ȘATALAN ALEXANDRA-MARIA

LAB1_IS

1. Instalarea mediului de lucru Visual Paradigm Community Edition.
   DA
2. Ce este UML și pentru ce este utilizat?
   UML este un limbaj de modelare generalizat standardizat în domeniul ingineriei software. Standardul este gestionat și a fost creat de OMG (Object Management Group). Include un set de tehnici de notare grafică pentru a crea modele vizuale de sisteme intensive în software.

3. Ce sunt modelele și care este utilitatea lor?
   1. **Definirea modelelor și rolul lor:**
      Modelele permit descrierea eficientă și elegantă a sistemelor complexe prin reducerea acestora la aspectele esențiale. Un **sistem** este un ansamblu integrat de componente interconectate care funcționează împreună pentru un scop comun. În tehnologia informației, accentul se pune pe **modelele software** care descriu structura și comportamentul sistemelor informatice.
   2. **Abstracția în modelare:**
      Modelele folosesc **abstracția** pentru a simplifica realitatea. Aceasta implică eliminarea detaliilor nesemnificative și păstrarea caracteristicilor relevante pentru scopul modelului. Alegerea corectă a mijloacelor de abstracție este crucială pentru a asigura simplitatea și flexibilitatea sistemului. Alegerea greșită poate duce la interfețe complicate și dificultăți în implementarea modificărilor.
   3. **Definiția modelului (după Herbert Stachowiak):**
      Un model este caracterizat prin:
      - **Mapare:** Reprezintă o imagine a unui obiect real sau artificial.
      - **Reducție:** Include doar atributele relevante pentru scopul modelului.
      - **Pragmatism:** Este creat pentru un anumit scop, utilizator și context specific.

4. **Proprietățile unui model de calitate (după Bran Selic):**

   - **Abstracție:** Reprezentare simplificată, focalizată pe esențial.
   - **Ușurință în înțelegere:** Elemente intuitive, posibil grafic, pentru a facilita interpretarea.
   - **Acuratețe:** Reprezintă fidel caracteristicile relevante ale sistemului.
   - **Predictibilitate:** Permite simularea și analiza comportamentului sistemului.
   - **Cost-eficiență:** Crearea modelului trebuie să fie mai ieftină decât dezvoltarea directă a sistemului.

5. Tipuri de modele:

- **Modele descriptive:** Ilustrează realitatea pentru o mai bună înțelegere (ex: o hartă a orașului).
- **Modele prescriptive:** Oferă un ghid de construcție al sistemului (ex: diagrame UML pentru dezvoltare software).

6. Aplicații ale modelelor în dezvoltarea software:

- **Modele ca schiță:** Folosite pentru a comunica idei generale, fără a include toate detaliile. Sprijină brainstorming-ul și luarea deciziilor.
- **Modele ca plan detaliat:** Conțin suficiente informații pentru a permite dezvoltatorilor să implementeze sistemul fără a lua decizii suplimentare.
- **Modele ca programe executabile:** Modele suficient de detaliate pentru a genera automat cod sursă, fiind utilizate în dezvoltarea software bazată pe modele (ex: în sisteme embedded).

---

7. Forward și Backward Engineering:

- **Forward engineering:** Modelul este folosit pentru a genera cod sursă.
- **Backward engineering:** Codul existent este analizat și transformat într-un model pentru a facilita înțelegerea și documentarea.

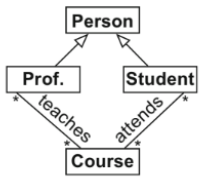
Modelele sunt esențiale în dezvoltarea software, ajutând la reducerea complexității, îmbunătățirea comunicării și asigurarea flexibilității în proiectare și implementare. Utilizarea corectă a modelelor poate crește eficiența și calitatea produsului software final.

4. Familiarizarea cu mediul de lucru mai sus amintit prin crearea anumitor diagrame din sectiunile 2.3.1 și 2.3.2 din [1] (class diagram – diagrama de clasă, use case diagram – diagrama de cazuri de utilizare, activity diagram – diagrama de activitate, sequence diagram – diagrama de secvente, state machine diagram – diagrama mașinii de stare).

