MALMÖ HÖGSKOLA

Programming Using C#, Basic Course

One and two dimensional arrays

Assignment 4

Apu Movies and Theatre

Cinema Booking System

Version 2

Mandatory

Farid Naisan
University Lecturer
Department of Computer Science
Malmö University, Sweden

Contents

Assignment 4: CBS Versions 2 & 3 - Arrays

1. Objectives

The main objectives of this assignment are:

- To work with one and two dimensional arrays.
- Use enumerations to group named constants.
- Use constructors to initialize an object.
- Enhance the GUI with new features and using a **ComboBox** and **RadioButtons**.
- Using **MessageBox** to interact with the user with notifications.
- Exercise more with parameterized methods.

2. Description

The project idea was described in the last assignment. There, it was explained that you had to write a program for automating the booking of tickets for a newly established movie theater in your town. The program was to be used by the cinema's cashiers who need to register the name of the customer and the price for the seat being reserved. . In this assignment, you are to produce a new version, version 2 of the same application by building on version1.

In the last version, you designed the GUI and coded for handling the input. In this assignment, it is intended to reuse these (with very little changes) exactly as our plans were. This assignment has a mandatory part for a Pass grade (Version 2a), and a second part that is mandatory only for a higher grade, Pass with Distinction (Version 2b). Then there is also another optional part, Version 3 (not graded) for those who wish to do more. The main goals to be accomplished in this as assignment are:

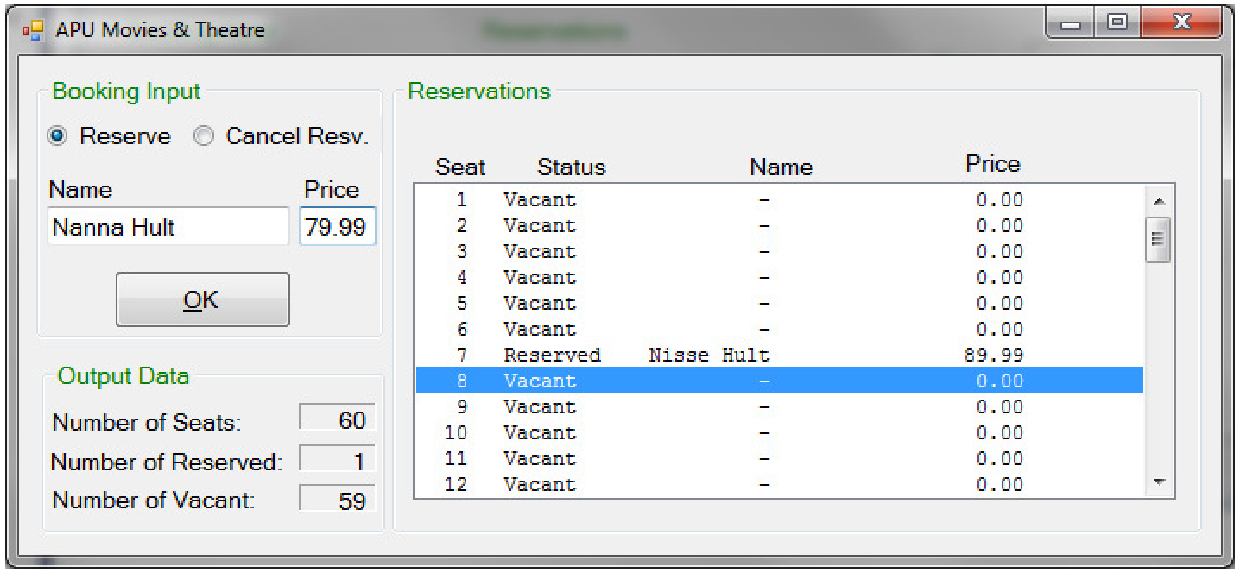
- Putting functionalities so the application works properly.
- The user should be able to reserve a vacant seat, by saving the customer's name and the ticket price.
- The user should be able to cancel an already reserved seat, by erasing the name of the customer and setting the price to 0.0.
- The program should handle the user input to avoid misbehavior.

You may certainly apply your own fantasy for designing the GUI in your way, apply other logics, provided that you take the requirements and the general quality standards set in this course (see separate document) into consideration. Otherwise follows the scenarios below:

3. General requirements

In order to understand the requirements, it should be explained that the idea behind this assignment is centered on using one or two dimensional arrays. We will be using two arrays, one with elements of the type **string** for storing names of the customers and one with elements of the type **double or decimal** for storing the prices. Although the two arrays are not by definition related to each other, it can be assumed that a seat number corresponds to the same index in both of the arrays. Throughout this document, we refer to these arrays as the **m_nameList** and **m_priceList** respectively.

