# Programare orientată pe obiecte

## Tema - Supermarket

**Deadline: 13.01.2017**
**Ora 23:55**

**Responsabili temă:** *Mihai Nan*, *Cristina Rînciog-Savin*

**Profesor titular:** *Carmen Odubăşteanu*

**Facultatea de Automatică şi Calculatoare**
**Universitatea Politehnica din Bucureşti**
**Anul universitar 2016 - 2017**
**Seria CC**

# 1 Obiective

În urma realizării acestei teme, studentul va fi capabil:

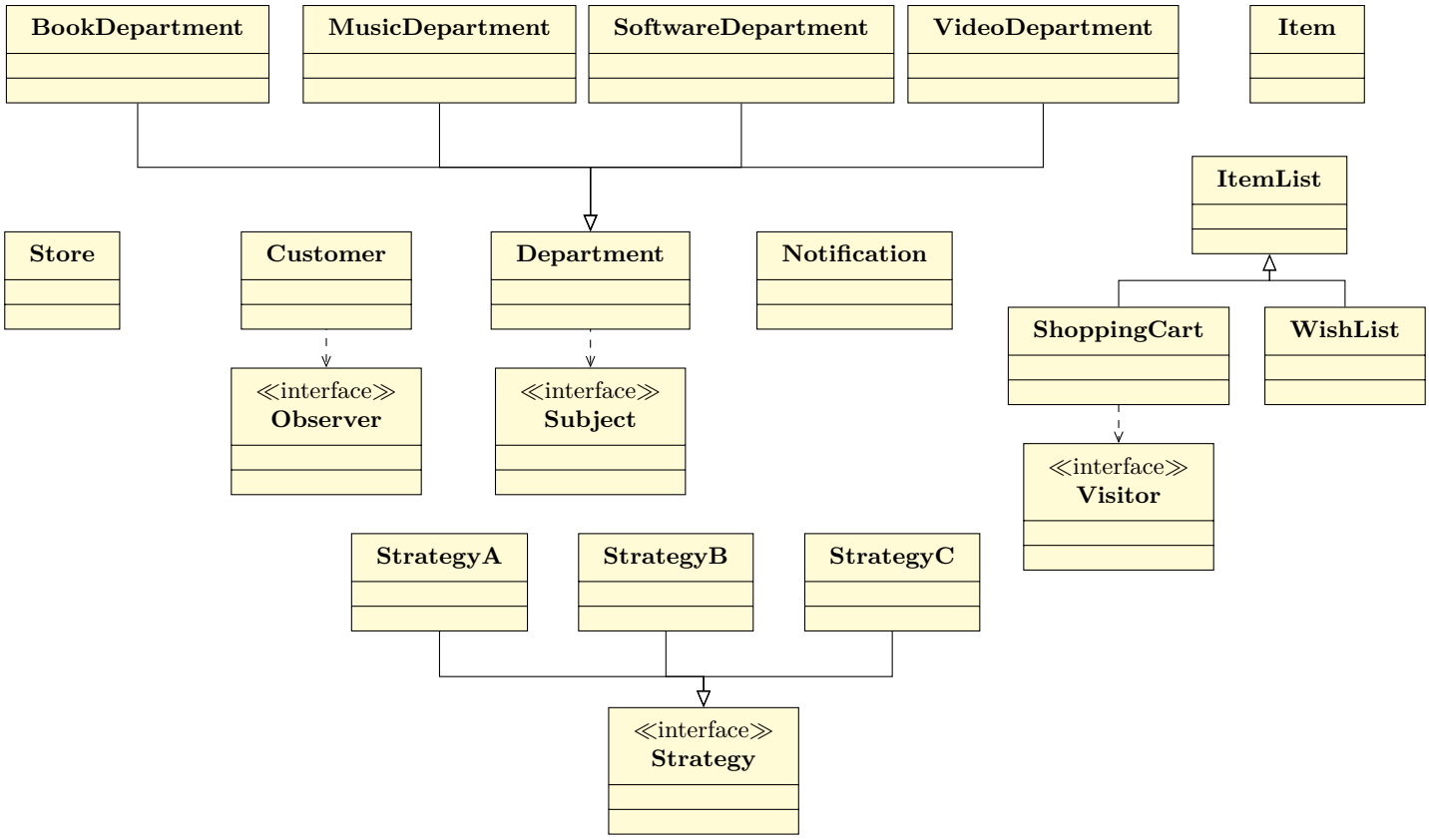
- să aplice corect principiile programării orientate pe obiecte studiate în cadrul cursului;
- să construiască o ierarhie de clase, pornind de la un scenariu propus;
- să utilizeze un design orientat-obiect;
- să implementeze o serie de structuri de date folosind programarea orientată pe obiecte;
- să trateze excepțiile ce pot interveni în timpul rulării unei aplicații;
- să transpună o problemă din viața reală într-o aplicație ce se poate realiza folosind noțiunile dobândite în cadrul cursului.

# 2 Descriere

Scopul acestei teme este de a implementa o simulare a unui magazin electronic, aplicând noțiunile studiate în cadrul acestui curs.

# 3 Arhitectura aplicației

## 3.1 Diagrama UML de bază




⚠ IMPORTANT !

⚠ La această diagramă de bază se adaugă toate atributele şi metodele descrise mai jos, precum şi cele pe care le veţi adauga voi!

## 3.2 Descrierea claselor

### 3.2.1 Store

Aceasta este clasa care modelează magazinul online. Un magazin este caracterizat prin nume, departamente și clienți. Pe lângă aceste date, magazinul va permite efectuarea următoarelor operații:

- ***enter(Customer)*** - clientul a intrat în magazin;
- ***exit(Customer)*** - clientul a ieșit din magazin;
- ***getShoppingCart(Double)*** - întoarce un obiect de tip **ShoppingCart**, având asociat bugetul indicat, ce urmează a fi folosit de către client;
- ***getCustomers()*** - întoarce toți clienții care sunt în magazin la acel moment;
- ***getDepartments()*** - întoarce departamentele magazinului respectiv;
- ***addDepartment(Department)*** - adaugă un departament;
- ***getDepartment(Integer)*** - returnează departamentul ce are ID-ul indicat.

### 3.2.2 Department

Un departament este reprezentat în aplicația noastră printr-o clasă abstractă care conține ca și atribute următoarele: numele departamentului, item-urile disponibile spre vânzare din departamentul respectiv, clienții care au cumpărat cel puțin un produs, clienții care își doresc cel puțin un produs aparținând departamentului (observers), ID-ul departamentului care trebuie să fie un identificator unic. Operațiile standard pentru un departament sunt următoarele:

- ***enter(Customer)*** - clientul a cumpărat cel puțin un produs aparținând departamentului;
- ***exit(Customer)*** - clientul s-a hotărât că nu mai dorește să cumpere vreodată vreun produs ce aparține departamentului;
- ***getCustomers()*** - întoarce toți clienții care au cumpărat cel puțin un produs al departamentului;
- ***getId()*** - întoarce ID-ul departamentului;
- ***addItem(Item)*** - adaugă un item în produsele departamentului;
- ***getItems()*** - întoarce produsele departamentului;
- ***addObserver(Customer)*** - metodă apelată pentru a înregistra un client ca un observator (această metodă trebuie apelată atunci când clientul își dorește cel puțin un produs din departamentul respectiv);
- ***removeObserver(Customer)*** - metodă apelată pentru a șterge un client ca observator (această metodă trebuie apelată când clientul nu mai are în **WishList** niciun produs aparținând departamentului);
- ***notifyAllObservers(Notification)*** - trimite notificarea către fiecare client (această metoda trebui apelată dacă este adăugat sau eliminat un produs din departament);
- ***accept(ShoppingCart)*** - se va detalia ulterior efectul acestei metode pentru fiecare departament în parte.

### 3.2.3 Item

Un **Item** este caracterizat printr-un nume, un **ID** unic și un preț. De asemenea, fiecare **Item** aparține unui **Department**. În momentul în care implementați această clasă ar trebui să țineți cont de principiul încapsulării datelor.

### 3.2.4 Customer

Atributele clasei **Customer** sunt următoarele: numele, un obiect de tip **ShoppingCart**, un obiect de tip **WishList** și o colecție de notificări.

### 3.2.5 ItemList

Clasa abstractă **ItemList** modelează o listă liniară dublu înlănțuită și sortată, având elemente de tip **Item**. Această clasă conține o clasă internă statică **Node**, folosită pentru reprezentarea unui nod al listei și o clasă internă **ItemIterator**, care implementează interfața **ListIterator**. Pentru a realiza ușor inserarea elementelor în listă astfel încât aceasta să rămână sortată, clasa va avea ca membru un **Comparator**. Pe lângă toate acestea, această clasă va implementa o parte din metodele din blocul de cod de mai jos. Metodele care nu se pot implementa în această clasă vor rămâne abstracte.

**Cod Java**

```
public boolean add(Item element);
public boolean addAll(Collection<? extends Item> c);
public Item getItem(int index);
public Node<Item> getNode(int index);
public int indexOf(Item item);
public int indexOf(Node<Item> node);
public boolean contains(Node<Item> node);
public boolean contains(Item item);
public Item remove(int index);
public boolean remove(Item item);
public boolean removeAll(Collection<? extends Item> collection);
public boolean isEmpty();
public ListIterator<Item> listIterator(int index);
public ListIterator<Item> listIterator();
public Double getTotalPrice();
```

### 3.2.6 BookDepartment

Metoda ***accept*** scade 10% de la prețul fiecărui produs din **ShoppingCart** ce aparține acestui departament.

