Assignment 4: Properties and Arrays

APU RECIPE BOOK

1. Objectives

The main objective of this assignment is to work with one-dimensional arrays and learn how to write and use properties to access values saved in the private fields of an object.

Arrays are used to save a list of values of a value type as well as a list of objects. Moreover, this assignment gives further training in using constructors, properties, methods and enums. In this assignment, you will also learn to use multiple forms and establish communication between them.

2. Description

In this assignment, we are developing a digital Recipe Book for use by households and also restaurants. The application should have the following features:

- Add a new recipe with a name and instructions (and other description).
- Add a list of ingredients to each recipe.
- Display a list of the registered recipes.
- Edit an existing recipe and its ingredients.
- Remove an existing recipe.

The application should use an array of Recipe objects.

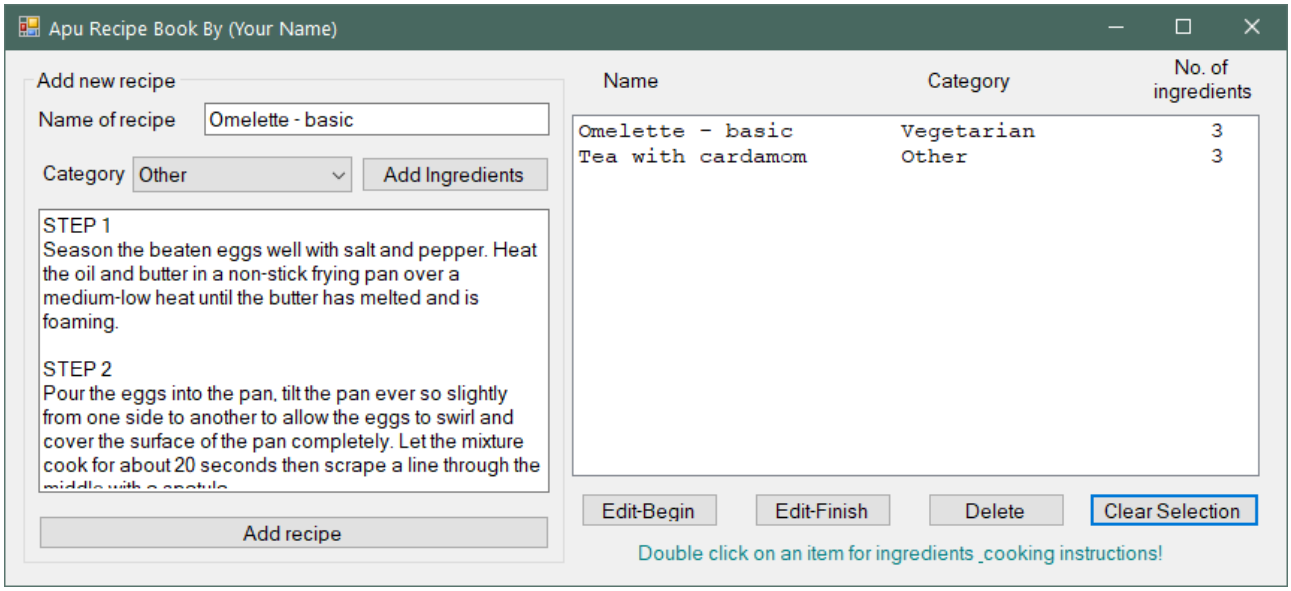
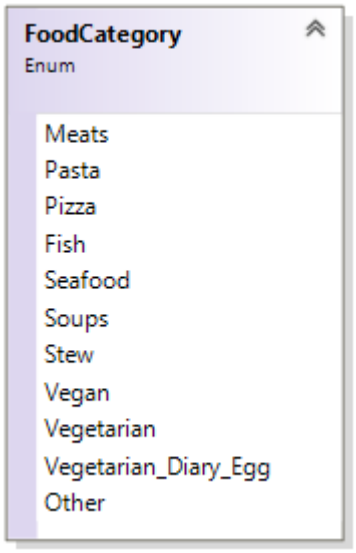


Figure 1: Running the program

Every recipe has:

- a list of ingredients (strings)
- a description (instructions)
- a category as listed in the enum FoodCategory.



