Drag-and-Drop

- A key user action with direct manipulation is *drag-and-drop*, which allows a user to "grab" an interface object, then freely move it across the display, finally "releasing" it at some meaningful location or target
- Drag-and-drop activities typically represent *move* or *copy* operations, which makes sense since the "real world" action being performed *is* a literal move
- Drag-and-drop is actually a specific instance of a more general mouse gesture of press-hold-move-release, along with rubberbanding, drag-adjusts, and more

Three Approaches to Drag-and-Drop

- *Intra-component:* Can be coded entirely from scratch; requires no interaction with other components or the window manager/operating system
- *Inter-component, intra-application:* Sometimes doable from scratch, though some assistance from the user interface framework makes it a little easier
- *Inter-application:* APIs are typically provided by the window manager or operating system, particularly for facilitating data transfer (thus, these are usually the same APIs for performing cut/copy/paste)

Anatomy of a Drag-and-Drop

- At the drag source:
  - Detect the initiation of the drag activity (must coordinate with other mouse actions such as selects)
  - Determine (and construct) the data representation(s) of the item being dragged
  - When the drag completes, perform some clean-up if necessary (e.g., if the drag activity represents a "move," then the dragged item needs to be removed from its original location)

- At the drag destination:
  - Detect an "offered" drop-off event, and provide feedback on the offer (selection, insertion point)
  - Examine the data that is being "offered" to see if the destination can handle it, and also provide feedback (change mouse icon, etc.)
  - Perform the operation that is represented by the drop-off gesture (transfer data, change state, etc.)
- At the operating system, window manager, or other overarching framework
  - Coordinate activities between the drag source and possible drag destinations (feedback, clipboard, etc.)

Drag-and-Drop as Data Transfer

- One of the most frequent uses of drag-and-drop is data transfer: moving a file from one place to another, placing a Web URL in a bookmarks bar, etc.
- Thus, not surprisingly, in many frameworks (including Swing), drag-and-drop is closely associated with other data transfer functionalities such as the clipboard
- Point to ponder: if you have full-featured cut/copy/paste functionality, then why bother with drag-and-drop? Is the implementation of drag-and-drop (which isn't trivial, as you'll see) worth the trouble then?

Data Transfer Concepts

- When data transfer activities (cut/copy/paste, drag-and-drop) cross application boundaries, we must ensure that the transferred data is meaningful to all applications involved — otherwise, why bother?
- Thus, a key idea in data transfer is support for *multiple representations* of the same item; for example, dragging an interface object for a file may transfer its name, its full path, its URL, or its entire content
- The recipient is expected to take the "best fit" representation of the transferred item

Drag-and-Drop in Swing

- All three types of drag-and-drop (intra-component, inter-component/intra-application, inter-application) are doable in Swing
- Some built-in Swing components support drag-and-drop "out of the box"
- Swing and the Java virtual machine coordinate so that data transfer can occur across Java and native applications — generally for free, as long as you use the APIs correctly

Intra-Component Drag-and-Drop in Swing

- Not surprisingly, creating custom components that can drag-and-drop within themselves is pretty much done entirely from scratch
- General approach: create a *controller* class that is both a MouseListener and MouseMotionListener, then add an instance of this class as a listener to your component; all drag-and-drop logic lives in this object
- Main activities: *mousePressed()* sets the initial state for the drag; *mouseDragged()* updates the status of the drag; *mouseReleased()* performs the "drop"

Inter-Component Drag-and-Drop in Swing

- First level: built-in inter-component drag-and-drop for selected built-in components (text fields, lists, tables, color choosers, file choosers — check online)
  - Check to see if inter-component drag-and-drop is available "out of the box" (if the component has *setDragEnabled()*, then it has some sort of built-in drag-and-drop logic)
  - Setting *dragEnabled* to true activates the built-in functionality — which may or may not fit what you want, but at least it's there for you to use

- Second level: customized (or enhanced) drag-and-drop for built-in components
  - Extend *TransferHandler* to manage and encapsulate the drag-and-drop logic (an easy special case is available when the setter property is known and standard)
  - Implement a *Transferable* to hold the information that is being dragged, using one or more *DataFlavor*s
  - Install mouse listener to detect and initiate drag-and-drop activity (by invoking the transfer handler)
- Replacing the transfer handler of a component that has built-in drag-and-drop will overwrite that functionality — if you don't want to lose it, you'll need to fall back on or reimplement the built-in transfer handler logic

Inter-Application Drag-and-Drop in Swing

- Inter-application drag-and-drop in Swing actually doesn't require any additional handling: *TransferHandler* and *Transferable*, through the Java virtual machine, will interact with the host system's own drag-and-drop/ data transfer functionality
- Orthogonality: cut-copy-paste uses the same API
- For additional details and sample code, visit the drag-and-drop/data transfer tutorial at this URL:

http://java.sun.com/docs/books/tutorial/uiswing/dnd/intro.html

Swing Drag-and-Drop Odds and Ends

- *Transferable*'s data representations are made portable via serialization; make sure your transfer objects implement *Serializable*
- The *TransferHandler* method *getVisualRepresentation()* is supposed to determine how a dragged object should look while it is in transit — at this writing, it actually doesn't work (bug #4816922)
  - The default visual representation is generally OK; the issue is that you can't change or customize it until this bug gets fixed