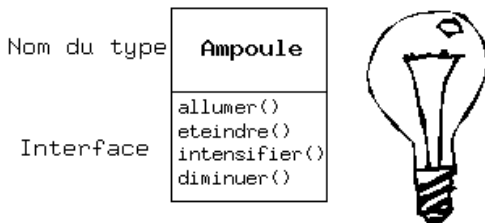
Un vélo :



- Les méthodes permettent de contrôler la manière dont les attributs de l'objet vélo sont modifiés.
- Avantages : modularité, réutilisation, indépendance des différentes parties du programme (facilite la maintenance et le debug).

## Autre exemple, avec la notation UML

Une ampoule :



- L'ensemble des méthodes d'un objet est appelé l'**interface de l'objet**.
- Une interface définit donc en quelque sorte tout ce qu'il est possible de "faire" avec un objet.

## Qu'est-ce qu'une classe ?

- Dans le monde réel, nous comprenons les objets en les associant à des catégories :
  - tous les vélos possèdent des **caractéristiques communes**.
  - peu importe leurs états individuels, ils sont tous faits des **mêmes composants** (vitesses, pédales, etc.)
  - Ils définissent donc une **classe (ou type) d'objet particulière**.
- En termes "orientés objets" :
  - un vélo particulier est simplement une **instance** de la **classe** vélo.
  - les attributs et les méthodes d'une instance de vélo sont les mêmes pour toutes les instances de vélos.
  - seules **les valeurs des attributs** (caractéristiques propres) sont susceptibles d'être **différentes**.
  - une **classe** est donc un **patron de conception** à partir duquel des objets individuels sont créés.



# Exemple de classe en Java

## Bicycle.java

```
public class Bicycle {  
    private int cadence = 0;  
    private int speed = 0;  
    private int gear = 1;  
  
    public void changeCadence(int newValue) {  
        cadence = newValue;  
    }  
  
    public void changeGear(int newValue) {  
        gear = newValue;  
    }  
  
    void speedUp(int increment) { // visibilité de la méthode restreinte au package  
        speed = speed + increment;  
    }  
  
    public void applyBrakes(int decrement) {  
        speed = speed - decrement;  
    }  
  
    public void printStates() {  
        System.out.println("cadence:"+cadence+" speed:"+speed+" gear:"+gear);  
    }  
}
```

## Remarques

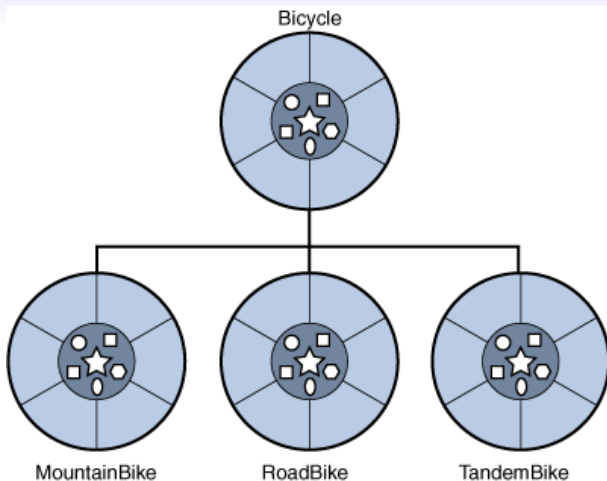
- la classe précédente ne contient pas de méthode main.
- ce n'est pas une application.
- c'est juste une classe permettant de créer des objets qui seront utilisés dans une application.

# Utilisation d'une classe dans une application Java

## BicycleDemo.java

```
public class BicycleDemo {  
  
    public static void main(String[] args) {  
  
        // Create two different Bicycle objects  
        Bicycle bike1 = new Bicycle();  
        Bicycle bike2 = new Bicycle();  
  
        // Invoke methods on those objects  
        bike1.changeCadence(50);  
        bike1.speedUp(10);  
        bike1.changeGear(2);  
        bike1.printStates();  
  
        bike2.changeCadence(50);  
        bike2.speedUp(10);  
        bike2.changeGear(2);  
        bike2.changeCadence(40);  
        bike2.speedUp(10);  
        bike2.changeGear(3);  
        bike2.printStates();  
    }  
}
```

# Héritage



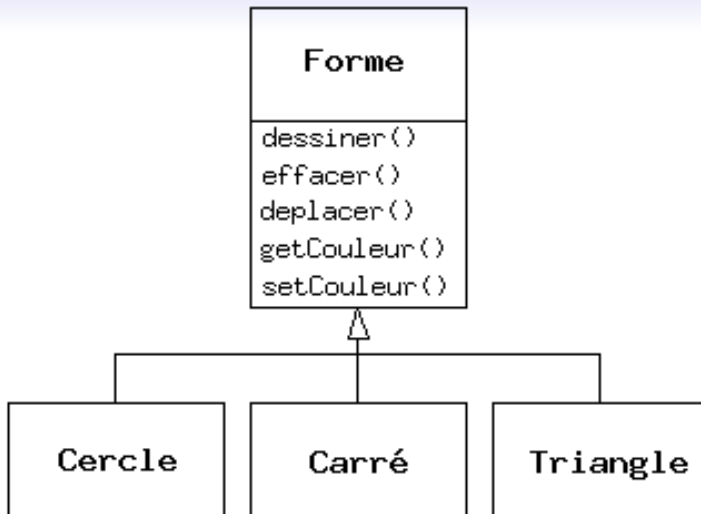
## Héritage : le mot clé *extends*

Pour hériter d'une classe, on utilise le mot clé *extends* :

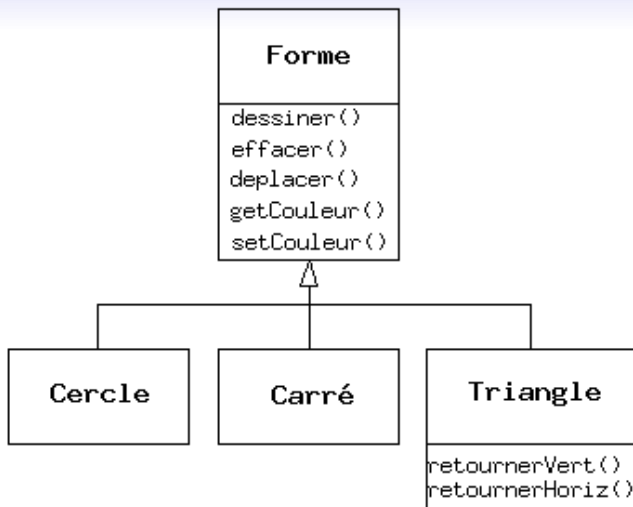
### MountainBike.java

```
public class MountainBike extends Bicycle {  
    // new fields and methods defining a mountain bike would go here  
}
```

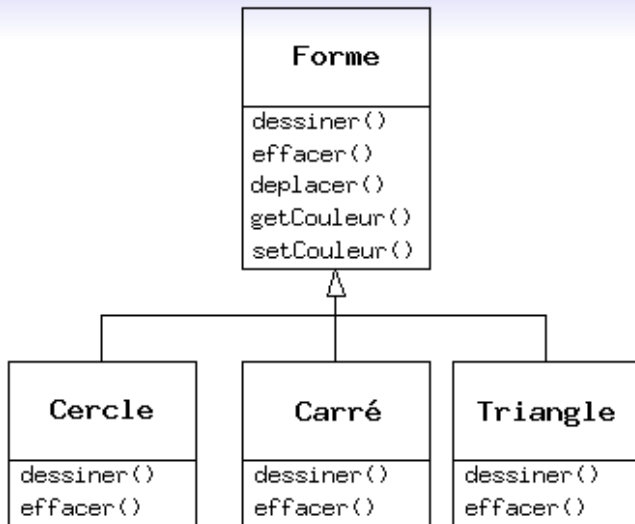
## Héritage (notation UML)



# Spécialisation



## Redéfinition de méthodes





# Interface en java

## Définition

- Les objets interagissent avec le monde extérieur grâce aux méthodes qu'ils définissent (encapsulation)
- Les méthodes d'un objet définissent donc son **interface** avec le monde extérieur
- En java, il est possible de définir, de manière abstraite (sans le code des méthodes), une interface.
- On peut ainsi définir toutes les méthodes nécessaires à un type d'objet particulier sans pour autant imposer le code des méthodes.
- On définit ainsi aussi des types qui permettent donc de **typer** les objets sans avoir recours à l'héritage.

## Interface, exemple :

### BicycleInterface.java

```
public interface BicycleInterface {  
    public void changeCadence(int newValue);  
    public void changeGear(int newValue);  
    public void speedUp(int increment);  
    public void applyBrakes(int decrement);  
}
```

## ACMEBicycle.java

```
public class ACMEBicycle implements Bicycle {  
  
    public void changeCadence(int newValue){  
        //code  
    }  
    public void changeGear(int newValue){  
        // code  
    }  
    public void speedUp(int increment){  
        // code  
    }  
    public void applyBrakes(int decrement){  
        // code  
    }  
}
```

### Une classe implémente une interface grâce au mot clé *implements*

- Lorsqu'une classe implémente une interface, toutes les méthodes de l'interface doivent être définies dans le corps de la classe.
- Il s'agit d'une sorte de contrat : "la classe s'engage à définir les méthodes de l'interface"

# Résumé global

- La programmation orientée objet (POO) :
  - la POO est un paradigme de programmation qui fait le lien avec la manière dont nous concevons le monde. [► La POO sur Wikipédia](#)
  - elle repose sur l'assemblage de briques logicielles, les *objets*, définies par un **état** et un **comportement**.
  - On parle d'**attributs** et de **méthodes**.
- Quelques principes de bases de la POO :
  - **encapsulation** : les traitements et les attributs internes à un objet ne doivent pas être visibles de l'extérieur
  - notion d'**interface** : les méthodes d'une classe d'objets. Elles définissent son comportement.
  - **héritage** et **spécialisation** : un programme correspond au final à définir et utiliser une hiérarchie d'objets.
  - la spécialisation d'une classe peut entraîner la **redéfinition** d'une partie de son comportement, c'est-à-dire de ses méthodes.

Ce cours reprend largement les tutoriaux en ligne proposés par Sun : [► The Java Tutorials](#)