DESIGN PATTERNS : UE18CS341

Project ID - 29

LOGGING SYSTEM : An Improvement on the Observer Pattern

Name	SRN
Anant Thazhemadam	**PES2201800014**
Afreen A Noorani	**PES2201800385**
Roshini Bhaskar K	**PES2201800122**

1. Abstract

We have implemented a variation of the Observer Pattern to mimic the logging system dependencies. Here Libraries usually depend on other libraries and updating any one Library leads to the change of states in all the Libraries depending on the one being changed. The state updates are cascaded through the libraries which resembles the Daisy Chain Pattern.
Singleton and Mediator Patterns concepts have been included into the implementation of Observer Pattern.
Concept of partial dependency where an observer depends only on a state of the Subject has also been implemented.

2. Interface

Clients can import the *logo* library which holds all the definitions for creating objects belonging to the *library* class and the *coordinate* and *line* class.

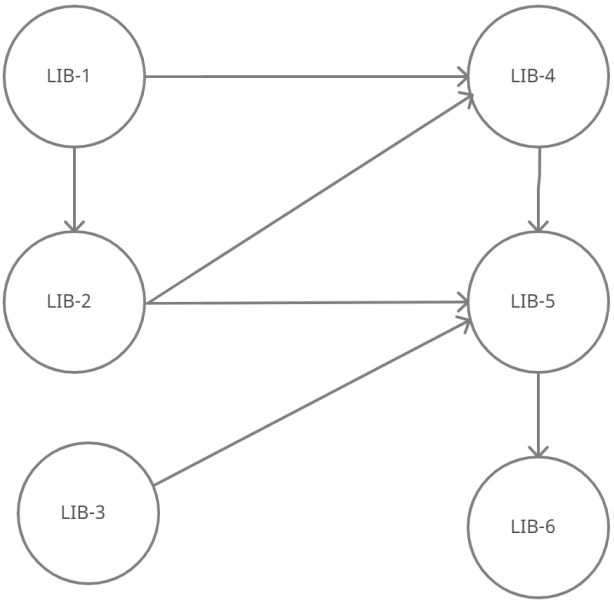
➔ ***Creating Libraries***

```
//creating objects
Library library_1("Imported Library", "LIB - 1", "1.3.8");
Library library_2("Imported Library", "LIB - 2", "3.3.7");
Library library_3("Imported Library", "LIB - 3", "4.6.9");
Library library_4("Imported Library", "LIB - 4", "1.1.3");
Library library_5("Imported Library", "LIB - 5", "0.3.2");
Library library_6("Imported Library", "LIB - 6", "1.7.8");
```

//attaching

```
library_1.attach(&library_4);
library_1.attach(&library_2);
library_2.attach(&library_4);
library_2.attach(&library_5);
library_3.attach(&library_5);
library_4.attach(&library_5);
library_5.attach(&library_6);
```

//Visualizing these dependencies



Change is state of ***library 3*** results in change in ***library 5*** and ***library 6***.
Change is state of ***library 1*** results in the change of state in ***libraries 2,4,5 and 6*** .

➔ ***Creating Coordinates and Lines***

//creating coordinates object

```
Coordinates z1 = Coordinates(5, 2);
Coordinates z2 = Coordinates(2, 1);
```

//z1 and z2 are attached to l1 inside the operator = member function

```
Line l1 = Line(z1, z2, true);
l1.disp() // displays the slope between the two coordinates z1 and z2
```

Change in the coordinates z1 or z2 will result in change in state of l1.

3. Implementation

The variation of **observer** pattern is created using the following structure.

➔ ***class Subject and class Concrete Subject***

Subject is an abstract class with no state information. This class defines the virtual *attach*, *detach* and *notify* behaviours. These functions call the *register_* , *unregister_* and *notify* functions of the **singleton** class Change Manager.
Concrete Subject inherits Abstract Subject. This class defines the *get_state* and *set_state* functions.

➔ ***class Observer and class Concrete Observer***

Object is an abstract class with a virtual destructor and a pure virtual function called *update*.
Concrete Observer inherits Abstract Object. This class defines the update function and has its own member functions for *get_state* and *display*.

➔ ***Change Manager***

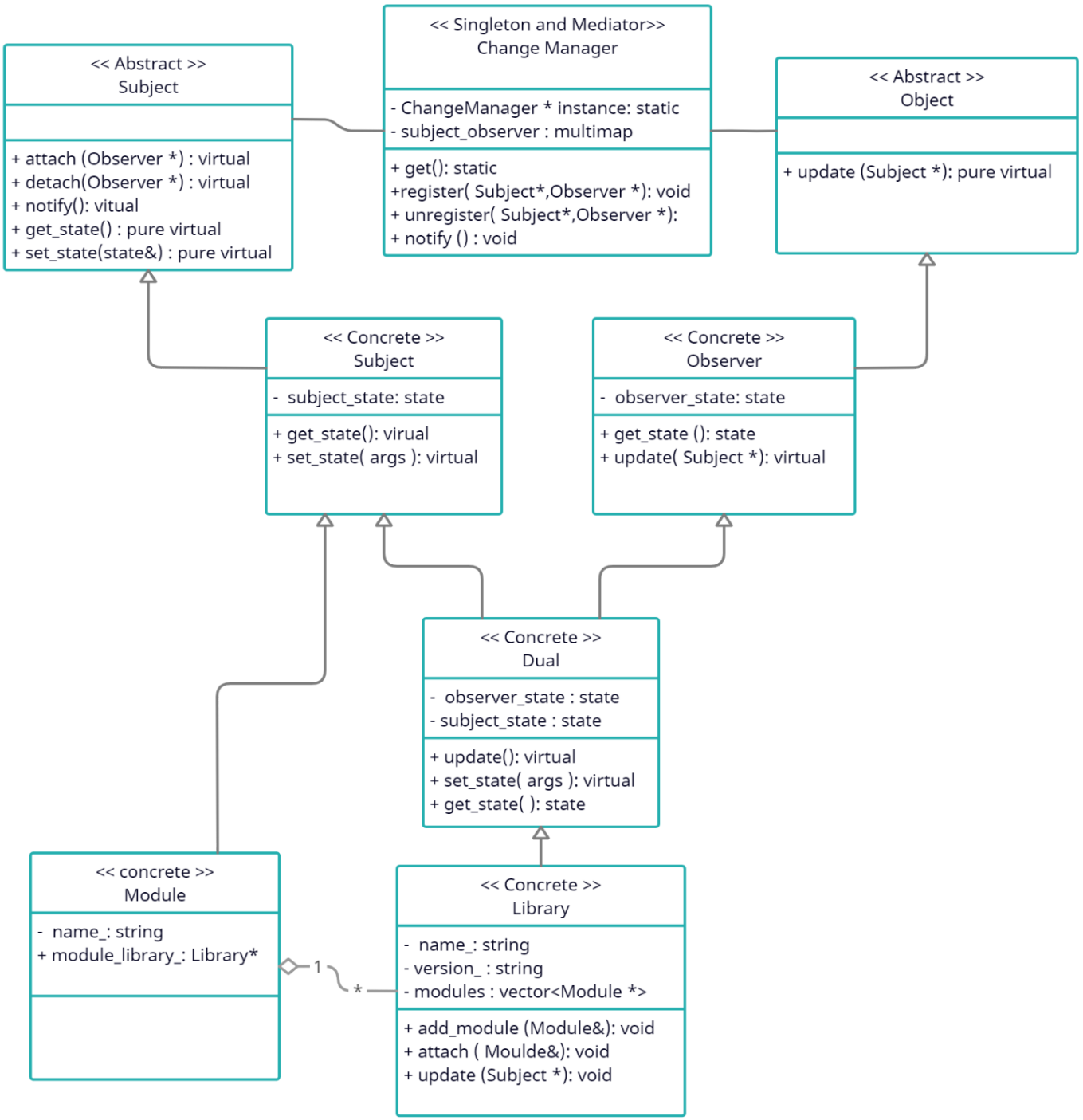
Change manager is a **singleton** class which is the **mediator** for Subjects and Observers. The singleton property allows the creation of only one Change Manager object through which all the communications are translated.
States that change manager holds are the ***subject_observer*** multimap and an additional ***lookup*** multimap that holds the mapping of the subject pointer and the dual pointers when behaving like a concrete subject.

- *lookup_convert*
  This is a member function of change manager which before registering an observer to the subject checks if the observer is the same as the subject. This check is necessary since a library object can behave as both subject and observer hence it would be meaningless for a library to attach to itself.
- *register_ method*
  This method inserts the subject and the observer that's being attached to it into the ***subject_observer*** multimap.
- *unregister_ method*
  Simply erases the observer from the subject in the ***subject_observer*** multimap.
- *notify method*
  Groups all the observers belonging to the subject in the ***subject_observer*** and calls update on all those observers.

➔ ***Library : Example 1***

This is an extension of the **observer** pattern where Library class inherits a class named Dual which is both an observer and subject. Dual class has definitions for both *update* and the *set_state* functions. The main purpose for creating a Dual class is to allow libraries objects to depend on each other.

A vector of pointers to Modules is stored in a Library class so each library knows which modules it stores.



- ***Modules to introduce partial dependencies between libraries object***
  Module class is a concrete subject that holds a pointer to library class.
- *add_module*
  This pushes the modules address into the vector *modules* in the Library class
- *attach*
  This attaches the *library_observer* to a module only (module.attach). And not the *library_subject* that holds the module. A partial dependency is maintained between *library_observer* and *library_subject*.
- *update*
  Simply calls update Subject of the parents method i.e Dual class update

➔ ***Geometry : Example 2***

Geometry example is built on the main idea of **observer** pattern. It consists of the following classes which builds the structure for the interface.

- ***Points***
  A structure type that takes in two coordinate points x and y and stores them as its state.
- ***Coordinates***
  Coordinate class inherits the Concrete Subject. Its state consists of *point_* which is of type Point structure.
  The *set_state* member function changes the *point_* and calls notify.
  Operator function for assignment is supported for this class. It takes care of self assignment before calling the *set_state* method.
- ***Line***
  Line inherits the concrete observer class. Its state includes *distance_* which calculates the distance between the coordinates and *slope_* that calculates the slope. Line class supports operator function for assignment which attaches the line(this *) to the coordinates *z1* and *z2*.
    - ***update function***
      The *update* function calls *update_slope* and *update_distance* which are member functions of Line class.

4. Snapshots

➔ Example 1:

- ***Library dependency (before review change)***

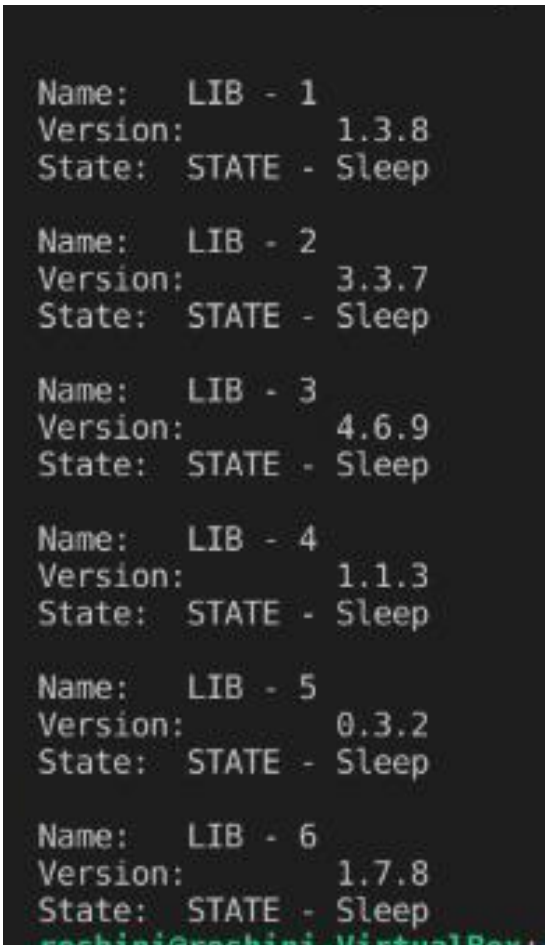
Creating libraries and dependencies

```
Library library_1("Imported Library", "LIB - 1", "1.3.8");
Library library_2("Imported Library", "LIB - 2", "3.3.7");
Library library_3("Imported Library", "LIB - 3", "4.6.9");
Library library_4("Imported Library", "LIB - 4", "1.1.3");
Library library_5("Imported Library", "LIB - 5", "0.3.2");
Library library_6("Imported Library", "LIB - 6", "1.7.8");

#ifdef DEBUG_DUAL
cout << "LIBRARY 1 : " << &library_1 << "\n";
cout << "LIBRARY 2 : " << &library_2 << "\n";
cout << "LIBRARY 3 : " << &library_3 << "\n";
cout << "LIBRARY 4 : " << &library_4 << "\n";
cout << "LIBRARY 5 : " << &library_5 << "\n";
cout << "LIBRARY 6 : " << &library_6 << "\n";
#endif

library_1.attach(&library_4);
library_1.attach(&library_3);
library_1.attach(&library_2);
library_1.attach(&library_1);
library_2.attach(&library_4);
library_2.attach(&library_5);
library_3.attach(&library_5);
library_4.attach(&library_5);
library_5.attach(&library_6);
```

Result when library 2 is changed :

```
Name:   LIB - 1
Version:        1.3.8
State:  Imported Library

Name:   LIB - 2
Version:        3.3.7
State:  STATE - LIQUID

Name:   LIB - 3
Version:        4.6.9
State:  Imported Library

Name:   LIB - 4
Version:        1.1.3
State:  STATE - LIQUID

Name:   LIB - 5
Version:        0.3.2
State:  STATE - LIQUID

Name:   LIB - 6
Version:        1.7.8
State:  STATE - LIQUID
-------------------------------------
```

Result when library 1 is changed : everything is updated



- ***Library Module wise updation***

```
Library library_1("Imported Library", "LIB - 1", "1.3.8");
Module new_mod("Module 1");
Module new_mod2("Module 2");

library_1.add_module(new_mod);
library_1.add_module(new_mod2);

Library library_2("Imported Library", "LIB - 2", "4.3.8");
Library library_3("Imported Library", "LIB - 3", "2.1.4");

std::cout << &library_1 <<"\n";
std::cout << &library_2 <<"\n";
std::cout << &library_3 <<"\n";
std::cout << &new_mod <<"\n";
std::cout << &new_mod2 <<"\n";

ChangeManager::get()->disp();


library_2.attach(new_mod);
library_3.attach(new_mod2);

ChangeManager::get()->disp();
```

Result :

```
Notifying 0x7ffe6518d6b8Module 2 Updated!

==========================================

Name:   LIB - 1
Version:        1.3.8
State:  Imported Library

Name:   LIB - 2
Version:        4.3.8
State:  Module 1 Updated!

Name:   LIB - 3
Version:        2.1.4
State:  Module 2 Updated!
roshini@roshini-VirtualBox:~/Desktop/DP/DP Project/loggo$
```

- ***Calculating slope between any two coordinates***

```
Coordinates z1 = Coordinates(5, 2);
Coordinates z2 = Coordinates(2, 1);
Coordinates z3 = Coordinates(2, 5);
Coordinates z4 = Coordinates(3, 1);



Line l1 = Line(z1, z2, true);

cout <<"\n---------------- \n BEFORE \n";
cout << l1;


l1.p_2_ = z3;


cout <<"\n---------------- \n AFTER \n";
cout << l1;
```

Result :

```
roshini@roshini-VirtualBox:~/Desktop/DP/DP Project/loggo$ ./loggo

---------------
 BEFORE

Coordinates 1:
x_ : 5
y_ : 2

Coordinates 2:
x_ : 2
y_ : 1

Distance:       4
Slope:  0.333333

---------------
 AFTER

Coordinates 1:
x_ : 5
y_ : 2

Coordinates 2:
x_ : 2
y_ : 5

Distance:       6
Slope:  -1
```