Universitatea Tehnică Cluj-Napoca

Restaurant Management System

~ Tema 4 ~

Stroe Mădălina Ionela

Grupa 302210

Semigrupa2

CUPRINS

- Obiectivul temei
- Analiza problemei, modelare, scenarii și cazuri de utilizare
- Proiectare
- Implementare
- Rezultate
- Concluzii
- Bibliografie

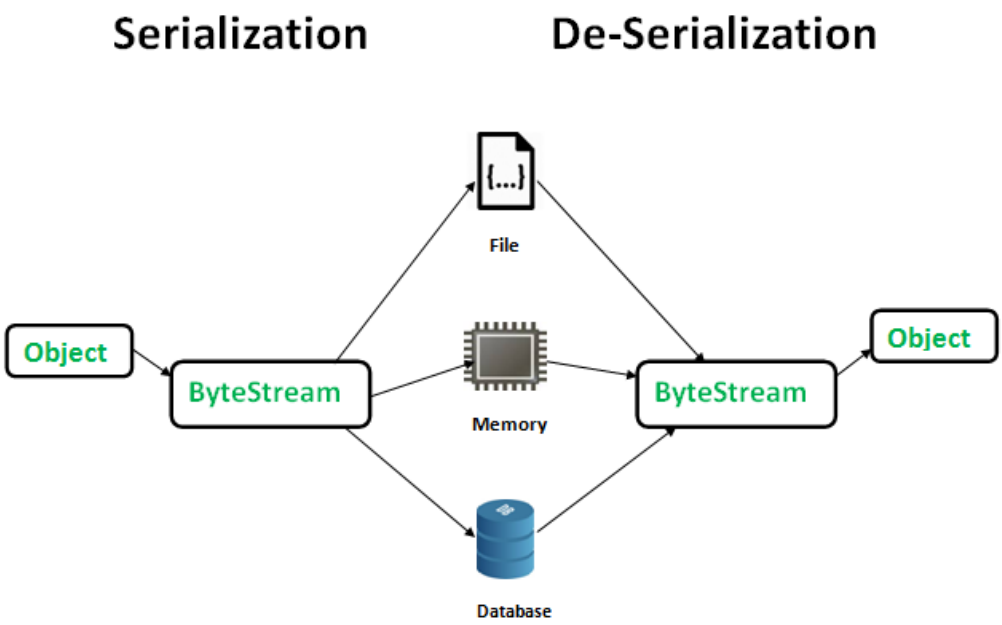
~ Obiectivul temei ~

Obiectivul acestei teme este de a realiza un sistem de manageriere a unui restaurant. Acest sistem are 3 tipuri de utilizatori: administrator, waiter și chef. Administratorul poate să adauge în meniu, să șteargă din meniu sau să modifice în meniu. Waiter-ul poate să creeze o comandă nouă, adăugând elemente din meniu, poate de asemenea să calculeze nota de plată pentru o comandă. Chef-ul este anunțat de fiecare data cand primește o comandă nouă.

Această temă îsi propune să ne perfecționeze abilitățile în Programare Orientată pe Obiecte și să propună utilizarea unor concepte noi precum serializare și deserializare, utilizare a Composite Design Pattern, Observer Design Pattern, lucrul cu HashTable, JTable, invarianți, etc.

Un concept nou și important pe care il vom aborda în această temă este serializarea, implicit deserializarea unui obiect.

În forma cea mai simplă, serializarea înseamnă salvarea și restaurarea obiectelor. Obiectele oricărei clase care implementează interfața Serializable, pot fi salvate într-un Stream(flux de date) și restaurate din acesta. Pachetul java.io conține două clase speciale ObjectInputStream, respective ObjectOutputStream pentru serializarea tipurilor primitive.



~ Analiza problemei ~

Luând cazurile, considerând că accesăm ca administrator, va trebui să putem să adăugăm în meniu diferite produse, simple sau compuse. Am decis că un produs compus va fi format din 2 produse simple. Pentru fiecare dintre acestea, trebuie să se introducă denumire, preț și cantitate.

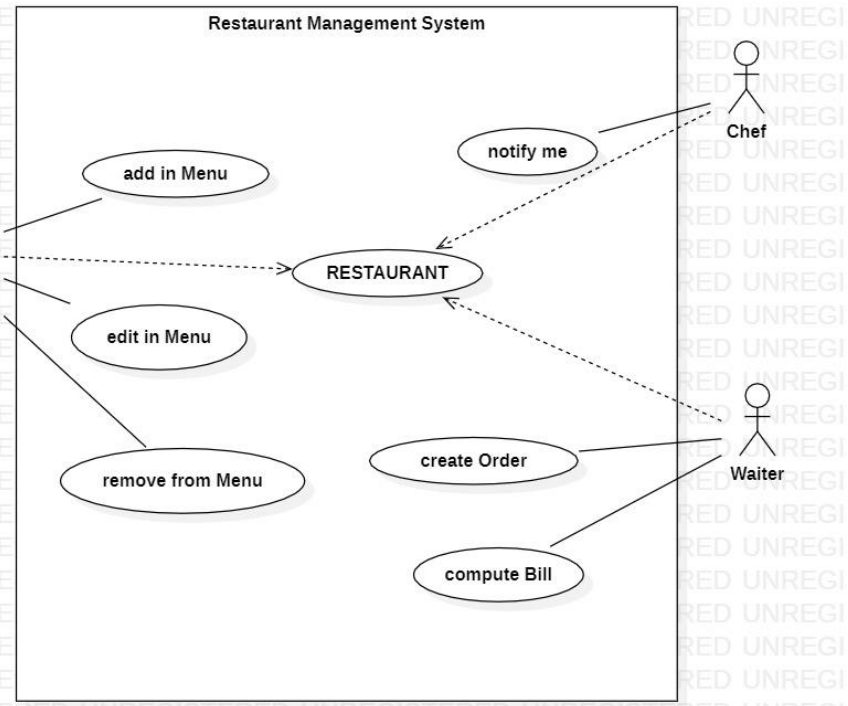
Pentru ștergerea unui produs, am decis că această operațiune se poate face după nume.

Pentru modificarea unui produs, putem alege dacă modificăm un composite sau un basic product. Putem schimba nume, preț sau cantitate, ori pe toate cele enumerate.

