**PROIECT PPOO**

Implementarea unui magazin online în varianta consolă

Student

Calotă Denisa-Andreea

Grupa 1105, E-Business

Academia de Studii Economice, București

2020

Cuprins

[1. Descrierea problemei propuse 3](#_Toc55209678)

[2. Implementarea soluției informatice 4](#_Toc55209679)

[2.1. Meniul aplicației 4](#_Toc55209680)

[2.2. Clase 7](#_Toc55209681)

[2.2.1. User 7](#_Toc55209682)

[2.2.2. Order 7](#_Toc55209683)

[2.2.3. Product 7](#_Toc55209684)

[2.2.4. Statistics 7](#_Toc55209685)

[2.2.5. Validations 8](#_Toc55209686)

[2.3. Stocarea persistenta și stocarea volatilă 8](#_Toc55209687)

[2.4. Gestionarea excepțiilor și validarea datelor introduse 9](#_Toc55209688)

[Concluzii 11](#_Toc55209689)

**Introducere**

Soluția informatică prezentată în cadrul acestei lucrări își propune să vină în ajutorul unui magazin online ce comercializează deserturi. În cadrul acesteia, utilizatorul are posibilitatea de a-și crea un cont personal, își poate vizualiza, respectiv actualiza datele personale, poate vizualiza meniul magazinului și poate plasa comenzi. Totodată, în cadrul aplicației, utilizatorul poate vizualiza și o analiza asupra comenzilor plasate, fiindu-i afișate statistici despre produsele preferate, respectiv despre istoricul achizițiilor făcute. Toate aceste funcționalități sunt îmbrăcate într-un meniu afișat la consolă, intuitiv și ușor de utilizat.

În implementarea acestei soluții, s-a utilizat limbajul Java, având ca principale motive siguranța pe care o oferă, portabilitatea, mentenanța ușoară, dar mai ales prezența instrumentelor de nivel înalt (*high-level programming tools*). Având la bază conceptul POO (programare orientată obiect), Java oferă o utilitate generală prin organizarea codului sursă în clase, lucru ce aduce o contribuție semnificativă la gestiunea proiectelor de dimensiuni mari.

Una dintre cele mai relevante caracteristici și principalul motiv pentru care acest limbaj de programare funcționează adecvat pe o varietate de platforme este independența sa față de mașina pe care este compilat. Această particularitate a limbajului este oferită de JVM – Java Virtual Machine, componentă care funcționează ca un liant între limbaj și mașină.

# Descrierea problemei propuse

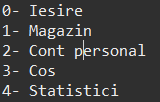
Aplicația este destinată unui magazin online ce comercializează deserturi, ecleruri și cheesecake, fiind proiectată din perspectiva clientului, acesta având acces la contul personal, la meniu, dar si la statistici asupra preferințelor personale.

# Implementarea soluției informatice

## Meniul aplicației

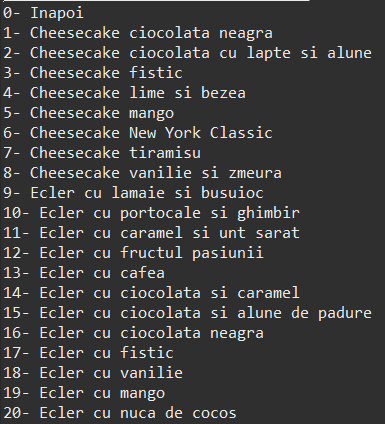
Meniul aplicației este organizat sub forma unei structuri arborescente, prezentate în figura 2.1.1, ce conține opțiunile principale – *Magazin, Cont personal, Coș, Statistici* și *Părăsirea magazinului.*

Figură 2.1‑1 Structura arborescentă a meniului



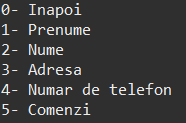
Figură 2.1‑2 Meniul principal la nivelul aplicației

În cadrul *Magazin,* utilizatorul poate vizualiza produsele disponibile în magazin, poate accesa aceste produse și poate vizualiza informații, precum descrierea și prețul, în final având posibilitatea de a adăuga produsul în coș.



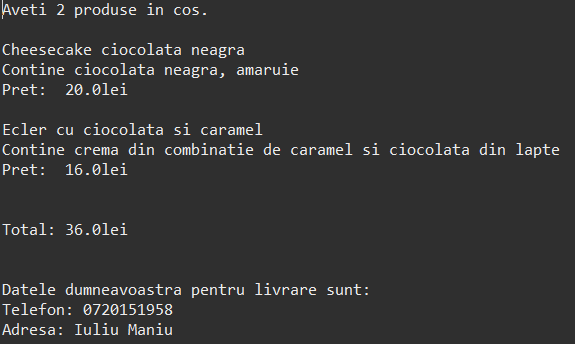
Figură 2.1‑3Meniu

În cadrul *Cont personal,* utilizatorul are acces la datele sale personale: *nume, prenume, număr de telefon* și *adresă,* pe care le poate actualiza. Totodată, sub acest meniu, se poate vizualiza și un istoric al comenzilor plasate.



Figură 2.1‑4Cont personal

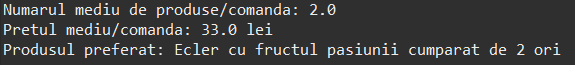
În *Coș,* sunt afișate produsele alese de utilizator pentru a fi cumpărate ulterior.



Figură 2.1‑5Cos

Accesând meniul *Statistici,* utilizatorul poate vizualiza o analiza asupra preferințelor de cumpărare, fiind disponibile:

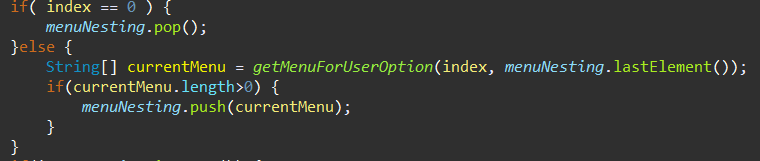
* Produsul preferat
* De câte ori a fost achiziționat produsul
* Numărul mediu de produse achiziționate per comandă
* Totalul mediul al unei comenzi.



Figură 2.1‑6Statistici

Prin accesarea *Ieșire,* se părăsește magazinul, moment în care modificările realizate sunt stocate persistent.

Pentru a asigura navigarea fluidă între elementele meniului, se utilizează o *Stivă,* astfel, la fiecare accesare a unui nou meniu, acesta va fi adăugat în stivă, iar la oricare ieșire din meniu, se va apela scoaterea din stivă.



Figură 2.1‑7Utilizarea stivei in cadrul meniului

## Clase

Entitățile identificate la nivelul problemei sunt: *utilizatorul, comanda* și *produsul,* care au fost transpuse în clasele: *User, Order,* respectiv *Product.* În cadrul acestora, se regăsesc atributele specifice fiecărei clase, împreună cu suprascriere metodei *toString().*

### User

În cadrul clasei User, sunt gestionate datele personale ale utilizatorului: *nume (String), prenume (String), număr de telefon (String), adresă (String)*, dar și o colecție ce conține *istoricul comenzilor (List<Order>).* Aceste atribute pot fi accesate atât prin setteri, cât și prin getteri.

### Order

Clasa Order conține atributele *date (String), products (List<Product>)* și *totalPrice (float),* care pot fi setate cu ajutorul *constructorului* și accesate cu ajutorul *getterilor*.

### Product

În cadrul clasei *Product*, se regăsesc atributele *name (String)*, *description (String)* și *price (float),* acestea fiind accesate prin intermediul *getterilor*. Instanțele acestei clase compun o colecție de obiecte inclusă în cadrul comenzilor (*Order).*

De asemenea, în cadrul proiectului se regăsesc și două clase ajutătoare, *Statistics* și *Validations,* create pentru a menține codul structurat și modularizat.

### Statistics

Clasa *Statistics* conține cinci atribute private, dintre care doar patru pot fi accesate, cu ajutorul *getterilor,* din exteriorul clasei. Aceste atribute sunt*: favouriteProduct (String), numberFavProduct (int), averageOrderPrice (float), averageNumberOfProducts (float) și topProducts (HashMap<String, Integer>). HashMap-ul* este singurul atribut utilizat doar în interiorul clasei, cu scopul de a contoriza numărul de produse achiziționate din fiecare tip.

În cadrul acestei clase, exista metoda *doStatistics,* ce primește ca parametru un *User*, și realiză o analiză asupra valorilor ce se află în colecția de comenzi stocată la nivelul unui utilizator.

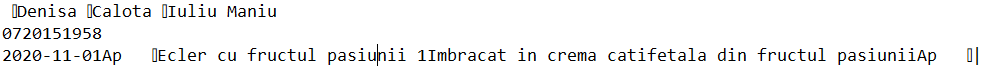
### Validations

Clasa *Validations* nu conține atribute și este populată cu multiple metode statice menite să asigure validarea datelor de intrare, cât și gestiunea excepțiilor. Această clasă va fi detaliată în subcapitolul *2.4. Gestiunea excepțiilor și validarea datelor introduse.*

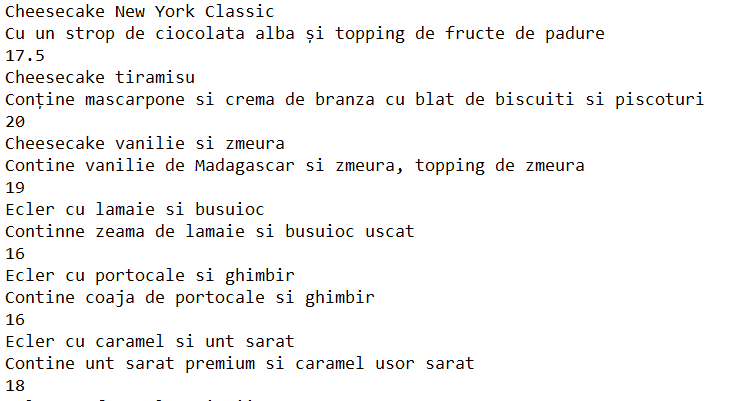
## Stocarea persistenta și stocarea volatilă

La nivelul aplicației, stocarea persistentă este asigurată cu ajutorul fișierelor, atât binare, cât și text. Sunt utilizate trei fișiere:

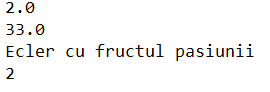
* Un fișier *binar* ce conține informații despre utilizator;



* Un fișier *text* ce conține produsele magazinului, împreună cu descrierea și prețul;



* Un fișier *text,* ce conține statisticile despre preferințele utilizatorului.



