Programmation Orientée Objet (OBJET)

TP 5 : Amélioration de « World of ECN »

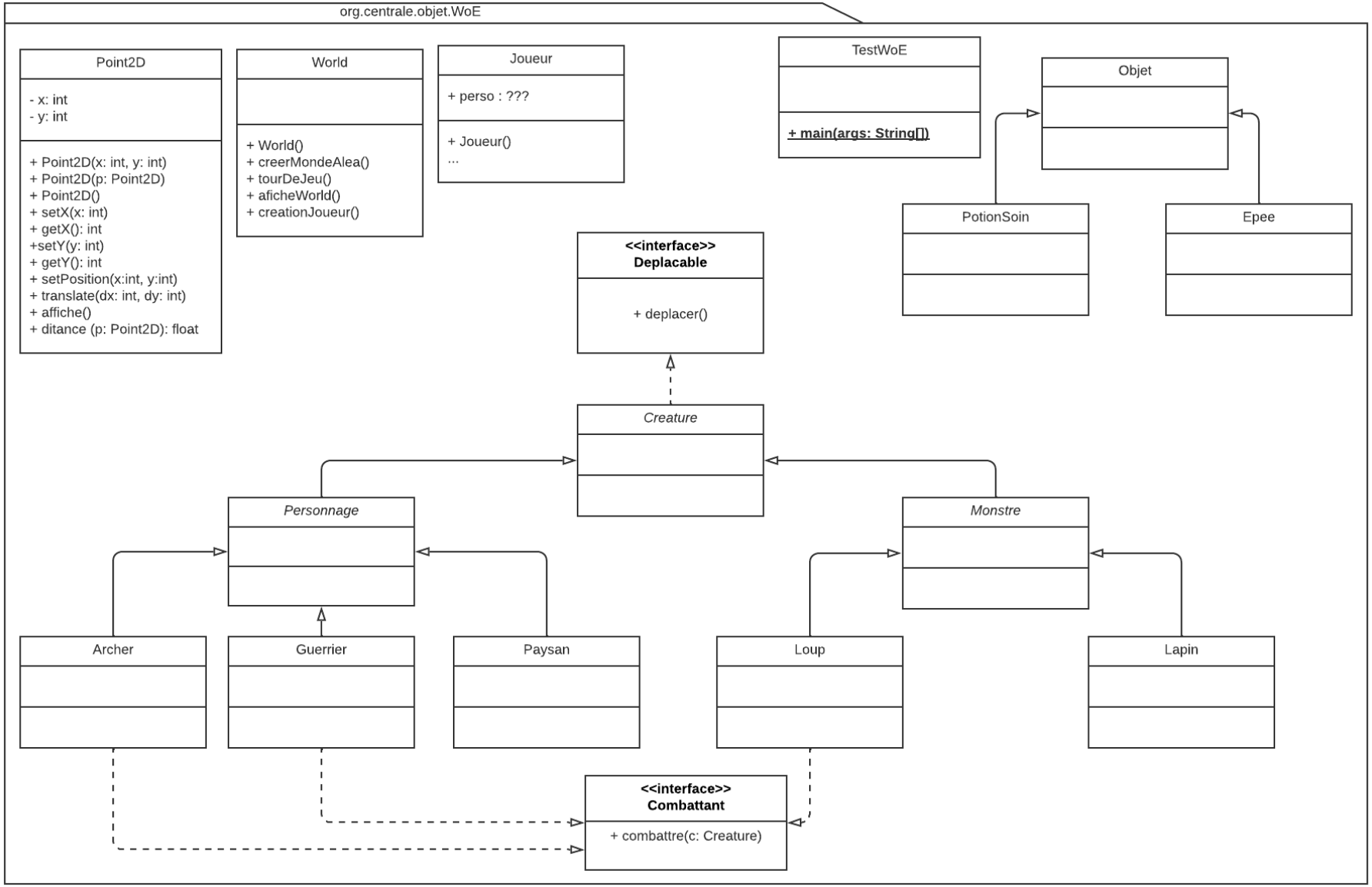
Ajout de nouvelles classes et modifications des classes existantes – Classes abstraites et Interfaces

