Tehnici de programare

Documentatie tema 4

Miklós Balázs

Grupa 30224

UTCN, Calculatoare

Restaurant management system

1. Obiectivul temei

Programul este folositor pentru a simula un sistem de control al unui restaurant, functionarea unui restaurant si facerea niste operatii cu produsele (meniul) restaurantului respectiv comenzile clientilor. Restaurantul are un administrator, un chef si un waiter.

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

In prima vedere a problemei am vazut ca aici trebuie sa folosim composite design pattern pentru a modela produsele restaurantului adica meniul restaurantului. Trebuiam sa folosim observer design pattern pentru a face notificarile daca ceva s-a schimbat in sistem(un nou order a fost facut sau un nou MunuItem a fost adaigat). In plus mai trebuiam sa folosim niste pre si postconditii pentru a verifica daca un order adaugat la lista de ordere (Map<Order, ArrayList<MenuItem>> orders) este valid sau un produs adaugat la lista produselor (ArrayList <MenuItem> menu) . Mai trebuiam sa facem un bill.txt pentru a salva comenzile facute si toata chestia trebuia serializat intr-un txt file unde programul ar trebui sa incarce si sa descarce datele la functionare.

Composite design pattern este o techinca de a modela un tip de relatie intre niste clase. In cazul nostru clasa MenuItem este clasa principala abstracta care poate sa fie BaseProduct sau CompositeProuct, avand intre ele relatie de mostenire. Fiind o clasa abstracta, MenuItem nu poate fi instantiat direct, deci putem definim un new BaseProduct sau CompositeProduct. CompositeProduct fiind un produs mai complicat, el foloseste ca relatie de agregare clasa MenuItem. Ceea ce inseamna ca un composite product poate sa fie un base product sau mai multe base product sau mai multe composite product. Acest tip de relatie este folositor in cazul in care vrem sa creem o relatie parinte copil infinita. De aceea este infinita fiindca un CompositeProduct poate sa aiba oricate de copii, daca unul dintre copii este CompositeProduct. Si asa mai departe putem sa mergem jos pe arborele de copil-parinte.

Observer design pattern este o tehnica care poate fi folosit si in cazul nostru. Clasa restaurant este o clasa Observable (extinde la clasa Observable) si Cheful este observerul. Aceasta inseamna ca la Chef o sa fie notificarile restaurantului. Adica cand se intampla o schimbare in sistem (adaugarea unui produs, modificarea unui produs, stergerea unui produs, adaugarea unei comanda, facerea Bill-ului) Cheful este notificat prin metoda notifyObservers. Cand lista produselor sau lista comenzilor este modificat, atunci se foloseste metoda setChanged.

Serialization este folosit pentru a nu pierde informatia prin inchiderea rularii a programului, datele intotdeauna vor fi incarcate intr-un fisier si vor fi descarcte de acolo la inceputul executiei pr ogramului. Si technica aceasta este folositoara in cazul nostru.

Folosirea asertiunilor si pre si post-conditiilor este buna pentru a valida, a verifica daca un order, comanda sau produs este valid sau nu pe care vream sa facem.

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

Programul contile 3 pachete: business_layer_pkg, data_layer_pkg si presentaion_layer_pkg. Fiecare pachet are propria sa rol in realizarea aplicatiei. Primul pachet este folosit pentru a face modelul aplicatiei. Aici sunt definite clasele BaseProduct, CompositeProduct, MenuItem, Order, Restaurant. Aici se observa folosirea technicii de composite design pattern intre clasele MenuItem, BaseProduct si CompositeProduct. Mai contine si o interfata numit RestaurantProcessing in care sunt definite operatiile facute asupra produselor si comenzilor de catre Administrator respectiv Waiter. Pachetul data later package contine clasele FileWriter si RestaurantSerializator. Aceste clase au scop de a scrie in respectiv a citi din fisiere. Aici se observa technica de serializare a aplicatiei, fiindca datele menu-lui si a listei de ordere este incarcat si incarcat intotdeauna la un fisier text, si comenzile facute sunt incarcate tot intr-un fisier .txt . Pachetul presentaton layer package are scop de a face interfata grafica de utilizator pentru Administrator, pentru waiter si pentru Chef. De aceeea clasele de aici (AdimnistratorGUI, WaiterGui si ChefGUI) extind la clasa Jframe, care creeaza fereastra pentru interfete.

Interfata grafica de utilizator AdministratorGUI contine urmatoare elemente. Prima data in panelul 1 are un Jtabel care arata datele meniului. In panelul 2 este o lista definit de unde putem alegem cu ce tip de produs vrem sa lucram (adaugam, stergem sau modificam), si mai este un buton pentru a arata tabelul modificat dupa o operatie numit Show menu items. Din lista de alegere putem alegem base si composite product. Mai jos panelurile 3, 4, 5 contin niste TextFielduri si butoane pentru a adauga, a sterge si a modifica un produs din lista de MenuItem menu.

Interfata de WaiterGUI contine cel mai sus tot un Jtable cu comenzile facute de catre waiterul. In partea de mijloc sunt niste butoane pentru a vede tabelul modificat de mai sus daca este facut vreo operatie fata de el, pentru a crea un nou order respectiv pentru a crea un order cu ID-ul introdus intr-un text-field. Cel mai jos apare un JtextArea, daca dam click pe butonul Compute Bill, in area aceasta aprare detaliile comandei la care vrem sa facem Bill-ul respectiv pretul care trebuie platit.

Diagrama UML am facut folosind modalitatile programului, softwarului Microsoft Word. Mai ales functiile insert shapes, insert text box etc.

4. Implementare

Pachtelul business layer pkg contine urmatoare clase:

Clasa **BaseProduct** extends **MenuItem** implements **Serializable** are variabilele de instanta int id, Stirng name, float price. Clasa are urmatoarele metode:

-public BaseProduct(int id, String name, float price)- constructorul.

-public int getID()- returneaza ID

-public void setID(int ID)- returneaza void, seteaza ID

-public String getName()- returneaza name

-public void setName(Stirng name)- returneaza void, modifica numele

-public float getPrice()- returneaza price.

-public void setPrice(float price)- seteaza price.

-public float computePrice()- returneaza float, metoda abstracta care este definit in clasa abstracta MenuItem. In cazul clasei BaseProduct doar returneaza pretul produsului.

Clasa **CompositeProduct** extends **MenuItem** implements **Serializable** are urmatoarele variabile de instanta: int ID, String name, float price, ArrayList <MenuItem> products. Clasa are urmatoarele metode:

-public CompositeProduct(int ID, String name, float price)- constructorul

-public int getID()- returneaza ID

-public void setID(int ID)- returneaza void, seteaza ID

-public String getName()- returneaza name

-public void setName(Stirng name)- returneaza void, modifica numele

-public float getPrice()- returneaza price.

-public void setPrice(float price)- seteaza price.

-public ArrayList<MenuItem> getProducts()- returneaza un ArrayList, lista din care este alcatuit un CompositeProduct.

-public float computePrice()- returneza float, metoda calculeaza pretul unui composit produs, parcurgand produsele simple a lui.

Clasa **MenuItem** implements **Serializable** este clasa parinte a claselor prezentate anterior, are urmatoarele variabile de instante: int ID, String name, float priece. Contine urmatoare metode:

-public BaseProduct(int id, String name, float price)- constructorul.

-public int getID()- returneaza ID

-public void setID(int ID)- returneaza void, seteaza ID

-public String getName()- returneaza name

-public void setName(Stirng name)- returneaza void, modifica numele

-public float getPrice()- returneaza price.

-public void setPrice(float price)- seteaza price.

-public float computePrice()-metoda abstracta definit de superclasa aceasta, folsite de catre clasele Base si Composite Product.

Clasa **Order** are urmatoarele variabile de instanta: int orderID, String date, int table.

-public Order(int ID, String date, int table)- constructorul

-public int getID()- returneaza ID

-public void setID(int ID)- returneaza void, seteaza ID

-public String getName()- returneaza name

-public void setName(Stirng name)- returneaza void, modifica numele

-public int getTable()-returneaza numarul mesei

-public void setTable(int nr)- modifica numarul unei masa la o comanda.

Clasa **Restaurant** extends **Observable** implements **RestaurantProcessing**. Are variabilele de instanta Map<Order,ArrayList<MenuItem>> orders, ArrayList<MenuItem> menu, ArrayList<MenuItem>baseProducts, ArrayList<MenuItem>compositeProducts. Are urmatoarele metode:
-public ArrayList<MenuItem> getBaseProducts() -returneaza lista produselor simple
-public void setBaseProducts(ArrayList<MenuItem> baseProducts)- returneaza void, modifica lista produselor complexe.
-public ArrayList<MenuItem> getCompositeProducts()- returneaza lista produselor mai complexe
-public void setCompositeProducts(ArrayList<MenuItem> compositeProducts)- modifica lista produselor complexe
-public ArrayList<MenuItem> getMenu()- returneaza lista meniului restaurantului
-public void createNewMenuItem(MenuItem m)- metoda definita de interfata RestaurantProcessing. Creaza un nou MenuItem si adauga la lista meniului daca a fost creat succes.
-public void editMenuItem(int ID, MenuItem newMenu)- face modificare la un MenuItem la lista meniului.
-public void createNewOrder(Order order, ArrayList <MenuItem>item, int[] menuID)- creaza un nou comanda, al treilea parametru a functiei indentifica numerele, ID-urile comenzilor pe care vrem sa cream.
-public float computePriceForAnOrder(Order order)- calculeaza pretul unui order
-public String generateBill(Order order)- genereaza un fisier txt pentru Bill
-public Map<Order,ArrayList<MenuItem>> getOrder()- returneaza lista de ordere

-public void setOrder(Map<Order,ArrayList<MenuItem>> order- modifica lista de ordere

Interfata **RestaurantProcessing** contine urmatoare metode:

-public void createNewMenuItem(MenuItem m);
-public void deleteMenuItem(int ID);
-public void editMenuItem(int ID, MenuItem newItem);
-public void createNewOrder(Order o, ArrayList<MenuItem> m, int[] menuID);
-public float computePriceForAnOrder(Order o);
-public String generateBill(Order order);

Interfata este implementata de catre clasa Restaurant.

Pachetul data layer pkg este folosit pentru a scrie respectiv a citi din fisiere. Comtine doua clase: FileWriter si RestaurantSerializator.

Clasa **FileWriter** are o singura variabila de instanta Restaurant restaurant. Metoda clasei este:

-public void writeBill()- scrie in fisierul Bill.txt comenzile facute de catre Waiter. Comenzile scrise in fisier sunt actualizate in fiecare modificare a lista de comenzi, de unde rezulta si modficarea fisierulu Bill.txt.

Clasa **RestaurantSerializator** contine urmatoarele metode statice:

-public static void writeFile()- returneaza void, scrie in fisierul binar datele meniu-lui, adica lista
-public static ArrayList<MenuItem> readFile()- returneaza un ArrayList<MenuItem>, meniul citit de la fisierul binar.

Functia read este folosita la inceperea programului, se citeste de la fisier datele meniului, iar functia write este folosita in toate cazuri in care se face orice fel de modificare asupra listei de MenuITem.

Pachetul presentation layer package este folosit pentru a crea GUI-urile pentru Administrator, Waiter respectiv Chef.

Clasa **AdministratorGUI** extend**s JFrame** are urmatoarele variabile de instanta: JPanel panels, Restaurant restaurant, Jtable menuTable, JscrollPane sp, JButton buttons, JTextField textfields, Choice menuItemTypes, int ID, JtextArea textArea. ID este initializat la 0 si intotdeauna este marit cand un nou produs este adaugata la lista produselor. Din butoane si text-fieldurile evident casunt mai multe. Metodele clasei:

-public AdministratorGUI(Restaurant r)- constructorul.
-public void createMainPanel()- creeaza panelul principal care are o orientare dupa axa Y, in el sunt adaugati celalalte paneluri.
-public void createPanel1()- returneaza void, creaza panelul 1
-public void createPanel2()- returneaza void, creaza panelul 2
-public void createPanel3()- returneaza void, creaza panelul 3
-public void createPanel4()- returneaza void, creaza panelul 4
-public void createPanel5()- returneaza void, creaza panelul 5
-public void createPanel6()- returneaza void, creaza panelul 6
-public void addActionListeners()- returneaza void, este o metoda unde sunt apelate celalalte metode care mai adauga cate un ActionListener pentru fiecare butoane.
-public Jtable createJTable()- returneaza un Jtable, creaza un tabel pentru lista produselor din restaurant.

-public void addShowMenuButtonListener()- returneaza void, adauga un actionListener pentru butonul ShowMenu
-public void addNewMenuItemBListener()- returneaza void, adauga un ActionListener pentru butonul addNewMenuItemB.
-public void addDeleteMenuItemBListener()- returneaza void, adauga un ActionListener pentru butonul deleteMenuItemB.
-public void addEditMenuItemBListener()- returneaza void, adauga un ActionListener pentru butonul editMenuItemB.

Clasa **WaiterGUI** extends **JFrame** are ca scop crearea fereastrei pentru Waiter. Are urmatoarele variabile de instanta: Restaurant restaurant, FileWriter fw, Jpanel panels, JTextField textfields, JButton buttons, int orderID. OrderID este initializat la 0 si intotdeauna este marit cand o noua comanda este adaugat. Evident ca din panele, butoane si textfieldurile sunt mai multe. Metodele clasei sunt:
-public AdministratorGUI(Restaurant restaurant)- este constructorul clasei, primeste ca parametru un restaurant.
-public void createMainPanel()- creeaza panelul principal care are o orientare dupa axa Y, in el sunt adaugati celalalte paneluri.
-public void createPanel1()- returneaza void, creaza panelul 1
-public void createPanel2()- returneaza void, creaza panelul 2
-public void createPanel3()- returneaza void, creaza panelul 3
-public void createPanel4()- returneaza void, creaza panelul 4
-public void createPanel5()- returneaza void, creaza panelul 5
-public void addActionListeners()- returneaza void, este o metoda unde sunt apelate celalalte metode care mai adauga cate un ActionListener pentru fiecare butoane.
-public Jtable createJTable()- returneaza un Jtable, creaza un tabel pentru lista comenzilor din restaurant.
-public void addShowOrdersListener()- returneaza void, adauga un ActionListener pentru metoda showOrders.
-public void addCreateOrderListener()- adauga un ActionListener pentru buttonul createOrder.
-public void createBillListene()- adauga un ActionListener pentru butonul createBill.

Clasa **ChefGUI** implements **Observer** are variabielele de instanta JtextArea ta, JPanel mainPanel, Restaurant restaurant. Metodele:
-public ChefGUI(Restaurant restaurant)- constructorul
-public void update(Observable o, Object arg)- metoda geneara automat de catre interfata Observer. Aici intotdeauna reactualiez textul lui JteaxtArea definita anterior cu noua comanda facuta.

5. Rezultate

Programul cel mai usor poate fi testat folosind interfata grafica de utilizator (GUI). Acolo apar niste butoane, pe fiecare butoan scrie functia lui, inaintea butoanelor de modificare a tabelelor din baza de data sunt niste textfield-uri in care utilizatorul trebuie sa introduca datele necesare pentru a face modificarile. Modificarile pot fi inserare, editare si stergere

Testarea de aceeapoate fi facut foarte usor, fiindca sunt implementate trei butoane de actualizarea a tabeleor din baza de date cu care putem sa vedem cum s-au modificat tabelele dupa orice update sau delete. Operatiile sunt facute corecte

6. Concluzii:

Aplicatiile de genu ceea ce este cerut in tema aceasta putem poate fi utilizat in cadrul unui restaurant, aplicatia este asa de facut, ca orice utilizator sa poate sa foloseasca fara greutati. Si in lumea reala putem sa intalnim cu situatii unde avem de voie sa folosim aplicatiile de genu aceasta. Dupa tema aceasta am invatat la folosesc composite design pattern, observer design pattern si serialization. Deci, tema aceasta a fost uzuala si beneficiala si pentru mine.

7. Bibliografie

http://www.tutorialspoint.com/java/java_serialization.htm

https://www.javaworld.com/article/2077258/observer-and-observable.html

8. Diagrama UML: