Programmation Orientée Objet (OBJET)

TP 6 : Finalisation de « World of ECN »

Enregistrement/Chargement de sauvegardes – Finalisation du projet

Jean-Marie Normand – Bureau A114
jean-marie.normand@ec-nantes.fr

1RE 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

2E PARTIE : ENREGISTREMENT/CHARGEMENT DE SAUVEGARDES

Sauvegarde/ Chargement

- Nous allons vous proposer 2 manières de sauvegarder et charger des parties de WoE
  1. Via des Bases de Données (cf. BDONN)
  2. Via des fichiers texte
- Nous vous proposons de choisir entre ce que vous souhaitez implémenter
- Implémenter les 2 solutions pourra s'avérer trop long pour le temps alloué au TP

2E PARTIE A : ENREGISTREMENT/CHARGEMENT DE SAUVEGARDES EN BASES DE DONNÉES

Stockage de WoE en Bases de Données

- Afin de pouvoir gérer la sauvegarde d'une partie et le chargement d'une partie précédemment sauvegardée, nous souhaitons le faire en utilisant des bases de données

- Vous utiliserez les modèles conceptuels et physiques que vous avez établis en BDONN pour sauvegarder une partie et pouvoir la charger ensuite

Connexion à une base de données

- Pour vous connecter à votre serveur de base de données, nous allons utiliser des fichiers de propriétés (`.properties`) qui permettent de stocker les paramètres d'une application plutôt de stocker ces valeurs « en dur » dans l'application
- Les fichiers de propriétés permettent de stocker des informations sous le format « clé-délimiteur-valeur », où le caractère délimiteur est souvent soit « `=` », soit « `:` » soit un `espace` ou une `tabulation` (mais peut être défini par l'utilisateur), p. ex.
  - `version=1.0` (la clé est « version » la valeur « 1.0 »)
  - `name : testApp` (la clé est « name » et la valeur est « testApp »
  - `date 2022-09-23` (la clé est « date » et la valeur est « 2022-09-23 »)
  - Les commentaires commencent par un `#` ou un `!`

Connexion à une base de données

- Pour charger les fichier de propriétés (`.properties`) nous allons utiliser la classe Java `ResourceBundle` ou plus précisément `PropertyResourceBundle`
- https://docs.oracle.com/javase/7/docs/api/java/util/ResourceBundle.html
- https://docs.oracle.com/javase/7/docs/api/java/util/PropertyResourceBundle.html
- Ces classes nous permettent de charger simplement des fichiers stockant des informations

Connexion à une base de données

- Afin de comprendre comment utiliser un fichier de propriétés (`.properties`) et une classe `ResourceBundle`, vous trouverez un exemple sur Hippocampus (fichier `UseProps.zip`)
- Nous vous invitons à lire le code et à essayer de le modifier pour voir comment utiliser un fichier `.properties` (rajouter des propriétés, modifier les valeurs ou les clés, etc.)

Connexion à une base de données

- Une fois le fonctionnement des `ResourceBundle` et les fichiers `.properties` expliquez nous :
  1. Comment vous pensez que ces fichiers peuvent être utilisés pour aider à vous connecter à des bases de données
  2. De quelles couples clé-valeur vous pensez avoir besoin pour vous connecter à des bases de données
  3. Pourquoi cette solution est meilleure que de rentrer les valeurs « en dur » dans votre code ?

Connexion à votre base de données !

- Avec ce que vous avez vu en BDONN, implémentez une connexion fonctionnelle à VOTRE base de données
- Listez les modifications que vous avez du apporter à vos classes Java ou à votre base de données pour pouvoir :
  1. Sauvegarder les informations de WoE dans votre base de données
  2. Pouvoir charger une partie de WoE à partir de votre base de données

