Programmation Java
A. Bourmaud, J. Maitre, P. Rodriguez,
Ch. Sempé, J-F. Viaud

Introduction à la programmation et à la modélisation en Java

TP°6 et 7 – Guerriers - Création de nouvelles classes d'objets

Vous allez écrire un programme permettant de simuler les combats entre des clans de guerriers.
Vous devrez compléter une classe *Guerrier* permettant de définir les caractéristiques d'un guerrier ainsi que les opérations applicables à un guerrier.

Vous créerez également la classe *Clan*, un clan étant simplement un ensemble de guerriers.

A. La classe *Guerrier (à compléter – disponible sur Moodle)*

Un guerrier est défini par son nom, son age, sa force, son expérience et son état de santé.

Le comportement d'un guerrier est lié à un certain nombre de paramètres communs à tous les guerriers. Ces paramètres sont définis dans la classe *Guerrier* sous la forme de constantes statiques. C'est le cas de l'état de santé maximum (état de pleine forme, ici fixé à 100) l'état de santé minimum (celui qui détermine que le guerrier est mort, fixé à 0), le seuil de faiblesse ...

Complément : en Java, *private final static int forceMin = 1 ;* permet de définir la constante (*final*) entière *forceMin* commune à tous les guerriers (*static*) et inaccessible depuis une autre classe (*private*).

Les constructeurs et certaines méthodes devront utiliser ces constantes.

1. Attributs, constructeur et méthodes de base

Dans la classe *Guerrier*, déclarez les attributs d'un guerrier, puis écrivez :

- le constructeur qui permet de créer un guerrier dont la force sera un entier aléatoire ∈ [*forceMin*, *forceMax*], la santé ∈ [*limiteFaiblesse*, *santeMax*], l'âge ∈ [*ageMinDepart*, *ageMaxDepart*], l'expérience vaudra *experienceMin*. Le nom du guerrier sera passé en paramètre au constructeur
- les observateurs *getForce* et *getExperience*
- la méthode *vieillir*, qui fait vieillir le guerrier d'un an
- les méthodes booléennes *estVivant* et *estMort*
- la méthode *toString* (qui prépare le travail de *System.out.print*)

Créez une classe *test.* Dans celle-ci, écrivez le programme principal qui crée deux guerriers et les fait vieillir de quelques années. Le programme affichera les guerriers après chaque vieillissement.

Ecrivez la méthode *modifierExperience*. Exemple : *g.modifierExperience(n)* augmente l'expérience du guerrier *g* de *n* points. De la même manière, écrivez la méthode *modifierSante.*

Créez la méthode *estFaible* qui renvoie vrai si l'état de santé du guerrier est inférieur au seuil de faiblesse.

Dans la classe *test*, affaiblissez un guerrier jusqu'à ce qu'il devienne faible. Le programme affichera le guerrier après chaque affaiblissement.

Complétez la méthode *toString* de façon à ce qu'elle génère le texte "en bonne santé", "faible" ou "mort" pour compléter la présentation du guerrier.

Ecrivez la méthode *estJeune* qui renvoie vrai si le guerrier a moins de 30 ans ;
Ecrivez la méthode *estVieux* qui renvoie vrai si le guerrier a plus de 50 ans ;
Ecrivez la méthode *estAdulte* qui renvoie vrai si le guerrier n'est ni jeune, ni vieux.

Utilisez les constantes afin d'éviter d'écrire des valeurs numériques "en dur" dans les méthodes.

Modifiez la méthode *toString* de façon à ce qu'elle génère le texte "jeune", "vieux" ou "adulte" pour compléter la présentation du guerrier.

Ecrivez la méthode *affichage* qui renvoie '/' pour un guerrier faible et '|' pour un guerrier fort.

Créez un constructeur, sans paramètre, qui permet de créer un guerrier dont le nom sera "Anonyme-<NUM>" où <NUM> correspondra au nombre de guerrier qui ont été créés (attribut « private static int nb »).

Complétez la classe *test* afin de vérifier le fonctionnement de ces nouvelles méthodes.
Utilisez le débugger pour visualiser le contenu d'un guerrier durant l'exécution du programme.

2. Combat

Dans la classe *Guerrier*, créez la méthode *public void combattre(Guerrier adversaire)*.

Cette méthode permet de faire combattre deux guerriers *a* et *b*, avec l'instruction *a.combattre(b)*.

Vous pourrez mettre en œuvre le calcul suivant pour calculer le résultat du combat :

résultat = différence d'expérience + différence de force + entier aléatoire ∈[-chanceMax ; chanceMax]

Si le résultat est négatif, l'adversaire gagne, sinon c'est le guerrier *this* qui l'emporte.
Le gagnant voit son expérience augmenter de *gainExperience* points.
La santé du perdant est modifiée de *impactBlessure* points.

Dans le programme principal, faites se battre deux guerriers jusqu'à la mort de l'un des deux.

3. Evolution

Un guerrier peut vieillir, voir sa santé se dégrader ou se renforcer, et mourir. Ces changements d'état interviennent de manière "aléatoire" au cours de la vie du guerrier.

Chaque fois qu'un guerrier vieillit, le programme doit faire évoluer son état de santé.