1.Class Diagram

Conceptele diagramei de clase provin din modelarea conceptuală a datelor și dezvoltarea software orientată pe obiect. Aceste concepte sunt utilizate pentru a specifica structurile de date și structurile de obiecte ale unui sistem. Diagrama de clase se bazează în principal pe conceptele de **clasă**, **generalizare** și **asociere**.

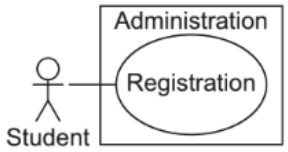
De exemplu, într-o diagramă de clase, poți modela clasele **Curs**, **Student** și **Profesor** care apar într-un sistem. Profesorii predau cursuri, iar studenții participă la cursuri. Studenții și profesorii au proprietăți comune, deoarece amândoi sunt membri ai clasei **Persoană**. Acest lucru este exprimat printr-o relație de generalizare.



2.Use Case Diagram

UML oferă **diagrama de cazuri de utilizare** pentru a-ți permite să definești cerințele pe care un sistem trebuie să le îndeplinească. Această diagramă descrie care utilizatori folosesc care funcționalități ale sistemului, dar nu abordează detalii specifice ale implementării. Unitățile de funcționalitate pe care sistemul le oferă utilizatorilor săi se numesc **cazuri de utilizare**.

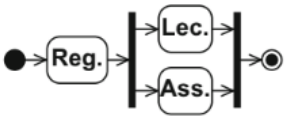
De exemplu, într-un sistem de administrare universitară, funcționalitatea **Înregistrare** ar reprezenta un caz de utilizare folosit de studenți.



3.Activity Diagram

Poți modela procese de orice tip folosind diagrame de activitate: atât procese de afaceri, cât și procese software. De exemplu, o diagramă de activitate poate arăta ce acțiuni sunt necesare pentru ca un student să participe la un curs și la o sarcină.

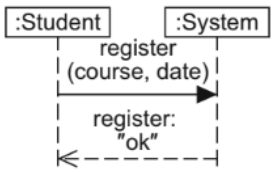
Diagramele de activitate oferă mecanisme de control al fluxului, precum și mecanisme de flux de date care coordonează acțiunile ce compun o activitate, adică un proces.



4.Sequence diagram

Diagrama de secvență descrie interacțiunile dintre obiecte pentru a îndeplini o anumită sarcină, de exemplu, înregistrarea la un examen într-un sistem de administrare universitară. Accentul este pus pe ordinea cronologică a mesajelor schimbate între partenerii de interacțiune.

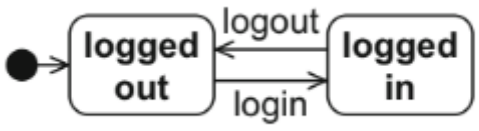
Diverse construcții pentru controlul ordinii cronologice a mesajelor, precum și concepte pentru modularizare, îți permit să modelezi interacțiuni complexe.



5.State machine diagram

Pe parcursul ciclului lor de viață, obiectele trec prin diferite stări. De exemplu, o persoană se află în starea delogat atunci când vizitează pentru prima dată un site web. Starea se schimbă în logat după ce persoana introduce cu succes numele de utilizator și parola (evenimentul de logare). Imediat ce persoana se deloghează (evenimentul de delogare), aceasta revine la starea delogat.

Acest comportament poate fi reprezentat în UML folosind diagrama mașinii de stare (*state machine diagram*). Această diagramă descrie comportamentul permis al unui obiect sub forma stărilor posibile și a tranzițiilor între stări, declanșate de diverse evenimente.



