

Contrôle 4: Informatique

Cours de mathématiques spéciales

13 juin 2018

Semestre de printemps

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

Nom :

Prénom :

SOLUTION

Question	Barème	Points
1	118	
2	40	
3	42	
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 **SDK 8.0**).
- A part les 2 photocopies 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.
- Suite à une instruction "**return**", une méthode (qui retourne un résultat ou qui est de type **void**) est quittée "immédiatement".
- Afin de manipuler des images correspondant à des "fichiers d'image" (avec des extensions comme **.jpg**, **.gif** ou **.png**), l'API Java propose l'interface (swing) **Icon** qui est implémentée, par exemple, dans la classe concrète (swing) **ImageIcon**.
- La classe **Button** dispose de deux méthodes permettant de manipuler une image associée à un bouton, à savoir :
 - la méthode d'instance publique "**setter**" nommée **setIcon** qui ne retourne pas de résultat et qui a un (seul) argument de type **Icon** qui précise l'adresse d'un objet correspondant à l'image qui sera affichée par le bouton ;
 - la méthode d'instance publique "**getter**" nommée **getIcon** qui n'a pas d'argument et qui retourne un résultat de type **Icon** qui correspond à l'image affichée par le bouton.

Question 1

Le but de cet exercice est de créer une interface graphique (GUI) permettant à un utilisateur (étudiant) de sélectionner un cours à option parmi plusieurs choix proposés. Plus précisément, au début de l'exécution du projet à réaliser, une interface graphique comme celle présentée dans la **Figure 1** doit apparaître dans le coin gauche supérieur de l'écran. Cette GUI correspond à un conteneur de premier niveau muni de trois boutons "poussoir" ("push buttons" en anglais) qui affichent au départ les noms des trois cours à option et des images associées. Par la suite, on considère que le premier bouton est celui de gauche (celui avec le texte associé **AdM** et l'image représentant Augustin-Louis Cauchy), le deuxième celui du milieu (celui avec le texte associé **GD** et l'image représentant Gaspard Monge) et le troisième (qui est aussi le dernier bouton) est celui de droite (celui avec le texte associé **Info** et l'image représentant Alan Turing). Cependant, afin de permettre seulement le choix du troisième cours, vous devez truquer cette interface graphique de la manière suivante :

- si le pointeur (ou le curseur) de la souris entre dans la zone de représentation du premier bouton (voir la **Figure 2**) ou du deuxième bouton (voir **Figure 3**), le bouton respectif doit échanger son texte et l'image associée avec le dernier bouton ;
- si le pointeur de la souris sort de la zone de représentation du premier ou du deuxième bouton, le bouton respectif doit afficher à nouveau son texte et son image de départ ;



Figure 1



Figure 2



Figure 3



Figure 4

- si le pointeur de la souris est dans la zone de représentation du dernier bouton, les trois boutons doivent afficher les textes et les images de départ (voir Figure 4) ;
- si le pointeur de la souris n'est pas dans la zone de représentation d'un bouton, les trois boutons doivent afficher les textes et les images de départ (voir Figure 1) ;
- si l'utilisateur appuie pour la première fois sur un des trois boutons et si le texte associé au bouton appuyé est bien **Info**, les trois boutons ne seront plus affichés et le container de premier niveau affichera en rouge le message **Info est le meilleur choix !** (voir la Figure 5).



Figure 5

On considère un projet Java qui implémente les consignes mentionnées ci-dessus et qui est muni d'un package nommé *cms_citr4* contenant deux classes publiques :

- la classe graphique *Choix* qui correspond à l'interface graphique décrite auparavant ;
- la classe principale *CP_Citr4Exo1* dont le code source n'est pas demandé et qui contient

la méthode publique et statique *main* qui crée une instance de la classe graphique *Choix*.

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

1.1 La classe *Choix* est publique, appartient au package *cms_citr4* et correspond à un container de premier niveau ("top level container" en anglais) qui écoute les (trois) boutons qui y sont placés.

A part plusieurs champs privés, cette classe contient un constructeur public sans argument, un gestionnaire qui gère les événements (sémantiques) lancés par des clics sur les boutons et la définition d'une classe interne et privée nommée *BigBrother* qui permet la création d'un écouteur (non graphique) qui gère les événements (de bas niveau) liés à la souris.