At program start, the elements of the **m_nameList** are initialized as **null** and the elements of the **m_priceList** are initialized to 0.0 (by the compiler).



[✓ Reserve] Cancel Resv.

Name	Nanna Hult
Price	79.99
Number of Seats:	60
Number of Reserved:	1
Number of Vacant:	59

```
Seat   Status           Name              Price
  1    Vacant             -                0.00
  2    Vacant             -                0.00
  3    Vacant             -                0.00
  4    Vacant             -                0.00
  5    Vacant             -                0.00
  6    Vacant             -                0.00
  7    Reserved     Nisse Hult            89.99
  8    Vacant             -                0.00
  9    Vacant             -                0.00
 10    Vacant             -                0.00
 11    Vacant             -                0.00
 12    Vacant             -                0.00
```

2.1 List all seats in a ListBox (or a ListView) when the program starts up. All seats are vacant at this time.

2.2 **Reserving a seat:** Consider the following scenario for a reservation to take place. The user

2.2.1 selects a seat by highlighting a row in the ListBox,

2.2.2 writes the name of the customer in the textbox at the left and fills the price in the second text box,

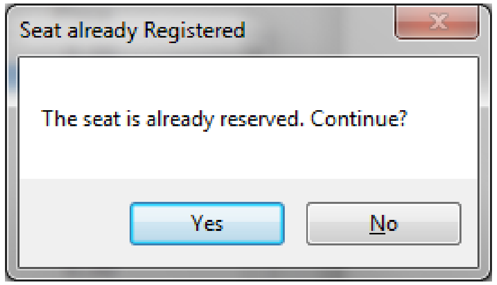
2.2.3 selects the option **Reserve**, and clicks the OK button.

2.2.4 The program checks the validity of the input (name and price). If the input is not valid, the user should be notified and the code execution should return to the GUI. The user can then correct the faulty input and try again.

2.2.5 If the input is validated, i.e., a name is given and the price is a valid number, the program then saves the name and the price in the related arrays using the seat number as index.

Hint: The index of highlighted row in the ListBox is available in your code through the ListBox's **SelectedIndex** property which will be an integer counted from 0.

2.2.6 If the **Reserve** option is selected and the user highlights an already reserved seat in the ListBox, and then presses OK button, the program should show a message box to let the user confirm the action before overwriting the existing reservation.


Seat already Registered

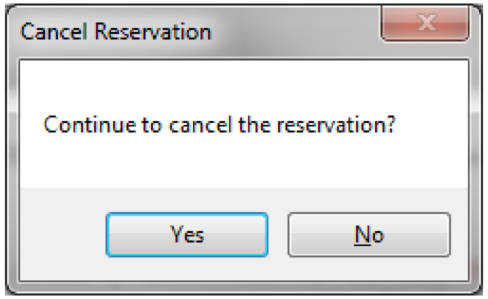
The seat is already reserved. Continue?

2.3 **Canceling** a seat: Consider the following scenario. The user

2.3.1 selects a row that is already reserved in the ListBox,

2.3.2 selects the **Cancel Reserv**. option,

2.3.3 receives a notification to confirm the cancelation.


Cancel Reservation

Continue to cancel the reservation?

2.3.4 If the user clicks the **Yes** button, the program should then set the value of the element in the **m_nameList** to null (or **string.Empty**) and the value of the element in the **m_priceList** to 0.0, using the selected seat number as index.

2.3.5 If the user clicks the **No** button, the code execution should return to the GUI.

2.4 The GUI is to be updated every time a reservation or a cancelation takes place. Update the output data and also the ListBox.

2.5 **Input control**:

2.5.1 The name of the customer should not be empty and should at least contain one character other than a blank space.

Hint: All string variables have a method named **Trim** which deletes blank spaces at the beginning and end of the string. You can take advantage of this feature to get rid of the blank space characters when reading the names from the GUI.

2.5.2 The price should be any amount that is equal or greater than zero.

2.6 All output controls are to be cleared from design text (TextBox1, etc). The **ComboBox** should be made read-only to prevent the user from entering text in the **TextBox** part of it. Set the ComboBox's **DropDownStyle** to **DropDownList** in VS.

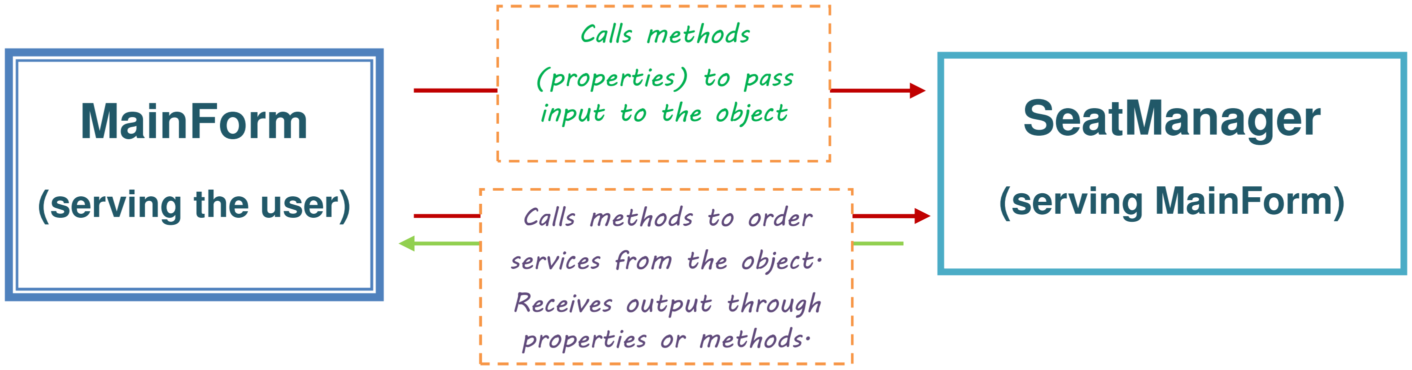
Hint: If your formatting is correct but you still do not get straight columns in the ListBox, it depends most likely on the choice of the font of your ListBox. To remedy this problem, you need to select a font with a constant width for all characters. Use **Courier New** and that will fix the problem.

Structural requirements

2.7 All user-interactions must be coded in the **MainForm**, but this class should not contain other logics than those related to IO (input/output).

2.8 The arrays as well as all other related logics must be coded in a separate class, **SeatManager. (SeatManager.cs). The MainForm** should use an instance of the **SeatManager** (aggregation) and communicate with the instance through the instance's methods (and constructors)

An OOP Note: The **SeatManager** class should not have any interaction with the user. Remember that the **MainForm** object is said to be a client object to **SeatManager**. Actually, the **SeatManager** should not need to know which clients (other classes) are using its objects. It should be independent of its clients. This times it is the **MainForm** that is using the **SeatManager**; another time it might be another object, totally different from the **MainForm** that will (should be able to) use it


MainForm
(serving the user)
Calls methods (properties) to pass input to the object
Calls methods to order services from the object. Receives output through properties or methods.
SeatManager
(serving MainForm)

3.1 The **SeatManager** should create the arrays with a capacity equal to the total number of seats, but we let the **MainForm** provide this value. A solution is to declare the total number of seats as a readonly instance variable in the **SeatManager** class. A **readonly** variable works as a constant but it doesn't need to be initiated at declaration. This value should come from the **MainForm** with the constructor call.

To make this work, **SeatManager** should provide a constructor with one parameter for accepting this data from its client, the **MainForm** . The **MainForm** may then instantiate the **SeatManager** (create an object of using the keyword new) and pass a value (say 350). This number should be used in the SeatManager class to set the capacity of the arrays.

3.2 You may alternatively write a constructor with two parameters, one parameter for the total number of rows and another parameter for the total number of columns. The total number of seats in theatre will then be the result of the multiplication of the two values (for example rows = 35 and columns = 10 giving a total number of seats = 35 x 10 = 350).

3.3 All tasks in the **SeatManager**, for instance **ReserveSeat**, **CancelSeat**, **GetNumberOfVacantSeats**, etc. should be coded in separate methods. The methods should not be long. Break down long methods into smaller ones.

3.4 **Use of collections is not allowed** in this assignment in order to make sure that you exercise and learn using one or two dimensional arrays (dependeng on which part of the assignment you are doing), as these are a part of the basics in learning a programming language. We will be using collections a lot in the future assignments. However, you may use collection provided that you also use a one or two dimensional array.

Remember:
- Make sure to always check the value of the **SelectedIndex**, because as soon as the ListBox loses focus, the value of the ListBox object's **SelectedIndex** is set to -1 automatically and if you use this as an index in your code, it will cause an Exception, resulting consequently in an abnormal program termination.