The user should be able to give a name, a description (cooking instructions and other comments), and a category type to define a recipe (an object of **Recipe**). She should also be able to specify a list of ingredients which can be saved as a list of strings. The program should then save the recipe as an object in an array of objects (**RecipeManager)**.

2.1 Add a recipe

When adding a new recipe, the user writes the name of the recipe in a TextBox intended for that, then selects a category from a ComboBox list , and writes instructions in the large TexBox. Before saving the recipe object, the user has to provide the ingredients. See Figure 1 on the previous page.

Adding ingredients

2.2 When the user clicks the **Add Ingredients**, a new Form is displayed for inputting ingredients related the current recipe, as in Figure 2.

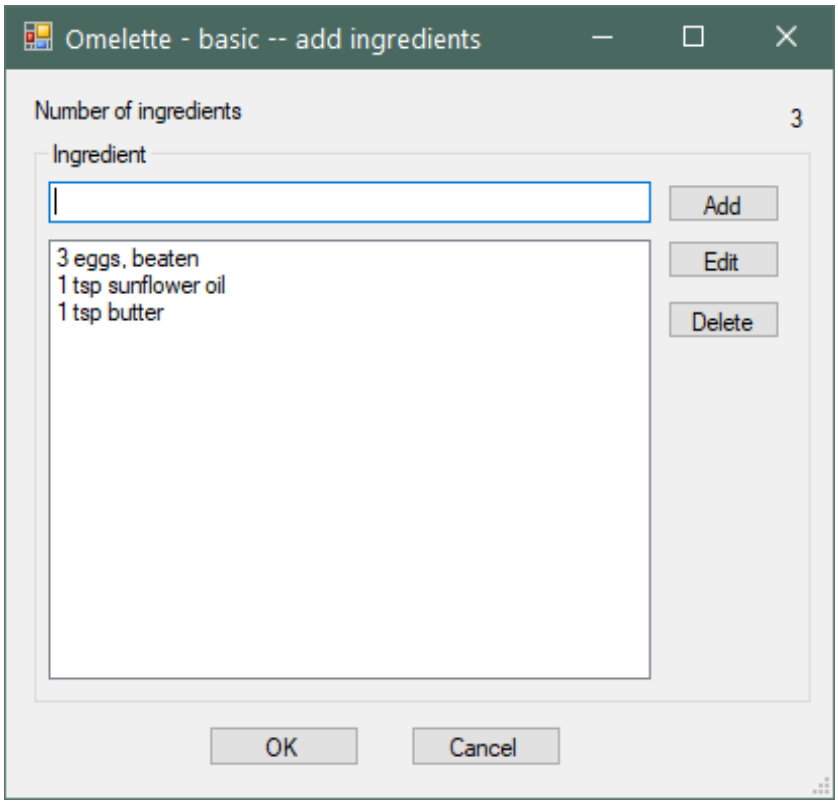


Figure 2: Add ingredients

2.3 For this part, a new Form (**FormIngredients**) is displayed, where the user uses the TextBox to specify a text for an ingredient, clicks the **Add** button to add to the ListBox.

2.4 After closing the form and returning to the main form, the user can click the **Add Receipe** button to register the current recipe with data from the GUI. **FormMain** saves the recipe object in the **recipeManager** and updates the ListBox (at the right side the GUI, Figure 1).

Viewing the ingredients and description

2.5 The ListBox containing the registered recipes becomes too long if all information for a recipe is to be listed. Therefore, for every recipe a chopped amount of information can be formatted and placed in the list. To see the ingredients and description for a certain recipe item, the user should double-click on an item in the ListBox and the program should display both ingredients and description (Figure 3). Alternatively, you can use a button to show the information.

Figure 3 demonstrates the feature (as an example) using a MessageBox. You may of course use a form if you would like to.

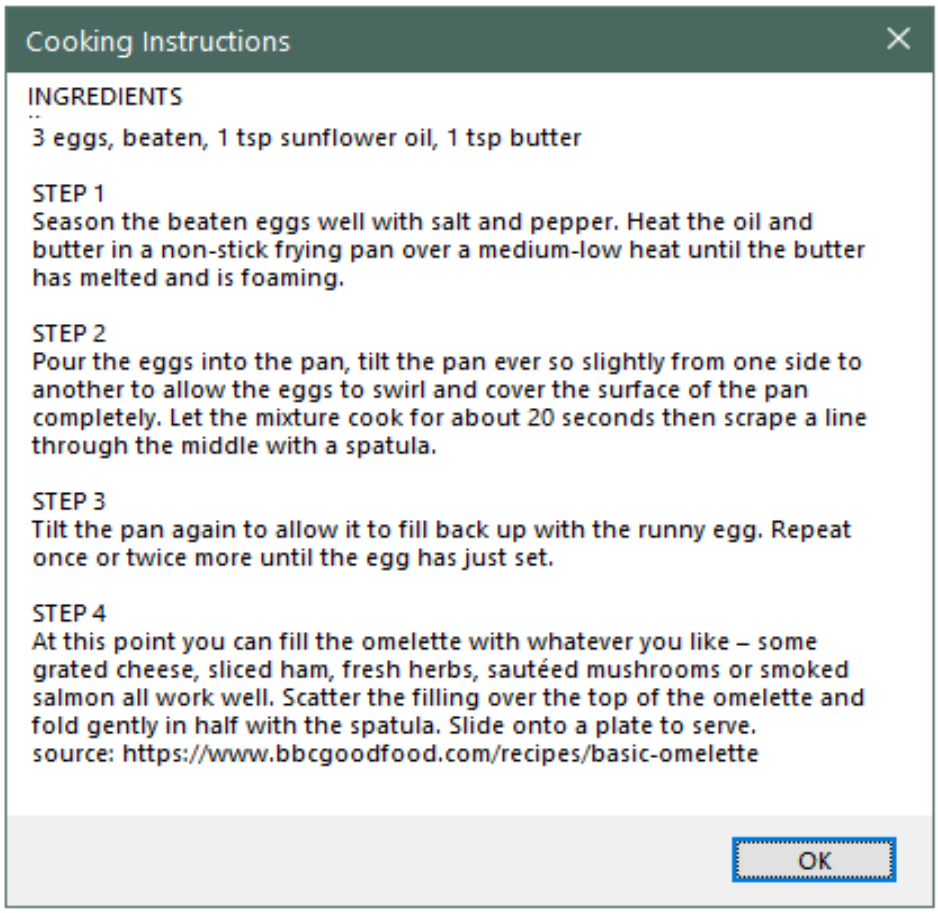


Figure 3: view ingredients and description

3. The Project

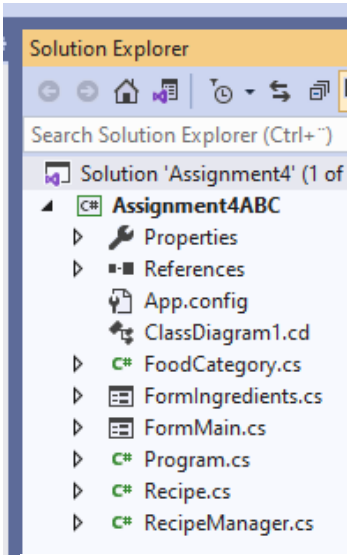


Figure 4: Sample project

Create a Windows Forms Desktop Application, but you may try working with Windows Presentation Foundation (WPF) if you would like to. The graphical components look similar but they work quite differently. It is also allowed to use .NET Core supporting Windows Forms.

Figure 4 presents a suggested project view; this, together with the class diagram shown in Figure 5 should give you an idea of how to organize the file structure for the application.

FoodCategory is an enum defining food categories, as described earlier.

