

CS 349

Undo



Undo 1

Feb 10 Announcements

- A01 remark requests due by 5:00pm Monday. Email the TA who marked it.
- Final exam schedule is posted
 - CS349: Wednesday, April 18, 12:30-3:00 RCH 103, 105, 112
- A03 is believed to be final; there were changes since it was posted in draft form



Undo 2

Why Undo?

- Why offer undo?
- What does it offer us?
- How is it used by people in practice?



Undo 3

Use: Correcting Errors

- Fix “mistakes” in input
 - A safety net for input techniques
 - Allows faster input
 - Allows for less planning
- Two types of errors:
 - User input error (human side)
 - Interpretation error (computer side)

Use: Supporting Exploration

“One of the key claims of direct manipulation is that users would *learn primarily by trying* manipulations of visual objects rather than by reading extensive manuals.” [Olsen, p. 327]

- Exploratory learning
 - Try things you don’t know the consequences of
 - Well-implemented undo can allow users to try without commitment
- Exploring alternative problem solutions
 - Again, try something without commitment

Use: Evaluation

- Fast do-undo-redo cycles
 - previous and current version are flashed in quick succession
 - provides in-place evaluation *across time*

Choices: Granularity

- What defines one undoable “operation”?
- Typing in MS Word
 - Apparently separated by a non-typing operation eg: bolding or switching to another app
- Typing in TextMate
 - A character
- Typing in TextPad
 - A line of text (always)
 - Probably because it is often used for programming where a line has a more specific meaning than in a word processor.
- Key question: What are appropriate undo “chunks”?



Granularity

- Example: drawing a free-hand line
 - User presses mouse button to begin drawing
 - User drags mouse with button pressed to define the line's path
 - User releases the mouse button at the end of the path
- Mouse down + Mouse drag + Mouse up
 - one conceptual unit
 - “undo” should probably undo the entire line, not just a small delta in the mouse position
 - mouse up defines “closure” of the conceptual unit or “operation”



Choices: Granularity

- Rules of thumb:
 - Do not record actions while actively interacting with a control.
 - Example:
 - Chunk *all* changes made in one user interface event into a single undo action.
 - Example:
 - Break input up based on discrete breaks in the input
 - Example:

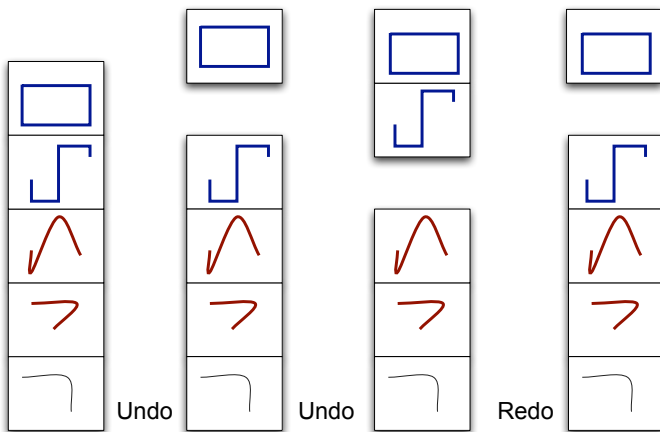


Choices: Implementation

- Need to keep a history of operations
- Undo:
 - Remove the most recent operation from the history
 - Restore the state to before the most recent operation
- Redo:
 - Reapply the most recently “undone” operation
 - Not available if there is no undone operation



Choices: Implementation



Choices: Implementation

- Two approaches to updating the model after an undo or redo:
 - Baseline and forward undo
 - rebuild the model from a known (saved) state by reapplying each operation in a forward direction
 - Command Objects and backup undo
 - for each operation, remember how to do it and how to undo it
 - Example:



Choices: Context

- Based on the previous illustration, we need two stacks. Where should they be kept?
 - System level?
 - Application level?
 - Document level?
 - Control level?
- Example: A form in Firefox vs. a form in Safari
- Choices impact your underlying implementation.

Choices: Context

- Option 1: associate an undo stack with each self-contained component of the interface.
 - Example: Firefox's handling of individual text fields.
- Option 2: associate an undo stack with each document's model in the MVC architecture.
 - Implications for multi-document applications?
 - Simplified conceptual model for the user: Edits are associated with an *overall* document rather than specific controls in the user interface.

