Program #6
Due: Thursday Dec 1st, 2016 at 11:30PM

Instructor	Dr. Stephen Perkins	
Office Location	ECSS 4.702	
Office Phone	(972) 883-3891	
Email Address	stephen.perkins@utdallas.edu	
Office Hours	Tuesday and Thursday 10:30am – 11:30am Tuesday and Thursday 1:00pm – 2:15pm and by appointment	
Grader	Section 502:	Sai Vamsi Muvva sxm154231@utdallas.edu Open Lab 2.103B1
	Section 504:	Gopichand Vanka gxv151030@utdallas.edu Open Lab 2.104A1 Tuesday/Thursday 3:00pm – 5:00pm

Purpose

Demonstrate the ability to create a program that utilizes the *Observer* design pattern. Demonstrate the ability to create abstract classes and implement derived classes. Demonstrate the ability to create and iterate over an *STL list* that contains callback functions.

Assignment

You will be creating a program that implements the *Observer* design pattern. This design pattern is utilized in almost all GUI systems and is the basis for distributed event handling. The goal of the program is to create a class (the Subject class for this assignment) that has a private variable (address) that can be modified via a standard mutator function (setAddress). This class has additional member functions that allow other classes (the observers) to register and deregister themselves with the Subject. If observers are registered with the subject, they will receive notifications (via a callback function) if the subject's address ever changes.

You are to create these observer classes BankObserver, SchoolObserver, CreditObserver. Each of the observers must be derived from this abstract base class:

```
class AbstractObserver
{
   public:
       virtual void subjectChanged(string)=0;
       virtual ~AbstractObserver(){}
};
```

Each should override the *subjectChanged* method by printing the string argument to the screen along with the name of it's class. For instance, the BankObserver might print the following:

The BankObserver received an address change notification: <string>

You will then create one instance of the *Subject* class and one instance each of three derived observer classes. You will **register** the instances of the observer classes with the instance of the Subject class. When registered, you will make a change to the instance of the subject class (using the *setAddress* method). This change should cause each of the registered observers to receive a callback with notification of the change. The *notify*() method implements this functionality. You must then **deregister** at least one of the observer instances and make a change to the subject instance. This will result in only the remaining registered observers receiving notification.

Here is the Class prototype for the Subject:

```
class Subject
{
	private:
		string address;
		list<AbstractObserver *> observers;
		void notify();

	public:
		Subject(string addr);
		void addObserver(AbstractObserver& observer);
		void removeObserver(AbstractObserver& observer);
		string getAddress();
		void setAddress(string newAddress);
};
```

Requirements

- Your code must extend and use the `AbstractObserver` class
- Your code must implement the Subject class
- Your code must exhibit the use of the *Observer* design pattern
- Your code must exhibit the use of the STL list data type
- Your code must exhibit the use of an STL list iterator
- Your code must exhibit correct operation with registered callbacks
- Your code must exhibit correct operation with deregistered callbacks

Deliverables

You must submit your homework through ELearning. You must include your source files.

No late homework is accepted.

Observer Pattern (http://en.wikipedia.org/wiki/Observer_pattern)

The **observer pattern** is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods. It is mainly used to implement distributed event handling systems. The Observer pattern is also a key part in the familiar Model View Controller (MVC) architectural pattern. [1] In fact the observer pattern was first implemented in Smalltalk's MVC based user interface framework.[2] The observer pattern is implemented in numerous programming libraries and systems, including almost all GUI toolkits.