CS 3340 - Homework 2

Course Registration System

Contents

[1 Overview 2](#_Toc127188357)

[2 Steps to Complete 2](#_Toc127188358)

[2.1 Partially Build User Interface 2](#_Toc127188359)

[2.2 Populate the Available Classes List 3](#_Toc127188360)

[2.3 Add\_Click Event – Basic Functionality 4](#_Toc127188361)

[2.4 Remove\_Click Event – Basic Functionality 4](#_Toc127188362)

[2.5 Add\_Click Event – Hours & Cost 4](#_Toc127188363)

[2.6 Remove\_Click Event – Hours & Cost 5](#_Toc127188364)

[2.7 Reset\_Click Event 5](#_Toc127188365)

[2.8 Add\_Click Event – Registration Hour Limit 5](#_Toc127188366)

[2.9 Add\_Click Event – Extra Options 5](#_Toc127188367)

[2.10 Enhance GUI 6](#_Toc127188368)

[2.11 MakeAvailable\_Click 6](#_Toc127188369)

[2.12 MakeAvailable\_Click – Prevent Adding Existing Course 7](#_Toc127188370)

[2.13 RemoveFromAvailable\_Click 7](#_Toc127188371)

[3 Grading Rubric 7](#_Toc127188372)

[4 Submission 7](#_Toc127188373)

[Appendix 1 n/a 8](#_Toc127188374)

# Overview

You will write a system that allows a student to register for classes. **This assignment will take a lot of time.**

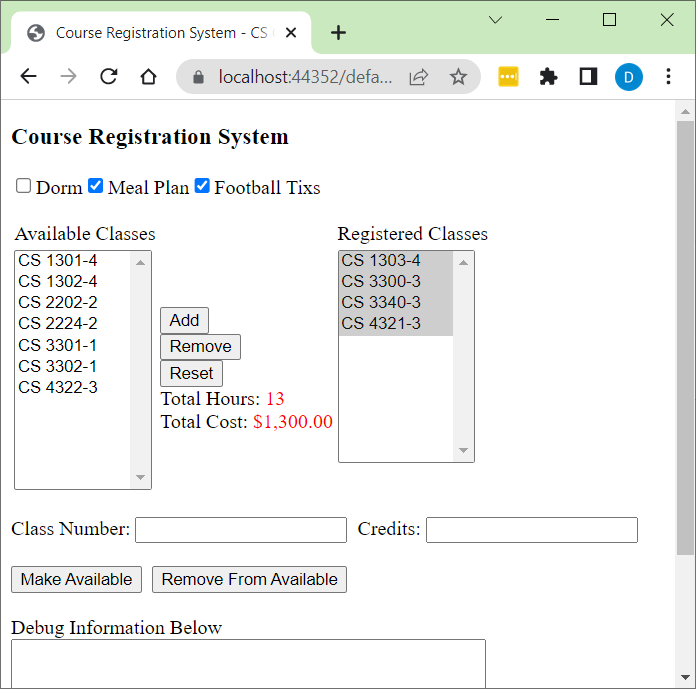


Figure 1

# Steps to Complete

I recommend you follow the steps below to build the page incrementally. Thus, all the elements shown in the Figure 1 above are not shown in the figures below. You will be adding them as you go.

## Partially Build User Interface

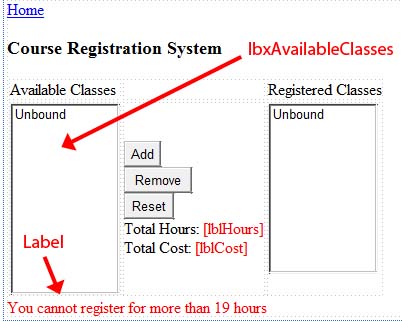
1. Create your *hw02* project with the name: *hw02\_lastName.*
2. Create a page named *default.aspx.* The design of the page is shown on the right (except you will NOT have a “Home” link).

Figure 2

1. Add the HTML title (in the head tag): “HW 2 – FirstName LastName”, substituting your name.
2. Build the page as shown in Figure 2 on the right. Note:
3. Make sure the name of the ListBox containing the Available Classes is “lbxAvailableClasses”. You can name the other components anything you like.
4. I recommend using a table to layout the part of the GUI with the ListBoxes. If you look carefully at the image on the right, you can see the table’s grid lines.
5. “Total Hours” is a label with the text, ”Total Hours”. Same for “Total Cost”.
6. **REQUIREMENT:** Consider the label, “You cannot register…”. This should not be visible when the page is first loaded.
7. **REQUIREMENT:** Consider the two labels, “lblHours” and “lblCost”. They should display “0” when the page is first loaded.

## Populate the Available Classes List

1. Populate the *ListView*.
2. Add the method below to the code-behind. The method builds and returns an array of *ListItems*. Do not modify this code. When the page is first loaded, you will call this method to build the list of available courses. This is hard-coded this so that everyones page initially shows the exact same courses in the proper format.

**Note: Both the *ListItem’s* *text* and *value* are in the format: “CS *courseNum-credits*”.**

private ListItem[] buildAvailableCourseList()

{

ListItem[] tempList = { new ListItem("CS 1301-4", "CS 1301-4"),

new ListItem("CS 1302-4", "CS 1302-4"),

new ListItem("CS 1303-4", "CS 1303-4"),

new ListItem("CS 2202-2", "CS 2202-2"),

new ListItem("CS 2224-2", "CS 2224-2"),

new ListItem("CS 3300-3", "CS 3300-3"),

new ListItem("CS 3301-1", "CS 3301-1"),

new ListItem("CS 3302-1", "CS 3302-1"),

new ListItem("CS 3340-3", "CS 3340-3"),

new ListItem("CS 4321-3", "CS 4321-3"),

new ListItem("CS 4322-3", "CS 4322-3")

};

return tempList;

}

1. Next, we call the preceding method in *Page\_Load* and bind the return to the ListBox. Add this code to *Page\_Load*.

if (!Page.IsPostBack) //For initial page creation

{

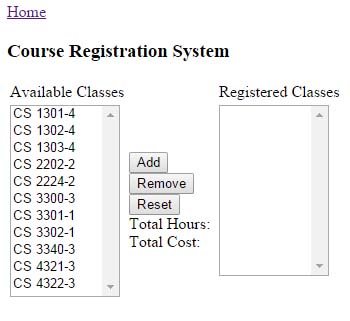
ListItem[] availableCourses = buildAvailableCourseList ();

lbxAvailableClasses.DataSource = availableCourses;

lbxAvailableClasses.DataTextField = "Text";

lbxAvailableClasses.DataValueField = "Value";

lbxAvailableClasses.DataBind();

 }

1. Run your page. It should look like as shown on the right (except yours should show “Total Hours: 0” and “Total Cost: $0.00”)

## Add\_Click Event – Basic Functionality

The key to this assignment is to express your algorithm for each event handler in words, then translate those words into helper methods. My solution has 15 helper methods. The first few event handlers, I lead you in that direction. If you don’t use this approach, you will have very large nested if blocks and lots of repeated code, both of which are not maintainable.

1. **REQUIREMENT:** Program the Add button so that when the user selects any number of courses from the Available Courses and chooses, “Add”, the courses are moved to the Registered Courses (and removed from the Available Courses). When complete, test your code.

Hints:

* Don’t bother with the Total Hours and Total Cost yet, nor the requirement for no more than 19 hours.
* Algorithm:

Add\_Click

list = gather selected from lbxAvailable

Add courses in list to lbxRegistered

Remove courses in list from lbxAvailable

* Refined algorithm

Add\_Click:

List<ListItem> list = getSelectedCourses(lbxAvailable)

addCourses(list, lbxRegistered)

removeCourses(list, lbxAvailable)

## Remove\_Click Event – Basic Functionality

1. **REQUIREMENT:** Program the Remove button so that when the user selects any number of courses from the Registered Courses and chooses, “Remove”, the courses are moved to the Available Courses (and removed from the Registered Courses). When complete, test your code.

Hints:

* This is exactly like the Add button.
* Algorithm:

Remove\_Click

List<ListItem> list = getSelectedCourses(lbxRegistered)

addCourses(list, lbxAvailable)

removeCourses(list, lbxRegistered)

## Add\_Click Event – Hours & Cost

1. **REQUIREMENT:** Modify the “Add” button so that it updates the Total Hours and Total Cost. Total cost should be in this format: “$#,###.##”. One way to do this is with the [currency format specifier](https://www.delftstack.com/howto/csharp/format-string-to-currency-in-csha). Note:

* Each credit hour costs $100
* The *Text* and *Value* both have the format: *Course Number-creditHours*. This format should not be altered. Thus, you need to strip the credit hours off of the *Value.* You can use the [*Split*](https://learn.microsoft.com/en-us/dotnet/api/system.string.split?view=net-7.0)method found in the String class to do this.
* You might want to add a multi-line textbox at the bottom of your page for debugging. I recommend using this to display the correct values for the hours for each course before programming the Total Hours and Total Cost.

Hints:

* Add a helper method, *updateHoursAndCost* to the end of *Add\_Click*
* Algorithm

updateHoursAndCost()

int totalHours = getTotalHours()

double cost = totalHours \* 100.0

lblHours.Text = totalHours

lblCost.Text = format(cost)

* *getTotalHours* is a helper method that simply loops through all the registered for courses in the ListBox and extracts the hours for each one, adds them up, and returns the total.

## Remove\_Click Event – Hours & Cost

1. **REQUIREMENT:** Modify the “Remove” button so that it updates the Total Hours and Total Cost when courses are removed (similar to the preceding requirement).

## Reset\_Click Event

1. **REQUIREMENT:** Program the “Reset” button so that the system is completely reset: no registered classes, total hours and cost are 0, and the original list of available courses is shown. This line of code will accomplish that:

Response.Redirect(Request.RawUrl);

## Add\_Click Event – Registration Hour Limit

1. **REQUIREMENT:** Modify the “Add” button so that the user cannot register for more than 19 hours. If a course puts the total hours **over 19**, it should not be added and an error message should be displayed as shown on the Figure 2 above.

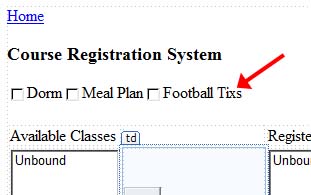
Note:

* If there are multiple selected courses to be added, then they are added until there is a violation of the hours requirement. For example, if there are currently 12 hours registered for and the user is trying to add to add courses with credits: 3, 3, 2, 4, then the first two are added (3 & 3), and the last two are not.
* The label should be hidden when it is no longer applicable. For instance, if a course is removed, or another added that does not put the total over 19 hours, or the system is reset.

Hint:

* You should not use a validator. You should program this logic into the Add click event.
* Use English to modify the algorithm for the existing *Add\_Click* event. Then, refine this with helper methods. You can make this very clean if you give it some thought.

## Add\_Click Event – Extra Options

1. Add a *CheckBoxList* as shown on the right.
2. **REQUIREMENT:** Modify the “Add” button so that it:

* Increases the total cost label for all checkboxes that are selected. The dorm costs $1000, the meal plan is $500, and the football tickets are $50.
* Decreases the total cost label for all checkboxes that have been unselected.

Note:

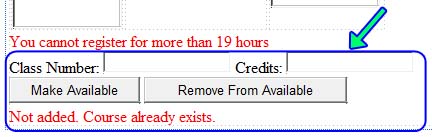
* Anytime Add or Remove is selected, these should be checked to update the total cost. For example, if the user currently has 1, 3-credit registered and Dorm checked, then the total cost is $1300.00. Then, if they select another 3-credit course and unselect Dorm, then the total cost will be $600. The same process occurs when remove is selected. (This is not the best UI design, but program it as described).

Hints:

* You can modify *updateHoursAndCost* to incorporate this be creating and calling a helper method, *getExtrasCost* that computes the costs associated with the “extras” (e.g. the checkboxes).

## Enhance GUI

1. Add the items shown box below to the bottom of the GUI (but above any debug TextBox you might have added). The text at the bottom (“Not added…”) should be a Label.



## MakeAvailable\_Click

1. **REQUIREMENT:** The user can add a new course to the available list, specifying the class number and number of credits in the two text boxes above. The “Make Available” button adds this new course to the available list when the user supplies a class number (*e.g.* CS 3320) and credits (*e.g.* 4). Add a click event for the “Make Available” button that adds the course to the available list.

Note:

* You’ll need to construct the Text & Value correctly (*e.g.* CS 3320-4).
* Don’t worry if the course already exists; we’ll deal with that later.

1. **REQUIREMENT:** When adding a new course, the number of credits must be between 1 and 10, inclusive. Add a Validation control (not shown in the GUI above) to enforce this. When invalid data is entered, this should show an informative error message (not shown in figure).

Hint:

* Set the “CausesValidation” property to *false* for only these three buttons: “Add”, “Remove”, “Reset” (not the “Make Available” and “Remove From Available” buttons). This prevents the validation from taking place when the user is Adding a course (or Removing, etc).
* You will probably need to add the highlighted code below to your *Web.config* file.

<configuration>

<appSettings>

<add key="ValidationSettings:UnobtrusiveValidationMode" value="None" />

</appSettings>

<system.web>

<compilation debug="true" targetFramework="4.5" />

<httpRuntime targetFramework="4.5" />

</system.web>

</configuration>

## MakeAvailable\_Click – Prevent Adding Existing Course

1. **REQUIREMENT:** Continuing from the previous requirement, if the **Class Number** (e.g CS 1301) already exists (either in Available or Registered), it should not be added and the error message, “Not added. Course already exists.” should be displayed. For example, “CS 1301-4” is in the list initially. If the user typed in: “CS 1301” (you should ignore any credits that are typed in), then this course would not be added.

Hint:

* You need to (a) loop through the Available courses, extract the course name and compare it to what the user typed in, (b) next, loop through the registered list and do the same.

## RemoveFromAvailable\_Click

1. **REQUIREMENT:** Program the “Remove From Available” button so that it takes the Class Number and removes this course from the Available list. Note:

* The user is not required to enter the credits when removing a class, thus, any value there should be ignored. So, if the user enters, “CS 1302”, if there is a course in Available, then it should be removed.
* Don’t worry about if the course is not found, or already registered for. These will be taken care of with the next two requirements.

## RemoveFromAvailable\_Click – Prevent Removing when can’t

The final two requirements must be implemented with a **custom validator**.

1. **REQUIREMENT:** Continuing from the previous requirement, if the course is in the Registered list, do not remove it, but display the error message, “Not removed. Course is registered for.”
2. **REQUIREMENT:** Continuing from the previous requirement, if the course is not found in either the Available or Registered lists then display the error message, “Course not found.”

# Grading Rubric

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Points** | **Description** |
| 1 | 10 | Add moves to registered |
| 2 | 10 | Add updates hours and cost |
| 3 | 10 | Remove moves to available |
| 4 | 10 | Remove updates hours and cost |
| 5 | 10 | Reset restarts app with no state |
| 6 | 10 | Can't register for more than 19 hours |
| 7 | 10 | Checkboxes update cost on postback |
| 8 | 5 | Make Available works when no problem |
| 9 | 5 | Make Available detects error when exists in available |
| 10 | 2 | Make Available detects error when exists in registered |
| 11 | 3 | Validator on number of credits |
| 12 | 5 | Remove from Available works when no problem |
| 13 | 5 | Remove from Available detects error when doesn't exist |
| 14 | 5 | Remove from Available detects error when exists in registered |
| **Total** | **100** |  |

# Submission

1. Zip your project into a file named: *hw02\_lastName.zip*. The following must be true:
2. When I unzip, it unzips to a folder named: *hw02\_lastName*
3. There is a *hw02.sln* (or *hw02\_*lastName) file in the project folder that when double-clicked launces your project in VS 2019.
4. Submit on Blazeview in the HW 2 dropbox.

Appendix

1. n/a