UNIVERSITEIT VAN AMSTERDAM 2018-2019

minor programmeren

Oefententamen Programmeren 2

lente 2019

Vul hier je naam en studentnummer in vóór je begint:	/ 21 p

1. Je mag de vragen in Engels of Nederlands beantwoorden.
2. Dit is een "gesloten boek"-tentamen. Je mag voor het invullen je pen of potlood gebruiken, maar verder niets. Schrijf duidelijk en niet te groot.
3. Leg je studentenkaart (of ander ID met foto) klaar op je tafel. We komen langs om te kijken of je hierboven je naam hebt ingevuld en of deze klopt met je ID.
4. Laat het weten als je kladpapier nodig hebt.
5. Als je vragen hebt over hoe we iets bedoelen, dan kunnen we dat waarschijnlijk niet beantwoorden zonder een deel van het antwoord weg te geven (maar voel je vrij om het te proberen!).
6. Je hoeft geen comments in je code te schrijven.

Football manager

Consider the following class. It is a simple *data class*, used to represent the results of a single football match.

```
class MatchResult():
    def __init__(self, club1, club2, goals1, goals2):
        # TODO
    def won(self):
        """determine which club has won the match"""
        # TODO
    def __str__(self):
        # TODO
```

Additionally, we will define a class that's able to calculate the number of wins for one club in a series of football matches.

```
class StatsCalculator:
    def __init__(self):
        self.results = []
    def add(self, match_result):
        """add given MatchResult to results list"""
        # TODO
    def get_stat(self, club_name):
        """calculate num of wins for specific club"""
        # TODO
```

And here is some testing code, which should tell you how the class is supposed to be used:

```
if __name__ == "__main__":
    calc = StatsCalculator()
    calc.add(MatchResult("ADO", "FEY", 3, 2))   # ADO won
    calc.add(MatchResult("HEE", "NEC", 1, 4))   # NEC won
    calc.add(MatchResult("AJA", "VIT", 0, 2))
    calc.add(MatchResult("HER", "AJA", 0, 2))
    calc.add(MatchResult("TWE", "AJA", 1, 1))
    calc.add(MatchResult("AJA", "AZ",  3, 2))
    print(calc.get_stat("AJA"))                 # should print: 2
```

1p **Question 1.**

Implement the `__init__` method for `MatchResult`, saving all incoming data as attributes.

```
def __init__(self, club1, club2, goals1, goals2):
```

1p **Question 2.**

Implement the `won` method for `MatchResult`. Simply return the name of the winning team.

```
def won(self):
```

2p **Question 3.**

Implement the `__str__` method for `MatchResult`. The string should look like: `ADO 3 - 2 FEY`

```
def __str__(self):
```

2p **Question 4.**

Implement the `add` method for `StatsCalculator`, saving the object to the `results` list.

```
def add(self, match_result):
```

3p **Question 5.**

Implement the `get_stat` method for `StatsCalculator`.

```
def get_stat(self, club_name):
```

Design challenge

6 p **Question 6.**

Model the following system with a class diagram: a network of vending machines.

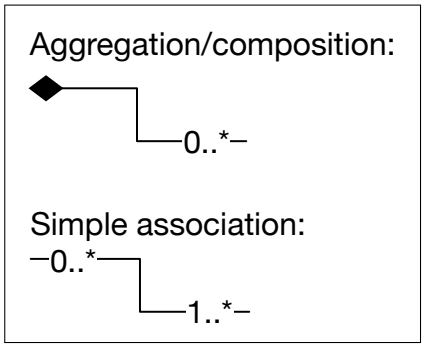
Some provider offers vending machines to universities, so people can buy things like chocolate bars, cookies and candies. Each item has a price and a name. Each university has a separate network of vending machines. The network includes smart cards to pay at the vending machines: these cards are linked to a certain amount of money that the owner has. Smart cards only work within a single university for the vending machines at that university.

The functions supported by the system are:

- Vending machines can be setup to contain different kinds of items
- Machines can be restocked by a service agent
- Vending machines can sell an item to a smart card owner
- Users can recharge their smart card
- Machines can be monitored remotely (get number of items sold, number of items sold per type, total revenue)

Draw a class diagram for this system. We're only interested in the core objects that are in this system. No need to specify how people interact with the system components. Clearly specify **associations** between related classes, and specify **methods** and **attributes** that would be needed to implement the functionality as described. The diagram will be graded on clarity and thoroughness.

As a reminder, here are the associations in UML class diagram notation that you might use:



Data structure

In many languages you may encounter a **set** type. An important idea of a set is that is cannot contain duplicate elements. For example, if you create a report of cities that you have visited, it need not contain duplicates. You could collect those in a set instead of a list.

Here's part of a `Set` class for Python. You could just create a new set using `Set()`, but you can also create a set from an existing list: `Set([4,3,4,9])`.

```
class Set():
    # initalize set from existing list
    def __init__(self, from_list):
        self.the_set = []
        for element in from_list:
            if element not in the_set:
                self.the_set.append(element)

    # get the number of elements in the set
    def size(self):
        return len(self.the_set)
```

3 p **Question 7.**

Implement the `add` method for `Set`, which can be used to add a new element to the set. It should not allow duplicate elements to be added!

```
def add(self, element):
```

3 p **Question 8.**

Implement the `filter` method for `Set`. It should keep only the elements that are not in the given list (thus, it removes any elements from the set that are in the given list).

```
def filter(self, the_list):
```