Programmation & Algorithmique 2

TP - Séance 2

3 mars 2022

Matière visée : Introduction au concept de classe et d'objet, récursivité.

1 Notion d'objet

1.1 Point

Définissez une classe **Point** (fichier Point.java) qui permette d'instancier des points dans $\mathbb{R}^2$. Le constructeur de cette classe initialisera les deux *variables d'instances* x et y qui caractérisent complètement un point (voir séance précédente). Définissez également un constructeur par défaut qui crée le point d'origine $(0, 0)$. Créez des *accesseurs* pour chacun des attributs.

1.2 Droite

Définissez une classe **Droite** (fichier Droite.java) qui permette d'instancier des droites dans $\mathbb{R}^2$. Le constructeur (qui prend trois `double` en paramètres) de cette classe initialisera les trois variables d'instances a, b et c qui caractérisent complètement une droite (voir séance précédente).

Définissez ensuite les méthodes suivantes :

1. `isHorizontal()` (méthode d'instance)
   Entrée: /
   Sortie: `true` si la droite sur laquelle la méthode est invoquée est horizontale, `false` sinon.
2. `intersection( )` (méthode d'instance)
   Entrée: un objet Droite `droite`
   Sortie: un objet Point qui est l'intersection entre la droite sur laquelle la méthode est invoquée et la droite `droite`, si elle existe. S'il n'y a pas d'intersection ou si les droites sont confondues, retouner `null`.
3. `create( )` (méthode de classe)
   Entrée: deux Points `p1` et `p2`
   Sortie: un objet Droite qui représente la droite passant par `p1` et `p2`
   Remarque : Cette méthode est static, car elle ne dépend pas d'un objet Droite particulier, mais il est clair qu'elle concerne le concept de "droite".

1.3 Testeur

Créez une classe **DroiteTest** (fichier DroiteTest.java) qui permet de tester vos méthodes de la classe `Droite`. Pour ce faire, vous devez (dans la méthode `main`) :

1. Créer les points $p1 = (1, 1)$, $p2 = (5, 3)$, $p3 = (0, 4)$ et $p4 = (4, 6)$;
2. Créer la droite $d1$ d'équation $y = 2$;
3. Créer la droite $d2$ passant par les points $p1$ et $p2$ (méthode `create`) ;
4. Créer la droite $d3$ passant par les points $p3$ et $p4$ (méthode `create`) ;

5. Indiquer quelles droites sont horizontales (méthode `isHorizontal`) parmi les deux droites que vous venez de créer ;
6. Afficher le point d'intersection entre $d1$ et $d2$ s'il existe. Afficher "les droites d1 et d2 sont parallèles" si le point d'intersection n'existe pas ou si les droites sont confondues.
7. Répéter le point 6 avec les droites $d1$ et $d3$ et les droites $d2$ et $d3$.

2 Récursivité

Créez une classe `Recursion` contenant des méthodes de classes récursives permettant de résoudre les problèmes suivants :

1. La factorielle d'un nombre entier positif `n` (notée `n!`) se calcule par la formule suivante : $\texttt{n}! = \prod_{k=1}^{n} k$ si $n >= 1$ et on définit $0! = 1$. On souhaite pouvoir calculer `n!` pour un $\texttt{n} \in \mathbb{N}$ donné.
   Pour vous permettre de vérifier votre algorithme, voici les valeurs des factorielles des nombres entre 0 et 10 : 1 1 2 6 24 120 720 5040 40320 362880 3628800
2. La suite de Fibonacci est une suite de nombres définie au départ pour représenter l'évolution d'une population de lapins. Le n^{e} terme de la suite (`Fibo(n)`) est défini comme la somme des $n-1^{\text{e}}$ et $n-2^{\text{e}}$ termes. Par convention, on définit le premier (`Fibo(0)`) et deuxième (`Fibo(1)`) terme comme valant 1 tous les deux. Le problème consiste à calculer le n^{e} terme de la suite de Fibonacci pour un `n` donné. Voici la suite de Fibonacci jusqu'au 10^{e} élément (`Fibo(9)`) : 1 1 2 3 5 8 13 21 34 55
3. Le triangle de Pascal est une représentation des coefficients binomiaux. *i.e.,* chaque case (n, p) dans le triangle contient le nombre de parties de p éléments dans un ensemble de n éléments (noté $\binom{n}{p}$ ou C_p^n). Si n ou p est négatif, on considère qu'il n'est pas possible de prendre p éléments parmi n et donc, $\binom{n}{p}$ vaut 0. Pour les même raisons, si $p > n$, $\binom{n}{p}$ vaudra aussi 0. On peut facilement déduire que $\binom{n}{0} = \binom{n}{n} = 1$ en se basant sur cette définition ensembliste. Il vous est demandé d'écrire une fonction récursive permettant de calculer la case (n, p) du triangle de Pascal pour n et $p \in \mathbb{Z}$ donnés. Ci-dessous se trouve une partie du triangle de Pascal. Les cases vides ou non représentées ont une valeur de zéro.

n \ p	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1

2.1 Les tours de Hanoï

Le problème des tours de Hanoï est constitué de `n` disques de diamètre allant de 1 à n et percés en leur centre. Ces disques sont placés sur trois tiges (appelées tours) alignées de gauche à droite comme illustré dans l'image ci-dessous qui représente la solution pour $n = 3$. Les règles du jeu interdisent de placer un disque i sur un disque j de diamètre plus grand. Le but du jeu est de déplacer tous les disques de la première tour à la troisième en ne déplaçant qu'un disque à la fois et en respectant la règle précédente. Il vous est demandé d'écrire une fonction qui resoud ce problème pour un $\texttt{n} \in \mathbb{N}\backslash\{0\}$ donné.

Pour y parvenir, réfléchissez au fonctionnement de la récursivité. *i.e.,* comment résoudre le problème à n disques si on peut résoudre le problème à $n-1$ disques ?

Afin de vous aider, vous trouverez sur moodle une archive contenant trois classes permettant l'affichage des déplacements effectués par votre programme. Pour l'utiliser, il vous suffit d'implémenter votre solution dans la méthode `solve` de la classe `Tester`. Celle-ci prend en paramètre un objet HanoiState dont la javadoc est fournie dans l'archive, un nombre de disques et les indices des trois tours dans l'ordre suivant : la tour de départ, la tour d'arrivée et la tour centrale. Par exemple, au départ, la tour où se trouvent les disques a l'indice 0, la tour d'arrivée (celle à droite) a l'indice 2 et la tour centrale a l'indice 1. Cela permet de changer les rôles des tours.