Contrôle 3: Informatique 12 avril 2017

Cours de mathématiques spéciales Semestre d'été

(écrire lisiblement et avec au maximum trois couleurs différentes s.v.p.)

Nom : ……...

Prénom : ……..….

Question	Barème	Points
1	65	
2	67	
3	68	
Total	200	

Note :

Indications générales

- Durée de l'examen : **105 minutes**.
- Posez votre **carte d'étudiant** sur la table.
- La réponse à chaque question doit être rédigée **à l'encre** sur la place réservée à cet effet à la suite de la question.
  Si la place prévue ne suffit pas, vous pouvez demander des feuilles supplémentaires aux surveillants ; chaque feuille supplémentaire doit porter **nom, prénom, n° du contrôle, branche et date**. Elle ne peut être utilisée que pour **une seule question**.
- Les feuilles de brouillon ne sont pas à rendre ; elles ne seront **pas corrigées** ; des feuilles de brouillon supplémentaires peuvent être demandées en cas de besoin auprès des surveillants.
- Les feuilles d'examen doivent être rendues **agrafées**.

Indications spécifiques

- Toutes les questions qui suivent se réfèrent au langage de programmation **Java** (à partir du **JDK 8.0**).
- A part les polycopiés du cours, **aucune** autre source bibliographique n'est autorisée et ne doit donc être consultée durant le contrôle.

Remarques initiales :

- Il est conseillé de lire chaque question jusqu'à la fin, avant de commencer la rédaction de la solution.
- N'oubliez pas d'importer les éléments dont vous avez besoin (et qui ne font pas partie du package ***java.lang***).
- Suite à une instruction "***return***", une méthode (qui retourne un résultat ou qui est de type ***void***) est quittée "immédiatement" (mais après l'exécution d'un éventuel bloc ***finally***).
- Implémenter le traitement d'une certaine exception revient normalement à utiliser un bloc ***catch*** approprié placé à la suite d'un bloc ***try*** contenant (au moins) la (ou les) instruction(s) "à risque".
- Les indices des éléments des tableaux commencent à zéro.
- N'oubliez pas les éventuels **casts** obligatoires.
- Le texte associé à un bouton swing peut être lu à l'aide de la méthode ***getText*** et attribué ou modifié à l'aide de la méthode ***setText***.

Question 1

Soit le fichier ***CP_Ctr3Exo1.java*** contenant le code source suivant :

```
package cms_ctr3;

class TropException extends Exception {
	int age;		String boisson;
	TropException(int age, String boisson) {
		super("Ca ne va pas ");
		this.age = age;		this.boisson = boisson;
	}
}

class TropJeuneException extends TropException {
	TropJeuneException(int age, String boisson) {
		super(age, boisson);
		System.out.println(getMessage() + "car trop jeune !");
	}
}

class TropVieuxException extends TropException {
	TropVieuxException(int age, String boisson) {
		super(age, boisson);
		System.out.println(getMessage() + "pour les sages !");
	}
}
```

```
class P {
	String nom;
	int age;

	P(String nom, int age) {
		this.nom = nom;
		this.age = age;
	}//fin du constructeur

	String boire(String boisson) throws TropException {
		try {
			if(age < 16 && !boisson.equals("du lait") &&
					!boisson.equals("de l'eau")) {
				throw new TropJeuneException(age, boisson);
			}
			if(age < 18 && (boisson.equals("du vin") ||
					boisson.equals("du cognac"))) {
				throw new TropJeuneException(age, boisson);
			}
			if(age > 70 && boisson.equals("du cognac")) {
				throw new TropVieuxException(age, boisson);
			}
		}
		catch(TropJeuneException tje) {
			System.out.println(tje.boisson.toUpperCase() + " à " +
					tje.age + " ans ?");
			if(tje.age < 14) {
				return "Désolé ! On peut vous servir du sirop !";
			}
			throw tje;
		} finally {
			System.out.println("Eau & lait = meilleures boissons !");
		}
		return nom + " boit " + boisson + ".";
	}//fin de boire
}//fin de la classe P
```

```
public class CP_Ctr3Exo1 {
	public static void main(String[] args) throws TropException {
		P[] personnes = { new P("Loïc", 1),
				  new P("Pline l'Ancien", 75),
				  new P("Gargantua", 10),
				  new P("Minor", 17),
				  new P("Zorro", 30)};
		String[] boissons = {"du lait", "du cognac", "du vin",
					"du vin", "du cognac"};
		for(int i = 0; i < personnes.length; i++) {
			try {
				System.out.println(personnes[i].boire(boissons[i]));
				System.out.println("Bravo " + personnes[i].nom +
								" !");
			}catch(TropJeuneException tje) {
				System.out.println("Vous êtes vraiment jeune !");
				if(tje.age >= 16)
					System.out.println("A la limite, prenez " +
								"une bière !");
			}catch(TropVieuxException tve) {
				System.out.println("Désolé ! On vous recommande " +
								"un verre de vin !");
			}
			System.out.println("**********************************");
		}

		System.out.println("L'abus d'alcool nuit gravement " +
								"à la santé !");
	}//fin de main
}//fin de la classe principale
```

Précisez à la page suivante les messages qui seront affichés à l'écran suite à l'exécution du projet correspondant au code source indiqué ci-dessus.

Eau & lait = meilleures boissons !

Loïc boit du lait.

Bravo Loïc !

****

Ca ne va pas pour les sages !

Eau & lait = meilleures boissons !

Désolé ! On vous recommande un verre de vin !

****

Ca ne va pas car trop jeune !

DU VIN à 10 ans ?

Eau & lait = meilleures boissons !

Désolé ! On peut vous servir du sirop !

Bravo Gargantua !

****

Ca ne va pas car trop jeune !

DU VIN à 17 ans ?

Eau & lait = meilleures boissons !

Vous êtes vraiment jeune !

A la limite, prenez une bière !

****

Eau & lait = meilleures boissons !

Zorro boit du cognac.

Bravo Zorro !

****

L'abus d'alcool nuit gravement à la santé !

Question 2

Le but de cet exercice est de concevoir une méthode qui permet au programmeur de connaître le mot qui se trouve à une certaine position (numérotée à partir de 1) sur une certaine ligne (numérotée à partir de 1) dans un fichier texte donné. Dans cet exercice, les mots représentent des simples sous-chaînes de caractères séparées par les "espaces blanches" habituelles.

Vous devez écrire le code source d'une classe publique appelé ***Auxiliaire*** qui se trouve dans un package nommé ***cms_ctr3***.
Cette classe contient une seule méthode publique nommée ***lire*** et qui peut être appelée sans avoir besoin de créer d'instance de la classe ***Auxiliaire*** (voir des exemples d'appels plus loin).

La méthode ***lire*** a trois arguments :

- le premier argument nommé ***nomFichier*** est de type chaîne de caractères et doit correspondre normalement au nom d'un fichier texte qui existe et qui est accessible en lecture ;
- le deuxième argument nommé ***noLigne*** est de type numérique entier et doit identifier normalement une certaine ligne du fichier texte mentionné ci-dessus ; les lignes ne sont pas numérotées dans le fichier texte et c'est la méthode ***lire*** qui doit les compter (la première ligne ayant le numéro 1) ;
- le troisième argument nommé ***noMot*** est de type numérique entier et doit identifier normalement un certain mot qui se trouve sur la ligne mentionnée ci-dessus ; les mots ne sont pas numérotés dans le fichier texte et c'est la méthode ***lire*** qui doit les compter (le premier mot ayant le numéro 1).

La méthode ***lire*** retourne un résultat de type chaîne de caractères qui correspond normalement au ***noMot***-ième mot qui se trouve sur la ***noLigne***-ième ligne du fichier texte ***nomFichier***. Cependant, la méthode ***lire*** doit traiter aussi les "cas spéciaux", en utilisant éventuellement les traitements de plusieurs exceptions prédéfinies. Par conséquent, vous devez respecter les consignes suivantes :

- si le premier argument de la méthode correspond à la valeur spéciale ***null*** :
  - on affiche dans la fenêtre console le message **Pas de fichier !** ;
  - on retourne la valeur spéciale ***null*** ;
- (autrement) on déclare une variable locale de type chaîne de caractères appelée ***extension*** ;
- grâce à la méthode ***substring*** de la classe ***String***, on stocke dans la variable locale ***extension*** les derniers quatre caractères de la chaîne ***nomFichier*** et, si l'appel de la

méthode ***substring*** lance une exception ***IndexOutOfBoundsException***, on retourne la chaîne de caractères **Pas de bon nom de fichier !** ;

- (autrement, c'est-à-dire si une extension de quatre caractères a pu être extraite) si la chaîne de caractères ***extension*** ne correspond pas à la chaîne littérale ***.txt***, on retourne la chaîne de caractères **Pas de bonne extension !** ;
- (autrement, c'est-à-dire si le nom du fichier a la bonne extension) on crée un objet ayant comme type une classe filtre appropriée pour la lecture du fichier texte indiqué par le premier argument et, si cette instruction lance une exception de type ***FileNotFoundException***, la méthode retourne la chaîne de caractères **Fichier non trouvé !** ;
- (autrement, c'est-à-dire si la création de l'objet filtre s'est bien passée) on parcourt et on lit le fichier texte indiqué par le premier argument ligne après ligne :
  - si on arrive à la ***noLigne***-ième ligne du fichier texte :
    - on tokenize convenablement cette ligne ;
    - si la ligne n'a pas suffisamment de mots (par rapport à la valeur du troisième argument de la méthode), on retourne la chaîne de caractères **Ligne trop courte !** ;
    - autrement, on retourne la chaîne de caractères **Le mot recherché est "XXX" !**, où **XXX** est le mot qui correspond à la position (sur la ligne tokenisée) indiquée comme troisième argument de la méthode ;
- (autrement, c'est-à-dire si on arrive à la fin du ficher texte lu et qu'il a moins de lignes que la valeur indiquée comme deuxième argument de la méthode), on retourne la chaîne de caractères **Pas assez de lignes !** .

A titre d'exemple, on considère un projet contenant la classe ***Auxiliaire*** et la classe principale ***CP_Ctr3Exo2*** dont le corps est donné ci-dessous.

```
package cms_ctr3;
import java.io.*;
public class CP_Ctr3Exo2 {
	public static void main(String[] args) throws IOException {
		System.out.println(Auxiliaire.lire("declaration.txt", 2, 4));
		System.out.println(Auxiliaire.lire("declaration.txt", 3, 1));
		System.out.println(Auxiliaire.lire("declaration.txt", 4, 6));
		System.out.println(Auxiliaire.lire("declaration.txt", 5, 1));
	}
}
```

De plus, dans le dossier du projet se trouve le fichier texte nommé **declaration.txt** et qui a le contenu suivant :

```
Tous les êtres humains naissent libres et égaux en
dignité et en droits. Ils sont doués de raison et de
conscience et doivent agir les uns envers les autres
dans un esprit de fraternité.
```

Les messages affichés dans la fenêtre console suite à l'exécution du projet mentionné ci-dessus sont les suivants :

```
Le mot recherché est : "droits." !
Le mot recherché est : "conscience" !
Ligne trop courte!
Pas assez de lignes !
```

Ecrivez ci-dessous le code complet de la classe ***Auxiliaire***.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

