

Windows Forms Sample Task

March 2023

1 Introduction

The purpose of this document is to demonstrate a sample task you might encounter during Windows Forms graded laboratories. The first part of this guide will show you an example solution for the laboratory part.

2 Laboratory Task

2.1 Description

The task is to create a simple room planner. In the task archive, you should be able to find **WinformsArch_Lab.exe** file - an example app of the laboratory part of the task.

Requirements

- Main window:
 - Starts in the center of the screen
 - Minimum window size: 400x300
 - When the window is resized, it behaves like in the example app.
 - Divided into two panels. You can change panels' widths using a splitter (vertical line between panels):
 - * Right panel: contains two group boxes: "Add furnitures" and "Created furnitures"
 - * Left panel: contains bitmap. Left panel scales automatically when the main window is resized, but the bitmap should stay in the fixed size.
 - Menu: with "New blueprint" button.
- New blueprint
 - Creates new bitmap with the size of the left panel (it should take all empty space)
 - Shortcut: F2
- Add furnitures groupbox
 - Contains four buttons.

- Buttons automatically change their position while resizing or moving the splitter.
- The scrollbar should be visible when needed.
- Buttons
 - Buttons size: 75x75px. Each button uses an image from project's resources (images cannot be loaded from an external location).
 - Button selection - Clicking the button changes background color. After clicking the button again, the background color changes back to white. When one button is selected and we click another button, the first one should be deselected and the new one should be selected.
 - Mouse cursor should change icon while hovering the buttons.
- Created furnitures groupbox
 - Contains a list of created furnitures. Each row should contain furniture's name and its position on the bitmap.
- Adding new furnitures
 - When a button is selected and the user clicks on the bitmap, the selected furniture should be drawn. The clicked point should be the center of the furniture (not it's upper-left corner). After adding new furniture the button is deselected.
 - The list in created furnitures groupbox should be updated.

Hints

- SplitContainer, TableLayoutPanel, FlowLayoutPanel
- RowSpan/ColumnSpan Property, Dock Property, Tag property, PictureBox, GroupBox, SplitterDistance
- MouseDown Event, Graphics methods, PictureBox.Refresh, WindowsStartLocation

Grading criteria

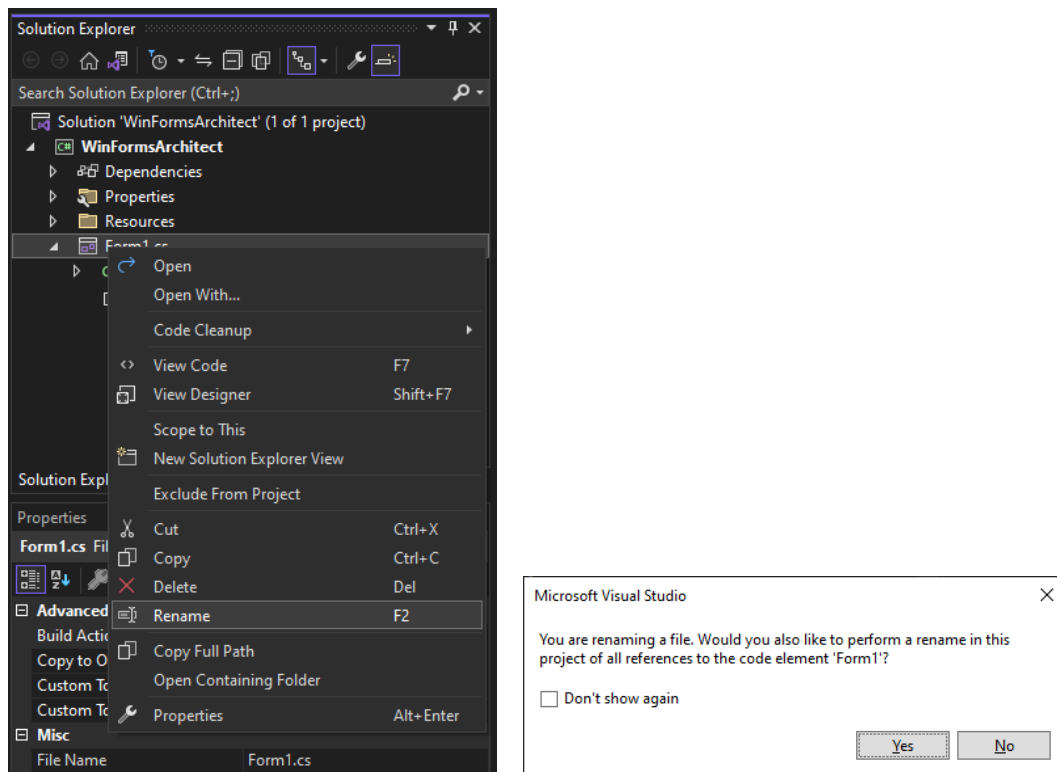
- "Main window" section - 2 points
- "New Blueprint" section - 2 points
- "Add furnitures groupbox" section - 2 points
- "Buttons" section - 2 points
- "Adding new furnitures" and "Created furniture groupbox" sections - 4 points

3 Solution

3.1 Creating a project

First, we create a Visual Studio project with Windows Forms template:

1. Choose File → New Project → Windows Forms App
2. Name it however you like. In Additional Information part set Framework to .NET Core or .NET 6/7.
3. In the created app you will have the main window already created (called "Form1"). This will be our app's main window. Rename it to MainWindow - in order to do this, select Form1.cs file in Solution Explorer, click RMB (Right Mouse Button) and choose Rename (or use F2 shortcut on Form1.cs file).