Nous vous fournissons la méthode *evoluer*, qui utilise les règles de santé suivantes applicables à chaque unité de temps (c'est-à-dire lors de chaque vieillissement) :

- un jeune guerrier sain a 60% de chance de se renforcer (et donc 40% de chance de s'affaiblir)
- un jeune guerrier affaibli a 50% de chance de se renforcer
- un adulte sain a 70% de chance de se renforcer
- un adulte affaibli a 60% de chance de se renforcer

- un vieux guerrier sain a 40% de chance de se renforcer
- un vieux guerrier affaibli a 20% de chance de se renforcer

Les renforcements ont une valeur de 5 points, les affaiblissements ont toujours une valeur de 7 points.

Remarque : la méthode *evoluer* utilise la fonction *chance.*
chance(0.8) est un booléen qui a 80% de chances d'être vrai :

private static boolean chance(double x){ return Math.random()<x ; }

La méthode *evoluer* doit être appelée par la méthode *vieillir* (après l'incrémentation de l'âge).
Dans la classe *test*, faites vieillir un guerrier jusqu'à sa mort.
Un guerrier mort ne vieillit plus, et sa santé n'évolue plus. Modifiez *vieillir* en conséquence.
Exécutez plusieurs fois le programme et remarquez que les résultats peuvent être très différents.

B. La classe *Clan*

Un clan est un ensemble de guerriers.

Créez la classe *Clan* qui regroupe deux attributs : *elements* (un *ArrayList* de guerriers) et *nom* (le nom du clan).

Ecrivez les constructeurs (avec et sans paramètre de nom) permettant de créer un clan vide.

Ecrivez la méthode *ajouter*, qui permet d'ajouter un guerrier au clan.

Ecrivez la méthode *vieillir*, qui fait vieillir d'un an tous les guerriers du clan.

Ecrivez la méthode *toString* qui prépare l'affichage de tous les guerriers du clan.

Dans la classe de test, écrivez le programme permettant de tester la classe *Clan.*

Dans *Clan*, écrivez la méthode *estDecime*, qui renvoie vrai si tous les guerriers du clan sont morts.

Dans la classe de test, créez un clan composé de plusieurs guerriers et faites le vieillir jusqu'à ce que tous ses guerriers soient morts.

Dans *Clan*, écrivez la méthode *guerriersValides*, qui renvoie tous les guerriers qui ne sont pas faibles.

Dans *Clan*, écrivez la méthode *nettoyer*, qui supprime tous les guerriers morts.

Ecrivez la méthode *affichage*, qui renvoie un *String* de la forme "||/|/|" pour afficher les guerriers faibles et forts d'un clan. Testez toutes ces méthodes.

C. Simulations

On veut simuler l'évolution d'un clan sans aucune intervention de l'utilisateur.

Dans une nouvelle classe que vous nommerez *Simulation*, écrivez le programme *simuler* permettant de créer un clan, puis de faire vieillir tous ses guerriers et d'arrêter la simulation lorsque tous les guerriers sont morts.

Le programme devra afficher l'âge du clan c'est-à-dire l'âge du guerrier le plus vieux. Vous devrez compléter la classe *Clan* ...

Pour réaliser des simulations à grande échelle, modifiez la procédure *simuler* afin qu'elle crée un clan de 10 guerriers, qu'elle simule son évolution jusqu'à la mort du dernier guerrier, et qu'elle affiche l'âge du clan.

Pour créer 10 guerriers sans avoir à saisir leur nom, vous utiliserez le constructeur non paramétré.

Complétez la procédure *simuler* pour qu'elle effectue 20 simulations et qu'elle calcule l'âge moyen d'un clan de 10 guerriers.

D. Requête

Dans la classe *Clan*, écrivez la méthode *nombreDecesAdultes*, qui renvoie le nombre de guerriers morts alors qu'ils étaient dans l'âge adulte dans le clan *this*. Testez son fonctionnement.

E. Combats entre clans

Les règles d'un combat se déroulant entre deux clans peuvent être les suivantes : chaque guerrier du clan le moins nombreux affronte un guerrier tiré au sort dans le camps adverse.

Pour mettre en œuvre cette règle, vous devez d'abord compléter la classe *Clan* en écrivant l'observateur *getNieme(int n)*, qui renvoie le n^{ième} *Guerrier* d'un clan.

Dans le programme principal, créez un combat entre deux clans et affichez le résultat.

Modifiez le programme pour que les deux clans combattent jusqu'à ce que l'un d'entre eux soit décimé.

Dans la classe *Clan*, écrivez la méthode *CombattreAMort* qui fait combattre deux clans jusqu'à ce que l'un des deux soit décimé.

F. Annexe : documentation concernant la structure de données ArrayList

```
ArrayList <Double> ens ;          // Déclaration de la variable ens en tant qu'ArrayList de Double
ens = new ArrayList() ;           // Construction de l'objet ens
ens.add(12.5) ;                   // Ajout d'éléments dans la structure de données ens
System.out.print(ens.get(2)) ;    // Accès à un élément
System.out.print(ens.size()) ;    // Nombre d'éléments
ens.remove(2) ;                   // Suppression de l'élément n°2
if(ens .isEmpty()) …              // Test de vacuité
```