



# INTRODUCTION POO

CLASSE – OBJET - ENCAPSULATION

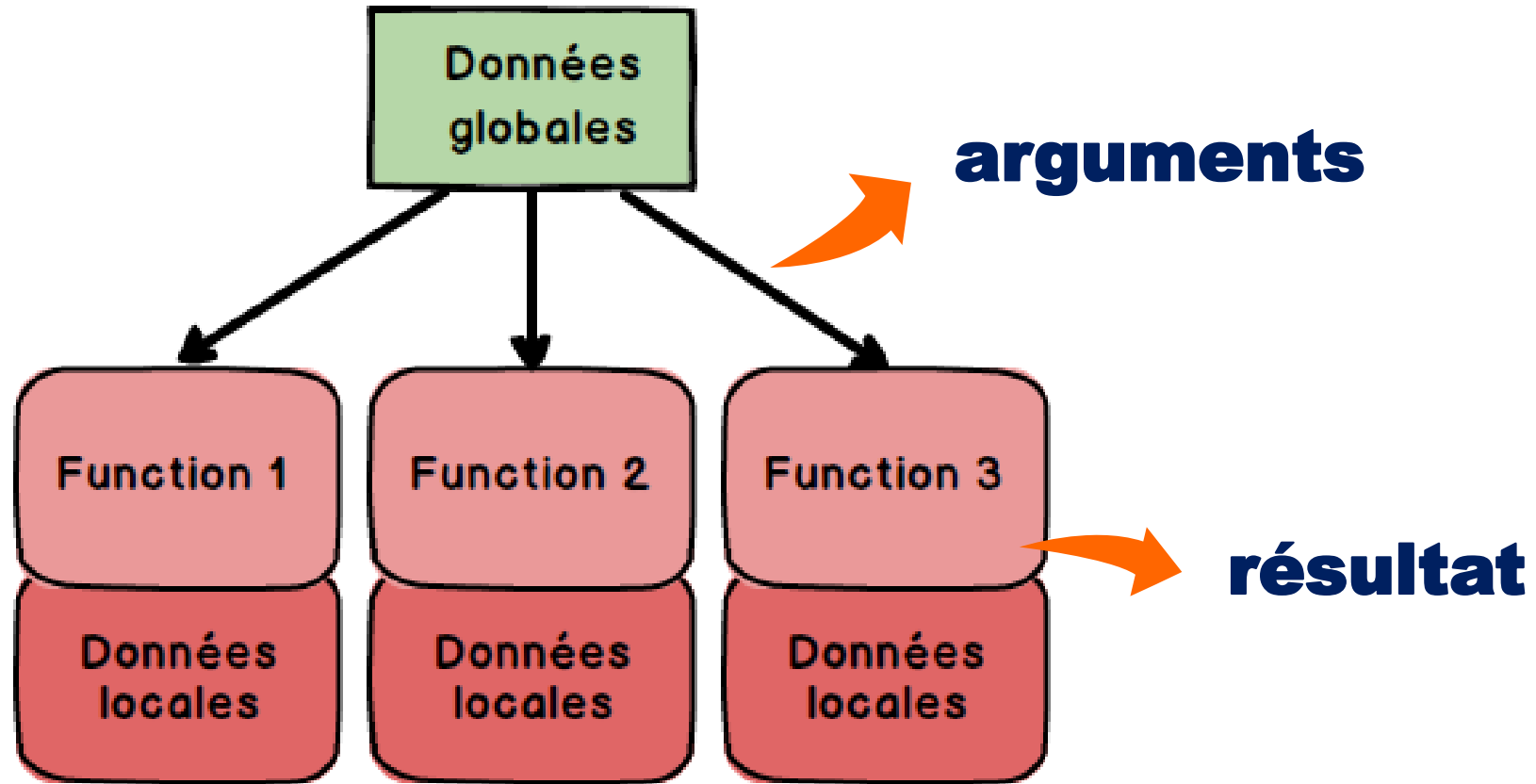
**BTS SIO1 – ANNÉE 2023/2024 – LYCEE PERGAUD**

# PROGRAMMATION PROCÉDURALE

La programmation procédurale est une méthode de programmation qui se concentre sur la création de fonctions pour effectuer des tâches spécifiques.

Ces fonctions sont des blocs de code qui prennent des entrées, effectuent des calculs et renvoient des résultats.

# PROGRAMMATION PROCÉDURALE



# PROGRAMMATION PROCÉDURALE



## Calcul de la surface d'un rectangle



### index.php

```
1 require_once "fonctions_rectangle.php";
2
3 // Déclaration d'un rectangle
4 $longueur = 12;
5 $largeur = 10;
6
7 // Calcul de la surface du rectangle
8 $surface = calculerSurfaceRectangle($longueur,$largeur);
9 echo "La surface du rectangle est égale à $surface";
```

### fonctions.php

```
1 // Fonction permettant de calculer la surface d'un rectangle
2 function calculerSurface(int $longueur, int $largeur) : int {
3     return $longueur * $largeur;
4 }
```



**ÇA  
MARCHE**

# PROGRAMMATION PROCÉDURALE

## → Lien sémantique entre données et traitements

### index.php

```
1 require_once "fonctions_rectangle.php";
2
3 // Déclaration d'un rectangle
4 $longueur = 12;
5 $largeur = 10;
6
7 // Calcul de la surface du rectangle
8 $surface = calculerSurfaceRectangle($longueur,$largeur);
9 echo "La surface du rectangle est égale à $surface";
```

**D**

### fonctions.php

```
1 // Fonction permettant de calculer la surface d'un rectangle
2 function calculerSurface(int $longueur, int $largeur) : int {
3     return $longueur * $largeur;
4 }
```

**T****D****longueur  
largeur****Lien sémantique****?****T****calculerSurface**

# PROGRAMMATION PROCÉDURALE

➔ Lien sémantique entre données et traitements

## index.php

```
1 require_once "fonctions_rectangle.php";
2
3 // Déclaration d'un rectangle
4 $longueur = 12;
5 $largeur = 10;
6
7 // Calcul de la surface du rectangle
8 $surface = calculerSurfaceRectangle($longueur,$largeur);
9 echo "La surface du rectangle est égale à $surface";
```

D

## fonctions.php

```
1 // Fonction permettant de calculer la surface d'un rectangle
2 function calculerSurface(int $longueur, int $largeur) : int {
3     return $longueur * $largeur;
4 }
```

T

D

*Lien sémantique*  
**RECTANGLE**

T

**longueur**  
**largeur**

**calculerSurface**



# REGROUPEMENT

## → Le lien sémantique



### fonctions.php

```
1 // Fonction permettant de calculer la surface d'un rectangle
2 function calculerSurface(int $longueur, int $largeur) : int {
3     return $longueur * $largeur;
4 }
```

