EPF – Lausanne *COURS DE MATHEMATIQUES SPECIALES*

30 mai 2012

Contrôle d'informatique no 4

Durée : 1 heure 45'

Nom : ……. Groupe :

Prénom : …….

No sujet	**1**	**2**	**3**
Nombre points	28 points	32 points	40 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.

Remarques initiales :

- Il est conseillé de lire chaque sujet jusqu'à la fin, avant de commencer la rédaction de la solution.
- N'oubliez pas d'importer les packages et les classes standards dont vous avez besoin.
- Au deuxième sujet, faites une bonne utilisation des méthodes ***print*** et ***println***.

1. Soit le fichier ***CP_Ctr4Exo1.java*** contenant le code source suivant :

```
package cms_ctr4;

class EgalExp extends Exception { }

class DefaiteExp extends Exception
{     int diff;
      public DefaiteExp(int diff)
      {     super("Quelle tragédie !");
            this.diff = diff;
      }
}

class AbsurdeExp extends RuntimeException { }

public class CP_Ctr4Exo1
{     static void interpreter(int tab[]) throws DefaiteExp
      {
            try
            {     if(tab[0]<0 || tab[1]<0)
                  {
                        throw new AbsurdeExp();
                  }
                  else if(tab[0] > tab[1])
                  {
                        System.out.println("Victoire !!!");
                  }else if(tab[0] == tab[1])
                  {
                        throw new EgalExp();
                  }else if(tab[0] < tab[1])
                  {
                        throw new DefaiteExp(tab[1]-tab[0]);
                  }
            }
```

```
            catch(EgalExp e1)
            {     System.out.println("Tout le monde d'accord !");
            }catch(DefaiteExp e2)
            {     if(e2.diff <= 2)
                        System.out.println("Comme c'est dur !");
                  else
                        throw e2;
            }finally
            {     System.out.println("Toujours présent !");
            }
            System.out.println("Fin d'appel !");
      }//fin de la méthode interpreter

      public static void main(String[] args)
      {
            int score[][] = {{2,2}, {-1, 0}, {1,2}, {3,0}, {0,3}};

            for(int i=0; i<score.length; i++)
            {     try
                  {     interpreter(score[i]);
                  }catch(AbsurdeExp e3)
                  {     System.out.println("Pas question !");
                  }catch(DefaiteExp e4)
                  {     System.out.println(e4.getMessage());
                  }finally
                  {     System.out.println("Game over !");
                  }
            }//fin de la boucle for
      }//fin de la méthode main
}//fin de la classe principale
```

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

2. Dans le sujet numéro 2, on vous demande d'écrire une application Java autonome qui lit un fichier texte donné, extrait certaines informations à partir du contenu lu et crée un nouveau fichier texte dans lequel elle écrit ces informations accompagnées éventuellement des commentaires (voir les exemples qui suivent).

Plus précisément :

- le fichier lu par le programme est un fichier texte appelé ***poeme.txt*** et il se trouve dans le dossier ***Litterature*** d'une partition Windows ***C:*** ; ce fichier a le contenu suivant garanti :
  - une première ligne qui indique l'auteur d'un poème ;
  - une deuxième ligne vide;
  - une troisième ligne qui précise le titre du poème ;
  - une quatrième ligne vide ;
  - les vers du poème qui tiennent chacun sur une ligne et qui sont groupés en strophes séparées par des lignes vides ;
  - chaque vers finit par un mot seul ou, éventuellement, suivi directement (c'est-à-dire sans espace) par un signe de ponctuation parmi les caractères suivants : virgule, point, point-virgule ou point d'exclamation ;
- le fichier écrit par le programme est un fichier texte appelé ***rimes.txt*** et il doit être placé au même endroit (c'est-à-dire dans le même dossier) que le fichier lu ; ce nouveau fichier doit respecter les consignes suivantes :
  - la première ligne a cette forme :

    Les rimes du poème "XXX" écrit par YYY :

    où ***XXX*** est le nom du poème (indiqué entre des guillemets) et ***YYY*** est l'auteur du poème ;
  - le nombre de lignes qui suivent est égal au nombre de strophes du poème lu ;
  - chaque ligne doit correspondre à une strophe et doit contenir autant de mots (séparés par des espaces) que de vers dans cette strophe ;
  - chaque nième mot d'une telle ligne est en fait le dernier mot du nième vers de la strophe correspondant à la ligne, sans l'éventuel signe de ponctuation qui termine le vers.

On donne ci-dessous, à titre d'exemple, le contenu d'un fichier texte ***poeme.txt*** qui doit être lu.

```
Charles BAUDELAIRE

L'albatros

Souvent, pour s'amuser, les hommes d'équipage
Prennent des albatros, vastes oiseaux des mers,
Qui suivent, indolents compagnons de voyage,
Le navire glissant sur les gouffres amers.

A peine les ont-ils déposés sur les planches,
Que ces rois de l'azur, maladroits et honteux,
Laissent piteusement leurs grandes ailes blanches
Comme des avirons traîner à côté d'eux.

Ce voyageur ailé, comme il est gauche et veule!
Lui, naguère si beau, qu'il est comique et laid!
L'un agace son bec avec un brûle-gueule,
L'autre mime, en boitant, l'infirme qui volait!

Le Poète est semblable au prince des nuées
Qui hante la tempête et se rit de l'archer;
Exilé sur le sol au milieu des huées,
Ses ailes de géant l'empêchent de marcher.
```

Il convient de préciser que, dans ce fichier texte à lire :

- on ne connaît pas le nombre de vers ni le nombre de strophes ;
- les lignes vides ne contiennent pas de caractères "blancs" ou "invisibles" (comme des espaces ou des tabulations) ;
- les lignes non vides ne commencent et ne finissent pas par des caractères "blancs" ou "invisibles".

Suite à l'exécution du programme que vous devez concevoir, un nouveau fichier texte ***rimes.txt*** sera créé et il aura le contenu suivant :

```
Les rimes du poème "L'albatros" écrit par Charles BAUDELAIRE :
d'équipage mers voyage amers
planches honteux blanches d'eux
veule laid brûle-gueule volait
nuées l'archer huées marcher
```

En conclusion, il s'agit d'un projet Java muni d'un package nommé ***cms_ctr4*** et qui contient une seule classe (principale), publique et appelée ***CP_Ctr4Exo2***. Dans la méthode ***main*** de cette classe, le programme doit lire, ligne après ligne, le fichier texte ***poeme.txt*** et écrire, au fur et à mesure, le nouveau fichier texte ***rimes.txt***.

Indications :

- chaque ligne lue et qui correspond à un vers dans le fichier ***poeme.txt*** devra être séparée en tokens adéquats (par une opération de "tokenisation" en fonction d'un séparateur convenablement choisi) ;
- ensuite, le nombre de tokens sera déterminé et les tokens seront lus successivement ;
- finalement, le dernier token lu à partir d'une telle ligne sera écrit dans le fichier ***rimes.txt*** :
    - sans le dernier caractère, si ce caractère est un des signes de ponctuation mentionnés auparavant ;
    - tel quel, si le dernier token représente un mot seul (qui n'est pas suivi par un signe de ponctuation) ;
- à part la deuxième et la quatrième ligne du fichier ***poeme.txt***, toutes les autres lignes vides séparent deux strophes et leur présence dans le fichier lu ***poeme.txt*** impose le passage à la ligne dans le fichier ***rimes.txt*** généré par le programme.

Vous devez écrire ci-dessous le contenu du fichier ***CP_Ctr4Exo2.java***, à l'aide des précisions données en commentaire. La solution que vous allez proposer doit utiliser le mécanisme de "tokenisation".

```
//définition du package
.....
.....

//importation des packages nécessaires (de l'API Java)
.....
.....
.....
.....
.....

//définir l'en-tête de la classe principale
.....
.....
.....

{
```

```
    //définir l'en-tête de la méthode main
……
……
……

    {
        //définir une variable locale ayant le type approprié et
        //qui permet de lire le fichier texte poeme.txt contenant
        //le poème à traiter
……
……
……
……

        //définir une variable locale ayant le type approprié et
        //qui permet d'écrire dans le fichier texte rimes.txt les
        //rimes utilisées
……
……
……
……

        //définir les variables locales utiles pour la résolution
        //du problème posé
……
……
……
……
……
……
……
……
……
……
……

        //lire successivement les quatre premières lignes du
        //fichier poeme.txt et extraire le titre et l'auteur
        //du poème
```

.....

.....

.....

.....

.....

//écrire la première ligne dans le fichier texte **rimes.txt**

.....

.....

.....

.....

.....

//lire successivement tout le reste du fichier texte
//**poeme.txt** (dont la taille n'est pas connue a priori),
//traiter les informations obtenues et remplir le fichier
//texte **rimes.txt** au fur et à mesure

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
            //"libérer" l'accès aux deux fichiers texte utilisés par
            //le programme pour qu'ils puissent être employés par la
            //suite par d'autres utilisateurs ou applications
```

.....

.....

.....

.....

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

3. On considère un projet Java muni d'un package nommé ***cms_ctr4*** et qui contient deux classes :

- une classe publique appelée ***JFrPrincipale*** dont l'instanciation correspond initialement à l'interface graphique utilisateur (GUI) présentée ci-dessous :

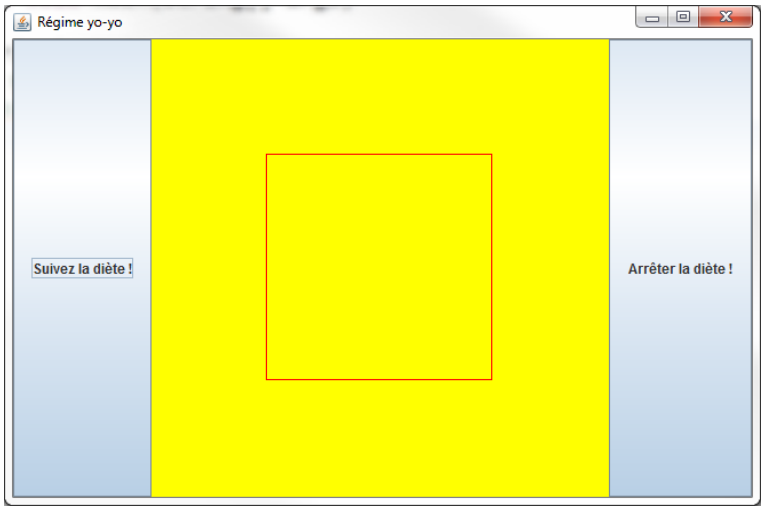

Figure 1

- la classe principale ***CP_Ctr4Exo3*** qui crée une instance de la classe graphique ***JFrPrincipale*** et dont le code source est indiqué ci-dessous.

```
package cms_ctr4;

