

**DOCUMENTATIE TEMA 4**

**Kasler Madalina**

**Grupa 30221**

**Profesor Laborator: Dorin Moldovan**

Proiectarea și implementarea unui sistem de management al livrării alimentelor pentru o companie de catering. Clientul poate

comandați produse din meniul companiei. Sistemul ar trebui să aibă trei tipuri de utilizatori care se conectează

folosind un nume de utilizator și o parolă: administrator, angajat obișnuit și client.

Administratorul poate:

 Importați setul inițial de produse care vor completa meniul dintr-un fișier .csv.

 Gestionați produsele din meniu: adăugați / ștergeți / modificați produse și creați produse noi

compus din mai multe produse (un exemplu de produs compus ar putea fi denumit „zilnic

meniu 1 ”compus dintr-o supă, o friptură, o garnitură și un desert).

 Generați rapoarte despre comenzile efectuate luând în considerare următoarele criterii:

o intervalul de timp al comenzilor - ar trebui generat un raport cu comenzile efectuate

între o anumită oră de început și o anumită oră de sfârșit, indiferent de dată.

o produsele comandate de mai multe ori de câte ori până acum.

o clienții care au comandat mai mult de un număr specificat de ori și valoarea

din comanda a fost mai mare decât o sumă specificată.

o produsele comandate într-o zi specificată cu numărul de ori pe care le au a fost comandat.

Clientul poate:

 Înregistrați-vă și utilizați numele de utilizator și parola înregistrate pentru a vă conecta în sistem.

 Vizualizați lista produselor din meniu.

 Căutați produse pe baza unuia sau mai multor criterii, cum ar fi cuvânt cheie (de exemplu, „supă”), evaluare, numărul de calorii / proteine ​​/ grăsimi / sodiu / preț.

 Creați o comandă formată din mai multe produse - pentru fiecare comandă data și ora vor fi

a persistat și va fi generată o factură care va lista produsele comandate și prețul total

a ordinului.

Angajatul este notificat de fiecare dată când o nouă comandă este efectuată de un client, astfel încât acesta să poată pregăti livrarea produselor comandate.

2. Obiectivul temei

2.1 Obiectivele principale

Ca si obiectiv principal al acestei teme, a fost creearea unei aplicatii care va pune la dispozitia un sisteme de management al unui sistem de catering. Aceasta aplicatie o sa permita utilizatorului sa introducă clienți, produse, dar să facă și comenzi, sa caute produse, sa vada produsele. Daca comanda a fost validă, se va putea genera și o factură pentru aceasta, cu numele clientului, produsele, cantitatea si costul. Daca comanda nu va fi valabila, se va afisa o eroare pentru a instiinta clientul că stocul de produse nu este suficient.

2.2 Obiectivele secundare

1. Serializare

Serializarea in Java este transformarea unui obiect (doar variabile nu si metode) intr-o secventa de octeti sau caractere din care ulterior poate fi refacut obiectul (deserializare).

Serializare unui obiect este necesara cand dorim un mod usor de a salva informatia sau de a o trimite ca flux de date catre alt sistem prin retea.

Pentru a serializa un obiect, o clasa trebuie sa implementeze interfata Serializable.

Pentru a controla ce atribute nu sunt serializate folosim la definire cuvantul transient.

ObjectOutputStream este clasa prin care serializam un obiect, metoda writeObject.

ObjectInputStream este clasa prin care deserializam un obiect, metoda readObject.

Pentru tipurile primitive de date Java pune la dispozitie si clasele DataOutputStream si DataInputStream.

2.Composite Pattern

Modelul compozit este un model de proiectare de partiționare și descrie un grup de obiecte care este tratat la fel ca o singură instanță a aceluiași tip de obiect. Intenția unui compozit este de a „compune” obiecte în structuri de copac pentru a reprezenta ierarhii parțiale. Vă permite să aveți o structură de copac și să solicitați fiecărui nod din structura de copac să efectueze o sarcină.

Așa cum este descris de Gof, „Compuneți obiecte în structura arborelui pentru a reprezenta ierarhii parțiale. Compozit permite clientului să trateze obiecte individuale și compoziții de obiecte în mod uniform ”.

Atunci când se ocupă de date structurate în arbori, programatorii trebuie adesea să discrimineze între un nod frunză și o ramură. Acest lucru face codul mai complex și, prin urmare, predispus la erori. Soluția este o interfață care permite tratarea obiectelor complexe și primitive în mod uniform.

În programarea orientată pe obiecte, un compozit este un obiect conceput ca o compoziție de unul sau mai multe obiecte similare, toate prezentând funcționalități similare. Aceasta este cunoscută ca o relație „are-a” între obiecte.Conceptul cheie este că puteți manipula o singură instanță a obiectului la fel cum ați manipula un grup de ele. Operațiunile pe care le puteți efectua pe toate obiectele compozite au adesea o relație cu cel puțin comun numitor.Modelul compozit are patru participanți:

Component - Component declară interfața pentru obiectele din compoziție și pentru accesarea și gestionarea componentelor sale copil. De asemenea, implementează comportamentul implicit pentru interfața comună tuturor claselor, după caz.

Leaf - Leaf definește comportamentul obiectelor primitive din compoziție. Reprezintă obiecte frunze în compoziție.

Compozit - Compozit stochează componente secundare și implementează operațiuni legate de copii în interfața componentelor.

Client - Clientul manipulează obiectele din compoziție prin interfața componentei.

Clientul folosește interfața clasei de componente pentru a interacționa cu obiecte din structura compoziției. Dacă destinatarul este o frunză, atunci cererea este tratată direct. Dacă destinatarul este un compozit, atunci acesta transmite de obicei cererea către componentele sale copil, efectuând eventual operații suplimentare înainte și după redirecționare.

Într-o organizație, are manageri generali și sub manageri generali, pot exista manageri și sub manageri pot exista dezvoltatori. Acum puteți seta o structură de copac și să cereți fiecărui nod să efectueze operațiuni comune, cum ar fi getSalary ().

Modelul de design compozit tratează fiecare nod în două moduri:

1) Compozit - Compozit înseamnă că poate avea și alte obiecte sub el.

2) frunză - frunză înseamnă că nu are obiecte sub ea.

3.Observer

Observatorul este un model comportamental de proiectare. Specifică comunicarea între obiecte: observabil și observator. Un observabil este un obiect care notifică observatorii despre schimbările în starea sa.

De exemplu, o agenție de știri poate notifica canalele atunci când primește știri. Primirea de știri este ceea ce schimbă starea agenției de știri și determină notificarea canalelor.

Modelul de observator este utilizat atunci când există o relație între mai multe între obiecte, cum ar fi dacă un obiect este modificat, obiectele sale dependente trebuie notificate automat. Modelul observator se încadrează în categoria modelului comportamental.

Implementare

Modelul observator folosește trei clase de actori. Subiect, observator și client. Subiectul este un obiect care are metode de atașare și detașare a observatorilor la un obiect client. Am creat o clasă abstractă Observer și o clasă concretă Subiect care extinde clasa Observer.

ObserverPatternDemo, clasa noastră demo, va folosi obiectul și obiectul clasei concrete pentru a arăta modelul observatorului în acțiune.

4. Principiul expresiei Lambda

Cu câțiva ani în urmă, experții au discutat despre extinderea limbajului Java pentru programarea funcțională. În 2009, a devenit clar faptul că Java fără lambda ar arăta învechit în comparație cu alte limbi de programare, deoarece hardware-ul multiprocesor modern și multi-core necesită suport simplu pentru paralelizarea programelor. De aceea avem nevoie de expresii lambda.

În Java, toate colecțiile au o metodă forEach, moștenită de la interfața super-interfață Iterable. Acesta a existat de la Java 5, a fost extins la versiunea 8. Metoda forEach iterează peste toate elementele colecției și aplică funcția fiecărui element.

O expresie lambda este o funcție anonimă care oferă un comportament parametrizat. Se compune dintr-o listă de opțiuni de returnare și de excepții care sunt selectate. O instanță a unei expresii lambda java poate fi atribuită oricărui Iterable care se potrivește cu definiția, cu condiția să fie funcțională.

Puncte importante pentru înțelegerea expresiilor lambda:

Elementul din stânga săgeții (->) este parametrii lambda. În acest caz, parametrul de intrare este definit ca param String.

În dreapta săgeții (->) este corpul lambdei.

Corpul este locul unde are loc procesarea reală a lambda, adică determină logica sa. De obicei, o expresie lambda java are logică simplă.

Opțiunile sintaxei pentru lista de parametri

Există mai multe opțiuni de sintaxă pentru lista de parametri. Iată o versiune simplificată a sintaxei lambda:

LambdaExpression.

LambdaParameters '->' LambdaBody LambdaParameters.

Identificator "(" ParameterList ")" LambdaBody.

Blocul expresiei.

Lista de parametri este fie o listă delimitată prin virgulă în paranteze, fie un identificator fără paranteze. Dacă este utilizată o listă de console, trebuie să decideți dacă tipul de parametru va fi specificat pentru toate sau dacă nu va fi specificat, atunci acesta va fi determinat automat de compilator.

6.Invariant

În programarea computerizată, în special programarea orientată pe obiecte, un invariant de clasă (sau invariant de tip) este un invariant utilizat pentru constrângerea obiectelor unei clase. Metodele clasei ar trebui să păstreze invariantul. Invariantul clasei constrânge starea stocată în obiect.

Invarianții de clasă sunt stabiliți în timpul construcției și întreținuți constant între apelurile către metodele publice. Codul din cadrul funcțiilor poate sparge invarianții atâta timp cât invarianții sunt restabiliți înainte ca o funcție publică să se încheie. Cu concurență, menținerea invariantului în metode necesită de obicei o secțiune critică pentru a fi stabilită prin blocarea stării folosind un mutex.

Un invariant de obiect, sau invariant de reprezentare, este o construcție de programare computerizată constând dintr-un set de proprietăți invariante care rămân necompromise indiferent de starea obiectului. Acest lucru asigură că obiectul va îndeplini întotdeauna condiții predefinite și că, prin urmare, metodele pot face referire întotdeauna la obiect fără riscul de a face prezumții inexacte. Definirea invarianților de clasă poate ajuta programatorii și testerii să prindă mai multe erori în timpul testării software-ului

Efectul util al invarianților de clasă în software-ul orientat obiect este îmbunătățit în prezența moștenirii. Invarianții de clasă sunt moșteniți, adică „invarianții tuturor părinților unei clase se aplică clasei în sine”

Moștenirea poate permite claselor descendente să modifice datele de implementare ale claselor părinte, deci ar fi posibil ca o clasă descendentă să schimbe starea instanțelor într-un mod care le-a făcut invalide din punctul de vedere al clasei părinte. Îngrijorarea pentru acest tip de descendenți care se comportă greșit este unul dintre motivele pentru care proiectanții de software orientați pe obiecte oferă favorizarea compoziției în locul moștenirii (adică moștenirea rupe încapsularea).

Cu toate acestea, deoarece invarianții de clasă sunt moșteniți, invariantul de clasă pentru o anumită clasă constă din orice afirmații invariante codate imediat pe acea clasă împreună cu toate clauzele invariante moștenite de la părinții clasei. Aceasta înseamnă că, chiar dacă clasele descendente pot avea acces la datele de implementare ale părinților lor, invariantul clasei îi poate împiedica să manipuleze acele date în orice mod care produce o instanță nevalidă în timpul rulării.

3. Proiectare

Pentru ca implementarea acestei aplicatii sa fie mult mai simplă, am ales să imi structurez proiectul in mai multe pachete: BusinessLayer, DataLayer, Presentation. In clasa BusinessLayer se gasesc cele 5 clase si o interfata care fac posibila implementarea aplicației. In primul rand avem clasele de BaseProduct, CompositeProduct si MenuItem in care exista atrubite pt aceste tipuri de obiecte. Un composite product este alcatuit din BaseProduct sau MenuItems, deci aceasta va contine un array de Menu Items. Interfata IDeliveryServiceProcessing va contine toate metodele care vor fi implementate pt Administrator si Client. Apoi va urma Clasa DeliveryService care va contine implementarile efective pt metodele din interfata DeliveryServiceProcessing pentru ca aceasta clasa va implementa acea interfata.

4.Implementare

Clasele MenuItem, BaseProduct si CompositeProduct sunt cele 3 clase care formeaza principalul model al aplicatiei, alaturi de clasa Order care se va baza pe aceste produse.

Clasa DeliverySerilzator va fi clasa in care se va face deserializarea si serializarea datelor.

Clasa DeliveryService va avea implementate metodele pentru adaugarea unui produs, pentru stergerea si modificarea unui produs, pentru vizualizarea tuturor produsele, pentru comanda, generat de facture.

Clasele Login,Administrator si Client si Employee sunt clasele de tip View care se vor ocupa de cele 4 ferestre care se vor deschide odata cu aplicația, pentru a se realiza interfața grafică.

5.Concluzii

În concluzie, realizarea acestui proiect a fost foarte utila, pentru ca am reamintit o mare parte din conceptele invatate in semestrele anterioare, atat la Programare Orientata pe Obiect cat și la Baze de Date.

O dezvoltare ulterioara ar fi adaugarea de bonusuri pentru clientii fideli in functie de comenzi, promotii pentru meniuri. si a aa a a aa a a a a