**Documentație Tema 4**

**Restaurant management**

**Student: Tocilă Emilia**

**Grupa 302210**

**Profesor laborator : Antal Marcel**

**Cuprins**

1. Obiectivul temei...........................................................................3
2. Analiza problemei, modelare, scenarii, cazuri de utilizare..........3
3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfața utilizator)........................................................................4
4. Implementare................................................................................5
5. Rezultate.......................................................................................9
6. Concluzii....................................................................................12
7. Bibliografie.................................................................................12

* **Obiectivul temei**
* **Obiectivul principal**

Considerați o aplicație pentru managementul unui restaurant. Sistemul trebuie să aibă 3 tipuri de utilizatori: administrator, ospătar și bucătar. Administratorul poate adăuga, șterge și modifica produsele existente din meniu. Ospătarul poate crea o nouă comandă pentru o masă, adăugând elemente din meniu și să creeze nota de plată pentru o comandă. Bucătarul este notificat de fiecare dată când trebuie să gătească pentru o comandă efectuată de ospătar.

* **Obiective secundare**
* Dezvoltarea de use case-uri si scenarii (cap. 2) – descrisă execuția programului
* Alegerea structurilor de date (cap. 3) – descrise structurile care pot implementa facil problema
* Împărțirea pe clase (cap. 3) – clasele ce vor dezvolta implementările
* Dezvoltarea algoritmilor (cap. 4) – implementarea algoritmilor & metodelor
* Implementarea soluției (cap. 4)
* Rezultate (cap. 5) – verificare
* **Analiza problemei, modelare, scenarii, cazuri de utilizare**

Pentru a ne asigura că restaurantul este ”format bine”, am folosit tehnica de programare design by contract. Pentru a notifica bucătarul, am folosit design pattern-ul ”Observer”. Pentru a crea elemente compuse ce sunt formate din alte elemente din meniu, am folosit design pattern-ul ”Composite”. Pentru a salva comenzile împreună cu produsele comandate, am folosit un HashMap. Pentru a păstra conținutul meniului și de asemenea comenzile, am folosit serializarea. Pentru a facilita realizarea aplicației, am stratificat arhitectura în pachete: dataLayer, bll (business logic), presentation, start.

**Diagrama use-case:**

Adăugare, ștergere, editare de produse din meniu

Administrator

Creare de comenzi

Generarea notei de plată

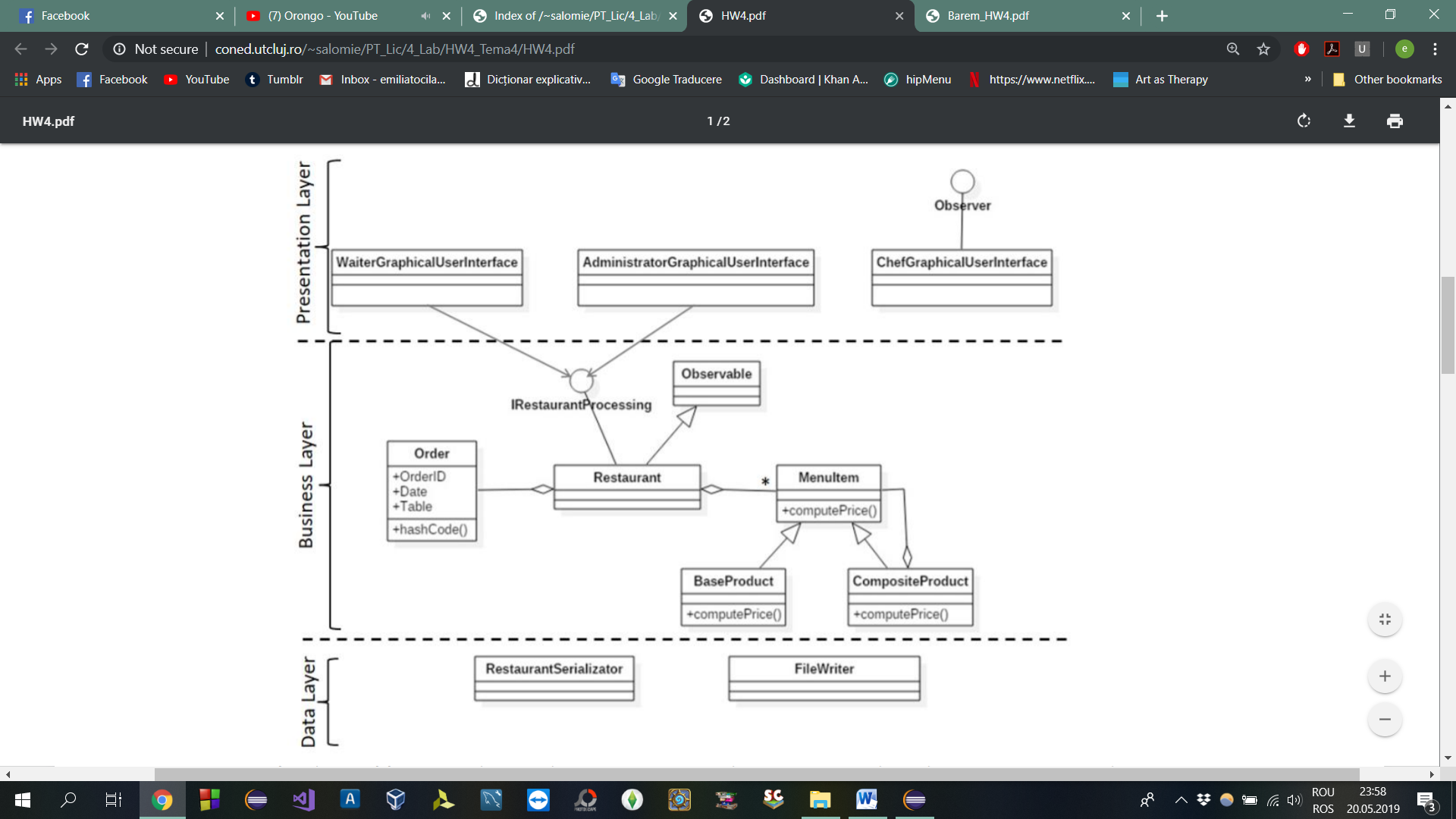
Chef

Waiter

**Scenariu:**

1. Administratorul creează, șterge sau modifică elementele din meniu.
2. Ospătarul alege produsele ce vor fi alese pentru noua comandă, plasează comanda.
3. Bucătarul este notificat cu privire la produsele ce trebuie pregătite.
4. Ospătarul generează crearea notei de plată.

* **Proiectare**
* **Diagrama UML**



* **Decizii de proiectare**

În pachetul bll există clasele BaseProduct, CompositeProduct, MenuItem, Order, Restaurant și RestaurantProcessing. Pachetul dataLayer conține clasa FileWriter. Pachetul presentation: Chef, Controller, View. Pachetul start: Main. Aceste pachete ”stratificate” au câte un rol special și apelează funcții din straturile de sub acestea. Pachetul bll conține clasele ce se ocupă cu gestionarea operațiilor pe elementele din meniul restaurantului și gestionarea comenzilor și textului pentru generarea notei de plată. Pachetul presentation conține clasele care definesc interfața grafică, de asemenea și clasa Chef ce implementează interfața Observer. Pachetul dataLayer se ocupă cu serializarea ArrayList-ului și HashMap-ului.

* **Structuri de date**

Pentru a reprezenta produsele compuse (CompositeProduct), am folosit un ArrayList<MenuItem> ce conține produsele de bază. În clasa restaurant, pentru a păstra toate elementele din meniu, am folosit de asemenea un ArrayList<MenuItem>. Pentru a păstra produsele comandate fiecărei comenzi, am folosit HashMap<Order, ArrayList<MenuItem>>. Cheia va fi comanda, iar valoarea va fi un ArrayList de produse.

* **Implementare**

**Clasa Order:**

Clasa Order are câmpurile orderID, date, table de tipurile int, String, int. Clasa are getter-e și setter-e, o metodă toString și de asemenea 2 metode suprascrise: equals și hashCode. Metoda equals va returna true dacă toate câmpurile obiectului sunt la fel cu ale celuia trimis ca parametru. Metoda hashCode returnează un nou hashCode calculat din suma valorilor câmpurilor (este unic). Clasa implementează interfața Serializable.

**Interfața MenuItem:**

Această interfață are metodele add, remove, getName, getPrice și setPrice.

public void add(MenuItem item);

public void remove(MenuItem item);

public String getName();

public double getPrice();

public void setPrice(double newPrice);

**Clasa BaseProduct:**

Clasa BaseProduct implementează interfețele MenuItem, Serializable. Câmpurile acestei clase sunt: name, price (String, double). Metodele add și remove nu conțin nicio linie de cod, deoarece aceste obiecte reprezintă ”produsele de bază”. Metodele getName, getPrice și setPrice sunt implementate precum niște getter-e și setter-e.

**Clasa CompositeProduct:**

Clasa CompositeProduct implementează interfețele MenuItem și Serializable. Câmpurile acestei clase sunt: name, price, baseProducts (String, double, ArrayList <MenuItem>). Câmpul baseProducts reprezintă elementele din care este alcătuit produsul compus, de tipul MenuItem. Metoda computePrice setează prețul obiectului de tipul CompositeProduct ca suma produselor din care este alcătuit. În constructorul clasei se setează câmpurile cu parametrii trimiși și se apelează metoda computePrice. Metodele add și remove adaugă sau șterg din ArrayList produsul trimis ca parametru. Metodele getName, getPrice și setPrice sunt implementat în același fel ca la clasa BaseProduct.

**Interfața RestaurantProcessing:**

Această interfață are metodele add, remove, edit, createOrder, computePrice, generateBillContent.

public void add(MenuItem item);

public void remove(String name);

public void edit(String name, double newPrice);

public void createOrder(Order order, ArrayList<String> newProducts);

public double computePrice(Order order);

public List<String> generateBillContent(int orderID);

**Clasa Restaurant:**

Clasa Restaurant extinde clasa Observable și implementează interfața RestaurantProcessing. Câmpurile acestei clase sunt statice : public static ArrayList<MenuItem> items; public static HashMap<Order, ArrayList<MenuItem>> itemsOfOrders;. ArrayList-ul conține toate produsele din meniu. HashMap-ul conține maparea produselor la comenzile acestora.

Metoda copyMenuItem returnează deep copy-ul ArrayList-ului ce conține toate produsele din meniu, cu scopul de a ușura ștergerea unui element folosind for-each-ul. Metoda wellFormed returnează un Boolean cu privire la faptul dacă restaurantul este format bine sau nu. Există niște condiții impuse ce trebuie verificate la fiecare operație, înainte și după acestea. (ex: să existe produse, să existe un număr între 3 și 150 de produse, să nu existe mai mult de 50 de comenzi, prețul produselor să fie mai mare decât 0 etc).

Toate metodele care efectuează operații pe ArrayList apelează metoda static FileWriter.serializationArrayList, iar toate metodele care efectuează operații pe HashMap apelează funcția FileWrite.serializationHashMap. Scopul acestor metode este de a suprascrie fișierele binare cu noile informații introduse.

Metoda add adaugă în ArrayList produsul trimis ca parametru dacă prețul acestuia este mai mare decât 0. Metoda remove șterge din ArrayList elementul dacă există mai mult de 3 produse. Metoda edit setează un nou preț unui produs, în cazul în care prețul este mai mare decât 0. Metoda createOrder adaugă în hashMap o legătură între un ArrayList de produse (produsele comandate) și comanda nouă. Când s-a creat o noua comandă se apelează metodele setChanged() și notifiyObservers(order) pentru a semnala bucătarului că trebuie să înceapă prepararea.

Metoda computePrice primește ca parametru o comandă și calculează prețul total, final al comenzii, însumând prețurile tuturor produselor comenzii.

Metoda generateBillContent returnează o listă de String-uri (liniile din bonul fiscal). Bonul va avea o structură de felul :

BON FISCAL

12.05.2019

Paine..................€1.0

Apa..................€1.5

Salata ..................€0.5

Supa..................€1.25

TOTAL..................€4.25

**Clasa FileWriter:**

Clasa FileWriter conține metodele ce serializează și deserializează ArrayList-ul și HashMap-ul. Metodele care serializează scriu datele într-un fișier binar. Metodele care deserializează pun în structurile noastre datele binare stocate în fișiere. Astfel, la închiderea și deschiderea programului, datele noastre sunt salvate.

**Clasa Chef:**

Clasa Chef conține un JFrame ca și câmp. Metoda update este suprascrisă, astfel încât atunci când bucătarul este notificat, pe ecran pa apărea un message dialog unde sunt specificate: id-ul comenzii, data, masa și produsele ce trebuie preparate.

**Clasa View:**

Se ocupă cu aspectul interfeței cu utilizatorul. Interfața va avea 2 tab-uri: Administrator și Waiter care vor fi adăugate unui JTabbedPanel.

Metoda generateTable primește un ArrayList<? extends Object> (o listă de obiecte de un tip generic) și returnează un JTable.

String[] columnNames = new String[list.get(0).getClass().getDeclaredFields().length];

Object[][] data = new Object[list.size()][list.get(0).getClass().getDeclaredFields().length];

Vectorul de String-uri va columnNames va conține numele câmpurilor (header-ul tabelei), ce sunt obținute prin reflexie parcurgând câmpurile clasei.

Obiectele, cu câmpurile sale, vor fi obținute prin reflexie astfel: se parcurge lista de obiecte, apoi încă într-un for each se parcurg câmpurile obiectului, se setează acestea ca fiind accesibile și se încearcă scrierea în matricea data la rândul și coloana corespunzătoare obiectul respective (fie el Integer, String etc).

Pentru a seta prima celula să nu fie editabilă (cea cu id-urile), se suprascrie metoda isCellEditable din clasa DefaultTableModel. Se returnează tabelul obținut.

Metoda createTable(A, W) crează un tabel astfel: se invocă metoda generateTable ce primește ca parametru Array-ul cu produse sau comenzi, iar tabelul rezultat este introdus într-un JScrollPane pentru a face posibilă vizualizarea header-ului. La final, JScrollPane-ul se adaugă JPanel-ului corespunzător clasei (administratorPanel, waiterPanel).

