

TP2 : Début du projet “World of Ecn”

Compréhension diagramme de classe UML

-

Écriture des premières classes Java

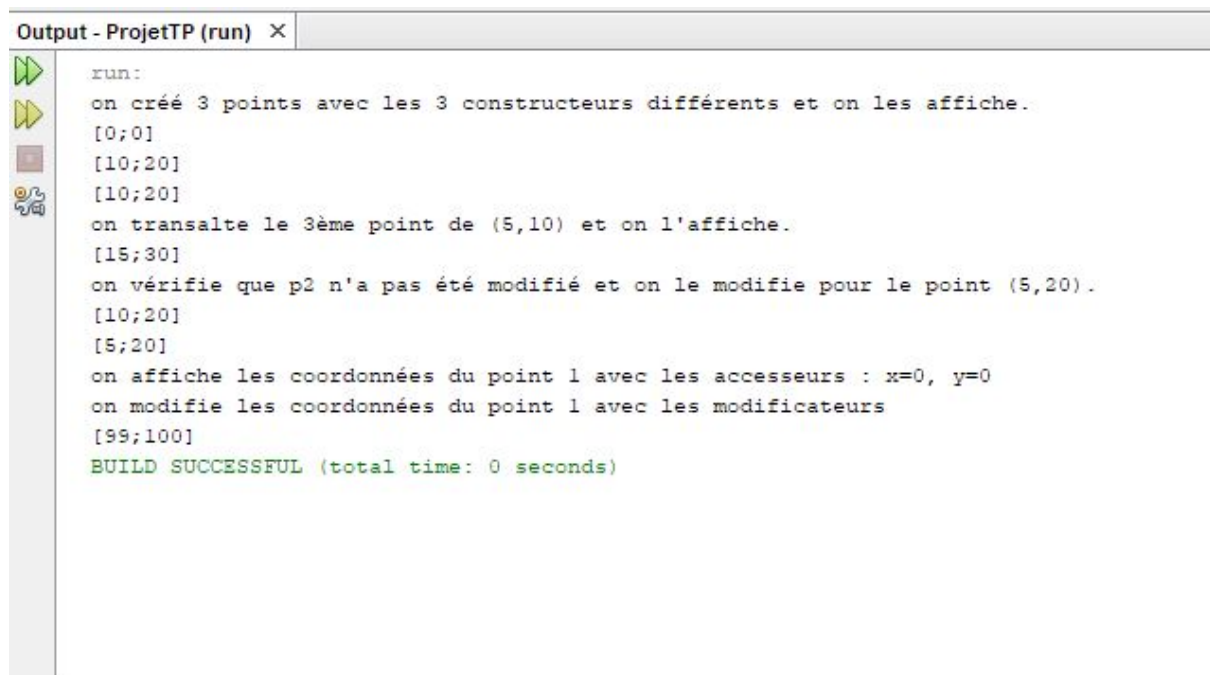
Ce TP a pour but de poser les bases des prochains TP, qui consisteront à la création d'un mini jeu-vidéo de type RPG. Il est très important puisque l'on va ici développer les premières briques sur lesquelles certains aspects clé du jeu reposent, il est donc primordial de bien réfléchir au modèle de donnée que l'on utilise afin de ne pas refaire de modification qui pourraient mettre en péril la suite du développement.

C'est dans cette optique que nous allons particulièrement nous intéresser au diagramme UML choisi, qui définit nos premières classes, ainsi qu'au principe d'héritage.

Nb : Vous pourrez retrouver tout notre code sur ce [github](https://github.com/Schipsi/ProjetTP)
<https://github.com/Schipsi/ProjetTP> en plus de l'archive jointe.

Mise en place du projet :

Tout d'abord, on crée un nouveau projet Java que l'on nomme ProjetTP, et où l'on récupère les fichiers **Point2D.java** et **TestPoint2D.java** réalisés précédemment.



```
Output - ProjetTP (run) X
run:
on crée 3 points avec les 3 constructeurs différents et on les affiche.
[0;0]
[10;20]
[10;20]
on translate le 3ème point de (5,10) et on l'affiche.
[15;30]
on vérifie que p2 n'a pas été modifié et on le modifie pour le point (5,20).
[10;20]
[5;20]
on affiche les coordonnées du point 1 avec les accesseurs : x=0, y=0
on modifie les coordonnées du point 1 avec les modificateurs
[99;100]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Le tout semble fonctionner correctement comme on peut le voir sur l'image ci-dessus.

Analyse du diagramme UML :

Ce diagramme contient tout d'abord les classes **Point2D**, **TestSeance1** et **World**, qui ont déjà été abordées dans le TP précédent ou sur le sujet, et nous ne les regarderons donc pas plus en détail.

Nous avons ensuite la classe **Personnage** que nous avons choisi de définir en tant que classe abstraite car nous considérons qu'un personnage a forcément une sous-classe, ainsi que les classes Archer et Paysan qui héritent de la classe **Personnage**. La classe personnage contient toutes les informations de base, comme les point de vie, la valeur d'attaque, etc. Cela permet de regrouper les attributs partagés par tous les personnages, tout en définissant seulement les attributs et méthodes qui nous intéressent pour une classe spécifique : c'est le cas par exemple pour l'attribut **nbFleches** de l'archer et ses méthodes associées **getNbFleches()** et **setNbFleches(int)**.

Cette architecture est d'autant plus intéressante qu'il sera simple par la suite de rajouter d'autres classes de personnages grâce à l'héritage de la classe Personnage. De la même manière, une méta classe **Monstre** (abstraite elle aussi) a été créé qui est l'équivalent de la classe **Personnage** mais pour les entités non-joueur. Un exemple de monstre a aussi été défini, la classe **Lapin**, qui hérite de la classe **Monstre** sans autres attributs. Ici aussi, il sera très facile par la suite de rajouter d'autres monstres par la suite.

Une alternative à ce diagramme aurait pu être de créer un classe **Entité**, qui regroupe à la fois les **Personnage** et les **Monstre** et contenant les attributs communs de ces deux classes, comme le nombre de points de vie par exemple.

Pour la génération des personnages et des monstres, il est à noter que nous avons généré en premier l'archer, puis le paysan à moins de 5 unité de distance (aléatoirement, en le recréant s'ils sont sur la même case), puis le lapin en suivant le même principe, en vérifiant la collision et la distance entre le lapin et le paysan.

Cet façon de faire est loin d'être efficace, il serait plus sûr de fonctionner avec des listes de personnages et de monstres indépendantes de la classe Monde ainsi que des méthodes **collision(Point2D, Point2D)** ainsi que d'autres comme **distance(Point2D, Point2D)**. Dans le futur la classe **World** va sans doute être totalement remanié.

On obtient le résultat suivant en générant les personnages dans le monde et en affichant respectivement l'archer, le paysan et le lapin.

Nous avons créé une méthode affiche spécifique pour le lapin, mais pas pour le paysan : on peut voir que celui-ci utilise celle présente par défaut dans la classe **Personnage**.

```
Output - ProjetTP (run) X
run:
Je suis un archer, je possède 16 fleches, 99 points de vie et 98 points de mana.
Je suis actuellement en [30,30] et je suis en mesure de t'infliger 13 dégats avec mes attaques et 4 dégats avec ma magie
(probabilité respective de toucher de 80 et 2)
Je suis un personnage, je possède 20 points de vie et 16 points de mana.
Je suis actuellement en [31,34] et je suis en mesure de t'infliger 2 dégats avec mes attaques et 1 dégats avec ma magie
(probabilité respective de toucher de 50 et 0)
Bonjour gentil voyageur , je suis un gentil Lapin et voici mes stats :
Point de Vie :6
Dégat d'attaque :1
Pourcentage d'attaque :20
pourcentage de Parade :10
position : (28,33)
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ce TP nous a permis de mettre en application des cas simple d'utilisation de l'héritage, en prenant en main à la fois l'utilisation de méthode spécifique pour une classe et l'héritage de ces méthodes par d'autres classes. Il nous fait questionner sur comment mettre en place de l'UML car celui fourni peut être amélioré, et certaines méthodes ne sont pas clairement spécifié (par exemple la méthode affiche).

Nous en avons aussi un peu plus appris sur la génération de nombre aléatoire, qui est une notion importante à maîtriser et à prendre en compte si l'on veut une application fonctionnelle et non un aléatoire "faussé".