MOOC Init Prog Java
Exercices facultatifs semaine 6

Fonctions simples (niveau 1)

Écrivez les méthodes suivantes, et testez-les en les appelant dans la méthode `main` de votre programme sur des exemples:

1. Écrivez une méthode `min2` qui reçoit deux arguments de type `double` et retourne le plus petit d'entre eux. Le type de retour devra donc être `double`.
2. Écrivez une méthode `min3` qui prend trois arguments de type `double` et retourne le plus petit d'entre eux. Comment utiliser la méthode `min2` du point précédent pour écrire le corps de `min3` en une ligne ?

Sapin (niveau 2)

On vous donne les deux méthodes suivantes:

```
void etoiles(int nbEtoiles)
{
  for(int i = 0; i < nbEtoiles; ++i) {
    System.out.print('*');
  }
}

void espaces(int nbEspaces)
{
  for(int i = 0; i < nbEspaces; ++i) {
    System.out.print(' ');
  }
}
```

1. Utilisez ces méthodes pour écrire une méthode qui affiche un triangle d'étoiles, et qui prend en paramètre le nombre de lignes du triangle:

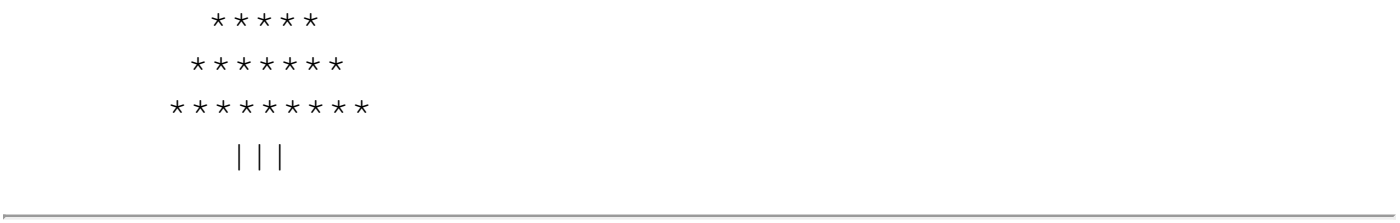
```
  *
 ***
*****
```

2. Utilisez cette méthode pour afficher un sapin:

```
   *
  ***
   *
  ***
 *****
   *
  ***
 *****
*******
   *
```

   Vous devrez modifier un peu la méthode écrite au point 1. afin que votre sapin ressemble à celui dessiné ci-dessus.
3. Le sapin précédent n'est pas très joli. Il est bien trop allongé et ne ressemble pas beaucoup à un vrai sapin. Adaptez votre code pour qu'il affiche le graphique ci-dessous:

```
   *
  ***
 *****
  ***
 *****
*******
```



Calcul approché d'une intégrale (niveau 2)

On peut montrer que pour une fonction suffisamment régulière (disons C-infinie), on a la majoration suivante :

$$\left| \int_a^b f(u)\,\mathrm{d}u - \frac{b-a}{840}\left[41f(a) + 216f(\frac{5a+b}{6}) + 27f(\frac{2a+b}{3}) + 272f(\frac{a+b}{2}) + 27f(\frac{a+2b}{3}) + 216f(\frac{a+5b}{6}) + 41f(b)\right]\right| \leq M_8 \cdot (b-a)^9 \frac{541}{315 \cdot (9!) \cdot 2^9}$$

où M_8 est un majorant de la dérivée huitième de f sur le segment [a,b].

Ecrivez un programme calculant la valeur approchée d'une intégrale à l'aide de cette formule, c'est-à-dire par

$$\frac{b-a}{840}\left[41f(a) + 216f(\frac{5a+b}{6}) + 27f(\frac{2a+b}{3}) + 272f(\frac{a+b}{2}) + 27f(\frac{a+2b}{3}) + 216f(\frac{a+5b}{6}) + 41f(b)\right]$$

Pour cela écrivez 3 méthodes :

1. Une méthode`f` de votre choix, qui corresponde à la fonction dont vous souhaitez calculer l'intégrale.
   (Essayez plusieurs cas avec x^2, x^3, ..., sin(x), 1/x, etc. Il faudra bien sur recompiler le programme à chaque fois.)
2. Une méthode `integre` qui, à partir de deux arguments correspondant à *a* et *b*, calcule la somme ci-dessus pour la fonction `f` ;
3. Une méthode qui demande à l'utilisateur d'entrer un nombre réel. S'en servir pour demander les bornes *a* et *b* de l'intégrale.

Utilisez ces méthodes dans la méthode `main()` pour réaliser votre programme.

