EXAMEN DE FIN D'ÉTUDES SECONDAIRES – Sessions 2024

QUESTIONNAIRE

Date :	24.05.24		*Horaire :*	08:15 - 12:15		*Durée :*	4 heures
Discipline :	PROGR	*Type :*	écrit	*Section(s) :*	Division technique générale Section informatique (GIN)		
					Numéro du candidat :		

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 - Le mot le plus long [41,5 points]

Le programme à compléter **TheLongestWord** est inspiré du jeu télévisé ***Le mot le plus long*** qui a une longue tradition à la télévision francophone. En France le jeu a été intégré dans l'émission ***Des chiffres et des lettres*** depuis 1972. Le principe du jeu est le suivant :

Une série de **n** lettres alphabétiques est tirée aléatoirement comme suit :
Le candidat peut choisir le nombre de voyelles *(DE: Vokale, EN: vowels)*, le reste des lettres sera rempli par des consonnes *(DE: Konsonanten, EN: consonants)*.
Le candidat a ensuite 30 secondes pour trouver le mot français le plus long en se servant uniquement des lettres tirées. Chacune des lettres de la série peut être utilisée une seule fois. Les mots sont acceptés uniquement s'ils se trouvent dans un dictionnaire donné.
A la fin, le jury présente le ou les mots les plus longs qui se trouvent dans le dictionnaire et qu'on peut former avec les lettres données.
Lors de la composition des mots, toutes les lettres sont en majuscules et tous les accents sont ignorés, c.-à-d. les lettres accentuées sont remplacées par les mêmes lettres non accentuées.

Exemple pour un jeu de 10 lettres avec 5 voyelles :

Lettres à utiliser :	"AEIYEHGBMV"
Exemples de mots acceptés:	"AGE", "AGEE", "HAIE", "MAGIE", "AIMEE", "ABIMEE"
Exemples de mots non acceptés:	"AGEE**S**", "AG EE", "GA**G**E", "ABMV"

Dans notre version du jeu, le joueur choisit d'abord le nombre **n** de lettres, puis il choisit le nombre **v** de voyelles qu'il désire. Lorsqu'il clique sur *'Générer la liste des lettres*' les **v** voyelles et **(n–v)** consonnes sont choisies au hasard par l'ordinateur. Le compte à rebours commence au même moment et le candidat peut entrer son mot au clavier.

Lorsque le compte à rebours est terminé, l'ordinateur vérifie que le mot donné par le candidat est formé par les lettres données et que le mot se trouve dans le dictionnaire. Si tel est le cas, le mot est accepté, sinon il est rejeté.

Enfin le candidat peut cliquer sur *'Trouver les mots les plus longs'* pour faire afficher tous les mots les plus longs possibles que l'ordinateur a trouvé dans le dictionnaire. Si p.ex. le mot le plus long est formé de 7 lettres, l'ordinateur affiche toutes les possibilités avec 7 lettres. L'ordinateur affiche aussi la longueur des mots les plus longs et le temps en millisecondes qu'il a fallu pour trouver ces mots.

Ouvrez le programme de démonstration **TheLongestWord.jar** et familiarisez-vous avec le programme et son fonctionnement !

Mémorisation du dictionnaire

Le fichier avec les mots du dictionnaire se trouve dans le fichier "**MotsFrancais.txt**" qui se trouve dans le dossier "**src/wordlist**" de votre projet. (Vous pouvez l'ouvrir en NetBeans en le double-cliquant).

Pour accélérer la recherche des mots, nous allons mémoriser les mots qui ont la même longueur dans une même liste chaînée. Ainsi, nous aurons un tableau (*array*) **arWordLists** de listes chaînées: Dans chaque élément de **arWordLists** se trouve une liste chaînée de mots de même longueur. P.ex. l'élément **arWordLists[5]** contiendra une liste chaînée avec tous les mots de longueur 5 du dictionnaire (comme "AIMEE", "AIMER", "MAGIE", etc.). La liste **arWordLists[0]** sera vide.
La liste chaînée sera à programmer par vous-même **sans** employer de collections prédéfinies.

Travail à faire:

Ouvrez le projet existant **TheLongestWord** qui contient déjà les paquets ainsi que la classe **MainFrame** avec l'interface graphique. Ouvrez les classes **WordNode**, **WordList** et **Dictionary** ainsi que la classe **Game**. Les en-têtes des méthodes de ces classes sont déjà préparées. Réalisez les parties manquantes selon les descriptions ci-dessous et le diagramme UML à la fin de l'énoncé de cette question.

WordNode et WordList [9 points]

Chaque instance de **WordNode** sert à mémoriser un mot **word** dans la liste chaînée. [0,5p]

Chaque instance de **WordList** sert à gérer les mots (d'une même longueur) dans une liste chaînée.

1. La méthode **addToStart** ajoute un nouveau mot au début de la liste. [0,5p]
2. La méthode **printAll** affiche ligne par ligne tous les mots de la liste dans la console. Cette méthode nous servira à vérifier le bon fonctionnement de la liste. [1p]
3. La méthode **size** retourne la longueur de la liste. [1,5p]
4. La méthode **contains** retourne *true* si et seulement si le mot donné comme paramètre se trouve dans la liste. La recherche doit se terminer dès que le mot est trouvé (sans effectuer de *break*, *return* ou similaire dans la boucle). [2,5p]
5. Une instance de **WordList** doit pouvoir être parcourue par un *for each*. P.ex.: **for (String s : wordList)** ... Faites les modifications nécessaires. [3p]

Dictionary [3,5 points]

Une instance de **Dictionary** sert à gérer un tableau (*array*) **arWordLists** de listes de mots du type **WordList**. Elle contiendra tous les mots du dictionnaire.
Rappel: **arWordLists [i]** contient la liste avec les mots de longueur **i**. [0,25p]

1. Le constructeur crée **arWordLists** et l'initialise avec autant de listes vides que l'indique son paramètre **size**. [1p]
2. La méthode **add** ajoute le mot **word** dans la liste de mots correcte du tableau, en fonction de la longueur du mot. [0,5p]
3. La méthode **get** retourne la liste chaînée d'indice **i** (c.-à-d. la liste qui contient les mots de longueur **i**). [0,25p]
4. La méthode **printSizes** affiche ligne par ligne la taille de toutes les listes du tableau dans la console. Cette méthode nous servira à vérifier le bon fonctionnement de la classe. [0,5p]
5. La méthode **printAll** affiche ligne par ligne tous les mots du dictionnaire dans la console. Cette méthode nous servira à vérifier le bon fonctionnement de la classe. [0,5p]
6. La méthode **contains** retourne *true* si et seulement si le mot donné comme paramètre se trouve dans le dictionnaire. La méthode profite de l'organisation de la classe pour limiter le temps de recherche au strict nécessaire! [0,5p]

Game [19 points]

Les constantes VOWELS et CONSONANTS sont prédéfinies et contiennent les voyelles et les consonnes de l'alphabet. Un commentaire contient toutes les lettres accentuées à traiter. Il sera utile pour copier les lettres accentuées si vous ne les trouvez pas sur le clavier.

1. Initialisez le dictionnaire, sachant que le mot le plus long du dictionnaire est composé de 25 lettres. [0,25p]
2. La méthode **dictionaryContains** retourne *true* si et seulement si le mot donné comme paramètre se trouve dans le dictionnaire. [0,25p]
3. La méthode **convertWord** sert à convertir un mot dans un mot compatible avec le jeu, c.-à-d. [3,25p] :
   - les mots contenant un trait d'union '-' sont ignorés (retour null),
   - toutes les lettres sont converties en majuscules,
   - toutes les lettres accentuées sont remplacées par les mêmes lettres non accentuées en majuscules.

