Université Grenoble Alpes
Licence 2 MIASHS
Inff3 – Algorithmique et programmation par objets
2022-2023

TP5 : Bataille Navale, première partie

Le but de ce TP est de commencer le développement d'un jeu de bataille navale (voir http://fr.wikipedia.org/wiki/Bataille_navale_%28jeu%29) . Dans un premier temps, nous allons nous concentrer sur la définition du système de coordonnée et des navires. Ensuite nous nous intéresserons à la grille de jeu et finalement dans un prochain TP nous réaliserons l'interface graphique.

Chacune des fonctionnalités (méthodes, constructeurs) devra être testée avec Junit, un framework de tests très utilisé en Java. Vous devez tester au fur et à mesure et ne pas attendre la fin. Le code devra être commenté en utilisant les commentaires Javadoc lorsque cela est approprié.
Les tests de la classe Coordonnee sont dans le projet de base fourni. Pour importer le projet, il faut (1) dézipper le projet dans votre workspace eclipse, (2) importer via File → Import → Maven → Existing Maven Projects.

Les classes seront créées dans un package `fr.uga.miashs.inff3.bataillenavale`. Tous les attributs seront privés.

1– Classe `Coordonnee`

La bataille navale utilise un système de coordonnées du type A1 où A désigne la colonne et 1 désigne la ligne.
Pour réaliser automatiquement ces conversions, nous allons définir une classe `Coordonnee`. Cette classe possède deux attributs représentant la ligne et la colonne sous forme d'indices commençant à 0. Pour convertir la colonne d'une coordonnée au format lettre en indice il faut soustraire le caractère `'A'` à la lettre que l'on veut convertir.
Par exemple, pour la coordonnée « C1 », l'attribut ligne sera initialisé à 0 et l'attribut colonne à 2.

1.1- Constructeurs

Ajoutez un premier constructeur qui prend en paramètre une chaîne de caractères contenant les coordonnées dans le système « bataille navale ». Pour réaliser ce constructeur, utilisez les méthodes `charAt(…)` et `subString(...)` de la classe `String`, et la fonction `Integer.parseInt(…)` qui permet de parser un entier à partir d'une chaîne de caractères.
NB: l'expression `'A'-'A'`, convertie en `int` donne 0 en Java (`'B'-'A'` donne 1, etc.).
Ajoutez un deuxième constructeur qui prend en paramètre la ligne et la colonne sous forme d'entiers.

1.2- Méthodes d'accès aux attributs

Définissez les deux méthodes `getLigne()`, et `getColonne()`.

1.3– Méthode `toString()`

Ajoutez la méthode `public String toString()` qui retourne une représentation en coordonnées « bataille navale ».
Testez systématiquement cette classe (constructeurs et méthodes) dans un programme Test. Vous en

ferez de même pour toutes les questions suivantes.

1.4– Méthode `equals(…)`

Ajoutez la méthode **`public boolean`** `equals(Object o)` qui retourne `true` ssi la coordonnée courante possède la même ligne et la même colonne que celle passée en paramètre.
Pour tester à l'exécution que o est bien une instance de `Coordonnee`, vous pouvez utiliser le code suivant :

```
if (o instanceof Coordonnee) {
      Coordonnee c = (Coordonnee) o;
      // On passe ici si o référence une coordonnée
      // il faut tester l'égalite
}
return false;
```

1.5– Méthode voisine `(…)`

Ajoutez la méthode **`public boolean`** `voisine(Coordonnee o)` qui retourne `true` si la coordonnée courante est voisine de celle passée en paramètre. On considère seulement le 4-voisinage (i.e. les coordonnées en diagonale ne sont pas voisines).

1.6– Méthode compareTo `(…)`

Ajoutez la méthode **`public int`** `compareTo(Coordonnee o)` qui permet de comparer deux coordonnées. Une coordonnée est plus petite qu'une autre si elle est placée avant dans le sens de la lecture (gauche à droite, haut en bas). Cette méthode retourne un nombre négatif si la coordonnée courante est plus petite que `o`, 0 si elle sont égales, un nombre positif si elle est plus grande que `o`.

2– La classe `Navire`

Un navire est représenté par une classe ayant les attributs suivants :

- **`private`** `Coordonnee debut;`
- **`private`** `Coordonnee fin;`
- **`private`** `Coordonnee[] partiesTouchees;`
- **`private int`** `nbTouchees;`

`debut` est la coordonnée du début et `fin` est la coordonnée de fin du navire. La ligne (resp. la colonne) de la coordonnée de début est toujours **inférieure ou égale** à la ligne (resp. la colonne) de la coordonnée de fin.
Le tableau `partiesTouchees` contient les coordonnées du navire qui sont touchées, et l'entier `nbTouchees` le nombre de parties du navire réellement touchées.

2.1- Constructeur de la classe Navire

Définissez le constructeur **`public`** `Navire(Coordonnee debut,` **`int`** `longueur,` **`boolean`** `estVertical)`.
Ce constructeur calculera la coordonnée de fin à partir des paramètres longueur et estVertical.
L'attribut `partiesTouchees` sera instancié et l'attribut `nbTouchees` initialisé à 0.

2.2- Méthode toString()

Réalisez la méthode **`public`** `String toString()` qui affichera une représentation textuelle d'un navire, .i.e. sa cordonnée de début, celle de fin ainsi que les coordonnées touchées.

2.3- Méthodes d'accès aux attributs

Définissez les deux méthodes `getDebut()`, et `getFin()`. Ces méthodes retournent la coordonnée de début et la coordonnée de fin du navire

2.4- Méthode `contient(...)`

Définissez la méthode **`public boolean`** `contient(Coordonnee c)`. Cette méthode retourne `true` si la coordonnée passée en paramètre est contenue dans le navire.

2.5- Méthode statique `intersectionNonVide(…)`

Définissez la méthode **`private static boolean`** `intersectionNonVide(`**`int`** `d1,` **`int`** `f1,` **`int`** `d2,` **`int`** `f2)`. Cette méthode retourne true ssi les intervalles `[d1; f1]` et `[d2; f2]` ont une intersection non vide. Cette méthode sera utile pour les deux méthodes suivantes.

2.6- Méthode `chevauche(…)`

Définissez la méthode **`public boolean`** `chevauche(Navire n)`. Cette méthode retourne `true` si le navire chevauche le navire passé en paramètre. Pour qu'un navire chevauche un autre navire, il faut que les intervalles formés par les coordonnées de début de fin s'intersectent à la fois au horizontalement (intervalles des abscisses) et verticalement (intervalles des ordonnées).

2.7- Méthode `touche(…)`

Définissez la méthode **`public boolean`** `touche(Navire n)`. Cette méthode retourne `true` si le navire touche le navire passé en paramètre (on considère que deux navires adjacents au niveau de la diagonale ne se touchent pas). Pour que deux navires se touchent il faut que les deux navires s'intersectent sur l'une des dimensions (horizontalement ou verticalement) et qu'ils se touchent sur l'autre dimension.

2.8- Méthode `estTouche(…)`

Définissez la méthode **`public boolean`** `estTouche(Coordonnee c)`. Cette méthode retourne `true` si la coordonnée `c` est contenue dans `partiesTouchees` , `false` sinon.

2.9- Méthode `recoitTir(…)`

Définissez la méthode **`public boolean`** `recoitTir(Coordonnee c)`. Cette méthode retourne `true` si le tir de coordonnée `c` touche le navire. L'attribut `partiesTouchees` et `nbTouchees` seront mis à jour si la coordonnée c touche le navire et si elle n'est pas déjà présente dans le tableau.

2.10- Méthodes `estTouche()` et `estCoule()`

Définir les méthodes **`public boolean`** `estTouche()` et **`public boolean`** `estCoule()`.
La méthode `estTouche()` retourne true si le navire a été touché au moins une fois.