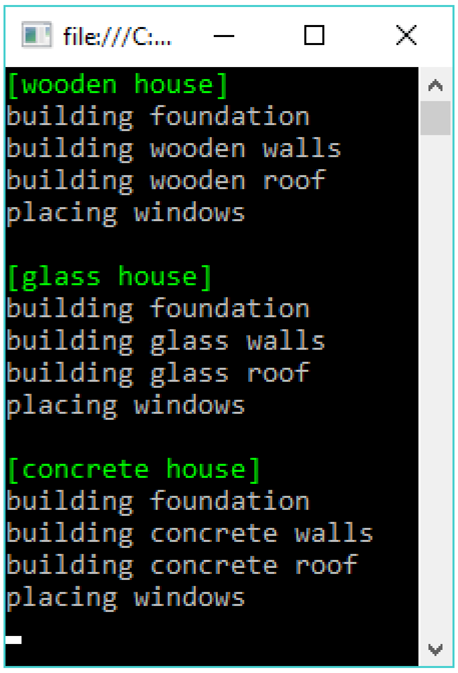
Assignment 1 (15 points)

Create a Console project in Visual Studio with name '**Assignment1**', and give the solution the name '**DesignPatterns-exam**'.
(the other assignments (2..4) will also be added as projects in this solution, see later)

In this assignment a (simple) Console application has to be made, in which several types of houses can be build. Building a house involves 4 fixed steps: 1. building the foundation, 2. building the walls, 3. building the roof,and 4. placing the windows. The building of the foundation and placing the windows is independent of the type of house, the building of the walls and the building of the roof do depend on the type of house.

Use the C#-code below (you can find it on Moodle, don't change it), implement the required classes (*using the appropriate Design Pattern*) in order to generate the output below.

```
private void Start()
{
  PrintHeader("[wooden house]");
  BaseHouse house1 = new WoodenHouse();
  house1.BuildHouse();

  Console.WriteLine();

  PrintHeader("[glass house]");
  BaseHouse house2 = new GlassHouse();
  house2.BuildHouse();

  Console.WriteLine();

  PrintHeader("[concrete house]");
  BaseHouse house3 = new ConcreteHouse();
  house3.BuildHouse();
}
```



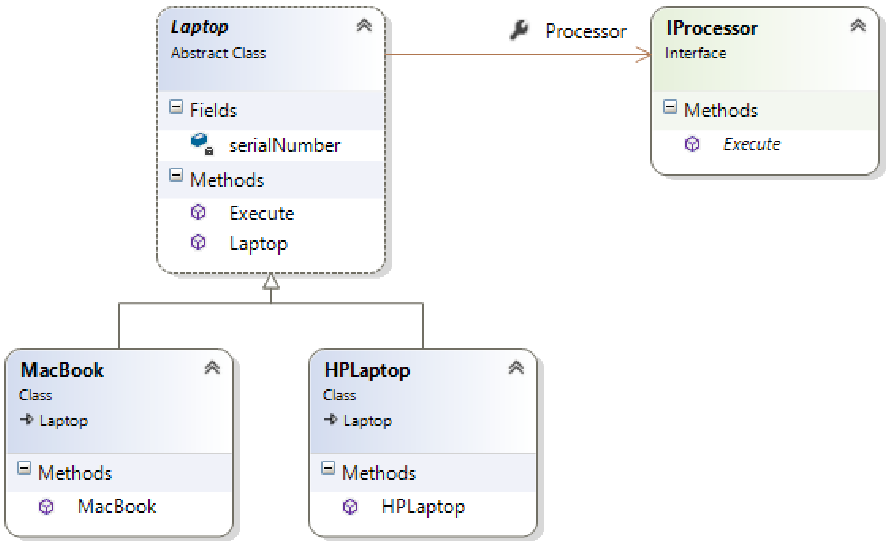
```
[wooden house]
building foundation
building wooden walls
building wooden roof
placing windows

[glass house]
building foundation
building glass walls
building glass roof
placing windows

[concrete house]
building foundation
building concrete walls
building concrete roof
placing windows
```

Assignment 2 (20 points)

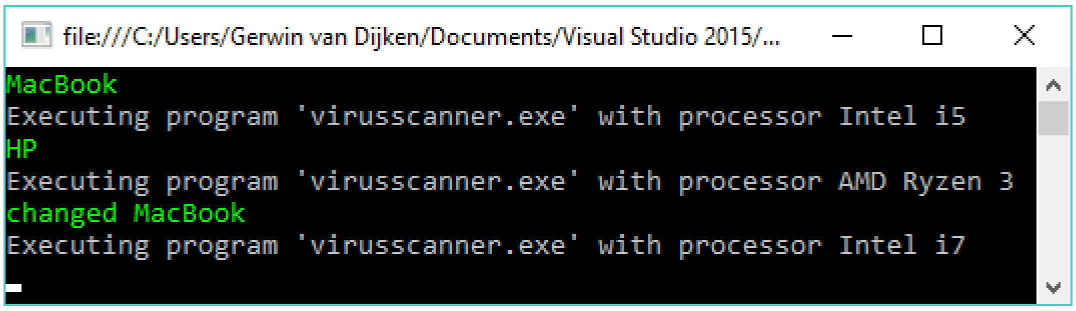
Add to solution 'DesignPatterns-exam' a Console project with name '**Assignment2**'.

Implement a Console application that contains different kind of laptops (derived classes from *abstract* class Laptop). Each laptop uses a certain kind of processor (Intel-i5, Intel-i7 or AMD-Ryzen-3): a MacBook uses by default an Intel-i5 processor, a HP-laptop uses by default an AMD-Ryzen-3 processor. The kind of processor (of a laptop) can be changed during the execution of the application. *Use the appropriate Design Pattern*.

With the (partially given) class diagram below and the C#-code (on Moodle, don't change it, except TODO-line) it has to be clear how to implement the application.



```
private void Start()
{
    PrintHeader("MacBook");
    Laptop macBook = new MacBook("S/N A1287");
    macBook.Execute("virusscanner.exe");

    PrintHeader("HP");
    Laptop hp = new HPLaptop("S/N 893419");
    hp.Execute("virusscanner.exe");

    PrintHeader("changed MacBook");
    // change macBook here... (TODO)
    macBook.Execute("virusscanner.exe");
}
```


file:///C:/Users/Gerwin van Dijken/Documents/Visual Studio 2015/...

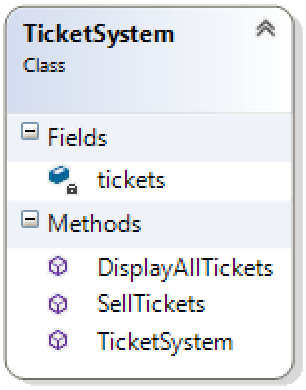
```
MacBook
Executing program 'virusscanner.exe' with processor Intel i5
HP
Executing program 'virusscanner.exe' with processor AMD Ryzen 3
changed MacBook
Executing program 'virusscanner.exe' with processor Intel i7
```

Assignment 3 (15 points)

Add to solution ‘DesignPatterns-exam’ a Console project with name ‘**Assignment3**’.

In an application a ticketsystem is used at several places. To avoid that too much tickets are being sold, there should not be multiple instances of this ticketsystem.

Implement class ‘TicketSystem’ (including the given members/methods) and modify this class to make sure there can be only one instance created from it. *Use the appropriate design pattern.*



Use a Dictionary for the tickets (`Dictionary<string, int>`), that can store for each artist (=key: string) the number of tickets (=value: int). Copy the code below to the constructor of class TicketSystem to have some (hardcoded) tickets:

```
    tickets = new Dictionary<string, int>();
    tickets.Add("Bruno Mars", 250);
    tickets.Add("Coldplay", 175);
    tickets.Add("Ed Sheeran", 150);
```

The definition of the 2 methods are:

```
public void DisplayAllTickets() { ... }
this method displays the number of tickets for all artists

public void SellTickets(string artist, int count) { ... }
this method decreases the number of tickets for the given artist (if present)
```

