**DOCUMENTATIE**

**TEMA 4**

**Student : Mihali Gabriel**

**Grupa : 30225**

Contents

[**1.Obiectivul temei:** 3](#_Toc103629300)

[**2. Analiza problemei, modelare, scenarii, cazuri de utilizare** 4](#_Toc103629301)

[**3.Proiectare:** 5](#_Toc103629302)

[**4.Implementare:** 6](#_Toc103629303)

[**5.Concluzii** 11](#_Toc103629304)

[**6.Bibliografie** 11](#_Toc103629305)

## **1.Obiectivul temei:**

* **Obiectivul principal :** Implementarea unei aplicatii, cu interfata grafica pentru management-ul unui food delivery system, cu un sistem de register-login, cu mai multi utilizatori care pot comanda produse.
* **Obiectivele secundare :** 
  + **Crearea unei clase User** : Clasa User va avea setter-e, getter-e si parametri care ajuta la implementarea aplicatiei, ca si username-ul parola, id, si va fi Serializable. Mai multe detaili despre clasa User in capitolul 4.
  + **Crearea unei clase MenuItem** : Clasa MenuItem, de tip abstract va face parte din Composite Design pattern si va fi serializabila ca si User. Aceasta are o metoda speciala abstracta, computePrice() care va fi suprascrisa in clasele copil.
  + **Crearea unei clase BaseProduct** : Clasa BaseProduct extinde clasa MenuItem si suprascrie metoda computePrice(). Aceasta clasa reprezinta produsele de baza, care nu sunt compuse din mai multe produse.
  + **Crearea unei clase CompositeProduct**. Clasa CompositeProduct extinde clasa MenuItem si pe deasupra contine si un ArrayList de MenuItem. Aici este folositor Composite Design Pattern-ul. Clasa CompositeProduct suprascrie computePrice(), calculand pretul intregului produs adunand preturile celorlalte produse. Mai multe despre Composite Design Pattern in capitolele 3 si 4.
  + **Crearea interfetelor pentru Employee, Admin si Client** : Aceste interfete vor fi lansate cu ajutorul unui username si parola, in functie de privilegiile utilizatorului respectiv. Mai multe despre interfete in capitolul 4.
  + **Deserializarea si serializarea claselor si a fisierelor sursa** : Folosind stream-uri si expresii lambda, se obtine deserializarea si serializarea unui fisier sursa .csv, si crearea mai multor fisiere in care se stocheaza datele despre aplicatie, si generarea anumitor facturi/rapoarte. Mai multe detalii despre metode de deserializare si serializare in capitolul 4.
  + **Crearea si implementarea clasei DeliveryService** : Clasa care se ocupa de actiunile pe datele aplicatiei, implementeaza interfata DeliveryServiceProcessing. Are mai multe metode care adauga produse, creeaza produse, creeaza comenzi, in functie de tipul de colectie/map pe care il folosesc. De asemenea, acest DeliveryService extinde clasa Observable si trimite notificari Observer-ului la orice comanda noua. Mai multe detalii despre aceasta clasa in capitolul 4**.**

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

* **Proiectati si implementati** un sistem de management al livrarii de mancare pentru o companie de catering. Clientul poate comanda produse din meniul companiei. Sistemul trebuie sa aiba trei tipuri de utilizatori : client, admin si angajat. Acestia se logheaza folosind un username si o parola.
* **Use cases :** 
  + **Actor :** employee
    - **Actions :**

1. Observe orders

* **Actor:** administrator
  + **Actions:**
    1. Add product
    2. Modify product
    3. Delete product
    4. Create product
    5. Generate reports
    6. View product
* **Actor :** client
  + **Actions:**
    1. Search products
    2. Place order
    3. Add item
    4. View product

# **3.Proiectare:**

* Aplicatia are 23 clase si un interface, distribuite in in 4 pachete, BusinessLogic, DataAcess, Controller si Presentation.
  + **Pachetul Presentation:** Are 11 clase, AddProductGUI, AdminGUI, ClientGUI, CreateProductGUI, EmployeeGUI, ImportProductGUI, LoginGUI, ModifyProductGUI, RegisterGUI, ReportsGUI, ViewProductGUI. Aceste clase reprezinta interfetele grafice care fac legatura cu utilizatorul. Aceste clase au fost implementate folosind Java Swing, fiind cat mai usor de inteles si placute visual.
  + **Pachetul DataAcess :** Este pachetul care face legatura cu datele externe ale aplicatiei, cu fisierele din sistem. Contine 5 clase : AppendableOutputObjectStream, Bills, Deserializator, Reports si Serializator. Aceste clase scriu in anumite fisiere care vor fi folosite ca si baza de date, prin serializare si preiau informatii din aceste fisiere folosind deserialziarea si expresii lambda.
  + **Pachetul Controller :** Este pachetul din care se porneste aplicatia si care contine o singura clasa, clasa Controller. Aceasta clasa controleaza aplicatia, defineste ActionListener-ele pentru interfete si apeleaza toate metodele necesare manevrarii eficienta a datelor in program.
  + **Pachetul BusinessLogic :** Este pachetul care contine partea de implementarea a datelor din aplicatie. Contine 6 clase si o interfata. 3 dintre aceste clase sunt legate prin Composite design pattern : BaseProduct, CompositeProduct si MenuItem. Acest Composite design pattern este necesar pentru universalizarea datelor in aplicatie. Restul claselor sunt : Order, User, DeliveryService si interface-ul DeliveryServiceProcessing.

Ca si **structuri de date speciale**, am folosit structura HashMap<>, pentru a crea un Map de comenzi, cu perechi <Order, ArrayList<MenuItem>>, prin care se leaga un Order de niste produse. De asemenea am folosit HashMap<> pentru a crea niste Map-uri de frecventa pentru rapoartele admin-ului. Am folosit ObjectInputStream, ObjectOutputStream, pentru serializare respectiv deserializare. Am folosit Collection pentru a returna metoda .values() a Map-urilor si Collectors pentru a calcula frecventa unor intrari din stream-urile produselor. FileInputStream si FileOutputStream sunt folosite impreuna cu clasele asemanatoare de tip ObjectOutputStream si ObjectInputStream pentru a realiza serializarea si deserialziarea.

# **4.Implementare:**

* **Pachetul Presentation : Contine clasele pentru interfata grafica : AddProductGUI, AdminGUI, ClientGUI, CreateProductGUI, EmployeeGUI, ImportProductGUI, LoginGUI, ModifyProductGUI, RegisterGUI, ReportsGUI, ViewProductGUI. Imagini cu interfetele AdminGUI, ClientGUI si EmployeeGUI sunt prezentate mai jos.**

In imaginea de mai jos este ilustrata interfata Administratorului:

Graphical user interface, application

Description automatically generated

In imaginea de mai jos este ilustrata interfata Clientului:

Graphical user interface

Description automatically generated

In imaginea de mai jos este ilustrata interfata AngajatuluiA picture containing square

Description automatically generated

* **Pachetul Controller : Contine clasa Controller, care porneste aplicatia si controleaza restul claselor din aplicatie.**
  + **Clasa Controller :** Din clasa Controller se porneste aplicatia. Aceasta are ca si campuri clasele din pachetul DataAccess, clasele din pachetul Presentation, adica interfetele, liste de clasele din pachetul BusinessLogic. Clasa controller implementeaza ActionListener-ele pentru butoanele din interfete, apeland metode din clasa DeliveryService si clasele din DataAcess, in functie de cerinte. Clasa Controller, folosind pachetul DataAccess genereaza diferite fisiere de tip .txt : fisierul userspasswords.txt care contine username-urile si parolele utilizatorilor, precum si tipul lor, orders.txt care contine toate comenzile facute pe aplicatie si products.txt care contine toate produsele aplicatiei, adaugate ulterior sau importate dintr-un anumit fisier.
* **Pachetul DataAccess: Contine clasele pentru prelucrarea datelor externe, AppendableObjectOutputStream, Bills, Deserializator, Reports si Serializator.**
  + **Clasa AppendableObjectOutputStream :** Este o clasa care extinde ObjectOutputStream si este folosita ca si inlocuitor pentru acesta. A fost necesara implementarea unei asemenea clase, pentru ca scrierea intr-un fisier serializat sa nu genereze un nou header mereu si sa distruga fisierul. Constructorul clasei primeste ca parametru un obiect de tip OutputStream si un boolean numit append, care reprezinta modul de scriere in fisier. Cand append este true, atunci fisierul in care se scrie exista, si nu se va mai genera header. Cand append este false se va genera si header pt fisier.
  + **Clasa Bills :** Aceasta clasa este folosita pentru a genera bill-urile comenzilor plasate de catre un anumit client. Are ca si camp un FileWriter cu ajutor caruia se scrie intr-un fisier initializat in metoda generateOrderBill(). Aceasta metoda scrie cu ajutorul metodei .append() intr-un fisier .txt detalii despre comanda primita ca parametru in metoda.
  + **Clasa Deserializator :** Aceasta clasa este folosita pentru a procesa fisierele externe ale aplicatiei, si ale transforma in date care pot fi manevrate de catre clasele din BusinessLogic. Metoda deserializeProducts() primeste ca parametru numele fisierului serializat cu produse si folosind FileInputStream si ObjectInputStream Aceasta metoda returneaza un HashSet<MenuItem>.Am ales HashSet deoarece aceasta colectie, cu ajutorul suprascrierii metodelor equals() si hashCode(), elimina obiectele duplicate din fisierul de tip .csv importat din cerinta.
  + **Clasa Reports :** Aceasta clasa genereaza rapoartele cerute de catre administratorul aplicatiei, la cerinta sa din interfata. Contine mai multe metode care creeaza fisiere de tip .txt : Report hour, Report products, Report products day etc. Aceste metode preiau ca si parametrii structura de date returnata de catre generarea raportului din DeliveryService si parametrul dupa care se genereaza raportul, si scriu folosind metoda append() in acele fisiere prezentate mai sus.
  + **Clasa Serializator :** Proceseaza datele interne ale aplicatiei, datele BusinessLogic si le serializeaza, scriindu-le in fisiere .txt, fiind stocate fizic in memoria aplicatiei, pentru rulari ulterioare. Metoda processCSV, primeste ca parametru path-ul unui fisier CSV si transforma acel fisier intr-un ArrayList<String>, folosind stream-uri si un .forEach(). Aceasta metoda este necesara pentru functia de importare a datelor din aplicatie. Metoda serializeProducts() primeste ca parametru un HashSet de produse, si, folosind FileOutputStream si AppendableObjectOutputStream scrie serializat intr-un fisier products.txt toate produsele din acel HashSet. Metoda serializeOrders() functioneaza pe acelasi principiu ca si serializeProducts() doar ca primeste ca si parametru un HashMap<Order, ArrayList<MenuItem>> si scrie serializat tot HashMap-ul intr-un fisier orders.txt. Metoda addProduct primeste ca parametru un produs si il adauga serializat in fisierul products.txt.
* **Pachetul BusinessLogic : Este pachetul care se ocupa de implementarea datelor, a tipurilor de date ale aplicatiei si a metodelor care lucreaza cu aceste date. Contine 6 clase : BaseProduct, CompositeProduct, DeliveryService, MenuItem, Order, User si un interface, DeliveryServiceProcessing.** 
  + **Clasa MenuItem :** Este clasa cea mai primitiva a ceea ce reprezinta produsul in aceasta aplicatie. Este o clasa de tip abstract, care implementeaza interfata Serializable, deoarece este serializata de multe ori in aceasta aplicatie. Are ca si campuri un serialversionUID universal, pentru a nu cauza exceptii de serializare la repornirea aplicatiei, un titlu de tip String, un rating de tip Double, calories, protein, fat, sodium, price de tip int. Are setter-e si getter-e corespunzatoare campurilor, iar metoda computerPrice() este de tip abstract urmand a fi implementata particular in fiecare copil al clasei MenuItem. De asemenea, clasa MenuItem face Override metodelor hashCode() si equals() pentru a putea fi stocate intr-un HashSet<>.
  + **Clasa BaseProduct :** Este clasa care reprezinta produsele de baza, necompuse din mai multe produse. Extinde clasa MenuItem, asadar primeste toate campurile clasei MenuItem : title, rating, calories, protein, fat, sodium, price si mai ales serialversionUID pentru a se pastra integritatea serializarilor. Face override la metoda computePrice() si la metodele equals() si hashCode() pentru a putea fi stocata intr-un HashSet<>.
  + **Clasa CompositeProduct :** Este clasa care reprezinta produsele compuse, de exemplu meniurile , care sunt alcatuite din mai multe produse de baza. Extinde clasa MenuItem, asadar poate fi si MenuItem, datorita design pattern-ului Composite. Primeste toate campurile clasei MenuItem, iar pe langa mai contine un ArrayList<MenuItem> de produse din care este alcatuit meniul. Face override la metoda equals() si hashCode() pentru a putea fi stocata intr-un HashSet<> , iar override-ul pe metoda computePrice() este necesar, deoarece pretul meniului este calculat prin parcurgerea acelui ArrayList<MenuItem> si adunarea preturilor produselor de baza.
  + **Interfata DeliveryServiceProcessing :** Este interfata folosita in DeliveryService si are metodele : importProducts(), createOrder(), addItem(), editItem() si searchItems().
  + **Clasa DeliveryService :** Este clasa care se ocupa de actiunile pe datele aplicatiei, mai ales pe comenzi, utilizatori si produse. Aceasta clasa manipuleaza datele in asa fel incat sa realizeze rapoartele, sa adauge produse, sa stearga produse, sa creeze produse, sa modifice produse, sa caute produse prin expresii lambda si sa creeze comenzi. Pentru a realiza aceste actiuni se foloseste de metodele : createOrder(), searchItems(), addItem(), deleteItem(), editItem(), importProducts(), reportHour(), reportFrequencyProduct(), reportDay(), createMaps(). Metoda createMaps() creeaza 2 map-uri : unul de freventa cumpararii produselor, iar celalalt de frecventa plasarii unei comenzi de catre utilizator. Pentru a face cautari de produse, metoda searchItems() se foloseste de stream-uri si de .filter() pentru a filtra listele dupa anumite criterii. Rapoartele se implementeaza asemanator cu .filter(), dar acestea se aplica pe map-uri nu pe liste, deoarece avem nevoie si de frecvente si de perechi de date, nu doar de intrari de date. Clasa DeliveryService extinde Observable, si notifica Observer-ul de creearea unei noi comenzi prin metoda notifyObservers(), apelata in metoda createOrder().
  + **Clasa Order :** Este clasa care contine comenzile efectuate de catre clientii aplicatiei. Este serializabila, contine campurile orderID, ClientID, si orderdate de tip Date, care reprezinta data la care a fost facuta comanda. Are getter-e si setter-e corespunzatoare. Aceasta clasa face override() la hashCode deoarece va fi inclusa intr-un hashMap.
  + **Clasa User** : Este clasa care contine utilizatorii care se logheaza in aplicatie. Este serializabila. Contine campurile user\_id, username, password si user\_type, cu setter-ele si getter-ele corespunzatoare.

# **5.Concluzii**

* In urma implementarii acestei aplicatii, au fost assimilate cunostinte noi despre serializarea si deserializarea datelor, despre stream-uri si expresii lambda. Au fost aprofundate cunostintele despre tipurile de colectii, in special despre Set-uri si Map-uri.
* Ca si dezvoltari ulterioare, poate fi dezvoltata aplicatia prin adaugarea unei baze de date de tip SQL, care va facilita si ajuta operatiunile pe date. De asemenea, aplicatia poate fi extinsa astfel incat sa permita logarea a mai multi utilizatori deodata.

# **6.Bibliografie**

<https://www.w3schools.com/java/java_hashmap.asp>

<https://dsrl.eu/courses/pt/materials/PT2021-2022_Assignment_4.pdf>

<https://dsrl.eu/courses/pt/materials/A4_Support_Presentation.pdf>

<https://www.baeldung.com/java-observer-pattern>