Asociația de proprietari

Trebuie sa scrieti programul de administrare a apartamentelor unei Asociatii de proprietari. Aceasta asociatie doreste sa-si extinda activitatea sa de gestiune a cheltuielilor legate de utilizarea apartamentelor si cu alte activitati cum ar fi si publicitate imobiliara privind proprietatile.

Programul pe care il veti scrie trebuie sa fie proiectat in conformitate cu conceptele programarii orientate pe obiecte si sa utilizeze sabloane de proiectare (design patterns) cunoscute. Vi se pune la dispozitie o parte din codul prin care s-a implementat sablonul DAO pe care trebuie sa-l completati, dar si sa scrieti cod nou corespunzator straturilor *bussiness logic* si *frontend*, conform specifcatiilor care urmeaza.

Este important sa respectati intocmai toate specificatiile care urmeaza, inclusiv denumirile de clase si interfete, iar iesirea produsa trebuie sa fie exact in formatul cerut deoarece programul dv. va putea fi verificat si automat cu un *java checker*.

Atentie! Sarcina pe care o aveti de realizat depinde de numarul atribuit dumneavoastra pentru aceasta tema. Acest numar este atribuit astfel:

- daca prima litera din numele dumneavoastra de familie este cuprinsa intre **A si H atunci aveti numarul 1**;
- daca prima litera din numele dumneavoastra de familie este cuprinsa intre **I si Z atunci aveti numarul 2.**

Programul pe care trebuie sa il realizati cuprinde metode comune celor doua numere, dar si metode specifice. Acolo unde sarcinile difera veti fi atentionati prin semnul grafic atentie (alaturat).
Acolo unde nu este nimic specificat inseamna ca sarcina trebuie rezolvata de ambele numere (de toti sudenti).

Important

Fiecare clasa trebuie sa aiba un antet prin care sa poata fi identificat autorul conform exemplului urmator

```
/**
 * @author Ionela IONESCU
 * @grupa 3132a
 * @nr 2
 */
```

In lipsa acestor comentarii tema nu va fi corectata!

1. Clasa abstracta *Apartament*

Clasa abstractă **Apartament** este utilizată pentru a stoca informații despre apartamentele unui imobil. Printre altele, următoarele date trebuie înregistrate ca variabile de instanță private și trebuie create metodele de acces public necesare (set... și get...)

- Id (număr de proprietate – număr intreg unic, dar nu neapărat consecutiv)
- suprafata locuintei in mp (ex. 64. 5)
- anul de construcție
- Adresă (stradă, număr de casă, scara, etaj, numar apartament)

Selectați tipurile de date adecvate.

Trebuie implementat un constructor care să permită setarea directă a variabilelor de instanță comune tuturor subclaselor. Verificați dacă valorile sunt plauzibile. (de exemplu, anul de construcție nu trebuie să fie în viitor etc.).

În cazul în care condițiile nu sunt îndeplinite, se aruncă o exceptie IllegalArgumentException cu un mesaj de eroare predefinit (`Valori incorecte pentru crearea unui apartament`).

Exceptia IllegalArgumentException deriva din java.lang.Exception si are un constructor cu un singur argument:

`public IllegalArgumentException(String mesaj)`

Acest mesaj poate fi obtinut cu metoda `getMessage()`, ca in exemplul de mai jos:

```
int an=2022;
try {
    if(an %2==0)
        throw new IllegalArgumentException("an par ="+ an);
} catch (IllegalArgumentException e){
    System.out.println(e.getMessage());
}
```

In urma executiei secventei de mai sus se afiseaza
`an par =2022`

Metoda `getVechime()` are rolul de a calcula vârsta in ani a unui **Apartament**.

2. Clasele Locuinta si SediuFirma

Sunt 2 subclase concrete ale clasei abstracte ***Apartament*** si reprezinta apartamente utilizate in scop de locuinta sau ca sedii firme.

Clasa **Locuinta** si clasa **SediuFirma** au variabile de instanță ca cele din exemplul de mai jos.