Choices: Undoable Actions

- Some things can't be undone:
 - Printing, Saving
 - Quitting program with unsaved data
 - Emptying trash
 - Ask for confirmation before doing a destructive, undoable, operation
- What about...
 - Changes to selections?
 - Window resizing?
 - Scrollbar positioning?
 - Example: Photoshop

Choices: Undoable Actions

- Rules of Thumb:
 - Any and all changes to a document's content should be undoable.
 - Changes to a document's *interface state* should be undoable if they are extremely tedious or require significant effort.

Choices: State Restoration

- What user interface state is restored after an undo or redo?
 - Compare OmniGraffle and TextEdit/TextMate
- Rules of Thumb:
 - User interface state should be meaningful after undo/redo action is performed.
 - Change selection to object(s) changed as a result of undo/redo. Scroll to show selection, if necessary.
 - Give focus to the control that is hosting the changed state.
 - These actions help users understand the result of the undo/redo operation.

Summary: Available Choices

- Granularity: how much should be undone at a time?
- Implementation: how do you do it?
- Context: what is the scope of an undo operation?
- Undoable actions: what can't/isn't undone?
- State restoration: how do you actually do it?
- If in doubt:
 - test the implementation with real users.
 - See if they find the choices made in undo semantics intuitive in the context of their work.

Implementation in Detail

- Saving and restoring state
- Model responsibility vs. UI responsibility
- Demo Code
 - Cocoa
 - Java

Impl: Saving & Restoring State

- For each operation (“chunk” of input from the user), place an object on the undo/redo stack.
- Do undo the operation, pop it off the stack and execute it.
- What’s the name of this Design Pattern?
- Example:
 - `someOperation.undo();`
 - `someOperation.redo();`

Impl: Saving & Restoring State

- The operation/command object restores a previous state in one of two ways:
 - Save changes to the state
 - Save the state
- Save changes to the state: typical in many cases
 - Word Processor
 - Vector drawing program
 - When doesn’t this work?

Impl: Saving State

- Consider a bitmap painting program
 - Do red stroke
 - Do black stroke
 - Undo
- If all we do is save the command to create/remove the black stroke, what is the result?
- Need to save at least part of the image that existed before the stroke was made.
 - Might require a lot of memory!



Impl: Saving & Restoring State

- If you can forward-correct an action (that is, perfectly restore from a previous state through actions alone), then just save the operations.
 - Exception: Operations that take a lot of time but don't take a lot of memory to save the change in state.
- If you cannot forward-correct an action (eg: cropping an image, paint-style drawing), you must save state so you can restore the previous state.
 - Options: store the entire state, or just the differences

Impl: Cocoa

- Uses Objective-C's dynamic nature to create an extremely elegant undo facility.
- Available by default in its document class (NSDocument) through NSUndoManager.

```

-(void) makeHotterBy:(int) increaseAmount {
    newTemperature += increaseAmount;

    // Record an "undo" by storing a call to makeColderBy
    NSUndoManager* undoManager = [myDoc undoManager];
    [undoManager prepareWithInvocationTarget:self];
    [undoManager makeColderBy:increaseAmount];
}

-(void) makeColderBy:(int) decreaseAmount {
    newTemperature -= decreaseAmount;

    // Record an "undo" by storing a call to makeHotterBy
    NSUndoManager* undoManager = [myDoc undoManager];
    [undoManager prepareWithInvocationTarget:self];
    [undoManager makeHotterBy:decreaseAmount];
}

```

25

Impl: Java

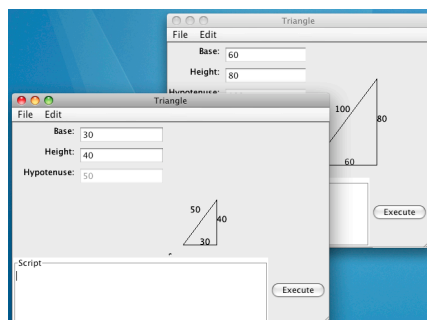
- Interfaces
 - StateEditable: implemented by models that can save/restore their state. Key methods: storeState, restoreState
 - UndoableEdit: implemented by command objects. Key methods: undo, redo.
- Classes
 - AbstractUndoableEdit: convenience class for UndoableEdit
 - StateEdit: convenience class for StateEditable; extends AbstractUndoableEdit. Key methods: init, end, undo, redo
 - UndoManager: container for UndoableEdit objects (command pattern). Key methods: addEdit, canUndo, canRedo, undo, ...
 - CompoundEdit: "A concrete subclass of AbstractUndoableEdit, used to assemble little UndoableEdits into great big ones."



