Benjamin BEAUCAMP
Muruo WANG

TP3 – Programmation Objet

Ajout de nouvelles classes et modifications des classes existantes

Introduction

Dans ce TP, nous allons

- ajouter une super classe **Creature** pour les classes **Personnage** et **Monstre** comme suggéré dans le rapport précédent.
- implémenter la méthode **deplace()**
- fournir la documentation en utilisant le mécanisme **Javadoc**
- implémenter le système de combat avec **combattre**

1) Déplacement des protagonistes

Le sujet nous demande d'implémenter la méthode **deplace()** dans les classes **Personnage** et **Monstre**, mais comme le diagramme UML propose d'implémenter une super-classe **Creature** plus tard, nous avons choisi de mettre **deplace()** dans la super-classe. C'est plus judicieux car le code de la méthode ne change pas entre un **Personnage** et un **Monstre**.

Pour le déplacement, on génère deux entiers aléatoirement dans [-1, 1] qui servent à translater la créature suivant x et y. Il est donc possible qu'une créature ne se déplace pas (1 chance sur 9).

On pourrait par contre surcharger la méthode **deplace()** pour une sous-classe de **Personnage** ou **Monstre** si on autorise un déplacement de plus d'une case, ou un déplacement en ligne uniquement par exemple. On pourrait imaginer que le lapin peut bondir et qu'il a donc le droit de se déplacer de deux cases avant mais de seulement une seule case dans toutes les autres directions.

Enfin, pour interdire le déplacement sur une position déjà occupée, il faudrait tenir dans **World** un tableau des positions déjà occupées, qui serait passé en paramètre de **deplace()** comme une liste de positions interdites. Dans **deplace()**, il faudrait commencer par vérifier qu'une position est libre avant de se déplacer, puis ne pas oublier de supprimer l'ancienne position et d'ajouter la nouvelle. Un **ArrayList** semble judicieux pour stocker ces positions car on a seulement besoin de chercher un élément et de le remplacer. De plus, si on souhaite ajouter un personnage en jeu, on peut facilement ajouter sa position à la fin du tableau. On pourrait aussi utiliser une table de hachage qui à une créature associe sa position.

Test du déplacement : (issu de TestSeance2.java)

On test le déplacement d'un archer, qui hérite de la méthdoe de **Creature**. On initialise donc un monde contenant seulement robin. On le déplace ensuite, et on voit que sa position a bien changé.

```
Monde aléatoire est crée !
Nom : Robin
Points de vie : 100
Position : [3 ; 8]
Points d'attaque : 100
Pourcentages d'attaque : 100
Points de parade : 100
Pourcentages de parade : 100
Points de mana : 5
Dégâts de magie : 10
Pourcentage de magie : 30
Pourcentage de résistance à la magie : 30
Distance d'attaque maximale : 5
Nombre de flèches  5

Nom : Robin
Points de vie : 100
Position : [4 ; 7]
Points d'attaque : 100
Pourcentages d'attaque : 100
Points de parade : 100
Pourcentages de parade : 100
Points de mana : 5
Dégâts de magie : 10
Pourcentage de magie : 30
Pourcentage de résistance à la magie : 30
Distance d'attaque maximale : 5
-Nombre de flèches  5
```

2) Ecriture et génération Javadoc

Nous avons ajouté la javadoc pour toutes les classes, leurs attributs et les méthodes principales (typiquement pas pour le constructeur avec tous les paramètres).

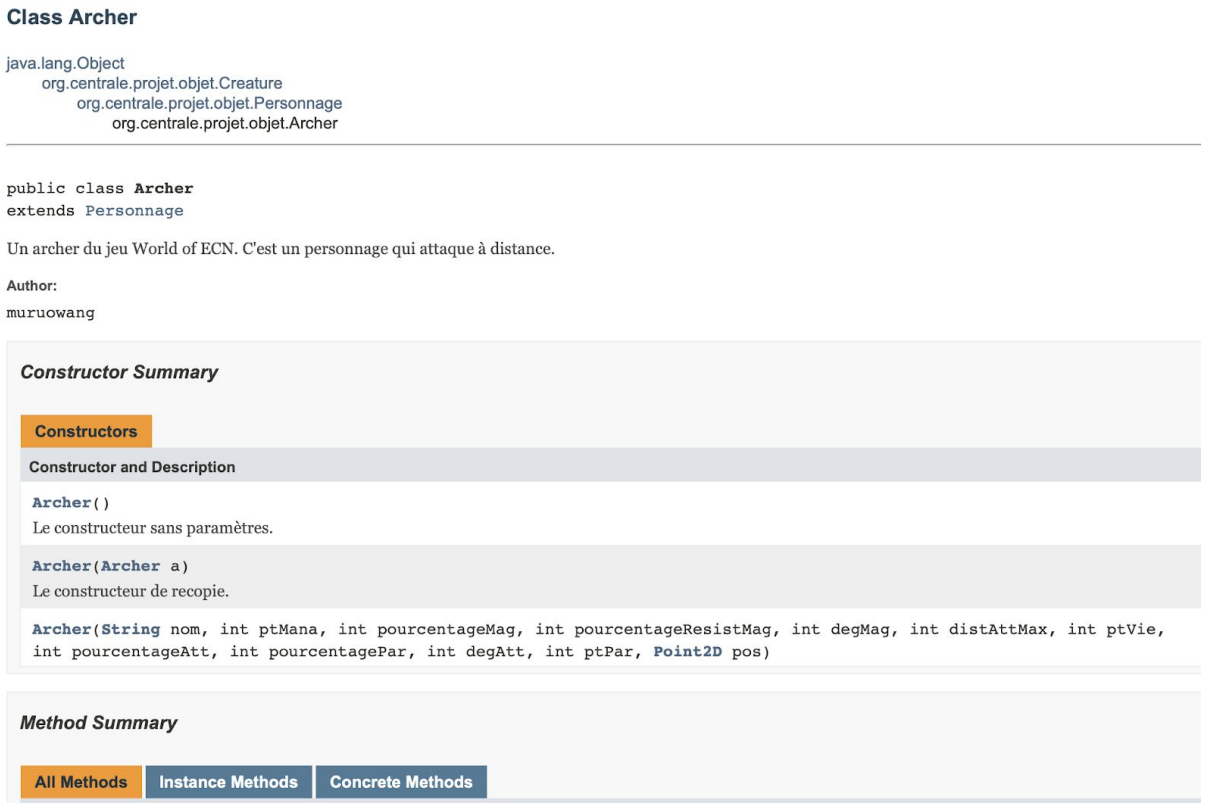
Quand on consulte la Javadoc, on a d'abord une vue d'ensemble du package :

Package org.centrale.projet.objet

Class Summary

Class	Description
Archer	Un archer du jeu World of ECN.
Creature	Une entité de base du jeu World of ECN.
Lapin	Un lapin du jeu World of ECN.
Loup	
Monstre	Un monstre du jeu World of ECN.
Paysan	Un paysan du jeu World of ECN.
Personnage	Un personnage du jeu World of ECN.
Point2D	Un point du plan, défini par deux coordonnées entières.
TestPoint2D	
TestSeance1	
TestSeance2	
World	Le monde 2D dans lequel évolue les créatures du jeu World of ECN.

Grâce aux lien hypertextes, on peut ensuite aller voir plus en détail une classe (par exemple **Archer**). On a ici un résumé des constructeurs et méthodes :



Class Archer

java.lang.Object
org.centrale.projet.objet.Creature
org.centrale.projet.objet.Personnage
org.centrale.projet.objet.Archer

```
public class Archer
extends Personnage
```

Un archer du jeu World of ECN. C'est un personnage qui attaque à distance.

Author:
muruowang

Constructor Summary

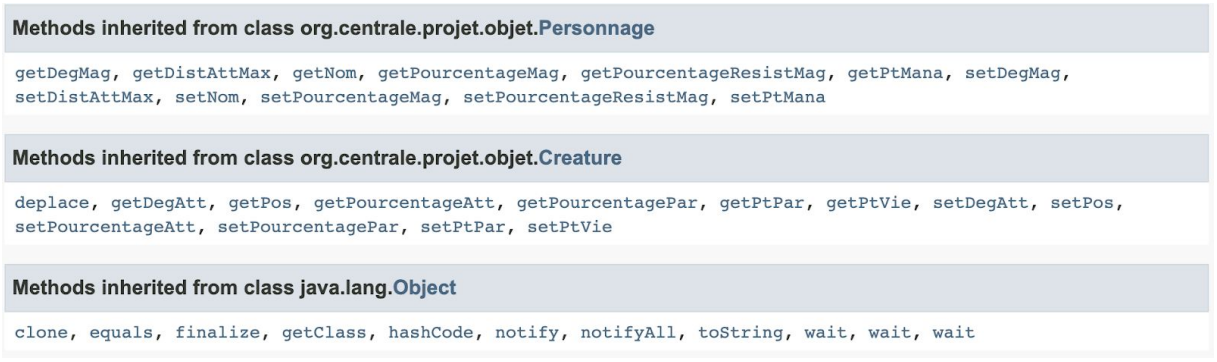
Constructors

Constructor and Description
`Archer()` Le constructeur sans paramètres.
`Archer(Archer a)` Le constructeur de recopie.
`Archer(String nom, int ptMana, int pourcentageMag, int pourcentageResistMag, int degMag, int distAttMax, int ptVie, int pourcentageAtt, int pourcentagePar, int degAtt, int ptPar, Point2D pos)`

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

On voit aussi ses super classes et les méthodes héritées, auxquelles on peut accéder avec un hyperlien.



Methods inherited from class org.centrale.projet.objet.Personnage

`getDegMag`, `getDistAttMax`, `getNom`, `getPourcentageMag`, `getPourcentageResistMag`, `getPtMana`, `setDegMag`, `setDistAttMax`, `setNom`, `setPourcentageMag`, `setPourcentageResistMag`, `setPtMana`

Methods inherited from class org.centrale.projet.objet.Creature

`deplace`, `getDegAtt`, `getPos`, `getPourcentageAtt`, `getPourcentagePar`, `getPtPar`, `getPtVie`, `setDegAtt`, `setPos`, `setPourcentageAtt`, `setPourcentagePar`, `setPtPar`, `setPtVie`

Methods inherited from class java.lang.Object

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Enfin, on accède au détail des attributs et méthodes (attention : on ne peut voir que les attributs et méthodes publics. Ici il n'y a pas d'attributs publics) avec le paragraphe de description complet et la spécification de chaque paramètre (type, nom, description). La javadoc nous indique même si la méthode est redéfinie :

affiche

```
public void affiche()
```

Description copied from class: `Personnage`
Affichage textuel de tous les attributs de la créature.

Overrides:
`affiche` in class `Personnage`

3) Copie d'objet

Pour tester notre constructeur de recopie, on crée un archer robin avec le constructeur sans paramètres. On crée ensuite guillaumeT comme une copie de robin avec le constructeur de recopie. Affichage de robin (et guillaumeT) :

```
Nom : Robin
Points de vie : 100
Position : [0 ; 0]
Points d'attaque : 100
Pourcentages d'attaque : 100
Points de parade : 100
Pourcentages de parade : 100
Points de mana : 5
Dégâts de magie : 10
Pourcentage de magie : 30
Pourcentage de résistance à la magie : 30
Distance d'attaque maximale : 5
Nombre de flèches  5
```

On déplace ensuite robin, et on constate que guillaumeT n'a pas bougé :

```
Nom : Robin
Points de vie : 100
Position : [0 ; -1]
Points d'attaque : 100
Pourcentages d'attaque : 100
Points de parade : 100
Pourcentages de parade : 100
Points de mana : 5
Dégâts de magie : 10
Pourcentage de magie : 30
Pourcentage de résistance à la magie : 30
Distance d'attaque maximale : 5
Nombre de flèches  5
Nom : guillaumT
Points de vie : 100
Position : [0 ; 0]
Points d'attaque : 100
Pourcentages d'attaque : 100
Points de parade : 100
Pourcentages de parade : 100
Points de mana : 5
Dégâts de magie : 10
Pourcentage de magie : 30
Pourcentage de résistance à la magie : 30
Distance d'attaque maximale : 5
-Nombre de flèches  5
```

Le constructeur de recopie de **Creature** effectue bien une copie profonde car nous appelons le constructeur de recopie de Point2D (au lieu de faire une simple copie avec **this.pos = a.pos**) :

```
public Creature(Creature c) {
    this.ptVie = c.ptVie;
    this.pourcentageAtt = c.pourcentageAtt;
    this.pourcentagePar = c.pourcentagePar;
    this.degAtt = c.degAtt;
    this.ptPar = c.ptPar;
    this.pos = new Point2D(c.pos);
}
```

4) Distance entre les protagonistes

Pour calculer la distance entre deux points, on implémente la méthode d'en tête **double distance(Point2D p)** qui calcule la distance euclidienne entre l'instance de la classe et p.

On utilise un type de retour double pour éviter de perdre en précision (car on manipule des puissances d'int).

5) Modification de la méthode creeMondeAlea

Comme nous avons ajouté de nouveaux personnages et monstres en jeu, il faut modifier l'initialisation aléatoire des positions. On garde la contrainte que toutes les créatures doivent être à des positions différentes et à une distance inférieure à 5.

Pour cela, on crée un tableau **positions** que l'on va remplir au fur et à mesure avec les positions initiales des créatures.

- On commence en initialisant un Point2D **newPos** de manière aléatoire, qu'on ajoute à **positions**.
- Ensuite, pour chaque position à créer, on initialise **newPos** aléatoirement puis on vérifie qu'il est bien différent et à une distance de moins de 5 de tous les autres Point2D de **positions**. Tant que cette condition n'est pas vérifiée, on génère une nouvelle valeur pour **newPos**. Finalement, on l'ajoute à **positions**.

Cet algorithme marche à condition qu'il y ait effectivement assez de positions vérifiant cette double contrainte. Ici, on se fixe une distance de 5 et on cherche à générer 8 positions, ce qui est largement suffisant.

6) Système de combat(1)

On a trois types de combats :

-combat à distance(Archer)

-combat magique (Mage)

-combat au corps à corps (Guerrier et Loup)

Test de combat au corps à corps

Dans la classe World, on a défini les positions (0,0) et (1,0) pour générer l'attaque(distance de 1), le résultat est le suivant:

```
Test du combat au corps à corps
Wolfie :
Points de vie : 50
Position : [1 ; 0]
Points d'attaque : 10
Pourcentages d'attaque : 50
Points de parade : 2
Pourcentages de parade : 20

Gros Bill attaque wolfie !
Jet d'attaque : 18. Pourcentage d'attaque : 70
Attaque au corps à corps réussie
Jet de parade : 19. Pourcentage de parade : 20
Parade réussie !
18 dégâts sont infligés

Wolfie :
Points de vie : 32
Position : [1 ; 0]
Points d'attaque : 10
Pourcentages d'attaque : 50
Points de parade : 2
Pourcentages de parade : 20
```

Gros Bill possède 20 points d'attaque. On voit que l'attaque et la parade ont réussis, Wolfie prend donc 20 - 2 = 18 dégâts et finit à 32 points de vie.

