INFORMATION & COMMUNICATION TECHNOLOGY ENGLISH COURSE

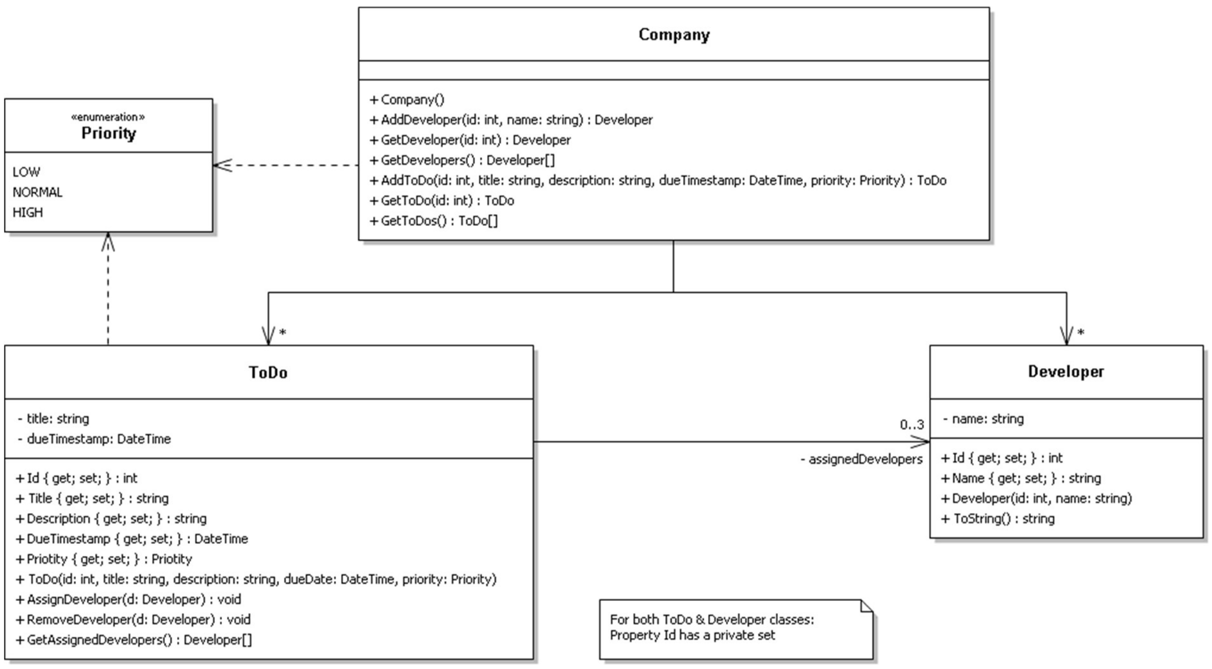
Course	:	OOD - Laptop assessment
Teachers	:	AVET, BAHR, HTVT, MANO, RIAD, RVL
Date	:	11th of May 2020
Time	:	09.30 – 12.00 (150 minutes)
Accepted resources	:	You are allowed to use everything on paper (books, notes, etc.) and on your laptop, but only what you bring in; you are not allowed to borrow anything from anybody else. During the assessment it is not allowed to use the network. You should make the exam yourself: no communication with MSDN or google for help and no communication with other students, like using Facebook, e-mail, Skype, Dropbox, mobile phone, etc.

Weight of requirements

		NFR-01	*NFR-02*	*FR-01*	*FR-02*
Weight	:	36%	30%	20%	14%

Extending a To-do application

You are tasked to extend an unfinished application for an IT Company. Currently, the application allows a user to store developers and to-do tasks (see the UML Class Diagram). It is also possible to assign up to three developers to work on a to-do task.



