Epreuve de Programmation Orientée Objet
Licence Informatique & ILOG 1 *

26 janvier 2004

Durée : 2 heures.
Tous les documents sont autorisés, sauf les livres.

1 Ce qui est évalué

1.1 Pour l'exercice 2.1

Votre capacité à :
- effectuer des choix corrects pour les arguments et le type de retour des méthodes ;
- effectuer des choix corrects pour la visibilité des champs et des méthodes ;
- déterminer et définir les assertions d'une classe en distinguant les rôles des différents types d'assertion (i.e. une assertion doit elle être spécifiée sous forme d'une précondition, d'une postcondition ou d'un invariant ?) ;
- donner l'implémentation d'une classe en conformité avec son contrat.
- utiliser correctement le langage JAVA pour implémenter une classe.

1.2 Pour l'exercice 2.2

Votre capacité à :
- déterminer et définir les assertions d'une classe en distinguant les rôles des différents types d'assertion (i.e. une assertion doit elle être spécifiée sous forme d'une précondition, d'une postcondition ou d'un invariant ?) ;
- utiliser correctement le langage JAVA pour implémenter une classe en utilisant l'héritage ;

2 Présentation du problème

On souhaite définir en JAVA une classe `ListePoints` représentant une liste ordonnée de points. Cette classe aurait les caractéristiques suivantes :
- la liste de points peut contenir un nombre quelconque de points (y compris 0) ;
- les ajouts et suppressions de points sont possibles, mais uniquement en fin de liste ;
- les modifications de points appartenant à la liste sont interdites, toute modification des coordonnées d'un point de la liste doit donc être rendu impossible, de même que le remplacement d'un point par un autre ;
- accès direct possible à un point de la liste par son numéro d'ordre dans la liste.

Dans un second temps, on souhaite définir en JAVA une classe `ListePointsGeometrie` représentant une liste ordonnée de points pour laquelle on pourrait tester que certaines propriétés géométriques sont satisfaites. Cette classe serait donc définie comme sous-classe de `ListePoints`.

Pour une instance de la classe `ListePointsGeometrie`, les notions de *côté* et de *diagonale* sont définies de la manière suivante :

côté un côté est un segment dont les extrémités sont deux points consécutifs dans la liste. Le premier et le dernier point de la liste forment un côté ;

diagonale une diagonale est un segment dont les extrémités sont deux points entre lesquels un point est intercalé dans la liste : par exemple, pour une liste de 5 points, les diagonales sont les segments formés des points de numéro 0 et 2 ; 1 et 3 ; 2 et 4 ; 3 et 0 (pour 4 points, cela correspond à la notion habituelle de diagonale, si on considère que ces 4 points définissent un polygone à 4 côtés).

*M. Champesme – Département d'Informatique – Institut Galilée

La classe `ListePointsGeometrie` devra contenir des méthodes pour calculer la longueur d'un côté ou la longueur d'une diagonale en fonction d'un numéro de sommet caractérisant ce côté ou cette diagonale. Par exemple, pour une liste de 5 points, le numéro de sommet 3 caractérise le côté [point 3, point 4] et la diagonale [point 3, point 0].

Les propriétés géométriques que l'on souhaite pouvoir tester sont :
- les points sont-ils deux à deux distincts ?
- tous les côtés sont-ils de longueur non nulle ?
- les diagonales sont-elles toutes de même longueur ?
- la liste de point constitue t'elle un rectangle ?

Exercice 2.1 Interface, contrat et implémentation partielle de la classe `ListePoints`.

1. Donnez les entêtes de toutes les méthodes de la classe `ListePoints` qu'il vous semblera utile de définir en fonction de la description ci-dessus. Vous devrez définir les trois constructeurs suivant :
   - un constructeur initialisant une liste vide ;
   - un constructeur initialisant une liste à partir d'un tableau de points ;
   - un constructeur initialisant une liste à partir d'une autre instance de `ListePoints` et du numéro d'ordre d'un élément de cette liste passée en paramètre. Ce numéro d'ordre indique que la nouvelle liste de points doit avoir pour premier élément, l'élément portant ce numéro dans la liste passée en paramètre : par exemple, si la liste passée en paramètre est composée des points (p0, p1, p2, p3, p4, p5) et que le numéro d'ordre passé en paramètre est 3, la nouvelle liste sera (p3, p4, p5, p0, p1, p2).
2. Ajoutez les assertions (y compris l'invariant) définissant le contrat de cette classe, sans oublier d'indiquer quelles méthodes sont "pures".
3. Définissez les attributs de la classe et donnez l'implémentation des constructeurs, de la méthode permettant l'accès direct à un point de la liste par son numéro, ainsi que des méthodes permettant l'ajout et la suppression de points. Votre implémentation devra être conforme au contrat exprimé à la question précédente. Vous devrez impérativement utiliser un tableau pour mémoriser les points de la liste.

Vous pourrez utiliserer les méthodes de la classe `Point` données en annexe A.

Exercice 2.2 Interface, contrat et implémentation partielle de la classe `ListePointsGeometrie`.

1. Donnez les entêtes de toutes les méthodes et constructeurs de la classe `ListePointsGeometrie` qu'il vous semblera utile de définir en fonction de la description ci-dessus. Cette classe doit être définie comme sous-classe de la classe `ListePoints` et doit permettre la création d'instance dans les mêmes conditions que la classe `ListePoints` (cf. définition des constructeurs de `ListePoints`).
2. Ajoutez les assertions (y compris l'invariant) définissant le contrat de cette classe, sans oublier d'indiquer quelles méthodes sont "pures".
3. Définissez les attributs de la classe et donnez l'implémentation du ou des constructeurs de la classe.

Exercice 2.3 Définition d'une classe `Rectangle`.

Utilisez les classes `ListePoints` et/ou `ListePointsGeometrie` pour donner le contrat et implémenter une classe `Rectangle` possédant les caractéristiques suivantes :
- possibilité de créer une instance à partir d'une liste de points ;
- accès direct possible à un point de la liste par son numéro d'ordre dans la liste ;
- possibilité d'obtenir une liste de points correspondant aux points du rectangle (i.e. points de même coordonnés et dans le même ordre) ;
- calcul de la largeur et de la longueur du rectangle.

A Classe Point

`public int x`

The x coordinate. If no x coordinate is set it will default to 0.

`public int y`

The y coordinate. If no y coordinate is set it will default to 0.

`public Point()`

Constructs and initializes a point at the origin (0, 0) of the coordinate space.

`public Point(int x, int y)`

Constructs and initializes a point at the specified (x, y) location in the coordinate space.

Paramètres :

`x` - the x coordinate.

`y` - the y coordinate.

`public Point(Point p)`

Constructs and initializes a point with the same location as the specified `Point` object.

Paramètres :

`p` - a point.

`public double getX()`

Returns the X coordinate of the point in double precision.

Returns : the X coordinate of the point in double precision

`public double getY()`

Returns the Y coordinate of the point in double precision.

Returns : the Y coordinate of the point in double precision

`public double distance(Point pt)`

Returns the distance from this `Point` to a specified `Point`.

Paramètres :

`pt` - the specified `Point`

Returns : the distance between this `Point` and the specified `Point`.

`public boolean equals(Object obj)`

Determines whether an instance of `Point` is equal to this point. Two instances of `Point` are equal if the values of their `x` and `y` member fields, representing their position in the coordinate space, are the same.

Paramètres :

`obj` - an object to be compared with this point

Returns : `true` if the object to be compared is an instance of `Point` and has the same values; `false` otherwise

---

`public Object clone()`

Creates a new object of the same class and with the same contents as this object.

Returns : a clone of this instance.