5. Ce tipuri de diagrame sunt prezentate în [1] – Capitolul 2? Care sunt asemănarile și deosebirile?

1. **Diagrame Structurale**
   Aceste diagrame se concentrează pe reprezentarea statică a sistemului, evidențiind componentele și relațiile dintre ele.
   - **Diagrama de Clase (Class Diagram):** Modelează structura sistemului prin clase, atribute, metode și relațiile dintre clase (asocieri, generalizări).
   - **Diagrama de Obiecte (Object Diagram):** Prezintă o instantanee concretă a obiectelor și relațiilor acestora într-un moment specific.
   - **Diagrama de Pachete (Package Diagram):** Grupează clase sau alte elemente în pachete pe baza coeziunii funcționale.
   - **Diagrama de Componente (Component Diagram):** Modelează componentele software și interacțiunile dintre acestea.
   - **Diagrama de Structură Compozițională (Composite Structure Diagram):** Descompune sistemul în părți mai mici și detaliază structura internă a componentelor.
   - **Diagrama de Implementare (Deployment Diagram):** Reprezintă arhitectura hardware și modul în care componentele software sunt distribuite pe noduri.
   - **Diagrama de Profil (Profile Diagram):** Extinde UML pentru a introduce concepte specifice domeniului.
2. **Diagrame Comportamentale**
   Aceste diagrame modelează aspectele dinamice ale sistemului și descriu comportamentul în funcție de evenimente și interacțiuni.
   - **Diagrama de Cazuri de Utilizare (Use Case Diagram):** Prezintă funcționalitățile sistemului din perspectiva utilizatorilor și relațiile dintre actori și cazurile de utilizare.
   - **Diagrama Mașinii de Stări (State Machine Diagram):** Modelează stările posibile ale unui obiect și tranzițiile între ele în funcție de evenimente.
   - **Diagrama de Activități (Activity Diagram):** Descrie fluxul de lucru sau logica unui proces prin activități și tranziții.

- **Diagrama de Secvență (Sequence Diagram):** Evidențiază interacțiunile dintre obiecte într-o ordine cronologică.
- **Diagrama de Comunicare (Communication Diagram):** Reprezintă schimbul de mesaje între obiecte, punând accent pe relațiile dintre acestea, nu pe ordinea temporală.
- **Diagrama de Timp (Timing Diagram):** Ilustrează schimbările de stare în timp pentru obiecte și interacțiunile lor.
- **Diagrama de Prezentare a Interacțiunilor (Interaction Overview Diagram):** Integrează mai multe diagrame de interacțiune pentru a descrie un flux complex de activități.

Asemănări între diagrame

- **Standardizare:** Toate diagramele urmează specificațiile UML și sunt utilizate pentru modelarea sistemelor software.
- **Claritate vizuală:** Folosesc simboluri grafice standardizate pentru a facilita înțelegerea de către dezvoltatori, analiști și alți participanți.
- **Reprezentare modulară:** Permite reprezentarea unui sistem complex prin componente mai mici și mai ușor de gestionat.
- **Suport pentru dezvoltare software:** Atât diagramele structurale, cât și cele comportamentale ajută la înțelegerea, proiectarea și documentarea sistemelor.

Deosebiri principale între diagramele structurale și comportamentale

1. **Scopul și perspectiva oferită:**
    - **Diagrame structurale:** Pun accent pe *ce* conține sistemul — clase, obiecte, componente și relații.
    - **Diagrame comportamentale:** Se axează pe *cum* funcționează sistemul — procese, interacțiuni și fluxuri de activități.
2. **Reprezentarea statică vs. dinamică:**
    - **Diagrame structurale:** Prezintă o imagine statică a arhitecturii sistemului.
    - **Diagrame comportamentale:** Ilustrează comportamentul sistemului în timp și modul în care reacționează la stimuli externi.
3. **Relațiile și interacțiunile:**
    - **Diagrame structurale:** Arată relațiile logice și fizice între elementele sistemului.
    - **Diagrame comportamentale:** Detaliază interacțiunile și fluxurile de control și date între componente.
4. **Exemple de utilizare:**

- **Diagrame structurale:** Folosite în etapele inițiale de proiectare pentru a defini arhitectura sistemului.
- **Diagrame comportamentale:** Folosite pentru a descrie comportamentele specifice și scenariile de utilizare în detaliu.

