

## Solution 1 : sauvegarder les états du TextEditor

Role : caretaker

<i>UndoManager</i>
List<EditorMemento> pastStates; List<EditorMemento> futurStates; Engine engine;
void store(); void undo(); void redo();

Role : Originator

<i>EngineImpl</i>
Memento getMemento(); void setMemento(Memento m);

Idée : quand on veut undo/redo, on remet l'engine à l'état adéquate (cf : EditorMemento)

Role : ConcreteMemento

<i>EditorMemento</i>
String bufferContent; int beginIndex; int endIndex;

Pros : rapide | Cons : besoin de beaucoup d'espace mémoire

## Solution 2 : sauvegarder toutes les commandes

Role : caretaker

<i>UndoManager</i>
List<Pair<CommandOriginator, Memento>> pastCommands List<Pair<CommandOriginator, Memento>> futurCommands
void store(Pair<CommandOriginator, Memento> m); void undo(); void redo();

Idée : quand on veut undo, on reset l'engine et on réapplique toutes les commandes enregistrées sauf la dernière

Pros : peu gourmand en mémoire | Cons : lent

Solution 3 : (solution 1 + solution 2) on enregistre tout les commandes et un sous échantillon d'état du TextEditor

Role : caretaker

<i>UndoManager</i>
List<EditorMemento> pastStates; List<EditorMemento> futurStates; List<Pair<CommandOriginator, Memento>> pastCommands; List<Pair<CommandOriginator, Memento>> futurCommands; Engine engine; int K;
void store(EditorMemento m); void store(Pair<CommandOriginator, Memento> m); void undo(); void redo();

Idée : on sous-échantillonne les états de l'éditeur (cf K). Quand on fait un undo, on remet le textEditor dans le dernier état connu, puis on réapplique les commandes, sauf la dernière

Pros : peu gourmand en mémoire, rapide | Cons :

#### Solution 4 : créer des commandes inversées

Idée : à partir d'une instance de commande donné, générer une commande inverse et l'appliquer

Pro : optimal | cons : implémentation complexe