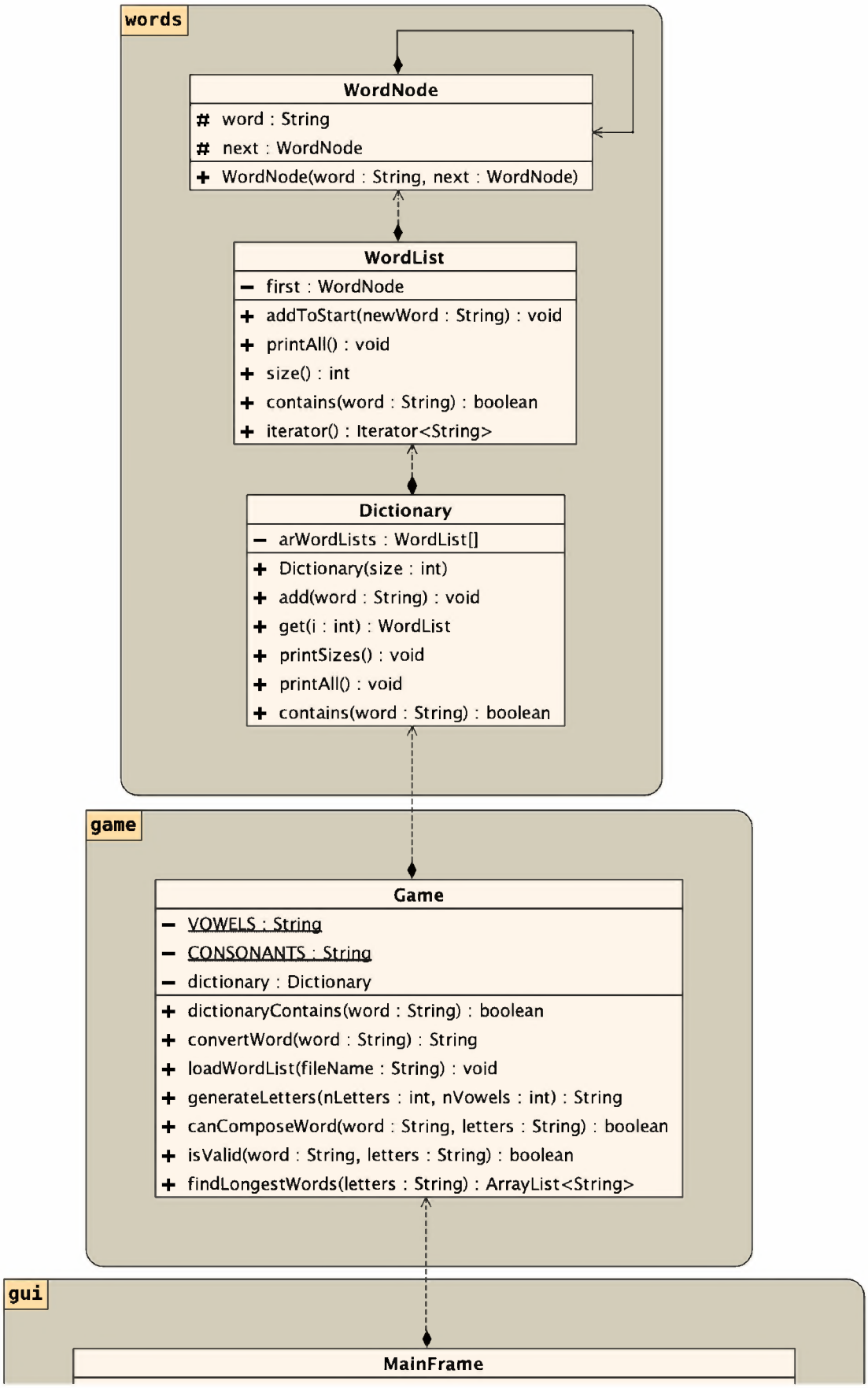
   Exemples : aimée -> AIMEE Âgée -> AGEE façonnées -> FACONNEES
4. La méthode **loadWordList** remplit le dictionnaire par les mots du fichier donné comme paramètre. Lors de la lecture, tous les mots sont convertis comme décrit ci-dessus. Les mots vides ou null sont ignorés.
   Faites appel ici aux méthodes **print...** du dictionnaire pour vérifier son bon fonctionnement. Si tout fonctionne bien, vous pouvez mettre ces appels en commentaires.
   Pour vérifier : il y a 49230 mots de longueur 9 et 50904 mots de longueur 10. [3p]
5. La méthode **generateLetters** retourne dans une chaîne de caractères une série aléatoire de **nLetters** lettres majuscules contenant exactement **nVowels** voyelles et (**nLetters**-**nVowels**) consonnes. Utilisez les constantes VOWELS et CONSONANTS. [2,5p]
6. La méthode **canComposeWord** retourne *true* si et seulement si le mot **word** peut être composé en utilisant uniquement les lettres données dans **letters**. Chaque lettre donnée dans **letters** ne peut être employée qu'une seule fois. La recherche doit se terminer dès que la réponse est connue (sans effectuer de *break*, *return* ou similaire dans une boucle). [5p]

7. La méthode **isValid** retourne *true* si et seulement si **word** se trouve dans le dictionnaire et peut être composé par les lettres données dans **letters** selon les règles du jeu. [0,25p]
8. La méthode **findLongestWords** retourne une liste avec les mots les plus longs du dictionnaire qui peuvent être composés par les caractères dans **letters**. Si le mot le plus long comporte **n** caractères, alors tous les mots de longueur **n** qui peuvent être composées par les caractères dans **letters** sont retournés dans la liste.
   Principe de base : Parcourez les mots du dictionnaire et vérifiez que vous pouvez les composer avec les caractères de **letters**. Profitez de la structure du dictionnaire et optimisez la recherche pour limiter le temps de recherche au strict nécessaire.
   Ne pas effectuer de *break*, *return* ou similaire dans une boucle. [4,5p]

MainFrame [10 points]

1. Au démarrage, chargez les mots contenus dans le fichier "**MotsFrancais.txt**" dans le dictionnaire du jeu. Au cas d'une exception, affichez un dialogue d'erreur avec un message explicatif (`JOptionPane.showMessage(...)`) et la description de l'erreur, puis fermez le programme. [1,5p]
2. Si la valeur de **lettersSlider** est modifiée, alors adaptez le maximum de **vowelsSlider** à cette valeur et mettez la valeur de **vowelsSlider** à la moitié de **lettersSlider** (-> voir modèle de démonstration). [0,5p]
3. Lors d'un clic sur **generateLettersButton**, utilisez les valeurs des deux glissières pour générer une série de lettres et affichez cette série de lettres dans **lettersLabel**. [0,5p]
   Le code déjà présent dans cette méthode de réaction sert à réinitialiser l'interface graphique. Entre autres, le bouton **findLongestWordsButton** est désactivé.
4. Réalisez la partie active du jeu [4,5p] :
   - Le compte à rebours de 30 secondes doit démarrer dès que la série de lettres a été générée.
   - A la fin du compte à rebours :
     - **findLongestWordsButton** est réactivé. Le mot entré par le joueur est converti et affiché dans le libellé.
     - Il est vérifié si le mot est acceptable selon les règles du jeu. Un message correspondant est affiché, ainsi que la longueur du mot s'il est accepté (voir modèle). La couleur du libellé (`setForeground(...)`) est vert foncé si accepté, sinon rouge foncé.
5. Lors d'un clic sur **findLongestWordsButton**, l'ordinateur recherche la liste des mots les plus longs et les affiche dans **wordList**. Le temps de recherche est affiché en millisecondes dans **timeLabel** (-> voir modèle). La longueur des mots les plus longs (si trouvés) est affichée dans **maxWorldLengthLabel**. [3p]

[UML -> voir page suivante]


words
WordNode
word : String
next : WordNode
+ WordNode(word : String, next : WordNode)
WordList
– first : WordNode
+ addToStart(newWord : String) : void
+ printAll() : void
+ size() : int
+ contains(word : String) : boolean
+ iterator() : Iterator<String>
Dictionary
– arWordLists : WordList[]
+ Dictionary(size : int)
+ add(word : String) : void
+ get(i : int) : WordList
+ printSizes() : void
+ printAll() : void
+ contains(word : String) : boolean
game
Game
– VOWELS : String
– CONSONANTS : String
– dictionary : Dictionary
+ dictionaryContains(word : String) : boolean
+ convertWord(word : String) : String
+ loadWordList(fileName : String) : void
+ generateLetters(nLetters : int, nVowels : int) : String
+ canComposeWord(word : String, letters : String) : boolean
+ isValid(word : String, letters : String) : boolean
+ findLongestWords(letters : String) : ArrayList<String>
gui
MainFrame

Question 2 - Spinning Fractal [18,5 points]

Le programme **SpinningFractal** dessine une fractale et permet de modifier ses paramètres par des glissières. A la fin l'angle peut aussi être modifié automatiquement par une minuterie (*timer*).

Ouvrez d'abord le programme de démonstration **SpinningFractal.jar** pour vous familiariser avec le programme.

Ouvrez le projet **SpiningFractal** en NetBeans et complétez-le d'après les indications ci-dessous.

