Licence Info 3éme année - module Programmation Objet 2

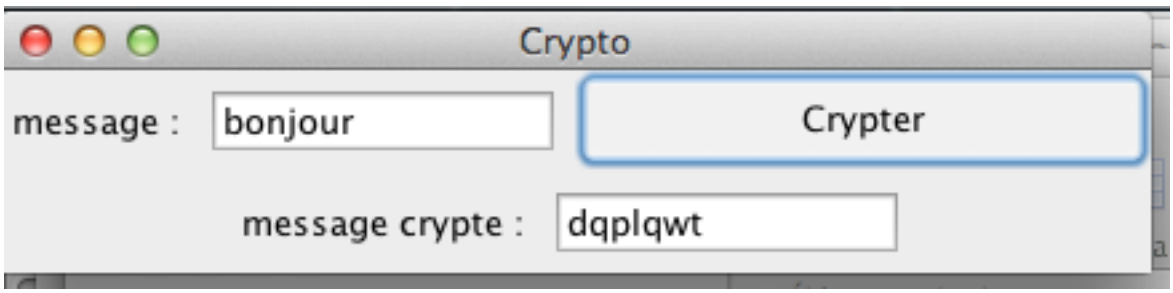
Examen – janvier 2016 – session 1 – durée : 2h

Documents autorisés : les polycopiés du cours **(pas de TD/TP)**

Partie A : Programmation graphique *(environ 10 points)*

L'objectif est de créer une interface permettant de crypter un message. Le principe de cryptographie utilisé ici est de remplacer chaque lettre d'un message par une autre à l'aide d'un tableau de chiffrement. Par exemple, un tel tableau avec le décalage circulaire de 2 :
- l1: abcdefghijklmnopqrstuvwxyz
- l2: cdefghijklmnopqrstuvwxyzab

Ainsi, le message « bonjour » sera crypté « dqplqwt ». L'interface est donnée au-dessous :



Concevoir une application avec MVC.

Question A.1 :

Nous vous donnons le modèle Crypto.java à compléter :

```
class Crypto // à compléter
 {
        private char[] l1,l2;

        // cree le tableau de chiffrement qui composé de l1 et l2, l1 contient 26 caractères de 'a' à 'z', et l2 est décalé circulairement les lettres de l1 d'un nombre offset
        public Crypto(int offset){
                l1 = new char[26];
                char c = 'a';
                for(int i=0;i<26;i++) {
                        l1[i]=(char) (c+i);

                }
                l2 = new char[26];
                c = 'a';
                for(int i=0;i<26;i++) {
                        if(((char) (c+i+offset)) > 'z'){
                                int t = 'z'-((char) (c+i));
                                l2[i]=(char) (c+offset-t-1);
                        }
                        else l2[i]=(char) (c+i+offset);
                }
        }
```

```
	// crypter un message sous forme d'une chaine de caractères
	public void crypte(String s) {
		char[] result = new char[s.length()];
		for(int i = 0;i<s.length();i++) {
			if(s.charAt(i) >= 'a' && s.charAt(i) <= 'z'){
				int j=0;
				while (l1[j]!=s.charAt(i)) {
					j++;
				}
				result[i] = l2[j];
			}
		}

		String res = new String(result);
		// à compléter
	}

}
```

Question A.2 :
Coder la classe contrôleur : ControleurGraphique.java.

Question A.3 :
Coder la classe vue : VueGraphique.java.