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

Sauvegardes en mode texte

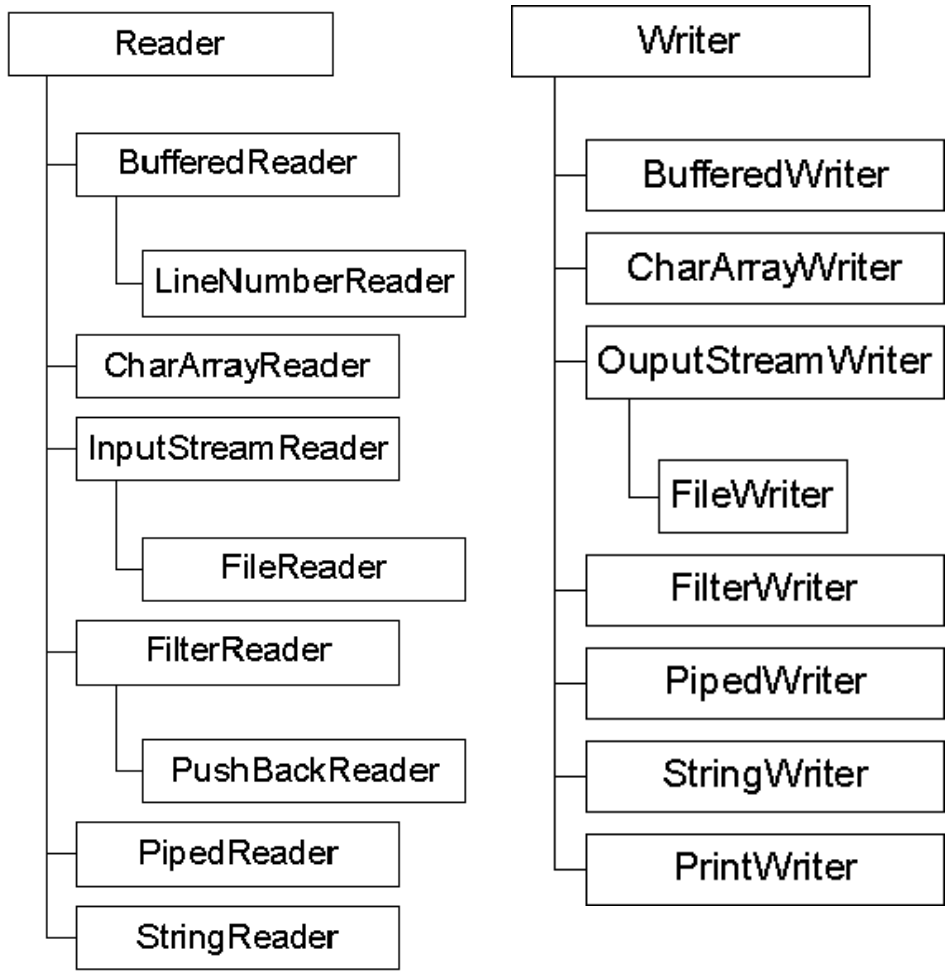
- Afin de pouvoir sauvegarder et reprendre une partie nous souhaitons utiliser des fichiers textuels (ce que l'on ne ferait pas dans de vrais projets)
- Nous proposons donc un format de fichier (décrit plus loin)
- Nous allons utiliser les mécanismes d'entrées/sorties de Java afin de pouvoir lire et 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 WoE, 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 une 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 → l'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

- Voici une description succincte du format (voir slides suivant pour 1 exemple) :
  - 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.
    - Le `Joueur` est toujours à la dernière ligne du fichier
    - L'`Inventaire` du joueur suit directement la description de celui-ci
    - Chaque ligne de l'`Inventaire` commence par « `Inventaire` » et est suivie de la description de l'objet correspondant
  - Une ligne se suffit à elle-même, par exemple un `Personnage` (quelle que soit sa classe) sera décrit sur une seule ligne **sauf pour le `Joueur` dont l'inventaire prend plusieurs lignes !**

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 degAtt ptParade %Att %Parade distAttMax posX posY
  - Attention à l'ordre des paramètres dansvos constructeurs !
- Les `Nourritures` (`Poisson`, `Miel`, `Eau`) sont indicatives car nous ne connaissons pas le nom des classes que vous avez utilisées

```
Largeur 25
Hauteur 25
Guerrier grosBill 250 10 8 80 60 1 8 3
Archer robin 75 20 5 75 30 5 5 20 15
Paysan peon 25 5 30 0 40 0 3 6
Loup 30 80 50 50 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
PotionSoin 25 18 18
Epee 50 12 18
PotionSoin 100 4 20
Poisson 20 2 13
PotionSoin 48 1 19
Miel 5 7 9
Joueur Guerrier bob 150 10 9 90 70 1 12 12
Inventaire PotionSoin 75
Inventaire Epee 125
Inventaire Eau 10
Inventaire PotionSoin 15
```

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, 10 points de dégâts d'attaque, 8 points de parade, 80% d'attaque, 60% de parade, 1 de distance d'Attaque maximum et se trouvant en position [8,3]

  - La ligne **« Archer … »** représente un `Personnage` de type `Archer` dont le nom est « robin » ayant 75 points de vie, 20 points de dégâts d'attaque, 5 points de parade, 75% d'attaque, 30% de parade, 5 de distance d'Attaque maximum, et se trouvant en position [5, 20] et ayant 15 flèches

  - La ligne **« Paysan … »** représente un `Personnage` de type `Paysan` dont le nom est « peon » ayant 25 points de vie, 5 points de dégâts d'attaque, 30 points de parade, 0% d'attaque, 40% de parade, 0 de distance maximum, et se trouvant en position [3,6]

Exemple de fichier de sauvegarde (3)

- Explications :
  - La ligne **« Loup 30 … »** représente une `Creature` de type `Loup` ayant 30 points de vie, 80 points de dégâts d'attaque, 50 points de parade, 50% d'attaque, 50% de parade, et se trouvant en position [19,3]
  - La ligne **« Lapin 4000 … »** représente une `Creature` de type `Lapin` ayant 4000 points de vie, 90 points de dégâts d'attaque, 1 points de parade, 1000% d'attaque, 1% de parade et se trouvant en position [10,10]
  - La ligne **« Loup 80 … »** représente une `Creature` de type `Loup` ayant 80 points de vie, 75 points de dégâts d'attaque, 30 points de parade, 45% d'attaque, 5% de parade, et se trouvant en position [2,13]
  - La ligne **« Loup 30 … »** représente une `Creature` de type `Loup` ayant 30 points de vie, 30 points de dégâts d'attaque, 30 points de parade, 20% d'attaque, 10% de parade, et se trouvant en position [5,14]

Exemple de fichier de sauvegarde (4)

- Explications :
  - La ligne « **Lapin 30 …** » représente une `Creature` de type `Lapin` ayant 30 points de vie, 20 points de dégâts d'attaque, 20 points de parade, 40% d'attaque, 10% de parade et se trouvant en position [23,23]
  - La lige « **NuageToxique …** » représente un `Objet` de type `NuageToxique` ayant 50% d'attaque, 5 points de dégâts d'attaque et se trouvant en position [20,5]
  - La ligne « **PotionSoin 25 …** » représente un `Objet` de type `PotionSoin` permettant de redonner 25 points de vie et se trouvant en position [18,18]
  - La ligne « **Epee 50 …** » représente un `Objet` de type `Epee` permettant d'augmenter de 50 les points de dégâts d'attaque du `Personnage` qui l'utilise et se trouvant en position [12,18]

Exemple de fichier de sauvegarde (5)

- Explications :
  - La ligne « **PotionSoin 100 …** » représente un `Objet` de type `PotionSoin` permettant de redonner 100 points de vie et se trouvant en position [4,20]
  - La ligne « **Poisson 20 …** » représente un `Objet` de type `Poisson` (dont la super classe directe serait `Nourriture`) permettant de redonner 20 points de vie et se trouvant en position [4,20]
  - La ligne « **PotionSoin 48 …** » représente un `Objet` de type `PotionSoin` permettant de redonner 48 points de vie et se trouvant en position [1,19]
  - La ligne « **Miel 5 …** » représente un `Objet` de type `Miel` (dont la super classe directe serait `Nourriture`) permettant de redonner 5 points de vie et se trouvant en position [7,9]

Exemple de fichier de sauvegarde (6)

- Explications :
  - 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]
  - Les lignes `Inventaire` listent le contenu de l'inventaire du Joueur humain. Le joueur possède donc dans son `Inventaire` :
  - La ligne **« Inventaire PotionSoin 75 … »** représente une `PotionSoin` permettant de redonner 75 points de vie au `Joueur`
  - La ligne **« Inventaire Epee 125 … »** représente une `Epee` permettant d'augmenter de 125 les points de dégâts d'attaque du `Joueur`
  - La ligne **« Inventaire Eau 10 … »** représente un `Objet` de type `Eau` (dont la super classe directe serait `Nourriture`) permettant de redonner 10 points de vie
  - La ligne **« Inventaire PotionSoin 15 … »** représente une `PotionSoin` permettant de redonner 15 points de vie au `Joueur`