- Intégrez d'abord la classe **SpinningFractal** dans **MainFrame** et **DrawPanel** pour pouvoir vérifier et tester que le dessin (p.ex. d'une ligne) fonctionne. Ajoutez pour cela aussi une méthode **draw** à **SpinningFractal**. Le fond du dessin est blanc et le dessin commencera toujours au centre de **drawPanel**. Effectuez ici un cast de **g** en **Graphics2D** pour pouvoir profiter des avantages de **Graphics2D** dans les méthodes de **SpinningFractal.** [1,5p]
- Ajoutez à **SpinningFractal** les attributs et manipulateurs *(Setters)* dont vous avez besoin pour que les glissières fonctionnent. Réalisez les modifications pour que les caractéristiques de la fractale puissent être réalisées à partir des glissières. (Décommentez pour cela les parties correspondantes dans **updateView**). [1p]
- Ajoutez à **SpinningFractal** la méthode **drawVector** qui sert à dessiner des segments (vecteurs) à l'écran (sur un canevas **g**). Paramètres de la méthode **drawVector** : [2p]

Graphics2D g	le canevas cible
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

Notre fractale peut être définie comme suit :

A partir d'un point partent **n (nBranches)** branches de longueur **length.** A la pointe de chaque branche se trouve récursivement une autre fractale. A chaque niveau récursif, la longueur des branches est réduite du facteur **shrinkingFactor** (exprimé en pourcentage). Des fractales dont la longueur des branches est inférieure à 4 points ne sont plus dessinées.
Les angles entre les branches sont tels que les branches sont réparties régulièrement dans un cercle. P.ex. pour n=4, l'angle entre les branches est 90 degrés.
Chaque fractale est tournée de la moitié de cet angle par rapport à la branche sur laquelle elle est placée. P.ex. pour n=4, la fractale est tournée de 45 degrés par rapport à la branche sur laquelle elle se situe.
L'angle **spinAngle** définit l'angle de la fractale de base par rapport à l'angle 0. L'angle de 0 degrés pointe horizontalement vers la droite. Si on augmente l'angle, on tourne dans le sens des aiguilles d'une montre. **spinAngle** est aussi ajouté à l'angle de chaque fractale par rapport à la branche sur laquelle elle se situe. (Ainsi à chaque niveau de récursivité, les branches tourneront plus vite si on change **spinAngle**. P.ex. si la branche au niveau 1 fait un tour, une branche au niveau 3 en fait 3.)

Utilisez le programme de démonstration avec les valeurs *Shrinking Factor* = 40 *et N. branches* = 1, 2, 3 ou 4 avec les *angles* de 0, 45, 60, 90 degrés pour vérifier ou clarifier les explications ci-dessus. Regardez aussi les impressions d'écran à la fin de l'énoncé.

- Définir une procédure privée récursive **drawFractal** qui sait dessiner des fractales comme décrites ci-dessus. Choisissez les paramètres de façon appropriée et servez-vous de la procédure **drawVector**. Adaptez aussi la méthode **draw** de **SpinningFractal** [4,5p]

Modifiez **drawVector** comme suit [1,75p] :

- L'épaisseur des branches (lignes) est d'un dixième de leur longueur.
  Utilisez : `g.setStroke(new BasicStroke( float width ), BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND);`
- Les couleurs sont définies comme suit :

longueur de la ligne en pixels	couleur
4 ... 8	magenta
9 ... 16	bleu
17 ... 32	cyan
33 ... 64	vert
> 64	vert foncé

Modifiez le programme comme suit :

- Une minuterie (*javax.swing.Timer*) permet de faire augmenter automatiquement l'angle **spinAngle** de la fractale. La minuterie sera définie dans la classe **SpinningFractal**. Lorsque la minuterie est active, l'angle est augmenté de 1 degré 50 fois par seconde. [1,75p]
- Utilisez le schéma **Observer** pour informer **drawPanel** de chaque modification d'un attribut dans **SpinningFractal**. La classe **MainFrame** effectue tous les changements directement dans **spinningFractal** et ne joue aucun rôle dans le dessin ou le rafraichissement du dessin de la fractale. (Il n'y a donc aucun appel à **repaint** dans **MainFrame**). [3,75p]
- La minuterie peut être activée et désactivée par la case à cocher **spin**. La glissière **spinAngleSlider** est active uniquement si la case à cocher **spin** est désactivée. [2,25p]