

Programmation Orientée Objet (OBJET)

TP 4 : Projet « World of ECN »

Gestion d'une multitude de protagonistes
dans notre jeu



Jean-Marie Normand – Bureau IM3-B
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 : PRÉSENTATION DU TRAVAIL À FAIRE

But du TP

- Aujourd'hui nous voulons pouvoir gérer de nombreux protagonistes dans notre jeu
- C'est la classe **World** qui contient pour le moment tous nos protagonistes → on va donc modifier cette classe !
- Pour le moment :
 - 1 protagoniste (Archer, Paysan, Lapin, etc.) = 1 attribut de la classe **World**
 - Comment gérer 100 protagonistes ? 1000 protagonistes ? 10000 ? etc.
 - Il est irréaliste de devoir gérer à la main 100, 1000 voire 1000 attributs !

But du TP (2)

- En cours nous avons vu comment Java permet de gérer des **ensembles d'objets** grâce à la **généricité** et aux **Collections** de l'API Java
- Les **Collections** sont des **conteneurs génériques** du package **java.util**
- Les principales **Collections** :
 - **ArrayList** : tableaux (vecteurs)
 - **LinkedList** : listes doublement chainées
 - **TreeSet** : ensemble (ordonné)
 - **HashMap** : table de hachage

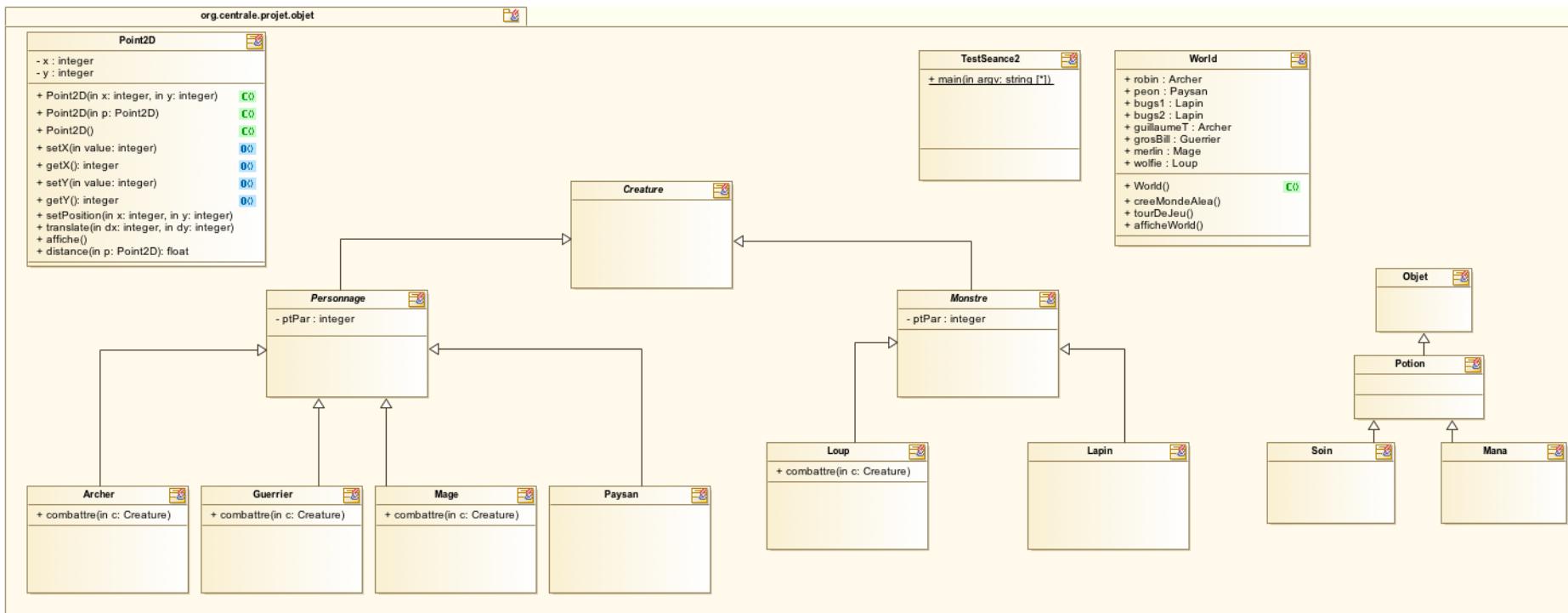
But du TP (3)

- L'objectif est de vous faire **manipuler les conteneurs génériques Java**
- De **comparer les performances** de différents conteneurs
- Étudier les **différentes manières de manipuler** ces conteneurs et les **impacts potentiels** de vos choix d'implémentation en termes de performances

2^E PARTIE : MISE À JOUR DE WOE !