Prin metoda init se inițializează cele 2 taburi, invocând la rândul ei metodele init(A, W). De asemenea se adaugă butoanele corespunzătoare (Show all, Add, Delete, Edit, Create New Order, Compute Bill).

Metoda clearPane șterge conținului JPanel-ului claselor (administratorPanel etc).

De asemenea, clasa view conține getter-e și setter-e (set---, getMainPane, getJTable) și metode ce crează listener pentru butoane.

**Clasa Controller:**

Se ocupă cu corelarea logicii din spatele programului cu partea cu care utilizatorul va interacționa direct. Clasele ---Listener (--- : showAll-, insert-, delete-, update- (A, W)) care implementează ActionListener au metoda actionPerformed, cu parametrul ActionEvent e, fac întocmai acest lucru.

Metoda din clasele showAllListener- apelează metoda clearPane și reinițializează panel-urile.

Metoda din clasa insertListenerA afișează un JOptionPane (option dialog) unde utilizatorul alege dacă acesta dorește să introduce un produs de bază sau unul compus. Dacă produsul este compus, utilizatorul a ales obiectele de bază ce compun noul produs. Se deschide un JDialog unde utilizator introduce datele noului obiect, se apelează metoda insert din clasa bll, și se reinițializează tabelele.

Metoda din clasa deleteListenerA funcționează astfel: utilizatorul selectează cu un click rândul obiectului pe care dorește să îl șteargă, apasă butonul și obiectul este șters.

JTable table = view.getJTable("administrator");

int []rows = table.getSelectedRows();

if (rows.length != 0){

for(int i = 0; i < rows.length; i++)

restaurant.remove(table.getValueAt(rows[i], 0).toString());

view.clearPane();

view.init();

}

else

JOptionPane.showMessageDialog(view.getMainPane(), "Select an item to be deleted!", "Try again", JOptionPane.PLAIN\_MESSAGE);

Metoda din clasa updateListenerA funcționează astfel: utilizatorul selectează rândul pe care dorește să îl actualizeze și cu un dublu click, selectează celula cu prețul pentru modificare, o modifică și apasă enter. După acest lucru, apasă butonul de update.

JTable table = view.getJTable("administrator");

int row = table.getSelectedRow();

if (table.isRowSelected(table.getSelectedRow())){

if (table.isEditing())

table.getCellEditor().stopCellEditing();

restaurant.edit(table.getValueAt(row, 0).toString(), Double.parseDouble(table.getValueAt(row, 1).toString()));

view.clearPane();

view.init();

}

else

JOptionPane.showMessageDialog(view.getMainPane(), "Select a product to be updated!", "Try again", JOptionPane.PLAIN\_MESSAGE);

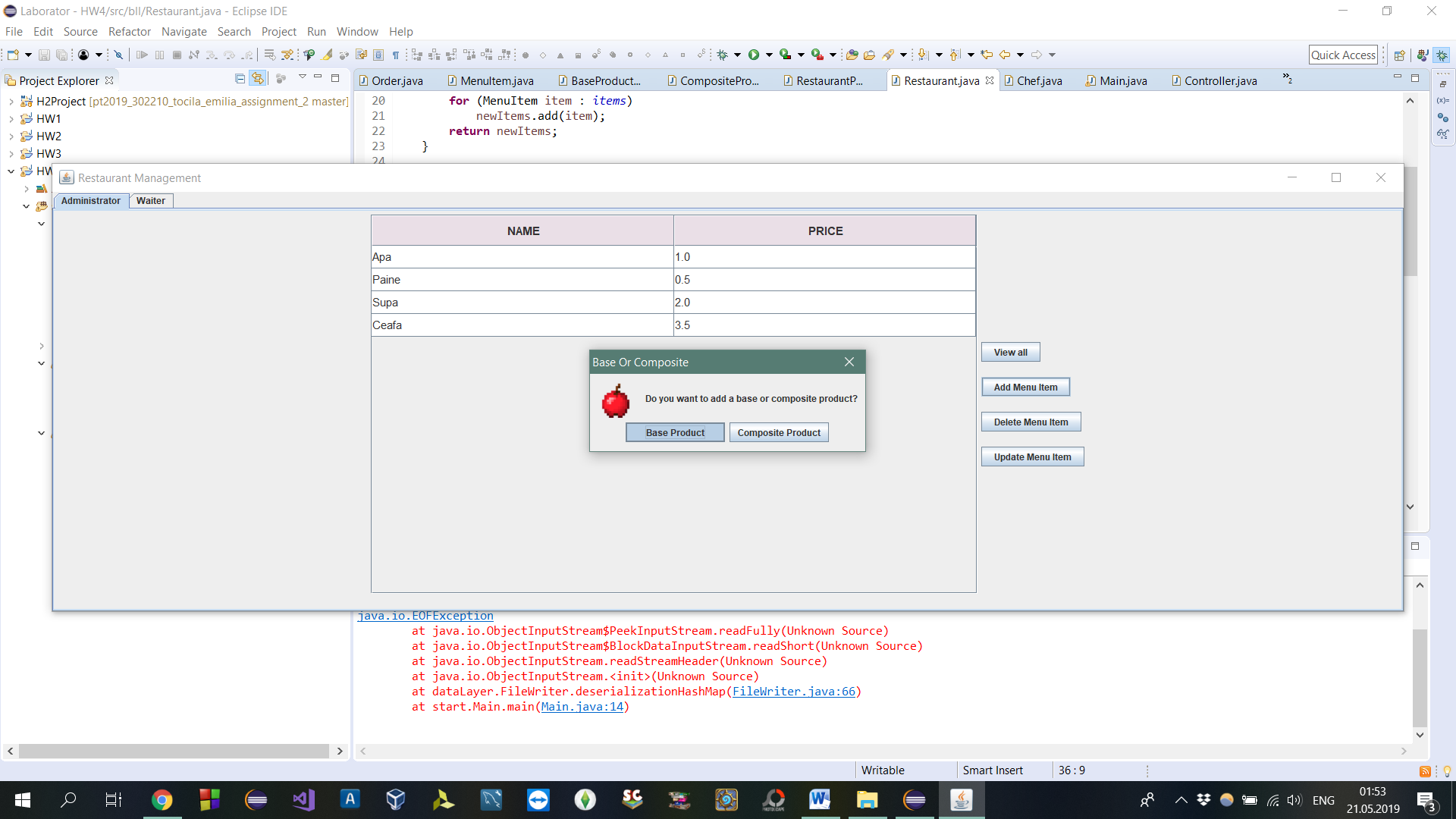
Metoda din clasa insertListenerW este implementată astfel: ospătarul alege produsele care formează noua comandă, apasă butonul Create New Order și introduce datele referitoare la noua comandă.

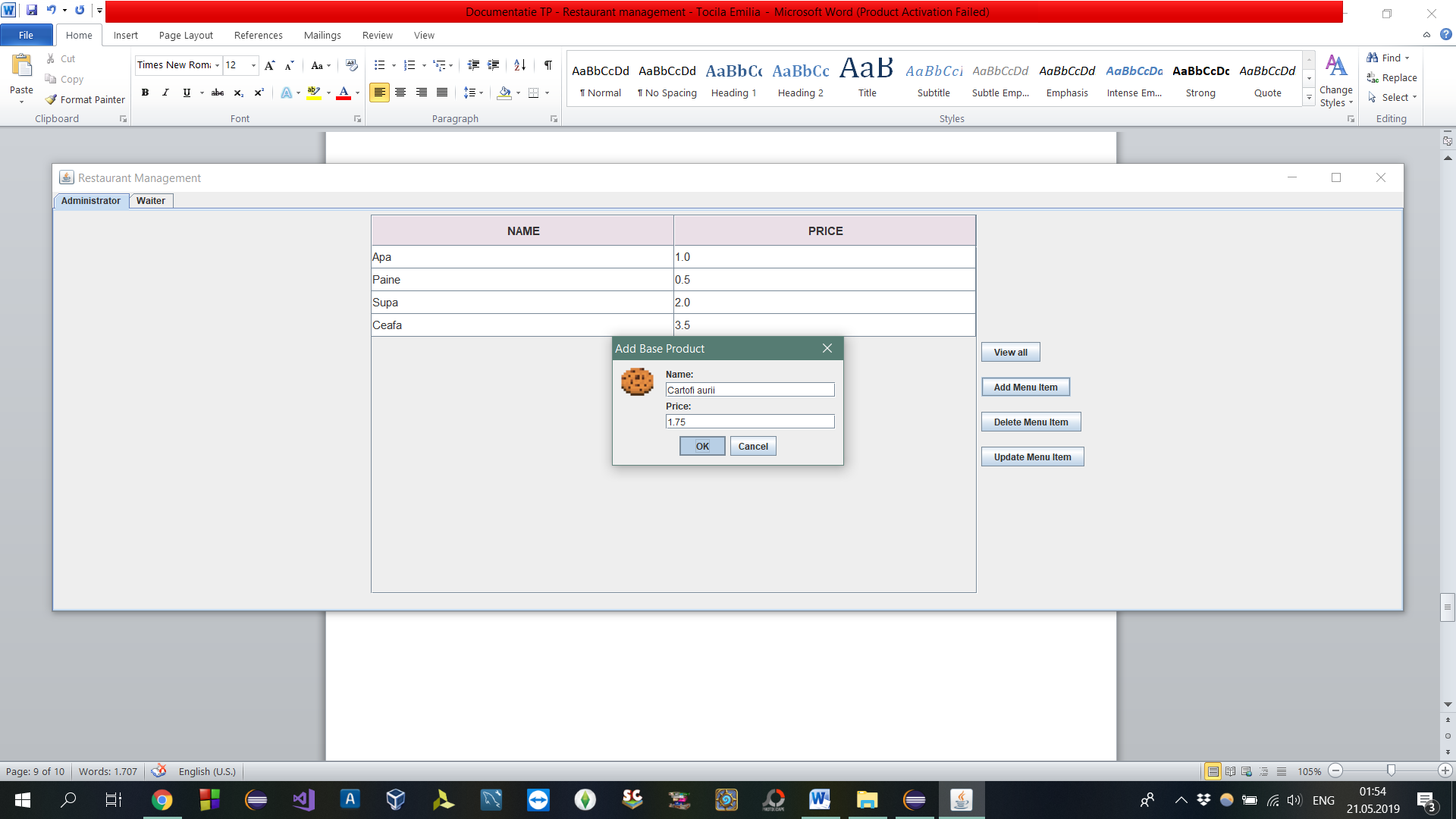
Metoda din clasa computeBillListenerW creează fișiere text ce conțin nota de plată aferentă fiecărei comenzi.

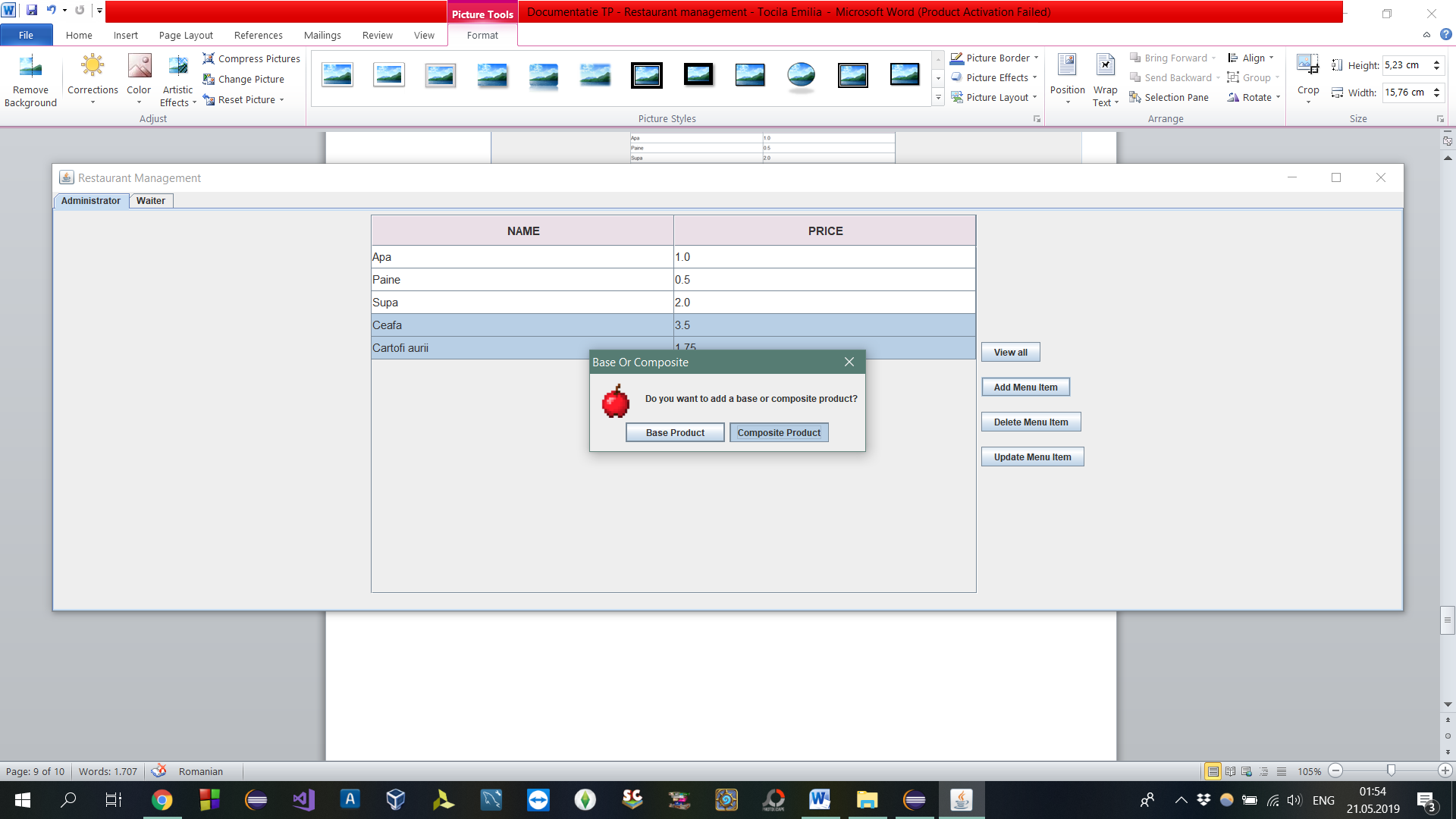
**Clasa Main:**

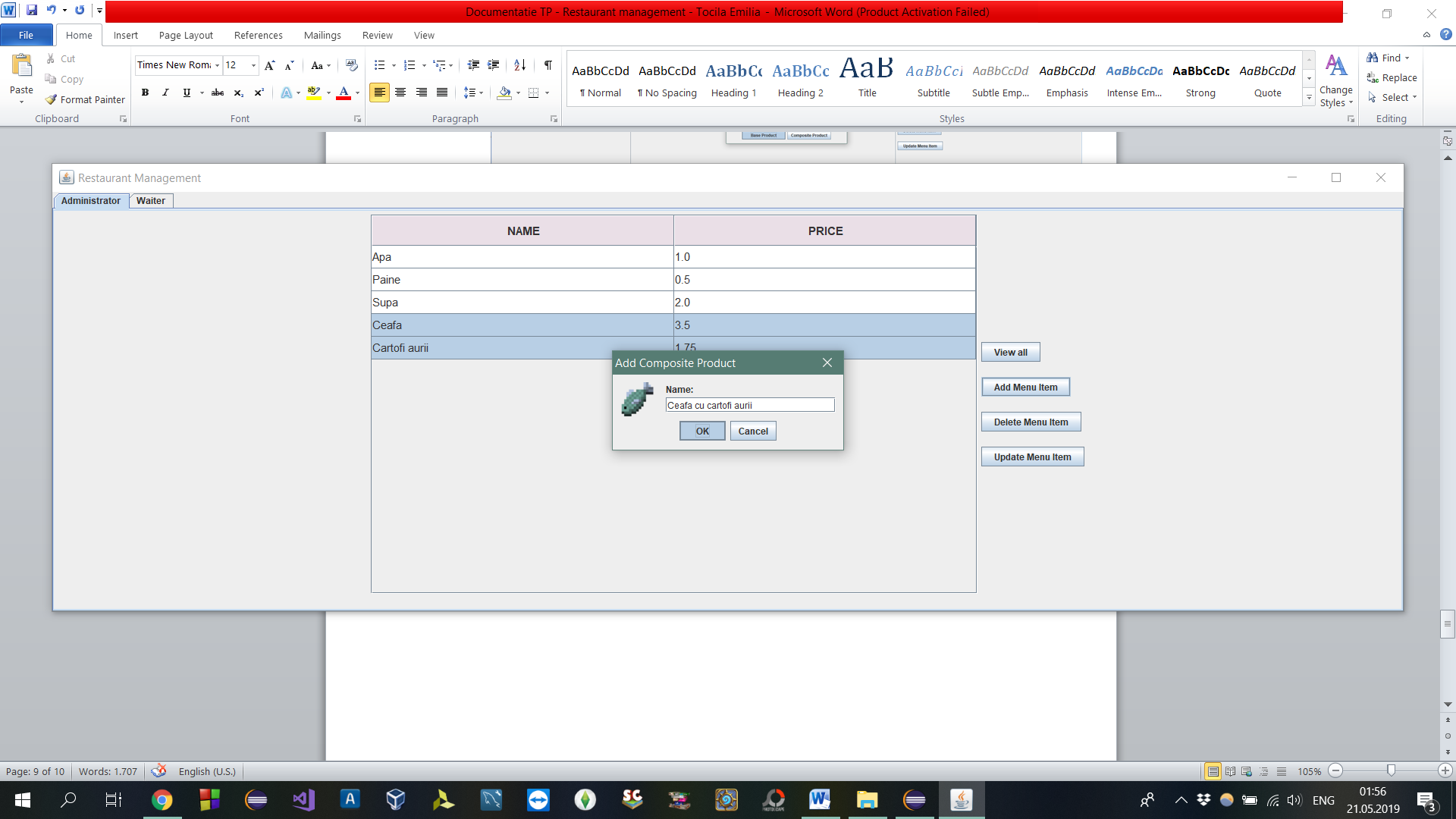
Clasa Main leagă View-ul de Controller și de model. Sunt instantiate obiecte din clasele Restaurant, Chef și FileWriter. Sunt deserializate ArrayList-ul și HashMapul. Este apelată metoda addObserver (restaurant.addObserver(chef); ).

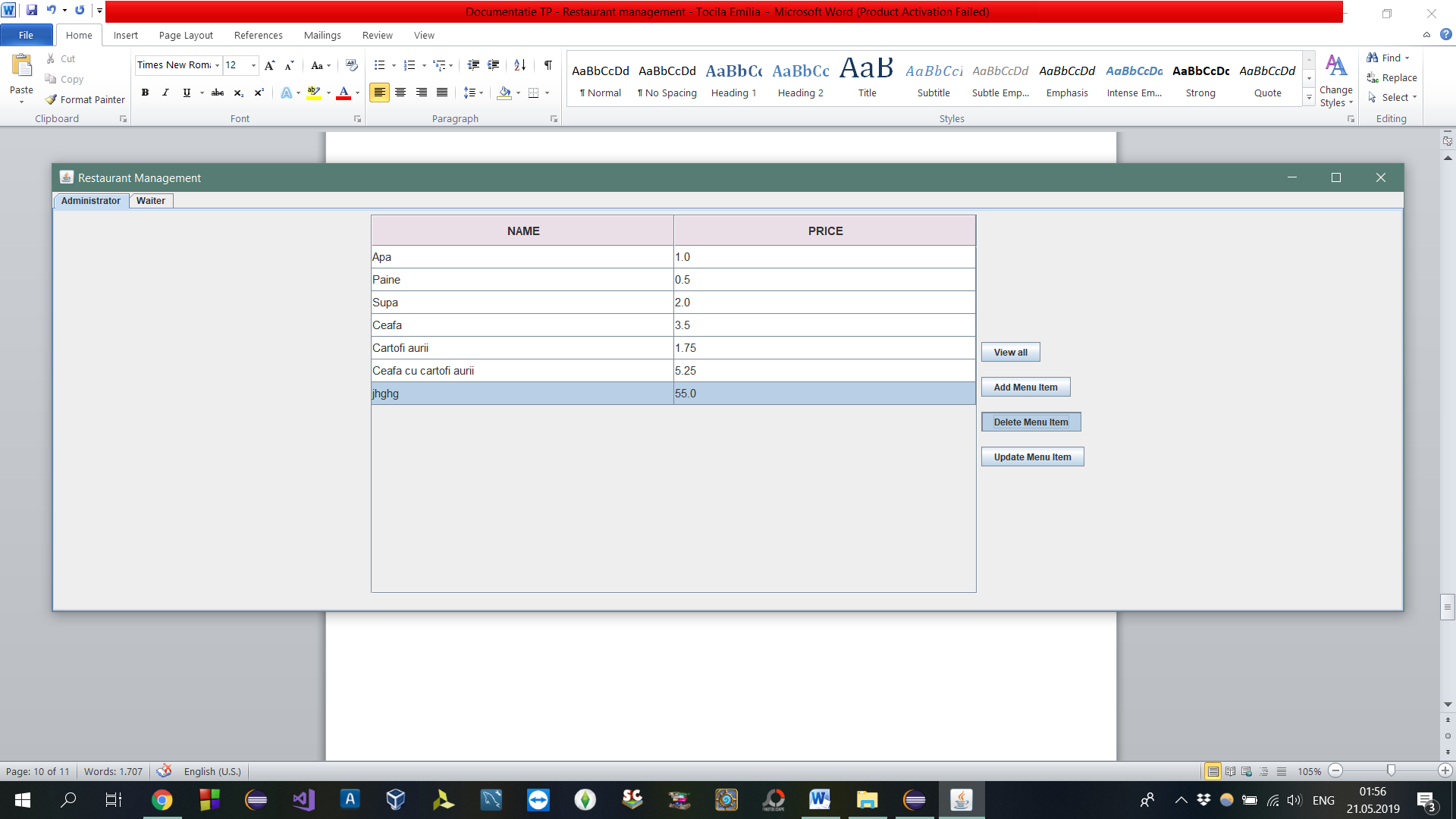
* **Rezultate**

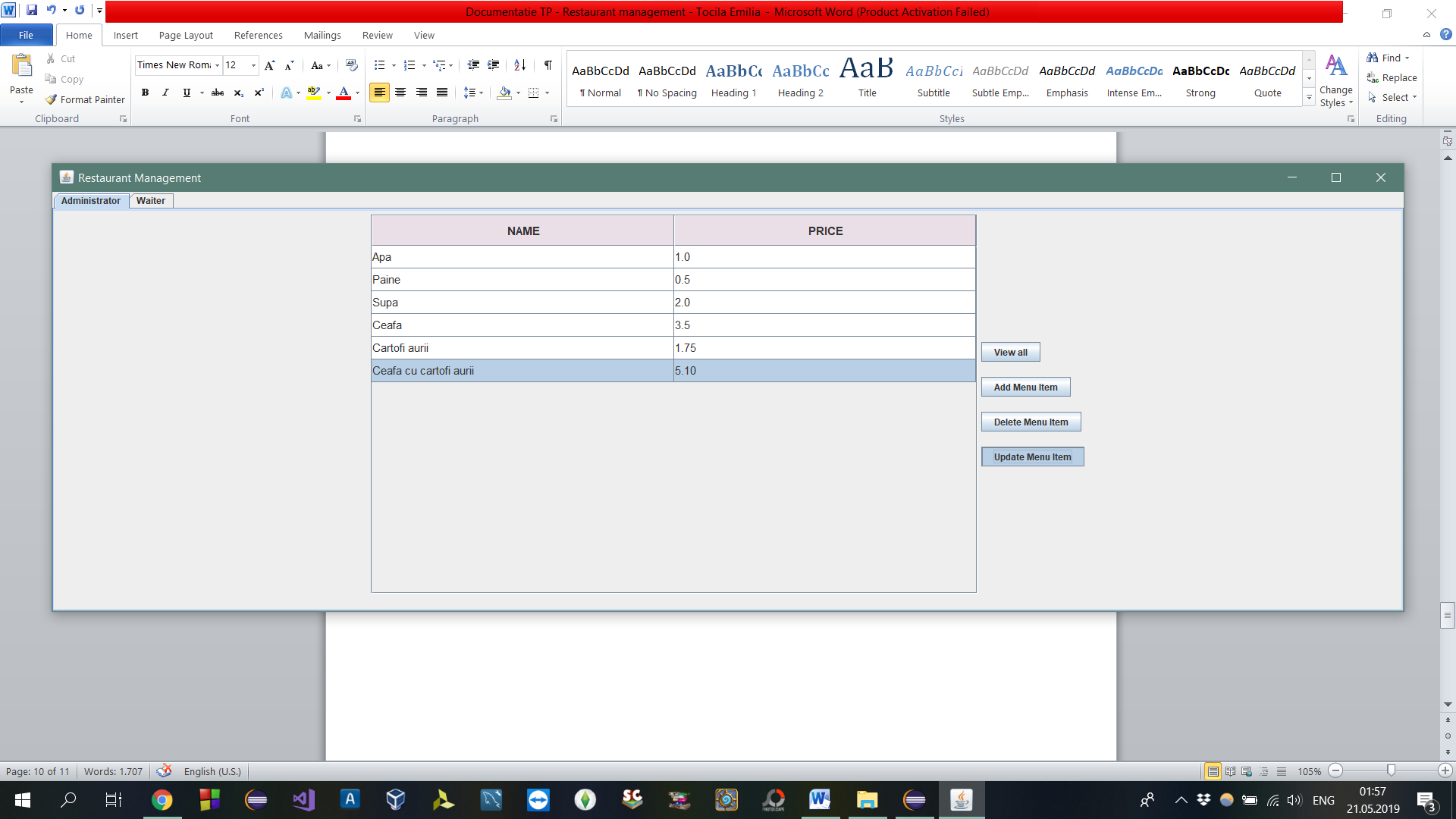


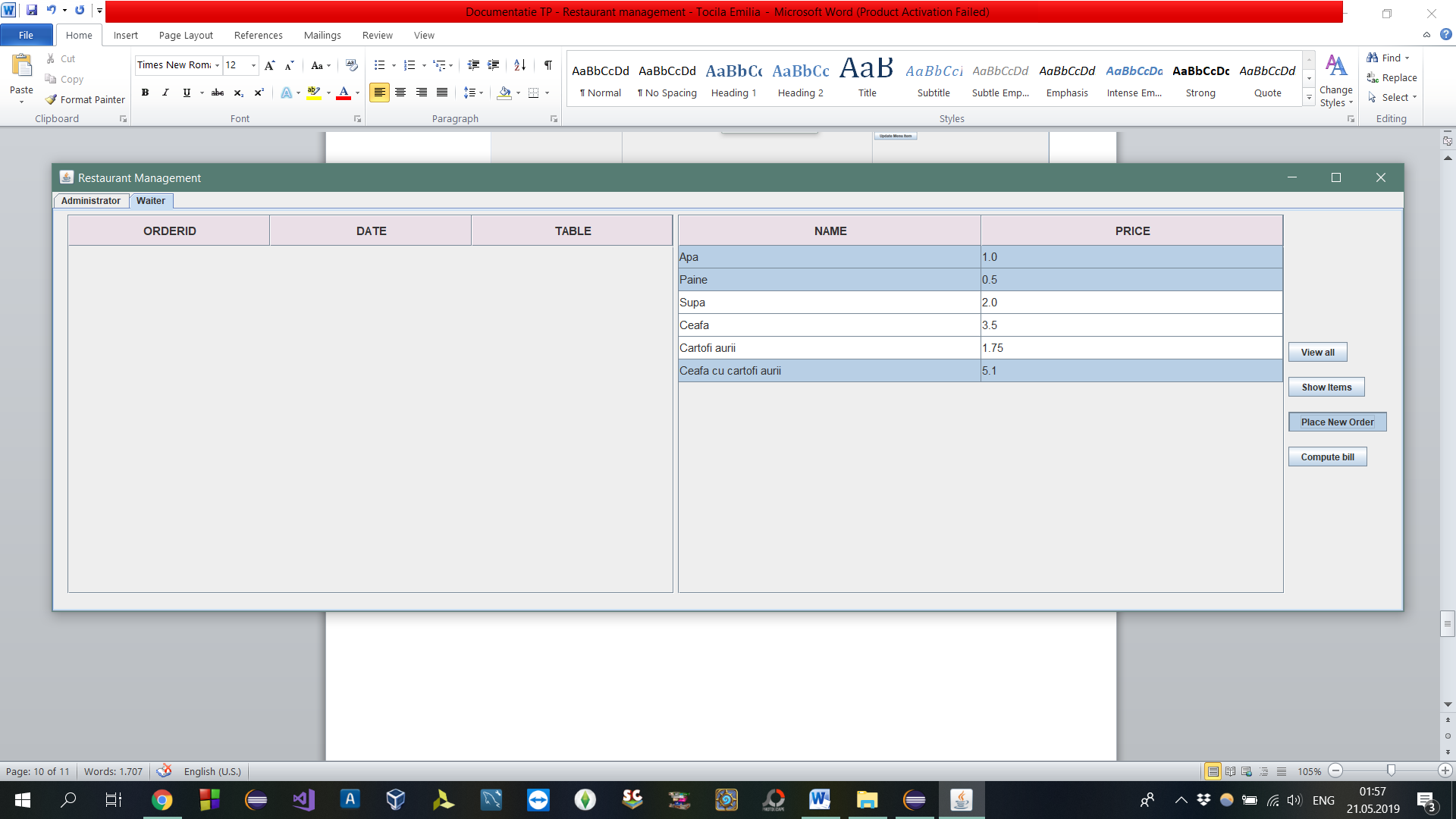


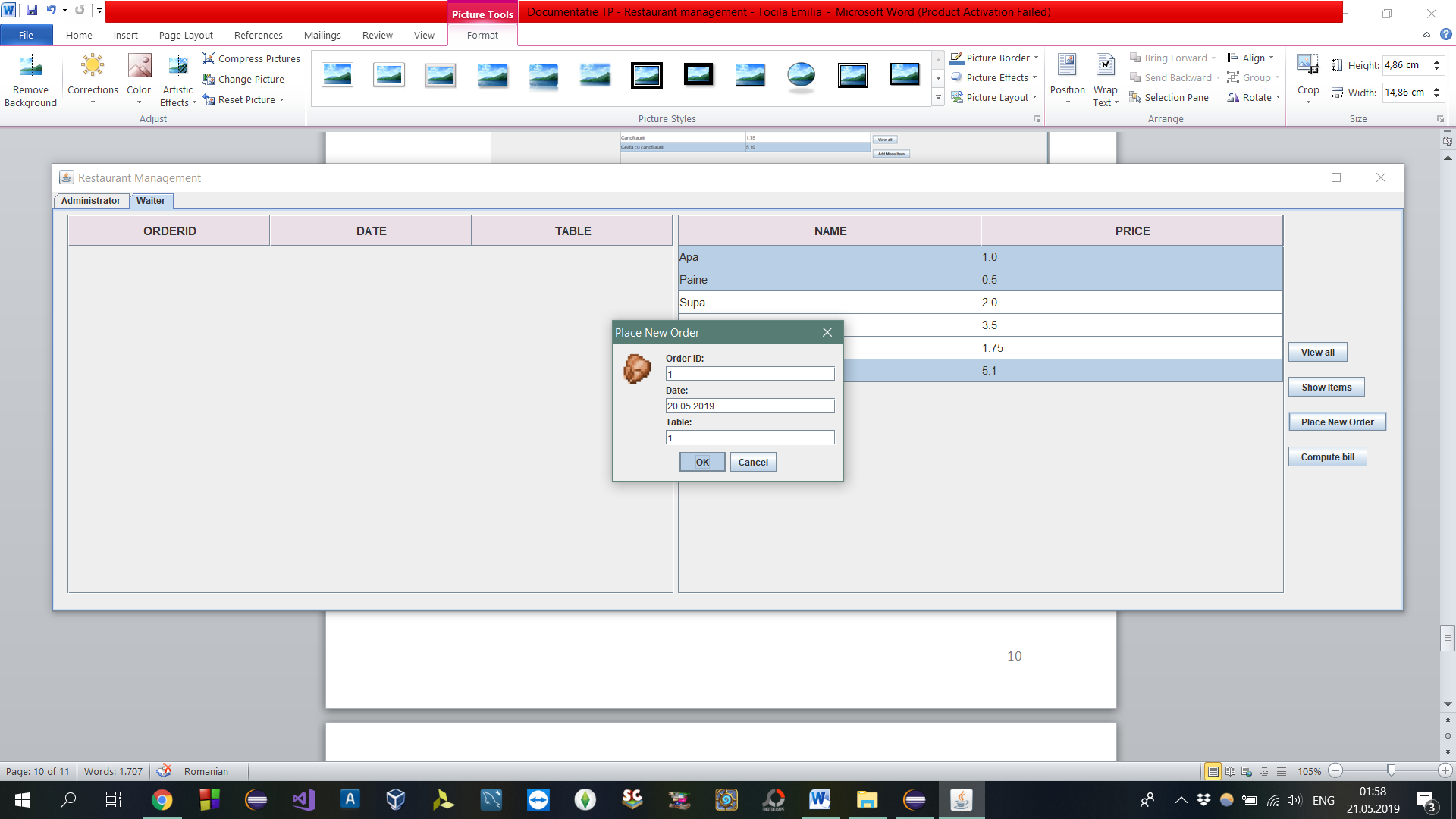


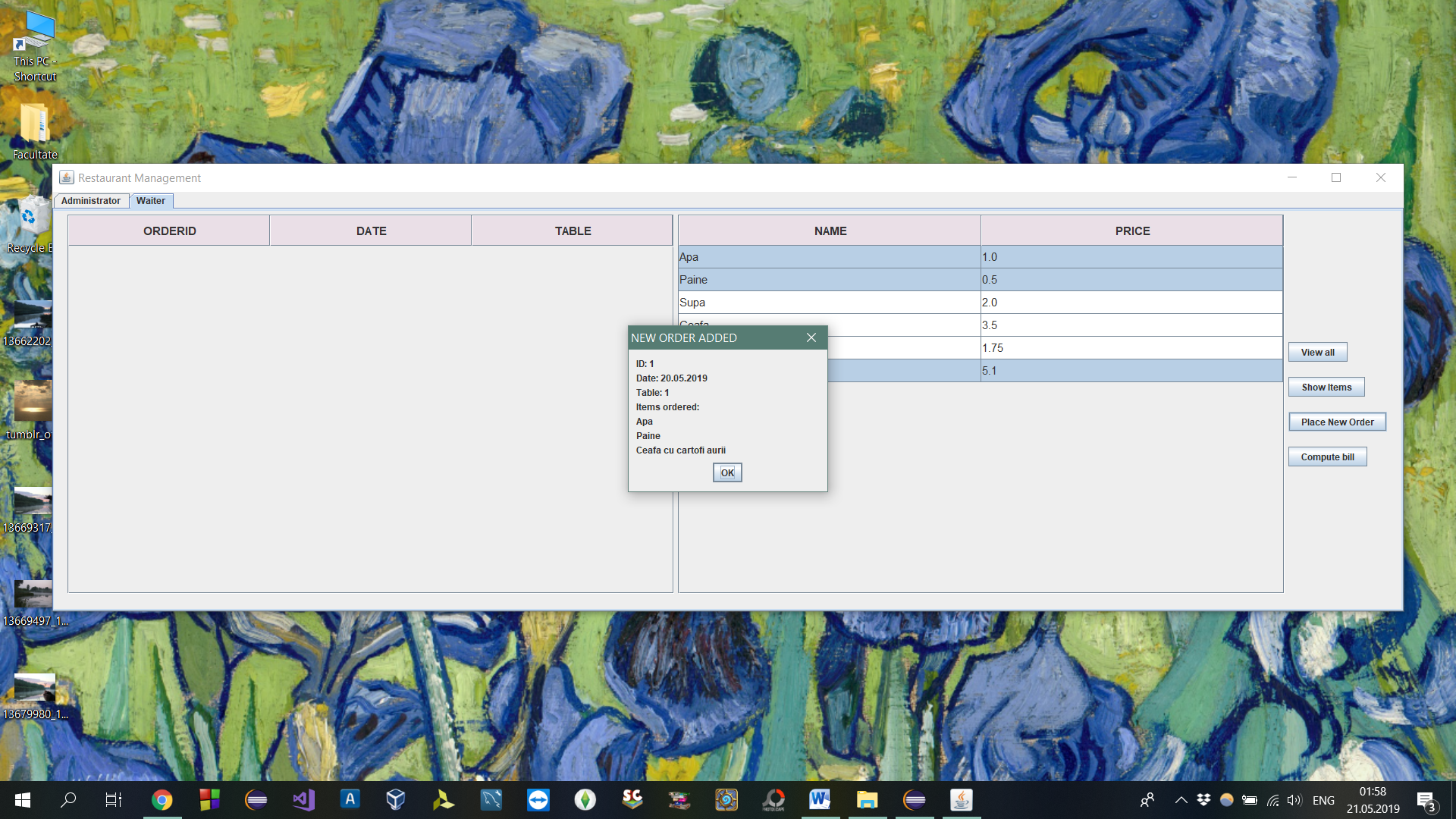




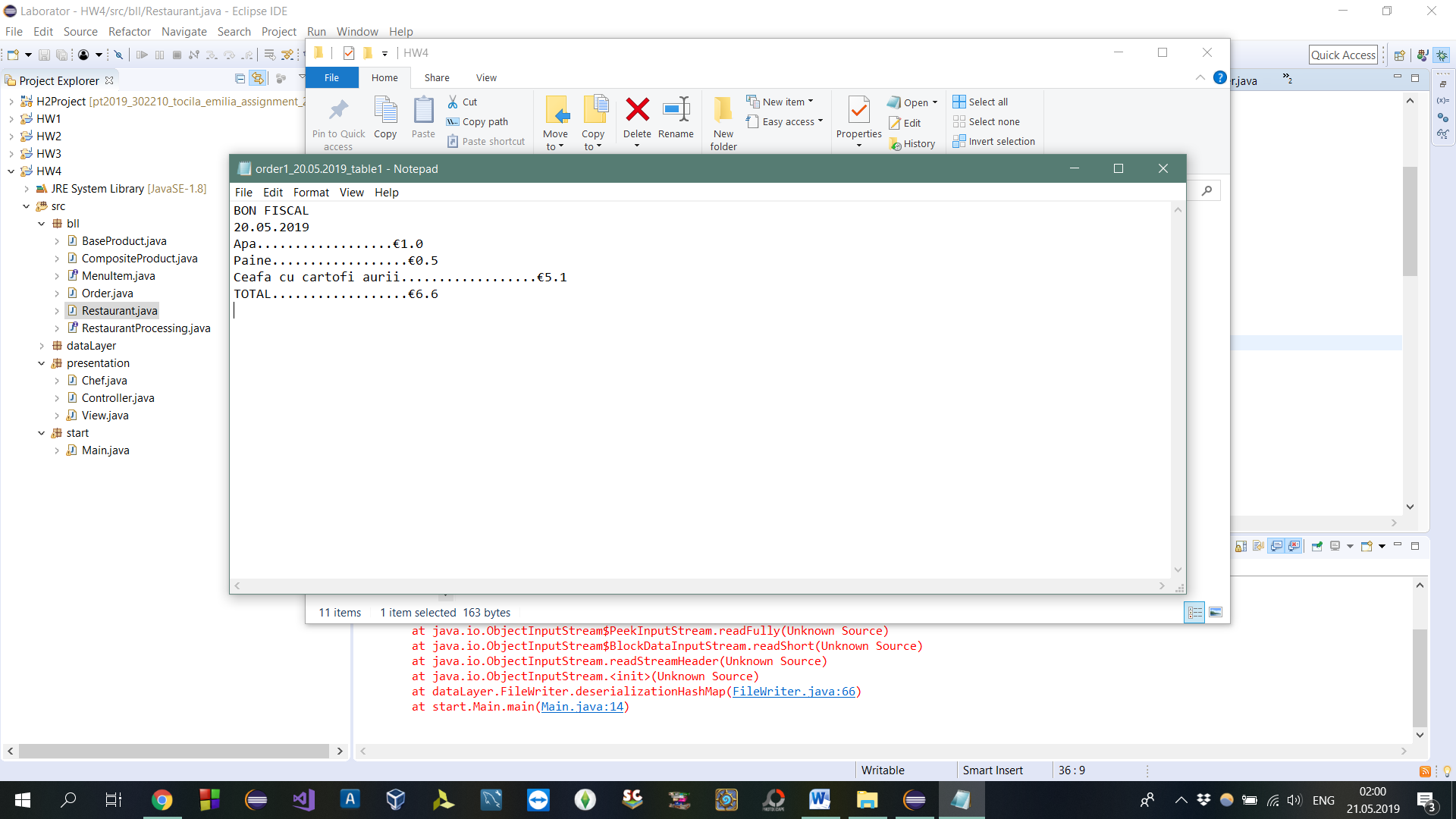












* **Concluzii**

Prin acest proiect am învățat cum să folosesc noi tehnici de programare, am

aprofundat noțiunile despre JCF (ArrayList, HashMap). Am aflat ce este serializarea și cât de folositoare este aceasta. Am identificat similarități între lucrul cu baza de date și folosirea fișierelor binare.

* **Bibliografie**

<https://stackoverflow.com/questions/29345792/java-jtable-getting-the-data-of-the-selected-row>

<https://docs.oracle.com/javase/7/docs/api/javax/swing/JTable.html>

<https://stackoverflow.com/questions/6555040/multiple-input-in-joptionpane-showinputdialog/6555051>

<https://stackoverflow.com/questions/33739941/read-cell-value-in-jtable-while-it-is-being-edited>

<https://stackoverflow.com/questions/35414482/get-column-value-from-jtable-after-table-is-been-edited>

<https://stackoverflow.com/questions/2885173/how-do-i-create-a-file-and-write-to-it-in-java>

<https://howtodoinjava.com/java/collections/arraylist/serialize-deserialize-arraylist/?fbclid=IwAR1yqubSwPqSQ5LO4fTmgewggy2k36gTzO8c6yj9PRNMb51VEp6sxr2gJTU>

<http://www.java2s.com/Code/JavaAPI/java.util/extendsObservable.htm?fbclid=IwAR2bvrrjhVi-c6zHdagM6udZlR_ji0hyGdxTgs3EPHHFP7lzKL1EItyXGuw>

Hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho hi ho