**Tehnici de programare**

**Tema 4:**

**No title found**

Student: Pop Tudor Andrei

Grupa:20229

Data: 06.06.2019

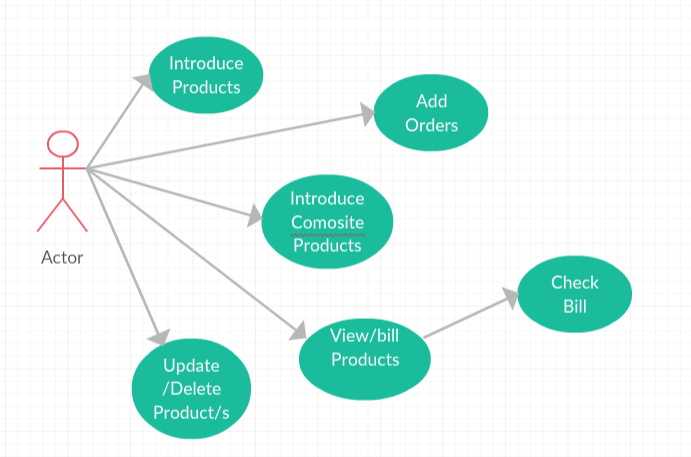
Cuprins:

1. Obiectivul temei
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)
4. Implementare
5. Rezultate
6. Concluzii
7. Bibliografie
8. Obiectivul temei

TP Lab–Homework 4 Objectives • Design by Contract Programming Techniques • Polymorphism • Design Patterns: Observer, Composite • JCF HashMap and HashSet implementations • Serialization Description Consider implementing a restaurant management system. The system should have three types of users: administrator, waiter and chef. The administrator can add, delete and modify existing products from the menu. The waiter can create a new order for a table, add elements from the menu, and compute the bill for an order. The chef is notified each time it must cook food ordered through a waiter. Consider the system of classes in the diagram below. To simplify the application you may assume that the system is used by only one administrator, one waiter and one chef, and there is no need of a login process. Solve the following: 1. Define the interface RestaurantProcessing containing the main operations that can be executed by the waiter or the administrator, as follows: • Administrator: create new menu item, delete menu item, edit menu item • Waiter: create new order; compute price for an order; generate bill in .txt format. 2. Define and implement the classes from the class diagram shown above: • Use the Composite Design Pattern for defining the classes MenuItem, BaseProduct and CompositeProduct • Use the Observer Design Pattern to notify the chef each time a new order containing a composite product is added. 3. Implement the class Restaurant using a predefined JCF collection which uses a hashtable data structure. The hashtable key will be generated based on the class Order, which can have associated several MenuItems. Use JTable to display Restaurant related information. • Define a structure of type Map> for storing the order related information in the Restaurant class. The key of the Map will be formed of objects of type Order, for which the hashCode() method will be overwritten to compute the hash value within the Map from the attributes of the Order (OrderID, date, etc.) • Define a structure of type Collection which will save the menu of the restaurant. Choose the appropriate collection type for your implementation. • Define a method of type “well formed” for the class Restaurant. • Implement the class using Design by Contract method (involving pre, post conditions, invariants, and assertions). 4. The menu items for populating the Restaurant object will be loaded/saved from/to a file using Serialization

Barem: Mandatory requirements for accepting the assignment: 1. OOP paradigms 2. Classes of maximum 200 lines (except the UI classes) 3. Methods of maximum 30 lines 4. Java naming conventions https://google.github.io/styleguide/javaguide.html Minimal requirements for accepting the assignment (5 points): • Implement the class diagram from the homework specification. Choose wisely the appropriate data structures for saving the Orders and the MenuItems. • Graphical interface: o Window for Administrator operations: add new MenuItem, edit MenuItems, delete MenuItems, view all MenuItems in a table (JTable) o Window for Waiter operations: add new Order, view all Orders in a table (JTable), compute bill for an Order • Documentation Requirements for maximum grade (5 points): Requirement Grading Use Composite Design Pattern for modelling the classes MenuItem, BaseProduct, CompositeProduct. 1 point Create bill in .TXT format. 1 point Design by contract: preconditions and postconditions in the RestaurantProcessing interface. Implement them in the Restaurant class using the assert instruction. Define an invariant for the class Restaurant. 1 point Window for Chef user: use Observer Design Pattern to notify each time a new Order is added 1 point Save the information from the Restaurant class in a file using serialization. Load the information when the application starts. 1 point Submission • Create a repository on bitbucket with the name: PT2019\_Group\_LastName\_FirstName\_Assignment\_4 • Push the source code and the documentation (push the code not an archive with the code) • Share the repository with the user utcn\_dsrl Tutorials • Java Serialization o http://www.tutorialspoint.com/java/java\_serialization.htm • Java HashMap o http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html • Java assert o http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html o http://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.html o http://stackoverflow.com/questions/11415160/how-to-enable-the-java-keywordassert-in-eclipse-program-wise

