# Waiter System (WaS) Workshop 6

**Objectives**

The main objectives of this **pilot system** are to enhance your understanding of 3 tier applications and OO principles and to enable you to

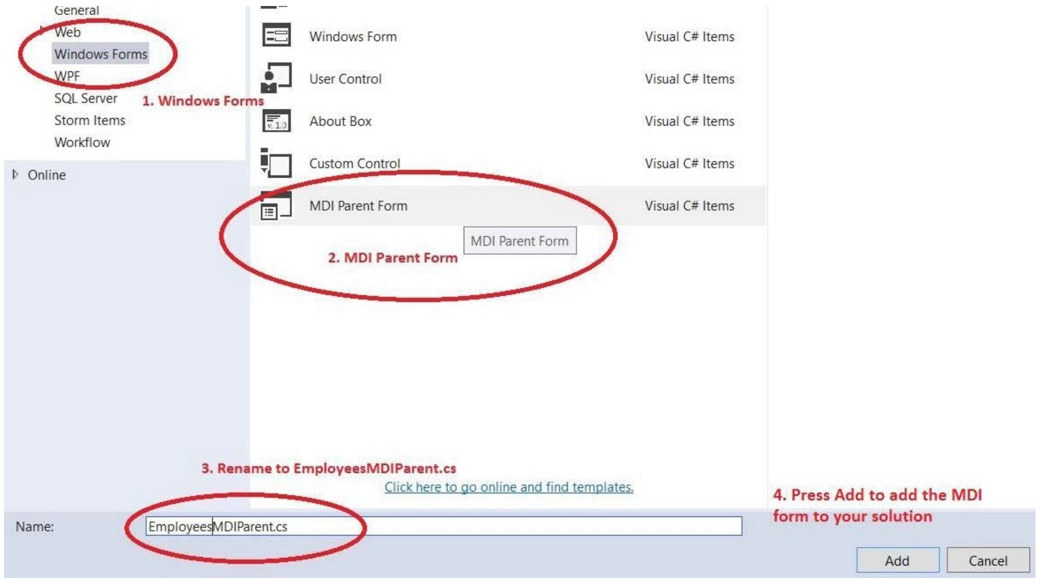
- Work with multiple forms.
- Use inheritance and composition.
- Use collections and control arrays
- Maintain all the waiters in the system

## Adding an MDI

**Open your application and ensure all tables have at least 5 records in them.**

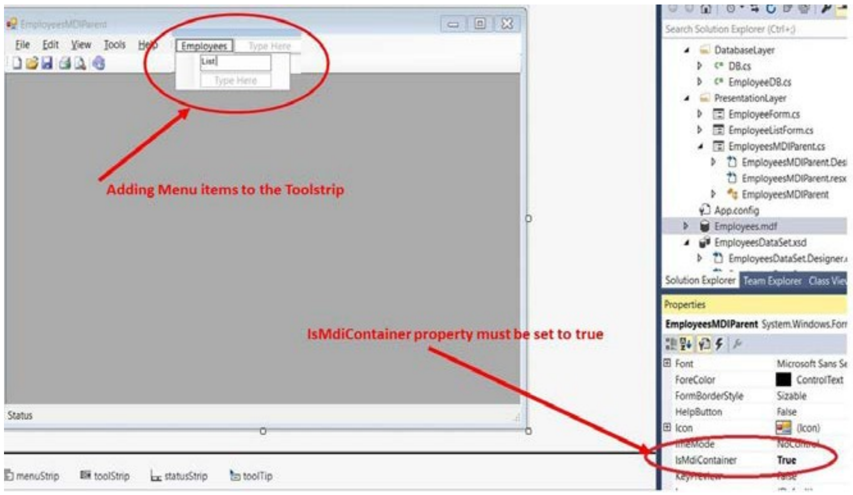
1. Add an MDI Form to the application. The steps below are given as a guidance.

**1.1** Right click on the Presentation folder in the Solution Explorer Window and select Add>Windows Form. Then select MDI Parent form as seen below.


General
Web
Windows Forms
WPF
SQL Server
Storm Items
Workflow
Online
1. Windows Forms
Windows Form Visual C# Items
User Control Visual C# Items
About Box Visual C# Items
Custom Control Visual C# Items
MDI Parent Form Visual C# Items
MDI Parent Form
2. MDI Parent Form
3. Rename to EmployeesMDIParent.cs
Click here to go online and find templates.
4. Press Add to add the MDI form to your solution
Add
Cancel

Name: EmployeesMDIParent.cs

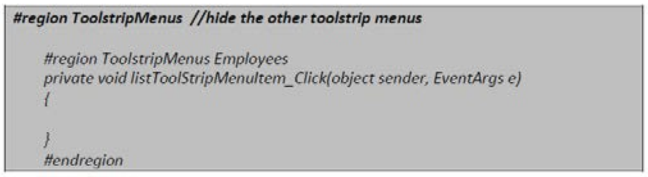
**1.2** Delete the Windows Menu Item on the MenuItem Toolstrip. Add a Menu Item Employees > List (See Figure below) and check that the IsMdiContainer property of this MDI Form is set to true



EmployeesMDIParent System.Windows.For

| | |
|---|---|
| Font | Microsoft Sans Se |
| ForeColor | ControlText |
| FormBorderStyle | Sizable |
| HelpButton | False |
| Icon | (Icon) |
| ImeMode | NoControl |
| IsMdiContainer | True |
| KeyPreview | False |

Once you have done the above, place all the current Toolstrip menus in a ToolstripMenus region to hide the clutter. Double click on the List menu item below the Main menu item Employees to create a toolstrip menu item event for listing. The code snippet below shows you what you should have

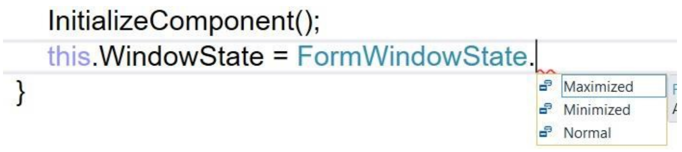


```
#region ToolstripMenus  //hide the other toolstrip menus

    #region ToolstripMenus Employees
    private void listToolStripMenuItem_Click(object sender, EventArgs e)
    {

    }
    #endregion
```

Maximise the MDI form. By setting the WindowState property in the constructor of the MDI form



```
    InitializeComponent();
    this.WindowState = FormWindowState.
}
```

## 2. Changes to the Employee Form

**2.1** Add a public Boolean (bool) variable, *employeeFormClosed*, to indicate whether the form is closed or not. Initialise this variable to false.

**2.2** Add a public property called ***RoleValue***, to set the *roleValue.* The header of the property method has been given for you below:

```
public Role.RoleType RoleValue
```

**2.3** Change the constructor of the form to receive a parameter *aController* of type *EmployeeController*. This will allow you to send a reference to the *employeeController* object every time the form is instantiated and loaded in memory. Assign *aController* to the *employeeController* object

**2.4** Set the value of *EmployeeFormClosed* variable to true in the exitButton_Click event

**2.5** If there is a line in the Load event of the form that instantiates an EmployeeController object, remove it

**2.6** Change the startup form in the Program.cs file to EmployeeMDIParent.

## 3. Adding a listing form

**3.1** Add a form *EmployeeListingForm* to the Presentation folder for viewing and manipulating employee data.

**3.2** Add a label *listLabel* and a ListView control, *employeeListView* to the Form. Change the heading of the form to *Employee Listing*.

**3.3** To be able to test if the form is open or closed we need to declare a public Boolean variable on form level. Call this variable *listFormClosed.*

**3.4** Set the Boolean value of listFormClosed to true in the FormClosed event of the form (first create a FormClosed event for the form).

**3.5** Declare a collection of *employees* on form level

**3.6** To enable you to list employees according to their roles, declare a form level variable *roleValue* which is associated with the Role class.

**3.7** Define a WriteONLY property *RoleValue* for this form to allow you to set the private variable (only set option).

**3.8** To enable the *EmployeeListingForm* form to communicate directly with the controller class, we will pass an object *empController* of the controller class to the *EmployeeListingForm* as follows:

3.8.1 Declare a reference *employeeController* to the Employee controller object on form level
3.8.2 Create an overloaded constructor which receives the controller class object *empController*
3.8.3 Assign *empController to the form level employeeController*

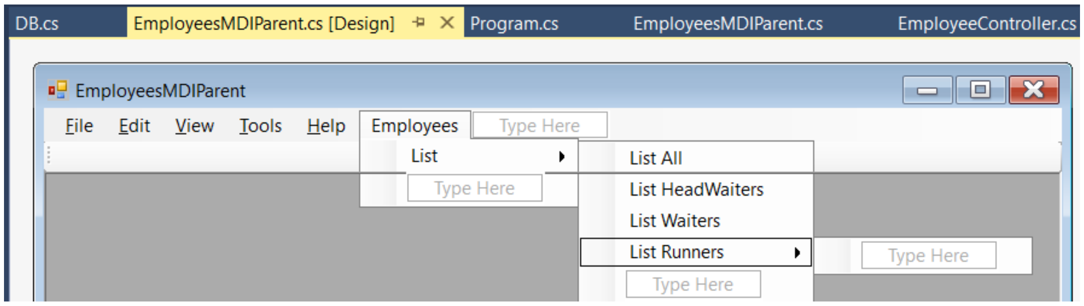
## 4. Setting up the MDI Form for utilising the child forms:

**4.1** The “child” forms live within the parent (MDI) form. We thus need to declare references to both the EmployeeForm as *employeeForm*; and the EmployeeListingForm as *employeeListForm* on form level.

**4.2** Declare a reference *employeeController* to the EmployeeController class.

**4.3** Instantiate an *employeeController* object in the EmployeeMDIParent constructor.

**4.4** Complete the Submenus for the employeesToolStripMenuItem. See below



***4.5*** Double click on each of the submenu items to generate each specific ToolStripMenuItem_Click event (eg listAllToolStripMenuItem_Click). Place all these events in the region titled: ***Employee ToolStrip Menus for Listing***

**4.6** Add a *Create a New ChildForm* region for both the EmployeeForm and the EmployeeListForm (same region for both).

4.6.1 Add two methods: *CreateNewEmployeeForm* and *CreateNewEmployeeListForm.* These methods are private, receive no parameters and return nothing. For each one of the two methods, do the following steps:

a. Instantiate a form object which needs to receive a reference to the *employeeController* to have access to the SAME data in memory.

b. Set MDIParent form property of each form to this MDI form, an example is given for the employee form in the CreateNewEmployeeForm:

***employeeForm.MdiParent = this;***

c. Set the starting position of both forms to be the Centre. An example is given for the employee form in the CreateNewEmployeeForm:

## 5. Changes to the EmployeeController class

Add a *FindByRole* method to the EmployeeController class to select only those employees of a specific role. This method searches through all the employees to find only those with the required role. The code is given below. You are required to finish it.

```
public Collection<Employee> FindByRole(Collection<Employee> emps, Role.RoleType roleVal)
        {
            Collection<Employee> matches = new Collection<Employee>();

            foreach (Employee emp in emps)
            {
// 5.1 Write the code here to check if the role has been found (i.e if the employee (emp)
role matches the roleVal, then)
                {
        //5.2 write the code to add the match to the collection, i.e add emp to matches –
        check your notices on how to add item to a collection

                }
            }
            //5.3 write the code to return the collection


        }
```

## 6. Adapt the code for the ListViewControl on the EmployeeListForm

**Add a method *setUpEmployeeListView* to set up the List View control. This method will have the task oflisting ALL the employees or to list the employees by role. Within this method:**

**6.1** Declare the following variables

- *employeeDetails* of datatype ListViewItem;
- *headW* of datatype HeadWaiter;
- *waiter* of datatype Waiter;
- *runner* of datatype Runner.

**6.2** The employees collection should be assigned to null.

**6.3** Write the code to clear current List View Control

**6.4** Write the code to set Up Columns of List View. An example is given below for two of the fields:

```
employeeListView.Columns.Insert(0, "ID", 120, HorizontalAlignment.Left);
employeeListView.Columns.Insert(1, "EMPID", 120, HorizontalAlignment.Left);

// … do this for the other generic elements
```

For the role field: use a switchstatement to check which role to add, then get all the employees from the *EmployeeController* object - (use the property) and assign to a local employees collection as follows (Note that you will have tofinish the program for the other roles):

```
 switch (roleValue)
{
     case Role.RoleType.NoRole:
          employees = employeeController.AllEmployees; listLabel.Text = "Listing of all employees";
          employeeListView.Columns.Insert(4, "Payment", 100,HorizontalAlignment.Center);
          break;
     case Role.RoleType.Headwaiter:
          //Add a FindByRole method to the EmployeeController
           employees = employeeController.FindByRole(employeeController.AllEmployees,Role.RoleType.Headwaiter);
          listLabel.Text = "Listing of all Headwaiters";
          //Set Up Columns of List View
          employeeListView.Columns.Insert(4, "Salary", 100,HorizontalAlignment.Center);
          break;
     //do for the other roles

}
```

**6.5** Once you get out of the switch statement, add employee details to each ListView item using theforeach statement, and then use a switch statement to determine the role, as follows:

```
foreach (Employee employee in employees)
{
     employeeDetails = new ListViewItem();
     employeeDetails.Text = employee.ID.ToString();
     // Do the same for EmpID, Name and Phone

     switch (employee.role.getRoleValue)
     {
          case Role.RoleType.Headwaiter:
               headW = (HeadWaiter)employee.role;
               employeeDetails.SubItems.Add(headW.SalaryAmount.ToString());
               break;

               //write the code to finish the other employee roles
      }
```

**6.6** Write the code to add the *employeeDetails* to the list view, just after the switch statement. Check your notes on how to add items to a listview.

**6.7** Write the code to refresh the listview, and also set the gridlines of the listview to true. This should be done after the foreach block of statement.

## 7. Changes to the EmployeeMDIParent form to manage the EmployeeListForm.

Add the following code for the ListAll ToolStripmenu event (The code is not complete, you are to complete the code in your program). In this case you have to test whether the form was instantiated or not. You should not have more than ONE instance of the form created in the parent container.

```
private void listAllToolStripMenuItem_Click(object sender, EventArgs e)
 {
     if (employeeListForm == null)
     {
        //7.1 write the code to call the CreateNewEmployeeListForm method
     }
     if (employeeListForm.listFormClosed)
     {
         //7.2 write the code to call the CreateNewEmployeeListForm method
     }
     employeeListForm.RoleValue = Role.RoleType.NoRole;
     //7.3 write the code to call the setUpEmployeeListView method
         //7.4 write the code to show the employeeListForm form

  }
```

Use that code as an example to write the code for the *headWaitersToolStripMenuItem, waitersToolStripMenuItem,* and *runnerToolStripMenuItem*

## 8. Add the Form Load and the Form Activated events to the EmployeeListForm

**8.1** Add the EmployeeListForm_Load event method definition to the form and add the following code to display the listview details within this method:

employeeListView.View = View.Details;

**8.2** Do the same for the EmployeeListForm_Activated event method definition to the form

**8.3** When a form is created, it must be loaded in memory and it must be activated to become the active form. Add these events in code to the constructor of the form. In other words, write the following code to set up Event Handlers for some form events in code rather than through the designer

*this.Load += EmployeeListingForm_Load;*
*this.Activated += EmployeeListingForm_Activated;*

Run your application, debug and ensure you can list employees, depending on their role, in your database.