Note : Vous pourrez trouver ici une démonstration de la formule [lien externe] donnée plus haut (Attention ! Dans la page en question $h = (b-a)/n$, ici $(b-a)/6$: 6*140 = 840).

Fonctions logiques (niveau 3)

La méthode `nonEt` vous est fournie :

```
boolean nonEt(boolean a, boolean b)
{
  return !(a and b);
}
```

qui renvoie la valeur de NON (A ET B). Comment écrire le corps des méthodes correspondant aux 3 opérateurs logiques NON, ET et OU:

```
boolean non(boolean a)

boolean et(boolean a, boolean b)

boolean ou(boolean a, boolean b)
```

en utilisant UNIQUEMENT la méthode `nonEt`, sans utiliser de `if`, ni d'opérateurs logiques `||`, `&&`, ou `!`.

Commencez par écrire la méthode `non` à l'aide de `nonEt`. Vous pouvez alors utiliser la méthode `non` en plus de `nonEt` pour écrire la méthode `et`. La méthode `ou` peut s'écrire en utilisant les méthodes `non` et `et`.

Recherche dichotomique (fonctions récursives, niveau 2)

Pour illustrer l'utilisation de la récursivité dans la recherche d'un élément dans une *liste ordonnée*, nous allons jouer à un petit jeu :

Pensez très fort à un nombre entre 0 et 100...
voilà !
L'ordinateur doit maintenant le trouver...

Principe

Pour ce faire, il pourrait énumérer tous les nombres les uns après les autres jusqu'à avoir la bonne réponse.
Mais, vous en conviendrez aisément, ce n'est pas une méthode très efficace...

Une autre méthode (plus efficace) est la méthode dite « par dichotomie » : on réduit le champ de recherche à chaque étape et on recommence la recherche sur le nouvel intervalle.

On commence avec la zone complète, puis on choisit un *pivot* (point central de la zone de recherche). En fonction de la comparaison de ce pivot avec l'élément recherché, plusieurs cas de figures peuvent se présenter:

- on a trouvé le bon élément -> il suffit de le retourner
- le pivot est plus grand que l'élément recherché -> on recommence la recherche sur le domaine délimité par le pivot (à gauche)
- le pivot est plus petit que l'élément recherché -> on recommence la recherche sur le domaine débutant par le pivot (à droite)

Mise en pratique

Dans le fichier `Dichotomie.java`

1. Définissez la méthode :

   ```
   int cherche(int borneInf, int borneSup);
   ```

   de sorte qu'elle effectue la recherche dans l'intervalle [`borneInf`, `borneSup`]

   Cette méthode devra choisir un pivot au milieu de l'intervalle (ou s'arrêter, avec un message d'erreur, si l'intervalle est vide), proposer le pivot à l'utilisateur, puis en fonction de sa réponse (<, > ou =) s'appeler elle-même sur le nouvel intervalle ainsi déterminé, ou arrêter la recherche.

   La méthode retourne la valeur trouvée (ou la borne inférieure en cas d'erreur).

2. Dans la méthode `main`, demandez à l'utilisateur de choisir (dans sa tête) un nombre entre 1 et 100 (définissez deux constantes `MIN` et `MAX`).
   Cherchez la solution à l'aide le la méthode `cherche` puis affichez le résultat trouvé.

Exemple de déroulement

```
Pensez à un nombre entre 1 et 100.

Le nombre est il <, > ou = à 50 ? <
Le nombre est il <, > ou = à 25 ? >
Le nombre est il <, > ou = à 37 ? <
Le nombre est il <, > ou = à 31 ? >
Le nombre est il <, > ou = à 34 ? <
Le nombre est il <, > ou = à 32 ? >
Le nombre est il <, > ou = à 33 ? =

Votre nombre était 33.
```

Recherche approchée de racine (niveau 2)

On souhaite écrire un programme calculant, de façon approchée, une solution à une équation de type « $f(x) = 0$ ».

La méthode à mettre en œuvre consiste à déterminer, à partir d'une approximation x_n de la solution, une meilleure solution x_{n+1}, et itérer ainsi jusqu'à ce que la valeur absolue de la différence entre x_n et x_{n+1} soit « suffisamment petite ».

Le calcul de x_{n+1} à partir de x_n se fait à l'aide de la formule suivante (« méthode de Newton ») :

$$x_{n+1} = x_n \frac{f(xn)}{f'(x_n)}$$

où f' est la dérivée de f.

Écrire un programme `Reseq.java` contenant la définition d'une méthode `f` (à choix) d'entête « `double f(double);` ».

