Observer pattern program

Program description

The program implements 4 types of forms, a main (subject) form and 3 observer forms – text, rectangle and circle.

The main form allows the user to increase or decrease the size of an unshown variable, as well as open new instances of the observer forms (by default 1 of each type is opened when the program starts).

The observer forms visualize the unshown variable that you can change through the main form as either a number, or a rectangle or circle of varying size, and are notified of any changes to that variable.

Object description

1. ISubject – interface that defines the main subject methods – adding, removing and notifying observers, as well as a method to return its current state
2. IObserver – interface that defines the main observer methods – in this case only to update itself on the subject’s state
3. Counter – the subject in the observer pattern, implements the *ISubject* interface

* List<IObserver> observers – a list of all the observers (objects that implement the *IObserver* interface) currently attached to the subject
* Int count – the current state of the subject (integer that increases or decreases)
* Public Counter() – class constructor that initializes the *observers* and *count* variables
* Void Increment() – method that increases *count* by 1 and notifies all observers
* Void Decrement() – method that decreases *count* by 1 if it is bigger than 0 and notifies all observers
* Void RegisterObserver(IObserver observer) – adds a new object that implements the *IObserver* interface to the *observers* list and updates only the new observer on its current state
* Void UnregisterObserver(IObserver observer) – removes the provided observer from the *observers* list
* List<IObserver> GetObservers() – returns the *observers* list
* Void NotifyObservers() – updates all observers in the *observers* list on its current state
* Int GetState() – returns the current state

1. Main – the main form of the program

* Counter subject – an instance of the Counter class
* Counter Subject – a read-only property that returns the *subject* instance
* Public Main() – class constructor that initializes the *subject* instance and the program’s form, after which it creates 1 instance of a form of each observer type, displays them and attaches them to the current subject
* Void obsClosing(IObserver obs) – notifies the subject when an observer form is closing so the subject can detach them
* btnIncrease\_Click() – calls the *subject.Increment()* method
* btnDecrease\_Click() – calls the *subject.Decrement()* method
* btnNewTextObs\_Click() – creates a new observer instance of the text observer type and displays its form
* btnNewRectangleObs\_Click() – creates a new observer instance of the rectangle observer type and displays its form
* btnNewCircleObs\_Click() – creates a new observer instance of the circle observer type and displays its form

1. formText – form that corresponds to a text observer

* Main parent – contains a reference to the parent form (*Main*)
* Public formText(Main m) – class constructor which takes a reference to the form that initializes it (current instance of the *Main* form) and assigns it to *parent*, as well as initializes the form
* Void setText(int count) – sets the text of the label *lblText* in the form to the provided number
* Void UpdateObs() – calls the *parent.Subject.GetState()* method (*GetState()* method of the current subject of this form’s parent *Main* form) and passes the returned state (number) to the setText() method in this form
* formText\_FormClosing() – notifies the form’s *parent* that the form is closing

1. formRectangle – form that corresponds to a rectangle observer

* Main parent – contains a reference to the parent form (*Main*)
* Public formRectangle(Main m) – class constructor which takes a reference to the form that initializes it (current instance of the *Main* form) and assigns it to *parent*, as well as initializes the form
* Void updateRectangle(int count) – redraws the rectangle in the form using the value of *count* to determine its size
* Void UpdateObs() – calls the *parent.Subject.GetState()* method (*GetState()* method of the current subject of this form’s parent *Main* form) and passes the returned state (number) to the updateRectangle() method in this form
* formRectangle\_FormClosing() – notifies the form’s *parent* that the form is closing

1. formCircle – form that corresponds to a circle observer

* Main parent – contains a reference to the parent form (*Main*)
* Public formCircle(Main m) – class constructor which takes a reference to the form that initializes it (current instance of the *Main* form) and assigns it to *parent*, as well as initializes the form
* Void updateCircle(int count) – redraws the circle in the form using the value of *count* to determine its size
* Void UpdateObs() – calls the *parent.Subject.GetState()* method (*GetState()* method of the current subject of this form’s parent *Main* form) and passes the returned state (number) to the updateCircle() method in this form
* formCircle\_FormClosing() – notifies the form’s *parent* that the form is closing

3 problems

Reusability – The loose coupling and simple template should make reusability easy

Maintainability – The basic implementation is relatively short and simple, as well as loosely coupled, so maintaining such programs should be easy

Extendibility – If all observers are interested in all the details about the subject’s current state, extendibility should be relatively easy. If observers are only interested in certain details about the form’s state though, it can prove difficult to determine which observers to notify, while notifying all of them every time can be a large waste of resources, especially if used in large-scale applications