| EXAMEN DE FIN D'ÉTUDES SECONDAIRES GÉNÉRALES<br>**Sessions 2023 – QUESTIONNAIRE ÉCRIT** | | | | | |
|---|---|---|---|---|---|
| *Date :* | 19.09.23 | *Durée :* | 08:15 - 12:15 | *Numéro candidat :* | |
| *Discipline :* | Programmation | *Section(s) :* | GIN | | |

**Indications importantes :**

- ☞ Renommez d'abord le dossier EEEE_GIN_x-xx en remplaçant EEEE par le code de votre établissement et x-xx par votre numéro d'examen !
- ☞ Les projets *NetBeans* à compléter se trouvent dans ce dossier.
- ☞ Des modèles de démonstration des programmes finis se trouvent dans le sous-dossier **dist**.
- ☞ Lisez les énoncés des deux questions chaque fois jusqu'à la fin avant de commencer.

## Question 1 – Transform2D [19 points]

Dans le programme **Transform2D**, des objets graphiques colorés, composés uniquement de lignes peuvent être placés sur une surface de dessin à l'aide de la souris. Leur représentation peut être transformée en changeant les valeurs d'une matrice de transformation à l'aide de 4 glisseurs. Le graphique montre alors les objets originaux ainsi que leur représentation transformée avec deux degrés différents de transparence

Trois types d'objets sont définis : des boîtes (**Box**), des personnes (**Person**) et des maisons (**House**). Chaque objet peut avoir une couleur différente (**color**). La position des objets est définie par les coordonnées de leur centre (**center**). Leur dimension est définie par leur largeur (**width**).
P.ex. : le constructeur de **Box** est défini comme suit :

```
public Box(Point center, int width, Color color) { ... }
```

Pour tester le fonctionnement, le programme permet de placer 4 sortes d'objets :

- Si le bouton *Shift* du clavier n'est pas enfoncé :
  - le bouton gauche de la souris ajoute une boîte bleue de largeur 50 pixels,
  - le bouton droit de la souris ajoute une boîte rouge de largeur 25 pixels.
- Si le bouton *Shift* du clavier est enfoncé :
  - le bouton gauche de la souris ajoute une personne orange de largeur 25 pixels,
  - le bouton droit de la souris ajoute une maison gris foncé de largeur 150 pixels.

Ouvrez le programme de démonstration **Transform2D-demo.jar** pour tester le fonctionnement du programme fini.

Le projet **Transform2D** sait déjà placer et dessiner des objets sur la surface de dessin. La classe **Matrix2D** qui sert à transformer les coordonnées de points et de listes de lignes existe déjà, mais elle n'est pas encore appliquée aux objets.

**Travail à réaliser :**

Ouvrez d'abord le projet **Transform2D** en NetBeans.

Dans la suite, les classes **Line** et **Matrix2D** ne doivent pas être changées. Elles ne sont pas considérées pour l'évaluation.

**A. Amélioration du code [9 points]**

Ouvrez les différentes classes et observez leur contenu, surtout celles du paquet **lineworld.items**. Vous allez voir qu'elles sont programmées de façon très naïve et simpliste, sans profiter des possibilités de la programmation orientée objet.

Sans changer la fonctionnalité du programme, améliorez le code des classes **en profitant au mieux des possibilités de la programmation objet (héritage, polymorphisme, accessibilité, ...).** Évitez la duplication superflue de structures de données et de code. Respectez le principe MVC (tel qu'il est déjà respecté dans le programme original).

A la fin, dans **Mainframe** et **DrawPanel**, l'ensemble de tous les objets graphiques doit être géré dans une seule classe nommée **Items** au lieu des trois classes **Boxes**, **Houses**, **Persons**. Créez une classe supplémentaire si nécessaire !

A la fin, supprimez les classes et le code qui n'est plus nécessaire.

*Remarque : Si par malheur vous avez perdu ou supprimé un bout du code original dont vous avez encore besoin, alors vous pouvez regarder dans le projet **Transfer2D-backup** qui contient une copie des fichiers originaux. Le projet **Transfer2D-backup** n'est pas évalué.*

**B. ArrayList --> Linked List [5 points]**

Dans les classes originales **Boxes**, **Houses** et **Persons**, les objets étaient gérés dans des listes du type **ArrayList**<...>. Seulement dans votre classe **Items**, remplacez **ArrayList**<...> par une liste chaînée que vous programmez par vos soins (**sans** utiliser la classe prédéfinie **LinkedList**!).

