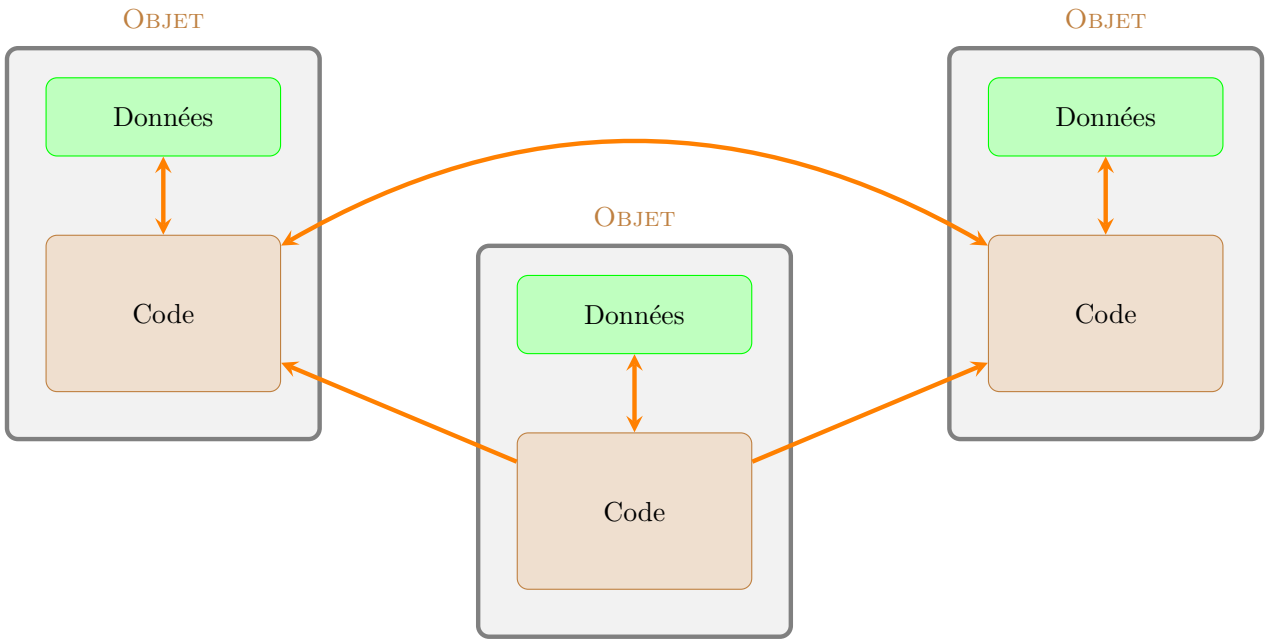
Classes / Objets

Le concept d'objet

La programmation orientée objet repose sur le concept d'**objet**.

Un objet contient des **données** et du **code**.

Un programme orienté objet consiste en interactions entre objets.

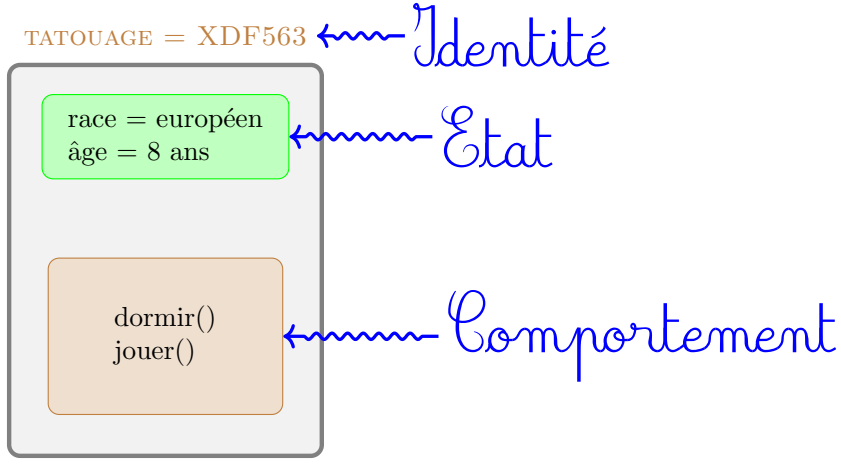


Le concept d'objet

Un objet est caractérisé par

- son **identité**, c'est-à-dire une information **unique** identifiant l'objet.
- un **état**, c'est-à-dire les données qu'il contient.
- un **comportement**, c'est-à-dire les actions possibles de l'objet.

Exemple



Le concept de classe

Les objets partageant les mêmes types de données et capables des mêmes actions définissent une **classe** d'objet.

Les objets seront alors appelés des **instances** de la classe.

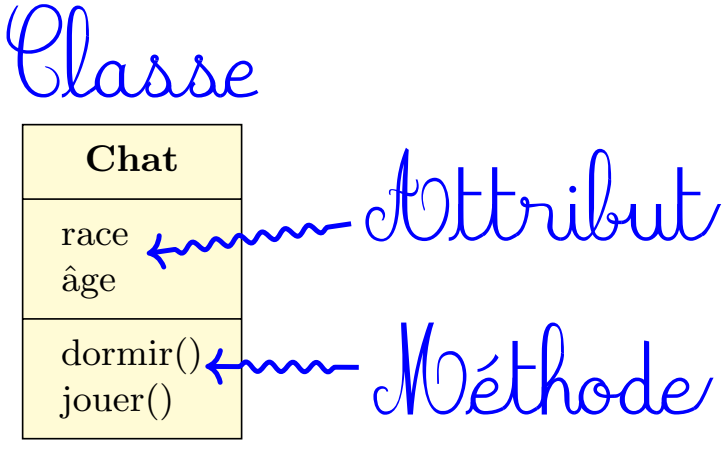
Exemple

Le concept de classe

La définition d'une classe d'objets indique

- la liste les variables qu'il faudra créer pour stocker les données des objets de la classe, qu'on appelle **attributs**.
- la liste les fonctions qui décriront le comportement des objets de la classe, qu'on appelle **méthodes**.

Exemple



Définir une classe d'objets

Pour définir une classe d'objets, on utilise le mot-clef `class` :

```
class Chat
```

> Le nom d'une classe commence toujours par une **lettre majuscule**

Définir une classe c'est définir un **nouveau** type d'objets : les objets `Chat` .

Définir les attributs / méthodes d'une d'une classe

Entre accolades, on donne la déclaration des attributs et méthodes des objets de la classe :

```
public class Chat {
  // Attributs
  String race;
  int age;

  // Méthodes
  void dormir() {
    System.out.println("le chat dort");}
  void jouer() {
    System.out.println("le chat joue");}
}
```

Le mot clé **public** placé devant le mot-clé **class** indique que les objet `Chat` pourront être utilisés par n'importe quel objet.

Types de données de base

Les types de données de base sont

- **boolean** pour les booléens qui ne pourra prendre que deux valeurs **true** ou **false** .
- **byte** , **short** , **int** et **long** pour les entiers.
- **float** et **double** pour les nombres décimaux.
- **char** pour les caractères.

Ces types de données sont parfois appelés les types **primitifs**.

> Dans la pratique, pour représenter des nombres, on utilise le plus souvent que les types **int** et **double** . Cela évite en général les problèmes de conversion.

Les classes de base

En plus des types de données de base, il existe des classes de base comme la classe `String` qui a été conçu pour faciliter la gestion des chaînes de caractères.

Bien qu'étant aussi des objets, les objets `String` sont créés simplement en indiquant la chaîne de caractères entres guillemets (caractère ") :

```
"Licence MIAGE" // Objet String
```

Il existe d'autres classes de bases comme les classes, appelées **classes enveloppes**. A chaque type de données de base, il est associé une classe : `Integer` pour le type **`int`** , `Double` pour le type **`double`** , etc.

Et quelques autres classes dont une dont nous reparlerons plus tard : la classe `Object` .

Encapsulation

L'**encapsulation** est le processus qui vise à définir le niveau d'accès des attributs d'un objet par les autres objets.

Pour mettre en œuvre ce mécanisme, le langage JAVA™ propose des mots-clés appelés **modificateurs d'accés**.

Dans un premier temps, nous n'en introduirons que deux :

- **public** : les attributs publics et méthodes publiques sont accessibles par toutes les instances.
- **private** : les attributs privés ne sont accessibles que par les instances de la classe

Encapsulation

```
class Chat {
  // Attributs privés
  private String race; // chaîne de caractères
  private int age;  // entier

  // Méthodes publiques
  public void dormir() {
    System.out.println("le chat dort");}
  public void jouer() {
    System.out.println("le chat joue");}
}
```

- Les attributs `race` et `age` ne sont accessibles que par les instances de la classe `Chat`.
- Les méthodes `dormir()` et `jouer()` sont accessibles par tous les objets et non seulement les instances de la classe `Chat`.

Définir une classe

Pour résumer à ce stade, définir une **classe** d'objets c'est

- déclarer les attributs des objets de la classe
- définir les méthodes des objets de la classe
- fixer le niveau d'accès de chacun des attributs / méthodes, c'est-à-dire indiquer ceux qui seront inaccessibles par les instances des **autres** classes (**private** , caché) et ceux qui seront accessibles par les instances des autres classes (**public** , visible).

Interface d'un objet

L'ensemble des attributs et méthodes publics d'un objet s'appelle son **interface**.

Interface de la classe Chat

void dormir()

faire dormir le chat

void jouer()

faire jouer le chat

Pour utiliser un objet `Chat`, il suffit de connaître l'interface de l'objet.

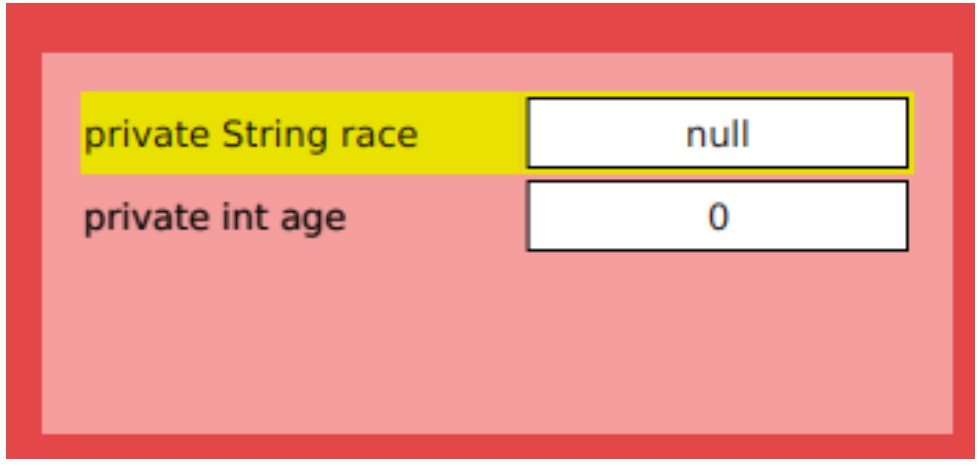
Créer une instance

Pour créer une instance de la classe `Chat`, il suffit d'écrire `new Chat()` :

- L'opérateur `new` alloue **dynamiquement**, (à l'exécution) un espace mémoire pour stocker les valeurs des attributs de l'instance ainsi que la création de liens qui permettront à l'instance d'appeler les méthodes de la classe.
- L'opérateur `new` retourne l'identité de l'objet, information **unique** pour chaque objet.

État d'une instance

On appelle **état** d'une instance le tuple des valeurs de ses attributs.
L'état de l'instance créée par l'instruction `new Chat()` est :



Le langage prévoit une initialisation par défaut des attributs de la classe `Chat` :

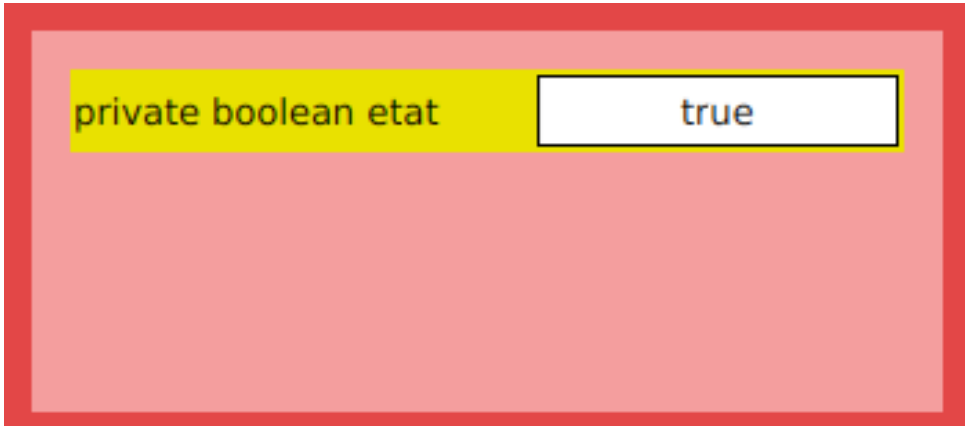
- Le type de la variable `age` étant un type `int`, elle est initialisée par défaut à `0`.
- Le type de la variable `race` étant une classe, la classe `String`, elle est initialisée par défaut à `null`.

Initialisation des attributs

Si on préfère donner une autre valeur que la valeur par défaut, il suffit de l'indiquer lors de sa déclaration.

```
public class Lampe
{
    /* variable d'instance : initialement une lampe
     est allumée */
    private boolean estAllumee = true;
 }
```

L'état de l'instance créée par l'instruction `new Lampe()` est :



Constructeurs

On peut aussi choisir de paramétrer l'état initial d'une instance en définissant ce qu'on appelle un **constructeur** :

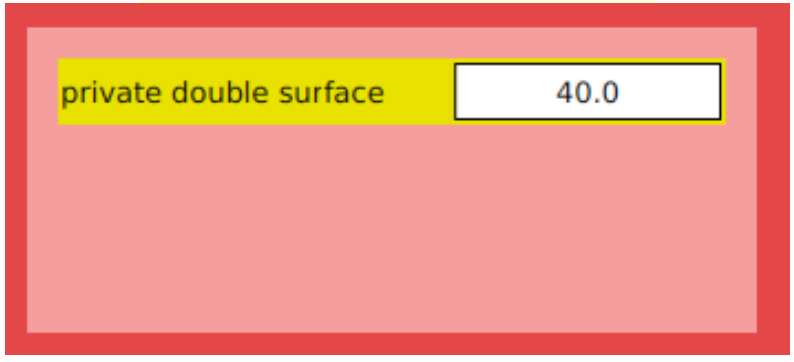
```
public class Logement
{
    private double surface;

    // Constructeur
    public Logement(double valeur) {
      surface = valeur; }
}
```

Le mot-clé **public** indique que n'importe quel objet pourra utiliser ce constructeur paramétré.

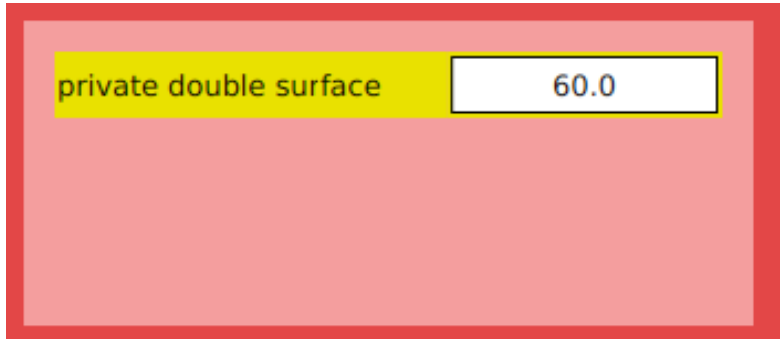
Constructeurs

Lors de la création d'une instance de `Logement`, on doit donc indiquer la valeur initiale de l'attribut `surface` entre parenthèses :

new Logement(40.0)



new Logement(60.0)

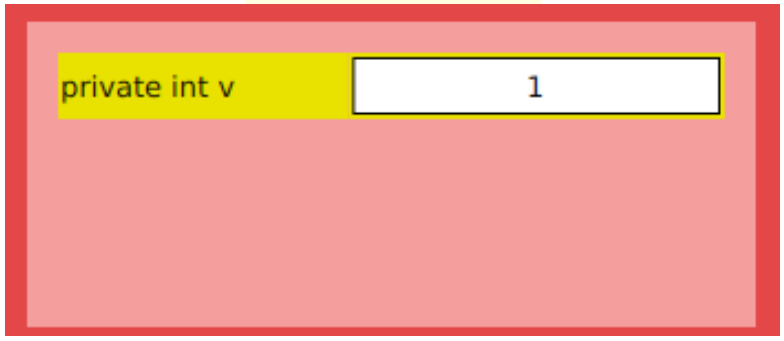


Constructeurs

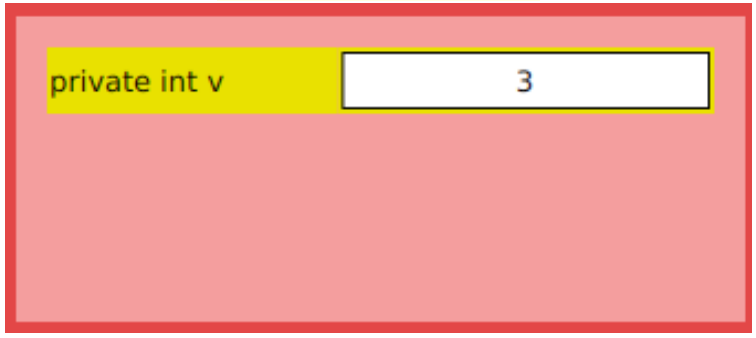
Il est possible de définir plusieurs constructeurs au sein d'une classe :

```
class Dice
{
    private int v;

    /* Constructeurs */
    public Dice() { v = 1; }

    public Dice(int x) { v = x; }
  }
```

new Dice()



new Dice(3)



Créer une instance

En résumé, il est très important de définir les états initiaux **possibles** des instances d'une classe en définissant autant de constructeurs que nécessaire.

Si on ne définit aucun constructeur dans une classe, le seul état initial possible d'une instance de la classe est formé des valeurs par défaut ou des valeurs données à la déclaration.

Réutiliser les objets

Deux appels successifs `new Chat()` créent deux objets **distincts**, c'est-à-dire d'identités différentes.

```
new Chat() // un objet Chat
new Chat() // un autre objet Chat
```

Pour pouvoir réutiliser plusieurs fois un objet, le langage java fournit un nouveau type de variables : les **références**.

- Une référence est une variable dont le type est le nom d'une classe.
- Une référence permet d'enregistrer l'identité d'un objet.

Réutiliser un objet

```
Chat r = new Chat();
```

La variable `r` est une référence de type `Chat` qui contient l'identité d'un objet `Chat`, c'est-à-dire une instance de la classe `Chat`.

Il est alors possible d'appeler à travers la référence `r` les méthodes de l'objet `Chat` :

```
r.dormir();
r.jouer();
r.dormir();
```

Références

En résumé,

- Une référence est une variable dont le type est une classe
- On peut modifier la valeur d'une référence et enregistrer successivement l'identité de différents objets.

```
Chat r;
r = new Chat(); // un premier objet Chat
r = new Chat(); // un deuxième objet Chat
```

- On peut recopier la valeur d'une référence dans une autre référence

```
Chat r1, r2;
r1 = new Chat();
r2 = r1;
```

Les deux références contiennent l'identité d'un même objet `Chat`.

Références

On peut tester l'égalité de deux références les opérateurs relationnels `==` ou `!=` :

```
Chat r1 = new Chat, r2 = r1, r3 = new Chat()
r1 == r2 // true
r1 != r2 // false
r1 == r3 // false
r1 != r3 // true
```

Autoréférence

Lors de la création d'un objet, il est enregistré l'identité de l'objet dans l'objet lui-même dans une référence appelée **autoréférence** : `this`.
Cette référence n'est accessible que par les méthodes de l'objet.

```
public class Entier
{
  private int valeur;

  public Entier(int valeur) {this.valeur = valeur;}
}
```

L'autoréférence `this` est requise dans l'exemple ci-dessus car l'attribut de l'objet et l'argument du constructeur portent le même nom : `valeur`.