FormMain is the starting object. It uses an object of the **FormIngredients** class to provide an interface for inputting ingredients, as shown in the run example images above (Form 2).

Recipe is a class that defines a recipe with name, category, ingredients and a description. All cooking instructions and other texts pertaining to a recipe are stored in this instance variable. The ingredients are saved in an array of strings in this class.

RecipeManager defines an array of recipes (Recipe objects) and performs all necessary operations on the list. Adding, removing and editing of a Recipe is the responsibilities of this class. This class calls the methods of the Recipe class when performing operations on a single object, e.g. when adding ingredients or querying the number of ingredients in a recipe.

Class Diagram: Classes and their associations

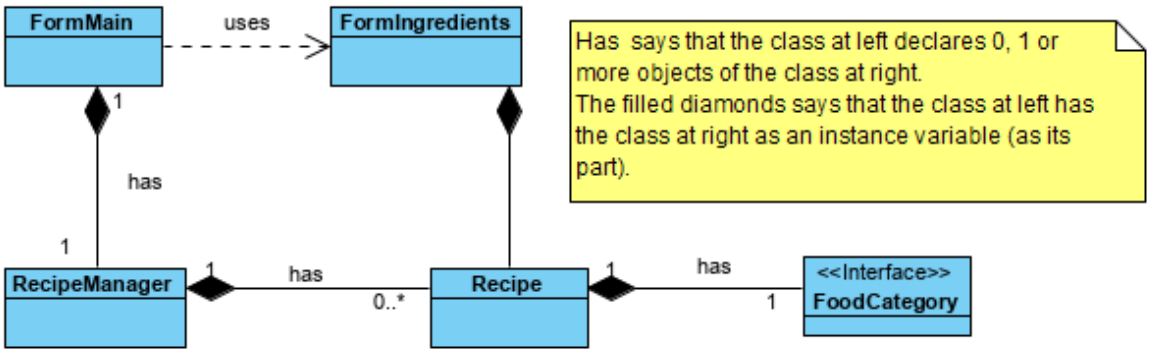


Figure 5: Class diagram

A detailed diagram for each of the classes is provided later in this document.

Note: If you would like to apply your own solution, have the requirements in mind and as long as you maintain a good code quality, there will be no problem in grading your assignment. In case of unsatisfactory results, you will get a chance to do complementary work and resubmit.

4. Features and requirements

4.1 All features visualized in Figure 1 should be implement. The user should be able to edit a selected recipe and its ingredients as well as remove a selected recipe from the array.

4.2 The clear button should unselected the highlighted item in the list box and an edit if not completed should be canceled.

4.3 Use properties to access private fields of the classes. Validate the data (value) in the set-accessor of the properties.

4.4 All instance variables are to be private. Auto-implemented properties are not allowed.

4.5 **RecipeManager** should use a private array of **Recipe** elements.

4.5.1 Use of collections (like List<T>, or Dictionary) are not allowed.

4.5.2 Do not use a property to provide access to the array.

4.5.3 You can use a method that returns an element at given index.

4.6 The maximum number of ingredients for a single recipe as well as the maximum number of recipes that the application can save can be hardcoded in the FormMain as two constants. These values should then be passed to the Recipe and the **RecipeManager** when calling their constructors. Suggested max values:

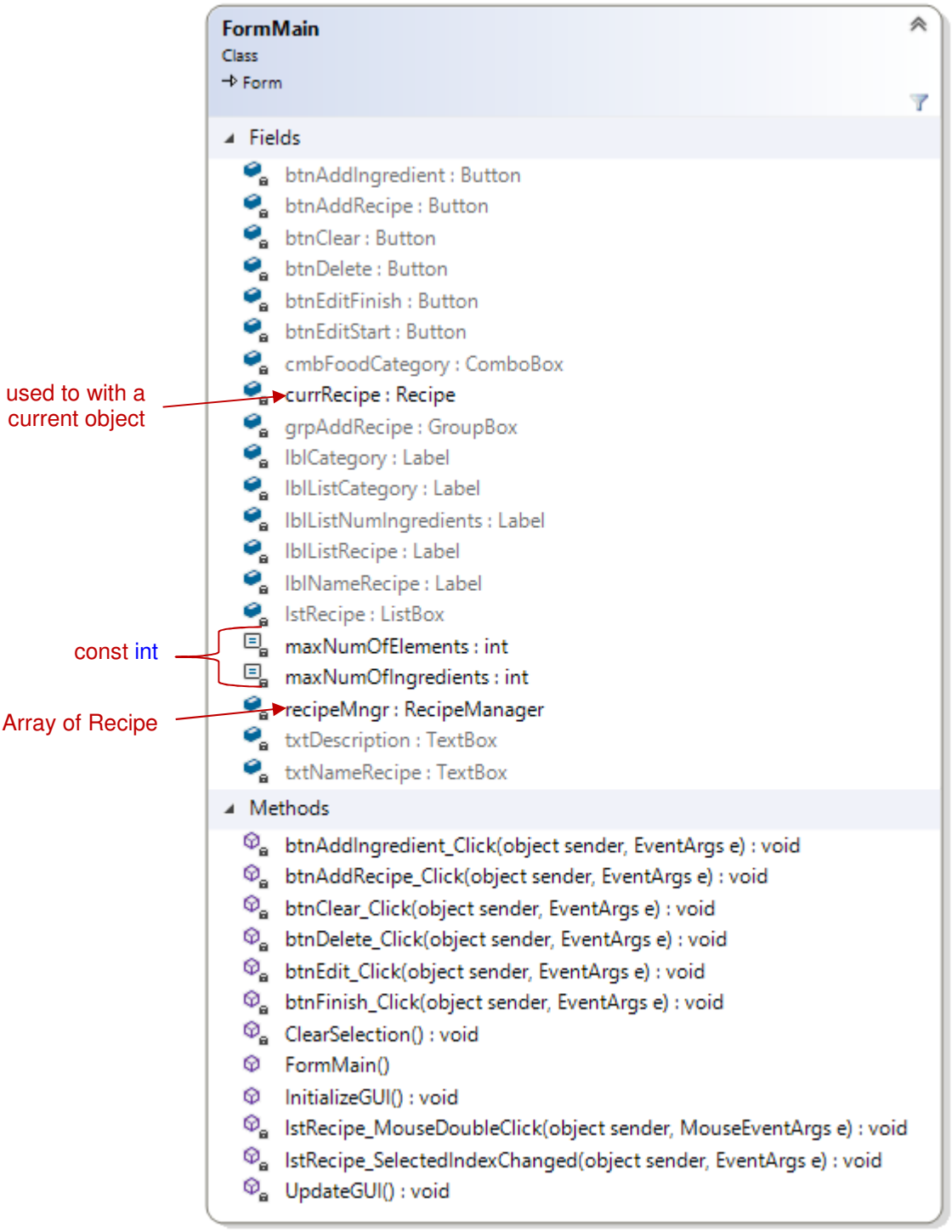
Max number of ingredients = 50;
Max number of recipes = 200

4.7 The application should work well and not crash or generate exceptions due to invalid user input or bugs in the program.

4.8 The user should select an option from the list in the ComboBox (Category) and not be able to write a text in the textbox part of the control. In other words, the ComboBox should be readonly. This can be achieved by setting the DropDownStyle property of the ComboBox to DropDownList.

5. Some quick help

FormMain Class diagram



5.1 **currRecipe** is used to store temporarily the data for a recipe being inputted by the user. When all data is ready, the object is sent to the **recipeManager** to be added in the array.

Important: Recreate currRecipe (currRecipe = new Recipe…) when you are done with one recipe and will start working with a new recipe. Why? A good question to discuss in the forum.