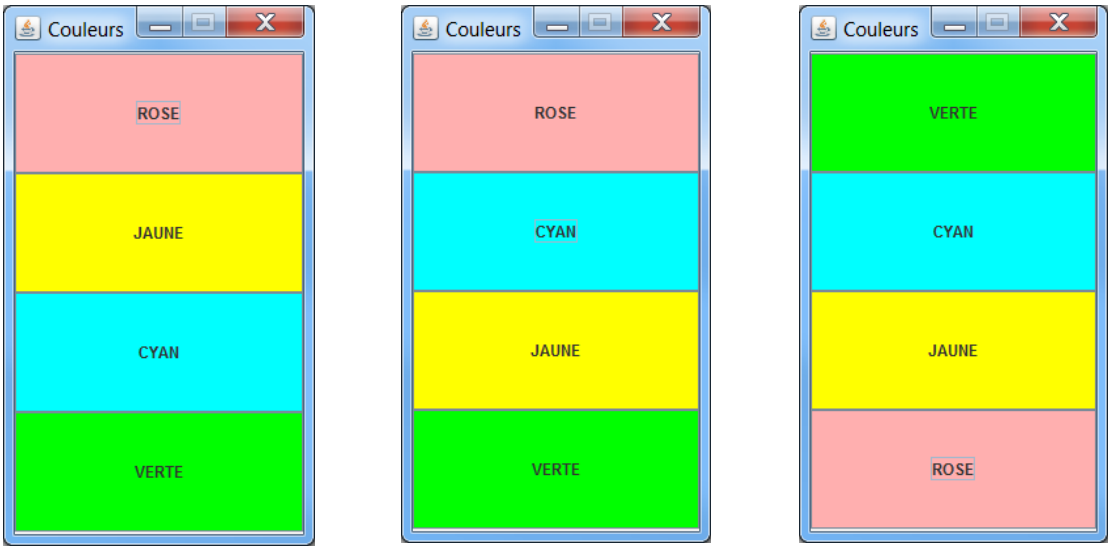
```
package cms_ctr3;

import java.io.*;
import java.util.StringTokenizer;

public class Auxiliaire
{
     public static String lire(String nomFichier,
               int noLigne, int noMot) throws IOException
     {
          if(nomFichier == null)
          {
               System.out.println("Pas de fichier!");
               return null;
          }
          String extension ;
          try {
               extension =
                nomFichier.substring(nomFichier.length()-4);
          }catch(IndexOutOfBoundsException exp)
          {
               return "Pas de bon nom de fichier !";
          }
          if(!extension.equals(".txt")) {
               return "Pas de bonne extension !";
          }
          BufferedReader br;
          try
          {
               br = new BufferedReader(new
                              FileReader(nomFichier));
          }catch(FileNotFoundException fnfe)
          {
               return "Fichier non trouvé !";
          }
          String ligne;
          int noLi = 0;
```

```
        while(true) {
            ligne = br.readLine();
            if(ligne == null) {
                break;
            }
            noLi++;
            if(noLi == noLigne)
            {
                StringTokenizer st = new
                            StringTokenizer(ligne);
                int nbTokens = st.countTokens();
                if(nbTokens < noMot)
                {
                    br.close();
                    return "Ligne trop courte!";
                }
                for(int i=0; i<noMot-1; i++)
                {
                    st.nextToken();
                }
                br.close();
                return "Le mot recherché est : \"" +
                            st.nextToken() + "\" !";
            }
        }//fin de la boucle while
        br.close();
        return "Pas assez de lignes ! ";
    }//fin de la méthode lire
}//fin de la classe principale
```

Question 3

Le but de cet exercice est de permettre à l'utilisateur d'interagir avec une interface graphique (GUI) qui sera décrite par la suite.

Plus précisément, au début de l'exécution du projet à réaliser, une interface graphique comme celle présentée dans la **Figure 3.1.a)** doit apparaître dans le coin supérieur gauche de l'écran. Cette GUI correspond à un container de premier niveau muni de quatre boutons swing. Les textes initiaux associés aux quatre boutons correspondent aux couleurs initiales de ces boutons, à savoir : rose, jaune, cyan et verte.



a) **b)** **c)**

Figure 3.1

Quand l'utilisateur appuie (avec la souris) sur un des trois premiers boutons (comptés du haut vers le bas), ce bouton échange la couleur et le texte associé avec le bouton placé en-dessous de lui. Par contre, quand l'utilisateur appuie sur le quatrième bouton (placé tout en bas), ce bouton échange la couleur et le texte associé avec le premier bouton (placé tout en haut).

La **Figure 3.1.b)** montre l'interface graphique obtenue à partir de celle présentée dans la **Figure 3.1.a)** si l'on clique avec la souris sur le deuxième bouton (ayant la couleur jaune avant le clic).

La **Figure 3.1.c)** montre l'interface graphique obtenue à partir de celle présentée dans la **Figure 3.1.b)** si l'on clique avec la souris sur le quatrième bouton (ayant la couleur verte avant le clic).

