Travaux pratiques d'informatique N° 14

Le but de cette séance est de vous permettre d'appliquer et de vérifier vos connaissances concernant les expressions lambda et les références de méthodes.

1. Créer un nouveau projet Eclipse appelé ***PrTP14Exo1*** muni d'un package nommé ***cms_tp14*** et qui permet à l'utilisateur de calculer des zéros des fonctions continues (réelles et d'une variable réelle).

Il faut respecter les consignes suivantes :

a) tout d'abord, dans une classe nommée ***DichotomieFigee***, implémenter la méthode numérique de la **bissection** pour une fonction donnée, en procédant ainsi :

- définir une méthode statique ("normale") nommée ***f*** et qui correspond à la fonction **f(x) = x – cos(x)** (qui est une fonction réelle et d'une seule variable réelle) ;
- définir une méthode statique nommée ***bissection*** qui doit avoir trois arguments (qui correspondent aux limites gauche et droite de l'intervalle pour la recherche d'un zéro et, respectivement, à la précision du calcul numérique) et qui retourne (éventuellement) un zéro de la fonction **f** dans l'intervalle précisé ;

b) ensuite, afin de rendre l'approche plus versatile, définir une interface fonctionnelle ad hoc nommée ***FI_Fonction*** avec la méthode fonctionnelle nommée ***apply*** et qui permet de manipuler des fonctions réelles d'une variables réelle ;

c) définir une classe nommé ***Fonctions*** qui prévoit :

- trois méthodes statiques ("normales") nommées ***f***, ***g*** et ***h***, et qui correspondent aux fonctions suivantes : **f(x) = x – cos(x)**, **g(x) = x/10 –sin(x) + 1/4** et **h(x) = exp(x) - 2** ;
- une méthode statique nommée ***choisirFonction*** qui permet de choisir (en fonction de son unique argument de type ***String***) une des trois fonctions mentionnées ci-dessus (grâce à l'interface fonctionnelle ad hoc ***FI_Fonction***) ;
- (facultatif) une méthode statique nommée ***chooseFunction*** qui permet de choisir (en fonction de son unique argument de type ***String***) une des trois fonctions mentionnées ci-dessus (grâce à l'interface fonctionnelle standard ***Function*** définie dans le package ***java.util.function***) ;

d) définir une classe nommée ***DichotomieAdHoc*** qui prévoit une méthode statique nommée ***bissection*** ; cette méthode est plus versatile que la méthode ***bissection*** de la classe ***DichotomieFigee*** car, grâce à un quatrième argument de type ***FI_Fonction***, elle permet de préciser, lors de son appel, la fonction pour laquelle on cherche un zéro ;

e) (facultatif) définir une classe nommée ***DichotomieStandard*** qui est similaire à la classe ***DichotomieAdHoc*** indiquée ci-dessus, mais qui utilise l'interface standard ***Function*** à la place de l'interface ad hoc ***FI_Fonction*** ;

f) définir la classe principale nommée ***CP_Dichotomie*** qui, pour chacune des fonctions mentionnées au point **c),** permet de calculer (et d'afficher dans la fenêtre console) un zéro dans l'intervalle fermé **[0, 1]** ; le calcul de chaque zéro devra être fait de plusieurs manières équivalentes ; par exemple, la méthode ***bissection*** de la classe ***DichotomieAdHoc*** (ou de la classe ***DichotomieStandard***) peut être appelée en fournissant comme quatrième argument effectif ;

- une variable locale qui stocke l'adresse d'une instance d'une classe anonyme qui implémente convenablement l'interface ad hoc ***FI_Fonction*** (ou l'interface standard ***Fonction***) ;
- (directement) l'adresse d'une instance d'une classe anonyme qui implémente convenablement l'interface ad hoc ***FI_Fonction*** (ou l'interface standard ***Function***) ;
- une variable locale qui stocke la référence d'une expression lambda (avec plusieurs syntaxes possibles) ;
- (directement) une expression lambda (avec plusieurs syntaxes possibles) ;
- une variable locale qui stocke la référence d'une méthode statique de la classe ***Fonctions*** ;
- (directement) la référence d'une méthode statique de la classe ***Fonctions*** ;
- une variable locale dont la valeur est obtenue grâce à un appel de la méthode ***choisirFonction*** (ou ***chooseFunction***) ;
- (directement) l'appel de la méthode ***choisirFonction*** (ou ***chooseFunction***).

Remarque : Vous trouvez sur le bureau virtuel du CMS une définition de la classe ***DichotomieFigee*** qui implémente notamment l'algorithme de **bissection** qui appartient à la catégorie de méthodes numériques dites de **dichotomie** (qui sont des méthodes permettant de calculer numériquement un zéro d'une fonction continue dans un intervalle donné).

