M2103
TP n° 3 – La classe *Monôme* en Java

Un monôme est représenté en mathématique sous la forme : $\mathbf{a_i\ x^i}$ où x représente une variable, a_i son coefficient (réel) et i son exposant (entier).

a) Opérations

unMonôme :	Réel x Entier → Monôme
coefficient :	Monôme → Réel
exposant :	Monôme → Entier
somme :	Monôme x Monôme → Monôme
produit :	Monôme x Monôme → Monôme
dérivée :	Monôme → Monôme

b) Pré-conditions

Pour m1 et m2 de type *Monôme* :
somme (m1, m2) est défini ssi exposant (m1) = exposant (m2)

c) Propriétés

Pour m, m1 et m2 de type *Monôme*, c de type *Réel* et e de *Entier* :

(P1) coefficient (unMonôme (c, e)) = c

(P2) exposant (unMonôme (c, e)) = e

(P3) somme (m1, m2) =
unMônome (coefficient (m1) + coefficient (m2), exposant (m1))

(P4) produit (m1, m2) =
unMônome(coefficient (m1) * coefficient (m2),
exposant (m1) + exposant (m2))

(P5) dérivée (m) =
si exposant (m) = 0 alors unMonôme (0.0, 0)
sinon unMônome (coefficient (m) * exposant (m), exposant (m) -1)

Questions :

1) Aller dans la javadoc standard Java pour rechercher la classe *ArithmeticException*. Le lien se trouve sous Moodle : *Lien vers la documentation Java Standard Edition 7*. Quel est le nom du paquetage dont est issue cette classe ? De quelle classe hérite-t-elle ?

2) Traduire le TAD *Monôme* en une classe Java.

3) Rajouter dans la classe Java une méthode `toString()` produisant une version chaine d'un monôme sous la forme « *coefficient***xe***exposant* ». Par exemple, `10xe2` représente le monôme $\mathbf{10\ x^2}$

4) Rajouter dans la classe Java une méthode `estNul()` déterminant si un monôme est nul.

5) Générer la documentation au format javadoc de la classe *Monôme*.

6) Ecrire une application Java cliente de *Monôme* qui crée 2 monômes nuls et qui propose le menu ci-dessous. Celui-ci orientera sur le traitement correspondant en fonction du choix de l'utilisateur (utilisation d'une structure de contrôle *switch*).

```
Quel est votre choix :
    1- modifier le premier monôme
    2- modifier le deuxième monôme
    3- afficher le premier monôme
    4- afficher le deuxième monôme
    5- calculer la somme des 2 monômes
    6- calculer le produit des 2 monômes
    7- calculer la dérivée du premier monôme
    8- calculer la dérivée du deuxième monôme
    9- quitter l'application
```

7) Afin de réduire la taille de la méthode *main*, créer dans la classe de l'application deux méthodes de classe spécifiées ainsi :

Class TestMonomeRefactore

- java.lang.Object
  - TestMonomeRefactore

```
public class TestMonomeRefactore
extends java.lang.Object
```

- **Constructor Summary**

Constructors
Constructor and Description
TestMonomeRefactore()

- **Method Summary**

Methods	
Modifier and Type	**Method and Description**
static void	**aff□c□erMen□**() affiche le menu de l'application
static Mono□e	**□ect□reMonome**(java.util.Scanner entr□e) lit un monôme sur l'entrée standard (coefficient et exposant)
static void	**ma□n**(java.lang.String□□ args) propose un menu orientant sur différents traitements pour tester la classe Monome

Methods inherited from class java.lang.Object

clone□ e□uals□ □inali□e□ get□lass□ □as□□ode□ noti□□□ noti□□□ll□ toString□ □ait□ □ait□ □ait

Constructor Detail

TestMonomeRefactore

public □estMono□e□e□actore()

Method Detail

afficherMenu

public static void a□□ic□erMenu()

affiche le menu de l'application

lectureMonome

public static Mono□e lectureMono□e(java.util.Scanner entr□e)

lit un monôme sur l'entrée standard (coefficient et exposant)

Parameters:

entr□e - scanner associé au flot standard d'entrée

Returns:

monôme résultat

main

public static void □ain(java.lang.String□□ args)

propose un menu orientant sur différents traitements pour tester la classe Monome

Parameters:

args -