Dico.java • Procédure de lancement

```
import javax.swing.*;
import java.awt.*;
import java.io.*;

public class Dico
{
    public static void main(String args[])
    {
        // Cree la fenetre principale de l'application :
        JFrame fenetre = new Interf();

        // Lorsque la fenetre sera fermee, l'application quittera :
        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Verification d'integrite des controles de la fenetre :
        fenetre.pack();

        // Affiche la fenetre pour notre cher utilisateur :
        fenetre.setVisible(true);
    }
};
```

Interf.java • Fenêtre principale

```
import javax.swing.*;
import javax.swing.table.*;
import javax.swing.filechooser.*;
import javax.swing.event.TableModelEvent;
import java.awt.*;
import java.awt.event.*;
import java.util.HashMap;
import java.util.Vector;
import java.util.Iterator;
import java.io.*;


public class Interf extends JFrame implements ActionListener
{

    // Filtre pour les fichiers contenant les mots a indexer :
    public class TextFileFilter extends javax.swing.filechooser.FileFilter
    {
        public boolean accept(File f)
        {
            return f.getName().toLowerCase().endsWith(".txt")
                || f.getName().toLowerCase().endsWith(".lem")
                || f.isDirectory();
        }

        public String getDescription()
        {
            return "Fichier lexique";
        }
    };


    // Filtre pour les fichiers proprietaires du programme stockant les index :
    public class IndexFileFilter extends javax.swing.filechooser.FileFilter
    {
        public boolean accept(File f)
        {
            return f.getName().toLowerCase().endsWith(".idx")
                || f.getName().toLowerCase().endsWith(".ref")
                || f.getName().toLowerCase().endsWith(".txt")
                || f.isDirectory();
        }

        public String getDescription()
        {
            return "Fichier index";
        }
    };


    // Membres donnees des menus :
    private JMenuBar menuBar;
    private JMenu menuFichier;
    private JMenuItem itemCreerIndex,
                      itemOuvrir,
                      itemEnregistrer,
                      itemEnregistrerSous,
                      itemQuitter;

    // Objets de la fenetre :
    private JTable table;
    private JScrollPane scrollPane;

    // Objets sous-jacents de la gestion des donnees de la JTable
    private DefaultTableModel modele;
    private TableSorter triage;
```

```
    // Parametres internes de la classe :
    private String nomFichierIndex;
    private Vector entetesColonnes;

    // Titre de l'application :
    private final String titreApplication="Dictionnaire";



    // Constructeur : initialisation de la fenetre :
    public Interf()
    {
        // Invoque le constructeur de JFrame en specifiant le titre de la fenetre :
        super();
        setTitle(titreApplication);

        // Definition des en-tetes de colonnes :
        entetesColonnes=new Vector();
        entetesColonnes.add("Mot");
        entetesColonnes.add("Nombre d'occurences");

        // Cree le modele de donnee (qui contiendra les infos de la table)
        // et y associe le TableSorter permettant d'en trier le contenu :
        modele=new DefaultTableModel(entetesColonnes, 0);
        triage=new TableSorter(modele);

        // Cree l'objet d'interface table et y ajoute la gestion des clics sur les colonnes :
        table=new JTable(triage);
        triage.addMouseListenerToHeaderInTable(table);

        // Taille par defaut permettant un affichage correct des donnees :
        table.setPreferredScrollableViewportSize(new Dimension(700, 200));

        // Cree le scrollpane et insere le tableau dedans :
        scrollPane=new JScrollPane(table);

        // Ajoute le scrollpane a la fenetre :
        getContentPane().add(scrollPane, BorderLayout.CENTER);


        // Definition de la barre de menu et de l'unique menu Fichier :
        menuBar=new JMenuBar();
        menuFichier=new JMenu("Fichier");

