Un éditeur de texte

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

- Objectifs
- Cahier des charges
- Textes et documents
- Vue d'ensemble
- Actions
- UndoHandler
- MenuManager
- Fichiers
- Etat du document
- Menus
- Barre d'outils

Objectifs

MinEdit0.bat

- Présenter des interactions d'un composant de texte avec son environnement
  - Ouverture et sauvegarde des fichiers
  - Couper-coller
  - Undo-Redo
- L'important est la cohérence de l'environnement
  - Entrées des menus activables seulement si cela a un sens
  - "Aide implicite" que cela apporte
- En revanche, on ignore le style du texte lui-même
  - Style des paragraphes
  - Polices de caractères

Cahier de charges

- Editeur SDI (single document interface)
  - un seul document présent
  - une seule fenêtre de manipulation du texte
- Autres modèles
  - une seule fenêtre, plusieurs documents
  - plusieurs fenêtres, une par document
- Commandes de manipulation de documents
  - nouveau, ouvrir, sauver, sauver sous
- Commandes de manipulation du texte
  - copier - couper, coller, tout sélectionner
  - annuler - rétablir
- Présentation de ces commandes sous forme
  - menu - toolbar - raccourcis clavier

Textes et documents

- Classes de textes

```
...
|
+--javax.swing.JComponent
      |
      +--javax.swing.text.JTextComponent
            |
            +--javax.swing.JTextArea
            |
            +--javax.swing.JTextField
            |
            +--javax.swing.JEditorPane
                  |
                  +--javax.swing.JTextPane
```

- Classes de documents

```
java.lang.Object
  |
  +--javax.swing.text.AbstractDocument     implements        Document
        |                                                      |
        +--javax.swing.text.PlainDocument                      | extends
        |                                                      |
        +--javax.swing.text.DefaultStyledDocument implements StyledDocument
              |
              +--javax.swing.text.html.HTMLDocument
```

Document / vue

- Un composant de texte présente une vue d'un document.
  - **`TextArea`** et **`TextField`** associés au **`PlainDocument`**
  - **`TextPane`** associé à **`StyledDocument`**

- Correspondance

```
JTextArea editor;
Document document = editor.getDocument();
```

```
editor.setDocument(new PlainDocument());
```

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Ecouter le texte

- Via le document, insertion, suppression, remplacement

```
Document document = editor.getDocument();
document.addDocumentListener( un listener );
```

- Un **DocumentListener** implémente trois méthodes

```
public void changedUpdate (DocumentEvent e);
public void insertUpdate (DocumentEvent e);
public void removeUpdate (DocumentEvent e);
```

appelées après modification d'un attribut, insertion, suppression.

Sélection

- On peut pister les déplacements du point d'insertion (caret)

```
Caret caret = editor.getCaret();
caret.addCaretListener( un CaretListener );
```

  par un **`CaretListener`**

- Un CaretListener possède une méthode **`caretUpdate`** appelée chaque fois que le point d'insertion bouge
- un **`CaretEvent`** fournit deux méthodes
  - **`getDot()`** qui donne le point actuel
  - **`getMark()`** qui donne le point précédent

```
public void caretUpdate(CaretEvent e) {
   int now = e.getDot();
   int before = e.getMark();
   boolean nowSelected = now != before;
   ...
}
```

- Un *mouvement de souris*, avec bouton enfoncé,
  - ne provoque pas d'évènement,
  - mais provoque un évènement quand on relâche le bouton

Manipulations de textes

- Manipulations de texte prédéfinies (sont en fait des méthodes de **`JTextComponent`**):

```
void editor.cut();
void editor.copy();
void editor.paste();
void editor.selectAll();
```

les dernières transfèrent dans le presse-papier système.

```
Clipboard clip =
   Toolkit.getDefaultToolkit().getSystemClipboard();
```

- Le **`DefaultEditorKit`** prédéfinit une trentaine d'actions sur les composants de textes.

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Vue d'ensemble

- Le texte
  - une seule zone de texte (**`JTextArea`**)
  - le document associé ne change pas, sauf pour la commande "nouveau".
- Les actions
  - chaque action (**`Action`**) (nouveau,..., tout sélectionner) est implémentée dans une classe séparée
- Les menus et la barre d'outils
  - construits à partir des actions
- Les gestionnaires de cohérence
  - de la cohérence des menu : une **`EditMenuManager`**
  - de la cohérence des undo-redo : un **`UndoHandler`**
  - de sauvegarde de documents modifiés : une **`StatusBar`**.

Composants

- Composants de la vue

```
JTextComponent editor;
JMenuBar menubar;
JToolBar toolbar;
StatusBar status;
```

- Composants de la gestion

```
File currentFile = null;
JFileChooser selecteurFichier;
UndoHandler undoHandler;
EditMenuManager editMenuManager;
```

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Les actions

- Une action par ... action

```
Action undoAction = new UndoAction();
Action redoAction = new RedoAction();
Action newAction = new NewAction();
Action openAction = new OpenAction();
Action saveAction = new SaveAction();
Action saveAsAction = new SaveAsAction();
Action exitAction = new ExitAction();
Action cutAction = new CutAction();
Action copyAction = new CopyAction();
Action pasteAction = new PasteAction();
Action selectAllAction = new SelectAllAction();
```

UndoAction

- Premier exemple : **undoAction**

```
class UndoAction extends AbstractAction {
  public UndoAction() {
    super("Undo", new ImageIcon("gifs/undo.gif"));
    setEnabled(false);
  }
  public void actionPerformed(ActionEvent e) {
    try { undoHandler.undo(); }
      catch (CannotUndoException ex) {}
    undoHandler.update();
    }
}
```

UndoHandler

- Il gère les undo, mais aussi l'état des boutons !

```
class UndoHandler extends UndoManager {
  public void undoableEditHappened(UndoableEditEvent e) {
    super.addEdit(e.getEdit()); // le “super” inutile
    update(); // en plus
  }
  public void update() {
    undoAction.setEnabled(canUndo());
    redoAction.setEnabled(canRedo());
  }
}
```

CutAction

- *Couper* implique
  - mettre dans la corbeille
  - mettre à jour les boutons

```
class CutAction extends AbstractAction {
  CutAction() {
    super("Cut", new ImageIcon("gifs/cut.gif"));
  }
  public void actionPerformed(ActionEvent e) {
    getEditor().cut(); // texte
    editMenuManager.doCut(); //boutons
  }
}
```

EditMenuManager

- Il gère
  - les transitions entre les 4 états du menu
  - la mise-à-jour de la vue (menu et toolbar) par la fonction **`update()`**

```
class EditMenuManager implements CaretListener {
  int state;
  static final int
    EMPTY = 0, CUTCOPY = 1, PASTE = 2, FULL = 3;

  void doInitial() {...}
  void doCopy() {...}
  void doCut() {...}
  void doPaste() {...}
  void doSelected() {...}
  void doDeselected() {...}
}
```

EditMenuManager (suite)

- Après une sélection :

```
void doSelected() {
  if (state == EMPTY) state = CUTCOPY;
  else if (state == PASTE) state = FULL;
  updateEnables(state);
}
```

- Après un copy :

```
void doCopy() {
  if (state == CUTCOPY) {
    state = FULL;
    updateEnables(state);
  }
}
```

EditMenuManager (suite)

- C’est aussi un **CaretListener**, pour écouter les sélections

```
public void caretUpdate(CaretEvent e) {
  int now = e.getDot();
  int before = e.getMark();
  boolean nowSelected = now != before;
  if (nowSelected)
    doSelected();
  else
    doDeselected();
}
```

EditMenuManager (fin)

