PROGRAMMING ASSIGNMENT 2

`A first step towards Object Oriented Programming`

Subject: Classes, objects, encapsulation, and inheritance[1]
Teaching Assistant: Selim Yilmaz
Release Date: 3/27/2019 (from 23:59:59)
Due Date: 4/17/2019 (until 23:59:59)

1 Introduction

In this assignment, you are expected to implement a simplified restaurant management system using *Java* programming language and to get familiar with object oriented programming. The software that you will implement is going to fulfill some basic operations that are included in most of today's restaurant management systems.

2 Restaurant Management System

The restaurant management software employs Tables, Orders, Items, People (as workers) and performs some operations depending on the provided commands. It basically relies on two types of workers to operate: *i*) `employers` who only create Tables and *ii*) `waiter` who handles all the stuffs regarding the orders. There are limited number of tables that can only be created by employer and every table has limited number of orders in which limited number of items are stored. Design remarks of the software you are asked to implement are explained in the following sections in detail.

2.1 Design remarks

As stated earlier, workers (employers and waiter) handle all the operations throughout the run. Every worker has *name*, *salary*, *authorization* which states permission to create a new table or to take any order. Table creation can only be performed by *employer* whereas orders can only be operated by *waiter*. Therefore, every employer should have also information carrying the number of tables that (s)he has created so far. Waiter, however, should carry information stating the number of orders operation by him/her. There are maximum `MAX_EMPLOYER` employers and `MAX_WAITER` waiters (=5 for both in this study) allowed. Employer should not be allowed to create more than `ALLOWED_MAX_TABLES` (=2 in this assignment) tables. In addition, waiter can deal with no more than `MAX_TABLE_SERVICES` (=3 in this assignment) tables at any time.

The number of tables created by employers should be limited to `MAX_TABLES` (=5 in this assignment). Every table should have an *ID* that uniquely identifies itself, *capacity* that states maximum number of customers it can take. In addition, it has information stating whether it is currently in service or not. The *creator* as well as the *waiter* information should also

[1] optional

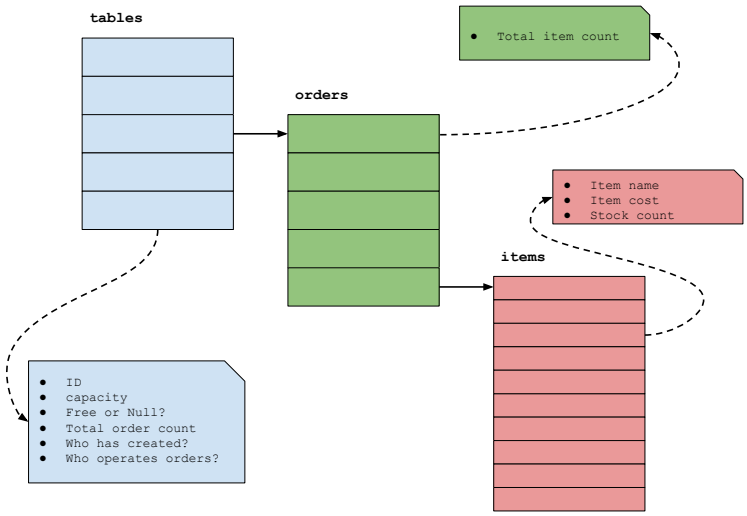

Figure 1: Relational scheme among table, order, and item objects

be stored within the table. As stated previously, every table possesses order information and maximum `MAX_ORDERS` (=5 in this assignment) orders should be provided.

An order simply records every items that are ordered by customer and there should not be more than `MAX_ITEMS` (=10 in this assignment) within a single order! Finally, every item should have name, cost, and amount information which states its name, cost, and the stock amount, respectively.

Please refer to Figure 1 in order to better grasp the relation between the design objects.

2.2 Initialization

Software should not take any parameter as argument but rather perform first some setup operations considering a provided input file, named `setup.dat`. This file always contains three types of commands which are `add_item` (to add a new item into the restaurant's stock), `add_employer` (to add a person as employer), and `add_waiter` (to add a person as waiter). Below, detailed description of each command is provided with its syntax:

- `add_item`: It takes item name, cost for a single item, and number of items that is added into the stock as arguments.

```
add_item [Name];[Cost];[Amount]
```

- `add_employer`: It takes employer's name as well as salary.

```
add_employer [Name];[Salary]
```

- `add_waiter`: It is similar to addition of a new employer and takes two inputs corresponding waiter's name and salary.

```
add_waiter [Name];[Salary]
```

After it completes initial setup progress, it take another input file, named `commands.dat`, comprising of a set of commands each of which yields different production. In the following; each command, with its syntax, is described in detail.

2.3 Commands

- `create_table [EMPLOYER_NAME];[CAPACITY]`: It creates a new table. It should be initial command so that the remaining commands become meaningful. It takes only two parameters; one for name that represents employer as creator, and other for the capacity stating maximum number of customer it can take for service.

```
Input:
create_table ahmet;4

Output:
PROGRESSING COMMAND: create_table
A new table has succesfully been added
```

In the case *i*) where non-existing employer attempts to create a new table, *ii*) where more than allowed maximum number of tables (`MAX_TABLES`) is being created, or *iii*) where given employer attempts to create more than maximum number of tables allowed to create (`ALLOWED_MAX_TABLES`), an appropriate message should be displayed as below.

```
Input:
create_table elif;2

Output:
PROGRESSING COMMAND: create_table
There is no employer named elif

Output:
PROGRESSING COMMAND: create_table
Not allowed to exceed max. number of tables, MAX_TABLES

Output:
PROGRESSING COMMAND: create_table
elif has already created ALLOWED_MAX_TABLES tables!
```

- `new_order [WAITER_NAME];[#CUSTOMER];[ITEM_NAME]-[ORDER_COUNT]`: It states the initial order and used when a new customer arrives at the restaurant. Therefore a free and appropriate table should first be allocated with this command. It takes three parameters. The first represents waiter's name, the second indicates the number of customer, the final parameter lists name-number pairs of items.

Note that there may be more than one pair each of which should be separated by colon mark. In addition, the *first* appropriate table should be reserved for the customers (e.g., it should be 2 if appropriate tables are 2, 5, and 7.). In the case where *i*) no appropriate table exists, *ii*) where no specified waiter exists, or *iii*) where a non-existing/unknown item is specified in the order; appropriate message should be displayed.

```
Input:
new_order kemal;3;Pizza-2:Coke-1

Output:
PROGRESSING COMMAND: new_order
Table (= ID 0) has been taken into service
Item Pizza added into order
Item Pizza added into order
Item Coke added into order

Input:
new_order kemal;4;Pizza-8:Coke-1

Output:
PROGRESSING COMMAND: new_order
Table (= ID 2) has been taken into service
Item Pizza added into order
Item Pizza added into order
Sorry! No Pizza in the stock!
Sorry! No Pizza in the stock!
Sorry! No Pizza in the stock!
Sorry! No Pizza in the stock!
Sorry! No Pizza in the stock!
Sorry! No Pizza in the stock!
Item Coke added into order

Output:
PROGRESSING COMMAND: new_order
Not allowed to service max. number of tables, MAX_TABLE_SERVICES

Output:
PROGRESSING COMMAND: new_order
There is no appropriate table for this order!

Output:
PROGRESSING COMMAND: new_order
There is no waiter named kemal
```

```
Input:
new_order kemal;1;Waffle-1:Tea-1

Output:
PROGRESSING COMMAND: new_order
Table (= ID 3) has been taken into service
Unknown item Waffle
Item Tea added into order
```

- `add_order [WAITER_NAME];[TABLE_ID];[ITEM_NAME]-[ORDER_COUNT]`: Similar to `new_order`, it inserts another order into a given table which should already be in service. The same warnings in `new_order` should also be displayed in this command except the one regarding `MAX_TABLE_SERVICES` as no new table is currently being taken into service. Special to this command, an appropriate message should be displayed as below in the case where a given table is not currently in service or where a waiter other than the one first operated to this table attempts to operate further orders!

  ```
  Input:
  add_order kemal;0;Water-1:Coffee-1

  Output:
  PROGRESSING COMMAND: add_order
  Item Water added into order
  Item Coffee added into order

  Output:
  PROGRESSING COMMAND: add_order
  This table is either not in service now or kemal cannot be
  assigned this table!

  Output:
  PROGRESSING COMMAND: add_order
  Not allowed to exceed max number of orders!
  ```

  It is important to note that every item should be separately inserted into the order and every time software progresses `add_order` command, a separate order should be created and inserted into the specified table. As expected, every time an item is added into any order, its stock amount should be decreased by one.

- `check_out [WAITER_NAME];[TABLE_ID]`: It indicates a case where customers taking service at table `TABLE_ID` want to checkout. As expressed in previous command, only waiter who operated to table with `TABLE_ID` is allowed to carry out this command, otherwise appropriate message should be displayed.

```
Input:
check_out kemal;1

Output:
PROGRESSING COMMAND: check_out
Pizza:  3.000 (x 2) 6.000 $
Coke:   1.500 (x 1) 1.500 $
Water:  0.500 (x 1) 0.500 $
Coffee: 0.750 (x 1) 0.750 $
Donut:  1.250 (x 2) 2.500 $
Tea:    0.200 (x 3) 0.600 $
Total:  11.850 $

Output:
PROGRESSING COMMAND: check_out
This table is either not in service now or kemal cannot be
assigned this table!

Output:
There is no waiter named kemal
```

From the output, it should be highlighted that you are expected to calculate item-based cumulative cost and display with 3 fraction digits by considering all the orders where each has one or more items. Total cost for the table should also be displayed (also with 3 fraction digits) at the end line. Do not forget to end every line with the currency mark ($ in this assignment). Not only should your software calculate the costs; but also it set the given table its initial state (i.e., set as 'not in service', not any waiter assigned, and etc.).

- `stock_status`: It simply displays the stock amount of all items.

```
Input:
stock_status

Output:
PROGRESSING COMMAND: stock_status
Pizza:      0
Hamburger:  2
Water:      3
Coke:       4
Coffee:     2
Tea:        5
Donut:      4
Doner:      6
```

It is seen from the output that stock status of all items (including ones with 0 stock) should be displayed in insertion order.

- `get_table_status`: It prints out all created tables with its current status (*Free*, or *Reserved*) in an insertion order as below. Note that, waiter's name should also be displayed in the case *Reserved*.

```
Input:
get_table_status

PROGRESSING COMMAND: get_table_status
Table 0: Free
Table 1: Free
Table 2: Reserved (kemal)
```

- `get_order_status`: It prints out information regarding table-order-item trio in an insertion order as below. Note that, table should also be displayed even if it is not currently in service.

```
Input:
get_order_status

Output:
PROGRESSING COMMAND: get_order_status
Table: 0
        3 order(s)
                3 item(s)
                2 item(s)
                5 item(s)
Table: 1
        0 order(s)
Table: 2
        1 order(s)
                3 item(s)
```

- `get_employer_salary`: In order to improve motivation, employers (as well as waiter) are awarded (as additional payment to their salary) every time they create a table. Total award (A) for an employer e should be calculated as follows:

$$A_e = T_e \times S_e \times 0.1$$

where T_e and S_e represent total number of tables created by employer e and the salary for employer e, respectively.

With this command, net salary (i.e., $S_e + A_e$) should be displayed for each employer in an insertion order.

```
Input:
get_employer_salary

Output:
PROGRESSING COMMAND: get_employer_salary
Salary for ahmet:  2750.0
Salary for zeynep: 3000.0
Salary for kamil:  3000.0
```

- `get_waiter_salary`: It is very similar to `get_employer_salary` but differs only in calculation, which should now be as follows for a waiter w:

$$A_w = O_w \times S_w \times 0.05$$

where O_w is total number of orders operated by waiter w. As in previous command, net salary (i.e., $S_w + A_w$) should be displayed for each waiter in an insertion order.

```
Input:
get_waiter_salary

Output:
PROGRESSING COMMAND: get_waiter_salary
Salary for kemal: 1440.0
Salary for ayse:  1500.0
Salary for ziya:  1575.0
```

3 A Complete Example

As stated earlier, the implementation you are expected to design will not take any parameter to run nor will write anything into any file! But instead, it makes use of two input files (`setup.dat` and `commands.dat`) to function properly. **Please note that your implementation must not wait user to provide any input from the keyboard**.

In the case your implementation is run with the following `setup.dat` as well as with the `commands.dat` files, software should display the output that is given thereafter.

Note that, the format of the output of your implementation should strictly match (including special characters, white spaces, and etc.) with the output given in this paper.

setup.dat

```
add_item Pizza;3;5
add_item Hamburger;1.5;2
add_item Water;0.5;4
add_item Coke;1.5;7
add_item Coffee;0.75;3
add_item Tea;0.2;5
add_item Donut;1.25;4
add_item Doner;2.5;6
add_employer ahmet;2500
add_employer zeynep;2500
add_employer kamil;3000
add_waiter kemal;1200
add_waiter ayse;1500
add_waiter ziya;1500
```

commands.dat

```
create_table ahmet;4
create_table zeynep;2
create_table elif;2
new_order kemal;3;Pizza-2:Coke-1
new_order kemal;6;Pizza-2:Coke-1
new_order ziya;1;Pizza-1:Coke-1
add_order kemal;0;Water-1:Coffee-1
check_out kemal;1
check_out ziya;1
create_table zeynep;6
new_order kemal;4;Pizza-8:Coke-1
stock_status
add_order kemal;0;Donut-2:Tea-3
get_order_status
check_out kemal;0
check_out kemal;1
check_out ismet;0
get_table_status
get_employer_salary
get_waiter_salary
get_order_status
```

```
**********************************
PROGRESSING COMMAND: create_table
A new table has succesfully been added
**********************************
PROGRESSING COMMAND: create_table
A new table has succesfully been added
**********************************
PROGRESSING COMMAND: create_table
There is no employer named elif
**********************************
PROGRESSING COMMAND: new_order
Table (= ID 0) has been taken into service
Item Pizza added into order
Item Pizza added into order
Item Coke added into order
**********************************
PROGRESSING COMMAND: new_order
There is no appropriate table for this order!
**********************************
PROGRESSING COMMAND: new_order
Table (= ID 1) has been taken into service
Item Pizza added into order
Item Coke added into order
**********************************
PROGRESSING COMMAND: add_order
Item Water added into order
Item Coffee added into order
**********************************
PROGRESSING COMMAND: check_out
This table is either not in service now or kemal cannot be assigned this table!
**********************************
PROGRESSING COMMAND: check_out
Pizza:  3.000 (x 1) 3.000 $
Coke:   1.500 (x 1) 1.500 $
Total:  4.500 $
**********************************
PROGRESSING COMMAND: create_table
A new table has succesfully been added
**********************************
PROGRESSING COMMAND: new_order
Table (= ID 2) has been taken into service
Item Pizza added into order
Item Pizza added into order
Sorry! No Pizza in the stock!
Sorry! No Pizza in the stock!
Sorry! No Pizza in the stock!
Sorry! No Pizza in the stock!
Sorry! No Pizza in the stock!
Sorry! No Pizza in the stock!
Item Coke added into order
**********************************
PROGRESSING COMMAND: stock_status
Pizza:      0
Hamburger:  2
Water:      3
Coke:       4
Coffee:     2
```

```
Tea:        5
Donut:      4
Doner:      6
**********************************
PROGRESSING COMMAND: add_order
Item Donut added into order
Item Donut added into order
Item Tea added into order
Item Tea added into order
Item Tea added into order
**********************************
PROGRESSING COMMAND: get_order_status
Table: 0
        3 order(s)
                3 item(s)
                2 item(s)
                5 item(s)
Table: 1
        0 order(s)
Table: 2
        1 order(s)
                3 item(s)
**********************************
PROGRESSING COMMAND: check_out
Pizza:  3.000 (x 2) 6.000 $
Coke:   1.500 (x 1) 1.500 $
Water:  0.500 (x 1) 0.500 $
Coffee: 0.750 (x 1) 0.750 $
Donut:  1.250 (x 2) 2.500 $
Tea:    0.200 (x 3) 0.600 $
Total:  11.850 $
**********************************
PROGRESSING COMMAND: check_out
This table is either not in service now or kemal cannot be assigned this table!
**********************************
PROGRESSING COMMAND: check_out
There is no waiter named ismet
**********************************
PROGRESSING COMMAND: get_table_status
Table 0: Free
Table 1: Free
Table 2: Reserved (kemal)
**********************************
PROGRESSING COMMAND: get_employer_salary
Salary for ahmet:  2750.0
Salary for zeynep: 3000.0
Salary for kamil:  3000.0
**********************************
PROGRESSING COMMAND: get_waiter_salary
Salary for kemal: 1440.0
Salary for ayse:  1500.0
Salary for ziya:  1575.0
**********************************
PROGRESSING COMMAND: get_order_status
Table: 0
        0 order(s)
Table: 1
```

```
        0 order(s)
Table: 2
        1 order(s)
                3 item(s)
```

4 Grading Policy

Task	Point
Report	10
`create_table`	5
`new_order`	15
`add_order`	10
`check_out`	15
`stock_status`	5
`get_table_status`	5
`get_order_status`	15
`get_employer_salary`	10
`get_waiter_salary`	10

5 Design Notes

- With this assignment which covers the subjects of *classes*, *objects*, *encapsulation*, and *inheritance*; you are expected to gain practice on the basics of object oriented programming (OOP). Therefore, you will **not be graded** if any paradigm other than OOP (i.e., structured programming) is developed!
- Inheritance is optional for this assignment. Therefore, you can handle with the assignment without it. But you are highly encouraged to include it to while implementing the assignment.
- Design your implementation in Java 8 in which Lambda expressions are introduced, which considerably simplify the development. So, you are highly encouraged to make use of lambda expressions in this assignment. Note that, Java 8 has already been installed on our department's dev server (type `java -version`).
- Reporting is mandatory. Do not waste your (neither my) time with trivial information. Instead, your report should cover *i*) problem definition (limited to max. 3 sentences), *ii*) solution approach (limited to max. 3 sentences), and *iii*) an explanation per class (limited to a single sentence for each).

6 Submission Notes

- Do not miss the deadline.
- Save all your work until the assignment is graded.
- The assignment must be original, individual work. All the duplicate or Internet works (even if a citation is provided) are both going to be considered as cheating.
- You can ask your questions through Piazza and you are supposed to be aware of everything discussed there.
- You will submit your work from our department's submission system with the file hierarchy as below: This file hierarchy must be zipped before submission (Not .rar, only .zip files are supported by the system)

  → <student id.zip>
  → src.zip <DIR>

- The class name in which main method belongs should be `Assignment2.java`. All classes should be placed in **src** directory in `src.zip`. Feel free to create subdirectories, corresponding the package(s), but each also should be in `src` directory.
- The location of `setup.dat` and `commands.dat` files should be the same with that of `Assignment2.java` file.