        // Definition et gestion des clics sur les items de menu :
        itemCreerIndex=new JMenuItem("Creer Index...");
        itemCreerIndex.setEnabled(true);
        itemCreerIndex.addActionListener(this);

        itemOuvrir=new JMenuItem("Ouvrir...");
        itemOuvrir.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O, Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
        itemOuvrir.addActionListener(this);

        itemEnregistrer=new JMenuItem("Enregistrer");
        itemEnregistrer.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S, Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
        itemEnregistrer.setEnabled(false);
        itemEnregistrer.addActionListener(this);

        itemEnregistrerSous=new JMenuItem("Enregistrer sous...");
        itemEnregistrerSous.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
ActionEvent.SHIFT_MASK+Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
        itemEnregistrerSous.setEnabled(false);
        itemEnregistrerSous.addActionListener(this);

        itemQuitter=new JMenuItem("Quitter");
        itemQuitter.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Q, ActionEvent.CTRL_MASK));
        itemQuitter.addActionListener(this);

        // Ajout des items au menu Fichier lui-même :
        menuFichier.add(itemCreerIndex);
        menuFichier.add(itemOuvrir);
        menuFichier.add(itemEnregistrer);
        menuFichier.add(itemEnregistrerSous);
        menuFichier.add(itemQuitter);

        // Ajout du menu Fichier et du menu a la fenetre :
        menuBar.add(menuFichier);
        setJMenuBar(menuBar);


        // Aucun nom par defaut pour enregistrer l'index :
        nomFichierIndex=null;
    }




    // Gestion des evenements du type clic sur un item de menu :
    public void actionPerformed(ActionEvent e)
    {
        // Distribution de chaque clic sur item au gestionnaire competent :
        if(e.getSource()==itemCreerIndex)
            actionCreerIndex();

        if(e.getSource()==itemOuvrir)
            actionOuvrir();

        if(e.getSource()==itemEnregistrer)
            actionEnregistrer();

        if(e.getSource()==itemEnregistrerSous)
            actionEnregistrerSous();

        if(e.getSource()==itemQuitter)
            System.exit(0);
    }
```

```
// Gestion centralisee des messages d'erreur - dans une boite de dialogue :
public void boiteErreur(String message)
{
    JOptionPane.showMessageDialog(this,message,"Erreur",JOptionPane.ERROR_MESSAGE);
}



// Creation d'un index (remplissage table) a partir d'un fichier texte :
private void actionCreerIndex()
{
    // Montre le selecteur de fichier pour specifier le fichier texte a analyser :
    JFileChooser selecFichier=new JFileChooser();
    selecFichier.setDialogTitle("Texte a indexer...");
    selecFichier.setFileFilter(new TextFileFilter());
    int retour=selecFichier.showOpenDialog(this);

    if(retour==JFileChooser.APPROVE_OPTION)
    {
        // L'utilisateur a clique sur OK et a bien choisi un nom de fichier :
        try
        {
            // Ouvre le fichier en lecture :
            FileReader lecteur=new FileReader(selecFichier.getSelectedFile().getName());
            BufferedReader fichier=new BufferedReader(lecteur);
            String ligne=new String();

            // Cette table de hachage contiendra l'index proprement dit :
            HashMap reference=new HashMap();

            // A chaque ligne du fichier, on lit le mot et on l'ajoute dans la base
            // s'il n'est pas deja reference, sinon on incremente son occurence...
            while((ligne=fichier.readLine())!=null)
            {
                    if(reference.containsKey(ligne))
                        reference.put(ligne, new Integer( ((Integer)reference.get(ligne)).intValue()+1 ) );
                    else
                        reference.put(ligne, new Integer(1));
            }

            // Convertit la table de hachage en un gros vecteur qui contiendra
            // lui-meme des vecteurs de 2 elements : le mot et son occurence.