2. Analiza problemei, modelare, scenarii, cazuri de utilizare

UML Case Diagram: 

Actorul – reprezinta utilizatorul, acesta introduce datele de intrare pentru simularea scenariului

Introducere produse – actorul va accesa interfata administatratorului pentru a introduce noi produse. Daca se doreste introducerea unor elemnte composite trebuie sa se apese butonul care semnaleaza programului aceasta intentie pentru a introduce elementele. Campul numelui trebuie sa fie completat inainte de a apasa acest buton

Vizualizarea datelor –actorul poate accesa unele din cele 2 comenzi disponibile pentru a vizualiza datele din program, se apasa butonul view fie in interfata waiterului unde se vor afisa order-urile prezente/existente, fie se apasa butonul cu acelasi numa din interfata adiministratorului canz in care se vor afisa toate produsele existente, fie ele compuse sau simple intr-un jTable.

Editare datelor – interfata Administratorului vine cu tool-urile pentru a edita numele si/sau pretul unui produs la dorinta actorului, efectele fiidn vizibile doar cand se reapasa butonul view pentru a observa efectele dorite. Tot in aceasta categorie, in aceeasi interfata exista optiunea de deleteProduct. Se introduce numele produsului care se doreste sters si se pasa butonul. Similar se poate observa efectul apasant butonul view

Factui- In interfara waiter se poate introduce id-ul(numarul) unei facturi care poate fi vizualizat in table cu toate facturile existente, dupa care se apasa butonul de “Bill”. Se va genera un fisier txt cu detaliile legate de order, dupa care acea comanda va fi stearsa din memeoria programului ca semn de achitare.

Un posibil mod de utilizare:

1. Se introduc mai multe produse simple.
2. Se introduce un produs compozit
3. Se introduce un produs compozit care contine un alt produs compozit
4. Se vizualizeaza produsele
5. Se editeaza/sterg anumite produse
6. Se vizualizeaza produsele
7. Se introduce un order conform explicatiilor aduse anterior
8. Se observa notificarea primita de ChefGUI
9. Se vizualizeaza datele apasand butonu view.
10. Se genereaza o facture pentru comanda plasat.
11. Se vizualizeaza datele.

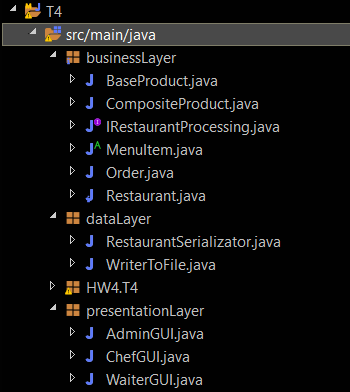
Programul genereaza automat data/identificatorii pentru itemii introdusi prin interfata si aceastea pot fi vizualizare apasand butonul

Daca datele introduse sunt valide, acestea pot fi modificare/refolosite in alte operatii in limita perimisiunilor bazei de date.

Pentru alte detalii de corectitudine a datelor se cere consultatea consolei programului pentru ca poate contine informatii despre formatul datelor ce trebuie introduse.

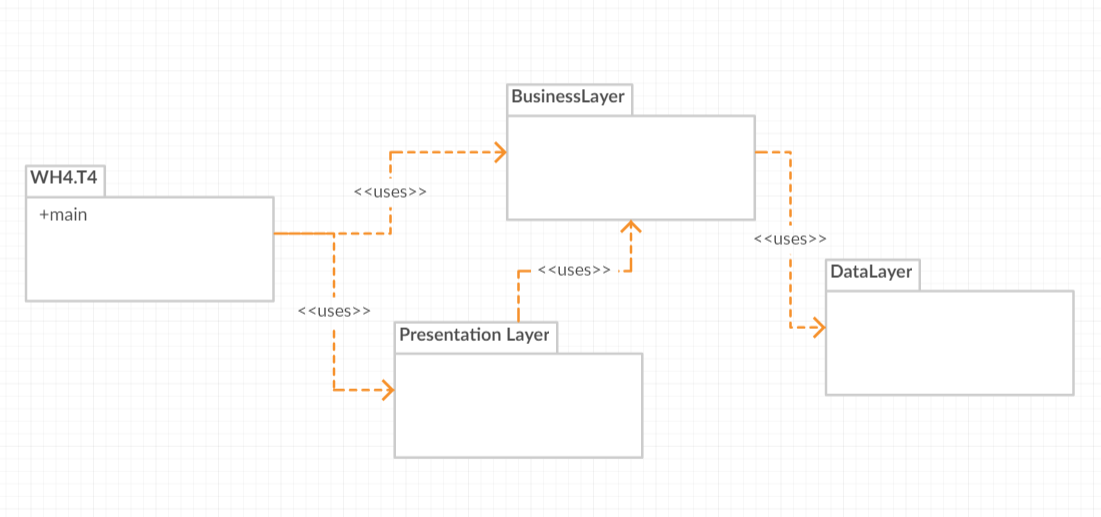
3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

Pentru proiectare respectat propunerea din barem: Layered Architecture (the application will contain at least four packages: dataAccessLayer, businessLayer, model and presentation), similar conditia Use relational databases for storing the data for the application, minimum three tables: Client, Product and Order.



Pachetul businessLayer se ocupa cu mare parte a logicii aplicatiei.

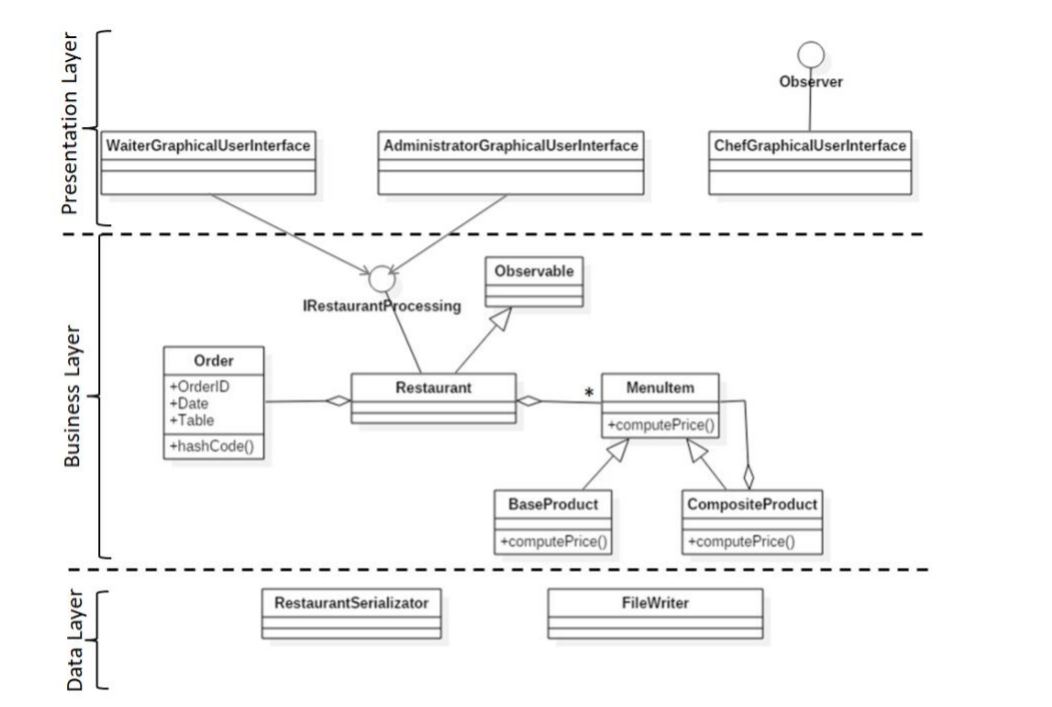
**Package Diagram:**



Pachetul WH4.T4 contine o clasa ce implementeaza metoda main(), care porneste interfata grafica. Interfata grafica este centrul programului, deoarece seleteaza si foloseste direct sau indirect fiecare functionalitate a celorlalte pchete.

Pachetul DataLayer se ocupa de scrierea in fisiere si de serializare, de asemenea ajuta la initializarea restaurantului, daca ar merge serializarea

Pachetul BusinessLayer impreuna cu PresentatioLayer asigura logica programului si o interfata de interactiune a userului cu acesta

**Class Diagram:**

Dupa cum se poate observa din ilustratia de mai sus, proiectul respecta cu strictete diagram de clasa propuse in cerinta

4. Implementare

Varianta acutala acestei aplicatii este rezultatul unui sir lung de incercari si esuari in care am incercat diferite metode, diferite abordari,variante asemanatoare cu cele prezentate la laborator ( poate chiar curs ) testate pe o perioada de 3 saptamani, recomandari de pe diferite site-uri cum ar fi StackOverflow/GeeksForGeeks, youtube si asa mai departe.

Codul poate contine diferite bucati de cod commentate, variabile nefolosite, metode neapelate sau care pur si simplu nu am nicio aplicabilitate toate din incercariile anterioare care pentru un motiv sau altul nu si-au gasit loc in variant “finala”

Clasa APP : in interiorul aceste clasa se afla metoda “public static void main(String[] args)” in interiorul acesteia declar un obiect din clasa “FirstWindow” si apelez cu acel obiect metoda “perform()”. Desigur, acest apel este inconjurat de o structura try/catch pentru a intercepta eventualele exceptii ce inevitabil pot aparea (Fara try/catch deoarece aceaesta structura isi are locul acum in “FirstWindow” )

Clasa **WaiterGUI**: in interiorul acestei clasa construiesc efectiv interfata grafica si implementez clasa “ProjectListener”. Constructorul acestei clase da forma aplicatiei, un obiect ”JFram” care contine restul obiectelor intr-un “GridLayout”. Fereastra in sine are rol de a procura si valida parametrii care trebuie sa dicteze cum decurge rularea programului. O data cu apasarea butoanelor care reprezinta comenzile pe care userul de poate executa, se va construi o fereastra de de tipul respective care va fi specializata in executarea unui singur sql query cu anumite date introduse de utilizator.

Clasa AdminGUI: Sunt specializare in executare unui singur querry dictat de datele introduse

Clasele din BussinessLayer: reprezinta marea majoritatea a logicii aplicatiei care este transmisa mai departe claselor care o mostenesc. Este construita dupa modelul care se afla in resuresele din utcn\_dsrl. Am mai adaugat dupa nevoie alte metode, am modificat pe cele din exemplu pentru a se mula peste modul de proiectare regasit in aplicatie.

**Interfata IRestaurantProcessing**

:

/\*\*

\* **@requires** order!=null

\* **@requires** food!=null

\* **@invariant** isWellFormed()

\*/

public void addOrder(Order order, ArrayList<MenuItem> food);

/\*\*

\* **@requires** o!=null

\* **@invariant** isWellFormed()

\*/

public void addItem(MenuItem o);

/\*\*

\* **@requires** name!=null

\* **@invariant** isWellFormed()

\*/

public void deleteItem(String name);

/\*\*

\* **@requires** item!=null

\* **@invariant** isWellFormed()

\* **@ensures** boolean

\*/

public boolean exists(MenuItem item);

/\*\*

\* **@requires** name!=null

\* **@invariant** isWellFormed()

\* **@ensures** MenuItem

\*/

public MenuItem findByName(String name);

/\*\*

\* **@invariant** isWellFormed()

\* **@requires** ItemName!=null

\* **@requires** filed!=null

\* **@requires** value!=null

\* **@ensures** void

\*/

public void editItem(String ItemName, String field, String value);

/\*\*

\* **@requires** ordId>0

\* **@invariant** isWellFormed()

\*/

public int computeOrderPrice(int ordId);

/\*\*

\* **@requires** orderId>0

\* **@invariant** isWellFormed()

\* **@ensures** Order

\*/

public Order getOrderById(int orderId);

/\*\*

\* **@requires** o!=null

\* **@invariant** isWellFormed()

\* **@ensures** void

\*/

public void deleteOrder(Order o);

/\*\*

\* **@invariant** isWellFormed()

\* **@ensures** String[][]

\*/

public String[][] avaibleOrders();

/\*\*

\* **@invariant** isWellFormed()

\* **@ensures** String[][]

\*/

public String[][] avaibleMenuItems();

/\*\*

\* **@invariant** isWellFormed()

\* **@ensures** void

\*/

public void saveData();

/\*\*

\* **@invariant** isWellFormed()

\* **@ensures** boolean

\*/

public boolean isWellFormed(); Aceste metode creaza query-urie care vor fi executate in continuare de metodele care fac legatura dintre baza de date si functionalitate programului. Acestea sunt urmatoarele:

public void insertObject(ArrayList<String> values)

public T findObject(int id)

public ResultSet findAllObjects()

public void deleteObject(int id)

public void updateObject(int id, String field, String value)

functionalitatea acestora poate fi dedusa foarte usor din numele lor: insert: insereaza in baza de date, find : caut din baza de date dupa un id dat, findAll: gaseste toate datele unui table, delete: sterge din baza de date dupa un anumit id. updateObj: va actualize un singur camp la un moment dat cu o valoare de pe campurile care trebuie completate in fereastra corespunzatoare acestei functionalitati.

private static final long serialVersionUID = -4561986279476793825L;

public String name;

public int price;

private boolean composed;

public abstract int computePrice();

public boolean isComposed() {

return composed;

}

public void setComposed(boolean composed) {

this.composed = composed;

}

@Override

public boolean equals(Object o) {

MenuItem i = (MenuItem)o;

if(name.equals(i.name) && price == i.price && composed == i.composed ) {

return true;

}

return false;

}

@Override

public String toString() {

return "MenuItem [name= " + name + ", price= " + price + "]";

}

Aceasta metoda folosete tehnica reflection pentru a ganera obiectele echivalente din aplicatie cu randurile tabelelor bazei de date. Ea sta la baza inca unei metode care obtine numele campurilor claselor pentru generearea coloanelor tabelului care apare la obtiunea findAll.

Clasa OrderDAO: Mosteneste clasa AbstractDAO, iar in plus fata de aceasta si fata de restul claselor dao, mai are cateva metode specifice care au rol in a updata tabela product de fiecare deta cand o noua comanda se plaseaza.

Implementeaza metodele:

private String selectQuantValQuery()

public int findQuantVal(int id)

Clasa public class CreateBillWin extends JFrame: mentionez aceasta clasa pentru a castiga cuvinte in plus la documentatie si pentru ca este relevanta deoarece pentru generarea facturii se face pe baza id-ului unei comenzi. Contructorul aceste clasa va genera tabelul (head + data ) folosind tehnica reflectiei conform cerintei minime. Constructorul executa un select statement pe fiecare tabela a bazei de date, obtinand astfel automat datele necesare completarii campurilor noilor facturi, iar de la user foloseste doar id-ul comenzii la care se doreste generarea facturii.

Tot in aceasta clasa se genereaza fisierul txt care reprezinta ultima factura generate de aplicatie

FileWriter wr = new FileWriter("LastEmittedBill.txt");

wr.write(bd.findObject(order.getOrdertableId()).toString();

wr.flush();

wr.close();

5. Rezultate

Conform asteptariilor, programul functioneaza bine, afisaza rezultatele dorite, ferestrele de erori apar cand trebuie, conexiunea cu baza de date functioneaza.

Datele pot fi introduse, pot fi vizualizate, modificate, sterse si utilizare de celetalte tabele ale bazei de date.

In anumite situatii se mai propaga excepti al caror stacktrace este afisat in consola, de asemenea apar anumite mesaje in consola care au un rol de debugging si nimic mai mult. Dar din lipsa timpului nu am mai apucat sa le elimin. Nu afecteaza functionalitatea programului.

6. Concluzii

- documentatia s-a putut scrie in ultima zi (\*noapte 2:15 AM, Test la EGC imediat dupa lab-ul de PT (y) )

- proiectul se poate inbunatatii la fiecare pas: optimizare, design, implementarea operatiilor pe mai multe threaduri, oprirea threadurilor trebuie revizuita

- codul contine multe ramasite de la incercarile anterioare, incluziv clase care nu au niciun impact asupra logicii in sine, dat daca sunt eliminate apar exceptii, iar programul refuza sa functionze

-am invatat multe despre cum functioneaza functia reflection-ul, desi nu la un nivel la care m-as putea descurca fara exemplul pus de dumneavoastra ca material ajutator

- Serializarea-ul este foarte foarte sensibila la numele metodelor, la tipurile de date, la contructorii care apar in clase, va arunca exceptii din motive neaspatate (raportat la nivelul me de cunostinte despre tehnica). Aceste exceptii si rezolvarea lor au fost foarte frustrante si au luat un timp considerabul de implementat.

7. Bibliografie

1) <https://en.wikipedia.org/wiki/Polynomial_long_divisio> -- pentru impartire

2) <https://creately.com/diagram-type/use-case> -- pentru diagrame

3) <https://stackoverflow.com/> -- pentru restul problemelor

4) Toate linkurile care apar in materialul ajutator