DOCUMENTAȚIE

TEMA 3

Pașcan Daniela Maria

Grupa 9

Cuprins

1. Obiectivul temei..............................................................................................................3

2. Analiza problemei, modelare, cazuri de utilizare............................................................3

3. Proiectare.........................................................................................................................6

4. Implementare...................................................................................................................9

5. Rezultate………............................................................................................................15

6. Concluzii.......................................................................................................................15

7. Bibliografie...................................................................................................................15

**1.Obiectivul temei**

Dorim să implementăm o aplicație care are ca scop gestionarea clienților, a produselor și plasarea de comenzi pentru un deposit.

1. *Analiza problemei, modelare, scenariu, cazuri de utilizare-* aici se vor reprezenta cerițele funcționale și cele non-funcționale, precum și use-case-urile

2*. Proiectare-* în această secțiune se va discuta despre diagrama UML a proiectului și despre organizarea acestuia în pachete și clase

1. *Implementare-* aici se va descrie în detalui fiecare clasă și metodele importante ale acesteia
2. *Rezultate-* la final vom discuta despre datele obținute în urma simulării

**2.Analiza problemei, modelare, cazuri de utilizare**

**2.1. Analiza problemei**

*Cerințe funcționale*

* Aplicația de simulare trebuie să permit utilizatorului să introducă datele pentru un nou client
* Aplicația trebuie să permită utilizatorul să introduce datele pentru un nou produs
* Aplicația trebuie să permită utilizatorului să modifice clienți și produse
* Aplicația trebuie să permită utilizatorului să șteargă anumiți clienți și produse după ID-ul introdus
* Aplicația trebuie să permită utilizatorului să vizualizeze tabelele cu clienți și cu produse
* Aplicația trebuie să permită utilizatorului să aleagă un client și un produs și să plaseze o comană, adăugând cantitatea produsului de cumpărat

*Cerințe non-funcționale*

* Aplicația de gestionare a unui depozit trebuie să fie intuitivă și simplu de utilizat de către utilizator
* Aplicația trebuie să aibă o bună performanță ( timpul de răspuns să fie minim, utilizarea sa să meargă linear)
* Aplicația de gestionare trebuie sa aibă scalabilitate ( să poată lucra cu un volum mai mare de informație și să permită mărirea sau extinderea sa)
* Aplicația de gestionare trebuie să țină cont de integritatea datelor, să afișeze rezultate corecte, complete și consistente și să păstreze datele neschimbate indiferent de cat timp sunt stocate sau cat de des sunt accesate
* Aplicația trebuie să fie ușor de reparat și de întreținut
* Trebuie să se țină cont de integritatea datelor în aplicație.

**2.2.Cazuri de utilizare**

A picture containing diagram, line, pattern

Description automatically generated

**Caz de utilizare**: Adăugare client

Actor primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul selectează opțiunea de adăugare a unui nou client
2. Aplicația va arăta căsuțele cu informațiile despre client care trebuie completate
3. Utilizatorul introduce în aplicație id-ul, numele, email-ul și adresa pentru noul client
4. Utilizatorul apasă butonul ADD pentru a introduce clientul în baza de date
5. Aplicația stochează clientul în baza de date și afișează un mesaj de success

Scenariu alternativ:

* Utilizatorul inserează valori invalide pentru parametrii clientului
* Se va afișa un mesaj de eroare iar utilizatorul va trebui să introducă datele corect
* Scenariul se întoarce în pasul 1

**Caz de utilizare**: Editare client

Actor primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul selectează opțiunea de editare a unui client

2. Utilizatorul inserează id-ul clientului pe care vrea să îl modifice

3. Aplicația va arăta căsuțele cu informațiile despre client care trebuie completate

4. Utilizatorul introduce în aplicație id-ul, numele, email-ul și adresa pentru client

5. Utilizatorul apasă butonul EDIT pentru a edita clientul

1. Aplicația stochează clientul editat în baza de date și afișează un mesaj de success

Scenariu alternativ:

* Utilizatorul inserează valori invalide pentru parametrii clientului sau schimbă id-ul
* Se va afișa un mesaj de eroare iar utilizatorul va trebui să introducă datele corect
* Scenariul se întoarce în pasul 1

**Caz de utilizare**: Șterge client

Actor primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul selectează opțiunea de ștergere a unui client

2. Utilizatorul inserează id-ul clientului pe care vrea să îl șteargă

3. Utilizatorul apasă butonul DELETE pentru a șterge clientul

1. Aplicația șterge clientul din baza de date și afișează un mesaj de success

Scenariu alternativ:

* Utilizatorul inserează id-ul unui client care nu există
* Se va afișa un mesaj de eroare iar utilizatorul va trebui să introducă datele corect
* Scenariul se întoarce în pasul 1

**Caz de utilizare**: Afișare table clienți

Actor primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul selectează opțiunea de afișare a tabelului de clienți

5. Utilizatorul apasă butonul EDIT pentru a edita clientul

1. Aplicația afișează tabelul cu clienți
2. Scenariu alternativ:

* Tabelul cu clienți nu are nici un client de afișat deci comanda va eșua
* Utilizatorul trebuie să introducă mai întâi clienți în table
* Scenariul se întoarce în pasul 1

**Caz de utilizare**: Plasare comandă

Actor primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul selectează opțiunea de plasare a unei comenzi

2. Utilizatorul inserează numele clientului și numele produsului pentru comandă

3. Utilizatorul inserează cantitatea din produs pe care vrea să o comande

5. Utilizatorul apasă butonul PLACE ORDER pentru a plasa comanda

6. Comanda este inserată în baza de date

Scenariu alternativ:

* Utilizatorul inserează valori invalide pentru numarul de produse care nu se mai află in stoc
* Se va afișa un mesaj de eroare iar utilizatorul va trebui să introducă datele corect
* Scenariul se întoarce în pasul 1

**3.Proiectare**

Programarea orientată pe obiecte este o paradigma de programare care se concentrează pe obiecte ca elemente principale de construire a aplicațiilor. În POO, obiectele sunt instanțe ale claselor, care conțin date și metode asociate acestor date. POO are mai multe beneficii, cum ar fi encapsularea datelor, reutilizarea codului și abstracția.

Pentru acest proiect am folosit tehnica de Reflection. Reflection în Java este o tehnică care permite examinarea și manipularea obiectelor la timpul de execuție. Aceasta oferă posibilitatea de a obține informații despre clase, metode, câmpuri și constructori, chiar dacă acestea nu sunt cunoscute la momentul compilării. Prin intermediul reflecției, poți accesa și modifica proprietăți private ale obiectelor, invoca metode și crea noi instanțe ale claselor.

De asemenea, am mai folosit și unealta JavaDoc. Javadoc este o unealtă utilizată în limbajul de programare Java pentru generarea automată a documentației pentru codul sursă. Aceasta oferă o modalitate convenabilă de a crea documentație în format HTML pentru clase, metode, câmpuri și alte elemente ale codului Java. Documentația generată de Javadoc este adesea utilizată pentru a împărtăși informații despre modul de utilizare și funcționalitatea codului cu alți programatori.

**3.1 Organizarea în pachete**

**A picture containing diagram, line, technical drawing, parallel

Description automatically generated**

**3.2. Organizarea pe clase**

**A picture containing text, diagram, line, parallel

Description automatically generated**

**3.3. Diagrama UML de clase**

**A screenshot of a computer

Description automatically generated with medium confidence**

**4.Implementare**

***Clasa Client***

Această clasă servește drept model pentru stocarea și manipularea informațiilor despre clienți în cadrul aplicației de gestionare a depozitului.

***Clasa Produs***

Această clasă servește drept model pentru stocarea și manipularea informațiilor despre produse în cadrul aplicației de gestionare a depozitului. Poate fi utilizată pentru a crea obiecte Products și pentru a accesa și actualiza detaliile acestora.

