Sofia University
Department of Mathematics and Informatics

Course : OO Programming C#.NET
Date: December 14, 2020
Student Name:

Lab No. 9b

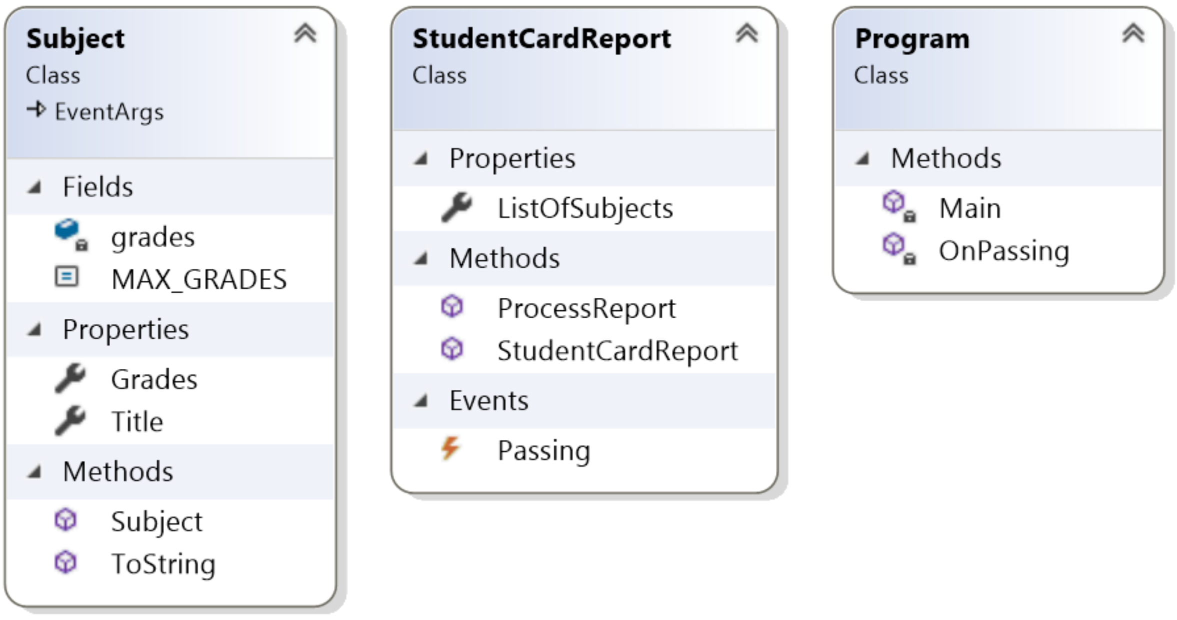
Submit the all C# .NET files developed to solve the problems listed below. Use comments and Modified-Hungarian notation.

Problem No.1

Write an application that handles a student report card. The application shall save the marks of **Computer Science**, **Math** and **English**. Each **Subject** has a **Title** and an array **grades** of **MAX_GRADES** elements, where each element represents a **grade** in the interval [0, 150] marks. If a student obtains at least **75** of the **150** marks per grade, an event named **Passing** will trigger and show a congratulation message on passing the exam. Otherwise, the application displays a message for assigning an "**F**" grade.