T

### index.php

```
1 // Déclaration d'un rectangle
2 $longueur = 12;
3 $largeur = 10;
```

D

+

Regroupement  
dans une  
**même entité**

# CONCEPT / NOTION

➔ Le lien sémantique



Lien sémantique est explicitement défini !  
C'est la notion de **RECTANGLE**

## UN RECTANGLE

longueur=12  
largeur=10

D

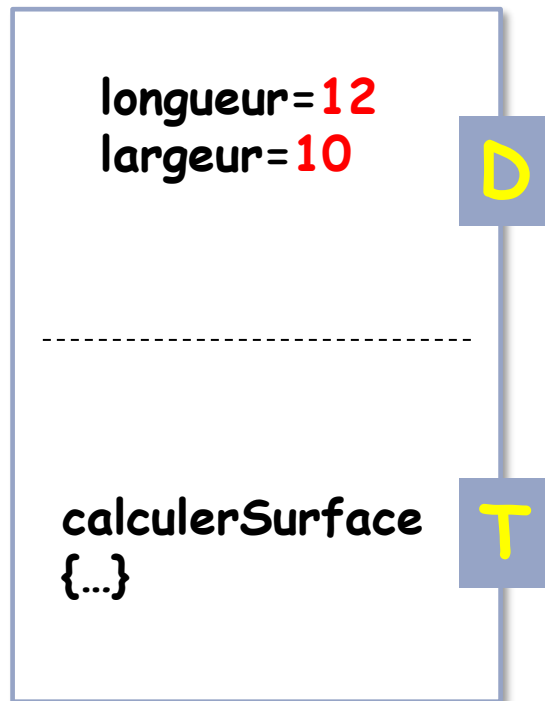
calculerSurface  
{...}

T



# NOTION D'OBJET

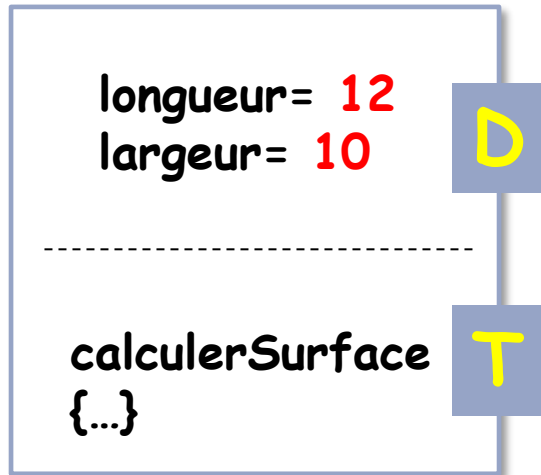
## UN RECTANGLE



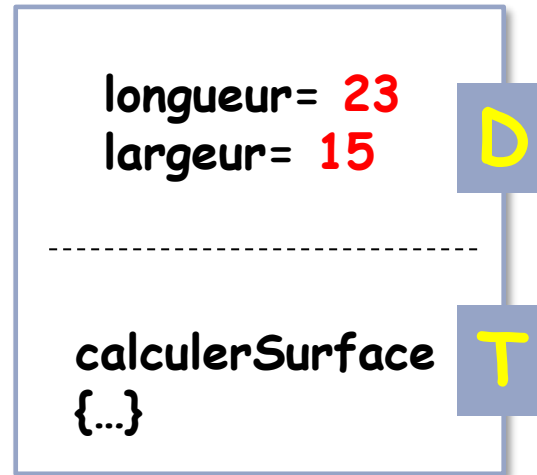
Un Objet

# NOTION DE TYPE

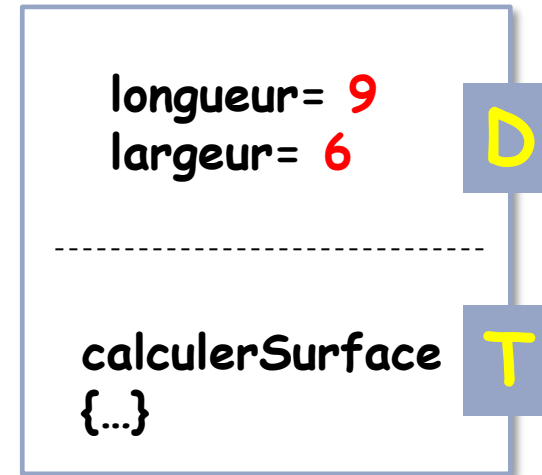
## Un objet RECTANGLE



## Un objet RECTANGLE



## Un objet RECTANGLE



**RECTANGLE** → Type de l'objet

# NOTION DE CLASSE

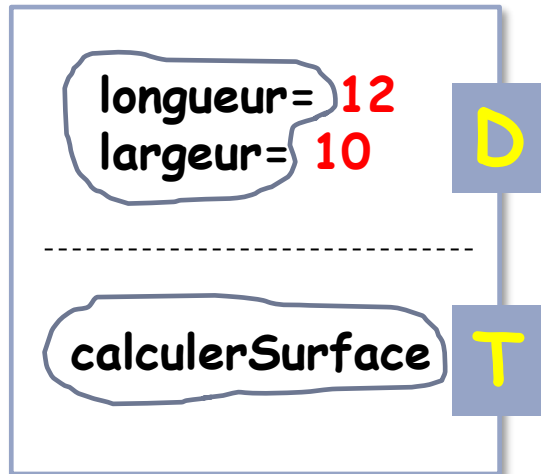
**RECTANGLE** → Type de l'objet

Type de  
l'objet → **Classe**

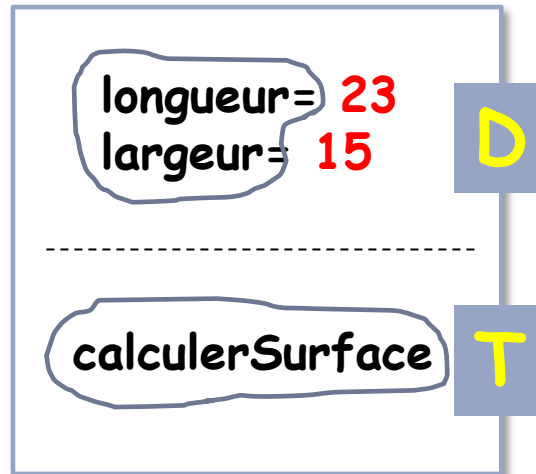
**RECTANGLE** → **Classe**

# NOTION DE TYPE

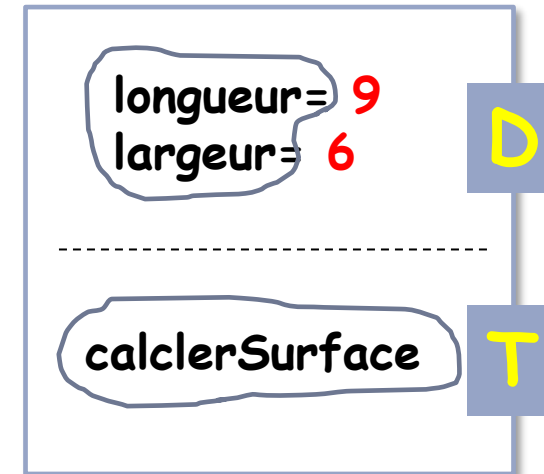
## Un objet RECTANGLE



## Un objet RECTANGLE



## Un objet RECTANGLE



Tous les objets de type **RECTANGLE** ont en commun :

- Une longueur et une largeur
- Un traitement `calculerSurface`

# NOTION DE CLASSE

→ Ces points communs sont regroupés et définis au niveau de la **classe** = le type

## Classe **Rectangle**

D

**Rectangle**

longueur  
largeur

T

calculerSurface  
{...}



La classe **Rectangle**  
sert de **modèle**

Chaque objet (un **rectangle**)  
créé aura une longueur, une  
largeur et un traitement  
calculerSurface

# PRINCIPE D'ENCAPSULATION

→ Le fait de regrouper dans une classe des données et des traitements : principe d'encapsulation



## RELATION CLASSE - OBJET

**Classe**



**Objet**



**Type**



**Variable**

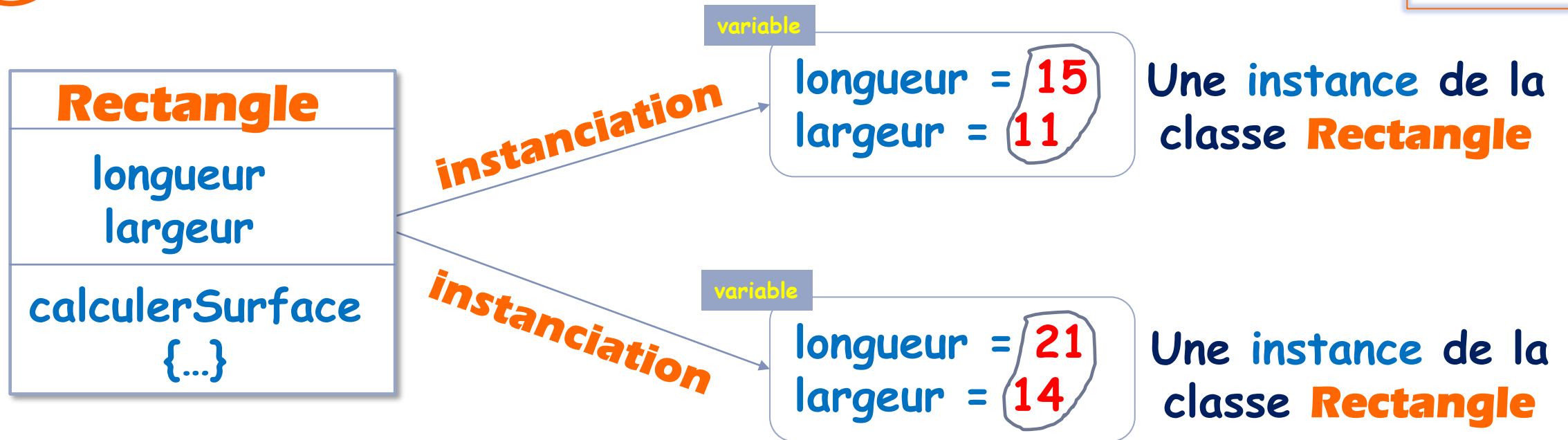
# INSTANCIATION

→ Le mécanisme permettant de créer des objets d'une **classe** : instanciation



→ Un objet est une instance d'une classe

objet  
=  
instance

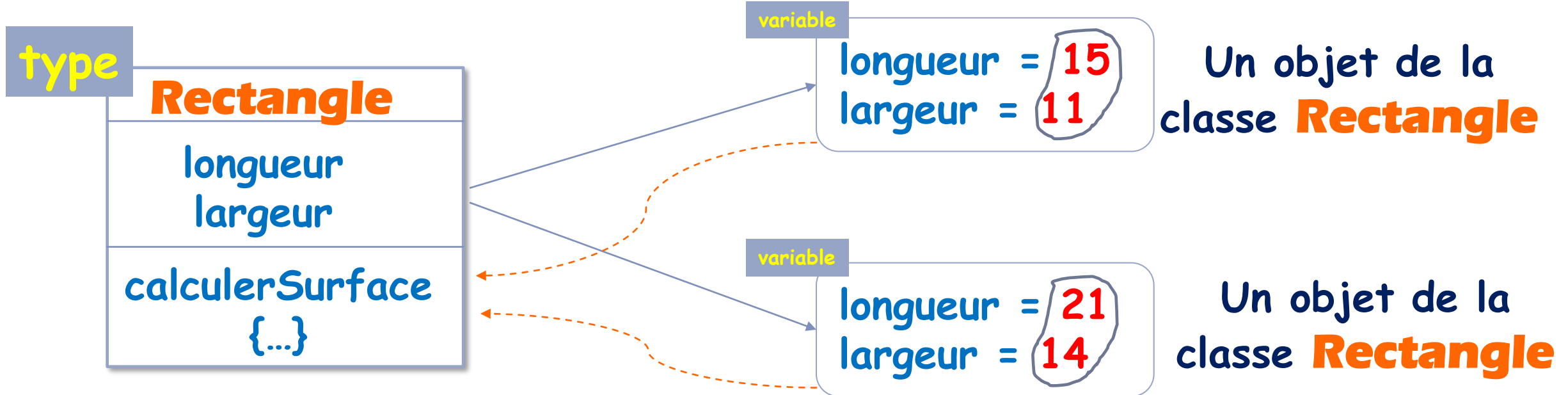




# INSTANCIATION

→ Chaque objet d'une **classe** a ses propres valeurs pour les attributs

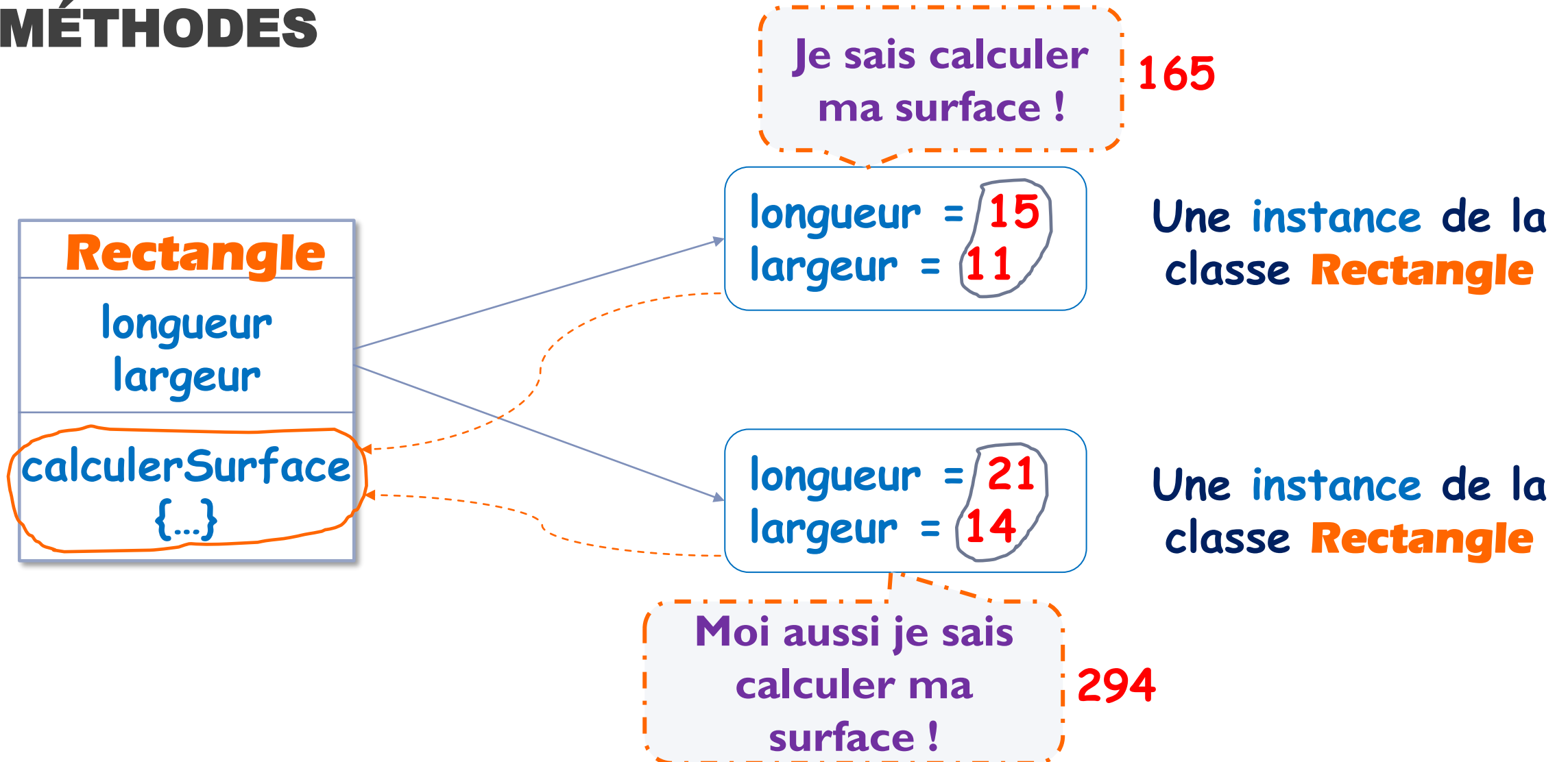
→ Chaque objet d'une **classe** partage les méthodes



