Week 9

Learning objectives

You will be able to apply object-oriented programming using the Python programming language:

- Create full dataclasses
- Use of class attributes & class methods (static methods)
- Integration of association between two classes
- File I/O (Json-files)
- Integrate exception handling

GitHub

All your solutions from week 9 should be posted on Github. To do so, follow the procedure explained on Leho. After each exercise, you can do a 'commit & push' so that your version on GitHub is always updated. Give an appropriate message each time.
Continue to work on unfinished exercises at home: regularly perform a 'push & commit' command so that all your solutions are available in your online GitHub repository.
When mastering a programming language such as Python, frequent practice is essential and a prerequisite. For this reason, you will find two additional sections in each lab document. These are labelled as follows

 Further reading – Doing your own research

This section goes beyond what you have seen this week. Often the exercises are just a little more difficult than what you did in the lab. You will need the Python documentation and Google for this investigation.
In this section you will research a topic that will be covered in theory or lab in the following weeks. These exercises you should also make.

Homework

In this section you will find exercises similar to those you have already done in the laboratory. These exercises have the same level of difficulty as the ones you did in the lab. It is only by doing the exercises "on your own" at home that you will consolidate the material.
Stuck on an exercise? Check the theory and see if you can find a similar exercise you did in the lab. Still not succeeding? Bring your preparation to the Study Morning!

Feedback lab week 8

 Find and solve the common errors in Shoppingcart.py.

Exercises

In your comments, always mention the number of the exercise!

Exercise 01

Find the class **Beer** from lab week 6 (or use the class Beer from the source material). Now modify this class:

- in each setter property, a check is made on the new value:
  - Beer name, brewery name, color: new value is a string and must not be empty;
  - Alcohol percentage: the new value is a float and lies between 0 and 100.

  If the new value does not match, a ValueError is returned.
- an object of the Beer class is correctly printed in a list.

Test your code by creating a correct Beer object. Then change the beer name to an empty string.

Use now **exception handling** in your test-code so the application does not crash.

```
CTAIBlond Howest - 25.0
Enter a new beername:>
Error: No valid beername!
```

```
Enter a new name:> CTAI
Enter a new brewery:> Howest
Enter a new color:> Brown
Enter a new alcoholpercentage:> -55
Making beer...
Error: No valid alcoholpercentage!
```

Exercise 02

Explore the file **beers.json** in your projectmap. What can you recognize?

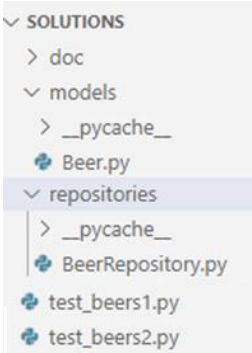
As you discussed in the theory lesson, the structure of a json file will always be different. Sometimes the structure is a list[], sometimes it is a dictionary{}.

The main objective of this exercise is to 'create a object of our class Beer' for each beer described in the json-file. We keep track of all created objects in a list. In the next step, we will then add some additional filtering functionality like filtering beers based on a given brewery name.

Step 1: Preparation

a) Add the json-file in the subfolder 'doc'.

b) In a new folder called 'repositories', add a new file '**BeerRepository.py'**. In it, create a new class **BeerRepository**. This class is responsible for processing data related to beers: reading files, filtering data, performing specific searches, etc.



Step 2: Read the beers

Add a public static method 'load_beers' in the class BeerRepository that can read the source file and return a list of beers.

- At the top, import the json library so that you can easily query a json file:
  `import json`

- Use the private static helper method '__read_local_json_file'. Copy this method from the source code or from the demos out the theory lesson. In this example, the helper method returns a list []: each element is a dictionary {}.

- Add the public static method **'load_beers'**. This method will further process the data from the json file.