Plus précisément, vous devez d'abord définir les champs d'instance privés suivants :

- le champ *noms* de type tableau de chaînes de caractères **Adm**, **GD** et **Info** ;

- le champ *images* de type tableau d'objets correspondant à des images et initialisé avec trois nouveaux objets créés pour l'occasion et qui correspondent aux fichiers d'images **im1.jpg**, **im2.jpg** et **im3.jpg** (qui se trouvent dans le dossier du projet en cours) ;
- le champ *boutons* de type tableau de boutons swing et initialisé avec l'adresse d'un nouveau tableau de 3 boutons créé pour l'occasion ;
- le champ *police* de type police de caractères et initialisé avec l'adresse d'un nouvel objet créé pour l'occasion et correspondant à une police **Arial**, grasse ("bold" en anglais) et de taille 48 pixels ;
- le champ *hb* de type classe interne *BigBrother* et initialisé avec l'adresse d'un nouvel objet écouteur *BigBrother* créé pour l'occasion (afin de réagir aux événements de bas niveau lancés par la souris) ;
- le champ *jpChoix* de type container intermédiaire et initialisé avec l'adresse d'un nouvel objet container intermédiaire créé pour l'occasion ;
- le champ *nomCourant* de type chaîne de caractères, sans valeur initiale explicite et qui permettra de stocker le texte associé au bouton sur lequel se trouve le pointeur de la souris ;
- le champ *imageCourante* dont le type lui permet de correspondre à une image, sans valeur initiale explicite et qui permettra de stocker l'adresse de l'objet qui correspond à l'image associée au bouton sur lequel se trouve le pointeur de la souris.

Dans la définition du constructeur public sans argument, vous devez :

- remplacer le gestionnaire de mise en forme ("Layout Manager" en anglais) par défaut du container intermédiaire avec un nouveau gestionnaire de mise en forme approprié pour y ajouter les trois boutons sur une même (et unique) ligne ;
- à l'aide d'une boucle appropriée, effectuer les opérations suivantes pour chacun des trois boutons :
 - o créer le nouveau bouton avec le "bon nom" (grâce au tableau *nom*s) et la "bonne image" (grâce au tableau *images*) associés, et ajouter le au tableau *boutons* ;
 - o proposer une taille de (280 pixels x 450 pixels) pour le bouton ;
 - o associer au bouton la police stockée dans le champ prévu à cet effet ;
 - o indiquer que le texte associé au bouton doit apparaître au-dessus de l'image associée au bouton ;
 - o indiquer que le texte associé au bouton doit être centré horizontalement ;
 - o demander au bouton de transmettre les éventuels événements de bas niveau reçus de la souris à l'écouteur *hb* prévu à cet effet ;

- o associer au bouton le conteneur de premier niveau comme écouteur (des événements sémantiques produits quand le bouton est appuyé) ;
 - o ajouter le bouton au container intermédiaire ;
- ajouter le container intermédiaire au container de premier niveau ;
- donner le titre *Choix Option* au container de premier niveau ;
- appeler une méthode qui assure que la taille du container de premier niveau s'adapte (à l'exécution et de manière optimale) à son contenu ;
- rendre le container de premier niveau non redimensionnable ;
- rendre le container de premier niveau visible ;
- prévoir une instruction qui impose l'arrêt de la machine virtuelle suite à la fermeture du container de premier niveau.

Selon les consignes données, écrivez ci-dessous le début du code source de la classe graphique *Choix* (notamment les champs et le constructeur de la classe).

```
//Déclaration du package

package cms_ctr4;

//Importations des packages nécessaires

import java.awt.*;

import java.awt.event.*;

import javax.swing.*;

//En-tête de la classe englobante Choix

public class ChoixFin extends JFrame

implements ActionListener

{
```

```
//Déclarations des champs privés (de la classe englobante Choix)

private String[] noms = {"Adm", "GD", "Info"};

private ImageIcon[] images =

    {
        new ImageIcon("im1.jpg"),
        new ImageIcon("im2.jpg"),
        new ImageIcon("im3.jpg")};

private JButton[] boutons = new JButton[3];

private Font police = new Font("Arial",
                                Font.BOLD, 48);

private BigBrother bb = new BigBrother();

private JPanel jpChoix = new JPanel();

private String nomCourant;

private ImageIcon imageCourante;
```

```
//Définition (en-tête et corps) du constructeur
//(de la classe englobante Choix)

public ChoixFin(){

    jpChoix.setLayout(new GridLayout(1, 3));

    for(int i=0; i<3; i++) {

        boutons[i] = new JButton(noms[i], images[i]);
        boutons[i].setPreferredSize(
                                new Dimension(280, 450));

        boutons[i].setFont(police);

        boutons[i].setVerticalTextPosition(
                                SwingConstants.TOP);

        boutons[i].setHorizontalTextPosition(
                                SwingConstants.CENTER);

        boutons[i].addMouseListener(bb);

        boutons[i].addActionListener(this);

        jpChoix.add(boutons[i]);

    }

    add(jpChoix);

    setTitle("Choix Option");

    pack();

    setResizable(false);

    setVisible(true);

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

} //fin du constructeur
```

1.2 Comme déjà mentionné, le container de premier niveau doit écouter et gérer les événements (sémantiques) produits quand l'utilisateur final de la GUI clique sur les boutons.

Par conséquent, la classe graphique **Choix** a dû implémenter la "bonne" interface et vous devez maintenant (re)définir (à l'intérieur de cette classe) le gestionnaire d'événements approprié.

Par conséquent, dans le corps de ce gestionnaire vous devez procéder ainsi :

- créer une variable locale nommée **source** ayant un type approprié et y stocker l'adresse du bouton appuyé ;
- si le texte associé au bouton appuyé correspond bien à la constante littérale chaîne de caractères **Info** :
 - o à l'aide d'une boucle appropriée, enlever les trois boutons du container intermédiaire ;
 - o créer une variable locale de type étiquette swing nommée **message** et y stocker l'adresse d'un nouvel objet créé pour l'occasion et correspondant à une étiquette dont le texte affiché est la constante littérale chaîne de caractères **Info est le meilleur choix** ! ;
 - o proposer une taille de (610 pixels x 200 pixels) pour l'étiquette ;
 - o associer à l'étiquette la police stockée dans le champ prévu à cet effet ;
 - o assurer que le texte de l'étiquette s'affiche en **rouge** ;
 - o ajouter l'étiquette au panneau intermédiaire ;
 - o appeler (à nouveau) la méthode qui assure que la taille du container de premier niveau s'adapte (à l'exécution et de manière optimale) à son contenu.

Ecrivez à la page suivante la (re)définition complète (c'est-à-dire l'en-tête et le corps) du gestionnaire des événements (sémantiques) déclenchés par les boutons.

```

@Override

public void actionPerformed(ActionEvent ae) {

    JButton source = (JButton)ae.getSource();

    //if(source.getText() == "Info") { //JUSTE

        if(source.getText().equals("Info")) {

            for(int i=0; i<3; i++) {

                jpChoix.remove(boutons[i]);

            }

            JLabel message = new JLabel(

                " Info est le meilleur !" );

            message.setPreferredSize(

                new Dimension(610, 200));

            message.setFont(police);

            message.setForeground(Color.RED);

            jpChoix.add(message);

            pack();

        }

    }

}

//fin du gestionnaire actionPerformed

```

11

1.3 La classe interne *BigBrother* est privée et permet la création du champ *bb* (de la classe englobante) afin d'écouter les événements de bas niveau liés aux actions de la souris associés aux trois boutons. Par conséquent, cette classe ne contient que les (re)définitions des gestionnaires d'événements.

Dans la (re)définition du "bon" gestionnaire qui traite les événements de bas niveau déclenchés à l'entrée du pointeur de la souris dans la zone de représentation graphique d'un bouton, vous devez procéder ainsi :

- si la source de l'événement est le troisième bouton, quitter immédiatement la méthode ;
- (autrement) créer une variable locale nommée *source* ayant un type approprié et y stocker l'adresse du bouton qui a produit l'événement traité (c'est-à-dire le bouton sur la surface duquel le pointeur de la souris est entré) ;
- stocker dans le champ *nomCourant* (de la classe englobante) le texte associé au bouton source ;
- stocker dans le champ *imageCourante* (de la classe englobante) l'adresse de l'objet qui correspond à l'image associée au bouton source ;
- donner au texte associé au bouton source la valeur du dernier élément du champ *noms* (de la classe englobante) ;
- donner comme image associée au bouton source l'image correspondant au dernier élément du champ *images* (de la classe englobante) ;
- donner au texte associé au troisième bouton la valeur stockée dans le champ *nomCourant* (de la classe englobante) ;
- donner comme image associée au troisième bouton l'image correspondant au champ *imageCourante* (de la classe englobante).

Dans la (re)définition du "bon" gestionnaire qui traite les événements de bas niveau déclenchés à la sortie du pointeur de la souris de la zone de représentation graphique d'un bouton, vous devez procéder ainsi :

- si la source de l'événement est le troisième bouton, quitter immédiatement la méthode ;
- (autrement) créer une variable locale nommée *source* ayant un type approprié et y stocker l'adresse du bouton qui a produit l'événement traité (c'est-à-dire le bouton dont la surface a été quittée par le pointeur de la souris) ;
- donner au texte associé au bouton source la valeur stockée dans le champ *nomCourant* (de la classe englobante) ;
- donner comme image associée au bouton source l'image correspondant au champ *imageCourante* (de la classe englobante) ;

12

- Ecrivez ci-dessous la définition de la classe interne *BigBrother* (c'est-à-dire son en-tête et son corps contenant les (re)définitions de tous les gestionnaires d'événements).

This image shows a full page of white paper with vertical dotted lines. The lines are evenly spaced and run from the top to the bottom of the page, providing a guide for writing or drawing. There are no margins, text, or other markings on the page.

```
private class BigBrother implements MouseListener {

    public void mouseClicked( MouseEvent me ) {

        if(me.getSource() == boutons[2]) {

            return;

        }

        JButton source = (JButton)me.getSource();

        nomCourant = source.getText();

        imageCourante = (ImageIcon)source.getIcon();

        source.setText(noms[2]);

        source.setIcon(images[2]);

        boutons[2].setText(nomCourant);

        boutons[2].setIcon(imageCourante);

    } //fin du gestionnaire mouseClicked

    public void mouseExited( MouseEvent me ) {

        if(me.getSource() == boutons[2]) {

            return;

        }

        JButton source = (JButton)me.getSource();

        source.setText(nomCourant);

        source.setIcon(imageCourante);

        boutons[2].setText(noms[2]);

        boutons[2].setIcon(images[2]);

    } //fin du gestionnaire MouseExited

    @Override public void mouseClicked(MouseEvent me) { }

    @Override public void mousePressed(MouseEvent me) { }

    @Override public void mouseReleased(MouseEvent me) { }

} //fin de la classe interne BigBrother
```



```

package cms_ctr4;

import java.util.ArrayList;

public class Personne<T> implements FI_Predicat<T> {
    public String nom = "Anonymus";
    public T id = null;

    public Personne(String nom, T id) {
        this.nom = nom;
        this.id = id;
    }

    public Personne(Personne<T> p) {
        this.nom = p.nom;
        this.id = p.id;
    }

    @Override
    public boolean tester(T t) {
        if(id == t) {
            return true;
        }
        return false;
    }

    public static <U> U getIdFromList(
        ArrayList<Personne<U>> liste, int i) {
        return liste.get(i).id;
    }
}

```

17

Question 3

Soit le code source suivant :

```

package cms_ctr4;

import java.util.ArrayList;

class Fruit { }
class Prune extends Fruit { }
class Nectarine extends Prune { }

public class CP_Ctr4Exo3 {
    public static void main(String[] args) {
        boolean bool;
        Fruit fruit;
        Prune prune;
        Nectarine nectarine;

        ArrayList<Fruit> fruits = new ArrayList<>();
        ArrayList<Prune> prunes = new ArrayList<>();
        ArrayList<Nectarine> nectarines = new ArrayList<>();

        ArrayList<? extends Prune> jokeExt = null;
        ArrayList<? super Prune> jokeSup = null;

        //ci-dessous on ajoutera (à tour de rôle) une seule des
        //instructions proposées par la suite

    } //fin de la méthode main
} //fin de la classe principale

```

Chacune des instructions suivantes est ajoutée, à tour de rôle, à la fin de la méthode *main*. Encerchez le numéro placé à droite de chaque instruction qui ajoutée seule à la fin de la méthode *main* produit une erreur à la compilation.
Approximativement la moitié de ces instructions produit des erreurs à la compilation.

18

```

prunes.add(new Fruit());
prunes.add(new Prune());
prunes.add(new Nectarine());
prunes = fruits;
prunes = nectarines;
jokExt = new ArrayList<? extends Prune>();
jokExt = fruits;
jokExt = prunes;
jokSup = nectarines;
jokSup = fruits;
jokSup = prunes;
jokExt = nectarines;
bool = jokExt.add(new Fruit());
bool = jokExt.add(new Prune());
bool = jokExt.add(new Nectarine());
fruit = jokExt.remove(0);
prune = jokExt.remove(0);
nectarine = jokExt.remove(0);
bool = jokExt.remove(new String("hello"));
bool = jokSup.add(new Fruit());
bool = jokSup.add(new Prune());
bool = jokSup.add(new Nectarine());
prune = jokSup.remove(0);
bool = jokSup.remove(new String("hello"));

```

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).