UNDO/REDO

CMPT 381

Outline

- The need for Undo
- Undo/Redo development issues
- Architectures for Undo/Redo
  - Baseline + Forward Undo
  - Backup Undo
- Scripting and Versioning

Why do we need Undo?

- A critical part of *direct manipulation*
  - Ben Shneiderman 1983
  - Principles:
    - Visibility of objects
    - Incremental action and rapid feedback
    - **Reversibility**
- Undo means users can fix errors
- Undo means users can explore
  - "fire and fix" operation

Undo/Redo Issues

- Pervasiveness: which actions are undoable?
  - Just model actions?
  - Interaction model actions? (e.g., selection, UI state)
- Granularity: how to represent actions?
  - 100 small contiguous move actions, or
  - 1 large start-to-finish action?
- Is undo needed?
  - Not if action can be reversed in the UI (e.g., scrolling)
  - How much user effort would be lost?
    - E.g., selections in Word vs. Photoshop

Architectures for Undo/Redo

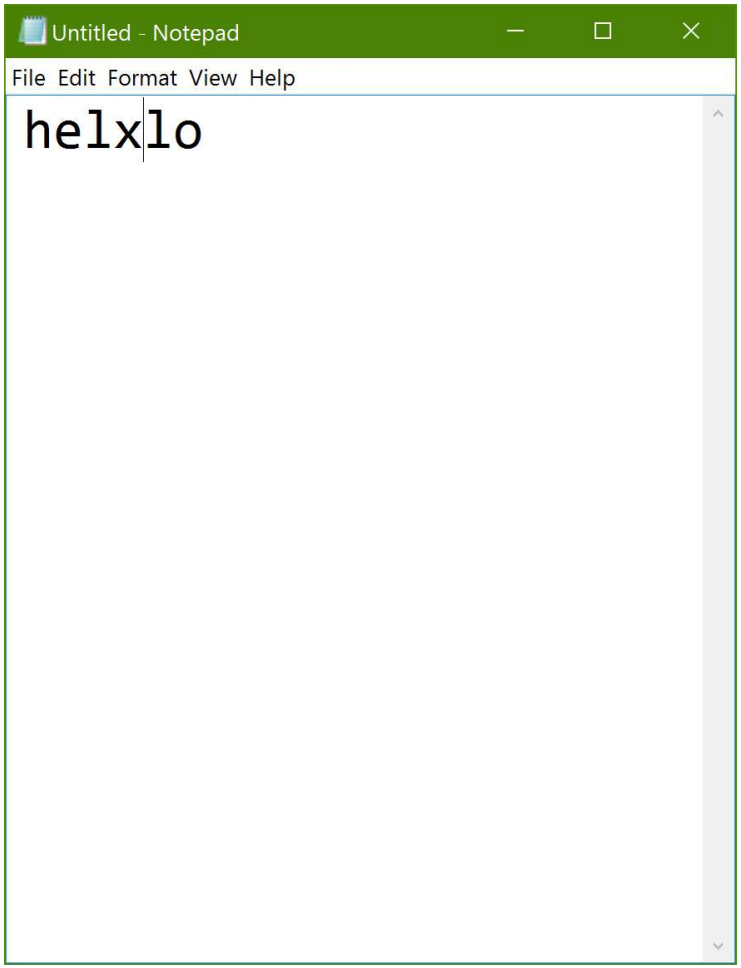
- Baseline + Forward Undo
- Backward Undo

Baseline + Forward Undo

- Stores original model (the baseline)
- Stores each action
  - Represented by the change to the model
- To undo last change:
  - Remove it from the change history
  - Recreate the model from the baseline

Baseline+Forward in a text editor

- Two commands:
  - insert(start,newString)
  - delete(start,end)
- Baseline:
  - Empty model
- Current history:
  - insert(0,”hekko”)
  - delete(2,4)
  - insert(2,”ll”)
  - insert(3,”x”)



Issues for Baseline + Forward

- Limitations:
  - Must recreate the model from baseline every time
    - Inefficient if many commands in the history
  - Each model-change method must log the action
- Possibilities:
  - Action records also provide a scripting mechanism
  - Multiple-action tasks can be recorded for later
    - E.g., “insert address to document” can be recorded

Backward Undo

- Restore the model to what it was before the change
  - More efficient than rebuilding from baseline
  - Need a transformation that is the inverse of the change
- Command pattern / Command objects
  - Create a command class for each model action
  - Object can run the action forward or backward
    - Forward: doIt()
    - Backward: undo()
  - Each class responsible for obtaining the information needed to achieve both directions of operation

Command interface

```
public interface Command {
  public void doIt( )
  public void undo( )
}
```

Backward Undo – JavaFX demo