DIAGRAMMES UML

420-V31-SF – Programmation de Jeux Vidéo III

fparadis@cegep-ste-foy.qc.ca

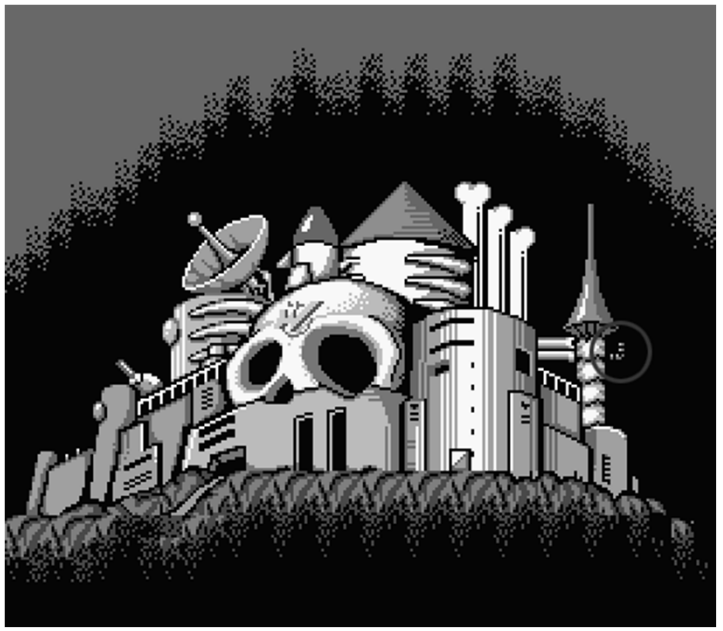
420-V31-SF – Programmation de Jeux Vidéo III

Diagramme de classe UML

fparadis@cegep-ste-foy.qc.ca

Avant de coder, planifier

- Dr Wily vous le dit: avant de vous lancer dans la construction de quoi que ce soit, il vous faut un plan.



Avant de coder, planifier

- Sans un plan, votre construction risque fort de ressembler à…ceci

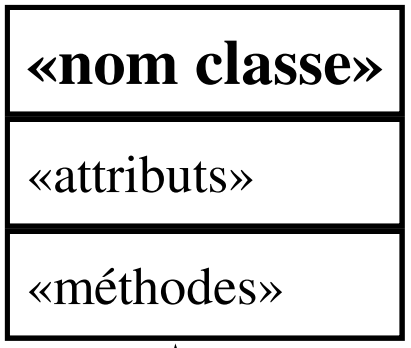
Diagramme de classe UML : Votre plan

- Donc avant de construire un système complexe, vaut mieux planifier.
- En Programmation Orienté Objet, il s'agit du diagramme de classe UML.
  - *1- Réfléchir aux classes que votre système va contenir.*
  - *2- Y placer des attributs et des méthodes*
  - *3- Faire les relations entre les classes*

Diagramme de classe UML : Votre plan

- Nous allons nous concentrer sur la technique, la systématisation viendra avec la pratique (on fera des cas)
- https://en.wikipedia.org/wiki/Class_diagram
- Allez lire ça, sans blague, tout y est.

Notation pour la classe



Section réservée au **nom** de la classe (caractères gras)

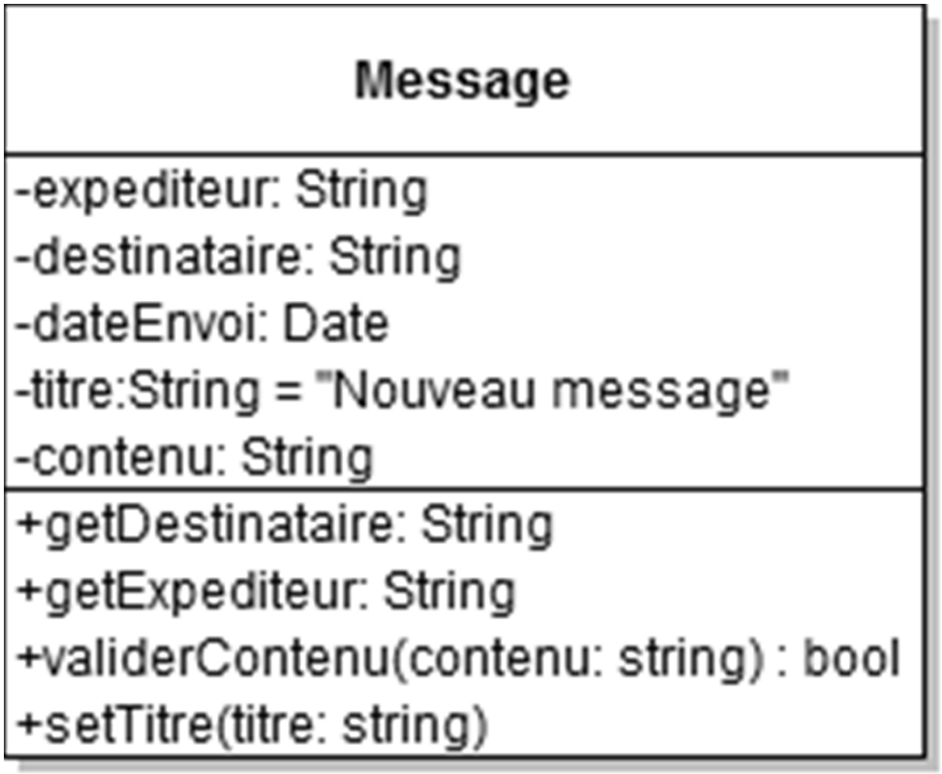
Section réservée aux **attributs** de la classe (avec classement par **visibilité**: publique, protégée ou privée)

Section réservée aux **méthodes** de la classe (avec classement par **visibilité**: publique, protégée ou privée)

(Pour le moment, considérez le mot-clé protected équivalent parfait de private)

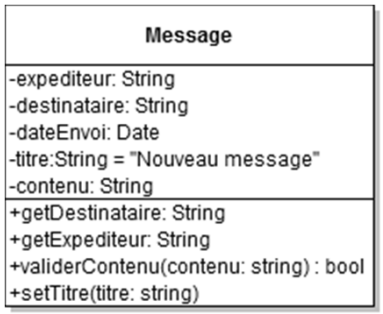
Notation pour la classe

Exemple d'une classe Message



Les attributs de la classe

- Un attribut a la syntaxe suivante:
  - `visibilite nom: type = valeur_initiale`



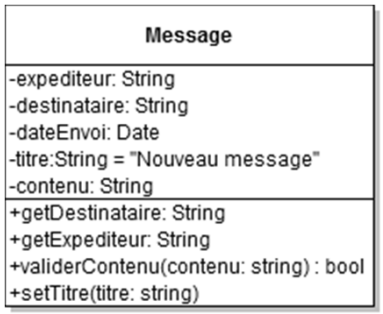
- La visibilité peut être:

```
+   publique: accès à tous.
#   protégée: accès limité à la classe elle-même
    et aux héritiers de la classe.
-   privée: accès limité à la classe elle-même.
```

- Le nom est l'identifiant de l'attribut.
- Le type est le type de donnée.
- La valeur initiale est la valeur à la création d'un attribut (si désiré, peut-être omise).

Les méthodes de la classe

- Un attribut a la syntaxe suivante:
  - `visibilite nom (paramètre): type_retour`



- La visibilité peut être:

```
+    publique: accès à tous.
#    protégée: accès limité à la classe elle-même
     et aux héritiers de la classe.
-    privée: accès limité à la classe elle-même.
```

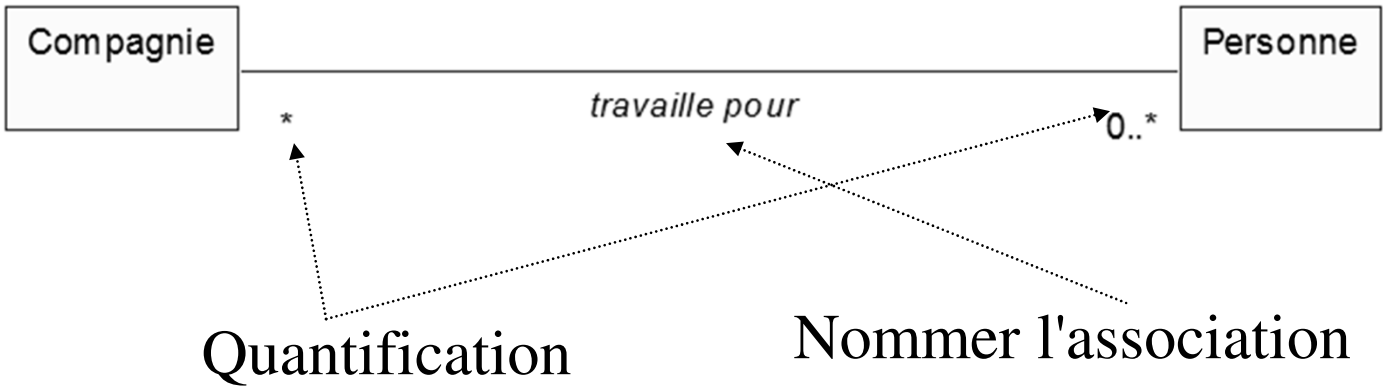
- Le nom est l'identifiant de la méthode.
- Les paramètres, notés -> (nom: type, nom: type…)
- Le type de retour, si méthode void alors on inscrit rien.

Collections

- Quand un attribut d'une classe est un tableau, peu importe le nombre de dimensions, un vecteur, une pile, une file, un graph, bref, peu importe la manière de classer une collections du même type d'objets, on fait suivre au type d'objet les braquettes d'un tableau -> `[]`
- Ex: un tableau de no de téléphones où on peut rejoindre une personne.
  - `nosTelephone: String[]`
- La liste des joueurs d'une équipe de hockey
  - `joueurs: Joueur[]`
- Dans un jeu de stratégie le graph des liens entre les différents points statégiques
  - `Liens: SratPoints[]`

Relations entre les classes

- Associations



  - *Relation faible*
  - *Un objet est lié à un autre de manière arbitraire*
  - *Utilisé comme attribut par une autre classe*
  - *Flèche utilisateur vers utilisé*
  - *Si les deux objets se réfèrent mutuellement, alors flèche double ou pas de flèche*
  - *On indique la cardinalité*

Multiplicité des relations

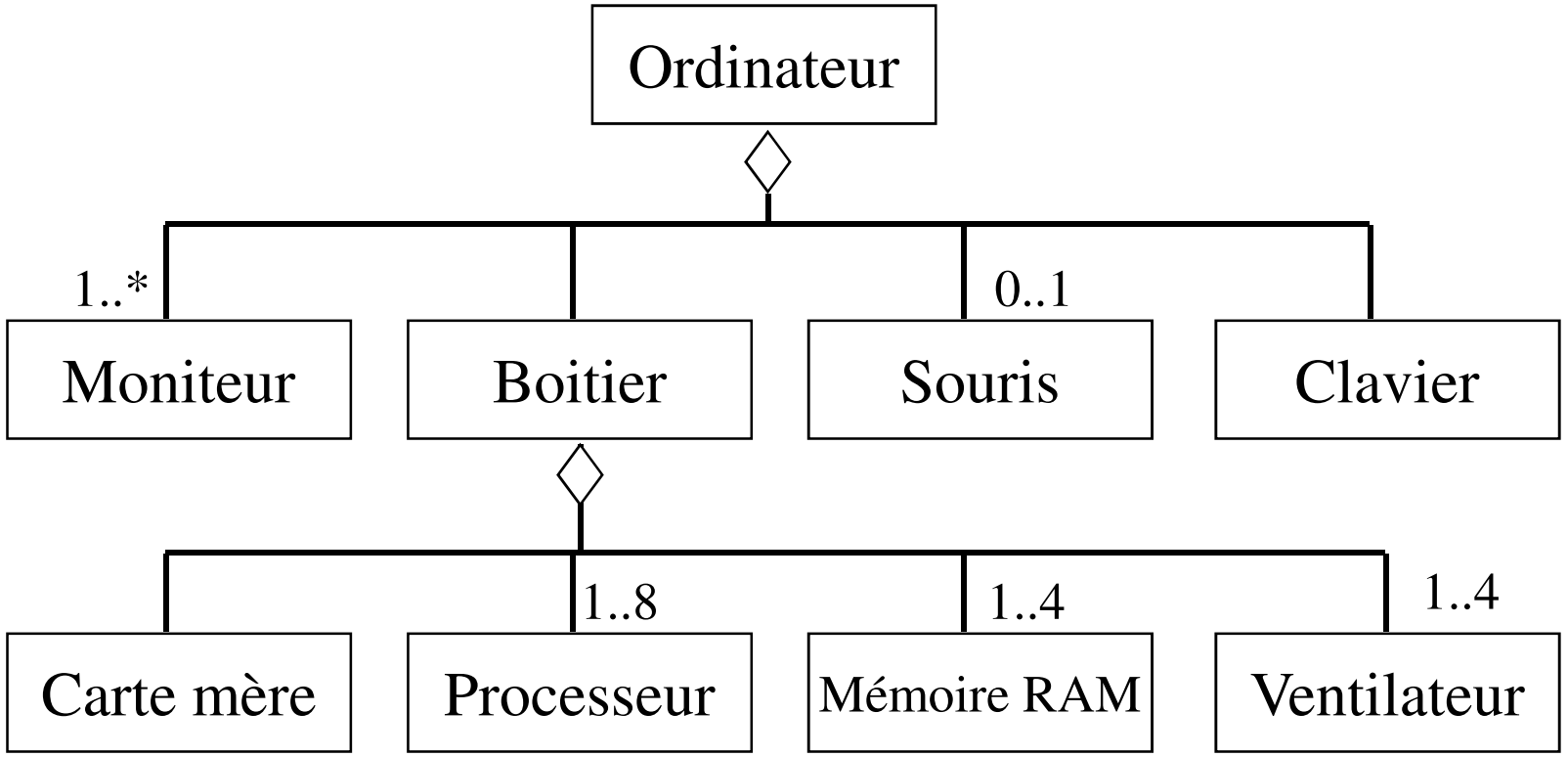
0..1	0 ou 1, préférable à 0,1
1	exactement 1,
	exactement 1,
0..*	0 à l'infini
*	0 à l'infini
1..*	1 à l'infini
1..6	1 à 6
1..3,7..10,15	1 à 3, 7 à 10 et 15

Composition

Relation forte

- Collection d'objets
- Si la collection est détruite, son contenu l'est aussi ou alors l'objet existe toujours mais n'a plus beaucoup de sens seul.
- Relation toujours de 1 et 0..*
- Représenté par un losange du coté de la collection.
- Ex: Paquet de cartes.

Composition



Composition

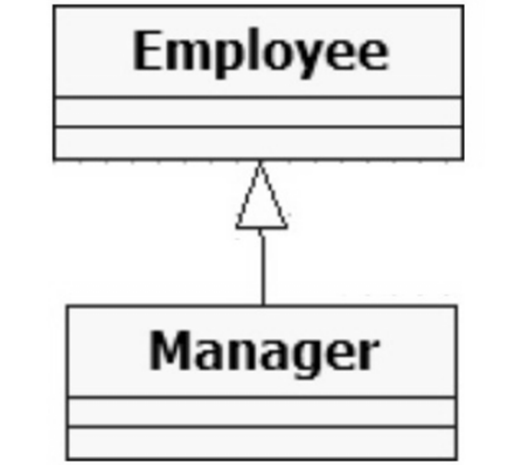
Attention

- Si la relation est une composition, il est ASSUMÉ que la classe composante a une collection des objets liés
- Dans ce cas il n'est pas obligatoire d'illustrer la collection.

Héritage

Relation très forte: EST_UN (IS_A)

Par contraste: Association et Composition A_UN



Héritage « EST_UN »

La flèche entière va de la sous-classe à la super-classe.

Notez bien

- Dans les faits, dans la composition, nous regroupons deux types de relation: La composition et l'agrégation.
- La distinction entre les deux n'est pas toujours évidente.
- Nous y reviendrons plus tard quand nous serons plus habitués avec UML

Outils de modélisation UML

- Visio
  - Pas le plus simple d'utilisation
  - Fait pour faire des schémas en général. On peut faire du UML avec, mais pas nécessairement optimisé pour.
  - Pas d'exportation du code
- Visual Studio
  - Pas le plus simple d'utilisation
  - Exporte le diagramme en code, mais seulement en C#
- VioletUML
  - Très simple d'utilisation, un vrai charme.
  - Fait pour faire du UML.
  - Pas d'exportation du code
  - http://sourceforge.net/projects/violet/files/violetumleditor/2.1.0/

Outils de modélisation UML

- StarUML
  - Assez simple d'utilisation
  - Fait pour faire du UML.
  - Excellente exportation du code en C++
  - License d'utilisation (essai illimité)
  - http://staruml.io/download

- Autre solution?
  - Si vous en connaissez? Aussi bien que StarUML et gratuit? je suis preneur!

Description de système

- Dans tous les travaux que vous aurez à faire. Vous devrez faire des descriptions de système.
- Ce ne seront pas des cas complets (un cas complet a toutes les méthodes et les attributs en place) mais des analyse préliminaires.
- Vous devrez présenter différentes classes, les relations entre celles-ci, ainsi que les attributs et méthodes que les classes auront selon vous à ce stade-ci.
- Devront être remises rapidement, afin d'avoir un plan de développement de votre projet ainsi que de faire valider votre plan.

Analyse de cas

- Les exercices que nous allons faire en UML seront des analyses de cas.
- On vous soumet une situation et vous devez en ressortir un schéma UML.
- **Analyse simple:** Vous ne mettez dans le UML que les attributs et méthodes que la situation décrit directement.
- **Analyse complète:** On ne place dans le système que les classes décrites dans la situation, par contre on extrapole un peu afin de faire de chaque classe une classe la plus complète possible, c'est-à-dire y placer les get, set et méthodes privées qui pourraient être nécessaires.

Exercice 1

- Analyses de cas
- Nous ferons un cas exemple ensemble