Le programme demandera un point de départ à l'utilisateur, itérera la recherche d'une solution à partir de ce point, et affichera le résultat trouvé.

Note : Il n'est pas nécessaire d'utiliser un tableau (de taille fixe ou dynamique) pour résoudre ce problème.

Indication : Pour obtenir la valeur de la dérivée d'une fonction en un point, on s'inspire simplement de sa définition :

$$f'(x) = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon) - f(x)}{\varepsilon},$$

dont on calcule une approximation pour une valeur ε donnée (par exemple 10^{-6}) : définir la constante globale epsilon de valeur « 1e-6 », puis la fonction « double df(double x) » calculant la valeur approchée de la dérivée de f() au point x par :

```
(f(x+epsilon)-f(x))/epsilon
```

Exemple (détaillé) de déroulement

La fonction utilisée dans cet exemple est $f(x) = (x - 1.0)(x - 1.5)(x - 2.0)$:

```
Point de depart ? 1.55
au point 1.55 :
f(x) = -0.012375
f'(x) = -0.2425
nouveau point = 1.49897
au point 1.49897 :
f(x) = 0.000257739
f'(x) = -0.249997
nouveau point = 1.5
```

```
au point 1.5 :
f(x) = -2.18843e-09
f’(x) = -0.25
nouveau point = 1.5
Solution : 1.5
```

Placement sans recouvrement (niveau 2)

Le but de cet exercice est de placer sans recouvrement des objets rectilignes sur une grille carrée. Cela pourrait être par exemple une partie d'un programme de bataille navale.

Dans le fichier `Recouvrement.java` :

1. Définissez une constante entière, nommée `DIM` et de valeur 10. Elle représentera la taille de la grille (carrée).

2. Ecrivez une méthode :

   ```
   boolean remplitGrille(boolean[][] grille,
                         int ligne, int colonne,
                        char direction, int longueur);
   ```

   dont le rôle est de vérifier si le placement dans une `grille` (voir ci-dessous) d'un objet de dimension `longueur` est possible, en partant de la coordonnée (`ligne`,`colonne`) et dans la direction définie par `direction` (Nord, Sud, Est ou Ouest).

   Si le placement est possible, la méthode devra de plus effectuer ce placement (voir ci-dessous la description de la grille).

   La méthode devra indiquer (par la valeur de retour) si le placement a pu être réalisé ou non.

3. Dans la méthode `main()`
   - Définissez une **variable** nommée `grille`, de type `boolean[][]` et de taille `DIM x DIM;`.
     La valeur `true` dans une case `[i][j]` de cette grille représente le fait qu'un (bout d')objet occupe la case de coordonnées (i, j).
   - Utilisez la méthode précédente pour remplir interactivement la grille, en demandant à l'utilisateur de spécifier la position, la taille et la direction des objets à placer. Indiquez à chaque fois à l'utilisateur si l'élément a pu ou non être placé dans la grille.
     Le remplissage se termine lorsque l'utilisateur entre une position/coordonnée strictement inférieure à 0.
   - Terminer alors en "dessinant" la grille : afficher un `'.'` si la case est vide et un `'#'` si la case est occupée.

Remarques :

- Dans l'interface utilisateur, pour indiquer les positions, utilisez au choix soit les coordonnées du Java : 0 à DIM-1 (plus facile), soit les coordonnées usuelles (1 à DIM, un peu plus difficile) , **MAIS dans tous les cas utilisez les indices de 0 à DIM-1** pour votre tableau (aspect programmeur).
- Pensez à effectuer des tests de validité sur les valeurs entrées (débordement du tableau).
- pour représenter la direction, vous pouvez utiliser un caractère (`'N'` pour nord. `'S'` pour sud, etc.) ou tout autre choix de modélisation que vous jugez pertinent.
- N'oubliez pas d'initialiser la grille en tout début de programme !

Exemple de fonctionnement (version facile : coordonées de 0 à 9 et lettres pour les directions) :

```
Entrez coord. x: 2
Entrez coord. y: 8
Entrez direction (N,S,E,O): E
Entrez taille: 2
Placement en (2,8) direction E longueur 2 -> succès
Entrez coord. x: 0
Entrez coord. y: 8
Entrez direction (N,S,E,O): S
Entrez taille: 5
Placement en (0,8) direction S longueur 5 -> ECHEC
Entrez coord. x: 0
Entrez coord. y: 9
Entrez direction (N,S,E,O): O
Entrez taille: 5
Placement en (0,9) direction O longueur 5 -> succès
Entrez coord. x: -1
```

Résultat des placements :