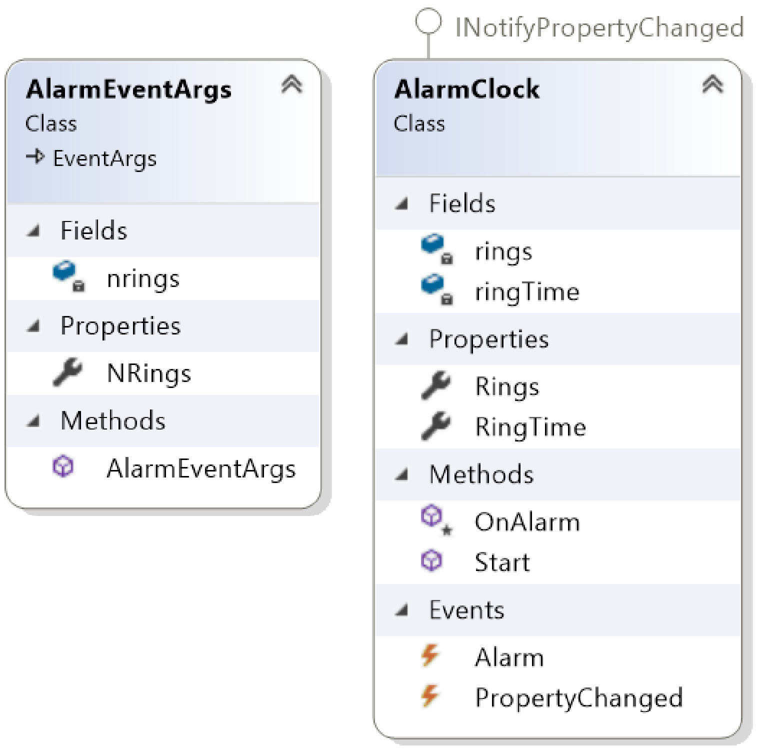
Use **EventHandler** to define the event **Passing**, where **class Subject** shall represent the event object and **class StudentGradeReport** represents the event source. Use method ProcessReport in **class StudentGradeReport** to browse **List<Subject> ListOfSubjects** elements and trigger event **Passing** once an element of array **grades** in a **Subject** is above 75 .Use the **public static void Main**() method in **class Program** to instantiate the event source. Write method **OnPassing** in **class Program** for handling the event **Passing** of the event source and subscribe for the event **Passing** published by the event source using this method.

Create a **List<Subject>** with **Random** values for the elements of array **grades** in each **Subject** to test handling of the event **Passing**.



Problem No. 2

Напишете приложение на C# , което моделира действието на часовник- будилник.



В основата на приложението да бъде дефинирането и обработката на събитието `Alarm` **и промяна на свойството** `RingTime`**.**

1. **Нека обектът** `Alarm`**EventArgs на събитието** `Alarm` **има свойството**

   ```
   public int Rings; // 0 by default
   ```

2. **Използвайте** `EventHandler` **за представяне за обработка на събитието**`Alarm` и `PropertyChangedEventHandler` **за представяне за обработка на събитието** `PropertyChanged`

3. **Дефинирайте** `class AlarmClock` – (будилникът, който ще е източник на събитието) **Нека** `AlarmClock` **публикува събитията** `Alarm` **и** `PropertyChanged`

   Нека `class AlarmClock` има свойства

   ```
   public int Rings;     // 1 by default
   public int RingTime;  // 0 by default
   ```

4. **При промяна на свойството** `RingTime` **да се обработи събитието** `PropertyChanged`
5. **При промяна на свойството** `RingTime` **да се обработи събитието** `PropertyChanged`
6. **Добавете в** `class AlarmClock` **следните методи:**

```
protected void OnAlarm(AlarmEventArgs e)
    {

        //Invoke  the event handler.
        Alarm?.Invoke(this,e);

    }
```

```
    }
    // event invoking method
    public void Start()
    {
       for (;;)
       {
          rings--;
          if (rings<0)
          {
             break;
          }

          else
          {
                AlarmEventArgs  e = new AlarmEventArgs  (rings);
                OnAlarm(this,e);
             }
          }
    }
```

7. **Дефинирайте** `class AlarmClockTest`**, който да се абонира за събитията, публикувани от** `AlarmClock` **обект. Инициализирайте** `AlarmClock` обекта да звъни 10 пъти.

   Напишете два метода за обработка на събитието `Alarm` на обекта `AlarmClock` и абонирайте тези методи за обработка на това събитие. **Всеки от тези методи да извежда броят на оставащите прозвънявания с различен текст(симулира различен тон на звънене).**

   Напишете метода за обработка на промяна на свойството `ringTime` на обекта `AlarmClock` **и абонирайте този метод за обработка на това събитие**. **Методът да извежда името на свойството, които се променя заедно с новата му стойност.**

   Напишете `main`**(), който изпълнява метода** `Start`() на `AlarmClock` обекта в `AlarmClockTest`

   Променете `AlarmClock` така че методът `Start`() да започва „звънене" след определено време. **Използвайте за целта**

```
using (var task = Task.Delay(timeInMilliSec))
{
   task.Wait();
}
```

Problem No.3

Consider a use case, where the main actors **`Employee`**, **`Manager`** and **`Store`** are represented in the following UML class diagram.

Class **`Store`** has a unique **`STORE_NAME`** like "Store 1", Store 2" etc. (**`static`** datamember **`cnt`**) and a **`List<Product>`** named **`listOfProducts`**, where each product is an instance of **`class Product`** shown on the same diagram. The **`get`** property **`ListOfProducts`** returns a list all the current entries in **`listOfProducts`** (not their copies), while the **`set`** property assigns a deep copy of **`value`** to **`listOfProducts`**.

Store instances are managed by an **`Employee`** (**`worker`** datamemeber) and a **`Manager`** (**`worker`** datamemeber).

Class **`Store`** publishes three events, an **`EventHandler`** event **`Appoint`** and two **`PropertyChangedEventHandler`** events **`PropertyChanged`**. Classes **`Employee`** and **`Manager`** (note the IS-A relation between them) serve as event objects for this events.

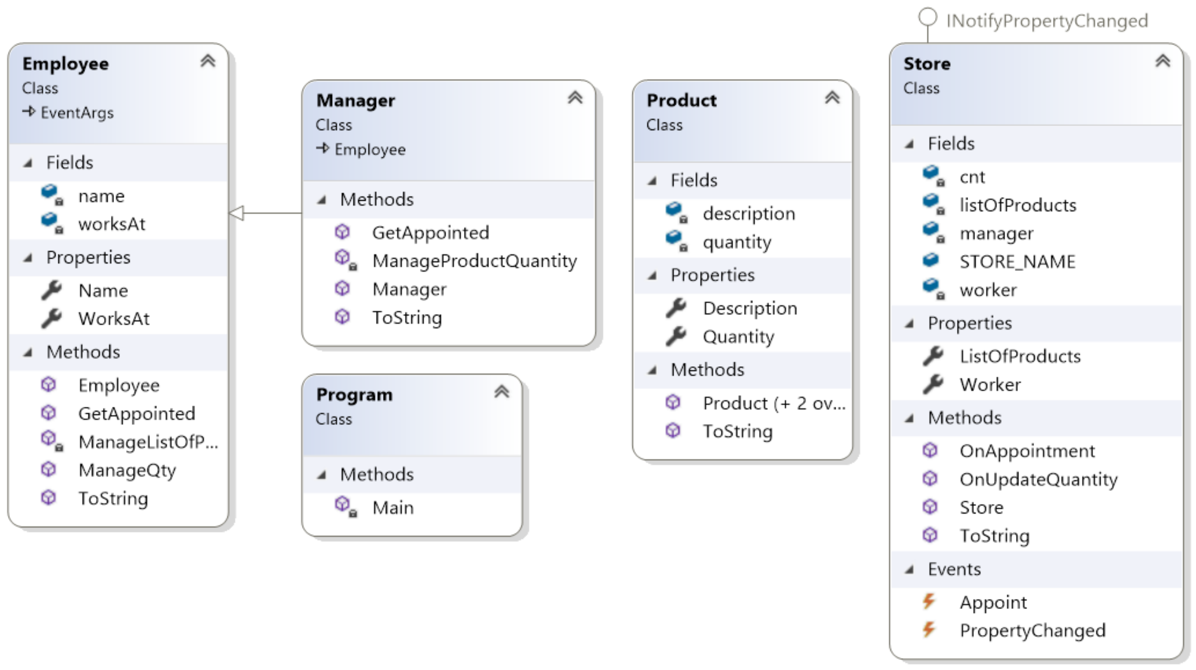
The event **`PropertyChanged`** triggers for name "**`ProductQuantity`**" in method **`OnUpdateQuantity(int index, int newQty`**`)` of class **`Store`** when the quantity of a **`Product`** with **`index`** in the **`listOfProducts`** is updated to **`newQty`** by the **`worker`** in the store`.` The method prints out data about the **`Product`** instance being updated with its current and the updated **`quantity`**. After that it triggers the method for handling a property named "**`ProductQuantity`**".