Jean-Marie Normand – Bureau E211
jean-marie.normand@ec-nantes.fr

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

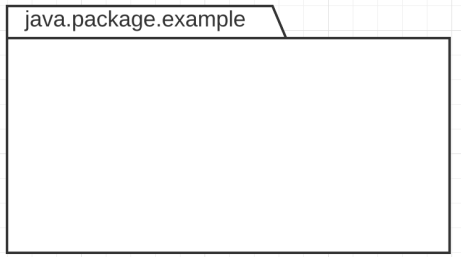
MàJ du diagramme de classe UML

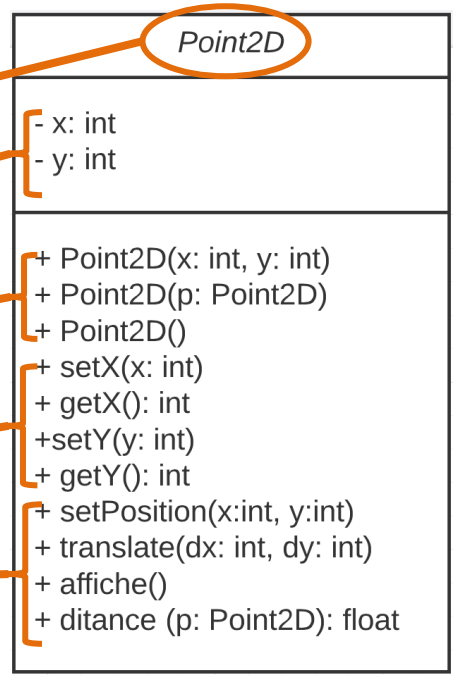
- En vous basant sur le diagramme UML suivant (disponible en version plus lisible dans le fichier `WoE-TP5.png`)

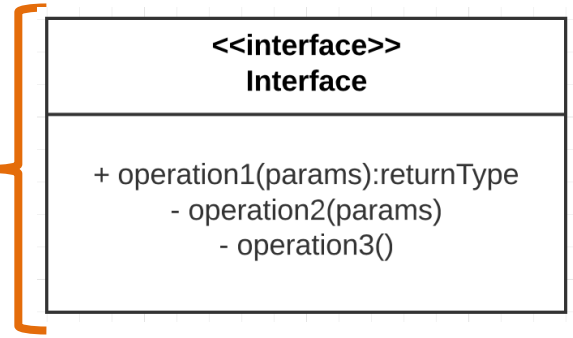


Notations UML

- Rappels UML :
  - Package
  - Classes :
    - Nom de la classe (en **ITALIQUE** pour une **classe abstraite**)
    - Attributs
    - Constructeurs
    - Accesseurs/Mutateurs
    - Méthodes
  - Lien d'héritage
  - Interface
  - Implémentation d'une interface