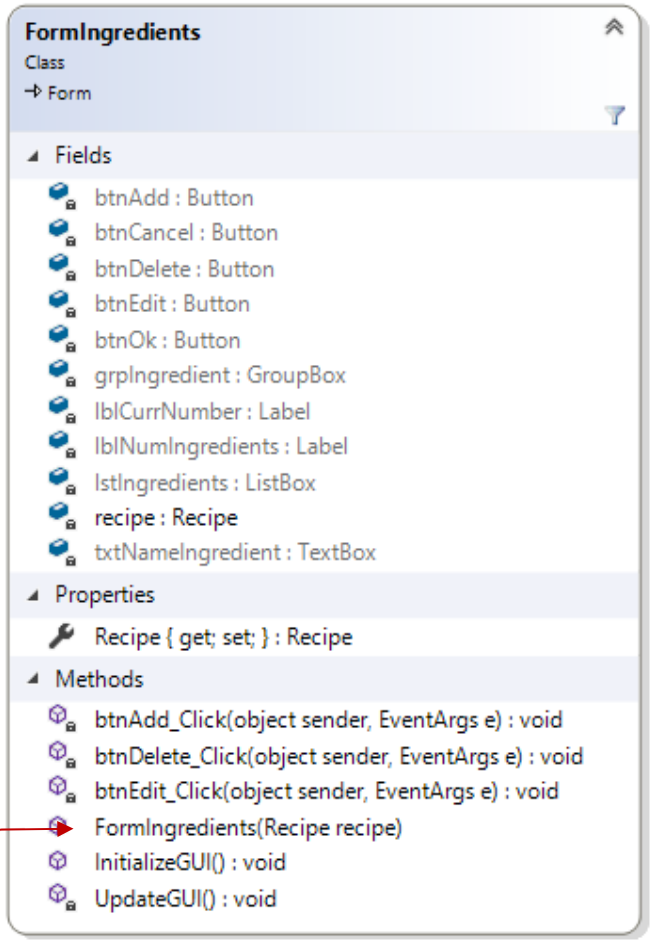
Note: Using the **currRecipe** makes life easier but it is meant that you must follow this solution. Take it as a hint!

Constructor call:

```
private Recipe currRecipe = new Recipe(maxNumOfIngredients);
```

5.2 The variable **maxNumOfIngredients** is an argument sent to the constructor. The whole statement creates an object of Recipe and the object is available through the reference variable **currRecipe**.

The FormIngredients class



*Constructor with a parameter of the type **Recipe**. (You can change the constructor that is prepared by VS)*
*FormMain passes the **currRecipe** to this form to be filled with ingredients. An alternative to this is to use the set-property connected to **recipe**.*

5.3 When the button **Add ingredients** on FormMain (Figure 1) is pressed, the **FormIngredients** can be loaded as shown below:

Note that currRecipe has the ingredients from **dlg** when FormIngredients is closed, due to the fact that it is passed by reference.

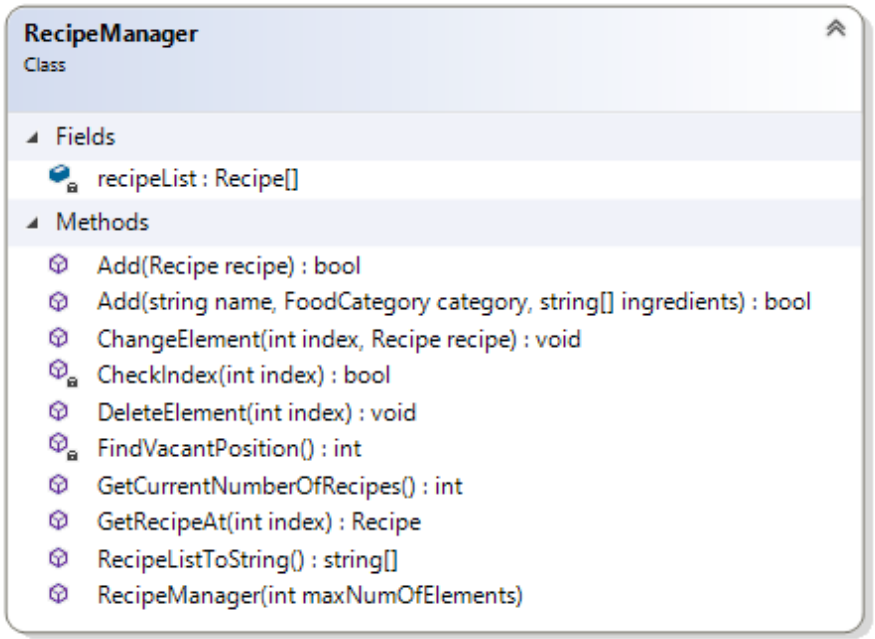
```
private void btnAddIngredient_Click(object sender, EventArgs e)
{
    //Creates new ingredients form and opens as a dialog box
    FormIngredients dlg = new FormIngredients(currRecipe);
    DialogResult dlgResult = dlg.ShowDialog();

    //Manages case where user click ok button
    if (dlgResult == DialogResult.OK)
    {
        //If no ingredients were added, shows message dialog
        if (currRecipe.CurrentNumberOfIngredients() <= 0)
        {
            MessageBox.Show("No ingredients specified", "Error");
        }
    }
}
```

