Task Manager in C# Using OOP

Introduction

Task managers are incredibly useful tools for organizing daily activities, assignments and projects effectively. Therefore, in this lesson you will create a Task Manager application using C# and Object-Oriented Programming (OOP) principles[1]. By the end you should have a working application for yourself, that I hope you will find useful!

For each task you will need to send me the files directly as a zip file containing the whole project. The zip file name should be ***TaskManager<Your name>TaskN*** like the following example: *TaskManagerExampleTask0*.

If you see a small number on top of something, it indicates the resource needed to be understood for that. It will only show the first time that the new thing is introduced.

Needed resources for the lesson:

1. https://www.w3schools.com/cs/cs_oop.php (and the subsequent lessons from OOP).
2. https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=net-8.0
3. https://learn.microsoft.com/en-us/dotnet/api/system.datetime?view=net-8.0
4. https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/nullable-value-types
5. https://youtu.be/CFRhGnuXG-4?si=lqq49X9ewdds2Coy
6. https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/conditional-operator
7. https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/tokens/interpolated
8. https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/lambda-expressions

Index

Task 0: Setting Up the Project

Before writing the program itself you should create the proper classes and files for the program to execute. Task management and such will be done through the **TaskManager** class, while the main functionality will be directly done inside the main file.

Instructions:

- Create a new project. It should be named ***TaskManager<Your name>***. For example, if your name is John, name it *TaskManagerJohn*. Remember that it is a purely console based app, so create it accordingly. Please make sure that all namespaces are the same as your project name.
- Create a **TaskManager.cs** file. Make inside a **TaskManager** class and a **Task** class with their empty constructors. These new classes should have no functionality for now.
- Create a **Program.cs** file with *"Hello Task Manager!"* as output to test if the file works.

Your project should look like:

```
TaskManagerYourName/
├── TaskManagerYourName.sln
├── TaskManagerYourName/
│   ├── TaskManager.cs
│   └── Program.cs
```

Running Program.cs should output:

```
Hello Task Manager!
```

Task 1: Making the Task class

Making use of the Object-Oriented Programming structure of this application, we will use the **Task** class as a container for all the information of a task. For now, we will not add any methods!

For your understanding, each task will contain an **id** that will identify it inside the list. This task will hold a **description** of the task. It can optionally have a **due date**. Tasks can be marked as either **completed** or not. Therefore, each task will contain four variables of information.

Instructions:

- Create the following variables with the correct types inside **Task** class:
    - **Id**: public int
    - **Description**: public string
    - **DueDate**: public nullable[4] DateTime[3]. Remember that **DateTime** comes from the namespace **System** (see needed resources).
    - **Complete**: public bool
- Make the constructor for **Task** accordingly. Remember that it is only called when creating a new **Task**, so *Complete* should be false by default.

There is no output for this task.

Optional:

You can make a **Task** object inside Program.cs to test if this works. Output the different attributes that you have created with the test object. Remember to clean this code from the file once you reach the next task, but save it, don't delete it entirely!

Task 2: Adding tasks

Now that we have defined what a **Task** is, we can make the proper method to add one with the **TaskManager** class. In order to do this, we will need to create a list of tasks with an index for the next task. This has to be managed in the constructor too. Once we do that, we can move to the adding task method, that will take a description and an optional due date.

Instructions:

- Inside **TaskManager** create the following variables:
  - **tasks**: private List[2] of **Task**
  - **nextTaskId**: private int
- Make The constructor of **TaskManager**. Remember that **List** comes from the namespace *System.Collections.Generic*, therefore, to make a new **List** it will be started as a method. By default *nextTaskId* is 1, as that is the starting index.
- Create a new method for **TaskManager** named **AddTask**, that will take as parameters *description* (string) and *dueDate* (nullable DateTime). This method should create a new **Task** object with the *nextTaskId*, *description* and *dueDate*. Then, this object has to be added to the *tasks* list. Output success. Finally, increase *nextTaskId* by 1.

Quick help! Here is the line the answer sheet has for outputting adding the task. Remember to do this before increasing *nextTaskId*:

```
Console.WriteLine($"Task {nextTaskId} has been added.");
```

There is no output for this task unless you do some tests.

Optional:

Try creating a **TaskManager** object inside Program.cs, while adding new tasks with the method. You should see the adding task output. Remember to clean this code from the file once you reach the next task, but save it, don't delete it entirely!

Task 3: List tasks

In the last task you created the method to add tasks, but there was no way of outputting them! We will show all the tasks stored inside **TaskManager** outputting them directly to the console.

Instructions:

- Create a new method for **TaskManager** named **ListTasks**, that will take no parameters.
- Inside of this method, if the *tasks* count is equal to 0 then output "No tasks." and end the function (never nester ideology[5]).
- Otherwise, continue the function outputting "Task list:".
- Then, for each task in *tasks* output the *Id, Description, DueDate* and status. The variable *status* should be either "Complete" or "Incomplete" using a ternary conditional operator[6] from the **Task** variable *Complete*. Remember that interpolated strings exist[7].

Quick help! Here is the line the answer sheet has for outputting the task:

```
Console.WriteLine($"ID: {task.Id} - Description: {task.Description} - Due Date: {task.DueDate} - Status: {status}");
```

There is no output for this task unless you do some tests.

Optional:

Create a **TaskManager** object inside Program.cs, adding new tasks. You should be able to show the tasks list with the different attributes you set for each task. Remember to clean this code from the file once you reach the next task, but save it, don't delete it entirely!

Task 4: Complete task

Now that you have a list of tasks it's time to be able to complete them. We will create a method that will modify only one **Task** inside *tasks*, setting the *Complete* attribute as true. This will internally mark it as complete, aside from the method output, so in the future when calling the tasks list we will see that task as completed.

Instructions:

- Create a new method for **TaskManager** named **MarkTaskComplete**, that will take as a parameter *taskId* (int).
- Make a new **Task** object *taskToComplete* that finds the task from *taskId* inside *tasks*. Use a lambda expression[8] for this, that looks for *Id* being equal to *taskId*, as *tasks* stores **Task** objects and those cannot be fetched only from id.
- If *taskToComplete* is null output "Task not found." and end the function (never nester ideology[5]).
- Otherwise, set *Complete* to true. As *taskToComplete* is an object directly looking at a **List** memory address, you can modify its values there. Output success of this.

Quick help! Here is the line the answer sheet has for outputting completing the task:

```
Console.WriteLine($"Task {taskId} marked as complete.");
```

As well, have the lambda expression if needed (you will need to discover how to find a task inside of a list still!):

```
...t => t.Id == taskId...
```

There is no output for this task unless you do some tests.

Optional:

Create a **TaskManager** object inside Program.cs, add a task and mark it as complete. You should be able to see in the tasks list how it says "Complete" instead of "Incomplete". Remember to clean this code from the file once you reach the next task, but save it, don't delete it entirely!

Task 5: Delete task

In the scenario that you accidentally input a task you shouldn't or finish it and just want to see it gone, we need the ability to delete them. Therefore, you will create a method to delete one **Task** from the list of *tasks*. It will look similar to the method you wrote for your last task.

Instructions:

- Create a new method for **TaskManager** named **DeleteTask**, that will take as a parameter *taskId* (int).
- Make a new **Task** object *taskToDelete* that finds the task from *taskId* inside *tasks*. Use the same code as you did with the prior task.
- If *taskToDelete* is null output "Task not found." and end the function (never nester ideology[5]).
- Otherwise, remove from *tasks taskToDelete* and output success.

Quick help! Here is the line the answer sheet has for outputting deleting the task:

```
Console.WriteLine($"Task {taskId} deleted.");
```

There is no output for this task unless you do some tests.

Optional:

Create a **TaskManager** object inside Program.cs, add a task and delete it. Once you output the tasks list it should be empty. Remember to clean this code from the file once you reach the next task, but save it, don't delete it entirely!

Task 6: Final program

This is the final task for your project. We will be adding the functionality of the application by making it usable. It will consist of a loop that will be repeated, each time you call it you will have an operation to do. For it we will use all the methods we wrote earlier. This is the longest task, as we will be writing directly on **Program.cs** *Main()*, so take it slowly. You can do this!

The actions that the user will be able to do are:

- "1. Add Task"
- "2. List Tasks"
- "3. Mark Task as Complete"
- "4. Delete Task"
- "5. Exit"

Instructions:

- Create a new **TaskManager** object named *taskManager*.
- Make a new boolean *exit* as false. We will use this condition to keep the loop alive.
- Make a while loop that checks if *exit* is false. This will keep the program running until *exit* becomes true.
- Output on different lines "Task Manager Menu:" and the actions above.
- Take input in *choice* as int. Ask "Enter your choice: ".
- From on here you will make a switch case structure, following the list of actions above as the case conditions. Remember that for each action you don't need to output anything after the method of **TaskManager** as it outputs already.
- For "1. Add Task": Take in *description* and *dueDate*. Remember that *dueDate* is optional, so to take it in as that you can use *string.IsNullOrWhiteSpace(yourInputString)*. Use the **AddTask** method.
- For "2. List Tasks" simply use the **ListTasks** method.
- For "3. Mark Task as Complete" take in the *taskId* as input and use the method **MarkTaskComplete**.
- For "4. Delete Task" take in *taskId* as input and use the **DeleteTask** method.
- For "5. Exit" set *exit* as true.
- For the default case output "Invalid choice. Please enter a number from 1 to 5.".

(possible functioning of the program is in the next page)


```
Task Manager Menu:
1. Add Task
2. List Tasks
3. Mark Task as Complete
4. Delete Task
5. Exit
Enter your choice: 1
Enter task description: Finish this lesson
Enter due date (YYYY-MM-DD, optional): 2024-07-01
Task 1 has been added.

Task Manager Menu:
1. Add Task
2. List Tasks
3. Mark Task as Complete
4. Delete Task
5. Exit
Enter your choice: 2
Task List:
ID: 1 - Description: Finish this lesson - Due Date: 01/07/2024 0:00:00 -
Status: Incomplete

Task Manager Menu:
1. Add Task
2. List Tasks
3. Mark Task as Complete
4. Delete Task
5. Exit
Enter your choice: 3
Enter task ID to mark as complete: 1
Task 1 marked as complete.

Task Manager Menu:
1. Add Task
2. List Tasks
3. Mark Task as Complete
4. Delete Task
5. Exit
Enter your choice: 2
Task List:
ID: 1 - Description: Finish this lesson - Due Date: 01/07/2024 0:00:00 -
Status: Complete

Task Manager Menu:
1. Add Task
2. List Tasks
3. Mark Task as Complete
4. Delete Task
5. Exit
```

```
Enter your choice: 4
Enter task ID to delete: 1
Task 1 deleted.

Task Manager Menu:
1. Add Task
2. List Tasks
3. Mark Task as Complete
4. Delete Task
5. Exit
Enter your choice: 2
No tasks.

Task Manager Menu:
1. Add Task
2. List Tasks
3. Mark Task as Complete
4. Delete Task
5. Exit
Enter your choice: 5
```

Lesson finished… Now what?

Now this app works as expected! Feel free to add memory to the tasks, having a .txt file or a .json file. You could even make a database with SQL! As for other tasks you could do, try adding more parameters to the tasks, like priority. Try ordering the uncompleted tasks first and the completed ones below. Finally, you could use an app framework and do UI for the project. The possibilities are endless, so tailor this app to fit your needs.

As well, now you have grasped most of the concepts of Object-Oriented Programming! That was the main goal of this lesson and if that is the only thing you did with this project, then, that is good enough. Thank you for reading!