Le monde de l'objet TD/TP n°2

La bataille peut commencer …

Enoncé

L'objectif est de continuer le voyage dans le monde de l'objet en Java en réalisant cette fois-ci une « Bataille des Etoiles ». Elle vous paraîtra sans doute comme une (r)évolution de la classique bataille navale mais la fin justifie les moyens : un sujet banal d'étude ne permet pas de cerner la plus value de la conception objet, ni la puissance d'un langage objet comme Java ...

Pour rester dans la continuité de vos dernières réflexions sur la Force et son Côté Obscur, nous resterons dans le décor de « Star Wars » et nous nous occuperons de la **bataille d'Endor de l'épisode VI** de la trilogie originelle (« Le Retour du Jedi »).

[... Cette bataille se déroule durant le duel final entre Luke Skywalker et son père corrompu Dark Vador sur la Seconde Étoile Noire). Ce duel marque la « mutation » inverse de Dark Vador en Anakin Skywalker ...]

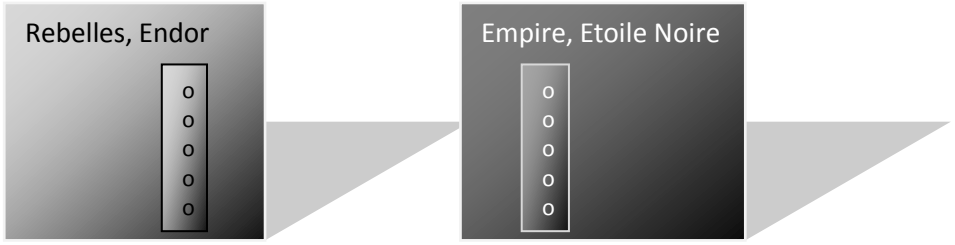
Sommaire

I. Synopsis de la bataille

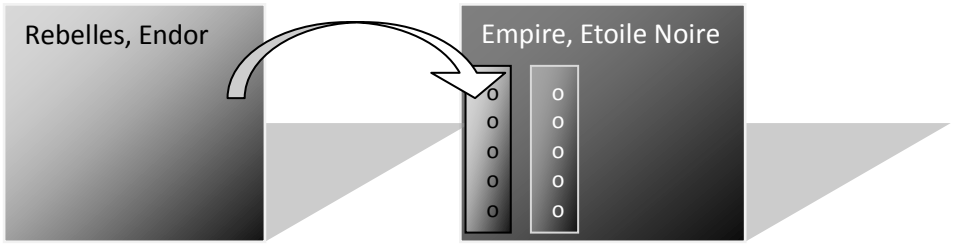
Cette bataille oppose l'Alliance Rebelle – dont les forces armées orbitent autour de la planète Endor – à l'Empire Galactique dont les vaisseaux gravitent autour de la Seconde Etoile Noire (orbitant elle-même autour d'Endor)

Le principe de la bataille consiste en une succession d'assauts entre les deux belligérants. Voici le scénario de base : une vague de vaisseaux de l'Alliance Rebelle déferle sur l'Etoile Noire et bombarde (un peu au hasard) les vaisseaux en stationnement de l'Empire Galactique. Une fois l'assaut des Rebelles terminé, sa flotte rentre vers Endor et compte ses victoires puis l'Empire lance son assaut vers Endor sur le même principe ... La bataille se termine lorsque toute la flotte d'un belligérant est entièrement détruite.

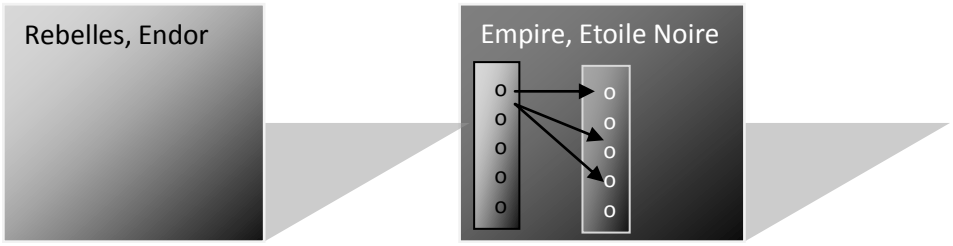
1. Avant la bataille la flotte de chaque camp est en stationnement sur son territoire représenté par une grille 10 x 10 : chaque vaisseau en stationnement a donc une position (x,y),



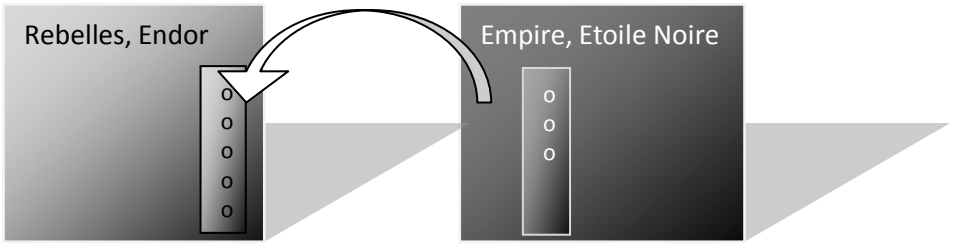
2. Assaut des Rebelles : la flotte se rend sur l'Etoile Noire et lance l'assaut une fois à destination,
3. Combats sur l'Etoile Noire : chaque assaillant rebelle attaque sur une position (x,y) et chaque



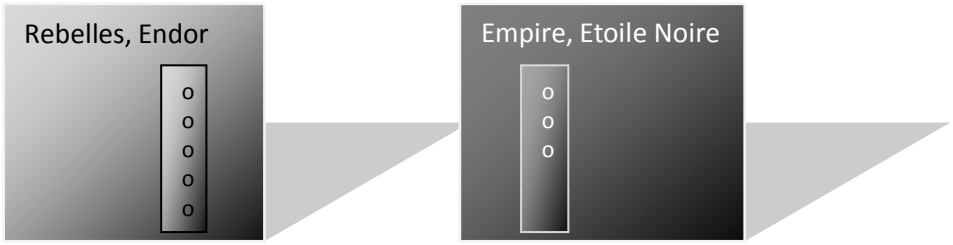
vaisseau de l'Empire répond à cette attaque (« touché », « détruit », « indemne »)



4. Retour de la flotte rebelle : chaque vaisseau fait son rapport c-à-d le résultat de son attaque en (x,y) et le bilan global de l'assaut : combien reste-t-il d'ennemis.



5. C'est maintenant le tour de l'Empire de lancer son assaut vers Endor mais vous constaterez que sa flotte est diminuée ...



II. Conception et réalisation

Les Vaisseaux

Que ce soit du côté de l'Alliance Rebelle ou de l'Empire Galactique, un vaisseau est muni des méthodes et des attributs suivants

- **Un modèle** (chaine de caractères)
- **Une position de stationnement** (x,y) sur la grille 10x10 de son territoire et fixée aléatoirement ou non à la création du vaisseau
- **Une force d'attaque** (variable selon le modèle mais constante dans le temps)
- **Un niveau de défense** (variable selon le modèle et décroissant suivant les attaques subies)
- **Un statut opérationnel** (méthode rendant un booléen vrai si le niveau de défense > 0)

1. Concevez et codez **une classe abstraite** « `Vaisseau` » répondant à ce cahier des charges
2. Redéfinissez/recodez la méthode **« toString() »** de la classe mère Object de manière à obtenir des informations de « debuggage » (cf. en Annexe et les tests ci-après)
3. Réalisez par l'intermédiaire de **classes concrètes** les modèles de Vaisseau suivant :

Classe	Modèle	Force d'attaque	Niveau de défense	Position
XWing	« X-Wing »	4	2	« random »
MilleniumFalcon	« Millenium Falcon »	8	10	« random »
TFighter	« T-Fighter »	2	4	« random »

4. Dessinez votre hiérarchie de classes.
5. Codez la classe de test (cf. en Annexe) « `TestVaisseau` » qui vous permettra de vérifier que vous pouvez instancier chacune des 3 classes concrètes aussi bien avec une position aléatoire que définie à la construction.

ATTENTION NOUS N'AVONS A CE POINT QU'UN MINIMUM DECRIVANT L'ETAT INITIAL D'UN VAISSEAU !

LORS D'UNE CONFRONTATION AVEC UN AUTRE VAISSEAU IL FAUDRA RAJOUTER DES COMPORTEMENTS (METHODES) ET DES ETATS (ATTRIBUTS) A GERER !

Les Belligérants

L'Alliance Rebelle et l'Empire Galactique sont des ennemis mortels, ils peuvent néanmoins être modélisés de la même manière car chaque belligérant possède au moins :

- **Un camp de ralliement** (une chaine de caractères figée à la construction du belligérant)
- **Une flotte de vaisseau** (un tableau de 4 vaisseaux « `Vaisseau[] flotte = new Vaisseau[4] ;` »
- **Une capacité à donner l'état de la flotte** (combien de vaisseaux opérationnels)

1. Concevez et codez une classe **abstraite** « **`class Belligerant`** » répondant à ce cahier des charges.
2. Redéfinissez/recodez la méthode **« toString() »** de la classe Object de manière à obtenir des informations de « debuggage » de la même manière que « Vaisseau » (cf. en Annexe et la classe de test ci-après).
3. Réalisez par l'intermédiaire de **classes concrètes les deux belligérants** de la bataille d'Endor :

Classe	Camp de ralliement	Flotte (nb vaisseaux)	Vaisseaux
EmpireGalactique	« Empire Galactique sur l'Etoile Noire »	4	4 T-Fighter
AllianceRebelle	« Alliance Rebelle sur Endor»	4	1 Falcon Millénium 3 X-Wing

4. Dessinez votre hiérarchie de classes.
5. Codez la classe de test « **`TestBelligerant`** » (cf. en Annexe) vous permettant de vérifier que vous pouvez instancier chacune des 2 classes concrètes et de vérifier leur bon fonctionnement (affichage des états).

ATTENTION NOUS N'AVONS A CE POINT QU'UN MINIMUM DECRIVANT L'ETAT INITIAL D'UN BELLIGERANT !

Interfaces de confrontation entre vaisseaux Combattant

Dans notre jeu de bataille par assaut, quand deux vaisseaux s'affrontent : l'un d'eux a un comportement de « **Combattant** » qui « **attaque** » une « **Cible** » (l'autre vaisseau) qui enclenche alors son mécanisme de « **défense** »

Nous avons donc deux **comportements** que nous modéliserons à l'aide de deux **interfaces** Java (de seulement trois lignes de code !)

Comportement « Combattant »	Comportement « Cible »

`1 public interface Combattant {`
`2  void attaquer(Cible c);`
`3 }`
	`1 public interface Cible {`
	`2  int seDefendre (int x, int y, int force);`
	`3 }`

Appeler la méthode «seDefendre» de la cible et stocker le résultat.	*En retour d'appel, « seDefendre » donne :* • *-1 : cible indemne* • *0 : cible touchée* • *1 : cible détruite*

Comprenez bien que c'est l'implémentation de la méthode « attaque » qui invoque la méthode « seDefendre » de la cible passée en paramètre.

1. Intégrez le fait que la classe « Vaisseau » réalise ces deux comportements.
2. Complétez votre diagramme de classes en ajoutant ces interfaces.
3. Codez l'implémentation de ces deux interfaces dans la classe « **`Vaisseau`** » (étant donné la dépendance de Combattant sur Cible : commencez par Cible !)
4. Pensez qu'un vaisseau attaque **seulement s'il est opérationnel** !
5. Codez la classe de test « **`TestCombattantCible`** » (cf. en Annexe) vous permettant de vérifier le bon fonctionnement de la confrontation entre deux vaisseaux concrets;

ATTENTION : IL VA MAINTENANT ETRE EVIDENT « QU'IL MANQUE DU CODE (METHODE ET ATTRIBUTS) » POUR QUE CELA FONCTIONNE AU REGARD DE LA CLASSE DE TEST (QUI VOUS AIDE BEAUCOUP)

Liste des choses à faire dans la classe Vaisseau (TODO List) :

- Un attribut (un point) **« localisationAttaque »** pour savoir où attaquer !
- Un moyen de donner l'ordre de mission, c.-à-d. affecter « localisationAttaque ») **avant l'assaut.** (méthode **« prendreOrdreAttaque(...)»**)
- Un attribut (entier) **« resultatAttaque »** stockant la réponse de la cible à l'attaque (retour de la méthode « seDefendre »), pensez qu'il s'agit peut être « du meilleur score » par assaut ...
- Un moyen de récupérer « resultatAttaque » **après l'assaut** (**« donnerResultatAttaque() »**)
- Augmenter significativement les informations affichées par **« toString() »**

Interfaces d'assaut entre belligérants

A l'instar des interfaces de confrontation entre vaisseaux, il existe un protocole similaire entre nos deux belligérants : un « Assaillant lance l'assaut sur une Forteresse » et la « Forteresse résiste à cet assaut » qui est constituée de la flotte de «Combattants» de l'Assaillant.

Nous avons donc ici aussi deux comportements que nous modéliserons à l'aide de deux interfaces Java (aussi compactes que précédemment !)

Comportement « Assaillant »	**Comportement « Forteresse »**

`public interface Assaillant {`
`  void lanceAssaut(Forteresse f);`
`}`
	`public interface Forteresse {`
	`  int resisteAssaut(Combattant[] flotte);`
	`}`

Préparer les ordres d'attaques *Envoyer la flotte sur la forteresse* *Traiter le résultat ?*	*En retour d'appel, « resisteAssaut » donne le nombre de cibles restantes*

Comprenez bien, ici aussi, que c'est l'implémentation de la méthode « lanceAssaut » qui invoque la méthode « resisteAssaut» de la Forteresse passée en paramètre ;

1. Intégrez le fait que la classe abstraite « **`Belligerant`** » réalise ces deux comportements.
2. Complétez votre diagramme de classes en ajoutant ces interfaces.
3. Codez l'implémentation de ces deux interfaces dans la classe « **`Belligerant`** ».
4. Codez la classe de test « **`TestAssaillantForteresse`** » (cf. en Annexe) vous permettant de vérifier la confrontation entre les deux instances concrètes de l' « **`AllianceRebelle`** » et l' « **`EmpireGalactique`** »

ATTENTION : IL FAUT CODER TRES PEU DE CHOSE POUR UN FONCTIONNEMENT MINIMUM (SANS INTELLIGENCE)

Liste des choses à faire dans la classe Belligerant (TODO List) :

- Gérer à minima la préparation de l'assaut : donner un ordre d'attaque à chaque vaisseau (au hasard) et envoyer la flotte à l'assaut

- Augmenter significativement les informations affichées par **« toString() »**

Le moteur de jeu final : « La Bataille d'Endor »

Maintenant vous devriez vous apercevoir que vous possédez tous les éléments nécessaires pour réaliser le synopsis qui vous a été présenté ... Il vous reste à faire la mise en scène au travers d'une classe principal « `BatailleEndor` » qui doit :

- Instancier l' « Empire Galactique » et l' « Alliance Rebelle»
- Décider qu'initialement la référence « **`Belligerant assaillant = ...`** » est affectée à l'instance « Alliance Rebelle »
- A fortiori, initialement la référence « **`Belligerant forteresse = ...`** » est l'instance de « Empire Galactique »
- Boucler sur les assauts tant que la forteresse en cours comporte encore des vaisseaux opérationnels (n'oubliez pas d'alterner l'assaillant et la forteresse)

```
public class BatailleEndor {

    private Belligerant assaillant ;
    private Belligerant forteresse ;

    public Belligerant lancerEtDonnerVainqueur() {

        // votre code d'assaut ici !

    }

    public static void main(String[] args) {
        int nbRound = 1000;
        int empireVainqueur = 0;
        int rebellesVainqueur = 0;
        for (int round = 0; round < nbRound; round++) {
            BatailleEndor bataille = new BatailleEndor();
            Belligerant vainqueur = bataille.lancerEtDonnerVainqueur();
            if (vainqueur instanceof AllianceRebelle) {
                rebellesVainqueur++;
            } else {
                empireVainqueur++;
            }
        }
        System.out.println(" Résultat pour "+nbRound
                +" : Empire ["+empireVainqueur+"]"
                +" Rebelles ["+rebellesVainqueur+"]");
    }
}
```

Remarquez le mot clef **`instanceof`** utilisé (et pratique, mais avec modération) pour tester l'appartenance à un type.

Affichage à l'exécution :

```
Résultat pour 1000 : Empire [72] Rebelles [928]
```

VOUS DEVRIEZ TROUVER QUE L'ALLIANCE REBELLE GAGNE 90% DU TEMPS

III. Evolutions

Une évolution structurelle de la flotte

Au lieu de conserver ce tableau pénible et statique de Vaisseaux dans le Belligérant, remplacez-le par une « **List<Vaisseau>** » Java ce qui vous permettra sans doute de simplifiez le code déterminant le nombre de vaisseaux opérationnels et surtout d'avoir une liste extensible sans vous soucier des débordements !

Essayer ce petit programme (qui mélange les arguments de la ligne de commande) pour comprendre le fonctionnement des listes (des collections) en Java :

```
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;

public class Shuffle {

    public static void main(String[] args) {
        List<String> list = new ArrayList<String>();
        for (int i = 0; i < args.length; i++) {
            list.add(args[i]);
        }
        Collections.shuffle(list, new Random());
        for (String s : list) {
            System.out.print(s);
            System.out.print(" ");
        }
        System.out.println();
    }
}
```

Exécution :

```
> java Shuffle 1 2 3 4 5 6 7 8 9
8 4 7 3 6 2 1 9 5
> java Shuffle a b c d
b d a c
>
```

Un peu de lecture,

En ligne :

http://java.sun.com/docs/books/tutorial/collections/index.html

Ou sur votre machine :

[HOME_TUTORIAL]/tutorial/collections/index.html

Un peu de pseudo-code pour démarrer :

```
public abstract class Belligerant implements Assaillant, Forteresse {

    private String ralliement;

    protected List<Vaisseau> flotteVaisseaux;

    public Belligerant(String ralliement) {

        this.ralliement = ralliement;

        flotteVaisseaux = new ArrayList<Vaisseau>();

    }

...
```

VOUS DEVEZ CHANGER AUSSI VOTRE CODE DANS ALLIANCEREBELLE & EMPIREGALACTIQUE

1. Combien de temps avez-vous mis pour cette évolution ?
2. Votre conception vous a-t-elle permis de gagner en productivité ? Pourquoi ?
3. Votez-vous pour ou contre la programmation Objet ? Pourquoi ?
4. De manière anecdotique avez-vous changé la règle du jeu pour les vaisseaux détruits lorsqu'ils repartent à l'assaut ?

Une évolution qualitative et quantitative des flottes

Ajoutez les modèles de vaisseau suivant :

Classe	Modèle	Force d'attaque	Niveau de défense
AWing	« A-Wing »	2	2
BWing	« B-Wing»	6	8
DVTFighter	« Dark Vador T-Fighter »	8	10

1. Combien de temps avez-vous mis pour cette évolution ?
2. Votre conception vous a-t-elle permis de gagner en productivité ? Pourquoi ?
3. Votez-vous pour ou contre la programmation Objet ? Pourquoi ?

Etoffez les deux flottes selon le schéma suivant :

Classe	Camp de ralliement	Flotte (nb vaisseaux)	Vaisseaux
EmpireGalactique	« Empire Galactique sur l'Etoile Noire »	9	8 T-Fighter 1 DVT-Fighter
AllianceRebelle	« Alliance Rebelle sur Endor»	9	1 Falcon Millénium 3 X-Wing 1 B-Wing 4 A-Wing

1. Combien de temps avez-vous mis pour cette évolution ?
2. Votre conception vous a-t-elle permis de gagner en productivité ?
3. Votez-vous pour ou contre la programmation Objet ? Pourquoi ?
4. Avez-vous favorisé la « Force » ou son « Côté Obscur » ? Quelles sont vos pourcentages ?

IV. Pour aller toujours plus loin : la vraie Force de l'Objet

Intelligence artificielle contre le hasard

Vous aurez sans doute remarqué que tout le travail est réalisé par les deux classes abstraites **Vaisseau** et **Belligerant** et que les classes concrètes ne font qu'initialiser des objets dans des états particuliers...

Nous pouvons **sans difficulté et « sans rien casser »** modifier le comportement hasardeux et sans intelligence des classes abstraites de manière quasiment infinie **dans les classes concrètes** :

- **Tester, en étendant EmpireGalactique avec une classe EmpireGalactiqueConcentre** si la concentration de la flotte sur un **même point** est favorable face à une AllianceRebelle «hasardeuse »
- **Tester, en étendant AllianceRebelle avec une classe AllianceRebelleBalayante** si des assauts avec une flotte **en formation « carré 3x3 » balayant** le territoire 10x10 de la Forteresse est favorable face à un EmpireGalactique «hasardeux » et à un « Empire Galactique Concentré »
- Finalement ne vaut-il pas mieux faire une **AllianceRebelleMemorisante** qui n'attaque pas les positions sans intérêt.
- Un **EmpireGalactiqueEsquivant** résiste-t-il mieux en déplaçant ses Vaisseaux d'une case quand ils sont touchés ?
- Imaginez des Vaisseaux « nettoyeurs» achevant une cible qu'ils ont touchés **malgré les ordres** (le « Millenium Falcon » est un candidat idéal)

Mode graphique et interactif de contrôle des assauts

Vous pourriez faire une IHM pour contrôler les vaisseaux de votre Alliance Rebelle ... sans rien changer au moteur du jeu.

Mode distant client / serveur

Vous pourriez faire une Etoile Noire résidant sur une machine (serveur) et une Planète Endor sur une autre machine qui lancerait le premier assaut à partir... sans rien changer au moteur du jeu mais, il est vrai, avec un peu plus de sueur.