

	<i>Université de Corse - Pasquale PAOLI</i>
	<b>Diplôme : MASTER1 Informatique DFS-DE</b>
<b>2025-2026</b>	
<b>Cours Patterns</b>	
<b>TD N°4 : Design Patterns Comportementaux</b>	
<b>Enseignant : Evelyne VITTORI</b>	

## Exercice 1 - Personnages et Bonus : pattern State

On souhaite modéliser la structure et le comportement des personnages dans un jeu d'aventure. On considère que chaque personnage est caractérisé par un nombre de points et un niveau (1, 2 ou 3 pour l'instant).

Au cours du jeu, le personnage peut augmenter son nombre de points en gagnant des bonus. Le gain d'un bonus dépend du niveau du personnage. Si le personnage est niveau 1, le gain d'un bonus lui rapportera 50 points, s'il est niveau 2 le gain d'un bonus sera de 1000 points et s'il est de niveau 3, le gain d'un bonus sera de 10000 points.

Le niveau d'un personnage doit pouvoir évoluer au cours du jeu. Lors de sa création, un personnage est par défaut de niveau 1. Ensuite, le passage de niveau se fait de manière automatique lorsque le personnage atteint un certain nombre de points (variable selon son niveau).

Au niveau 1, le passage automatique se fera lorsque le personnage atteindra 100 points. Au niveau 2, le passage au niveau suivant se fera lorsque le personnage atteindra 5000 points. Pour le niveau 3, pour l'instant aucun passage de niveau ne doit être implémenté mais l'on sait déjà que le nombre de points requis pour passer au niveau suivant sera de 50000.

La conception doit permettre d'ajouter un nouveau niveau en limitant la modification à une seule des classes existantes et en particulier sans avoir à modifier la classe Personnage.

**Question 1.1 :** Définissez un diagramme de classe utilisant le pattern State pour modéliser cette situation.

**Question 1.2 :** Coder en Java la solution proposée en 1.1. en définissant une méthode test située dans une classe TestPersonnage et permettant d'exécuter les actions suivantes :

- créer un personnage de nom Zenon,
- afficher son niveau et son nombre de points (résultat : *Zenon, 0 point, niveau 1*)
- enregistrer le gain d'un bonus pour Zenon
- enregistrer le gain d'un bonus pour Zenon
- afficher à nouveau son statut (résultat : *Zenon, 100 points, niveau 2*)

**Question 1.3 :** Définissez un diagramme de séquence représentant l'invocation de la méthode test définie en 3.2 sur un objet de la classe TestPersonnage.

**Question 1.4 :** Identifiez les modifications à réaliser pour ajouter un niveau 4 atteint lorsque le personnage totalise 50000 points au niveau 3 et caractérisé par un gain bonus de 50000.

## **Exercice 2 - Base Militaire : Pattern Observer**

---

Une base militaire stratégique est défendue par des compagnies de soldats et par des avions. Lorsqu'une menace arrive sur la base stratégique, celle-ci doit prévenir en temps réel chacun de ses défenseurs (compagnies et avions) (affichage d'un message sur les postes des défenseurs).

De même lorsque le calme revient à la base après une attaque, les défenseurs doivent en être informés.

Lorsqu'un défenseur est détruit (compagnie détruite ou avion scratché), la base doit à son tour en être informée immédiatement (affichage d'un message sur le poste de la base).

La conception de l'application doit permettre d'ajouter de nouveaux types de défenseurs sans avoir à modifier les classes existantes.

**Question 2.1** Dessinez un diagramme de classe modélisant ce problème en utilisant le pattern Observer.

**Question 2.2** Codez en Java une version simplifiée de vos classes (affichage sur une console unique en simulant l'affichage sur plusieurs postes par des messages) et testez leur fonctionnement en implémentant le scénario suivant :

- Création d'une base stratégique de nom « Solenzara » ;
- Engagement de la compagnie « Intrepide » de type « Hara Kiri » pour défendre la base ;
- Engagement de l'avion « Aiglon » de type A320 pour défendre la base
- Attaque de la base ;
- Scratch de l'avion Aiglon ; (l'avion Aiglon ne fait plus partie des défenseurs)
- Retour au calme à la base Solenzara ;
- Destruction de la compagnie Intrepide.

L'exécution du programme doit donner lieu à un affichage console similaire à l'affichage suivant :

```
🔇 Base : Ajout du défenseur -> Compagnie Intrepide de spécialité Hara Kiri
🔇 Base : Ajout du défenseur -> Avion Aiglon de type A320

🔊 Base : Notification aux défenseurs !
🛡️ Compagnie Intrepide [Hara Kiri] : Réception de l'alerte -> (03/03/2025)⚠️
Base attaquée !
📡 Avion Aiglon [A320] : Réception de l'alerte -> (03/03/2025)⚠️ Base attaquée !
🔇 Fin de la notification

🔥 Avion Aiglon a été détruit !
🔇 Base : Suppression du défenseur -> Avion Aiglon de type A320

🔊 Base : Notification aux défenseurs !
🛡️ Compagnie Intrepide [Hara Kiri] : Réception de l'alerte -> (03/03/2025)✅
Le calme est revenu.
🔇 Fin de la notification

✖️ Compagnie Intrepide a été anéantie !
```

## Exercice 3 - Pilotage d'un robot : pattern Command

On dispose d'une classe Robot comportant plusieurs méthodes permettant de piloter un robot (cf figure 1).

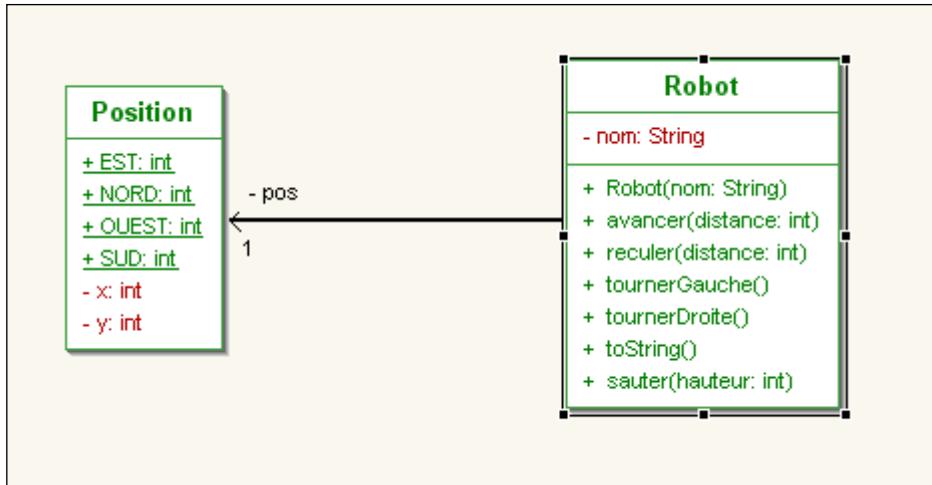


Figure1 : Diagramme de classe Robot

On souhaite modéliser une application permettant de définir des scénarios de pilotage d'un robot.

On suppose que les actions tournerGauche et tournerDroite sont limitées à des angles de rotation fixes de 45°.

Un scénario est composé d'une liste ordonnée d'actions de pilotage d'un robot.

### Exemple de scénario :

#### Scénario « Test1 »

1. Avancer de 10 unités
2. Tourner à gauche
3. Reculer de 20 unités
4. Tourner à droite
5. Sauter de 2 unités
6. Avancer de 3 unités

Un scénario doit pouvoir être annulé après son exécution. L'annulation d'un scénario consiste à exécuter les actions inverses (« undo ») de chacune des actions composant le scénario. A la fin de l'annulation, le robot concerné doit se retrouver dans sa position initiale (avant l'exécution du scénario).

Chaque action de pilotage du robot peut en effet être annulée (« undo ») par la réalisation d'une action d'annulation :

Action	Annulation
avancer(n)	reculer(n)
reculer(n)	avancer(n)
tournerGauche()	tournerDroite()
sauter(a)	Ne rien faire

### Exemple de scénario d'annulation

L'annulation du scénario Test1 évoqué précédemment se traduit par l'exécution des actions suivantes :

### Annulation du Scénario « Test1 »

1. Reculer de 3 unités
2. Ne rien faire
3. Tourner à gauche
4. Avancer de 20 unités
5. Tourner à droite
6. Reculer de 10 unités

L'application à modéliser doit utiliser la classe Robot existante.

Elle doit permettre :

- de créer un scénario vide
- d'ajouter des actions dans un scénario
- de supprimer des actions dans un scénario
- d'exécuter un scénario (exécution de la liste des actions composant le scénario)
- d'annuler l'exécution d'un scénario (exécution de la liste des actions « undo » associées à chacune des actions composant le scénario)

**Question 3.1** Appliquez le pattern Command pour modéliser cette application et définissez le diagramme de classe correspondant.

**Question 3.2** Récupérez sur l'ENT les classes Robot et Position, codez en Java une version simplifiée de vos classes et testez leur fonctionnement en implémentant le scénario suivant :

- création d'un robot
- création du scénario test1 (voir ci-dessus)
- exécution du scénario
- annulation de cette exécution.

L'exécution du programme doit donner lieu à un affichage console similaire à l'affichage suivant :

```
Position initiale : Robot Toto : (0,0, NORD)
EXECUTION DU SCENARIO TEST 1
Commande N°1
Toto avance de 10 : nouvelle position (0,10, NORD)
Commande N°2
Toto tourne à gauche : nouvelle position (0,10, OUEST)
Commande N°3
Toto recule de 20 : nouvelle position (20,10, OUEST)
Commande N°4
Toto tourne à droite : nouvelle position (20,10, NORD)
Commande N°5
Toto saute de 2 : nouvelle position (20,10, NORD)
Commande N°6
Toto avance de 3 : nouvelle position (20,13, NORD)
ANNULATION DU SCENARIO TEST 1
Annulation Commande N°6
Toto recule de 3 : nouvelle position (20,10, NORD)
Annulation Commande N°5
Annulation Commande N°4
Toto tourne à gauche : nouvelle position (20,10, OUEST)
Annulation Commande N°3
Toto avance de 20 : nouvelle position (0,10, OUEST)
Annulation Commande N°2
Toto tourne à droite : nouvelle position (0,10, NORD)
Annulation Commande N°1
Toto recule de 10 : nouvelle position (0,0, NORD)
Position finale : Robot Toto : (0,0, NORD)
```

**Question 3.3** Quels sont à votre avis les avantages de l'utilisation du pattern Command au niveau de l'évolutivité de l'application ?