Benjamin BEAUCAMP
Muruo WANG

TP2 – Programmation Objet
Compréhension diagramme de classes UML

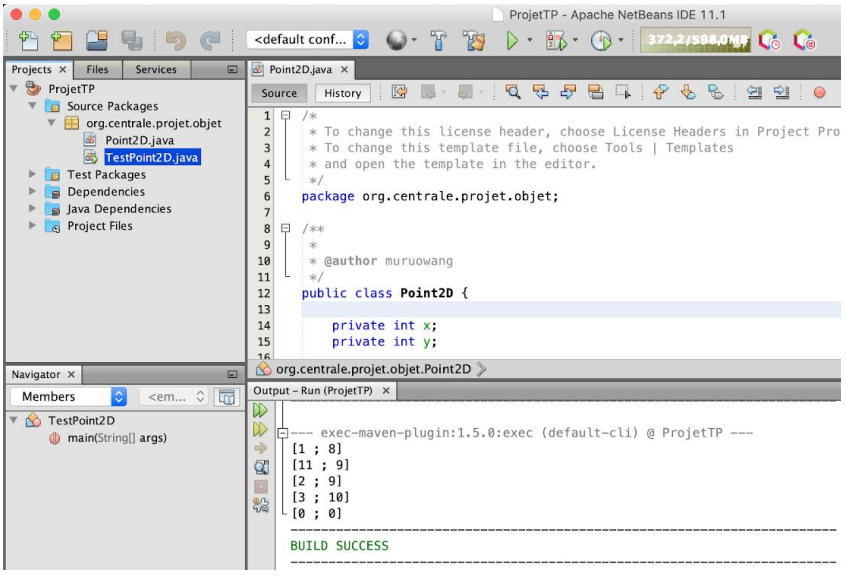
Introduction :

Dans ce TP, nous allons implémenter les premières classes de World of ECN. Cela nous permettra de mettre en pratique ce que nous avons vu en cours sur l'héritage :

- L'écriture du constructeur d'une classe enfant avec super()
- La redéfinition d'une méthode
- L'appel à super pour accéder à une méthode redéfinie de la classe mère

Nous allons aussi implémenter des constructeurs de recopie et découvrir comment générer des nombres pseudos-aléatoires en Java.

1) Création du projet



Après avoir ajouté "Point2D" et "TestPoint2D", le projet compile et affiche les résultats attendus (voir TP précédent).

2) Diagramme de classe UML

Dans le diagramme il y a deux super classes : Personnage et Monstre. Elles représentent respectivement une personne humaine et un animal dans le jeu. Chacun possède des

caractéristiques décrivant ses capacités au combat (attaque, défense), et les humains possèdent des caractéristiques supplémentaires (magie).

Paysan et Archer héritent de Personnage car ils sont des cas particuliers : un archer est un humain qui peut en plus tirer à l'arc, il possède donc des flèches. On peut de plus l'initialiser avec une distance d'attaque supérieur à celle d'un personnage générique. Le paysan ne possède pas de capacités particulières mais on peut par exemple choisir de lui donner des caractéristiques initiales différentes d'un personnage générique.

Il est est de même pour Lapin qui est un cas particulier d'un monstre.

On a choisi de créer ces super classes afin de regrouper les attributs et méthodes des entités du jeu. Même si plusieurs types de personnage et plusieurs types de monstres seront présent en jeu, ceux-ci auront tous des caractéristiques de base en commun. Cela nous permettra aussi de créer des interactions entre entités du jeu sans avoir à spécifier une fonction pour chaque type de personnage et chaque type de monstre grâce aux types génériques.

On aurait pu choisir de créer une super classe "Entité vivante" de laquelle héritent Personnage et Monstre afin de regrouper les attributs de point de vie, dégâts d'attaque, position, ... et d'affichage de ces attributs.

3) Nouvelles méthodes pour Point2D

Pour compléter le TP précédent, nous avons commencé par implémenter le constructeur de recopie dans la classe Point2D, qui sera utile pour définir le constructeur de recopie de Personnage.

```
public Point2D(Point2D p) {
    this.x = p.x;
    this.y = p.y;
}
```

Nous avons ensuite ajouté la méthode **equals** pour vérifier si deux points sont à la même position ainsi que la méthode **distance** qui permet de calculer la distance (en valeur absolue) à un autre Point2D. Ces deux méthodes sont utiles pour générer les positions des personnages dans la classe World.

```java
// méthode pour vérifier  si les deux points  sont  à la même position
public boolean equals(Point2D p){
    if (p == null) {
        return false;
    } else {
        return (this.x == p.getX() && this.y == p.getY());
    }
}

// méthode pour calculer la distance entre deux points
// valeur absolue
public int distance(Point2D p){
    int dis = Math.abs(this.x - p.getX()) + Math.abs(this.y - p.getY());
    return dis;
}
```

Enfin, nous avons ajouté la méthode **toString()** que nous allons utiliser dans la méthode **affiche()** de Personnage.

```java
// afficher les positions des objets
public String toString(){
    return ("[" + this.x + " ; " + this.y + "]");
}
```

4) Implémentation des classes et test

Création de classe “World”

Dans la méthode **creeMondeAlea**, on a généré trois positions aléatoires par le fonction de l’objet **Random**. Pour calculer la distance entre eux, on a appelé la méthode **distance** (défini dans classe Point2D). Afin d’éviter la superposition des personnages, nous avons utilisé une boucle while pour générer une position aléatoirement jusqu’à ce qu’il n’y ait plus de conflit.

Output de classe TestSeance1.java

- On a testé la génération d’un monde aléatoire avec “Archer” “Paysan” et “Lapin”, vérifié qu’il n’y avait pas de superposition de personnages et que la distance entre deux personnages était bien inférieure à 5.
- Pour tester les classes de personnages, on a défini les valeurs de “Archer”. On a choisi “Archer ” car “Archer” est sous-classe de superclasse “Personnage”, si classe “Archer” marche bien, il n’y a pas de problèmes pour classe “Personnage”.
- Puis pour tester la méthode “Set”, on a redéfini le nombre de flèches.
- On a fait le même test pour “Lapin”(sous-classe de “Monstre”).
- Enfin, nous avons testé le constructeur de recopie de Personnage pour vérifier si la copie est profonde. Nous avons créé deux personnages p1 et p2, puis après avoir modifié la position de p1 on voit que celle de p2 n’a pas été impactée (attention, dans le test p1 et p2 s’appellent tous les deux Personnage car nous avons utilisé le constructeur sans paramètre).

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ ProjetTP ---
Monde aléatoire est crée !
Archer se situe en [9 ; 6]
Archer possède 100 points de vie et 0 dégâts d'attaque
Archer possède 0 flèches

Paysan se situe en [6 ; 5]
Paysan possède 100 points de vie et 0 dégâts d'attaque

L'information de monstre
ptVie =  100
pourcentageAtt =  0
pourcentagePar = 0
degAtt = 0
pos = [9 ; 7]

Initialisation d'un nouvel archer :
Le lapin possède 7 fleches
Modificaition le nombre fleches :
Archer_test se situe en [28 ; 10]
Archer_test possède 99 points de vie et 17 dégâts d'attaque
Archer_test possède 4 flèches

Initialisation d'un nouvel lapin :
Le lapin possède 20 points de vie
Modificaition les points de vie de lapin :
L'information de monstre
ptVie =  30
pourcentageAtt =  5
pourcentagePar = 19
degAtt = 6
pos = [0 ; 10]
Test du constructeur de recopie :
Personnage se situe en [10 ; 2]
Personnage possède 100 points de vie et 0 dégâts d'attaque
Personnage se situe en [0 ; 0]
Personnage possède 100 points de vie et 0 dégâts d'attaque
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 1.177s
```

5) Conclusion

Ce TP nous a permis de mettre en oeuvre l'héritage, la redéfinition et la surcharge de méthodes. Nous avons aussi pu utiliser les fonctionnalités de l'IDE pour générer très facilement les getter et les setter (vu au dernier TP). Nous avons aussi utilisé la méthode **toString()** pour afficher nos objets qui

est plus pratique que d'utiliser les getter (sans en oublier) dans un System.out.println. Enfin, nous avons implémenté un constructeur de recopie profonde fonctionnel.