MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

DOCUMENTAȚIE TEMA 4

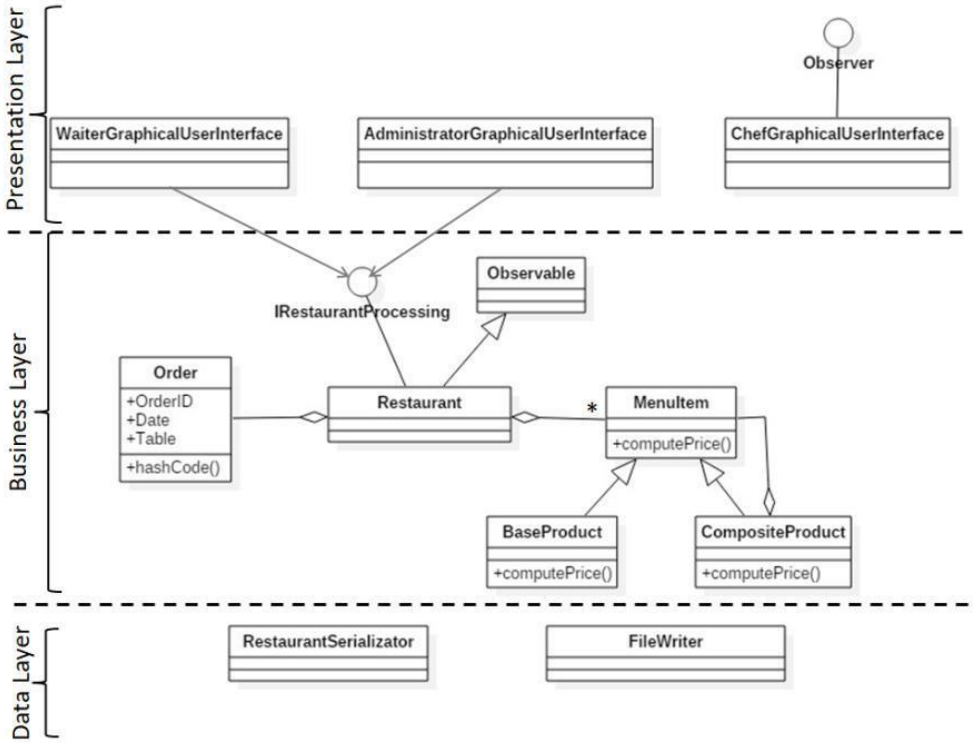
RESTAURANT MANAGEMENT SYSTEM

Trifu Diana-Maria
Grupa 30223
Profesor Laborator: Dorin Moldovan

Cuprins:

1. Cerințe funcționale

Propuneți, proiectați și implementați o aplicație pentru managementul unui restaurant. Sistemul de management al restaurantului ar trebui sa aiba 3 tipuri de utilizatori: administrator, chelner și bucătar. Administratorul poate adăuga, șterge și modifica produsele existente în meniu. Chelnerul poate crea o nouă comandă pentru o masa, poate adăuga elemente din meniu și poate procesa o factură pentru o anumită comandă. Bucătarul este notificat de fiecare data când trebuie să gătească mâncarea care a fost comandată prin intermediul chelnerului. Se consideră sistemul de clase din diagram de mai jos:



Pentru a simplificaa aplicația, se presupune că există un singur administrator, un singur bucătar și un singur chelner, iar astfel nu este nevoie de niciun fel de process de login.

Considerații de implementare:

- **Folosirea limbajului de programare Java**
- **Implementarea claselor cu maxim 300 de linii de cod**
- **Implementarea metodelor cu maxim 30 de linii de cod**
- **Folosirea javadoc pentru documentarea claselor și generarea fișierelor JavaDoc corespunzătoare**
- **Pentru fiecare comandă procesată de chelner se va genera o factură în format text**

2. Obiectivul temei

2.1 Obiectivele principale

Obiectivul principal al proiectului este de a creea o aplicație care să pună la dispoziția utilizatorului un sistem de management al unui restaurant. Această aplicație va permite administratorului să introducă prin intermediul ei diverse produse în meniu, să le șteargă, sau să le modifice, dar va oferi totodată și posibilitatea chelnerului de a înregistra și procesa comenzi. În cazul unei comenzi valide, în cazul în care există un CompositeProduct, se va trimite o notificare către bucătar pentru ca acesta să înceapă să gătească produsl respectiv. Pentru încercarea unei operațiuni invalide, atât din partea chelnerului, cât și a administratorului, se va afișa o nouă fereastră cu eroarea.

2.2 Obiectivele secundare

Obiectivele secundare ale temei reprezintă pașii careau fost urmați pentru atingerea obiectivului final.

1. Invariantul clasei și pre-/postcondițiile

Un invariant de clasă este o expresie care este întotdeauna adevărată despre o variabilă obiect a clasei în orice moment în timp ce un program client se execută.

O precondiție este ceva pe care un programator client îl garantează unei metode. De obicei, o precondiție este o afirmație despre parametrii metodei care este garantată a fi adevărată.

O postconditie este ceva pe care metoda il garanteaza programatorului client care se va intampla ca urmare a apelului la metoda. Uneori, post-condițiile sunt introduse în comentarii javadoc. Postcondițiile sunt cel mai bine implementate cu ajutorul lui "assertion", indifferent dacă sunt sau nu specificate în metode publice.