- La mise-à-jour des boutons est paresseuse

```
public void updateEnables(int state) {
  switch (state) {
  case EMPTY :
    cutAction.setEnabled(false);
    copyAction.setEnabled(false);
    pasteAction.setEnabled(false);
    break;
  case CUTCOPY:
    cutAction.setEnabled(true);
    copyAction.setEnabled(true);
    pasteAction.setEnabled(false);
    break;
  case PASTE: ...
  case FULL: ...
  }
}
```

Ouvrir un fichier

- Il faut
  - s'assurer que le fichier courant n'est pas modifié
  - s'il est modifié, demander une éventuelle sauvegarde
  - ouvrir un dialogue de choix de fichier
  - lire ce fichier
- Ces opérations sont assumées par la méthode **`actionPerformed()`**

```
class OpenAction extends AbstractAction {
  OpenAction() {
    super("Ouvrir...", new ImageIcon("gifs/open.gif"));
  }
  public void actionPerformed(ActionEvent e) {...}
}
```

Ouvrir un fichier (suite)

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

```
public void actionPerformed(ActionEvent e) {
  if (!isConfirmed(
    "Voulez vous sauver le texte courant\n"+
    " avant d'ouvrir un autre fichier ?",
    "Sauver avant d'ouvrir ?")) return;
  int answer = selecteurFichier.showOpenDialog(frame);
  if (answer != JFileChooser.APPROVE_OPTION)
    return;
  currentFile = selecteurFichier.getSelectedFile();
  try {
    FileReader in = new FileReader(currentFile);
    getEditor().read(in, null);
    in.close();
  }
  catch (IOException ex) { ex.printStackTrace(); }
  status.setSaved();
  frame.setTitle(currentFile.getName());
}
```

Ouvrir un fichier (fin)

```
boolean isConfirmed(String question, String titre) {
  if (!status.isModified()) return true;
  int reponse = JOptionPane.showConfirmDialog(null,
            question, titre, JOptionPane.YES_NO_CANCEL_OPTION);
  switch(reponse) {
    case JOptionPane.YES_OPTION: {
      saveAction.actionPerformed(null);
      return !status.isModified();
    }
    case JOptionPane.NO_OPTION: return true;
    case JOptionPane.CANCEL_OPTION: return false;
  }
  return false;
}
```

Etat du document

- Il n'existe pas de fonction qui indique une modification du document
- StatusBar assume ce rôle ...

```
class StatusBar extends JPanel implements DocumentListener {
  boolean modStatus = false; // true = modified;

  public boolean isModified() { return modStatus; }

  public void changedUpdate(DocumentEvent ev) { setModified();}
  public void insertUpdate(DocumentEvent ev) { setModified();}
  public void removeUpdate(DocumentEvent ev) { setModified();}
  public void setSaved() {
    modStatus = false;
    getEditor().getDocument().addDocumentListener(this);
    saveAction.setEnabled(false);
  }
  public void setModified() {
    modStatus = true;
    getEditor().getDocument().removeDocumentListener(this);
    saveAction.setEnabled(true);
  }
}
```

Les menus

- Dans le menu “Fichier”, on ajoute des raccourcis

```
protected JMenuBar createMenubar() {
  JMenuBar mb = new JMenuBar();
  JMenu menu;
  JMenuItem item;
  menu = new JMenu("Fichier");
  item = menu.add(newAction);
  item.setIcon(null); item.setMnemonic('N');
  item = menu.add(openAction);
  item.setIcon(null); item.setMnemonic('O');
   ...
  menu.addSeparator();
  item = menu.add(exitAction);
  mb.add(menu);
   ...
  return mb;
}
```

La barre d'outils

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

- On ajoute les tooltips, des espaces et de la glue

```
private JToolBar createToolbar() {
  JButton b;
  JToolBar tb = new JToolBar();
  b = tb.add(newAction);
  b.setText(null);
  b.setToolTipText("nouveau");
    ...
  tb.add(Box.createHorizontalStrut(10));
  b = tb.add(copyAction);
  b.setText(null);
  b.setToolTipText("copier");
    ...
  tb.add(Box.createHorizontalGlue());
  return tb;
}
```