Observer Pattern

Observer pattern is a proper way to notify many listeners in case of one Subject changes. One of the most common usages of observer is in Model–view–controller **(**MVC) software architecture. Although In our project they didn’t use MVC architecture, Observer is a proper pattern to update view as in following I will explain.

Reference: <http://en.wikipedia.org/wiki/Observer_pattern>

**Why observer?**

In AssignmentPlanner class, both following methods are called time to time to update the calendar and LeftPanel view:

updateCalendar(eventsLoadedStartDate, eventsLoadedEndDate);

updateLeftPanel();

**Snippet code**

|  |
| --- |
| private void deleteFinishedEventsToolStripMenuItem\_Click(object sender, EventArgs e) {  //display confirmation message  DialogResult reallyDelete = MessageBox.Show("Are you sure you really want to delete all finished class events? This will remove all graded assignments and associated grading information for the past events associated with finished classes.", "Really Delete Finished Class Events?", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);  if (reallyDelete == DialogResult.Yes) {  Database.modifyDatabase("DELETE FROM Event WHERE EndDateTime < DATETIME('now', 'localtime') AND EventID IN(SELECT EventID FROM GradedAssignment NATURAL JOIN Class WHERE FinalLetterGrade IS NOT NULL);");  calendarEvents.Clear();  //reset calendar view to default settings  …  updateCalendar(eventsLoadedStartDate, eventsLoadedEndDate);  updateLeftPanel();  }  }  private void deleteFinishedGeneralEventsToolStripMenuItem\_Click(object sender, EventArgs e) {  //display confirmation message  DialogResult reallyDelete = MessageBox.Show("Are you sure you really want to delete all finished general events? This will not delete any current class assignments or events.", "Really Delete Finished General Events?", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);  if (reallyDelete == DialogResult.Yes) {  Database.modifyDatabase("DELETE FROM Event WHERE EndDateTime < DATETIME('now', 'localtime') AND EventID NOT IN(SELECT EventID FROM GradedAssignment);");  calendarEvents.Clear();  …  updateCalendar(eventsLoadedStartDate, eventsLoadedEndDate);  updateLeftPanel();  }  }  private void updateLeftPanel() {  updateEvents();  updateGrades();  updateGPA();  }  //another Example of repetitive using these updates  enableDisableToolStripMenuItem.Checked = PlannerSettings.Default.SyncEvents;  updateEvents();  updateGrades();  updateGPA();  updateCalendar(eventsLoadedStartDate, eventsLoadedEndDate); |

I think better way for implementing of these updates is using an observer pattern to update both these calendar and leftPanel. In this way by each change on form’s data such as add, delete and modification; observer will be responsible for these updates.

**Mechanics**

1. Add Listener class to AssignmentPlanner class

**public** **interface** Listener {

**public** **abstract** **void** viewchangs();

       }

**static** **private** ArrayList<Listener> *listeners* = **new** ArrayList<Listener>();

**static** **public** **void** addListener(Listener listener) {

*listeners*.add(listener);       }

**static** **public** **void** notifyListeners() {

**for** (**int** i = 0; i < *listeners*.size(); i++) {

                     Listener v = *listeners*.get(i);

                     v.viewchangs();           }    }

1. Make an update method :

Public void Update (){

updateCalendar(eventsLoadedStartDate, eventsLoadedEndDate);

updateLeftPanel();

}

1. Replace all

updateCalendar(eventsLoadedStartDate, eventsLoadedEndDate);

updateLeftPanel();

with

notifyListeners();