Programmation Orientée Objet
IG2I, LE2 – 2018-2019

TP10 – Synthèse

Diego Cattaruzza, Philippe Kubiak, Maxime Ogier

Contexte et objectif du TP

Dans ce TP nous allons développer un outil informatique qui permet de gérer le chargement des navires porte-conteneurs. Il faut placer les conteneurs de telle sorte que le centre de gravité des conteneurs soit au même endroit que le centre de gravité du navire.

Ce TP propose de faire une synthèse des notions qui ont été abordées dans le cours de POO2.

Ce TP permet d'illustrer les notions suivantes :

- création d'un modèle objet ;
- utilisation des piles : la classe **ArrayDeque** ;
- visualisation avec une interface graphique ;
- un peu d'algorithmique.

Bonnes pratiques à adopter

Pour réaliser ce TP, il vous est très fortement recommandé d'utiliser la Javadoc pour comprendre le fonctionnement des méthodes que vous appelez. Par ailleurs, il est aussi important que votre propre code soit commenté au format Javadoc (commentaire commençant par "`/**`" au-dessus des définitions des attributs et méthodes). Ceci vous permet (1) de spécifier rigoureusement le comportement d'une méthode au moment où vous l'implémentez, (2) de savoir ce que fait une méthode sans avoir besoin de regarder son code (qui d'ailleurs peut ne pas être accessible), et donc (3) de faciliter le partage et la réutilisabilité du code.

1 Création du modèle objet

On cherche à réaliser un programme informatique pour gérer le chargement d'un navire porte-conteneurs, comme présenté dans la Figure 1. Un tel navire comporte $n \times m$ emplacements disposés selon une grille rectangulaire de taille $n \times m$ (n lignes et m colonnes). Les conteneurs sont empilés les uns sur les autres à chaque emplacement. Chaque emplacement peut être associé à un empilement (potentiellement vide) de conteneurs. De plus il ne doit pas y avoir plus de $nbMaxC$ conteneurs dans un empilement. Un conteneur est loué par une entreprise pour un trajet et pèse un certain poids selon la marchandise placée à l'intérieur.

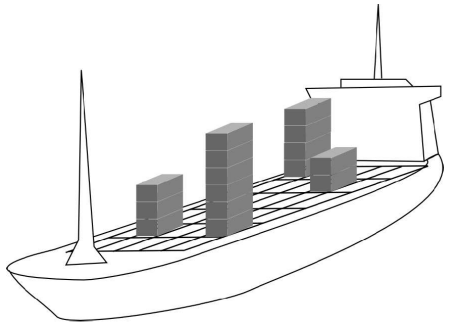

Figure 1: Un porte-conteneurs.

Le problème du chargement du navire est d'équilibrer la charge. Lorsque le chargement est bon, le centre de gravité de la grille de piles est proche du centre de la grille. Le modèle informatique s'appuie sur trois classes que vous développerez : **PorteConteneurs**, **Empilement** et **Conteneur**. Dans ce modèle, un conteneur est simplement représenté par le nom du client qui le loue et son poids (un entier). Un empilement de conteneurs est vu comme une collection ordonnée de conteneurs (le dernier élément de cette collection est le conteneur situé en haut de l'empilement) et un porte-conteneurs est vu comme une matrice d'empilements. Le centre d'un empilement p_{ij} situé à la ligne i et à la colonne j de la grille est le point de coordonnées $(100 \times j + 50,\ 100 \times i + 50)$. Remarquez bien dans le calcul des coordonnées que j est utilisé pour le calcul de l'abscisse et i pour le calcul de l'ordonnée.

On obtient les coordonnées (x_G, y_G) du centre de gravité du navire de la façon suivante :

$$x_G = \frac{\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} ((\text{poids de } p_{ij}) \times (\text{abscisse du centre de } p_{ij}))}{\text{poids total du chargement}}$$

$$y_G = \frac{\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} ((\text{poids de } p_{ij}) \times (\text{ordonnée du centre de } p_{ij}))}{\text{poids total du chargement}}$$

Une illustration graphique est proposée dans la Figure 2.

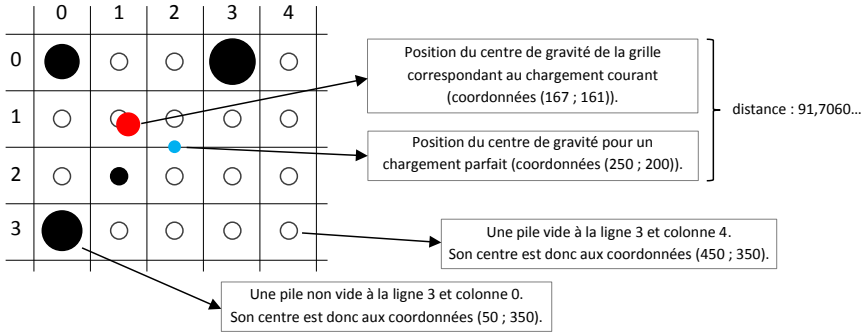

Figure 2: Vue du dessus d'une grille d'empilements ($n = 4$ et $m = 5$). Un empilement non vide est un disque noir de diamètre proportionnel à son poids et situé au centre de sa case. Un empilement vide est schématisé par un cercle.

1.1 La classe Conteneur

Un conteneur est défini par un numéro d'identification unique, le nom de son client, et son poids en kilos (un entier positif).

Question 1. Créez un nouveau projet nommé **Chargement**. Copiez les paquetages de votre TP Paint dans ce projet. Ajoutez un paquetage **porteconteneurs**, dans lequel vous ajoutez une classe **Conteneur**. Ajoutez dans cette classe les attributs nécessaires, 3 constructeurs (par défaut, par données et par copie), et les méthodes que vous jugez nécessaires. Redéfinissez la méthode **toString**. Ajoutez une méthode principale afin de vérifier que tout fonctionne correctement avec le code suivant :

```
1  Conteneur c1 = new Conteneur();
2  Conteneur c2 = new Conteneur("IG2I", 400);
3  Conteneur c3 = new Conteneur(null, 200);
4  Conteneur c4 = new Conteneur("IG2I", -20);
5  Conteneur c5 = new Conteneur(c4);
6  Conteneur c6 = new Conteneur(null);
7  System.out.println(c1);
8  System.out.println(c2);
9  System.out.println(c3);
10 System.out.println(c4);
11 System.out.println(c5);
12 System.out.println(c6);
```

1.2 La classe Empilement

Un empilement représente une collection ordonnée de conteneurs. Il est aussi caractérisé par un nombre maximal de conteneurs qui peuvent être empilés, ainsi qu'un point qui représente les coordonnées de son centre dans la grille (voir les explications sur les coordonnées de ce point au début du sujet).

La particularité d'un empilement est que son fonctionnement est de type *LIFO* : le dernier conteneur ajouté à la pile sera le premier à sortir de la pile. Pour modéliser cette structure de données en Java, il est recommandé d'utiliser la classe générique **ArrayDeque<E>** où **E** représente le type des éléments de la pile. Cette classe implémente l'interface **Deque<E>** qui représente une file à double extrémités (qui peut donc être utilisée en *LIFO* ou *FIFO*). Quelques méthodes utiles de la classe **ArrayDeque<E>** pour réaliser les opérations de type *LIFO* sont :

- **int size()** : renvoie le nombre d'éléments dans la pile ;
- **boolean isEmpty()** : renvoie **true** s'il n'y a pas d'éléments dans la pile, **false** sinon ;
- **void push(E e)** : ajoute l'élément **e** au dessus de la pile ;
- **E peek()** : renvoie l'élément en haut de la pile (sans le retirer de la pile) ;
- **E pop()** : renvoie l'élément en haut de la pile et le retire de la pile ;
- **Iterator<E> iterator()** : renvoie un itérateur sur les éléments de la pile.

Question 2. Ajoutez dans le paquetage **porteconteneurs** une classe **Empilement**. Définissez ses attributs. Vous utiliserez obligatoirement une collection de type **Deque<Conteneur>** (avec une implémentation **ArrayDeque<Conteneur>**) pour modéliser la collection de conteneurs. Ajoutez un constructeur **Empilement(int nbMaxC, int i, int j)** où **nbMaxC** représente le nombre maximal de conteneurs qui peuvent être mis sur la pile, **i** et **j** représentent respectivement la ligne et la colonne de la grille sur laquelle est situé l'empilement. N'hésitez pas à réutiliser la classe **Point** du TP Paint.

Question 3. Ajoutez et implémentez dans la classe **Empilement** les méthodes suivantes :

- **public boolean ajouterConteneur(Conteneur c)** : ajoute le conteneur **c** sur le haut de la pile si cela est possible, renvoie un booléen indiquant si l'ajout a pu être effectué ou non ;
- **public int poids()** : retourne le poids total de la pile de conteneurs ;
- **public boolean deplacerConteneurVers(Empilement e)** : transfère le conteneur situé en haut de l'empilement courant dans l'empilement **e** si cela est possible, cette méthode renvoie un booléen indiquant si le transfert a été effectué ou non.

Testez dans la méthode principale de la classe que tout fonctionne correctement avec le code suivant :

```
Conteneur c1 = new Conteneur("IG2I", 10);
Conteneur c2 = new Conteneur("IG2I", 10);
Conteneur c3 = new Conteneur("IG2I", 15);
Conteneur c4 = new Conteneur("IG2I", 20);
Conteneur c5 = new Conteneur("IG2I", 20);
Empilement e1 = new Empilement(3, 0, 0);
Empilement e2 = new Empilement(3, 1, 0);
if (! e1.ajouterConteneur(null))
   System.out.println("null pas ajoute a e1");
if (! e1.ajouterConteneur(c1))
   System.out.println("c1 pas ajoute a e1");
if (! e1.ajouterConteneur(c2))
   System.out.println("c2 pas ajoute a e1");
if (! e1.ajouterConteneur(c3))
   System.out.println("c3 pas ajoute a e1");
if (! e1.ajouterConteneur(c4))
   System.out.println("c4 pas ajoute a e1");
System.out.println("poids e1 : "+e1.poids());
if (! e2.deplacerConteneurVers(e1))
   System.out.println("e2 -> e1 imp.");
if (! e2.ajouterConteneur(c4))
   System.out.println("c4 pas ajoute a e2");
if (! e2.ajouterConteneur(c5))
   System.out.println("c5 pas ajoute a e2");
if (! e1.deplacerConteneurVers(e2))
   System.out.println("e1 -> e2 imp.");
if (! e1.deplacerConteneurVers(e2))
   System.out.println("e1 -> e2 imp.");
System.out.println("poids e1 : "+e1.poids());
System.out.println("poids e2 : "+e2.poids());
```

1.3 La classe PorteConteneurs

Un porte-conteneurs peut être vu comme une matrice d'empilements (voir la Figure 1). Ainsi, chaque empilement est repéré selon la ligne et la colonne de cette matrice. Le porte-conteneurs est donc aussi caractérisé par le nombre de lignes et le nombre de colonnes dans la matrice.

Question 4. Ajoutez dans le paquetage **porteconteneurs** une classe **PorteConteneurs**. Définissez ses attributs. Ajoutez un constructeur **public PorteConteneurs(int n, int m, int nbMaxC)**, où **n** est le nombre de lignes et **m** le nombre de colonnes de la matrice qui caractérise le porte-conteneurs, et **nbMaxC** est le nombre maximum de conteneurs dans un empilement.

Question 5. Ajoutez dans la classe **PorteConteneurs** une méthode **public void charger(Conteneur c, int i, int j)** permettant de charger le conteneur **c** sur l'empilement situé à la ligne **i** et la colonne **j** si cela est possible. En cas de chargement impossible, la méthode affiche dans la console : "chargement impossible". Si les coordonnées passées en paramètre ne sont pas correctes, la méthode affiche dans la console : "coordonnées non valides". Testez le bon fonctionnement de cette méthode.

Question 6. Ajoutez dans la classe **PorteConteneurs** une méthode **public int poidsTotal()** retournant le poids total de l'ensemble des conteneurs sur le porte-conteneurs. Testez le bon fonctionnement de cette méthode.

Question 7. Ajoutez dans la classe **PorteConteneurs** une méthode **public Point centreGravite()** retournant le centre de gravité de la grille (la matrice) d'empilements. Reportez-vous à la description du sujet pour le calcul des coordonnées. Testez le bon fonctionnement de cette méthode avec le code suivant (et vérifiez que vous trouvez le point de coordonnées (167 ; 161)) :

```
PorteConteneur pcont = new PorteConteneur(4,5,2);
pcont.charger(new Conteneur("IG2I",10), 0, 0);
pcont.charger(new Conteneur("IG2I",10), 0, 0);
pcont.charger(new Conteneur("IG2I",15), 0, 3);
pcont.charger(new Conteneur("IG2I",15), 0, 3);
pcont.charger(new Conteneur("IG2I",15), 3, 0);
pcont.charger(new Conteneur("IG2I",10), 3, 0);
pcont.charger(new Conteneur("IG2I",10), 2, 1);
System.out.println("Centre de gravite : "+pcont.centreGravite());
```

Question 8. Ajoutez dans la classe **PorteConteneurs** une méthode **public double qualiteDuChargement()** retournant la distance euclidienne entre le centre de gravité correspondant à un équilibrage parfait de la grille et son centre de gravité réel (correspondant au chargement courant). Ainsi une distance de 0 indique un équilibrage parfait. Référez-vous à la Figure 2 pour un exemple. Testez le bon fonctionnement de cette méthode en rajoutant à la fin du test précédent la ligne suivante (et vérifiez que vous trouvez une valeur proche de 91,7060) :

```
System.out.println("Qualite chargement :
     "+pcont.qualiteDuChargement());
```

2 Visualisation du chargement

On souhaite à présent utiliser le travail qui a été réalisé dans le TP Paint afin de pouvoir visualiser en 2D le chargement des conteneurs sur le porte-conteneurs. Les empilements seront dessinés avec des disques (cercle plein) dont le rayon sera fonction du poids de l'empilement. Si le poids est nul, alors on dessinera simplement un cercle. Le centre de gravité et le centre de gravité idéal seront dessinés avec des disques de couleur. Un exemple est présenté dans la Figure 3.

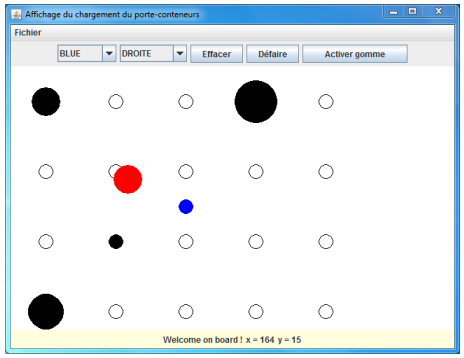

Figure 3: Visualisation de l'exemple présenté dans la Figure 2.

Question 9. Ajoutez dans la classe **Cercle** un constructeur **public Cercle(Point centre, int rayon, Color couleur)** qui initialise les attributs de la classe **Cercle**.

Question 10. Ajoutez dans le paquetage **modele** une classe **Disque** qui hérite de **Cercle**. Un disque est un cercle plein, il faudra donc redéfinir la méthode **seDessiner**. Pour remplir un cercle (ou plus généralement une forme ovale), on peut utiliser la méthode **fillOval** de la classe **Graphics**.

Question 11. Ajoutez dans la classe **Empilement** une méthode **public Forme getDessin()** qui renvoie :

- si le poids de l'empilement est strictement positif : un disque de couleur noire dont le centre est celui de l'empilement et le rayon est égal au poids de l'empilement ;
- si le poids de l'empilement est nul : un cercle de couleur noire dont le centre est celui de l'empilement et le rayon est égal à 10.

Question 12. Ajoutez dans la classe **PorteConteneurs** une méthode **public Collection<Forme> getDessin()** qui renvoie une collection (à vous d'en déterminer l'implémentation) contenant toutes les formes pour dessiner le chargement du porte-conteneurs. N'oubliez pas d'y inclure un disque bleu de rayon 10 pour le centre de gravité idéal, ainsi qu'un disque rouge de rayon 20 pour le centre de gravité du chargement.

Question 13. Ajoutez dans la classe **Fenetre** une méthode **public void dessinerPorteConteneurs (PorteConteneurs p)** qui permet de réaliser le dessin en 2D du chargement du porte-conteneurs dans la zone graphique de la fenêtre. Testez le bon fonctionnement de la visualisation dans la méthode principale de la classe **PorteConteneurs** en rajoutant au test de la Question 7 les lignes suivantes :

```
1  Fenetre f = new Fenetre();
2  f.dessinerPorteConteneurs(pcont);
```

3 Amélioration du chargement

Dans cette section, l'idée est de déplacer les conteneurs de telle sorte que le centre de gravité soit le plus proche possible du centre de gravité idéal. Nous éviterons ainsi que le navire chavire !

Question 14. Ajoutez dans la classe **PorteConteneurs** une méthode **public double evaluerQualiteDeplacement(Empilement p1, Empilement p2)** retournant la qualité du chargement si on transférait le conteneur situé en haut de l'empilement **p1** dans **p2**. Testez le bon fonctionnement de cette méthode.

Question 15. Ajoutez dans la classe **PorteConteneurs** une méthode **public void amelioreEquilibrage()** qui déplace les conteneurs d'un empilement à un autre afin d'améliorer la qualité de l'équilibrage. On s'arrêtera lorsque aucun déplacement ne permet d'améliorer l'équilibrage. Testez sur l'exemple proposé à la Question 13.

Question 16. La méthode précédente donne de bons résultats. Mais on peut certainement faire mieux ! Par exemple, si aucun déplacement ne permet d'améliorer l'équilibrage, alors il est possible qu'une combinaison de deux mouvements permette d'améliorer l'équilibrage. Proposez une méthode **public void amelioreEquilibrage2()** qui permette d'obtenir de meilleurs résultats que la méthode précédente. Vous pouvez aussi raisonner autrement que par des mouvements d'un empilement vers un autre.