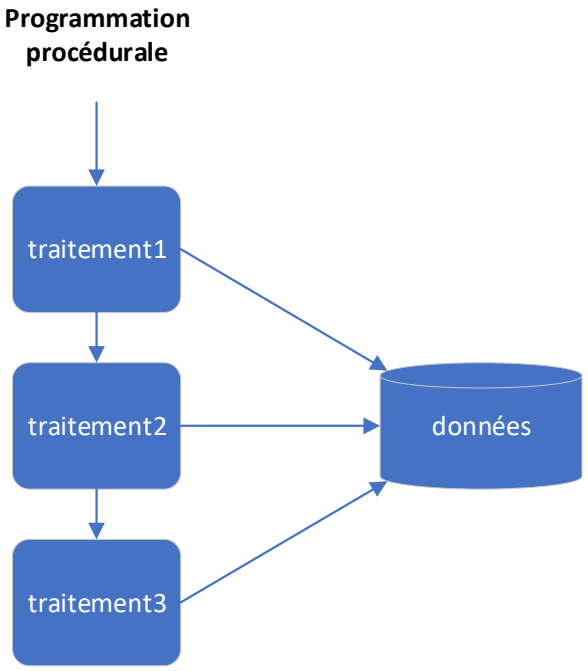
Introduction à la programmation orientée objet

Approche classique de développement d'un programme

- La méthode **classique**, également appelée **procédurale**, est axée sur la **conception descendante** (TopDown): on décompose le traitement principal du problème en sous-traitements élémentaires indépendants correspondants à des sous-programmes de réalisation simple et facile. Chaque sous-programme est soit une procédure, soit une fonction.
- **L'unité de décomposition est donc le traitement** (l'action).
- **Les données et les traitements** apparaissent comme des entités **séparées** dans un programme.
- Les données sont accessibles et modifiables par le code contenu dans les procédures et fonctions par le biais des paramètres.



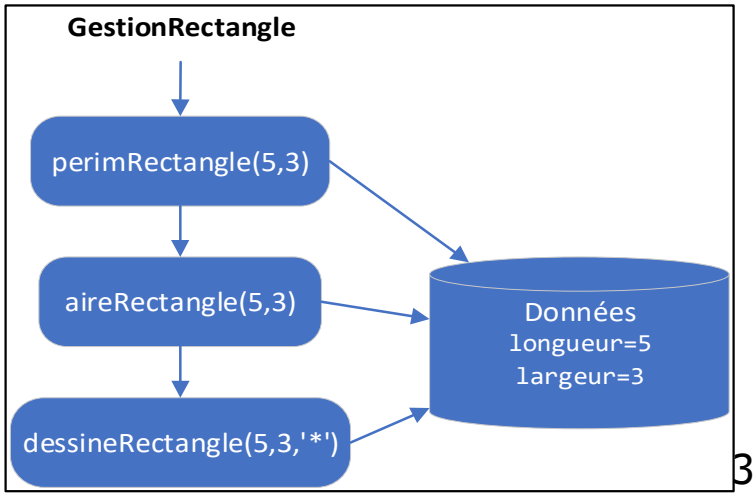
Approche classique de développement d'un programme: exemple

On veut calculer l'aire et le périmètre d'un rectangle à partir de sa longueur et sa largeur. On veut afficher une représentation d'un rectangle en utilisant un symbole, par exemple '*'. Les sous-traitements sont:

- Le calcul du périmètre: perimRectangle()
- Le calcul de l'aire : aireRectangle()
- L'affichage du rectangle: dessineRectangle()

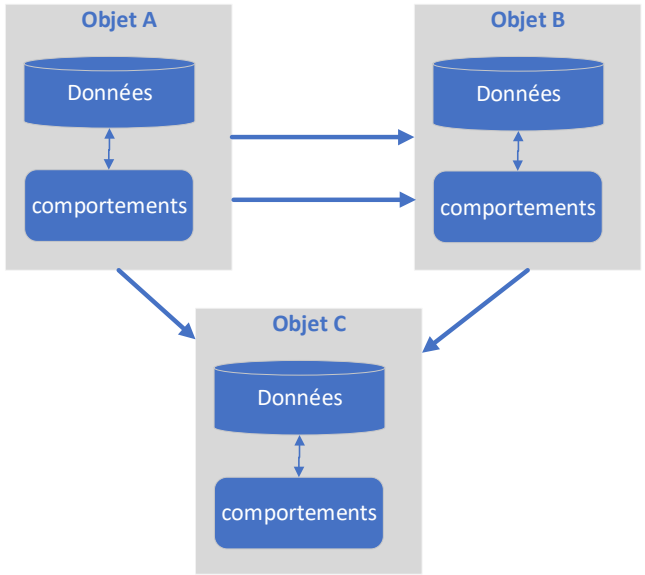
```
public class GestionRectangle {
    public static int perimRectangle(int pLong, int pLarg) {
        int perim;
        perim = (pLong + pLarg) * 2;
        return perim;
    }
    public static int aireRectangle(int pLong, int pLarg) {
        return pLong * pLarg;
    }
    public static void dessineRectangle(int pLong, int pLarg, char pSymbole) {
        System.out.println("Représentation graphique du rectangle:");
        for (int i = 0; i < pLong; i++) {
            for (int j = 0; j < pLarg; j++) {
                System.out.print(pSymbole);
            }
            System.out.println();
        }
    }
    public static void main(String[] args) {
        int longueur, largeur;
        char symbole;
        Scanner clavier = new Scanner ((System.in));
        System.out.print("Entrez la longueur du rectangle: ");
        longueur = clavier.nextInt();
        System.out.print("Entrez la largeur du rectangle: ");
        largeur = clavier.nextInt();
        System.out.println("Le périmètre est: " + perimRectangle(longueur, largeur));
        System.out.println("L'aire est: " + aireRectangle(longueur, largeur));
        System.out.print("Entrez le symbole à utiliser pour dessiner le rectangle: ");
        symbole = clavier.next().charAt(0);
        dessineRectangle(longueur, largeur, symbole);
    }
}
```

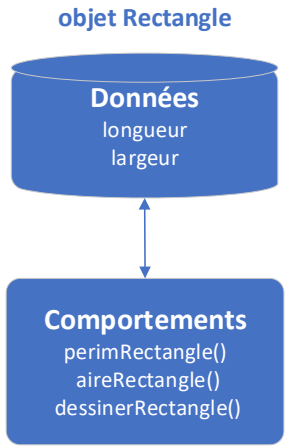
```
Entrez la longueur du rectangle: 5
Entrez la largeur du rectangle: 3
Le périmètre est: 16
L'aire est: 15
Entrez le symbole à utiliser pour dessiner le rectangle: *
Représentation graphique du rectangle:
***
***
***
***
***
```



