“Magazin”, Aplicatie realizata in C++ folosind QtCreator pentru GUI

De Marinescu Dragos.

1. Introducere

„StoreApplication” este o aplicație dezvoltată folosind C++ și Qt pentru gestionarea unui magazin. Aplicația permite adăugarea, modificarea, stergerea și vizualizarea produselor dintr-un magazin, precum și gestionarea unui coș de cumpărături. De asemenea, include funcționalități pentru sortare și filtrare a produselor, precum și exportul coșului de cumpărături în format CSV sau HTML.

2. Structura Proiectului

2.1. Fișierele și Modulele Principale

- **main.cpp**: Punctul de intrare al aplicației, care inițializează aplicația Qt și pornește interfața grafică.
- **Validator.h** și **Validator.cpp**: Conțin logica pentru validarea datelor de intrare.
- **ui.h** și **ui.cpp**: Implementarea interfeței utilizatorului fără interfață grafică (console).
- **repo_magazin.h** și **repo_magazin.cpp**: Definirea și implementarea repository-ului pentru gestionarea produselor.
- **service_magazin.h** și **service_magazin.cpp**: Serviciul de gestionare a produselor, care folosește repository-ul și validatorul.
- **magazinGUI.h** și **magazinGUI.cpp**: Implementarea interfeței grafice folosind Qt.
- **CosCRUDGUI.h** și **CosCRUDGUI.cpp**: Interfața grafică pentru gestionarea coșului de cumpărături.
- **CosReadOnlyGUI.h** și **CosReadOnlyGUI.cpp**: Interfața grafică pentru vizualizarea coșului de cumpărături.

- **MagazinGUIModele.h** și **MagazinGUIModele.cpp**: Modele pentru interfața grafică.
- **TableModel.h** și **TableModel.cpp**: Modele pentru afișarea datelor în tabele.

3. Descrierea Funcționalităților

3.1. Funcționalități de Bază

1. **Adăugare Magazin**:
    - Permite adăugarea unui nou produs în magazin, specificând numele, tipul, prețul și producătorul. Este validată conform constrângerilor definite în Validator.
2. **Modificare Magazin**:
    - Permite modificarea detaliilor unui produs existent. Verificările sunt făcute pentru a asigura că datele sunt valide.
3. **Stergere Magazin**:
    - Permite ștergerea unui produs din magazin pe baza numelui și tipului.
4. **Vizualizare și Filtrare**:
    - Vizualizează produsele existente și permite filtrarea acestora după nume, preț sau producător.
5. **Sortare**:
    - Sortarea produselor după nume, preț sau combinație nume/tip.
6. **Raport**:
    - Generarea unui raport privind tipurile de produse din magazin.
7. **Gestionarea Coșului de Cumpărături**:

- Adăugarea de produse în coș, ștergerea produselor și exportul coșului în fișiere CSV sau HTML.

3.2. Interfața Grafică

Interfața grafică este implementată folosind Qt și include:

- **MagazinGUIModele**: Gestionează interacțiunea cu datele magazinului.
- **CosCRUDGUI** și **CosReadOnlyGUI**: Permite utilizatorului să vizualizeze și să gestioneze coșul de cumpărături.

4. Principii de Programare Orientată pe Obiect (OOP)

4.1. Clase și Moștenire

- **Clasele principale**: Magazin, ServiceMagazin, RepoMagazin, Validator.
- **Moștenire**: Nu este folosită direct în codul furnizat, dar este folosită extensiv în designul interfețelor Qt.

4.2. Encapsulare

- **Clasele**: Toate clasele sunt bine encapsulate, cu membri de date protejați și metode publice pentru manipularea acestora.

4.3. Polimorfism

- **Interfețele grafice și modelele**: Utilizarea polimorfismului în Qt pentru a crea interfețe grafice care se adaptează la diferite tipuri de date.

5. Pattern-uri de Design

5.1. Pattern-ul Observer

- **Observer Pattern**: Este utilizat pentru actualizarea automată a interfeței utilizatorului atunci când datele din RepoMagazin se schimbă. Clasa MagazinGUIModele observă modificările din ServiceMagazin și actualizează interfața.

Module:

1. Modulul domain

Descriere

Modulul domain definește clasa Magazin, care reprezintă un produs într-un sistem de magazin. Acesta include metodele de acces pentru a obține atributele produsului și funcții de comparare pentru ordonarea produselor.