# MÉTHODES

- Une méthode représente un traitement que l'on peut réaliser sur les instances (les objets) d'une **classe**
- Une **classe** peut définir plusieurs méthodes (c'est d'ailleurs souvent le cas)
- Les méthodes d'une **classe** définissent les actions que ses instances sont capables de réaliser

# MÉTHODES



# CONCEPTEUR - UTILISATEUR

- Une **classe** est implémentée\* par un développeur/concepteur
- Une **classe** est utilisée par un développeur/utilisateur



Je suis le  
concepteur de la  
classe **Rectangle**

\*



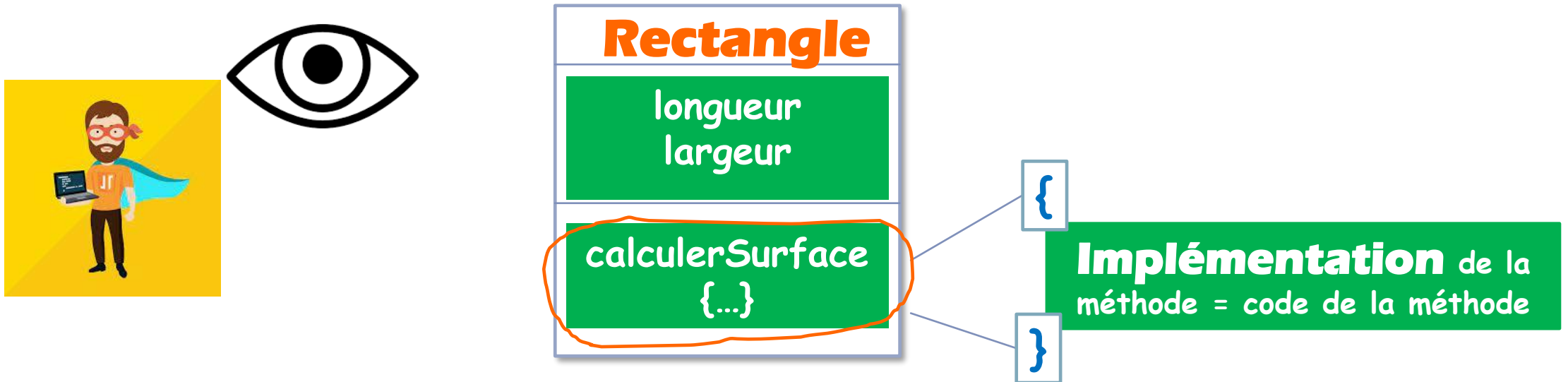
Je suis un  
utilisateur de la  
classe **Rectangle**



## Implémenter = Coder

# CONCEPTEUR

→ Le concepteur d'une **classe** a une visibilité totale sur le code de **classe** : il fait ce qu'il veut !



# UTILISATEUR

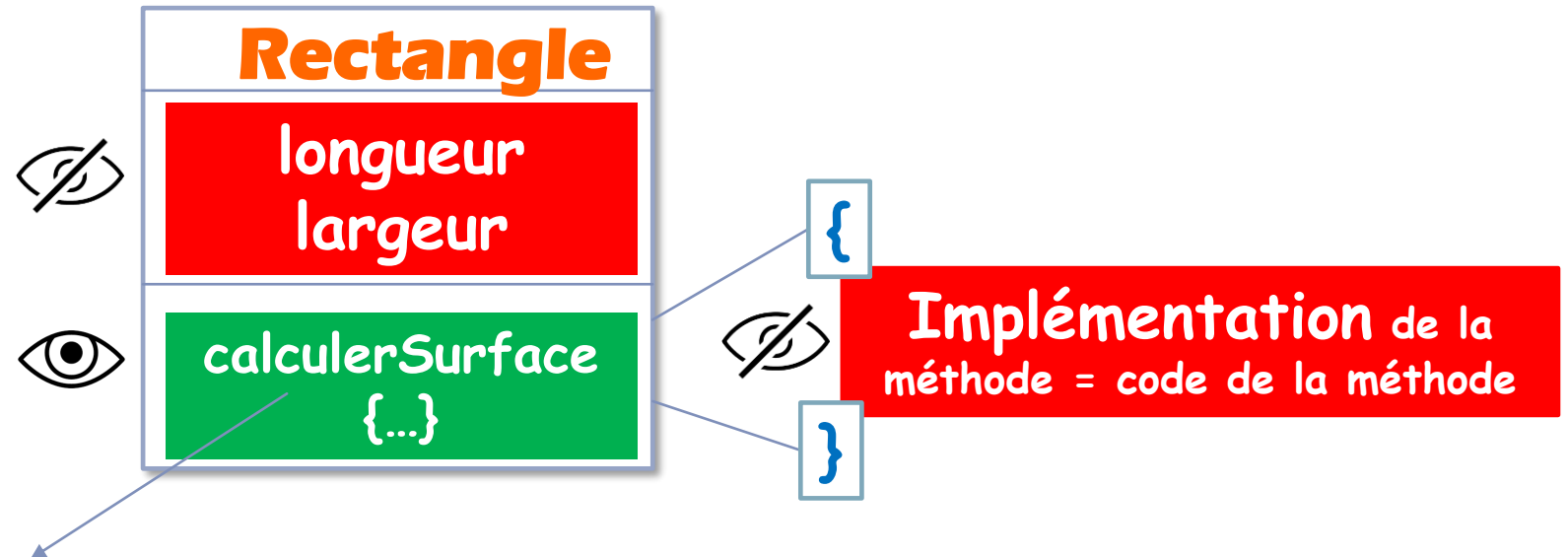
➔ L'utilisateur d'une **classe** a une visibilité réduite sur le code de **classe** : il ne peut que l'utiliser !



**Utiliser une classe nécessite uniquement de connaître les méthodes qui représentent les actions que sont capables de réaliser ses instances**

# UTILISATEUR

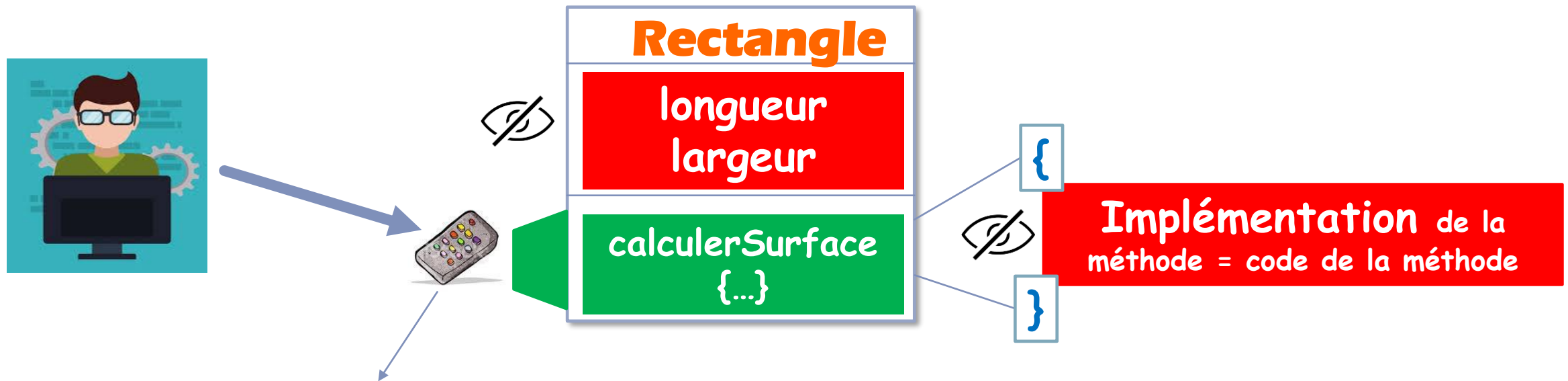
→ L'utilisateur d'une **classe** a une visibilité réduite sur le code de **classe** : il ne peut que l'utiliser !



**Signature** de la méthode

# UTILISATEUR

→ L'utilisateur d'une **classe** doit uniquement connaître les signatures des méthodes



**Interface** : signatures des méthodes



# UTILISATEUR

- L'utilisateur d'une **classe** a juste besoin de connaître l'**interface** proposée par la **classe**
- L'utilisateur d'une **classe** n'a pas à connaître les attributs ni le code (implémentation) des méthodes



**Attributs** + **Code méthodes** = **Détails d'implémentation**



Principe d'**encapsulation**

## PRINCIPE D' ENCAPSULATION (SUITE)

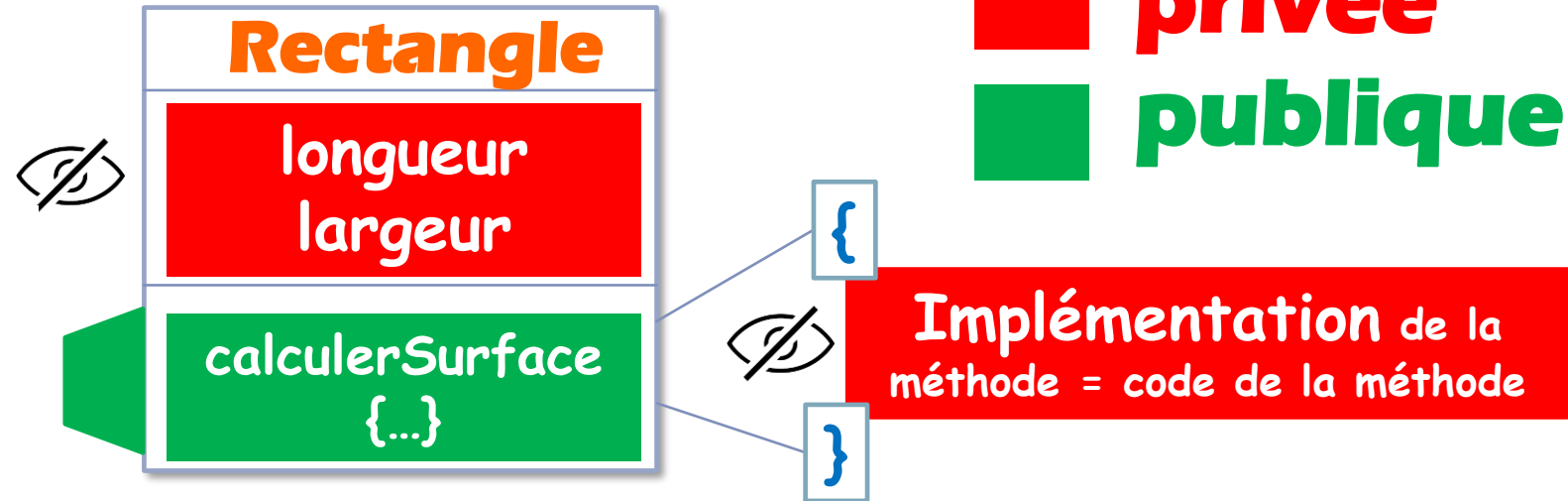
- ➔ Principe d'**encapsulation** (encore lui !) consiste à cacher les **détails d'implémentation** aux utilisateurs d'une **classe**
- ➔ D'un point de vue utilisateur, une **classe** (et ses instances) est vue comme une **boite noire**




# ENCAPSULATION

➔ Afin de respecter le principe d'encapsulation et de le mettre en œuvre, une classe doit être "découpée" en 2 parties : une **partie privée** et une **partie publique**

**Interface**



## PARTIE PRIVÉE

- La **partie privée** : les détails d'implémentation de la **classe**
- Cette partie est **inaccessible** (invisible) de l'extérieur de la **classe** → utilisateurs de la classe 
- L' utilisateur d'une **classe** n'a pas à connaître les **détails d'implémentation** pour l'utiliser

**Partie privée** = **Fonctionnement interne**

## PARTIE PUBLIQUE

- La **partie publique** : l'**interface** de la **classe**
- Cette partie est **accessible** (visible) de l'**extérieur** de la classe → utilisateurs de la classe 
- Un utilisateur d'une **classe** doit connaître son **interface** pour l'utiliser



**Partie publique**



**Mode d'emploi**

## ATTRIBUTS – PARTIE PRIVÉE

- Les **attributs** de doivent **JAMAIS** être déclarés dans la **partie publique** : pas d'accès direct !