Class **`Manager`** defines **`internal`** method **`ManageProductQuantity`**`()` for handling changes in property "**`ProductQuantity`**" (only instances of **`Manager`** are subscribed for this event**)**

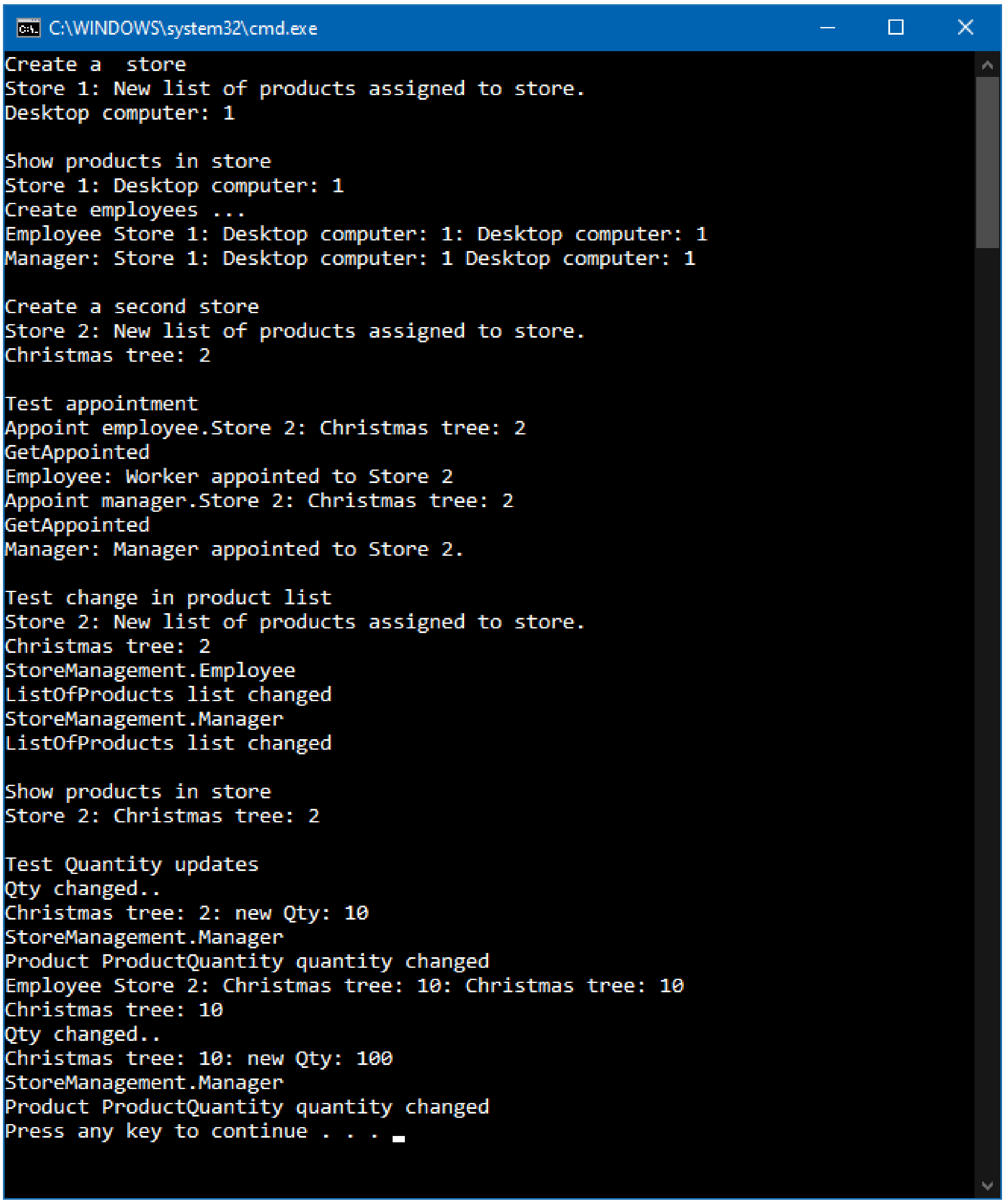
The **`OnUpdateQuantity()`** method is called by the **`worker`** at the **`Store`** using method **`ManageQty(Product p, int qty)`** defined in **`class Employee`**. Method **`ManageQty()`** delegates the execution of this task to method **`OnUpdateQuantity()`** of the store, where the worker is employed.

