IPO – TP 9 : classes abstraites

http://www.lri.fr/~blsk/IPO/

On souhaite réaliser un jeu dans lequel différentes créatures se déplacent dans un labyrinthe : des personnages cherchant à s'échapper, et des monstres susceptibles de manger ces personnages. Le labyrinthe prendra la forme d'un ensemble de cases organisées dans un tableau à deux dimensions. Chaque case peut être soit une case intraversable (un mur), soit une case libre, soit une sortie. En plus des personnages et des monstres, les cases peuvent contenir des obstacles, que les créatures sont susceptibles de détruire.

Un objectif de ce TP est d'organiser le code des différentes sortes de cases et des différents occupants du labyrinthe en une hiérarchie de classes et de classes abstraites, de sorte à éviter au mieux les redondances dans le code. On fournit sur la page du cours deux squelettes de classes `Jeu` et `Terrain`, une énumération `Direction`, et le descriptif d'un terrain exemple (`laby1.txt`).

Cases Chaque case est caractérisée par ses coordonnées (numéro de ligne et numéro de colonne dans le tableau). Chaque case libre ou sortie peut contenir au plus une entité (obstacle ou créature). Une case intraversable ne contient rien.

1. Définir une classe abstraite `Case` avec deux attributs `lig` et `col` désignant les coordonnées de la case, et une méthode abstraite de signature `boolean estLibre()`. Note : une fois une case créée, les valeurs des deux attributs `lig` et `col` ne changent jamais.
2. Définir une classe abstraite `Entite`, avec un attribut entier `resistance` et une méthode abstraite de signature `String toString(String background)`.
3. Définir une classe `CaseTraversable` étant `Case` et possédant un attribut `contenu` (une entité), et les méthodes suivantes.
   - `Entite getContenu()` : renvoie le contenu,
   - `void vide()` : retire le contenu, et
   - `void entre(Entite e)` : définit le nouveau contenu, en supposant que la case ne contenait pas encore d'entité.
4. Définir trois classes `CaseIntraversable`, `CaseLibre` et `Sortie` respectant la description précédente. Chacune doit étendre `Case` (éventuellement via `CaseTraversable`), et concrétiser la méthode `estLibre()` de sorte à ce qu'elle renvoie **`true`** si et seulement si la case concernée est susceptible d'accueillir une entité *et* n'est pas déjà occupée.
5. Ajouter des méthodes `toString` aux classes des cases de sorte que chacune s'affiche sous la forme d'une chaîne de 3 caractères :
   - "###" pour les cases intraversables,
   - " " pour les cases libres,
   - "( )" pour les sorties.
6. Ajouter à la classe `Terrain` une méthode `void print()` qui affiche le terrain complet en utilisant les représentations précédentes.

Obstacles Les obstacles sont des entités, qui peuvent se trouver soit sur une case libre soit sur une sortie. Les obstacles ne peuvent pas se déplacer, mais sont détruits lorsque leur `resistance` atteint zéro.

7. Définir une classe `Obstacle` concrétisant `Entite`. On veut deux constructeurs : un constructeur par défaut fixant la résistance à 3, et un constructeur prenant une résistance en paramètre. La méthode `String toString(String background)` doit être concrétisée de la manière suivante.
   - On suppose que `background` est une chaîne de longueur 3 représentant la case sur laquelle l'obstacle est placé.
   - Si l'obstacle a une résistance supérieure ou égale à 3, la chaîne produite est "@@@".

— Si l'obstacle a une résistance de 1, on n'affiche qu'un symbole `'@'`, entouré du premier et du dernier caractères de `background`. Dans ce cas on doit donc obtenir `" @ "` avec l'appel `toString("  ")` et `"(@)"` avec l'appel `toString("( )")`.
— Similairement, si l'obstacle a une résistance de 2 on forme une chaîne avec deux symboles `'@'` et un symbole de `background`.

Entités mobiles Les personnages et les monstres sont deux cas particuliers d'entités mobiles. Chacun possède une direction (susceptible d'évoluer avec le temps).

8. Créer une classe abstraite `EntiteMobile`.
9. Créer les classes `Personnage` et `Monstre`. Chacune doit posséder une version concrète de la méthode `String toString(String background)` renvoyant une chaîne formée d'un symbole central entouré du premier et du dernier caractères de `background`. Le symbole central dépend du type de créature et de sa direction :

	`Personnage`	`Monstre`
nord	`'^'`	`'m'`
sud	`'v'`	`'w'`
est	`'>'`	`'»'`
ouest	`'<'`	`'«'`

   On doit donc renvoyer par exemple :
   — `" < "` pour un personnage tourné vers l'ouest sur un fond `"  "`, et
   — `"(w)"` pour un monstre tourné vers le sud sur un fond `"( )"`.
   Note : les affichages des personnages et des monstres étant similaires, il est possible (mais non obligatoire) de partager une partie du code à écrire dans la classe `EntiteMobile`.
10. Modifier les méthodes `toString` des classes des cases susceptibles d'accueillir une entité de sorte que, lorsqu'une case possède effectivement un contenu, on affiche la combinaison produite par les méthodes `String toString(String background)`.

Note : à ce stade, vous pouvez utiliser le constructeur fourni dans la classe `Terrain`, qui initialise un terrain à partir d'une description donnée par un fichier texte (voir `laby1.txt`).

Actions Un tour de jeu consiste à faire agir une créature tirée au hasard parmi les créatures présentes. Les créatures obéissent aux règles suivantes.

— Un `Personnage` sur une `Sortie` sort du jeu (et on incrémente un compteur des personnages sauvés).
— Un `Personnage` ou un `Monstre` face à une case libre avance.
— Un `Personnage` face à un `Obstacle`, décrémente la résistance de l'`Obstacle` de 1.
— Un `Monstre` face à un `Obstacle` ou un `Personnage` décrémente la résistance de sa cible de 1.
— Dans tous les autres cas : changement de `Direction` aléatoire.

Un `Personnage` ou un `Obstacle` dont la résistance tombe à zéro disparaît du jeu.

11. Faire que chaque `EntiteMobile` contienne une méthode **`void`** `action(Case courante, Case cible)`, qui fait agir la créature selon le descriptif ci-dessus. On suppose que `courante` est la case occupée par la créature, et `cible` la case qui se trouve devant elle. Note : cela nécessite également d'inclure un moyen de décrémenter la résistance d'une `Entite`, et de détecter les cas où une `Entite` doit être retirée du jeu.
12. Écrire dans la classe `Jeu` une méthode **`void`** `tour()` qui sélectionne de manière aléatoire une case contenant une créature, et fait agir cette créature. Ajouter une méthode `main` qui enchaîne les tours de jeu et affiche le terrain à chaque étape.

Extensions

13. Faire qu'à chaque tour, chaque créature joue exactement une fois.
14. Arrêter le jeu lorsque tous les personnages sont sauvés ou mangés.
15. Ajouter à volonté d'autres types de cases et d'autres types de créatures.