Violation du principe d'**encapsulation**

- Un utilisateur doit pouvoir accéder aux attributs de manière indirecte : par des méthodes !



Respect du principe d'**encapsulation**

# RÉSUME ENCAPSULATION



Principe **fondamental en POO** consistant à

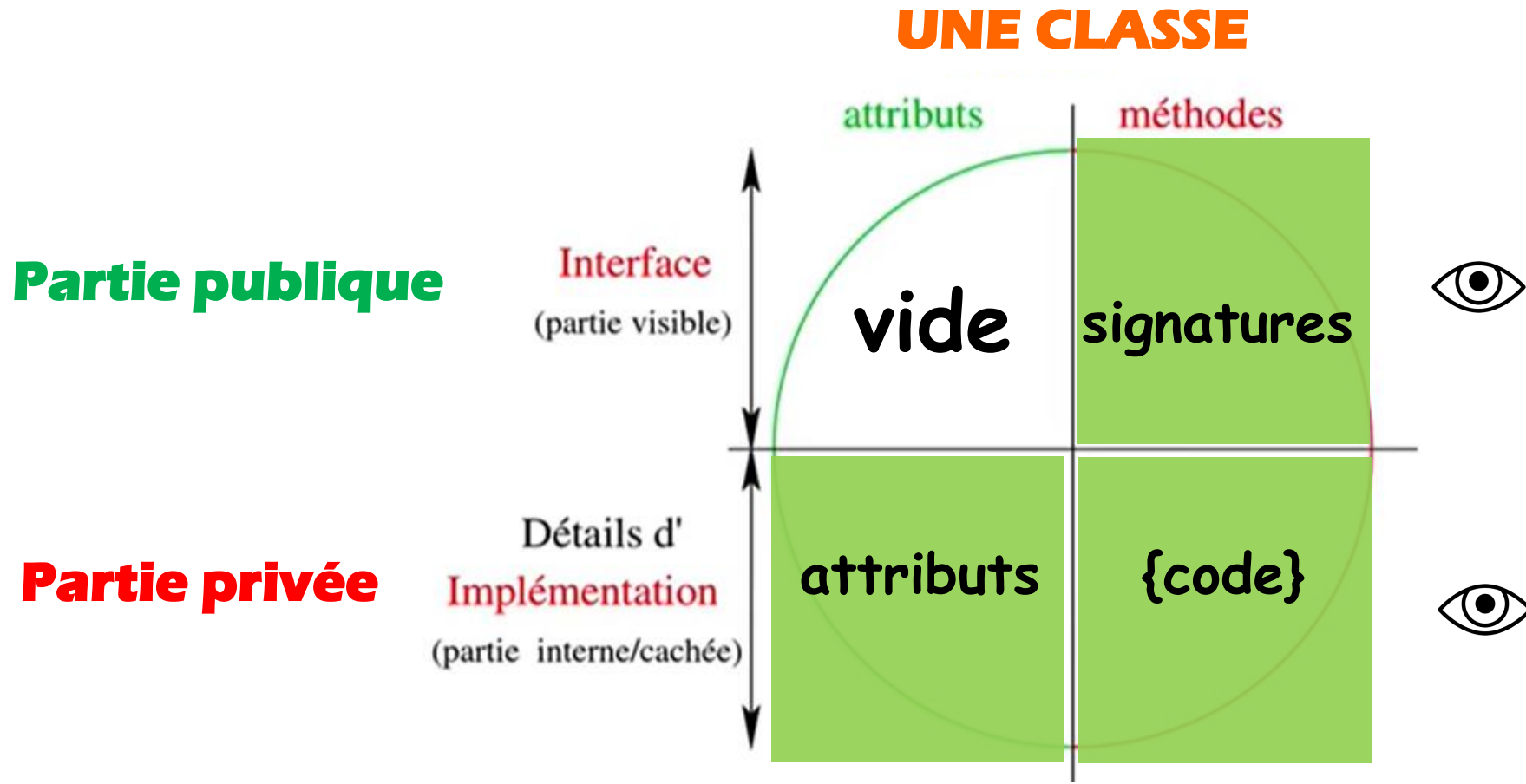


Regrouper des données (**attributs**) et des traitements (**méthodes**) au sein d'une même **entité** : la **classe**



Cacher les **détails d'implémentation** aux utilisateurs de la **classe** en leur proposant une **interface**

# VUE CONCEPTEUR DE LA CLASSE





## VUE UTILISATEUR DE LA CLASSE