Cette liste chaînée commence dans ***Items*** et elle est gérée dans ***Items***. Vous n'avez **pas besoin** de créer des classes supplémentaires *LinkedList* ou *Node*, ni une liste générique. Les nouveaux éléments sont à ajouter à la fin de la liste.

*Remarque : Si vous n'avez pas réussi à réaliser le point A. et si vous n'avez pas de classe **Items**, alors réalisez le point B. dans la classe **Boxes** en ajoutant un commentaire bien visible au début de la classe :* `// ***** Point B. ArrayList --> LinkedList *****`

**C. Dessin des objets transformés [5 points]**

Tout en respectant les principes donnés sous le point A., modifiez le dessin des objets comme suit :

- Les lignes originales de chaque objet sont dessinées avec la couleur donnée (attribut **color**), mais de façon presque transparente (valeur *Alpha*: 32, si nécessaire consultez JavaDoc)
- Modifiez le programme, de façon que la matrice **matrix2D** qui est créée et actualisée dans **MainFrame**, soit aussi disponible lors du dessin des objets graphiques.
- Utilisez la méthode **transform** de **matrix2D** pour transformer les coordonnées des lignes des objets. Cette méthode retourne une liste avec les lignes transformées comme résultat.
- Les lignes transformées de chaque objet sont dessinées avec la couleur donnée (attribut **color**), mais à moitié transparente.

## Question 2 – CrossFractal [9,5 points]

Un modèle de démonstration du projet fini se trouve dans votre dossier sous **dist/CrossFractal-demo.jar**. Démarrez ce programme maintenant pour vous familiariser avec son fonctionnement !

Ouvrez ensuite le projet **CrossFractal**. Avant d'ouvrir la classe **MainFrame**, effectuez un *'Clean and Build'* du projet (pour garantir que **DrawPanel** est bien intégré dans **MainFrame**).

Le code dans **MainFrame** et dans **DrawPanel** restera inchangé. Toutes vos modifications se feront dans la classe **CrossFractal**.

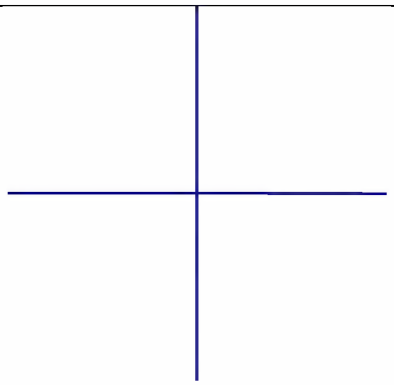
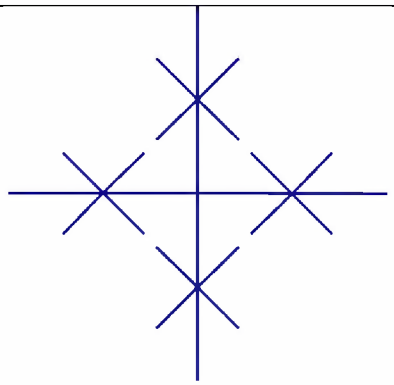
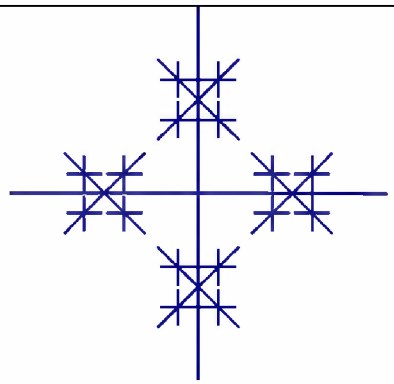
La fractale **CrossFractal** est construite de façon récursive:

Sur une croix de base sont placées 4 croix dont les centres se trouvent chaque fois au milieu de chaque bras de la croix de base. Les angles des quatre croix sont déphasés de 45 degrés par rapport à la croix de base. Ensuite, on fait de même récursivement pour les 4 croix qu'on vient de placer (qui serviront alors de bases aux prochaines croix) etc.

Les croix qui sont placées sur la croix de base ont un diamètre plus petit que la croix de base. Le facteur de réduction est fixé par l'attribut **shrinkingFactor** de **CrossFractal**. **shrinkingFactor** a une valeur entre 0,000 et 0,999. Si p.ex. **shrinkingFactor** est égal à 0,250, alors le diamètre de chaque croix est un quart (25%) de sa croix de base.