2. Créer un nouveau projet Eclipse appelé ***PrTP14Exo2*** muni d'un package nommé ***cms*** et qui permet à l'utilisateur de calculer des intégrales définies des fonctions (continues, réelles et d'une variable réelle) sur des intervalles donnés.

Il faut respecter les consignes suivantes :

a) tout d'abord, dans une classe nommée ***SimpsonFigee***, implémenter la méthode d'intégration numérique dite de **Simpson** pour une fonction donnée, en procédant ainsi :

- définir une méthode statique ("normale") nommée ***f*** et qui correspond à la fonction **f(x) = sin(x) + cos(3x) + 5/2** (qui est une fonction réelle et d'une seule variable réelle) ;
- définir une méthode statique nommée ***simpson*** qui doit avoir trois arguments (qui correspondent aux limites gauche et droite de l'intervalle d'intégration et, respectivement, à la précision du calcul numérique) et qui retourne la valeur (approchée) de l'intégrale définie de la fonction **f** sur l'intervalle précisé ;

b) ensuite, afin de rendre l'approche plus versatile, définir une interface fonctionnelle ad hoc nommée ***FI_Fonction*** avec la méthode fonctionnelle nommée ***apply*** et qui permet de manipuler des fonctions réelles d'une variables réelle ;

c) définir une classe nommé ***Fonctions*** qui prévoit :

- trois méthodes statiques ("normales") nommées ***f***, ***g*** et ***h***, et qui correspondent aux fonctions suivantes : **f(x) = sin(x) + cos(3x) + 5/2**, **g(x) = |x|** et **h(x) = 1/x** ;
- une méthode statique nommée ***choisirFonction*** qui permet de choisir (en fonction de son unique argument de type ***String***) une des trois fonctions mentionnées ci-dessus (grâce à l'interface fonctionnelle ad hoc ***FI_Fonction***) ;
- (facultatif) une méthode statique nommée ***chooseFunction*** qui permet de choisir (en fonction de son unique argument de type ***String***) une des trois fonctions mentionnées ci-dessus (grâce à l'interface fonctionnelle standard ***Function*** définie dans le package ***java.util.function***) ;

d) définir une classe nommée ***SimpsonAdHoc*** qui prévoit une méthode statique nommée ***simpson*** ; cette méthode est plus versatile que la méthode ***simpson*** de la classe ***SimpsonFigee*** car, grâce à un quatrième argument de type ***FI_Fonction***, elle permet de préciser, lors de son appel, la fonction qui est intégrée ;

e) (facultatif) définir une classe nommée ***SimpsonStandard*** qui est similaire à la classe ***SimpsonAdHoc*** indiquée ci-dessus, mais qui utilise l'interface standard ***Function*** à la place de l'interface ad hoc ***FI_Fonction*** ;

f) définir la classe principale nommée ***CP_Simpson*** qui permet de calculer (et d'afficher dans la fenêtre console) l'intégrale définie pour chacune des trois fonctions mentionnées au point **c)** sur l'intervalle **[1, 5]**, **[0, 2]** et, respectivement, **[1, e]** ; le calcul de chaque intégrale définie doit être fait de plusieurs manières équivalentes ; par exemple, la méthode ***simpson*** de la classe ***SimpsonAdHoc*** (ou de la classe ***SimpsonStandard***) peut être appelée en fournissant comme quatrième argument effectif ;

- une variable locale qui stocke l'adresse d'une instance d'une classe anonyme qui implémente convenablement l'interface ad hoc ***FI_Fonction*** (ou l'interface standard ***Fonction***) ;
- (directement) l'adresse d'une instance d'une classe anonyme qui implémente convenablement l'interface ad hoc ***FI_Fonction*** (ou l'interface standard ***Function***) ;
- une variable locale qui stocke la référence d'une expression lambda (avec plusieurs syntaxes possibles) ;
- (directement) une expression lambda (avec plusieurs syntaxes possibles) ;
- une variable locale qui stocke la référence d'une méthode statique de la classe ***Intégrant*** ;
- (directement) la référence d'une méthode statique de la classe ***Intégrant*** ;
- une variable locale dont la valeur est obtenue grâce à un appel de la méthode ***choisirFonction*** (ou ***chooseFunction***) ;
- (directement) l'appel de la méthode ***choisirFonction*** (ou ***chooseFunction***).

Remarque : Vous trouvez sur le bureau virtuel du CMS une définition de la classe ***SimpsonFigee*** qui implémente notamment l'algorithme de **simpson** qui appartient à la classe de méthodes numériques d'intégration de fonctions réelles d'une variable réelle.