2E PARTIE B.1 : CHARGEMENT D'UN FICHIER 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 ;
          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) {
          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 chaine de caractères en sous-chaines

- Nous avons maintenant une chaine de caractères (un objet de type `String`) contenant une ligne complète de notre fichier
- On souhaite maintenant **découper cette chaine 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 chaine 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 delimiteurs = " ,.;";
20
21      // on declare un 'tokenizer' qui va decouper
22      // 'test' en fonction de l'ensemble des delimiteurs
23      StringTokenizer tokenizer = new StringTokenizer(test, delimiteurs);
24
25      // Parcours de l'ensemble des unites lexicales de 'test'
26      // hasMoreTokens() retourne 'vrai' tant qu'il reste des 'mots'
27      // dans 'test' separes par un des delimiteurs (espace,
28      // virgule, etc.) declares plus haut
29      while(tokenizer.hasMoreTokens()) {
30        // nextToken() retourne la prochaine unite lexicale decoupee par les delimiteurs
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 méthode `chargementPartie` dans la classe `World` qui :
  - A pour but de **charger un fichier de sauvegarde** et de
  - **Rempli les attributs de la classe `World`** permettant de reprendre la simulation là où elle avait été enregistrée

Travail à faire (1)

- **Écrivez** la méthode `chargementPartie` de la classe `World` telle que :
  - Elle prend en entrée un attribut représentant le **nom du fichier de sauvegarde** à charger
  - Elle utlise utilisant un `BufferedReader`
  - Elle « charge » dans `World` les attributs 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
- 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 (ou pas) d'une méthode annexe `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 chaine de caractères représentant la ligne correspondante à un élément de cette classe

  - **Commentez** ces deux solutions et choisissez-en une en **justifiant votre choix** !

2E PARTIE B.2 : 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 méthode de la classe `World` 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)

```java
import java.io.*;

public class TestBufferedWriter {

  public static void main(String args[]) {
    BufferedWriter bufferedWriter = null;
    String filename = "monFichier.txt";

    try {
      // Creation du BufferedWriter
      bufferedWriter = new BufferedWriter(new FileWriter(filename));
      // On ecrit dans le fichier
      bufferedWriter.write("Ecriture ligne un dans le fichier");
      bufferedWriter.newLine();
      bufferedWriter.write("Ecriture ligne deux dans le fichier");
    }
    // on attrape l'exception si on a pas pu creer le fichier
    catch (FileNotFoundException ex) {
      ex.printStackTrace();
    }
    // on attrape l'exception si il y a un probleme lors de l'ecriture dans le fichier
    catch (IOException ex) {
      ex.printStackTrace();
    }
    // on ferme le fichier
    finally {
      try {
        if (bufferedWriter != null) {
          // je force l'écriture dans le fichier
          bufferedWriter.flush();
          // puis je le ferme
          bufferedWriter.close();
        }
      }
      // on attrape l'exception potentielle
      catch (IOException ex) {
        ex.printStackTrace();
      }
    }
  }
}
```

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 chaine 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 chaine 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 chaine de caractères **avec** et **sans caractère de retour à la ligne** intégré à la chaine 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 méthode `sauvegardePartie` dans la classe `World` 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 méthode `sauvegardePartie` de la classe `World` telle que :
  - Elle prend en entrée un attribut représentant le **nom du fichier de sauvegarde** à charger
  - Elle utilise un objet de type `BufferedWriter`
  - Elle enregistre sous forme de fichier le contenu des attributs de la classe `World` contenant l'ensemble des éléments du jeu
- **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 (ou pas) d'une méthode annexe `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

3E 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 (Fichier ou Base de données)
- **Terminez** la partie
- **Relancez** le jeu et **chargez** la partie sauvegardée précédemment (via un fichier ou une base de données)
- **Illustrez** bien le fait que les deux parties sont identiques

4E 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 !


Success
NEXT EXIT