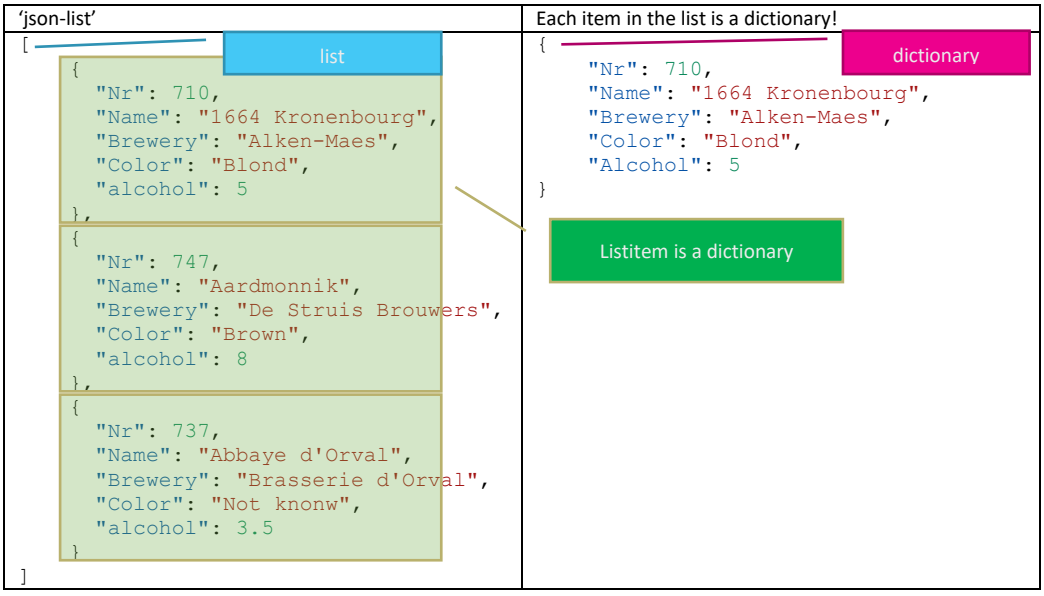
BierRepository.py

```
from models.Beer import Beer
import json
from typing import List

class BeerRepository:

    @staticmethod
    def __reading_local_json_file(filename: str):
        fo = open(bestandsnaam)
        response_json = fo.read()
        fo.close()
        return json.loads(response_json)

    @staticmethod
    def load_beers(filename: str) -> List[Beer]:
        result = []
        list_beers = BeerRepository.__reading_local_json_file(filename)
        for dict_beer in list_beers:
            pass
            #....
        return result
```

Loop over each element in the 'json-list': each element is represented as a dictionary with associated (key,value) pairs. Query each value with the appropriate key. With all collected info create now an object of your class Beer.

Remark:

- *the keys are case sensitive.*
- *Use exception handling when you go through the list: not every value is correct.*



Step 3: **filter the beers**

- Make sure that the list of beer objects can be sorted from small to large based on alcohol percentage.
    - *Which operator should you overload for this? In which class?*

- Add a public static method 'filter_beers_brewery' with parameters a list of Beer class objects and a brewery name. Loop through all the beers and return only those from the given brewery.

BierRepository.py

```
from models.Beer import Beer
import json
from typing import List

class BeerRepository:

    @staticmethod
    def filter_beers_brewery(list_beers:List[Beer], searched_brewery:str) -> List[Beer]:
        result = []
        for beer in list_beers:
            #...

        return result
```

Exercise 03

For this exercise we will use a json file '**highways.json**'. The json file returns some information about traffic jams on Belgian highways on 28 November.

Study the json file carefully and complete the following statements.
- *The root of the json file is* ..Dictionary.....
- *It consists of* ..List..... *of* ....dictionaries.....
- *Each* ..Dictionary..... *represents one* ...HighWay.....

- *Under which key are the files?* ..Traffic Jams.....
- *This is made up of* ..a List..... *of* ..Dictionaries.....
- *Each* .....Key..... *represents one* ...record.....

For this exercise, we only want to keep the following information for each highway
- ID
- total length
- traffic intensity
- year of construction
- number of lanes
- traffic Jams

For each traffic jam, we only keep the following information
- Location
- Length in km
- timestamp
- cause
- weather conditions

We read the json file and a list of highways will returned. A list of traffic jams is kept in each highway.

Part 1: Preparation

Place the json file in a separate subfolder called 'doc'.

Now program all the data classes in the 'model' folder first:

(a) Class '**TrafficJam**'. Provide the class with a constructor, properties, tostring method, repr method, eq method. Provide the necessary checks in the setters (including exception handling). Use the datetime datatype for the time. Output when printing a TrafficJam object: see test code below.

Tip: To convert the time to a string, do the following
```
self.time.strftime('%d/%m/%Y %H:%M') #vb: 01/12/2024 22:50
```

(b) Create the class '**Highway**'.
- Give the class a constructor, properties, tostring method, repr method, eq method. Provide the necessary checks in the setters (including exception handling).
- For the traffic jams, create an empty list in the init method. For this attribute you provide only a get property.
- For the other attributes, provide a get&set property.
- Add a method 'add_traffic_jam': this method has one parameter, which is an object of the TrafficJam class. This method adds the object to the list of traffic jams.

    - In the method, check that the parameter is indeed an object of the TrafficJam class.
    - Check if the parameter does not already exist in the list.
      Which method of the class is used for this?
- Output when printing an object: see below.
- Add an extra print method print_traffic_jams, which will print all the traffic jams of the highway one below the other.

Now test your data classes first: create an object of the Highway class. Also create and add some traffic jams to the highway.

test_highway.py
highway_E403 = Highway("E403",60.5,50000,1985) trafficjam1 = TrafficJam("Roeselare",2.5,datetime.now(),"Accident","Fog") trafficjam2 = TrafficJam("Brugge",3.7,datetime.now(),"Roadworks on the highway","Sunny") highway_E403.add_traffic_jam(trafficjam1) highway_E403.add_traffic_jam(trafficjam2) print(highway_E403) print() highway_E403.print_traffic_jams()
Terminal E403 (construction year: 1985, number of lanes: 2, total length: 60.5) registered traffic jams on E403 are: Traffic jam at Roeselare - 02/12/2024 23:48 (length: 2.5 km, cause: Accident) Traffic jam at Brugge - 02/12/2024 23:48 (length: 3.7 km, cause: Roadworks on the highway)

Part 2A: Reading the highways from the json-file

a) Add a new file **HighwayRepository**.py in a new 'repositories' folder. Inside create a new class HighwayRepository. This class will be responsible for processing the data from the json file: reading the file, filtering on the data, specific searches, etc.

b) Import the json library:
import json

c) Use the private static helper method '__read_local_json_file'. Copy this method from the previous exercise.

d) Add the public static method '**load_highways**'. This method will process the data from the json-file.

Note: How can you retrieve the list of highways from the json file? What is each item in this list?

Each element contains information from one highway. Now loop over all the elements. Each element is a dictionary from which you can retrieve specific information.

Now test this first version of load_highways: print the full list of highways.

test_highway.py
list_highways = HighwayRepository.load_highways("doc/highways.json") print(list_highways)
Terminal [E40, E17, E411]

Part 2B: Reading the traffic jams from the json-file

Now modify the load_highways method internally so that the traffic jams belonging to a highway also been read and processed. Each time, an object of the TrafficJam class is created and added to the highway (use the add_traffic_jam method).

Explore the json-file: how will you access the traffic jams? What does this list consist of?

Hint: converting a string to a datetime:

```
timestamp = datetime.fromisoformat(dict_traffic_jam["time"])
```

Now test this new version of load_highways: print the complete list of highways. Loop over each highway and also print now the traffic jams.

test_highway.py

```
# Part 2A
list_highways = HighwayRepository.load_highways("doc/highways.json")
print(list_highways)

# Part 2B
for highway in list_highways:
    highway.print_traffic_jams()
    print("*****************************************")
```

Terminal

```
[E40, E17, E411]
registered traffic jams on E40 are:
      Traffic jam at Ternat - 28/11/2024 08:30 (length: 5.2 km, cause: Accident)
      Traffic jam at Zwijnaarde - 28/11/2024 17:45 (length: 3.1 km, cause: Heavy traffic)
****************************************
registered traffic jams on E17 are:
      Traffic jam at Lokeren - 28/11/2024 09:15 (length: 7.4 km, cause: Roadworks)
      Traffic jam at Deerlijk - 28/11/2024 18:20 (length: 4.3 km, cause: Accident)
****************************************
registered traffic jams on E411 are:
      Traffic jam at Waver - 28/11/2024 07:50 (length: 6.5 km, cause: Heavy traffic)
      Traffic jam at Arlon - 28/11/2024 16:10 (length: 2.8 km, cause: Roadworks)
****************************************
```

Part 3A: Filter the highways

We want to select only the very busy highways from our list. So add a public static method **filter_very_busy_highways** to the HighwayRepository class. This method has the read list as its only parameter. The method returns only those highways whose traffic volume is greater than 100000 cars per day. Try it out!

test_highway.py

```
# Part 3A
busy_highways = HighwayRepository.filter_very_busy_highways(list_highways)
print("\nThe very busy highways are those with a minimum of 100.000 vehicles per day.")
print(f"These are: {busy_highways}")
```

Terminal

```
The very busy highways are those with a minimum of 100.000 vehicles per day.
These are: [E40, E17]
```

Part 3B: Filter the traffic jams

We want to collect all traffic jams with a certain minimum length. Therefore, we add the public static method **filter_long_traffic_jams** to the HighwayRepository class. This method has two parameters: the

list of highways and a minimum length. The method loops over all the highways and returns only those traffic jams whose minimum length is greater than the specified length. Try it out!

test_highway.py

```
min_length_traffic_jam = float(input("\nEnter a minimum lenth of traffic jam:> "))
print(f"The traffic jams that meet this criterion are: ")
searched = HighwayRepository.filter_long_traffic_jams(list_highways, min_length_traffic_jam)
for traffic_jam in searched:
    print(traffic_jam)
```

Terminal

```
Enter a minimum lenth of traffic jam:> 5
The traffic jams that meet this criterion are:
Traffic jam at Ternat - 28/11/2024 08:30 (length: 5.2 km, cause: Accident)
Traffic jam at Lokeren - 28/11/2024 09:15 (length: 7.4 km, cause: Roadworks)
Traffic jam at Waver - 28/11/2024 07:50 (length: 6.5 km, cause: Heavy traffic)
```