EPF – Lausanne *Cours Polymaths*

11 juin 2009

Contrôle de Méthode de calcul numérique

Durée : 1 heure 45'

Nom : …… Groupe : ☐

Prénom : ……

Barème sur 90 points

No	1	2	3
Total points	30 points	30 points	30 points

Remarque générale : Toutes les questions qui suivent se réfèrent au langage de programmation Java (à partir du JDK 5.0) et les réponses doivent être rédigées à l'encre et d'une manière propre sur ces feuilles agrafées.

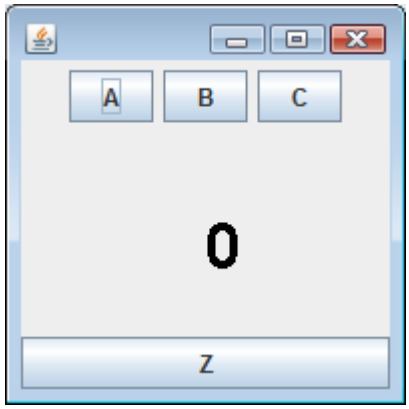
Sujet no 1.

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

```
package cms_ctr4;

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

public class JFFenestra extends JFrame
{
      char car;
      //ci-dessous, on crée un tableau bidimensionnel de char
      //et on stocke son adresse dans le champ (la variable) tab
      char tab[][] = { {'0','0','0'}, {'1','2','3'},
                       {'4','5','6'}, {'7','8','9'} };
      int i = 0, j = 0;
      JButton jbTab[ ] = new JButton[3];
      JButton jbSud;
      String textes = "ABC";
      ClasseInterne ci = new ClasseInterne( );
      JPanel jpan = new JPanel( );
```

```
	public JFFenestra( )
	{
		setSize(200, 200);

		for(int k=0; k<3; k++)
		{
			jbTab[k] = new JButton(textes.substring(k, k+1));
			jbTab[k].addActionListener(ci);
			jpan.add(jbTab[k]);
		}

		add(jpan, BorderLayout.NORTH);

		jbSud = new JButton("Z");
		jbSud.addActionListener(ci);
		add(jbSud, BorderLayout.SOUTH);

		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setVisible(true);
	}//fin du constructeur

	@Override
	public void paint(Graphics g)
	{
		super.paint(g);
		g.setFont(new Font("Arial", Font.BOLD, 32));
		g.drawString(tab[i][j] + "", this.getSize().width/2,
					this.getSize().height/3*2);
	}//fin de la méthode paint

	class ClasseInterne implements ActionListener
	{
		@Override
		public void actionPerformed(ActionEvent e)
		{	car = ((JButton)e.getSource()).getText().charAt(0);
			switch(car)
			{
				//Attention aux opérateurs de préincrémentation
				case 'A' : i = ++i % 4;
					break;

				case 'B' : j = ++j % 3;
					break;

				case 'C' : repaint( );
					break;

				case 'Z' : i = 0; j = 0;
			}//fin de switch

		}//fin de la méthode actionPerformed

	}//fin de la classe interne

}//fin de la classe graphique englobante
```

Suite à l'instanciation de la classe **JFFenestra** qui vient d'être présentée, l'utilisateur dispose d'une interface graphique (GUI) comme celle montrée dans la figure ci-dessous :



A, **B**, **C** et **Z** désignent, par la suite, les **boutons** dont les textes associés sont, respectivement, **A**, **B**, **C** et **Z**.

1.1 Indiquer quel sera le "**chiffre final**" affiché dans la zone centrale de la fenêtre graphique si, juste après l'affichage de la GUI, l'utilisateur clique, dans l'ordre, sur les boutons suivants :

a) A → A

.....

b) A → A → C

.....

c) B → B

.....

d) B → B → C

.....

e) A → B → A → C

.....

f) A → B → A → C →A →C

.....

g) A → B → Z → B → A → C → Z

.....

h) A → A → C

.....

1.2 Indiquer l'ordre du plus simple enchaînement des clics sur des boutons afin d'afficher dans la zone centrale de la fenêtre graphique le "**chiffre final**" :

a) 1

.....

b) 3

.....

c) 6

.....

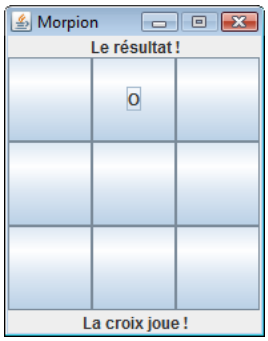
d) 9

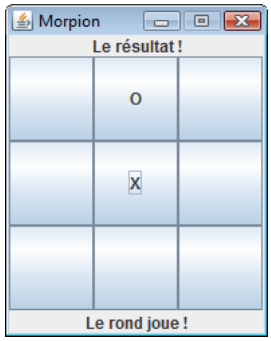
.....

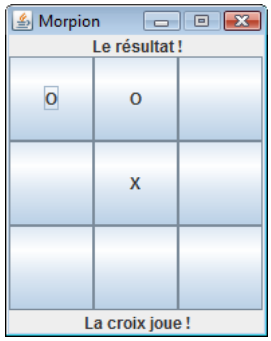
Sujet no 2.

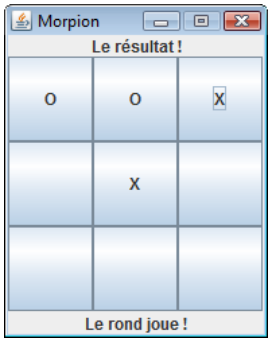
Le but de cet exercice est d'écrire une classe graphique Java qui permet de jouer de manière "naïve" (sans aucune stratégie implémentée) au jeu **Morpion**. A l'instanciation, cette classe doit afficher une fenêtre comme celle présentée ci-dessous.

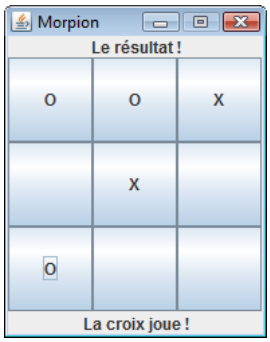
Dans les images suivantes, on montre à titre d'exemple le début d'un jeu **Morpion**, en utilisant l'interface graphique qui doit être réalisée.

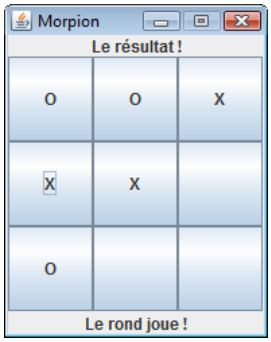












Plus précisément, il faut compléter le code source présenté plus loin, en respectant les indications données en commentaire ainsi que les consignes suivantes :

- la classe graphique :
  - fait partie du package ***cms_ctr4*** ;
  - est publique et s'appelle ***JFJeu*** ;
  - correspond à un **container de premier niveau** (et elle doit dériver d'une classe prédéfinie appropriée) ;

    - peut **écouter des événements** de type ***ActionEvent*** (et elle doit implémenter une interface prédéfinie appropriée) ;
- dans la région "nord" de la fenêtre principale se trouve une **étiquette** dont le texte associé reste inchangé, à savoir : "**Le résultat !**" ;
- dans la région "sud" de la fenêtre principale se trouve une **étiquette** dont le texte initial est "**Le rond joue !**" ;
- dans la zone "centrale" de la fenêtre principale se trouve un "**panneau**" qui est muni d'une "**grille**" avec 9 **boutons** qui, initialement, n'affichent aucun texte ;
- quand l'utilisateur clique pour la première fois sur un **bouton**, le **bouton** poussé doit afficher le texte "**O**" et l'**étiquette** placée au sud doit changer le texte affiché en "**La croix joue !**" ;
- quand l'utilisateur clique ensuite sur un autre bouton, le **bouton** poussé doit afficher le texte "**X**" et l'**étiquette** sud doit changer le texte affiché en "**Le rond joue !**" ;
- par la suite, les deux dernières étapes se succèdent à chaque clic sur un "nouveau" **bouton**, mais les clics sur les **boutons** qui affichent déjà un texte doivent rester sans effet.

Afin d'obtenir le fonctionnement décrit ci-dessus, vous pouvez :

- inscrire la fenêtre graphique en tant qu'écouteur auprès des 9 boutons ;
- redéfinir la méthode qui sera appelée automatiquement chaque fois que l'utilisateur clique sur un bouton.

En fait, il suffit de prévoir dans la classe graphique un champ de type primitif booléen appelé **rond** initialisé à "vrai" et de redéfinir la méthode gestionnaire d'événements selon la marche à suivre suivante :

- récupérer (avec le bon type) la source de l'événement ***ActionEvent*** produit ;
- si le texte correspondant au bouton appuyé est la chaîne vide :
    - si la valeur du champ **rond** est "vrai" :
        - mettez le texte du **bouton** appuyé à "**O**" ;
        - changez la valeur du champ **rond** ;
        - mettez le texte de l'**étiquette** sud à "**La croix joue !**";
    - autrement :
        - mettez le texte du **bouton** appuyé à "**X**" ;
        - changez la valeur du champ **rond** ;
        - mettez le texte de l'**étiquette** sud à "**Le rond joue !**".

Complétez ci-dessous le contenu du fichier source ***JFJeu.java*** qui définit la classe ***JFJeu.***

```
//Déclarez le package

……

//Importez les packages prédéfinis nécessaires

……

……

……

……

//Précisez l'en-tête complet de la classe graphique JFJeu

……

{//début du corps de la classe
      //Déclarez un champ de type "étiquette" nommé jlScore

……

      //Déclarez un champ de type "étiquette" nommé jlJoueur

……

      //Déclarez un champ de type "panneau" nommé jpanTable

……

      //Déclarez un champ de type primitif booléen nommé rond
      //et initialisé à "vrai"

……

      //Déclarez un champ de type "bouton" nommé jbCourant

……
```

```
      //Précisez l'en-tête du constructeur public de la classe JFJeu
```

……

```
      {//début du constructeur
            //Créez un objet "étiquette" affichant le texte "Le résultat !"
            //et stockez son adresse dans le champ jlScore
```

……

```
            //Rien à modifier ou compléter
            jlScore.setHorizontalAlignment(SwingConstants.CENTER);

            //Créez un objet "étiquette" avec le texte "Le rond joue !"
            //et stockez son adresse dans le champ jlJoueur
```

……

```
            //Rien à modifier ou compléter
            jlJoueur.setHorizontalAlignment(SwingConstants.CENTER);

            //Créez un objet "panneau" et stockez son adresse
            //dans le champ jpanTable
```

……

```
            //Associez au panneau jpanTable un nouveau gestionnaire de mise
            //en forme de type "grille" de 3 x 3
```

……

```
            //Indiquez, pour le "panneau" jpanTable, une taille préférée de
            //180 pixels x 180 pixels
```

……

```
            //Précisez l'en-tête d'une boucle for qui s'exécutera 9 fois
```

……

```
            {//début de la boucle for
                  //Créez un nouvel objet de type "bouton" et stockez son
                  //adresse dans le champ jbCourant
```

```
……

                //Inscrivez la fenêtre graphique (qui sera créée avec ce
                //constructeur)comme écouteur des événements ActionEvent
                //auprès du "bouton" jbCourant

……

                //Ajoutez le "bouton" jbCourant au "panneau" jpanTable

……

            }//fin de la boucle for

            //Ajoutez "l'étiquette" jlScore dans la région "nord" de la
            //fenêtre graphique

……

            //Ajoutez le "panneau" jpanTable dans la région "centrale" de
            //la fenêtre graphique

……

            //Ajoutez "l'étiquette" jlJoueur dans la région "sud" de la
            //fenêtre graphique

……

            //Ecrire une instruction qui va donner à la fenêtre graphique
            //une taille "idéale" et minimale par rapport à son contenu

……

            //Indiquer "Morpion" comme titre de la fenêtre graphique

……

            //Rendez la fenêtre graphique non redimensionnable

……
```

```
            //Rendez la fenêtre graphique visible
```

……

```
            //Ecrire une instruction qui va arrêter la machine virtuelle
            //Java à la fermeture de la fenêtre graphique
```

……

```
      }//fin du constructeur

      @Override
      //Précisez l'en-tête et le corps de la méthode qui va s'exécuter
      //automatiquement chaque fois que l'utilisateur clique sur un des
      //9 boutons affichés dans la fenêtre graphique
```

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

}//fin de la classe graphique JFJeu

Sujet no 3.

Le but de cet exercice est d'écrire une application Java autonome qui extrait des informations à partir d'un **fichier texte** au format **HTML**.

Plus précisément, on dispose d'un **fichier texte** appelé "***page.html***" correspondant à une page Web qui sera disponible sur un serveur et pourra être visitée à l'aide d'un navigateur (browser). Ce fichier peut contenir des balises (tags) **APPLET** ayant, éventuellement, une ou plusieurs balises ***PARAM***.
L'application Java qui sera réalisée doit :

- lire le **fichier texte** appelé "***page.html***" qui est stocké dans le dossier ***Sources*** de la partition ***F:*** ;
- déterminer le nombre de balises **APPLET** et extraire le nom qualifié du fichier ***.class*** contenant le bytecode de chaque applet, ainsi que les noms des éventuels paramètres associés à chaque applet ;
- écrire dans un **fichier texte** appelé "***resultats.txt***" (qui sera créé pour l'occasion sur la partition ***F:*** et dans un dossier déjà existant ***Programmation***) :
    - sur la première ligne, le nombre d'applets référencés par la page Web ;
    - sur des lignes distinctes, le nom du fichier ***.class*** qui désigne le bytecode de chaque applet, ainsi que les noms de ses éventuels paramètres.

On peut consulter plus loin l'exemple d'un fichier texte "***page.html***" et du fichier texte "***resultats.txt***" obtenu suite à l'exécution de l'application Java.

Le projet à concevoir est formé d'une seule classe publique appelée ***CP_Ctr4Exo3***, placée dans un package nommé ***cms_ctr4***.

La classe ***CP_Ctr4Exo3*** contient la méthode ***main*** qui réalise la succession des opérations principales suivantes :

- on crée une connexion **en lecture** vers le fichier texte "***page.html***" déjà existant dans le dossier ***Sources*** de la partition ***F:*** ;
- on fait une première lecture de ce fichier et on détermine le nombre d'applets qu'il mentionne, en comptant, par exemple, le nombre de balises **APPLET** (ouvrantes ou fermantes) ;
- on ferme la connexion entre le programme Java et le fichier texte "***page.html***" ***;***

- si le nombre d'applets référencés est zéro, on affiche un message approprié dans la fenêtre console (par exemple : ***Pas de balise APPLET dans le fichier HTML !***) et on arrête le programme (par un appel à la méthode ***exit*** avec un code de retour ***-1***) ;
- autrement, le programme continue son exécution comme indiqué par la suite ;
- on crée une nouvelle connexion **en lecture** vers le même fichier texte "***page.html***" ;
- on crée une connexion **en écriture** vers un fichier texte "***resultats.txt***" (qui sera créé dans le dossier ***Programmation*** de la partition ***F:***) ;
- on écrit sur la première ligne du fichier "***resultats.txt***" un message indiquant le nombre d'applets référencés dans la page Web ;
- on lit le fichier texte "***page.html***" ligne après ligne et, au fur et à mesure qu'on identifie les informations recherchées, à savoir le nom du fichier ***.class*** contenant le bytecode de l'applet (correspondant à l'attribut ***CODE*** de la balise ***APPLET***) et les noms des éventuels paramètres (correspondant à l'attribut ***NAME*** de la balise ***PARAM***), on les écrit sur des lignes distinctes dans le fichier texte "***resultats.txt***" ;
- finalement, on ferme les connexions entre le programme Java et les fichiers textes "***resultats.txt***" et "***page.html***".

Indications :

- on compte sur une mise en page du fichier "***page.html***" comme celle donnée comme exemple plus loin ;
- on ne traite que les lignes non vides du fichier "***page.html***" ;
- après chaque lecture d'une ligne non vide du fichier "***page.html***", on élimine d'abord les éventuels espaces ou tabulations du début ou de la fin de la ligne (par un appel à la méthode **trim**) ;
- puis, afin de trouver des informations pertinentes sur la ligne, on peut utiliser la classe ***StringTokenizer*** en précisant de**s** caractère**s** séparateur**s** adéquat**s** dans la chaîne de caractères passée comme deuxième argument au constructeur de cette classe ;
- le programme Java doit fonctionner correctement si les noms de balises et les mots clés sont écrits avec des lettres minuscules ou majuscules dans le fichier texte "***page.html***" ;
- dans le fichier texte "***resultats.txt***", les noms des fichiers correspondant au bytecode des applets doivent apparaître sans guillemets, tandis que les noms des paramètres doivent apparaître entre des guillemets.

Exemple d'un fichier texte "***page.html***" :

```
<html>

	<head>
		<title>Contrôle 4</title>
	</head>

	<body>
		<hr>
		<table>
			<tr>
				La première applet
				<APPLET  codebase="..\bin"  CODE="cms.JAUno.class"
					width=175 height=150
					<PARAM NAME=toto value="Valeur toto">
				</APPLET>
			</tr>
			<hr>
			<tr>
				La deuxième applet
				<APPLET  codebase="..\bin"  CODE="cms.JADuo.class"
					width=175 height=150>
				</APPLET>
			</tr>
			<hr>
			<tr>
				La troisième applet
				<APPLET  codebase="..\bin"  CODE="cms.JATrio.class"
					width=175 height=150   name="applet_2">
					<PARAM NAME=code_1 value="Valeur 1">
					<PARAM NAME=code_2 value="Valur 2">
				</APPLET>
			</tr>
		</table>
		<hr>
	</body>

</html>
```

Exemple du fichier texte "***resultats.txt***" obtenu à partir du fichier texte "***page.html***" suite à l'exécution de l'application autonome :

```
Il y a 3 applets !
Le bytecode : cms.JAUno.class.
Un paramètre de nom "toto".
Le bytecode : cms.JADuo.class.
Le bytecode : cms.JATrio.class.
Un paramètre de nom "code_1".
Un paramètre de nom "code_2".
```

Ecrivez ci-dessous le contenu du fichier ***CP_Ctr4Exo3.java*** qui définit la classe ***CP_Ctr4Exo3.***

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

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....