4. Requirement for a passing grade (Grade G or ECT C)

4.1 **SeatManager** should store the customer names in a one-dimensional array **(m_nameList)** and the prices in another one-dimensional array **(m_priceList)**. The numbering of seats in this case will be a sequence beginning from 0 to total number of seats -1. The figure below shows a visual image of the seat numbering

n = total number of seats.

0	**1**	**2**	**3**	**4**	**5**
6	7	8	9	10	.
.	.	.	.	.	.
.	.	.	.	.	n-1

The numbers 0, 1, …n-1 show the indexes of the elements (position numbers), not the values. The value in each cell will be the name of the customer in the **m_nameList**, and a value equal to or greater than zero in the **m_priceList**. .

```
int index = 7;
m_nameList[index] = "Nisse Hult";  //seat nr 7, (second row, second column),reserve
m_priceList[index] = 89.90;
```

5. Requirement for a passing with distinction grade (Grade VG or ECT A and B)

The main requirement here is to use two-dimensional arrays instead of one dimensional arrays. The rest of the requirements (those that are applicable) are as above. You can assume that the seats are numbered by rows and columns as shown in the table below. All rows have the same number of columns.

5.1 The **SeatManager** class should store the customer names and ticket prices in two two-dimensional arrays. A seat in this case is indexed by a combination of the row and column number.

5.2 The GUI should allow the user to filter the list of seats by options, "**All seats**", "**Only reserved seats**" and "**Only vacant seats**". The filtering should work well so the ListBox should show only seats that match the option. The number of items displayed in the ListBox should also match the number of seats that match the option. A solution with empty lines to hide items in the ListBox is not accepted.

5.3 The options are to be grouped using en enum.

5.4 Showing only reserved sears and only vacant seats are for the purpose of displaying. When these choices are selected, reservation and cancellation of seats will requires some extra programming. Therefore, you may disable the OK button when any of these two options are selected or let the user be notified that reservations can only be managed when the option "All seats" is active (see the run example below).

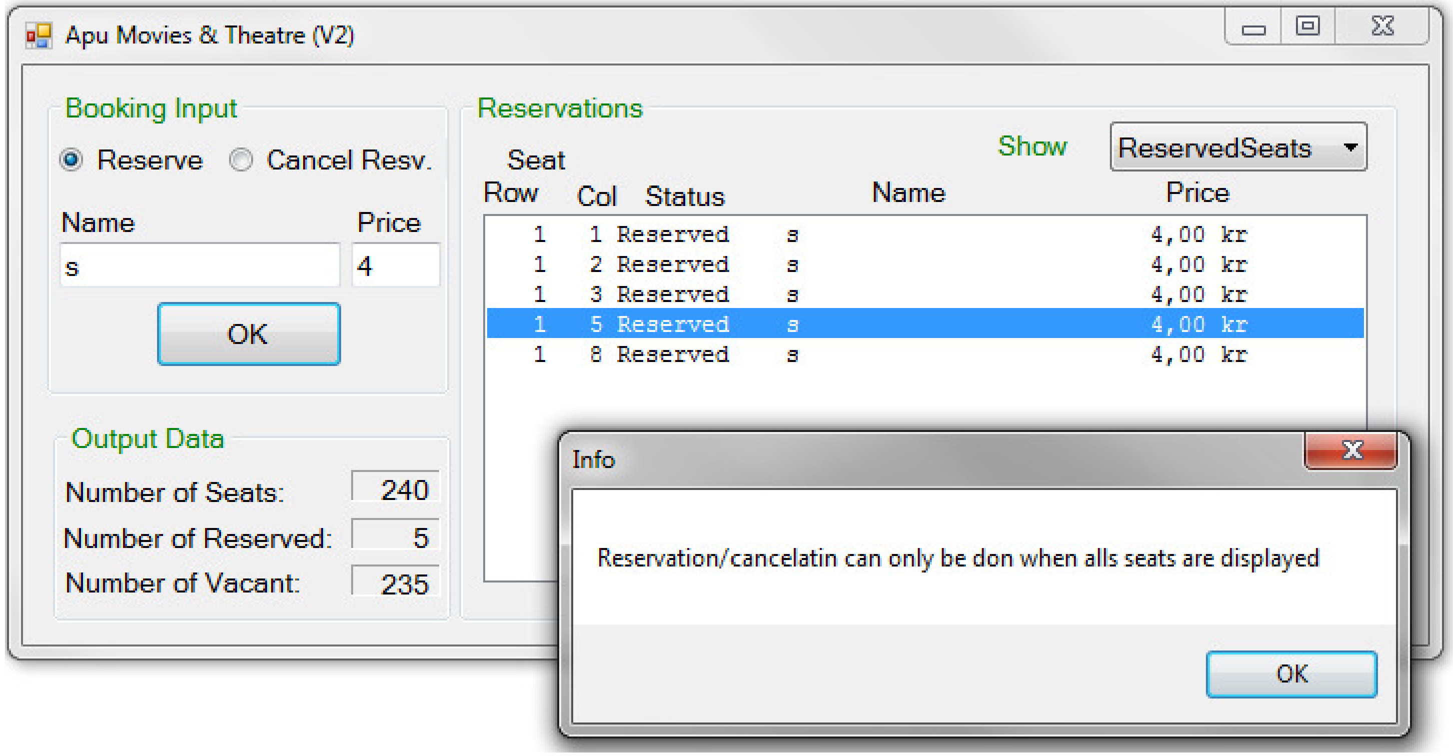
	0	*1*	*2*	*3*	*4*	*5*
0	0,0	0,1	0,2	0,3	0,4	0,5
1	1,0	1,1	1,2	1,3	1,4	1,5
2	2,0	2,1	2,2	2,3	2,4	2,5
3	.	.	.	.	.	.
4	.	.	.	.	.	x

