Cut, Paste, Drag-and-Drop

Benefits, data formats, and Java implementation

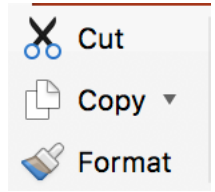
Direct Manipulation Principles

- There is a visible and continuous representation of the domain objects and their actions. Consequently, there is little syntax to remember.
- The instruments are manipulated by physical actions, such as clicking or dragging, rather than by entering complex syntax.
- Operations are rapid and incremental
- Their effects on domain objects are immediately visible.
- Reversibility of (almost) all actions
- Users can explore without severe consequences
- Operations are self-revealing
- Syntactic correctness – every operation is legal

(from User Interface Design & Evaluation, p. 213-214)

Transferring Data

- Early uses of Direct Manipulation treated text as a common domain-object.
  - Early metaphor: cutting (using scissors) and pasting blocks of text
  - This has been extended to other types of content
  - e.g. MS Office toolbars *still* use scissors to represent cut operations.



- Cut and paste and drag and drop allows for (relatively) easy data transfer within and between applications
  - This has become standard and expected behaviour, for most types of data in DM interfaces.

The clipboard

The clipboard is a “holding area” for data that you are moving within or between applications (ubiquitous, “standard” design).

- Copy information (or pointer to information) to clipboard
- Other applications can read data from the clipboard

Issue: any application can read this information

- Is this a potential security risk?
- Clipboard is deliberately not accessible to apps running in a web browser (“sandboxing”)

Issue: The clipboard requires common data formats to work seamlessly

- Text is no problem
- What about other formats?

Consider graphics, as a more complicated example than text.

How do we deal with:

- Drawings in vector-based drawing programs?
- Bitmap images?
- Images from different file formats (JPEG, TIFF, GIF…)
- 3D graphics?
- PostScript drawings?
- Charts?
- Proprietary graphics formats?

The Clipboard

Solution: When data is placed on clipboard, application indicates the formats in which it can provide the data

- Example: “I can provide a vector image, bitmap image, or text”
- Simplest case: immediately place each supported data format on the clipboard, most preferred to least preferred

Mac Human Interface Guidelines specify that all applications

- put either plaintext or an image on the clipboard
- accept both plaintext and images from the clipboard
- should always be able to cut/paste something!

Design: data is not always copied to the clipboard immediately

- Why not?
- What are implications?

Placing Data on Clipboard

Data may be available in many formats

- Wasteful to put all formats on clipboard at once

Data may never be pasted

- Again, wasteful to commit memory to a copy unless it is needed
- Particularly after a “cut” operation, which can be used in place of delete …

However, if data is not immediately placed on clipboard:

- Must create a copy if user changes data locally
- Must put it on clipboard if application exits
  - Or at least prompt user

Clipboard a function of the underlying windowing system, toolkit

- Java will do it differently from Cocoa from Windows…

Java Clipboards

Relevant packages:

- java.awt.datatransfer (Clipboard, Drag and Drop)
  - Clipboard
  - DataFlavor
  - Transferable
  - Toolkit

Local and system clipboards

Local clipboards are named clipboards holding data only accessible by the current application

- new Clipboard(“My clipboard”);
- Local clipboards often used to maintain all of the formatting information when cutting/pasting within the same program. Use a system clipboard for transfer between programs.

System clipboard is operating-system-wide clipboard

- `Toolkit.getDefaultToolkit().getSystemClipboard()`

Copying Data to Clipboard

Basic steps:

- Get clipboard
- To copy, create a Transferable object
  - Defines methods for responding to queries about what data formats (DataFlavors) are available
  - Defines method for getting data of specified type
- Set clipboard contents to the new Transferable object

Transferable object encapsulates all the data to handle the copy operation later

- Similarities to what other object?

