Problem: avoid inter-widget dependencies

- When widgets talk directly to each other the GUI is hard to modify
  - removing/changing a widget has (severe) repercussions
  - widget code clutters up the GUI class
  - solution: make widgets into classes, encapsulate state and widget behavior into a class, often derived from a widget class: e.g., see `FileList`, `FileDisplay` in scoogui
- How do application and view/GUI communicate with each other?
  - Direct communication leads to very tight coupling, hard to modify classes
  - use a mediator/controller class through which all communication is processed
Mediator (GOF 273)

- **Problem: when to use mediator pattern:**
  - lots of objects communicate in well-defined, but complex ways. Many interdependencies exist, unstructured, hard to get a big picture.
  - Object re-use is hard because each object is so specialized and refers to lots of other objects
  - behavior is distributed among several classes

- **Mediator is controller, colleagues are objects/widgets**
  - colleagues are decoupled
  - control is centralized
  - simplifies colleague/widget protocols
  - colleague/widget classes are more re-usable
scoogui: mediator and widgets (also see MVC)
Library information

● **Library**: compiled .o files grouped together
  ➤ linked (at compile or run time)
  ➤ pre-process, compile, link: where are .h files used?
  ➤ order of linking/loading libraries matters, only needed source is linked

  -ltapestry -lscoobi vs. -lscoobi -ltapestry

● **Static library**: linked at compile time, **Dynamic/Shared library**: linked at run time
  ➤ not all systems/architectures support dynamic libraries
  ➤ either the code or the environment must know where shared library is located: -R gmake or **LD_LIBRARY_PATH** environment variable