

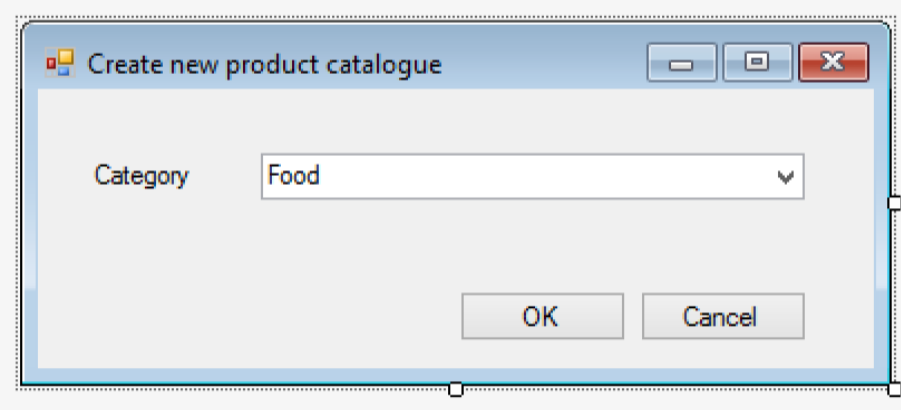
# Screen mockups for “Product Inventory” app

This document gives some example form mockups, which shall assist you in designing the GUI for the product inventory application. Notice that these screens are suggestions only and represent the simple-most way to implement the application. You are free to deviate from the design (eg, use different controls, layout controls differently, etc.). In order to earn full marks, you are required to have **different forms** for the following features:

- Creating a new product catalogue
- Adding a new product into a catalogue
- The main form, which shows the currently existing catalogues and their products and from where you can invoke all other features

## Create new catalogue form

This form allows the user to create a new product catalogue by selecting a product category from an **enum** in your program. You are free to (optionally) have additional properties in your **ProductCatalogue** class.

A screenshot of a Windows-style dialog box titled "Create new product catalogue". The dialog has a standard title bar with minimize, maximize, and close buttons. Inside the dialog, there is a label "Category" followed by a text box containing the word "Food" and a small downward arrow indicating a dropdown menu. At the bottom of the dialog, there are two buttons: "OK" and "Cancel".

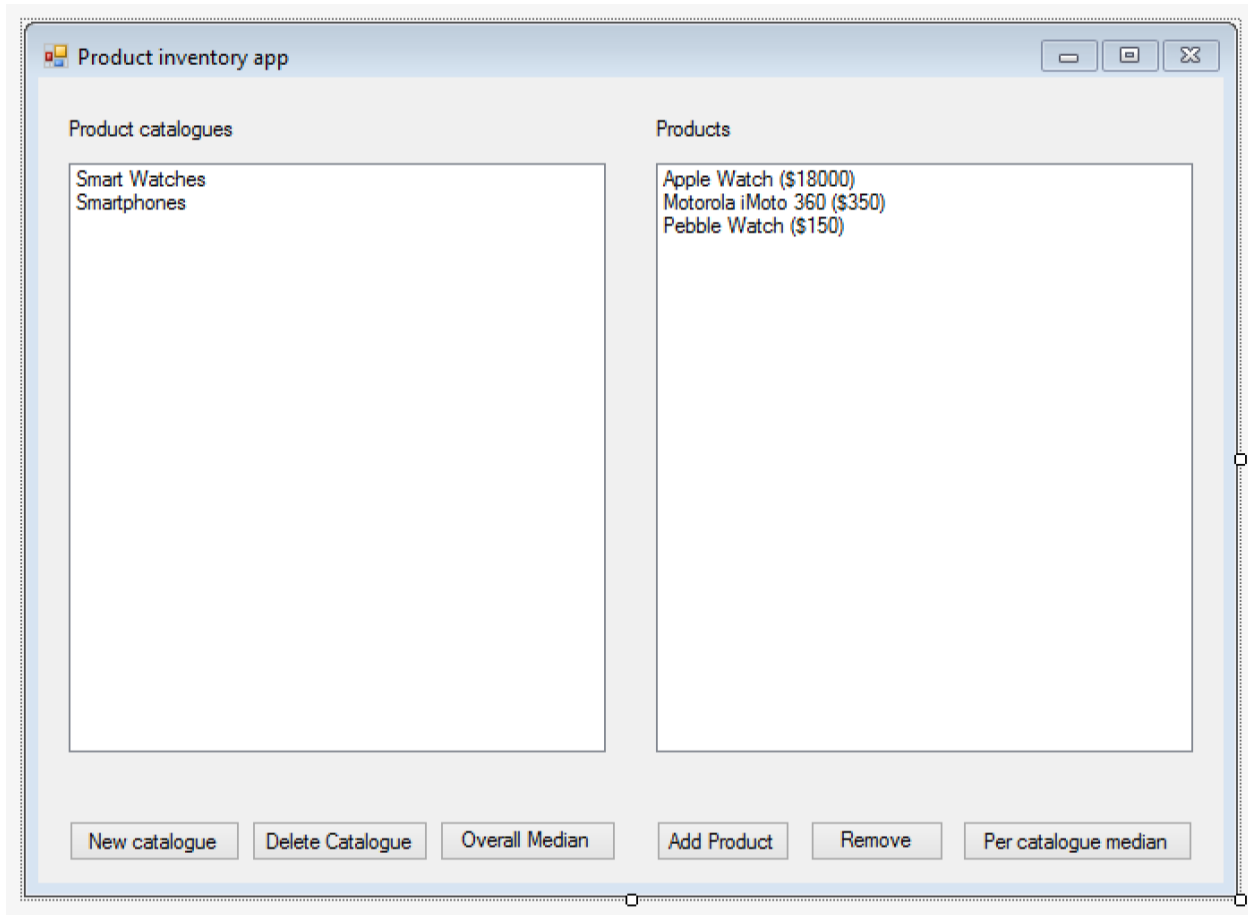
## Add new product form

This form allows the user to add a new product into an existing product catalogue by selecting the product catalogue and entering the product name and price. You are free to omit the product catalogue selection if an affected product catalogue was already selected on the main screen.

The image shows a standard Windows application dialog box. The title bar at the top reads 'Add new product' and includes standard minimize, maximize, and close buttons. The main area of the dialog contains three labeled input fields. The first, 'Catalogue', is a dropdown menu currently displaying 'Food (9 products)'. The second, 'Product name', is a text box containing 'Apple Watch'. The third, 'Product price', is a text box containing '18000'. At the bottom center of the dialog are two buttons: 'OK' and 'Cancel'.

## Main form

Finally, the main form is the application's entry point that is always visible during the runtime of your application. All other features are started by having the user interact with controls on this form. This example mockup uses a two-column design, where the product catalogues are shown in a **ListBox** on the left side and the products within the currently selected product catalogue are shown in another **ListBox** on the right side. All other actions can be invoked by clicking on the buttons below the two **ListBoxes**.

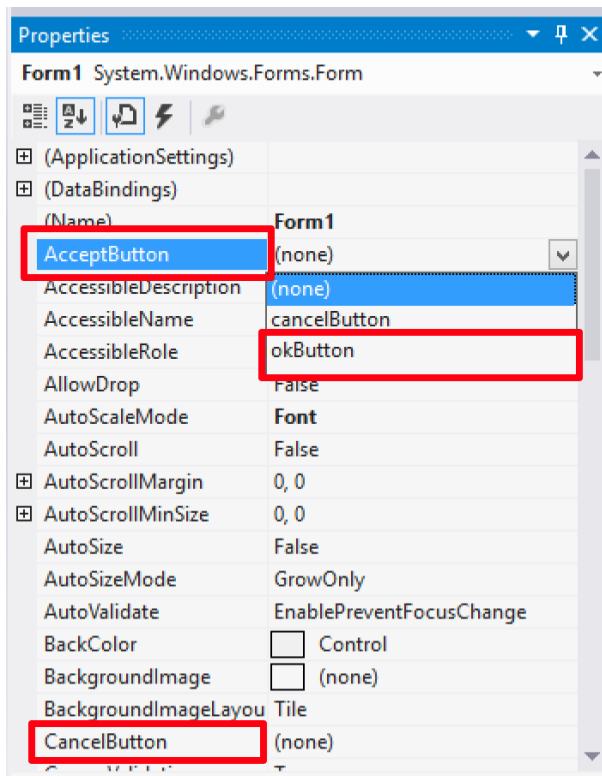


## Further hints

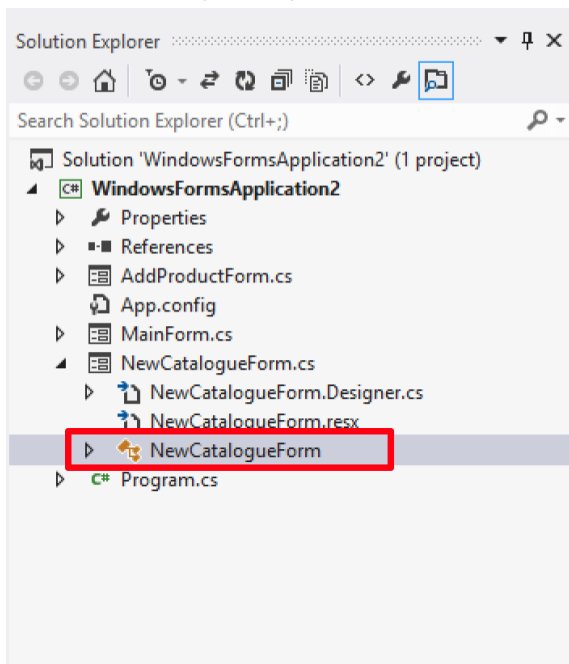
When creating the former two forms (for creating a catalogue and adding a product), use the property sheet in Visual Studio to set the *AcceptButton* and *CancelButton* properties to the corresponding buttons:

- First set the *Name* properties of the two buttons to some meaningful identifiers, like *okButton* and *cancelButton*.

- The set the form properties *AcceptButton* and *CancelButton* to the matching buttons:



- Then go to the source code of your form. To do so, double-click the matching symbol in the *Solution Explorer* panel:



- In the form class, you should then add a public method, which returns (or uses *out* parameters) to pass back the user input/selections (by retrieving the relevant properties of the controls on the form). For instance, in order to return a variable of the

enum type *ProductCatalogue*, the following method first accesses the *string* property *SelectedText* of the *ComboBox* property *categorySelectionBox* on the form. The static method *Enum.Parse* then takes that string alongside *typeof(ProductCategory)* and returns an enum variable, which still needs to be “cast” into the enum *ProductCategory*, before it can be returned:

```
public ProductCategory GetCatalogueCategory()
{
    return (ProductCategory)Enum.Parse(typeof(ProductCategory),
categorySelectionBox.SelectedText);
}
```

- In the main form, we want to invoke another form in a way that we suspend other user interactions with the other form is shown and depending on the “outcome” of the other form (ie, whether the user has clicked on “OK” or “Cancel”) continue processing the data from the other form. To do so, you need to add the following code to the event handler, where you open the new form:

```
NewCatalogueForm form = new NewCatalogueForm();
if (form.ShowDialog() == DialogResult.OK)
{
    ProductCategory category = form.GetCatalogueCategory();
}
```