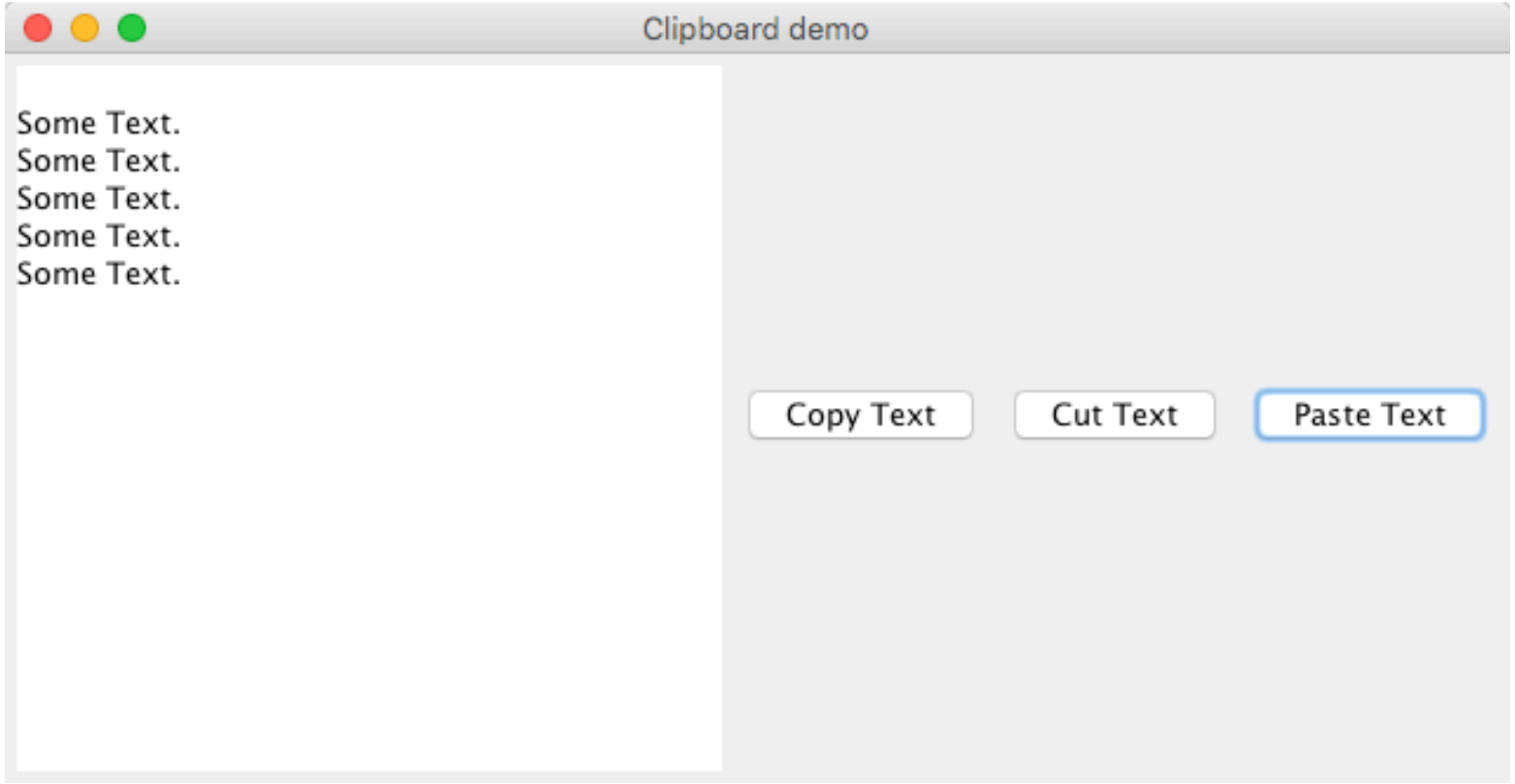
Transferable

- Encapsulates all data to copy in an object
- Similar in spirit to UndoableEdit
- Methods:
  - DataFlavor[ ] getTransferDataFlavors( )
  - boolean isDataFlavorSupported(DataFlavor flavor)
  - Object getTransferData(DataFlavor flavor)

Pasting Data from Clipboard

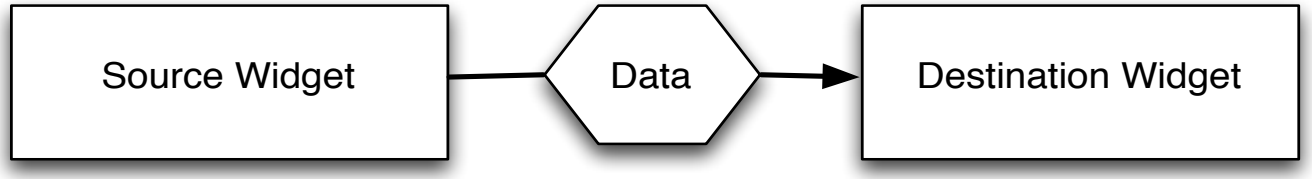
- Basic steps:
  - Get Clipboard
  - Get the Transferable from the Clipboard
    - See if it supports the desired format (DataFlavor)
    - Get the data, casting it to the proper Java object

Code Demo: Cut-and-Paste



Drag ‘n Drop

- Uses same Transferable, DataFlavor objects to pass information around
- Attach a TransferHandler to each widget to manage data transfer
- Need to detect the start-of-drag gesture



- has a TransferHandler that implements createTransferable and exportDone
- has a mouseListener to detect the start-of-drag gesture

- has a TransferHandler that implements importData

Supporting Drag ‘n Drop

“Dragging” refers to copying something from your control;
“Dropping” refers to receiving data dragged from elsewhere

To support dragging:

- Set a transfer handler for each component that supports dragging
- In the source of the drag, define a mouse listener that knows when a drag has started.
- When a drag has started, get the component’s transfer handler and call its exportAsDrag method
- Override createTransferable and exportDone to do the right thing

To support dropping:

- Set a transfer handler for each component that supports dropping
- Override the importData method

Summary

- Make data available in a variety of data formats, ordered by the provider's preference.
- Target chooses the most applicable format to use.