2. Serializarea obiectelor

Serializarea este o metodă ce permite transformarea unui obiect într-o secvență de octeți sau caractere din care să poată fi refăcut ulterior obiectul original. Cu alte cuvinte, serializarea permite salvarea într-o manieră unitară a tuturor informațiilor unui obiect pe un mediu de stocare extern programului. Procesul invers, de citire a unui obiect serializat pentru a-i reface starea originală, se numește deserializare. Într-un cadru mai larg, prin serializare se înțelege procesul de scriere/citire a obiectelor. Tipurile primitive pot fi de asemenea serializate. Utilitatea serializarii constă în următoarele aspecte:

- **Asigură un mecanism simplu de utilizat pentru salvarea și restaurarea a datelor.**
- **Permite persistența obiectelor, ceea ce înseamna că durata de viață a unui obiect nu este determinată de execuția unui program în care acesta este definit - obiectul poate exista și între apelurile programelor care îl folosesc. Acest lucru se realizează prin serializarea obiectului și scrierea lui pe disc înainte de terminarea unui program, apoi, la relansarea programului, obiectul va fi citit de pe disc și starea lui refacută.**
- **Compensarea diferențelor între sisteme de operare - transmiterea unor informații între platforme de lucru diferite se realizează unitar, independent de formatul de reprezentare a datelor, ordinea octeților sau alte detalii specifice sistemelor repective.**
- **Transmiterea datelor în rețea - Aplicațiile ce rulează în rețea pot comunica între ele folosind fluxuri pe care sunt trimise, respectiv receptționate obiecte serializate**

3. Modele de design

Composite Design Pattern

În inginerie software, acesta este un model de proiectare a compartimentării. Modelul compus descrie un grup de obiecte care sunt tratate la fel ca o singură instanță a aceluiași tip de obiect. Intenția unui

„composite" este de a „compune" obiecte în structuri de arbori pentru a reprezenta ierarhii parțiale. Implementarea acestui model permite clienților să trateze uniform obiectele și compozițiile individuale.

Observer Design Pattern

Acest model este folosit atunci când există o relație unu-la-multe între obiecte, cum ar fi dacă un obiect este modificat, obiectele sale depenedente trebuie notificate automat. Modelul observator se încadrează în categoria modelului comportamental.

Design by contract

Design by contract (DBC) este o metodă propusă inițial de Betrand Meyer pentru proiectarea sistemelor orientate pe obiect și componente. Principala caracteristică a DBC este că clasele își definesc comportamentul și interacțiunea prin „contracte". Un contract în acest context constă în esență dintr-un invariant al clasei și condiții pre și post pentru toate metodele interfeței de clasă.

3. Analiza Problemei

Programarea orientate pe obiect oferă posibilitatea începerii dezvoltării unui proiect fără nevoia efectivă de a cunoaște de la început toate amănuntele funcționării efective a aplicației care urmează să fie dezvoltată. Această strategie folosită este cunoscută sub numele de "Top Down." O abordare de acest tip este esențială într-un proiect de acest tip, deoarece îți oferă o perspectivă mult mai bună asupra sub-sistemelor care alcătuiesc întregul. O astfel de metodă oferă posibilitatea proiectantului să își faca o imagine de ansamblu asupra sistemului, fără a fi nevoit să detalize fiecare nivel al subsistemelor.

3.1 Scenarii

Administratorul are posibilitatatea de a adăuga, șterge și modifica produse în/din meniul restaurantului. În cazul în care acesta încearcă să realizeze o acțiune invalidă(ștergerea/modificarea unui produs inexistent, adăugarea unui produs cu același nume cu unul deja existent) acesta va fi atenționat că acea operațiune este invalidă, prin intermediul unuei notificări. La fel ca și în cazul administratorului, chelnerul va fi atenționat în cazul în care încearcă să creeze o nouă comandă la o masa care este deja ocupată sau încearcă să proceseze plata pentru o comandă inexistentă.

3.2 Modelare

Utilizitarul va putea accesa aplicația fie prin intermediul Eclipse, fie prin intermediul fișierului cu extensia .jar, care ofera o compilare imediată a programului printr-un simplu click. Produsele adăugate în meniu și comenzile înregistrate care doresc a fi salvate de utilizator la ultima accesare, prin simpla apăsare a butonului „SAVE" vor fi serializate în fișierul "restaurant.ser". La redeschiderea aplicației, elementele salvate la ultima accesare a aplicației vor fi deserializate și vor fi vizibile în aplicație. În cazul în care chelnerul va dori să proceseze plata unei comenzi, aceasta va fi ștersă din tabelul interfeței și un fișier text va fi creat cu detaliile plății, sub forma unei facturi.

