



# Les méthodes d'instance

CSC 3101

Algorithmique et langage de programmation

Gaël Thomas



# Petits rappels : l'objet

- Une structure de données (tuple ou tableau) s'appelle un **objet**
- Un **objet** possède un **type**
- Le **type d'un objet** s'appelle une **classe**
- Si la classe d'un objet  $o$  est  $C$ ,  
alors on dit que  **$o$  est une instance de  $C$**
- En Java, on ne manipule que des **références** vers des objets

# Limite de la programmation impérative

Prog. impérative = celle que vous utilisez jusqu'à maintenant

Un programme est constitué de

- Structures de données
- Et de méthodes qui manipulent ces structures

Pour réutiliser une structure de données dans un autre projet

- Il faut trouver la structure de données et la copier
- Mais il faut aussi **trouver les méthodes qui manipulent la structure de données et les copier**

⇒ nécessite une structure claire du code

# Solution partielle utilisée en CI3

Faire preuve de discipline quand on programme

- Regrouper les méthodes qui manipulent une structure de données dans la classe qui définit la structure de données
- Éviter de manipuler directement une structure de données à partir de l'extérieur de la classe

Solution partiellement satisfaisante car aucune aide fournie par le langage

Méthode d'instance : aide à faire preuve de discipline

# La méthode d'instance

But : simplifier la définition des méthodes qui manipulent une structure de données


Principe : associer des méthodes aux instances

- Méthode d'instance = méthode **sans** mot clé `static`
- Méthode qui manipule une structure de données
- Associée à la classe dans laquelle la méthode est définie
- Reçoit un **paramètre caché nommé `this`** du type de l'instance
  - ⇒ pas besoin de spécifier explicitement ce paramètre
  - ⇒ simplifie le code

# Méthode d'instance vs de classe

## Avec méthode d'instance

```
class Monster {  
    int health;  
  
    void kill() {  
        this.health = 0;  
    }  
}
```



## Avec méthode de classe

```
class Monster {  
    int health;  
  
    static void kill(Monster m) {  
        m.health = 0;  
    }  
}
```

Pas de `static`

⇒ paramètre `Monster this` implicitement ajouté

# Méthode d'instance vs de classe

Avec méthode

```
class Mons
  int heal

  void kil
    this.h
  }
}
```

On dit que **this** est le **receveur** de l'appel

classe

```
Monster m) {
```

Pas de `static`

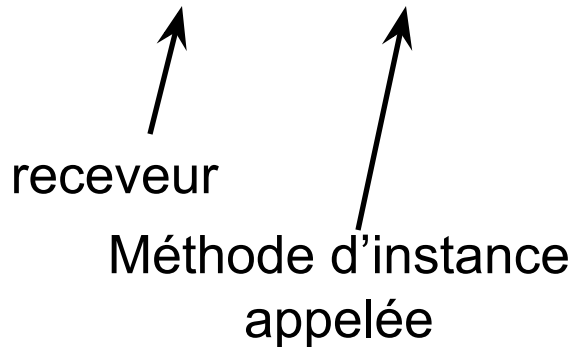
⇒ paramètre `Monster this` implicitement ajouté

# Invocation d'une méthode d'instance

Le receveur d'un appel de méthode d'instance se met à gauche

```
Monster aMonster = Monster.create(aPicture);
```

```
aMonster.kill();
```



Un peu comme si on invoquait

```
Monster.kill(aMonster);
```

(i.e., `this` reçoit la valeur `aMonster`)

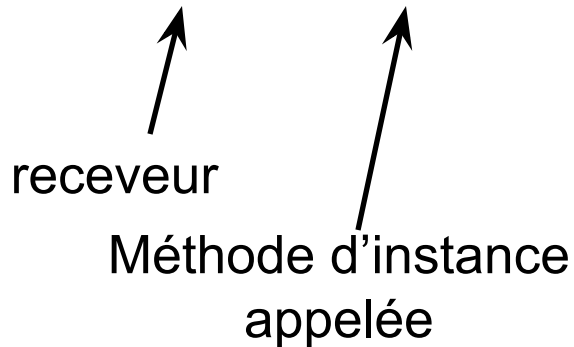


# Invocation d'une méthode d'instance

Le receveur d'un appel de méthode d'instance se met à gauche

```
Monster aMonster = Monster.create(aPicture);
```

```
aMonster.kill();
```



Remarque :

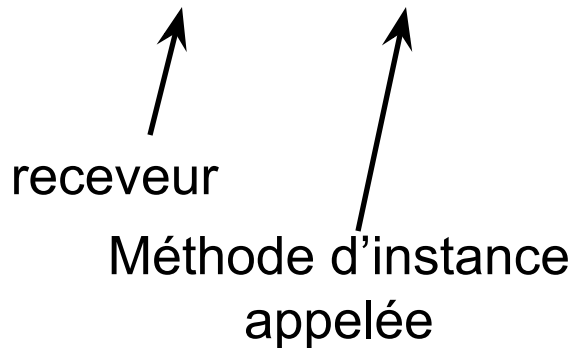
On appelle le `kill` de `Monster`  
car la classe du receveur  
(`aMonster`) est `Monster`

# Invocation d'une méthode d'instance

Le receveur d'un appel de méthode d'instance se met à gauche

```
Monster aMonster = Monster.create(aPicture);
```

```
aMonster.kill();
```



Remarque :

On appelle le `kill` de `Monster` car la classe du receveur (`aMonster`) est `Monster`

Remarque : si `aMonster` vaut `null`  
⇒ erreur de type `NullPointerException`

# Omission de `this`

```
class Monster {  
    int health;  
  
    void kill() {  
        this.health = 0;  
    }  
}
```



```
class Monster {  
    int health;  
  
    void kill() {  
        health = 0;  
    }  
}
```

En l'absence d'ambiguïté, `this` peut être omis

(`health` champ de `Monster`

⇒ remplacé par `this.health` à la compilation)

# Omission de `this`

**Attention !**

N'oubliez pas qu'il y a un receveur  
`this` caché pour les méthodes  
d'instance

En l'

(head ... par `this.health`)

# En cas d'ambiguïté, utilisez `this`

Java utilise la portée lexicale : le compilateur cherche le symbole le plus proche en suivant les blocs de code

```
class Monster {  
    int    health;  
    int    x;  
    int    y;  
  
    void move(int x, int y) { this.x = x; this.y = y; }  
}
```

`this` est nécessaire pour lever l'ambiguïté entre le champ `x` et l'argument `x`

# Notions clés

Avec `static`, la méthode s'appelle une **méthode de classe**

- Marquée par le mot clé `static`
- Uniquement les paramètres explicites

```
static void kill(Monster monster)
```

Sans `static`, la méthode s'appelle une **méthode d'instance**

- Reçoit un paramètre caché nommé `this` du type de la classe

```
void kill()
```

↔

```
static void kill(Monster this)
```

- Invocation avec receveur à **gauche** : `monster.kill()` ;

# Notions clés

Attention !

Ce cours est court, mais **essentiel**

Ne confondez pas les méthodes de classe et les méthodes d'instance !

- `monstre.gauche : monster.kill () ;`

se

se