

**Documentatie tema 4**

**Sistem pentru Organizarea unui Restaurant**

**Codrescu Razvan-Gabriel**

**Grupa 30224**

**Profesor Laborator Antal Marcel**

**Cuprins**

**1.Obiectivul temei**

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

**3. Proiectare (diagrame UML, structure de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)**

**4. Implementare**

**5. Cazuri de testare**

**6.Use-case**

**7. Concluzii, ce s-a invatat din tema, dezvoltari ulterioare**

**1.Obiectivul temei**

Propuneți, proiectați și implementați o aplicație pentru managementul unui restaurant. Sistemul de management al restaurantului ar trebui sa aiba 3 tipuri de utilizatori: administrator, chelner și bucătar. Administratorul poate adăuga, șterge și modifica produsele existente în meniu. Chelnerul poate crea o nouă comandă pentru o masa, poate adăuga elemente din meniu și poate procesa o factură pentru o anumită comandă. Bucătarul este notificat de fiecare data când trebuie să gătească mâncarea care a fost comandată prin intermediul chelnerului. Se consideră sistemul de clase din diagram de mai jos:   
Pentru a simplifica aplicația, se presupune că există un singur administrator, un singur bucătar și un singur chelner, iar astfel nu este nevoie de niciun fel de process de login. Considerații de implementare:

• Folosirea limbajului de programare Java

• Implementarea claselor cu maxim 300 de linii de cod

• Implementarea metodelor cu maxim 30 de linii de cod

• Folosirea javadoc pentru documentarea claselor și generarea fișierelor JavaDoc corespunzătoare

• Pentru fiecare comandă procesată de chelner se va genera o factură în format text

In rezolvarea acestei cerinte am implementat clasele necesare si le-am organizat in asa fel incat proiectul sa respecte MVC(model view controller – pentru o mai buna practica) si au fost realizate inclusiv cerintele pentru serializare si observer.

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

Principiile de baza sunt:

**1.Abstractizare** = Este posibilitatea ca un program să separe unele aspecte ale informației pe care o manipulează, adică posibilitatea de a se concentra asupra esențialului. Fiecare obiect în sistem are rolul unui “actor” abstract, care poate executa acțiuni, își poate modifica și comunica starea și poate comunica cu alte obiecte din sistem fără a dezvălui cum au fost implementate acele facilitați.Are loc dupa analiza problemei si specificarea cerintelor.

**2.Încapsularea** = numită și ascunderea de informații: Asigură faptul că obiectele nu pot schimba starea internă a altor obiecte în mod direct (ci doar prin metode puse la dispoziție de obiectul respectiv); doar metodele proprii ale obiectului pot accesa starea acestuia. Fiecare tip de obiect expune o interfață pentru celelalte obiecte care specifică modul cum acele obiecte pot interacționa cu el.

**3.Polimorfismul** = Este abilitatea de a procesa obiectele în mod diferit, în funcție de tipul sau de clasa lor. Mai exact, este abilitatea de a redefini metode pentru clasele derivate. De exemplu pentru o clasă Figura putem defini o metodă arie. Dacă Cerc, Dreptunghi, etc. ce vor extinde clasa Figura, acestea pot redefini metoda arie.

**4.Moștenirea** = Organizează și facilitează polimorfismul și încapsularea, permițând definirea și crearea unor clase specializate plecând de la clase (generale) deja definite - acestea pot împărtăși (și extinde) comportamentul lor, fără a fi nevoie de a-l redefini. Aceasta se face de obicei prin gruparea obiectelor în clase și prin definirea de clase ca extinderi ale unor clase existente. Conceptul de moștenire permite construirea unor clase noi, care păstrează caracteristicile și comportarea, deci datele și funcțiile membru, de la una sau mai multe clase definite anterior, numite clase de bază, fiind posibilă redefinirea sau adăugarea unor date și funcții noi.

**3. Proiectare (diagrame UML, structure de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)**

Pentru rezolvarea corecta a cerintelor, am implementat urmatoarele clase, salvate in pachete diferite:

1.model: BaseProduct, CompositeProduct, IRestaurantProcessing, MenuItem, Order, Restaurant, RestaurantSerialization

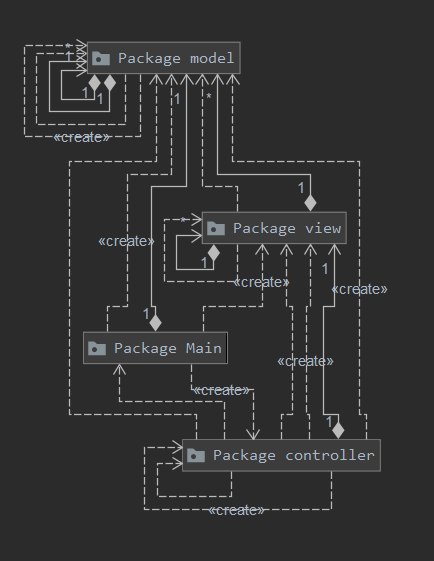
2.view: AddItemFrame, AdministratorFrame, ChefFrame, EditItemFrame, FrameStack, InitialFrame, OrderFrame, UniversalFrame, WaiterFrame.

