MODEL 2024

A.

1.Argumentati necesitatea realizarii modelelor in procesul de dezvoltare a softului

In procesul de dezvoltare a softului modelarea este importanta pentru ca permite abstractizarea si analiza problemei pentru a planifica corect implementarea si a evita greseliile, precum si pentru a facilita intelegerea intre membrii echipei.

2.Ce diferentiaza relatia de compunere de cea de agregare intre clase + exemple

Agregarea este un tip special de asociere care reprezinta o relatie de tip parte-intreg, unde partile pot exista independent de intreg. Un astfel de exemplu este o masina, care are 4 roti, dar aceste roti poti exista si fara sa fie puse unei masini.

In schimb, compunerea este o forma mai stricta de agregare unde partile sunt dependente de intreg. Un astfel de exemplu este o casa, care este compusa din camere, dar aceste camere nu pot exista in afara unei case.

3.Sablonul de proiectare Strategy

Sablonul de proiectare Strategy este folosit pentru incapsulare algoritmilor. Presupune existenta unui numar de algoritmi diferiti pentru rezolvarea aceleiasi probleme si necesitatea de a interschimba dinamic algoritmul folosit.

O clasa context este cea care primeste cererile de la client si in functie de specificatii va folosi o anumita strategie (un anumit algoritm) pentru procesarea cererii.

Exp: O aplicatie de gps care poate sa faca rute pentru masini, mersul cu autobuzul, mersul cu bicicleta sau pentru pietonii folosind strategii diferite pentru fiecare ruta.

4.UML + OCL, exemplu de invariant care sa foloseasca iteratorul forAll pe colectii

BILET 1

A.

1.Explicati diferenta dintre domeniul problemei si domeniul solutiei

Domeniul problemei se refera la contextul si mediul in care o problema exista si trebuie rezolvata, iar domeniul solutiei se refera la modul in care problema este abordata si rezolvata.

Domeniul problemei – intelegerea si definirea problemei

Domeniul solutiei – dezvoltarea si implementarea unei solutii tehnice pentru o problema

2.Definiti conceptul de scenariu

Un scenariu este o instanta a unui caz de utilizare, ce descrie o secventa concreta de evenimente, reprezentand o interactiune tipica a unui utilizator cu sistemul.

3.Dati exemplu de diagrama de obiecte corespunzatoare unei diagrame de clase

EZ

4.Explicati modul in care numele de roluri si multiplicitatea specificate la nivelul diagramei de clase influenteaza codul sursa aferent acelei diagrame

Numele de la capetele relatiilor determina numele de atribute din clase, multiplicitatea va determina daca un atribut din clasa este simplu sau o colectie sau poate fi null.

5.Explicati semnificatia stereotipurilor <<entity>>, <<boundary>>, <<control>>

Entity – responsabile de informatia persistenta din sistem.

Boundary – reprez inerfata actorilor cu sistemul.

Control – responsabile de coordonare claselor entity si boundary sau de realizarea cazurilor de utilizare.

6.Trei exemple de stiluri arhitecturale

Model-view-controller, client-server, peer-to-peer, three-tier-architecture, four-tier-architecture, repository.

B.

Depozit de produse. La un depozit de produse se foloseste o aplicatie pentru a tine evidenta produselor din depozit. Pentru fiecare produs se pastreaza denumirea si cantitatea disponibila. Cand un client vine sa cumpere produse, se alege produsul din lista celor existente, se introduce cantitatea dorita si se apasa butonul "CUMPARARE". Dupa apasare cantitatea disponibila se actualizeaza. Operatorul poate sa adauge noi produse. Pentru un nou produs se introduce denumirea si cantitatea disponibila. Dupa adaugarea unui nou produs lista produselor se actualizeaza. La pornirea aplicatiei informatiile despre produse se citesc dintr-o baza de date relationala, iar la inchiderea aplicatiei informatiile din baza de date sunt actualizate.

1.Identificati actorii si cazurile de utilizare aferente acestora

Actori: client, operator, sgbd

Cazuri de utilizare: (alege produs) – hmm, cumparare produs, adaugare produse

2.Diagrama cazurilor de utilizare

EZ

3.Identificati conceptele, atributele acestora si relatiile dintre ele

Concepte: depozit, produs (denumire + cantitate), achizitie (de produs), client, operator

Atribute: EZ

Relatii: efectiv relatiile dintre ele (probabil ca trebuie adaugata alte clase ici colo)

4.Sa facem modelul cu toate chestiilde de la 3

C.

1.Justificati utilitatea sablonului de proiectare Proxy

Sablonul Proxy asigura un surogat in scopul controlarii accesului la acesta, iar acesta poate fi util in folosirea unui proxy la distanta care ascunde faptul ca un obiect se afla in alt spatiu de adresa sau in folosirea unui proxy virtual pentru crearea de obiecte la cerere cu scopul de a imbunatatii performanta.

2.Definiti conceptul de delegare

Delegarea reprezinta o alternativa la mostenirea implementarii unei clase, atunci cand se doreste implementarea unei functionalitati prin reutilizare.

3.Principiul deschis/inchis

Clasele (`Entitatiile software`) ar trebui sa fie deschise pentru extindere, dar inchise pentru modificari.

4.Regula privind mostenirea contractelor referitoare la preconditii

Clientul asteapta ca un contract formulat relativ la clasa de baza sa fie respectat si de clasele derivate. Unei metode dintr-o subclasa ii este permis sa slabeasca preconditiile metodei pe care o supradefineste.

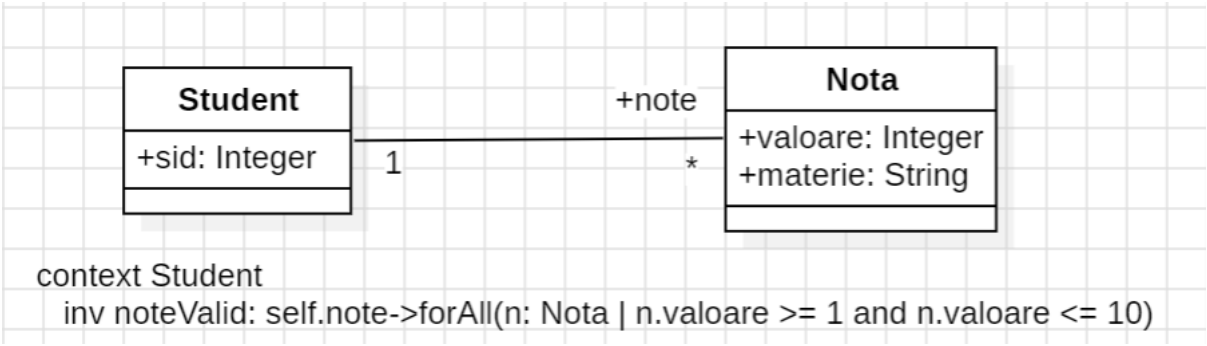
(O metoda care supradefineste trebuie sa fie cel putin la fel de puternica precum cea supradefinita)

(O clasa trebuie sa respecte toti invariantii superclasei, eventual poate introduce noi invarianti)

5.Principiul non-redundantei in contextul Design by Contract

In corpul functiei nu trebuie verificate preconditii

6.UML si OCL – invariant care sa foloseasca un iterator pe colectii



D. Diagrama de secventa, comunicare si diagrama detaliata – EZ

BILET 6

A.

1.Enumerati activitatile tehnice ale dezv. de soft orientate obiect

-colectarea cerintelor(cazuri de utilizare), analiza cerintelor(model conceptual si dinamic), proiectarea de sistem(obiective de proiectare), proiectarea obiectuala, implementarea(transf in cod a conceptelor), testarea.

2.Ilustrati, folosind un exemplu o modalitate de transformare a unei clase de asociere in clase de asocieri obisnuite: EZ

3.Asociere one-to-many bidirectionala: Meh

C.

1.Testarea bazata pe echivalente

Tehnica de testare blackbox care minimizeaza numarul de cazuri de test, prin partitionarea intrarilor posibile in clase de echivalenta si selectarea unui caz de test pentru fiecare astfel de clasa.

Testarea comportamentului aferent unei clase de echivalenta se poate realiza prin testarea unui singur membru al clasei.