Company
+ Company()
+ AddDeveloper(id: int, name: string) : Developer
+ GetDeveloper(id: int) : Developer
+ GetDevelopers() : Developer[]
+ AddToDo(id: int, title: string, description: string, dueTimestamp: DateTime, priority: Priority) : ToDo
+ GetToDo(id: int) : ToDo
+ GetToDos() : ToDo[]
«enumeration»
Priority
LOW
NORMAL
HIGH
*
*
ToDo
- title: string
- dueTimestamp: DateTime
+ Id { get; set; } : int
+ Title { get; set; } : string
+ Description { get; set; } : string
+ DueTimestamp { get; set; } : DateTime
+ Priotity { get; set; } : Priotity
+ ToDo(id: int, title: string, description: string, dueDate: DateTime, priority: Priority)
+ AssignDeveloper(d: Developer) : void
+ RemoveDeveloper(d: Developer) : void
+ GetAssignedDevelopers() : Developer[]
0..3
- assignedDevelopers
Developer
- name: string
+ Id { get; set; } : int
+ Name { get; set; } : string
+ Developer(id: int, name: string)
+ ToString() : string
For both ToDo & Developer classes:
Property Id has a private set

The GUI of the application is postponed till the logic layer is done (i.e. now it is an empty form), but to ensure the application works properly Unit Tests are created.

You are now required to add the following requirements to the application:

- **NFR-01: Maintainable code is achieved by proper object-oriented design**
  You are expected to apply the *object-oriented design principles* covered during the lectures.
- **NFR-02: Code is tested/testable**
  You are expected to write unit tests for the code you implemented, testing both happy flow (main success scenario) and edge cases (extensions) mentioned in the FR and step descriptions. Note that you must ensure that the complete public interface of a class is tested.
- **FR-01: Include Meetings**
  Extend the application with *Meeting*s, which have at least a *unique id, title, description, priority, when* (date & time) and *who are invited*.
- **FR-02: Include filtering by priority**
  It must be possible to view *ToDo*s and *Meeting*s with a certain priority. The *ToDo*s and *Meeting*s are sorted together by their date & time and then title.

The steps described on the next pages contain more (implementation) details of these requirements.

Important!
The supplied start-up project contains code which throws exceptions. You must make sure to also throw exceptions when appropriate, but they do not have to exactly match the ones in the supplied code.

For example, when a public method is unable to add an object to a list, an exception should be thrown instead of doing nothing.
Another example, when an invalid value is assigned via a property, do not set a corrected value but throw an exception.

Step 1:

Analyse the supplied start-up project and extend the application to also support *Meetings* for the `Company`; make sure you apply proper object-oriented design.
A meeting has the following data:

Data	Constraints
Unique id	A unique number representing the meeting. This must be unique for all `Meeting`- and `ToDo`-objects in the whole `Company`-object. E.G. when a `Meeting`-object has the id 1 no other `Meeting` or `ToDo`-object may have it, but a `Developer`-object can have the id 1.
Title	A title for the meeting. This is required (i.e. cannot be empty).
Description	A description of what the meeting is about. This is optional.
Date and time	This is to indicate when the meeting is.
Developers	This is to indicate who are invited to the meeting. There must always be at least be 2 developers attending and it must not be possible to have the same developer invited/stored multiple times.
Priority	This to indicate what the priority of the meeting is. This is required and has the same priorities as a to-do.

<u>Without violating</u> the above-mentioned constraints, it must be possible to:

- *Add a new developer to the meeting*
- *Remove an invited developer from the meeting*
- *See the invited developers as an array*

You must also create the Unit Tests to ensure that:

- Adding a Meeting is handled correctly when valid or invalid data is supplied.
- When 'interacting' with a `Meeting`-object via its public interface, none of the above-mentioned constraints is violated.

When required, you must refactor the start-up code it to a proper object-oriented design. Be sure to not remove existing functionalities (e.g. public methods).
Also do not forget to refactor the Unit Test class `CompanyTest` to match the new implementation and make sure nothing has 'broken' after your changes.

Step 2

Extend the application to also support viewing of *To-dos* and *Meeting*s with a certain *priority*. Note that this does not mean you have to create the GUI, but just the code in the logic-layer to support this.

You must also create the Unit Tests to ensure that:

- The correct *To-dos* and *Meeting*s*,* if any, are returned.

Step 3

Extend the viewing by priority (see previous step) to return the objects sorted by their *date and time* in an ascending order (e.g. the earlier ones should be first).
When the *date and time* is the same, the order between those objects is based around the *title* in an ascending order (e.g. A before Z).

You must also create the Unit Tests to ensure that:

- The returned *To-dos* and *Meeting*s are in the order described above.

- End of laptop assessment -