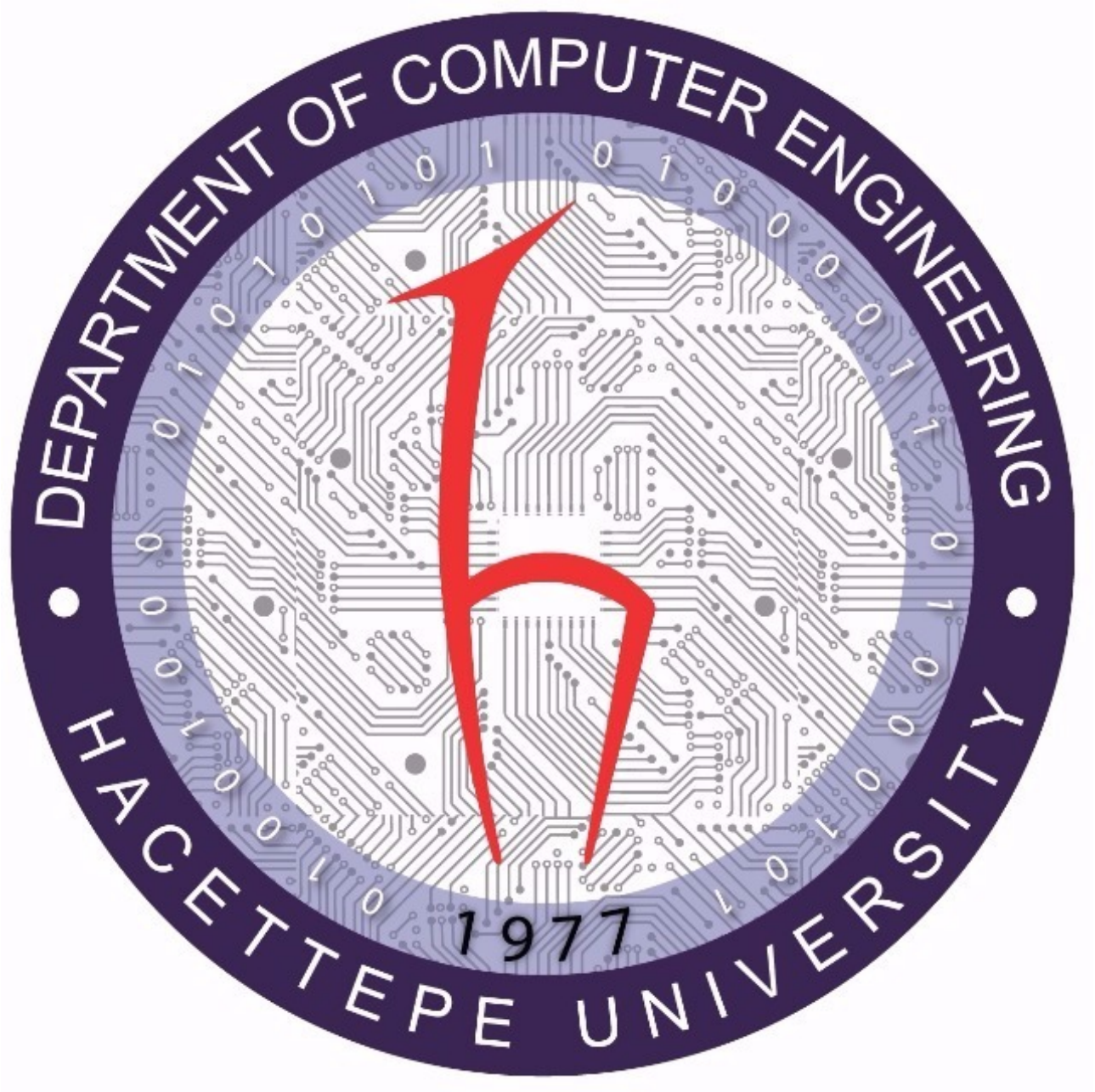

HACETTEPE UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

GÖKHAN ÖZELOĞLU
21627557

ASSIGNMENT 3

2.1. SOFTWARE USAGE

NOTES :
1) Names must be like this on files ( ) :

-HotPepper
-Salami
-Onion
-Soudjouk
-AmericanPan
-Neapolitan
-AddCustomer
-RemoveOrder
-CreateOrder
-AddPizza
-PayCheck
-ListCustomers
-RemoveCustomers

This program takes 3 input files on command line as an argument. There is not any special file name. The sample command line is like this:

$ java Main <inputfile> <customerfile> <orderfile>

All of files are being read by program. After executing of program, <customerfile> and <orderfile> are updated.

1-<inputfile>: This file includes commands to make operation on <customerfile> and <orderfile>.

2-<customerfile>: This file contains some of informations about the customer's. Every customer has an **ID, name, surname, phone number and address**. ID's are unique for every customer.