L'attribut **depth** de **CrossFractal** indique la profondeur de la récursion.

**Exemples (pour shrinkingFactor = 0,300 ) :**

| pour *depth*=1 : | pour *depth*=2 : | pour *depth*=3 : |
|---|---|---|
|  |  |  |

Remarques :

- Tous les calculs sont effectués avec des nombres réels. Les valeurs sont arrondies uniquement lors du dessin à l'écran.
- Rappel : les méthodes trigonométriques de Java calculent avec des angles en radians.

**Travail à réaliser :** Complétez les méthodes dans la classe CrossFractal.

Dans la classe **CrossFractal**, développez la méthode privée ***getEndPoint*** qui sert à déterminer le point final d'un segment d'une croix, pour un point de départ, un angle et une longueur donnés. Cette méthode vous aidera à trouver facilement les coordonnées cartésiennes (point(x,y)) pour des coordonnées polaires (point, longueur, angle) données. Paramètres de ***getEndPoint*** :

| | |
|---|---|
| **Point2D p** | les coordonnées du point de départ du segment (nombres réels) |
| **double length, double angle** | la longueur du segment et son angle (en degrés) |

Valeur retour de la méthode :

| | |
|---|---|
| **Point2D** | les coordonnées de la fin (du dernier point) du segment |

[2p]

Définissez la méthode privée ***drawLine*** qui sert à dessiner une ligne entre deux points donnés du type *Point2D*. Paramètres de la méthode ***drawLine***:

| | |
|---|---|
| **Graphics g** | le canevas cible |
| **Point2D start, end** | les coordonnées des points de départ et d'arrivée |

[0,5p]

Définissez la méthode privée ***drawCross*** qui sert à dessiner une croix inclinée de *angle* degrés ! La méthode ***drawCross*** se sert des méthodes ***drawLine*** et ***getEndPoint*** pour tracer les 4 lignes de la croix. Paramètres de la méthode ***drawCross*** :

| | |
|---|---|
| **Graphics g** | le canevas cible |
| **Point2D center** | les coordonnées du centre de la croix |
| **double length, double angle** | le diamètre de la croix et l'angle d'inclinaison de la croix (en degrés) |

[1,5p]

Définissez la méthode privée récursive ***drawFractal*** qui sait dessiner une fractale de croix ! Servez-vous des procédures ***getEndPoint*** et ***drawCross*** !
Paramètres de la procédure ***drawFractal*** :

| | |
|---|---|
| **Graphics g** | le canevas cible |
| **Point2D center** | les coordonnées du centre de la fractale |
| **double length** | le diamètre de la croix de base de la fractale à dessiner |
| **double angle** | l'angle d'inclinaison de la croix de base de la fractale à dessiner (en degrés) |
| **int depth** | la profondeur de la récursion |

Complétez la méthode **draw(Graphics g, int width, int height)** dans **CrossFractal** pour lancer la représentation. La croix de base de toute la fractale est carrée, c.-à-d. ses bras horizontaux et verticaux ont la même longueur. Cette longueur est la longueur maximale possible sur le canevas. Les bras de la croix de base sont parallèles aux bords du canevas.

[5,5p]

## Question 3 – JavaCodeScanner [31,5 points]

Avec le programme **JavaCodeScanner**, il est possible de parcourir une arborescence sur le disque et d'établir une liste de tous les fichiers Java (extension .java) qui se trouvent dans le dossier donné ou récursivement dans l'un de ses sous-dossiers. Les fichiers trouvés sont ajoutés à la liste, c.-à-d. les fichiers qui s'y trouvent déjà ne sont pas effacés de la liste. Il est aussi possible de choisir un seul fichier java et de l'ajouter à la liste.

La liste des fichiers est affichée dans le programme et l'utilisateur peut sélectionner un fichier de la liste. Le texte (code Java) contenu dans le fichier sélectionné est alors affiché dans un composant du type *JTextArea*.

Deux options (cases à cocher) permettent de modifier la représentation du code Java :

- **Suppress single line comments ( //... )**
  Les commentaires à une seule ligne ne sont pas affichés.
- **Suppress multiple lines comments ( /* ... */ )**
  Les commentaires à plusieurs lignes ne sont pas affichés.

Le bouton ***Clear*** efface la liste des fichiers.

Le bouton ***Export All...*** crée un fichier texte avec le code contenu dans tous les fichiers qui se trouvent actuellement dans la liste.