The event **`PropertyChanged`** triggers for name " **`ListOfProducts`**" in the **`set`** property **`ListOfProducts`** of class **`Store`** when a new list with products is assigned to the store`.` The **`PropertyChanged`** event for name " **`ListOfProducts`**" is handled by **`internal`** method **`ManageListOfProducts`**`()` defined in class **`Employee.`** The method displays the type of **`Employee`** that handles the **`event`** and a message with the **`PropertyName`**. (instances both of **`Employee`** and **`Manager`** are subscribed for this event**)**

The event **`Appoint`** triggers when an **`worker`** or a **`manager`** gets appointed to a store by means of the **`OnAppointment`**(**`Employee employee`**) method in class **`Store`**. Depending on the contents of the **`employee`** parameter this method initializes datamember **`worker`** or **`manager.`** Besides, the method subscribes the newly appointed **`worker`** or **`manager`** to the method **`GetAppointed`**() used to handle this event in classes **`Employee`** and **`Manager`**. It also subscribes the appointed worker and manager to **`ManageListOfProducts`**`() and respectively` **`ManageListOfProducts`**`() and` **`ManageProductQuantity().`** The implementation of method **`GetAppointed`**() in both classes updates the **`worksAt`** datamember with the place of employment (reference to the **`event source`**, instance of **`class Store`**). This method also prints out a message that the **respective** **`Employee`** or **`Manager`** are appointed with text showing the **`Employee`** working position and the store **`STORE_NAME`** of employment.



Test the project solution with sample data. Shown below is sample output in the program execution.



```
C:\WINDOWS\system32\cmd.exe
Create a  store
Store 1: New list of products assigned to store.
Desktop computer: 1

Show products in store
Store 1: Desktop computer: 1
Create employees ...
Employee Store 1: Desktop computer: 1: Desktop computer: 1
Manager: Store 1: Desktop computer: 1 Desktop computer: 1

Create a second store
Store 2: New list of products assigned to store.
Christmas tree: 2

Test appointment
Appoint employee.Store 2: Christmas tree: 2
GetAppointed
Employee: Worker appointed to Store 2
Appoint manager.Store 2: Christmas tree: 2
GetAppointed
Manager: Manager appointed to Store 2.

Test change in product list
Store 2: New list of products assigned to store.
Christmas tree: 2
StoreManagement.Employee
ListOfProducts list changed
StoreManagement.Manager
ListOfProducts list changed

Show products in store
Store 2: Christmas tree: 2

Test Quantity updates
Qty changed..
Christmas tree: 2: new Qty: 10
StoreManagement.Manager
Product ProductQuantity quantity changed
Employee Store 2: Christmas tree: 10: Christmas tree: 10
Christmas tree: 10
Qty changed..
Christmas tree: 10: new Qty: 100
StoreManagement.Manager
Product ProductQuantity quantity changed
Press any key to continue . . .
```

Problem No.4

Write an application that handles a BankAccount transactions. The user can credit and debit his account. When the account balance fall below a predefined minimum as a result of money withdrawal he gets a warning message that his balance has fallen below the predefined minimum. When the user credits and debits the account trigger another property event in case the user attempts to enter a negative value. Handle this event by prompting the user to enter a nonnegative number. Use TryParse() method for handling the input task.