Mémo PlantUML

Principe plantUML

- PlantUML est un langage permettant de représenter facilement des diagrammes UML dont les diagrammes de classe et de séquences.

- Pour décrire un diagramme, il suffit d'écrire un fichier texte en respectant la syntaxe plantUML. Ce fichier peut ensuite être compilé pour générer une image ou un fichier pdf.
  - Page plantUML : http://fr.plantuml.com/
  - Compilateur en ligne : http://plantuml.com/plantuml/uml/
  - Bibliotheque java : http://plantuml.com/api.html

- Structure d'un fichier
  - Un fichier plantUML débute par "**`@startuml`**" et finit par "**`@enduml`**"
  - Le titre se déclare derrière le mot-clef "**`title`**"
  - Les commentaires se déclarent après une chaîne constituée **de trois apostrophes**

Diagramme de classe

```
@startuml
title Groupe de heros

'''classes
class Groupe{
  + ajouterHeros(Heros)
}
abstract class Heros{
  - nom : String
  + attaquerMonstre(Monstre)
}
class Guerrier{
  - degats : int
  + attaquerMonstre(Monstre)
}
class Magicien{
  - magie : int
  + attaquerMonstre(Monstre)
}

'''relations
Heros <|.. Guerrier
Heros <|.. Magicien
Groupe "1" --> "*" Heros : - heros
Magicien "1" -> "*" Sort : - sorts
@enduml
```

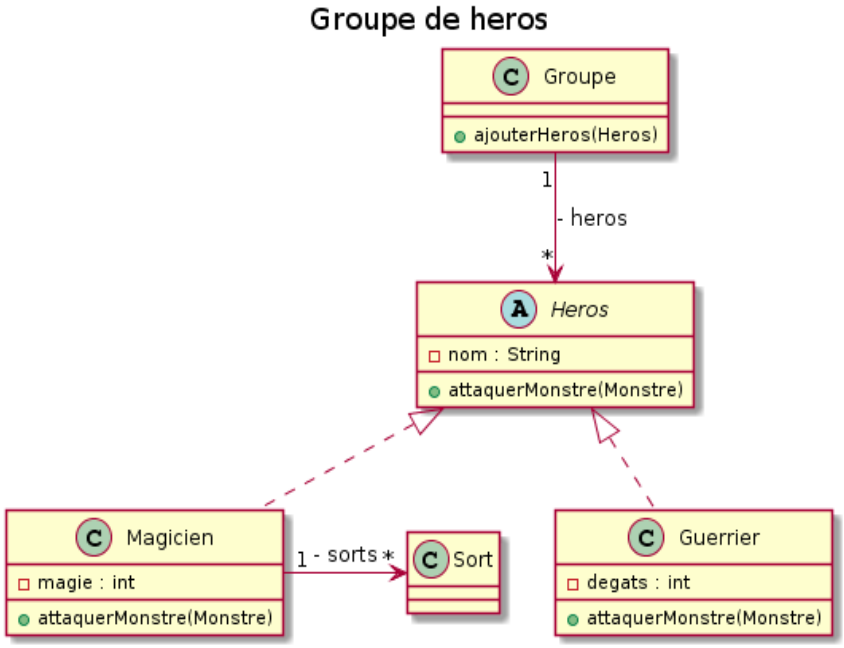
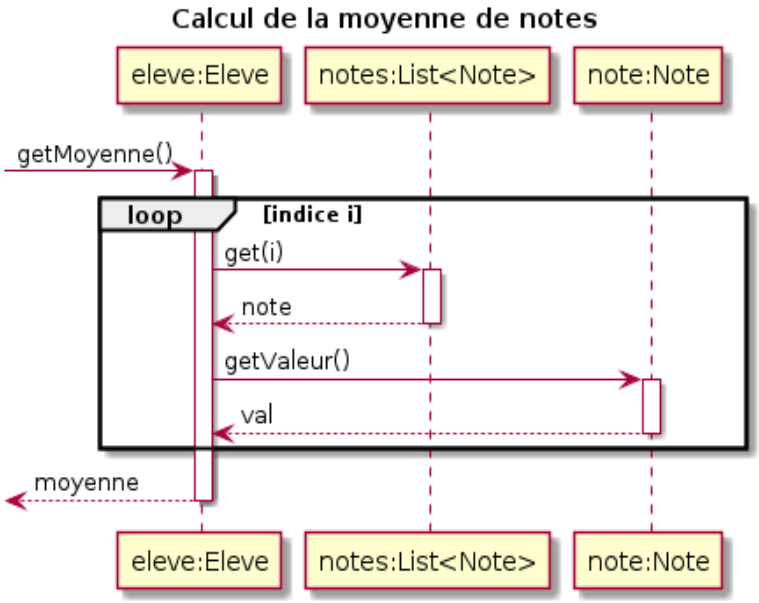


Diagramme de séquence

```
@startuml
title Calcul de la moyenne de notes

participant "eleve:Eleve" as eleve
participant "notes:List<Note>" as notes
participant "note:Note" as note

[-> eleve : getMoyenne()
activate eleve

  loop indice i
    eleve -> notes : get(i)
    activate notes
    eleve <-- notes : note
    deactivate notes

    eleve -> note : getValeur()
    activate note
    eleve <- note : val
    deactivate note
  end

[<-- eleve : moyenne
deactivate eleve
@enduml
```



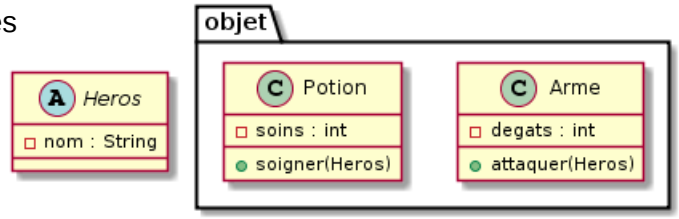
Mémo PlantUML – Diagramme de classe

- **Classe**
  - Classe / Interface se déclarent comme en java
  - Classe abstraite : **`abstract`** devant la classe
  - Méthodes et attributs automatiquement séparés

- **Méthodes et attributs**
  - Public ⇒ caractère **+** devant méthode/attribut
  - Privé ⇒ caractère **-** devant méthode/attribut
  - Abstract ⇒ **`{abstract}`** devant la méthode
  - Static ⇒ **`{static}`** devant la méthode

- **Package**
  - Créer des package en regroupant des classes entre accolades
  - Mot clef **`package`**

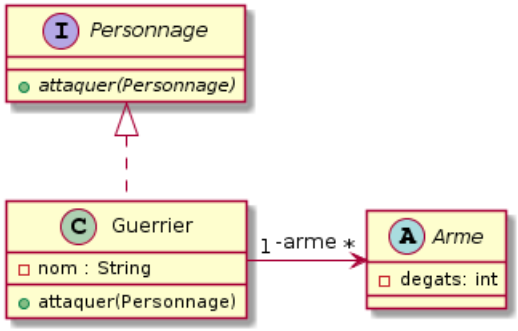
```
@startuml
Abstract class Heros {
  - nom : String
}
package objet{
  class Arme{
    - degats : int
    + attaquer(Heros)
  }
  class Potion{
    - soins : int
    + soigner(Heros)
  }
}
@enduml
```



- **Relations**
  - Chaque relation s'exprime par une ligne
  - **Héritage**
    - **`Mamifere <|-- Ours`**
  - **Implémentation**
    - **`Animal <|.. Mamifere`**
  - **Association**
    - **`Ours "1" -> "2" Patte : pattes`**
      - *Cardinalité* : guillemets à coté de la relation
      - *Nom* de la relation : aprés : fin ligne

- **Orientations des relations**
  - Par défaut, les entités sont placées correctement
    - **`<--`** ou **`-->`** désigne un lien vertical
    - **`<-`** ou **`->`** désigne un lien horizontal
  - On peut forcer l'orientation avec une direction
    - **`<-left-`** ou **`<-right-`** en horizontal
    - **`<-up-`** ou **`<-down-`** en vertical

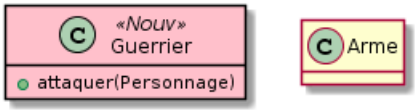
```
@startuml
Interface Personnage {
  + {abstract} attaquer(Personnage)
}
Class Guerrier{
  - nom : String
  + attaquer(Personnage)
}
Abstract Class Arme{
  - degats: int
}

Personnage <|.. Guerrier
Guerrier "1" -> "*" Arme : -arme
@enduml
```



- **Masquer des éléments**
  - Il est possible de cacher / montrer les éléments
    - Mots clefs **`hide`** ou **`show`**
    - Pour les attributs → **`fields`**
    - Pour les methodes → **`methods`**

- **Modifier les couleurs de rendu**
  - définir des catégories avec
    - **`skinparam class{ ... }`**
      - **`backgroundColor<<Categ>> Pink`**
      - **`borderColor<<Categ>> Black`**
  - Associer les catégories à la déclaration des classes
    - **`Class Guerrier<<Categ>>`**

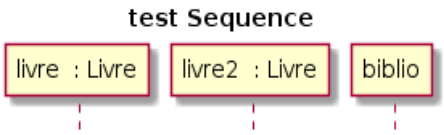
```
@startuml
skinparam class{
  BackgroundColor<<Nouv>> Pink
  BorderColor<<Nouv>> Black
}
Class Guerrier <<Nouv>>{
  - nom: String
  + attaquer(Personnage)
}
Class Arme {
  - degats: int
}
hide fields
@enduml
```



Mémo PlantUML – Diagramme de Séquence

- **Déclarer des objets**
  - La déclaration des objets est facultative
  - Les objets sont ajoutés en fonction des appels
  - Il est utile de nommer pour les utiliser facilement
    - On définit à la déclaration un identifiant
    - **`participant "livre: Livre" as id`**
  - Les objets sont affichés dans l'ordre avec des lignes de vie vide

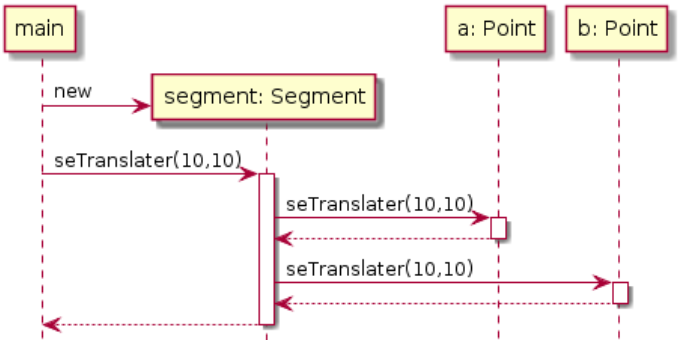
```
@startuml
title test Sequence
participant "livre  : Livre" as l1
participant "livre2  : Livre" as l2
participant "biblio" as biblio
@enduml
```



- **Envoi de message entre objets**
  - Déclarer un message correspond à une ligne
    - **`A -> B : message`**
  - Les messages sont affichés dans l'ordre
  - Un **appel synchrone** nécessite deux messages
    - appel lui-même
      - **`A -> B : methode()`**
    - retour en pointillé
      - **`B --> A : valeurDeRetour`**
  - Un appel à un **`constructeur`** est précédé
    - **`create objet`**
  - **`[`** représente des messages entrants

```
@startuml
participant "main" as m
participant "segment: Segment" as s
participant "a: Point" as a
participant "b: Point" as b

create s
m -> s : new
m -> s :  seTranslater(10,10)
activate s
  s -> a: seTranslater(10,10)
  activate a
  a --> s
  deactivate a
  s -> b: seTranslater(10,10)
  activate b
  b --> s
  deactivate b
m <--s
deactivate s
@enduml
```

- **Barre de vie**
  - Ajouter la barre de vie
    - Début après appel avec **`activate objet`**
    - Fin après retour avec **`deactivate objet`**
  - Un appel complet s'écrit donc en 4 lignes
    - **`rect -> p : seTranslater(10,10)`**
    - **`activate p`**
    - Appels internes
    - **`p --> rect`**
    - **`deactivate p`**



- **Fragment combinés**
  - Déclaration des fragments combines en encadrant les appels par des mots clefs
  - **Boucles**
    - Début → **`loop`** ***`message`***
    - Fin → **`end`**
  - **Conditionnelles**
    - Début → **`alt`**
    - Sinon → **`else`**
    - Fin → **`end`**

```
participant "b: Bilbio" as b
participant "l: List<Livre>" as l
[-> b :  parcourirLivres()
activate b
  loop i
    b -> l: get(i)
    activate l
    l --> b : livre
    deactivate l
  end
[<--b
deactivate b
```