ACADEMIA DE STUDII ECONOMICE BUCUREȘTI

FACULTATEA DE CIBERNETICĂ, STATISTICĂ ȘI INFORMATICĂ ECONOMICĂ

**IMPLEMENTAREA OPERAȚIILOR CRUD PENTRU GESTIUNEA UNUI MAGAZIN ON-LINE**

Student: Lungu E. Mihai-Adrian

Grupa: 1110

2021

***CUPRINS:***

**Capitolul 1 – Cunoașterea domeniului.............................................................................3**

* 1. Analiza temei și a cerințelor de implementare...........................................................................3
  2. Detalii de implementare și colecții utilizate...............................................................................4

**Capitolul 2 – Implementarea aplicației............................................................................7**

2.1. Prezentarea rezultatului.……………………………………………………………………....7

**Bibliografie.......................................................................................................................10**

**Capitolul 1 – Cunoașterea domeniului**

* 1. **Analiza temei și a cerințelor de implementare**

Această temă presupune dezvoltarea unei aplicații Java destinată gestiunii unui magazin on-line, în care să se poată efectua toate operațiile CRUD. Pentru acest lucru, implementarea nu presupune lucrul cu baze da date, ci s-a optat pentru varianta în care s-a găsit o altă soluție de persistență adiacentă, și anume lucrul cu fișiere de tip text.

Prin operații de tip CRUD, ne referim la faptul că aplicația dezvoltată trebuie să fie capabilă să creeze noi produse, să le poată citi și să le afișeze, să le poată edita individual pe fiecare dintre ele, și să le poată șterge la cerere în funcție de un anumit critieriu.

Se presupune asigurarea unui mecanism de salvare a datelor la închiderea aplicației și restaurarea acestora la relansarea aplicației prin actualizarea fișierului cu datele de intrare, lucrul cu minim două masive de tipuri de date fundamentale și minim două colecții diferite de obiecte din clasele definite.

Totodată, toate modificările asupra datelor și obiectelor de lucru se fac în memorie, în cadrul colecțiilor definite, iar salvarea în fișier se realizează la închiderea aplicației. Trebuie efectuată validarea datelor de lucru, atât cele citite de la consolă, cât și cele preluate din fișiere de intrare, utilizând mecanismul de gestiune a excepțiilor, atât prin folosirea claselor standard de excepții, cât și prin definirea de către programator a unor excepții specifice.

Trebuie asigurat un flux funcțional logic al aplicației, din momentul deschiderii și încărcării cu date, până la momentul închiderii, prin utilizarea unui meniu cu opțiuni clare oricărui utilizator.

În final, aplicația nu trebuie să conțină erori de compilare sau de execuție.

* 1. **Detalii de implementare și colecțiile utilizate**

Pentru început, Java este un limbaj de programare orientat-obiect, care împrumută o mare parte din sintaxă din alte limbaje de programare, având un model al obiectelor mult mai simplu, suportând totodată folosirea de concepte de programare avansate, precum firele de execuție. [Burak Kanber, Lee Jacobson – *Genetic Algorithms in Java Basics,* pag. 21]

Pentru dezvoltarea aplicației, s-a folosit framework-ul Swing care este un instrument GUI (graphical user interface) pentru Java. Acesta este un API care ne ajută să furnizăm o interfață grafică cu utilizatorul destinată aplicațiilor Java.

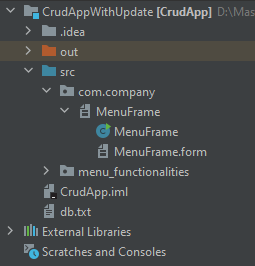
Scopul pentru care a fost dezvoltat acest framework, este pentru a oferi un set mai sofisticat de componente GUI față de alte framework-uri concurente, precum ar fi panoul de tab-uri, scroll panes, tabele și liste.

La nivelul acesti aplicații, ne-am folosit de container-ul JFrame care este un container care moștenește clasa java.awt.Frame. Acesta funcționează ca și fereastră principală în care componentele precum etichetele, butoanele sau câmpurile de text sunt adăugate pentru a crea o intrefață grafică. [https://www.javatpoint.com/java-jframe]

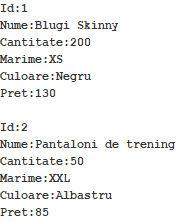
Pentru a crea această interfață ne-am folosit de un “design form” pe care l-am numit “MenuFrame.form”. L-am folosit pentru a putea aranja toate butoanele, listele derulante și câmpurile destinate inserării de text în pozițiile pe care ni le-am dorit, specificându-le tuturor un anumit id pentru identificare, un nume care să fie afișat pentru ca utilizatorul să știe ce înseamnă fiecare dintre ele și o anumită valoare sau culoare.

După ce ne-am folosit de acest “design form” pentru a desena modul în care trebuie să arate interfața, alegem să creăm acțiuni la nivel de butoane și câmpuri destinate introducerii de date, pe baza unei interfețe în cadrul căreia sunt apelate metodele CRUD. Astfel, apare clasa “MenuFrame” care conține tot codul necesar pentru a pune interfața în funcțiune.

Mai jos am introdus o captură de ecran care să explice mai bine structura fișierelor.

****

Pentru început, am create un fișier de tip “txt” pe care l-am numit “bd.txt”, pe care l-am folosit pe post de bază de date și pe care l-am prepopulat cu câteva date inițiale care să ne ajute să exemplificăm modul în care funcționează metodele implementate.

****

Ulterior, am creat metoda de “**read**” în care am preluat datele din fișierul de tip text și le-am afișat în interfața creată anterior. Pentru acest lucru, ne-am folosit de clasa “File” și “Scanner” prin care i-am indicat programului ce fișier să citească. Ulterior, am început să citim linie cu linie din fișierul text prin metoda “hasNextLine()”, printând pe ecran într-un “text area” fiecare linie. În final, ne-am asigurat că am închis scanner-ul pentru a evita alte posibile conflicte. Clasa “Scanner” poate să fie folosită atât pentru a citi date dintr-un fișier text, cât și de la tastatură.[https://www.java67.com/2012/11/how-to-read-file-in-java-using-scanner-example.html]

Totodată, aici ne-am folosit și de prima colecție, un “ArrayList” de tip “Integer”, care este un obiect care ne ajută să grupăm mai multe elemente similar într-o singură unitate. Colecțiile sunt folosite pentru a stoca, a prelua, a manipula și a comunica date agregate. [https://docs.oracle.com/javase/tutorial/collections/intro/index.html]

În interiorul acestuia, am salvat toate id-urile existenta în fișierul text pe care ulterior le-am adăugat într-o listă derulantă, care este de foarte mare ajutor atunci când vrem să actualizăm un produs sau să ștergem unul, deoarece nu mai trebuie să facem verificări în plus pentru a vedea dacă id-ul introdus de către utilizator există sau nu pentru a putea fi modificat conținutul sau șters. Astfel, ne asigurăm de faptul că utilizatorul va avea la dispoziție doar id-urile existente la alegere. Pentru a reuși să preiau toate id-urile, m-am folosit de un “StringTokenizer” care are delimitatorul “:” și care verifică dacă rândul pe care eram situat începe cu “Id:”, iar în caz afirmativ îmi introducea în listă valoarea id-ului.

De altfel, în cadrul metodei de afișare a datelor, la început ne asigurăm că golim toate câmpurile sunt goale, pentru a nu persista conținutul introdus după ce una dintre metode (create, delete sau update) a fost deja folosită și ca data viitoare când utilizatorul vrea să revină pentru a face un nou proces, să nu trebuiască să le șteargă el personal.

Pentru metoda de “**create**” s-au folosit aceleași funcții ca și la citirea din fișier, doar că înainte de a începe orice fel de proces ne-am asigurat că toate datele introduse sunt valide și că respectă condițiile impuse de programator. În cazul în care există probleme depistate la nivelul introducerii de date, nu se începe introducerea de noi date în fișier, ci programul se oprește, iar utilizatorul este avertizat ce a greșit când a introdus datele. Pentru opțiunea de “mărime” și “culoare” au fost create două liste derulante din care utilizatorul poate să aleagă orice variantă își dorește.

În cadrul acestei metode, apare și funcția “FileWriter”, căreia i-am menționat în ce fișier să introducă datele primite din interfață. Astfel, noul șir creat este adăugat la sfârșitul conținutului deja existent în cadrul fișierului text. În final, utilizatorul este informat că a reușit să introducă noile date cu succes.

În cadrul metodei “**delete**”, m-am folosit de lista derulantă creată în momentul în care am citit datele din fișierul de tip text pentru a selecta un produs pe care doresc să îl șterg din baza de date. Astfel, am creat o nouă colecție de tip “ArrayList” care de această dată nu are un tip de date special, în care memorez datele deja existente în fișierul text, mai puțin datele produsului pe care vreau să îl șterg. La sfârșit, rescriu datele din fișier cu cele din colecția creată anterior, iar utilizatorul este înștiințat că a reușit să șteargă produsul cu succes.

În cazul în care dorim să parcurgem colecția, acest lucru îl putem face doar prin prisma unui iterator, deoarece fiecare colecție are un anumit tip de ordonare. Astfel, iteratorul ne permite să navigăm prin colecție și să ajungem la orice linie sau înregistrare dorim. [Bruce Eckel – Thinking in Java, Fourth Edition, pag. 580]

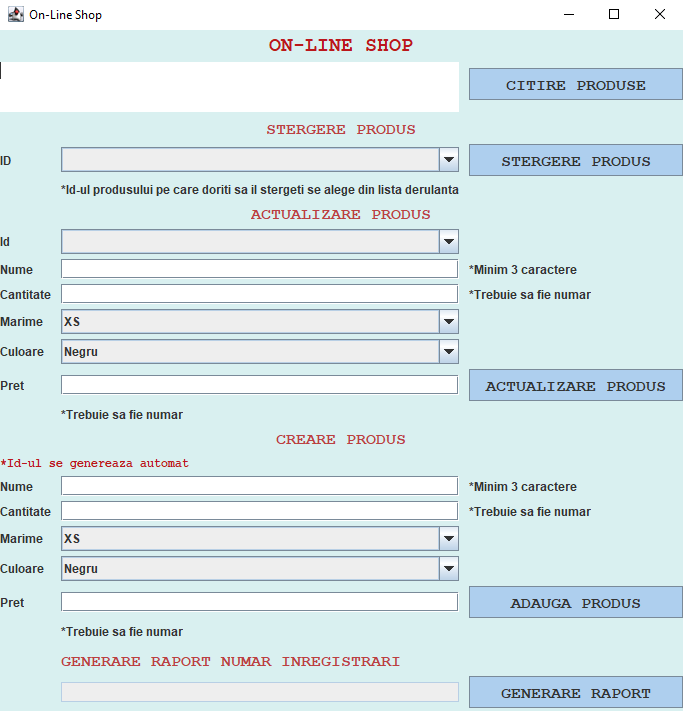
În cadrul metodei de “**update**”, ca și în cazul metodei de adăugare de nou produs, primul pas este să verificăm dacă datele introduse de către utilizator sunt valide. În caz afirmativ, începe procesul de actualizare de date. Utilizatorul, are atât câmpuri în care trebuie să introducă date, precum numele, cantitatea și prețul, cât și date pe care trebuie să le aleagă din liste derulante, precum id-ul produsului, mărimea și culoarea. Pentru început, începem să recitim de la început fișierul text în care am memorat datele despre respectivele produse create anterior, până ajungem la produsul cu id-ul specificat de către utilizator, folosindu-ne de “StringTokenizer” cu delimitatorul “:”. Când am ajuns la această linie, începem să rescriem valorile atributelor prin adaugându-le într-o colecție de tip “ArrayList”. Totodată, lista conține și toate datele existente înainte de produsul pe care vreau să îl actualizez, ci și cele care erau după el. În final, utilizând funcția “FileWriter”, rescriu fișierul text cu noile date și înștiințez utilizatorul că modificările au fost realizate cu succes.

**Capitolul 2 – Implementarea aplicației**

* 1. **Prezentarea rezultatului**

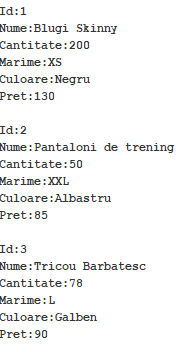
După ce toate metodele CRUD au fost implementate și au fost adăugate validările necesare, urmează partea de testare a aplicației, verifcând ce am reușit să implementez și ce procent din cerințele proiectului am reușit să acopăr.

Astfel, am dat drumul aplicației, putând pentru prima dată să vedem cum arată interfața, putându-se observa foarte clar fiecare secțiune dedicată metodelor CRUD.

****Pentru început, apăsăm pe butonul de “Afișare Produse” care va intra în baza noastră de date de tip “txt” și va afișa în locul indicat tot conținutul fișierului.

După ce am apăsat butonul de “read”, putem observa că în interiorul listelor derulante putem vedea id-urile existente în baza de date, asupra căror putem efectua modificări.

Ulterior, mergem la secțiunea de adăugare și încercăm să introducem în baza noastră de date un nou produs. Completăm toate câmpurile necesare, apăsăm butonul de “Adăugare Produs” și suntem anunțați de către sistem că am introdus noul produs cu succes.

****1

Totodată, dacă introducem date care nu respectă condițiile de validare, sistemul o să ne avertizeze de fiecare dată ce am scris greșit.

După ce acționăm metoda de “read”, o să fie reafișate toate datele din baza de date, iar toate listele derulante vor fi actualizate cu noul produs.

În cazul metodei de “delete” și de “update”, trebuie să ne asigurăm în prealabil că înainte de acționarea metodei, să citim baza de date pentru ca în listele derulante să fie introduse toate id-urile.

Astfel, atunci când acționăm metoda de „delete”, tot ce trebuie să facem este să alegem din lista derulantă produsul pe care dorim să îl ștergem și să apăsăm pe butonul de delete. După ce citim din nou baza de date, o să observăm că produsul a fost șters.

La fel și în cazul metodei de update, în care trebuie să alegem id-ul din lista derulantă și să introducem noile date aferente produsului selectat.

**Bibliografie**

[1] Bruce Eckel – Thinking in Java, Fourth Edition

[2] Burak Kanber, Lee Jacobson – *Genetic Algorithms in Java Basics*, Editura apress, 2015)

[3] <https://docs.oracle.com/javase/tutorial/collections/intro/index.html>

[4] <https://www.javatpoint.com/java-jframe>

[5] <https://www.java67.com/2012/11/how-to-read-file-in-java-using-scanner-example.html>