Clase și Funcționalități

- *Clasa Magazin*
  - *Atribute:*
    - *type (string) - Tipul produsului.*
    - *name (string) - Numele produsului.*
    - *producer (string) - Producătorul produsului.*
    - *price (double) - Prețul produsului.*
  - *Metode:*
    - *string get_type() const: Returnează tipul produsului.*
    - *string get_name() const: Returnează numele produsului.*
    - *string get_producer() const: Returnează producătorul produsului.*
    - *double get_price() const: Returnează prețul produsului.*
- *Funcții de Comparare:*
  - *bool cmp_name(const Magazin& m1, const Magazin& m2): Compară două obiecte Magazin după nume.*
  - *bool cmp_price(const Magazin& m1, const Magazin& m2): Compară două obiecte Magazin după preț.*

- ***bool cmp_name_type(const Magazin& m1, const Magazin& m2): Compară două obiecte Magazin după nume și tip, utilizat pentru ordonarea complexă.***

Principii de Design

- ***Encapsulation: Atributele private ale clasei Magazin sunt accesibile doar prin intermediul metodelor getter, protejând datele interne.***
- ***Abstracție: Funcțiile de comparare permit manipularea obiectelor Magazin fără a dezvălui detalii interne, facilitând sortarea și compararea produselor.***

---

2. Modulul cos_magazin

Descriere

Modulul cos_magazin gestionează coșul de magazin (CosMagazin), care permite adăugarea, ștergerea și manipularea produselor din coș. Include și funcționalități pentru adăugarea aleatorie a produselor.

Clase și Funcționalități

- ***Clasa CosMagazin***
  - ***Atribute:***
    - ***cosMagazin (vector<Magazin>) - Lista produselor aflate în coș.***
  - ***Metode:***
    - ***void adauga_magazin_cos(const Magazin& m): Adaugă un produs în coș și notifică observatorii.***
    - ***void adauga_magazin_random_cos(vector<Magazin> magazine, int nr): Adaugă un număr specificat de produse aleatorii în coș și notifică observatorii.***
    - ***const vector<Magazin>& get_all_cos(): Returnează lista produselor din coș.***

        - *void sterge_cos(): Șterge toate produsele din coș și notifică observatorii.*

Principii de Design

- *Observer Pattern: Implementarea pattern-ului Observer permite notificarea automată a interfețelor grafice despre modificările din coș.*
- *Separation of Concerns: Modulul se ocupă doar cu logica de gestionare a coșului, separând funcționalitatea de prezentare și manipulare a datelor.*

---

3. Modulul undo

Descriere

Modulul undo gestionează acțiunile de anulare (undo) pentru modificările efectuate asupra colecției de produse, permițând revenirea la starea anterioară a coșului după acțiuni precum adăugarea, modificarea sau ștergerea produselor.

Clase și Funcționalități

- *Clasa ActiuneUndo (clasă abstractă)*
    - *Metode:*
        - *virtual void doUndo() = 0: Metodă virtuală pură pentru revenirea la starea anterioară.*
- *Clasa UndoAdauga*
    - *Atribute:*
        - *magazin_adaugat (Magazin) - Produsul adăugat.*
        - *rep (RepoAbstract&) - Referință la depozitul de produse.*
    - *Metode:*
        - *void doUndo() override: Îndepărtează produsul adăugat anterior.*

- ***Clasa UndoModifica***
  - ***Atribute:***
    - ***magazin_vechi (Magazin) - Produsul înainte de modificare.***
    - ***magazin_nou (Magazin) - Produsul după modificare.***
    - ***rep (RepoAbstract&) - Referință la depozitul de produse.***
  - ***Metode:***
    - ***void doUndo() override: Reîntoarce produsul la starea anterioară.***
- ***Clasa UndoSterge***
  - ***Atribute:***
    - ***magazin_sters (Magazin) - Produsul șters.***
    - ***rep (RepoAbstract&) - Referință la depozitul de produse.***
  - ***Metode:***
    - ***void doUndo() override: Reintroduce produsul șters în depozit.***

Principii de Design

- ***Command Pattern: Permite definirea și gestionarea acțiunilor de undo/redo prin crearea de comenzi individuale pentru fiecare acțiune.***
- ***Single Responsibility Principle: Fiecare clasă de undo se ocupă cu un singur tip de acțiune, fie că este vorba de adăugare, modificare sau ștergere.***

---

4. Modulul TabelModel

Descriere

Modulul TabelModel definește un model de date pentru vizualizarea produselor într-un tabel utilizând Qt. Acesta permite integrarea cu componentele de interfață grafică Qt pentru a afisa datele despre produse.

