CS 3340 – HW 4 – Ticket Sales System

Contents

[1 Introduction 1](#_Toc194498318)

[2 System Description 2](#_Toc194498319)

[2.1 *default.php* 2](#_Toc194498320)

[2.2 *summary.php* 4](#_Toc194498321)

[3 Grading Rubric 5](#_Toc194498322)

[4 Steps to Complete 6](#_Toc194498323)

[Appendix 1 n/a 10](#_Toc194498324)

# 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

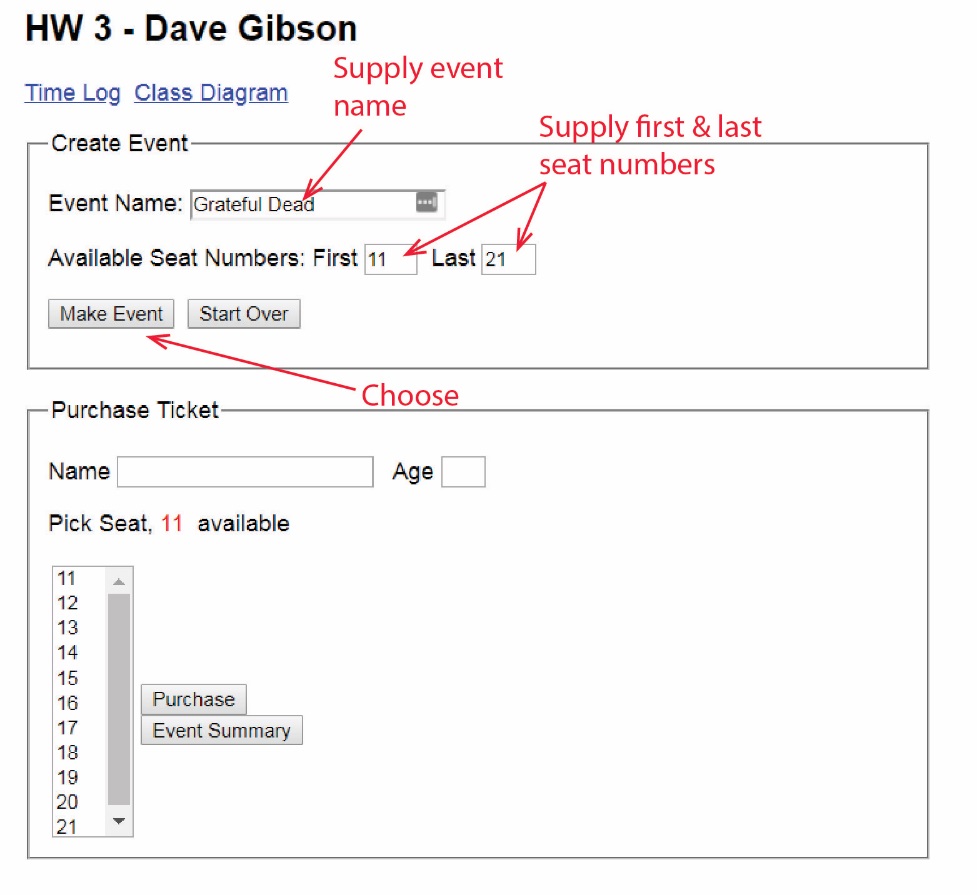
# System Description

## *default.php*

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. When the main page, *default.php* is first displayed, the user should create an event as shown below.

Notes:

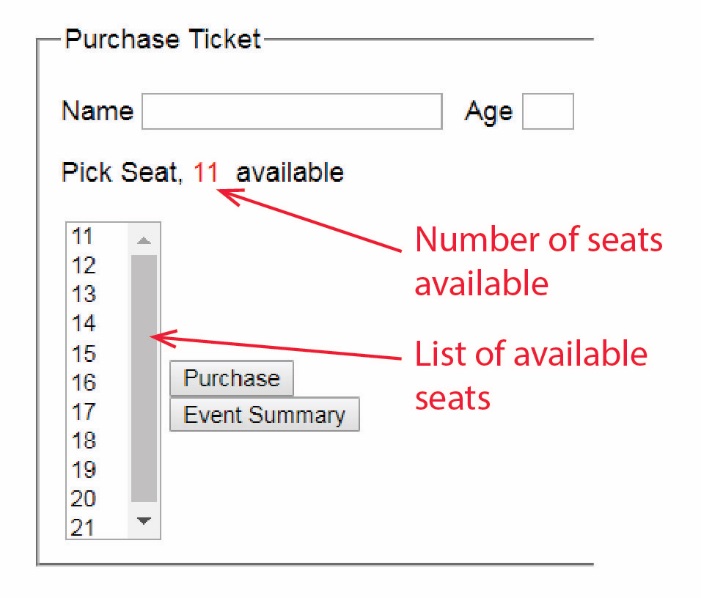
* The page should appear roughly as shown. (borders around the regions are not required)
* Use level 3 headers for “Create Event” and “Purchase Ticket”
* There is a link to a page which shows your class diagram (discussed later). **Ignore the *Time Log* link.**
* All buttons are submit buttons.



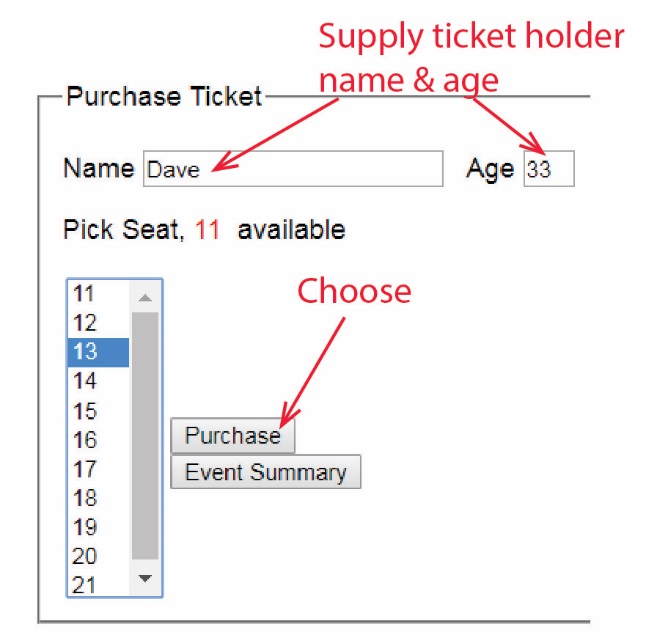
1. The user enters and event name, and first and last ticket numbers. When the user presses, “Make Event”, the “Purchase Ticket” pane looks as shown below.

Notes:

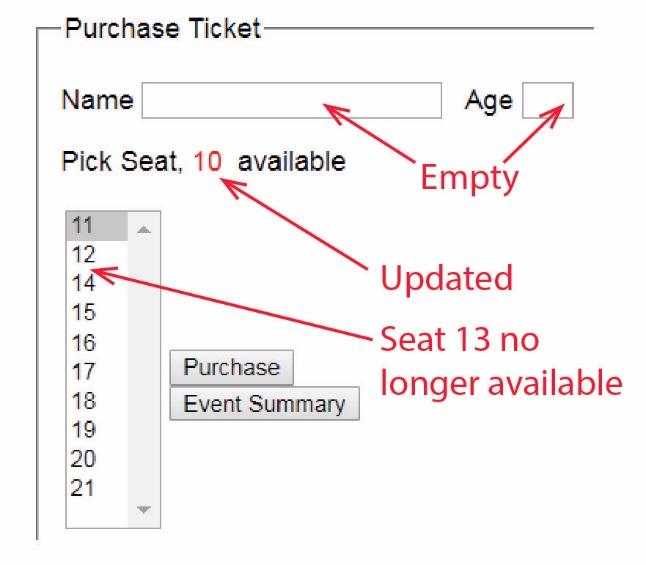
* You will build the list box (select) programmatically
* The “Make Event” button postsback to itself.



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



1. The “Purchase” button postsback to itself.
2. A ticket is purchased. If ticket holder is 12 or under tickets are $5; otherwise, $10.
3. The display is updated as indicated below

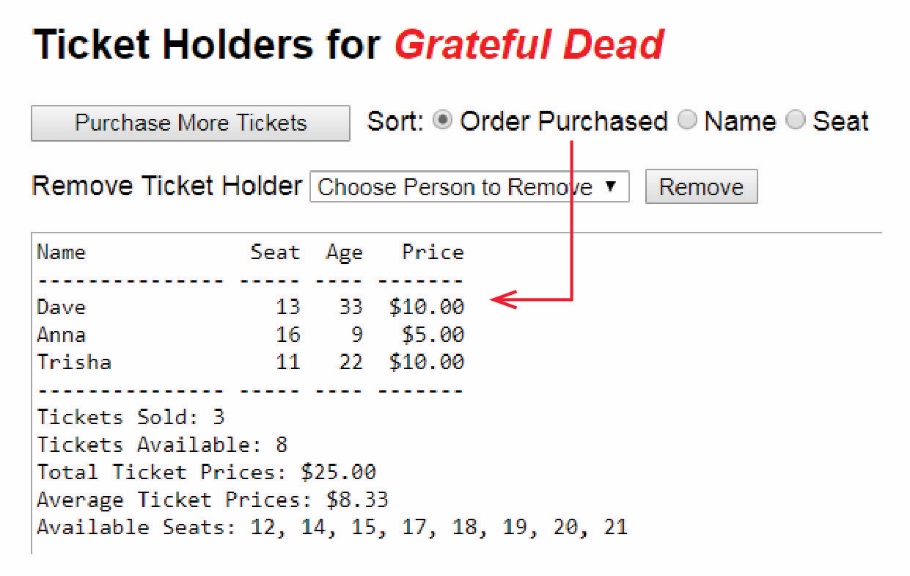


1. The user can continue to purchase tickets.
2. When the user presses, “Start Over” (postback to self), the system forgets everything: the event, and any tickets that have been purchased and resets the three text fields in the “Create Event” pane to be empty.
3. At any time, user can press, “Event Summary” (postback to self), in which case the user will be redirected to *summary.php*.

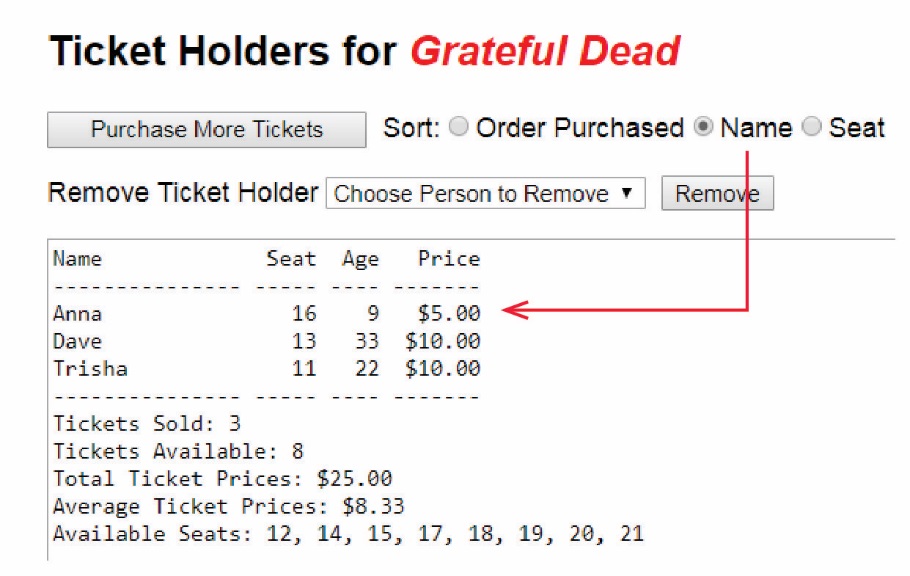
## *summary.php*

1. When the user presses 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. **It does not have to have the first option, “Choose Person to Remove”. This will be extra credit.**



