Examen juin 2006

3 heures
documents de cours, TD et TP autorisés
livres, portables et calculatrices interdits

Exercice 1 : Bataille Navale

A la "bataille navale", le plateau de jeu est représenté par une grille rectangulaire sur laquelle on peut poser des bateaux. Les bateaux sont larges d'une case et longs d'un nombre variable de cases. Ils peuvent être posés verticalement ou horizontalement sur le plateau.
Le plateau est masqué au joueur qui "attaque" et celui-ci doit couler tous les bateaux du joueur "défenseur". Pour cela, il propose une position du plateau. Si cette position n'est pas occupée par un bateau, le défenseur annonce "*dans l'eau*", dans le cas contraire il annonce "*touché*" si toutes les cases occupées par le bateau touché n'ont pas déjà été visées par l'attaquant, et "*coulé*" si toutes les autres cases du bateau ont déjà été touchées.

On s'intéresse à la programmation en JAVA d'un ensemble de classes permettant la programmation de ce jeu, mais on ne programmera pas tout le jeu. Les classes sont à placer dans le paquetage `naval`.

Les bateaux. Un objet bateau est défini par sa longueur. La longueur d'un bateau détermine son nombre de "points de vie" en début du jeu. A chaque fois qu'il est touché, sur une case jamais touchée auparavant bien sûr, ce nombre diminue et un bateau est coulé quand il arrive à 0 point de vie.

Q 1 . Donnez le code JAVA de la case `Bateau` qui correspondrait à ce diagramme UML :

Bateau
-vie : int
+Bateau(longueur : int) +estCoule():boolean +touche() +getVie():int

Dans l'eau, touché , coulé. On appelle `Reponse` le type permettant de représenter les 3 réponses possibles après une proposition d'un attaquant. Il s'agit d'un type énuméré pouvant prendre les 3 seules valeurs : `DANS_LEAU`, `TOUCHE` et `COULE`.

Q 2 . **Donnez** le code JAVA pour le type `Reponse`.

Le plateau On décide de représenter le plateau de jeu par un tableau à 2 dimensions de *cases*. Une case peut être vide ou occupée par un bateau. Le code de la classe `Case` est donné en annexe. L'attribut `bateau` d'une instance de cette classe vaut `null` si aucun bateau n'occupe cette case. Pour une case, l'information `visee` permet de savoir si l'attaquant a déjà visé ou non cette case, qu'elle comporte un bateau ou non. La classe représentant le plateau s'appelle `Mer`

Q 3 . **Donnez** le code définissant les attributs de la classe `Mer` ainsi que le constructeur sachant qu'initialement toutes les case sont vides (pas de bateau) et que les dimensions du plateau sont fixées à la construction.

Q 4 . **Donnez** le code de la méthode :

```
public Reponse vise(Position p)
```

dont le résultat est la réponse lorsque l'attaquant vise la case représentée par la position `p`. Le diagramme de la classe `Position` est donnée en annexe.

Cette méthode doit gérer l'évolution de l'attribut `visee` de la case à la position `p` ainsi que, le cas échéant, la gestion des points de vie du bateau qui s'y trouve.

Affichage On s'intéresse maintenant à l'affichage du plateau de jeu. On souhaite un affichage sur la sortie standard. Le plateau est affiché ligne par ligne et case par case. Cet affichage doit être différent selon que l'on est défenseur ou attaquant. On considère que la case de coordonnées (0,0) est la case en haut à gauche (la plus au nord-ouest) du plateau.
Pour le défenseur, le caractère affiché pour une case est `'.'` si la case est vide, `'B'` si elle est occupée par un bateau non touché et `'*'` si le bateau occupant cette case a été touché.
Pour l'attaquant, le caractère affiché est `'?'` pour une case qui n'a jamais été visée, il est `'.'` pour une case visée vide, et `'*'` pour une case occupée par un bateau touché.