public class CP_Ctr4Exo3 {
     public static void main(String[] args)
     {
          new JFrPrincipale();
     }//fin de la méthode main
}//fin de la classe principale
```

Dans ce troisième sujet, on vous demande d'écrire le code source de la classe graphique ***JFrPrincipale*** qui fait partie du package ***cms_ctr4***.

Suite à l'instanciation de cette classe ***JFrPrincipale*** par la méthode ***main*** de la classe principale ***CP_Ctr4Exo3***, une fenêtre (un container graphique de premier niveau) est affichée à l'écran (voir la **Figure 1** ci-dessus). On peut y distinguer :

- un bouton poussoir (à cliquer) dont le texte associé est ***Suivez la diète !*** et qui est placé dans la région "ouest" ;
- un bouton poussoir (à cliquer) dont le texte associé est ***Arrêtez la diète !*** et qui est placé dans la région "est" ;

- un panneau de taille 400 x 400 pixels qui affiche (dessine) un carré rouge sur un fond jaune et qui est placé dans la région "centre".

Par la suite, le carré mentionné ci-dessus (et qui est toujours dessiné et redessiné dans le panneau central) sera nommé tout simplement "le carré".

Tout au début, le carré :

- est figé ;
- a son côté initial égal à la moitié du côté du panneau central (voir la **Figure 1**) ;
- a le coin supérieur gauche situé à une distance égale à 100 pixels vers la droite et à 100 pixels vers le bas par rapport au coin supérieur gauche du panneau central.

A l'aide des deux boutons, l'utilisateur peut démarrer ou arrêter une animation durant laquelle la position du coin supérieur gauche du carré ne change pas mais le côté du carré change automatiquement entre une valeur maximale égale à la taille initiale et une valeur minimale correspondant à un côté qui vaut la moitié de la taille maximale (voir la **Figure 2** ci-dessous).

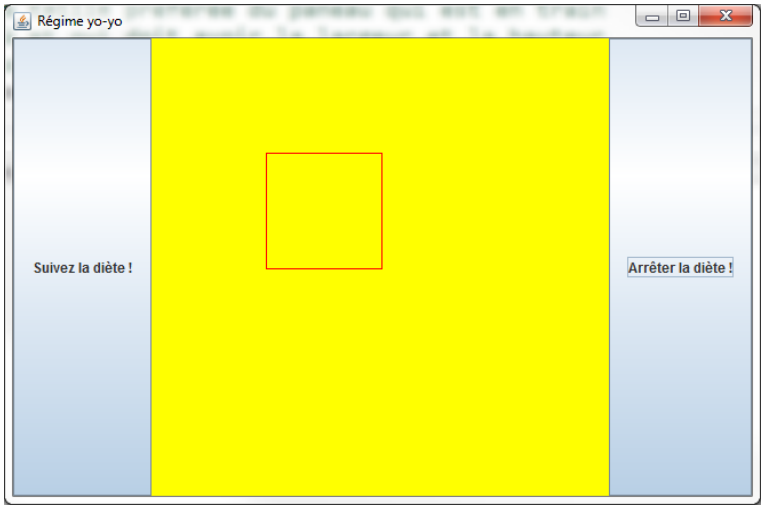

Figure 2

Plus précisément, les deux boutons sont réactifs dans le sens où :

- chaque fois que l'utilisateur appuie sur le bouton placé à l'ouest, l'animation commence (ou, éventuellement, continue) ;
- chaque fois que l'utilisateur appuie sur le bouton placé à l'est, l'animation s'arrête (ou, éventuellement, reste arrêtée).

Afin de réaliser l'animation avec une période donnée (par exemple 10 millisecondes), il faut tenir compte des consignes suivantes :

- si le côté du carré est en train d'augmenter :
    - on calcule la nouvelle valeur du côté comme étant l'ancienne valeur à laquelle on ajoute la valeur d'un pas donné (par exemple 1 pixel) ;
    - si cette nouvelle valeur ne dépasse pas la valeur maximale du côté, on redessine le carré avec la taille augmentée ;
    - autrement, on dessine le carré avec sa taille maximale et "on note" que la "prochaine fois" la taille du carré doit commencer à diminuer ;
- autrement (c'est-à-dire si le côté du carré est en train de diminuer) :
    - on calcule la nouvelle valeur du côté comme étant l'ancienne valeur de laquelle on soustrait la valeur d'un pas donné (par exemple 1 pixel) ;
    - si cette nouvelle valeur ne descend pas au-dessous de la valeur minimale du côté, on redessine le carré avec la taille diminué ;
    - autrement, on dessine le carré avec sa taille minimale et "on note" que la "prochaine fois" la taille du carré doit commencer à augmenter.

La réalisation du design de l'interface graphique présentée dans les **Figures 1** et **2** est détaillée sous forme de commentaires dans le canevas de la classe ***JFrPrincipale*** présenté ci-dessous. Cette classe publique est la fille d'un container graphique swing de premier niveau.

Le panneau graphique central est un conteneur intermédiaire "personnalisé" qui est obtenu par l'instanciation d'une classe interne appelée ***JPanDessin*** et qui dérive de la classe prédéfinie ***JPanel***.

La gestion des événements produits quand l'utilisateur clique avec la souris sur les boutons est assurée par l'instance même de la classe graphique qui implémente une "interface écouteur" appropriée. Par conséquent, la classe graphique doit s'inscrire comme écouteur auprès de ses deux boutons et doit (re)définir la méthode "gestionnaire d'événements" appropriée.

Le changement automatique et continu de la taille du carré entre ses valeurs maximale et minimale est réalisé grâce à un objet Java d'un type adéquat et qui est le "moteur de l'animation". Cet objet envoie périodiquement des événements de type ***ActionEvent*** vers le conteneur de premier niveau qui, grâce à la même méthode "gestionnaire d'événements" évoquée ci-dessus, appelle la méthode qui fait varier la taille du carré.

On présente ci-dessous le squelette de la classe ***JFrPrincipale*** et il faut compléter ce canevas en fonction des indications données en commentaire.

```
//déclaration du package
……
……

//importation des packages nécessaires (de l'API Java)
……
……
……
……
……

//déclaration de l'en-tête de la classe graphique englobante qui est
//publique, hérite d'un container de premier niveau et implémente
//une interface qui lui permet d'écouter les événements produits par
//les boutons swing et par le "moteur de l'animation"
……
……
……

{
     private JButton jbEst;
     private JButton jbOuest;
     private JPanDessin conPan;
     //déclarer un champ sans valeur initiale et ayant le type
     //approprié pour que l'on puisse y stocker l'adresse de l'objet
     //qui sera le "moteur de l'animation"
……
……

     public JFrPrincipale()
     {
          this.setTitle("Régime yo-yo");
          this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          //créer un bouton ayant comme texte associé la chaîne
          //Suivez la diète !
          //et stockez son adresse dans le champ jbOuest
……
……
```

```
            //rendez le bouton créé ci-dessus réactif en lui associant
            // le container de premier niveau en tant qu'écouteur
……
……

            //créer un bouton ayant comme texte associé la chaîne
            //Arrêtez la diète !
            //et stockez son adresse dans le champ jbEst
……
……

            //rendez le bouton créé ci-dessus réactif en lui associant
            //le container de premier niveau en tant qu'écouteur
……
……

            conPan = new JPanDessin();
            this.add(jbEst, BorderLayout.EAST);
            this.add(jbOuest, BorderLayout.WEST);
            this.add(conPan, BorderLayout.CENTER);
            this.pack();
            this.setVisible(true);

            //créer un objet qui sera le "moteur de l'animation" et
            //stocker son adresse dans le champ tim ;
            //quand il est "démarré", cet objet doit lancer à chaque
            //10 ms un événement ActionEvent à l'intention du
            //container de premier niveau
……
……

      }//fin du constructeur de la classe JFrPrincipale

      //redéfinir la méthode gestionnaire d'événement qui s'exécute
      //automatiquement chaque fois qu'un des deux boutons est
      //appuyé ainsi que chaque fois que le "moteur de l'animation"
      //envoie un événement
      @Override
……
……
……
……
……
……
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
}//fin de la méthode gestionnaire d'événements

//définition de l'en-tête de la classe interne appelée
//JPanDessin et qui hérite de la classe prédéfinie JPanel ;
//un objet de ce type va occuper la zone centrale du container
//de premier niveau et servira pour dessiner le carré de taille
//variable
```

.....

.....

.....

```
{
        private int tailleJPan = 400;
        private int taille = tailleJPan/2;
        private int tailleMax = tailleJPan/2;
        private int tailleMin = tailleMax/2;

        //définir un champ privé, de type entier, nommé pas
        //et avec la valeur initiale 1 ;
        //la valeur de ce champ est utilisée pour augmenter ou
        //pour diminuer la taille du carré
.....

.....

        //définir un champ privé, de type logique, nommé
        //estCroissant et avec la valeur initiale vrai ;
        //la valeur de ce champ indique si la taille du carré
        //doit augmenter ou diminuer
.....

.....

        //définir l'en-tête du constructeur de la classe interne
        //(sans modificateur d'accès et sans argument)
.....

.....

        {
              //mettre la couleur de fond du panneau créé à la
              //valeur jaune (yellow en anglais)
.....

.....

              //préciser la taille préférée du panneau créé qui
              //doit avoir la largeur et la hauteur égales à la
              //valeur du champ tailleJPan
.....

.....

        }//fin du constructeur de la classe interne

        //préciser l'en-tête de la méthode redéfinie qui dessine
        //le panneau JPanDessin ; chaque nouvel appel à cette
        //méthode dessine un carré dont la taille augmente ou
        //diminue et crée ainsi l'effet d'animation
        @Override
.....

.....
```

{

//appeler la méthode d'origine de la classe mère

………………………………………………………………………………………………………

………………………………………………………………………………………………………

//donner la couleur rouge (**red** en anglais) à
//"l'outil de dessin"

………………………………………………………………………………………………………

………………………………………………………………………………………………………

//suivez les indications données dans l'énoncé afin
//de réaliser l'animation (en utilisant les valeurs
//des champs **estCroissant** et **pas** définis auparavant)

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

………………………………………………………………………………………………………

……

……

……

……

……

……

……

……

```
            }//fin de la méthode qui dessine le panneau
      }//fin de la classe interne JPanDessin
}//fin de la classe englobante JFrPrincipale
```