***Clasa Order***

Această clasă servește drept model pentru stocarea și manipularea informațiilor despre o comandă în cadrul aplicației de gestionare a depozitului. Poate fi utilizată pentru a crea obiecte Order și pentru a accesa și actualiza detaliile acestora.

***Clasa Bill***

Această clasă servește drept model pentru stocarea informațiilor despre o factură generată pentru o comandă în cadrul aplicației de gestionare a depozitului. Poate fi utilizată pentru a crea obiecte Bill și pentru a accesa detaliile facturii, cum ar fi numele clientului, produsul și prețul facturat.

***Clasele ProductDAO, ClientDAO, OrderDAO, BillDAO***

Aceste clase extend clsaa AbstractDAO pentru a implementa anumite metode de editare, inserare, editare și afișare a clienților, produselor și a comenzilor din baza de date.

***Clasa AbstractDAO***

Această clasă se numește "AbstractDAO" și este o clasă generică care servește drept clasă de bază pentru toate clasele DAO (Data Access Object) din aplicație. Scopul clasei este să ofere metode generice pentru operațiile comune de acces și manipulare a datelor într-o bază de date.

**Metoda findAll()**

Metoda findAll() este o metodă esențială în cadrul clasei AbstractDAO și are rolul de a recupera toate înregistrările din tabela asociată clasei type. Această metodă urmărește un flux bine definit pentru a obține și a returna rezultatele dorite.

În începutul metodei, sunt inițializate variabilele de conexiune la baza de date: connection, statement și resultSet. Apoi, se construiește interogarea SELECT utilizând metoda createSelectQueryforAll(), care returnează o interogare SQL pentru a selecta toate înregistrările din tabela respectivă.

Un obiect ArrayList<T> numit rez este creat pentru a stoca înregistrările rezultate din baza de date. Acesta va fi ulterior returnat ca rezultat al metodei.

În interiorul blocului try, se realizează următoarele acțiuni. Mai întâi, se deschide conexiunea la baza de date utilizând ConnectionFactory.getConnection(). Apoi, se pregătește declarația SQL utilizând connection.prepareStatement(query), unde query reprezintă interogarea SELECT generată anterior.

În continuare, interogarea este executată apelând metoda executeQuery() pe obiectul statement. Rezultatele obținute sunt stocate în obiectul resultSet.

A screen shot of a computer program

Description automatically generated with medium confidence

**Metoda deleteById()**

Metoda deleteById(int id) este responsabilă de ștergerea unei înregistrări din tabela asociată clasei type, utilizând identificatorul specificat id. Această metodă urmează un flux bine definit pentru a executa ștergerea și returna rezultatul acesteia.

A screen shot of a computer code

Description automatically generated with low confidence

**Metoda findById()**

Metoda findById(int id) este responsabilă de găsirea și returnarea unei înregistrări din tabela asociată clasei type, utilizând identificatorul specificat id. Această metodă urmează un flux bine definit pentru a executa interogarea și a returna rezultatul.

A screen shot of a computer program

Description automatically generated with low confidence

**Metoda createObjects()**

Metoda createObjects(ResultSet resultSet) este responsabilă de transformarea rezultatului obținut dintr-un obiect ResultSet într-o listă de obiecte de tip T. Această metodă utilizează informații despre clasa T și rezultatul interogării pentru a crea și inițializa obiecte corespunzătoare înregistrărilor găsite.

**A picture containing text, screenshot, software, multimedia software

Description automatically generated**

**Metoda insert()**

Metoda insert(T t) este utilizată pentru a insera un obiect de tip T în baza de date. Aceasta primește ca parametru obiectul pe care dorim să-l inserăm și îl adaugă în baza de date utilizând valorile câmpurilor obiectului.

A screen shot of a computer program

Description automatically generated with medium confidence

**Metoda update()**

Metoda update(T t) este utilizată pentru a actualiza un obiect de tip T în baza de date. Aceasta primește ca parametru obiectul pe care dorim să-l actualizăm și efectuează modificările corespunzătoare în baza de date.

**A screen shot of a computer program

Description automatically generated with low confidence**

**Metoda makeTable()**

Metoda makeTable(List<T> list) este utilizată pentru a crea și returna un obiect JTable pe baza unei liste de obiecte de tip T. Această metodă este utilă în contextul creării unei interfețe grafice pentru afișarea datelor din lista într-un tabel.

A picture containing text, screenshot, font

Description automatically generated

A picture containing text, screenshot, font

Description automatically generated

**Clasa ConnectionFactory**

Clasa ConnectionFactory este responsabilă de gestionarea conexiunii la baza de date în cadrul aplicației. Ea implementează un design pattern numit Singleton, pentru a asigura că există o singură instanță a clasei și o singură conexiune deschisă către baza de date.

De asemenea, clasa conține un obiect de tip Logger pentru a înregistra evenimente și erori, folosind API-ul java.util.logging.

Metoda createConnection() este responsabilă pentru crearea unei conexiuni la baza de date utilizând URL-ul, numele de utilizator și parola specificate. Dacă nu poate fi stabilită conexiunea, se înregistrează un mesaj de avertizare și se afișează stiva de apel pentru a identifica problema.

Clasa ConnectionFactory facilitează gestionarea conexiunii la baza de date, asigurând că aceasta este realizată într-un mod corect și sigur. Prin utilizarea design pattern-ului Singleton, se asigură că o singură instanță a clasei este creată și utilizată în cadrul aplicației, evitând astfel crearea redundantă a conexiunilor și menținând un control asupra resurselor.

**Clasa ProductBLL, Clasa OrderBLL, Clasa ClientBLL, Clasa BillBLL**

Aceste clase reprezintă calsele de logică. Această clase gestionează operațiile specifice legate de clienții, produsele, comenzile și facturile din cadrul sistemului.

**Clasa Controller**

Clasa Controller este clasa de control care gestionează evenimentele și acțiunile utilizatorului în cadrul interfeței grafice a aplicației. Aceasta implementează interfața ActionListener, astfel încât să poată asculta și răspunde la evenimentele generare de către utilizator.

În esență, clasa Controller funcționează ca intermediar între interfața grafică și nivelul de logică a aplicației, gestionând evenimentele utilizatorului și coordonând acțiunile corespunzătoare.

**Clasa View**

Reprezintă interfața grafică cu utilizatorul.

A screenshot of a computer

Description automatically generated

A picture containing text, screenshot, font, yellow

Description automatically generated

A screenshot of a computer screen

Description automatically generated with low confidence

**5.Rezultate**

Această aplicație trebuie să se descurce cu cerințele utilizatorului, astfel încât acesta să poată face operațiile dorite pentru clienți și produse. Cel mai important, această aplicație trebuie să îi permită utilizatorului să plaseze o comandă, care va fi în limita stocului disponibil și va fi stocată în tabelul corespunzător.

**6. Concluzii**

În concluzie această temă m-a ajutat să stăpânesc mai bine lucrul cu interfețe deoarece a fost nevoie de mai multe pentru a procura utilizatorului o experiență plăcută, și o interfață ușor de utilizat.

A fost folositoare și învățarea tehnicii de reflection pentru ca am aflat că poate să reducă considerabil cantitatea de cod ce trebuie scris într-un stfel de proiect.

**7.Bibliografie**

<https://www.dummies.com/article/technology/programming-web-design/java/how-to-use-javadoc-to-document-your-classes-153265/>

<https://app.diagrams.net/?src=about>

<https://www.digitalocean.com/community/tutorials/java-records-class>

<https://www.edureka.co/blog/logger-in-java>

<https://colorhunt.co/palette/206a5d81b214ffcc29f58634>

<https://stackoverflow.com/questions/13935934/java-jtable-column-headers-not-showing>