Q 5 . **Complétez** le code JAVA de la méthode `affichage` décrite ci-dessous. Pour cela vous utiliserez la méthode `getCaractere` de la classe `Case` (cf. Annexe) après en **avoir donné** le code. La signification du paramètre de cette méthode est la même que celui du paramètre de la méthode `affichage`.

```
/** affiche le plateau ligne par ligne, case par case en
* proposant un affichage différencié pour le défenseur et
* l'attaquant.  Le paramètre defenseur détermine l'affichage
* à réaliser
* @param defenseur true si l'affichage est pour le défenseur,
   false si il est pour l'attaquant.
*/
public void affichage(boolean defenseur) {
   à compléter
}
```

Placement de bateau On s'intéresse maintenant au placement des bateaux sur le plateau.
Les bateaux ne peuvent être posés que selon des directions imposées. La méthode permettant de poser un bateau est la suivante (voir en annexe pour le diagramme UML de la classe d'exception) :

```
/** pose le bateau b à partir de la position p dans la
* direction d.  Le nombre de case occupée est bien sûr
* déterminé par la longueur du bateau.
* @param b le bateau à poser
* @param p la position de la première case où l'on pose le bateau
* @param d la direction dans laquelle on pose b à partir de p
* @exception PoseImpossibleException si on ne peut poser b à
* partir de p dans la direction d parce que b sortirait du plateau.
*/
```

Une direction correspond à un décalage de plus ou moins 1 dans le sens des `x` ou des `y` pour une position donnée (on rappelle que la case de coordonnées (0,0) est la case en haut à gauche, c'est-à-dire la plus au nord-ouest). On définit pour cela l'interface `Direction` ainsi :

```
public interface Direction {
/** fournit la position voisine de p dans cette direction
* @return la position voisine de p dans cette direction
*/
public Position positionVoisine(Position p);
}
```

Q 6 . **Donnez** le code d'une classe `DirectionNord` qui définit une direction dans laquelle la "position voisine" est au nord de la position donnée (décalage selon les `y`).

Q 7 . **Même question** avec `DirectionEst` pour définir la direction "vers l'est" (décalage selon les `x`).

Q 8 . **Donnez** le code JAVA de la méthode `poseBateau`. Cette méthode lève une exception si en posant le bateau case par case à partir de la position de départ, on est amené à sortir des bornes du plateau.

Q 9 . **Ecrivez** une méthode `main` qui :

- crée un plateau de jeu 10 cases de large et 15 cases de haut,
- crée 1 bateau de longueur 3 et le place sur le plateau à partir de la position (2,3) dans la direction est,
- crée 1 bateau de longueur 4 et le place sur le plateau à partir de la position (6,5) dans la direction nord,
- affiche la réponse obtenue lorsqu'un attaquant vise la case (3,10),
- affiche le plateau pour le défenseur puis pour l'attaquant.

Exercice 2 : Palmarès

On souhaite gérer un palmarès d'une compétition dans lequel on mémorise le meilleur résultat de chacun des participants à la compétition.
L'ajout d'un participant se fait lors de la mémorisation de son premier résultat à la compétition.
Les fonctionnalités dont on souhaite disposer sont les suivantes :

▷ enregistrer un résultat pour un participant,

▷ connaître le meilleur résultat d'un participant donné, une exception `ParticipantInconnuException` est levée si le participant est inconnu (voir Annexes),

▷ afficher tous les participants et leur résultat (un participant par ligne),

▷ connaître le participant qui a le meilleur résultat,

▷ connaître le meilleur résultat.

Les participants sont définis par le type (minimal) suivant :

Participant
+hashCode():int +equals(o :Object):boolean +toString():String

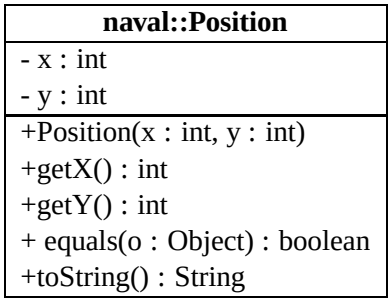
On supposera que les résultats sont des entiers.

Q 1 . **Donnez** le code JAVA de la classe `Palmarès`.

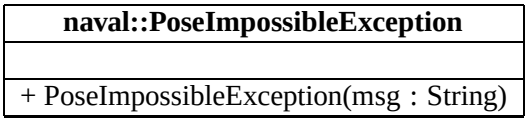
Annexes

```
package naval;
public class Case {
   private Bateau bateau;
   private boolean visee;
   public Case() {
      this.bateau = null;
      this.visee = false;
   }
   public Bateau getBateau() {
      return this.bateau;
   }
   public void setBateau(Bateau bateau) {
      this.bateau = bateau;
   }
   public boolean aEteVisee() {
      return this.visee;
   }
   public void visee() {
      this.visee = true;
   }
   public char getCaractere(boolean defenseur) {
      ... voir question 5
   }
}
```

Le diagramme de la classe `naval.Position` est le suivant :



Le diagramme de la classe `naval.PoseImpossibleException` est le suivant :



Le diagramme de la classe `ParticipantInconnuException` est le suivant :