Assignment 1

Software Development 2022

Department of Computer Science
University of Copenhagen

Johan Topp <joto@di.ku.dk>
Sia Sejer <sia.sejer@di.ku.dk>

Version 1.0: February 4th

Due: February 11th

Abstract

For this exercise there is an individual part consisting of the development of C# code as well as a technical report. Both ***must*** be done alone. See section 6 for details on how to hand in. In this exercise you will begin your work with Object-Oriented programming in Visual Studio Code using C# and .NET Core 6.0.

Contents

1 Installing Required Software

During this course you will be using the editor called Visual Studio Code to write your programs and the `CLI`-tool `dotnet` to build and run your programs and their tests.

.NET Core 6.0

You need the `.NET Core 6.0 SDK`. Find it here:
`https://dotnet.microsoft.com/en-us/download/dotnet/6.0`

Visual Studio Code

You need the editor `Visual Studio Code`. Either download it via the link below or install it through your package manager.
`https://code.visualstudio.com/`

C# Extension

You also need the C# extension for `VsCode`. Install it by navigating to the extensions-pane in the editor and searching for C#. The full title of the extension is:
`C# for Visual Studio Code (powered by OmniSharp)`.

2 Hello world

Creating a project

Using the dotnet cli tool, create and build a project. First, create a new directory for your project, name it helloWorld. Navigate to the directory in a terminal and create a new C# console project with the following command.
`$ dotnet new console`.

This will create a few files:

`Program.cs` An "empty" C# program containing a main-method.

`helloWorld.csproj` A file containing metadata about the project. The `dotnet` tool needs this in order to know how and what to compile.

`obj/` A folder containing various things related to the project, that `dotnet` handles for us.

Test out the new project (and your installation of `.NET Core SDK`) by building and running it.

Building: `dotnet build`

Running: `dotnet run`

If the application prints `Hello World!` when run, everything works as expected.

3 DIKULecture

This exercise is meant as a first foray into OOP, `Visual Studio Code` and `C#`. Object-oriented Programming is a tool to model the real world. We will investigate this by modeling the most exciting part of the student life post COVID-19: Going to an online lecture.

Create a new directory, `DIKULecture`. Enter it and create a new console project. Now it's time to open up `Visual Studio Code`. In `VsCode`, click `File -> Open` and navigate to the `DIKULecture` directory you just created. You want to open the *directory*, not a specific file in it! Click `Open`. You should now be able to see the project in the `Explorer` pane of `VsCode`. `VsCode` might warn you about missing assets. If so, click yes to add them.

Click on `Program.cs` in the `Explorer` pane, to open the file for editing.

3.1 Building from VS Code

It is very convenient to be able to build your project directly when editing. `VsCode` supplies a built-in terminal we can use. If it does not appear by itself, click `View -> Terminal`. Try to build and run your project from the terminal inside VsCode. The program should print `Hello World!`.

Modelling the DIKULecture

Doing some very simple Object-Oriented Analysis, we decide that the DIKULecture is indeed a type of chat room, and might share properties with other chat rooms. All UCPH chat rooms, including the DIKULecture, is associated with a topic, that is to discussed. Creating a `ChatRoom`-class seems like a good abstraction, in case we want to add more chat rooms for other events than lectures at a later point. Note that for this assignment, we would like all fields to be private. If other classes need to access or change these fields, it should be done using public methods or public setters and getters.

- Create a new file, `ChatRoom.cs`. It should be in the same folder as `Program.cs`
- In this file, define a class, called `ChatRoom`
- The namespace should be DIKULecture
- Add a field, `name` to the class. The field should be private.
- Add a constructor that takes a single argument, `String name`, and sets the `name`-field of the `ChatRoom` class.

Remember to check that your project still builds. Do this often when adding or modifying code!

Now for the next step - Adding the actual DIKULecture!

- Create a new file, `Lecture.cs`
- `Lecture` should inherit the `ChatRoom`-class. To do this, you need to create a constructor that takes one argument, `String name` and does nothing but pass along its argument to the base-class (`ChatRoom`). Note that a class does not inherit private fields unless it has public setters and getters, so you should make sure that your name field has a getter and setter, which technically means it is a property of the class.

- Add a private field `numOfstudentsOnline`. Give it a default value of 0.
- Add a constructor to the `Lecture` class that takes one argument:

  3.1. `String name` → pass it along to `ChatRoom`

Once again, ensure the project builds.

Making the Lecture pretty

Open the `Program.cs` file, that contains the main method of the program. This is the entrypoint and is where execution of our program starts. The main method of `Program.cs` should at this point contain a single line: `Console.WriteLine("Hello world");`

- Create a new instance of a Lecture-object on the line before the `Console.WriteLine`
- Try to print your Lecture by giving it as an argument to the `WriteLine-method` (i.e. replace the `String` with your `Lecture` object).

If you run the program now, and have followed this assignment-text, you should see the following output: `DIKULecture.Lecture`. That doesn't tell us a lot about the Lecture - only that it is an *instance* of the `Lecture`-class, and that it belongs to the `DIKULecture` *namespace*. Luckily for us, object oriented programming has an easy solution for this.

Overriding ToString

All classes inherit from a class called `Object`. `Object` defines some useful methods that are relevant for all classes we can think of. One of these methods is the `ToString()`-method, which is called on an object whenever we implicitly try to convert our object to a String. When we tried to print the `Lecture` *instance* earlier, this is what happened.

Open `Lecture.cs` for editing.

- Override the `ToString()` method. `public override String ToString() { return String.Format( ...);}`
- You want to print the name of the lecture as well as the number of online students.

Try to run your program again. The output should now be the name of the lecture and the current number of students online (0).

Adding more classes

A lecture needs people - we need to model them!

- Create a class `Person`
- Add private fields for `name`, `occupation` and `age`.
- Add a constructor that takes all three arguments.

Both `Students` and `Speakers` are indeed `Persons`, and as such both classes could inherit from `Person`.

Create two more classes, `Student` and `Speaker` respectively.

- Make the `Student` class a subclass of `Person`

- Make the `Speaker` class a subclass of `Person` as well

A Student should be able to join a lecture.

- Add a private field to the `Student` class, `isInLecture`.
- Initialize the field to `false`
- Add another field to the `Student` class `lecture`. This field should be set to an instance of the lecture that the student is currently in and will be initialized later.

A Speaker should be able to begin a lecture and speak to the lecture.

- Add a field to the Speaker class, `isInLecture`.
- Initialize the field to `false`.
- Add another field to the Speaker class `Lecture`. This field should be set to an instance of the lecture that the speaker will speak to.

3.2 Time for some (inter)action - Adding methods

We like to think of a class as an encapsulated unit of data and related behavior. Up until now we have declared some classes with various fields (the data). These fields have gotten their value through the constructor or simply by specifying a value right there in the class declaration. Now we would like to add methods that can manipulate the values of the fields in each instance of the class (the behavior related to the data).

To do this, we will give the `Student` class a `Join` method and a `Listen` method. Also, we will give the `Speaker` class a `Broadcast` method and a `Speak` method.

3.2.1 Requirements

We want to accomplish the following with our program:

- Students can join a lecture, if they have not already joined another lecture.
- Students can listen to the lecture they have joined.
- Speakers can begin a lecture, if they have not already begun another lecture.
- Speakers can speak (broadcast[1]) to a lecture.
- Speakers should be able to alter the name of the lecture

3.2.2 Implementing the Requirements

When a `Student` joins a `Lecture`, we need to

1. Check if the `Student` already has joined a `Lecture`

`false` Then we have to change the state of our objects.

   - Increment the `numOfStudentsOnline`-field in the `Lecture` instance.
   - Change the `isInLecture`-field of the lecture-joining object.
   - Set the `Lecture` field of the lecture-joining object.

[1] You can do this by giving the Lecture class a field `Information`, that the Speaker can set with the Broadcast method, and the Student can get with the Listen method.

`true` The student cannot join the lecture.

When a `Student` listens to a `Lecture`, we need to

1. Check if the `Student` is in a lecture

`false` We can not listen to the information.

`true` We have to give the information to the student.

When a `Speaker` begins a `Lecture`, we need to

1. Check if the `Speaker` is already in a lecture

`false` We cannot begin the lecture.

`true` Then we need to change the state of our objects.

- Change the `isInLecture`-field of the lecture-beginning object.
- Set the `Lecture` field of the lecture-beginning object.

When a `Speaker` speaks at a `Lecture`, we need to

1. Broadcast information to the lecture.
   - Change the `information`-field of the speakers lecture object.

When a Speaker wants to change the name of the lecture, we need to

1. check if the Speaker is already in a lecture

`false` Nothing to change

`true` Change the name field of the lecture

There are many ways to implement this behavior and many choices to make while doing so. Think about what class each of the responsibilities belong to. Should a student be able to increment the `studentOnline`-field directly, or should they ask the Lecture to join through a method? Should Lecture have a `Join`-method that as argument takes a `Student` or `Speaker` instance, or should the `Student` and `Speaker` classes have methods that as argument take a `Lecture` instance? Think about who (what class) owns the data and thus who should be responsible for manipulating it. Also think about what restrictions - needed modifications - the design poses on the code base. As an example, if functionality where to be extended such that students could ask (broadcast) questions to the lecture and speaker.

4 Deliverable

- Implement the requirements (3.2.1) in any way you see fit.
- Write a short program in the `main` method of `Program.cs`, that exhibits that all the requirements have been implemented. Instantiate some `Students`, a `Speakers` and two `Lectures`, join a lecture, and let some students try to join more than one at once. Have a `Speaker` begin a lecture, and show that the `Students` can listen to what the `Speakers` has broadcasted.
- Write a short document (using LaTeX), no longer than 2-3 pages, explaining the choices you made in your implementation and your reasoning behind those choices. Besides explaining the choices in your implementation, your document needs to answer/discuss the following, in your own words:

- What is a class?
- What is an instance?
- What is a field?
- What is a method?
- What does it mean that a field or method is public or private?

We expect a couple of sentences for each question, not more.

Your `pdf` should not contain any code. For this first assignment, we are only concerned with the content of your text. What we do not expect:

- A proper report structure.
- Academic and/or Object-Oriented nomenclature - We want you to think and reason about objects, their interactions and how we can use object-oriented programming to solve problems - in the first week we do not expect you to know the proper names for anything related to Object-oriented analysis, design and programming.

5 Cleaning up your code

To keep your TA happy, and receive more valuable feedback, you should:

- Remove commented out code.
- Make sure that your files and folders have the correct names.
- Make sure the code compiles without errors and warnings.
- Make the code comprehensible (perform adequate renaming, separate long methods into several methods, add comments where appropriate).
- Make sure the code follows our style guide[2].
- Make sure to make a clean build: `dotnet clean`.
- Remove `obj/` and `bin/` directories as well as any auto-completion data or tagging databases that VS Code.

6 Submission

Your work must be submitted through Absalon. You should submit two files (seperately):

- Your report as a report called firstName-A1.pdf. Make the first letter lowercase. The same applies if you are in a group; firstNames_-A1.pdf.
- Your code as a zip-file. Zip the entire directory of your project, including the `.sln` and `.proj` files. The zip file should be named firstName-A1.zip and if in group firstNames-A1.zip

When submitting code make sure that you only submit what is required to run the code. In VS Code that is usually the `.csproj`, `.sln` as well as any `.cs` files. Exclude any OS specific files. When in doubt, attempt to simulate running the code from the zip file, i.e. copy and extract and run.

[2] See `https://github.com/diku-dk/su21-guides/blob/main/guides/CSharpStyle.md`.