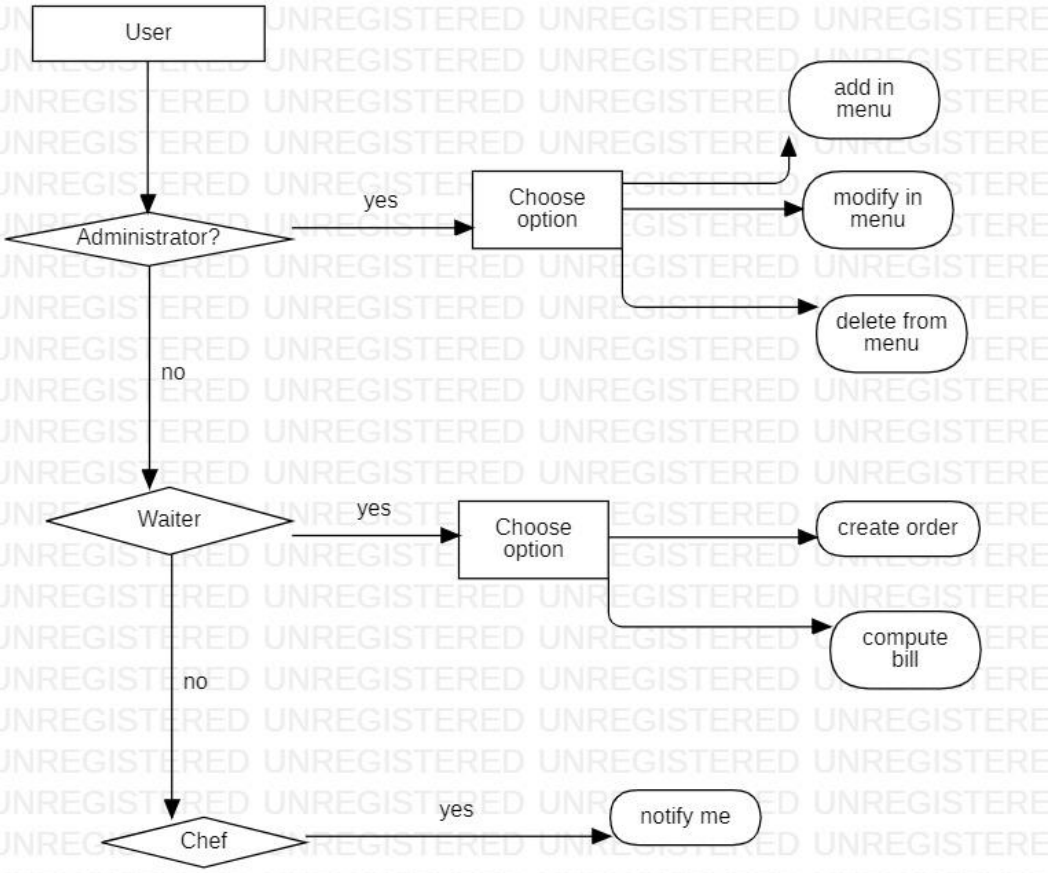
În rolul de Waiter trebuie să putem crea o comandă, adică să adăugam elemente din meniu în lista de comenzi, dar fără ca acestea să fie șterse din lista de meniuri. Meniul ar trebui să fie disponibil pentru ca u un client să știe ce opțiuni are. De asemenea, trebuie să calculăm și nota de plată pentru o comandă.

În rolul de Chef, pentru a putea procesa comanda, trebuie să se primească o notificare atunci când apare o comandă nouă.

Toate aceste operațiuni vor fi realizate prin intermediul restaurantului.



Pașii execuției sunt:



~ Proiectare ~

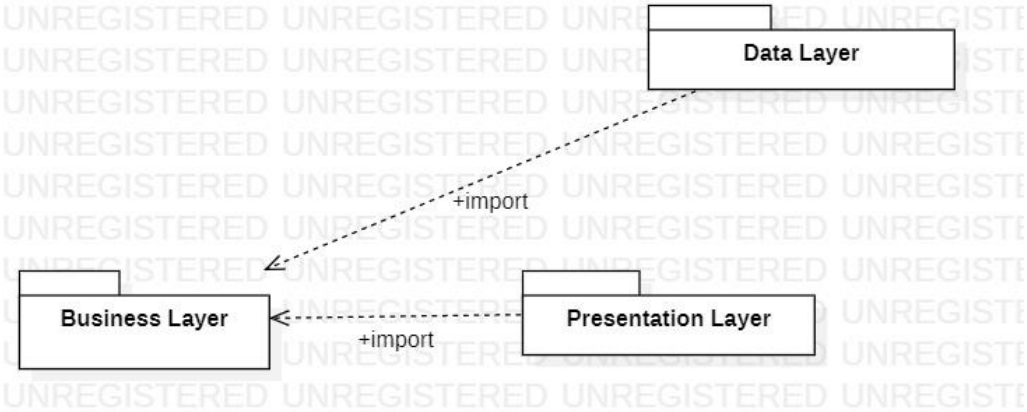
Pentru realizarea acestei teme, am luat în considerare urmaătoarele decizii de proiectare:

- Pentru selectarea utilizatorului dorit vom avea 3 butoane care ne vor duce în ferestre diferite de unde vom putea realiza mai departe cerințele
- Dacă am ajuns în interfața pentru administrator, vom avea posibilitatea să alegem ce operațiune dorim să îndeplinim: adaugă în meniu, modifică în meniu sau șterge din meniu.
    - Dacă vrem să adăugăm în meniu, atunci avem posibilitatea de a adăuga base product cu nume, cantitate, preț, sau composite product cu nume și lista de itemi din meniu.
    - Dacă vrem să modificăm în meniu, avem de ales tipul produsului de modificat și noile attribute.
    - Dacă vrem să ștergem din meniu, ștergerea va fi facută după nume, în funcție de tipul itemului.
- Dacă am ajuns în interfața waiter, avem posibilitatea de a alege dacă vrem să adaugăm o comandă sau să generăm nota de plată pentru o comanda existenă.
    - Dacă vrem să adăugăm o comandă, trebuie să decidem ce vrem să adăugăm în comandă, după tip, nume, etc.

- Dacă vrem să calculăm prețul unei comenzi, vom căuta după ID.

- Dacă am ajuns în interfața chef, suntem notificați atunci când a fost preluată o comandă.