On considère un projet Java qui implémente les consignes mentionnées ci-dessus et qui est muni d'un package nommé ***cms_ctr3*** contenant deux classes publiques :

- la classe graphique ***Fenestra*** qui correspond à l'interface graphique décrite auparavant;

- la classe principale ***CP_Ctr3Exo3*** dont le code source est donné ci-dessous et qui contient la méthode publique et statique ***main*** (qui crée une instance de la classe graphique ***Fenestra***).

```
package cms_ctr3;
public class CP_Ctr3Exo3 {
	public static void main(String[] args) {
		new Fenestra();
	}
}
```

Dans ce troisième exercice, on vous demande d'écrire le code source de la classe graphique ***Fenestra*** en fonction des indications données ci-dessous.

3.1 La classe ***Fenestra*** est publique, appartient au package ***cms_ctr3*** et correspond à un container de premier niveau (top level container). De plus, elle est à l'écoute des quatre boutons placés dans son **contentPane**.

Le corps de la classe ***Fenestra*** contient trois champs nommés ***boutons***, ***couleurs*** et ***textes***, un constructeur, une méthode appelée ***getIndice*** et un gestionnaire d'événements approprié.

Le champ ***boutons*** est privé et sert à garder les adresses des quatre boutons qui sont placés (du haut vers le bas) dans le container de premier niveau. Il est de type tableau de boutons swing et doit être initialisé avec l'adresse d'un nouveau tableau de quatre éléments créé pour l'occasion.
Le champ ***couleurs*** est privé et sert à garder les adresses des couleurs initiales des quatre boutons (dans l'ordre du haut vers le bas). Il est de type tableau de couleurs et doit être initialisé avec l'adresse d'un nouveau tableau de quatre éléments créé pour l'occasion et qui a lui-même comme éléments initiaux les couleurs rose (en anglais, pink), jaune (en anglais, yellow), cyan (en anglais, cyan) et verte (en anglais, green).
Le champ ***textes*** est privé et sert à garder les textes initiaux associés aux quatre boutons (dans l'ordre du haut vers le bas). Il est de type tableau de chaînes de caractères et doit être initialisé avec l'adresse d'un nouveau tableau de quatre éléments créé pour l'occasion et qui a lui-même comme éléments initiaux les chaînes de caractères ***ROSE***, ***JAUNE***, ***CYAN*** et ***VERTE***.

Ecrivez ci-dessous le début du code source de la classe graphique ***Fenestra*** selon les consignes données (en vous arrêtant juste avant le début de la définition du constructeur de la classe).

```
package cms_ctr3;

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class Fenestra extends JFrame
                      implements ActionListener
{
     private JButton[] boutons = new JButton[4];
     private Color[] couleurs = {Color.PINK, Color.YELLOW,
                                 Color.CYAN, Color.GREEN};
     private String[] textes = {"ROSE", "JAUNE",
                                "CYAN", "VERTE"};
```

3.2 Le **constructeur** de la classe graphique est public, n'a pas d'argument et doit réaliser le design de l'interface graphique. De plus, il doit permettre la communication entre la fenêtre et ses quatre boutons. Plus précisément, dans ce constructeur vous devez :

- prévoir une instruction qui impose l'arrêt de la machine virtuelle suite à la fermeture du container de premier niveau, container appelé par la suite tout simplement la fenêtre ;
- donner le titre ***Couleurs*** à la fenêtre ;
- fixer la taille de la fenêtre à 225 pixels x 400 pixels ;
- rendre la fenêtre non redimensionnable ;
- associer à la fenêtre un nouveau gestionnaire de mise en forme créé pour l'occasion et correspondant à une grille avec quatre lignes et une seule colonne ;
- à l'aide d'une boucle appropriée :
    - créer chaque bouton swing et stocker son adresse dans le "bon" élément du tableau ***boutons*** (le premier indice du tableau doit correspondre au bouton placé tout en haut de la fenêtre et le dernier indice du tableau au bouton placé tout en bas de la fenêtre) ;
    - associer à chaque bouton la couleur correspondant à l'élément de même indice du tableau ***couleurs*** ;
    - associer à chaque bouton le texte correspondant à l'élément de même indice du tableau ***textes*** ;
    - mettre chaque bouton sous l'écoute de la fenêtre ;
    - ajouter chaque bouton à la fenêtre ;
- rendre la fenêtre visible.

La méthode ***getIndice*** est privée et doit permettre de savoir si l'adresse d'un certain bouton indiqué comme argument de la méthode se trouve parmi les éléments du tableau ***boutons*** et, si c'est le cas, quel est l'indice qui lui correspond. Par conséquent, cette méthode a un seul argument de type bouton swing nommé ***jbt*** et retourne un résultat de type numérique entier. Plus précisément, la méthode ***getIndice*** :

- doit vérifier, à l'aide d'une boucle appropriée, si l'adresse ***jbt*** correspond à un des éléments du tableau ***boutons***, cas où la méthode retourne immédiatement l'indice de cet élément ;
- autrement (c'est-à-dire si aucune correspondance n'est trouvée), la méthode doit retourner la valeur "spéciale" ***-1***.

Ecrivez ci-dessous les définitions (c'est-à-dire les en-têtes et les corps) du constructeur de la classe graphique et de la méthode ***getIndice***.

```
public Fenestra()
{
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     setTitle("Couleurs");
     setSize(225, 400);
     setResizable(false);

     setLayout(new GridLayout(4,1));

     for(int i=0; i<4; i++) {
          boutons[i] = new JButton();
          boutons[i].setBackground(couleurs[i]);
          boutons[i].setText(textes[i]);
          boutons[i].addActionListener(this);
          add(boutons[i]);
     }
     setVisible(true);
}//fin du constructeur


private int getIndice(JButton jbt) {
     for(int i=0; i<boutons.length; i++) {
          if(jbt == boutons[i]) {
               return i;
          }
     }
     return -1;
}//fin de la méthode getPosition
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

3.3 Le gestionnaire d'événements approprié doit rendre les quatre boutons sensibles aux clics de la souris afin d'assurer la réactivité de l'interface graphique. Plus précisément, la méthode gestionnaire d'événements que vous devez (re)définir doit respecter les consignes suivantes :

- si la source de l'événement traité par le gestionnaire est un objet dont le type n'est pas celui d'un bouton swing (ou compatible par polymorphisme), la méthode ne doit faire aucun traitement spécifique (et elle peut être quittée immédiatement) ;
- (autrement, c'est-à-dire si la source de l'événement traité est bien un bouton swing mais) si l'adresse de la source de l'événement traité par le gestionnaire ne se retrouve pas parmi les éléments du tableau ***boutons*** (et vous pouvez obtenir cette information grâce à la méthode ***getIndice***), la méthode ne doit faire plus aucun traitement spécifique (et elle peut être quittée immédiatement) ;
- (autrement, c'est-à-dire si la source de l'événement traité par le gestionnaire est bien un des quatre boutons) le bouton source de l'événement traité (et dont l'indice correspondant dans le tableau ***boutons*** a été obtenu grâce à la méthode ***getIndice***) doit échanger la couleur et le texte associé avec le "bon" bouton parmi les trois autres boutons, comme expliqué au début de cette **Question 3**.

Vous devez (re)définir le gestionnaire d'événements approprié ci-dessous.

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

```
    @Override
    public void actionPerformed(ActionEvent ae)
    {
        if(!(ae.getSource() instanceof JButton)) {
            return;
        }

        JButton btSource = (JButton)ae.getSource();
        int pos = getIndice(btSource);

        if(pos == -1) {
            return;
        }

        JButton btEnBas = boutons[(pos+1) % 4];

        Color coulSource = btSource.getBackground();
        String texteSource = btSource.getText();
        Color coulEnBas = btEnBas.getBackground();
        String texteEnBas = btEnBas.getText();

        btSource.setBackground(coulEnBas);
        btSource.setText(texteEnBas);
        btEnBas.setBackground(coulSource);
        btEnBas.setText(texteSource);
    }//fin du gestionnaire d'événements
}//fin de la classe graphique
```

A part ce texte, cette page doit rester normalement blanche (mais vous pouvez l'utiliser pour la rédaction de vos réponses si la place prévue à cet effet s'est avérée insuffisante).