3-<orderfile>: This file stores the orders for some customers. There are customer and order ID's, pizza and topping names and soft drink information.

*All of files are separated by space character.
*None of file should not contain empty line. This will occur error and program terminates immediately.
*<customerfile> and <orderfile> are updated after executing of program.
*Also, program creates an output file in name ***"output.txt"*** which contains commands of input file's outputs.

4-output.txt: This file writes the result messages of every each line. User can understand what happened during execution of the program.

2.2. ERROR MESSAGES

There are 3 different error messages in this program. One of them occurs if user try to add more than 3 topping to pizza. I controlled this possible error situation with using *if-else* statements and makeControlOrderNum method. If customer wants to give order more than 3 toppings, program gives an error message as this : "***You cannot give order more than included 3 toppings!".***

Second error message occurs if the customer wants to give an order except 4 toppings. The message for this error is "***You have tried to give non-existing topping!***".

Third error message occurs if non-existing customer is tried to being removed. *"**You cannot remove non-existing customer**"*

Furthermore, I had to use *try-catch* blocks while reading file. Somehow, if one of file cannot being read or open unknown reason, program prints out the screen which kind of error occurs.

3. SOFTWARE DESIGN NOTES

3.1 Description of the Program

3.1.1 Problem

The problem is making operations on the input files with using *Decorator Pattern* and *Data Access Object.* Creating classes and their methods related to decorator pattern. Also, all of data should stores with using Data Access Object.

3.1.2 Solution

I used interface, inheritance, encapsulation and polymorphism in this assignment. Due to nature of decorator pattern, I declared Pizza interface. This interface is being implemented by DecoratorPizza class. Also, DecoratorPizza class is extended by Salami, Soudjouk, HotPepper, Onion, AmericanPan, Neapolitan classes. Also, CustomerDAO and OrderDAO classes are interface. These interfaces are implements by CustomerDAOImp and OrderDAOImp classes. These interfaces has methods related to their operation of them. Methods are filled in CustomerDAOImp and OrderDAOImp classes.

Data Access Object is a pattern which stores different kind of information with not effecting each other. So, I declared two different interfaces and their implementation classes. These classes are not effecting each other. For instance, when user want to make changes on order file, this operation is not being affected customer file. When program need to get information about order or customer, program access its class by using accessor methods. Program does not need to access customer class while tries to get information about order. In my solution, customer and order informations are being stored in 2 different arraylist. Operation on these arraylists provides with using objects which in *Customer* or *Order* type. Objects are created from Customer and Order classes.

Customer class stores these informations : ***Customer ID** (*String customerID*)
***Customer Name** (*String customerName*)
***Customer Surname** (*String customerSurname*)
***Customer Phone Number** (*String customerPhoneNumber*)
***Customer Address** (*String customerAddress*)

Order class stores these information : ***Order ID** (*String orderID*)
***Customer Name** (*String customerID*)
***Soft Drink**(*int orderSoftDrink*)
***Pizza** (*Pizza orderPizza*)

An object is being created by one of these classes with using constructors. After that, object is being added to array list. Removing operations is happening with deleting object from arraylist. If an object removes from array list, its all informations deletes from the program's database. So, there is no need to make any operation to remove all of informations from different array lists.

In *decorator pattern* part, I declared an interface in name Pizza. This interface is being implemented by DecoratorPizza class. Also, DecoratorPizza class is being extended by topping classes(HotPepper, Onion,

Soudjouk, Salami) and pizza classes (AmericanPan and Neapolitan). In my implementation, AmericanPan or Neapolitan class must be inside of the constructor. Toppings are being added over this pizza. Program creates new pizza's with adding toppings. While adding toppings, also, pizza's ingredients and cost are being added. These 2 informations are being holded on DecoratorPizza class's field as an object. If cost and pizza name informations are being needed to in anywhere, calling cost() and getDescription() methods is enough.

OPERATION CLASS

void readInputFile : This method takes 3 parameters. These parameters are inputFile, customerFile orderFile.
Also, creates an output file in name "*output.txt*". Before reading input file, program calls the "*readCustomerFile*" and "*readOrderFile*" classes. Program jumps into these two methods. Then, program reads the input file with using while loop line by line. After assigning line as a string a variable, program make controls the command with *if-else* statements. Every *if-else* statements, program calls the methods from CustomerDAOImp and OrderDAOImp classes to make operations.

void readCustomerFile : This method takes only one parameter which is customerFile in *File* type. Program reads the customer file line by line and adds them in customerArrayList before implementing of input file.

void readOrderFile : This method takes one parameter which name is orderFile in *File* type. Program reads order file line by line before implementing of input file. While reading order file, program creates pizza objects and adds them in orderArrayList.

void writeCustomerFile : This method writes the customer file when all of input operations are finished. So, customer file is being updated in this case. Customer file must be ordered by customer ID. So, firstly, program opens a new array list and customer ID's are being added to this array list with converting their types from string to integer. After that, *Collections.sort();* built-in method is being sorted. While writing on file, nested for loop is being used. First for loop is looking for ID's, second for loop is looking for customer list. If ID's are being matched, program writes the file.

void writeOrderFile : This method writes the order file when all of input operations are finished. So, order file is being updated in this case. Order file must be ordered by order ID. So, firstly, program opens a new array list and order ID's are being added to this array list with converting their types from string to integer. After that, *Collections.sort();* built-in method is being sorted. While writing on file, nested for loop is being used. First for loop is looking for ID's, second for loop is looking for order list. If ID's are being matched, program writes the file.

int makeControlOrderNum : This is a helper method to know how many pizza tries to added. Counts the space characters with using while loop. Then returns the number of (pizza+topping). If there are more than 3 topping, it returns 0.

String reversePizza : This method is making swap operation. Firstly, program splits the string separated by space. Then, changes the places of array's elements with using for loop. After for loop, array is being made just a string with using while loop.

CustomerDAOImp CLASS

List<Customer> getCustomerArrayList : Returns the customerArrayList

void addCustomer : This method takes parameters which are customer's information in their types. Also, takes 2 extra parameters to write on output file and boolean parameter to know where the program is used this method. If program is used in readInputFile method, boolean is true. So, program writes the messages on output. If boolean is false, program does not write any message. Just adds the new customer.

void removeByID : This method removes the customer from customer list. Removes the customer by customer ID's. Program looks for ID's in for loop. If ID's are matched, removes the customerArrayList's element by using *remove()* method.

void customerList : When ListCustomer command is being catched by program, just calls this method. Firstly, sorts the names by using *Colloections.sort()*, then, writes the file with nested for loop.

OrderDAOImp CLASS

List<Order> getOrderArrayList : Returns the orderArrayList

void createOrder : Creates order with using order ID, customer ID. If boolean is true, program writes the related message to output file. If it is false, program just adds to array list.

void addDrink : This method calls when user wants to add new soft drink to order. Adding soft drink operation is making with order ID. Searches on the order list, if founds the order ID, add new soft drink. Also, if boolean is true, program writes the message on output file.

void addPizza : When user wants to add new pizza, this method is being called. Compares the order ID's and then, new pizza adds the order. After adding pizza, program writes the message on output file.

void payCheck : This method calls when payCheck command is appear on input file. Program send just order ID to this method. Then, compares orders with using for loop and *if-else* statements. Also, program calculates total cost in *for* loop. After that, writes the output file related messages.

void addPizzaFromFile : This method is only using for adding new pizza's while reading order file. Creates new order, then, adds the orderArrayList.

void removeOrderID : When remove order command is come up with order ID, this method is being called by program. Firstly, calculates how many the same order are there in orderArrayList. Then, loop turns number of ID times. Program removes orders in every loop.

3.1.3. ALGORITHM

* Program starts with taking command line arguments. Command line must be like this:
<$ java Main <InputFile> <CustomerFile> <OrderFile>
* Creates file objects
* Creates operate object from Operation class, then, calls readInputFile method.

---

* In readInputFile method, creates *BufferedReader* object to read input file.
* Then, calls the readCustomerFile and readOrderFile methods to read customer and order files.
* When two of them are being read, orderArrayList and customerArrayList array lists are stores file's informations.
* Then, program comes back to readInputFile method.
* Execution of program continues with reading input file line by line.

* In while loop, program reads the line and assign an variable. This operation continues until last line.
* After assigning line a variable, program controls the command by using *substring()* and *if-else* statements.
* In every *if-else* statement, program calls the related methods.

---

* After execution of Operation class, program returns the Main class.
* Calls again the Operation class's methods.
* These 2 methods are being created to write order and customer files.

HOW TO CREATE PIZZA? (DECORATOR PATTERN)

In my program, Pizza is a interface which includes *cost()* and *getDescription()* methods. This interface class is being implemented by DecoratorPizza class. *cost()* and *getDescription()* methods declared in this class. This class holds 3 variable in field. Pizza and toppings comes this class adding by adding, and finally, assigns in this field. Also, DecoratorPizza class is extended by HotPepper, Salami, Onion, Soudjouk, AmericanPan, Neapolitan classes. Due to not knowing which pizza is being created and which kind of toppings are being added to pizza, I used reflection. Firstly, program creates topping's and pizza's classes by using *forName()* method. Then, declares classes methods to Constructor objects. Finally, program creates object from Pizza class by using *newInstance()* method. Program starts to create and add new pizza inside of the object declaration. It goes to outside of the constructor. In every situation, program firstly creates pizza (AmericanPan or Neapolitan). After that, continues with adding topping if any topping is written on input. In every adding new instance, program adds costs and descriptions. After creating pizza, if you prints out the pizza name with using accessor method, you will get an string separated by spaces. Also, you will get total cost of pizza with using *cost()* method.