Assignment 2: Webshop

A webshop is selling all kinds of products: books, hardware, hobby material, software and toys. The webshop provides a way for others to receive updates on its catalogue (new products, price change, out of stock).

Several parties (let's call then subscribers) are interested in this, there is a toy collector that wants to instantly know when new toys are for sale, and a bargain hunter that wants to see all the products that are cheaper than € 10,-.

The webshop should be able to view, add, update and remove its products, and the subscribers should be able to see these changes.

You are going to develop an application that has three forms. One form for the webshop, one form for the toy collector and one form for the bargain hunter.

In the image below you may find the GUI:

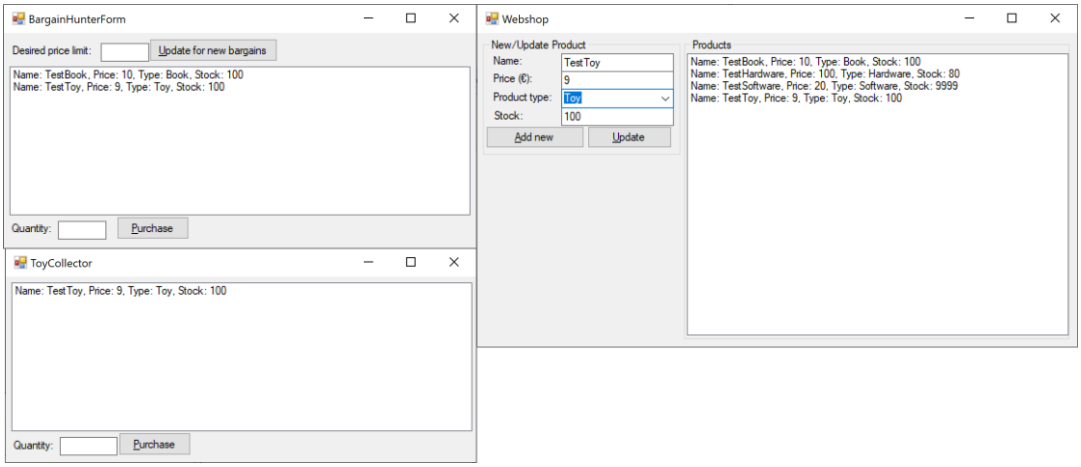

Figure 1: the provided GUIs

User Interaction:

We can make a distinction between three kinds of users for this system. There is the webshop administration (let's call them admin), the bargain hunter form user (let's call them hunters) and the toy collector form user (let's call them collectors).

Admin:

The admin will make use of the webshop (GUI on the right side) and can add and update products. By filling in the required information in the groupbox "New/Update Product" and clicking the button "Add new" a new product is added to the webshop and displayed accordingly in the listbox on the right. All products should have a unique name, so adding a product with a name that already exists should display a message with an appropriate warning. If the user wants to update a product, they can click on one in the listbox and all data should appear in the groupbox "New/Update Product". The user can then make changes to the product and click on the button "Update".

Collector:

The toy collector has its own form, whenever a new toy is added to the webshop the collector should see the toy in their form. If the collector wants to purchase this toy, they can click on the toy they are interested in, enter the quantity that they want to get in the textbox "Quantity:", and click on the button "Purchase". If they want to purchase more than the webshop has in stock, the purchase did not happen. An appropriate message should appear to inform the collector of this.

Hunter:

The bargain hunter interacts similarly with the application as the collector does, except they can also enter a price limit for new/updated products they want to see (existing products do not have to disappear).

Extra notes:

General:

- Make sure that all three forms open at startup.
- Use events and delegates in this assignment.

WebshopForm:

- If the user clicks on *btnAddNew*, a new product will be added to the webshop based on the values of the input fields above. The product will appear in *lbProducts*. The *BargainHunterForm* and *ToyCollectorForm* instances will be informed.
- If the user clicks on a product in *lbProducts*, the input fields above *btnUpdate* will contain their related information. After the user made the required changes and presses *btnUpdate*, the webshop will update the product. *lbProducts* will be updated accordingly. The *BargainHunterForm* and *ToyCollectorForm* instances will be informed.

ToyCollectorForm:

- Products appear in *lbToys* based on the information received from the webshop.
- If the user clicks on a product in *lbToys* the maximum amount of *nudQuantity* will change according to the stock of the selected product. The user can select a quantity to purchase and click on *btnPurchase* to purchase the products.

BargainHunterForm:

- The user can type an amount in *tbPriceLimit* and click on *btnUpdate* to change the highest price of products that will appear in *lbBargains*.
- Products appear in *lbBargains* based on the information received from the webshop.
- If the user clicks on a product in *lbBargains* the maximum amount of *nudQuantity* will change according to the stock of the selected product. The user can select a quantity to purchase and click on *btnPurchase* to purchase the products.

Steps:

Step 1:

Make a class diagram.

Identify the various entities/classes that should exist in this application.

- What are their responsibilities?
- Is SRP applied?
- Are your entities separated over the three layers?
- What data should they store?
- What should they be able to do?
- How do they relate to each other?
- Are all possible scenarios covered in your diagram?

Step 2:

Implement the application.

Download the application base from Canvas and implement your class diagram.

- Can you implement it without deviating from your class diagram?
- If not, stop implementation and change your class diagram before continuing.

There is a class diagram available on Canvas. *But for your own benefit, only use this as point of reference **after** finishing this exercise yourself (all steps)!*