Test de combat à distance

Les deux archers sont initialisés avec les mêmes caractéristiques. Le résultat est le suivant:

```
Test du combat à distance
Robin attaque guillaumeT
Jet d'attaque : 63. Pourcentage d'attaque : 70
Attaque à distance réussie
Jet de parade : 31. Pourcentage de parade : 60
Parade réussie !
10 dégâts sont infligés

Nom : Robin
Points de vie : 100
Position : [0 ; 0]
Points d'attaque : 15
Pourcentages d'attaque : 70
Points de parade : 5
Pourcentages de parade : 60
Points de mana : 5
Dégâts de magie : 10
Pourcentage de magie : 30
Pourcentage de résistance à la magie : 30
Distance d'attaque maximale : 5
Nombre de flèches  4

Nom : guillaumT
Points de vie : 90
Position : [3 ; 2]
Points d'attaque : 15
Pourcentages d'attaque : 70
Points de parade : 5
Pourcentages de parade : 60
Points de mana : 5
Dégâts de magie : 10
Pourcentage de magie : 30
Pourcentage de résistance à la magie : 30
Distance d'attaque maximale : 5
Nombre de flèches  5
```

On a vérifié tous les valeurs, et après l'attaque robin a perdu une flèche donc on peut dire que le combat à distance marche bien.

Test de combat magique

Un magicien possède par défaut 7 points de mana. On voit bien ici qu'un point de mana a été utilisé (tentative d'attaque). Un loup possède par défaut 50 points de vie, il a bien perdu 20 points de vie au cours de l'attaque.

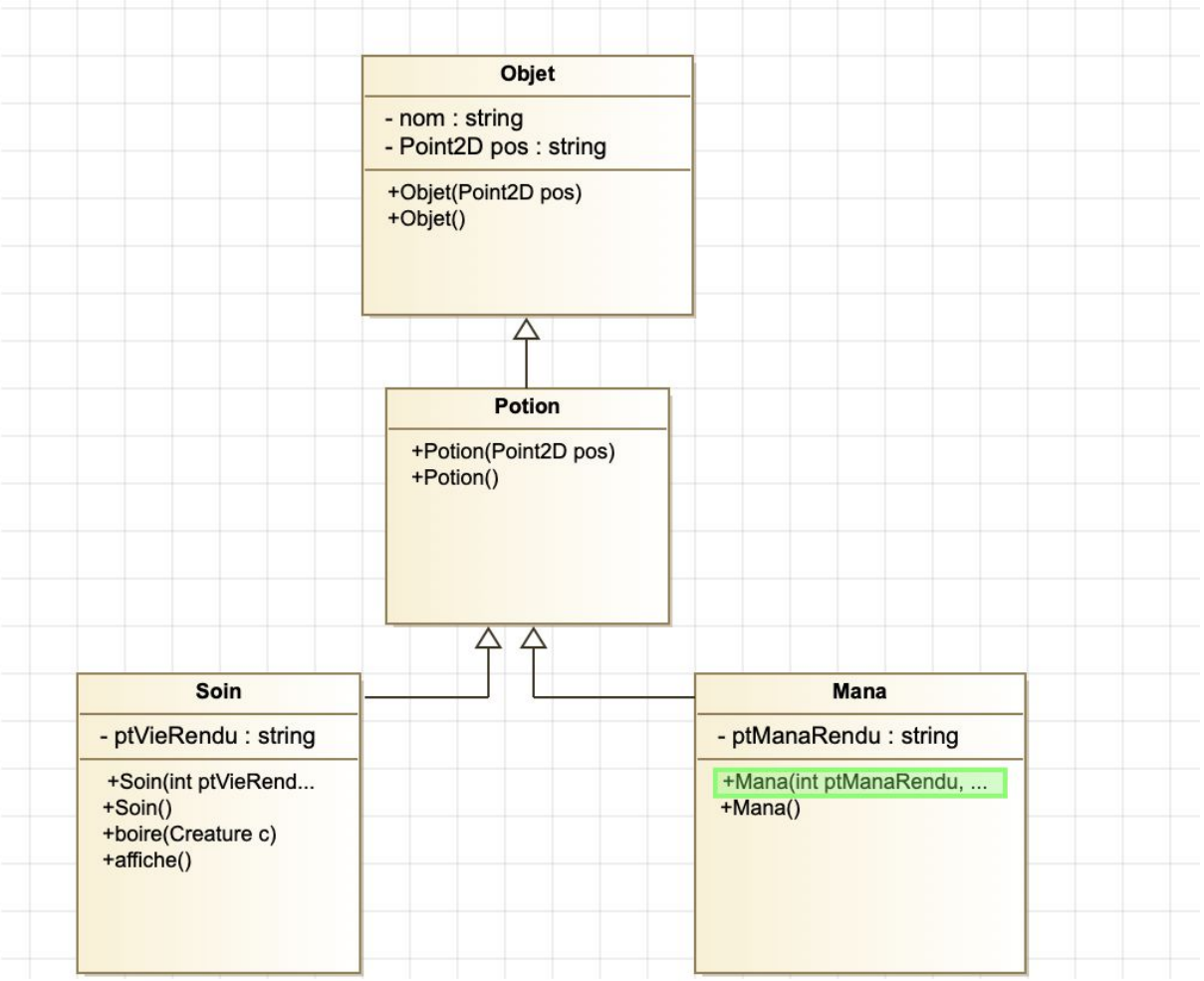
Le résultat est suivant:

```
Test du combat magique
Merlin attaque wolfie !
Jet d'attaque : 51. Pourcentage d'attaque : 80
Attaque magique réussie !
Jet de parade : 79. Pourcentage de parade : 20
Parade échouée !
20 dégâts sont infligés

Nom : merlin
Points de vie : 100
Position : [0 ; 0]
Points d'attaque : 15
Pourcentages d'attaque : 70
Points de parade : 5
Pourcentages de parade : 60
Points de mana : 6
Dégâts de magie : 20
Pourcentage de magie : 80
Pourcentage de résistance à la magie : 30
Distance d'attaque maximale : 3

Wolfie :
Points de vie : 30
Position : [2 ; 0]
Points d'attaque : 10
Pourcentages d'attaque : 50
Points de parade : 2
Pourcentages de parade : 20
```

On a vérifié les valeurs et après l'attaque le magicien a perdu un point de mana, c'est logique.

7) Objets

Nous avons implémenté le schéma suivant :



Remarque : dans le schéma, tous les attributs sont des string car nous n'avons pas réussi à changer leur type en **integer** ou **Point2D** dans Modelio.

La fonction **boire()** permet à une créature ou un personnage (suivant la potion) de gagner des points de vie ou de mana supplémentaires. Une fois que la potion est bue, sa quantité de points de vie ou de mana rendu passe à 0 (en attendant que nous implémentions un mécanisme permettant de gérer les potions dans World).

8) Conclusion

Dans ce TP, en utilisant un tableau on a défini une algorithme pour générer les positions des 8 protagonistes de manière aléatoire. On a aussi généré une documentation en utilisant Javadoc, ce qui sera utile quand on travaillera pour les projets et en entreprise. Pour les variables privées, elles ne sont pas présentées dans la documentation, c'est logique parce qu'on ne peut pas les utiliser les variables privées directement en dehors de classe. Nous avons aussi implémenté un système classique de combat pour un RPG, mais un personnage ne peut pas encore mourir (il ne se passe rien si ses points de vie passent à 0 ou moins). Enfin, nous avons commencé à implémenter des objets en jeu mais nous ne savons encore pas comment nous allons gérer les interactions entre les créatures et les potions.