Week 16-17: Final individual assignment

This is the final individual assignment to give you better insight about your proficiency related to the covered concepts and the learning outcomes. Your teacher will give you a formative indication and feedback to help you understand your proficiency.

The assignment exists out a description, example data, list of requirements and an (incomplete) UML class diagram. You are required to implement an application based on these. The supplied class diagram is to give you a hint how the classes should be, but you will need to complete it with the actual class members (e.g., instance variables, methods, enums, etc.); i.e. completed till step 3.1.

Note that this assignment is individual and is used by your teachers to evaluate if you reached the expected proficiency for this semester. For this reason, it must be your own code and you are not allowed to copy any part of it from your fellow peers. Your teacher will give you feedback about what you submit and we assume that it is your own code and that you understand it fully.

Overview of assignments

- Library (page 1 & 3)
- Car sales (page 4 & 5)
- Work tasks (page 6 & 7)

Constraints

- You have to do one of the assignments and your teacher will tell you which one.
- You can use the supplied (incomplete) UML class diagrams as a guideline for your solution; note that you may also come with your own, sensible, class design
- Hand in your final version before the given deadline via Canvas.
- Your teacher(s) will schedule, in week 18, a meeting with you. During this meeting you can explain your submitted solution and you will receive formative feedback.

Deliverables
For this final individual assignment you are expected to submit in Canvas:

- Solution (i.e., source code) as a Windows Forms Application; make sure you submit everything so it can be opened in Visual Studio
- When present, UML class diagram
- When present, database diagram; you can get this from MS SQL Management Studio and do make sure all data is included, i.e., include the tables as a 'standard'-table view and the relationships.

Library

In this assignment you are tasked to create an application that allows a user to manage the books of a library, register who and when a book has been borrowed, and, if applicable, when it was returned.

Below you can find examples of book details[1]:

Title	Harry Potter and the Philosopher's Stone			
Author	J.K. Rowling			
Genre	Fantasy		*Pages*	352
Publication date	1 September 2014		*ISBN13*	9781408855898
Description	Harry Potter has never even heard of Hogwarts when the letters start dropping on the doormat at number four, Privet Drive. Addressed in green ink on yellowish parchment with a purple seal, they are swiftly confiscated by his grisly aunt and uncle. Then, on Harry's eleventh birthday, a great beetle-eyed giant of a man called Rubeus Hagrid bursts in with some astonishing news: Harry Potter is a wizard, and he has a place at Hogwarts School of Witchcraft and Wizardry. An incredible adventure is about to begin!			
Current borrower	*(nobody)*			
History of borrowers	*No.*	*Whom*	*Borrow date*	*Return date*
	10001	Donald Duck	1 September 2014	15 September 2014
	34571	Eve McFarm	12 June 2020	10 July 2020
	00004	John Doe	23 May 2021	30 August 2021