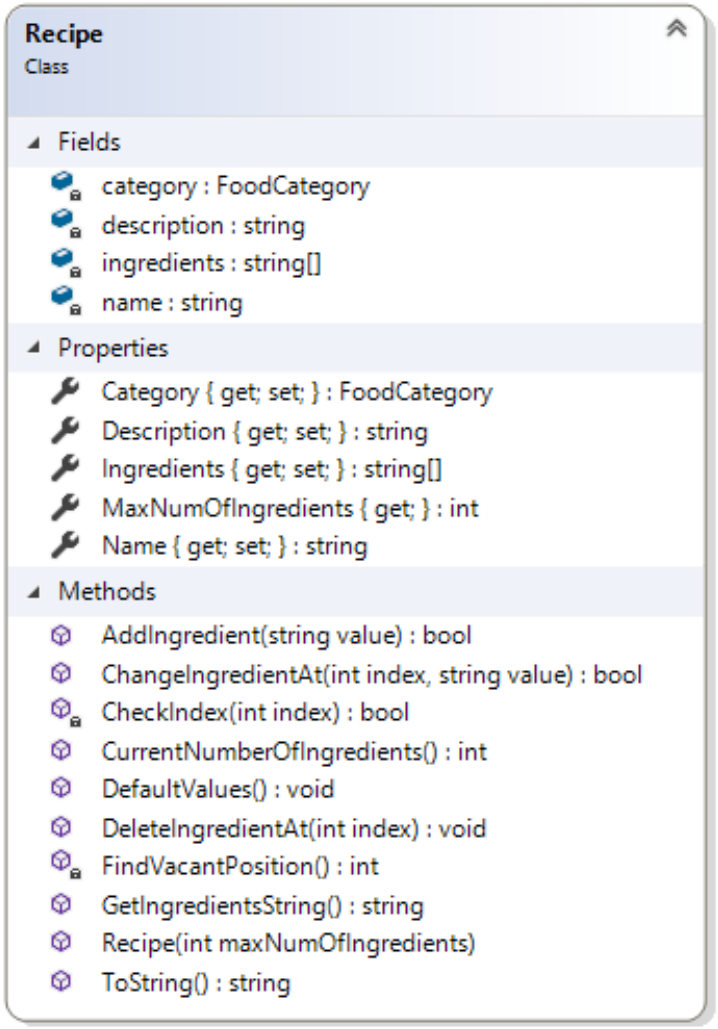
In the above code, the data is saved in the currRecipe but not yet added to **mngrRecipe** object. This is done when the user clicks the **Add recipe** button.

RecipeManager class diagram

The class diagram for the RecipeManager is given below:



The class diagram for Recipe:



Creating arrays and array elements:

- If you have an array of reference types, i.e. objects, you create the array first and then create each element of the array

```
public class RecipeManager
{
    private Recipe[] recipeList; //Declares an array for storing recipes

    /// <summary>
    /// Constructor for the class
    /// </summary>
    /// <param name="maxNumOfElements"></param>
    public RecipeManager(int maxNumOfElements)
    {
        //Creates array that has a size equal to the input variable
        //but all elements are null at the creation
        recipeList = new Recipe[maxNumOfElements];
    }
```

To add an element that comes through a method argument, it must not be null (meaning that it must be created in the caller method).

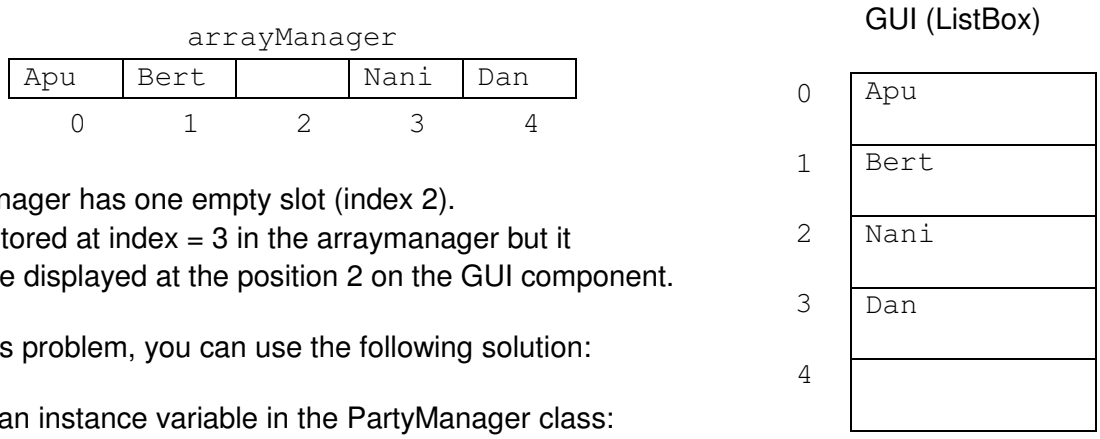
```
public bool Add(Recipe recipe)
{
   //check so recipe is not null before adding to the
   //recipList.

   }
```

6. Keep array elements arranged from index =0 without empty slots.

When an element is deleted from a position in the array, it provides an empty slot. Having empty (unused) element in different positions in the array provides problems in keeping track of filled and empty locations. The indexes will then not match the index of a list component on the GUI where empty elements are not be displayed.

Assume we have an array of strings.



arrayManager has one empty slot (index 2).
Nani is stored at index = 3 in the arraymanager but it should be displayed at the position 2 on the GUI component.

To fix this problem, you can use the following solution:

Declare an instance variable in the PartyManager class:

```
private int numOfElems = 0; //num of elems with values
```

When adding a new element, add the new element at the index = numOfElems and then increment the variable:

```
if (numOfElems < guestList.Length)
{
    guestList[numOfElems] = FullName(firstName, lastName);
    numOfElems++;
}
```

Every time an element is removed (deleted), decrement the variable: numOfElems--; and meanwhile move all element which are at the right of the index being removed one step to the left.

Use the code below in the RemoveAt method (or DeleteAt)

```
guestList[index] = string.Empty;
numOfElems--;
MoveElementsOneStepToLeft ( index );


private void MoveElementsOneStepToLeft(int index)
{
    for (int i = index+1; i < guestList.Length; i++)
    {
        guestList[i-1] = guestList[i];  //move 1 step to left
        guestList[i] = string.Empty;  //empty its place
    }
}
```

This solution can be used with array of other types and objects as well. With array of objects, use null instead of string.Empty in above. Actually null can be used with array of string too as string is also an object in C#. With array of numeric elements, the default value (empty value) should be used based on the problem.

7. Submission

Compress all your files and folders (particularly the folder Properties) that are part of your project into a ZIP or RAR file. Upload the compressed file via the same page where you downloaded the assignment.

Good Luck!

Farid Naisan,

Course Responsible and Instructor