**DOCUMENTAȚIE**

**TEMA 4**

**RESTAURANT**

Student: Mateș Claudia

Grupa: 30224

**CUPRINS**

1. Cerință ........................................................................................................................ 3
2. Obiective .................................................................................................................... 3
   1. Obiective principale ............................................................................................. 3
   2. Obiective secundare ............................................................................................ 3
3. Analiza problemei și usecase .................................................................................... 3
4. Diagrama UML .......................................................................................................... 6
5. Structuri de date ....................................................................................................... 6
6. Implementare ........................................................................................................... 7
7. Rezultate ................................................................................................................... 9
8. Concluzie ................................................................................................................... 9
9. Dezvoltări ulterioare ................................................................................................. 10
10. **Cerințăv v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v**

Se cere implementarea managemantului unui restaurant care să aibă un administrator, bucătar și un chelner. Administratorul poate adăuga noi elemente meniului sau le poate șterge, iar chelnerul poate efectua aceleași operații pentru o comandă.

1. **Obiective v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v**
   1. Obiective principale

Implementarea managementului unui restaurant cu bucătar, administrator și chelner.

* 1. Obiective secundare

|  |  |  |
| --- | --- | --- |
| Obiectivul | Descrierea | Capitol |
| Composite pattern | Utilizat pentru crearea claselor MenuItem, BaseProduct, CompositeProduct | 6. Implementare |
| HashMap | Utilizat pentru a ține detaliile legate de comenzi | 5. Structuri de date |
| Design by Contract | Utilizat pentru a ține detaliile despre pre și post condiții | 6. Implementare |
| Fișier text | Utilizat pentru a creea factura pentru fiecare comandă | 5. Structuri de date  6. Implementare |
| Observable | Pentru comunicarea dintre chelner și bucătar | 6. Implementare |

1. **Analiza problemei și utilizare r r r r r r rr r r r r r r r r r r r r r r r r r r r r r r r r**

**Use case: s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s**

Nume use case: crearea unui base producct

Actori: administratorul;

Trigger: apăsarea butonului „Create Base Item”;

Precondiții:

* Pornirea aplicației
* Introducerea datelor în casetele corespunzătoare

Postcondoții:

* Introducerea datelor în arrayList de MenuItems

**Use case: s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s**

Nume use case: crearea unui composite product

Actori: administratorul;

Trigger: apăsarea butonului „Create composite Item”;

Precondiții:

* Pornirea aplicației
* Introducerea datelor în casetele corespunzătoare

Postcondoții:

* Introducerea datelor în arrayList-ul de MenuItems

**Use case: s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s**

Nume use case: ștergerea unui MenuItem

Actori: administratorul;

Trigger: apăsarea butonului „delete Item”;

Precondiții:

* Pornirea aplicației
* Introducerea datelor în casetele corespunzătoare

Postcondoții:

* Ștergerea datelor din arrayList

**Use case: s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s**

Nume use case: vizualizarea unui Meniului

Actori: administratorul;

Trigger: apăsarea butonului „ View Menu”;

Precondiții:

* Pornirea aplicației

Postcondoții:

* Crearea unui Jtable conținând elementele meniului

**Use case: s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s**

Nume use case: crearea unei comenzi

Actori: chelnerul și bucătarul;

Trigger: apăsarea butonului „Create Order”;

Precondiții:

* Pornirea aplicației
* Introducerea datelor în casetele corespunzătoare

Postcondoții:

* Introducerea datelor în map
* Notificarea bucătarului

**Use case: s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s**

Nume use case: vizualizarea comeenzilor

Actori: chelnerul;

Trigger: apăsarea butonului „View Orders”;

Precondiții:

* Pornirea aplicației
* Introducerea datelor în casetele corespunzătoare

Postcondoții:

* Crearea unui Jtable conținând comenzile

**Use case: s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s**

Nume use case: crearea chitanței

Actori: chelnerul;

Trigger: apăsarea butonului „Create Bill”;

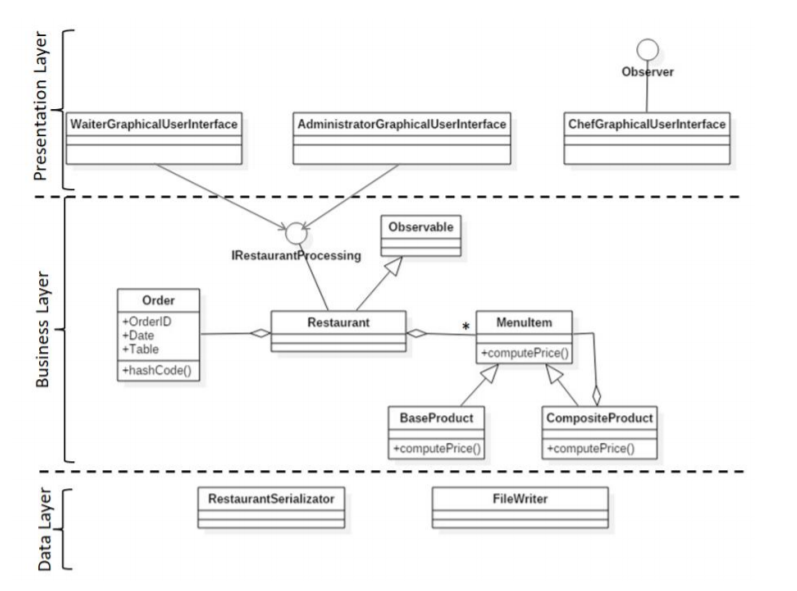
Precondiții:

* Pornirea aplicației
* Introducerea datelor în casetele corespunzătoare

Postcondoții:

* Crearea unui fișier text conținând cheia și numele comenzii șiprțil acesteia

1. **Diagrama UML v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v**



1. **Structuri de date v v v v v v v v v v v v v v v v v v v v v v v v v v v v**

Programul folosește ArrayListuri pentru a reține meniul creat de administrator.

Un map care are un obiect de tip Order drept cheie și datele un ArrayList de MenuItems care practic ține detaliile despre comenzi. E utilizat pentru a crea tabelul de vizualizare și evident pentru creearea facturii.

Fișiere text utilizate pentru crearea facturii.

1. **Implementarea**

Programul conține 3 pachete în cadrul cărora acumulează nouă clase. Unul din aceste pachete este view conținând cele 3 interfețe: WaiterGUI, AdministratorGUI, ChefGUI.

ChefGUI implementează interfața Observer astfel poate fi notificată când primește o nouă comandă. Pentru a notiffica bucătarul, in metoda update, metoda interfeței care trebuie să fie suprascrisă în mod obligatoriu în cadrul acestei clase, am creat un thread care la fiecare comandă nouă afișează mesajul : „New Order”, iar înntre două comenzi afișează mesajul: „Waiting”. Interfața WaiterGUI conține butoane, text fielduri și un combobox, acesta din urmă e pentru a selecta mâncarea dorită din meniu. Elementele comboboxului sunt adăugate în momentul în care administratorul crează un nou element pentru meniu. AdministratorGUI conține doar textfielduri și butoane pentru a adauga datele necesare și pentru a efectua operațiile administratorului.

Al doilea pachet este Order, iar acesta conține 5 clase. Trei dintre ele sunt utilizate pentru meniu: MenuItem, BaseProduct, CompositeProduct. Cele 3 clase sunt create folosind ccomposite design pattern. Composit design pattern este un model de despărțire și descrie un grup de obiecte care sunt tratate la fel ca o singură instanță a aceluiași tip de obiect. Intenția unui compozit este de a "compune" obiecte în structuri de copaci pentru a reprezenta ierarhiile parțiale. Vă permite să aveți o structură arborescentă și să cereți fiecărui nod din structura copacului să efectueze o sarcină.

Composite design pattern are 4 participanti:

Componentă - Componenta declară interfața pentru obiectele din compoziție și pentru accesarea și gestionarea componentelor copilului. De asemenea, implementează comportamentul implicit pentru interfața comună pentru toate clasele, după caz.

Leaf - Leaf definește comportamentul pentru obiectele primitive din compoziție. Acesta reprezintă obiectele frunzelor din compoziție.

Compozit - Composite stochează componente copil și implementează operațiunile legate de copil în interfața componentă.

Client - Clientul manipulează obiectele din compoziție prin intermediul interfeței componente. Clientul utilizează interfața de clasă componentă pentru a interacționa cu obiectele din structura compoziției. Dacă destinatarul este o frunză, atunci cererea este tratată direct. Dacă destinatarul este un compozit, atunci, de obicei, înaintează cererea către componentele copilului său, efectuând eventual operațiuni suplimentare înainte și după expediere. MenuItem fiind clasa abstractă extinsă de celelalte 2. BaseProduct crează elemente sipmle în meniu, iar Composite product crează proiecte puțin mai complexe. Metoda compositePrice() este suprascrisă de ambele subclase. Restul metodelor sunt pentru adăugare de elemente, ștergere de elemente și pentru a obține numele elementului.

Clasa Observer conține ID ul unei comenzi și numărul mesei. Pe baza acestora calculează hashcodul, fiind necesar sa suprascrie metoda hashCode(). Formula pentru a calcula hash-ul aleasă este: „(orderID+table)/3”. Pe lângă această metodă mai arre 2 settere pentru a seta cele 2 atribute.