Ouvrez le programme de démonstration **JavaCodeScanner-demo.jar** pour tester le fonctionnement du programme fini!

**Travail à réaliser :**

Ouvrez le projet **JavaCodeScanner** en NetBeans.

Le projet contient déjà l'interface graphique dans **MainFrame.java** et une classe vide **CodeUtils.java**.

La classe **JavaFiles.java** contient déjà une liste **alFileNames** pour les noms de fichiers (avec leur chemins absolus) ainsi que quelques méthodes standard (**add**, **clear**, **size**, **get**, **toArray**). En plus cette classe contient la déclaration des deux attributs publics qui devront refléter l'état des deux options décrites ci-dessus (**suppressSingleLineComments** et **suppressMultiLineComments**).

Pour toute la suite : Toutes les exceptions qui pourraient apparaître dans une classe modèle lors du traitement de fichiers doivent être renvoyées et traitées dans la vue (**MainFrame**). Un dialogue d'erreur doit être affiché avec un message donnant la description spécifique de l'erreur.

Exemple:

**A. Ajout de fichiers à la liste [12,25 points]**

- Dans **JavaFiles**, créez la méthode **récursive**

  ```
  public void scan(String pathName, String type)
  ```

  qui sert à ajouter à **alFileNames** tous les fichiers portant l'extension donnée par **type** qui se trouvent dans le dossier donné par **pathName** ou récursivement dans l'un de ses sous-dossiers, sous-sous-dossiers, sous-sous-sous-dossiers etc. [6p]
  - La liste **alFileNames** contiendra le chemin absolu pour chaque fichier.
  - Le paramètre **pathName** doit contenir le chemin absolu du fichier ou du répertoire à scanner.
  - Si **pathName** contient le nom et le chemin d'un fichier, et si ce fichier porte l'extension donnée par **type**, alors le fichier est ajouté à la liste.
  - La vérification de l'extension **type** ignore la casse (majuscule, minuscule). P.ex. si l'extension donnée est "java", les fichiers portant l'extension "Java" ou "JAVA" sont aussi ajoutés.
  - Si le chemin donné est **null** ou n'existe pas, alors une exception du type **FileNotFoundException** est lancée avec un message correspondant.

  Indication : Utilisez les méthodes de la classe **java.io.File**, notamment: **isDirectory**, **isFile**, **listFiles**. Consultez *JavaDoc* à ce sujet.

- Dans **MainFrame**, ajoutez une instance **javaFiles** et une méthode

  ```
  public void updateList()
  ```

  qui actualise le contenu de la **JList fileList** ainsi que le contenu de **listSizeLabel** qui contient toujours le nombre actuel des fichiers dans **javaFiles**. [1,25p]

- Ajoutez la réaction au bouton **scanButton** qui sert à choisir un répertoire ou un fichier à ajouter à l'aide d'un dialogue d'ouverture de fichiers. [5p]
  - Définir un attribut **currentFilePath** qui contient le chemin où le dialogue doit s'ouvrir. **currentFilePath** est actualisé après chaque action d'ouverture ou de sauvegarde de fichiers (voir sous C.). De cette façon les dialogues s'ouvrent toujours à l'endroit où le dernier dialogue a été fermé avec succès. **currentFilePath** est initialisé par le chemin du projet même.
  - Ajoutez au dialogue d'ouverture un filtre avec une description pour les fichiers *.java*.
  - Si l'utilisateur confirme son choix dans le dialogue, faites appel à la méthode **scan** que vous avez définie en utilisant l'extension *"java"* comme type. Actualisez la vue.
  - Pour permettre la sélection de fichiers et de répertoires, activez l'option suivante pour votre dialogue:

    ```
    ....setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES)
    ```

**B. Affichage du contenu d'un fichier [7,25 points]**

- Dans **JavaFiles**, créez la méthode

  ```
  public String getContent(int index)
  ```

  qui retourne dans une seule chaîne de caractères le contenu entier du fichier de **alFileNames** dont l'indice est donné comme paramètre. Dans la chaîne de caractères retournée, le caractère `'\n'` est utilisé pour séparer les lignes lues dans le fichier. [3p]

