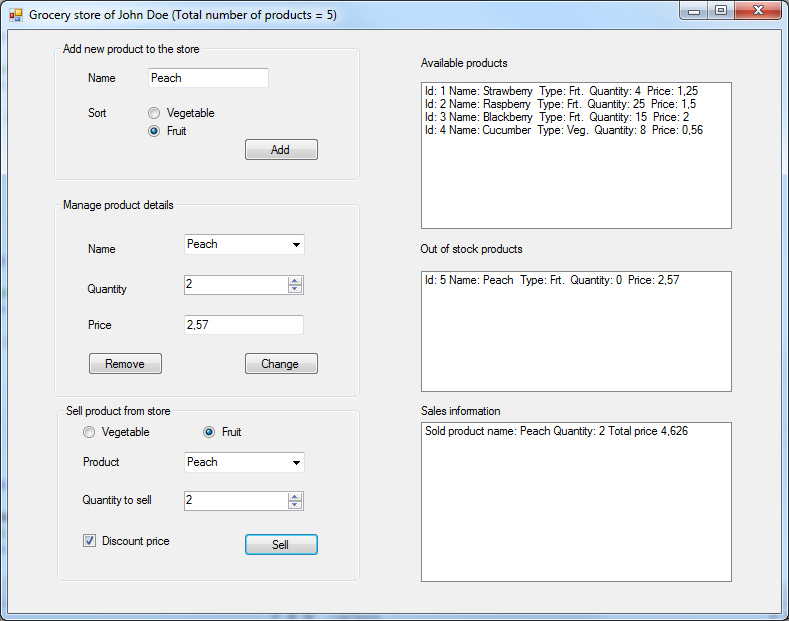
# Exam PCS2 – Grocery Store June 2015

You are asked to develop an application for managing your own grocery store.



Now open the startup-project either in Visual Studio 2012 or Visual Studio 2013 (it is mandatory to use Visual Studio 2012 or Visual Studio 2013 in this exam). You can see that there is one form with a lot of controls on it (see the picture above). During this exam, you will add two classes: class Product and class ProductStore. Of course, you will also add some code to the Form as well.

Exam points:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Assignment | 1 | 2 | 3 | 4 | 5 | 6 | 7 | Bonus | Total |
| Points | 20 | 10 | 10 | 10 | 10 | 10 | 20 | 10 | 100 |

**Important note about naming -** Whenever this exam paper suggests to create a class, variable, property or method with a certain name (e.g., property called Title), you are required to indeed use that name (e.g., Title) in your C# program.

A**ssignment 1: Class Product (10 + 5 + 5 pts)**

Add a class called Product to the project (in a separate file). An object of this class will represent one product in the grocery store (e.g., strawberry, lemon, cucumber …). Implement this class by doing the following:

1. Add all properties listed below. Wherever possible make an auto-implemented property. If it is not possible to make an auto-implemented property, you can define a private instance variable with a custom property (where you implement set and get accessors).
   * Property called **Name** should store the name of the product (as a string). It should not be possible to change the value from outside the Product class (so e.g. from the GroceryForm class).
   * Property called **Type** should store the information of the product type. There are only three possible product type values: fruit, vegetable, and unknown. It should not be possible to change the value from outside the Product class (so e.g. from the GroceryForm class).
     + Because there are only three possible product types, you should make a special ProductType enumeration type with values FRUIT, VEGETABLE, and UNKNOWN.
     + Property Type should then be of the type ProductType.
   * Property called **Quantity** should be the quantity of the product available in the store. The quantity should be of type int, but can never become negative. (I.e., the getter of the property has no restrictions, its setter must prevent that the quantity gets a value less than zero. Make the value equal to zero in that situation.)
   * Property called **Price** should store the price of the product (as a double precision number) . It cannot store negative values. (I.e., the getter of the property has no restrictions, its setter must prevent that the price gets a value less than zero. Make the value equal to zero in that case).
   * Property called **ProdId** should store the product identification number (as an int). It should not be possible to change the value from outside the Product class. The product identification number should be a number that is unique for each product that is added to the store.
2. Add one **constructor** to the Product class. This constructor should have two parameters, called name and type, to initialize values of the properties Name and Type. The value of the property Quantity must always be initialized to 0 (zero). The value of the property Price must be initialized to 0.0. Finally, the property ProdId should be assigned a unique integer value (hint: within class Product, introduce a private instance variable noCreated of type int, that is available to all the objects of type Product. Use its value to assign an appropriate value to ProdId).
3. Add method **AsString()** which returns all information about the product as one string. Use a format similar to the one shown in the ListBox in the screen shot on the first page. Notice that in this string, the product type is represented as Frt. (for value FRUIT), Veg. (for value VEGETABLE), and Unk. (for value UNKNOWN).

**Assignment 2: Class ProductStore (3 + 3 + 2 + 2 pts)**

Add a class called ProductStore to the project (in a separate file). An object of this class will represent a list of products (called the product assortiment) in the grocery store.

1. Add the following properties/fields:

* Property called **Owner** should store the full name of the person owning the grocery store (as a string). If possible make Owner an auto-implemented property. If it is not possible to make an auto-implemented property, you can define a private instance variable with a custom property (where you implement set and get accessors).
* Add a private instance variable called **productAssortiment**, which should be a list (not array) for storing all products in the grocery store. This field (instance variable) must be inaccessible for other classes!

1. Add one **constructor** that initializes all properties and fields. This constructor should have one parameter to initialize the value of property Owner. The list of products (productAssortiment) should not contain any product at the start.
2. Add method **GetAllProducts** that returns a list containing all products in the productAssortiment.
3. Add method **GetNoOfProducts** that returns how many products there are in the productAssortiment.

**The following assignments are about the form and classes Product and ProductStore.**

All controls (buttons, labels) you need are already in the form. You don’t need to add any new controls during this exam.

A**ssignment 3: At startup (2 + 2 + 1 + 5 pts)**

1. Add one private field called **productStore** of the type ProductStore to the form. Make sure it is created and initialized on startup with your name.
2. Add a private method called **UpdateGroceryStoreText** that sets the new title on the GroceryForm like in the screenshot. The information from the productStore (your name, and the total number of products) must be used in this title. The method should not return a value.
3. Call method UpdateGroceryStoreText once when the GroceryForm is created in order to show the correct information in the title of the form.
4. Add a private method called **UpdateListBoxes** that shows all products in the productStore in two list boxes lbAvailableProducts and lbOutOfStockProducts by doing the following:
   * Get all products from the productStore using the method GetAllProducts.
   * Clear the list boxes lbAvailableProducts and lbOutOfStockProducts and fill it with information about all products.
   * Add information about each product in a new line in one of the list boxes (add the result of method AsString() of the product, so that the information is displayed clearly in the list box).
   * If a product is out of stock (i.e. the quantity of the product is zero) then the information of that product should be added to the lbOutOfStockProducts, if the product is available (i.e. the quantity of the product is larger than zero), then the information of that product should be added to the lbAvailableProducts.

**Assignment 4: Add a product (2 + 4 + 4 pts)**

Add functionality to add a new product:

1. A product will only be added to the productStore, if it is not yet in the productAssortiment of that productStore. A private method **InAssortiment** with one parameter productName of type string will be used to check whether a product with name productName is already in the productAssortiment of the productStore. The method will return true if the product is in the productAssortiment, and false otherwise. Implement the method InAssortiment (in class **ProductStore**).
2. Add method **AddProduct** to the **ProductStore** class, which will be used to add a new product to the productAssortiment in the product store (provided that the product is not yet in the productAssortiment of the product store). The method should use two parameters to indicate the required information about the product to be added: productName and type. Use the method InAssortiment to check whether a product with productName is already in the productAssortiment. If the product is not yet in the productAssortiment, a new Product object should be created and added to the productAssortiment, and the method should return true. If the product is already in the productAssortiment, the method should return false.
3. Add functionality to the **“Add”-button**. Here you should do the following:

