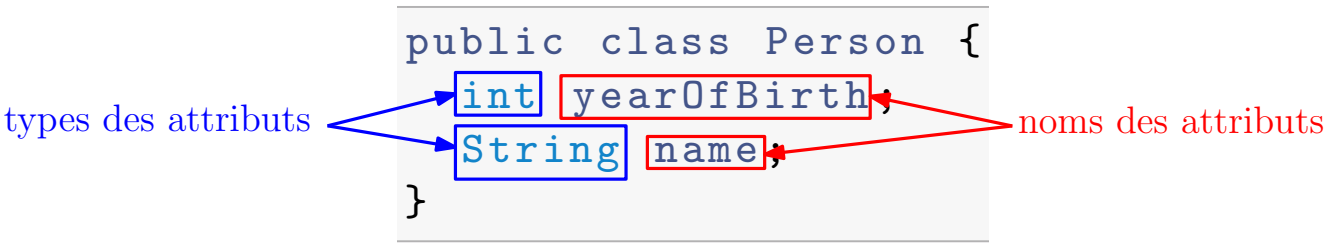
Objets et classes

Dans la programmation orientée objet, un objet sert à modéliser un concept. Par exemple, dans le cas d'un programme gérant des commandes d'articles pouvant être faites par des clients, on modélisera les commandes, les articles et les clients par des objets. Chaque entité, c'est-à-dire chaque commande, client et article correspondra à un objet distinct.

Attributs

Un objet est composé de données qui peuvent être des références vers d'autres objets ou bien des types primitifs tels que des entiers, des nombres à virgule ou des booléens. Les propriétés sont représentées par des variables appelées attributs ou encore variable d'instance. Par exemple, un objet modélisant une personne pourrait avoir comme donnée une année de naissance (nommée `yearOfBirth` de type entier : type primitif `int`) et un nom (nommée `name` de type chaîne de caractères : type `String`). L'ensemble des valeurs des attributs d'un objet constitue l'*état* d'un objet. Par exemple, une personne ayant pour année de naissance `2018` et un nom `"Paul Calcul"` a un état qui correspond à ces deux valeurs. Si on change son nom en `"Jean-Michel Goodenough"`, son état change.

Afin de pouvoir construire un tel objet en java, on doit déclarer une classe qui définit un modèle d'objets. Les objets construits à partir d'une classe sont appelés les *instances* de la classe. Une classe définit donc un type d'objet en spécifiant la structure de ses attributs (mais pas que comme nous allons le voir par la suite). Une classe définit donc un type d'objet. Pour définir une classe nommée `Person` nous permettant de construire les objets représentant des personnes comme décrit ci-avant, on peut utiliser le code suivant :



Pour accéder à un attribut d'un objet, il faut utiliser l'opérateur `.` suivi du nom de l'attribut. Si un objet de type `Person` est stocké dans une variable de nom `somebody`, il suffit d'écrire `somebody.name` pour accéder à son nom.

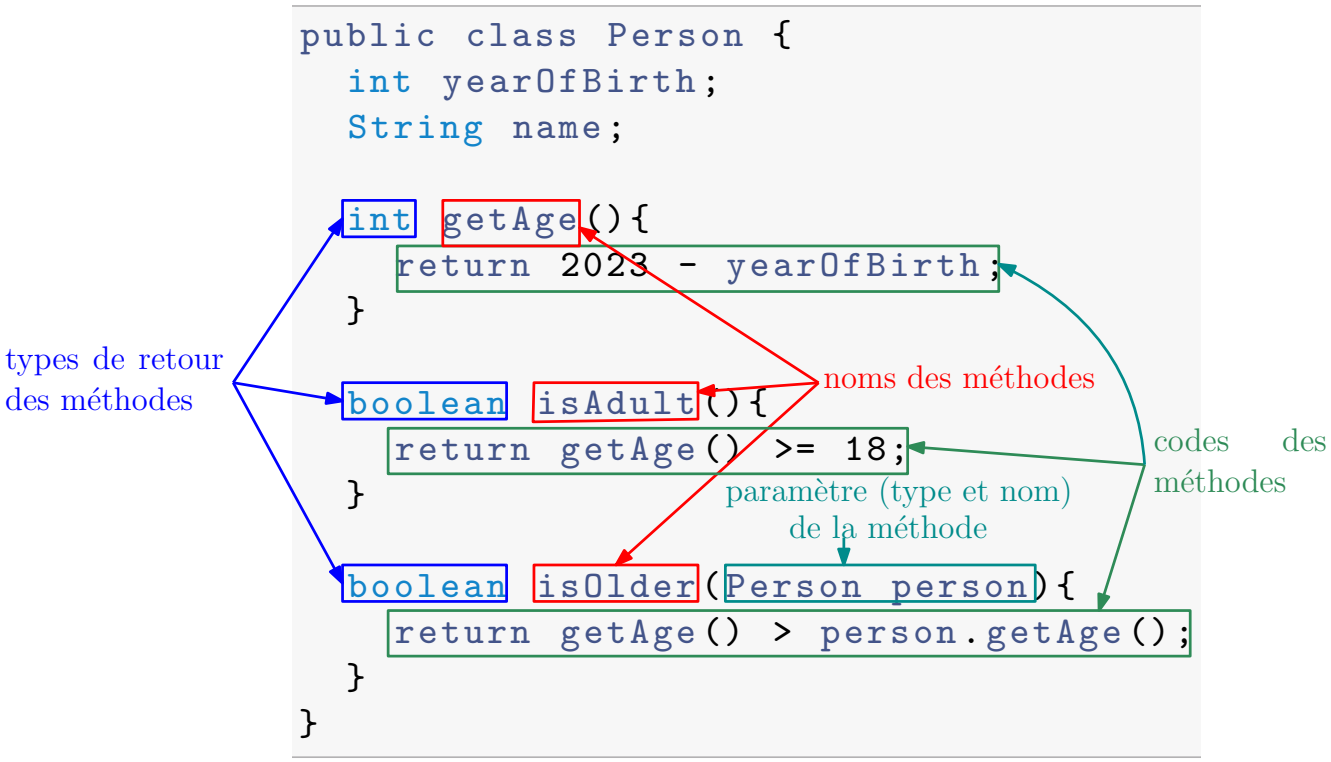
Méthodes

Un objet en plus de sa structure de données (ses attributs) possède aussi un ensemble d'opérations qu'on peut lui appliquer. Ces opérations prennent la forme de méthodes qui sont des fonctions que l'on peut appliquer sur l'objet. Les méthodes définissent le comportement de l'objet (définit ce qui se passe lorsqu'une opération est appliquée sur un objet). Il est à noter que comportement agit sur l'état et l'état influence le comportement.

Comme pour les attributs, les méthodes doivent être déclarées dans la classe de l'objet. Pour la classe `Person` qu'on a déjà défini on peut rajouter :

- une méthode `getAge` permettant de récupérer l'age de la personne (par soucis de simplicité du code on supposera qu'on est en 2023) ;
- une méthode `isAdult` permettant de déterminer si la personne est adulte (au moins 18 ans) ou pas ;
- une méthode `isOlder` permettant de déterminer si une autre personne (donné en paramètre) est plus âgée ou pas.

La déclaration de ces trois méthodes peut se faire avec le code suivant :



Pour appeler (ou invoquer) une méthode d'un objet, il faut utiliser l'opérateur `.` suivi du nom de la méthode avec entre parenthèses les paramètres demandés.

On peut remarquer dans l'exemple les choses suivantes :

- Il est possible d'accéder aux attributs de l'objet avec lequel la méthode est invoquée directement avec leur nom comme `yearOfBirth` dans la méthode `getAge`.
- Il est possible d'invoquer une méthode dans le code d'une méthode : soit directement avec l'objet avec

lequel la méthode originelle a été invoqué (comme `getAge()` pour le code de `isAdult`) ou sur un autre objet (comme `person.getAge()` dans le code de `isOlder`).

Constructeurs

Pour créer un objet, il faut généralement utiliser un constructeur. Chaque appel à un constructeur crée un nouvel objet (instance) qui obéit au modèle défini par la classe du constructeur.

Un constructeur a deux rôles :

1. créer les attributs de l'objet (la structure de l'état) ⇒ réserver l'espace mémoire (automatique en Java) ;
2. donner les valeurs initiales aux attributs (« initialiser l'objet » : code du constructeur).

Chaque classe doit donc définir comment sont initialisés les attributs et par conséquent définir un ou plusieurs constructeurs. Un constructeur en `Java` est définie de manière similaire à une méthode. Il y a néanmoins quelques différences :

— il n'y a pas de type de retour pour le constructeur et
— le nom du constructeur est le nom de la classe (et donc il commence par une majuscule).

Par exemple, une classe `Person` avec deux attributs `yearOfBirth` et `name` peut avoir le constructeur suivant :

```
1  public class Person {
2    int yearOfBirth;
3    String name;
4
5    // Constructeur
6    Person(int yearOfBirth, String name) {
7      this.yearOfBirth = yearOfBirth;
8      this.name = name;
9    }
10 }
```

L'appel au constructeur se fait à l'aide du mot-clé **`new`** suivi du nom de la classe avec entre parenthèses les arguments souhaités : **`new`** `nomDeLaClasse(arguments)`. Pour créer une instance de `Person`, on peut donc utiliser l'instruction suivante : **`new`** `Person("Alice", 2001)`.

L'appel à un constructeur a pour résultat une référence vers l'objet créé. Cette référence est une adresse vers l'identité de l'objet. Elle peut être stockée dans une variable (de type objet). La référence permet d'accéder à l'objet, mais n'est pas l'objet lui-même. Une variable objet contient donc l'information pour accéder à l'objet (et potentiellement le modifier). Deux variables ayant la même référence d'un objet (par exemple après une affectation `a = b;`) pointe vers un objet unique ⇒ toute modification de l'objet est visible via les deux variables.

En Java, si une classe ne définit pas de constructeur, alors il y a un constructeur par défaut (constructeur sans paramètre n'ayant pas d'instruction réservant juste l'espace mémoire pour l'objet).

Il est aussi possible de donner des valeurs par défaut aux attributs lors de leur déclaration via une opération d'affectation. Cette valeur peut éventuellement être remplacée par des affectations dans le constructeur.

Par exemple avec le code ci-dessous, l'appel au constructeur `new Person("Alice")` construit une instance de `Person` avec comme valeur des attributs `1900` (la valeur par défaut) pour `yearOfBirth` et `"Alice"` pour `name`.

```
1  public class Person {
2    int yearOfBirth = 1900;
3    String name;
4
5    Person(String name) {
6      this.name = name;
7    }
8
9    Person(int yearOfBirth, String name) {
10     this.yearOfBirth = yearOfBirth;
11     this.name = name;
12   }
13 }
```

Le comportement du constructeur à deux attributs ne change pas et il n'y a aucun problème à avoir deux constructeurs, car le compilateur peut déterminer suivant le nombre et le type des arguments, lequel est appelé.