            Vector couple, indexVirtuel=new Vector();
            Iterator itcles=reference.keySet().iterator();
            Object element;
            while(itcles.hasNext())
            {
                element=itcles.next();
                couple=new Vector();
                couple.addElement(element);
                couple.addElement(reference.get(element));
                indexVirtuel.addElement(couple);
            }

            // Recree un modele de donnees a partir de notre vecteur de vecteur :
            modele=new DefaultTableModel(indexVirtuel, entetesColonnes);
            triage.setModel(modele);

            // Rafraichissement de l'objet graphique pour refleter la modif des donnees :
            modele.newDataAvailable(new TableModelEvent(modele));

            // On trie immediatement par ordre lexicographique les mots :
            triage.sortByColumn(0);

            // Nous avons un index, on peut autoriser l'enregistrement d'un fichier index :
            itemEnregistrer.setEnabled(true);
            itemEnregistrerSous.setEnabled(true);
            nomFichierIndex=null;
            setTitle(titreApplication + " - Sans Titre");

        }
        catch(FileNotFoundException except)
        {
            boiteErreur("Impossible de lire le fichier selectionne. Verifier que le fichier\nn'a pas ete efface et que "
                        + "vous possedez les droits suffisants pour le lire.");
        }
        catch(IOException except)
        {
            boiteErreur("Le fichier specifie n'a pas un format valide.\nAbandon de l'operation d'indexage.");
        }
    }
}



// Item enregistrer - ne redemande pas le nom du fichier SI il a deja ete entre :
private void actionEnregistrer()
{
    if(nomFichierIndex==null) actionEnregistrerSous();
    else enregistrerIndex();
}



// Item enregistrer sous - demande le nom du fichier et enregistre :
public void actionEnregistrerSous()
{
    // Demande le nom de fichier dans lequel sauver :
    JFileChooser selecFichier=new JFileChooser();
    selecFichier.setDialogTitle("Enregistrer sous...");
    IndexFileFilter filtre=new IndexFileFilter();
    selecFichier.setFileFilter(filtre);
    int retour=selecFichier.showSaveDialog(this);
```

```
        // Si l'utilisateur a valide le nom, garde le nom en memoire et sauve :
        if(retour==JFileChooser.APPROVE_OPTION)
        {
            // Recupere le nom du fichir choisi par l'utilisateur :
            nomFichierIndex=selecFichier.getSelectedFile().getName();

            // Ajoute l'extension si celle-ci n'y est pas :
            if(!filtre.accept(selecFichier.getSelectedFile()))
                nomFichierIndex=nomFichierIndex+".idx";

            setTitle(titreApplication + " - " + nomFichierIndex);

            // Enregistrement normal :
            enregistrerIndex();
        }
    }



    // Enregistrement proprement dit d'un index (contenu table) dans un fichier de type index :
    private void enregistrerIndex()
    {
        try
        {
            FileOutputStream ostream = new FileOutputStream(nomFichierIndex);
            ObjectOutputStream p=new ObjectOutputStream(ostream);

            // Grace a la magie de la serialisation, cette simple fonction enregistre la
            // totalite de l'objet dans le fichier specifie !
            p.writeObject(modele);

            p.flush();
            ostream.close();
        }
        catch(FileNotFoundException except)
        {
            boiteErreur("Impossible d'ecrire dans le specifie. Verifier le fichier\nn'a pas ete protege et que "
                        + "vous avez les droits suffisants pour ecrire dans ce repertoire.");
        }
        catch(IOException except)
        {
            boiteErreur("Il y a eu un dysfonctionnement lors de l'enregistrement de l'index.");
        }
    }



    // Item Ouvrir - re-ouvre un fichier index et affiche son contenu dans la table :
    public void actionOuvrir()
    {
        // Selecteur de fichier pour choisir le fichier a lire :
        JFileChooser selecFichier=new JFileChooser();
        selecFichier.setDialogTitle("Ouvrir un index...");
        selecFichier.setFileFilter(new IndexFileFilter());
        int retour=selecFichier.showOpenDialog(this);

        if(retour==JFileChooser.APPROVE_OPTION)
        {
            // L'utilisateur a clique sur OK et a choisi un fichier :
            try
            {
                FileInputStream istream=new FileInputStream(selecFichier.getSelectedFile().getName());
                ObjectInputStream p=new ObjectInputStream(istream);

                // Recupere l'objet grace a la magie de la serialisation :
                modele=(DefaultTableModel)p.readObject();
                istream.close();
                triage.setModel(modele);

                // De nouvelles donnees ont ete chargees, il s'agit de rafraichir le controle :
                modele.newDataAvailable(new TableModelEvent(modele));

                // Et de trier immediatement par mots :
                triage.sortByColumn(0);

                // OK, on peut a present enregistrer l'index :
                itemEnregistrer.setEnabled(true);
                itemEnregistrerSous.setEnabled(true);
                nomFichierIndex=selecFichier.getSelectedFile().getName();
                setTitle(titreApplication + " - " + nomFichierIndex);
            }
            catch(FileNotFoundException except)
            {
                boiteErreur("Impossible de lire le fichier selectionne. Verifier que le fichier\nn'a pas ete efface et que "
                            + "vous possedez les droits suffisants pour le lire.");
            }
            catch(IOException except)
            {
                boiteErreur("Le fichier specifie n'a pas un format valide.\nAbandon de l'operation d'indexage.");
            }
            catch(ClassNotFoundException except)
            {
                boiteErreur("Erreur interne lors du chargement du fichier.\nCes programmeurs ne valent pas un clou !");
                // Esperons que cette erreur n'arrive jamais...
            }
        }
    }

};
```