- Dans **MainFrame**, ajoutez une méthode
  `public void updateContentArea()`
  qui actualise le contenu de la **JTextArea fileViewTextArea** en la remplissant par le contenu du fichier qui est actuellement sélectionné dans **fileList**. Si aucun fichier n'est sélectionné, alors **fileViewTextArea** reste vide. Appelez cette méthode à chaque changement dans la vue. (Le contenu d'une **JTextArea** est changé par **setText(...)** comme pour les composants **JTextField**, mais avec la différence que **JTextArea** sait représenter des textes à plusieurs lignes.) [4p]
- Dans **MainFrame**, réalisez la réaction au bouton **clear** qui efface le contenu de la liste des fichiers et actualise la vue en conséquence. [0,25p]

**C. Exportation du contenu de tous les fichiers [4 points]**

- Dans **JavaFiles**, créez la méthode
  `public void exportAllContents(String exportFileName)`
  qui crée un nouveau fichier texte avec le contenu de tous les fichiers qui se trouvent actuellement dans la liste **alFileNames**. Le chemin absolu avec le nom du fichier à créer est donné dans **exportFileName**. Le contenu de chaque fichier est précédé par un entête contenant le chemin et le nom du fichier. L'entête se présente comme suit :

```
/* ================================================================================
<chemin et nom du fichier.java>
 * ================================================================================ */
```

  A la fin du contenu de chaque fichier est ajouté un caractère de saut de page `'\f'`. [2,5p]
- Dans **MainFrame**, réalisez la réaction au bouton **Export All...** qui sert à choisir un fichier pour l'exportation à l'aide d'un dialogue de sauvegarde de fichiers. Si l'utilisateur confirme son choix dans le dialogue, le fichier est créé. (L'extension du fichier est au choix de l'utilisateur, donc pas besoin de définir de filtre spécifique) [1,5p]

**D. Les filtres de suppression de commentaires [8 points]**

- Dans **MainFrame**, veillez à ce que l'état des deux cases à cocher (**suppress...Comments**) soit toujours reflété aux attributs prévus dans **javaFiles**. Profitez pour cela de la méthode **updateContentArea**. Utilisez l'événement **itemStateChanged** des cases à cocher. [1p]

  Remarques pour la suite :

  - Les caractères `'\n'` (nouvelle ligne) et `'\t'` (tabulation) ont une longueur de **1** dans une chaîne de caractères, même s'ils sont notés avec 2 caractères. (Le symbole '\' est utilisé dans le code pour que le système puisse distinguer ces caractères de formatage des caractères normaux.)
  - Pour simplifier, nous allons ignorer le fait que des séquences "//", "/*" ou "*/" peuvent se trouver à l'intérieur d'une chaîne de caractères dans le code java.
    P.ex.:

    ```
    System.out.println("This is a double slash: // !");
    ```

    Dans ce cas, notre programme supprimera la partie `"// !");"` jusqu'à la fin de la ligne, même s'il ne s'agit pas d'un commentaire.

Les méthodes suivantes sont à réaliser uniquement avec les méthodes et techniques qui sont au programme du cours (**indexOf**, **substring**, **replace**, **charAt**, ...), donc **sans** utiliser par exemple **removeAll** ou des *'regular expressions' !*

Dans les méthodes suivantes, le paramètre **code** contient le texte complet d'un fichier *java*. Dans **code**, les lignes du programme sont séparées par un caractère de retour à la ligne `'\n'`.

- Dans **CodeUtils**, créez la méthode
  ```
  public static String suppressSingleLineComments(String code)
  ```
  qui supprime de **code** les commentaires à une seule ligne. Ces commentaires commencent par la séquence `"//"` et se terminent à la fin de la ligne (donc à l'encontre d'un caractère de fin de ligne `'\n'`). Le symbole de retour à la ligne lui-même n'est pas supprimé. Le texte ainsi modifié est renvoyé comme résultat. [4p]

- Dans **CodeUtils**, créez la méthode
  ```
  public static String suppressMultiLineComments(String code)
  ```
  qui supprime de **code** les commentaires à plusieurs lignes. Ces commentaires commencent par la séquence `"/*"` et se terminent par la séquence `"*/"`. Il peut donc s'agir de commentaires multi lignes simples `"/* ... */"` ou de commentaires *JavaDoc* `"/** ... */"`. Les séquences `"/*"` et `"*/"` sont aussi supprimées, ainsi que tout ce qui se trouve entre les deux (donc aussi d'éventuels retours à la ligne). [2p]

- Dans la méthode **getContent** de **JavaFiles**, veillez à ce que les méthodes définies dans **CodeUtils** soient appliquées au contenu du fichier en dépendance des attributs du même nom (**suppressSingleLineComments** et **suppressMultiLineComments**). [1p]