Concepts de l'orienté objet

Branches SRT et ISI – Unité de valeur LO02

Guillaume Doyen

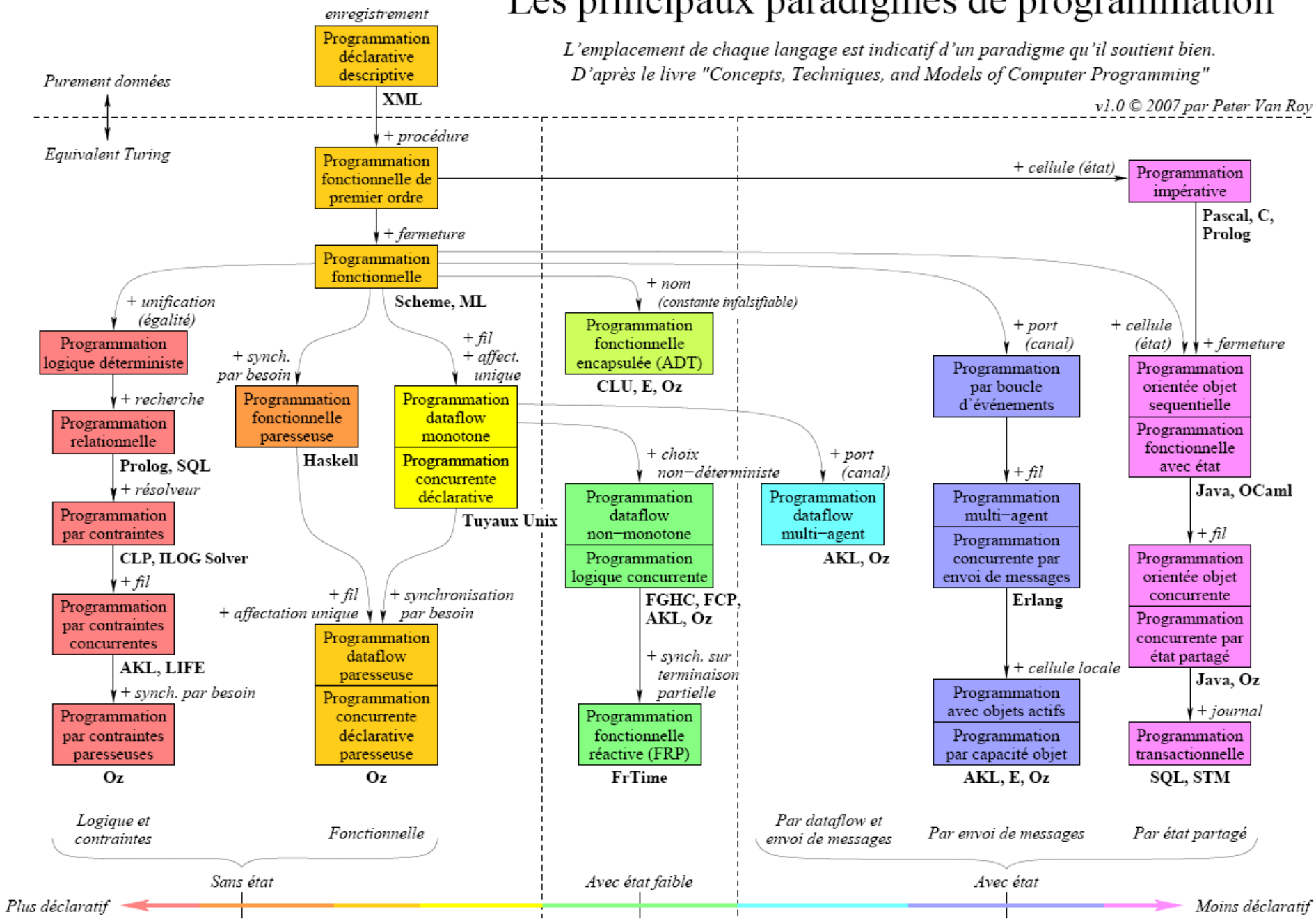
Contact : guillaume.doyen@utt.fr

Plan du cours

- **Introduction**
  - Les paradigmes de programmation
- **L'objet**
  - Principe
  - Les classes
- **Les concepts associés**
  - L'encapsulation
  - L'héritage
  - Le polymorphisme