Rappel du diagramme de classe UML

- Le projet WoE est (à peu près) représenté par le diagramme UML suivant (fichier [UML-Seance2.png](#))
- Vous avez potentiellement fait des modifications !!!





Gestion de multiples protagonistes

- Nous souhaitons pouvoir gérer un grand nombre (ce nombre n'étant pas connu à l'avance) de protagonistes dans notre jeu WoE
- La classe **World** contenant pour le moment l'ensemble de nos protagonistes nous allons la modifier !
- **Quelle(s) solution(s)** proposez vous pour pouvoir avoir un nombre non connu à l'avance de :
 - **Personnages** de type :
 - Archer
 - Paysan
 - Guerrier
 - Mage
 - **Monstres** de type :
 - Loup
 - Lapin
 - **Objets** de type :
 - Potion de soins
 - Potion de mana



Gestion de multiples protagonistes

- **Précisez en le justifiant** quelle solution (issues de celles listées précédemment) vous choisissez de mettre en œuvre dans votre projet
- **Listez** les modifications à apporter à la classe **World** pour mettre en œuvre votre solution
- **Implémentez** votre solution et **justifiez** le choix du conteneur Java
- **Illustrez** le bon fonctionnement de votre solution en créant un nombre tiré aléatoirement (nombre différent pour chaque « type » de **Personnage**) de :
 - Archer
 - Paysan
 - Lapin
 - Guerrier
 - Loup

Javadoc

- Rappel : on suppose dans la suite de ce projet que vous devez écrire la Javadoc pour :
 - Toutes les classes
 - Tous les attributs
 - Les méthodes principales de chaque classe
- Nous ne rappellerons plus cette attente qui devra devenir automatique lors des prochaines séances du projet



Parcours des conteneurs Java

- Nous souhaitons maintenant pouvoir parcourir tous les éléments contenus dans un conteneur Java afin de les afficher
- **Écrivez une boucle de parcours de votre/vos conteneur(s) basée sur le nombre d'éléments stockés (i.e. la taille du conteneur)**
- **Affichez** l'élément courant à chaque tour de boucle
- **Illustrez** le bon fonctionnement de votre boucle de parcours

Mesure de temps en Java

- Il existe deux moyens de mesurer le temps en Java :
 - `System.currentTimeMillis()` voir la documentation
[http://docs.oracle.com/javase/7/docs/api/java/lang/System.html#currentTimeMillis\(\)](http://docs.oracle.com/javase/7/docs/api/java/lang/System.html#currentTimeMillis())
 - `System.nanoTime()` voir la documentation
[http://docs.oracle.com/javase/7/docs/api/java/lang/System.html#nanoTime\(\)](http://docs.oracle.com/javase/7/docs/api/java/lang/System.html#nanoTime())
- Ces méthodes retournent toutes les deux une valeur de type primitif `long` (entier long)
- `System.currentTimeMillis()` retourne le temps écoulé depuis le 1^{er} janvier 1970 en millisecondes → lié à l'horloge su système
- `System.nanoTime()` retourne le temps mesuré par la JVM → n'est pas lié à une quelconque notion d'horloge
- `System.nanoTime()` est à préférer lorsque l'on veut calculer le temps écoulé
- En théorie `System.currentTimeMillis()` peut être utilisée mais on le déconseille pour avoir des résultats précis

Mesure du temps en Java (2)

- Exemple avec `System.currentTimeMillis()`:

```
// Exemple utilisation System.currentTimeMillis()
long debut = System.currentTimeMillis();
int pipo=1;
for(int cpt=0;cpt<1000000;cpt++){
    pipo = pipo + pipo*2;
}
long fin = System.currentTimeMillis();
System.out.println("Temps ecoule en ms : "+(fin-debut));
```

Mesure du temps en Java (3)

- Exemple avec `System.nanoTime()`:

```
// Exemple d'utilisation de System.nanoTime()
long debutN = System.nanoTime();
int pipon=1;
for(int cpt=0;cpt<1000000;cpt++){
    pipon = pipon + pipon*2;
}
long finN = System.nanoTime();
System.out.println("Temps ecoule en ns : "+(finN-debutN));
```



On ne se marche pas sur les pieds (ni sur les pattes) !