- or -

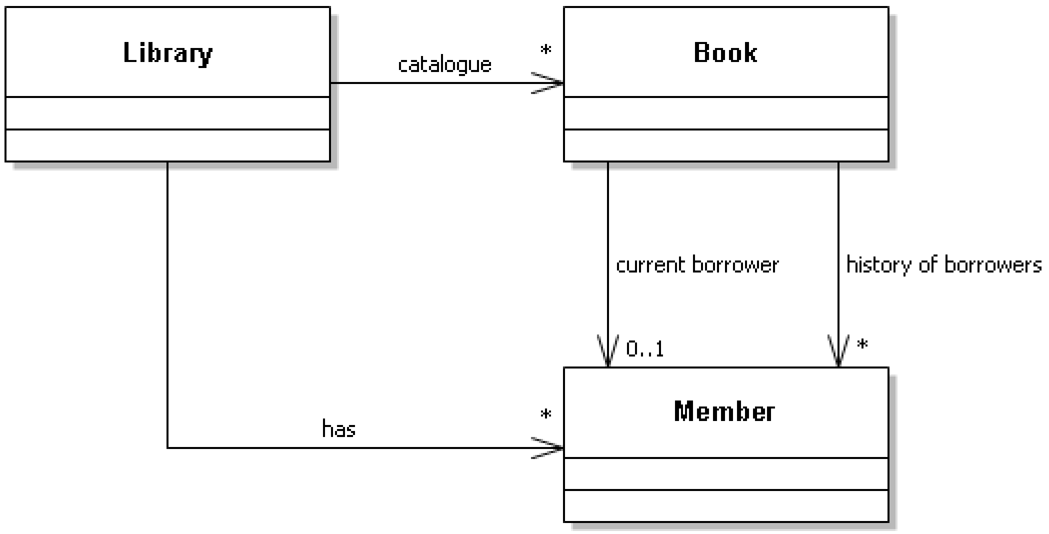
Title	The Da Vinci Code		
Author	Dan Brown		
Genre	Thriller	*Pages*	464
Publication date	18 March 2003	*ISBN13*	9780385504201
Description	While in Paris on business, Harvard symbologist Robert Langdon receives an urgent late-night phone call: the elderly curator of the Louvre has been murdered inside the museum. Near the body, police have found a baffling cipher. While working to solve the enigmatic riddle, Langdon is stunned to discover it leads to a trail of clues hidden in the works of Da Vinci -- clues visible for all to see -- yet ingeniously disguised by the painter. In a breathless race through Paris, London, and beyond, Langdon and Neveu match wits with a faceless powerbroker who seems to anticipate their every move. Unless Langdon and Neveu can decipher the labyrinthine puzzle in time, the Priory's ancient secret -- and an explosive historical truth -- will be lost forever.		
Current borrower	John Doe (no. 00004) on 28 April 2022		
History of borrowers	*(no history)*		

[1]: Note that you should not try to attempt to recreate the GUI like the examples; this is just a way of representing it.

With this app it should at least be possible to:

1. Add new books.
2. Search for and view a book's details; see the examples above
    - Search should be based on title, author, and/or genre; it should be possible to combine these (e.g., books with the substring *vinci* in the title and the genre *thriller*).
3. Remove books
4. Register when a book has been borrowed and by whom; you may assume that the library only has one copy of each book.
    - Take note that members can borrow multiple books (e.g., see John Doe with number 00004).
5. Register when a borrowed book has been returned.
6. Save and load all object data using a file
7. Optional: In addition to files, save and load all data from a database.
8. Optional: If you fancy an extra challenge, you can also assume a library has 1 or more copies of a book.

To help you towards the right direction you can use the (incomplete) UML class diagram as basis[2]:



[2]: This design does not consider the optional requirement (i.e., you might need to extend/change the class diagram to support those).

- End of Library assignment -

Car sales

In this assignment you are tasked to create an application that allows a user to manage the cars sold by a car dealership based on CSV imports, register car sales, and generate markdown files of sold cars.

Below you can find an example of an export of sold cars as markdown[1]:

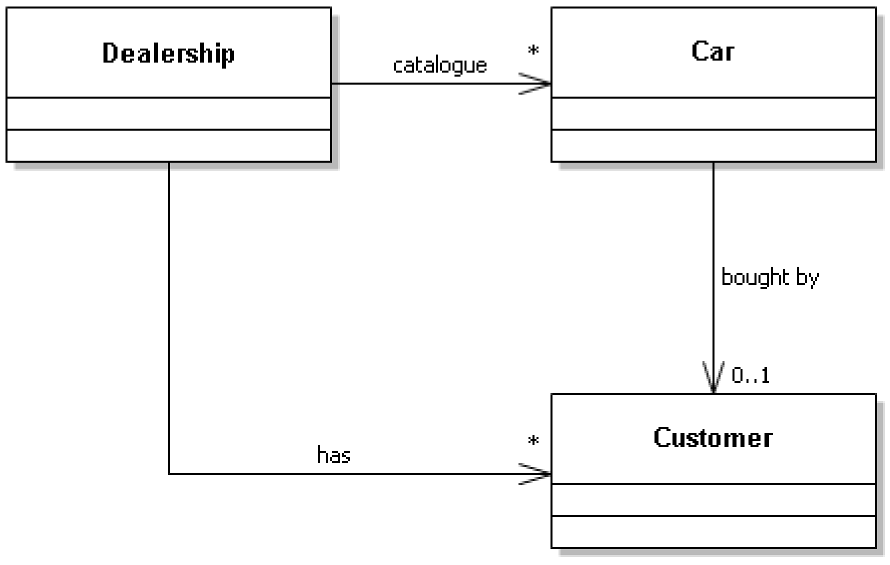
```
# 12 February 2022 | Cadillac Escalade ESV (2012)

- *Price*: 11511.57
- *Customer*: Eve McFarm
- *Phone no.*: 0624361890
- *Address*: Farlane 254b
- *Zip code & city*: 0925XD Farmwards

---

# 12 February 2022 | Porsche 928 (1995)

- *Price*: 57446.46
- *Customer*: Donald Duck
- *Phone no.*: 0401234567
- *Address*: Rachelsmolen 1
- *Zip code & city*: 5612MA Eindhoven

---

# 29 April 2022 | Maserati Spyder (2003)

- *Price*: 62057.81
- *Customer*: Eve McFarm
- *Phone no.*: 0624361890
- *Address*: Farlane 254b
- *Zip code & city*: 0925XD Farmwards

---
```

[1]: Markdown is, for example, used in GitLab to write styled text (e.g., readme.md, issues, etc.). You can use a markdown editor to see how the examples visually looks like; for example, https://dillinger.io. For more information about markdown see https://markdownguide.org

With this app it should at least be possible to:

1. Update available cars from a CSV file; the file that is imported represent the actual available cars; this means that if you:
   - First import *MOCK_CAR_DATA1.csv*: there are 50 new cars
2. If you then import *MOCK_CAR_DATA2.csv*: 3 new cars should be added, 35 cars stay the same, and 15 cars will be removed
3. Remove all cars and, if present, any connected sales data
4. Search for and view a car's details; see CSV files
   - Search should be based on brand, model, and/or price; it should be possible to combine these (e.g., cars with the substring *da* in the brand and the substring *lu* in the model).
5. Register car sales; see example above to determine what data is required and make sure a car can only be sold once
6. Export sold cars to a text file using markdown syntax; see examples above
7. Save and load all object data using a file
8. Optional: In addition to files, save and load all data from a database.
9. Optional: If you fancy an extra challenge, you can also include the possibility to add accessories when selling a car.

To help you towards the right direction you can use the (incomplete) UML class diagram as basis[2]:



[2]: This design does not consider the optional requirement (i.e., you might need to extend/change the class diagram to support those).

- End of Car Sales assignment -

Work tasks

In this assignment you are tasked to create an application that allows a user to manage task for their employees, the employees are imported from a CSV import, and see an overview of everything.

Below you can find examples of a task[1]:

Task	Lorem ipsum dolor sit amet, consectetur adipiscing elit				
Department(s)	■ Human Resources ☐ Support	☐ Research and Development ■ Marketing	■ Sales		
Status	Open	*Deadline*	27 July 2022	*By*	
Description	Vestibulum urna diam, feugiat tempus mollis sit amet, ornare ut elit. Aliquam vitae aliquam ligula. Ut a rutrum felis, a pellentesque quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque nisi ante, viverra quis dapibus quis, cursus sed nulla. Integer et magna velit. Cras luctus, arcu eget facilisis feugiat, nulla velit molestie turpis, pharetra hendrerit mi nisl sagittis ex. Integer sit amet scelerisque turpis, non molestie ex. In at nunc dolor. Sed sit amet urna lorem. Vestibulum elit purus, tincidunt vel ex in, placerat volutpat est. Mauris vulputate ornare ipsum vitae dapibus. Nam fringilla tellus et mi ornare scelerisque. Pellentesque tristique laoreet facilisis. Curabitur vel pulvinar ante. Sed at ipsum ac velit viverra pretium id a nibh.				

- or -

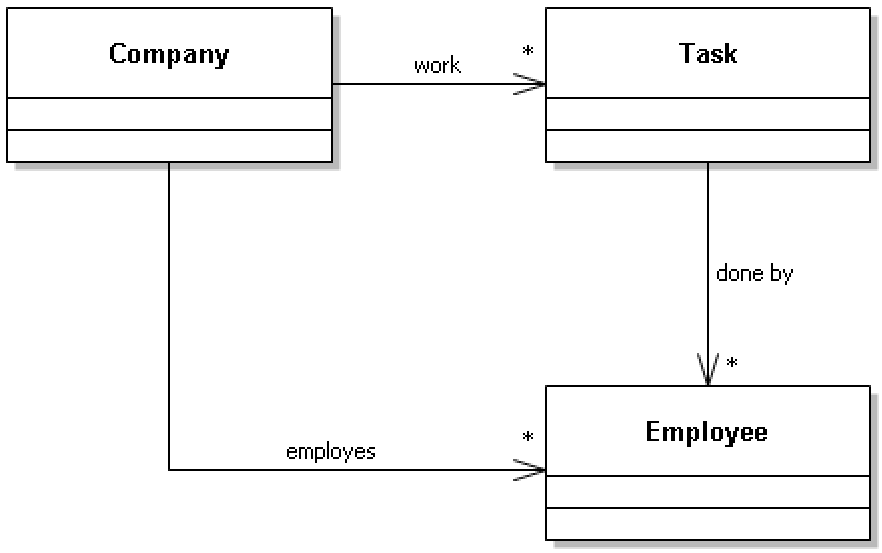
Task	Sed ac sem tempor, imperdiet ex ac, lobortis nisi				
Department(s)	☐ Human Resources ☐ Support	■ Research and Development ☐ Marketing	☐ Sales		
Status	Completed	*Deadline*	19 May 2022	*By*	Joséphine Newarte Angélique Sinncock
Description	Curabitur justo turpis, consectetur et porttitor non, mollis eget nisl. Nunc fermentum neque ut nisl finibus, eget ornare metus commodo. Curabitur elementum id nisi eu vestibulum. In sit amet luctus dui, ut tincidunt mi. Phasellus nisi nulla, porta sit amet egestas ac, tincidunt at enim. Suspendisse eu sapien eget urna sollicitudin condimentum vitae sit amet ipsum.				

[1]: Note that you should not try to attempt to recreate the GUI like the examples; this is just a way of representing it.

With this app it should at least be possible to:

1. Import employees of the five (5) department from the CSV file *MOCK_EMPLOYEE_DATA.csv*
   - It has: 14 Human Resources employees, 16 Marketing employees, 13 Sales employees, 15 Support employees, and 10 Research and Development employees.
2. Create new tasks; see example above to determine what data is required
   - The possible statuses are: *Open*, *In Progress*, *Completed*, *Blocked,* and *Cancelled*
3. Remove tasks; this can only be done for tasks with the status *Open* and nobody has been assigned to it.
4. Update a by assigning an employee(s) to a task and/or changing its status; note that only employees with a matching department can be assigned to a task.
5. View a task via a dashboard where tasks can be searched/filtered based on: (part of) title, status, and/or department; it should be possible to combine these (e.g., tasks the substring *ac* in the title from the department *Support*).
6. Save and load all object data using a file
7. Optional: In addition to files, save and load all data from a database.
8. Optional: If you fancy an extra challenge, you can also include the possibility for an employee to login and see the tasks assigned to her/him. This employee should only be able to view her/his tasks and change its status.

To help you towards the right direction you can use the (incomplete) UML class diagram as basis[2]:



[2]: This design does not consider the optional requirement (i.e., you might need to extend/change the class diagram to support those).

- End of Work Tasks assignment -