Num of rows = 5 (indexed 0 to 4)
Num of cols = 6 (indexed 0 to 5)

Total num of seats = 5 *6 = 30

x = [total cows-1, total columns -1]

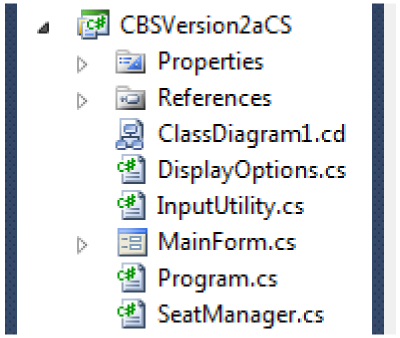
The number 0,0, 0,1, …2,5 denote row, col and are not values!



```
  1   1 Reserved   s                    4,00 kr
  1   2 Reserved   s                    4,00 kr
  1   3 Reserved   s                    4,00 kr
  1   5 Reserved   s                    4,00 kr
  1   8 Reserved   s                    4,00 kr
```

Reservation/cancelatin can only be don when alls seats are displayed

6. The Project



Copy the last assignment project to a new folder to reuse the files in this assignment. You can rename files, if needed, e.g. Assignment3.sln to Assignment4.sln. Open the solution and continue to code for this assignment.

7. CBSVersion3 – Array of Objects (Optional – not graded)

This version is optional and is presented as extra exercise. It is desired by the cinema owner to store more data about a seat,

- Seat category (Business Class, Economy Class, Handicap Seat, Staff Seat) etc).
- The phone number of a customer.
- The first name and the last name separately.
- Rows may have different number of seats.

As you may have noticed from the descriptions so far, each time we have to add and handle a new type of data for a seat, there comes a lot coding, if we are determined to continue using one or two dimensional arrays. To create more arrays is definitely not an effective way.

A good solution is to define a class **Seat** and encpsulate every data desired about a seat, and write methods to handle all operations about a seat. In the **SeatManager** class, you can then declare a fixed-size array (as in the previous versions) with element of the Seat class. Fixed-sized arrays are sometimes referred to as static array, meaning that the size of the array is determined at compile time. We will be utilizing mostly dynamic arrays of objects in the future. During the rest of this course as well as in advanced courses (and also in our daily programmer lives), we will be using collections, which are advanced list types but are easy to work with. .NET supports many collection types.

Create a new project in your solution and name it CBSVersion3. Copy files from Version 2, or create new ones and then copy code after your needs.

Design your GUI using your own fantasy. Write the **Seat** and **SeatManager** classes first and then come back to the MainForm and write code so everything works the way expected.

Once again, the only reason to not choosing this solution for the assignment is to learn and exercise with basic array types as an important part of language basics. We will have ample time to use collections and dynamic lists in our future solutions.

8. Help and guidance

A detailed guidance and instructions are availabe in a separate document in the module.

Good Luck!

Farid Naisan,

Course Responsible and Instructor