Clasa Restaurant se poate considera cea mai complexă. Ea extinde clasa Observable necesar pentru a putea trrimite mesaje spre ChefGUI. Această clasă implementeză metoda start care va fi apelată pentru a porni aplicația, are o metodă pentru inițializarea action listenerilor pentru butoanele din interfețele WaiterGUI și AdministratorGUI și o metodă noty() care notifică Observerul. Tot în această clasă sunt pornite interfețele și tot de aici se crează fișierul text ce va servii drept chitantă. Tehnica de dezvoltare a software-ului Design by Contract (DBC) asigură software de înaltă calitate, garantând că fiecare componentă a unui sistem se ridică la nivelul așteptărilor sale. Ca dezvoltator care utilizează DBC, specificați contractele componente ca parte a interfeței componentei. Contractul specifică ce așteaptă acea componentă a clienților și ce clienți se pot aștepta de la acesta.

Bertrand Meyer a dezvoltat DBC ca parte a limbajului său de programare Eiffel. Indiferent de originea sa, DBC este o tehnică de design valoroasă pentru toate limbile de programare, inclusiv Java.

Centrală a DBC este noțiunea de afirmație - o expresie booleană despre starea unui sistem software. În timpul execuției, evaluăm afirmațiile la anumite puncte de control în timpul executării sistemului. Într-un sistem software valabil, toate afirmațiile sunt evaluate ca adevărate. Cu alte cuvinte, dacă orice afirmație este evaluată la falsă, considerăm că sistemul software este invalid sau rupt.

Noțiunea centrală a DBC se referă într-o oarecare măsură la macrocomanda #asset în limbajul de programare C și C ++. Cu toate acestea, DBC preia afirmații la un nivel de zillion.

În DBC, identificăm trei tipuri diferite de expresii:

[Aflați Java de la concepte de început la modele de design avansate în acest curs cuprinzător de 12 părți! ]

* precondiţii
* postconditii
* invarianți

precondiţii

Condițiile preliminare specifică condițiile care trebuie să fie păstrate înainte ca o metodă să poată fi executată. Ca atare, ele sunt evaluate chiar înainte ca o metodă să fie executată. Condițiile prealabile implică starea sistemului și argumentele transmise în metodă.

Condițiile preliminare specifică obligațiile pe care trebuie să le îndeplinească un client al unei componente software înainte de a putea invoca o anumită metodă a componentei. Dacă o condiție prealabilă eșuează, un bug se găsește într-un client al componentei software.

postconditii

În schimb, postcondițiile specifică condițiile care trebuie să se mențină după ce metoda finalizează. În consecință, postcondițiile sunt executate după terminarea unei metode. Postcondițiile implică starea vechiului sistem, noua stare a sistemului, argumentele metodei și valoarea returnată a metodei.

Postcondițiile specifică garanțiile pe care o componentă software le oferă clienților săi. Dacă o situație postcondiționată este încălcată, componenta software are un bug.

invarianți

Un invariant specifică o condiție care trebuie să dețină oricând un client poate invoca metoda unui obiect. Invarianții sunt definiți ca parte a unei definiții de clasă. În practică, invarianții sunt evaluați oricând înainte și după ce se execută o metodă pe orice instanță de clasă. O încălcare a unui invariant poate indica o eroare în client sau în componenta software.

Aserțiuni, moștenire și interfețe

Toate afirmațiile specificate pentru o clasă și metodele acesteia se aplică și pentru toate subclasele. De asemenea, puteți specifica afirmațiile pentru interfețe. Ca atare, toate afirmațiile unei interfețe trebuie să fie valabile pentru toate clasele care implementează interfața.

Classa StartApp din pachetul Start pornește aplicația.

1. **Rezultate**

Administaratorul va adăuga elemente creând astfel un meniu, chelnerul va prelua comenzile despre care bucătarul va fi notificat. Comenzile si meniul pot fi vizualizate în tabelele respective create pentru ele. Tot un rezultat este și factura creată.

1. **Concluzii**

Proiectul oferă o soluție simplistă pentru managementul unui restaurant. Acțiunile simulate corespund celor reale.

1. **Dezvoltări ulterioare**

Aplicația poate fi dezvoltată prin adăugarea opțiunii de angajați multiplii, adică mai mulți chelneri, mai mulți bucătari sau poate chiar mai mulți administratori.

Aplicația ar putea calcula nu doar prețul unor comenzi pentru a fi adăugate facturii, ci ar putea calcula și venitul zilnic al restaurantului.

Adăugarea feedbackului din partea bucătarului spre chelner.

În urma calculului venitului zilnic s-ar putea realiza o statistică de vinit lunar.

Dacă se va cunoaște venitul restaurantului se pot adăuga opțiuni de salarizare.

S-ar putea lua în calcul ordinea comenzilor și timpul de servire.

E posibilă introducerea unui feedback de la clienți.

Și în final i s-ar putea adăuga o parte de rezervări care arputea rula online.