- **Implémentez** l'interdiction à vos multiples protagonistes de se trouver sur une case déjà occupée
- **Illustrez** le bon fonctionnement de cette fonctionnalité en positionnant 100 protagonistes de votre choix dans un monde dont la taille serait 50x50 cases
- **Ajoutez une constante** à la classe **World** permettant de représenter la taille du monde selon une dimension

Comparaisons des différents conteneurs Java

- Nous souhaitons **comparer les différents conteneurs Java** en termes **d'efficacité** et ce **en fonction du code écrit** par le programmeur
- Pour ce faire nous allons :
 - **Stocker** les protagonistes dans différents types de conteneur
 - **Effectuer un parcours basé sur la taille** (cf. cours) pour accéder aux protagonistes
 - **Effectuer un parcours basé sur les itérateurs Java** (cf. cours) pour accéder aux protagonistes
 - **Mesurer le temps** dans chacun des cas
 - **Tracer des graphes** montrant l'évolution du temps de calcul en fonction du nombre de protagonistes stockés dans les conteneurs



Comparaisons des différents conteneurs Java (2)

- Utilisez un conteneur de type `LinkedList` pour stocker vos protagonistes
- Ajoutez 100 protagonistes aléatoirement dans votre conteneur (assurez vous que votre monde soit suffisamment grand)
- Mesurez le temps nécessaire pour calculer le barycentre de l'ensemble des protagonistes avec une boucle basée sur la taille du conteneur
- Mesurez le temps nécessaire pour calculer le barycentre de l'ensemble des protagonistes avec une boucle basée sur les itérateurs
- Recommencez l'opération avec :
 - 100 protagonistes
 - 1000 protagonistes
 - 10000 protagonistes
 - 100000 protagonistes
 - 1000000 (si le temps est trop long arrêtez l'opération après quelques minutes)
- Tracez un graphe illustrant l'évolution du temps nécessaire avec chacun des deux types de boucles en fonction du nombre d'éléments contenus



Comparaisons des différents conteneurs Java (3)

- **Utilisez** un conteneur de type `ArrayList` pour stocker vos protagonistes
- **Ajoutez** 100 protagonistes aléatoirement dans votre conteneur (assurez vous que votre monde soit suffisamment grand)
- **Mesurez le temps** nécessaire pour **calculer le barycentre de** l'ensemble des protagonistes avec **une boucle basée sur la taille du conteneur**
- **Mesurez le temps** nécessaire pour **calculer le barycentre de** l'ensemble des protagonistes avec **une boucle basée sur les itérateurs**
- **Recommencez** l'opération avec :
 - 100 protagonistes
 - 1000 protagonistes
 - 10000 protagonistes
 - 100000 protagonistes
 - 1000000 (si le temps est trop long arrêtez l'opération après quelques minutes)
- **Tracez un graphe** illustrant l'évolution du temps nécessaire avec chacun des deux types de boucles en fonction du nombre d'éléments contenus



Comparaisons des différents conteneurs Java (4)

- **Commentez** dans votre rapport les résultats des graphes établis précédemment. Vous paraissent-ils normaux ? **Justifiez**
- **Précisez** dans votre rapport **en le justifiant** si il a été compliqué de passer d'une **LinkedList** à une **ArrayList** (ou inversement)
- **Serait-ce également vrai** pour un autre type de conteneur (p. ex. **TreeSet** ou **Hashmap**) ?
- **Précisez dans votre rapport en le justifiant** si vous pensez que les deux types de conteneurs utilisés jusqu'à présent (i.e. **LinkedList** et **ArrayList**) pour stocker nos protagonistes sont les mieux adaptés à cette tâche ?
- Auriez-vous une alternative à proposer ? **Justifiez**



Retour à WoE

- Les études précédentes sur les conteneurs Java vous ont-elles donné matière **à modifier la solution proposée précédemment** (slide 10) ?
 - Si oui : pourquoi ? **Justifiez**
 - Si non : pourquoi ? **Justifiez**
- **Implémentez** une méthode dans la classe **World** permettant la création de plusieurs centaines de protagonistes tout en leur interdisant d'être à une distance inférieure à 3 unités
- **Illustrez** le bon fonctionnement de cette méthode dans votre rapport



Retour à WoE (2)

- Pour que nos protagonistes puissent évoluer dans le monde de manière cohérente il faut leur interdire de se déplacer sur une case déjà occupée (sauf par un objet « ramassable ») !
- Rappel : c'est la classe **World** qui est chargée de « **gérer** » le monde et c'est elle qui **connaît tous les protagonistes** de notre jeu
- **Proposez** un ensemble de solutions permettant de résoudre ce problème :
 - **Expliquez en le justifiant** quel choix vous ferez parmi toutes les solutions proposées (vous n'avez pas à l'implémenter aujourd'hui)



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-TP4-NomBinome1-NomBinome2.zip** (avec **NomBinome1 < nomBinome2** dans l'ordre alphabétique) et contenant :
 - Votre rapport au format **.pdf** dont le nom respectera la convention suivante :
OBJET-TP4-NomBinome1-NomBinome2.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 !



Centrale
Nantes

