Java
Utilisation de classes en Java

Objectifs

- Notion de Programmation Orientée Objet (POO)
- Distinguer une classe et une instance

Programmation Orientée Objet

- Regrouper des variables qui ont du sens ensemble
  - Par exemple : les variables année, mois, jour dans une classe « Date »

- Associer les fonctions qui manipulent ces valeurs
  - Par exemple : la fonction permettant de savoir si une année est bissextile, la fonction permettant de connaître le nombre de jours écoulés depuis le début de l'année

Programmation Orientée Objet

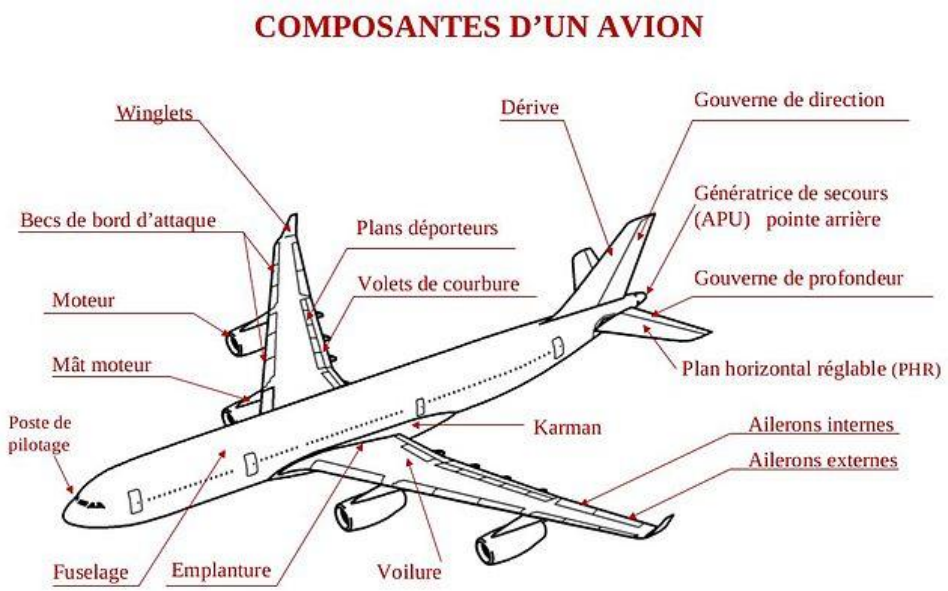
- Une classe, c’est quoi ?

« Un ensemble de valeurs et d’opérations logiquement liées entre elles »

Programmation Orientée Objet

- Une classe, c'est quoi ?

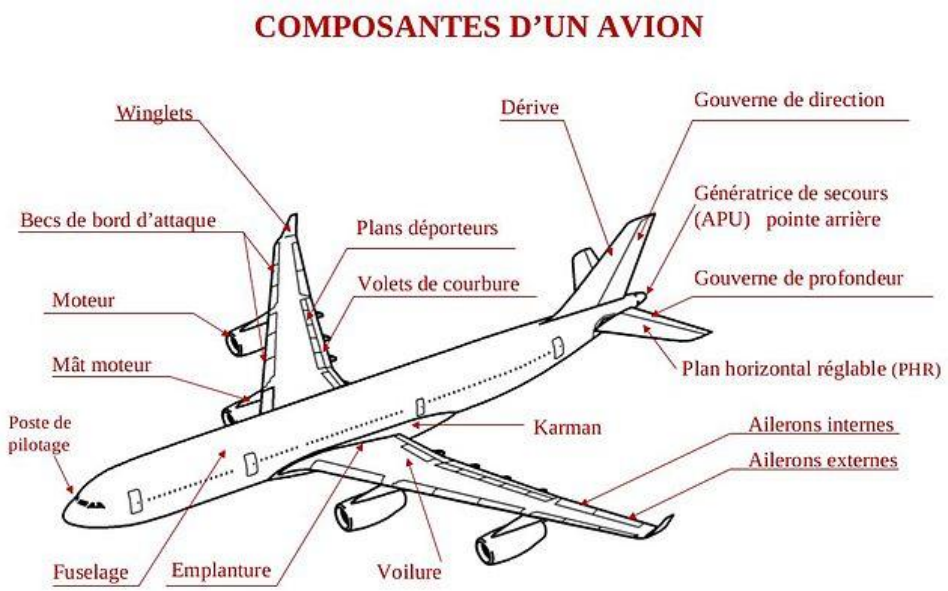
« Un ensemble de valeurs et d'opérations logiquement liées entre elles »



Programmation Orientée Objet

- Une classe, c'est quoi ?

« Un ensemble de valeurs et d'opérations logiquement liées entre elles »



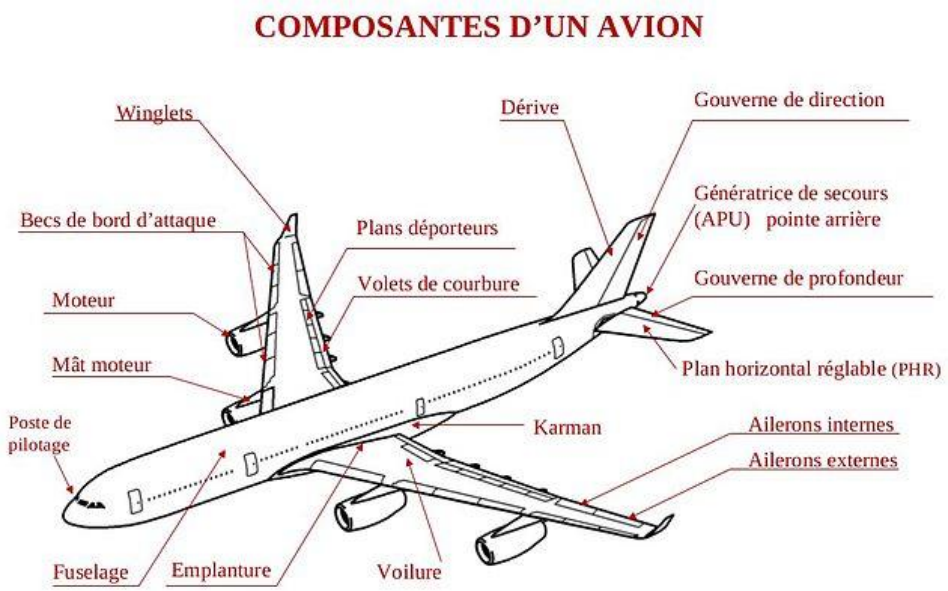
- Et une instance ?

« Un élément issu de cet ensemble »

Programmation Orientée Objet

- Une classe, c’est quoi ?

« Un ensemble de valeurs et d’opérations logiquement liées entre elles »



- Et une instance ?

« Un élément issu de cet ensemble »

Programmation Orientée Objet

- En schéma

Des instances

Une classe

une instance est appelée par l'instruction new

Programmation Orientée Objet

- Le constructeur

  - Le constructeur est une sorte de fonction
  - Il alloue la place nécessaire pour contenir la nouvelle instance
  - Il initialise cette instance

Comme les tableaux, une instance est un type « référence » !

CAD, une modif se fait sur l'originale et non sur une copie

Programmation Orientée Objet

- Utilisation d'un constructeur de la classe Locale

```java
package demos;

import java.util.Locale;

public class DemoLocale {

    public static void main(String[] args) {
        // Déclaration d'une variable de type Locale
        Locale localeFR;

        // Création d'une nouvelle instance de Locale en utilisant le constructeur
        localeFR = new Locale("FR", "fr");

        // Ou bien en une seule ligne :
        Locale localeEN = new Locale("EN", "en");
    }
}
```

On met toujours des majuscule lorsqu'on crée uen classe

Programmation Orientée Objet

- Les méthodes d'instance

  - Les méthodes sont les différentes opérations proposées par la classe
    - Par exemple : savoir si une année est bissextile, connaître le nombre de jours écoulés depuis le début de l'année, pour la classe Date

Programmation Orientée Objet

- Quelques méthodes de la classe Locale

```
package demos;

import java.util.Locale;

public class DemoLocale {

    public static void main(String[] args) {
        Locale localeFR = new Locale("FR", "fr");
        Locale localeEN = new Locale("EN", "en");

        System.out.println(localeFR.getCountry());
        System.out.println(localeFR.getDisplayName());
        System.out.println(localeEN.getDisplayLanguage(localeFR));
    }
}
```

Programmation Orientée Objet

- Quelques méthodes de la classe Locale

```
package demos;

import java.util.Locale;

public class DemoLocale {

    public static void main(String[] args) {
        Locale localeFR = new Locale("FR", "fr");
        Locale localeEN = new Locale("EN", "en");

        System.out.println(localeFR.getCountry());
        System.out.println(localeFR.getDisplayName());
        System.out.println(localeEN.getDisplayLanguage(localeFR));
    }
}
```

constructeur de la classe en jaune

Locale
+ language : String + country : String + variant : String
+ Locale(language : String, country : String) + getCountry () : String + getDisplayName() : String + getDisplayLanguage (locale : Locale) : String

En bleu des méthodes associées (sortes de fonction spécifiques à une classe

Programmation Orientée Objet

- Quelques méthodes de la classe Locale

Locale
+ language : String + country : String + variant : String
+ Locale(language : String, country : String) + getCountry () : String + getDisplayName() : String + getDisplayLanguage (locale : Locale) : String

Un diagramme de classe

```
package demos;

import java.util.Locale;

public class DemoLocale {

    public static void main(String[] args) {
        Locale localeFR = new Locale("FR", "fr");
        Locale localeEN = new Locale("EN", "en");

        System.out.println(localeFR.getCountry());
        System.out.println(localeFR.getDisplayName());
        System.out.println(localeEN.getDisplayLanguage(localeFR));
    }
}
```

Programmation Orientée Objet

La surcharge :

- Plusieurs méthodes peuvent porter le même nom au sein d'une classe) condition qu'elles puissent être différenciées
  - Le nombre d'argument est différent
  - Le type des arguments est différent

Programmation Orientée Objet

La surcharge :

- Plusieurs méthodes peuvent porter le même nom au sein d'une classe) condition qu'elles puissent être différenciées
  - Le nombre d'argument est différent
  - Le type des arguments est différent

Locale
+ language : String + country : String + variant : String
+ Locale(language : String) + Locale(language : String, country : String) + Locale(language : String, country : String, variant : String) + getCountry () : String + getDisplayName() : String + getDisplayLanguage () : String + getDisplayLanguage (locale : Locale) : String

Programmation Orientée Objet

La surcharge :

- Plusieurs méthodes peuvent porter le même nom au sein d'une classe) condition qu'elles puissent être différenciées
  - Le nombre d'argument est différent
  - Le type des arguments est différent

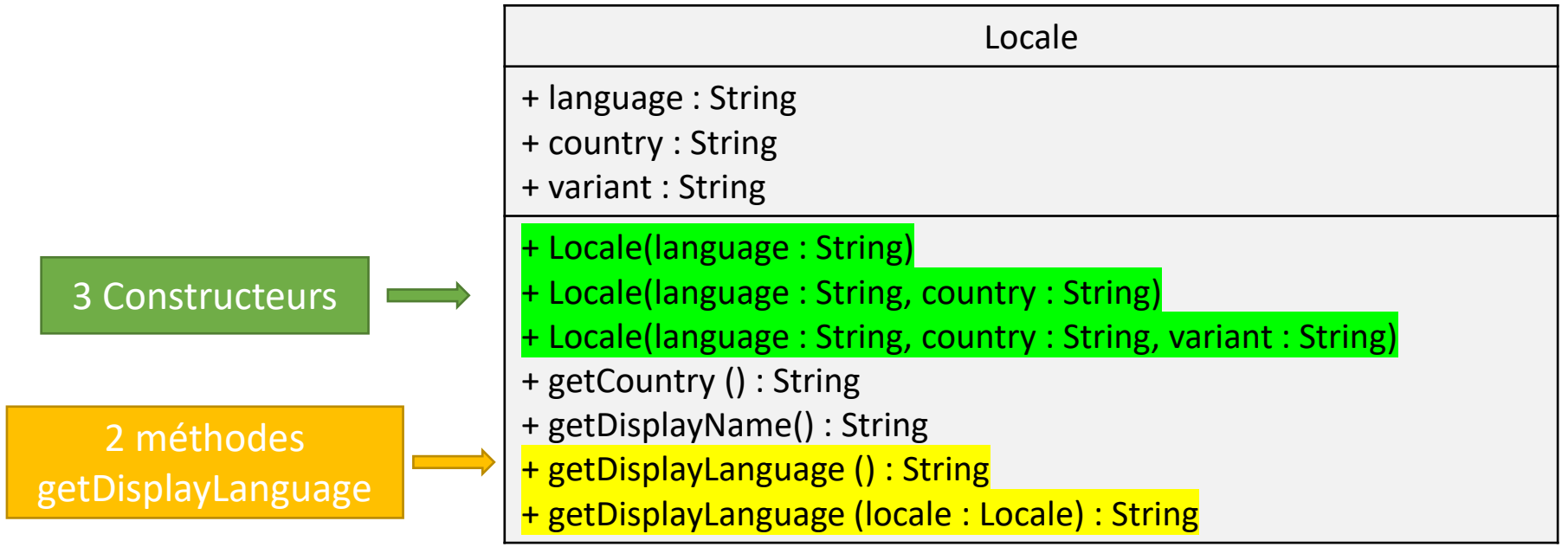
3 Constructeurs →

Locale
+ language : String + country : String + variant : String
+ Locale(language : String) + Locale(language : String, country : String) + Locale(language : String, country : String, variant : String) + getCountry () : String + getDisplayName() : String + getDisplayLanguage () : String + getDisplayLanguage (locale : Locale) : String

Programmation Orientée Objet

La surcharge :

- Plusieurs méthodes peuvent porter le même nom au sein d'une classe) condition qu'elles puissent être différenciées
  - Le nombre d'argument est différent
  - Le type des arguments est différent



Programmation Orientée Objet

La surcharge :

- Plusieurs méthodes peuvent porter le même nom au sein d'une classe) condition qu'elles puissent être différenciées
  - Le nombre d'argument est différent
  - Le type des arguments est différent

```
public static void main(String[] args) {
    Locale locale = new Locale("DE");
    Locale localeEN = new Locale("EN", "en");
    Locale localeFR = new Locale("FR", "fr", "ma");

    System.out.println(localeFR.getCountry());
    System.out.println(localeFR.getDisplayName());
    System.out.println(localeEN.getDisplayLanguage(localeFR));
}
```

ici c'est "EN","gb"

ex.:français de france de marseille

Locale
+ language : String + country : String + variant : String
+ Locale(language : String) + Locale(language : String, country : String) + Locale(language : String, country : String, variant : String) + getCountry () : String + getDisplayName() : String + getDisplayLanguage () : String + getDisplayLanguage (locale : Locale) : String

Programmation Orientée Objet

Les éléments d'instances et les éléments de classe

- Elément d'instance :
  - individuel, propre à chaque instance
    - couleur, numéro de série, prix…
- Elément de classe :
  - collectif, concerne l'ensemble des instances
    - prochain numéro de série, nombre d'instances
  - commun, la même chose quelque soit l'instance
    - nombre de places, langues disponibles

Programmation Orientée Objet

Les éléments d'instances et les éléments de classe

```java
public class DemoLocaleStatic {

    public static void main(String[] args) {
        // getAvailableLocales est une méthode de classe
        Locale[] locales = Locale.getAvailableLocales();
        for (Locale current : locales) {
            // getDisplayLanguage est une méthode d'instance
            System.out.println(current.getDisplayLanguage());
        }
    }
}
```

permet d'afficher toutes les langues supportées par java

La classe String

- La classe String est immuable
  - Aucune méthode ne modifie l'instance
  - Les méthodes censées modifier la valeur retournent en réalité une nouvelle instance de String

```java
package demos;

public class DemoString {
    public static void main(String[] args) {
        String nom = "Dupont";
        String nomUp = nom.toUpperCase();
        String nomD = nom.replace('T', 'D');
        System.out.println(nom);
        System.out.println(nomUp);
        System.out.println(nomD);
    }
}
```

Les méthodes de String ne modifie jamais un String.
String est l'une des seules classes qu'on instancie sans "new".
En revanche, de manière générale, une méthode peut très bien modifier une instance.

Atelier 2
Langues

Java
Utilisation de classes en Java