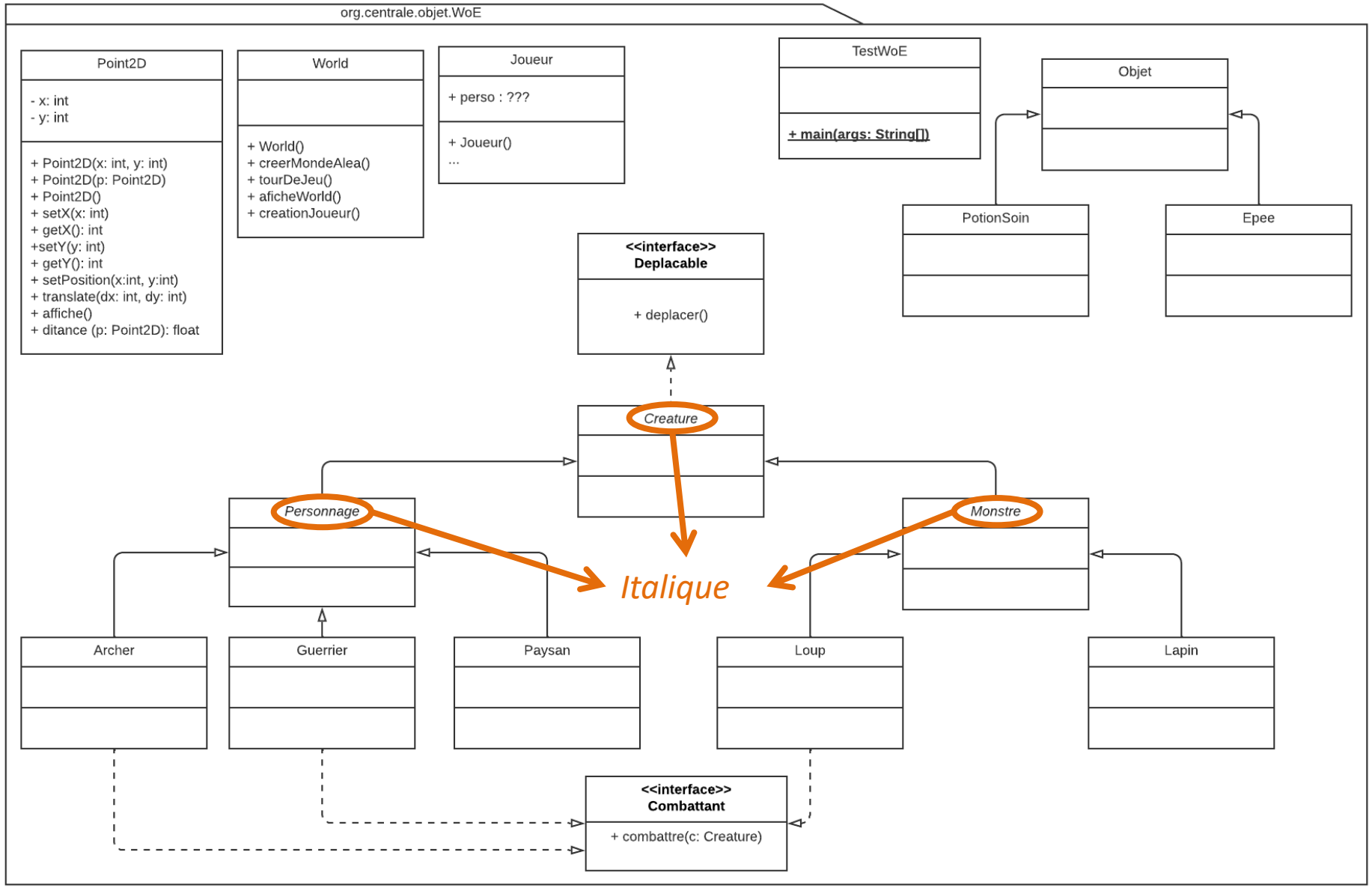






MàJ du diagramme de classe UML

- 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

- À partir du diagramme UML :
  - **À propos des modifications proposées** :
    - **Réfléchissez à** 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
    - Quel intérêt voyez-vous à l'interface `Combattant` ?
    - **Pensez-vous que cela doive impacter** la manière dont vous proposez de stocker plusieurs créatures et objets (dont le nombre est inconnu à l'avance) dans la classe World

Travail à faire

- Nous avons rajouté une classe `Joueur` :
  - Cette classe permet de gérer **un joueur humain** lequel pourra choisir son `Personnage` parmi les classes existantes « jouables » (toute classe n'est pas jouable, p. ex. on ne peut pas jouer un `Paysan`)
  - **Modifiez** le diagramme UML et **implémentez** la solution retenue pour respecter la contrainte de jouabilité limitée à certaines classes de personnages
  - **Implémentez** une méthode permettant à l'utilisateur de choisir son type de `Personnage` jouable :
    - **Demandez en mode textuel** à l'utilisateur de rentrer une chaine de caractères correspondant à une classe jouable (p. ex. « Guerrier » etc.)
    - Demandez-lui **le nom** de son `Personnage` jouable
    - **Générez aléatoirement** le reste des attributs de ce `Personnage` jouable (les valeurs aléatoires doivent dépendre de la classe choisie, p. ex. un `Guerrier` aura plus de degAtt qu'un `Archer` etc.)

