CS 342: Object-Oriented Software Development Lab

Lab 6 Intro

David L. Levine
Christopher D. Gill
Department of Computer Science
Washington University, St. Louis
levine,cdgill@cs.wustl.edu

http://classes.cec.wustl.edu/~cs342/
Lab 6: Observable Sort

- Build a Sorter that:
  - sorts input contained in an `Array<T>`
  - encapsulates the data and other information in a `Sort_Info` object
  - uses selection sort (in this Lab)
  - allows multiple Observers of the sort results (and intermediate progress, Lab 7)
Lab 6 Collaborators

1. create_sorter
2. create
3. register_observer, request Sort_Info response
4. start_sort
5. publish_sort_update
6. update_sort
Sorter Public Interface

// Manager functions.
static Sorter *create_sorter (const Sort_Info_Base &);
// Factory Method for creating a Sorter.

// Implementor functions, for use by Sorter clients.
int register_observer (Sort_Observer &sort_observer);
// Register an observer for the sort. Returns 0 on success,
// -1 on failure (reached maximum number of observers).

int virtual start_sort ();
// Starts the sort operation. Returns 0 on success,
// or -1 if the start had already been started.
Sorter Protected Interface

// Implementor functions, for use by Sort algorithms.
void virtual publish_sort_update (
    const Sort_Response &sort_observation);
// Used by Sort algorithms to publish updates on Sort
// status to registered observers.
Sort_Info_Base Class

- Non-template sort info interface

```cpp
enum Status { UNSORTED, PARTIALLY_SORTED, SORTED };

enum Sort_Type {
  ANY_SORT,  // Allows Sorter to select the sort algorithm
  SELECTION_SORT,
  MERGE_SORT,
  QUICK_SORT
};
```

virtual void print (ostream &) const = 0;
virtual void read (istream &) = 0;
// Pure virtual, but can still have
// implementations in this base class.
Sort_Info Template Class

- Derived from Sort_Info_Base, and adds data (in Array)

**public:**

// Manager functions.
Sort_Info (const Sort_Type sort_type,
           const Array<T> &data,
           const Status status = UNSORTED);
// Constructor. Copies the data to an internal Array.

virtual ~Sort_Info () ;

virtual void print (ostream & ) const;
// Output the contents to the specified stream.

virtual void read (istream &);
// Read the object from the stream. Must be compatible with // print (ostream &).

private:
    Array<T> data_; // Elements to be sorted.
Callbacks

class Sort_Observer
{
    public:
        // Manager functions.
        virtual ~Sort_Observer () ;

        // Implementor functions.
        void virtual update_sort ( 
            const Sort_Info_Base &sort_observation) = 0 ;
        // Callback, to receive a sort observation.
};
Concrete Sort_Observer

class Main_Observer : public Sort_Observer
{
public:
    Main_Observer () {}
    virtual ~Main_Observer ();

    void update_sort (const Sort_Info_Base &sort_observation)
    {
        if (sort_observation.status () == Sort_Response::SORTED)
        {
            // print out the sorted array here
        }
    }

main (int, char *[])
{
    Sorter *si = create_sorter [...];

    Main_Observer main_observer;
    si->register_observer (main_observer);
    si->start_sort ();
}
Sort_Viewer_Client Interface

- C++ interface to Java viewer
- Provides three functions for viewer interaction:
  1. establish a connection to a Java viewer (defaults to a viewer listening at port 3420 on the local machine)
  2. send a state update to the viewer
  3. disconnect from the viewer
Sort Viewer Client connect_to_viewer

/*
 * Connects to the Java viewer on the host, at the given port.
 * fs is an fstream that can be used to send messages to the
 * viewer, using send_to_viewer ().
 * sockfd is set to the socket’s file descriptor. It must
 * closed by the caller, using disconnect_from_viewer ()
 * when the viewer connection is no longer needed.
 * Returns 0 on success, -1 on failure.
 */

int
connect_to_viewer (int &sockfd,
            fstream *fs,
            const char *const host = JAVA_VIEWER_HOST,
            const int port = JAVA_VIEWER_PORT);
Sort_Video_Viewer send_to_viewer

/*
 * Use print () virtual function to send Sort_Response to
 * the viewer.  handle was created by connect_to_viewer()
 */

int
send_to_viewer (fstream *fs, const Sort_Info_Base &sort_info)
{
    sort_info.print (*fs);
    return 0;
}
Sort_Viewer_Client disconnect_from_viewer

```c
int disconnect_from_viewer (const int sockfd, fstream *fs)
{
    if (close (sockfd))
    {
        perror ("close");
        return -1;
    }

    delete fs;

    return 0;
}
```