Approche de développement d'un programme orienté objet

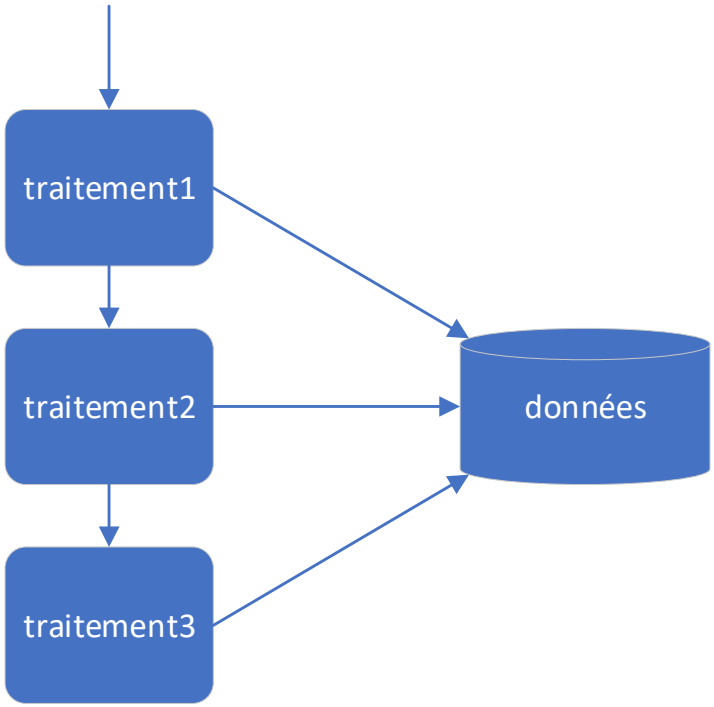
- La technique d'analyse de programmation orientée objet consiste d'abord à déterminer **les entités (objets)** appartenant au problème à modéliser ainsi que les opérations qui leur sont applicables et ensuite à se préoccuper de l'enchainement de ces opérations. On parle d'analyse de **conception ascendante**.
- **L'unité de décomposition est l'objet**.
- En d'autres termes, l'approche orientée objet consiste en:
  1. Identifier les objets du système. Chaque objet possède des informations qui le caractérisent (données) et un ensemble de compétences ou comportements (fonctions). Les données et les comportements sont réunis dans la même entité.
  2. Faire collaborer ces objets pour qu'ils accomplissent la tâche voulue : les objets interagissent en s'envoyant des messages (appels de fonctions)
- En programmation orientée objet, les données et les traitements sont regroupés en une seule et même entité qui est l'objet.



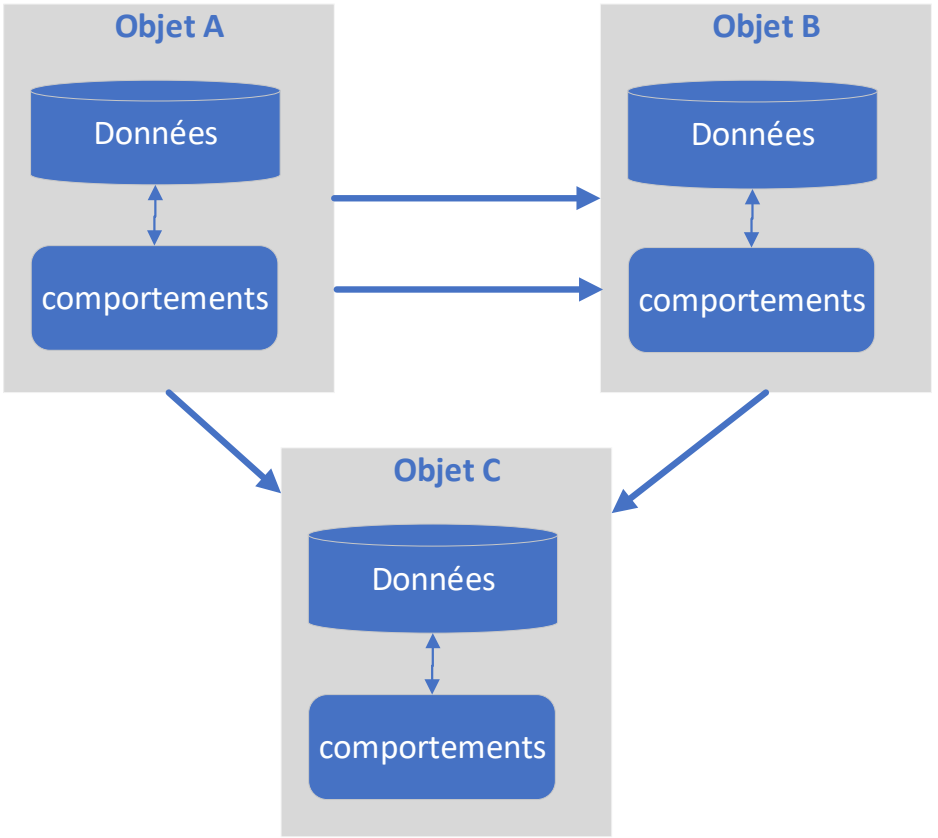


Approche de développement procédurale VS objet

procédurale



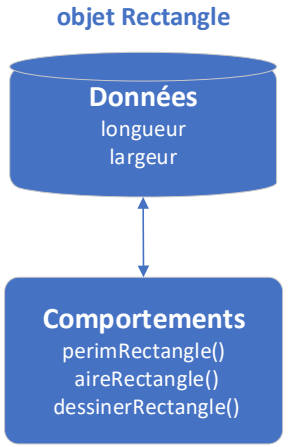
objet



Objet?

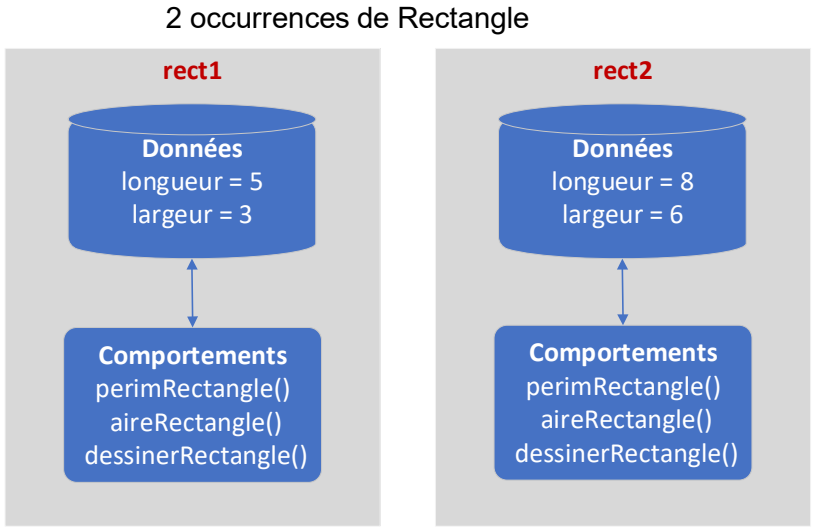
- Les objets sont à la base de la programmation orientée objet.
- Un **objet** possède **des attributs** (ou propriétés) qui définissent son état et des fonctionnalités permettant d'effectuer des **opérations** sur l'objet .
- Les attributs sont représentés par des variables d'instances et les fonctionnalités par des méthodes.

  Un objet = attributs + méthodes
- Exemple:
  - Un objet rectangle possède une longueur et une largeur. Ces caractéristiques sont des attributs de l'objet. Elles sont représentées par des variables d'instance: longueur et largeur.
  - Les opérations qu'on est capable d'effectuer sur un objet rectangle sont: calculer son aire, calculer son périmètre et dessiner le rectangle en utilisant un symbole. Ces opérations seront représentées par des méthodes: perimRectangle(), aireRectangle(), dessineRectangle().



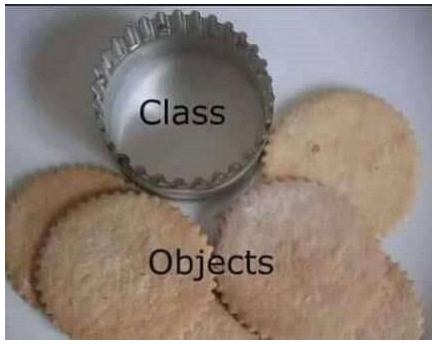
Occurrence d'un objet

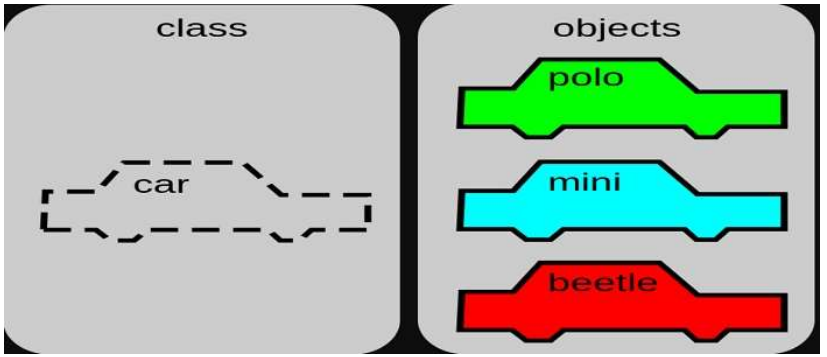
- On peut avoir **plusieurs occurrences** d'un objet. Chaque objet possède une valeur pour ces différents attributs qui font que cet objet est unique (état de l'objet).
  - L'objet rect1 aura une longueur de 5 et une largeur de 3.
  - L'objet rect2 aura une longueur de 8 et une largeur de 6.
- Chaque occurrence d'un objet possède le même comportement:
  - l'objet rect1, tout comme l'objet rect2, pourra calculer son aire, calculer son périmètre et se dessiner en utilisant un symbole.



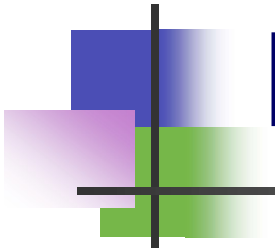
Classe?

- Chaque objet est décrit par **une classe qui contient toutes les caractéristiques qu'un objet peut avoir**. Il s'agit du **plan** à partir duquel on va pouvoir construire des objets.
- Autrement dit, les classes sont des **modèles** (comme les emporte-pièces) servant à faire des objets (comme les biscuits).





- **Tout objet appartient à une classe** qui représente **un type d'objet**. On dit aussi qu'un objet est une **instance** d'une classe.
- Exemples:
  - Rectangle est une classe (type d'objet). Les objets **rect1 et rect2 sont de type Rectangle**. On peut dire également que **rect1 et rect2 appartiennent à la classe Rectangle** ou sont **deux instances de la classe Rectangle.**
  - Car est une classe (type d'objet). Les objets polo, mini et beetle sont des objets de type Car. On peut dire également que ces 3 objets appartiennent à la classe Car ou sont 3 instances de la classe Car.

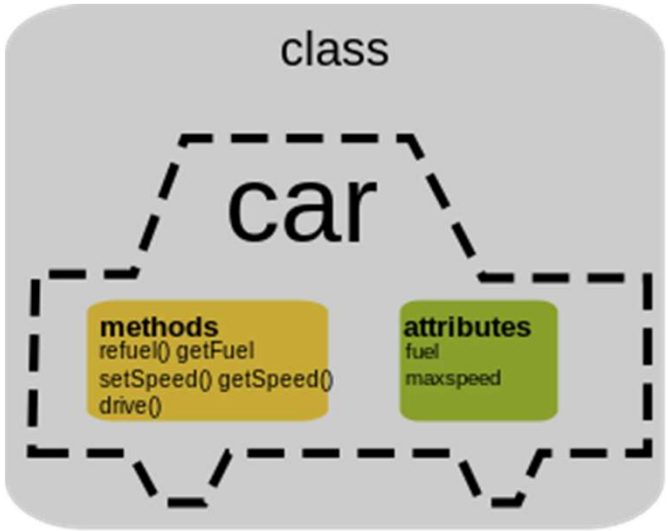


Exercice1

1. Quels sont les attributs (propriétés) et les méthodes (opérations) que peut avoir un objet de type Car (une voiture) ?
2. Décrivez l'état (ensemble des attributs) de deux occurrences de type Car.

Exercice1: corrigé

1. Quels sont les attributs (propriétés) et les méthodes (opérations) que peut avoir un objet de type Car (une voiture) ?
   - Attribut :
     - marque, modèle, immatriculation, couleur, consommation (nombre de litres / 100 km), vitesse maximale
   - Méthodes:
     - avancer, reculer, accélérer, tomber en panne, faire le plein d'essence
2. Décrivez l'état (ensemble des attributs) de deux occurrences de la classe Car.
   - Première occurrence: polo:
     - Marque: Toyota
     - Modèle: Corolla
     - Immarticulation: Y1B2C3
     - Couleur: verte
     - Consommation: 6
     - vitesse maximale: 180
   - Deuxième occurrence: beetle
     - Marque: Hundai
     - Modèle: Elantra
     - Immarticulation: Z9S2C3
     - Couleur: rouge
     - Consommation: 7
     - vitesse maximale: 170



Déclaration d'une classe

- Une classe, qui définit un type d'objet, est composée de la structure suivante :
  - Un nom.
  - **Des attributs** et **des méthodes** contenus **dans une même capsule**.
- Pour déclarer une nouvelle classe, il suffit d'utiliser le mot-clé `class`, suivi du nom de la classe. En java, par convention, **le nom de la classe commence par une lettre majuscule.**
- Si on veut que notre classe soit accessible par les autres classes, on précède le mot `class` par `public`.

```
public class NomDeLaClasse {
    Déclaration des variables d'instance (attributs)

    Déclaration des méthodes d'instance (fonctionnalités)
}
```

- Notes:
  - La déclaration des attributs peut être placée avant ou après la déclaration des méthodes.
  - Le terme **instance** signifie **spécifique à chaque occurrence (instance) de la classe**. Nous verrons dans le cours suivant qu'il existe des attributs et des méthodes de classe.

Les attributs d'instance

- Les **attributs** d'instance décrivent les informations caractéristiques des objets de la classe. Ils sont aussi appelés communément **champs d'instance**, **variables d'instance** ou **données membres de la classe.**
- Ils sont définis à l'aide d'instructions de déclaration de variables. Ces variables peuvent être de type simple (int, float, char…) ou de type composé (String, tableau...).

```
public class Rectangle {

   private int longueur;
   private int largeur;

   public int perimRectangle() {
            int perim;
            perim = (longueur + largeur) * 2;
            return perim;
   }
   public int aireRectangle() {
            return longueur * largeur;
   }
   public void dessineRectangle(char caractere) {
      for (int i = 0; i < longueur; i++) {
         for (int j = 0; j < largeur; j++) {
             System.out.print(caractere);
         }
         System.out.println();
      }
    }
}
```

2 attributs d'instance accessibles par toutes les méthodes de la classe .

Accès direct aux attributs de la classe

Les méthodes d'instance

- Les méthodes d'instance représentent les traitements et comportements des objets de la classe. Ce sont les méthodes qui agissent sur les données.
- Elles sont aussi appelées **méthodes membres de la classe** ou **fonctions**.
- Elles se construisent comme de simples méthodes: composées d'un en-tête et d'instructions**.**
- Attention! Une variable locale à une méthode n'est pas un attribut d'une classe**.**

```
public class Rectangle {
   private int longueur;
   private int largeur;

   public int perimRectangle() {
      int perim;
      perim = (longueur + largeur) * 2;
      return perim;
   }
   public int aireRectangle() {
      return longueur * largeur;
   }
   public void dessineRectangle(char caractere) {
      System.out.println("Représentation graphique du rectangle:");
      for (int i = 0; i < longueur; i++) {
         for (int j = 0; j < largeur; j++) {
             System.out.print(caractere);
         }
         System.out.println();
      }
    }
}
```

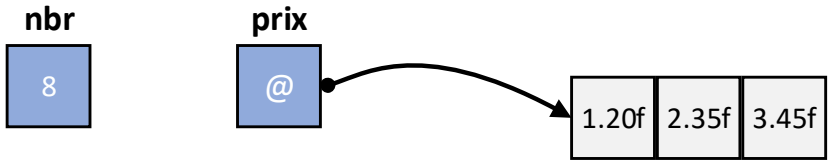
`perim` est une variable locale à la méthode . Ce n'est pas un attribut.

Méthode avec paramètre pour recevoir une donnée de l'extérieur de l'instance

La variable en mémoire (rappel)

- Il y a deux types de variables: les variables de type primitif (byte, short, int, long, float, double, boolean, char) et les variables de type objet (String, les tableaux, Scanner…).
- **Une variable de type primitif (simple) stocke sa valeur en mémoire**.
- **Une variable de type objet ne stocke pas directement l'objet, mais une référence** (adresse mémoire) vers l'objet, qui lui, est stocké ailleurs dans la mémoire. Une variable de ce type est appelée variable de référence.

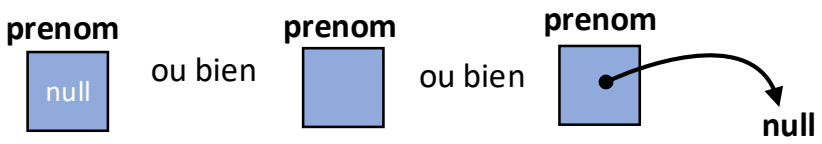
```
int nbr = 8;  //variable de type primitif(simple)
float[] prix = {1.20f, 2.35f, 3.45f} ; //variable de type objet
```



La valeur null

- En java, la valeur **null** est utilisée pour indiquer **qu'aucune valeur n'est affectée à une variable de référence**.
- Lorsqu'une variable de référence (un objet) est déclarée, mais n'est pas encore initialisée, elle contient la valeur de référence **null** (elle ne pointe vers nul part)

```
String prenom;
// prenom ne contient aucune référence: l'objet prenom n'existe pas, il doit
// être créé à l'aide de l'opérateur new.
```



Construction des objets (1/6)

- La classe est un modèle qui s'apparente à la déclaration d'un nouveau type de données.
- On peut **déclarer une variable du type de cette classe**. Cette variable est appelée **objet** ou **instance d**e la classe.
- L'opération permettant de créer un objet à partir d'une classe s'appelle **instanciation**.
- Exemple:
  - `Rectangle rect = new Rectangle (8,7) // rect est une instance de Rectangle`
- Pour pouvoir instancier des rectangles comme ci-haut, nous ajouterons une méthode spéciale appelée **constructeur** à l'intérieur de la classe. C'est une **méthode publique sans le mot void qui porte le même nom que la classe.**

```
public class Rectangle {

   private int longueur;
   private int largeur;

   public Rectangle(int longu, int larg) {
      longueur = longu;
      largeur = larg;
   }
   public int perimRectangle() {
      //…
   }
   public int aireRectangle() {
      return longueur * largeur;
   }
   public void dessineRectangle(char caractere) {
      //…
}
```

Constructeur

Construction des objets (2/6)

- La **construction**, aussi appelée **instanciation** ou **création**, d'un objet à partir d'une classe nécessite 2 étapes:
  1- la déclaration de l'objet
  2- l'initialisation de l'objet
- Exemple: construction d'un objet de la classe Rectangle

```
Rectangle rect1; // déclaration
rect1 = new Rectangle(5,3) // initialisation
```

- On peut rassembler ces deux étapes sur une seule ligne.

```
Rectangle rect1 = new Rectangle(5,3) // déclaration et initialisation
```

- Lorsqu'une variable est déclarée, mais n'est pas encore initialisée, elle contient la valeur de référence `null`.

Construction des objets (3/6)

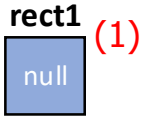
- Les 2 étapes de construction d'un objet :

- **1- La déclaration de l'objet:**

  Elle consiste en la création d'une variable dont le type correspond au nom de la classe de l'objet (1),

  ```
  Rectangle rect1;
  ```

  - Après exécution de cette ligne de code, une nouvelle variable est créée, mais ne contient aucune référence (elle contient la valeur **`null`**).

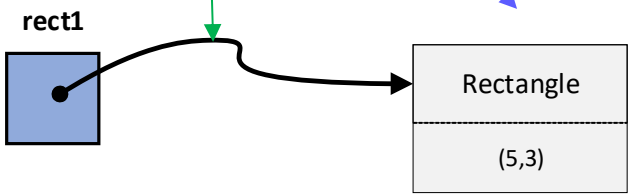


- **2- L'instanciation de l'objet en utilisant l'opérateur new.**

  ```
  rect1 = new Rectangle(5,3)
  ```

  - L'exécution de cette instruction par la machine virtuelle Java se fait en deux temps:
    - L'opérateur new, suivi du constructeur à sa droite, crée un nouvel objet en mémoire (une instance de la classe Rectangle) (2)
    - L'opérateur new renvoie la référence (adresse mémoire) de l'objet créé, et celle-ci est affectée à la variable rect1 grâce à l'opérateur d'affectation = (3)

```
rect1 = new Rectangle(5,3) // déclaration et initialisation.
```

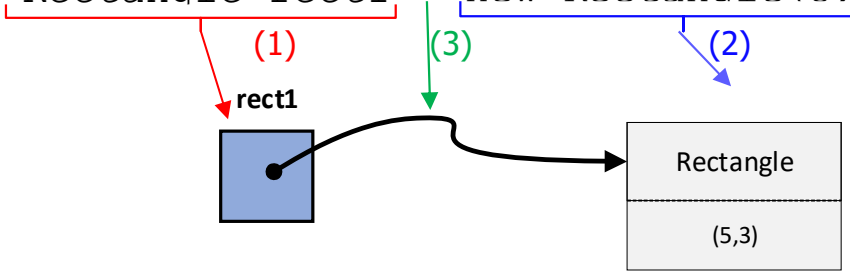


La flèche représente une référence.

Construction des objets (4/6)

Il est possible de déclarer et d'initialiser en une seule instruction comme pour les variables de type simple:

```
Rectangle rect1 = new Rectangle(5,3) // déclaration et initialisation.
```



La flèche représente une référence.

Construction des objets (5/6): exemple

```
public class Rectangle {

   private int longueur;
   private int largeur;

   public Rectangle(int longu, int larg) {
            longueur = longu;
            largeur = larg;
   }
   public int perimRectangle() {
            int perim;
            perim = (longueur + largeur) * 2;
            return perim;
   }
   public int aireRectangle() {
            return longueur * largeur;
   }
   public void dessineRectangle(char caractere) {
      System.out.println("Représentation graphique du rectangle:");
      for (int i = 0; i < longueur; i++) {
         for (int j = 0; j < largeur; j++) {
              System.out.print(caractere);
         }
         System.out.println();
      }
    }
}
```

- On peut créer des instances de la classe Rectangle:

```
Rectangle rect1 = new Rectangle(9, 3);
Rectangle rect2 = new Rectangle(7, 2);
```

`rect1` et `rect2` sont 2 **instances** de la classe `Rectangle`

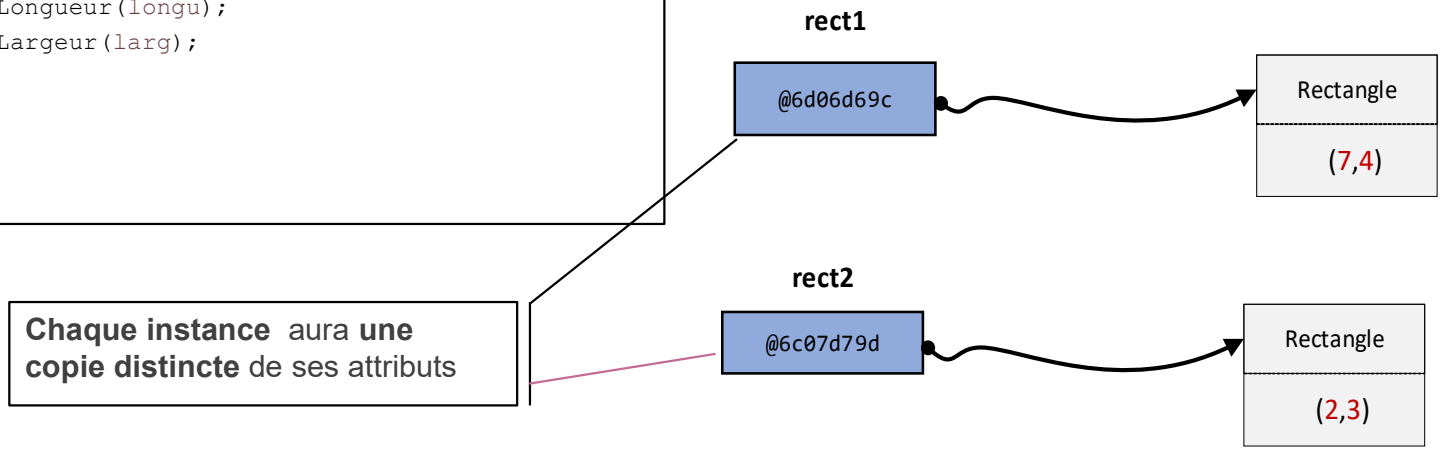
Classe Rectangle

Construction des objets (6/6)

- Chaque nouvel objet de la classe Rectangle initialisé **aura une copie distincte de ses attributs ou variables** d'instances.
- Dans l'exemple ci-dessous, `rect1` aura comme longueur la valeur 7 et comme largeur la valeur 4, alors que `rect2` aura comme longueur la valeur 2 et comme largeur la valeur 3.

```
public class Rectangle {

   // Attributs(ou ou variables d'instances)
   private int longueur;
   private int largeur;

   public Rectangle(int longu, int larg) {
      setLongueur(longu);
      setLargeur(larg);
}

  ...
}
```

```
public static void main(String[] args) {
   Rectangle rect1, rect2;
   rect1 = new Rectangle(7, 4);
   rect2 = new Rectangle(2, 3);
}
```



Exercice2

- Créez les objets Cours zc6 et zh5 ci-dessous.
- Voici les étapes à suivre:
  1. Créez la classe des objets. Donnez un nom significatif, par exemple **Cours.** Ajoutez les attributs et un constructeur.
  2. Créez deux instances zc6 et zh5 de cette classe en utilisant l'opérateur new.

zc6	zh5
Code: 420-ZC6-MO	**Code**: 420-ZH5-MO
Titre: Algorithmie et programmtion	**Titre**: Base de données
Session: 1	**Session**: 3
Nombre d'heures par semaine : 6	**Nombre d'heures par semaine:** 5

- Ajoutez une méthode qui affiche les informations d'un cours à la console sous le format ci-dessous:

```
Informations du cours :420-ZH5-MO
**********
Titre : Base de données
Session : 3
Nombre d'heures par semaine : 5
```

- Ajoutez une méthode qui calcule le nombre d'heures d'un cours par session sachant qu'une session a 15 semaines.

Exercice2 : corrigé

```java
public class Cours {
    private String code;
    private String titre;
    private int session;
    private int nbHeures;
    // constructeur
    public Cours(String pCode, String pTitre, int pSession, int pNbHeures) {
        code = pCode;
        titre = pTitre;
        session = pSession;
        nbHeures = pNbHeures;
    }
    public void afficher() {
        System.out.println("Informations du cours : " + code);
        System.out.println("*******************************");
        System.out.println("Titre : " + titre);
        System.out.println("Session : " + session);
        System.out.println("Nombre d'heures par semaine : " + nbHeures);
    }
    public int heuresParSession() {
        return 15 * nbHeures;
    }
}
```

```java
public class GestionCours {

    public static void main(String[] args) {
        //création de l'objet zc6 en une seule ligne
        Cours zc6 = new Cours("420-ZC6-MO", "Algorithmie et programmation", 1, 6);
        //création de l'objet zc6 en deux lignes
        Cours zh5;
        zh5 = new Cours("420-ZH5-MO", "Base de données", 3, 5);
        zh5.afficher();
        System.out.println("Nombre d'heures par session : " + zh5.heuresParSession());
    }
}
```

Visibilité des membres d'une classe : public et private

- La visibilité des membres (attributs et méthodes) d'une classe définit les endroits d'où ils peuvent être utilisés.
- Les méthodes et attributs **privés** (`private`) d'une classe ne sont accessibles que par les méthodes de la classe.
- Les méthodes et attributs **publics** (`public`) d'une classe sont accessibles de l'extérieur de la classe.
- Nous verrons plus tard dans le cours d'autres types de visibilité.

```
public class Rectangle {

   private int longueur; // cet attribut n'est pas accessible à l'extérieur de la classe
   private int largeur; //cet attribut n'est pas accessible à l'extérieur de la classe

   public Rectangle(int longu, int larg) {
      longueur = longu;
      largeur = larg;
   }
   public int perimRectangle() {  // méthode accessible à l'extérieur de la classe
      int perim;
      perim = (longueur + largeur) * 2;
      return perim;
   }
   public int aireRectangle() {  // méthode accessible à l'extérieur de la classe
      return longueur * largeur;
   }

   private boolean isCarre() { // méthode non accessible à l'extérieur de la classe
      //…
   }
    //…
}
```

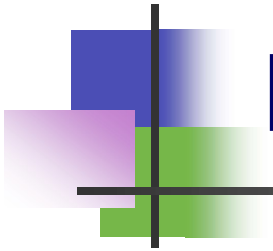
Syntaxe à point

- On peut accéder aux différents éléments accessibles d'un objet avec la syntaxe à point suivante: **`nomObjet.nomElementAccessible.`**

Exemple: `rect1.aireRectangle()`

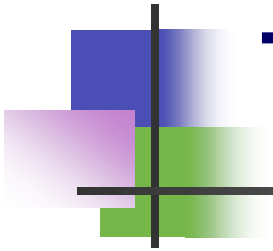
```
public class GestionRectangle {
     public static void main(String[] args) {
        //déclarer une variable de type rectangle
       Rectangle rect1;

       // construire un rectangle de longueur 7 et de largeur 4
       rect1 = new Rectangle(7, 4);

       // Afficher l'aire de rect1
       System.out.print("l'aire du rectangle 1: ");
       System.out.println(rect1.aireRectangle());

       // Afficher le périmètre de rect1
       System.out.print("le périmètre du rectangle 1: ");
       System.out.println(rect1.perimRectangle());

       // dessiner rect1 à la console avec le symbole *
       System.out.println("Représentation graphique du rectangle 1:");
       rect1.dessineRectangle('*');
    }
}
```

```
l'aire du rectangle 1: 28
le périmètre du rectangle 1: 22
Représentation graphique du
rectangle 1:
****
****
****
****
****
****
****
```



Principe d'encapsulation

- La programmation orientée objet repose sur trois piliers: l'encapsulation , l'héritage et le polymorphisme. L'héritage et le polymorphisme seront vus dans les prochains cours.
- **L'encapsulation est un mécanisme consistant à rassembler les données et les méthodes au sein d'une structure en cachant l'implémentation de l'objet au monde extérieur**. Ainsi, l'accès aux données ne doit se faire que par les méthodes (les services proposés). L'encapsulation permet donc de garantir l'intégrité des données contenues dans l'objet.
- Voici comment mettre en œuvre l'encapsulation :
  - **Privatiser les attributs** en utilisant le mot clé `private`. L'utilisation d'une autre visibilité doit être justifiée.
  - Si on a besoin d'accéder à un attribut à partir d'une autre classe, **créer une méthode d'accès** appelée accesseur (getter en anglais).
  - Si on a besoin de modifier un attribut à partir d'une autre classe, **créer une méthode appelée mutateur** (setter en anglais).
  - **Ne pas créer d'accesseurs/mutateurs si ce n'est pas nécessaire.**



Types de méthodes

Un objet a différents types de méthodes.

- Constructeur
  - Son rôle est d'initialiser l'objet.
- Accesseur (en anglais Getter)
  - Sous forme de `getNomDeVariable()` ou `isNomDeVariable()` pour les booléens.
  - Son rôle est d'accéder (en lecture) à un attribut de l'objet .
- Mutateur (en anglais Setter)
  - Sous forme `setNomDeVariable()`
  - Son rôle est de modifier un attribut d'un objet.
- Méthode métier (business method en anglais)
  - Effectue des calculs en fonction des données.

Accesseur (getter en anglais)

- Si on a besoin d'accéder en lecture à un attribut protégé à partir d'une autre classe, on crée une méthode appelée accesseur (getter en anglais).
- Cette méthode a les caractéristiques suivantes:
  - Elle **doit être `public`** pour pouvoir être appelée à partir d'une autre classe.
  - Elle doit avoir comme **type de retour le type de la variable à renvoyer**.
  - Elle ne possède pas habituellement de paramètre.
  - Par convention, **son nom doit être préfixé par «get» et suivi par le nom de la variable**, comme getLargeur, getLongueur, getNote. Pour les booléens, le préfixe sera «**is**».

Exemple de la classe Rectangle:

```
public class Rectangle {
   private int longueur;
   private int largeur;

 //accesseur de longueur
   public int getLongueur() {
     return longueur;
   }

//accesseur de largeur
   public int getLargeur() {
     return largeur;
   }
//...
}
```

```
public class AutreClasse {
  public static void main(String[] args) {
     int lng, larg;
     Rectangle rect = new Rectangle(5, 3);
     // obtenir la longueur
     lng = rect.getLongueur();
      // obtenir la largeur
     larg = rect.getLargeur();
}
```

Les attributs longueur et largeur sont accessibles à partir de toutes les méthodes de la classe même s'ils sont privés (private). Par contre, ils **ne sont pas accessibles de l'extérieur de la classe.**

Mutateur (setter en anglais)

- Si on a besoin de modifier un attribut protégé à partir d'une autre classe, on crée une méthode appelée mutateur (setter en anglais).
- Cette méthode a les caractéristiques suivantes:
  - Elle doit être **une méthode** `void` (elle ne retourne aucun résultat).
  - Elle **doit être** `public` pour pouvoir être appelée à partir d'une autre classe.
  - Elle doit **avoir comme paramètre la valeur à assigner à la donnée membre**. Le paramètre doit donc être du type de la donnée membre.
  - Par convention, **son nom doit être préfixé par «set» et suivi par le nom de la variable,** comme setLargeur, setLongueur, setNote.

Exemple de la classe Rectangle:

```
public class Rectangle {
   private int longueur;
   private int largeur;

   public void setLongueur(int longu) {
      longueur = longu;
   }
   public void setLargeur(int larg) {
      largeur = larg;
   }
   //...
}
```

```
public class AutreClasse {
   public static void main(String[] args) {
      Rectangle rect = new Rectangle(5, 3);
      rect.setLongueur(10); //mofier la longueur de rect
      rect.setLargeur(20); // mofier la largeur de rect
   }
}
```

Les attributs longueur et largeur sont accessibles à partir de toutes les méthodes de la classe même s'ils sont privés (`private`).
Par contre, ils **ne sont pas accessibles de l'extérieur de la classe.**

Validation des données dans le mutateur

- On peut valider un attribut avant de le modifier dans un mutateur. Dans ce cas, il est intéressant de faire **appel à ce mutateur dans le constructeur pour initialiser l'attribut**.

```
public class Rectangle {

      private int longueur;
      private int largeur;

      public Rectangle(int longu, int larg) {
          setLongueur(longu); //appel du mutateur pour valider longueur
          setLargeur(larg); //appel du mutateur pour valider largeur
      }

      public void setLongueur(int longu) {
          if (longu > 0) { // sinon longueur = 0, valeur par défaut
              longueur = longu;
          }
      }

      public void setLargeur(int larg) {
          if (larg > 0) { // sinon largeur = 0, valeur par défaut
              largeur = larg;
          }
      }
}
```

Référence this

- À l'intérieur d'une classe, le mot clé **`this` fait référence à l'objet courant,** c'est-à-dire **l'objet qui est en train d'être manipulé.**
- L'utilisation la plus courante de `this` est d'éliminer la confusion entre un attribut d'une classe et un paramètre d'une méthode ou d'un constructeur de mêmes noms.

```
public class Rectangle {
   private int longueur;
   private int largeur;

   public void setLongueur(int lo) {
      longueur = lo;
   }

   public void setLargeur(int largeur) {
      this.largeur = largeur;
   }
   ...
}
```

Ici, `this` est implicite (sous-entendu) , il n'est pas nécessaire.
`longueur` est équivalent à `this.longueur`.

Ici, `this` est nécessaire, car l'attribut `largeur` est masqué par le paramètre `largeur` de la méthode `setLargeur()`.

- **`this`** peut également être utilisé dans d'autres situations qu'on verra plus loin, comme appeler le constructeur de la classe actuelle.

Référence this

This implicite: Dans ce code, le this n'a pas besoin d'être spécifié, il est implicite. Cela permet d'avoir un code plus clair.

```
public class Rectangle {
   private int longueur;
   private int largeur;

   public Rectangle(int pLongueur, int pLargeur) {
     longueur = pLongueur;
     largeur = pLargeur;
   }

   public int getLongueur() {
       return longueur;
   }

   public void setLargeur(int pLargeur) {
      largeur = pLargeur;
   }

   public int perimRectangle() {
      int perim;
      perim = (longueur + largeur) * 2;
      return perim;
  }
//…
}
```

Ici, `this` est implicite (sous-entendu) , il n'est pas nécessaire.
Ce code est équivalent à:

```
this.longueur = pLongueur
this.largeur = pLargeur
```

Exercice3:

- Ajoutez le mot clé `this` aux endroits nécessaires

```
public class Rectangle {
   private int longueur;
   private int largeur;

   public Rectangle(int largeur, int longueur) {
     longueur = longueur;
     largeur = largeur;
   }

   public int getLongueur() {
      return longueur;
   }

   public void setLargeur(int largeur) {
       largeur = largeur;
   }

   public int perimRectangle() {
      int perim;
      perim = (longueur + largeur) * 2;
      return perim;
  }
//…
}
```

Exercice3: Corrigé

- Indiquez si le mot clé `this` est nécessaire devant les attributs longueur et largeur de la classe `Rectangle`.

```
public class Rectangle {
   private int longueur;
   private int largeur;

   public Rectangle(int largeur, int longueur) {
     this.longueur = longueur; // this est nécessaire
     this.largeur = largeur; // this est nécessaire
   }

   public int getLongueur() {
      return this.longueur; // this est inutile, Il est implicite
   }

   public void setLargeur(int largeur) {
       this.largeur = largeur; // this est nécessaire
   }

   public int perimRectangle() {
      int perim;
      perim = (this.longueur + this.largeur) * 2;
      //Les 2 this sont inutiles ici-haut. Ils sont implicites.
     return perim;
  }
//…
}
```

Exercice4

1. On modifie les noms des paramètres des méthodes de la classe `Cours` comme indiqué ici-bas. Complétez les méthodes ainsi que le constructeur de cette classe. Téléchargez le nouveau code de la classe Cours dans Moodle.
2. Doit-on modifier la classe `GestionCours` à la suite de ces changements?
3. Une session est valide si elle est comprise entre 1 et 6. Modifiez le mutateur de session pour ajouter sa validation. Mettez à jour le constructeur.
4. Un code cours est valide s'il a 10 caractères, sinon la chaine «INCONNU» est enregistrée dans la variable. Le code doit être converti en majuscule dans tous les cas. Modifiez le mutateur du code de cours pour ajouter sa validation.
5. Mettez à jour le constructeur.

```java
public class Cours {
	private  String code;
	private String titre;
	private int session;
	private int nbHeures;

	// constructeur
	public Cours(String code, String titre, int session, int nbHeures) {
		// à compléter

	}
	// mutateurs
	public void setCode(String code) {
		// à compléter
	}
	public void setTitre(String titre) {
		// à compléter
	}
	public void setNbHeures(int nbHeures) {
		// à compléter
	}
	public void setSession(int session) {
		// à compléter
	}
…
}
```

```java
    // Accesseurs
    public String getCode() {
        return code;
    }
    public int getSession() {
        return session;
    }

    //Méthodes métiers
    public void afficher() {
        System.out.println("Informations du cours: " + code);
        System.out.println("******************************");
        System.out.println("Titre: " + titre);
        System.out.println("Session: " + session);
        System.out.println("Nombre d'heures par semaine: " + nbHeures)
    }

    public int heuresSession() {
        return nbHeures * 15;
    }

}
```

Exercice4: corrigé

Question1:

```
public class Cours {
        private String code;
        private String titre;
        private int session;
        private int nbHeures;
        // constructeur
        public Cours(String code, String titre, int session, int nbHeures) {
                this.code = code;
                this.titre = titre;
                this.session = session;
                this.nbHeures = nbHeures;

        }
        // mutateurs
        public void setCode(String code) {
                this.code = code;
        }
        public void setTitre(String titre) {
                this.titre = titre;
        }
        public void setNbHeures(int nbHeures) {
                this.nbHeures = nbHeures;
        }
        public void setSession(int session) {
                this.session =session;
        }
        // Accesseurs
        public String getCode() {
                return code;
        }

        public int getSession() {
                return session;
        }
}
```

Exercice4: corrigé

Question 2: non

Questions 3, 4 et 5

```
public Cours(String code, String titre, int session, int nbHeures) {
	setCode(code);		// setCode() valide le code avant de la modifier
	this.titre = titre;
	setSession(session); // setSession() valide la session avant de la modifier
	this.nbHeures = nbHeures;

}

public void setSession(int session) {
	if (session <= 6 && session >= 1) {
		this.session = session;
	}
}

public void setCode(String code) {
	if (code.length() == 10) {
		this.code = code.toUpperCase();
	} else {
		this.code = "INCONNU";
	}
}
```

Afficher les informations d'un objet

- La variable `rect` ne stocke pas directement l'objet créé, mais une référence vers l'emplacement mémoire où se trouve cet objet. Par exemple:

```
Rectangle rect = new Rectangle(8, 4);
System.out.println(rect); // affiche Rectangle@6d06d69c
```

- Si on veut afficher les informations de `rect`, on utilise les accesseurs et les autres méthodes comme suit:

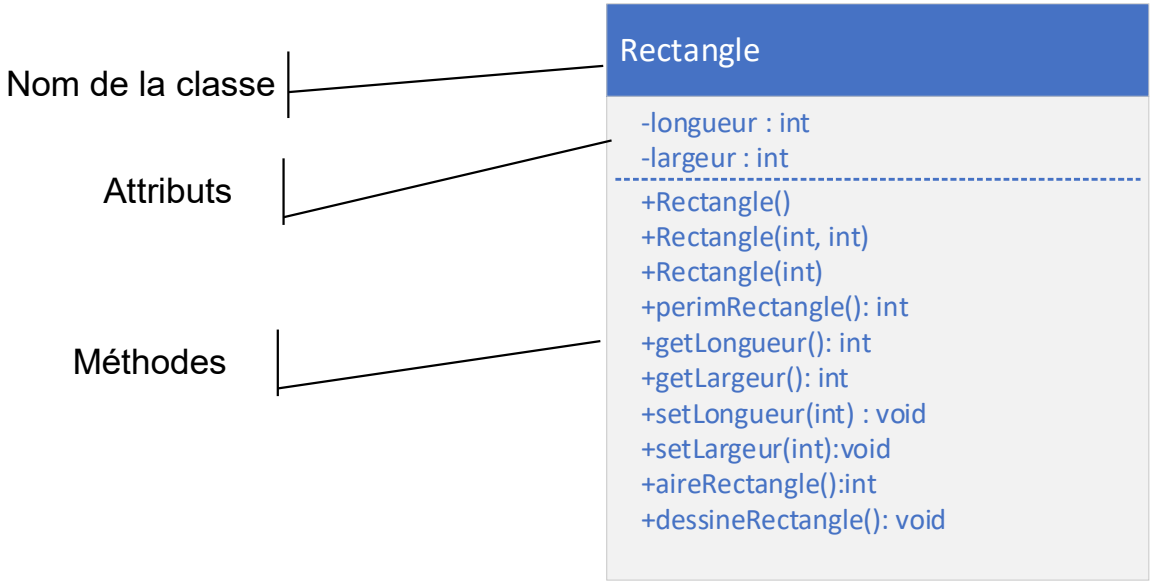
```
System.out.println("Largeur:" + rect.getLargeur() + "\n" + "Longueur: " +
rect.getLongueur() + "\n" + "périmètre:" + rect.perimRectangle());
```

- L'affichage à la console est comme suit:

```
Largeur:4
Longueur: 8
périmètre:24
```

Modélisation d'une classe

- Voici comment une classe peut être représentée en suivant les conventions de la modélisation objet **UML** (Unified Modeling Language).
- **On spécifie le nom de la classe, les attributs et les méthodes** dans un rectangle comme ici-bas.
- La visibilité des membres (attributs et méthodes) est représentée par les différents symboles suivants :
  - **private ou −** : le membre est accessible uniquement dans la classe en question.
  - **public ou +** : le membre est accessible dans toutes les autres classes du programme.
  - **protected ou #** : le membre est accessible uniquement dans les classes dérivées de la classe en question ainsi qu'aux classes se trouvant dans le même package. On verra la notion de classe dérivée dans les prochains cours.
  - **package ou ~ ou rien** : le membre est accessible uniquement dans les classes du même paquetage.
- Note: nous verrons plus en détail cette notion de visibilité dans les autres cours.

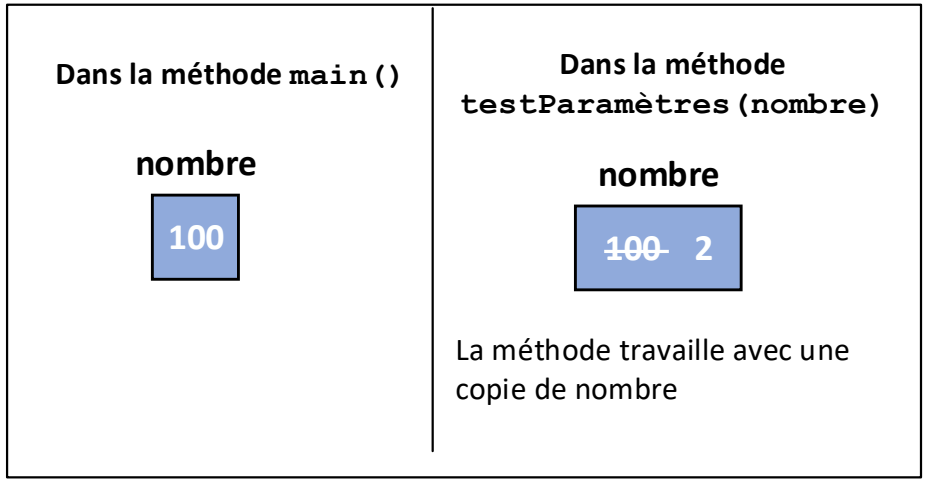


Paramètre par valeur

- Lorsqu'on passe un paramètre de type primitif (`byte, short, int, long, float, double, boolean, char`) à une méthode, ce paramètre est passé par valeur. C'est une copie de cette variable qui est utilisée par la méthode au moment de son appel.
- Par conséquent, **la variable initiale n'est pas accessible (donc non modifiable) par la méthode appelée.**

```
public class TestParametres {

public static void testParametres(int n) {
    n = 2;
    System.out.println("Dans la méthode");
    System.out.println("nombre: " + n);
}
public static void main(String[] args) {
   int nombre = 100;
    System.out.println("Avant l'appel de la méthode");
    System.out.println("nombre: " + nombre);
    testParametres(nombre);
    System.out.println("Après l'appel de la méthode");
    System.out.println("nombre: " + nombre);
}
}
```

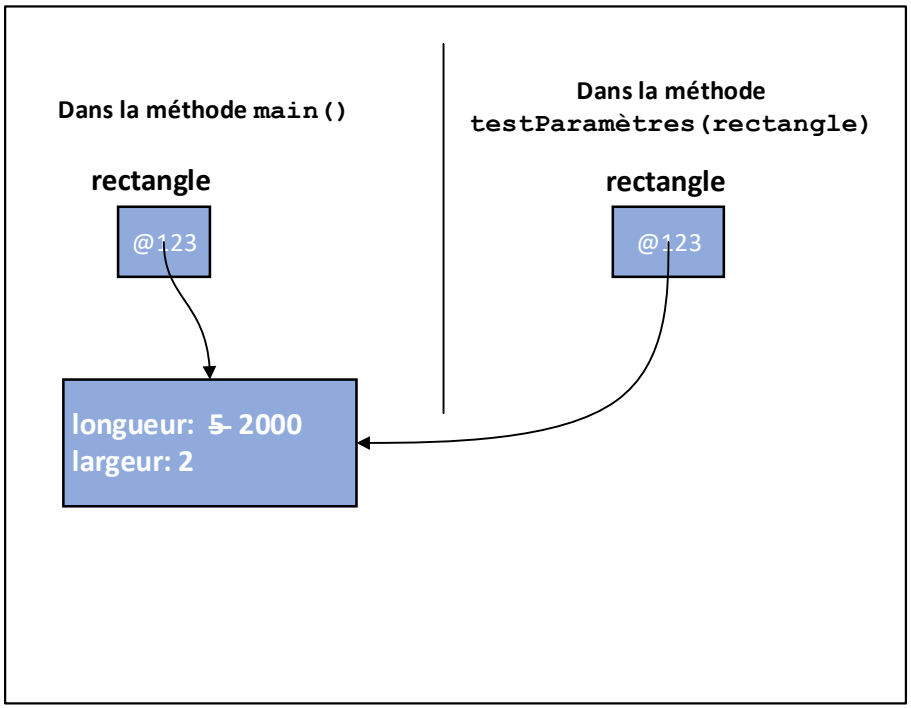
```
Avant l'appel de la méthode
nombre: 100
Dans la méthode
nombre: 2
Après l'appel de la méthode
nombre: 100
```



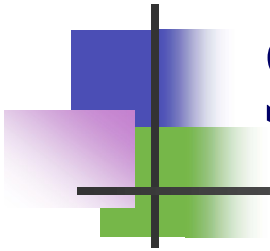
Paramètre par référence

- Lorsqu'on passe un paramètre de type objet à une méthode (Rectangle, tableau,…), ce paramètre est passé par référence: c'est l'adresse de l'objet placé en paramètre lors de l'appel qui est utilisée.
  - Par conséquent, **l'objet initial est accessible (donc modifiable) par la méthode appelée.**

```
public class TestParametres {

public static void testParametres(Rectangle rect) {
   rect.setLongueur(2000);
}

public static void main(String[] args) {
   Rectangle rectangle = new Rectangle(5, 2);
   System.out.println("Avant l'appel de la méthode");
   System.out.println("longueur du rectangle: " +
   rectangle.getLongueur());
   testParametres(rectangle);
   System.out.println("Après l'appel de la méthode");
   System.out.println("longueur du rectangle: " +
   rectangle.getLongueur());
}
}
```




```
Avant l'appel de la méthode
longueur du rectangle: 5
Après l'appel de la méthode
longueur du rectangle: 2000
```



Sources

- Apprendre Java et la programmation orientée objet: https://www.ukonline.be/cours/java/apprendre-java
- Tasso, A. (s.d.). Le livre de JAVA premier langage avec 109 exercices corrigés. Éditions EYROLLES.
- Les images: https://webcourses.ucf.edu/courses/1249560/pages/what-is-object-oriented-programming