(a) Renaming file Form1.cs

(b) Renaming prompt

Figure 1: Rename main window class

Notice you will receive prompt asking to perform rename of all references to "Form1". Select "Yes", as it will change the name of the main window class, the main window file name, as well as the name of the resource file (.resx file) and *.Designer.cs file.

3.2 Main Window appearance

1. To add all the controls in this tutorial, we will use the Designer and the Toolbox.
 - (a) To open Designer (if it's not visible) click RMB on MainWindow.cs and choose View Designer (or use shortcut Shift+F7).
 - (b) To open the partial code of MainWindow class (the code you will actually use) - click RMB on MainWindow.cs and choose View Code (or use shortcut F7).
 - (c) To open the Toolbox tab (if it's not visible) choose in Visual Studio menu View → Toolbox (the default shortcut for this is Ctrl+Alt+X).
2. Set the window's title bar text to a more appropriate one. In order to do this, click on the form in the Designer, go to its Properties and change it in **Appearance** → **Text** property.
3. The window should start in the center of the screen. In order to meet this condition change window property **Layout** → **StartPosition** to **CenterScreen**.
4. One of the main window requirements is to set its Minimum Size. To do this, change property **Layout** → **MinimumSize** to **400x300**.
5. To add any of the controls from the Toolbar to the window, just drag and drop chosen control onto the window opened in Designer mode:

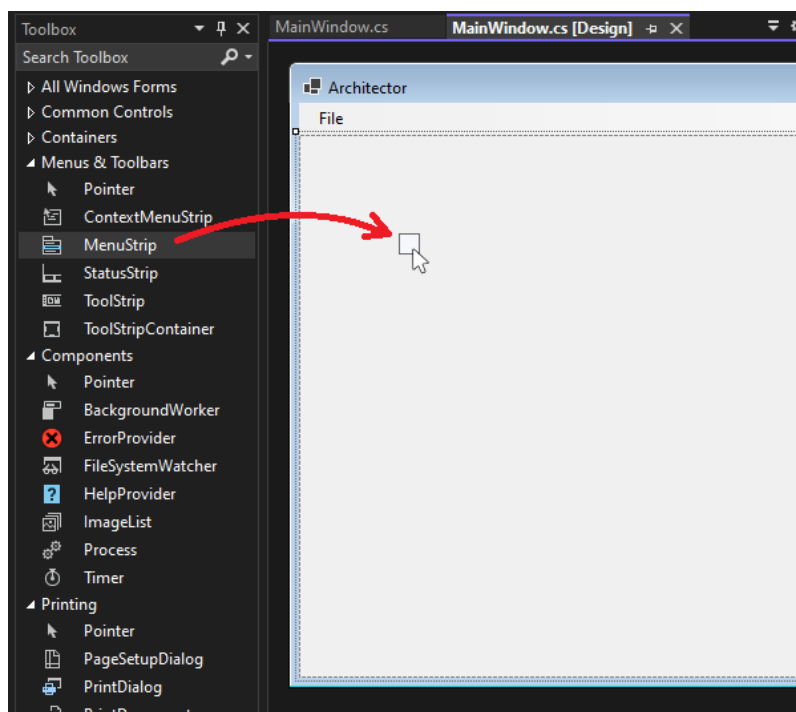


Figure 2: Drag and Drop controls

6. Setup the window layout:

- (a) Add MenuStrip to the window. Add Item "File", and under it add "New Blueprint" subitem.
- (b) Add SplitContainer by dragging it onto the window.
- (c) Right panel should contain two group boxes that resize according to the window size - the height of both of them should be half the height of the right panel of SplitContainer. To ensure the height of GroupBoxes is correct add TableLayoutPanel with only one column and two rows. Set **Layout** → **Dock** to **Fill**. Ensure that both rows have SizeType set to Percent and Value set to 50% (**Layout** → **Rows**). Add one GroupBox to the top row and one to the bottom row. Change their names to appropriate ones by setting property **Appearance** → **Text** and **Layout** → **Dock** to **Fill**.
- (d) Add PictureBox to the left panel of the SplitContainer. Set **Layout** → **Dock** to **Fill**. Change property **Design** → **Name** to Canvas. You can refer to the control by this property, so remember to name your controls in a way that can relate to their responsibilities.

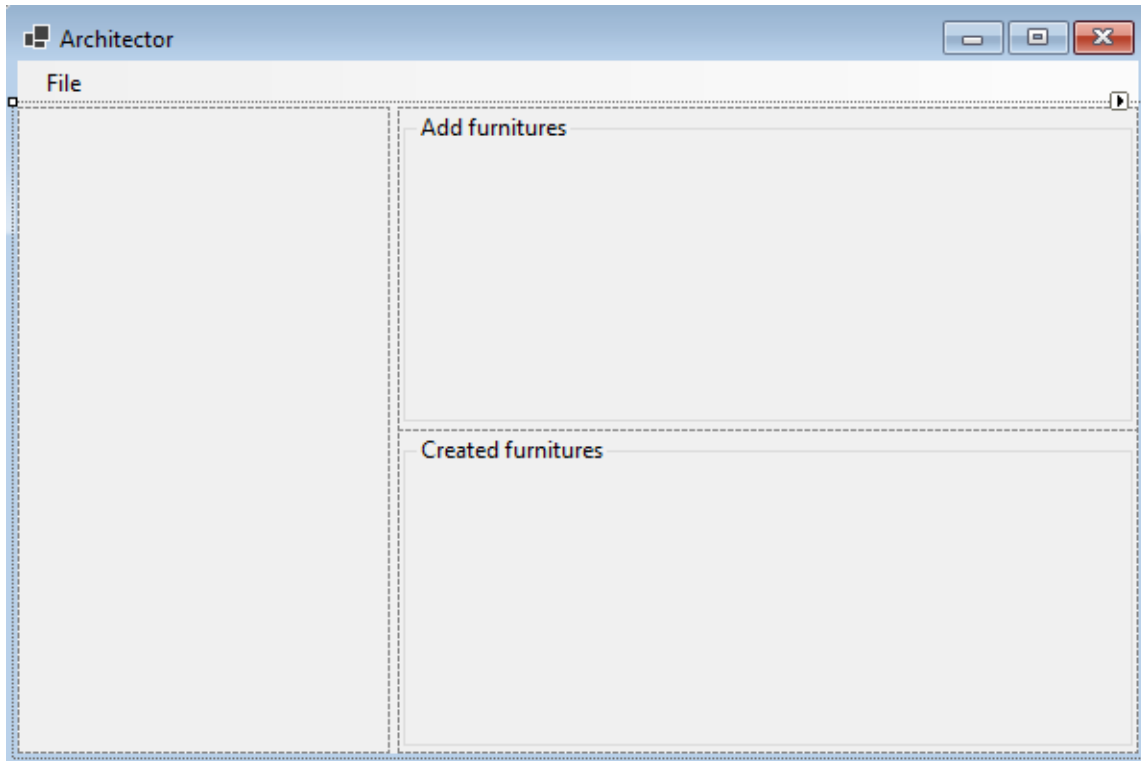


Figure 3: Complete Main Window appearance

3.3 Binding Bitmap to PictureBox

1. The last thing to complete the Main Window requirements is to setup a bitmap that will be bound with PictureBox. In order to do this create class Blueprint that will be responsible for managing the bitmap and the elements we emplace on it. Add a property of type Bitmap to it.

```
1 public class Blueprint
2 {
3     public Bitmap Bitmap { get; private set; }
4 }
```

2. Add the constructor with two parameters: width and height.

```
1 public Blueprint(int width, int height)
2 {
3     Bitmap = new Bitmap(width, height);
4 }
```

3. Add object of Blueprint type to MainWindow. Add private property and initialize it in new function **Initialize()**. Create Blueprint object in it, passing the width and height of the PictureBox and assign its Bitmap property to Canvas.Image:

```
1 public partial class MainWindow : Form
2 {
3     private Blueprint blueprint;
4     public MainWindow()
5     {
6         InitializeComponent();
7         Initialize();
8     }
9
10    private void Initialize()
11    {
12        blueprint = new Blueprint(Canvas.Width, Canvas.Height);
13        Canvas.Image = blueprint.Bitmap;
14    }
15 }
```

4. The bitmap should still not be visible when you run your application because every pixel of the created bitmap has the alpha value set to 0. Change it by adding private field backgroundColor to the Blueprint class and setting it to the chosen color. Then create public function **Draw()** responsible for clearing bitmap and redrawing it with all the content it will contain. Call the function in the constructor of the Blueprint class.

```

1  public class Blueprint
2  {
3      ...
4      private Color backgroundColor = Color.White;
5
6      ...
7      public Blueprint(int width, int height)
8      {
9          ...
10         Draw();
11     }
12     public void Draw()
13     {
14         using (Graphics g = Graphics.FromImage(Bitmap))
15         {
16             g.Clear(backgroundColor);
17         }
18     }
19 }

```

This ends the first part of the task - creating a Main Window with a given appearance and binding the bitmap to the specific control responsible for showing it.

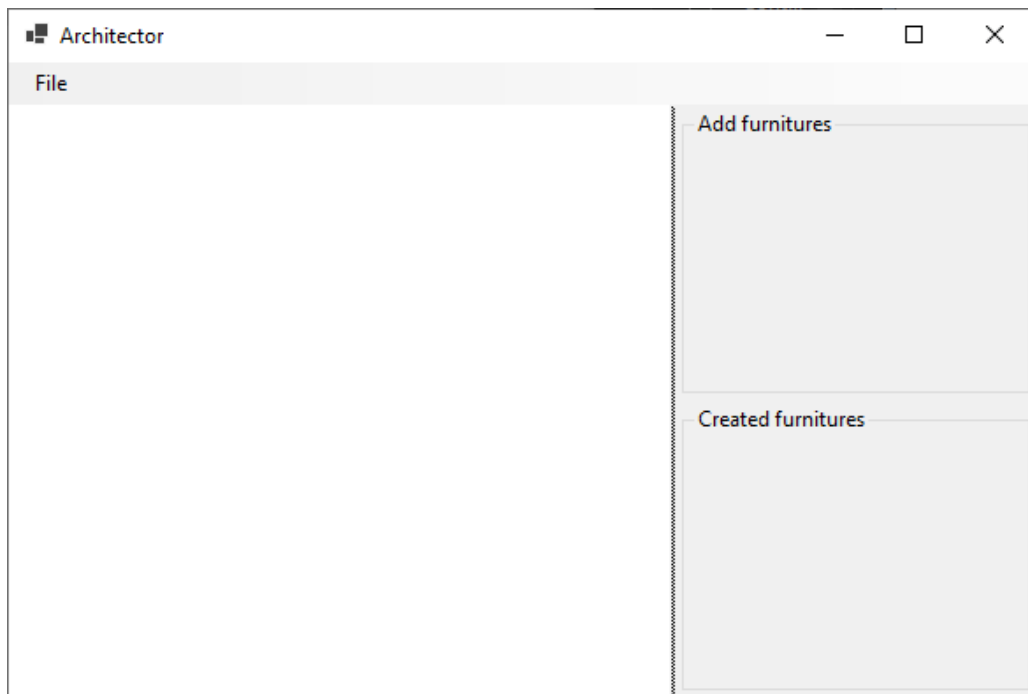


Figure 4: Bound Bitmap to PictureBox

3.4 New Blueprint functionality

1. Add shortcut to the New Blueprint menu item by selecting it and changing property **Misc** → **ShortcutKeys** to **F2**. Ensure you have **Misc** → **ShowShortcutKeys** set to **True**.
2. With the current implementation, you only have to call function `Initialize()` when the New Blueprint menu item is chosen. To do that, add event handler to **Action** → **Click** Event. In function handling this event call `Initialize()` function:

```
1 public partial class MainWindow : Form
2 {
3     ...
4     private void newBlueprintToolStripMenuItem_Click(object sender,
5         EventArgs e)
6     {
7         Initialize();
8     }
9 }
```

3.5 Buttons

1. First, add resources to your project. Open file **Properties** → **Resources.resx** and choose **AddResource** → **Add Existing File**. Change the filter of files you want to add to Bitmaps and find the folder with images attached to the task. You should be able to select all of them and press Add. All items should be right now copied to the folder with your project.

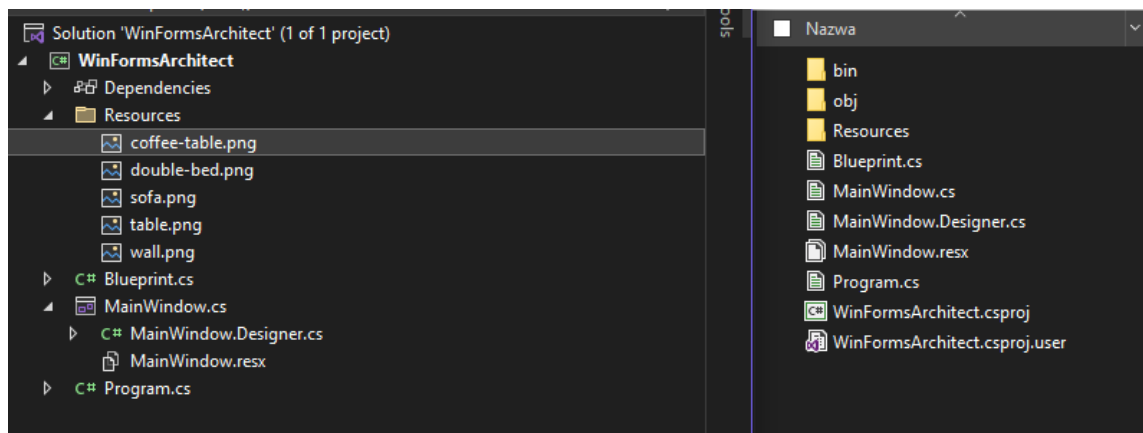


Figure 5: Added Resources to the project

You should inspect that the paths in the project are **relative** to the project. **This is crucial because the projects with absolute paths will not be graded during laboratories!** That means,

- (a) in the properties of the image files property **FullPath** should be pointing to the path inside the folder with your project.

E.g. path to project: C:/Users/SuperUser/MyWindowsFormsProject →
path to image C:/Users/SuperUser/MyWindowsFormsProject/Resources/image.png

- (b) in the Resources.resx file, when you open it with text editor as Notepad or VS Code, the path to the image should begin with "." or ".."

E.g.

```
...  
<data name="table" type="System.Resources.ResXFileRef, System  
  .Windows.Forms">  
<value>..\Resources\table.png;System.Drawing.Bitmap, System.  
  Drawing, Version=4.0.0.0, Culture=neutral, PublicKeyToken  
  =b03f5f7f11d50a3a</value>  
</data>  
...
```

2. Add FlowLayoutPanel to "Add Furnitures" GroupBox. Set property **Layout** → **Dock** to **Fill**. This will hold our buttons in a way that we won't have to calculate their positions when the main window is resized.
3. Add 4 buttons to the FlowLayoutPanel, as there are 4 images with furniture attached to the task. Set their size (**Layout** → **Size**) to **75x75**. Rename them appropriately so that their name will suggest what kind of furniture you will add with this button (e.g. buttonCoffeeTable, buttonTable).
4. Now let's set the buttons' images. For each button click **Appearance** → **BackgroundImage** property. When clicked, it should open a window with the import settings. Under **Project resource file** → **Properties/Resources.resx** choose the appropriate image file.

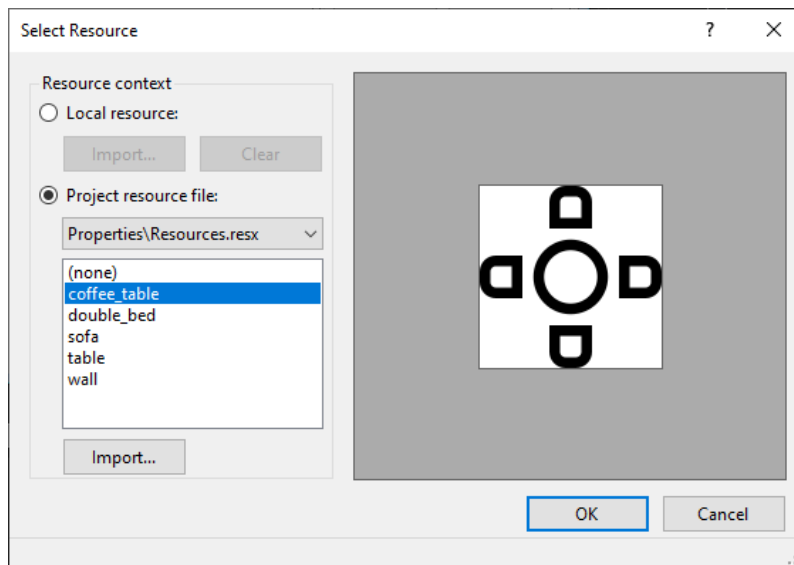


Figure 6: Window with import options when choosing button background

The buttons still appear slightly wrong as the image on them is too big for the button itself. To fix this, for each button change **Appearance** → **BackgroundImageLayout** to **Stretch**. Also, remove the text on the buttons (**Appearance** → **Text**) and set the **Appearance** → **BackColor** to **Web** → **White** (as this is the requested normal state for our button).

5. The last part to setup is the proper behavior of the buttons upon being pressed. We will create a function that will be assigned to each button's Event **Action** → **Click**. Inside your `MainWindow` class create function **private void buttonFurniture_Click(object sender, EventArgs e)**. Add private **Button selectedButton** field to the `MainWindow` class and set its initial value to `null`. It will hold the information of the currently chosen button from our panel. Add the implementation of `buttonFurniture_Click` function:

```

1  public partial class Form1 : Form
2  {
3      ...
4      private Button selectedButton = null;
5
6      ...
7      private void buttonFurniture_Click(object sender, EventArgs e)
8      {
9          var btn = sender as Button;
10         if (selectedButton == null)
11         {
12             btn.BackColor = Color.AntiqueWhite;
13             selectedButton = btn;
14         }
15         else if (selectedButton == btn)
16         {
17             btn.BackColor = Color.White;
18             selectedButton = null;
19         }
20         else
21         {
22             btn.BackColor = Color.AntiqueWhite;
23             selectedButton.BackColor = Color.White;
24             selectedButton = btn;
25         }
26     }
27     ...
28 }

```

In the function above parameter *object sender* contains the information which control has sent this event. *EventArgs e* parameter serves to send the additional event data that occurred during invoking the event. `EventArgs` is the base class for classes that contain the event data, so the functions handling specific events will have typically this parameter as `EventArgs` derived class (e.g. for mouse or keyboard events accordingly `MouseEventArgs` or `KeyEventArgs`).

6. Assign this function to all the buttons' Click Event, as all of the buttons should work the same way. In Designer window, for each button set Event **Action** → **Click** to **buttonFur-**

niture_Click.

3.6 Adding furniture images to bitmap

1. First, add new class Furniture representing furniture objects on the bitmap. In the beginning, it will hold created furniture object image and its position on the canvas.

```
1 public class Furniture
2 {
3     private Point position;
4     private Image icon;
5
6     public Furniture(Point position, Image icon)
7     {
8         this.position = position;
9         this.icon = icon;
10    }
11 }
```

2. Add to the Blueprint class list of objects of Furniture class to hold information of created objects on the canvas. Add function AddElement to the Blueprint class to be able to add Furniture objects outside this class:

```
1 public class Blueprint
2 {
3     ...
4     public List<Furniture> FURNITURES { get; }
5
6     ...
7     public Blueprint(int width, int height)
8     {
9         Bitmap = new Bitmap(width, height);
10        FURNITURES = new BindingList<Furniture>();
11        Draw();
12    }
13    public void AddFurniture(Furniture furniture)
14    {
15        FURNITURES.Add(furniture);
16    }
17    ...
18 }
```

3. Now let's bind AddFurniture function with mouse down event on the canvas. Go to Designer and choose Canvas. Add event handler to **Mouse** → **MouseDown** Event. Name the function **Canvas_MouseDown**. When one of the buttons is chosen (that is, if selectedButton is not null), we should create Furniture object in the location of the mouse click and deselect the chosen button. Such behavior is achieved by sample implementation below:

```

1 private void Canvas_MouseDown(object sender, MouseEventArgs e)
2 {
3     if (e.Button == MouseButton.Left)
4     {
5         if (selectedButton != null)
6         {
7             blueprint.AddFurniture(new Furniture(e.Location,
8                 selectedButton.BackgroundImage));
9             selectedButton.BackColor = Color.White;
10            selectedButton = null;
11        }
12    }
13 }

```

4. Still, when running the application, the furniture images are not visible on the canvas. To make it visible, you should add function **Draw(...)** to the Furniture class and call it in Draw function of the Blueprint class. Add Draw function to the Furniture class:

```

1 public class Furniture
2 {
3     ...
4     public void Draw(Graphics g)
5     {
6         var center = new Point(position.X - icon.Size.Width / 2,
7             position.Y - icon.Size.Height / 2);
8         g.DrawImage(icon, center);
9     }
10 }

```

Now call Furniture's Draw(...) function in Draw() method of the Blueprint class:

```

1 public class Blueprint
2 {
3     ...
4     public void Draw()
5     {
6         using (Graphics g = Graphics.FromImage(Bitmap))
7         {
8             g.Clear(backgroundColor);
9             foreach (Furniture furniture in FURNITURES)
10                furniture.Draw(g);
11        }
12    }
13 }

```

5. Last but not least, call the Blueprint's Draw function upon clicking the mouse in MainWindow:

```

1  public partial class MainWindow : Form
2  {
3      ...
4      private void Canvas_MouseDown(object sender, MouseEventArgs e)
5      {
6          if (e.Button == MouseButtons.Left)
7          {
8              if (selectedButton != null)
9              {
10                 blueprint.AddFurniture(new Furniture(e.Location,
11                 selectedButton.BackgroundImage));
12                 selectedButton.BackColor = Color.White;
13                 selectedButton = null;
14                 blueprint.Draw();
15                 Canvas.Refresh();
16             }
17         }
18     }
19 }

```

Remember to call Refresh function on the Canvas (PictureBox). It is essential because the control doesn't know the image set to the property has been changed.

3.7 Adding created furniture objects to the list

To add the list we will create a control known as ListBox.

The easiest way to add objects to the ListBox is to either add the entry manually to the list itself upon clicking on the canvas or to use a BindingList. The first one may seem to be a bit quicker, however it has its drawbacks. With the first approach, the list in the ListBox will not be bound to the created objects in any way.

Imagine we have the limit of furnitures on the bitmap, and you want to remove the oldest object in the Blueprint class when you hit the limit. In this example, you would have to add the ListBox reference to the Blueprint class to remove this object also from the list. Or synchronize them in MainWindow, or use another way to synchronize the Furniture list in the Blueprint class and elements in the ListBox. Yuck.

Class BindingList serves to synchronize data in "Two Way" mode. In other words, if data would be added in the UI, it will be added to the list behind, and vice versa.

1. Add ListBox to the "Created Furniture" GroupBox. Rename it "furnitureListBox". Set **Layout** → **Dock** to Fill.
2. Change the property Furnitures of the Blueprint class from type **List<Furniture>** to **BindingList<Furniture>**. BindingList<> class implements IBindingList interface which supports two-way databinding. In other words, binding the list of BindingList<> class to the DataSource of some control will cause the control to update its content when you add an object to the list.

Remember to change the initialization in the constructor of the Blueprint class:

```

1  public class Blueprint
2  {
3      ...
4      public BindingList<Furniture> Furnitures { get; }
5
6      public Blueprint(int width, int height)
7      {
8          Bitmap = new Bitmap(width, height);
9          Furnitures = new BindingList<Furniture>();
10         Draw();
11     }
12     ...
13 }

```

3. Bind the ListBox DataSource to the Furniture list. Do this in Initialize() function of the MainWindow class:

```

1  public partial class MainWindow : Form
2  {
3      ...
4      private void Initialize()
5      {
6          blueprint = new Blueprint(Canvas.Width, Canvas.Height);
7          Canvas.Image = blueprint.Bitmap;
8          furnitureListBox.DataSource = blueprint.Furnitures;
9      }
10     ...
11 }

```

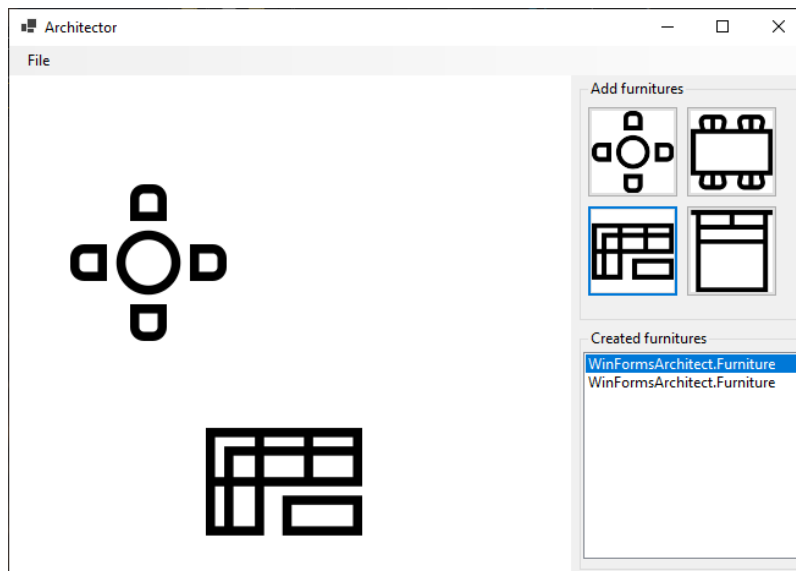


Figure 7: Binded ListBox to the Furniture list

When you run your code right now, you can see that when you add the furniture object on the canvas the ListBox is updated with new data. The new line displays your class name. Instead of the class name, it should display the name of the object and its position. To fix this, bound objects should override their ToString() method.

4. Override method ToString() in the Furniture class:

```
1 public class Furniture
2 {
3     ...
4     public override string ToString()
5     {
6         return position.ToString();
7     }
8     ...
9 }
```

Now the ListBox shows the position of the created objects. The last thing to add is to display the object name in the ListBox entry.

5. Add private field holding the name of the object to the Furniture class. Add a third parameter to the Furniture class constructor with the name of the object. Change the ToString() method so that it will use the name of the furniture:

```
1 public class Furniture
2 {
3     ...
4     private string name;
5
6     public Furniture(Point position, Image icon, string name)
7     {
8         this.position = position;
9         this.icon = icon;
10        this.name = name;
11    }
12    public override string ToString()
13    {
14        return name + " " + position.ToString();
15    }
16    ...
17 }
```

6. To set the proper name of the object upon creating it, the name of the object should be known at the moment of selecting the proper button. Of course, it can be done by creating a proper enum with parsing to the String type, but it also can be done by setting the Tag of the buttons in the MainWindow. In the Designer find the buttons' property **Data** → **Tag** and fill it with proper names (e.g. Coffee Table, Sofa).

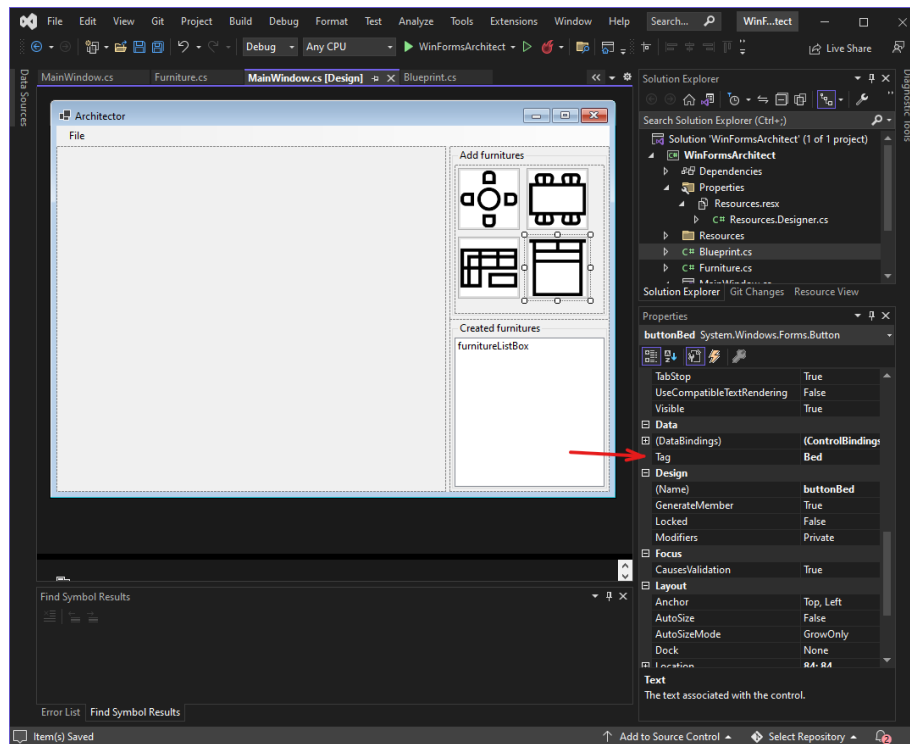


Figure 8: Setting buttons Tag properties

7. Now adapt creating furnitures in Canvas_MouseDown() function to the new constructor:

```

1 public partial class MainWindow : Form
2 {
3     ...
4     private void Canvas_MouseDown(object sender, MouseEventArgs e)
5     {
6         if (e.Button == MouseButtons.Left)
7         {
8             if (selectedButton != null)
9             {
10                blueprint.AddFurniture(new Furniture(e.Location,
11                selectedButton.BackgroundImage,
12                selectedButton.Tag.ToString()));
13            }
14        }
15    }
16 }
17 ...
18 }

```

This concludes the lab part of the sample task.

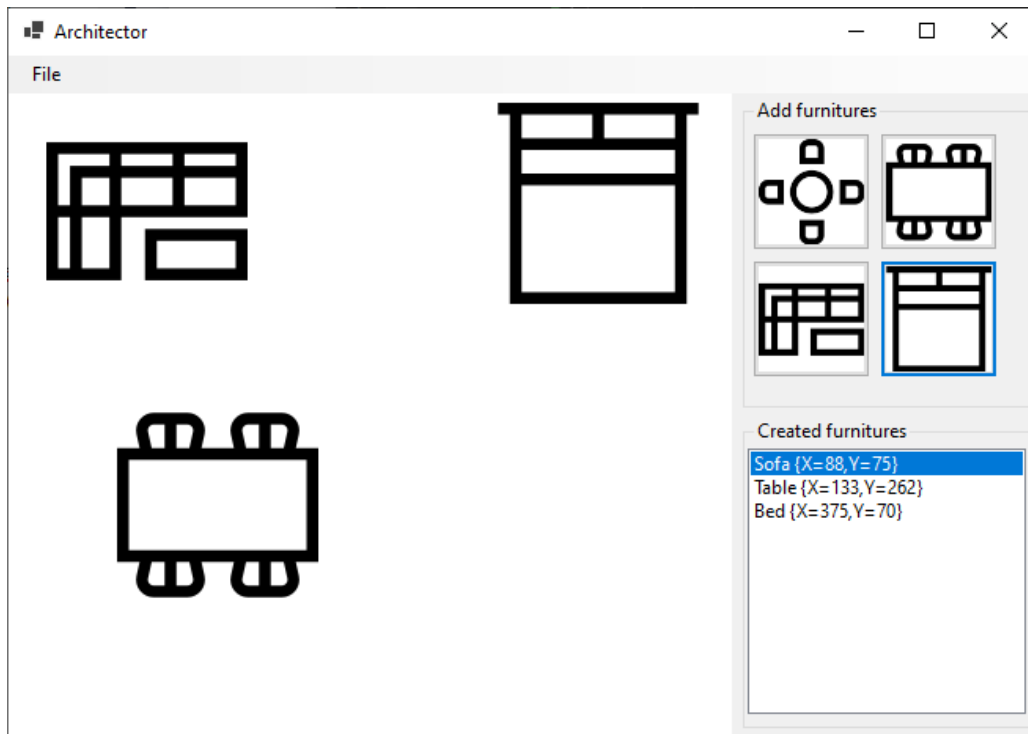


Figure 9: Completed Task

4 Home Task

Description

This section contains requirements for the home part of the task presented in Section 2. As usual, to pass the home part, all features from the lab part must be completed. In the task archive, you should be able to find **WinformsArch_Home.exe** file - an example app of the home part of the task.

The solution for the home part will not be part of this guide - it should be done at home as an exercise.

Requirements

- Bitmap resizing
 - When the main window of the application is increased, the bitmap should also change its size.
 - When the main window of the application is decreased, the bitmap should preserve its size and the scroll bars should become visible.

- The scrollbars should ignore the mouse wheel event.
- Drawing walls
 - The user can draw wall segments after selecting appropriate button in "Add Elements" groupBox.
 - After clicking with left mouse button a new segment is added to the wall.
 - Wall segments should have "smooth" connections - any holes and empty spaces are unacceptable.
 - Listbox should display the position of the first point added to wall segment.
 - After clicking right mouse button wall creation is stopped. Another way to stop building the wall is deselecting the wall button.
- Selecting elements
 - The user can select an element by clicking with left mouse button on it.
 - ListBox's selection should be updated.
 - When an element is selected it is drawn with alpha set to 50% (it's semi-transparent)
- Moving elements
 - The user can move an element by clicking on it with left mouse button and moving the cursor (with left mouse button still pressed).
 - ListBox must update furniture's actual position.
 - When an element is moving it is also selected.
- Deleting elements
 - The user can delete an element by selecting it and clicking Delete on the keyboard.
 - ListBox must be updated.
- Rotating elements
 - The application must support rotating the elements (furnitures and wall segments).
 - Furnitures rotate around its centers and wall segments around the first point added to the segment).
 - To rotate an element, the user has to select the element, and scroll using the mouse wheel.
- Open and Save
 - The application must support saving and loading created blueprints.
 - After clicking "Save blueprint" from the menu, the appropriate dialog box opens. It allows to enter a name of the file, but it should force you to save it with the extension (you can define any extension, e.g. *.bp).
 - All necessary information about the blueprint should be saved. After loading a file, the bitmap (and listBox) should look the same as in the moment of saving.

- After clicking "Open blueprint" from the menu, the appropriate dialog box opens. It should force you to open files only with your defined extension (e.g. *.bp).
- The user must be informed about the result of open or save operations.
- Localization
 - The application should support two languages (English and any other).
 - The application's default language is English.
 - Language can be changed using Menu (File → Language). Each text in the application should be changed to an equivalent in a different language.
Note: this should be done using a localization mechanism - changing the Text / Title property will not be scored
 - After clicking "Open blueprint" from the menu, the appropriate dialog box opens. It should force you to open files only with your defined extension (e.g. *.bp).
 - After changing the localization all controls should be reloaded correctly.
 - The main window should remain in the same position and be the same size.
 - Remember about the messages and texts in open/save file dialogs and listbox.

Hints

- MouseEventArgs, MouseEventArgs
- Form.Localizable, CultureInfo.CurrentCulture, ResourceManager
- OpenFileDialog, SaveFileDialog
- DrawImage, ImageAttributes, SetColorMatrix, TranslateTransform, RotateTransform
- GraphicsPath, Matrix.RotateAt, GraphicsPath.IsOutlineVisible
- HandledMouseEventArgs, BindingList

Grading

- "Bitmap resizing" section - 1 point
- "Drawing walls" section - 2 points
- "Selecting elements" section - 1 point
- "Moving elements" section - 1 point
- "Deleting elements" section - 1 point
- "Rotating elements" section - 2 points
- "Open and Save" section - 2 points
- "Localization" section - 2 points