CS 3340 – HW 3 – Ticket Sales System

Contents

[1 Introduction 1](#_Toc160367657)

[2 System Description 2](#_Toc160367658)

[2.1 *Default.aspx* 2](#_Toc160367659)

[2.2 *Summary.aspx* 4](#_Toc160367660)

[3 Grading Rubric 5](#_Toc160367661)

[4 Suggestions 5](#_Toc160367662)

[Appendix 1 n/a 5](#_Toc160367663)

# Introduction

This is an individual assignment. You will write a system that allows a user to create an event (concert, *etc.* anything with assigned seating. The user can also purchase tickets for the event. On a summary page, the user can see all the tickets that have been sold.

Required Skills: OO modeling and state management

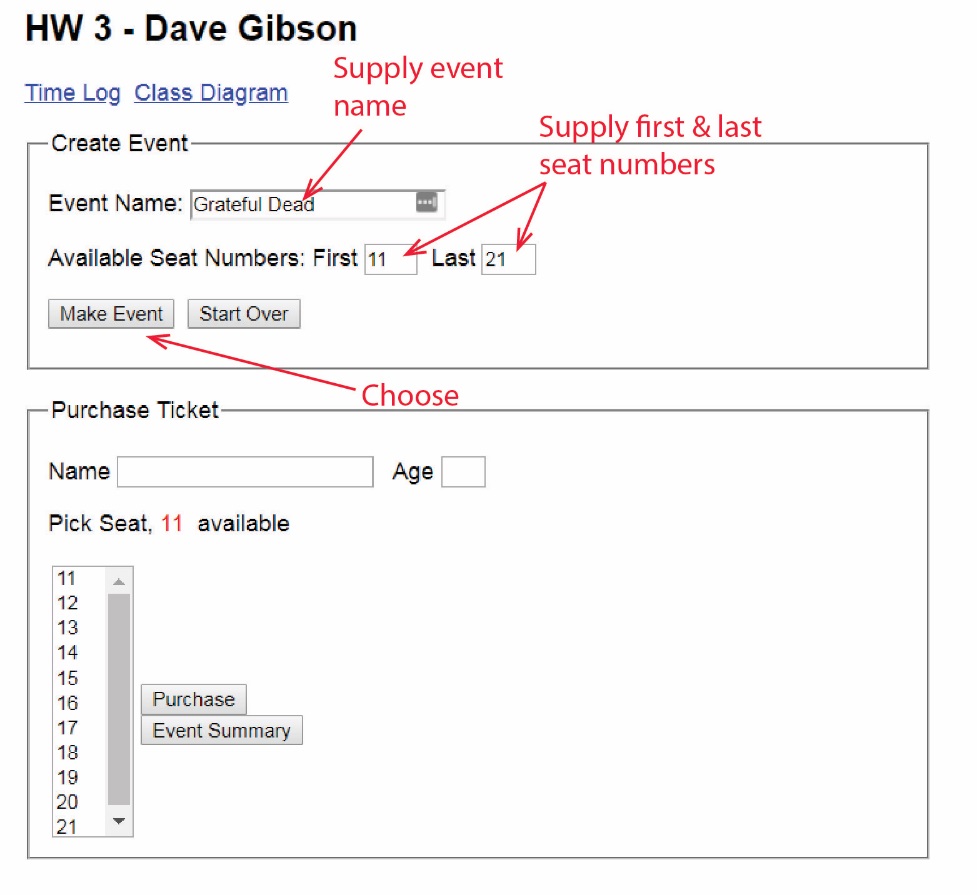
# System Description

## *Default.aspx*

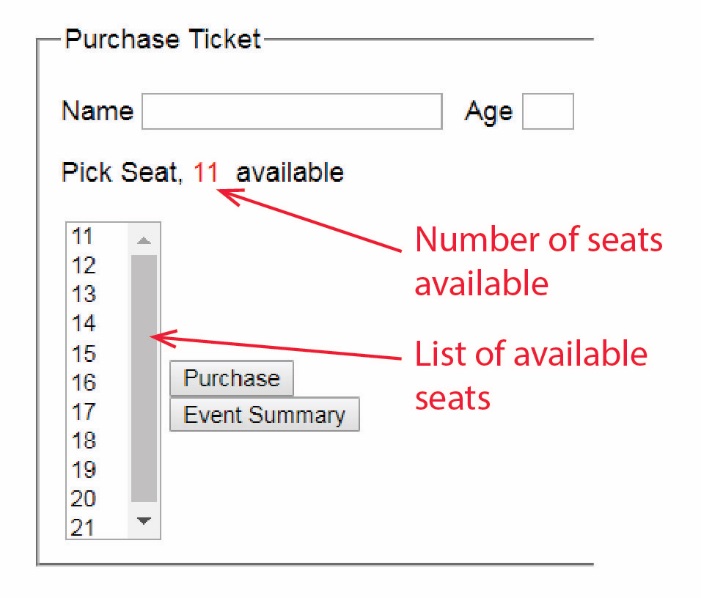
1. You must use an object-oriented approach to state management. This means you need to write and use classes to represent the data necessary to implement the system as described.
2. Create a class diagram using the class design tool in VS (see Lab 6). It should show only the domain classes (not the web form) and should show: (a) all instance variables, (b) the full signature of all methods, (c) the relationship between the classes.
3. When the main page, *Default.aspx* is first displayed, the user should create an event as shown below.

Notes:

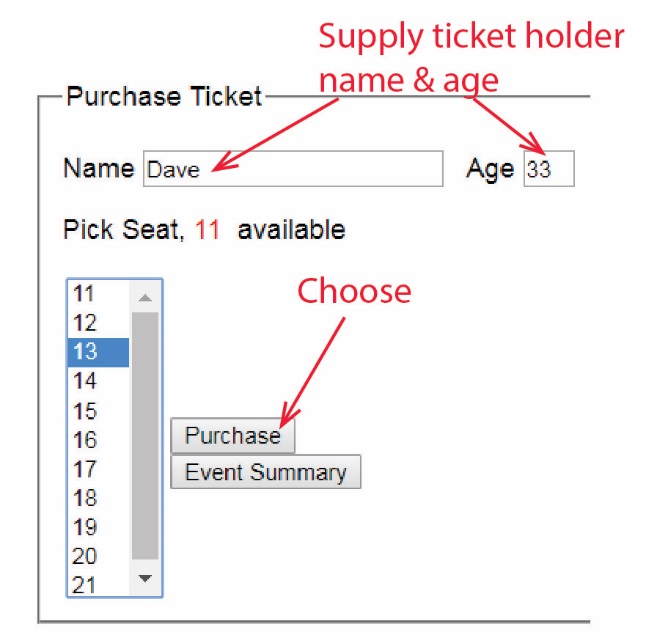
* The page should appear roughly as shown. I used a *Panel* for the two regions, though you don’t have to. If you do: set the *GroupingText* property for the title (Create Event) which will automatically put the border. Do not set any border properties as it will cause an additional border to appear.
* There is a link to a page which shows your class diagram. Ignore the *Time Log* link.



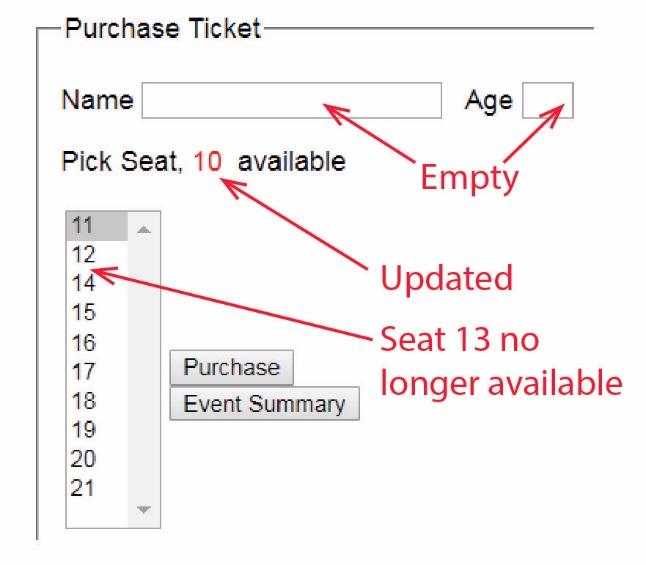
1. When the user presses, “Start Over”, the system forgets everything: the event, and any tickets that have been purchased and resets the three text fields to be empty.
2. When the user presses, “Make Event”, the “Purchase Ticket” pane looks like this:



1. The user supplies the data to purchase a ticket and presses, “Purchase”.



1. A ticket is purchased. If ticket holder is 12 or under tickets are $5; otherwise, $10.
2. The display is updated



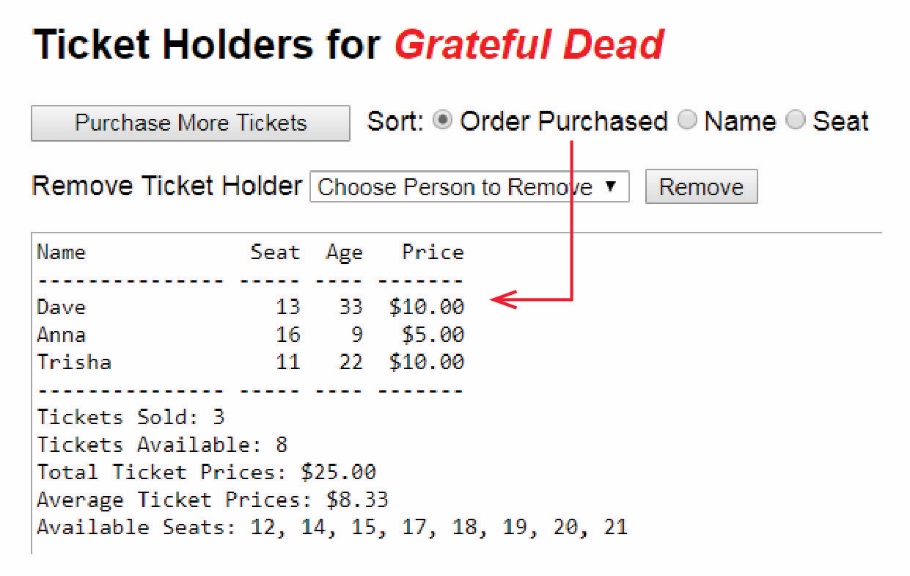
1. The user can continue to purchase tickets.
2. At any time, user can press, “Event Summary” in which case *Summary.aspx* is displayed.

