

TP5 – Programmation Objet

Introduction

Dans le TP précédent, nous avons ajouté dans WoE la possibilité de gérer un très grand nombre de personnages grâce aux collections de Java. Dans ce TP, nous allons permettre à un joueur humain de jouer à WoE en contrôlant un personnage. Nous allons aussi générer des exceptions et apprendre à les gérer avec des blocs try/catch/finally.

1) Mise à jour du diagramme de classe UML

a) Commentaire du diagramme

Les classes **Creature**, **Personnage** et **Monstre** sont devenues abstraites pour 2 raisons :

- Elles n'ont pas d'intérêt à être instanciées, l'abstraction permet d'interdire cette instanciation.
- Certaines de leurs méthodes sont redéfinies systématiquement par leurs classes filles, on peut donc les définir comme abstraites pour qu'elles soient utilisables grâce au polymorphisme mais aussi qu'elles soient spécifiques à chaque sous-classe. Exemple : on peut définir la méthode d'affichage de **Creature** comme abstraite pour que **Personnage** et **Monstre** l'implémentent différemment, mais lorsqu'on manipulera un objet de type **Creature** on pourra appeler la méthode d'affichage.

L'interface **Deplacable** permet de s'assurer que tout créature aura une fonction de déplacement. L'intérêt d'utiliser une interface est de pouvoir l'utiliser avec une autre classe ultérieurement, même si elle n'est pas dans la hiérarchie de **Creature**.

L'interface **Combattant** permet elle de s'assurer que certaines sous-classes de **Creature** sont capables de se battre. Si on avait utilisé une méthode abstraite dans **Creature**, on aurait imposé à toutes les sous-classes d'implémenter cette méthode pour être instanciables. Pour **Paysan**, on aurait dû écrire une "fausse" méthode qui ne fait rien.

Pour gérer un nombre inconnu à l'avance de créatures et d'objets, nous allons utiliser des collections, et plus précisément des ArrayList afin de pouvoir facilement ajouter et accéder à des personnages ou des objets. Au vu des modifications du diagramme de classe, nous allons modifier le choix fait au TP précédent qui était d'avoir une ArrayList par sous-classe (Archer, Guerrier, Loup, Mana, ...). L'implémentation choisie est précisée en 2.c).

b) Implémentation de la classe Joueur

Nous avons créé une méthode **creeJoueur** dans **World** qui demande à l'utilisateur de choisir sa classe et le nom de son personnage. Cette méthode est ensuite appelée dans **creeMondeAlea** autant de

fois qu'il y a de joueurs. Nous n'avons pas eu le temps d'initialiser les attributs de chaque personnage de manière aléatoire, mais il suffit pour cela de changer le constructeur par défaut de chaque classe.

2) Nouvelle mise à jour du diagramme de classe UML

a) Nouveautés

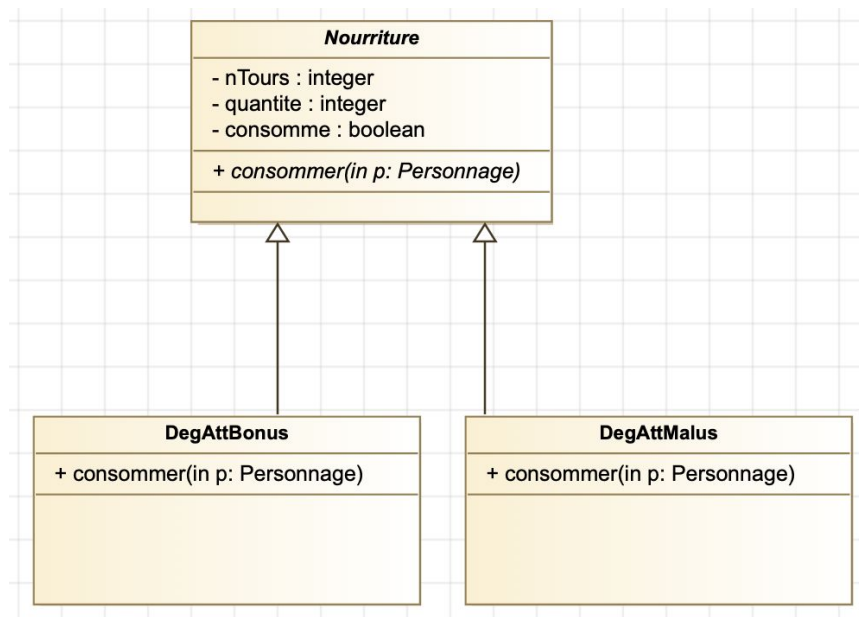
La super-classe **ElementDeJeu** a été ajoutée pour pouvoir factoriser du code entre les objets et les créatures. Nous avons par exemple choisi de mettre un attribut **position** dans cette super-classe. On peut aussi définir une méthode abstraite **affiche** qui sera définie dans les sous-classes. Elle n'implémente pas **Deplacable** ni **Combattant** car cela voudrait dire que toutes ses sous-classes doivent implémenter les méthodes de ces interfaces, ce qui n'est pas le comportement voulu.

La classe **NuageToxique** hérite d'**Objet** qui hérite d'elle même de **ElementDeJeu**, ce qui lui permet d'avoir une position sur le plateau, et aussi des caractéristiques d'un objet. Elle implémente des interface car le nuage toxique va pouvoir se déplacer durant un tour de jeu mais aussi attaquer les créatures. Implémentation :

- combattre : Nous vérifions que **NuageToxique** est dans la même position que **Creature**(Comme le nuage est devant sa tête). Si oui, il va attaquer. Sinon il ne va pas attaquer.
- déplacer : Nous vérifions que **NuageToxique** est dans le plateau d'abord, puis il peut déplacer dans le plateau.

b) Nourriture

Nous proposons l'implémentation suivante pour **Nourriture** et des bonus/malus.



La méthode **consommer** va faire diminuer de 1 la quantité **nTours** (nombre de tours restants de l'effet) et diminuer ou augmenter la caractéristique concernée du personnage *p* si **consomme** est faux (la nourriture n'a pas encore été consommée). Quand **nTours** vaudra 0, on restaurera à la caractéristique concernée la quantité déduite. Nous n'avons implémenté que ces deux classes qui donnent un bonus/malus sur la compétence de dégât d'attaque.

Il faudra donc à chaque début de tour de jeu parcourir l'attribut de type `List<Nourriture>` de chaque personnage et appliquer la méthode **consommer**.

c) Modifications de nos choix d'implémentation

Ces modifications du diagramme de classe nous ont poussé à changer la manière dont nous gérons les entités sur le plateau.

Nous allons utiliser une `LinkedList` pour gérer l'ensemble des positions occupées sur le plateau. Cette structure est adaptée car on va supprimer une position de la liste quand un élément du plateau se déplace, puis ajouter sa nouvelle position.

La fonction **déplacer** prendra donc en paramètre une liste de positions occupées sur lesquelles l'élément déplaçable ne peut pas se déplacer. Pour le moment, nous ne permettons pas le déplacement sur une case choisie, le déplacement se fait de manière aléatoire sur l'une des cases adjacentes libres. Pour le simplifier, le déplacement n'est pas aléatoire, nous vérifions la position est occupée ou pas, si la position est occupée, nous allons imprimer "Déplacement impossible! La case est occupée."

Nous allons utiliser une `LinkedList` pour gérer les créatures, une pour les joueurs et une pour les objets. La structure de liste est ici plus adaptée que celle de tableau car il y aura des suppressions d'éléments en cours de jeu (quand une créature meurt ou un objet est consommé).

Il nous a semblé plus adapté d'utiliser la généricité pour manipuler nos objets. Par exemple, si on cherche l'ensemble des créatures qui sont à distance d'attaque d'un personnage donné, il est plus simple de ne parcourir qu'une seule liste. De plus, si on veut ajouter une nouvelle classe dans notre jeu, nous n'aurons pas à créer un nouvel attribut et à modifier toutes les méthodes de **World**.

La génération du monde aléatoire est alors faite en initialisant un nombre aléatoire d'archers, de paysans, de loups, de potions de soins, ... ce qui est assez laborieux. Une méthode plus efficace (mais que nous n'avons pas eu le temps d'implémenter) serait d'associer une probabilité d'apparition à chaque créature et chaque objet puis de générer les créatures et les objets grâce à ces probabilités et à la réflexivité (avec une liste de classes de créatures et une liste de classes d'objets).

Pour cette génération aléatoire, nous avons ajouté les attributs **maxCrea** et **maxObj** qui fixe le nombre maximal de créatures et d'objets dans chaque classe. Nous avons aussi ajouté **nJoueurs** qui spécifie le nombre de joueurs humains dans le jeu.

d) Tests

Test de la génération aléatoire du monde et de l'initialisation des joueurs :

Dans ce test, on génère un nombre aléatoire d'archers, de guerriers, de lapins, de potions, ... et on initialise un joueur.

On vérifie ensuite que toutes les positions sont bien distinctes, et on affiche le nombre d'entités générées ainsi que le premier élément de chaque liste pour montrer qu'ils sont bien initialisés.

```
Test de la création aléatoire du monde :
Joueur 0, entrez vos choix.
Quel classe désirez-vous jouer ?
Choix possibles : Archer, Guerrier, Mage|
Mage
Entrez un nom pour votre personnage :
merlin
Est-ce-qu'il y a des positions en double ? false

Personnages des joueurs :
Nom : merlin
Points de vie : 100
Position : [46 ; 21]
Points d'attaque : 15
Pourcentages d'attaque : 70
Points de parade : 5
Pourcentages de parade : 60
Points de mana : 7
Dégâts de magie : 20
Pourcentage de magie : 80
Pourcentage de résistance à la magie : 30
Distance d'attaque maximale : 3

Nombre de créatures générées : 90
Première créature de la liste :
Nom : Archer
Points de vie : 100
Position : [45 ; 6]
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
Nombre d'objets générés : 13Premier objet de la liste :
Soin se trouve en [2 ; 17]
Elle rend 20 points de vie
```

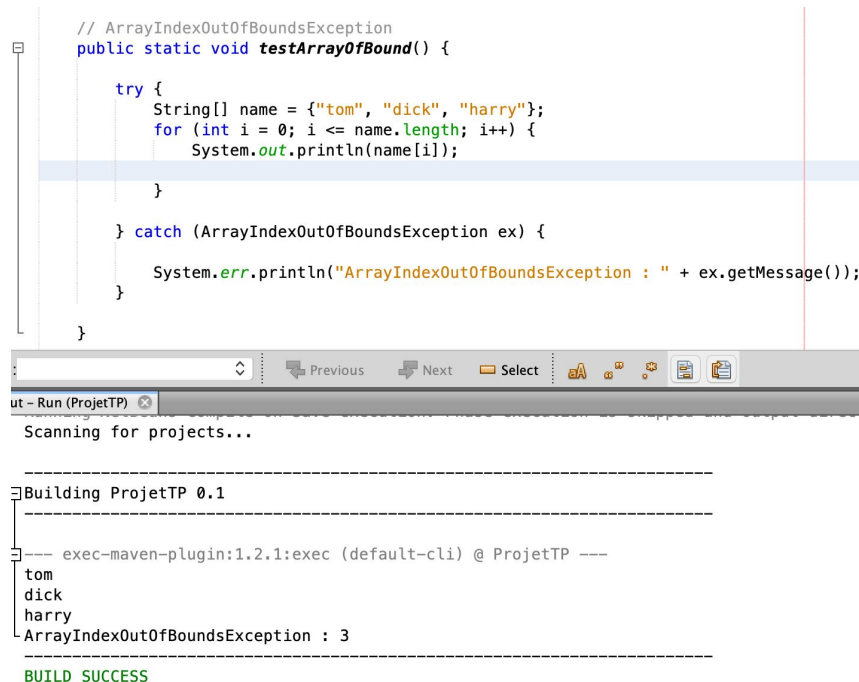
3) Exceptions

a) NullPointerException

```
// NullPointerException
public static void testNullPointerException() {
    try {
        World w = null;
        w.creeMondeAlea();
    } catch (NullPointerException ex) {
        System.err.println("NullPointerException : " + ex.getMessage());
    }
}
```

Nous avons défini classe **World** comme NULL, c'est compilé mais il va déclarer "NullPointerException : null". Si nous ne définissons pas EXCEPTION dans CATCH, il ne va pas compiler.

b) ArrayIndexOutOfBoundsException



```
// ArrayIndexOutOfBoundsException
public static void testArrayOfBound() {
    try {
        String[] name = {"tom", "dick", "harry"};
        for (int i = 0; i <= name.length; i++) {
            System.out.println(name[i]);
        }
    } catch (ArrayIndexOutOfBoundsException ex) {
        System.err.println("ArrayIndexOutOfBoundsException : " + ex.getMessage());
    }
}
```

Scanning for projects...

Building ProjetTP 0.1

--- exec-maven-plugin:1.2.1:exec (default-cli) @ ProjetTP ---

tom
dick
harry
ArrayIndexOutOfBoundsException : 3

BUILD SUCCESS

La longueur de table "name" est 3, mais dans la boucle on commence par 0, c'est-à-dire qu'on doit arrêter la boucle à i=2, pas à i=3

c) ArithmeticException

```
// ArithmeticException
public static void testArithmeic() {
    try {
        int x = 19;
        int y = 0;
        int z = x/y;
        System.out.println("La valeur de x est " + z);
    } catch (ArithmeticException ex) {
        System.err.println("ArithmeticException : " + ex.getMessage());
    }
}
```

C'est une opération arithmétique illégale lancée parce que le dénominateur ne peut pas être 0.

d) ClassCastException

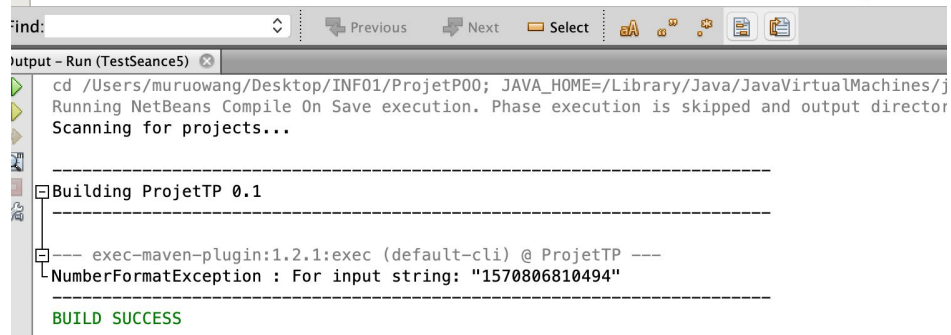
```
// ClassCastException
public static void testClassCast() {
    try {
        Object i = Integer.valueOf(42);
        String s = (String) i;
    } catch (ClassCastException ex) {
        System.err.println("ClassCastException : " + ex.getMessage());
    }
}
```



String n'est pas une sous-classe de **Int**, donc ce n'est pas impossible de caster **Int** comme **String**

e) NumberFormatException

```
// NumberFormatException
public static void testNumberFormat() {
    try {
        String today = "" + System.currentTimeMillis();
        int todayInt = Integer.parseInt(today);
    } catch (NumberFormatException ex) {
        System.err.println("NumberFormatException : " + ex.getMessage());
    }
}
```



Le chiffre est supérieur que le maximum de type **Int**, il faut utiliser type **Long**.

Après la modification, le résultat est suivant :

```
// NumberFormatException
public static void testNumberFormat() {
    try {
        String today = "" + System.currentTimeMillis();
        long todayLong = Long.parseLong(today);
        System.out.println("L'heure est " + todayLong);
    } catch (NumberFormatException ex) {
        System.err.println("NumberFormatException : " + ex.getMessage());
    }
}
```

```
cd /Users/muruowang/Desktop/INF01/ProjetP00; JAVA_HOME=/Library/Java/JavaVir
Running NetBeans Compile On Save execution. Phase execution is skipped and o
Scanning for projects...

-----
Building ProjetTP 0.1
-----
--- exec-maven-plugin:1.2.1:exec (default-cli) @ ProjetTP ---
L'heure est 1570807064443
-----
BUILD SUCCESS
```

f) StackOverflowError

```
// StackOverflowError
public static void testStackOverflow() {
    try {
        testStackOverflow();
    } catch (StackOverflowError ex) {
        System.err.println("StackOverflowError : " + ex.getMessage());
    }
}
```

```
cd /Users/muruowang/Desktop/INF01/ProjetP00; JAVA_HOME=/Library/Java/JavaVirtualM
Running NetBeans Compile On Save execution. Phase execution is skipped and output
Scanning for projects...

-----
Building ProjetTP 0.1
-----
--- exec-maven-plugin:1.2.1:exec (default-cli) @ ProjetTP ---
StackOverflowError : null
```

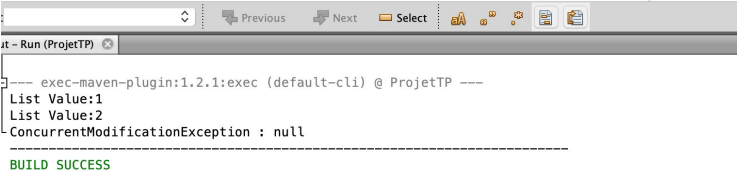
Dans ce programme, la méthode `testStackOverflow()` est appelée dans une boucle infinie. La méthode va être ajoutée dans thread stack infiniment.

g) `ConcurrentModificationException`

```
// ConcurrentModificationException
public static void testConcurrentModifiaction() {
    try {
        List<String> list = new ArrayList<>();
        list.add("1");
        list.add("2");
        for (Iterator<String> it = list.iterator(); it.hasNext(); ) {
            String value = it.next();

            if (value.equals("2")) {
                it.remove();
                list.add("6");
            }

            System.out.println("List Value:" + value);
        }
    } catch (ConcurrentModificationException ex) {
        System.err.println("ConcurrentModificationException : " + ex.getMessage());
    }
}
```

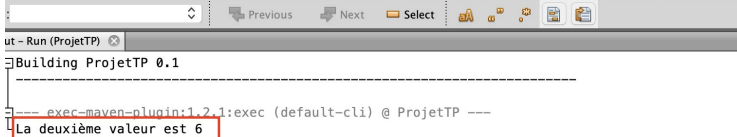


Ici, nous voudrions supprimer la valeur "2" et la remplacer par "6". On ne peut pas supprimer un élément d'une Collection quand on est en train de parcourir la liste.

Après générer un nouveau tableau, on peut parcourir le tableau précédent et faire la modification dans le tableau actuel, le résultat est suivant:

```
// ConcurrentModificationException
public static void testConcurrentModifiaction() {
    try {
        List<String> list = new ArrayList<>();
        list.add("1");
        list.add("2");

        List<String> newList = list.stream()
            .map(s -> s.equals("2") ? "6" : s)
            .collect(Collectors.toList());
        System.out.println("La deuxième valeur est " + newList.get(1));
    } catch (ConcurrentModificationException ex) {
        System.err.println("ConcurrentModificationException : " + ex.getMessage());
    }
}
```



Conclusion

Dans ce TP, nous avons défini certaines classes comme “Abstract”. “Interface” nous permet de ne pas oublier d’implémenter des méthodes et peut concerner différentes classes qui ne font pas partie de la même hiérarchie. Nous avons aussi changé la structure de notre classe **World** pour tirer parti de la généricité en Java, et nous avons commencé à implémenter la possibilité pour un joueur humain de contrôler un personnage. Enfin, nous avons généré des exceptions et mis en oeuvre des blocs try/catch/finally pour les récupérer et les traiter.