Question A.4 :
Nous vous donnons la classe de l'interface CryptoGUI.java à completer :

```
public class CryptoGUI extends JFrame {
  public CryptoGUI() {
	super("Crypto");
	this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	// à compléter : créer le modèle crypto, le contrôleur controlGra et la vue vueGra, ainsi que leur liens
	JPanel panel = new JPanel();
	panel.setLayout(new GridLayout(2,1));
	panel.add(controlGra);
	panel.add(vueGra);

	this.setContentPane(panel);
	this.pack();
	this.setVisible(true);
	}

	public static void main(String[] args) throws Exception {
		javax.swing.SwingUtilities.invokeLater(new Runnable() {
			public void run() {
				new CryptoGUI();
			}
		});
	}
}
```

Partie B : Carton … générique *: généricité et Exception (environ 4 points)*

Nous souhaitons disposer d'une classe générique Carton qui permette, entre-autre, de stocker des bouteilles :

la classe Bouteille :

```
package df.exam;

public class Bouteille {
   private String quoi;
   public Bouteille(String quoi) {
      this.quoi = quoi;
   }
   public String toString() {
      return "bouteille de "+quoi;
   }
}
```

Et la classe CartonDeBouteilles :

```
package df.exam;

import java.util.Scanner;

public class CartonDeBouteilles {
   public static void main(String args[]) {
      Carton<Bouteille> monCarton = new Carton<>(6);
      try {
         Bouteille bouteille = new Bouteille("Chinon");
         monCarton.ajouterUn(bouteille);
         bouteille = new Bouteille("Chinon Médaille d'Or");
         monCarton.ajouterUn(bouteille);
         bouteille = new Bouteille("Touraine");
         monCarton.ajouterUn(bouteille);
      } catch (Exception e) {
         System.out.println("erreur : "+e.getMessage());
         System.exit(1);
      }
      System.out.println("t(erminer), f (pour afficher), c(apacité), "
                         +"q(uantité), r(etirer), a(jouter) quoi");
      Scanner sc = new Scanner(System.in);
      String rep;
      do {
         rep = sc.next().trim();
         if (rep.equals("f"))
            System.out.println("détails : "+monCarton.toString());
         else if (rep.equals("c"))
            System.out.println("capacité : "+monCarton.getCapacite());
         else if (rep.equals("q"))
            System.out.println("quantité : "+monCarton.getQuantite());
         else if (rep.equals("r"))
         try {
            Bouteille bouteille = monCarton.retirerUn();
            System.out.println("bouteille retirée : "+bouteille.toString());
         } catch (IllegalStateException ise) {
            System.out.println("erreur retirer du carton : "+ise.getMessage());
         }
         else if (rep.equals("a")) {
            try {
               rep = sc.next().trim();
               Bouteille bouteille = new Bouteille(rep);
               monCarton.ajouterUn(bouteille);
               System.out.println("bouteille ajoutée ");
            } catch (IllegalStateException ise) {
```

```
               System.out.println("erreur ajouter au carton : "+ise.getMessage());
            } catch (Exception e) {
               System.out.println("erreur ajouter au carton : "+e.getMessage());
            }
            rep = "";
         }
      } while (! rep.equals("t")) ;
   }
}
```

Voici une exécution :

```
$ java CartonDeBouteilles
t(erminer), f (pour afficher), c(apacité), q(uantité), r(etirer), a(jouter) quoi
f
détails : carton { bouteille de Chinon, bouteille de Chinon Médaille d'Or, bouteille de
Touraine }
c
capacité : 6
q
quantité : 3
r
bouteille retirée : bouteille de Touraine
f
détails : carton { bouteille de Chinon, bouteille de Chinon Médaille d'Or }
a Vouvray
bouteille ajoutée
f
détails : carton { bouteille de Chinon, bouteille de Chinon Médaille d'Or, bouteille de
Vouvray }
t
```

Voici la classe Carton qui devrait permettre de stoker des éléments de même type :
Compléter la classe pour qu'elle soit générique.
Ne changez pas et ne recopiez pas les méthodes getCapacite() et toString().

```
package df.exam;

import java.util.LinkedList;
import java.util.List;

public class Carton {
   private List contenu;
   private int capacite;
   public Carton(int capacite) {
      this.capacite = capacite;
      this.contenu = new LinkedList();
   }

   public int getCapacite() {
      return capacite;
   }

   ....

   public String toString() {
      StringBuffer result = new StringBuffer("carton { ");
      int max = contenu.size() - 1;
      for ( int i=0; i < max; i++)
         result.append(contenu.get(i).toString()+", ");
      if (max >= 0)
```

```
            result.append(contenu.get(max).toString());
        return result.toString() + " }";
    }
}
```

Partie C : Vote , élection *: concurrence, sérialisation et Exception (environ 5 points)*

Nous souhaitons mettre en place des élections :

la classe Vote est déjà écrite :
elle comptabilise les voix des candidats FdB et Nabo.

```
package df.exam;

public class Vote  {
   private int scoreFdB;
   private int scoreNabo;
   public Vote(int scoreFdB, int scoreNabo) {
      this.scoreFdB = scoreFdB;
      this.scoreNabo = scoreNabo;
   }
   public int getScoreFdB() {
      return scoreFdB;
   }
   public void setScoreFdB(int scoreFdB) {
      this.scoreFdB = scoreFdB;
   }
   public void addScoreFdB(int scoreFdB) {
      this.scoreFdB += scoreFdB;
   }
   public int getScoreNabo() {
      return scoreNabo;
   }
   public void setScoreNabo(int scoreNabo) {
      this.scoreNabo = scoreNabo;
   }
   public void addScoreNabo(int scoreNabo) {
      this.scoreNabo += scoreNabo;
   }
}
```

la classe BureauDeVote est déjà écrite :
dans la vraie vie, les résultats ne sont pas si aléatoire ! Enfin, j'y crois encore … un peu.

```
package df.exam;

import java.util.concurrent.Callable;

public class BureauDeVote implements Callable<Vote>{
   public Vote call() throws Exception {
      try {
         Thread.sleep((int)(Math.random()*10000));
      } catch (InterruptedException ie) {
       return null;
      }
      int scoreFdB = (int)(Math.random()*1000);
      int scoreNabo = (int)(Math.random()*1000);
      return new Vote(scoreFdB, scoreNabo);
```

```
   }
}
```

Voici la classe Election qui devrait permettre de :
- afficher le résultat du précédent vote, s'il y a ;
- démarrer les « votations » des 10 bureaux de vote ;
- attendre les résultats des bureaux pour publier le résultat final ;
- sauvegarder le résultat de ce vote.

```
package df.exam;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class Election {
   public static void main(String [] args){
      // affichage du resultat du vote précédent
      File f = new File("resultatVote");
      Vote dernierVote;
      ...
      System.out.println("précédent vote : FdB "+dernierVote.getScoreFdB()
                         +" voix, Nabo "+dernierVote.getScoreNabo()+" voix");

      // lancement des votations dans les bureaux de vote
      ExecutorService executeur = Executors.newCachedThreadPool();
      List<Future<Vote>> resultats = new ArrayList<>();
      BureauDeVote unBureauDeVote;
      Future<Vote> unResultat;
      for (int i = 0; i < 10; i++) {
         unBureauDeVote = new BureauDeVote();
         unResultat = executeur.submit(unBureauDeVote);
         resultats.add(unResultat);
      }

      // attente des résultats
      Vote totalVotes = new Vote(0,0);
      ...
      System.out.println("résultats vote : FdB "+totalVotes.getScoreFdB()
                       +" voix, Nabo "+totalVotes.getScoreNabo()+" voix");

      // sauvegarde du résultat du vote
      ...
   }
}
```

Voici une exécution :

```
$ java Election
précédent vote : FdB 6015 voix, Nabo 4687 voix
résultats vote : FdB 6391 voix, Nabo 4305 voix
```

Vous compléterez la classe .
Ne changez pas la partie « lancement des votations dans les bureaux de vote ».
Les 2 questions sont indépendantes.
Vous prendrez en compte les Exceptions.

Question A :
Écrire les parties affichage du résultat du vote précédent et sauvegarde du résultat du vote.
Cela a-t-il des conséquences ?

Question B :
Écrire la partie attente des résultats de telle sorte que votre solution soit « thread-safe » et sans attente active.