Table des matières

Diagramme de classe et d'objet UML 1
EnvironnementVisual Paradigm 1
Avant de commencer 2
Le diagramme de classe 6
Ajouter une classe sur un diagramme. 7
Ajouter des attributs 9
Ajouter une constante 10
Ajouter une opération (équivalent d'une méthode en Java) 12
Affichage UML 14

Diagramme de classe et d'objet UML

EnvironnementVisual Paradigm

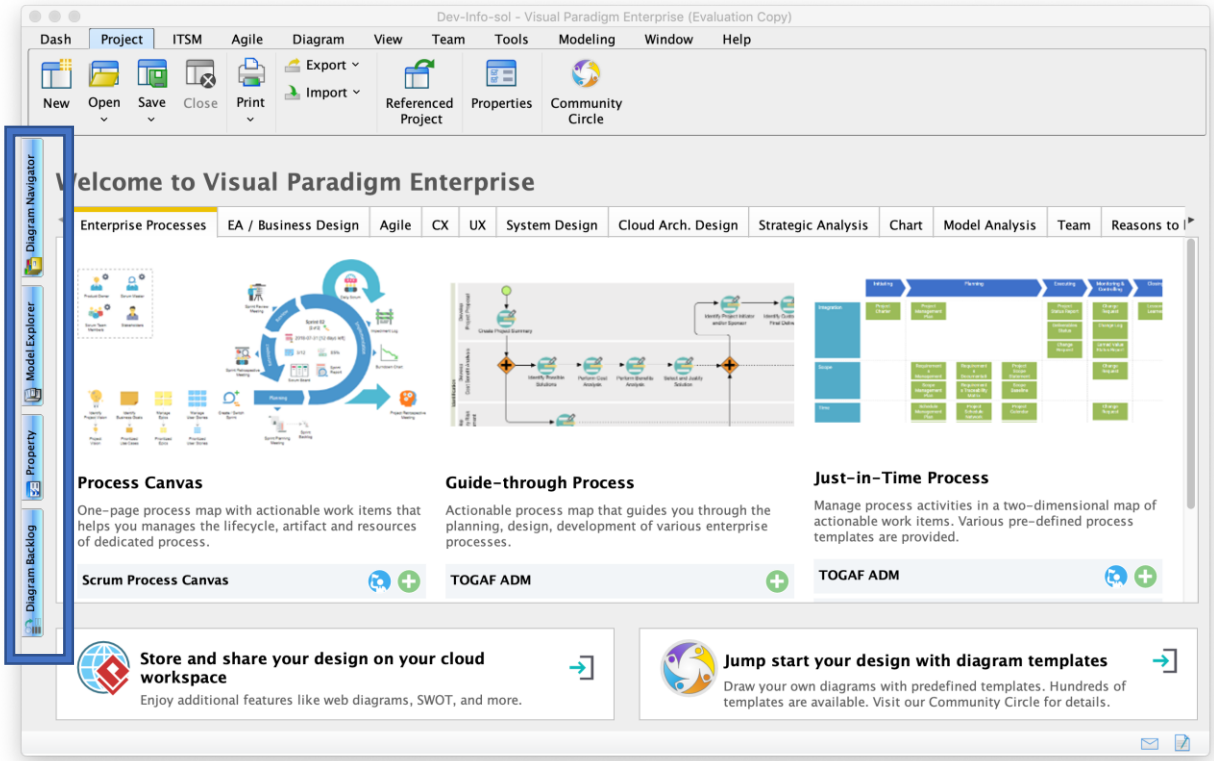
Le logiciel VisualParadigm (**VP**) version 16.2 est un logiciel puissant pour créer des diagrammes UML des diagrammes d'entité ERD et d'autres schémas d'analyse. Il est l'un des plus utilisés et il offre une version communautaire gratuite et le Cégep de Limoilou possède une version de site éducative. Si vous voulez l'installer sur votre PC, vous devez utiliser la version qui est sur le réseau du cégep :

Commun/cours/!logiciel/!Visual_Pradigm/16

Vous n'aurez qu'à suivre les instructions qui se trouvent dans le fichier qui se trouve dans ce dossier : *Procédure_Activation_Visual_Paradigm.docx*

C'est lui que nous allons utiliser dans le cours.

En démarrant VP, vous allez voir :



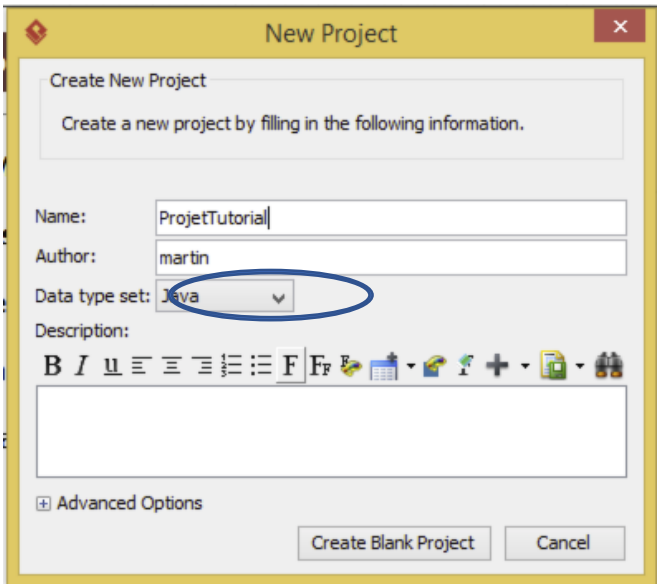
Sur le côté vous trouvez 4 onglets. Nous n'allons utiliser que les trois premiers.

