

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**CATEDRA CALCULATOARE**

Tehnici de programare

Documentație

**FOOD DELIVERY MANAGEMENT SYSTEM**

Bartalus Izabella

Grupa 30221

**Cuprins**

1. Obiectivul temei

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

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

4. Implementare

5. Rezultate

6. Concluzii

7. Bibliografie

1. Obiectivul temei

Tema are la baza programarea pe obiectele si paradigmele acesteia . Programarea orientată pe obiecte ( POO , uneori și Programarea orientată obiect , uneori denumită ca și în limba engleză , Object Oriented Programming ( OOP ) ) este o [paradigmă de programare](https://ro.wikipedia.org/wiki/Paradigm%C4%83_de_programare" \o "Paradigmă de programare) , axată pe ideea încapsulării , adică grupării datelor și codului care operează asupra lor , într-o singură structură . Un alt concept important asociat programării orientate obiect este polimorfismul , care permite abstractizări ce permit o descriere conceptuală mai simplă a soluției . Principiile de baza ale acesteia , pe care le-am folosit in cod sunt : Abstractizarea  – 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. Procesele, funcțiile sau metodele pot fi de asemenea abstracte, și în acest caz sunt necesare o varietate de tehnici pentru a extinde abstractizarea : 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 . 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. vor extinde clasa Figura, acestea pot redefini metoda arie. 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. Se utilizează ideea: ”Anumite obiecte sunt similare, dar în același timp diferite”. O clasă moștenitoare a uneia sau mai multor clase de bază se numește clasă derivată. Esența moștenirii constă în posibilitatea refolosirii lucrurilor care funcționează.

Principalul obiectiv al acestei teme este implementarea unei aplicatii care simuleaza un sistem de livrari de mancare . Aceasta aplicatie are o parte care administreaza sistemul ( adauga produse in meniu , le editeaza sau le sterge ) , o parte care se ocupa cu comenzile ( angajatul ) si partea de client care pot face comenzi .

Ca obiective secundare avem :

* Utilizarea unui “ design patterns “ din Java ( “ Composite “ si “ Observer “ ) ;
* Utilizarea unor colectii predefinite in implementarea sistemului de livrari ( “ HashMap “ ) ;
* Serializarea si deserializarea datelor .

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

* Analiza problemei

Principalul scop este de a dezvolta un program ce implementeaza toate cerintele cerute , si , in acelasi timp , foloseste paradigmenele Programarii Orientate pe Obiect . Pentru a face asta , avem nevoie sa gasim un mod sa ne stocam informatiile , si pentru asta , alegem “ HashSet “ pentru a retine meniului sistemului , o lista cu toate comenzile effectuate , si un “ HashMap “ , care are ca si cheie comanda , si ca valoare , o lista de “ MenuItems “ . Oricum , adevarata provocare este sa implementam un anumit “ desing pattern “ , si sa serializem sis a deserializam obiectele noastre , pentru a nu pierde datele cand aplicatia este inchisa .

* Modelare

1. **Composite Design Pattern** este un “ design pattern “ de partitionare si descrie un grup de obiecare ce e tratat ca o singura instanta a aceluias tip de obiect . Intentia sa este sa “ compuna “ obiectele in 3 structuri , ce reprezinta ierarhii partiale . Acest lucru ti permite sa ai o structura de tip “ arbore “ , si sa faci fiecare nod din structura arborelui sa efectuezre un anumit task .

Composite Design Patern are 4 participanti :

1. Componenta : Componenta declara interfata pentru obiectele din compozitie si pentru accesul si controlul componentelor-copii . Implementeaza , de asemenea , in mod implicit , si un comportament pentru interfata comuna tuturor claselor , dupa caz .
2. Frunzaele : Frunzele definesc comportamentul pentru obiectele primitive din compozitite .
3. Compozitia : Compozitia stocheaza componentele-copii si implementeaza operatii legate de copii , in interfata de componente .
4. Clientul : Clientul manipuleaza compozitia obiectului prin interfata componentelor .
5. **Observer Pattern** este utilizat cand avem relatii de tip “ one – to – many “ intre obiecte , astfel incat daca un obiect este modificat , obiectele sale dependente sunt informate automat . “ Observer Pattern “ se incardreaza in categoria modelului comportamental .

“Observer Pattern “ utilizeaza 3 clase :” Subject “ , ” Observer “ si “ Client “ . “ Subject “ este un obiect cu metode ce ataseaza si detaseaza observatorii de obiectele “ client “ .

Pentru a ne administra datele in mod eficient , utilizam :

1. **HashSet :** “ HashSet “ implementeaza intergata “ Set “ , sprijinita de un “ hash table “ , care este , de fapt , o instant “ HashMap “ . Nu ofera nicio garantie cu privire la ordinea de interatie a setului , ceea ce inseamna ca , clasa nu garanteaza ordinea constanta a elementelor de-a lungul timpului . Aceasta clasa permite elemenul “ null “ . Clasa poate oferi , de asemenea , performanta timpului constant , pentru operatii de baza , ca “ add ” , “ remove “ , “ contains “ and “ size “ , presupunand ca functia “ hash “ disperseaza elementele corespunzator .

2. **HashMap** : “ HashMap “ asigura implementarile de baza ale intefetei “ Map “ . Stocheaza informatiile in perechi ( Key , Value ) . Pentru a accesa o valoare , trebuie sa ii cunoastem cheia . “ HashMap “ este cunoscuta ca “ HashMap “ , deoarece utilizeaza o tehnica numita “ Hashing “ . “ Hashing “ este o tehnica de convertire a unui String mare , intr-un string mic , ce reprezinta de fapt acelasi String . O valoare mai mica ajuta in indexarea si cautarea mai rapida . “ HashSet “ utilizeaza intern “ HashMap “ . Intern , foloseste un “ link list “ pentru a stoca perechi “ key - value “ .

Pentru a ne pastra datele dupa ce aplicatia se inchide , folosim “ serializarea “ :

Java ofera un mecanism , numit “ object serialization “ , unde un obiect poate fi reprezentat ca o secventa de biti , ce include informatiile obiectului , precum si informatii despre tipul obiectului si tipurile de date stocate in obiect .

Dupa ce un obiect serializat a fost scris intr-un fisier , el poate fi citit din fisier si deserializat , ceea ce reprezinta , de fapt , tipul informatiei , si bitii ce reprezinta obiectul si informatiile lui pot fi utilizate pentru a crea obiectul in memorie .

* Scenarii

Pentru ca programul sa ruleze corect , atunci cand un utilizator ( fie el administrator , client sau angajat ) incearca sa se conecteze la sistemul de livrari , acesta trebuie sa introduca un username valid si o parola valida . In cazul in care un client nu are cont , acesta se poate folosi de butonul ‘ REGISTER ‘ care il va duce sper un nou frame unde acesta isi va putea crea un cont nou.

* Cazuri de utilizare

Atunci cand se apasa butonu ‘Run’, se deschide interfata grafica , care este usor de utilizat. Apar 4 butoane: Administrator , Client , Employee . Utilizatorul va apasa butonul aferent operatiei pe care doreste sa o realizeze si va deschide unul dintre cele 4 frame – uri.

In cadrul frame – ului ‘ Administrator ‘ utilizatorul poate adauga un produs nou , sterge un produs existent , edita detaliile unui produs , poate vedea lista cu toate produsele si poate genera diferite tipuri de rapoarte .

In cadrul frame – ului ‘ Clients ‘ utilizatorul poate vizualiza tabelul de produse , poate cauta produsele dupa anumite criterii sau poate creea o comanda .

In cadrul frame – ului ‘ Employee ‘ utilizatorul va fi informat de fiecare data cand un client plaseaza o comanda pentru a fi preparata si livrata .

Frame – ului ‘ Register ‘ va fi folosit doar de catre clientii care inca nu au un cont in cadrul sistemului si doresc sa isi faca unul .

1. Proiectarea ( decizii de proiectare , diagrama UML , structuri de date , proiectare clase , interfete , relatii , packages , algoritmi , interfata utilizator )

Pentru a respecta paradigmele POO , programul a fost impartit in mai multe pachete cu nume sugestive care contin diferite clase :

* Pachetul ‘ businessLayer ’ care contine clasele : BaseProduct , CompositeProduct , DeliveryService, MenuItem, Order si interfata IDeliveryService ;
* Pachetul ‘ dataLayer ‘ care contine clasele : DeliverySerializator , FileWriter ;
* Pachetul ‘ presentation ‘ care contine clasele : AdministratorView , ClientView , EmployeeView , RegisterFrame , View ;
* Pachetul ‘ start ‘ care contine clasa : Start .

Interfata grafica cuprinde urmatoarele elemente:

* Frame *=* “rama” in care se adauga toate elementele de care avem nevoie pentru buna functionare a programului . E practice o fereastra care , dupa utilizare , apasand butonul ‘X’ din dreapta sus se poate inchide ( Exit\_on\_close ) ;
* Butoane – avem 3 butoane in frame – ul principal fiecare cu alta functionalitate :
* CLIENTS – deschide frame - ul pentru client care contine la randul lui alte butoane cu operatiile care se pot realiza de catre clientii sistemului de livrari ;
* ADMINISTRATOR – deschide frame – ul pentru administrator care contine la randul lui alte butoane cu operatiile care se pot realiza de catre administratorul sistemului de livrari ;
* EMPLOYEE – deschide frame – ul pentru angajati care contine la randul lui alte butoane cu operatii care se pot realiza de catre angajatii sistemului de livrari ;
* TextField = spatii dreptunghiulare in care se pot introduce date de la tastatura , dar pe langa asta pot fi folosite si pentru a afisa rezultatul fara a se putea introduce date de la tastatura .
* Label = o etichetă , care poate fi titlul , informații , indicații etc . , care pot ajuta utilizatorul să folosească progamul. In cazul nostru e folosit doar pentru afisarea informatiilor aflate deasupra casetelor de text .

1. Implementare

* Clasa “DeliverySerialization “ are 2 metode :
* Serialization ( ) : ia datele din parametrul “restaurant “ si le scrie in fileStreamObject si le pune intr-un fisier
* Deserialization ( ) : creeaza o noua instanta a restaurantului folosind informatiile salvate in fisier in urma serializarii .
* Clasele : “ BaseProduct “ & “ CompositeProduct “ au doar metodele getters si setters si metoda pentru calcularea pretului definite in clasa abstracta “ MenuItem “
* Clasa “ Order “ : contine getters si setters pentru fiecare atribut , si , in plus , avem si clasa hashCode ,care calculeaza codul functiei Hash al fiecarei comenzi .
* Clasa “ DeliveryService “ are mai multe metode :
* Metoda “ createNewMenuItem() : Metoda care adauga la meniul restaurantului un alt produs(un alt obiect de tip "menuItem " )
* Metoda “ findBaseProduct ( ) : Metoda care gaseste in meniu un produs "menuItem",dat ca parametru .Aceasta metoda cauta produsul "menuItem",si printre componentele unui produs compus Daca il gaseste, returneaza produsul,iar in caz contrar,returneaza null.
* Metoda “ findMenuItem ( ) : Gaseste un produs in meniul restaurantului .
* Metoda “ findItemInOrder ( ) : Metoda care cauta in lista de comenzi un produs de tip "MenuItem",dat ca parametru.
* Metoda “deleteItemMenu “ : Sterge din meniu produsul "menuItem",dat ca parametru
* Metoda “ editMenuItem “ : Metoda care inlocuieste un produs al meniului cu unul nou
* Metoda “ findOrder “ : Metoda care verifica daca exista o comanda la o anumita masa
* Metoda “noItems “ : Metoda care verifica daca avem o comanda goala(fara produse)
* Metoda “createNewOrder “ : Metoda care creeaza o noua comanda
* Metoda “ computePrice “ : Metoda care returneaza suma totala a unei comenzi
* Metoda “generateBill “ : Metoda care genereaza factura unei comenzi

1. Rezultate

Rezultatele in urma rularii acestui program sunt stocate in fisierul “delivery.ser” , iar ,datorita deserializarii ,aceste operatii sunt vizibile si dupa ce programul este inchis,putandu-se vedea in ferestrele administratorului , si a osparatului , tabele cu meniul restaurantului si cu comenzile efectuate .

Se poate observa in folderul incarca, rezultatele obtinute in urma generarii unor bonuri , sub forma unor fisiere text .

1. Concluzii

In concluzie , aceasta tema ne-a initrodus in lumea “ Design-Pattern-ului “ din Java ( Composite si Observer ,in mod special ) ; si , de asemenea , ne introduce conceptual de serializare .

1. Bibliografie
2. <http://www.tutorialspoint.com/java/java_serialization.htm>
3. <https://www.baeldung.com/java-serialization>
4. <https://www.geeksforgeeks.org/serialization-in-java/>
5. <https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html> <http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html>
6. <http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html>
7. <http://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.html>
8. <http://stackoverflow.com/questions/11415160/how-to-enable-the-java-keywordassert-in-eclipse-program-wise>
9. <https://intellij-support.jetbrains.com/hc/en-us/community/posts/207014815-How-toenable-assert>
10. <https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag>