

Programmation Orientée Objet (OBJET)

TP 6 : Finalisation de « World of ECN »

Enregistrement/Chargement de sauvegardes –
Finalisation du projet

Jean-Marie Normand – Bureau E211

jean-marie.normand@ec-nantes.fr



Instructions

- Suivez les slides les uns après les autres
- A la fin de chaque séance de TP, vous devrez nous rendre un rapport par binôme
- Ce rapport devra contenir :
 - Une introduction et une présentation rapide du sujet de la séance
 - Les réponses aux questions posées dans les slides repérés par une icône de panneau STOP
 - Une conclusion
- La notation tiendra compte du respect de ces consignes



1^{RE} PARTIE : MISE À NIVEAU (RATTRAPAGE DU RETARD ÉVENTUEL)

Avant toute chose

- Assurez vous d'avoir bien terminé les séries de TP précédentes
- Si ce n'est pas le cas, prenez le temps d'arriver dans un **état fonctionnel de WoE**
- En particulier avec une version permettant à un humain de contrôler un **Personnage** au clavier

2^E PARTIE : ENREGISTREMENT/CHARGEMENT DE SAUVEGARDES EN MODE TEXTE

Sauvegardes en mode texte

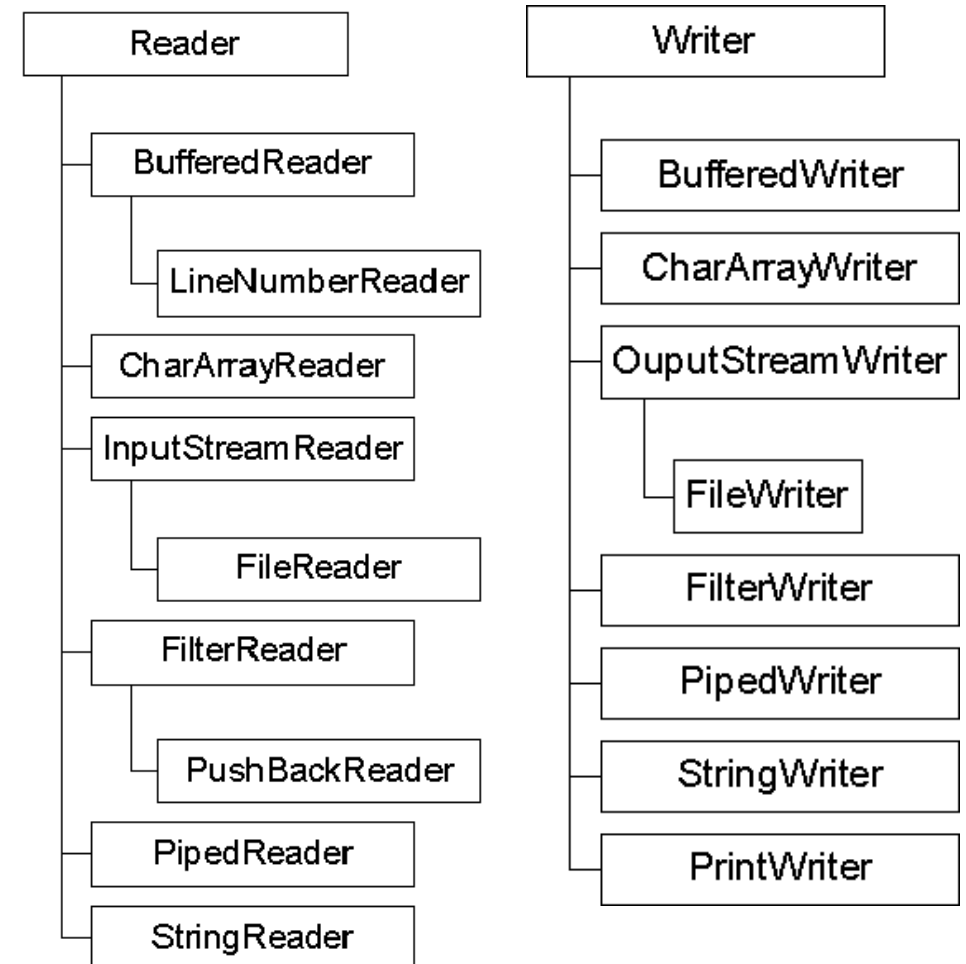
- Afin de pouvoir sauvegarder et reprendre une partie nous souhaitons utiliser des fichiers texte
- Nous proposons donc un format de fichier (décrit plus loin)
- Nous allons utiliser les mécanismes d'entrées/sorties de Java afin de pouvoir lire et écrire dans des fichiers
- Ces mécanismes se trouvent dans le paquetage `java.io` dont la documentation en ligne est ici :
- <http://docs.oracle.com/javase/8/docs/api/java/io/package-summary.html>
- Dans ce projet nous nous limiterons aux fichiers texte contenant uniquement des caractères et des chaînes de caractères, les principales classes que nous manipulerons seront ainsi (voir leur documentation en ligne) :
 - `File`, `FileReader`, `FileWriter`
 - `BufferedReader`, `BufferedWriter`

Les flux en Java

- Java utilise des **flux** (**stream** en VO) pour gérer les entrées/sorties entre un programme Java et une source extérieure (autre application Java, fichier stocké sur un disque dur, des informations stockées dans la mémoire vive, connexion réseau, etc.)
- Les flux permettent de gérer ces échanges de données et ce de **manière toujours séquentielle**
- On distingue généralement :
 - Les flux d'entrée (**input stream**)
 - Les flux de sortie (**output stream**)
 - Les flux de **traitement de caractères** (données textuelles)
 - Les flux de **traitement d'octets** (données brutes)
- Dans la suite nous nous intéresserons aux **flux de traitement de caractères** et utiliserons un type de flux d'entrée (**lecture d'une sauvegarde**) et un type de flux de sortie (**écriture d'une sauvegarde**)

Les flux de traitement de caractères

- Flux transportent des données sous forme de caractères
- Classes Java qui gèrent ces flux héritent de 2 classes abstraites : **Reader** et **Writer**
- Beaucoup de classes ayant des caractéristiques différentes → utilisation dépend des besoins du programmeur !
- Attention ! **La plupart des méthodes de ces classes sont susceptibles de lever des exceptions !**



Et pour WoE ?

- Avant d'aller plus loin sur les classes Java permettant de gérer des flux : intéressons nous au format de fichier que nous allons utiliser pour les sauvegardes de partie et le chargement d'une partie sauvegardée
- Ce format sera volontairement simpliste mais pourra nous faire manipuler les flux en Java

Format du fichier de sauvegarde

- Avant d'aller plus loin sur les flux, intéressons nous au format du fichier de sauvegarde
- Description succincte du format :
 - Chaque ligne du fichier texte représente une seule information sur le plateau de jeu
 - Une information peut correspondre à :
 - La largeur du plateau de jeu
 - La hauteur du plateau de jeu
 - Un des éléments du jeu :
 - Un Personnage
 - Un Monstre
 - Un Bonus
 - Etc.
 - Une ligne se suffit à elle-même, par exemple un Personnage (quelle que soit sa classe) sera décrit sur une seule ligne

Exemple de fichier de sauvegarde

- Vous trouverez un exemple de fichier de sauvegarde dans [Sauvegarde-WoE.txt](#) sur Hippocampus
- **Attention ! L'ordre des valeurs peut dépendre de vos constructeurs !**
- Nous considérons ici l'ordre suivant pour les Personnages :
 - **nom ptVie pt Magie %Att %Par %Magie %ResistMagie degAtt degMagie distMax ptParade posX posY**
 - La caractéristique **ptPar** n'apparaît pas forcément en avant dernière position dans vos constructeurs
- De plus les [Nourriture\(s\)](#) sont manquantes car nous ne connaissons pas le nom des classes que vous avez utilisées

```
Largeur 25
Hauteur 25
Guerrier grosBill 250 0 80 60 0 10 15 0 1 5 8 3
Mage merlin 50 50 20 25 85 80 2 20 7 2 15 22
Voleur arsene 75 0 60 80 0 25 15 0 1 5 5 7
Archer robin 75 0 50 60 0 25 10 0 10 5 20 11
Paysan peon 25 0 0 30 0 0 1 0 1 5 3 6
Loup 30 50 50 25 5 19 3
Lapin 4000 90 1 1000 1 10 10
Loup 80 75 30 45 5 2 13
Loup 30 30 30 20 10 5 14
Lapin 30 20 20 40 10 23 23
NuageToxique 50 5 20 5
Soin 25 18 18
Mana 50 4 4
Soin 100 4 20
Mana 100 7 1
Soin 48 1 19
Mana 24 0 0
Joueur Guerrier bob 150 0 90 50 0 10 20 0 1 5 12 12
```

Exemple de fichier de sauvegarde (2)

- Explications :
 - La ligne « **Guerrier ...** » représente un **Personnage** de type **Guerrier** dont le nom est « grosBill » ayant 250 points de vie, 0 points de magie, 80% d'attaque, 60% de parade, 0% de magie, 10% de résistance à la magie, 15 points de dégâts d'attaque, 0 points de dégâts magiques, 1 de distance maximum, 5 points de parade et se trouvant en position [8,3]
 - La ligne « **Mage ...** » représente un **Personnage** de type **Mage** dont le nom est « merlin » ayant 50 points de vie, 50 points de magie, 20% d'attaque, 25% de parade, 85% de magie, 80% de résistance à la magie, 2 points de dégâts d'attaque, 20 points de dégâts magiques, 7 de distance maximum, 2 points de parade et se trouvant en position [15,22]
 - La ligne « **Voleur ...** » représente un **Personnage** de type **Voleur** dont le nom est « arsene » ayant 75 points de vie, 0 points de magie, 60% d'attaque, 80% de parade, 0% de magie, 25% de résistance à la magie, 15 points de dégâts d'attaque, 0 points de dégâts magiques, 1 de distance maximum, 5 points de parade et se trouvant en position [5,7]
 - La ligne « **Archer ...** » représente un **Personnage** de type **Archer** dont le nom est « robin » ayant 75 points de vie, 0 points de magie, 50% d'attaque, 60% de parade, 0% de magie, 25% de résistance à la magie, 10 points de dégâts d'attaque, 0 points de dégâts magiques, 10 de distance maximum, 5 points de parade et se trouvant en position [20,11]

Exemple de fichier de sauvegarde (3)

- Explications :
 - La ligne « **Paysan ...** » représente un **Personnage** de type **Paysan** dont le nom est « peon » ayant 25 **points de vie**, 0 **points de magie**, 0% **d'attaque**, 30% **de parade**, 0% **de magie**, 0% **de résistance à la magie**, 1 **points de dégâts d'attaque**, 0 **points de dégâts magiques**, 1 de **distance maximum**, 5 **points de parade** et se trouvant en **position** [3,6]
 - La ligne « **Lapin 4000 ...** » représente une **Creature** de type **Lapin** ayant 4000 **points de vie**, 90% **d'attaque**, 1% **de parade**, 1000 **points de dégâts d'attaque**, 1 **points de parade** et se trouvant en **position** [10,10]
 - La ligne « **Loup 30 ...** » représente une **Creature** de type **Loup** ayant 30 **points de vie**, 50% **d'attaque**, 50% **de parade**, 25 **points de dégâts d'attaque**, 5 **points de parade** et se trouvant en **position** [19,3]
 - La ligne « **NuageToxique ...** » représente un **Objet** de type **NuageToxique** ayant 50% **d'attaque**, 50% **de parade**, 25 **points de dégâts d'attaque**, 5 **points de parade** et se trouvant en **position** [20,5]
 - La ligne « **Soin 25 ...** » représente un **Objet** de type **Soin** (dont la classe mère directe est **Potion**) permettant de redonner 25 **points de vie** et se trouvant en **position** [18,18]
 - La ligne « **Joueur ...** » représente le **Joueur humain** dont l'attribut **perso** est de type **Guerrier** (voir plus haut pour la description correspondantes) et se trouvant en **position** [12,12]

3^E PARTIE : CHARGEMENT D'UN FICHER DE SAUVEGARDE EXISTANT

Chargement d'un fichier de sauvegarde

- Cette partie abordera 3 problématiques :
 - Le fonctionnement de la classe `BufferedReader`, qui permet de lire un **fichier texte ligne par ligne** en **retournant une chaîne de caractères par ligne**
 - Comment **parcourir une chaîne de caractères** contenant une ligne entière d'un fichier texte (i.e. **comment découper cette chaîne mot par mot**)
 - Comment **mettre en œuvre une sauvegarde** de notre partie de WoE et les conséquences sur nos classes
 - <http://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html>

Utilisation d'un `BufferedReader`

- La lecture d'un fichier texte en Java va s'effectuer en utilisant la classe `BufferedReader`
- Comme il existe de nombreuses classes pour la lecture de flux (voir slide n°8), voici pourquoi nous avons choisi `BufferedReader` :
 - elle nous permet de **lire un fichier ligne par ligne**
 - elle **offre de très bonnes performances**, via l'utilisation d'un tampon (*buffer*, l'explication de ces bonnes performances est hors programme)
- À suivre, un exemple d'utilisation de la classe `BufferedReader` (que vous trouverez aussi sur Hippocampus) qui va :
 - **ouvrir** un fichier texte nommé « **source.txt** »
 - **lire ce fichier ligne par ligne** avec la méthode `readLine()`
 - **afficher** chaque ligne du fichier qui vient d'être lue
- **Attention ! Dans NetBeans, pour que votre application trouve une ressource (par exemple un fichier texte qui vous souhaitez ouvrir) il doit se trouver à la racine de votre projet (i.e. au même niveau que les répertoires **src** **lib**, etc.)**

Utilisation d'un `BufferedReader` (2)

```
11  import java.io.*;
12
13  public class TestBufferedReader {
14      protected String source;
15
16      public TestBufferedReader(String source) {
17          this.source = source;
18          lecture();
19      }
20
21      public static void main(String args[]) {
22          TestBufferedReader testBufferedReader = new TestBufferedReader("source.txt");
23      }
24
25      private void lecture() {
26          try {
27              String ligne ;
28              BufferedReader fichier = new BufferedReader(new FileReader(source));
29              ligne = fichier.readLine();
30              while (ligne != null) {
31                  System.out.println(ligne);
32                  ligne = fichier.readLine();
33              }
34
35              fichier.close();
36          } catch (Exception e) {
37              e.printStackTrace();
38          }
39      }
40  }
```

Précisions sur l'exemple de `BufferedReader`

- Notons que la création d'un objet de type `BufferedReader` (cf. ligne 28 de l'exemple) passe par la création d'un objet de type `FileReader`
- Toutefois, comme celui-ci ne nous est pas indispensable, nous pouvons appeler le constructeur de `FileReader` à l'intérieur de l'appel du constructeur de `BufferedReader`
- De plus `BufferedReader` possède une méthode `readLine()` retournant une chaîne de caractères (objet de type `String`) correspondant à la ligne courante du fichier et décale un curseur interne lui permettant de parcourir tout le fichier

Décomposition d'une chaîne de caractères en sous-chaînes

- Nous avons maintenant une chaîne de caractères (un objet de type `String`) contenant une ligne complète de notre fichier
- On souhaite maintenant **découper cette chaîne de caractère mot par mot**
- Par exemple pour la ligne « `Largeur 10` » nous voulons pouvoir récupérer chacun des deux mots : `Largeur` et `10` !

StringTokenizer

- Pour ce faire nous allons utiliser une classe Java: `StringTokenizer`
- Elle permet de découper une chaîne de caractères (objet de type `String`) selon un **ensemble de délimiteurs de mots**
- Ces **délimiteurs sont des caractères** (par exemple l'espace, la virgule, et.)
- Les **unités lexicales résultantes** (les mots séparés par un des délimiteurs) peuvent donc **ensuite être parcourues pour traitement**
- <http://docs.oracle.com/javase/8/docs/api/java/util/StringTokenizer.html>

Utilisation d'un StringTokenizer

```
13 import java.util.StringTokenizer;
14
15 public class TestStringTokenizer {
16
17     public static void main(String args[]) {
18         String test = "Largeur 10";
19         String delimiters = " ,.:";
20
21         // on declare un 'tokenizer' qui va decouper
22         // 'test' en fonction de l'ensemble des delimiters
23         StringTokenizer tokenizer = new StringTokenizer(test, delimiters);
24
25         // Parcours de l'ensemble des unites lexicales de 'test'
26         // hasNextTokens() retourne 'vrai' tant qu'il reste des 'mots'
27         // dans 'test' separes par un des delimiters (espace,
28         // virgule, etc.) declares plus haut
29         while(tokenizer.hasMoreTokens()) {
30             // nextToken() retourne la prochaine unite lexicale decoupee par les delimiters
31             String mot = tokenizer.nextToken();
32             // pour l'exemple, on transforme 'mot' en lettres minuscules
33             mot = mot.toLowerCase();
34             // on affiche 'mot' qui est maintenant en minuscules
35             System.out.println(mot);
36         }
37     }
38 }
```

Résultat :

largeur

10

Chargement d'une sauvegarde de WoE !

- Vous avez toutes les informations nécessaires pour écrire une classe `ChargementPartie` qui :
 - A pour but de **charger un fichier de sauvegarde** et de
 - **Retourner un objet de type** `World` permettant ainsi de reprendre la simulation là où elle avait été enregistrée



Travail à faire (1)

- **Écrivez** la classe `ChargementPartie` ayant :
 - un attribut représentant le **nom du fichier de sauvegarde** à charger
 - un attribut de type `BufferedReader`
 - un **constructeur** prenant en paramètre le **nom du fichier à charger**
 - une méthode `chargerPartie()` retournant un objet de type `World` contenant l'ensemble des éléments du jeu qui étaient sauvegardés dans le fichier texte
- **Proposez** au lancement d'une partie la possibilité de charger une sauvegarde existante
- Veuillez à bien **commenter** votre code et à fournir la **Javadoc**
- **Lisez le slide suivant avant de commencer !**



Travail à faire (2)

- Vous pourrez avoir besoin d'une méthode `creerElementJeu(params)` vous permettant de gérer la création des différents éléments du jeu :
 - Les paramètres `params` sont laissés **volontairement libres** pour que vous choisissiez vous mêmes
 - **Justifiez** dans le rapport le pourquoi de ces paramètres et en quoi leur choix est judicieux
- Pour la création des `ElementDeJeu`, il existe plusieurs solutions possibles :
 1. **Parcourir** toute la ligne correspondant à une classe et **appeler explicitement le constructeur** de la classe en question
 2. **Écrire** dans chaque classe **un nouveau constructeur** prenant en paramètre une chaîne de caractères représentant la ligne correspondante à un élément de cette classe
 - **Commentez** ces deux solutions et choisissez-en une en **justifiant votre choix** !

4^E PARTIE : SAUVEGARDE D'UNE PARTIE

Création/Ecriture dans un fichier

- Nous savons donc maintenant charger un fichier de sauvegarde et manipuler des objets de types :
 - BufferedReader
 - StringTokenizer
- Nous allons maintenant pouvoir écrire une classe permettant la création d'un fichier respectant le format de sauvegarde et ainsi terminer notre mécanisme de sauvegarde/chargement de partie !

Utilisation d'un `BufferedWriter`

- De manière identique à la lecture d'un fichier texte en `Java`, basée sur l'utilisation de la classe `BufferedReader`, notre mécanisme de sauvegarde va être basé sur l'utilisation de la classe `BufferedWriter` (voir slide 8)
- `BufferedWriter` possède de nombreux avantages, en particulier :
 - elle nous permet **d'écrire un fichier ligne par ligne**
 - elle offre de **très bonnes performances**, via l'utilisation d'un tampon (*buffer*, l'explication de ces bonnes performances est hors programme)
- <http://docs.oracle.com/javase/8/docs/api/java/io/BufferedWriter.html>

Utilisation d'un `BufferedWriter` (2)

- À suivre, un exemple d'utilisation de la classe `BufferedWriter` (que vous trouverez aussi sur Hippocampus) qui va :
 - **ouvrir** un fichier texte nommé « **source.txt** »
 - **écrire une ligne** avec la méthode `write()`
 - **écrire un retour à la ligne** avec la méthode `newline()`
 - **écrire une deuxième ligne** avec la méthode `write()`
- Attention ! Dans NetBeans, pour que votre application trouve une ressource (par exemple un fichier texte qui vous souhaitez ouvrir) il doit se trouver à la racine de votre projet (i.e. au même niveau que les répertoires **src lib**, etc.)

Utilisation d'un `BufferedWriter` (3)

```
11 import java.io.*;
12
13 public class TestBufferedWriter {
14
15     public static void main(String args[]) {
16         BufferedWriter bufferedWriter = null;
17         String filename = "monFichier.txt";
18
19         try {
20             // Creation du BufferedWriter
21             bufferedWriter = new BufferedWriter(new FileWriter(filename));
22             // On ecrit dans le fichier
23             bufferedWriter.write("Ecriture ligne un dans le fichier");
24             bufferedWriter.newLine();
25             bufferedWriter.write("Ecriture ligne deux dans le fichier");
26         }
27         // on attrape l'exception si on a pas pu creer le fichier
28         catch (FileNotFoundException ex) {
29             ex.printStackTrace();
30         }
31         // on attrape l'exception si il y a un probleme lors de l'ecriture dans le fichier
32         catch (IOException ex) {
33             ex.printStackTrace();
34         }
35         // on ferme le fichier
36         finally {
37             try {
38                 if (bufferedWriter != null) {
39                     // je force l'écriture dans le fichier
40                     bufferedWriter.flush();
41                     // puis je le ferme
42                     bufferedWriter.close();
43                 }
44             }
45             // on attrape l'exception potentielle
46             catch (IOException ex) {
47                 ex.printStackTrace();
48             }
49         }
50     }
51 }
```

Précisions sur l'exemple de `BufferedWriter`

- Notons que la création d'un objet de type `BufferedWriter` (ligne 21) passe par la création d'un objet de type `FileWriter`
- Toutefois, comme celui-ci ne nous est pas indispensable, nous pouvons appeler le constructeur de `FileWriter` à l'intérieur de l'appel du constructeur de `BufferedWriter`
- De plus `BufferedWriter` possède :
 - une méthode `write()` prenant en paramètre une chaîne de caractères (objet de type `String`) et **écrivant cette chaîne dans le fichier**
 - une méthode `newline()` qui **écrit dans le fichier un caractère de retour à la ligne** (ces caractères variant d'un système à l'autre Java propose une méthode évitant ainsi les problèmes)
 - Si votre chaîne de caractères écrite avec la méthode `write()` possède déjà un caractère de retour à la ligne (comme par exemple « **\n** ») alors l'appel à la méthode `newline()` est optionnel

Précisions sur l'exemple de `BufferedWriter` (2)

- Voici un exemple de création de chaîne de caractères **avec** et **sans caractère de retour à la ligne** intégré à la chaîne de caractères

```
12 public class TestString {  
13  
14     public static void main(String args[]) {  
15         String sansRetourLigne = "Bla";  
16         String avecRetourLigne = "Bla\n";  
17     }  
18 }
```



Sauvegarde d'une partie de WoE !

- Vous avez toutes les informations nécessaires pour écrire une classe `SauvegardePartie` qui :
 - A pour but de **sauvegarder l'état courant d'une partie de WoE**
 - En **respectant le format de fichier** présenté ci-avant (voir slides 10-13)
 - **Doit permettre de choisir :**
 - si vous souhaitez **demander le nom de la sauvegarde** à l'utilisateur
 - **choisir un nom automatiquement** (en évitant d'écraser une sauvegarde déjà effectuée et donc en générant un nom unique et de manière automatique)



Travail à faire (1)

- **Écrivez** la classe `SauvegardePartie` ayant :
 - un attribut représentant le **nom du fichier de sauvegarde** à charger
 - un attribut de type `BufferedWriter`
 - un **constructeur** prenant en paramètre le **nom du fichier à sauvegarder**
 - une méthode `sauvegarderPartie()` prenant en paramètre un objet de type `World` contenant l'ensemble des éléments du jeu et éventuellement le nom du fichier
- **Proposez** à l'utilisateur à chaque tour de boucle la possibilité de sauvegarder la partie
- Veillez à bien **commenter** votre code et à fournir la **Javadoc**
- **Lisez le slide suivant avant de commencer !**



Travail à faire (2)

- Vous pourrez avoir besoin d'une méthode `getTexteSauvegarde()` vous permettant de gérer la sauvegarde des différents éléments du jeu :
 - Chaque **classe peut ainsi fournir la ligne de texte correspondant à sa propre sauvegarde !**
- **Veillez à bien détailler** dans le rapport :
 - les **choix** que vous faites
 - la **manière dont vous proposer de créer un nom unique** de manière automatique (p. ex. sauvegarde1.txt puis sauvegarde2.txt etc.)
 - listez également les **mécanismes de la POO** vous permettant de mettre en œuvre ces sauvegardes

5^E PARTIE : ILLUSTRATION D'UN CHARGEMENT ET D'UNE SAUVEGARDE DE PARTIE



Illustrez le bon fonctionnement du mécanisme de sauvegarde/chargement

- **Créez** une partie aléatoire (comme lors du TP précédent) avec :
 - Des **Personnages**, des **Creatures**, des **Objets** générés de manière aléatoire
 - Un Joueur humain
- **Faites jouer** plusieurs tours de jeu
- **Sauvegardez** la partie
- **Terminez** la partie
- **Relancez** le jeu et **chargez** la partie sauvegardée précédemment
- **Illustrez** bien le fait que les deux parties sont identiques

5^E PARTIE : FINALISATION DE WOE



Interface Graphique (Minimaliste)

- Afin de pouvoir jouer à WoE il nous faut une **interface graphique** même **minimaliste** (le mode texte est bien évidemment suffisant car nous n'avons pas vu les GUI en cours)
- Proposez un moyen de **visualiser le plateau de jeu de manière textuelle** (éventuellement en proposant une légende, p. ex. G = Guerrier, etc.)
- En plus de l'affichage du monde de WoE, veillez à bien **présenter au joueur humain les choix qui s'offrent** à lui à chaque tour de jeu (**déplacement/combat/sauvegarde**)



Finalisation de WoE

- Et voilà ! Nous avons vu (presque) tous les concepts que nous souhaitions aborder dans le projet du cours de Programmation Orientée Objet
- Maintenant vous devez avoir une **version minimaliste mais fonctionnelle** de WoE !
- Il ne vous reste plus qu'à **finaliser** les dernières **classes et méthodes** que vous avez pu laisser de côté
- Veillez à bien **commenter votre code** et à bien **écrire la Javadoc**
- Veillez aussi à **nettoyer votre code** !



Conclusion

- Ajoutez à votre rapport :
 - **L'illustration du bon fonctionnement** de votre fonction principale (sortie textuelle des tests effectués)
- Rendez une archive au format **.ZIP** nommée **OBJET-TP6-NomBinome1-NomBinome2.zip** (avec **NomBinome1 < nomBinome2** dans l'ordre **alphabétique**) contenant :
 - Votre rapport au format **.pdf**
 - Tous vos fichiers **.java**
 - **Veillez à bien avoir écrit la Javadoc** de tous les **attributs** de vos classes et des **principales méthodes** (déplacer, combattre, etc.)
 - **Faites générer la Javadoc** par NetBeans, **joignez** l'ensemble des fichiers résultats à l'**archive .zip dans un dossier documentation**
- Le respect de ces consignes est pris en compte dans la note !