Clase și Funcționalități

- ***Clasa TableModel***
  - ***Atribute:***
    - ***magazine (vector<Magazin>) - Lista produselor de afișat.***
  - ***Metode:***
    - ***int rowCount(const QModelIndex& parent = QModelIndex()) const override: Returnează numărul de rânduri (produse) din tabel.***
    - ***int columnCount(const QModelIndex& parent = QModelIndex()) const override: Returnează numărul de coloane (atribute) din tabel.***
    - ***QVariant data(const QModelIndex& index, int role = Qt::DisplayRole) const override: Returnează datele pentru o celulă specifică din tabel.***
    - ***void setMagazine(const vector<Magazin>& magazine): Actualizează lista de produse și notifică vizualizarea.***

Principii de Design

- ***Model-View Pattern: TableModel urmează modelul Model-View din Qt, separând datele de logica de prezentare.***
- ***Encapsulation: Datele interne ale modelului sunt protejate și gestionate prin metode publice.***

---

5. Modulul CosReadOnlyGUI

Descriere

Modulul CosReadOnlyGUI oferă o interfață grafică pentru vizualizarea produselor din coș sub formă de histogramă. Utilizatorul poate vedea reprezentarea grafică a produselor, inclusiv prețul acestora.

Clase și Funcționalități

- ***Clasa HistogramGUI***
  - ***Atribute:***
    - ***cos (CosMagazin&) - Referință la coșul de produse.***
  - ***Metode:***
    - ***void update() override: Reînnoiește interfața grafică atunci când coșul se actualizează.***
    - ***void paintEvent(QPaintEvent* ev) override: Desenează histogramă pe baza produselor din coș.***

Principii de Design

- ***Observer Pattern: HistogramGUI se actualizează automat când coșul de produse se schimbă, respectând pattern-ul Observer.***
- ***Single Responsibility Principle: Clasa se ocupă exclusiv cu prezentarea vizuală, delegând logica de manipulare a datelor altor module.***

---

6. Modulul CosCrudGUI

Descriere

Modulul CosCrudGUI oferă o interfață grafică pentru gestionarea produselor din coș, inclusiv adăugarea și ștergerea produselor. Acesta utilizează Qt pentru a construi interfețe intuitive și interactive pentru utilizatori.

Clase și Funcționalități

- ***Clasa CosGUILista***
  - ***Atribute:***

        - ***cos (CosMagazin&) - Referință la coșul de produse.***
        - ***lst (QListWidget*) - Widget pentru listarea produselor.***
        - ***btn (QPushButton*) - Buton pentru curățarea coșului.***
        - ***btnrandom (QPushButton*) - Buton pentru adăugarea aleatorie de produse.***
    - ***Metode:***
        - ***void loadlist(const vector<Magazin>& m): Încarcă lista produselor în widget.***
        - ***void initGUI(): Inițializează interfața grafică.***
        - ***void connectSignals(): Conectează semnalele și sloturile pentru gestionarea evenimentelor.***

Principii de Design

- ***Observer Pattern: CosGUILista se actualizează automat când coșul se schimbă, respectând pattern-ul Observer.***
- ***Single Responsibility Principle: Clasa se ocupă exclusiv cu logica interfeței grafice și interacțiunea cu utilizatorul.***

---

7. Modulul observer

Descriere

Modulul observer implementează pattern-ul Observer, care permite obiectelor să fie notificate despre schimbările de stare ale altor obiecte. Acesta include atât interfața pentru observatori, cât și gestionarea notificărilor.

Clase și Funcționalități

- ***Clasa Observer (interfață)***
    - ***Metode:***

        - ***virtual void update() = 0: Metodă virtuală pură pentru actualizarea observatorului.***

- ***Clasa Observable***
    - ***Atribute:***
        - ***observers (vector<Observer*>) - Lista observatorilor abonați.***
    - ***Metode:***
        - ***void addObserver(Observer* obs): Adaugă un observator la lista de observatori.***
        - ***void removeObserver(Observer* obs): Elimină un observator din lista de observatori.***
        - ***void notify(): Notifică toți observatorii despre schimbările de stare.***

Principii de Design

- ***Observer Pattern: Permite crearea unei relații de tip subiect-observator între obiecte, facilitând notificările automate.***
- ***Single Responsibility Principle: Clasa Observable este responsabilă doar pentru gestionarea observatorilor și notificarea acestora.***