Diagrama de pachete:



Ca structuri de date am folosit ArrayList<T>, care este un o listă/ vector a cărui dimensiune este redimensionabilă.

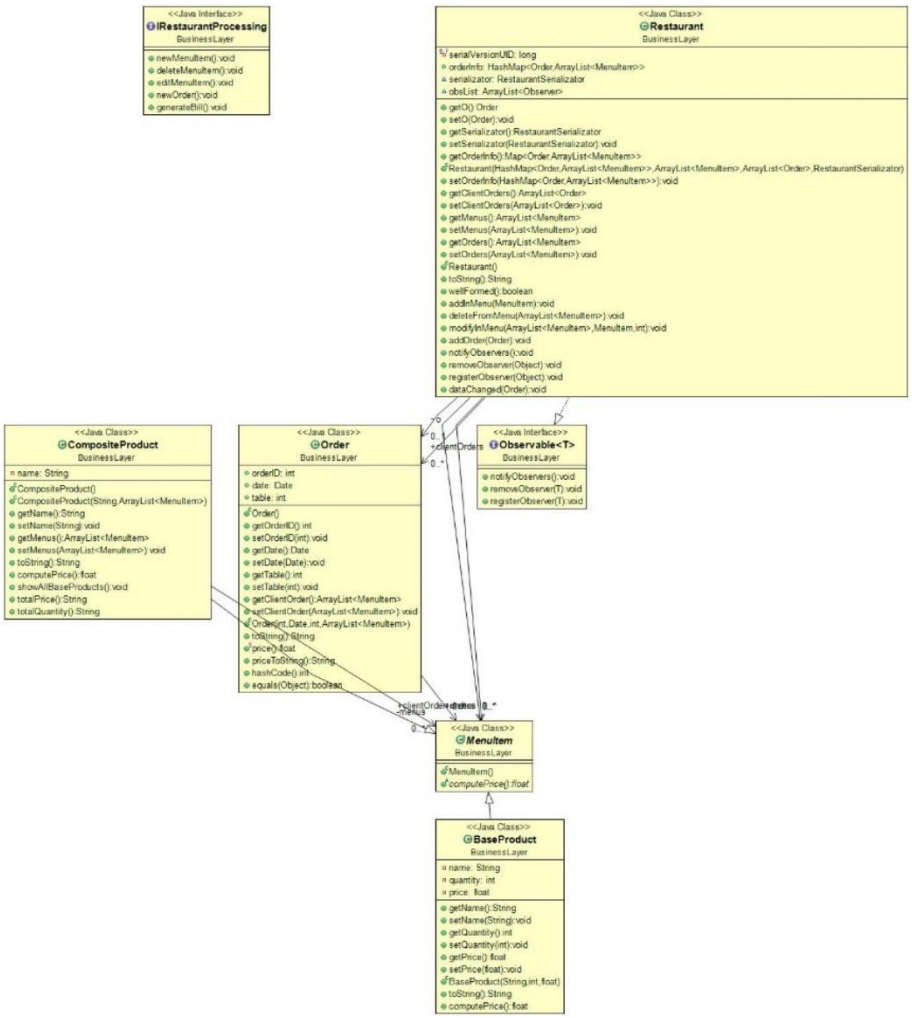
De asemenea, am folosit si HashMap care stochează elemente în perechi de forma [key, value]. Un obiect este utilizat drept *key* pentru alt obiect *value*. Pot fi stocate tipuri diferite pentru key și values sau același tip.

Algoritmii folosiți sunt de inserare în ArrayList, de ștergere din ArrayList, de modificare pe o anumită pozitie într-un ArrayList.

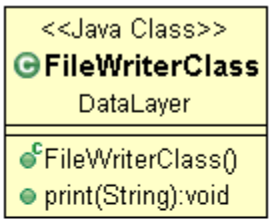
Pe de altă parte, am folosit și adăugarea elementelor într-un HashMap, căutarea unui element într-un HashMap după o anumită proprietate.

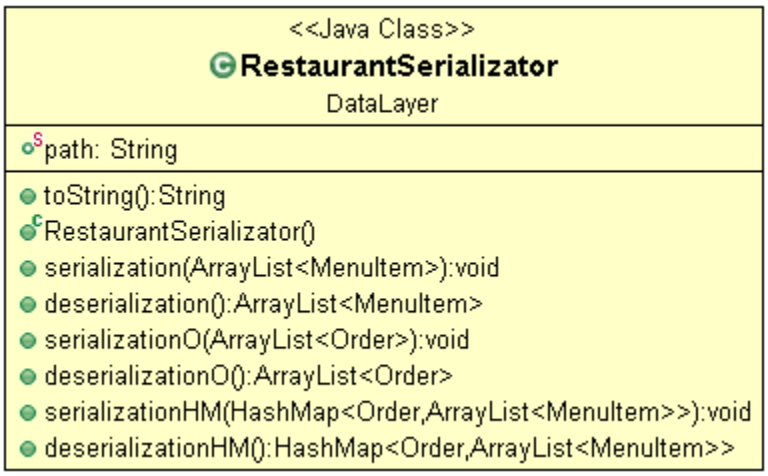
În figura de mai jos putem vedea relațiile dintre clase. Unified Modeling Language (prescurtat UML) este un limbaj standard pentru descrierea de modele si specificatii software. Diagrama de clase UML Este folosită pentru reprezentarea vizuală a claselor și a interdependențelor, taxionomiei și a relațiilor 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.

Pentru Business Layer:



Pentru Data Layer:





Pentru Presentation Layer:

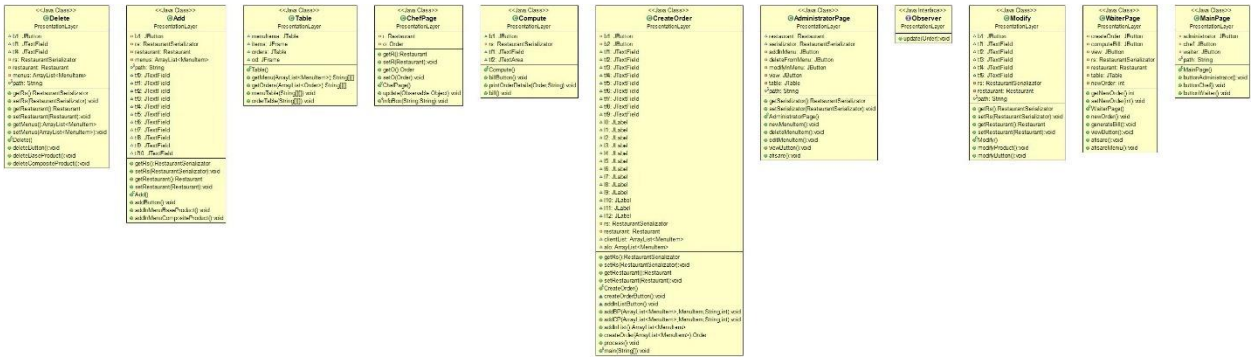
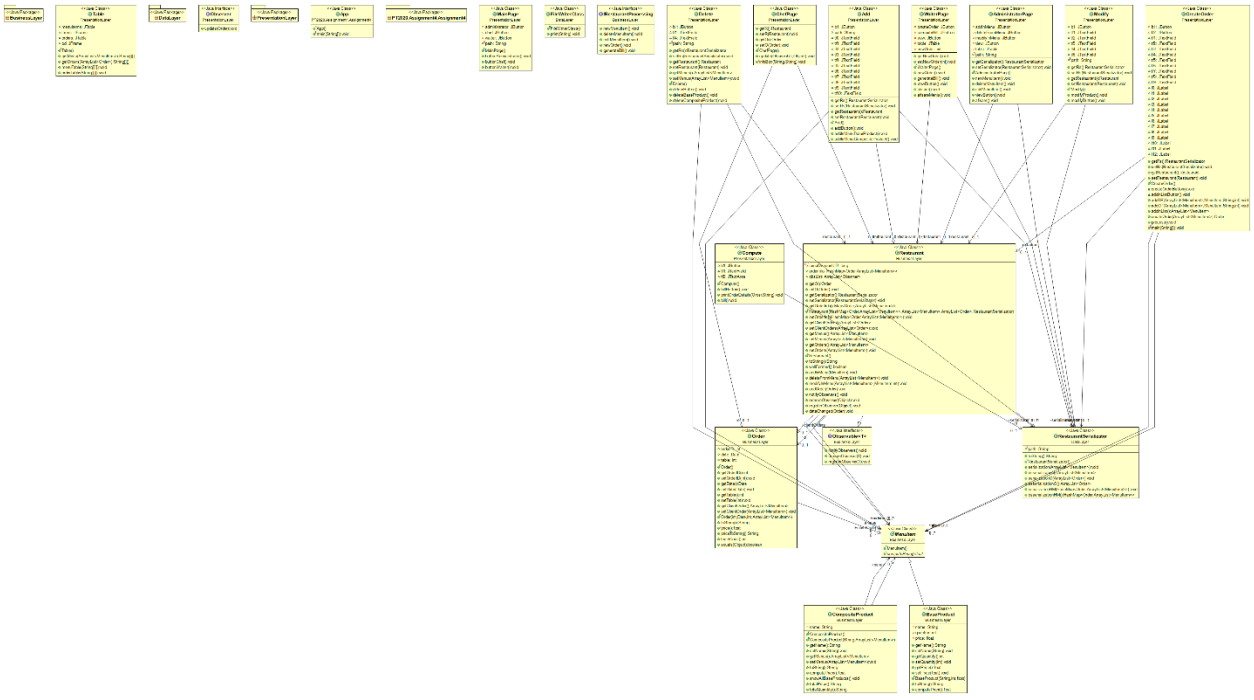


Diagrama de clase:



~ Implementare ~

În această secțiune vom lua pe rând fiecare clasă și vom discuta câmpurile și metodele importante.

Pentru clasa BaseProduct avem String nume, int quantity, float price. În acest fel vom crea base products de care ne vom ajuta în operațiunile effectuate ulterior.

Clasa CompositeProduct conține String nume și un ArrayList de MenuItem, în cazul nostru, un Composite Product va conține mai multe Base Products.

Spre deosebire de Base Product, unde prețul nu are nevoie să fie calculate pentru ca este direct introduce atunci când obiectul este creat, în cazul unui composite product avem o metoda float computePrice care calculează prețul total al Composite Product-ului în funcție de prețurile itemilor conținuți în acea listă. Tot cu ajutorul acestei abordări calculăm și cantitatea.

Clasa abstracta MenuItem conține metoda abstractă computePrice.

Interfața IRestaurantProcessing are metodele newMenuItem, deleteMenuItem, editMenuItem, newOrder, generateBill.

Intefața Observable are metodele notify, remove, register Observer.

Clasa Order are un int orderID, Date date, int table, ArrayList care va conține itemii aleși de client pentru comandă.

Tot în această clasă calculăm și prețul pentru comandă, calculând separat pentru fiecare item din lista comenzii, prețul efectiv, însumând ulterior.

Metoda hashCode ne va folosi mai târziu pentru căutarea comenzii în hashMap.

Clasa Restaurant este cea care conține metode importante pentru funcționarea acestui sistem, spre exemplu metoa addInMenu care permite adăugarea unui MenuItem în lista de meniuri a Restaurantului, apoi aplicăm serializarea.

Același pattern va fi urmat și de metoda deleteFromMenu. O mica diferență vom avea în metoda de modifyInMenu, unde vom modifica în listă, pe o anumită poziție, un element deja existent.

Metoda addOrder este utilizată pentru adăugarea unei comenzi în lista și în HashMap, urmând apoi procesul de serialiazare.

În DataLayer Package avem clasele FileWriterClass și RestaurantSerializator.

În FileWriterClass avem metoda de print. Această metodă va fi folosită pentru a afișa într-un fișier cu format .txt prețul comenzii.

RestaurantSerializator implementeaza interfața Serializable. Am creat metodele de serialization si deserialization pentru itemii din meniu.

În PresentationLayer Package, pentru Administrator, avem clasa Add, totodata ajutând la realizarea operațiunii de adăugare în meniu prin intermediul Administratorului. Informațiile despre produs sunt preluat din JTextField-uri, apoi vom crea produsul în funcție de tipul acestuia și urmează a fi adăugat în în listă. BaseProduct-ul va avea name, quantity, price, pe când CompositeProduct-ul va conține 2 BaseProduct-uri, pentru care va fi introdus numele, apoi un nume general al CompositeProduct-ului, urmat de quantity și price.

Clasa Delete ne ajută să realizăm ștergerea după nume a unui BaseProduct sau a unui CompositeProduct. Pentru realizarea acestei operațiuni am decis să facem deserializare pentru lista de meniuri, apoi să căutăm elemental care se dorește a fi șters și să îl adăugăm într-o listă pentru a Evita eventualele erori/excepții de a realiza 2 operațiuni simultane pe aceeași listă. La final, din lista de itemi din meniu vom șterge elementele care se găsesc în lista denumită toBeRemoved.

Clasa Modify are ca scop modificarea numelui, cantității sau prețului unui produs din lista de meniuri. Aceasta realizează preluarea datelor din JTextField-uri. Modificarea se va face după nume. Dacă vrem să modificăm un BaseProduct, adăugăm numele itemului dorit în căsuța name, iar dacă vrem să modificăm un CompositeProduct, în căsuța a cărei valoare este initial null, adăugăm numele, apoi și eventual numele BaseProduct-ului care se dorește a fi modificat. Item-ul nou creat va fi adăugat pe poziția corectă în listă, cu ajutorul unei variabile auxiliare.

Tot Administratorul va putea să afișeze un table cu toate produsele din meniu, într-un tabel. Clasa Table conține metodele de preluare a informațiilor cu care se va popular ulterior tabelul.

Pentru Waiter, avem două opțiuni: să cream o comandă sau să facem nota de plată.

Pentru crearea comenzii avem clasa CreateOrder. Aici decidem numele produsului pe care vrem să îl adăugăm, cât și cantitatea, id-ul comenzii și masa.

În funcție de produsul ales, daca e de tip Base sau Composite, am decis să facem următorul lucru cu cantitatea. Dacă clientul dorește o cantitate X dintr-un anumit produs verificăm în aceste cazuri:

Daca X e mai mic decât cantitatea furnizată in meniu pentru produsul respectiv, in comandă vom adăuga X în dreptul cantității.

Dacă X e mai mare, înseamnă că este cerut mai mult decât putem să oferim pentru acel produs, așa că maximul este deja cantitatea produsului, ce va fi afișată în dreptul cantității în comandă.

Pentru un CompositeProduct, se va adăuga cantitatea minima disponibilă, obținută în urma comparărilor cu fiecare BaseProduct din listă.

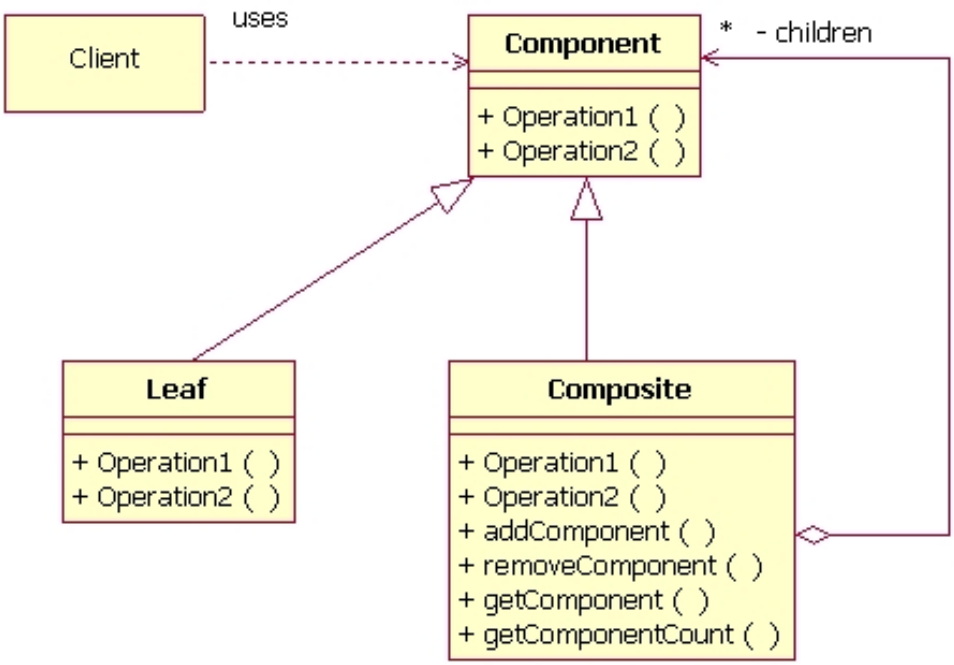
Dacă am selectat opțiune de calculare preț(ComputePrice), vom face acest după id-ul comenzii.

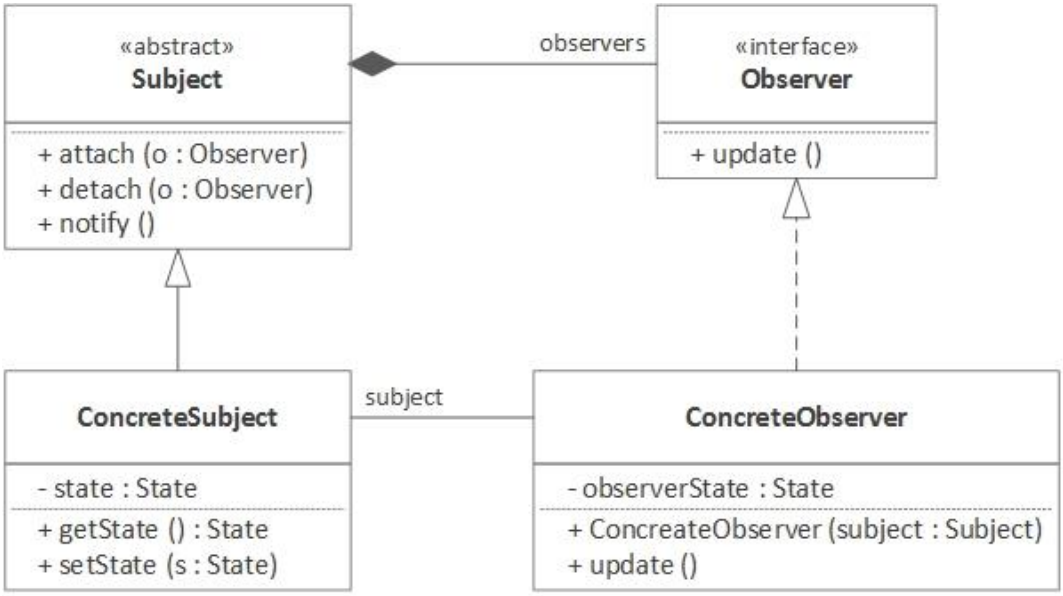
Într-o variabilă de tip HashMap vom stoca lista de comenzi în care vom căuta comanda cerută în funcție de id, apoi vom afișa detaliile comenzii într-un JTextArea și prețul în fișierul cu format .txt.

Restul claselor prezintă interfețe utilizator pentru preluarea comenzilor.

Pentru Chef am ne-am folosit de Observer Design Patter pentru a putea să anunțăm atunci când este preluată o comandă nouă.

Composite Design Patter a fost utilizat atunci când am creat clasele BaseProduct și CompositeProduct, conform teoriei.

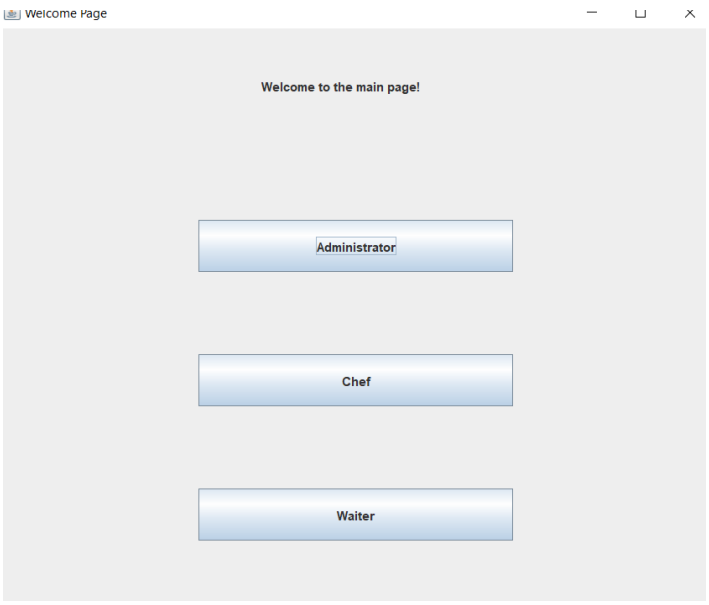




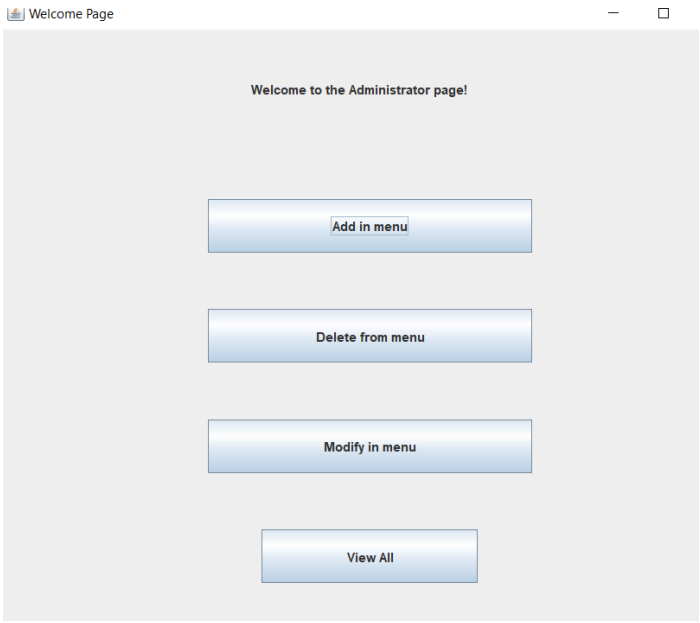
~ Rezultate ~

În urma rulării programului, am obținut următoarele rezultate:

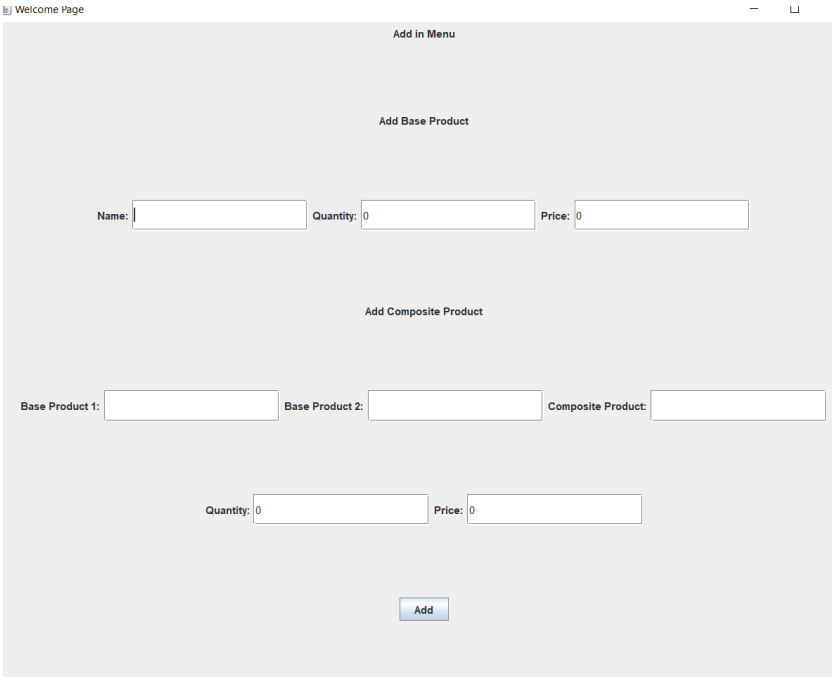
Interfața principal de unde putem alege tipul de utilizator.



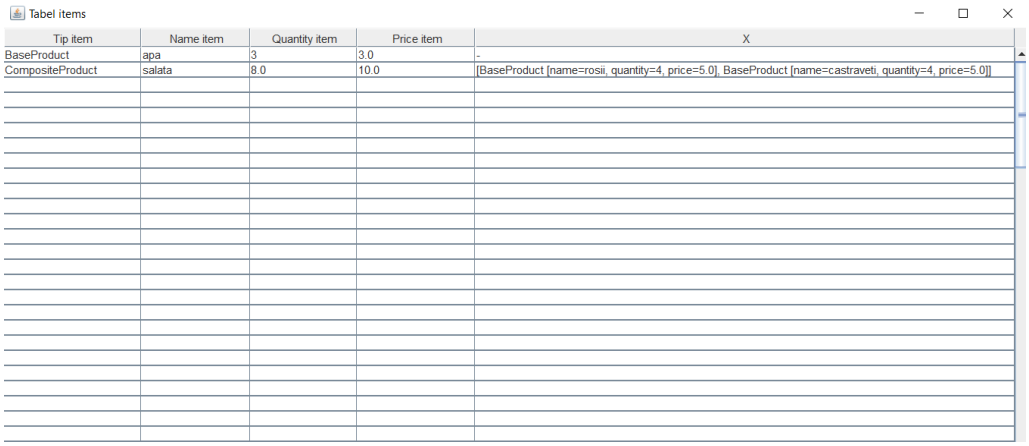
Mai departe, alegem Administrator:



Alegem apoi optiunea de adaugare în meniu:

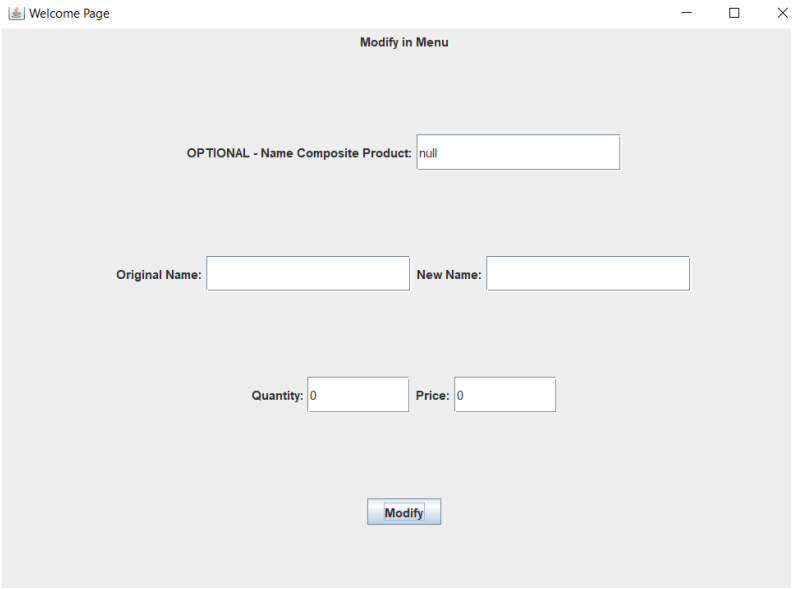


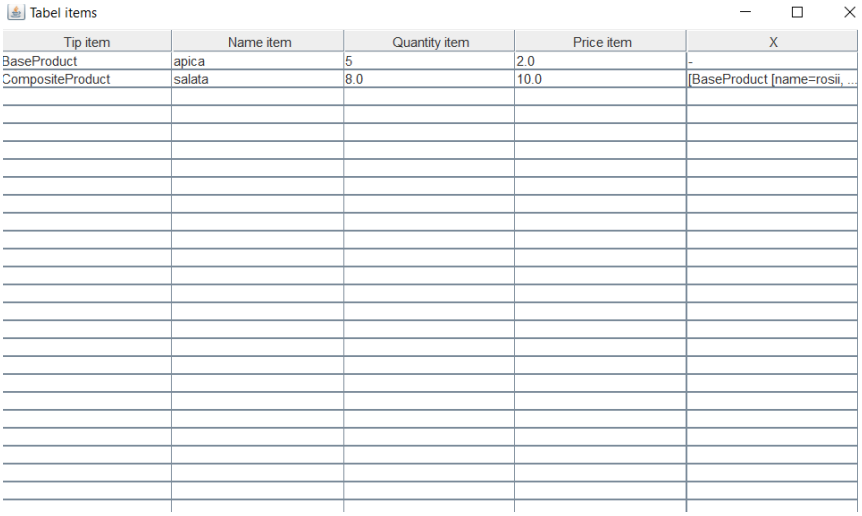
În meniu am adăugat un BaseProduct si un CompositeProduct pe care le vizualizam în tabel:



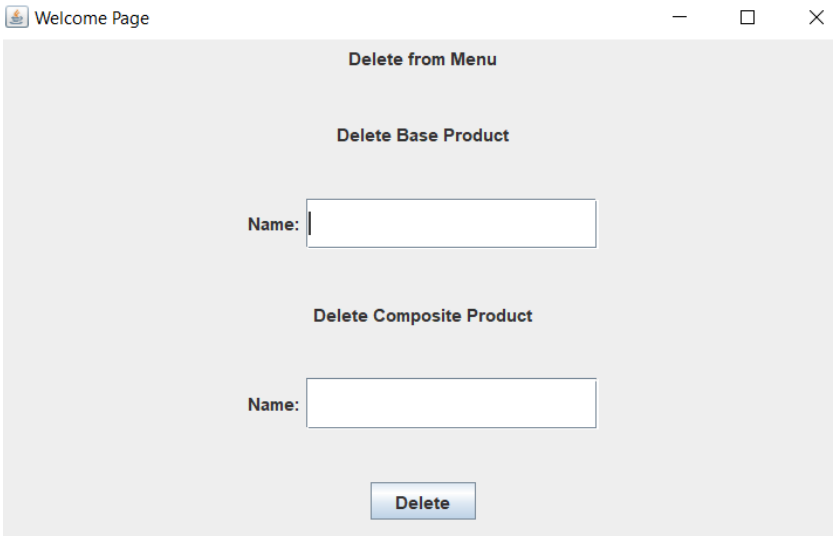
Alegem altă opțiune, de modificare:

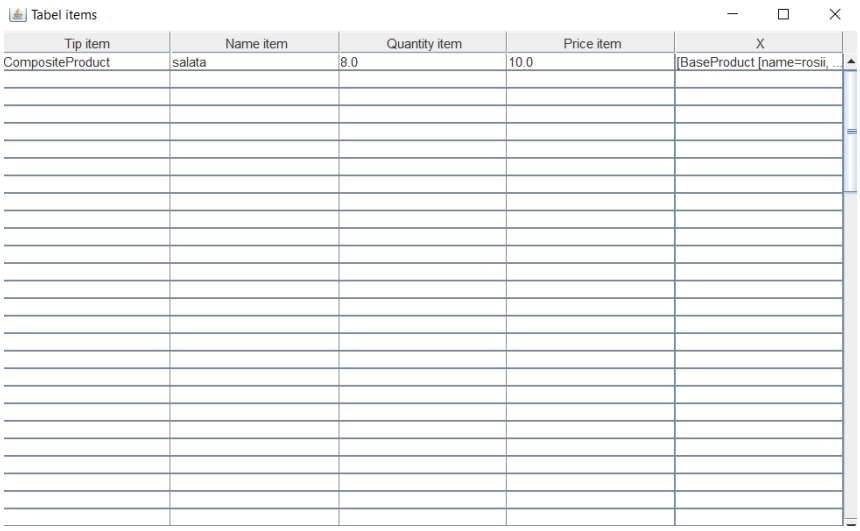
Schimbam apa in apica:



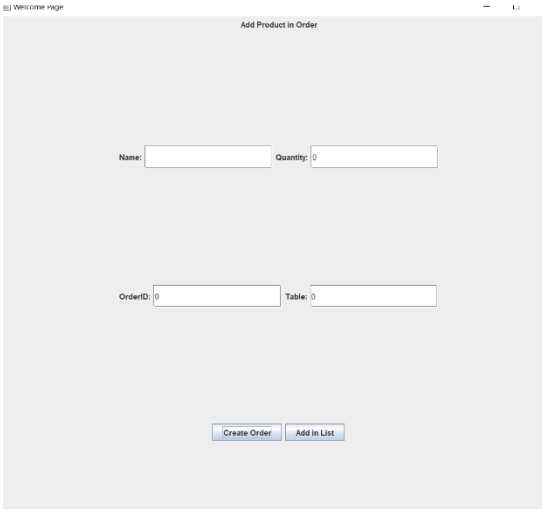


Și decidem să stergem ceva:

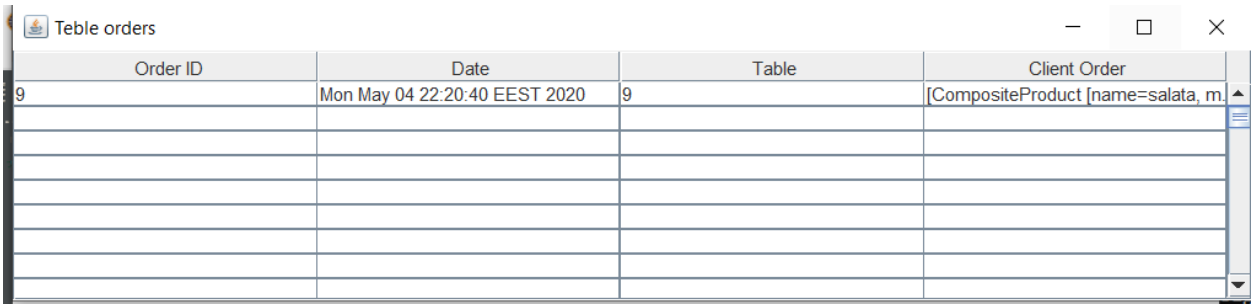




Mai departe, vrem să cream o comanda:

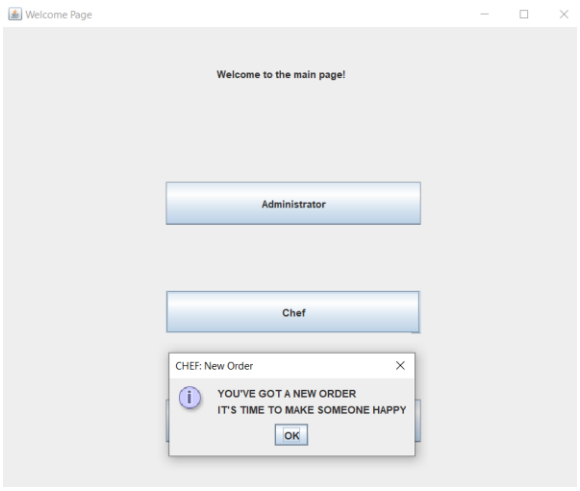


Și să o vizualizăm:

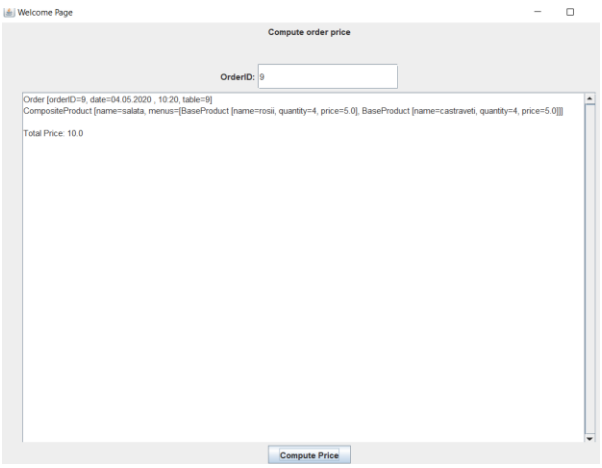


Ce ne dorim mai departe este să anuntăm chef-ul:

Apăsând pe butonul Chef din pagina principală, ajungem aici:



Să presupunem apoi că vrem să calculăm nota de plată pentru comandă:



~ Concluzii ~

Din rezolvarea acestei teme am învățat să lucrez cu Observer Design Pattern și Composite Design Pattern, acestea fiind noțiunile introduse.

De asemenea, am avut ocazia de a folosi seralizarea și deserializarea pentru lucrul cu produsele,

Am avut ocazia să folosesc și HashMap și, totodată, să înțeleg mai bine modul de funcționare.

Pe lângă toate acestea, la fel ca până acum, unul din obiectivele acestei teme era și de a fundamenta cunoștințele anterioare de programare orientată pe obiecte.

~ Bibliografie ~

http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_4/Assignment_4_Indications.pdf

https://www.baeldung.com/java-serialization

https://www.geeksforgeeks.org/serialization-in-java/

https://www.geeksforgeeks.org/composite-design-pattern/

https://www.baeldung.com/java-composite-pattern

https://dzone.com/articles/composite-design-pattern-java-0

https://www.tutorialspoint.com/design_pattern/observer_pattern.htm

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

https://howtodoinjava.com/design-patterns/behavioral/observer-design-pattern/

https://www.vogella.com/tutorials/DesignPatternObserver/article.html

https://www.youtube.com/watch?v=98DiwRp-KZk

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

https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html