## *Summary.aspx*

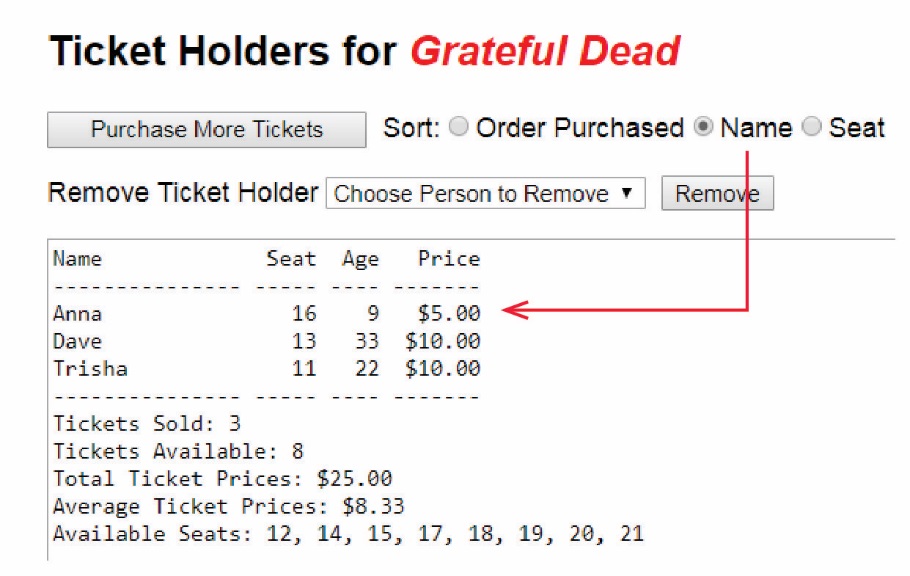
1. When the user press the “Event Summary”button on *Default*, the *Summary* page displays all the information shown below. Note:

* The drop down displays a list of names of the ticket holders. The first entry can be added after all other names have been added (or after any databind if you use that technique) with a line like this:

ddlTicketHolders.Items.Insert(0, new ListItem("Choose Person to Remove", "0"));



1. The user can return to the main page by choosing, “Purchase More Tickets” in which case *Default.aspx* is displayed and the number of seats and the choices available should be correct (correspond with the state of the event on the summary page).
2. The user can choose a radio button and the list of ticket holders will immediately be sorted according to the choice. The default order of ticket holders is the order they were purchased. For example:



1. The user can select a person from the drop down and chose, “Remove” in which the person is removed from the drop down and all data and statistics are recomputed and displayed, omitting the removed ticket holder.

# Grading Rubric

Develop the system as described above.

|  |  |  |
| --- | --- | --- |
| **Req't** | **Points** | **Description** |
| 1 | 10 | Custom classes defined for state management |
| 2 | 10 | Class diagram |
| 3 | 10 | Make Event button shows number of tickets available and corresponding seats listed in DropDownList |
| 4 | 10 | Purchase removes one from tickets available, removes from DropDownList, and clears text boxes |
| 5 | 10 | Event name shown on Summary page |
| 6 | 7 | Summary of tickets purchased is displayed on Summary page |
| 7 | 6 | Number of tickets sold and available shown on Summary page |
| 8 | 6 | Average & total ticket price shown on Summary page |
| 9 | 6 | List of seats available shown on Summary page |
| 10 | 10 | Sell more tickets button updates main main page properly |
| 11 | 8 | Sorting on Summary page works |
| 12 | 7 | Removing ticket purchaser on Summary page works |
| **Total** | **100** |  |

# Suggestions

1. Design your class model first. Study the problem carefully and encapsulate the data you need to store. Identify methods you will need. Don’t spend too much time on this as you will almost certainly not get it all and have to modify it later. Then, write the classes and test. In general, you want your event handlers to be as short as possible, delegating the work to the classes. For example, I recommend writing methods in the appropriate class to return the statistics in the Summary text area: TicketsSold(), TicketsAvailable(), *etc.* Then, in the page itself, I would write a helper method to actually build the display. This follows the pattern of separating domain logic and presentation logic.
2. Implement with baby steps.
3. For Requirement 5, the display doesn’t have to perfectly lined up, but should be reasonable to look at. If you want to make it very neat, here is some help:

<http://www.csharp-examples.net/align-string-with-spaces/>

1. For the sorting on the Summary page, this is a very simple example of custom sorting:

<http://stackoverflow.com/questions/20902248/sorting-a-list-in-c-sharp-using-list-sortcomparisont-comparison>

The first technique is Lambda Expression, the second is a Linq, the third is a Delegate (somewhat the approach from CS 1302); any is fine. And, there are other ways to sort.

.

Appendix

1. n/a

.