(420-3P5-AB) Programming III
Fall 2020

Course Project – V1.0
Worth 15%

Final Due Date: 17 December 2020 (On Lea, end of the day)

Deductions for late submissions is 10%/day, up to a maximum of 2 days.
This document has 5 pages

Objective
The objective of this project is to build a simple C# WPF application. You should utilize the OOP knowledge you have acquired in this course to build better software. In addition, your application should maintain state by saving and loading data from and to text-based files.

Project Requirements

Project Phases
The project has two phases:

1. Demo & Report
2. Implementation

Phase 1: Demo & Report (worth 20% of final grade)
Due during the week of submission: December 14, 15, 16 & 17, 2020.
After choosing the topic of the project, each team will need to prepare a document with requirements and description of the project. The document may borrow minor parts from this project document but needs to customize it as per the app developed.
During the submission week, each team will be allocated a time slot of 10 minutes to demo the developed application. For this meeting, each team should have the following:

1. A document with a brief description of the developed app.
    a. Development approach.
    b. Design of OOP classes.
2. A functional demo to show what they have worked. The app does not have to be 100% ready but there should be enough functionality for the demo.
3. UML class diagrams for each of the required classes.

Not demoing your work will result in automatic deduction of 20%.

How to prepare for the demo?

1. Be ready before your demo time by 5 minutes.
2. App loaded and ready to be tested.
   Prepare your own demo: a walk through the developed app to show features and how it works.
3. Visual Studio open with the code of the app loaded with all the files open. You will need to demonstrate class designs and any special features of the app in the code.
4. Document stated above open.

Note: Time yourself for the demo. Extra time will not be provided.

Phase 2: Implementation of the app (worth 80% of final grade)

Implementation is due December 17th, 2020.

Category of Evaluation	**Worth**
Proper use of OOP concepts.	30%

Functionality and code organization
• App does everything and works as expected.
• Exception handling: add try and catch where needed and provide proper error messages. App should not crash.
• Project Organization: keeping project files in different folders:
  ○ Models folder: for classes
  ○ Images folder: for images
  ○ Etc.
	30%

Appearance: UI Design	15%
Comments	5%

Proposed Topic (1)

Inventory Tracker

The **Inventory Tracker** is a simple application that allows groceries to track available stock of items in their stores. A grocery owner will add all available items in their store into the app (Item description is provided below). The owner should be able to update available quantities. The app should provide owners with information about all items. In addition, the app should provide a shopping list report for items that are not available in minimum quantities.

Item Class

Item Name	Name of the item
Available quantity	Quantity available in store. Value cannot be negative.
Minimum quantity	The minimum quantity that a store should always have at all times. Value cannot be less than 1.
Location	Provides information where the item is located in the store. Value: can be null. Example: Isle number 5.
Supplier	Provides information where to buy the item from. This could be a predefined list and provided through a dropdown menu. Value: can be null. Examples: Costco, Walmart, ABC, Other
Category	Create couple of categories in an enum. Examples: Pantry, Diary, Drinks, Frozen Food, Fruit & Vegetable, Bakery, Cleaning Supplies, Other

Inventory Class

Add Item	Adds item to a list of available items
Remove Item	Removes the item from the list
Update Item	Provide functionality to update all the Item data fields. Most important is to update available quantities.
GeneralReport	Provides a report that shows all items with available quantities and minimum quantities.
ShoppingList	Provided a report for items that need to be purchased because there is not enough quantity available. Example: Jam has available quantity of 3 and minimum quantity of 5, so jam should be part of the shopping list report. Pepper has available quantity of 5 and minimum quantity of 2, so pepper should not be part of the shopping list report.
LoadItems	A method to load items from a file(s) A text or csv file(s) can be used as you see fit.
SaveItems	Saves any changes made on the item list or the items.

Build a WPF GUI that will allow users to fulfill the above requirements. You will need to create several windows to achieve the requirements. Data can be transferred between different windows using the class constructor as shown in class demo.

Feel free to add any additional any classes or class members: data fields, properties, methods, setters and getters you think are required. Maintain OOP concepts as you design your classes.

Proposed Topic (2)

Jeopardy Game

The **Jeopardy Game** is a simple knowledge game that is played in teams. For simplicity you may assume that the game is play with 2 teams only. The teams are provided with a board showing a set of topics and points. First team selects a category and a point value. The teams are provided with a question related to the chosen topic. The first team who thinks they know the question should signal to answer. If their answer is correct, they are acquired the points for the question. If the answer is wrong, the team loses the points (subtracted from score) and the other team has a chance to answer to acquire the points. If the answer is wrong, the answer is revealed, and no one gets the points. Each answered question is removed (blocked). The next question (topic and point) is chosen by the winning team, the team with higher score or alternatively. The game finishes when there are no more questions available to answer. The team with the higher score is the winner.

A sample of the game can be found at: https://www.playfactile.com/csharpbasics/play
There are several other implementations that you could google to have an idea about your GUI design.

Here are some of the classes required for this app:

Question Class

Question	The question shown to the teams.
Answer	The answer to the question.
Point	Value points of the question.
Topic	Topic of the question.
Chosen	A Boolean value to indicate if the question was already chosen

Team Class

Team Name	The team name to be displayed when playing.
Score	The points acquired by each team. Value starts a 0 at the beginning of the game and may become negative during the game.

Game Class

Start Time	Indicates when the game started
Finish Time	Indicates when the game finished
Add Question	Used to add a question manually to the list of questions in the game.
Remove Question	Used to remove a question from the list of questions.
Load Questions	Used to load questions from a file
Save Questions	Save all available questions to a file of choice
Save Game	A method to save a game after it has finished to a file (log) Indicate game start and finish time, team names, scores and winning team.

The game admin should be provided with a window to add or remove questions to a list or load questions from a file. The players should have a different window showing the game board, where teams are added and the game starts. Data can be transferred between different windows using the class constructor as shown in class demo.

Feel free to add any additional any classes or class members: data fields, properties, methods, setters and getters you think are required. Maintain OOP concepts as you design your classes.

Hint

C# classes **File** and **Directory** offer many methods that provide several functionalities. These can ease implementation in your app.

- Reading and writing to files.
- Creating files and deleting files.
- Creating and deleting directories.
- Checking content of directories.

Requirements and Deliverables

Your submission must meet the following requirements:

1. **Updated Project Document**
    a. State all team members (Names and Student IDs) and the parts each member worked on.
    b. Update the document with all details provided by the teacher during the demo.
    c. Updated UML Class diagram.
    d. Optional sections (bonus):
        i. "Future Work" section: discuss features that can be added to application.
        ii. "App Snapshots": Take snapshots of the developed app in different states, add them to the document with explanatory titles or caption underneath.
2. **Developed App**
    a. Make sure to clean the solution: **Build** menu → **Clean Solution.**
3. **Zipp All Content**
    a. Requirement document.
    b. App solution folder.
    c. Demo presentation (if available) and any other support documents.
    DO NOT USE 7Zip or RAR files. (Assignment that do not use zip will not be corrected)
4. **Submit zipped file through LEA:** Do not send the file through MIO.
5. You may have one submission per team.