TP 1 : Récursivité

Objectif du TP

L'objectif de cette séance est de pratiquer la programmation récursive en Java.

Recommandations

Immédiatement après chaque séance de TD/TP, chaque étudiant ou groupe d'étudiants enverra un mail à l'enseignant responsable du groupe un compte rendu de l'activité réalisée pendant la séance au format PDF dans lequel les réponses aux questions auront été saisies. Il est fortement conseillé de faire des copies d'écran de vos programmes pour aller plus vite.

Le sujet du mail devra obligatoirement respecter le format suivant : **[S3T ou S3D ou S3A][M313][Gx][Seance_y] Nom1 / Nom2** ou *x* représente le numéro de groupe et *y* le numéro de la séance (par exemple : mailto:denis.pallez@unice.fr?subject=[S3A][M313][G1][Seance2]Turing). Le fichier PDF devra également être renommé de la façon suivante : **<NomEtudiant1_NomEtudiant2>_TP<N° du TP>_<date de la séance de TP en anglais>.pdf** (par ex : Pallez_TP1_20141306.pdf).

Vous ne pouvez pas faire plus de 2 séances avec le même binôme et vous ne pouvez pas faire plus de la moitié des séances seul. Tout manquement à ces règles pourra entrainer des retraits de points sur la note de contrôle continu.

Exercice 1 Vous avez dit « Pas besoin d'ordinateurs … ! »

Pour chacune des fonctions suivantes, **l'exécuter à la main sur un exemple (donc sans la programmer sur ordinateur)**, dire pour quelles valeurs du paramètre *n* elle se termine, et quand c'est le cas ce qu'elle fait ou calcule et de quelle type est la fonction récursive :

a)
```
public static int f(int n) {
    if (n == 0) return 1;
    else return f(n+1);
}
```

b)
```
public static int sommeBis(int n) {
    if (n == 0) return 0;
    else {
        int result = sommeBis(n-1);
        result += n;
        return result;
    }
}
```

c)
```
public static int g(int n) {
    if (n <= 1) return 1;
    else return 1 + g(n-2);
}
```

Exercice 2 Récurrence

Montrer par récurrence que $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1, \forall n \geq 0$.

Exercice 3 Ecriture de nombres

a) Ecrire une méthode récursive `descente(int n)` qui affiche les entiers de 1 à *n* dans l'ordre décroissant.
b) Ecrire une méthode récursive `montee(int n)` qui affiche les entiers de 1 à *n* dans l'ordre croissant cette fois.
c) Ecrire une méthode récursive `rebond(int n)` qui donne le même résultat que le bloc d'instructions `descente(n) ; montee(n) ;` mais qui n'utilise pas ces méthodes.

d) Ecrire une méthode récursive `rebondSur0(int n)` qui donne le même résultat que le bloc d'instructions `descente(n) ; affiche(0) ; montee(n) ;` mais qui n'utilise pas ces méthodes.

e) Ecrire une méthode `rechute(int n)` qui donne le même résultat que `montee(n) ; descente(n) ;` mais sans ces méthodes.

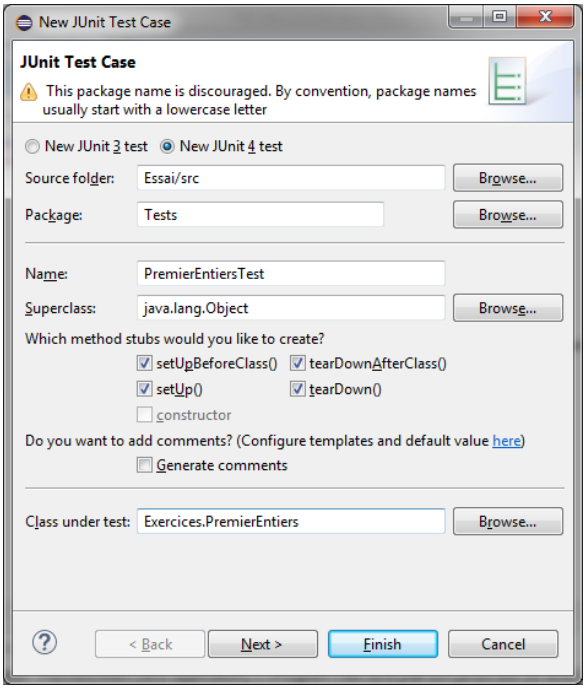
Exercice 4 Suite des n premiers entiers

a) Créer un nouveau projet Eclipse ;

b) Dans `src`, ajoutez les packages `Exercices` et `Tests` ;

c) Ajoutez une nouvelle classe `PremierEntier` qui contiendra une méthode récursive non terminale `SommePremierEntiers` qui calcule la somme des *n* premier entiers ;

d) Clic droit sur la classe `PremierEntier` et ajoutez un nouveau `JUnit Test Case` en modifiant le package en `Tests` comme ci-contre. En cliquant sur `Next`, il est possible de sélectionner les méthodes pour lesquelles on souhaite créer le test. Sélectionnez la méthode `SommePremierEntiers` ;



e) Il sera nécessaire d'importer le package `Exercices` qui contient la classe à tester ;

f) Définissez une nouvelle méthode Somme dans la classe de test qui calcule la somme des *n* premiers entiers en utilisant la formule vue en cours ;

g) Modifiez la méthode de test `testSommePremier-Entiers` générée automatiquement par Eclipse en remplaçant l'instruction `fail` par `assertEquals` qui vérifie que les deux paramètres sont identiques. Cette instruction devra faire appel à Somme et à `SommePremierEntiers` ;

h) Pour lancer le test, il suffit de faire un clic droit sur la classe de test et de l'exécuter (`Run As`) comme `JUnit Test`.

i) Dans la suite des TPs, prenez l'habitude de créer au moins une fonction de test par méthode développée.

Exercice 5 Fibonacci

a) A l'aide du cours, écrire une méthode récursive `int Fibonacci(int n)` qui calcule le terme F_n.

b) Ajouter une instruction pour que la méthode affiche le message « calcul de F(n) » au début de son exécution.

c) Arbre des appels : Il représente la manière dont les appels récursifs sont imbriqués : l'appel de `Fibonacci(n)` provoque les appels de `Fibonacci(n-1)` et `Fibonacci(n-2)`, qui eux-mêmes … Modifier la méthode pour qu'elle affiche non seulement les appels récursifs mais aussi cette structure d'imbrication. Pour cela, la méthode affichera le message « - - … - Calcul de F(n) » au début de son exécution où le nombre de tirets est la profondeur de l'appel suivi de l'affichage initial. *Indication : on pourra ajouter un argument à la méthode.*

d) En observant le résultat, on se rend compte que de nombreux calculs sont exécutés plusieurs fois. Puisque chaque appel provoque 2 sous appels récursifs, il y a une explosion exponentielle du nombre d'appels. Pour cela, on souhaite créer une nouvelle méthode récursive Fibo_econome(int n) qui renvoie un couple de résultats (F_n,F_{n+1}) ; de cette facon, un seul appel récursif est nécessaire. *Indication : il sera nécessaire de créer un nouveau type de données contenant 2 valeurs numériques.*

e) Ajouter à cette nouvelle méthode l'affichage des appels récursifs et la structure d'appel comme en c).

Exercice 6 Palindrome récursif

a) À l'aide de la classe `java.util.Scanner`, lisez une chaine de caractères au clavier. Écrivez une méthode récursive qui permet de savoir si la chaine est un palindrome. Vous n'utiliserez que les méthodes `charAt()`, `substring()` et `length()` sur la chaine de caractères.
b) Modifier la fonction récursive précédente pour ne pas tenir compte des espaces.
c) Pensez à écrire les méthodes de tests.

Exercice 7 Récursivité croisée

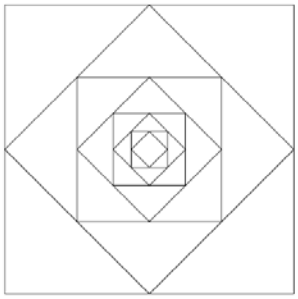
Vous trouverez ci-dessous le code source minimum pour afficher une fenêtre graphique dans laquelle vous pourrez dessiner (méthode `paint`).

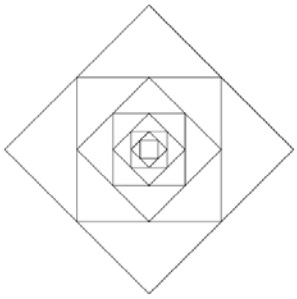
```
import java.awt.Graphics;
import javax.swing.JFrame;

public class FenetreG extends JFrame {
    public void paint(Graphics g) {

    }

    public static void main(String[] args) {
        FenetreG fenetre = new FenetreG() ;
        fenetre.setDefaultCloseOperation(EXIT_ON_CLOSE) ;
        fenetre.setExtendedState(MAXIMIZED_BOTH) ;
        fenetre.setVisible(true) ;
        fenetre.repaint();
    }
}
```



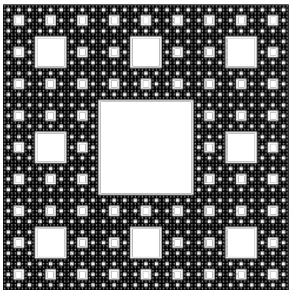


On souhaite créer deux méthodes mutuellement récursive `Droit` et `Penche` qui permettent de construire les deux figures ci-dessus uniquement en dessinant des lignes graphiques (méthode `drawLine(x1,y1,x2,y2)` sur un objet `Graphics`).

a) Réfléchissez aux paramètres nécessaires pour dessiner les 2 formes en sachant que les deux méthodes auront exactement les mêmes.
b) Identifier la récurrence entre `Droit` et `Penche`.
c) Programmer la méthode `Droit`.
d) Programmer la méthode `Penche`.
e) Et avec une seule méthode récursive ?

Exercice 8 Eponge de Menger

Afin de dessiner l'éponge de Menger (ci-contre), on remarque que le grand carré est divisé en 9 petits carrés ; le carré central est blanc, mais les 8 petits carrés extérieurs représentent tous le même dessin qui n'est en fait que le grand carré réduit. Il en est de même des 8 petits carrés d'un carré extérieur et ainsi de suite. Ainsi, dessiner le grand carré, c'est dessiner 8 fois ce même carré avec un coté réduit d'un tiers. On vient d'identifier la récurrence du dessin. Par conséquent, on peut en déduire l'algorithme ci-contre.



Le problème dans l'algorithme proposé est qu'il ne termine jamais et qu'on ne dessine jamais rien. Pour résoudre ces problèmes, il suffit simplement d'ajouter un

```
void Menger(int taille) {
    Menger(taille/3) au Nord-Ouest
    Menger(taille/3) au Nord
    Menger(taille/3) au Nord-Est
    Menger(taille/3) à Est
    Menger(taille/3) à Ouest
    Menger(taille/3) au Sud-Ouest
    Menger(taille/3) au Sud
    Menger(taille/3) au Sud-Est
}
```

cas d'arrêt qui consiste à dessiner un point quand la taille est trop petite. En effet, dessiner un carré inférieur à un pixel n'apporte pas grand-chose.

a) Implémenter cet algorithme. La taille initiale de l'éponge de Menger doit être une puissance de 3 (3^5 donne un bon résultat).
b) Afin de mieux voir le tracé de l'éponge, ajoutez une temporisation après chaque appel récursif (`try {Thread.sleep(nombre_de_millisecondes) ;} catch(Exception e) { } ;`).

Pour aller plus loin

Exercice 9 Puissance 4

On souhaite mettre en œuvre le jeu du puissance 4.

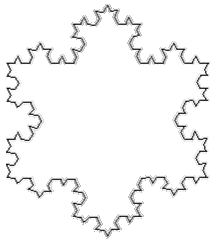
a) Réfléchissez à une modélisation du plateau de jeu et implémentez la ;
b) Implémentez les méthodes, si possible récursives, pour savoir si un joueur a gagné.

Exercice 10 Triangle Sierpinski

Sur le même principe que l'Exercice 8, programmez l'affichage du triangle de Sierpinski. 

Exercice 11 Le flocon de Koch

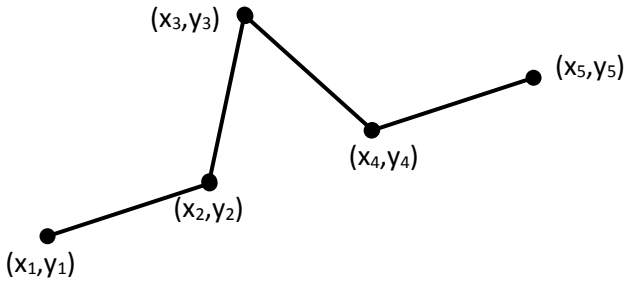
Le principe de récurrence du flocon de Koch est le suivant :



1. Considérer un segment et le diviser un segment en 3 parties identiques ;
2. Identifier le sommet du triangle équilatéral ayant pour base le segment médian de la 1ère étape ;
3. Recommencer l'opération avec chacun des segments.
4. Si le segment est trop petit, on trace un segment.

Les propriétés sont les suivantes :

$x_2 = x_1 + \frac{x_5 - x_1}{3}$, $x_4 = x_1 + \frac{2(x_5 - x_1)}{3}$, $x_3 = \frac{x_1 + x_5}{2} + \frac{\sqrt{3}(y_5 - y_1)}{6}$



Sources pour ce TP

Claire David, Structures de données et objets Java 2005-2006.

Tutoriel Eclipse – Junit : Mon premier test automatiques