

# Interfaces Graphiques

Undo-Redo

Focus, Toolbar

Table, Tree

Jean-Baptiste.Yunes@u-paris.fr

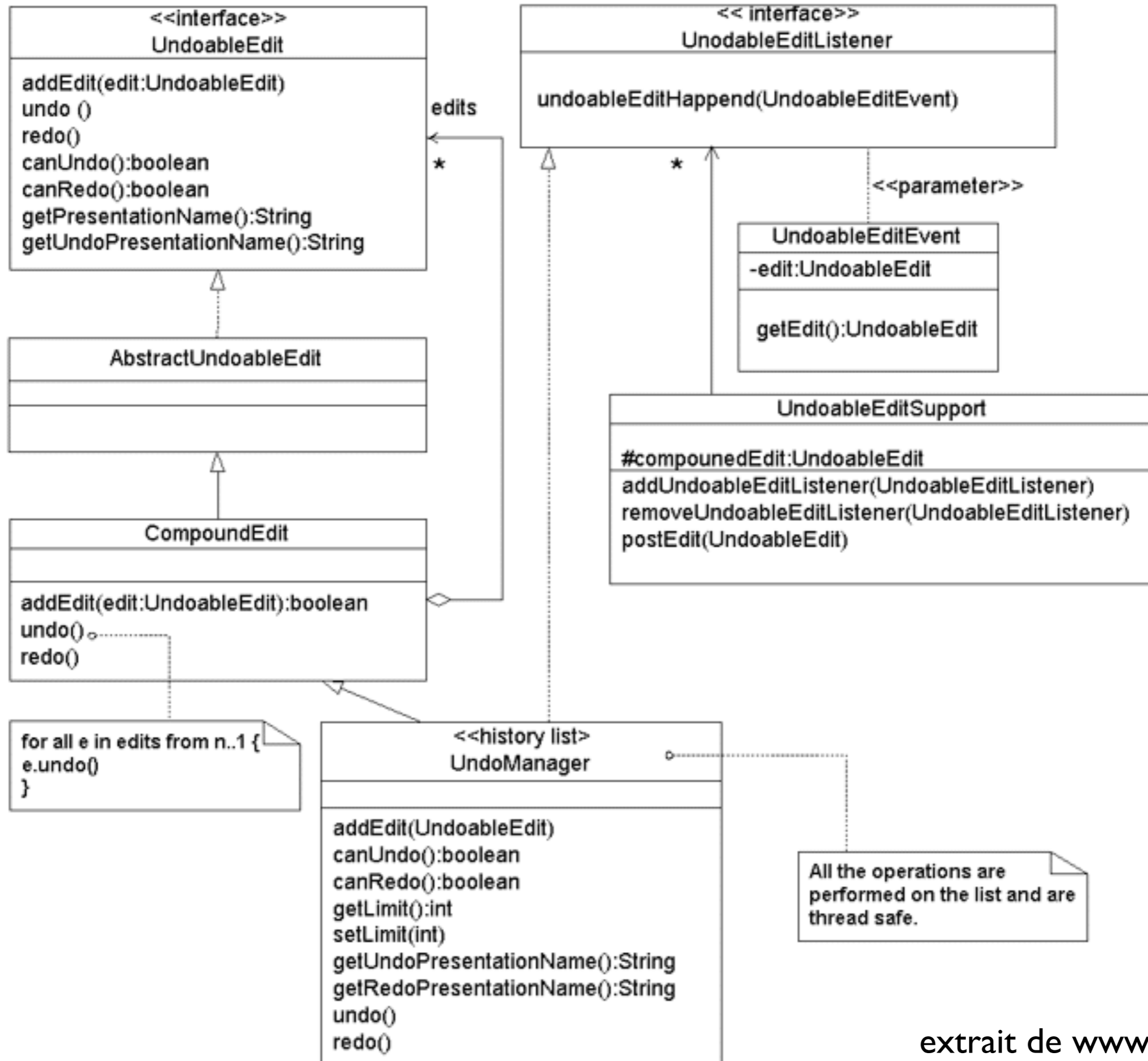
Université Paris Cité

©2026

- Annuler — Rétablir
- Do — Undo
- Undo — Redo

- Annuler / Rétablir : Undo / Redo
- Design pattern : Command
  - permet d'encapsuler tout ce qui est nécessaire à appeler une méthode dans le futur...
  - par exemple, quelque chose qui permettra de défaire une action que l'on fait (Undo)
    - voir refaire une action que l'on défait (Redo)...

