School of Engineering

Applied MSc in Data Science & Artificial Intelligence

Applied MSc in Data Engineering & Artificial Intelligence

Course: **Software Engineering Part 2**

Project: **OrderBook**

Instructor: **Hanna Abi Akl**

Project Objectives:

The project requires you to use Python Object-Oriented Programming (OOP) knowledge to create the classes for an application that collects all tasks for a software company. This will let you further practice working on objects which contain references to other objects.

The project is constructed in "Parts" that are ordered and serve as a guide towards the final implementation.

Please note that this is an individual project and any suspicion in similarities between solutions will automatically result in a failing grade for the students involved. ChatGPT-like

AI-generated solutions will also be penalized (believe me, I can tell when they write your code 😊).

Part 1: Task

Please create a class named Task which models a single task in a software company's list of tasks. Tasks have:

- A description
- An estimated number of hours for completion
- The name of the programmer assigned to it
- A status field to indicate if it is finished
- A unique identifier

You can use the following code to test your class:

```
t1 = Task("program hello world", "Eric", 3)
print(t1.id, t1.description, t1.programmer, t1.workload)
print(t1)
print(t1.is_finished())
t1.mark_finished()
print(t1)
print(t1.is_finished())
t2 = Task("program webstore", "Adele", 10)
t3 = Task("program mobile app for workload accounting", "Eric", 25)
print(t2)
print(t3)
```

Your program should print out this:

Sample output

```
1 program hello world Eric 3
1: program hello world (3 hours), programmer Eric NOT FINISHED
False
1: program hello world (3 hours), programmer Eric FINISHED
True
2: program webstore (10 hours), programmer Adele NOT FINISHED
3: program mobile app for workload accounting (25 hours), programmer Eric NOT
FINISHED
```

Some clarifications:

- The state of a task (finished or not finished) can be checked with a function that returns a Boolean value
- A task cannot be finished when it is created
- A task should be marked finished by calling a method

- The Id of a task is a running number starting with 1, i.e. the first task has id 1, the second task has id 2, and so forth.

Part 2: OrderBook

Please write a class named **OrderBook** that collects all the tasks ordered from a software company. The tasks should be created using the **Task** class you implemented in Part 1.

Your class should contain the following methods:

- A method to add a new order to the OrderBook. The OrderBook stores the orders internally as Task objects. The method should take a *description*, a *programmer* and a *workload* as arguments.
- A method that returns a list of all the tasks stored in the OrderBook.
- A method that returns a list of the names of all the programmers with tasks stored in the OrderBook. The list should contain each programmer only once.

The basic version of your class should work as follows:

```
orders = OrderBook()
orders.add_order("program webstore", "Adele", 10)
orders.add_order("program mobile app for workload accounting", "Eric", 25)
orders.add_order("program app for practising mathematics", "Adele", 100)

for order in orders.all_orders():
    print(order)

print()

for programmer in orders.programmers():
    print(programmer)
```

Executing the above should print out:

Sample output

```
1: program webstore (10 hours), programmer Adele NOT FINISHED
2: program mobile app for workload accounting (25 hours), programmer Eric NOT
FINISHED
3: program app for practising mathematics (100 hours), programmer Adele NOT
FINISHED

Adele
Eric
```

Part 3: Listing task owners more efficiently

In Part 2, you were asked to implement a method to return a list of programmers that are task owners. The previous implementation is not efficient since it only prints out the names of the programmers without knowing who works on what. Implement a new method that returns a dictionary of key-value pairs, with keys corresponding to the names of the different programmers and values corresponding to the list of task identifiers that are assigned to them.

Part 4: Additional features for OrderBook

Please add the following methods to your **OrderBook** class definition:

- A method that takes the id of the task as its argument and marks the relevant task as finished. If there is no task for the given id, the method should raise a ValueError exception.
- A method that returns a list of the finished tasks from the OrderBook.
- A method that returns a list of the unfinished tasks from the OrderBook.

Your class should now work with the following program:

```
orders = OrderBook()
orders.add_order("program webstore", "Adele", 10)
orders.add_order("program mobile app for workload accounting", "Eric", 25)
orders.add_order("program app for practising mathematics", "Adele", 100)

orders.mark_finished(1)
orders.mark_finished(2)

for order in orders.all_orders():
    print(order)
```

Executing the above program should print out this:

Sample output

```
1: program webstore (10 hours), programmer Adele FINISHED
2: program mobile app for workload accounting (25 hours), programmer Eric
FINISHED
3: program app for practising mathematics (100 hours), programmer Adele NOT
FINISHED
```

Part 5: The finishing touches

Please add a new method to your **OrderBook** class: **status_of_programmer** should take the name of a programmer and return a tuple. The tuple should contain the number of finished and unfinished tasks the programmer has assigned to them, along with the estimated hours in both categories. The first item in the tuple should be the number of finished tasks, the second item should be the number of unfinished tasks, and the third and fourth items should be the sums of workload estimates for the finished and unfinished tasks respectively. If a name is given for a programmer that does not exist, the method should raise a ValueError exception.

Your class should now work with the following program:

```
orders = OrderBook()
orders.add_order("program webstore", "Adele", 10)
orders.add_order("program mobile app for workload accounting", "Adele", 25)
orders.add_order("program app for practising mathematics", "Adele", 100)
orders.add_order("program the next facebook", "Eric", 1000)

orders.mark_finished(1)
orders.mark_finished(2)

status = orders.status_of_programmer("Adele")
print(status)
```

Executing the above program should print out this:

Sample output

```
(2, 1, 35, 100)
```

Part 6: Putting it all together

Using the classes you implemented, write a program that for administering the tasks ordered from a software company. Your program should consist of a main function that accepts the following commands with their corresponding arguments:

- **0**: exit
- **1**: add order
- **2**: list finished tasks
- **3**: list unfinished tasks
- **4**: mark task as finished
- **5**: programmers
- **6**: status of programmer

An example behavior is shown below. To make the implementation easier, your program can accept each command with its arguments as a list of arguments to a method for example but should give the same behavior as the example shown. Note that the example shows the different inputs in every block along with the expected output on the last line of the block.

```
command: 1
description: program the next facebook
programmer and workload estimate: jonah 1000
added!

command: 1
description: program mobile app for workload accounting
programmer and workload estimate: eric 25
added!

command: 1
description: program an app for music theory revision
programmer and workload estimate: nina 12
added!

command: 1
description: program the next twitter
programmer and workload estimate: jonah 55
added!

command: 2
no finished tasks

command: 3
1: program the next facebook (1000 hours), programmer jonah NOT FINISHED
2: program mobile app for workload accounting (25 hours), programmer eric NOT
FINISHED
3: program an app for music theory revision (12 hours), programmer nina NOT
FINISHED
4: program the next twitter (55 hours), programmer jonah NOT FINISHED

command: 4
id: 2
marked as finished

command: 4
id: 4
marked as finished

command: 2
2: program mobile app for workload accounting (25 hours), programmer eric
FINISHED
4: program the next twitter (55 hours), programmer jonah FINISHED

command: 3
1: program the next facebook (1000 hours), programmer jonah NOT FINISHED
3: program an app for music theory revision (12 hours), programmer nina NOT
FINISHED

command: 5
jonah
eric
nina

command: 6
programmer: jonah
tasks: finished 2 not finished 1, hours: done 55 scheduled 1000
```

Part 7: Handling errors in user input

To make the application better, you are required to make it recover from erroneous user input. Any input which does not follow the specified format should produce The error message *erroneous input*.

Sample output

```
command: 1
description: program mobile app for workload accounting
programmer and workload estimate: eric xxx
erroneous input

command: 1
description: program mobile app for workload accounting
programmer and workload estimate: eric
erroneous input

command: 4
id: 1000000
erroneous input

command: 4
id: XXXX
erroneous input

command: 6
programmer: unknownprogrammer
erroneous input
```

Project Deliverables:

Each student must submit the following files:

● A **classes.py** script: The script containing the class implementations.

● A **main.py** script: The script containing the same test code for class objects as shown in the examples as well as the final application program code to test Parts 6 and 7.

Project Evaluation:

The project will be evaluated using the following rubric:

● Part 1 **[10 point]**

- Part 2 **[20 point]**
- Part 3 **[10 point]**
- Part 4 **[20 point]**
- Part 5 **[10 point]**
- Part 6 **[20 point]**
- Part 7 **[10 point]**