Les principaux paradigmes de programmation

L'emplacement de chaque langage est indicatif d'un paradigme qu'il soutient bien.
D'après le livre "Concepts, Techniques, and Models of Computer Programming"



La programmation orientée objet

- **Principe**
  - Un ensemble d'objets informatiques
    - Forment un programme
    - Communiquent entre eux par le biais de messages
    - Changent d'état
- **Exemples**
  - La bibliothèque
    - Le programme est constitué d'objets : livres, bibliothèque adhérent, bibliothécaire, …
    - Les objets communiquent : un adhérent peut emprunter un livre, le bibliothécaire gère la bibliothèque, …
  - Partie de poker
    - Le programme est constitué de cartes, tas de cartes, joueurs, …

Plan du cours

- **Introduction**
  - Les paradigmes de programmation
- **L'objet**
  - Principe
  - Les classes
- **Les concepts associés**
  - L'encapsulation
  - L'héritage
  - Le polymorphisme

L'objet

- **Définition**
  - Un objet est une abstraction d'une « donnée du monde réel »
    - Il peut représenter un objet concret
      - Ex : une voiture, une personne, une maison, …
    - Mais aussi toute forme de concept abstrait
      - Ex : une stratégie, une transaction, …
  - Il est caractérisé par
    - Son identité
      - Il possède un identifiant attribué par le système
    - Son état
      - C'est la liste des valeurs de variables qui le caractérisent
    - Son comportement
      - L'ensemble des opérations applicables à l'objet

L'objet

- **Exemple : la voiture**
  - Identifiant

```
maVoiture, laVoitureDePaul, ...
```

  - Etat

```
carburant = 23 // Litres
vitesse = 54 // km/h
kilomètres = 34789 // km
couleur = rouge
```

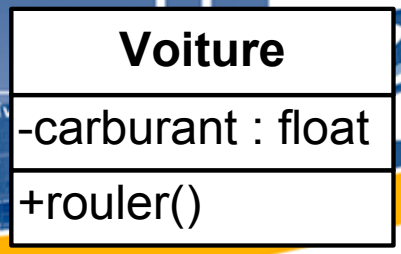
  - Operations

```
rouler(), stopper(), accélérer(), freiner(), tournerAGauche(),
tournerADroite(), reviser(), mettreDuCarburant(), ...
```

L'objet

- **Terminologie**
  - L'état
    - On parle d'attribut, de propriété
  - Le comportement
    - On parle de méthode, d'opération, d'envoi de message
  - Ces termes sont employés indifféremment par la suite

La classe 2/



- **Définition**
  - La classe représente le modèle qui décrit les objets du programme
  - Tous les objets d'un même type peuvent être décrits par une classe
    - Ex : voitures, personnes, …
    - La classe représente donc un type
      - Au même titre que les types primitifs (entier, flottant, caractère, …)
  - La classe est définie par
    - Les propriétés communes à ses objets (attributs et opérations)
      - On les appelle les membres
    - Un mécanisme d'instanciation qui permet de créer des objets ayant ces propriétés

La classe 2/

- **Classes et objets**
  - La classe est un modèle de description
    - Analogie avec un moule de fonderie
  - Les objets sont des instances de classes
    - Des réalisations
  - Le mécanisme qui permet de créer un objet s'appelle l'instanciation
- **Qu'est ce qu'un programme OO ?**
  - La définition de classes
  - Au sein de ces classes
    - L'instanciation d'objets
    - L'envoi de messages

Plan du cours

- **Introduction**
  - Les paradigmes de programmation
- **L'objet**
  - Principe
  - Les classes
- **Les concepts associés**
  - L'encapsulation
  - L'héritage
  - Le polymorphisme

L'encapsulation 1/

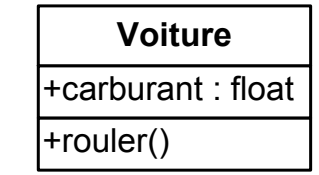
- **Vue interne/externe des objets**
  - Un objet dispose d'une mécanique interne et offre une vue à l'extérieur
  - Ex : une voiture
    - offre une vue à ses utilisateurs (carrosserie, habitacle, …) mais possède son fonctionnement interne propre (moteur, transmission, …)
    - Impossibilité pour l'utilisateur de régler le carburateur, de changer les rapports de boite, …
  - Nécessité de différentier la vision interne d'un objet de celle offerte à ses utilisateurs
  - C'est l'encapsulation
    - Concept essentiel dans la POO

L'encapsulation 2/

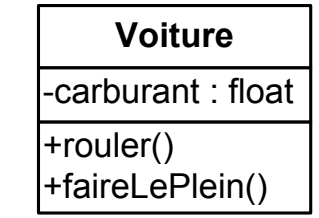
- **Définition**
  - Ajout de modificateurs de visibilité aux membres d'une classe (attributs et méthodes)
  - Au minimum, une classe définit la visibilité publique et privée pour ses membres
    - Publique : le membre est accessible depuis l'extérieur
      - Possibilité par un objet extérieur de modifier la valeur du membre, si c'est un attribut
    - Privée : le membre n'est pas accessible depuis l'extérieur
      - On a dans ce cas encapsulé le membre dans la classe
- **Usage**
  - La majorité des attributs sont privés
    - Pour éviter tout changement de leur valeur non contrôlé par l'objet à qui ils appartiennent
  - La majorité des méthodes sont publiques
    - Pour permettre la modification des attributs sous contrôle

L'encapsulation 3/

- **Exemple**
  - Une voiture est caractérisée par sa quantité de carburant dans le réservoir
    - La quantité de carburant est donc un attribut
    - Si l'attribut est publique, tout objet extérieur à la classe peut changer la valeur de quantité de carburant
      - Vider le réservoir, mettre -7 litres, mettre 34567124 litres, …
    - Si l'attribut est privé, le seul moyen de changer la quantité de carburant est d'appeler une méthode publique de l'objet qui va réaliser cette opération
      - Cette méthode peut vérifier la cohérence de la nouvelle quantité de carburant
      - L'objet garde le contrôle de la valeur de ses attributs



Sans encapsulation



Avec encapsulation

L'encapsulation

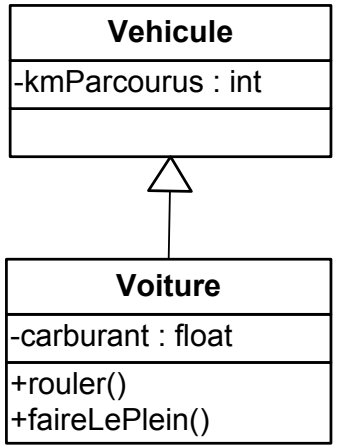
- **Conséquence**
  - L'objet qui encapsule des membres masque une partie de soi à l'extérieur
    - Il offre une interface à l'extérieur
  - C'est une abstraction de son implantation
    - L'utilisateur n'a pas besoin de savoir comment sont réalisées les opérations d'un objet qu'il utilise
    - Il accède à un service offert par cet objet

L'héritage 1/

- **Objectif**
  - Organiser les classes selon le principe de généralisation ou spécialisation
  - Réduire la quantité de code produit en factorisant le code dans les classes les plus générales possibles
  - Accroit la modularité du code
- **Exemple**
  - Une voiture est un véhicule ; une moto est un véhicule
    - Une voiture est un véhicule au sens général
    - Une voiture est un véhicule spécial
  - On peut placer dans la classe véhicule les membres qui sont communs à la voiture et la moto

L'héritage 2/

- **Définition**
  - L'héritage est le mécanisme de transmission des membres d'une classe à ceux dont les classes héritent
  - On peut mettre en place autant de niveaux d'héritage que souhaité
- **Terminologie**
  - Classes dont une autre hérite : super classe, classe ancêtre, classe mère, …
  - Classe qui hérite d'une autre : sous classe, classe fille, …
- **Impact sur les types**
  - Une classe est de son propre type mais aussi du type de ses classes mères
    - Ex : Une voiture est un véhicule
    - Contre ex : Un véhicule n'est pas forcément une voiture

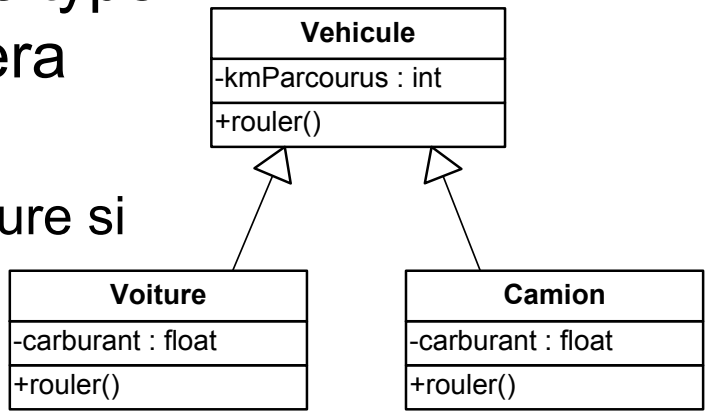


Le polymorphisme 1/

- **Définitions**
  - Redéfinition : *redéfinition d'une opération définie dans une classe mère, dans une classe fille avec un code différent*
  - Polymorphisme : *faculté d'une opération à pouvoir s'appliquer à des objets de classes différentes*
  - Liaison dynamique : *mécanisme de sélection de code d'une opération à l'exécution en fonction de la classe d'appartenance de l'objet receveur*

Le polymorphisme 2/

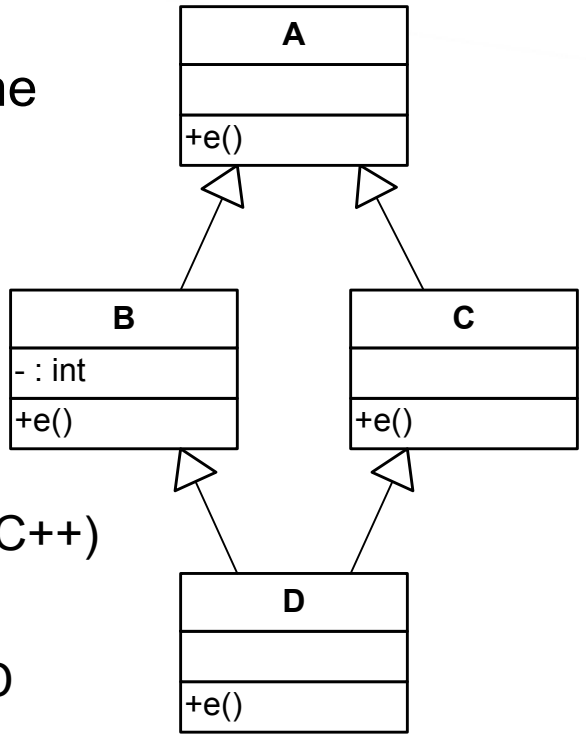
- **Exemple**
  - Lorsqu'un objet extérieur envoie le messages rouler() à des instances des classes Voiture et Camion, c'est le type effectif de l'instance qui déterminera dynamiquement le code exécuté
    - Opération définie dans la classe Voiture si l'instance est de type Voiture



- **Conclusion**
  - L'appelant envoie un message et son traitement diffère en fonction du type effectif de l'objet

Héritage multiple

- **Héritage simple et multiple**
  - Héritage simple : une classe n'hérite que d'une super-classe
  - Héritage multiple : une classe peut hériter de plusieurs super-classes
- **Quelques remarques**
  - L'héritage multiple est très controversé
    - Seuls certains langages le proposent (par ex. C++)
  - Problème de l'héritage « en losange »
    - Soit A une classe. B hérite de, C hérite de A. D hérite de B et C
    - Si B et C définissent une opération de même nom : comment y accéder depuis D ?



Synthèse

- **Introduction**
  - Les paradigmes de programmation
- **L'objet**
  - Principe
  - Les classes
- **Les concepts associés**
  - L'encapsulation
  - L'héritage
  - Le polymorphisme