# Annuler — Rétablir



- l'interface `UndoableEdit` ou la classe abstraite `AbstractUndoableEdit`
- encapsule ce qui est nécessaire pour réaliser une ou deux des actions suivantes :
  - `undo()`
    - censée annuler une action passée
  - `redo()`
    - censée rétablir une action précédemment défaite

- La classe `UndoManager` est un gestionnaire d'historique d'actions
- on peut y ajouter des actions
  - `addEdit(UndoableEdit)`
- tester
  - `canUndo()` pour tester si on peut annuler
  - `canRedo()` pour tester si on peut rétablir
- demander
  - `undo()` pour annuler une action
  - `redo()` pour rétablir une action

- La difficulté reste ensuite de garder l'interface cohérente
- mise à jour des options de menus
  - actives ou non
  - étiquette si actives
- autres éléments d'interface, s'ils existent

- Exemple : `UndoRedo.java`
- Attention `Observer/Observable` est désormais obsolète depuis la version 9...

- Navigation
- Focus Traversal

- Focus Ordering
- la navigation au clavier est importante, en particulier lors de saisies
- un formulaire
  - permettre de passer d'un champ à l'autre facilement
  - un ordre logique doit être proposé (parfois différent de l'ordre graphique)

- Focus Traversal Cycle
  - une collection de composants ayant un même ancêtre
- Focus Cycle Root
  - l'ancêtre d'un cycle :
    - JApplet, JDesktopPane, JDialog, JEditorPane, JFrame, JInternalFrame, JWindow
  - peut contenir d'autres racines...
  - `setFocusTraversalPolicy(FocusTraversalPolicy);`

- `FocusTraversalPolicy`
- la classe abstraite permettant de définir la navigation :
  - `Component getInitialComponent(Window);`
  - `Component getFirstComponent(Container);`
  - `Component getLastComponent(Container);`
  - `Component getDefaultComponent(Container);`
  - `Component GetComponentAfter(Container, Component);`
  - `Component  
GetComponentBefore(Container, Component);`

- **Exemple** : `FocusTraversal.java`

- `JToolBar`
  - un container particulier
    - une barre d'outils déplaçable et détachable!
  - `setFloatable(boolean)`
    - déplaçable/détachable
- Il existe des icônes préparées pour l'apparence Java :
  - *Java Look and Feel Graphics Repository*

- **Example : `ToolBarExample.java`**

- `JTable` :
- permet de lister des éléments, chaque ligne représentant un élément, chaque colonne un attribut de l'élément à lister

- `JTable` :
- on doit lui associer un modèle (`TableModel`), lequel fournira les données à afficher
- on peut lui associer des tris
  - certains tris peuvent aussi opérer un filtrage...
  - `setRowSorter ( RowSorter < M > )`
  - ou `setAutoCreateRowSorter ( boolean ) ;`
- on peut modifier son apparence
  - les `TableCellRenderer`...

- `TableRowSorter<M>` :
- permet de contrôler quel `Comparator` doit être employé pour trier les entrées des colonnes de la table
- méthode la plus importante
  - `Comparator<?> getComparator(int);`
- en fait une `DefaultRowSorter<M, Integer>`
  - un `RowSorter<M>` capable de filtrer des entrées de type `Integer`
    - `setRowFilter(RowFilter<M, I>);`

- `RowFilter<M, I>`
  - permet d'obtenir le filtrage des entrées de type `I` depuis le modèle `M`
- Il en existe de prédéfinis :
  - `dateFilter`, `numberFilter`, `regexpFilter`
  - et les opérateurs `orFilter`, `andFilter`, `notFilter`
- sinon définir la méthode abstraite
  - `boolean include(RowFilterEntry<M, I> entry);`

- L'apparence de chaque case d'une `JTable` est assurée par utilisation d'un `Component`
- `TableCellRenderer` est une interface
  - permettant d'obtenir pour une position donnée dans la table et une valeur à afficher le composant servant à réaliser l'affichage
- Il existe une implémentation par défaut étendant un `JLabel`
  - `DefaultTableCellRenderer`
    - à sous-classer pour obtenir l'effet voulu...

- On peut modifier de façon globale l'apparence d'une donnée selon son type :
  - JTable
    - `setDefaultRenderer(Class<?>, TableCellRenderer);`
- On peut modifier l'apparence pour une colonne donnée :
  - TableColumn (à obtenir depuis une JTable par `getColumnModel` puis `getColumn`)
    - `setCellRenderer(TableCellRenderer);`

- **Exemple : JTableExample.java**

- JTable ok mais JTree ?
- Comme pour une JTable :
  - un modèle pour les données
    - TreeModel
    - TreeNode
- Exemple : JTreeExample.java