Assignment for Class Computer Science for Big Data C1

Structured Data and Object-oriented Programming in Python, Points: 10

Object-oriented programming (OOP) uses *classes* to define new data types. It is useful to model real-world concepts and "objects" as programming constructs. It helps organizing code by assigning functions to classes and encapsulate state (data) in objects that are created from classes at run-time.

See the Python course (link in Moodle, chapter *(3:43:56) Classes & Objects*) for an introduction to OOP in Python.

Structured Data refers to data that is represented in a predefined static structure called *schema*. Data is often stored in relational databases (RDBMS) that use tables with columns as schema and rows to store data objects as rows in the table.

The UML class diagram below (see https://en.wikipedia.org/wiki/Class_diagram) defines the data structure of a simple order processing system with classes and relations between classes.

The open (white) diamond describes an **aggregation** relationship of a customer "owning" orders (* or 0..* for any number of orders, including no order). Aggregation means there is no "list of orders" in the Customer class. The relation is represented as a (foreign key) reference in the class Order back to the owning customers. In contrast, **composition** with a black diamond expresses a "part_of" relationship between classes Order and OrderItem. Each order contains at least one (1..*) ordered item in its list-attribute: *items[OrderItem]*. Class OrderItem then contains no information about the relation.

Within classes, the first section defines *attributes* holding data in objects of that class. The minus sign (-) indicates that attributes are private and encapsulated inside the class and cannot directly be accessed or altered from outside. Plus (+) refers to public attributes, which can directly be accessed from outside.

The second section defines functions (methods) that provide operations for the class. Getter-functions return values of private attributes, e.g. *get_id()*. Setter-functions control write access to attributes.

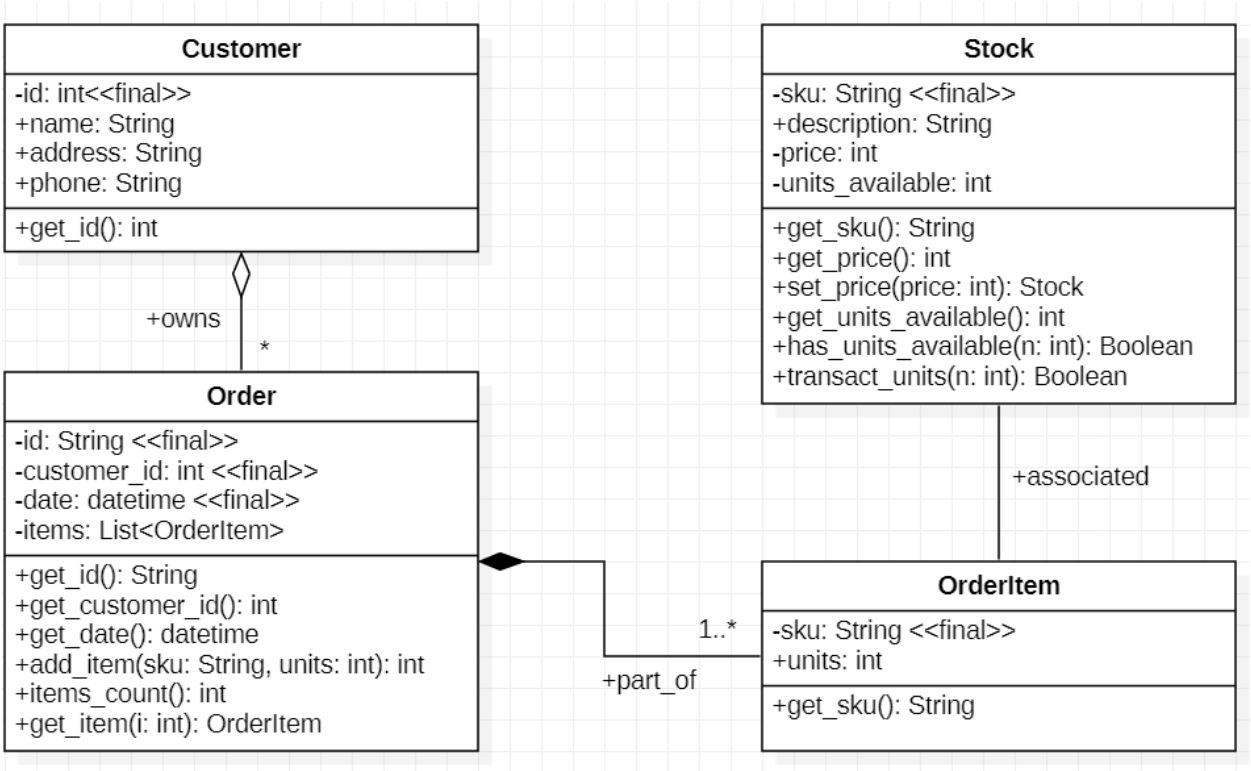
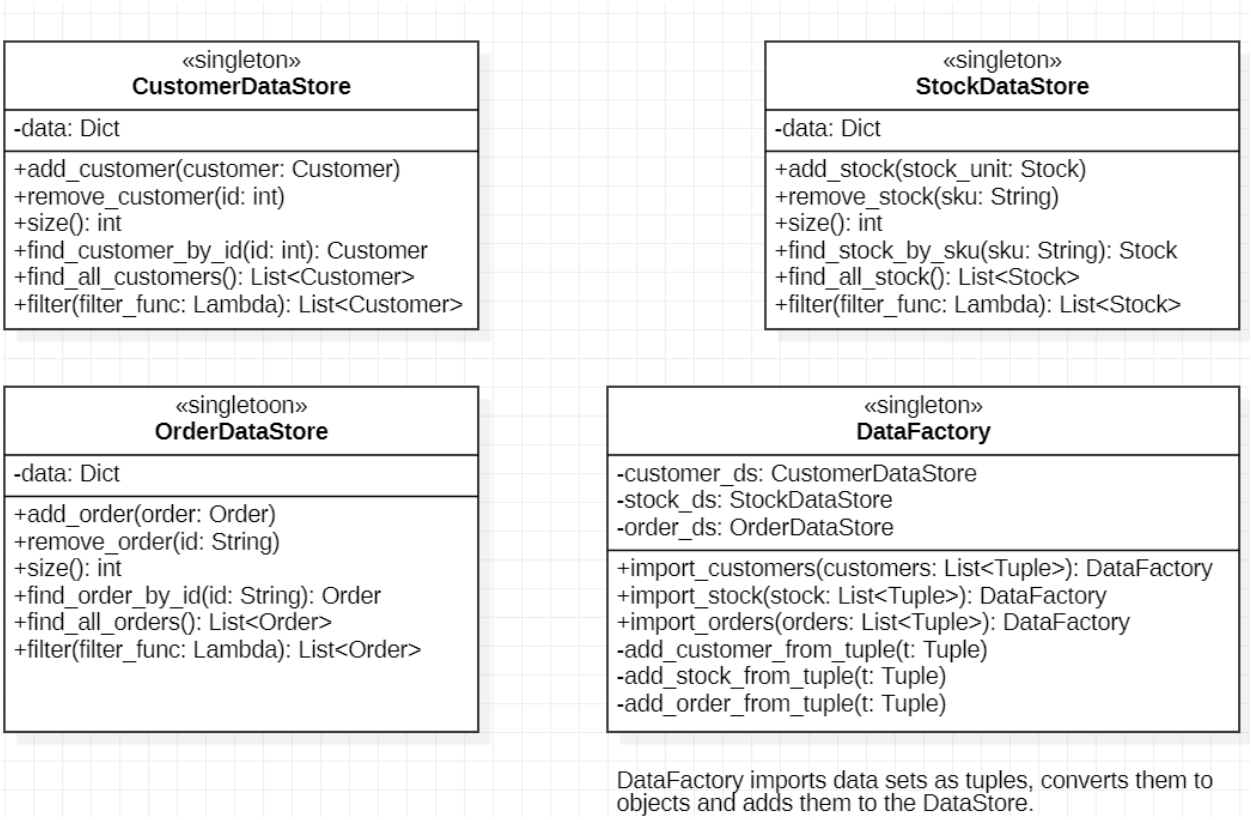


Fig 1: Data model with four classes and relations for a simple order processing system.

Data Stores contain and provide access to objects. The following diagram defines classes for three data stores. In contrast to business classes (defining the structure of data relevant for a business such as Customer, Order and Stock), data store objects exist only as singular objects, which is indicated by the <<singleton>> stereotype in the diagram. Data store methods allow adding and removing objects to and from the store and execute queries for finding or filtering objects.

DataFactory is a special class that facilitates data import (from tuple lists in this case), convert data into business objects and store them in one of the data stores.



1.) Clone the project from https://github.com/sgra64/C1_order_system.git or download the project as *C1_order_system.zip* from Moodle and set up a new project in your IDE. [1 Pts]

Inspect its packages:

- *data* – contains sample data sets for customer, stock and order data.
- *datamodel* – contains classes *Customer*, *Stock* and *Order* (which includes class *OrderItem*),
- *datastore* – with classes for *DataFactory* and the classes for data stores

2.) Understand the *data sets* in the *data* package.

3.) Understand the role of the top-level *__init__.py* file and answer questions:

a) How long do *DataStore* objects exist that are referenced by *cds*, *sds* and *ods* variables?

b) What could be the reason that no global reference variable was used for *DataFactory*?

c) What enables function chaining in *DataFactory's* import-functions? [1 Pts]

4.) Run the driver code *C1_driver.py* in your IDE or run in a terminal: `python C1_driver.py`.

The code is still incomplete, only 1000 customers are properly imported, but no stock data and no orders, yet. *CustomerDataStore* accessed by variable `cds` is already properly working. Therefore, query:

```
_last_name = "Cantrell"
_filtered_by_name = cds.filter(lambda c: last_name_func(c) == _last_name)
```

returns a set of five customers with that name as shown in the expected output:

```
--> 1000 customers, 0 stock items, 0 orders loaded.
0 orders in OrderStore
1000 customers in CustomerStore
customer 898179: Joan Hendricks
--> find customers with last name 'Cantrell':
 - Rudyard Cantrell, 6424 Simms Street, #71, CO 80004 Arvada
 - Megan Cantrell, 1720 Quacco Road, A, GA 31322 Pooler
 - Marah Cantrell, 4603 Grant Street Northeast, DC 20019 Washington
 - August Cantrell, 2433 Southwest 36th Street, OK 73109 Oklahoma City
 - Alden Cantrell, 6420 Via Baron, CA 90275 Rancho Palos Verdes

--> customer 258090 has 0 orders
==> total order value is: 0.0
```

5.) Refer to *CustomerDataStore.py* and complete the implementations in *Stock*- und *OrderDataStore.py* and the import functions in *DataFactory*. Successful implementations should produce output:

`--> 1000 customers, 871 stock items, 124 orders loaded.` [2 Pts]

6.) Complete the implementations of *Stock.py* and *Order.py* classes in the *datamodel* package. Refer to class *Customer.py* for a complete implementation.

Pay attention to private attributes (with double underscore) in class diagrams.

Look up one stock and one order object from stock and order data stores and print the full set of attributes to demonstrate correct class implementations. [2 Pts]

7.) Complete the implementations of functions in `C1_driver.py`:

- `print_orders(_customer_id: int)`, which uses
- `print_order_items(o: Order) -> int:`

With a correct implementation, output should be as follows: [2 Pts]

```
--> 1000 customers, 871 stock items, 124 orders loaded.
124 orders in OrderStore
1000 customers in CustomerStore
customer 898179: Joan Hendricks
--> find customers with last name 'Cantrell':
...skipped names

--> customer 258090 has 2 orders
- order 00-769-34958 (1 item):
  - 1 x Remote Operation Unit Cable UC-V1000 = 499.0
  --> order value is: 499.0
- order 00-210-52970 (1 item):
  - 3 x EOS 4000D Gehäuse + EF-S 18-55mm f/3.5-5.6 III Value Up Kit = 1287.0
  --> order value is: 1287.0
==> total order value is: 1786.0
```

8.) Test your code with the following customer ids (remove comments in *C1_driver.py*):

```
# find all orders for each customer
print_orders(258090)    # customer 258090: 2 orders with 1 item each
print_orders(368075)    # customer 368075: 1 order with 1 item each
print_orders(986973)    # customer 986973: 1 order with 4 items
print_orders(193667)    # customer 193667: 4 orders with 1 item each
print_orders(973407)    # customer 973407: 0 orders, is customer
print_orders(333333)    # customer 333333: 0 orders, not a customer
print_orders(-1)        # customer -1: illegal customer id
```

Expected output is: [2 Pts]

```
--> customer 368075 has 1 orders
- order 00-609-44671 (1 item):
  - 5 x Schutzfilter 82mm = 595.0
  --> order value is: 595.0
==> total order value is: 595.0

--> customer 986973 has 1 orders
- order 00-200-46326 (4 item):
  - 1 x EF 35mm f/2 IS USM = 599.0
  - 4 x Speedlite 270 EX II = 876.0
  - 1 x EF-S 18-135mm f/3.5-5.6 IS USM = 549.0
  - 1 x Schutzfilter 72mm = 59.0
  --> order value is: 2083.0
==> total order value is: 2083.0

--> customer 193667 has 4 orders
- order 00-532-44167 (1 item):
  - 3 x IXUS 185 - Silber Essentials Kit = 357.0
  --> order value is: 357.0
- order 00-554-82798 (1 item):
  - 1 x XA11 BP-820 Power Kit = 1549.0
  --> order value is: 1549.0
- order 00-423-97519 (1 item):
  - 3 x Augenkorrekturlinsen EOS Ed mit Augenmuschel Ed . 4 Dioptrien = 177.0
  --> order value is: 177.0
- order 00-564-57334 (1 item):
  - 1 x BP-820Akku = 119.0
  --> order value is: 119.0
==> total order value is: 2202.0

--> customer 973407 has 0 orders
==> total order value is: 0.0

--> customer 333333 has 0 orders
==> total order value is: 0.0

--> customer -1 has 0 orders
==> total order value is: 0.0
```