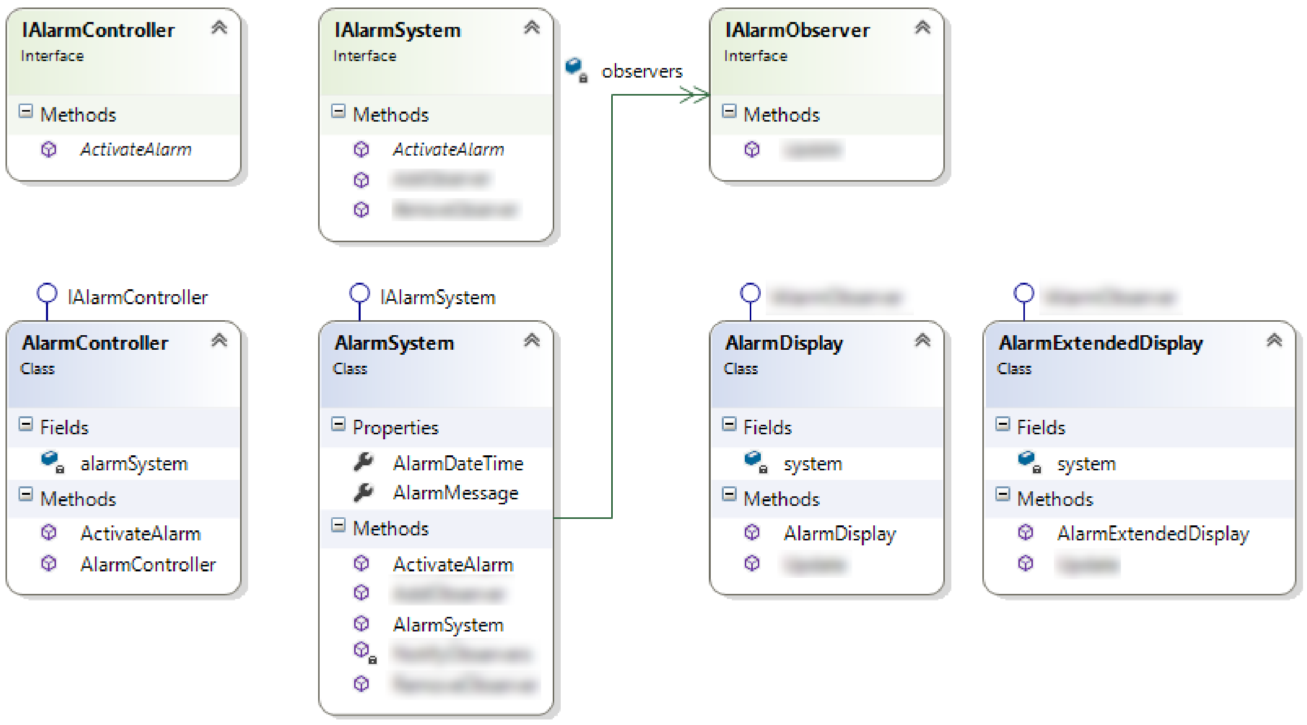
Implement a simple program to show how a TicketSystem can be created and used. Also show that using 2 ‘different’ ticket systems will use one instance.

Assignment 4 (30 points)

Add to solution 'DesignPatterns-exam' a Console project with name '**Assignment4**'.

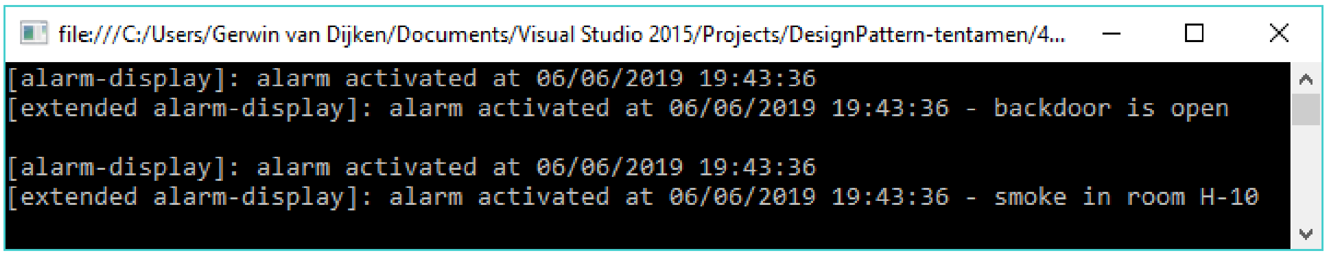
Create a Console application according to the classdiagram below, in which an alarm-system can activate several alarms (messages). Multiple displays can 'listen' to this alarm-system. Make sure that all (subscribed) displays are updated, every time an alarm is activated.

There are 2 display types: a normal display (only displaying the date/time of the alarm) and an extended display (displaying the date/time of the alarm and the message of the alarm).
(format date/time: {0: dd/MM/yyyy HH:mm:ss})



You can test your classes/interfaces with the C#-code on the next page (code can be found on Moodle, <u>don't change this code, except for creating a controller and the displays</u>).

```
private void Start()
{
  // create alarm system
  IAlarmSystem alarmSystem = new AlarmSystem();

  // create controller
  ... (TODO)

  // create displays
  ... (TODO)

  // activate the alarm system a few times (for testing)
  controller.ActivateAlarm("backdoor is open");
  Console.WriteLine();
  controller.ActivateAlarm("smoke in room H-10");
  Console.WriteLine();
}
```


file:///C:/Users/Gerwin van Dijken/Documents/Visual Studio 2015/Projects/DesignPattern-tentamen/4...

```
[alarm-display]: alarm activated at 06/06/2019 19:43:36
[extended alarm-display]: alarm activated at 06/06/2019 19:43:36 - backdoor is open

[alarm-display]: alarm activated at 06/06/2019 19:43:36
[extended alarm-display]: alarm activated at 06/06/2019 19:43:36 - smoke in room H-10
```