PC Configurator App

Application description

We started implementing OOP application for a company that sells PC parts (Part), modules (Module) and configurations (Configuration). The application should allow creating different computer configurations consists of specific parts and modules. The complete list of all computer parts is defined in the following table:

#	Part	Price
1	HDD	30.00
2	HDD SSD	120.0
3	HDD data Cable	15.25
4	HDD power cable	5.25
5	HDD External Box	9.00
6	HDD usb cable	3.25
7	HDD connector	2.25
8	Camera external	15.00
9	HDMI cable	20.50
10	VGA cable	4.25
11	USB cable	3.25
12	Wifi Mouse	5.25
13	Wifi Keyboard	5.25
14	Desk mont part	15.25
15	Monitor	115.25
16	Cooler	7.25
17	RAM	45.25
18	Graphic card	86.25
19	CPU	95.25
20	Mother board	85.00
21	PC box	35.00
22	Wifi sound	19.00

Apart from the separate parts, the company provides users with modules and configurations. One module consists of multiple parts. For modules, the company offers an additional discount.

One configuration consists of multiple parts. Also, the configuration can be a combination of modules and parts. In the context of this, the company offers a discount for the configurations as well. But there is one limitation where the company does not provide a discount for individual parts.

Modules examples:

Module 1 - Full HDD set: HDD, HDD SSD, HDD data Cable, HDD power cable, HDD usb cable, HDD connector.

Module 2 - Peripherals: HDMI cable, Wifi Mouse, Wifi Keyboard, Cooler, Wifi sound.

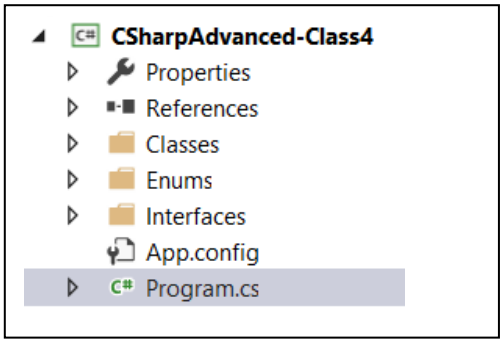
Configuration example:

Configuration 1: Module1, Module2, Monitor (x2), RAM, Graphic card, CPU, Mother board, PC box.

Technical implementation description

The program consists of three folders:

1. Enums
2. Interfaces
3. Classes
4. Program.cs



The program is partially solved.

The Parts objects from the table presented above are given/created in Program.cs. Then, you can also find the creation of two modules Module1 and Module2, and finally one configuration known as Configuration 1. All creations are placed in regions meaning that as you implement the specific functionalities the corresponding regions should become uncommented. Different calls (within the regions) are given to methods that should be first implemented in the order given below (Tasks) for use.

There is one abstract class defined within the initial code base.

```
public abstract class Item
    {
        public string Name { get; set; }
        public double Price { get; set; }
        public int Quantity { get; set; }
        public double Discount { get; set; }
    }
```

Classes Part, Module, and Configuration inherit the Item class.

The Part class implements the IPrice interface, while the Module and Configuration classes, besides IPrice, further implement the IDiscount interface. Consider all the interfaces and methods within the classes.

The Module class has the AddPartToModule() method that adds Part to a module. Consider this method. Also consider the constructors for this class. This class has private property _parts which stores the parts from which a single module is composed.

The Configuration class consists of methods AddPartToProduct() and AddModuleToProduct() that are not implemented. Consider also the constructors of this class. This class has private properties _parts and _modules that store the parts and modules from which a configuration is composed.

Tasks

Tasks should be implemented successively. Before you start, make an analysis if the code is well structured and organized.

Analysis: Review the program before you start coding. View classes with their partial implementations, interfaces, and enums.

Modules:

1. Implement GetPrice() and SetDiscount() methods in Module entity.
2. Uncomment regions “HDD module” and “Peripherals module” and ensure that the code is errors free and working as expected. (Note: commented lines within the regions should not be uncommented)

Configurations:

1. Implement AddPartToProduct() and AddModuleToProduct() methods in Configuration entity. (Note: the similar approach as with AddPartToModule() method from the Module entity.
2. Implement GetPrice() and SetDiscount() in the same entity.
3. Uncomment region named as “Configuration 1”and try to run the program. The program should work without compile and run time errors.

Extension method:

1. Complete the implementation of PriceWithCurrency() method in Extensions entity.
2. Uncomment the lines where PriceWithCurrency() method is used (Module and Configuration entities) and try to start the program.