Locuinta
{Tip=L, id=11, suprafata=64.5, anConstructie=2020, strada='Eroilor', nr=109, scara=B, etaj=3, nrApt=14, nrPersoane=5}

SediuFirma
{Tip=SF, id=190, suprafata=50.0, anConstructie=1990, strada='Zorilor', nr=9, scara=A, etaj=2, nrApt=12, denumire='S.C. Tester Prim', CUI=11223344}]

Aceste clase au constructori specifici care verifica datele primite ca parametri si, daca e cazul, se aruncă o exceptie IllegalArgumentException cu un mesaj de eroare predefinit (`Valori incorecte pentru crearea unei locuinte`) sau `Valori incorecte pentru crearea unui sediu de firma`).

Metoda *toString*() (moștenită de la Object) trebuie să fie suprascrisă astfel încât toate datele să fie returnate sub forma unui șir de caractere în conformitate cu exemplul de mai sus.

3. Gestiunea mediului de stocare persistenta - Data Access Object

Aveti la dispozitie pachetul ***Dao*** continand interfata **Dao**, clasa **SerializareDaoComplet** si clasa abstracta **Entitate**.

In implementarea pusa la dispozitie memorarea persistenta a datelor este realizata prin serializare intr-un fisier de catre clasa **SerializareDaoComplet** (care implementeaza interfata **Dao**).

Este obligatoriu sa proiecati aplicatia utilizand interfata **Dao** psi restul claselor din pachetul ***Dao*** pentru ca aplicatia sa nu se modifice daca se adopta o alta metoda de stocare a datelor (spre exemplu stocarea datelor intr-o baza de date). Daca se schimba solutia de stocare, atunci clasa **SerializareDaoComplet** va fi inlocuita cu alta clasa care tine cont de noua solutie de stocare (spre exemplu **SerializareDaoDB**).

NU MODIFICATI NIMIC DIN PACHETUL DAO

3.1. Interfata *Dao*

Această interfață definește metodele de accesare a datelor in mod independent de implementarea concretă a memoriei de date persistente (a se vedea design pattern-ul Data Access Object).

Interfața **Dao** conține metode abstracte pentru citirea și stocarea obiectelor de tip T (care extind clasa **Entitate**) avand cheia de tip K.

```
package ro.usv.dao;
import java.util.List;
public interface Dao<T,K> {
    T get( K id );
    void delete(K id);
    void save(T t);
    void update(T t);
    List<T> getAll();
    void saveAll(List<T> lst);
    void deleteAll();
}
```

3.2. Clasa SerializareDaoComplet

Reprezinta o implementare a interfetei **Dao** in care persistenta datelor este asigurata prin serializarea intr-un fisier a listei de obiecte.

```
class SerializareDaoComplet<T extends Entitate, K> implements Dao<T, K>
```

Clasa are 2 constructori:

1. constructorul fara argumente – serializarea are loc intr-un stream in memorie (se realizeaza o pseudo-persistenta utila doar pentru teste)
2. constructorul cu un singur argument, un șir de caractere reprezentand numele unui fișier - serializarea obiectelor va avea loc in acel fisier, asigurandu-se astfel persistenta datelor si dupa incetarea executiei programului.

În cazul erorilor rezultate în urma operațiilor cu fișiere, se afiseaza unul din mesajele de eroare

```
Eroare în timpul serializării
Eroare în timpul deserializării
```

iar programul se termină cu **System.exit(1)**.

Clasa **SerializareDaoComplet** pusa la dispozitie implementeaza in mod concret (fara exceptia UnspportedOperation) toate metodele din interfata **Dao.**

3.3. Clasa abstracta *Entitate*

Trebuie sa fie extinsa de clasele obiectelor ce vor fi stocate in mediul de stocare persistenta prin DAO.

4. Clasa *AsociatieProprietariServ*

Clasa ***AsociatieProprietariServ*** (de gestionare a proprietăților) este cea care implementează *logica de afaceri* punand la dispozitie o serie de ***servicii*** legate de gestiunea unei asociatii de proprietari.

Clasa ***AsociatieProprietariServ*** reprezinta un strat de proiectare logica separat si aflat deasupra stratului DAO realizand independenta aplicatiei fata de modul concret in care este realizata persistenta datelor.

Clasa ***AsociatieProprietariServ*** foloseste operatiile CRUD puse la dispozitie de clasa ***SerializareDaoComplet***.

Clasa ***AsociatieProprietariServ*** va avea 2 constructori:

1. Unul fara argumente – se va utiliza serializarea in memorie a datelor (pseudo-persistenta, utilizata in teste)
2. Unul cu un singur argment – un nume de fisier in care se vor serializa datelor (asigurandu-se persistenta)

Clasa trebuie sa aiba o variabilă de instanță privată de tipul (interfetei) **Dao**, pe care o foloseste pntru a accesa datele de tip ***Apartament*** memorate in mediul de stocare persistenta sau pseudo-persistenta. Aceasta variabila privata de tip **Dao** memoreaza o referinta catre o instanta a clasei concrete ***SerializareDaoComplet,*** instanta care poate fi construita apeland

- fie constructorul fara argumenete (serializarea are loc in memorie),
- fie constructorul cu un singur argument, transmitandu-i numele unui fisier, caz in care datele vor fi serializate in acest fisier.

Totodata, in clasa ***AsociatieProprietariServ*** exista metoda

```
public void setStocare(String nume)
```

care poate schimba instanta curenta a clasei **SerializareDaoComplet** cu una noua in care fisierul in care se vor serializa in continuare obiectele de tip ***Apartament*** are numele `nume`. Daca `nume` este `null` sau este sirul vid, atunci datele vor fi serializate in memorie.

Unele metode din aceasta clasa apeleaza doar metodele corespunzatoare din clasa **SerializareDaoComplet**:

- Metoda *getApartamentenById(int id)* returnează un obiect pe baza id-ului. Dacă nu se găsește apartamentul returneaza `null`. Spre exemplu, aceasta metoda returneaza doar rezultatul transmis de metoda *get(id)* din clasa **SerializareDaoComplet.**
- Metoda *getApartamentente()* returnează toate obiectele-**Apartament** stocate în mod persistent sub forma unei liste java.util.List.
- Metoda saveApartament(**Apartament** …) are ca scop salvarea persistentă a unui obiect **Apartament**. Asigurați-vă că, atunci când salvați un **Apartament** nou, nu se utilizează ID-ul unui **Apartament** deja salvat. În acest caz aruncă o excepție IllegalArgumentException cu un mesaj de eroare corespunzător.
- Metoda *deleteApartment(int id)* are rolul de a șterge un **Apartament** din sursa persistenta de date. În cazul în care apartamentul nu există, se arunca exceptia IllegalArgumentException cu un mesaj de eroare corespunzător.

- Metoda *deleteAllApartment()* are rolul de a șterge toate **Apartament**ele din sursa persistenta de date.

Daca aceste metode sunt deosebit de simple, celelalte, care sunt mai strans legate de logica de afaceri (n.b. gestiunea asociatiei), efectueaza prelucrari suplimentare apelurilor metodelor specificate de interfata **Dao**.

O parte din metodele clasei ***AsociatieProprietariServ*** pe care trebuie sa le programati sunt in sarcina ambelor numere, altele depind de numarul atribuit dv. (1 sau 2), asa cum se va preciza in continuare.

Metode ce vor fi programate de ***ambele numere***
• furnizarea datelor unui singur **Apartament** - *getApartamentenById(int id)*
• furnizarea datelor tuturor apartamentelor - *getApartamentente()*
• adăugarea unui nou **Apartament** in mediul de stocare persistenta - *saveApartament*(**Apartament** …)
• stergerea apartamentului care are un anumit ID - *deleteApartment(int id)*
• stergerea tuturor apartamentelor - *deleteAllApartment()*
• schimbarea fisierului in care se vor serializa obiectele (setStocare(numeFisier))

Metode ce vor fi programate doar de cei cu ***numarul 1***
• determinarea numărului total al tuturor apartamentelor locuinta sau sedii de firma (conform valorii parametrului *tip*)
• gasirea ID-ul (ID-urile) **Apartament**ului(elor) situate la un etaj mai mic sau cel mult egal cu cel dat ca parametru (*etaj*)
• gasirea ID-ul (ID-urile) **Apartament**ului(elor) situate pe o anumita strada

Metode ce vor fi programate doar de cei cu ***numarul 2***
• determinarea valorii medii a suprafatelor apartamentelor de tip *locuinta* sau *sedii de firma* (conform valorii parametrului *tip*); daca nu sunt apartamente de tipul cerut metoda va returna -1.
• gasirea ID-ul (ID-urile) celui (celor) mai recent construite **Apartament**(e)
• gasirea ID-ul (ID-urile) **Apartament**ului(elor) care au suprafata mai mare sau egala cu o anumita valoare

5. **Interfata *IAsociatieProprietariServ1*** sau ***IAsociatieProprietariServ2***

Clasa ***AsociatieProprietariServ*** implementeaza interfata ***IAsociatieProprietariServ1*** sau ***IAsociatieProprietariServ2***
Alegeti interfata corespunzatoare numarului atribuit dv. (***1*** sau ***2***)

Daca aveti numarul 1:

```
public interface IAsociatieProprietariServ1 {
    public void setStocare(String nume);
    public Apartament getApartamentById(int id);
    public List<Apartament> getApartamentente();
    public void saveApartament(Apartament ap);
    public void deleteApartment(int id);
    public void deleteApartmente();
```

```
    public int countApartamente(String tip);
    public List<Long> findIdsNewerThan(int nrAni);
    public List<Long> findIdsByStreet(String numeStrada);
}
```

sau daca aveti numarul 2:

```
public interface IAsociatieProprietariServ2 {
    public void setStocare(String nume);
    public Apartament getApartamentById(int id);
    public List<Apartament> getApartamentente();
    public void saveApartament(Apartament ap);
    public void deleteApartment(int id);
    public void deleteApartmente();

    public double getAverageSurface(String tip);
    public List<Long> findIdsFloorSmallerThan(int etaj);
    public List<Long> findIDsSurfaceGreaterThan(double smin);
}
```

6. Frontend

Scrieți un program client Java **AsociatieProprietariClient** care, utilizând clasa ***AsociatieProprietariServ***, pregateste interfața de linie de comandă descrisă in continuare.

Asigurați-vă că output-ul programului corespunde exact exemplelor de mai jos!

Programul trebuie să citeasca un **fisier de comenzi** în următorul format:

```
<Comanda 1>
<Comanda 2>
...
<Comanda n>
```

unde

- `<Comanda i>`: este de forma

  `<cuvant_rezervat> <parametri>`

  Nmarul de parametri depinde de comanda. Exista si comenzi fara parametri.
- <cuvant_rezervat> denumeste comanda si poate fi unul din sirurile urmatoare **`file, add, clear, delete, list, rem, stop, count, floor, street, avgsurf, newer, surfgt`**

O parte din comenzile pe care trebuie sa le programati in aceasta clasa sunt comune ambelor numere, altele depind de numarul atribuit dv. (**1** sau **2**)

Comenzi comune ambelor numere

- **`add, clear, delete, file, list, rem, stop`**

*Comenzi ce vor fi implementate de cei cu **numarul 1***

- **`count, newer, street`**

*Comenzi ce vor fi implementate de cei cu **numarul 2***

- **`avgsurf, floor, surfgt`**

Fisierul de comenzi

- Este un fisier text al carui nume il preluati din primul argument al liniei de comanda -> args[0] contine numele inclusiv calea absoluta; veti face doar

```
Scanner scnr = new Scanner( new File( args[0] ) );
```

-
- *Atentie. In fisierele in*.txt puse la dispozitie (in1.txt pentru numarul 1 si in2.txt pentru numarul 2, va trebui sa completati „...” pe 2 linii.*

In fisierul de comenzi fiecare comanda este inregistrata pe *o singura linie*. Programul va citi fisierul de comenzi linie cu line, va afisa linia citita si daca este o comanda corecta o va executa, altfel va afisa un mesaj de eroare corespunzator.

Comenzile scrise incorect vor fi semnalizate astfel:

- daca cuvantul cheie nu este recunoscut atunci se va afisa mesajul

```
Eroare. Comanda neimplementata
```

- daca numarul de parametri este incorect atunci se va afisa mesajul

```
Eroare. Numarul parametrilor nu este corect
```

- daca formatul este necorespunzator (se asteapta un numar si parametrul nu reprezinta un numar) se va afisa mesajul

```
Eroare. Format incorect pentru parametru(i)
```

La intalnirea comenzii **stop** programul se va opri**.** Daca nu exista comanda **stop** atunci programul se va opri dupa ce s-au citit toate liniile din fisierul de comenzi.

Comenzile pe care trebuie sa le programeze **ambele numere**
`add, list, clear, delete, file, rem, stop`

Comanda add *parametri*

Adauga un apartament ***in mediul de stocare persistenta*** (save). Parametrii sunt specifici fiecarui tip de apartament si sunt dati exact in ordinea in care sunt prezentati de metoda *toString*() a clasei de apartamente.

Exemple:

```
add L 1 51. 1990 Zorilor 10  A 2 12 3
add SF 190  50 1990 Zorilor 9  A  2  12 "S.C. Tester Prim" 11223344
add L 11 510. 1963 Universitatii 10  A
Eroare. Numarul parametrilor nu este corect
```

Dacă parametrii conțin spații libere folositi ghilimele.
Efect: Se creeaza obiecte de tipul respectiv care sunt memorate persistent.

Comanda list [id]

Comanda *afiseaza toate datele referitoare la apartamentul cu id-ul specificat (daca exista).* Daca parametrul lipseste se afiseaza toate apartamentele din mediul de stocare.
Exemplu:

```
list
```

Rezultat:

```
[
{Tip=L, id=1, suprafata=51.0, anConstructie=1990, strada='Zorilor',
nr=10, scara=A, etaj=2, nrApt=12, nrPersoane=3},
{Tip=L, id=11, suprafata=64.5, anConstructie=2020, strada='Eroilor',
nr=109, scara=B, etaj=3, nrApt=14, nrPersoane=5},
{Tip=L, id=101, suprafata=60.0, anConstructie=2012, strada='Zorilor',
nr=15, scara=A, etaj=1, nrApt=5, nrPersoane=2},
```

```
{Tip=SF, id=190, suprafata=50.0, anConstructie=1990, strada='Zorilor',
nr=9, scara=A, etaj=2, nrApt=12, denumire='S.C. Tester Prim',
CUI=11223344}]
```

Comanda **list <id>**: Returneaza toate datele unui singur apartament
Exemplu:
list 190
Rezultat:

```
{Tip=SF, id=190, suprafata=50.0, anConstructie=1990, strada='Zorilor',
nr=9, scara=A, etaj=2, nrApt=12, denumire='S.C. Tester Prim',
CUI=11223344}
```

list 1900
Rezultat:

```
Eroare. Nu exista apartament cu id=1900
```

> *Nota. In cele ce urmeaza nu vom mai insera cuvantul **Rezulat**, ceea ce urmeaza unei comenzi fiind afisarea produsa dupa executia comenzii.*

Comanda `clear`

Elimina din mediul de stocare persistenta toate apartamentele.
Exemple si rezultate

```
clear
S-au eliminat toate apartamentele
clear
S-au eliminat toate apartamentele
list
[]
```

Comanda `delete id`

Elimina din mediul de stocare persistenta apartamentul cu ID-ul *id*
Exemple si rezultate

```
delete 190
delete 1900
Eroare. delete: obj cu id=1900 nu exista
```

Comanda `file numefisier`

Stabileste care este numele fisierului in care se vor serializa datele
Exemplu:
file asocGEnescu.ser
Numele acestui fisier trebuie transmis clasei **AsociatieLocatariServ**

In lipsa unei comenzi **file** si pana la prima comanda **file** se considera ca serializarea are loc in memorie. In acest ultim caz se apeleaza constructorul fara argumente al clasei **AsociatieLocatariServ** care, la randul lui, va apela constructorul fara argumente al clasei **SerializareDaoComplet**).

Comanda `rem`

Reprezinta un comentariu (**rem**ark). Nu are parametri, se afiseaza doar linia curenta si se trece la urmatoarea comanda fara a se executa nimic altceva.
Exemplu si rezultat

```
rem completati ... cu informatiile din text ref. la ap. cu id=11
rem completati ... cu informatiile din text ref. la ap. cu id=11
```

(doar afisarea liniei).

Comanda `stop`

Nu are parametri, opreste executia programului.
Exemplu si rezultat

```
stop
La revedere!
```
(afisarea mesajului si terminarea programului).

Comenzile pe care trebuie sa le programeze doar cei cu **numarul 1**
`count, newer, street`

Nota. In exemplele de mai jos s-a considerat ca in mediul de stocare persistenta exista cele 4 apartamente afisate in exemplul anterior cu comanda **list**.

Comanda `count [tip]`
Parametrul *tip* poate fi ***L*** si atunci se numara locuintele sau ***SF*** si atunci se afiseaza numarul de sedii de firma.
Daca parametrul *tip* lipseste atunci se returneaza numarul total al apartamentelor (locuinte + sedii firma).
Exemple si rezultate

```
count L
Nr.locuinte: 3
count SF
Nr.sedii firme: 1
count
Nr.apartamente: 4
count ?
Nu sunt apartamente de tipul ?
```

Daca dupa aceasta comanda se dau urmatoarele comenzi se obtin rezultatele:

```
delete 190
Eroare. delete: obj cu id=190 nu exista
count SF
Nr.sedii firme: 0
```

Comanda `newer nrAni`
Afiseaza ID-ul/ID-urile apartamentelor care s-au construit cu cel mult *nrAni* in urma
Exemple si rezultate

```
newer 10
Ap. construite cu cel mult 10 ani in urma: [11, 101]
newer 5
Ap. construite cu cel mult  5 ani in urma: [11]
newer 1
Ap. construite cu cel mult  1 ani in urma: []
```

Comanda `street str`
Afiseaza ID-ul/ID-urile apartamentelor situate pe strada cu numele precizat in parametrul *str*
Exemple si rezultate

```
street Zorilor
Ap. str. Zorilor[1, 101, 190]
street "Stefan cel Mare"
Ap. str. Stefan cel Mare[]
```

Dacă parametrii conțin spații libere folositi ghilimele (ca si la comanda **add**)

Comenzile pe care trebuie sa le programeze doar cei cu **numarul 2**
`avgsurf, floor, surfgt`

Nota. In exemplele de mai jos s-a considerat ca in mediul de stocare persistenta exista 4 apartamente.

```
list
[
{Tip=L, id=1, suprafata=51.0, anConstructie=1990,
strada='Zorilor', nr=10, scara=A, etaj=2, nrApt=12,
nrPersoane=3},
{Tip=L, id=11, suprafata=64.5, anConstructie=2020,
strada='Eroilor', nr=109, scara=B, etaj=3, nrApt=14,
nrPersoane=5},
{Tip=L, id=101, suprafata=60.0, anConstructie=2012,
strada='Zorilor', nr=15, scara=A, etaj=1, nrApt=5,
nrPersoane=2},
{Tip=SF, id=190, suprafata=50.0, anConstructie=1990,
strada='Zorilor', nr=9, scara=A, etaj=2, nrApt=12,
denumire='S.C. Tester Prim', CUI=11223344}]
```

Comanda `avgsurf [tip]`

Afiseaza suprafata medie a apartamentelor de tipul precizat de parametrul *tip*.
Daca parametrul *tip* este **L** atunci se afiseaza suprafata medie a locuintelor, daca este **SF** atunci se afiseaza suprafata medie a sediilor de firma.
Daca parametrul *tip* lipseste atunci valoarea medie a suprafetelor apartamentelor va fi calculata pentru toate apartamentele (locuinte + sedii firma).

Exemple si rezultate

```
avgsurf L
Suprafata medie a locuintelor: 58.5
avgsurf SF
Suprafata medie a sediilor de firme: 50.0
avgsurf
Suprafata medie a apartamentelor: 56.375
avgsurf X
Nu sunt apartamente de tipul X
```

Daca se efectueaza comenzile urmatoare se obtin rezultatele

```
delete 190
avgsurf SF
Nu sunt apartamente de tipul SF
```

Comanda `floor etaj`

Afiseaza ID-ul/ID-urile apartamentelor situate la etaje mai mici sau cel mult egale cu parametrul *etaj*.

Exemple si rezultate

```
floor 1
Ap. situate la un etaj <=1 [101]

floor 0
Ap. situate la parter []
```

Comanda `surfgt smin`

Afiseaza ID-ul/ID-urile apartamentelor care au suprafata mai mare sau cel putin egala cu valoarea parametrului *smin*.
Exemple si rezultate

```
surfgt 60
Ap. cu suprafata cel putin egala cu 60.0: [11, 101]
surfgt 100
Ap. cu suprafata cel putin egala cu 100.0: []
surfgt 0
Ap. cu suprafata cel putin egala cu 0.0: [1, 11, 101, 190]
```

Observatie. Numerele reale nu se formateaza.

7. Mesaje de eroare

Toate excepțiile aruncate de metodele din pachetul Dao si cele aruncate de metodele din clasele realizate de dv. trebuie să fie „prinse" (capturate). In cazul capturarii unei exceptii programul afiseaza un mesaj de eroare dupa care trece la comanda urmatoare din fisierul de comenzi.

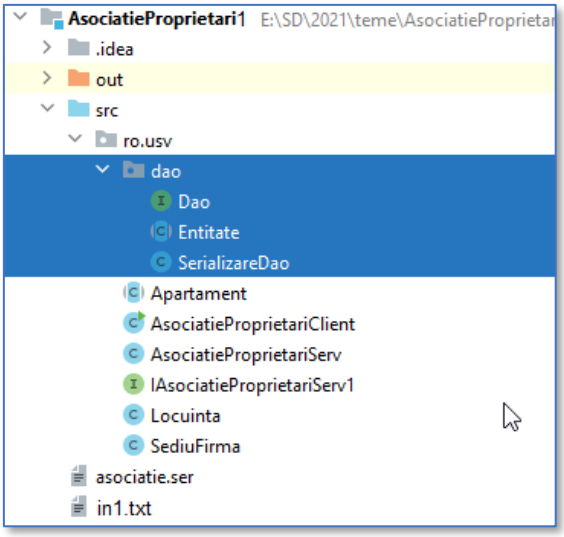
Exemple de mesaje de eroare provenind din exceptii:

```
Eroare. get: parametru id=null
Eroare. save: parametru obj. null
Eroare. save: obj. exista deja id=1
Eroare. delete: obj cu id=190 nu exista
Eroare. Format incorect pentru parametru(i)
Eroare. Valori incorecte pentru crearea unui apartament
Eroare. Valori incorecte pentru crearea unui sediu de firma
Eroare. ...
```

Modalități de prezentare

Termen limită: ~~Luni, 10.01.2022, ora 10:01,~~ **13.01.2022 ora 23:59.**

Proiectul trebuie sa se numeasca *AsociatieProprietari1* sau *AsociatieProprieteari2* corespunzator numarului dv. Structura proiectului trebuie sa fie urmatoarea (exemplu pentru **numarul 1)**:



Cu albastru s-a marcat pachetul **ro.usv.dao** care va este pus la dispozitie (atentie, nu modificati nici o clasa sau interfata din acest pachet; daca aveti probleme va veti adresa titularului acestui curs).

Ca bază pentru dezvoltarea programului dumneavoastră, utilizați clasele Java conținute în arhiva ro_usv_Dao.zip ce contine pachetul `ro.usv.dao`.

Numele claselor si/sau interfetelor pe care trebuie sa le realizati trebuie sa fie cele precizate in acest material! ATENTIE. Clasa principala trebuie sa fie NEAPARAT **`ro.usv.AsociatieProprietariClient`**

Programul trebuie depus sub forma unei arhive zip care va purta numele dv (ex.IonescuIonela.zip) si care trebuie sa contina:

- folderul src
- fisierul in1.txt (respectiv in2.txt) – va trebui sa completati „...” de la 2 linii
- ~~fisierul ou1.txt (respectiv out2.txt)~~

Pentru trimiterea temei

- veti trimite arhiva ZIP a intregului proiect; in IntelliJ se obtine astfel File -> Export -> Project to Zip File
- numele fisierului de comenzi va fi preluat din linia de comanda -> in args[0] il gasiti cu tot cu calea sa absoluta, veti face doar `Scanner scnr = new Scanner( new File( args[0] ) );`
- programul va afisa rezultatele in fisierul standard de iesire (System.out.print sau println).
- Cei care au realizat programul astfel incat sa produca un fisier de iesire (out1.txt sau out2.txt) pot face modificarea (exemplu): in loc de: `PrintWriter logger = new PrintWriter( new File("out2.txt"));`
- - vor scrie: **`PrintWriter logger = new PrintWriter(System.out);`**

Termenul de predare este 13.01.2022 ora 12:00

Informații suplimentare si idei de rezolvare în acest sens pot fi găsite pe Classroom (curs si consultatii).

Pentru nelamuriri va rog sa folositi stream-ul de la Assignement-ul de pe Classrom corespunzator temei.

SUCCES!