4. Proiectare

4.1. Structuri de date și clasele folosite

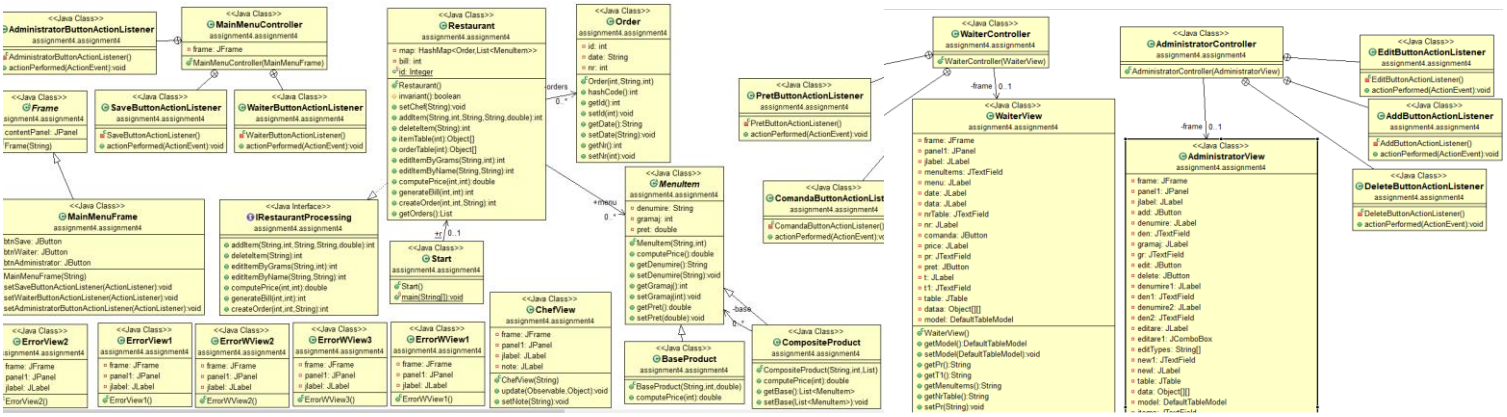
Clasele principale folosite sunt: MenuItem, BaseProduct, CompositeProduct, Order, Restaurant și clasele de View și Controller care alcătuiesc împreună interfețele pentru administrator și chelner. Clasa "Restaurant" implementează toate metodele din interfața „IrestaurantProcessing". În această clasă sunt definite pricipalele metode necesare pentru acțiunile premise administratorului și chelnerului. În clasele care

alcătuiesc interfețele celor doi posibili utilizatori se realizeaza conexiunea între interfața grafica accesata propriu-zis și metodele implementate în clasa ”Restaurant”. Cele trei clase: MenuItem, BaseProduct și CompositeProduct definesc tipul obiectului ce poate fi introdus de către administrator în meniu. Pentru implementarea acestor clase se folosește ”Composite Design Pattern”. Clasa Order definește un obiect de tip comandă, care ulterior, împreuna cu o listă de produse din meniu, vor fi adăugate în clasa ”Restaurant” înr-o HashMap pentru a forma comanda preluată de chelner de la o anumită masa. Cheia este reprezentată de un obiect de tip comandă, iar valoarea elementului de un ArrayList în care se află produse de tip ”MenuItem” pe care chelnerul le-a preluat de la client.

Ca și structuri de date folosite am ales ArrayList-urile și HashMap. ArrayListurile reprizintă o structură de date ușor de utilizat datorită numeroaselor metode care sunt deja gata implementate, ceea ce oferă o mobilitate foarte mare și ușurință în lucrul cu acestea. Un HashMap, stochează articole în perechi „cheie / valoare” și se pot accesa printr-un index de alt tip .Un obiect este utilizat ca cheie (index) pentru un alt obiect (valoare), putând stoca diferite tipuri. Atât cheile, cât și valorile dintr-un HashMap sutn de fapt obiecte. Acestea pot fi atât de tipuri deja implementate(String, Integer), cât și de tipuri nou-create de programator.

4.2. Diagrama de clase

Unified Modeling Language (prescurtat UML) este un limbaj standard pentru descrierea de modele și specificații software. Diagrama de clase UML este folosită pentru reprezentarea vizuală a claselor și a interdependențelor, taxionomiei și a relațiilor de multiplicitate dintre ele. Diagramele de clasă sunt folosite și pentru reprezentarea concretă a unor instanțe de clasă, așadar obiecte și a legăturilor concrete dintre acestea.



5. Implementare

AdministratorController – Controller-ul pentru interfața administratorului. În această clasă sunt descrise comportamentele celor 3 butoane ale interfeței. Butonul ”Add item!” va apela metoda AddItem din clasa Restaurant și se va adăuga produsul în meniu în cazul în care nu mai există un alt produs cu acest nume, iar în caz contrar va apărea un mesaj de eroare care va atenționa utilizatorul asupra acestei greșeli;

AdministratorView – View-ul interfeței administratorului. În această clasă este descris design-ul interfeței care cuprinde: 2 JComboBox-uri, 3 Jbutton-uri, 1 JTable și 7 JTextField-uri. În cazul în

care se dorește adăugarea unui nou produs în meniu trebuie specificat tipul produsului(BaseProduct sau CompositeProduct), nume și gramele. În cazul în care este vorba despre un produs de tip Composite trebuie completat, opțional, câmpul "Items", iar în cazul unui produs de tip Base acest câmp nu trebuie completat, însa trebuie completat câmpul "Price", care în cazul produsului de tip Composite nu este necesar, deoarece prețul unui produs de tip Composite este calculat în funcție de prețul produselor ce intră în componența sa, plus un adaos în funcție de gramajul produsului. În cazul în care se dorește editarea unui produs trebuie selectat tipul de editare dorit: gramaj sau nume. Pentru a edita produsul este nevoie să se indice și numele produsului și noua valoare a gramajului sau noul nume dorit. Pentru a șterge un produs trebuie specificat doar numele acestuia;

BaseProduct – În această clasă este descris un tip de produs din meniu. Acest produs are un nume, un preț și un gramaj.