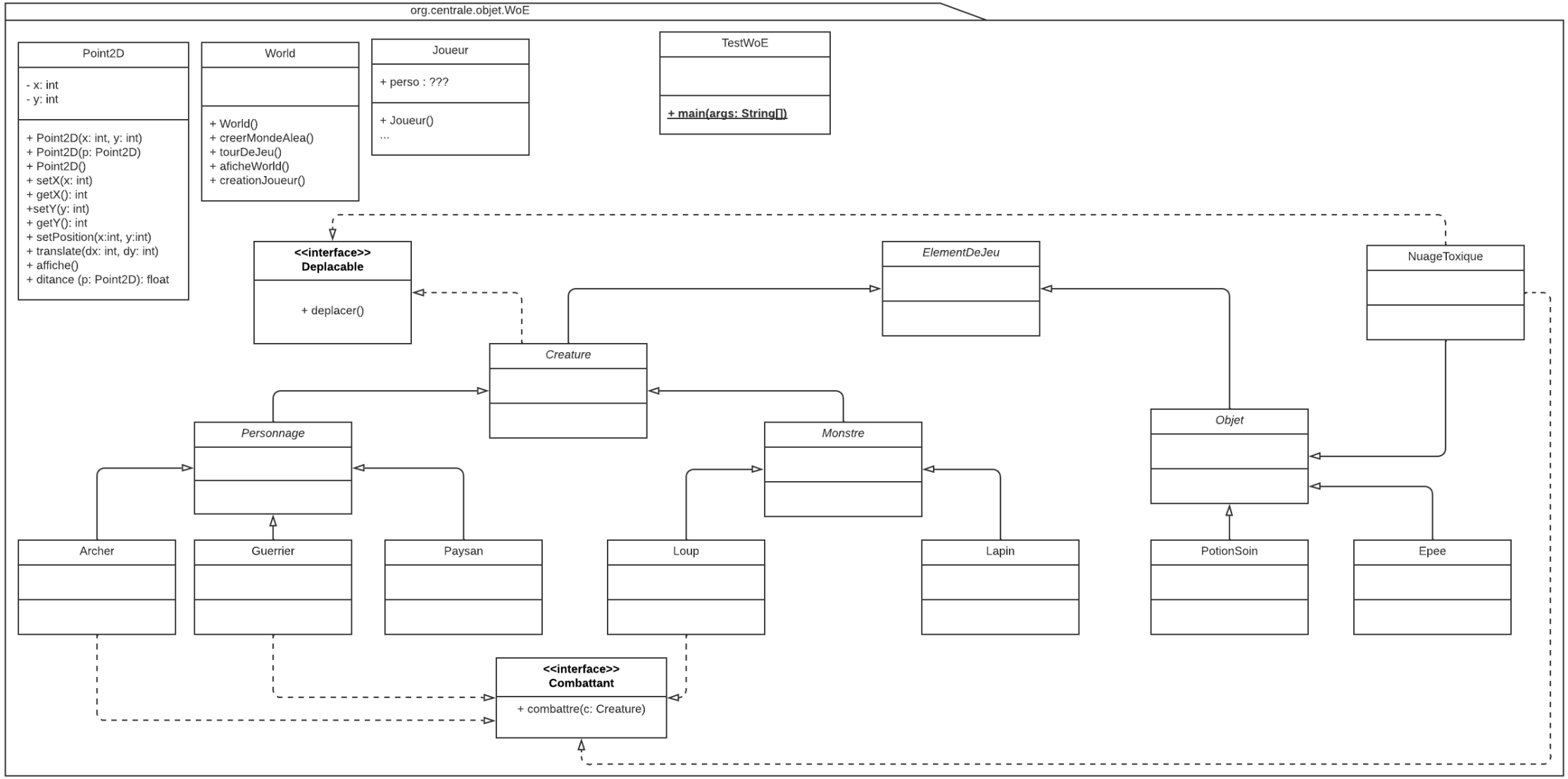
Travail à faire

- 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 s'il préfère se déplacer ou combattre
- **Assurez-vous** :
  - Du **bon fonctionnement de la création d'un joueur humain**
  - De la **création aléatoire** d'un certain nombre (> 50) de `Creature` contrôlées automatiquement par le jeu
  - De la création aléatoire de quelques objets de type `PotionSoin` et `Epée`
- **Effectuez** quelques tours de jeu :
  - **Déplacez** le joueur humain dans le monde
  - Amenez le **à combattre**
  - Permettez-lui de **ramasser quelques potions ou épées**

2E PARTIE : AJOUT DE NOUVELLES CLASSES ET INTERFACES

Nouvelle mise à jour du diagramme de classe UML !