- **L'explorateur de modèle** : vous permettent de consulter tous les éléments (classes, diagrammes, package…) du modèle en cours
- **Le navigateur de diagramme** : Cette vue retrouve tous les diagrammes dans votre projet et elle vous les présente par type de diagramme
- **Les propriétés** : montre les propriétés relatives à l'élément sélectionné.

Avant de commencer

Il faut d'abord créer un projet.

- Appuyer sur le bouton *Project* puis sur le sous-menu *New* en haut à gauche dans le ruban *Project*.
- Entrez le nom de votre projet *ProjetTutoriel* puis ***n'oubliez pas de sélectionner JAVA*** **comme Data type set**. Cela va assurer que les types utilisés sont ceux du Java.

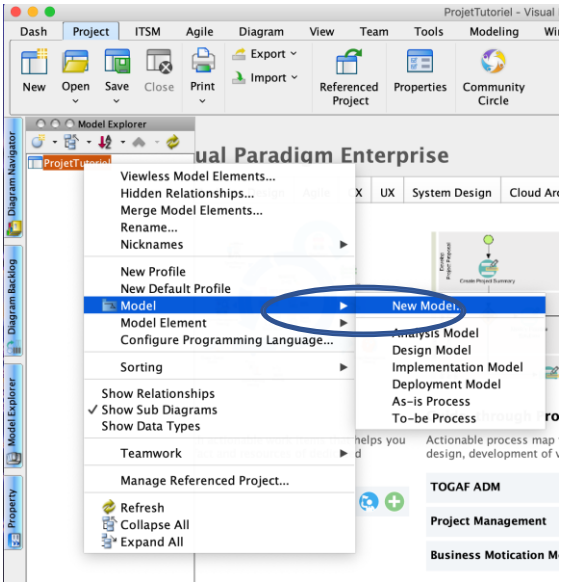


Un workspace VP peut contenir plusieurs projets, mais un seul projet peut être ouvert à la fois. Pour travailler avec simultanément avec plusieurs idées (alternatives), il faut créer plusieurs *modèles indépendants*. Un modèle est un ensemble d'éléments qui fonctionnent ensemble. Les modèles sont indépendants les uns des autres. Il est possible d'avoir plusieurs modèles dans un projet VP et il est possible de dupliquer des modèles pour concevoir simultanément plusieurs alternatives.

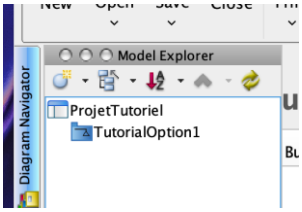
On vous recommande donc de **toujours créer un modèle** avant de fabriquer vos diagrammes. Le modèle permet entre autres de partager plusieurs éléments de modèles entre les différents diagrammes.

Pour créer un modèle :

- allez dans l'*explorateur de modèle* et à l'aide du menu contextuel sur votre projet sélectionner le menu **Model/New Model**.

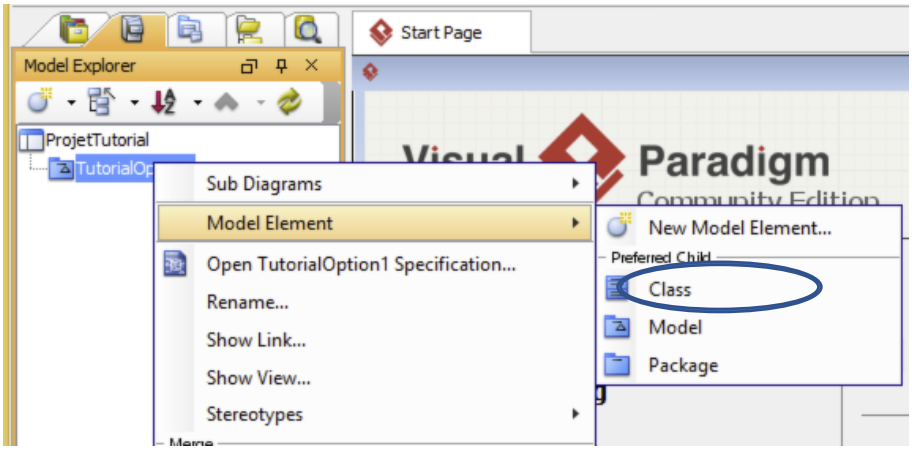


- Entrez ensuite le nom que vous voulez donner à ce modèle (par exemple : *tutorialOption1*)



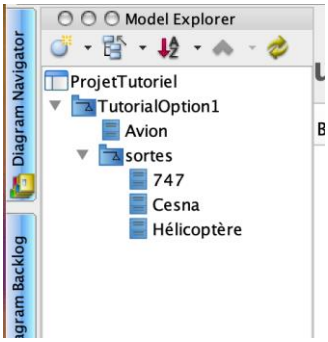
Noter qu'Il est possible d'imbriquer des modèles les uns dans les autres (comme les dossiers d'un système d'exploitation). On peut alors parler de sous modèles. Nous n'en avons pas besoin pour l'instant.

- Vous pourrez ensuite créer des classes à l'aide du menu contextuel sur votre nouveau modèle. Model Element/ Class
- Créez maintenant une classe Avion

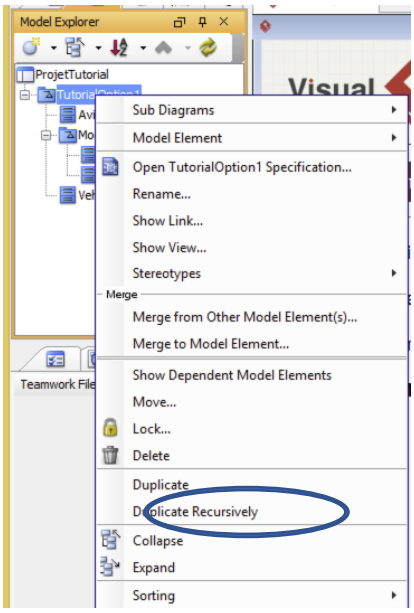


Vous pouvez créer plusieurs classes ou même des sous-projets dans votre projet en répétant les étapes précédentes.

- Créez les éléments (un sous-modèle et 3 classes) tel qu'illustrés dans l'image suivante :

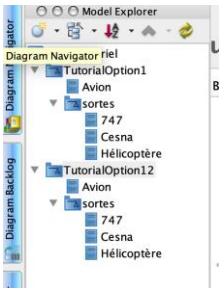


Très souvent on désire dupliquer un modèle existant pour esquisser une nouvelle hypothèse (pour proposer un design alternatif à vos collègues par exemple). VP offre 2 options pour faire cela simplement : **Duplicate** (seulement le modèle sélectionné) et **Duplicate recursively** (tous les éléments et sous éléments du modèle sélectionné incluant les sous-modèles). En général, on préfère donc *Duplicate recursively*.



En faisant *Duplicate recursively* on crée une copie complètement indépendante, mais conforme de notre modèle. On pourra la modifier à notre guise sans abimer le modèle original. On peut donc facilement créer plusieurs versions de modèles afin d'étudier et comparer plusieurs pistes de design.

- Maintenant, faites une copie récursive du modèle afin de créer une seconde option de design.

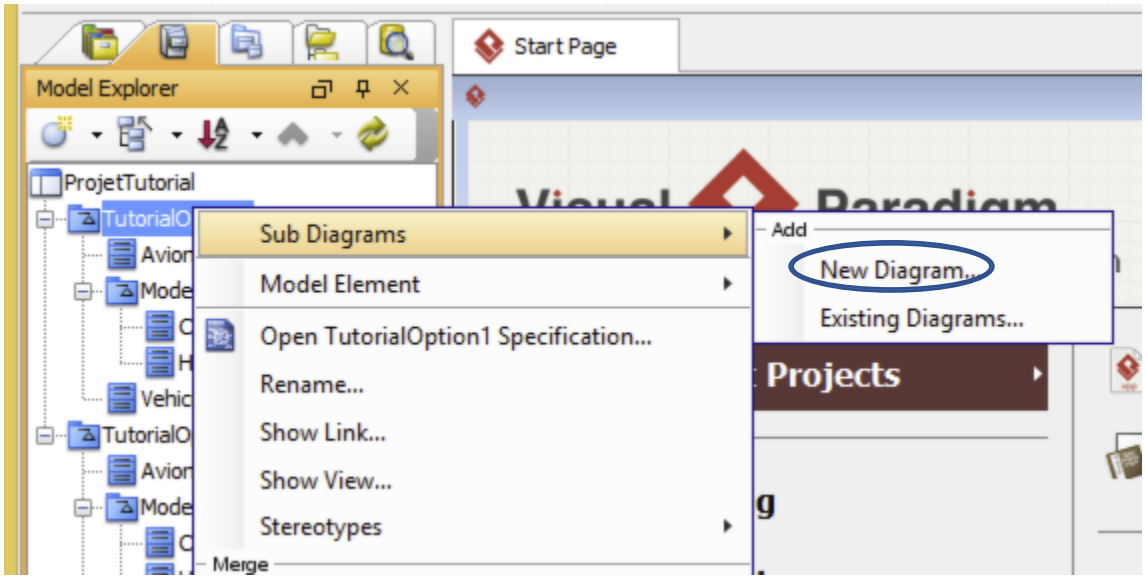


Une fois le projet bien préparé avec un modèle bien défini, on peut s'attaquer aux différents diagrammes.

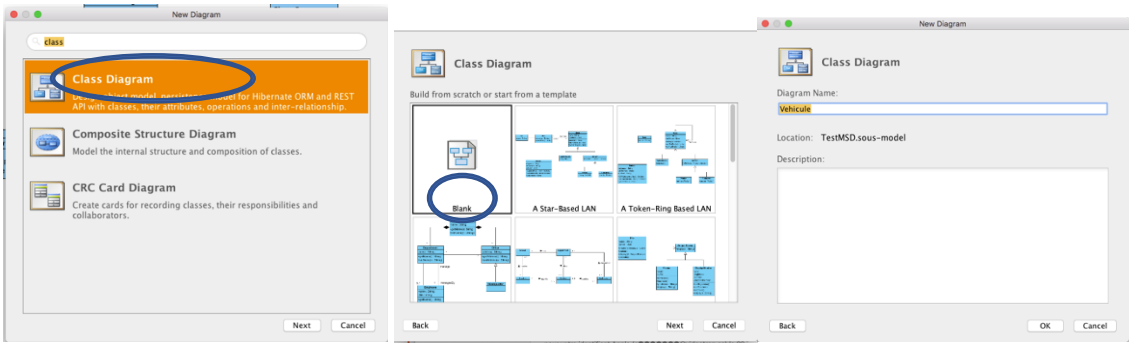
Le diagramme de classe

Le diagramme de classe est un diagramme conçu pour montrer rapidement **l'organisation** des différentes classes qui constituent un programme. Pour créer un diagramme de classe

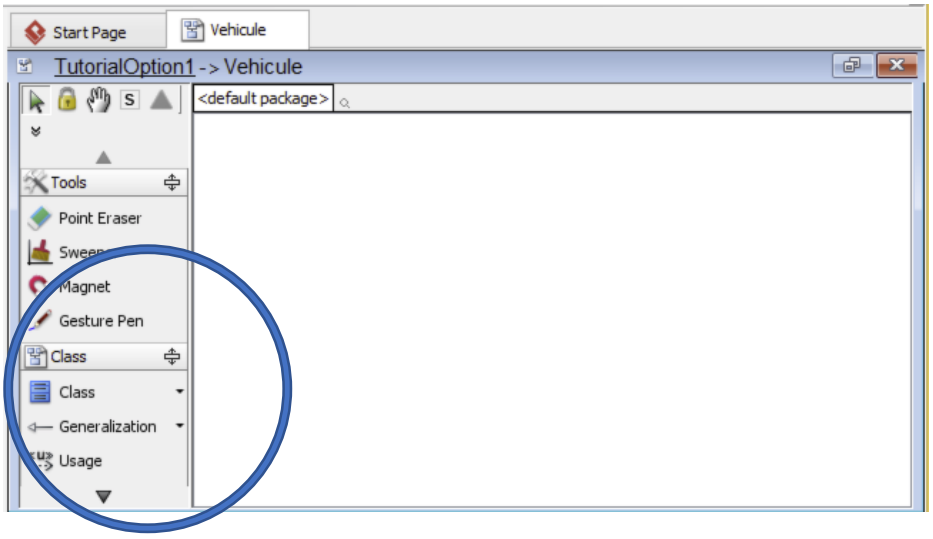
- Ouvrez le menu contextuel sur votre modèle *TutorialOption1* (Sub Diagrams / New Diagram)



- Pour trouver plus facilement le diagramme de classe, il suffit de taper class dans le champ de recherche en haut de la fenêtre. Choisissez le Diagramme de classe « Blank » et donner lui un nom (ici *Vehicule*)



Le diagramme sera ajouté dans votre modèle ainsi que dans le navigateur de diagramme. Il sera également ouvert automatiquement. Vous pouvez alors commencer à créer votre modèle. La zone de travail ressemble alors à :

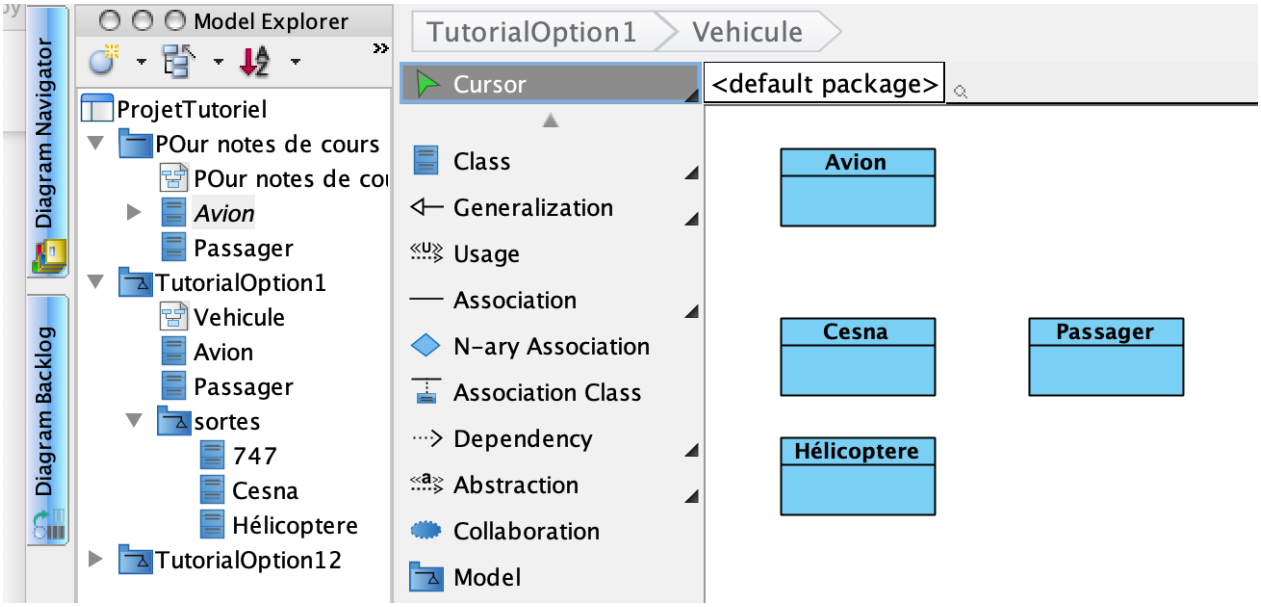


Si vous ne voyez pas les icônes ou les noms, vous pouvez les faire afficher avec le menu contextuel sur cette barre d'outils.

Ajouter une classe sur un diagramme.

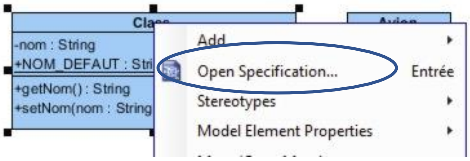
Pour ajouter une classe :

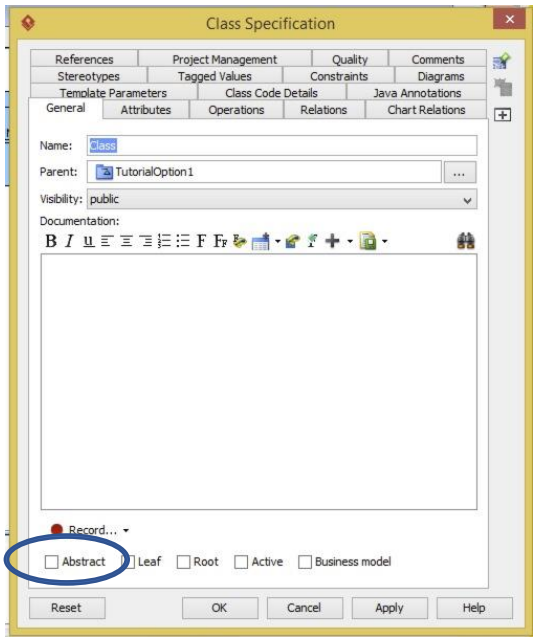
- Faire glisser un élément du modèle directement sur le diagramme. Faites glisser les classes *Avion, Cesna* et *Helicoptere* sur le diagramme.
- Cliquer sur le bouton class puis cliquer sur le diagramme à l'endroit où l'on veut voir apparaître la classe (glisser-déposer fonctionne également). Ajoutez une classe Passager en procédant ainsi.



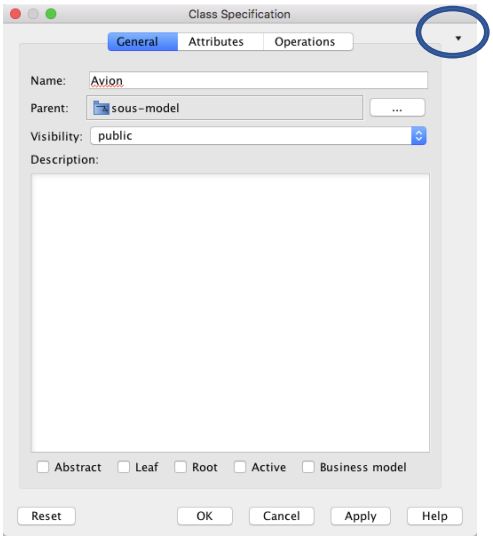
Remarquez que la classe *Passager* a été ajoutée dans votre modèle.

Pour spécifier davantage les caractéristiques de votre classe utiliser le menu contextuel sur le nom de la classe et sélectionner ***Open specification***. Vous aurez alors plus d'options.

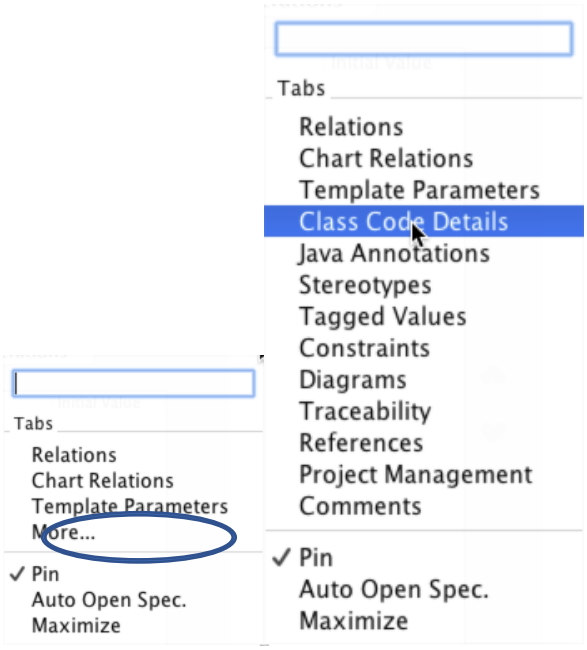




- Dans cet onglet vous pouvez dire que la classe est *abstract* (une classe qui ne peut pas être instanciée).
- L'interface ne vous montre que les choses les plus importantes, il faut appuyer sur la flèche dans le coin supérieur droit de la fenêtre pour avoir plus de détail



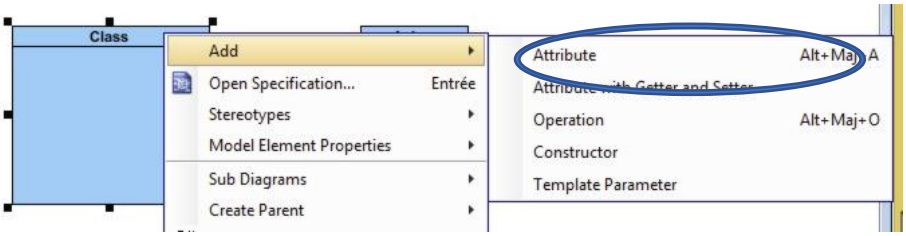
- Sélectionner *More…* et puis *Class Code Details*



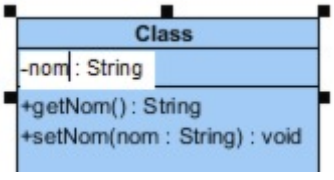
- On peut spécifier s'il s'agit d'une interface.
- Si la classe est *finale* (donc qu'on ne peut plus hériter de cette classe)
- Notez qu'en Java une classe ne peut pas être statique (sauf les classes imbriquées)

Ajouter des attributs

Pour ajouter des attributs, il suffit de cliquer sur le nom de la classe et de sélectionner le menu *Add / Attribute ou Add / Attribute with Getter and Setter*. Si vous choisissez le second menu, l'accesseur et le mutateur seront automatiquement créés pour votre attribut.



On entre alors en mode Édition dans l'éditeur de diagramme. (On peut également entrer en édition en double cliquant sur un élément).



Notez qu'en UML on déclare un attribut comme suit

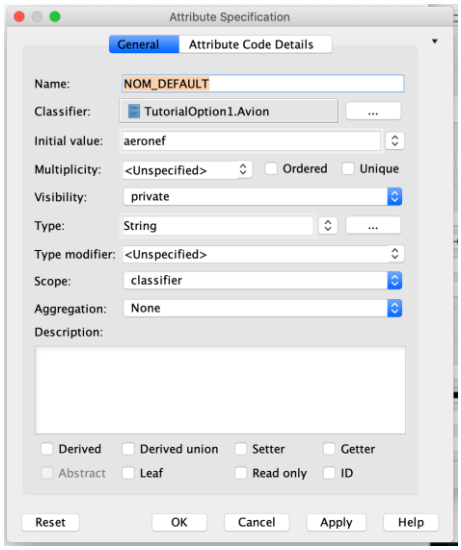
<visibilité> <nomDeAttribut> : <type>

Contrairement au Java le type est à la fin et non au début!

On rappelle que la visibilité d'un attribut doit toujours être privée ou protected sauf si c'est une constante.

Chaque attribut possède également une spécification. On peut y régler :

- Le nom
- Le type (Classifier)
- La multiplicité (nombre d'instances permises)
- La visibilité
- Le scope (static ou normal)
- Dans le bas on peut également cocher certaines options
    - Avec getter et/ou setter
    - Read only (constante)



Ajoutez maintenant les attributs suivants sur l'objet *Avion* :

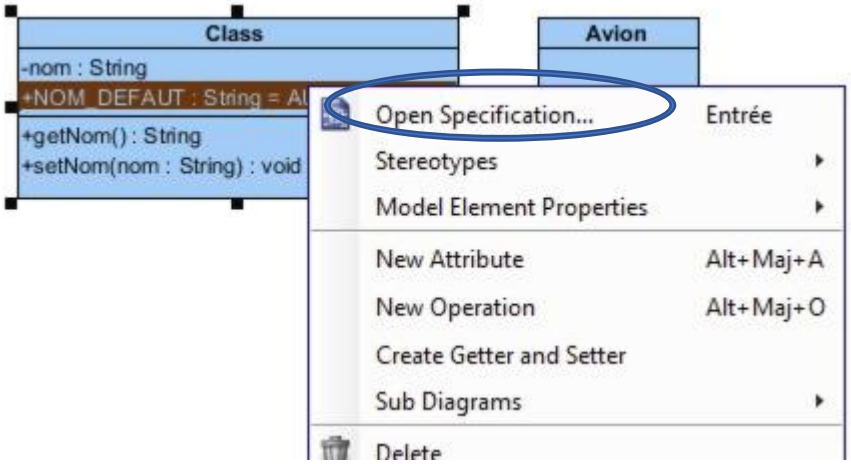
Nom de l'Attribut	visibilité	type	Valeur initiale	multiplicité
poids	private	float	1000	
nombrePassager	protected	int		
GRAVITE	public	float	9,8	
passagers	private	Passager		De 0 à 500
nombreAvion	private	int		

Ajouter une constante

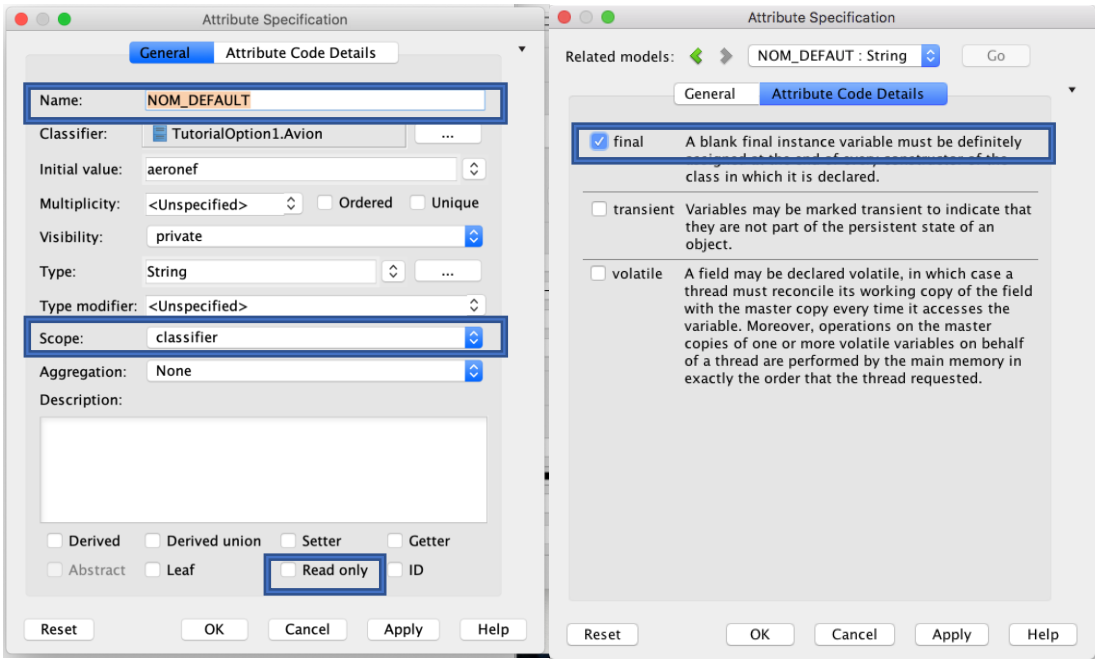
Pour ajouter une constante, il faut d'abord ajouter un attribut. Ici on ajoute l'attribut NOM_DEFAUT. Pour en faire une constante, il faudra ouvrir les spécifications de l'attribut.

- Sélectionnez attribute

- Ouvrir le menu contextuel **sur l'attribut**
- Choisissez *Open Specification*



- Une constante est statique. Pour créer un élément **statique** il faut changer le *Scope* d'instance vers *classifier*.
- On peut donner une valeur initiale au besoin.
- Une constante est souvent *publique,* on peut donc sélectionner la visibilité *public.*
- N'oubliez pas de mettre le nom de votre constante tout en majuscule.
- Choisissez le type de votre constante (ici *String*).
- Cochez *read-only* pour rendre l'attribut final.
- Il faut ensuite aller dans l'onglet *Attribute Code Details*. Cochez la case *final*. (Notez que l'option *transient* est réglable ici).



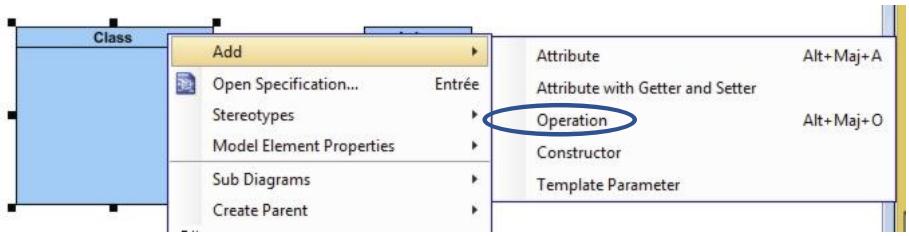
*TRUC : Visula Paradigm permet de faire des **glisser-déposer** avec les éléments. Vous pouvez donc facilement déplacer des éléments (attributs ou opérations) d'une classe à l'autre. De plus, si vous appuyer sur la touche ALT durant le déplacement VP effectuera une duplication (copie dans l'emplacement cible) au lieu d'un déplacement.*

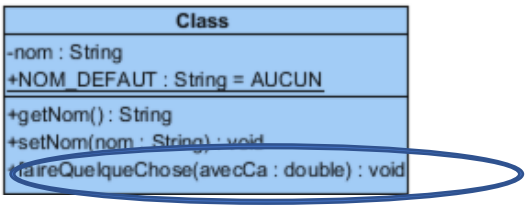
Ajoutez maintenant les constantes suivantes à la classe Avion.

Nom de l'Attribut	visibilité	type	Valeur initiale	multiplicité
GRAVITE	public	float	9,8	
NOM_DEFAUT	public	String	aeronef	

Ajouter une opération (équivalent d'une méthode en Java)

Pour ajouter des opérations, il suffit de cliquer sur le nom de la classe et de sélectionner le menu *Add / Operation*.





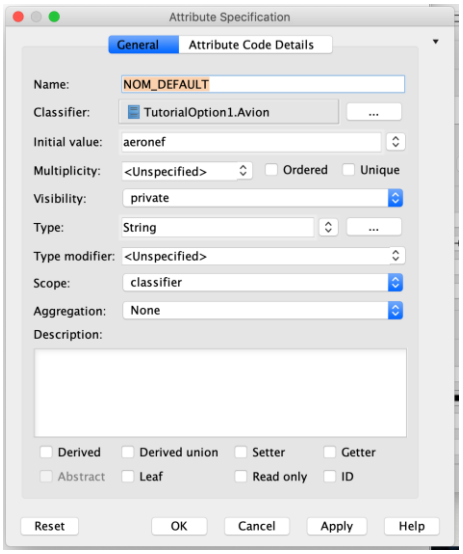
Ici on a ajouté l'opération *faireQuelqueChose*(). On remarque que la syntaxe est différente de celle du Java.

Chaque opération possède également une fenêtre pour changer ses spécifications :

En UML une opération s'écrit :

<visibilité><nom de l'opération>(<parametres>);<type de retour>

Une Opération possède également une boîte de spécification qui peut nous permettre de faire certains réglages:

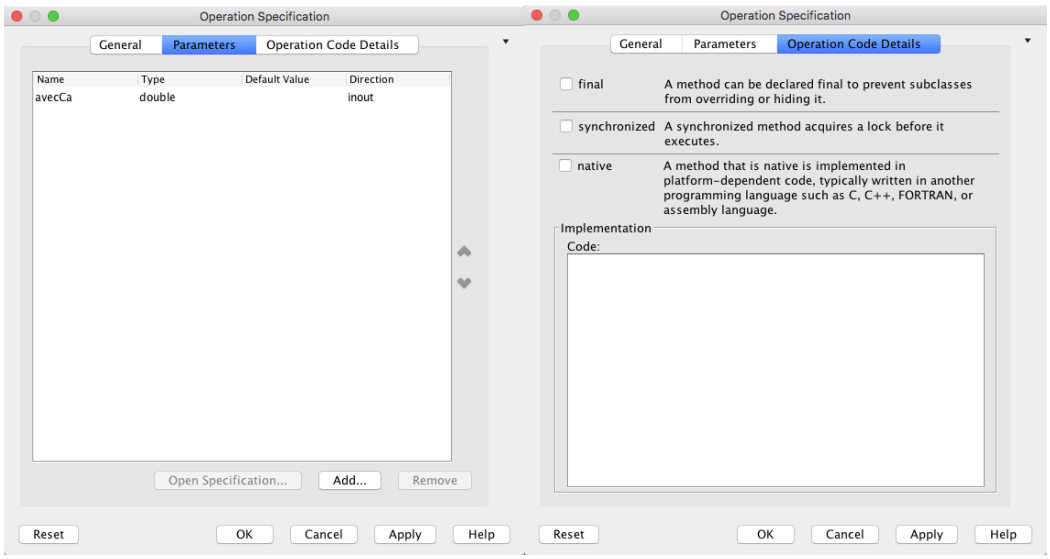


Ici on peut :

- modifier son nom
- Son type de retour
- Déterminer si elle est statique (Scope classifier)
- Sa visibilité

L'opération comporte également des onglets pour : spécifier ses paramètres, ajuster les détails du code. On peut modifier toutes les caractéristiques de chaque paramètre:

- Son nom
- Son type
- Sa direction et sa valeur par défaut ne sont pas utilisées en Java. Ils indiquent respectivement si les données entrent ou sortent de la méthode et la valeur du paramètre lorsqu'elle n'est pas fournie par l'appelant.



Ajoutez maintenant les opérations suivantes sur la classe Avion :

Nom de la méthode	visibilité	Type de retour
atterrir	package	
débarquer	public	Passager[]
décoller	package	
embarquer	public	Boolean
voler	public	

Affichage UML

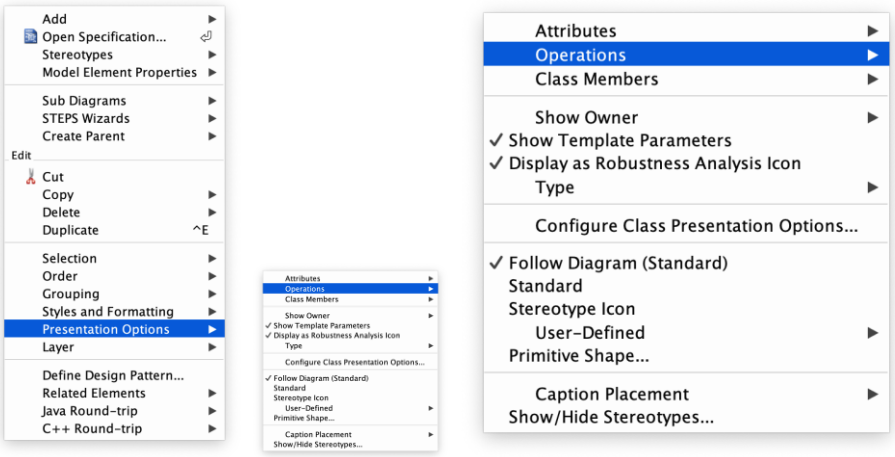
L'UML utilise certaines conventions pour afficher les éléments particuliers. Le tableau suivant en fait un résumé

Élément	Affichage
public	+
privé	-
protégé	#
package	~
statique	Souligné
abstrait	italique
constant	Tout en majuscule

Il se peut que certains éléments ne soient pas affichés sur le diagramme UML même s'ils sont dans le modèle.

IMPORANT : Un diagramme UML ne montre jamais tout le code. Ce serait inutilement compliqué et difficile à consulter. Il ne montre que les éléments qui sont intéressants pour un aspect donné du code. Par exemple, un diagramme pourrait ne montrer que les champs qui vont être transférés en BD. Si un élément n'est pas sur un diagramme, ça ne veut pas dire qu'il n'existe pas.

Pour décider de ce qui est affiché, on doit utiliser le menu *Presentation Options.* L'image suivante montre ce menu pour les opérations d'une classe.



Exercices complémentaires :

1. Modélliser une classe *Étudiant*. Il doit avoir :
    a. un nom
    b. un numéro de matricule
    c. une date d'inscription
    d. une moyenne générale
    e. Un nom par défaut
    f. Un numéro de matricule initial (999999)
    g. Une méthode qui permet d'inscrire l'étudiant
    h. Une méthode qui permet d'entrer un résultat
    i. Un constructeur qui reçoit une valeur initiale pour chacun des attributs de la classe
2. Modéliser une classe *Personnage*. Il doit avoir :
    a. Un attribut *allies* qui est un tableau de personnages
    b. Un attribut *pointDeVie* qui commence à 25
    c. Un attribut *combatEffectues* qui est un tableau de Combat
    d. Une méthode combattre qui reçoit un second personnage en paramètre et qui retourne le vainqueur