2.Asociere calificata: Meh

3.Sablonul Adapter

Sablonul Adapter permite interfata unei clase existente sa fie folosita ca o alta interfata, actioneaza ca o legatura intre doua interfete care sunt incompatibile.

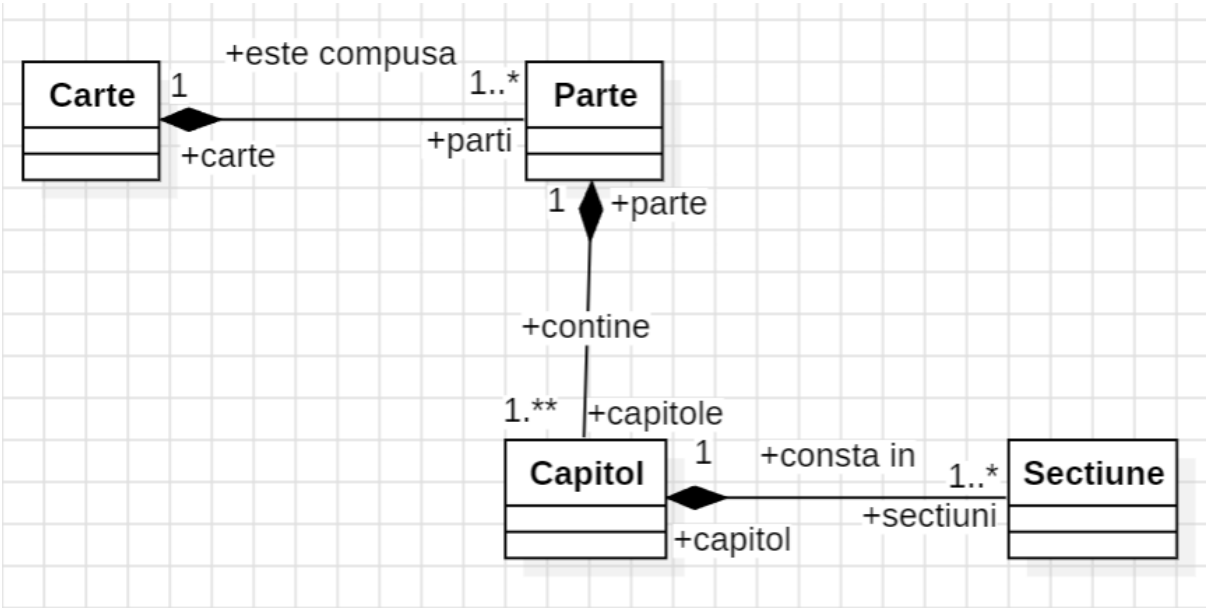
SUBIECTE A

A10.

1.Utilitatea realizarii diagramelor de interactiune in etapa de analiza a sistemelor soft

Se conceptualizeaza sabloanele de comunicare intr-o multime de obiecte care interactioneaza si se conceptualizeaza functionalitatiile pe care le ofera sistemul.

2.



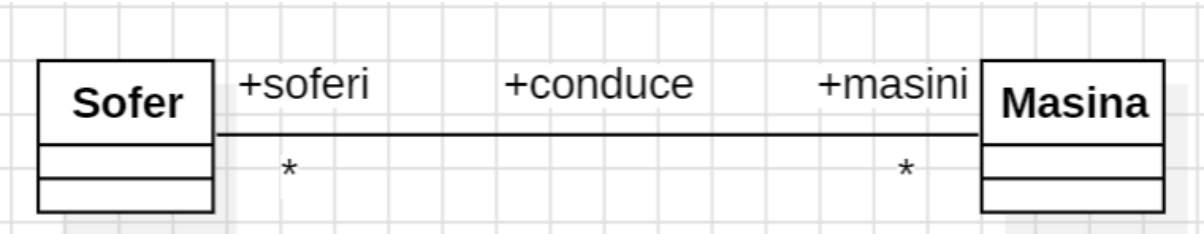
3.Care sunt regulile de reprezentare a numerelor de roluri si multiplicitatea capetelor de asociere in codul sursa? + Exemple (diagrama/cod sursa)

Numele de pe capetele relatiilor din diagrame trebuie sa coincida cu numele atributelor din codul sursa, iar multiplicitatea pe diagrame reprezinta daca un atribut este o referinta simpla sau o colectie de referinte la clasa asociata.

+ Exemple (Common sense duh)

A13.

3.Asociere many-to-many bidirectionala + transformare in cod sursa




```
public class Sofer {
private Set<Masina> masini;
public Sofer() {
        masini = new HashSet();
}

public addMasina(Masina masina) {
        if (!masini.contains(masina)) {
                masini.add(masina);
                masina.internalAddSofer(this);
        }

private internalAddMasina(Masina masina) {
        if (!masini.contains(masina))
                masini.add(masina);
}
}

public class Masina {
private Set<Sofer> soferi;

public Masina() {
        soferi = new HashSet();
}
public addSofer(Sofer sofer) {
        if(!soferi.contains(sofer)) {
                soferi.add(sofer);
```

```
            sofer.internalAddMasina(this);
        }
}

private internalAddSofer(Sofer sofer) {
        if (!soferi.contains(sofer))
            soferi.add(sofer);
}
```

SUBIECT C

C2.

1.Care este esenta abordarii Design by Contract referitor la asigurarea fiabilitatii sistemelor soft?

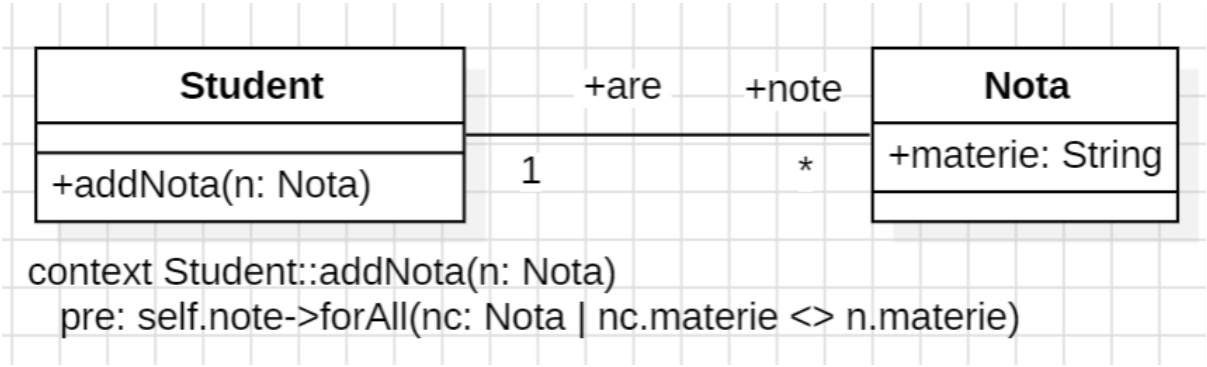
Design by Contract este o metodă de dezvoltare software care descrie interacțiuni între componente și utilizează pre/post-condiții, invarianți și assertions(verificări) pentru a asigura corectitudinea și robustețea programului.

2.Modalitati reprezentare a ierarhiilor de mostenire…

-mapare verticala: definirea unui tabel cu atributele proprii ale acesteia + o referinta spre tabela parinte

-mapare orizontala: se introduc atributele tabelei parinte in tabelele fii

3.UML + OCL, preconditie cu iterator pe colectii



C4.

1.Definiti conceptul de delegare

Delegarea reprezinta o alternativa la mostenirea implementarii unei clase, atunci cand se doreste implementarea unei functionalitati prin reutilizare.

2.Dati un exemplu de transformare la nivelul modelului (model-to-model) + explicatii

Obiectivul este simplificarea, detalierea sau optimizarea modelului initial, în conformitate cu cerintele din specificatie.

3 clase diferite au acelasi atribut email: String, se face o superclasa de la care se mosteneste care are String.

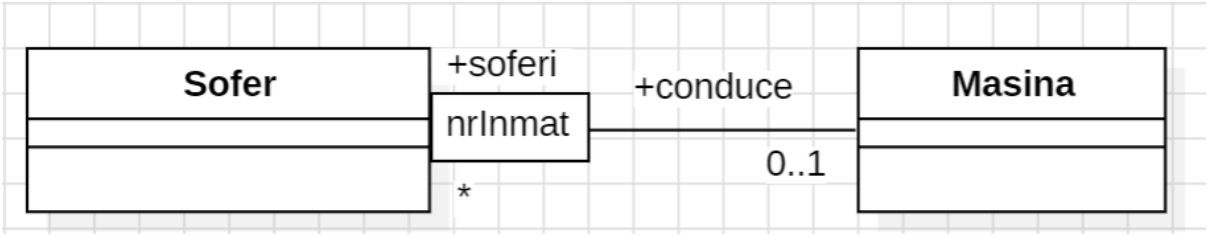
3.Enuntati invariant + OCL corespunzator

context Produs

inv cantitateValida: self.cantitate > 0

C5.

2.




```
public class Sofer {
private Map<String, Masina> masini;
public Sofer() {
masini = new HashMap();
}

public addMasina(String nrInmat, Masina masina) {
        // daca nu contine mapul
        masini.put(nrInmat, masina);
        masina.internalAddSofer(this);
}

private void internalAddMasina(String nrInmat, Masina masina) {
        if (!masini.containsKey(nrInmat)) {
```

```
                masini.put(nrInmat, masina);
}

public class Masina {
private Set<Sofer> soferi;
public Masina() {
        soferi = new HashSet();
}

public void addSofer(Sofer sofer) {
        // daca nu este continut in set
        soferi.add(sofer);
        sofer.internalAddMasina(this);
}

private void internalAddSofer(Sofer sofer) {
        // daca nu contine
        soferi.add(sofer);
}
}
```

3. 3. Folosind UML si OCL, dati un exemplu de post-conditie care sa foloseasca unul dintre iteratorii pe colectii.

```
context Sofer::AddMasina(nrInmat: String, masina: Masina)
  post: self.masini = self.masini@pre->including(masina)
```

C7.

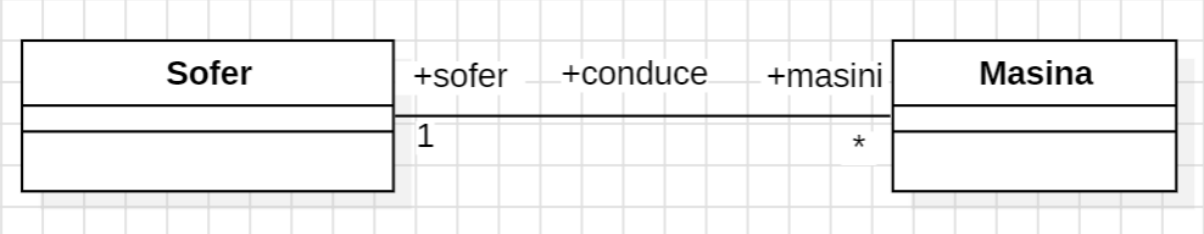
1.Justificati necesitatea utilizarii limbajului OCL in modelare

-a aparut din necesitatea de a acoperi problemele de expresivitate ale UML

-OCL se traduce mai bine in cod, datorita pre si post conditiilor

2. LMAO

3.One-to-many bidirectionala diagrama + cod




```
public class Sofer {
private Set<Masina> masini;

public Sofer() {
        masini = new HashSet();
}

public addMasina(Masina masina) {
        if (!masini.contains(masina)) {
                masini.put(masina);
                masina.internalSetSofer(this);
}
}

public class Masina {
private Sofer sofer;

private internalSetSofer(Sofer sofer) {
        this.sofer = sofer;
}
```

C8.

1.Open/close – open to extensions, closed to modifications

2.Reprez asocieri in modelul obiectual – baze de date

◦ Asocierile one-to-one si one-to-many se reprezinta folosind chei straine

◦ Asocierile many-to-many se reprezinta folosind tabele de leg atura

3. Common sense

C10.

1.Separarea interfetelor

Clasele care implementeaza o interfata trebuie sa implementeze doar metodele pe care le intereseaza, altfel o interfata mare trebuie impartita in interfete mai mici, mai specifice.