Interacțiunea cu fișiere are loc la lansarea aplicației, prin inițializarea utilizatorului și încărcarea produselor, dar și la închiderea aplicației, prin suprascrierea datelor despre utilizator și statistici.

Stocarea *volatilă* se face cu ajutorul colecțiilor de date imbricate (*lista de comenzi, ce conține o listă de produse),* dar și cu ajutorul vectorilor cu date primitive (*vectorii ce conțin meniurile)*.

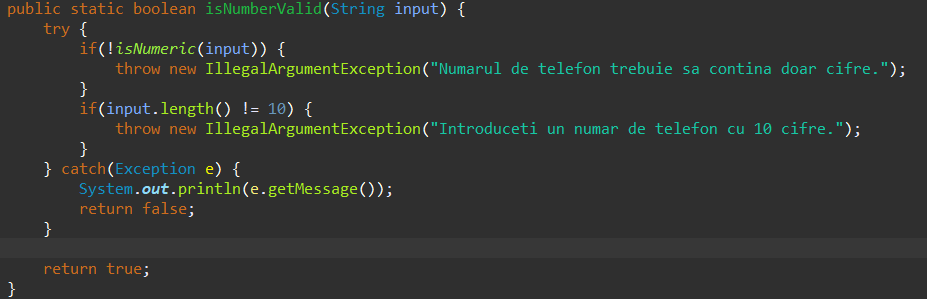




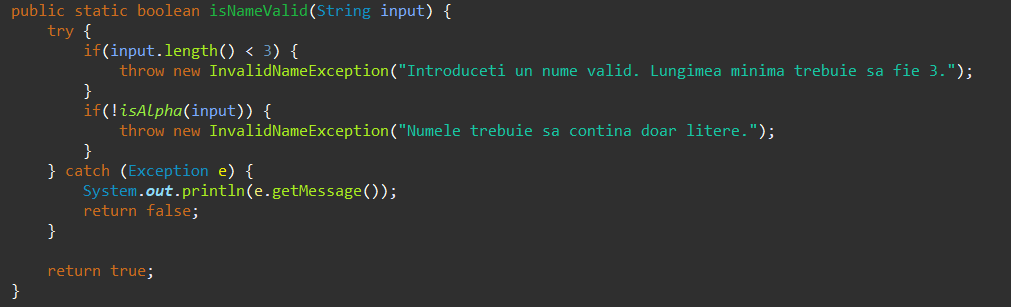


## Gestionarea excepțiilor și validarea datelor introduse

Pentru a asigura funcționarea fără erori a aplicației, s-a ales utilizarea atât a excepțiilor default, cât și a excepțiilor personalizate.



Figură 2.4‑1 Exceptie default



Figură 2.4‑2 Exceptie personalizata

Aceste excepții sunt utilizate în cadrul clasei *Validations*, în cadrul căreia există opt metode statice, utilizate pentru validarea datelor introduse la tastatură, respectiv validarea datelor citite din fișier. Aceste metode sunt:

* *isNameValid*, ce verifică lungimea inputurilor, dar și componența acestora, permițând doar caractere *a-Z.*
* *isNumberValid,* ce verifică validitatea numărului de telefon: lungime fixă de 10 caractere, de la 0 la 9.
* *isAddressValid,* asigură o lungime a adresei de minim 7 caractere.
* *isDateValid*, ce folosește mecanismul try-catch, în cadrul căruia se încearcă transformare unui *String* în *Date.*
* *isPriceValid,* ce asigură existența prețurilor strict pozitive.
* *isAlpha, isNumeric, isAlphaNumeric,* funcții ajutătoare, folosite în metodele prezentare anterior menționate, care verifică dacă un *String* este compus din litere, din cifere, respectiv litere și cifre.

# Concluzii

Aplicația prezentată este o soluție versatilă ce poate fi transpusă în mediul online, pentru un magazin de deserturi. Implementarea acesteia cu limbajul Java, permite utilizarea ei pe multiple sisteme de operare, aducându-i un grad de utilizabilitate crescut.

În procesul de implementare, a fost utilizată o modalitate de versionare, GIT, ce a permis revenirea la versiuni anterioare, în cazul în care implementarea nu urma traiectoria dorită, dar și stocarea online a soluției.

Ca și direcții viitoare de dezvoltare a aplicației, aceasta poate permite atât introducerea unui administrator de sistem, cât și a multiplilor utilizatori ai magazinului.

# Bibliografie

1. Brian D. Hahn, Katherine M. Malan, Essential Java for Scientists and Engineers, Butterworth-Heinemann, 2002
2. Java Create and Write To Files - <https://www.w3schools.com/java/java_files_create.asp>