ChefView – În această clasa este descris view-ul interfeței notificării care are apare pe ecran și anunță bucătarul că un produs de tip Composite a fost comandat de către un client și trebuie să gătească;

CompositeProduct – În această clasă este descris un tip de produs din meniu.Acest produs are un nume, un gramaj, un preț și un ArrayList de produsede tip BaseProduct. În cazul acestui produs prețul este calculat în funcție de prețul produselor care intră în componența acestuia, plus un adaos în funcție de gramajul acestuia;

ErrorView1 – În această clasă este descrisă fereastra de eroare care apare în cazul în care se dorește încercă introducerea unui produs care există deja;

ErrorView2 – În această clasă este descrisă fereastra de eroare care apare în cazul în care se dorește încercă ștergerea sau modificarea unui produs care nu există în meniu;

ErrorWView1 – În această clasă este descrisă fereastra de eroare care apare în cazul în care se dorește încercă crearea unei comenzi pentru o masă care este deja ocupată;

ErrorWView2 – În această clasă este descrisă fereastra de eroare care apare în cazul în care se dorește încercă procesarea unei comenzi care nu există;

ErrorWView3 – În această clasă este descrisă fereastra de eroare care apare în cazul în care se dorește încercă introducerea unui produs care nu există într-o nouă comandă;

IRestaurantProcessing – Interfața în care sunt descrise toate metodele care definesc operațiunile ce pot fi executate de către administrator și chelner;

MainMenuController – Controller-ul meniului principal al aplicației. Pentru fiecare dintre butoanele Waiter și Admin se va deschide interfața corespunzătoare utilizatorului, iar în cazul butonului Save se va serializa obiectul "r" de tip Reastaurant.

MainMenuFrame – View-ul meniului principal al aplicației. În aceast clasă sunt descrise 3 butoane: Save, Admin și Waiter. Fiecare buton este denumit sugestiv, asfel încât să se poată anticipa rolul fiecăruia. În cazul butonului Save se va serializa obiectul de tip Restaurant, în cazul butonului Waiter se va deschide interfața corespunzătoare chelnerului, iar în cazul butonului Admin interfața corespunzătoare administratorului.

MenuItem – În această clasă este descris tipul de produs din meniu. Acest produs are un nume, un preț și un gramaj.

Order – În această clasă este descris un obiect care deține informații despre comanda preluată: id-ul comenzii, numărul mesei de la care a fost preluată comanda și data;

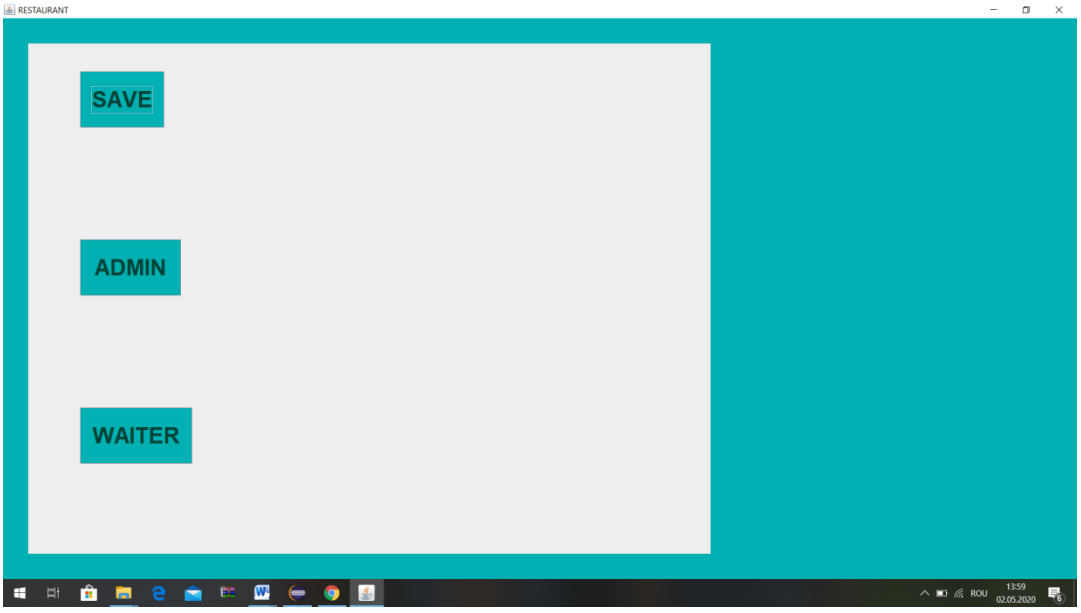
Start – În această clasă se află metoda main. Tot în această clasă este instanțiat un obiect de tip restaurant și tot aici se realizează și deserealizarea.

WaiterController – Controller-ul interfeței chelnerului. În această clasă este descris comportamentul celor 2 butoane. În cazul butonului ”Make order!” se va crea o nouă comandă în cazul în care toate datele necesare creării acesteia sunt valide. În cazul în care se dorește crearea unei comenzi cu un produs care nu există în meniu, sau o nouă comandă pentru o masă deja ocupată va apărea un mesaj de eroare care va atenționa utilizatorul asupra acestui fapt. În cazul butonului ”Compute price and generate bill!” se va calcula prețul total al produselor comandate la masa respectiva și va fi generată o factura, sub forma unui fișier text, în care vor fi prezentate detaliile comenzii și suma totală care trebuie achitată;

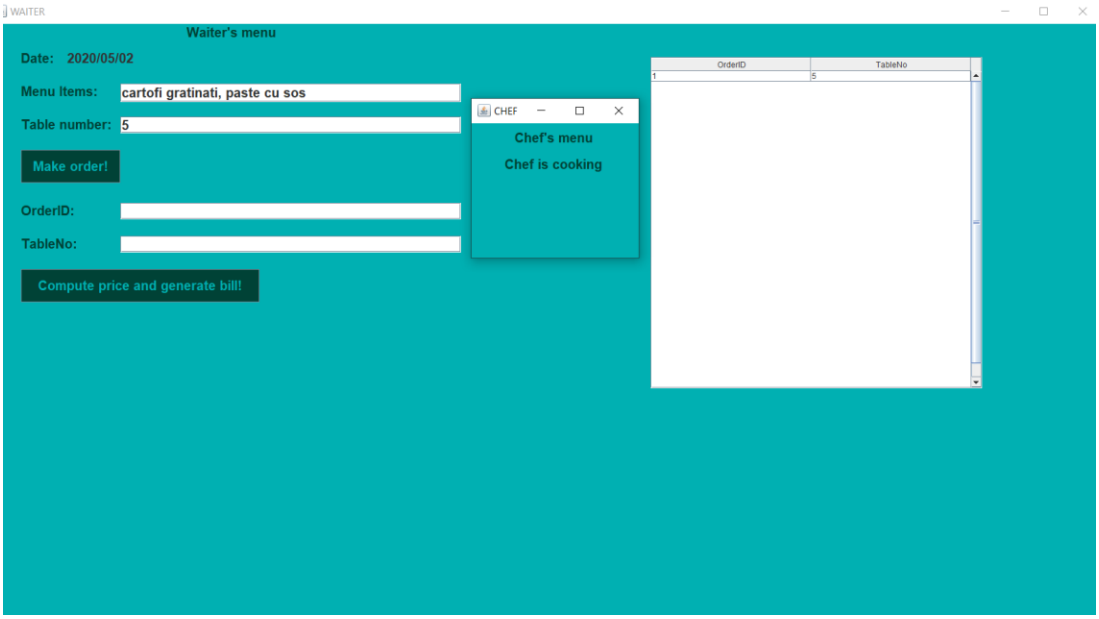
WaiterView - View-ul interfeței chelnerului. În această clasă este descris design-ul interfeței care cuprinde: Jbutton-uri, 1 JTable și 4 JTextField-uri. În cazul în care se dorește introducerea unei noi comenzi trebuie introduse unul sau mai multe produse existente în meniu(produsele comandate de client) și numărul mesei de la care a fost luată comanda. Data va fi preluată automat de către program, iar id-ul comenzii se autoincrementează automat. În cazul în care se dorește procesarea unei comenzi trebuie precizat id-ul comenzii și numărul mesei. Dacă se încearcă procesarea unei comenzi inexistente, sau a unei comenzi care a fost deja procesată, va apărea un mesaj care va atenționa utilizatorul asupra acestei erori.

6. Rezultate

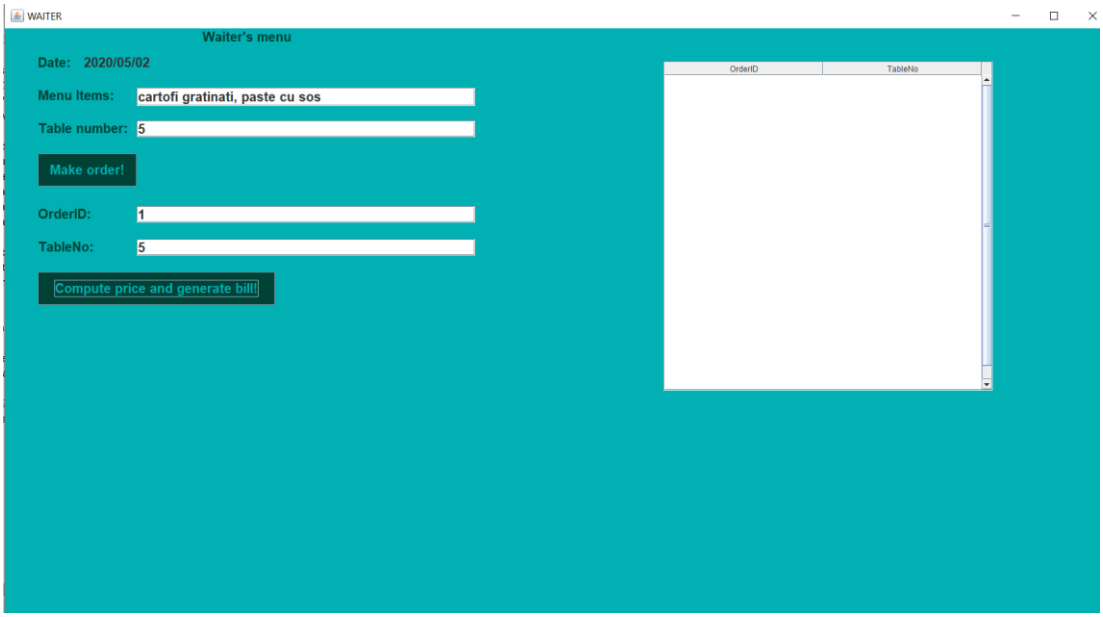
-Interfața principală a aplicației



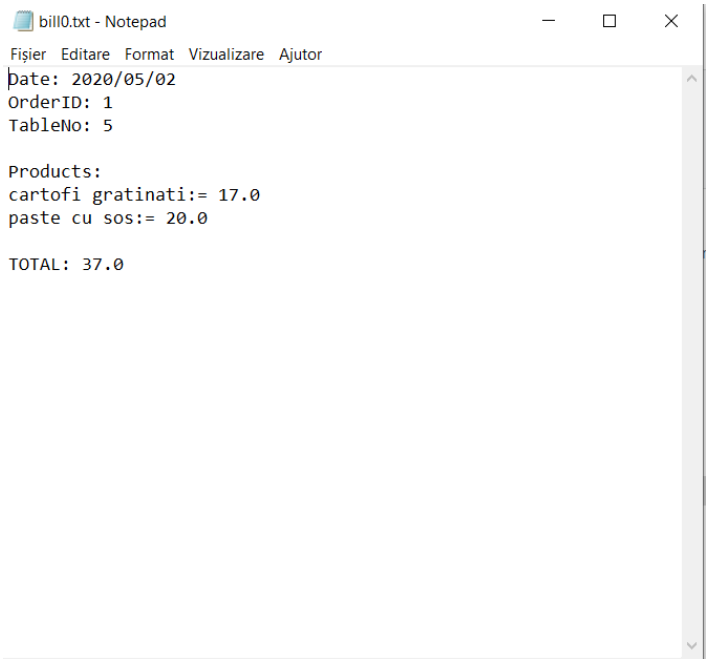
-**Exemplu adăugare o nouă comandă**



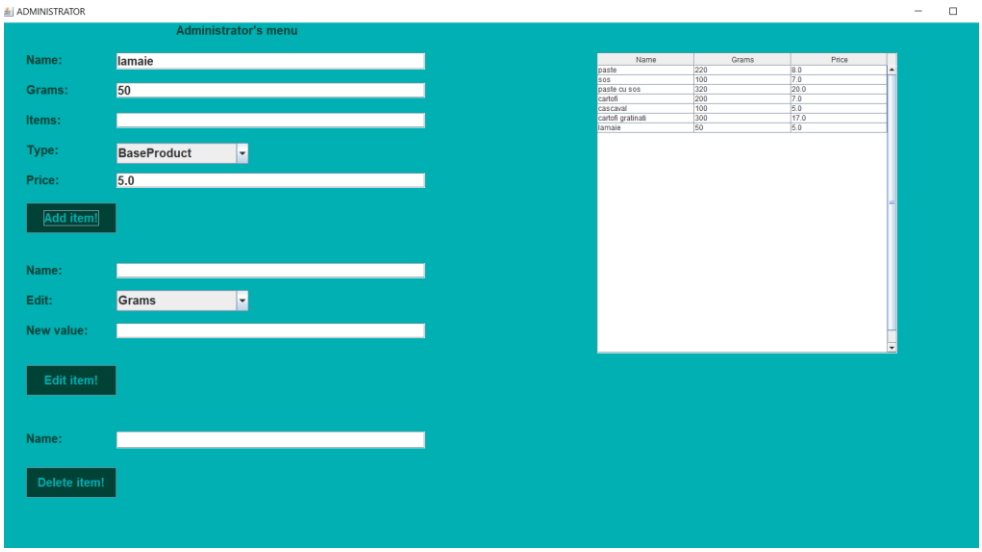
-Exemplu după procesarea comenzii



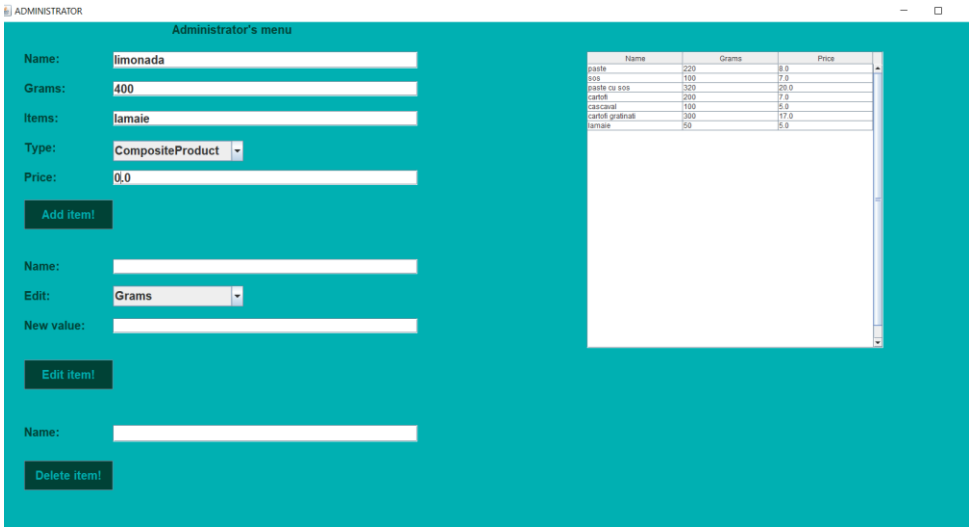
-Exemplu factură



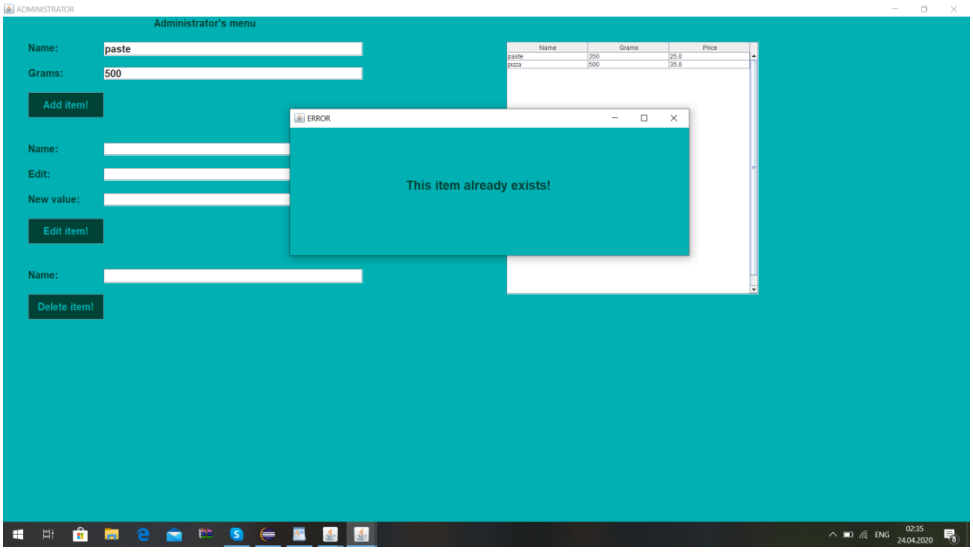
-Exemplu adăugare un nou produs(base) în meniu



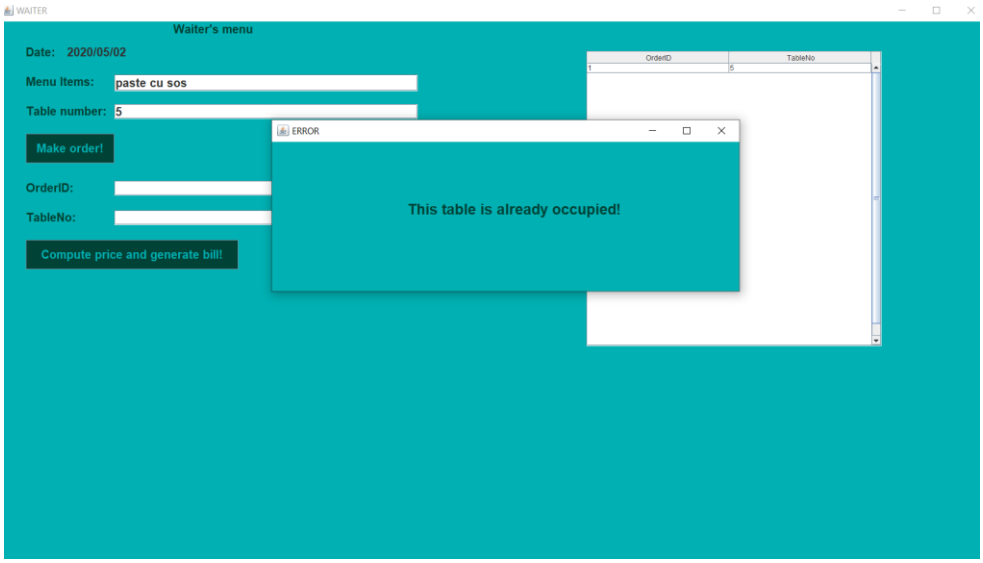
-Exemplu adăugare un nou produs(composite) în meniu (în cazul unui produs de tip Composite, prețul nu contează)



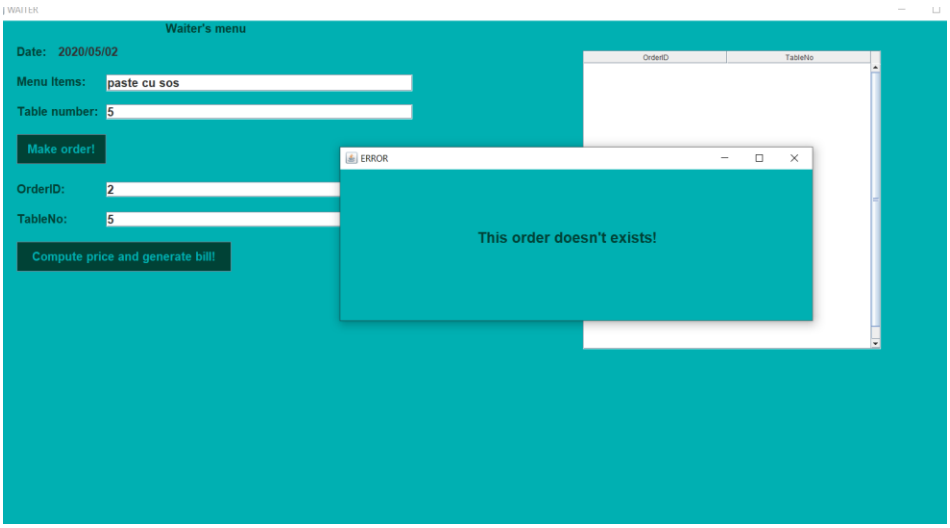
-Exemplu eroare introducere un produs deja existent



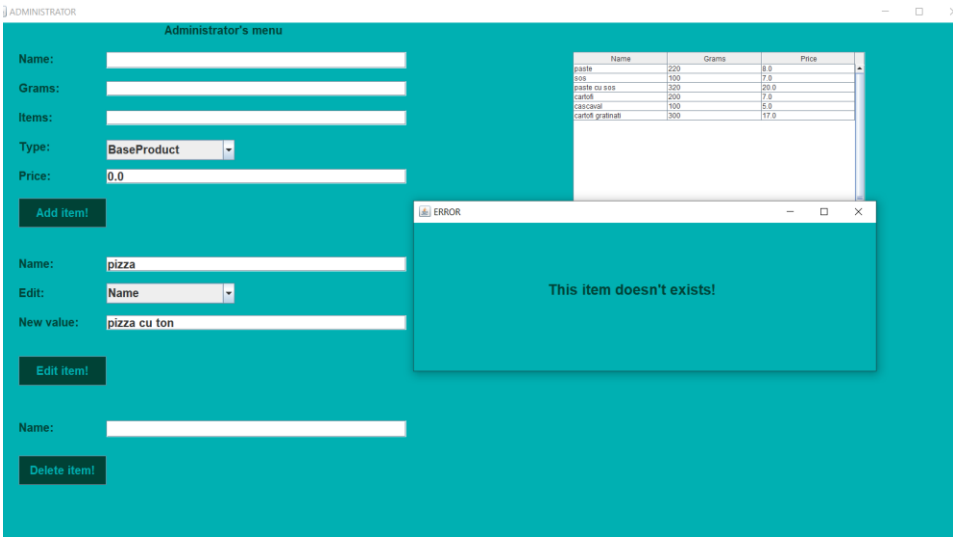
-Exemplu eroare încercare introducere o nouă comandă la o masă la care deja există comandă



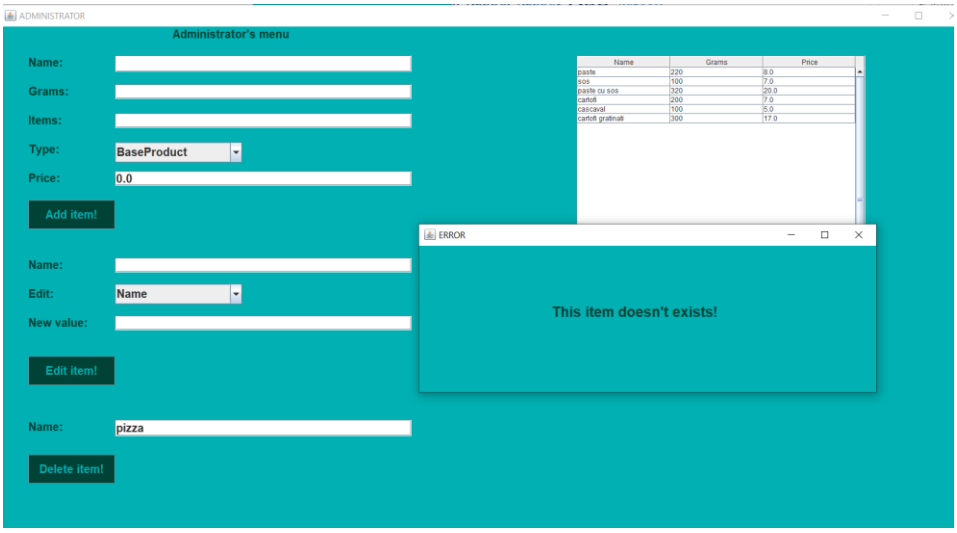
-Exemplu eroare încercare procesare o comandă inexistentă



-Exemplu eroare încercare editare un produs inexistent



-Exemplu eroare încercare ștergere un produs inexistent



7. Concluzii

În concluzie realizarea acestui proiect a fost foarte utilă , amintind de o mare parte din principalele concepte ale programării orientate pe obiect învățate semestrul trecut. Datorită modului în care a fost gândită și proiectată aplicația, dezvoltările ulterioare pot fi foarte ușor implementate în orice moment și nu doar de către cel ce a facut-o, ci și de către alți programatori care ar dori să aducă îmbunătățiri ulterioare aplicației.

O dezvoltare ulterioară care ar putea fi făcută pentru ca aplicația să fie mai ușor de utilizat ar putea fi, spre exemeplu, ca interfața grafică să permita logarea mai multor chelneri și administratori. O altă îmbunătățire ar putea fi reprezentată de posibilitatea de a adăuga ulterior mai multe produse unei anumite comenzi(fără a se primi eroare).

8. Bibliografie

- **https://www.baeldung.com/javadoc**
- **https://www.baeldung.com/java-copy-on-write-arraylist**
- **http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html**
- **http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_4/Assignment_4_Indications.pdf**
- **https://www.w3schools.com/java/java_files_read.asp**
- **https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag**
- **https://www.baeldung.com/java-serialization**
- **Cursurile și laboratoarele de "Programare orientate pe obiect" și "Tehnici de Programare"**