TD n°2 – Les types en Java

L'objectif de ce TD est de prendre en main l'IDE Eclipse et de manipuler les types usuels de données et les tableaux en Java.

Utiliser l'IDE Eclipse

Lire le tutoriel disponible à l'adresse : http://www.eclipsetotale.com/articles/premierPas.html

Plus précisément :

- Sur une machine du département, ouvrir Eclipse Mars situé dans *C:/eclipsemars/*
- Choisir comme *workspace* (espace de travail), le répertoire M213 créé sur votre espace personnel
- Créer un nouveau projet Java (*Java Project*) intitulé TD2
- Dans ce projet (voir le « *PackageExplorer* » sur la gauche de l'écran d'Eclipse), cliquer avec le bouton droit de la souris sur le répertoire *src* et choisir « *New > package* ». **Créer 1 package par exercice du TD**. *Le nom des packages commence par une lettre minuscule.*
- Une fois le package créé, clic droit sur le package pour ajouter une classe (New > *Class*)

Ensemble d'entiers

Le but de cet exercice est de réaliser une classe `EnsembleEntierBorne` permettant de manipuler des ensembles d'entiers de taille bornée.

1) **Attributs :** Chaque instance de `EnsembleEntierBorne` est définie par 2 attributs :
   a. une constante entière `MAXIMUM` qui ne peut être initialisée qu'une seule fois, et
   b. un tableau de booléens où chaque case indique si oui ou non l'entier (correspondant à l'indice) est contenu dans l'ensemble.
2) **Constructeurs :** l'unique constructeur prend en paramètre un entier. L'ensemble ne pourra contenir que des entiers compris entre 0 (inclus) et cet entier (inclus). *Le tableau a donc une case de plus que cette valeur.*
3) **Méthodes :**
   a. `add(…)` Cette procédure prend un entier en paramètre et le rajoute à l'ensemble (si l'entier est déjà dans l'ensemble ou ne peut être contenu dans l'ensemble, la procédure est sans effet).
   b. `remove(…)` Comme pour la procédure `add`, mais cette fois-ci il s'agit d'enlever un élément.
   c. `doesContain(…)` Cette fonction prend un entier en paramètre et renvoie vrai si et seulement si l'entier est dans l'ensemble.
   d. `toString()` Affiche l'ensemble sous forme `{2,4,6,7,99}` (on peut éventuellement laisser une virgule en trop dans une première version, sinon utiliser la fonction `substring` de la classe `String` afin de supprimer la dernière virgule inutile).
   e. `intersect(…)` Cette fonction prend un `EnsembleEntierBorne` en paramètre et renvoie l'ensemble correspondant à l'intersection entre l'ensemble courant et celui entré en paramètre.
4) **Main :** Dans une classe `TestEnsemble`, instancier des objets `EnsembleEntierBorne` et tester les méthodes développées.

5) En utilisant la description des ensembles d'entiers bornés de l'exercice précédent (constructeur + méthodes), ajouter dans la fonction `main` le code Java correspondant à l'algorithme suivant :

variables
maxim : entier
premiers : ensemble d'entiers entre 0 et maxim
i, j : 0..maxim

début
pour i **de** 2 **à** maxim **faire**
inclure i dans premiers
finpour
pour i **de** 2 **à** maxim **faire**
si i *appartient* à premiers **alors**
pour j **de** 2i **à** maxim **par pas de** i **faire**
enlever j de premiers
finpour
finsi
finpour
afficher premiers
fin

Que fait cet algorithme ? Regarder ce qu'il se passe sur différents exemples.

6) Placer le code de cet algorithme dans une fonction associée à la classe `TestEnsemble`. Modifier le `main` pour appeler cette fonction et la tester sur différents exemples.

Classe Segment

Après avoir inclus la classe `Point` dans le package *segment*, ouvrir le code source et l'analyser. Dans cette classe, définir la méthode `public boolean equals(Point p)` qui retourne `true` si le point courant et celui passé en paramètre sont confondus, `false` sinon.
Rappel : *On dit que 2 points A (xA, yA) et B (xB, yB) sont confondus si leurs coordonnées sont « proches » :* $|xA - xB| \leq \varepsilon$ *et* $|yA - yB| \leq \varepsilon$.

Ecrire ensuite la classe `Segment` dans laquelle un segment est **composé** de 2 points **distincts**, que l'on nommera *origine* et *extrémité*.

- Dans un premier temps, on se limitera à la déclaration des attributs, la définition des constructeurs, des accesseurs en lecture, et de la méthode `toString`.
- Détailler en Java un programme de test de la classe `Segment` (une classe `TestSegment`). Compiler et exécuter avec succès.
- Ajouter dans la classe `Segment` la méthode `longueur` qui retournera la longueur de l'objet courant. (Modifier `TestSegment` pour faire appel à la nouvelle méthode)
- Introduire dans la classe `Segment` deux nouvelles méthodes `projX` et `projY` permettant respectivement de calculer le projeté d'un segment support sur l'axe des abscisses et sur l'axe des ordonnées. (Mettre à jour `TestSegment`).

Exercice complémentaire : Carré magique (tableaux à 2 dimensions)

Un *carré magique* est une matrice carrée n x n contenant tous les entiers entre 1 et n^2, et telle que la somme des entiers de chaque ligne, de chaque colonne, et des deux diagonales sont identiques.

Dans ce TD, nous allons créer des carrés magiques de taille impaire (n impair).

Exemples :

Carrés magiques d'ordre 3 et 5

4	9	2
3	5	7
8	1	6

11	24	7	20	3
4	12	25	8	16
17	5	13	21	9
10	18	1	14	22
23	6	19	2	15

L'algorithme pour générer un carré magique de taille n impaire est le suivant :

- On place le 1 dans la case située une ligne en dessous de la case centrale (qui existe toujours vu que n est impair).
- Lorsque l'on a placé l'entier x dans la case (i, j), on place l'entier $x + 1$ dans la case $(i + 1, j + 1)$, mais :
  - Si un indice devient égal à n, il revient à 0
  - Tant que l'on tombe sur une case déjà occupée, par exemple (l, k), on essaie de placer le nombre en $(l + 1, k - 1)$ (attention encore aux bornes de l et k !)

Notez que cet algorithme suppose que le carré est représenté par un tableau Java à 2 dimensions et donc que les indices commencent à 0.