3.controller: AddItemController, AdministratorController, ChefController, EditItemController, InitialController, NewOrderController, WaiterController.

4.Main: Start.

Clasele BaseProduct si CompositeProduct vor extinde clasa MenuItem, ceea ce inseamna ca fiecare obiect va avea cate un nume si un pret. De asemenea, in clasa Composite product vom avea si un ArrayList de MenuItem, pentru a retine din ce produse este alcatuit produsul respectiv. IRestaurantProcessing este interfata care va fi implementata in clasa Restaurant. In aceasta clasa va fi scrisa explicit fiecare functie care opereaza pe tabel. In clasa Order vom retine masa unde a fost ceruta comanda, data si pretul. In RestaurantSerialization se va face serializarea datelor respectiv scrierea si citirea in fisierul de serializare. In pachetul view, am implementat cate un frame pentru fiecare comanda care poate fi executata, folosind un stack de frameuri. In pachetul controller se afla controllerul fiecarui buton care poate fi apasat si care va deschide o fereastra noua. In main, am intializat un obiect de tipul Restaurant si un obiect de tipul InitialController, pentru a deschide frame-ul initial in care se afla cele 3 persoane care pot opera in restaurant. De precizat este faptul ca cheful este notificat cand este facuta o comanda, insa butonul propriu-zis de chef, nu va face nimic, deoarece nu l-am implementat sa faca un lucru anume.

In figura de mai jos este prezentata diagram UML:



**4. Implementare**

Clasele din view nu le voi explica pe toate deoarece sunt asemanatoare. Fiecare mosteneste UniversalFrame, care are date toate datele necesare(culoare, font si dimensiuni), dupa aceea in constructor am stabilit locul si dimensiunea fiecarui buton. De precizat este faptul ca unde a fost necesar, am creat o metoda displayTable, in care imi initializez un defaultTable, pe urma stabilesc fontul, culoarea, dimensiunea si un JScrollPane, pentru ca in cazul in care sunt prea multe elemente pentru a le putea vedea in tabel, sa apara o bara de scroll in partea dreapta a tabelului. Tabela va fi umpluta cu datele corespunzatoare din ArrayListul din Restaurant.

In clasa MenuItem vom implementa Serializable pentru a putea efectua ulterior serializari ale obiectelor in functiile urmatoare. Exista 2 parametrii: numele produsului(name) si pretul acestuia(price). Am realizat gettere si settere pentru fiecare parametru.

In clasa Order vom retine masa la care s-a facut comanda, data la care s-a facut comanda, si pretul comenzii, dupa care am realizat gettere si settere pentru fiecare din datele de mai sus.

In RestaurantSerialization sunt doua functii : read si write, in care se va face scrierea in fisier si citirea din fisier(serializarea datelor).

Restaurant va extinde Observable si va implementa interfata IRestaurantProcessing(alaturi de toate metodele din interfata respectiva). La inceputul clasei am declarat 4 obiecte:

menuItems – un ArrayList de MenuItem care va reprezenta lista de obiecte din meniu;

orders – un HashMap realizat cu Order si o lista de MenuItem care va reprezenta lista comenzilor;

rs – un obiect de tip RestaurantSerialization;

serFile – fisierul in care se va realiza serializarea;

Prima metoda este aceea de createMenuItem, care va adauga la lista de obiecte din meniu un nou obiect.

Cea de-a doua metoda, deleteMenuItem va realiza stergerea unui produs din meniu, dar si a tuturor produselor care il contin. Pentru aceasta, se va parcurge tot meniul si in cazul in care gaseste nume asemanator cu cel cautat se va sterge. Daca produsul este de tipul CompositeProduct(menu instanceof CompositeProduct) si contine produsul cautat, atunci va fi sters si el.

In updateMenuItem se va face schimbarea datelor unui produs din selectat din tabel, cu date noi introduse de catre utilizator. Mai intai se va cauta produsul, daca acesta este gasit, exista 2 posibilitati: fie produsul este baseProduct, caz in care se vor modifica si pretul si numele produsului, fie acesta este compositeProduct, caz in care se poate modifica doar numele acestuia.

In createOrder vom primi ca parametrii un obiect de tip Order si o liste de MenuItem iar cheful va fi notificat ca o comanda a fost facuta.

In computePrice se va calcula pretul unei comenzi date ca parametru.

In generateBill se va crea un fisier de tipul .txt in care se vor scrie urmatoarele date despre comanda data ca parametru : produsele comandate alaturi de pretul lor si totalul comenzii, la sfarsitul bonului.

In pachetul de controller vom avea cate un controller nou pentru fiecare buton care va fi apasat si care poate sa execute operatii asupra tabelelor de obiecte sau de comenzi. Controllerul principal este InitialController, in care doar se vor apela ActionListenerii pentru fiecare din butoanele care urmeaza sa aleaga tipul de operator care va executa modificari asupra datelor. De asemenea fiecaruia dintre cele 3 apeluri li se transmite si fisierul in care se face serializarea.

In AdministratorController avem 4 posibilitati : AddItem, EditItem, DeleteItem si Back. In afara de DeleteItem, care va executa stergerea randurilor selectate din tabelul alaturat, fiecare buton va deschide un frame nou iar butonul de Back va redeschide fereastra initiala. AddItemController va prezenta la randul sau 4 posibilitati. Initial se va alege ce fel de produs se va adauga. In cazul in care s-a selectat BaseProduct, va aparea fieldul in care sa se introduca numele si pretul iar in cazul in care este selectat CompositeProduct, va aparea doar fiedul cu nume si se va selecta din tabela alaturata produsele pe baza caruia se va realiza noul produs compus. Aceste operatii se vor executa doar in momentul apasarii butonului Submit. La apasarea acestui buton, daca produsul este composite se va cauta fiecare produs cu numele egal cu numele produselor selectate si pretul noului produs compus va fi alcatuit din suma tuturor produselor care il alcatuiesc. Dupa apasare, se va face pop la frameul curent si se va reveni la frameul initial cu operatiile administratorului.

Pentru EditItemController se va deschide un frame nou si exista 4 posibilitati: selectarea produsului(simplu sau compus), intoarcerea la fereastra anterioara si submit la modificari. La apasarea butonului submit, se verifica daca produsul este composite sau base, daca este base se vor modifica pretul si numele cu cele introduse de catre utilizator, daca este composite, se va modifica doar numele sau. La final se va face pop pentru a reveni la fereastra initiala.

In WaiterFrame avem 4 optiuni, revenirea la frameul precedent cu ajutorul butonului back, ComputePrice care va calcula totalul unei comenzi realizate, generateBill care va crea un fisier cu toate datele necesare unui bon si adaugarea unei comenzi care va deschide o noua fereastra. In fereastra noua deschisa de NewOrderController vom avea un camp in care sa introducem masa la care s-a facut comanda si meniul cu produsele din care se vor selecta produsele comandate cu .getRows().

In clasa Start, doar se va initializa un obiect de tipul Restaurant si un InitialController, pentru a se deschide fereastra principala. Se va transmite ca parametru args[0], deoarece programul trebuie rulat din commandBlock si pentru aceasta se va crea un fisier .jar.

**5. Cazuri de testare**

Pentru testarea programului, se va crea un fisier necesar executiei cerintei serializarii datelor pe parcursul executarii programului.

**6.Use-case**

Pasii prin care trece un utilizator pentru a utiliza aplicatia sunt:

1.Se va executa programul creat, introducand un fisier in care sa se faca serializarea.

2.Va alege rolul pe care vrea sa il execute in restaurant.

3.In functie de rolul ales, v-a executa operatiile dorite.

4.Daca meniul este gol, utilizatorul trebuie mai intai sa introduca date de la tastatura pentru a pune elemente in meniu(se va introduce un nume si un pret in campurile corespunzatoare).

5.In cazul introducerii unor date eronate, produsele nu vor fi introduse.

6.Dupa executarea operatiunii de umplere a meniului, toate actiunile devin disponibile pentru executare.

**7. Concluzii, ce s-a invatat din tema, dezvoltari ulterioare**

In urma realizarii acestei teme, mi-am imbunatatit cunostintele despre lucrul cu o interfata grafica, am invatat sa folosesc Serializable si Observable. De asemenea am invatat sa folosesc date de tipul HashMap. Printe imbunatatirile care se pot aduce asupra acestui program, se numara urmatoarele :

-adugarea mai multor validatori pentru date;

-adaugarea posibilitatii de a avea multipli utilizatori pentru fiecare rol(realizarea de conturi pentru fiecare).

-imbunatatirea interfetei prin adaugarea de obiecte noi si prin transpunerea intr-o inerfata more friendly user.

-realizarea rolului de chef care sa execute fiecare comanda primita sis a returneze un timp in care va fi gata comanda ceruta.

-crearea unor tabele cu mai multe date, pe langa nume si pret;

-pentru fiecare composite product acesta sa poata afisa si base producturile din care face parte.

-eficientizarea programului la un nivel mai avansat, programul fiind intr-o stare curent functionabila, dar nu si eficienta.