Patterns as solutions to problems

- A design pattern is the solution to a problem in a context
  - Has a name that helps in remembering/understanding
  - Has forces that describe the situations in which applicable
    - Should supply pros/cons in using the pattern
  - Has a description that summarizes purpose
- Adapter
  - You have a class that's close to what you want, but the interface isn't quite right, or some functionality is missing
  - Use an *adapter*, adapt the existing class to a new interface
  - Also known as *wrapper*, similar to Proxy but changes interface/adds functionality

Decorator

- Add responsibility to object dynamically, flexible alternative to subclassing for adding functionality
  - Add responsibility to objects without affecting other objects (transparently)
  - Remove responsibilities
  - Extension by subclass impractical (subclass explosion)
- Component is the base class
  - Decorator is a component that contains a component
  - Used "as-a" component, decorates and forwards

Inheritance and STL in OOLS

- Sorter (Comparer) and Filter objects
  - Appear to use inheritance, `virtual operator ()`
  - In STL inheritance rarely (never?) works
    - Template parameters don't support inheritance
    - Objects often copied/passed-by-value
- Solution? Use base-class through decoration by subclass
  - Base class maintains pointer to "real" sorter
  - Base class function always used, forwards to virtual
- Look at Filter and Sorter base class implementations
  - How does storing `this` work?

`getopt_long` details

- `#include <getopt.h>` works in Eclipse, but requires header file on Unix system
  - Implementation linked in by –liberty (libiberty.a)
  - In Eclipse part of standard g++/mingw libraries
- See oolsmain.cpp for details on initializing structures and calling function to process options
  - Notice requirement for short args, could be generated automatically (see header file for `struct option`)
- Switch statement is frought with peril, but liveable
  - Alternative, map to commands