MODÉLISATION ET INVERSION EN GÉOPHYSIQUE
4 - Programmation orientée objet

Bernard Giroux
(bernard.giroux@ete.inrs.ca)

Institut national de la recherche scientifique
Centre Eau Terre Environnement

Version 1.0.0
Hiver 2017

Introduction

Motivation

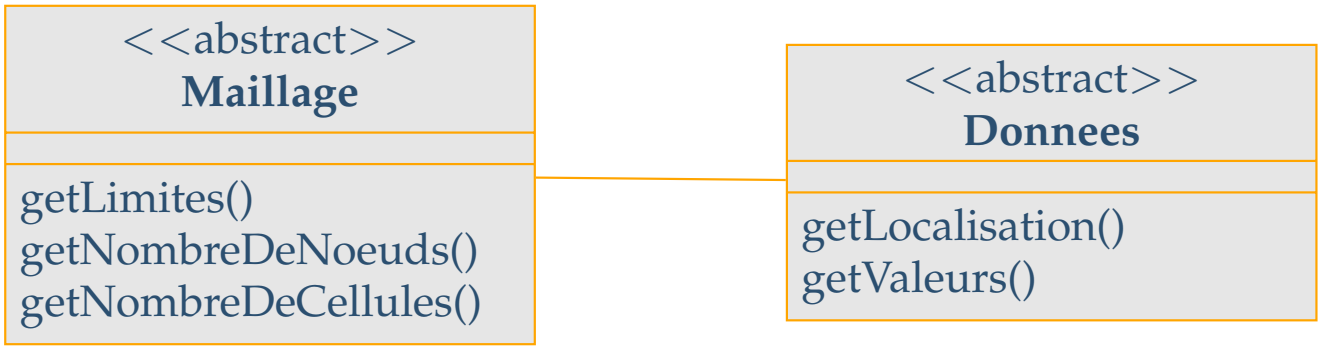
- Le développement de logiciels d'envergure moyenne à grande pose plusieurs défis sur le plan de la programmation :
  - structure cohérente entre les éléments ;
  - maintenance de cette structure ;
  - organisation du code.
- La programmation orientée objet (POO) offre plusieurs avantages pour faciliter ces développements :
  - modularité ;
  - abstraction ;
  - encapsulation ;
  - sûreté ;
  - productivité et réutilisabilité.

Modularité

- Division d'un logiciel en modules indépendants ;
- Ces modules regroupent les données et les opérations *associées* ;
  - Exemple : un module `Maillage` contenant
    - les coordonnées des noeuds,
    - les indices des noeuds formant les faces,
    - une fonction pour importer un maillage à partir d'un fichier,
    - une fonction pour retourner les limites du maillage,
    - une fonction pour retourner le nombre de faces ou de voxels,
    - etc
- Classe : formalisme permettant de représenter le module.
- Objet : représentation du module, *instance* de la classe qui existe dans la mémoire de l'ordinateur.
- La modularité permet une organisation hiérarchique du code.

Abstraction

- Abstraction :
  - La spécification des fonctionnalités d'une classe est définie ;
    - Classe dite *abstraite*
    - Les fonctionnalités constitue l'*interface*
  - La *représentation* (l'implémentation) n'est pas définie ;
    - l'implémentation se fait dans une classe *dérivée*.
- Il ne peut exister une instance d'une classe abstraite.
- L'abstraction permet
  - de définir de façon générale les relations entre les classes,
  - permet de définir une structure cohérente pour un programme/logiciel.

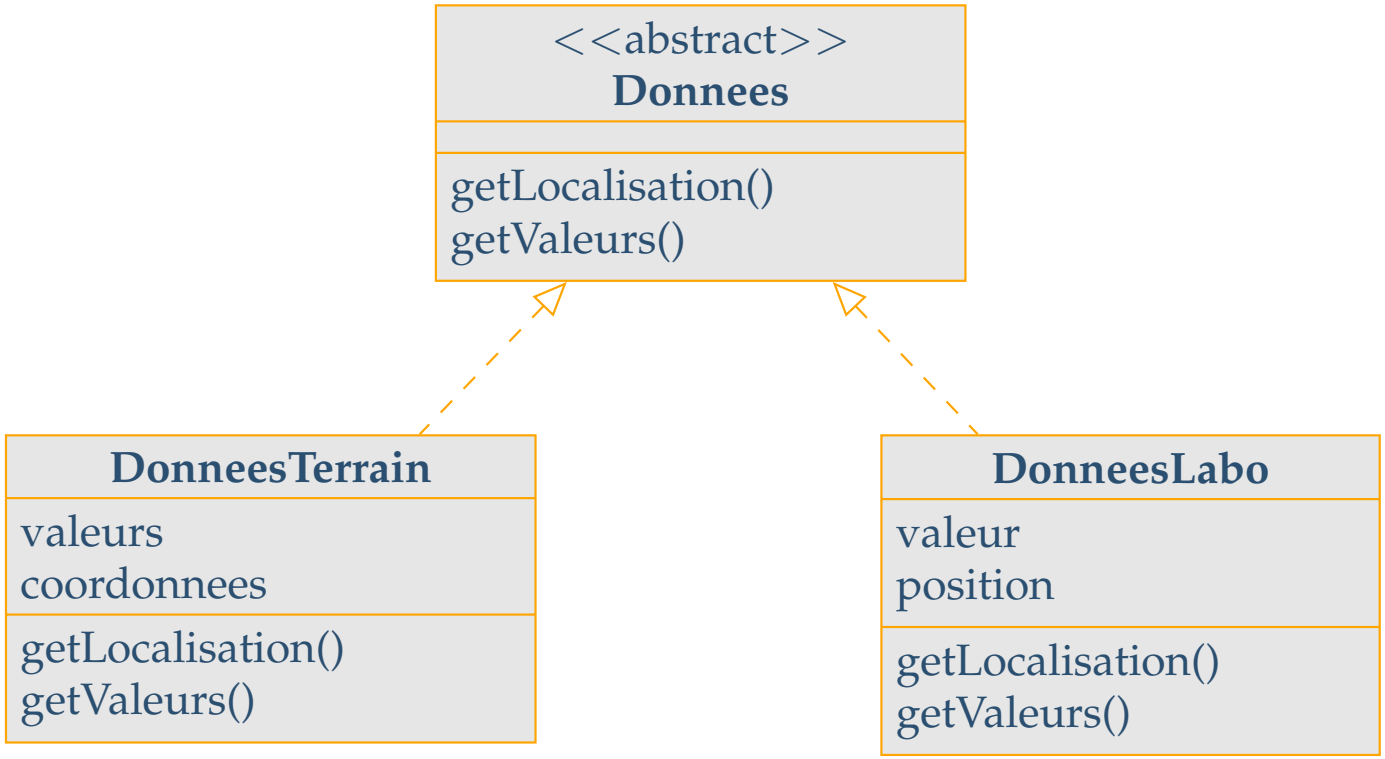


Encapsulation et sûreté

- Encapsulation : fonctionnalité du language qui permet la modularité ;
  - Une classe permet de lier des variables, ou attributs, à des fonctions, ou méthodes
  - Un object contient les valeurs des variables, les fonctions appelées par l'objet produisent un résultats propre à ses valeurs.
- Permet de changer librement l'implémentation.
- Possibilité de restreindre l'accès direct aux données ;
  - Des variables *privées* ne sont acessibles que par les méthodes de la classe.
  - Tenter d'accéder aux variables privées produit une erreur.
  - Permet de produire un code plus "sûr".

Productivité et réutilisabilité

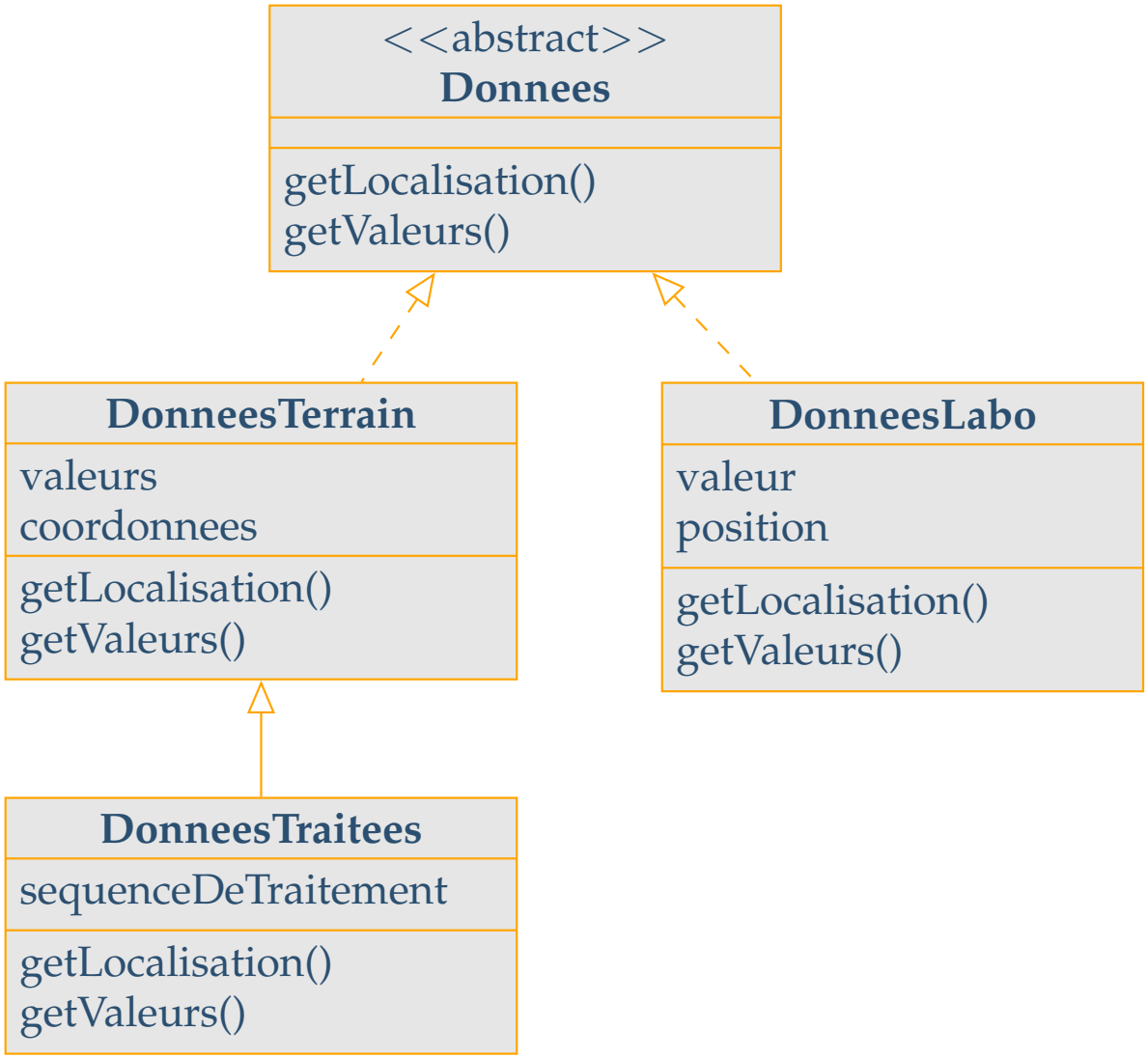
- La POO favorise *en principe* la productivité et la réutilisabilité
  - Travail en équipe : la modularité simplifie l'organisation du code et sa gestion et peut améliorer la productivité.
  - L'abstraction et l'*héritage* permettent, dans une certaine mesure, la réutilisabilité du code.



Héritage

- Chaque classe possède des caractéristiques (attributs et méthodes) qui lui sont propres.
- On dit d'une classe dérivée qu'elle hérite de la classe *mère*
  - la classe *fille* peut alors utiliser les caractéristiques de la classe mère ;
  - le code écrit pour la classe mère *n'a pas* à être écrit à nouveau pour la classe fille ;
  - une classe fille *peut* réimplémenter une méthode de la classe mère.
- Propriétés de l'héritage :
  - Transitivité : si B hérite de A et si C hérite de B alors C hérite de A
  - Non réflexif : une classe ne peut hériter d'elle-même
  - Non symétrique : si A hérite de B, B n'hérite pas de A
  - Sans cycle : il n'est pas possible que B hérite de A, C hérite de B et que A hérite de C.

Héritage



Polymorphisme

- Avec le polymorphisme, une interface est partagée par des classes différentes ;
  - e.g. : multiplication définie pour des scalaires, des matrices ou des vecteurs.
- Sortes de polymorphisme :
  - *Ad hoc* : interface implémentée par plusieurs fonctions de même nom ayant des arguments différents ;
  - paramétré : interface implémentée par une seule fonction prenant en arguments un type *générique* pour chaque entité (templates en C++) ;
  - par sous-typage : classes filles possédant chacune son implémentation d'une méthode de la classe mère.

Références

Références

- Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley