

Programmation Orientée Objet (OBJET)

TP 5 : Amélioration de « World of ECN »

Ajout de nouvelles classes et modifications des
classes existantes –
Classes abstraites et Interfaces



Jean-Marie Normand – Bureau E211

jean-marie.normand@ec-nantes.fr

Instructions

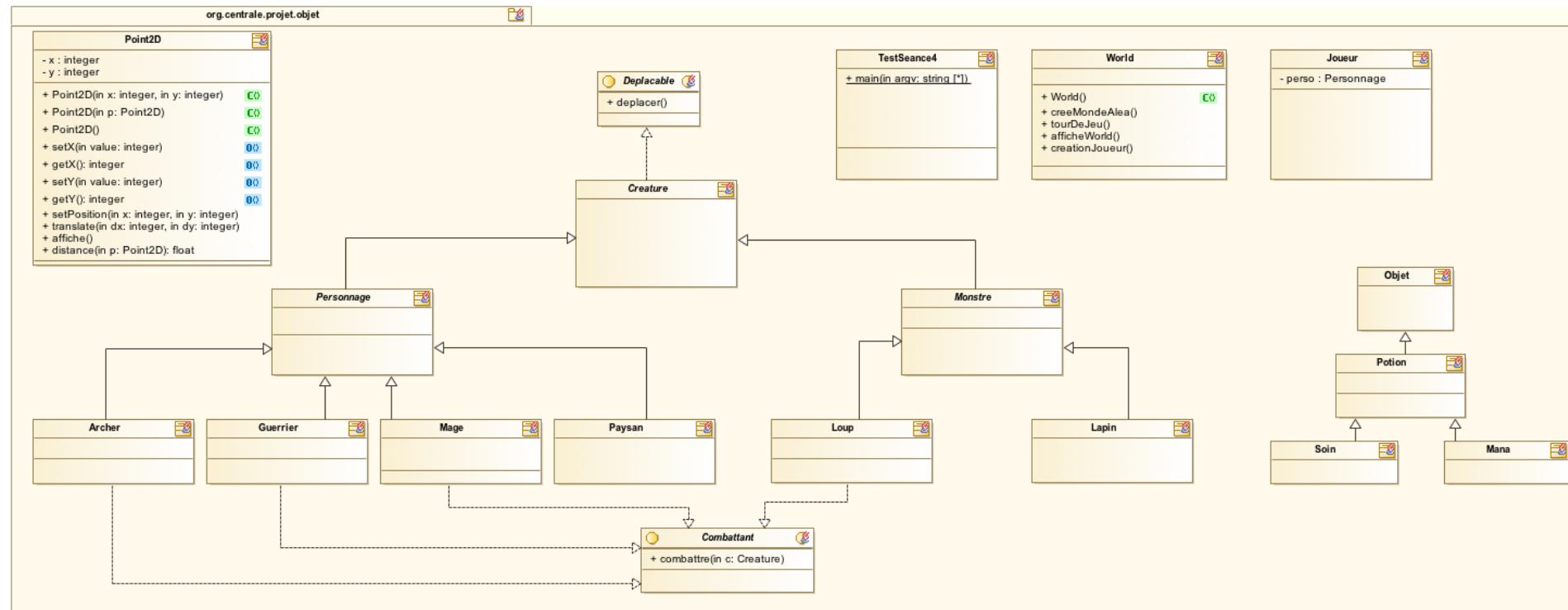
- Suivez les slides les uns après les autres
- A la fin de chaque séance de TP, vous devrez nous rendre un rapport par binôme
- Ce rapport devra contenir :
 - Une introduction et une présentation rapide du sujet de la séance
 - Les réponses aux questions posées dans les slides repérés par une icône de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



1^{RE} PARTIE : COMPRÉHENSION D'UN DIAGRAMME DE CLASSES UML

MàJ du diagramme de classe UML

- En vous basant sur le diagramme UML suivant (disponible en version plus lisible dans le fichier [UML-Seance4.png](#)) :



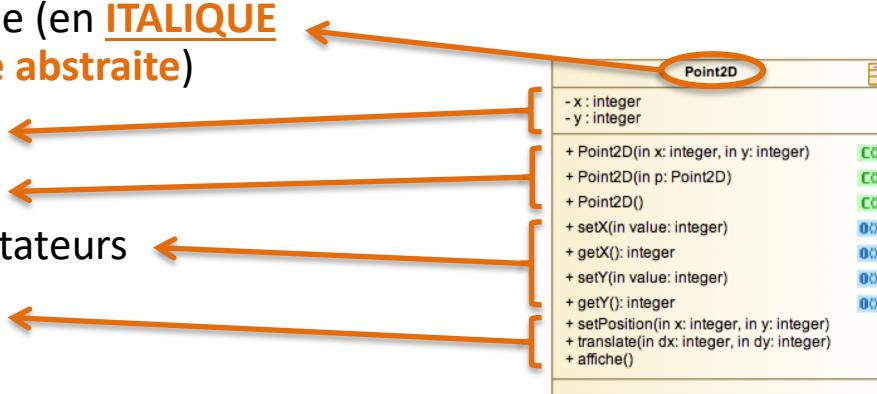
Notations UML

- Rappels UML :

 - Package

 - Classes :

 - Nom de la classe (en **ITALIQUE** pour une **classe abstraite**)



 - Attributs

 - Constructeurs

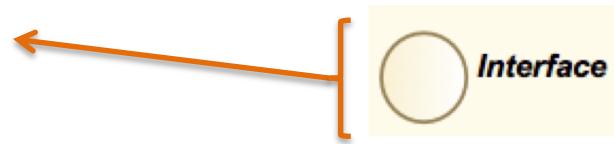
 - Accesseurs/Mutateurs

 - Méthodes

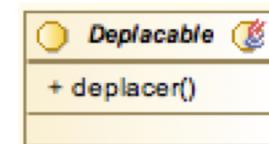
 - Lien d'héritage

 - Interface (dans le logiciel Modelio l'affichage diffère si l'interface possède une méthode ou non)

 - Implémentation d'une interface

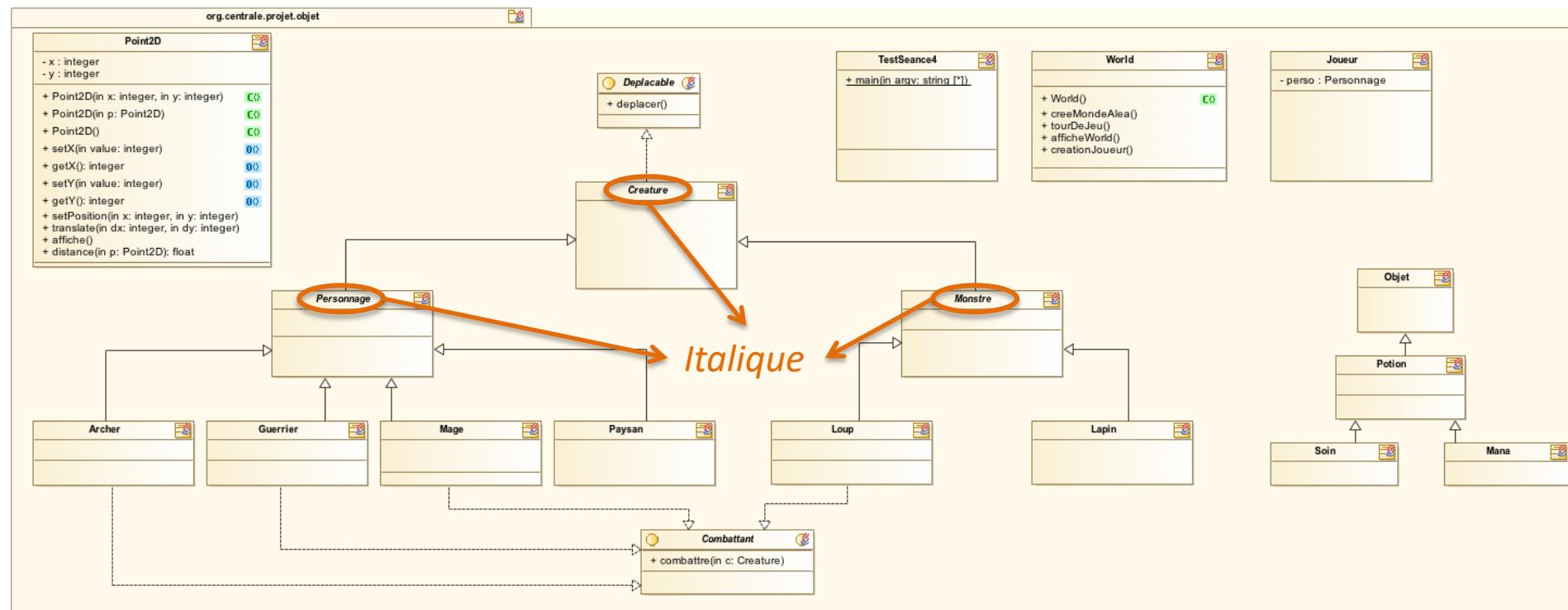


ou



MàJ du diagramme de classe UML

- Précisions :



World of ECN – WoE

- Le diagramme de classes représente une **évolution de WoE** (néanmoins toujours incomplet pour l'instant)
- Nous avons **ajouté** de nouvelles classes
- Nous avons **modifié** les classes existantes
- Nous avons **ajouté deux interfaces**
- Les éléments UML sont maintenant **volontairement « vides » → à vous d'y mettre ce que vous y jugez bon !!!**



Travail à faire

- A partir du diagramme UML :
 - **Commentez en argumentant** les modifications proposées :
 - **Expliquez** pourquoi certaines classes sont devenues abstraites
 - Quel intérêt voyez-vous à l'interface **Deplacable** ?
 - NB : vous pouvez modifier les paramètres à passer à la méthode **deplacer** si besoin est
 - Quel intérêt voyez-vous à l'interface **Combattant** ?
 - **Rappelez** la manière dont vous proposez de stocker plusieurs créatures et objets (dont le nombre est inconnu à l'avance) dans la classe World et **justifiez votre choix**



Travail à faire

- À partir du diagramme UML :
 - Nous avons rajouté une classe **Joueur** :
 - Cette classe permet de gérer **un joueur humain** lequel pourra choisir son **Personnage** parmi les classes existantes
 - **Implémentez** une méthode permettant à l'utilisateur de choisir son type de **Personnage** :
 - **Demandez en mode textuel** à l'utilisateur de rentrer une chaîne de caractères correspondant à une classe (p. ex. « Guerrier » etc.)
 - Demandez-lui **le nom** de son **Personnage**
 - **Générez aléatoirement** le reste des attributs de ce **Personnage** (les valeurs aléatoires doivent dépendre de la classe choisie, p. ex. un **Guerrier** aura plus de **degAtt** qu'un **Mage** etc.)



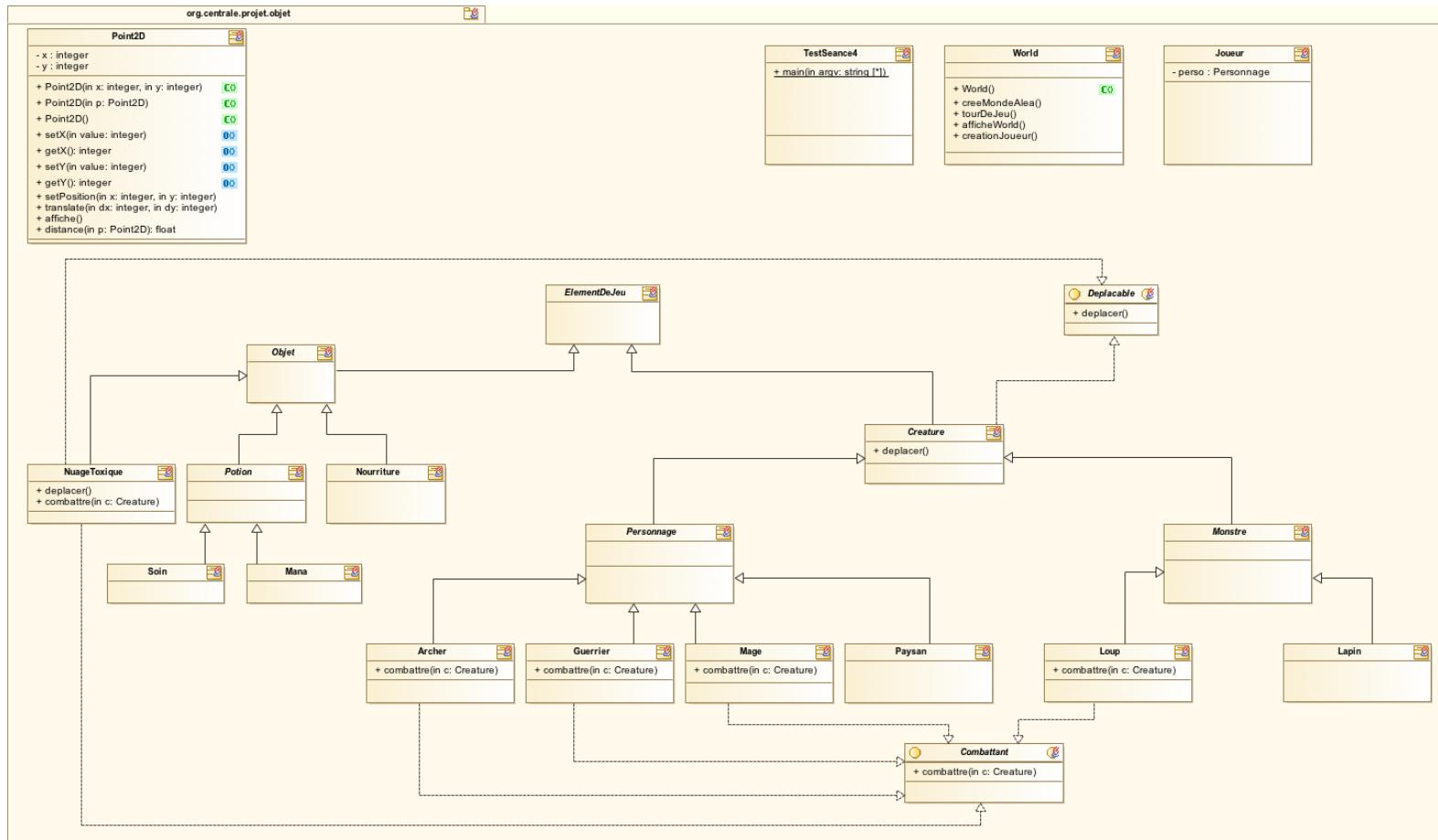
Travail à faire

- Le **joueur humain** :
 - **doit pouvoir contrôler son déplacement** (contrairement aux autres objets de type **Creature**) → sans pour autant se déplacer sur des cases occupées
 - **doit pouvoir choisir de combattre** si il le souhaite
- À chaque tour de jeu il convient donc de demander au joueur humain ce qu'il veut faire
- **Implémentez** une fonction permettant à un joueur humain de choisir à chaque tour de jeu si il préfère se déplacer ou combattre
- Dans votre rapport, illustrez :
 - le **bon fonctionnement de la création d'un joueur humain**
 - la **création aléatoire** d'un certain nombre (> 50) de **Creature** contrôlées automatiquement par vous
 - la création aléatoire de quelques **Potion**
 - quelques tours de jeu pendant lesquels le joueur humain est amené **à se déplacer et à combattre**
 - lui permettant de **ramasser quelques potions**

2^E PARTIE : AJOUT DE NOUVELLES CLASSES ET INTERFACES

Nouvelle mise à jour du diagramme de classe UML !