* Try to add a new product to the productStore object. The first step is to validate the input of tbName. If no value has been entered in tbName, a message box should be displayed that asks the user to enter a product name. If tbName contains a value, then try to add a product using the method AddProduct you just added to the ProductStore class and the values from tbName, rbFruit and rbVegetable.
* If the product was successfully added (i.e. method AddProduct has returned true), then update the list boxes, update the grocery store text, and add the product name to the items in the combobox cbProduct. Use the methods UpdateListBoxes and UpdateGroceryStoreText implemented before.
* If the product was not successfully added (i.e. method AddProduct has returned false) then a message box should be displayed telling the user that the product is already in the productStore.

**Assignment 5: Managing product details (5 + 5 pts)**

Add functionality to manage product details:

1. Add method **ChangeProductDetails** to the **ProductStore** class, which should set the price and the quantity of one of the products in the productAssortiment in the productStore. This method should have three input parameters: a parameter productName of type string, a parameter quantity of type int, and a parameter price of type double. You may safely assume that a product with name productName is in the productAssortiment. For this product, the method should set the quantity and price, using the values of the corresponding input parameters. The method should not return a value.
2. Add functionality to the **“Change”-button**. After clicking this button, the quantity and price of the product that is currently selected in the cbProduct should be set. Use the values from nudQuantity and tbPrice to set the quantity and price. Use the method ChangeProductDetailsto change the information for the selected product. Finally, update the product information in the list boxes, using the method UpdateListBoxes.

Note: if no item is selected (or available) in the combo box cbProduct, the method should do nothing.

**Assignment 6: Removing a product (5 + 5 pts)**

Add functionality to remove a product from the list:

1. Add method **RemoveProduct** to the **ProductStore** class, which should remove the selected product from the productAssortiment in the productStore. This method has one input parameter: a parameter productName of type string, that indicates the product to be removed from the productAssortiment. Note: You may **NOT** assume that a product with name productName is in the productAssortiment. If found in the productAssortiment, the method should remove the product with name productName from the productAssortiment, otherwise it should not do anything. The method should not return a value.
2. Add functionality to the **“Remove”-button**. After clicking this button, the product that is currently selected in the cbProduct should be removed from the productAssortiment. Use the method RemoveProduct to remove the selected product. Next, the selected item should also be removed from the dropdown list of the combo box cbProduct. Finally, update the product information in the list boxes and on the form, using the method UpdateListBoxes and UpdateGroceryStoreText.

Note: if no item is selected (or available) in the combo box cbProduct, the method should do nothing.

**Assignment 7: Selecting the type of product to sell (2 + 2 + 2 + 2 + 2 + 3 + 3 + 4 pts)**

Once the grocery store contains products, they can be sold. In order to sell a product, the product should first be selected. This is done by first selecting the type of product (fruit or vegetable) to sell by checking one of the radio buttons rbSelectVegetable and rbSelectFruit . Once the product type is selected, all available products of the selected type are shown in the combo box. This functionality will be implemented in a number of steps:

First we will implement a number of methods in the Product class.

1. Add a public method called **InStock** to the **Product** class. This method has no parameters. The method returns true if product p is in stock (i.e. its quantity is larger than zero) and false otherwise.
2. Add a public method called **IsFruit** to the **Product** class. This method has no parameters. The method returns true if product p is of type FRUIT, and false otherwise.
3. Add a public method called **IsVegetable** to the **Product** class. This method has no parameters. The method returns true if product p is of type VEGETABLE, and false otherwise.

Then we implement some methods in the ProductStore class:

1. Add a public method **GetNoOfFruitsInStock** to the **ProductStore** class. This method has no parameters. It should return the number of products of type FRUIT that are in stock as an integer. Use the methods InStock and IsFruit implemented before.
2. Add a public method **GetNoOfVegetablesInStock** to the **ProductStore** class. This method has no parameters. It should return the number of products of type VEGETABLE that are in stock as an integer. Use the methods InStock and IsVegetable implemented before.
3. Add a public method **GetFruitsInStock** to the **ProductStore** class. This method has no parameters. It should return an array of strings, containing the names of the products of type FRUIT that are currently in stock. Use the methods InStock, IsFruit, and GetNoOfFruitsInStock to implement this method.
4. Add a public method **GetVegetablesInStock** to the **ProductStore** class. This method has no parameters. It should return an array of strings, containing the names of the products of type VEGETABLE that are currently in stock. Use the methods InStock, IsVegetable, and GetNoOfVegetablesInStock to implement this method.

Now, we implement the functionality in the GroceryForm to fill the combo box cbVegFruit when selecting a product type, by checking one of the radio buttons rbSelectVegetable or rbSelectFruit.

1. Add the code for the event handlers **rbSelectVegetable\_CheckedChanged** and **rbSelectFruit\_CheckedChanged** in the GroceryForm class.

When the rbSelectVegetable radio button is checked, the combo box cbVegFruit should contain a list with names of all vegetables that are currently in stock. In the implementation of the event handler **rbSelectVegetable\_CheckedChanged**, use the method GetVegetablesInStock implemented before.

When the rbSelectFruit radio button is checked, the combo box cbVegFruit should contain a list with names of all fruits that are currently in stock. In the implementation of the event handler **rbSelectFruit\_CheckedChanged**, use the method GetFruitsInStock implemented before

**BONUS ASSIGNMENT : Selling a product (2 + 1 + 3 + 3 + 1 pts)**

**(NOTE: This assignment is only required for those students that did NOT receive a confirmation from their teacher that they earned the 10 bonus points)**

We are now ready to implement the functionality when pressing the **Sell** button:

1. First, we implement a public method **SellProduct** in the **ProductStore** class. This method has two parameters: a parameter productName of type string, and a parameter quantityToSell of type int (that indicates the quantity to be sold of the product with name productName). The method SellProduct should return a boolean value.

Note that the quantityToSell of a selected product (indicated in the nudQuantityToSell) may exceed the quantity of that product that is currently available (indicated in the Quantity property of the product).

If the quantityToSell does exceed the currently available quantity of the product, then the method SellProduct should set the Quantity property of the product with name productName to zero, and the method should return false.

If the quantityToSell does **NOT** exceed the currently available quantity of the product, the method SellProduct should decrease the value of the Quantity property of the product with name productName by quantityToSell, and the method should return true.

1. Implement a public method **GetProductPrice** in the **ProductStore** class. This method has one input parameter: a parameter productName of type string. The method should return the price of the product with name productName in the list of products as a double.
2. After clicking the **Sell** button, the indicated quantity of the product that is currently selected in the cbVegFruit should be sold, if possible. Use the value from nudQuantityToSell to determine the requested quantity to sell. Use the method SellProductto sell the requested quantity of the selected product.

Check the result of invoking the SellProduct method. If SellProduct returned false, we should display a message box with a representative message. In this case all available items should be sold (i.e. the resulting Quantity of the product should be zero).

1. Next, add a line to the list box lbSalesInformation indicating the name of the product, the quantity that is sold, and the total price. (Note: in this assignment you may NOT assume that a sufficient quantity of the selected product is available in order to sell the requested quantity.)

The line that is added to the list box lbSalesInformation should indicate the actual product quantity that is sold. So, e.g. is quantityToSell = 5, but the available quantity of the product is only 3, we will sell 3 products, not 5.

In order to be able to determine the actually sold quantity of the product, we need to know the quantity that is available of the selected product. To retrieve that information, implement a public method **GetProductQuantity** in the ProductStore class. This method should have one input parameter productName of type string and should return the available quantity of the product with name productName in the productAssortiment.

Use the method GetProductPrice to determine the standard price for the selected product.

Make sure that in the calculation of the total price, the information of the checkbox cbxDiscount is taken into account. When this checkbox is unchecked, the selected product is sold at the original price. However, if the cbxDiscount checkbox is checked, the selected product should be sold at a 10% discount.

1. Finally, update the product information in the list boxes, using the method UpdateListBoxes.