*Part 1*

**1. Description**

**BoardR** is a task-management system which will evolve in the next several weeks. During the course of the project, we will follow the best practices of Object-Oriented Programming and Design.

**2. Project information**

* Framework: **.NET Core 3.0**
* Language: **C# 8**

**3. Goals**

Your goal is to design and implement the main building blocks of the application - the **BoardItem** class and the **Board** class.

**4. Setup**

Create a new console application and name it **BoardR**  
**Make sure that you are using .NET Core, not .NET Framework**  
Leave the main method empty for now.  
Create two classes - Board and BoardItem, each in a separate file.

**5. BoardItem class**

**Description**

This class will model our idea of an **Item** that we can put in a **Board** (check [Trello](https://trello.com/)). A BoardItem can be used to describe anything - a **task**, **bug**, **note**, **assignment**...  
A minimum viable BoardItem should have at least a **title** (describes what this item is about), **dueDate** (when it should be done), and **status** (describes the state of this item - being worked on, being completed, etc..)  
For a useful BoardItem, the **title** should be non-empty and should have at least several characters. The **dueDate** should probably be in the future - we can't expect a task to be finished before we created it. There must be some rules on how a BoardItem changes its state - for example, from a state you can only *advance* to the next one or *rollback* to the previous one.  
When creating a BoardItem, we must be forced to provide title and date, and we must start from the initial state. (a constructor with the right arguments will come in handy)  
We usually have more than one BoardItem, so this class **can't be static**.

**Constructor**

* Should accept a title, a date and assign those to their respective fields
* A new BoardItem must have its Status as Open
* Example:

var item = new BoardItem("Refactor this mess", DateTime.Now.AddDays(2));

Console.WriteLine(item.title); // Refactor this mess

Console.WriteLine(item.dueDate); // 25/01/2020 12:09:49 PM (this will vary depending on when you run the code)

Console.WriteLine(item.status); // Open

**Fields**

* **title**: *string*, should never be null or empty, length should be [5..30] ([How to read bracket notation for ranges?](https://stackoverflow.com/questions/4396290/what-does-this-square-bracket-and-parenthesis-bracket-notation-mean-first1-last))
* **dueDate**: *DateTime*, should never be in the past
* **status**: Open <-> Todo <-> InProgress <-> Done <-> Verified think of a way to express this possible **set of values**.

Hint: you can **validate** in the constructor

**Methods**

* **RevertStatus()** - returns the status to a previous state - e.g. from **Todo** to **Open**, from **Done** to **InProgress**, etc (no effect if the status is **Open**)
* **AdvanceStatus()** - advances the status to a next state - e.g. from **Open** to **Todo**, from **Done** to **Verified**, etc (no effect if the status is **Verified**)
* Example:

var item = new BoardItem("Refactor this mess", DateTime.Now.AddDays(2));

Console.WriteLine(item.status); // Open

item.AdvanceStatus();

Console.WriteLine(item.status); // Todo

item.AdvanceStatus();

Console.WriteLine(item.status); // InProgress

item.RevertStatus();

Console.WriteLine(item.status); // Todo

* **ViewInfo()** - returns information about the current BoardItem in format '{title}', [{status}|{dueDate(dd-MM-yyyy)")}]" ([How to format date and time?](https://docs.microsoft.com/en-us/dotnet/standard/base-types/custom-date-and-time-format-strings))
* Example:

var item = new BoardItem("Refactor this mess", DateTime.Now.AddDays(2));

Console.WriteLine(item.ViewInfo());

// 'Refactor this mess', [Open|25-01-2020]

**6. Board class**

**Description**

The **Board** class will be used to organize all the BoardItems that we create. In the future, we might want to use it for searching, grouping, viewing, storing... For now, the **Board** will be no more than a collection of BoardItems. In the next couple of days, we will enhance it.  
The **Board** class must be able to keep track of all **BoardItems** in our application. If we have more than one instance, there is no reliable way to control which items go into which board.  
The easiest way to **prevent a class from being instantiated is to mark it as static**. This way, we are sure that all the items will go into a single **Board**.

**Constructor**

Nothing to instantiate

**Fields**

List of BoardItems - we must be able to add items to the board

**Methods**

We will add some in the next chapter.

**Example code**

var item = new BoardItem("Refactor this mess", DateTime.Now.AddDays(2));

item.AdvanceStatus();

var anotherItem = new BoardItem("Encrypt user data", DateTime.Now.AddDays(10));

Board.items.Add(item);

Board.items.Add(anotherItem);

foreach (var boardItem in Board.items)

{

boardItem.AdvanceStatus();

}

foreach (var boardItem in Board.items)

{

Console.WriteLine(boardItem.ViewInfo());

}

// Output:

// 'Refactor this mess', [InProgress|25-01-2020]

// 'Encrypt user data', [Todo|02-02-2020]