1. The user can return to the main page by choosing, “Purchase More Tickets” (postback to self and redirect) in which case *default.php* 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). In other words, if a person has been removed, then their ticket should now be available in the “Purchase Ticket” pane.
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.

|  |  |  |
| --- | --- | --- |
| **Criterion** | **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" button removes one from tickets available, removes from DropDownList, and clears name and age text boxes |
| 5 | 10 | Event name shown on Summary page |
| 6 | 5 | "Start Over" button works properly. |
| 7 | 10 | List of ticket holders is displayed on Summary page |
| 8 | 5 | Number of tickets sold and available shown on Summary page |
| 9 | 5 | Average & total ticket price shown on Summary page |
| 10 | 5 | List of seats available shown on Summary page |
| 11 | 10 | "Purchase More Tickets" button updates default page properly |
| 12 | 8 | Sorting on Summary page works |
| 13 | 7 | Removing ticket purchaser on Summary page works |
| **Total** | **105** |  |
|  |  |  |
| Extra Credit |  |
| 14 | 5 | "Choose Person to Remove" first item in dropdown and works properly. |
| 15 | 5 | Very neatly organized Gui and some styling |
| **Total** | **115** |  |

Time estimates based on the Steps to Complete below

|  |  |  |
| --- | --- | --- |
| **Steps to Complete** | **Task** | **Time (hr)** |
| 1 | Class model | 0.25 |
| 2a | Code Classes | 0.25 |
| b | Write default | 0.50 |
| c | Write summary | 0.50 |
| d | Modify default | 0.25 |
| 3a | Modify event class | 0.25 |
| b | Modify default | 1.25 |
| 4a | Create TicketHolder class | 0.25 |
| b | Mofidy Event class | 0.25 |
| c | Modify default | 2.50 |
| 5a | Evaluate Event class | 0.25 |
| b | Modify summary-helper | 2.00 |
| c | Modify summary | 0.25 |
| 6 | Start Over | 0.25 |
| 7a | Add radio buttons | 0.25 |
| b | Sorting | 1.00 |
| 8a | HTML for remove | 0.25 |
| b | Helper function for remove | 0.50 |
| c | Code to display | 0.25 |
| d | Update after remove | 0.50 |
| 9a | Class diagram | 0.50 |
|  | **Total** | **12.25** |

# Steps to Complete

You can take the description above and write the system. Or, in the steps below, I’ll provide a baby-steps approach.

1. Design your class model first, sketch on paper. 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. Probably, you’ll want something like: Event 🡪\*TicketHolder
2. **Goal: Capture event name and carry in session between the two pages.** Do the following:
3. Write the class to hold the “Event”, with just a field for the event name, a getter for that and a constructor.

**See:**

* + ***lesson2/ex4-session-v1/Order.php, Product.php***
  + [PHP OOP - Classes and Objects](https://www.w3schools.com/php/php_oop_classes_objects.asp)
  + [Classes & Objects – The Basics](https://www.php.net/manual/en/language.oop5.basic.php)

1. Write *default.php* with only:
2. A text field to supply the event name
3. A “Make Event” button which postsback to itself, creates the Event, and puts it in session. Add some code to display what is in session, at the bottom of the page, in, say, a paragraph, for testing and debugging purposes. **(See *lesson2/ex4-session-v1/add\_order.php, add\_products.php*)**
4. Test.
5. Write *summary.php* with only:
6. The title at the top with the event name displayed (pulled from the Event object in session)
7. A “Purchase More Tickets” that postsback to itself and redirects to *default*. **(See *lesson2/*** ***ex3-session.php, ex3-results.php*)**
8. Modify *default.php*:
9. Add a “Event Summary” button which postsback to itself and then redirects to *summary.php.*
10. Test thoroughly. Make sure you are passing the Event session variable back and forth between the two pages.
11. **Goal: “Start Over” works.** When the user presses, “Start Over” (postback to self), the system forgets everything: the event, and any tickets that have been purchased. To clear the session, see code below. Hint: you need to detect the button that indicates to Start Over.

session\_unset();  
session\_destroy();

1. **Goal: “Make Event” populates listbox.** Do the following:
2. Modify the Event class to hold the first and last ticket numbers. Add those to the Event constructor.

**Hints:**

* **It will be useful to create an additional instance variable, *available\_seats* which is an array. The constructor can populate it with a loop that goes: i=first\_seat; i<=last\_seat. This will come in handy several places!**

1. Modify *default.php*:
2. Add the text fields to enter first and last ticket number.
3. Add all the HTML for the “Purchase Ticket” area, except the list box (you’ll build that programmatically).
4. Write a helper function to build the list box.

**Hints:**

* **See *lesson2/ex5-radiopostback.php* for the IDEA, but note that you are building a <*select>* with <*option>*.**
* **Set the <select> *size* attribute to make it taller (display more values), say, <select size=7>, or whatever.**
* **You need to set the *value* attribute on the <option> because that is what will be posted back.**
* **You can use a loop to build the <option>s. Just loop over the *available\_seats* instance variable from above.**

1. Write code to call the function to display the list box (in between the HTML above). Don’t worry about positioning things; it is fine for now if the buttons are below the list box.
2. Test. Make sure “Make Event” builds the list box correctly. Use the “Start Over” button to test.
3. **Goal: “Purchase” updates listbox, *etc*.** Do the following:
4. Create a Ticket Holder class.
5. Modify the Event class to hold an array of Ticket Holders. Add methods to add and get a TicketHolder. You may not need a *getTicketHolder* method, but it might be useful. I found it useful to make a method, *getTicketHolders* that returns the array of TicketHolders. **(See *lesson2/ex4-session-v1/Order.php)***
6. Modify *default.php*:
7. The Purchase button posts back to itself
8. Add text fields and labels for entering name and age.
9. Add code so that when a ticket is purchased, a Ticket Holder object is created, put into the Event, which is put into session.

**Hints:**

* **See *lesson2/ex4-session-v1/add-products.php***
* **When you add the TicketHolder to the event, you will also need to update the *available\_seats* instance variable. You’ll need to find the location of the selected seat in the array:**

**Searching an array**

<https://www.w3schools.com/php/func_array_search.asp> The example shows a map (called an associative array in PHP), but it works the same with a regular, indexed array) For example:

<?php

$a=array(2,5,6,7,8,11);

echo array\_search(6,$a); // displays “3”, the index.

?>

* Then, you’ll need to remove it from the array:

**Removing an element from an array: splice**

<https://www.w3schools.com/php/php_arrays_remove.asp>

1. Add code so that the label showing how many tickets are available is updated.
2. Add code so that the list box is updated.
3. Write code to call the function and display the list box.
4. Note that if your event is not in session, i.e. first time on page, or a Start Over, the code above (iii and iv above) will bomb. So, you’ll need to not display the purchase stuff when the event is not in session.

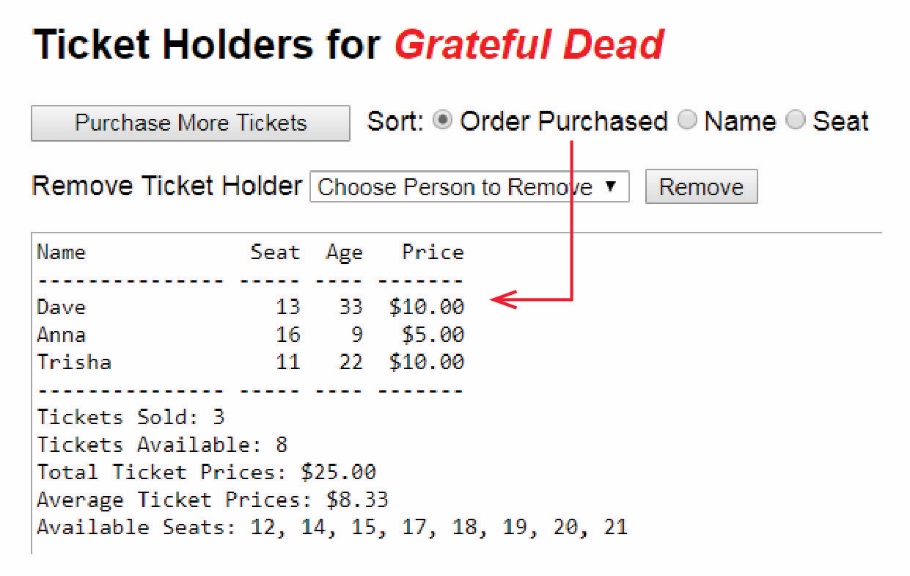
Hint:

You can conditionally display HTML as shown below. Note that the open “{“ is PHP, followed by HTML, the followed by PHP closing “}”

<?php  
 if (event is in session) {  
?>

// Some HTML statements!  
<?php  
 }  
?>

1. Test thoroughly.
2. **Goal: “Event Summary” updates the display.** Do the following:
3. For now, we are only working on the area shown below (ignore the red arrow) (NOT the radio buttons, drop down, and remove). Thus, you should take some time to see if Event class has all the data you need to build this. You may need some more variables and methods in that class. The easiest way to approach this whole assignment is to put all the information you need in the classes, and define useful methods. **Do that now.**



1. Modify *summary.php*: Write a helper method to build this display. It is shown in text area, but it doesn’t have to be. You can display anyway that is neat and arranged closely to what is shown. The top part doesn’t have to be laid out perfectly with the spaces (tabs?) shown above, just something that is readable.
2. Modify *summary.php* with a call to display the summary.
3. Test thoroughly. Be sure and test “Purchase more tickets” and make sure things are updating properly on both pages.
4. **Goal: Sort rearranges the summary display.** Do the following:
5. Add the HTML for the radio buttons
6. Add code so that the sorting takes place and updates display. I recommend helper method(s). Some hints to make the radio buttons post back immediately.

**Hints:**

* **See *lesson2/ex5-radiopostback.php* to illustrate radio buttons auto-postback using JavaScript.**
* **I recommend writing methods in the Event class to return the data sorted, *e.g. ticketHolderByName, ticketHoldersBySeat*. You don’t need a separate method for “by order purchased”, that should already be your TicketHolders array in the EventClass. Note: you will need to copy the TicketHolders array before sorting.**
* **Copying an array of objects (second answer):**

<https://stackoverflow.com/questions/1532618/is-there-a-function-to-make-a-copy-of-a-php-array-to-another>

* **Sort by a string property from inside a class (first answer, see: “From inside a class”):**

<https://stackoverflow.com/questions/4282413/sort-array-of-objects-by-one-property>

* **Sort by a numeric property from inside a class (The answer is the first item on page. A simpler version, since the property is an integer is found further down in the comments, by “Gabriel Somoza”, just as we did in Java when writing a Comparator):**

<https://davidwalsh.name/sort-objects>

* **It will be a bit harder to have your code maintain the appropriate radio button to be checked. You can skip that.**

1. **Goal: “Remove” removes a ticket holder and updates the summary display.** Do the following:
2. Add the HTML needed.
3. Write a helper method to build the dropdown.
4. Write code to display the dropdown.
5. Write code so that when Remove is pressed, the ticket holder is removed and everything is updated. A helper method is recommended.
6. **Goal: Create and link a class diagram.** Hand drawn, photo is fine. However, it must be legible and oriented properly. It should show: (a) all instance variables, (b) the full signature of all methods, (c) the relationship between the classes. Use UML notation including data types.

Appendix

1. n/a

.