- Répercutez les nouveautés du diagramme UML suivant (disponible en version plus lisible dans le fichier `WoE-TP6.png`) dans votre base de code



Nouveautés

- **D'après vous, pourquoi a t'on apporté** les modifications suivantes au diagramme de classe :

  - 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

Nourriture

- 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` (sauf les points de vie) de manière temporaire (pour un certain nombre de tours de jeu) !
  - Implémentez au moins (attention au noms de classes) :
    - 1 **Bonus** laissé à votre choix
    - 1 **Malus** laissé à votre choix
    - Chaque bonus/malus doit impacter seulement une caractéristique (p. ex. « feuille d'épinard » augmente la caractéristique « degAtt » de 2 points pour 3 tours de jeu)
    - **Effectuez les modifications** que vous jugez nécessaires à la classe `Nourriture` (méthodes, attributs, etc.)

Objets Utilisables

- La nourriture, comme les potions ou les épées sont des « objets utilisables » que le joueur utilise immédiatement lorsqu'il se déplace sur la case de l'objet

- Afin de savoir quels objets sont « utilisables » lors d'un tour de jeu, nous proposons l'écriture d'une **Interface** «`Utilisable` » qui indique quelles sont les classes qui peuvent être utilisées par le joueur (p. ex. `Nourriture`)

Objets Utilisables

- Pour connaître l'ensemble des objets utilisés à un instant `t`, le joueur dispose d'une `Collection<Utilisable>` appelée « `effets` »
  - Cette collection est initialement vide et se remplit au fur et à mesure que le `Personnage` « active » un objet `Utilisable`
  - Les caractéristiques du `Personnage` doivent donc être **modifiées pendant toute la durée d'activation** des `Utilisable` de la `Collection` → en particulier **lors des calculs effectués pour les combats**
  - La durée de chaque `Utilisable` de la `Collection<Utilisable>` est décrémentée de 1 à chaque tour de jeu
  - Les `Utilisable` dont la durée atteint 0 doivent être retirés de la `Collection<Utilisable>` (attention à comment on supprime 1 objet d'une collection)

Inventaire

- Chaque joueur peut avoir un **inventaire** qui lui permet de stocker l'ensemble des objets utilisables qu'il ramasse lors de son parcours dans WoE
- Lorsqu'un joueur se déplace sur une case contenant un objet Utilisable, au lieu d'utiliser l'objet, celui-ci va maintenant aller dans son `Inventaire` et n'est pas activé automatiquement

- Lors d'un tour de jeu, le joueur doit pouvoir choisir :
  - De se déplacer
  - De combattre un ennemi à sa portée
  - D'utiliser un des objets de son `Inventaire`, qui disparaît de son inventaire pour passer dans la collection des `effets` actifs

Inventaire (2)

- Pour utiliser un objet de son inventaire, le joueur doit d'abord choisir quel objet utiliser
- Pour ce faire, vous devez donc :
  - Afficher le contenu de l'`Inventaire`
  - Chaque ligne doit posséder **un numéro** et correspondre à **un objet** de l'`Inventaire`
  - Le joueur doit indiquer le numéro correspondant à l'objet qu'il souhaite utiliser
  - Cet objet est utilisé et disparaît de l'`Inventaire`

Version Fonctionnelle

- Nous avons fait des modifications significatives sur le diagramme de classes

- **Mettez à jour votre diagramme de classe (utilisez p. ex. le plugin easyUML de NetBeans)**

- **Soyez en mesure de proposer une illustration** des nouveautés :
  - Un `Joueur` humain se déplaçant dans le monde pouvant :
    - De déplacer
    - Récolter puis activer de la `Nourriture` et/ou des `Potions`
    - 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
- 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/8/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/8/docs/api/java/lang/ArrayIndexOutOfBoundsException.html

`ArithmeticException`

- Une opération arithmétique illégale a été calculée
- Cf. http://docs.oracle.com/javase/8/docs/api/java/lang/ArithmeticException.html

ClassCastException

- Une opération de transtypage illégale a été effectuée
- Cf. http://docs.oracle.com/javase/8/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/8/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/8/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/8/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`)


Success
NEXT EXIT