### 3.2.7 MusicDepartment

Metoda ***accept*** adaugă la buget 10% din valoarea totalului produselor ce aparțin departamentului si sunt si în coșul de cumpărături.

### 3.2.8 SoftwareDepartment

Metoda ***accept*** modifică prețul fiecărui produs din **ShoppingCart** ce aparține acestui departament aplicând o reducere de 20% dacă bugetul nu mai permite cumpărarea unui alt produs din acest departament. Cu alte cuvinte, dacă bugetul rămas disponibil pentru acel coș de cumpărături este mai mic decât prețul celui mai ieftin produs ce aparține departamentului software.

### 3.2.9 VideoDepartment

Metoda ***accept*** modifică prețul fiecărui produs din **ShoppingCart**, ce aparține acestui departament, dacă valoarea totalului produselor ce aparțin departamentului este mai mare decât cel mai scump produs din acest departament, aplicând o reducere cu 15%. De asemenea, se va adăuga la bugetul coșului de cumpărături 5% din valoarea totalului produselor ce aparțin departamentului, indiferent dacă este sau nu îndeplinită condiția anterioară.

⚠ IMPORTANT !

⚠ Toate aceste tipuri de departamente vor extinde clasa abstractă **Departament**. Implementările metodei ***accept***, în toate clasele de mai sus, solicită vizitarea instanței curente de către vizitator.

**Observație**

Prețurile produselor se vor modifica doar pentru produsele din **ShoppingCart**. Astfel, trebuie să vă asigurați că prețurile produselor din magazin sau din alte coșuri de cumpărături nu se vor modifica.

Nu uitați să modificați corespunzător bugetul coșului de cumpărături după apelul unui astfel de metode.

#### 3.2.10 ShoppingCart

Clasa **ShoppingCart** va moșteni clasa **ItemList** și va implementa interfața **Visitor**. De asemenea, această clasă va implementa toate metodele rămase abstracte în clasa **ItemList**. Pentru fiecare **ShopppingCart** se va aloca, la instanțiere, un buget și se pot adăuga elemente în listă atâta timp cât bugetul nu a fost depășit. În cadrul **ShoppingCart**-ului elementele vor fi sortate crescător după preț, iar dacă două produse au același preț, se vor sorta alfabetic după numele produsului.

#### 3.2.11 WishList

Această clasă va moșteni clasa ***ItemList*** și va conține produsele pe care un client le dorește, dar nu le poate achiziționa momentan.

**Observație**

În cazul bonusului, clasa va conține ca și atribut un obiect de tip **Strategy** ce se va folosi în cadrul operației ***executeStrategy*** care va selecta un produs din listă, conform strategiei utilizatorului. Această operație va returna produsul selectat în urma aplicării strategiei și îl va elimina din listă, urmând ca acesta să fie adăugat în **ShoppingCart**.

#### 3.2.12 Notification

Fiecare client deține o colecție de notificări pe care le primește atunci când este modificat prețul unui anumit produs dintr-un departament care s-ar putea să-l intereseze: are cel puțin un produs aparținând departamentului în lista sa de preferințe; când este adăugat un nou produs în departamentul respectiv sau când este eliminat un produs. Clasa **Notification** va conține ca și atribute următoarele: data la care a fost trimisă notificarea, tipul notificării (va conține o enumerare, **NotificationType**, cu următoarele constante **ADD**, **REMOVE**, **MODIFY**), **ID**-ul departamentului și **ID**-ul produsului. Atunci când un client primește o notificare el trebuie să țină cont de ea. Astfel, dacă prețul produsului a fost modificat, se verifică în coșul de cumpărături și în lista de preferințe dacă există produsul respectiv și se modifică prețul. Aveți grijă să modificați și bugetul, în cazul coșului de cumpărături, atunci cand modificați prețul unui produs. Daca un produs a fost șters din magazin, el se va șterge și din listele clientului, dacă există.

## 4 Șabloane de proiectare

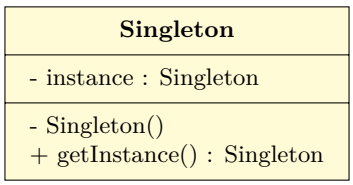
Un șablon de proiectare descrie o problemă care se întâlnește în mod repetat în proiectarea programelor și soluția generală pentru problema respectivă, astfel încât să poată fi utilizată oricând, dar nu în același mod de fiecare dată. Soluția este exprimată folosind clase și obiecte. Atât descrierea problemei cât și a soluției sunt abstracte astfel încât să poată fi folosite în multe situații diferite.

Scopul șabloanelor de proiectare este de a asista rezolvarea unor probleme similare cu unele deja întâlnite și rezolvate anterior. Ele ajută la crearea unui limbaj comun pentru comunicarea experienței despre aceste probleme și soluțiile lor.

## 4.1 Identificarea sabloanelor de proiectare

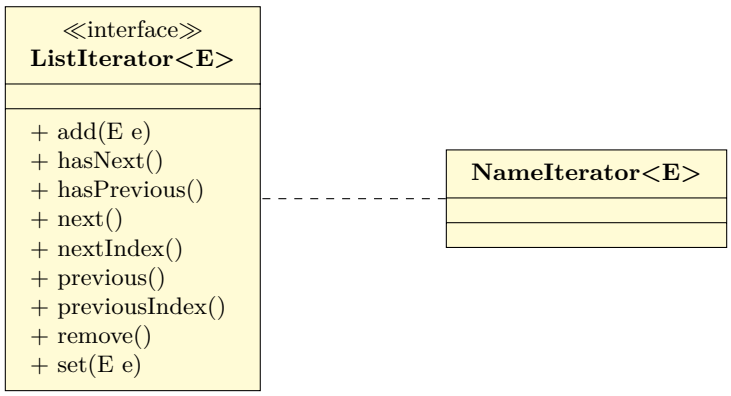
### 4.1.1 Singleton pattern

Pentru a ne asigura că fiecare client are acces la informațiile aceluiași magazin, avem nevoie să menținem o referință către un obiect de tip **Store**, în mai multe clase. Astfel, vom utiliza șablonul **Singleton** pentru a restricționa numărul de instanțieri ale clasei Store la un singur obiect. Pentru a nu consuma inutil resursele sistemului, în implementarea voastră, veți folosi instanțierea întârziată.



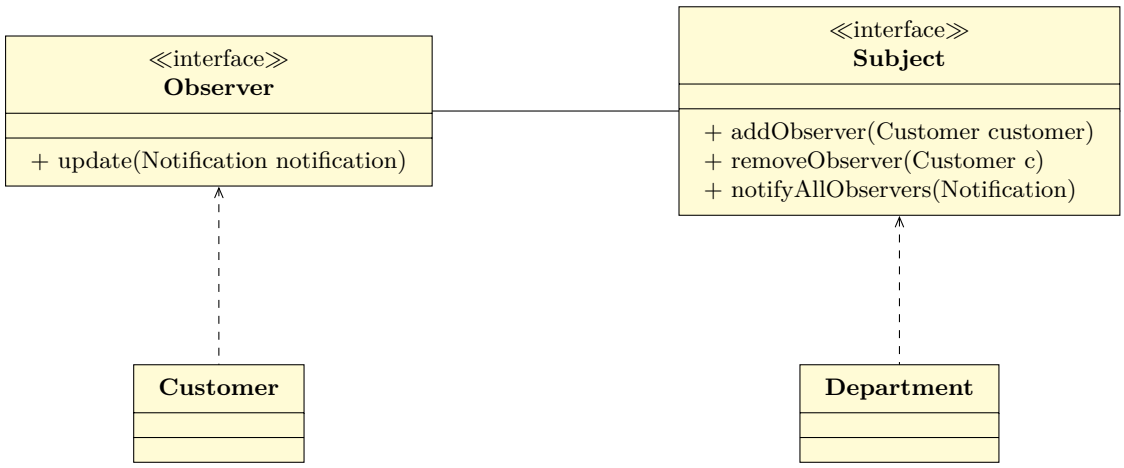

### 4.1.2 Iterator pattern

Având în vedere că în cadrul aplicației folosim liste ordonate, avem nevoie de un mecanism prin care să traversăm aceste structuri de date. De aceea, veți folosi în implementare pattern-ul **Iterator** care asigură o cale de accesare secvențială a elementelor unui obiect agregat, fără a expune reprezentarea lui de bază.



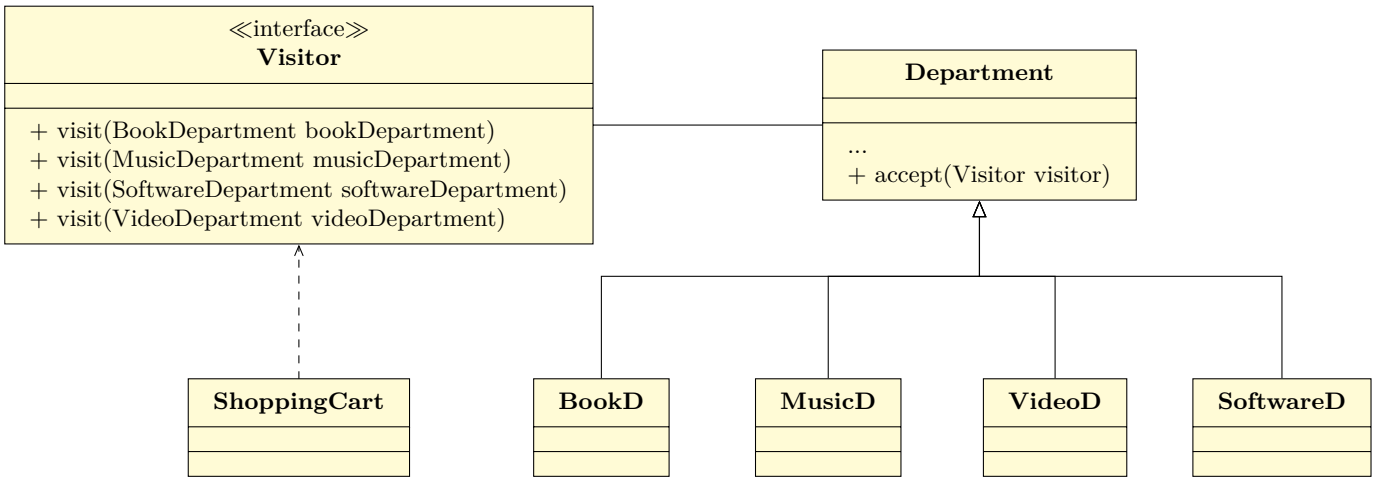

### 4.1.3 Observer pattern

Design pattern-ul **Observer** definește o relație de dependență unul la mai mulți între obiecte astfel încât atunci când un obiect își schimbă starea toți dependenții lui sunt notificați și actualizați automat. Acest pattern implică existența unui obiect denumit subiect care are asociată o listă de obiecte dependente, numite observatori, pe care le apelează automat de fiecare dată când se întâmplă o acțiune. În cadrul acestei aplicații vom folosi pattern-ul **Observer** pentru a notifica clienții de fiecare dată când se modifică prețul unui produs, se adăugă un produs sau este eliminat un produs dintr-un departament.




#### 4.1.4 Visitor pattern

În cazul aplicării pattern-ului **Visitor**, avem următorul scenariu: clientul folosește o colecție de obiecte cu tipuri diferite și dorește să aplice pe aceste obiecte operații specifice, în funcție de tipul lor. Cu alte cuvinte, clientul uzitează obiecte **Visitor** create pentru fiecare operație necesară. Acesta parcurge colecția și, în loc sa aplice direct, pe fiecare obiect, operația, îi oferă acestuia un obiect de tip **Visitor** și apelează metoda de *vizitare*. Astfel, pe obiectul **Visitor** se apelează metoda ***visit*** corespunzătoare obiectului, realizându-se, în implementarea acesteia, operația dorită. În aplicația noastră, avem clasa **ShoppingCart** care implementează o interfață **Visitor**, iar această clasă reprezintă o colecție de produse aparținând unor departamente specifice.




# 5 Cerințe

## 5.1 Cerința 1 - Implementarea şi testarea aplicației (fără interfaţă grafică) - 120 puncte

Pentru a putea realiza o testare a aplicației, trebuie sa implementați o clasă **Test** ce conține metoda ***main*** care parsează o serie de fișiere de intrare și realizează o serie de operații indicate. Rezultatul acestor operații va fi depus într-un fişier de ieşire, ***output.txt*** (aveţi un exemplul în arhiva temei).

## 5.2 Formatul fișierelor

În continuare, se va detalia formatul fiecărui fișier de intrare ce se va oferi pentru verificarea funcționalității aplicației.

### 5.2.1 store.txt

Pe prima linie a fișierului se găsește numele magazinului, după care sunt specificate departamentele. Pentru fiecare departament sunt specificate: numele departamentului și ID-ul acestuia, numărul de produse și specificarea fiecărui produs(`Nume;ID;preț`).

**Exemplu**

```
Carrefour
BookDepartment;1
11
Amintiri din copilarie;100;5.0
Solenoid;101;57.8
Ochii jupanitei;102;28.0
Legendele Olimpului;103;26.80
Enigma Otiliei;104;10.53
Poezii alese - Mihai Eminescu;105;3.75
Recreatia mare;106;9.75
Legende istorice - Dimitrie Bolintineanu;106;5.53
Biblia pierduta;107;26.11
Cincizeci de umbre intunecate;108;28.02
Fericirea incepe azi;109;23.17
MusicDepartment;2
5
Andrea Bocelli-Passione;200;37.99
George Enescu - Octet in C major;201;22.92
Andre Rieu-Live In Maastricht;202;61.74
Madrigal - Documente Ale Culturii Muzicale;203;22.92
Antonio Vivaldi - Four Seasons;204;29.88
SoftwareDepartment;3
3
Microsoft Office 365 Personal;300;289.99
FacturarePRO Standard;301;150.00
CorelDRAW Graphics Suite;302;1659.00
VideoDepartment;4
4
Me before you;400;44.96
Jungle book;401;53.99
The martian;402;50.00
Hugo;403;39.90
```

### 5.2.2 customers.txt

Pe prima linie a fișierului se găsește $N$, numărul total de clienți, iar pe următoarele $N$ linii sunt specificați clienții (`nume;buget;strategie`).

**Exemplu**

```
5
Alina;750;A
Maria;950;B
Andrada;1650;C
Andrei;1500;A
Mihai;500;A
```

**Observație**

Strategia va fi luată în calcul doar în cazul implementării bonusului.

### 5.2.3 events.txt

Pe prima linie a fișierului se găsește numărul de evenimente, iar pe următoarele linii sunt descrise evenimentele.

Evenimentele posibile sunt următoarele:

- `addItem;ItemID;ShoppingCart/WishList;CustomerName` - adaugă produsul indicat în coșul de cumpărături sau la lista de dorințe pentru clientul specificat;
- `delItem;ItemID;ShoppingCart/WishList;CustomerName` - operația inversă pentru `addItem`;
- `addProduct;DepID;ItemID;Price;Name` - adaugă produsul în magazin;
- `modifyProduct;DepID;ItemID;Price` - modifică prețul unui produs existent în magazin;
- `delProduct;ItemID` - elimină produsul din magazin (dacă se afla în lista vreunui client, este eliminat);
- `getItem;CustomerName` - întoarce produsul ales conform strategiei definite în cadrul fișierului ***customers.txt*** pentru clientul specificat;
- `getItems;ShoppingCart/WishList;CustomerName` - întoarce produsele din lista clientului;
- `getTotal;ShoppingCart/WishList;CustomerName` - întoarce totalul produselor din coșul de cumpărături sau lista de dorințe pentru clientul indicat;
- `accept;DepID;CustomerName` - apelează metoda ***accept*** pentru departamentul și clientul indicați;
- `getObservers;DepID` - întoarce observatorii departamentului;
- `getNotifications;CustomerName` - întoarce notificările clientului.

**Exemplu**

```
24
addItem;100;ShoppingCart;Andrei
addItem;104;WishList;Mihai
addProduct;1;110;4.25;Mara - Ioan Slavici
addProduct;1;111;4.45;Proza si teatru - V. Alecsandri
addProduct;4;404;9.75;Big fish
delProduct;404
addItem;403;ShoppingCart;Maria
addItem;200;WishList;Mihai
addItem;403;WishList;Mihai
getItems;WishList;Mihai
getItem;Mihai
getItems;WishList;Mihai
getItems;ShoppingCart;Mihai
addItem;402;ShoppingCart;Mihai
getTotal;ShoppingCart;Mihai
accept;4;Mihai
getTotal;ShoppingCart;Mihai
addItem;201;ShoppingCart;Mihai
getTotal;ShoppingCart;Mihai
getItems;WishList;Mihai
getObservers;1
getObservers;2
getObservers;3
getObservers;4
```

⚠ IMPORTANT !

⚠ Se garantează că **;** apare doar ca delimitator între câmpuri. Puteți găsi rezultatele acestor evenimente în fișierul din arhiva temei.

**Observatie**

Atunci când se modifică prețul unui produs, se șterge sau adaugă un produs, se va trimite o notificare către fiecare observator al departamentului din care face parte produsul.

### 5.3 Cerința 2 - Interfața grafică - 80 puncte

Va trebuie să realizați o interfață grafică folosind componenta **Swing**. Din această interfață vor putea executa comenzi atât magazinul, cât și clienții.

Prima pagină a interfeței grafice va cuprinde un buton de încărcare a fișierelor ***store.txt*** și ***customers.txt***. Pe baza acestor fișiere se va crea instanța magazinului și ierarhia de obiecte adiacente.

Pentru administrare magazinului va trebui să implementați următoarele pagini:

- pagina de afișare și administrare a produselor din magazin cu următoarele operații:
  - afișarea produselor existente cu posibilitatea de ordonare: alfabetică după denumirea produsului, crescătoare / descrescătoare după prețul produsului;
  - adăugarea unui nou produs (dacă produsul există deja, utilizatorul va fi atenționat);
  - ștergerea sau editarea unui produs, iar atunci când un produs este șters din magazin se va elimina și din orice listă a utilizatorilor;

Pentru paginile destinate unui client, va trebui întâi selectat clientul din lista de clienți a magazinului, iar pentru clientul selectat vor trebui implementate următoarele pagini:

- pagina de vizualizare și editare **ShoppingCart**;
  - Această pagină va afișa produsele existente în coșul de cumpărături al clientului și toate informațiile adiacente acestora (ID produs, nume, preț, ID departament).
  - Pagina va permite clientului adăugarea / ștergerea unui produs din lista de cumpărături.
  - Pagina trebuie să ofere, de asemenea, posibilitate de comandare a produselor prin apăsarea unui buton de trimitere a comenzii. Apăsarea acestuia va avea ca efect creare pentru clientul curent a unui nou **ShoppingCart** gol, având ca buget suma rămasă disponibilă în urma plasării comenzii.
  - Pagina va conține informații cu privire la **ShoppingCart**: totalul produselor din coș, bugetul maxim al coșului. Așezarea tuturor componentelor menționate în pagină ține de implementarea voastră, având libertate maximă.
- pagina de administrare și vizualizare **WishList**;
  - Această pagină va afișa produsele existente în lista de dorințe a clientului și toate informațiile adiacente acestora (ID produs, nume, preț, ID departament).
  - Pagina va permite clientului adăugarea / ștergerea unui produs din lista de dorințe.

**Observație**

Sunteți liberi să adaugați orice funcționalități suplimentate doriți sau le considerați utile din punct de vedere al interfeței grafice.

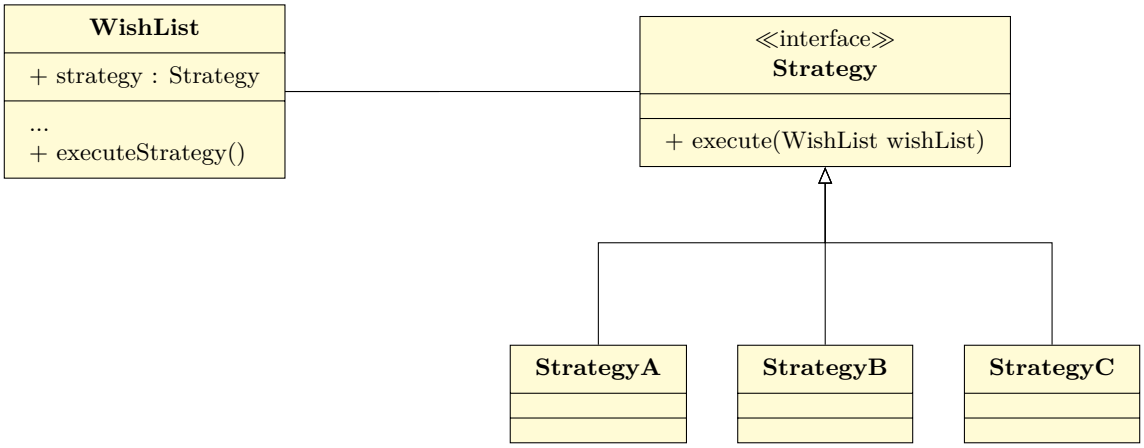
Se vor acorda **10 puncte** bonus pentru o interfață grafică intuitivă și complexă, frumos realizată, care pune la dispoziție toate operațiile implementate de arhitectură.

## 6 Bonusuri

### 6.1 Strategy pattern - 15 puncte

Șablonul arhitectural **Strategy** a fost conceput pentru a putea selecta algoritmul optim pentru necesitatea fiecărui client, în funcție de specificațiile acestuia. În cazul nostru, avem o listă de preferințe și dorim să selectăm

un produs din lista respectivă. Pentru aceasta, putem folosi strategii ca: alegem produsul cel mai ieftin / cel mai scump, selectăm produsul pentru care am primit ultima notificare etc. Astfel, acest șablon îi oferă clientului posibilitatea de a alege singur ce strategie folosește pentru a selecta un produs din lista respectivă.




- **StrategyA** - selectează produsul cel mai ieftin;
- **StrategyB** - selectează primul produs, având denumirile ordonate alfabetic;
- **StrategyC** - selectează produsul adăugat cel mai recent în **WishList**.

## 6.2 Builder pattern - 5 puncte

Acest pattern poate fi folosit în implementarea restaurantelor de tip fast food care furnizează meniul pentru copii. Un meniu pentru copii constă de obicei într-un fel principal, unul secundar, o băutură și o jucărie. Pot exista variații în ceea ce privește conținutul mediului, dar procesul de creare este același. Fie că la felul principal se alege un hamburger sau un cheesburger procesul va fi același. Vânzătorul le va indica celor din spate ce să pună pentru fiecare fel de mâncare, pentru băutură și jucărie. Toate acestea vor fi puse într-o pungă și servite clienților.

Acest șablon dorește separarea construcției de obiecte complexe de reprezentarea lor astfel încât același proces să poată crea diferite reprezentări. **Builder**-ul creează părți ale obiectului complex de fiecare dată când este apelat și reține toate stările intermediare. Când departamentul este terminat, clientul primește rezultatul de la builder. În acest mod, se obține un control mai mare asupra procesului de construcție de noi obiecte. Spre deosebire de alte pattern-uri, din categoria creational, care creau produsele într-un singur pas, pattern-ul **Builder** construiește un produs pas cu pas la comanda coordonatorului. În cadrul acestei aplicații, pattern-ul este folosit pentru a instanția departamentele. Pentru a înțelege cum ar trebui folosit acest pattern, puteți urmări exemplul de mai jos.

⚠ IMPORTANT !

⚠ Trebuie să vă asigurați că veți putea instanția, folosind mecanismul implementat pe baza pattern-ului **Builder**, orice tip de departament.

**Cod Java**

```java
public class User  {
    private final String firstName; // required
    private final String lastName; // required
    private final int age; // optional
    private final String phone; // optional
    private final String address; // optional
    private User(UserBuilder builder) {
        this.firstName = builder.firstName;
        this.lastName = builder.lastName;
        this.age = builder.age;
        this.phone = builder.phone;
        this.address = builder.address;
    }
    public String getFirstName() {
        return firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public int getAge() {
        return age;
    }
    public String getPhone() {
        return phone;
    }
    public String getAddress() {
        return address;
    }
    public String toString() {
        return "User:"+this.firstName+" "+this.lastName+" "+this.age+" "+this.phone+" "+
            this.address;
    }
    public static class UserBuilder {
        private final String firstName;
        private final String lastName;
        private int age;
        private String phone;
        private String address;
        public UserBuilder(String firstName, String lastName) {
            this.firstName = firstName;
            this.lastName = lastName;
        }
        public UserBuilder age(int age) {
            this.age = age;
            return this;
        }
        public UserBuilder phone(String phone) {
            this.phone = phone;
            return this;
        }
        public UserBuilder address(String address) {
            this.address = address;
            return this;
        }
        public User build() {
            return new User(this);
        }
    }
    public static void main(String[] args) {
        User user1 = new User.UserBuilder("Lokesh", "Gupta")
                .age(30)
                .phone("1234567")
                .address("Fake address 1234")
                .build();
        User user2 = new User.UserBuilder("Jack", "Reacher")
                .age(40)
                .phone("5655")
                //no address
                .build();
    }
}
```

### 6.3 Interfața grafică - 20 puncte

În ceea ce privește interfața grafică, vi se propun următoarele bonusuri:

- Pagina de vizualizare și editare **ShoppingCart** va conține și funcționalitatea de sugerare a unui produs existent în **WishList**, conform strategiei clientului. Produsul sugerat va fi adăugat la **ShoppingCart** dacă bugetul o permite și va fi eliminat din pagina de WishList.
- O pagină pentru fiecare departament în care se vor afișa clienții acelui departament, observatorii săi, cel mai cumpărat produs, cel mai dorit produs și cel mai scump produs.
- Pagina în care un client își poate accesa lista de notificări primite.

## 7 Punctaj

| Cerinta | Punctaj |
|---|---|
| Cerința 1 | 120 puncte |
| Cerința 2 | 80 puncte |
| Bonus Design Pattern Strategy | 15 puncte |
| Bonus Design Pattern Builder | 5 puncte |
| Bonus interfață grafică | 20 + 10 puncte |

## 8 Restricții și precizări

**Atentie!**

Soluții de genul folosirii de variabile interne pentru a stoca informații ce țin de tipul obiectului vor fi depunctate.

Tipizați orice colecție folosită în cadrul implementării.

Respectați specificațiile detaliate în enunțul temei și folosiți hinturile menționate.

**Tema este individuală! Toate soluțiile trimise vor fi verificate, folosind o unealtă pentru detectarea plagiatului.**

Tema se va prezenta la ultimul laborator din semestru.

Tema se va încărca pe site-ul de cursuri până la termenul specificat în pagina de titlu. Se va trimite o arhiva ***.zip*** ce va avea un nume de forma **grupa_Nume_Prenume.zip** (ex. ***326CC_Popescu_Andreea.zip*** și care va conține următoarele:

- un folder **SURSE** ce conține doar sursele Java, un **Makefile** care să compileze aceste surse și să permită rularea clasei **Test**, având cel puțin 2 reguli: ***build*** și ***run***) și fișierele de test;
- un folder **PROIECT** ce conține proiectul ***NetBeans*** sau ***Eclipse***;
- un fisier **README** în care veți specifica numele, grupa, gradul de dificultate al temei, timpul alocat rezolvării și veți detalia succint modul de implementare, specificând și alte obervații, daca este cazul. Lipsa acestui fișier duce la o depunctare de ***10 puncte***.