Undo 26

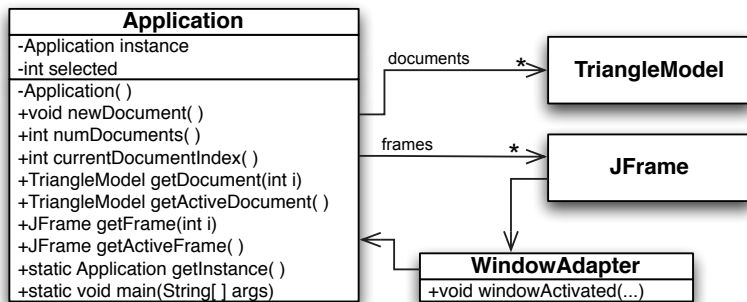
Undo Example: Triangles

- Demo
 - Now have multiple documents
 - Three views:
 - textview (simple to undo/redo)
 - graphical view (more complex)
 - scripting view (later topic).
 - Complete code will not be released, but lots of details in the following slides that will be posted.



Undo 27

Ex: Application Class



```

public void newDocument() {
    final TriangleModel model = new TriangleModel();
    final UndoMgr undo = new UndoMgr();

    JFrame frame = this.makeView(model, undo);

    this.documents.add(model);
    this.frames.add(frame);
    this.selected = frames.size()-1;

    frame.setVisible(true);
}
  
```

```

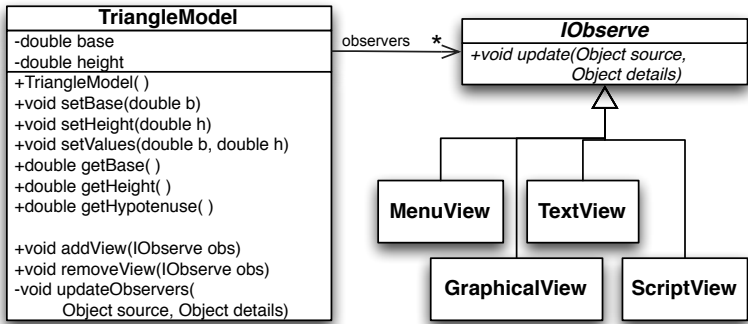
private JFrame makeView(TriangleModel model, UndoMgr undo) {
    GraphicalView vGraphical = new GraphicalView(model, undo);
    TextView vText = new TextView(model, undo);
    ScriptingView vScript = new ScriptingView(model, undo);
    JMenuBar menus = new MenuView(model, undo);

    JFrame frame = new JFrame("Triangle");
    // layout the frame here

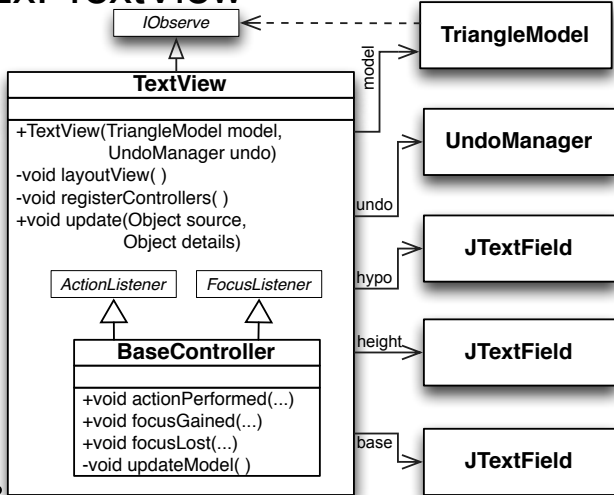
    // When this frame is activated, record new active window
    frame.addWindowListener(new WindowAdapter() {
        public void windowActivated(WindowEvent e) {
            selected = frames.indexOf(e.getWindow());
        }
    });

    return frame;
}
  
```

Ex: TriangleModel



Ex: TextView



```

pri... class BaseCon... impl... ActionListener, FocusListener {
public void actionPerformed(ActionEvent evt)
{ this.updateModel(); }

public void focusGained(FocusEvent evt)
{ baseTF.selectAll(); }

public void focusLost(FocusEvent evt)
{ this.updateModel(); }

private void updateModel() {
double oldBase = model.getBase();
double newBase = Double.parseDouble(baseTF.getText());
if (newBase != oldBase) {
UndoableEdit cmd = new TriangleBaseUndoableEdit(
model, oldBase, newBase);
undo.addEdit(cmd); // from the outer class
cmd.execute();
}
}
}
}

```

```

public class TriangleBaseUndoableEdit extends AbstractUndoableEdit {
    private TriangleModel model;

    // We have to store both old and new values (rather than just
    // a delta) because the model may reject or modify a value
    // that we set. That would throw off a cumulative change.
    protected double oldBase;
    protected double newBase;

    private TriangleBaseUndoableEdit(TriangleModel model,
        double oldBase, double newBase) {
        this.model = model;
        this.oldBase = oldBase;
        this.newBase = newBase;
    }
}

```

34

```

public void undo() {
    this.model.setBase(this.oldBase);
}

public void execute() {
    this.model.setBase(this.newBase);
}

public void redo() {
    this.execute();
}
}

```

35

Ex: Undo for Varying Changes

- The GraphicalView has changes that are similar to the TextView: modify the controller(s) to
 - capture the old values of the base and height on MousePress
 - capture the new values of the base and height on MouseRelease
- Additional checks for whether changes in both X and Y are allowed, or only in X.

Ex: Keeping Undo Menus Current

- We need to keep the undo/redo menus current. UndoManager has methods canUndo and canRedo to make this easy, but no way to inform observers.
- Therefore, extend UndoManager
- Add an Observer from the MenuView



Undo 37

```
public class MenuView extends JMenuBar {
    private TriangleModel model;
    private UndoMgr undo; // UndoManager extended to handle observers
    private JMenu file = new JMenu("File");
    private JMenu edit = new JMenu("Edit");

    // Actions can be interpreted by menus, toolbars, etc.
    private AbstractAction newAction = new AbstractAction("New") {
        public void actionPerformed(ActionEvent e) {
            Application.getInstance().newDocument();
        }
    };

    private AbstractAction undoAction = new AbstractAction("Undo"){
        public void actionPerformed(ActionEvent e) {
            MenuView.this.undo.undo();
        }
    };
};
```

38

```
...
// Undo manager should inform observers when a command is
// added or it performs an undo or redo. We then update
// menus.
this.undo.addObserver(new IObserve() {
    public void update(Object subject, Object detail) {
        undoAction.setEnabled(MenuView.this.undo.canUndo());
        redoAction.setEnabled(MenuView.this.undo.canRedo());
    }
});

...
// Set accelerator keys for the menu items.
this.undoAction.putValue(Action.ACCELERATOR_KEY,
    KeyStroke.getKeyStroke(KeyEvent.VK_Z,
        ActionEvent.META_MASK));
this.redoAction.putValue(Action.ACCELERATOR_KEY,
    KeyStroke.getKeyStroke(KeyEvent.VK_Z,
        ActionEvent.META_MASK |
        ActionEvent.SHIFT_MASK));
}
```

39

Ideas for Improving Undo

- Branching Histories
 - Fully record every state that is visited
 - Issues
 - User may not want every state saved
 - No real elegant interfaces for browsing the histories
- Editable Histories
 - Directly edit past state; changes propagate down
 - Issue: changes made earlier in history may result in incompatible states later in the history.

Scripting

- The command objects in the undo/redo stacks can form the basis for scripting the application.
- Need methods to:
 - parse text input (eg from a file) into appropriate command objects
 - hard part is figuring out how to refer to specific parts of the model
- More about scripting using interpreters in a future lecture