- Implémentez le diagramme UML suivant (disponible en version plus lisible dans le fichier [UML-Seance5.png](#)) :





Nouveautés

- Avant de répondre aux questions qui suivent veillez à bien faire attention à la remarque concernant vos réponses présente dans le slide intitulé « Nouveautés (2) » !!
- Expliquez en justifiant les modifications apportées au diagramme de classe précédent :
 - Classe **ElementDeJeu** :
 - Pourquoi a-t-on ajouté cette nouvelle super-classe ?
 - Quel peut être son rôle ?
 - Pourquoi n'implémente-t-elle pas **Deplacable** ?
 - Pourquoi n'implémente-t-elle pas **Combattant** ?
 - Classe **NuageToxique** :
 - Expliquez son positionnement dans la hiérarchie des classes
 - Pourquoi implémente-t-elle des interfaces ?
 - Proposez une implémentation de cette classe :
 - Détaillez le comportement des méthodes **deplacer** et **combattre** que vous souhaitez mettre en place pour cette classe



Nourriture

- **Expliquez en justifiant** les modifications apportées au diagramme de classe précédent :
 - Classe **Nourriture** :
 - Cette classe doit permettre de représenter différents **bonus/malus** que les objets de type **Personnage** sont susceptibles de pouvoir ramasser sur le plateau de jeu
 - Ces bonus/malus doivent pouvoir modifier les caractéristiques d'un **Personnage** (pas les points de vie ni les points de mana) de **manière temporaire** (pour un certain nombre de tours de jeu) !
 - Proposez une modification de notre diagramme de classes illustrant la création d'un moins :
 - 1 **Bonus** laissé à votre choix
 - 1 **Malus** laissé à votre choix
 - Chaque bonus ou malus doit impacter seulement une caractéristique !
 - Veillez à donner à vos classes des noms explicites !
 - **Précisez les modifications** que vous apporteriez à la classe **Nourriture** :
 - » Méthodes
 - » Attributs
 - » Etc.



Nourriture (2)

- **Expliquez en justifiant** les modifications apportées au diagramme de classe précédent :
 - Classe **Nourriture** (suite) :
 - Pour implémenter ce mécanisme, nous proposons de rajouter un attribut de type **List<Nourriture>** à tous les **Personnages**
 - Cette liste est initialement vide et se remplit au fur et à mesure que le **Personnage** ramasse des bonus/malus dans le monde
 - Les caractéristiques du **Personnage** sont donc **modifiées pendant toute la durée d'activation** du bonus/malus → en particulier **lors des calculs effectués pour les combats**



Nourriture (3)

- Expliquez en justifiant les modifications apportées au diagramme de classe précédent :
 - Classe **Nourriture** (suite) :
 - À la fin de chaque tour de jeu, chaque Personnage doit donc parcourir sa liste de Nourriture, pour mettre à jour la durée des bonus/malus en cours (diminuer la durée de 1)
 - À chaque calcul impliquant les caractéristiques vous devez donc parcourir la liste des ces bonus/malus afin de calculer le bonus ou malus à appliquer aux caractéristiques du Personnage
 - Lorsqu'un bonus/malus arrive à une durée de 0 vous pouvez le retirer de la liste :
 - Attention à la manière dont on retire des éléments d'une liste !



Nouveautés (2)

- Ces nouveautés **remettent-elles** en cause certains choix d'implémentation de votre jeu ?
 - En particulier pour la classe **World** ?
 - **Justifiez** vos réponses !
 - La **qualité** et la **précision** de vos réponses aux questions de ces deux slides est fortement prise en compte dans la note
 - **Proposez une illustration** des nouveautés :
 - Un **Joueur** humain se déplaçant dans le monde pouvant récolter de la **Nourriture** et/ou des **Potions**, et se battre contre des **Creature** contrôlées automatiquement

WoE to be continued...

- Laissons de côté WoE pour un moment
- Nous y reviendrons un peu plus tard pour y ajouter la possibilité de sauvegarder et de charger notre monde via des fichiers texte
- En attendant intéressons-nous un peu aux exceptions !

Génération de quelques **Exceptions**

- On se propose d'utiliser le projet WoE pour illustrer le mécanisme des **Exceptions** en Java
- Dans la suite nous **présenterons brièvement** quelques unes des exceptions les plus communes
- Votre **objectif sera de tenter d'écrire du code produisant ces exceptions**
- Enfin vous utiliserez les blocs **try/catch/finally**

NullPointerException

- L'application Java essaye d'accéder à un objet dont la référence est en fait à `null`
- Cf.
<http://docs.oracle.com/javase/7/docs/api/java/lang/NullPointerException.html>

ArrayIndexOutOfBoundsException

- On a tenté d'accéder à un tableau dans un indice illégal
- Cf.
<http://docs.oracle.com/javase/7/docs/api/java/lang/ArrayIndexOutOfBoundsException.html>

ArithmeticException

- Une opération arithmétique illégale a été calculée
- Cf.
<http://docs.oracle.com/javase/7/docs/api/java/lang/ArithmeticException.html>

ClassCastException

- Une opération de transtypage illégale a été effectuée
- Cf.
<http://docs.oracle.com/javase/7/docs/api/java/lang/ClassCastException.html>

NumberFormatException

- La conversion d'une chaîne de caractères vers un type numérique a échoué car le format n'est pas le bon
- Cf.
<http://docs.oracle.com/javase/7/docs/api/java/lang/NumberFormatException.html>
- Attention à ne pas confondre avec **InputMismatchException** qui peut être levée par la classe **Scanner**

StackOverflowError

- La pile d'appel Java a explosé !
- Cf.

[http://docs.oracle.com/javase/7/docs/api/java/lang/
StackOverflowError.html](http://docs.oracle.com/javase/7/docs/api/java/lang/StackOverflowError.html)

ConcurrentModificationException

- Avec un peu de (mal)chance vous l'avez déjà rencontrée lors de l'écriture de la suppression de la **Nourriture** dans la liste de bonus/malus !
- Cette exception peut survenir lorsque l'on essaye de supprimer un élément d'une **Collection** Java alors qu'on la parcourt
- Cf.
<http://docs.oracle.com/javase/7/docs/api/java/util/ConcurrentModificationException.html>



Exceptions

- Pour chaque exception présentée ci-avant :
 - **Proposez** quelques lignes de code générant cette exception
 - Le cas échéant **expliquez** pourquoi cette exception survient
 - Veillez à clairement **présenter dans votre rapport** les lignes de code et le résultat de l'exécution de l'application avec l'exception affichée



Exceptions

- Pour le cas particulier de **NumberFormatException** :
 - **Illustrez** un problème pouvant arriver lors d'une saisie d'information au clavier par un utilisateur (par exemple lors de la création de son Personnage)
 - **Proposez** une méthode contenant un bloc **try/catch** (et éventuellement **finally** si vous le souhaitez) permettant de gérer cette exception
 - **Proposez** une alternative au bloc **try/catch** en écrivant une méthode qui va faire remonter l'exception vers sa méthode appelante (cf. mot clé **throw**)



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-TP5-NomBinome1-NomBinome.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 finale !

