Université de Nice - Sophia Antipolis
Faculté des Sciences

DEUG MIAS MP1

Programmation 2000-01

9. LA CLASSE VECTOR. LES POLYGONES

A. Résumé des épisodes précédents

Dans la feuille 8, vous avez appris à construire une **interface graphique** [GUI] minimale mais assez complète, comprenant des textes à afficher [classe `Label`], des textes éditables[1] [classe `TextField`], des zones à dessiner [sous-classe de `Canvas`], des boutons générant des actions [classe `Button`]. Tous ces « composants graphiques » étaient placés à l'intérieur d'une fenêtre [classe `MP1Frame`, sous-classe de `Frame`]. A certains de ces composants étaient accrochés des « écouteurs » [*listeners*] dont le rôle était d'être tenus au courant des évènements survenant dans le composant et de réagir en conséquence.

B. Et alors, que demander de plus ?

Comme tout bon scientifique, un programmeur risque d'être exigeant : faire des animations, jouer des morceaux MP3, interfacer des appareils de mesure physique, etc. Mais tout cela est plus difficile bien que parfaitement réalisable en Java. Nous nous intéresserons seulement à la petite bestiole située à droite de votre clavier : **la souris** ! Nous souhaitons pouvoir choisir des points dans un *canvas* en cliquant avec la souris [*mouse*]. Un ensemble de points [polygone] étant ainsi défini, il sera possible de faire de la « géométrie algorithmique » [*computational geometry*] qui est un secteur très actif de la programmation, apprécié des mathématiciens qui voient leur géométrie pleinement opérationnelle, des physiciens qui peuvent simuler des processus dynamiques, et des… programmeurs de jeux pour qui Lara Croft n'est pas autre chose qu'un grand nombre de polygones en mouvement !

C. Préliminaire : la classe `Vector`

Dans quelle structure de donnée stocker les points que l'utilisateur choisira ? Ne connaissant pas à l'avance le nombre de ces points, il est difficile d'opter pour un tableau, sauf à le surdimensionner par prévoyance et à se compliquer la vie en gaspillant de la place-mémoire au passage. Il se trouve que Java dispose de nombreuses structures de données toutes prêtes dans son API, pour représenter des *collections* d'objets. En particulier, dans le package `java.util` se trouve la classe `Vector`. Un « vecteur » comme on dit en Java, ressemble à un tableau sauf que sa taille peut varier : on peut lui rajouter ou lui enlever des éléments, opérations interdites dans le cadre rigide d'un tableau. En contrepartie, il n'admet pas de syntaxe avec les [ ] et se manipule comme une classe usuelle, avec des constructeurs et des méthodes.

> *The **Vector** class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a **Vector** can grow or shrink as needed to accommodate adding and removing items after the **Vector** has been created.*

Exercice 9.1 Amenez votre planche de surf sur la plage `Vector` de l'API, et complétez les cadres ci-dessous :

`import` ____

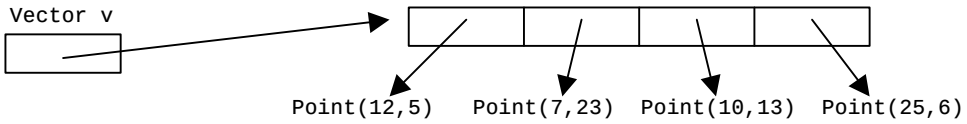
Constructeurs : ____

[1] Rappelons qu'un *label* remplace un *Console.print(…)* et qu'un *textfield* remplace un *Console.read(…).*

Méthodes d'instance :

```
.....  size ( )
.....  elementAt (.....)
.....  setElementAt (.....)
.....  addElement (.....)
.....  removeElementAt (.....)
.....  insertElementAt (.....)
.....  toString ( )
```

***ATTENTION** : contrairement à un tableau usuel, il n'est pas possible de placer des `int` et plus généralement des données d'un type primitif à l'intérieur d'un vecteur. Il ne peut y avoir que des **objets**, instances de classes ! Un vecteur ne contient donc que des **références** vers des objets : des `Point`, des `Integer`, des Double, des `Employé`, etc.*



Une conséquence désagréable est qu'il faudra penser à opérer une conversion de type en récupérant un élément de ce vecteur, ici par exemple : `Point p = (Point)(v.elementAt(2)) ;`

Exercice 9.2 En utilisant `Console.readDouble(…)`, demandez à l'utilisateur d'entrer plusieurs notes sur 20, que vous stockerez dans un vecteur. La lecture termine dès qu'une note $\notin$ [0,20] est entrée, qui n'est pas mise dans le vecteur. Vous afficherez à la `Console` le nombre de notes entrées ainsi que leur moyenne. Le vecteur se laisse-t-il afficher ?…

D. Un polygone est un vecteur de points !…

L'API nous offre une classe `Polygon`, mais un polygone est représenté par deux tableaux de nombres, un pour les abscisses et un autre pour les ordonnées, ce qui rend son utilisation inconfortable. Nous allons donc définir une classe `Polygone` où un polygone sera représenté par un *vecteur* d'objets de type `Point` [la classe `Point` étant celle de l'API].

Exercice 9.3 Complétez le fichier `Polygone.java` que vous trouverez sur la page Web du cours 9…

Exercice 9.4 Dans une classe Test`Polygone`, programmez une procédure :

```
static void pointMinY (Polygone v)
```

cherchant le point d'ordonnée minimale d'un polygone `v` [c'est-à-dire le point le plus haut pour un *canvas* Java]. Au cas où il y en aurait plusieurs, on calculera le point d'abscisse minimale. Une fois le point-solution trouvé, on le permutera avec le 1er sommet, de sorte qu'il soit en tête du vecteur.

Exercice 9.5 Toujours dans la classe Test`Polygone`, programmez une procédure `triSel(Polygone v)` qui permute les sommets d'un polygone de sorte que le résultat soit ordonné par abscisses croissantes [en cas d'abscisses égales, on ordonnera par ordonnées croissantes]. *Vous avez étudié l'algorithme de « tri par sélection » au 1er semestre…*

E. Compléments : préparation de la feuille 10

Soit `v` un polygone dont le 1er sommet p_0 est le point le plus haut [cf. l'exercice 9.4]. Modifiez l'algorithme précédent `triSel(…)` pour que les points restants $p_1,\dots,p_{n-1}$ deviennent ordonnés par angles polaires croissants par-rapport au premier point p_0. Voir les détails mathématico-géométriques sur la page Web du cours…

Afficher un polygone aléatoire de 20 sommets dans un canvas. Dessiner chaque sommet par un tout petit disque rouge. Relier par un segment bleu les sommets consécutifs p_ip_{i+1} et par un trait noir le point p_0 à chaque autre sommet p_i…

```
//  Polygone.java
//  Un polygone est un objet ayant en variable privée un vecteur de Point. Cette classe
//  spécialisée sur les vecteurs de points évitera de sempiternelles conversions de type...

import ...;
import ...;

class Polygone
{      private Vector v;
```

```
        // se contente de créer le vecteur v
        Polygone()
        {       v = new ... ;
        }

        // crée un vecteur de n éléments. Chaque élément sera un Point dont les
        // coordonnées seront des entiers aléatoires de [10..max-10] pour ne pas
        // trop s'approcher des bords !
        Polygone(int n, int max)
        {       v = new ... ;
                for(... ; ... ; ...)
                  v.add(...) ;
        }

        // on conserve le nom size() qui est parlant pour le nombre de sommets
        int size()
        {       return ... ;
        }

        // on remplace elementAt par un pointAt plus précis :
        Point pointAt(int i)
        {       return ... ;
        }

        // idem avec setElementAt :
        void setPointAt(Point p, int i)
        {       ... ;
        }

        // idem
        void removePointAt(int i)
        {       ... ;
        }

        // le toString de la classe Point affiche "Point(x,y)" que l'on remplace
        // par un <x,y> plus concis, de sorte à afficher un polygone sous la forme
        // par exemple :  "Polygone[ <23,10> <12,55> <44,40> <27,52> ]"
        ... toString()
        {       String s = "Polygone[ ";
                for (... ; ... ; ...)
                {  Point pi = ...;   // le Point numéro i du vecteur v
                   s = s + ... ;
                }
                return s + "]";
        }
}
```