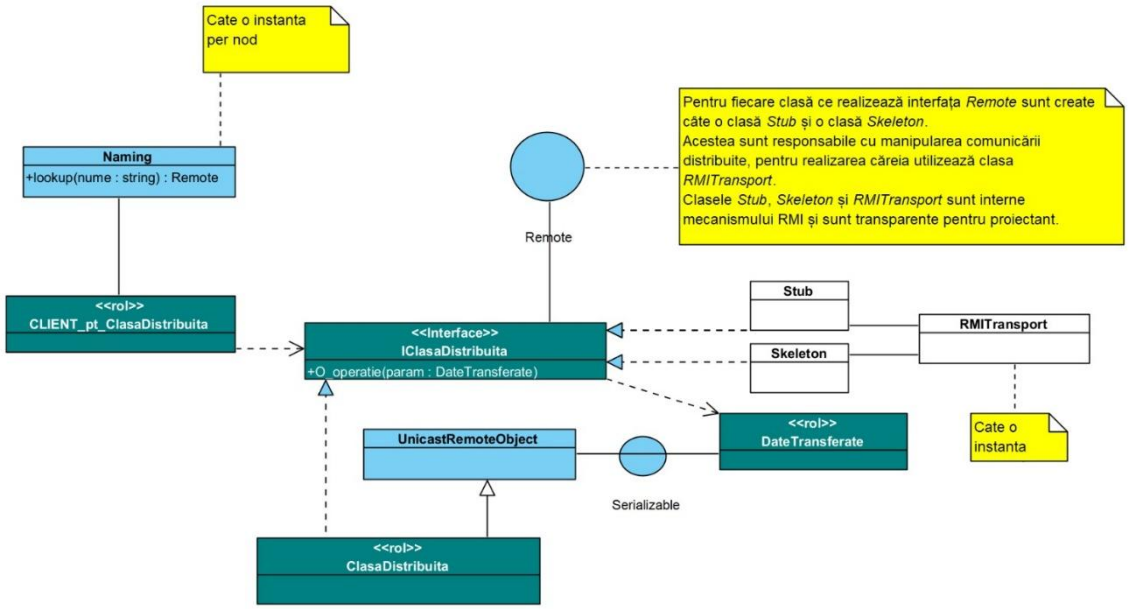
6. Caracterizați succint fiecare diagramele din [1] – Capitolul 2 (cele 5 menționate mai sus)

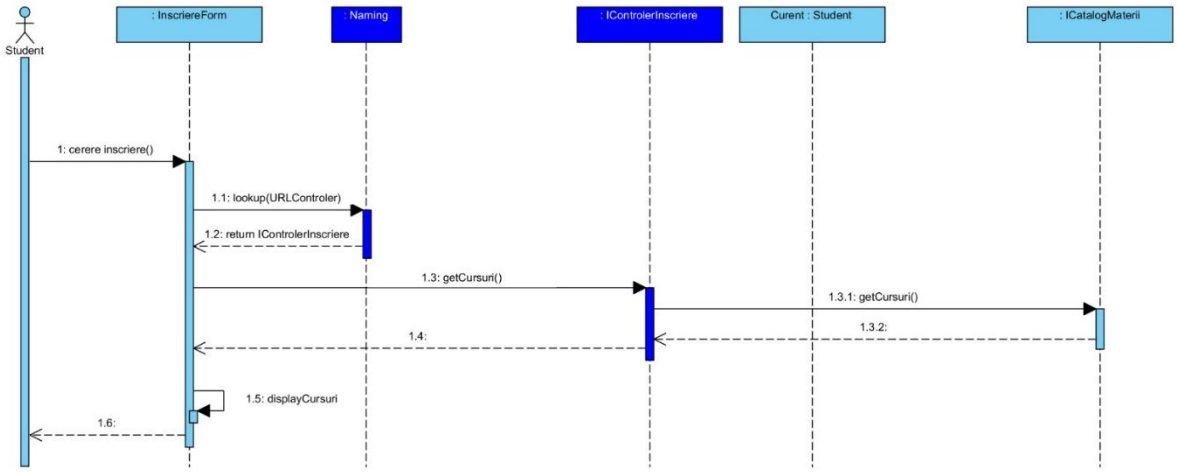
Am facut la 4.

7. Folosind Visual Paradigm, realizați diagramele de clasă:

• O_DiagramaDeClase.png

• O_DiagramaDeSecvente.png




Student
: InscriereForm
: Naming
: IControlerInscriere
Curent : Student
: ICatalogMaterii
1: cerere inscriere()
1.1: lookup(URLControler)
1.2: return IControlerInscriere
1.3: getCursuri()
1.3.1: getCursuri()
1.3.2:
1.4:
1.5: displayCursuri
1.6: