# Analiza și proiectarea sistemelor software

Curs 1

Site curs:
https://sites.google.com/view/apsswcm

wrl5y5l

- Fișa disciplinei
- Site curs : https://sites.google.com/view/apsswcm
- Inginerie software - laborator an II licență (pentru recapitulare)

https://sites.google.com/view/ingswcm/inginerie-software/laborator

- Recapitulare diagrame UML cunoscute

- Inregistrare studenti la Classroom APSSw : 6jmm7fw

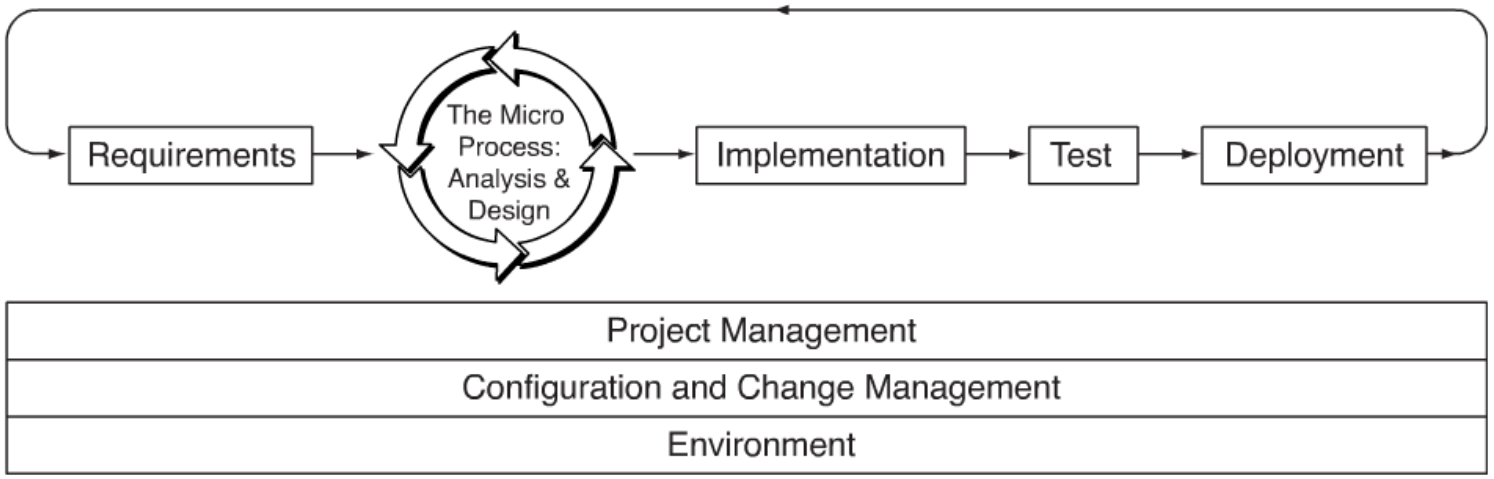
# BIBLIOGRAFIE

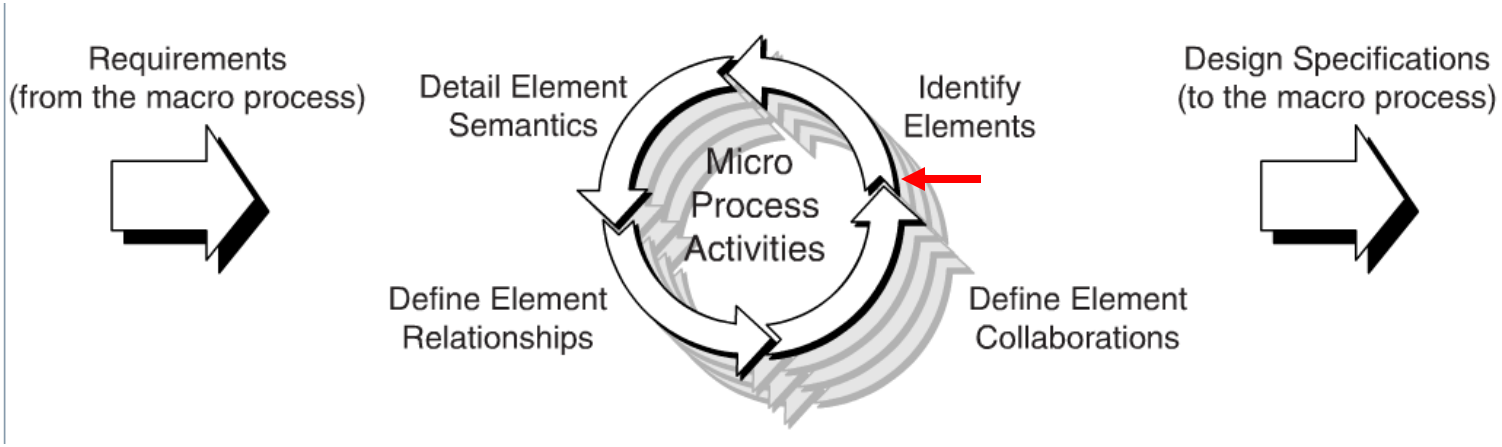
**Bibliografie –** format electronic disponibil la

https://drive.google.com/drive/folders/1Mcc8DfMyqbCaLHqHpyrJlEFiDOZjDlj-?usp=sharing


1. E.Gamma, R.Helm, R.Johnson, J.Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, ISBN 978-0201633610, Addison Wesley Professional, 1994, http://www.uml.org.cn/c++/pdf/DesignPatterns.pdf
2. Tom Pender, *UML Bible*, John Wiley & Sons, 2003
3. Roger S. Pressman, *Software Engineering. A Practitioner's Approach* ed.7, McGraw-Hill International Edition, 2010, cap. 6 – 14.
4. Ian Sommerville, "Software Engineering" 10-th Edition, Addison-Wesley, 2016, cap. 5,6,7,18,21.
5. http://swebokwiki.org/Chapter_2:_Software_Design, last modified in 24 August 2015
6. G.Booch et. all, *Object-Oriented Analysis and Design with Applications* 3rd Edition, Addison-Wesley Professional, 2007
7. Doug Rosenberg, Matt Stephens, *Use Case Driven Object Modeling with UML: Theory and Practice*, Apress, 2007
8. https://martinfowler.com/tags/domain%20driven%20design.html
9. Domain Driven Design Community : https://dddcommunity.org/
10. http://www.webml.org
11. https://sites.google.com/view/apsswcm

# CONTEXTUL




Booch G., Maksimchuk R.A., Engle M.W., Zoung B.J., Conallen J., Houston K.A, **Object-Oriented Analysis and Design with Applications, Third Edition,** Addison-Weslez Professional, 2007

# CONTEXTUL

ANALIZA:

Cerinţe → modelul analiză = reprezentări ale

- *datelor*
- *funcţiilor*
- *comportamentului*.

PROIECTAREA:

- Modelul analiză → modelul proiect = reprezentări ale
  - *Arhitecturii (date și procesare)*
  - *interfeţelor*
  - *componentelor software (date şi procesare)*.

# PLAN CURS

- Analiza sistemelor software: concepte şi abordări
- Modelare date
- Modelare funcții
- Modelare comportament
- Elemente de analiză structurată
- Elemente de analiză OO
- Principii practice ale modelării

# ANALIZA SISTEMELOR SOFTWARE: **CONCEPTE**

MODELUL ANALIZĂ (set de modele) = prima reprezentare tehnică a sistemului.

- Date
- Funcţii
- Comportament

Construit plecând de la *cerinţele* clientului – le reprezintă *din puncte de vedere diferite*.

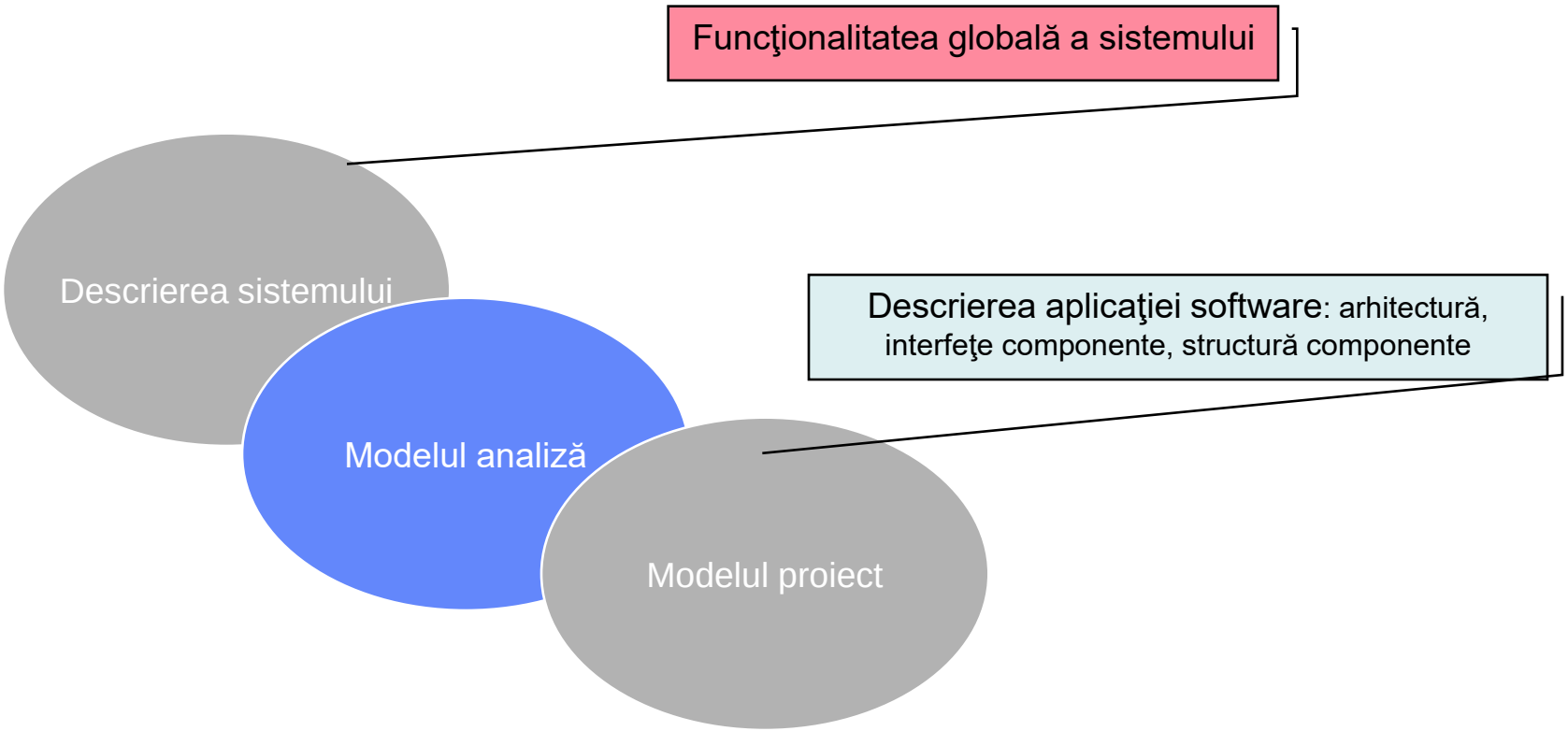
Simplu de înţeles şi de verificat în raport cu: *corectitudine*, *completitudine* şi *consistenţă*.

Rezultate:

- specificaţie cerinţe *clară* şi *completă,*
- reprezentare *detaliată* a modelului analiză al produsului software.

# ANALIZA SISTEMELOR SOFTWARE: **CONCEPTE**

## Modelul analiză între descrierea sistemului şi modelul proiect



# ANALIZA SISTEMELOR SOFTWARE: **CONCEPTE**

## Modelul analiză

CIM (Computation Independent Model)

- nivel superior de abstractizare pentru modelarea cerințelor sistemului software.

PIM (Platform Independent Model)

- descrierea sistemului independent de platforma pe care se va executa.

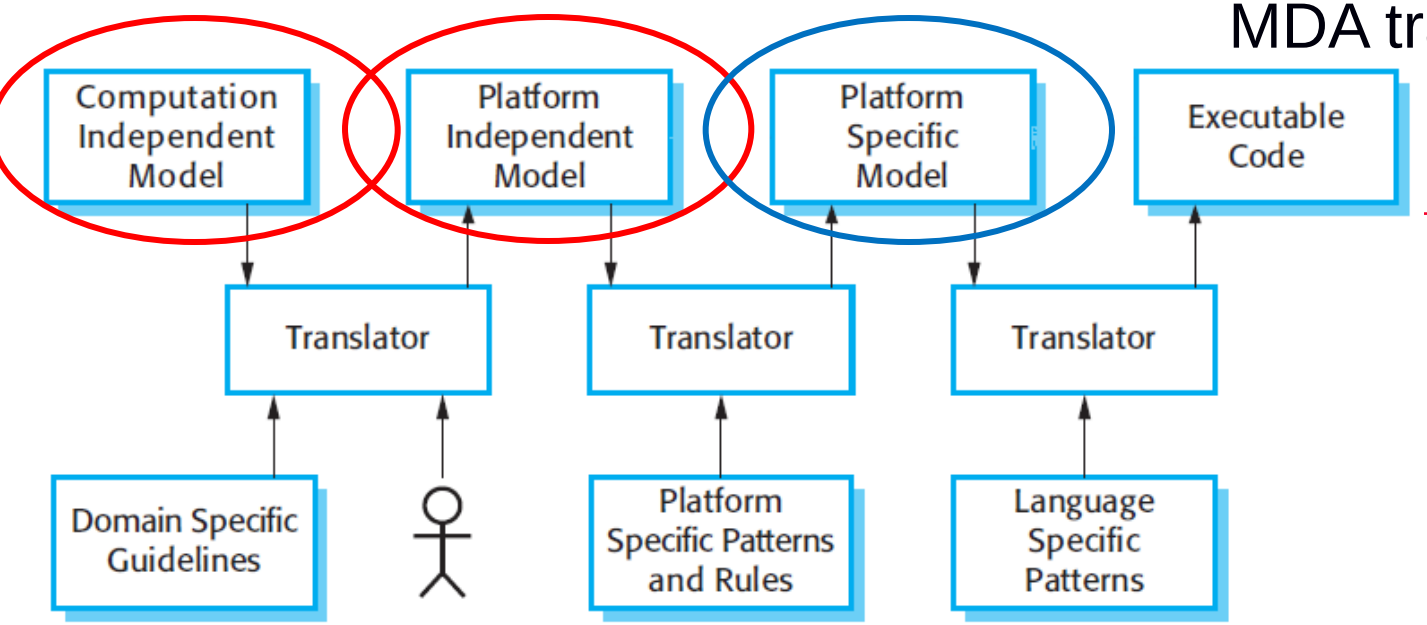
(technology free, domain specific)

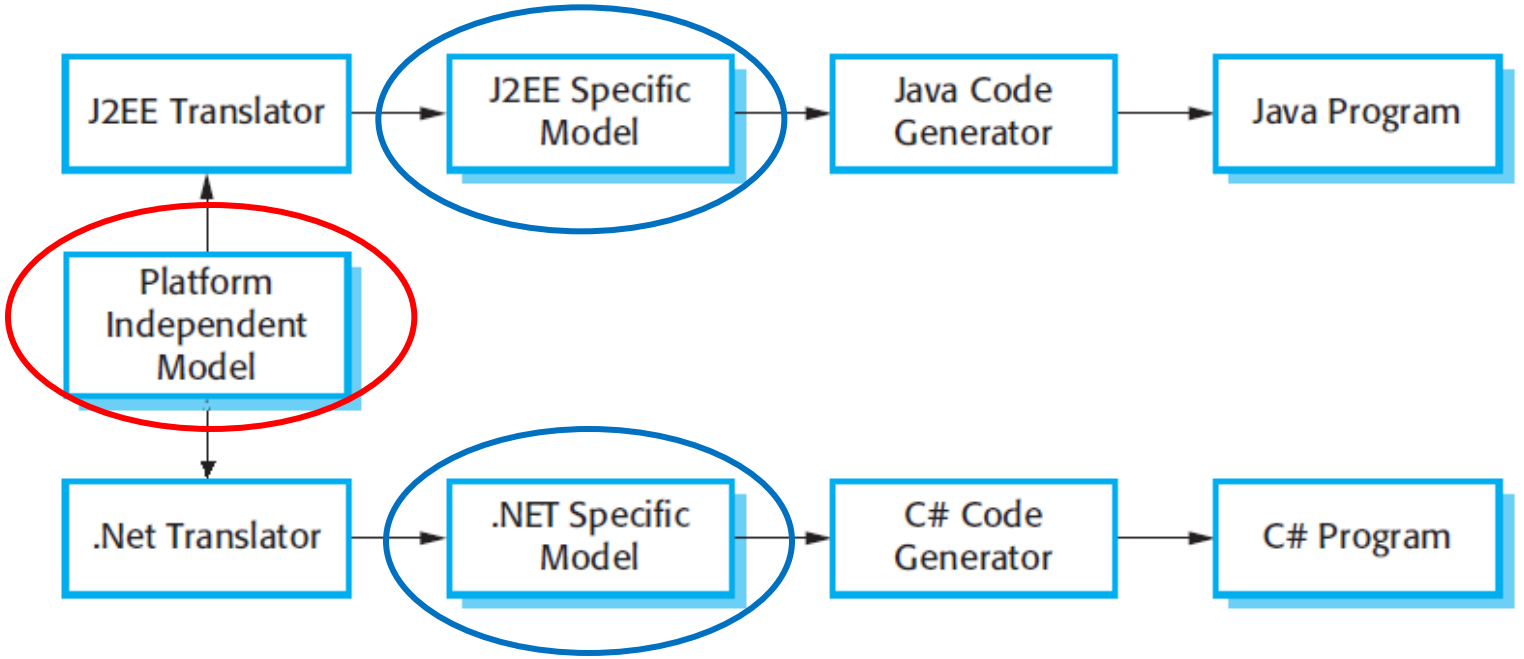
## Modelul proiect

PSM (Platform Specific Model)

- descrierea sistemului adaptat la platforma pe care se va executa

# MDA transformations



# ANALIZA SISTEMELOR SOFTWARE: **CONCEPTE**

## MODELUL ANALIZĂ - SFATUL EXPERTULUI

1. Nivel relativ ridicat de *abstractizare*: concentrarea pe cerinţele vizibile la nivelul problemei sau domeniului; fără detalii.
2. Fiecare element al modelului analiză trebuie să contribuie la *înţelegerea* ansamblului cerinţelor software şi să ofere *introspecţie* în domeniile *datelor* (informaţiilor), *funcţiilor* şi *comportamentului* sistemului.
3. *Delegarea* către proiectare a responsabilităţii, consideraţiilor de infrastructură şi a altor modele extra-funcţionale.
4. Minimizarea gradului de *cuplare* în reprezentările relaţiilor între clase şi între funcţii.
5. Modelul trebuie să poată oferi *valoare multiplă*: validare cerinţe, bază pentru proiectare, planificare teste de acceptare.
6. Păstrarea *simplităţii* modelului: eliminarea elementelelor care nu oferă informaţii noi; utilizarea de notaţii simple.

# ANALIZA SISTEMELOR SOFTWARE: **CONCEPTE**

ANALIZA DOMENIULUI:

Identificarea, analiza şi specificarea *cerinţelor comune* din cadrul unui anumit domeniu de aplicaţie.

Artefactele acesteia pot fi *reutilizate* în mai multe proiecte pentru domeniul respectiv.

**Scop:**

Identificarea şi crearea claselor şi a funcţiilor şi trăsăturilor *comune*, larg aplicabile, astfel încât să poată fi *reutilizate*.

Caz particular:

*Analiza OO* : identificarea, analiza şi specificarea capabilităţilor comune, reutilizabile, în cadrul unui anumit domeniu de aplicaţie, în termeni de *obiecte*, *clase*, *subansamble* şi *cadre* (frameworks).

## ANALIZA DOMENIULUI:

Intrări:

- literatura tehnică de specialitate
- aplicaţiile existente
- sondaje la client
- sfat expert
- cerinţe curente şi viitoare

Ieşiri:

- taxonomii de clase
- standarde pentru reutilizare
- modele funcţionale
- limbaje de domeniu (DSL – domain specific languages)

# ANALIZA SISTEMELOR SOFTWARE: **ABORDĂRI**

## ANALIZA STRUCTURATĂ

*Separare* date de procesele care le transformă.

Modelare *date* prin atribute şi relaţii.

Modelare *procese* prin transformările pe care le realizează asupra datelor.

## ANALIZĂ ORIENTATĂ OBIECT

Modelează *datele şi operaţiile* asupra lor sub formă de clase de obiecte.

În cadrul unei aplicaţii obiectele *colaborează* între ele.

- Abordări potenţial complementare.
- Pot fi utilizate combinat pentru obţinerea unui set optim de reprezentări.

# PLAN CURS

- Analiza sistemelor software: concepte şi abordări
- Modelare date
- Modelare funcții
- Modelare comportament
- Elemente de analiză structurată
- Elemente de analiză OO
- Principii practice ale modelării

# MODELAREA DATELOR

**Obiect de date** = reprezentarea unei *informaţii compuse*.

Informaţie compusă = are mai multe *proprietăţi* (atribute).

⇓

Obiect de date reprezentat în termenii unui *set de atribute*.

Obs. Conţine numai date !!!

Exemple:

- entitate externă ce produce sau consumă informaţii, produs (ex. raport, afişare), apariţie (ex. apel telefonic), eveniment (ex. alarmă), rol (ex. student), unitate organizaţională (ex. facultate), loc (ex. bibliotecă), structură (ex. fişier).

Reprezentare tabelară: (coloană – atribut), (rând – instanţă date).

↑

**Obiect de date**

# MODELAREA DATELOR

**Atribut** = defineşte o proprietate a unui obiect de date.

Tipuri de atribute:

- numire instanţă; include atributele de **indentificare** instanţă (componentele cheii primare)
- descriere instanţă
- referinţă la o altă instanţă

Obs. Nu există referinţe la operaţii !!!

Alegerea setului de atribute este determinată de contextul problemei.

# MODELAREA DATELOR

**Relaţie** = defineşte o conexiune relevantă între două obiecte de date.

Ex. Studenţi **asistă** Cursuri.

**Cardinalitate relaţie** = specificarea *numărului de instanţe* ale fiecărui obiect de date în raport cu relaţia.

**Modalitate relaţie** = obligativitatea existenţei relaţiei

0 – relaţie opţională

1 – relaţie obligatorie

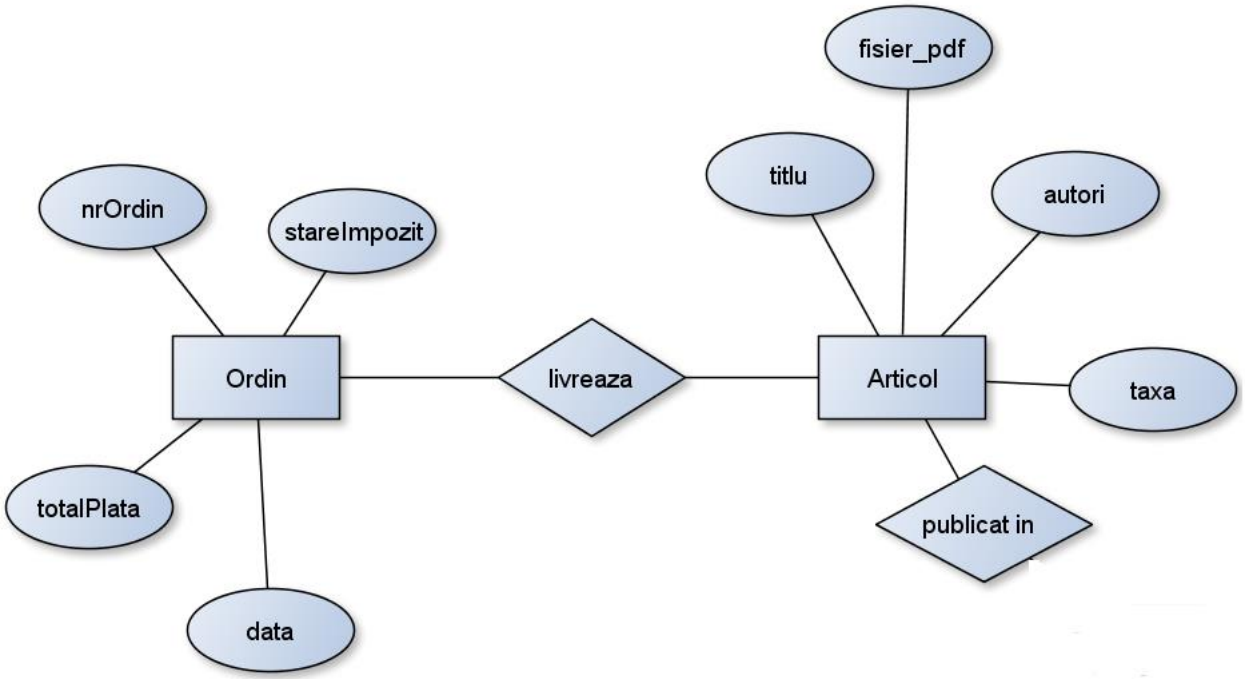
Reprezentare grafică:

- diagrame Entity-Relationship (ERD)
- diagrame de clase (fără operații)

# MODELAREA DATELOR

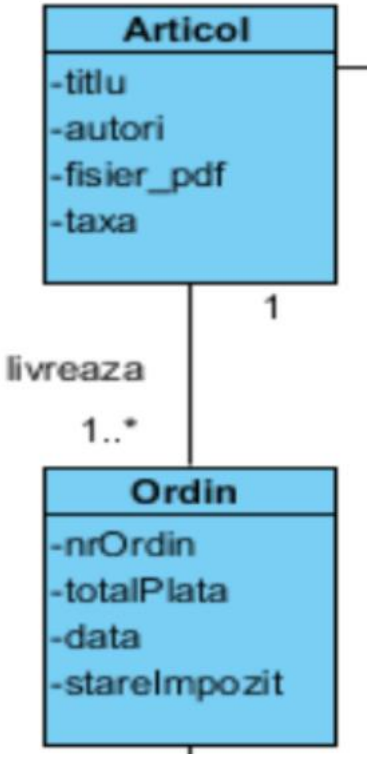
diagramă Entity-Relationship (ERD)

(extras)




diagramă de clase (fără operații)

(extras)



## Instrumente software.

Automatizarea creării de:

- diagrame ER sau diagrame de clase
- dicţionare de obiecte de date
- modele corelate cu acestea

Generează schema bazei de date.

Exemple:

Visual Paradigm, Oracle Designer, etc

(Sursa : Gemini

Online: DBDiagram.io, QuickDBD, Lucidchart, DrawSQL

Aplicații desktop: ERD Plus, Dia, Visual Paradigm, MySQL Workbench)

# MODELAREA DATELOR
## Evaluare formativă

Ce conține un obiect de date ?

Un obiect de date este definit printr-un set de _____.

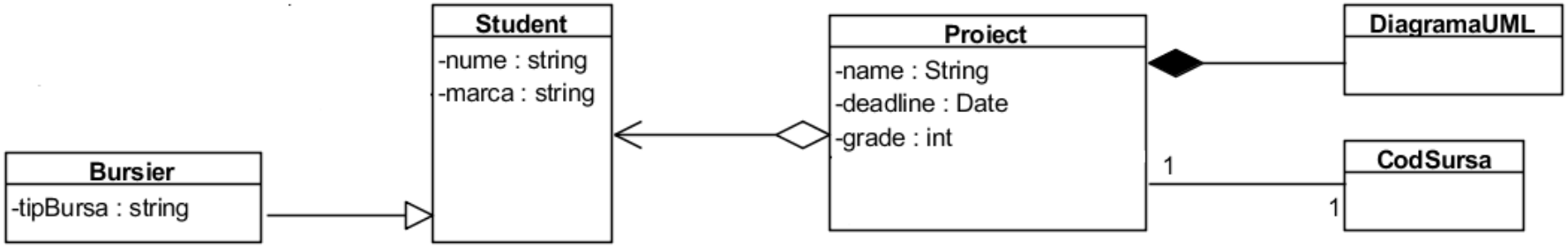
O conexiune relevantă între obiecte de date este definită printr-o _____.

Tipuri de atribute:

- numire instanţă; include atributele de **indentificare** instanţă (componentele cheii primare)
- descriere instanţă
- referinţă la o altă instanţă

Ce reprezintă un atribut de tip referință la o altă instanță ?

Exemplu.



# PLAN CURS

- Analiza sistemelor software: concepte şi abordări
- Modelare date
- Modelare funcții
- Modelare comportament
- Elemente de analiză structurată
- Elemente de analiză OO
- Principii practice ale modelării

# MODELARE BAZATĂ PE SCENARII

Se poate începe cu reprezentarea procesului business cu o *diagramă de activitate*.

Utilitate : identificare cazuri de utilizare.

DIAGRAMA *SWIMLANE*

Perspectivă organizată a diagramei de activitate prin gruparea activităţilor în ***partiţii***.

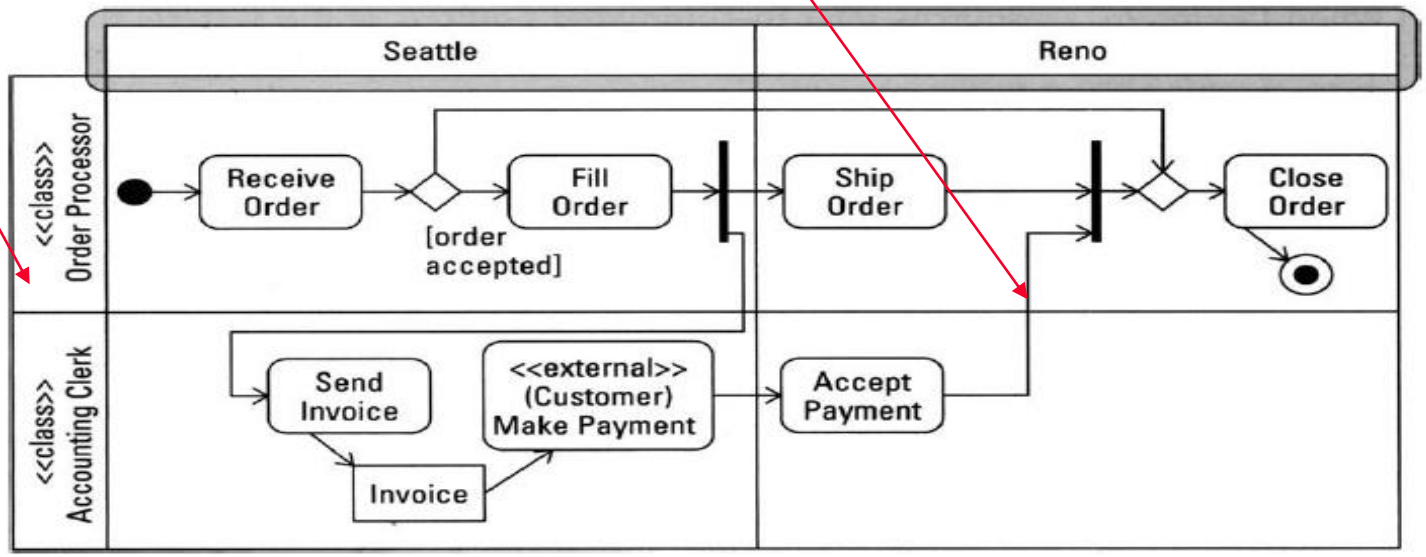
Utilizată, de obicei, pentru reprezentare procese business.

Criteriu tipic de partiţionare: Entitatea (actor, clasă, etc.) sau grupul de entităţi responsabile cu realizarea fiecărei activităţi.

Diagrama poate avea 1 sau 2 *dimensiuni*.

Se pot reprezenta *subpartiţii*.

Este vizibil transferul responsabilităţii.

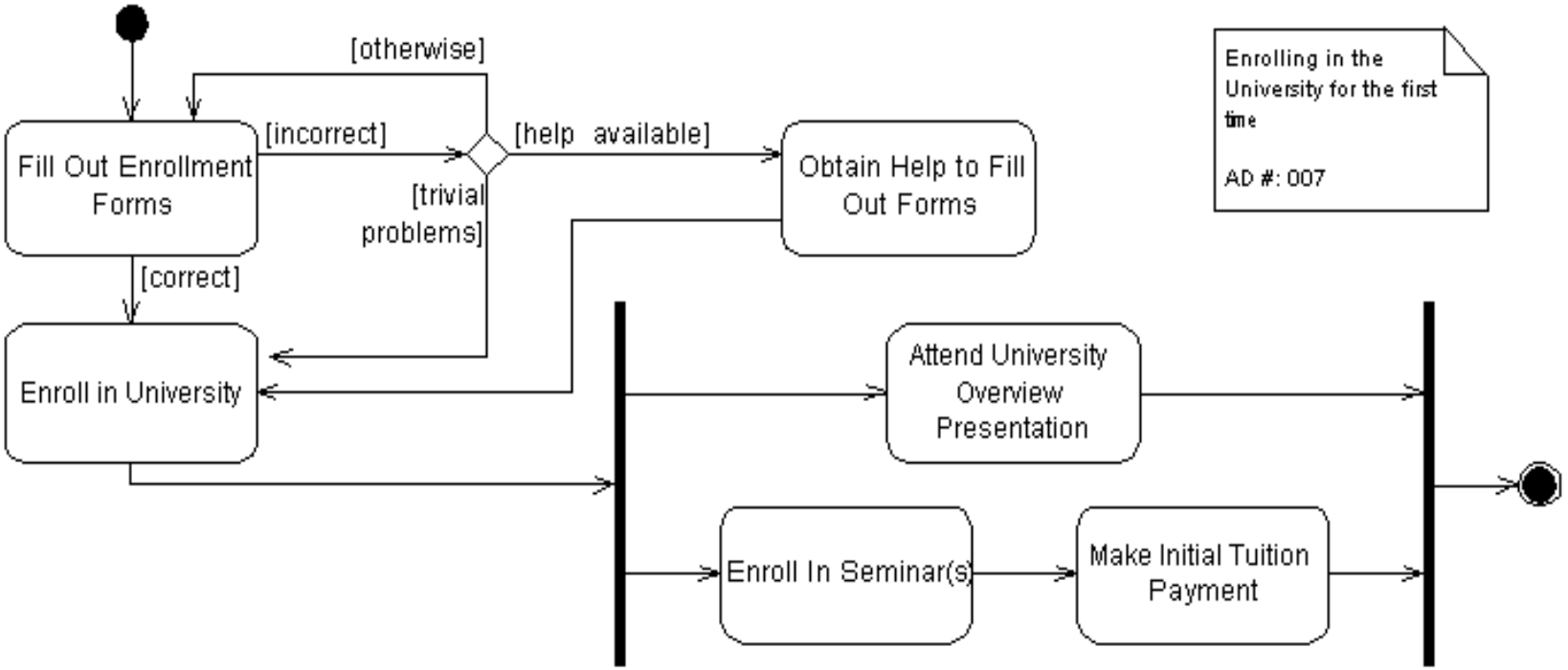


# MODELARE BAZATĂ PE SCENARII

Se poate începe cu reprezentarea procesului business cu o ***diagramă de activitate***.

Utilitate : identificare cazuri de utilizare.

Exemplu de modelare a unui proces business, din punct de vedere al unui singur actor, cu o diagramă de activitate:



# USE CASE

**Use Case:** Totalitatea modurilor (*utile*) de utilizare a unui sistem pentru a realiza un *anumit scop* pentru un *anumit utilizator*.

Setul de cazuri de utilizare ale unui sistem = toate modurile *utile* de a folosi sistemul, ilustrând *valoarea produsă de acesta*.

Clarifică ce FACE și ce NU FACE sistemul.

Use Case:

- O *secvență de acțiuni* realizate de sistem care conduce la un *rezultat observabil* de *valoare* pentru un anumit *utilizator.*
- *Comportament* specific al unui sistem care participă într-o *colaborare* cu un utilizator pentru a *livra ceva de valoare pentru acel utilizator*.
- Cea mai mică *unitate de activitate* care oferă un *rezultat semnificativ* utilizatorului.
- Context pentru un *set de cerințe corelate*.

# MODELARE BAZATĂ PE SCENARII

Etapele modelării bazată pe cazuri de utilizare:

- Reprezentarea diagramei de cazuri de utilizare
- Detalierea cazurilor de utilizare
  - Dezvoltarea diagramelor de secvenţe la nivel de sistem
  - (opțional) Dezvoltarea diagramei de activitate pentru fiecare caz de utilizare
- Realizarea prototipului UI
  - Ecranul de pornire
  - Pentru fiecare caz de utilizare – ecranele și fluxul acestora

# MODELARE BAZATĂ PE SCENARII

## CAZUL DE UTILIZARE

Interacţiunile producătorilor/consumatorilor de informaţii (*actori*) cu *sistemul*.

**Caz de utilizare** = descrierea unui anumit scenariu de utilizare din punct de vedere al unui actor (goal oriented).

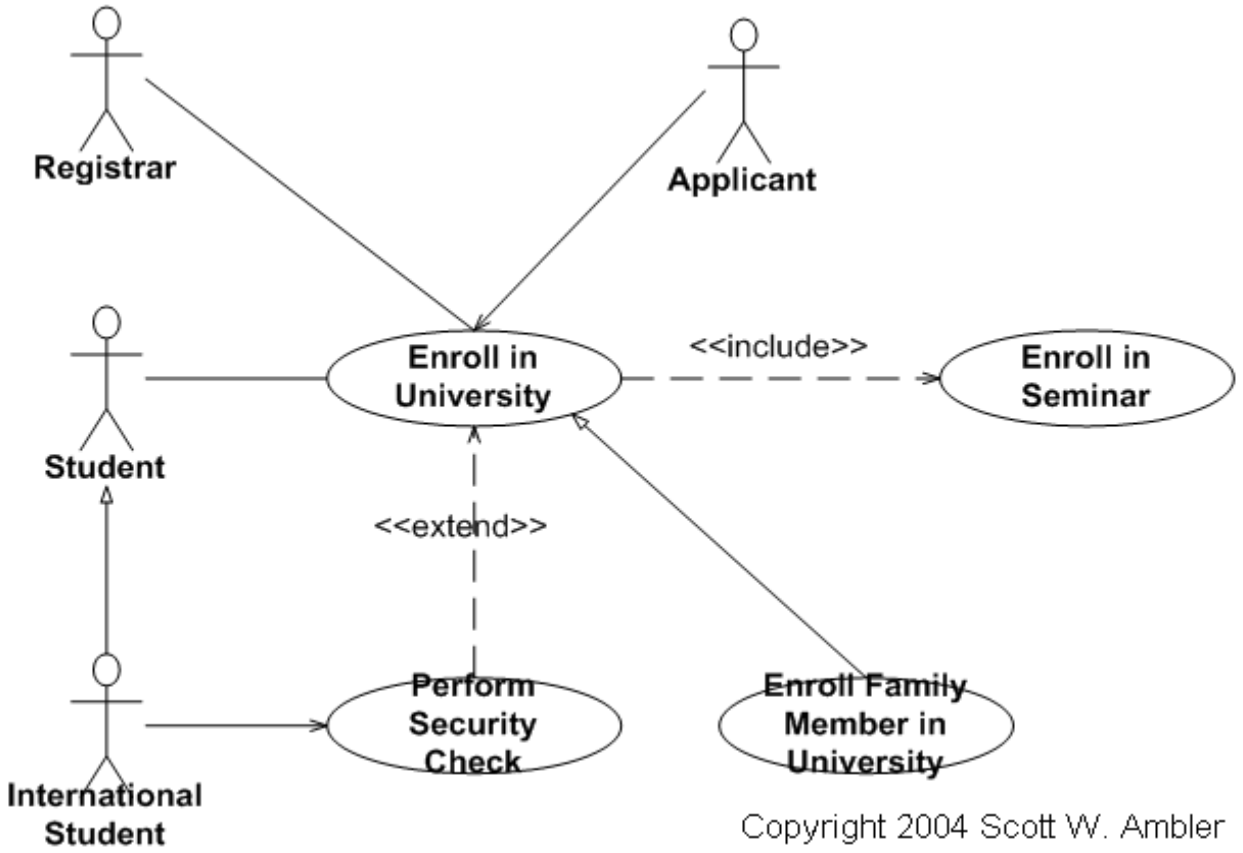
**Actor** = entitate (persoană/sistem/dispozitiv) externă sistemului, asociată unui rol jucat într-un context dat.

Actorul solicită un *serviciu* sistemului prin intermediul unei *interfeţe.*

# MODELARE BAZATĂ PE SCENARII

CAZUL DE UTILIZARE **- format de reprezentare**

- grafic (UML)



# MODELARE BAZATĂ PE SCENARII

CAZUL DE UTILIZARE **- detalii**

- narativ informal
- secvenţă ordonată de acţiuni ale utilizatorului
  - scenariul primar- scenariul de bază
  - scenarii alternative- situații speciale, tratare erori
- structurat, conform unui şablon

**Exemplu de şablon:**

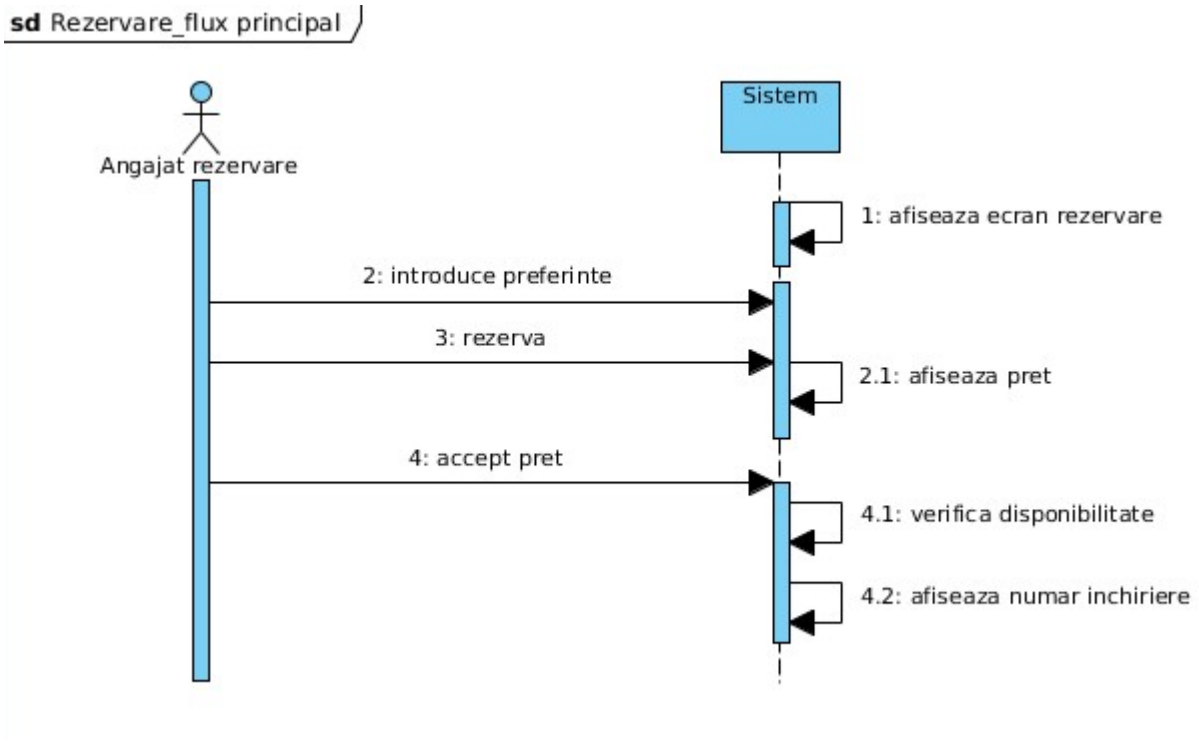
| | |
|---|---|
| **Nume caz de utilizare** | Prioritate |
| **Actor primar** | Când va fi disponibil în procesul de dezvoltare a aplicaţiei |
| Obiectiv (în context) | Frecvenţa de utilizare |
| **Precondiţii** | Calea de acces a actorului |
| Declanşator | Actori secundari |
| **Scenariu principal** | Căi de acces pentru actorii secundari |
| **Excepţii** | Probleme rămase deschise |
| **Postcondiții** | |

# MODELARE BAZATĂ PE SCENARII

## DIAGRAMA DE SECVENŢE LA NIVEL DE SISTEM

*Interacţiunea* actorului cu sistemul la nivelul *interfeţei* acestuia.

Exemplu:

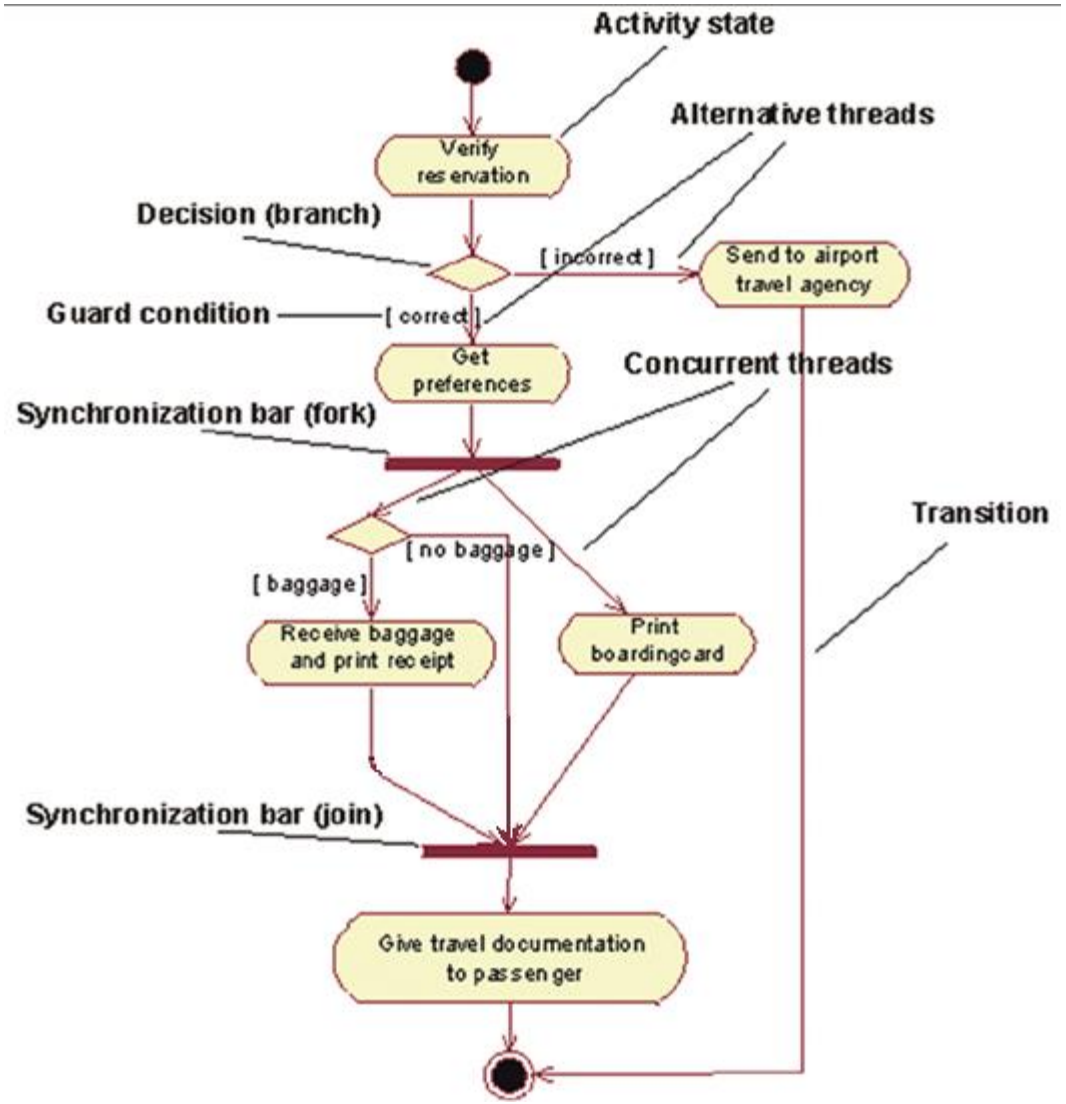


# MODELARE BAZATĂ PE SCENARII

## DIAGRAMA DE ACTIVITATE

Reprezentare grafică a *procesului* (*flux* de acţiuni şi activităţi) din cadrul unui scenariu.

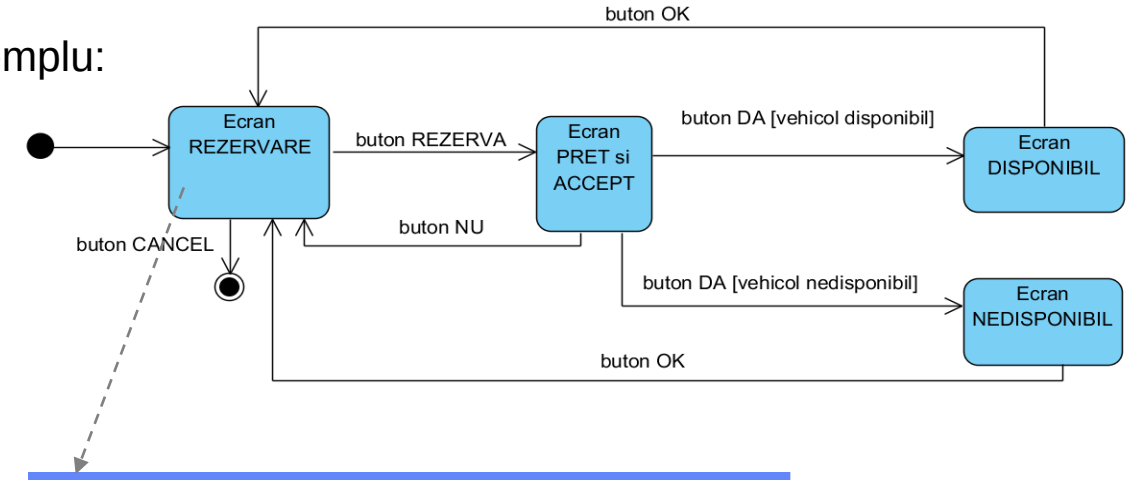
Echivalentul UML pentru diagramele fluxurilor de activităţi şi fluxurilor de date.



# MODELARE BAZATĂ PE SCENARII

## PROTOTIPUL INTERFEŢEI CU UTILIZATORUL (GUI)

Exemplu:





Flux ecrane

Se recomandă folosirea diagramei de stări și tranziții.

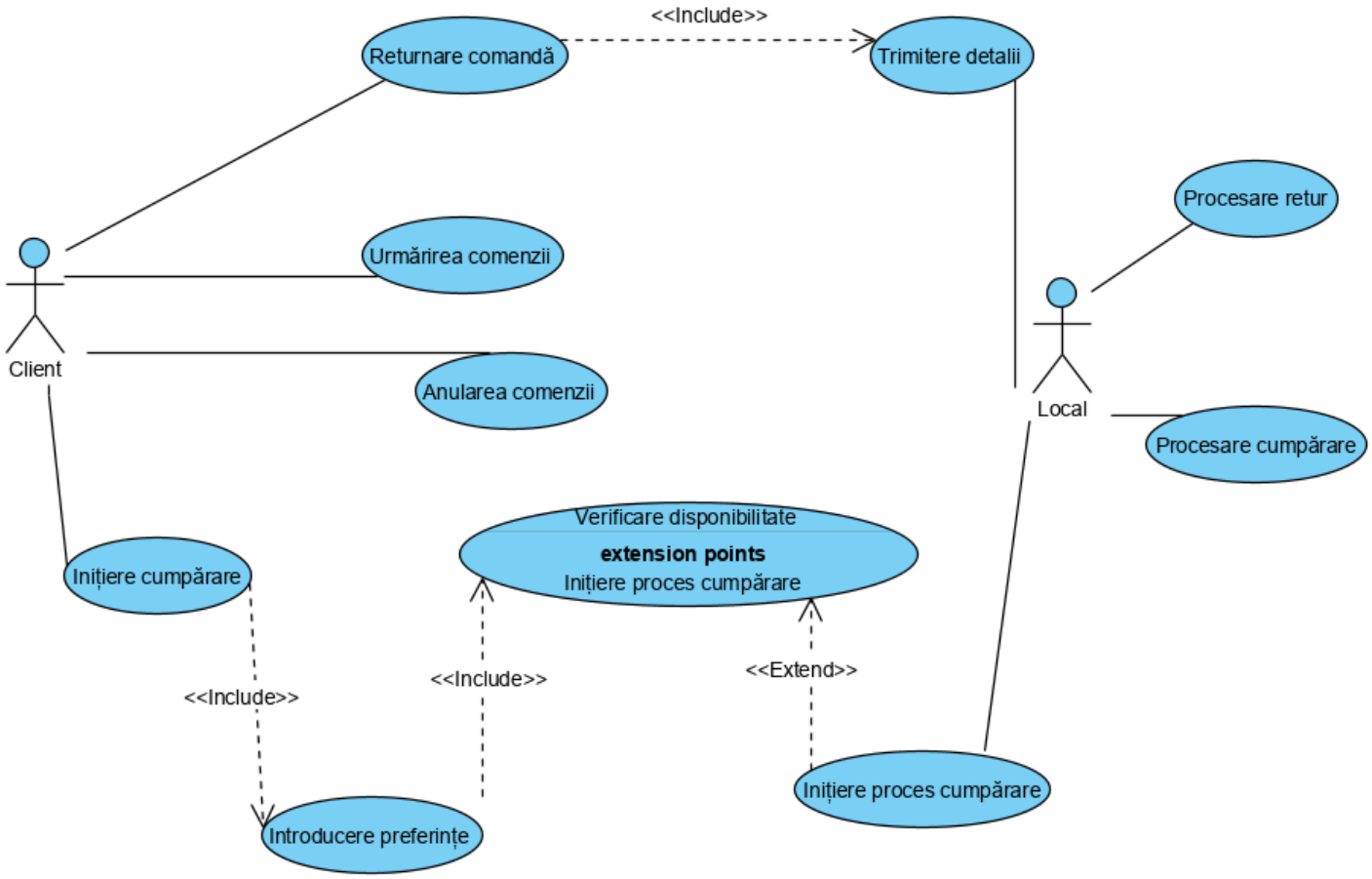
Conţinut ecran

# MODELARE BAZATĂ PE SCENARII - Evaluare formativă

Ce se reprezintă pe diagrama de activitate? Indicați variante de utilizare a acesteia?

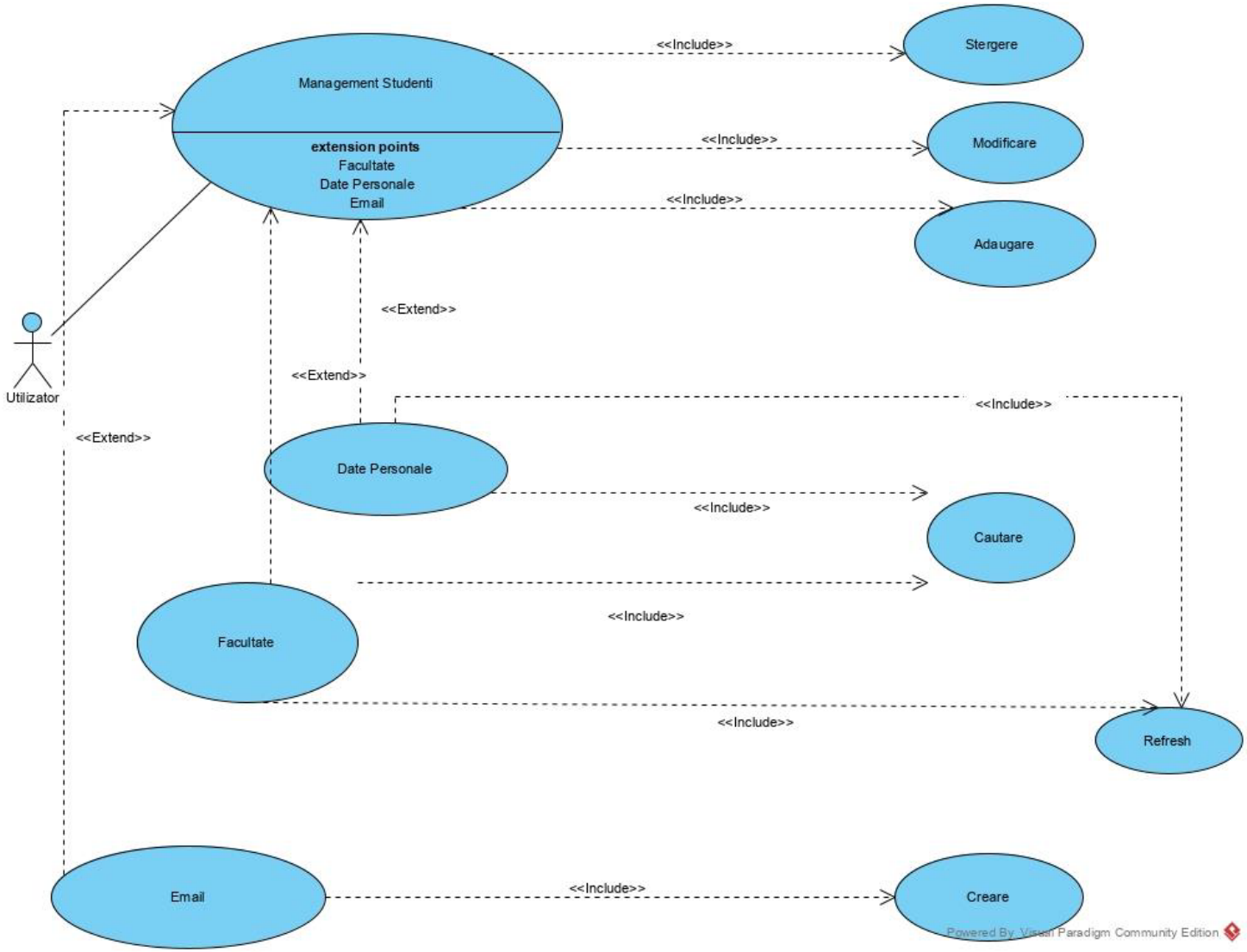
Ce se reprezintă pe diagrama cazurilor de utilizare?

Este corectă următoarea diagramă de cazuri de utilizare? De ce?



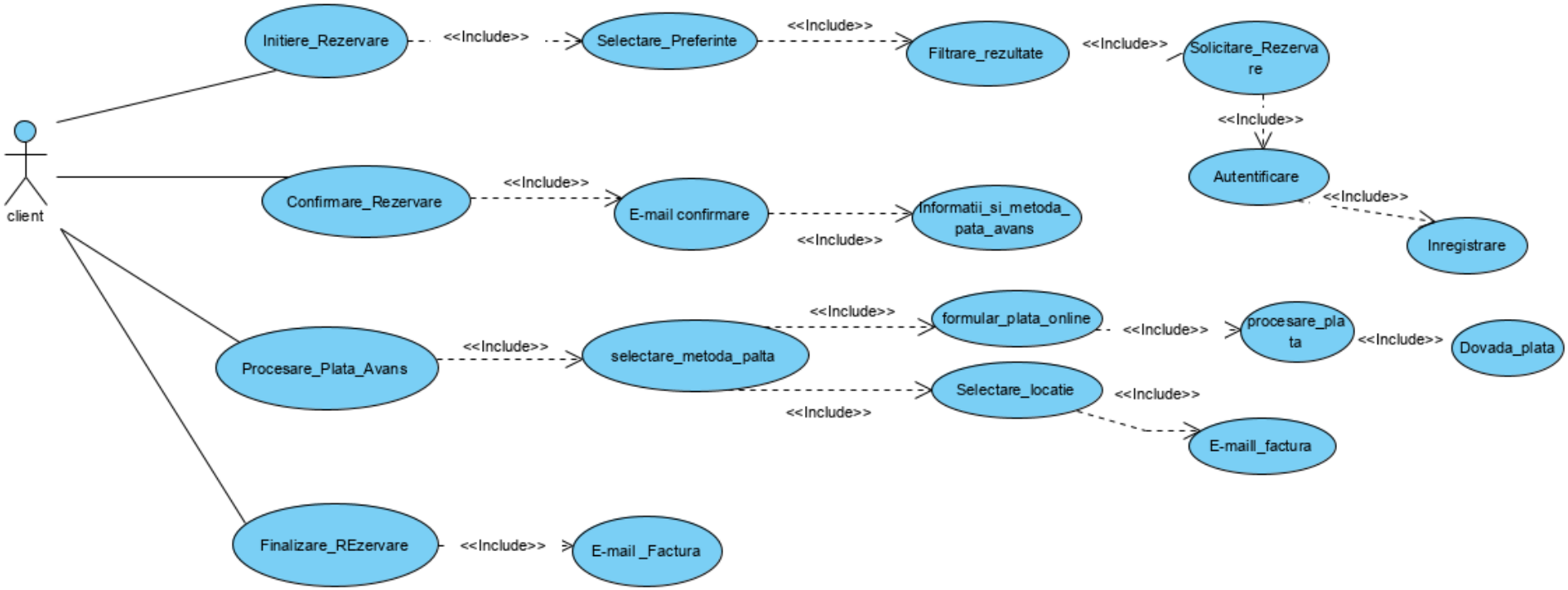
# MODELARE BAZATĂ PE SCENARII - Evaluare formativă

Este corectă următoarea diagramă de cazuri de utilizare? De ce?



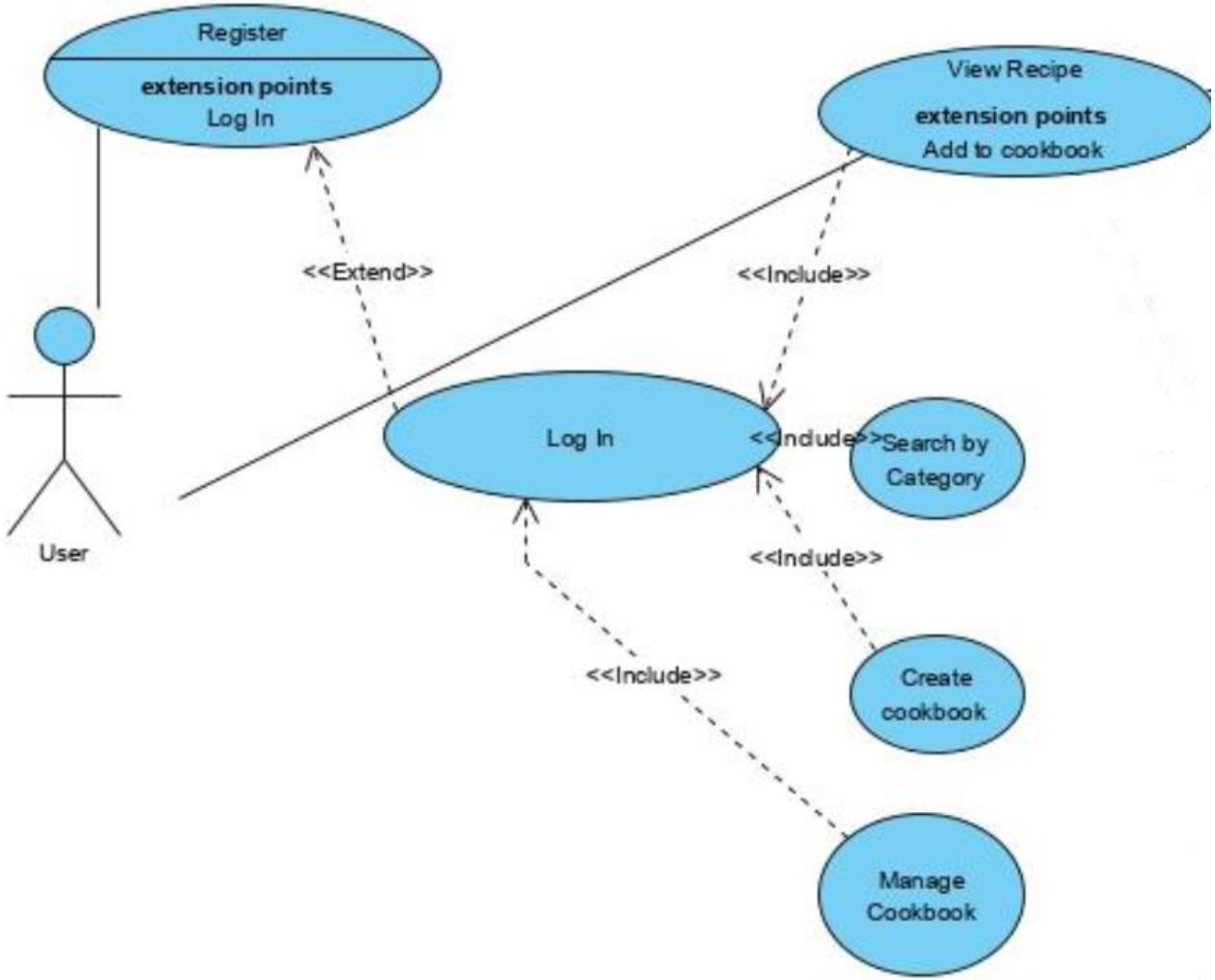
# MODELARE BAZATĂ PE SCENARII - Evaluare formativă

Este corectă următoarea diagramă de cazuri de utilizare? De ce?



# MODELARE BAZATĂ PE SCENARII - Evaluare formativă

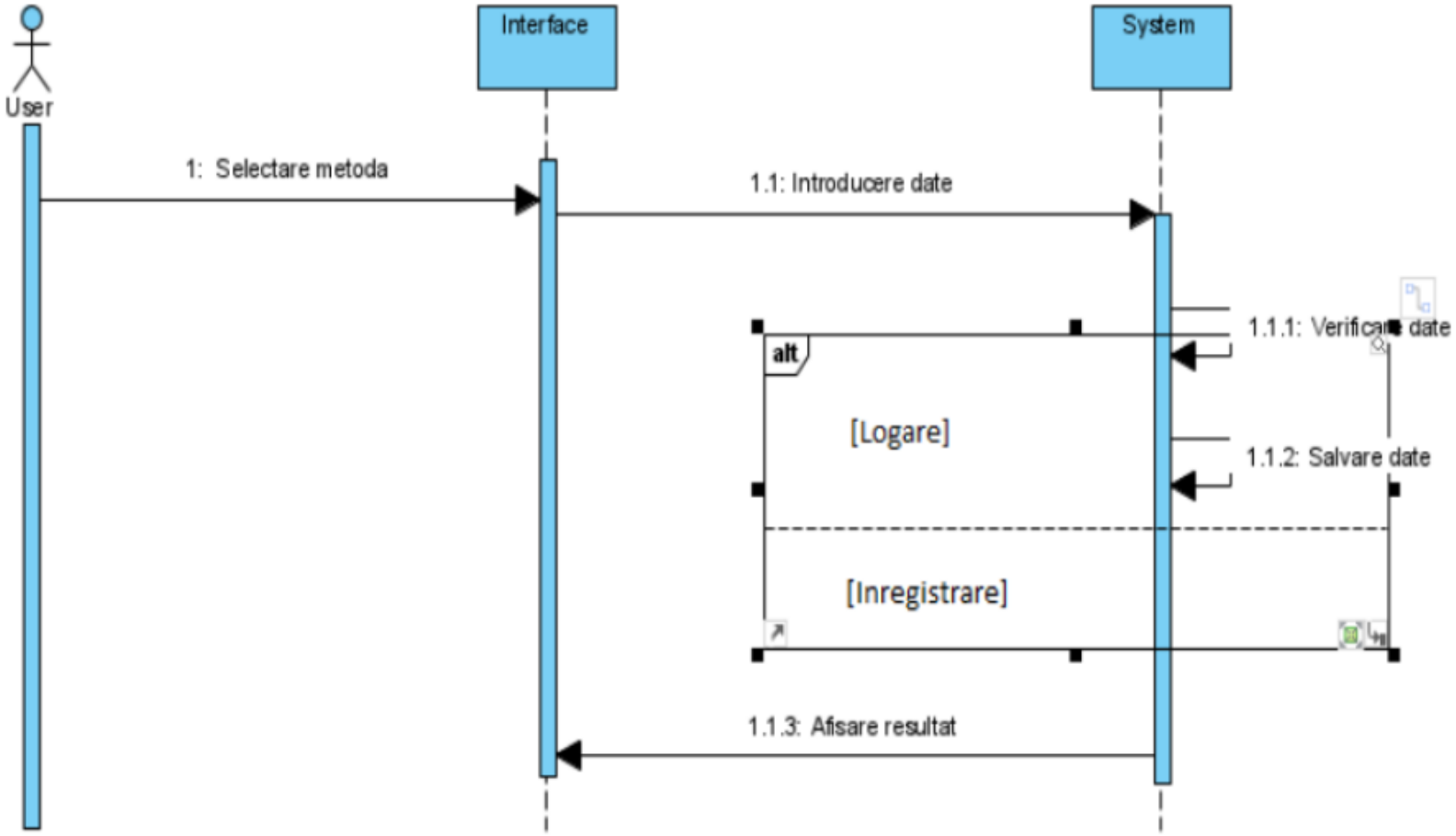
Este corectă următoarea diagramă de cazuri de utilizare? De ce?



# MODELARE BAZATĂ PE SCENARII - Evaluare formativă

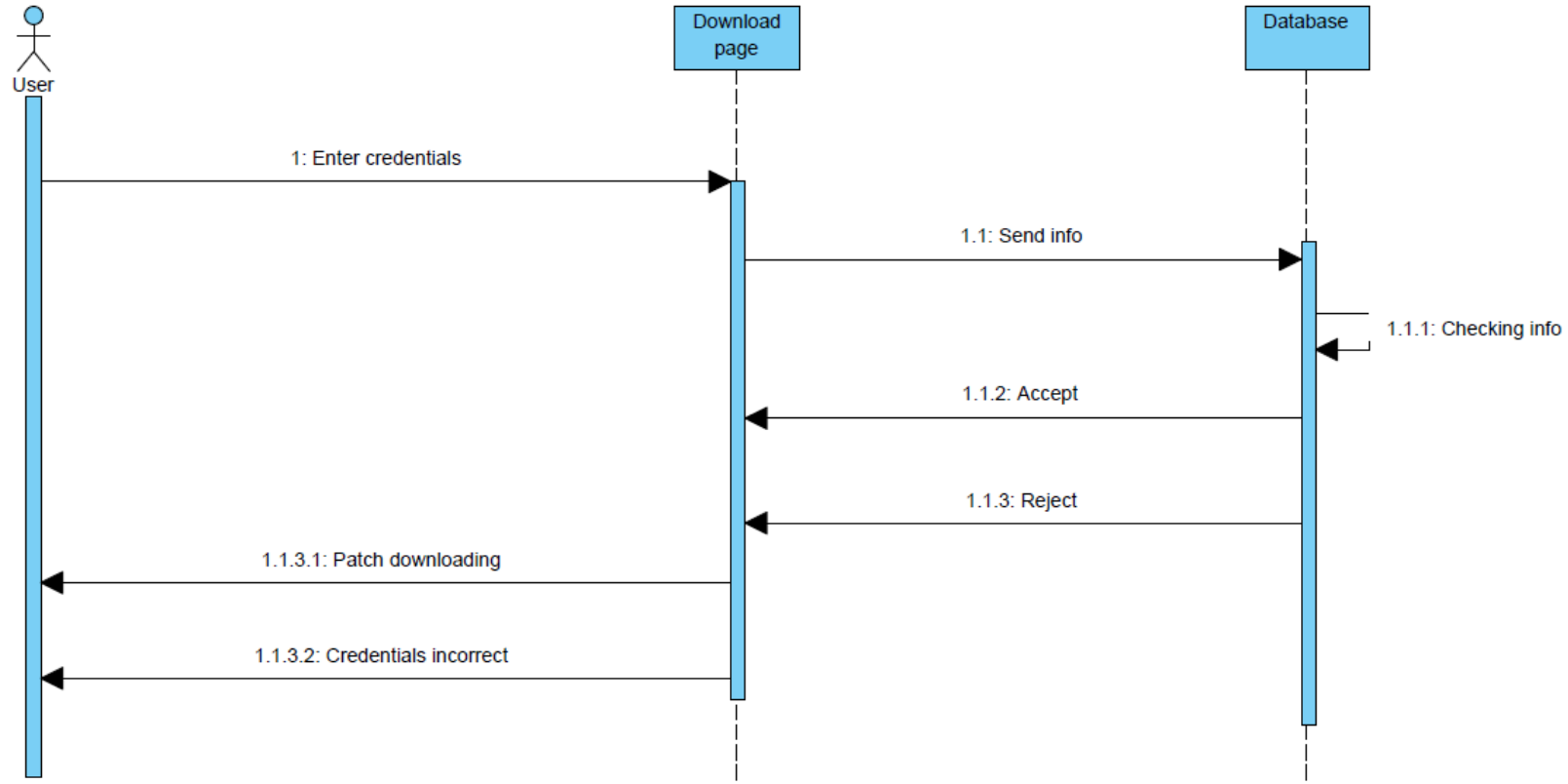
Ce se reprezintă pe diagrama de secvențe la nivel de sistem?

Este corectă următoarea diagramă de secvențe la nivel de sistem? De ce?



# MODELARE BAZATĂ PE SCENARII - Evaluare formativă

Este corectă următoarea diagramă de secvențe la nivel de sistem? De ce?



# MODELARE BAZATĂ PE SCENARII - Evaluare formativă

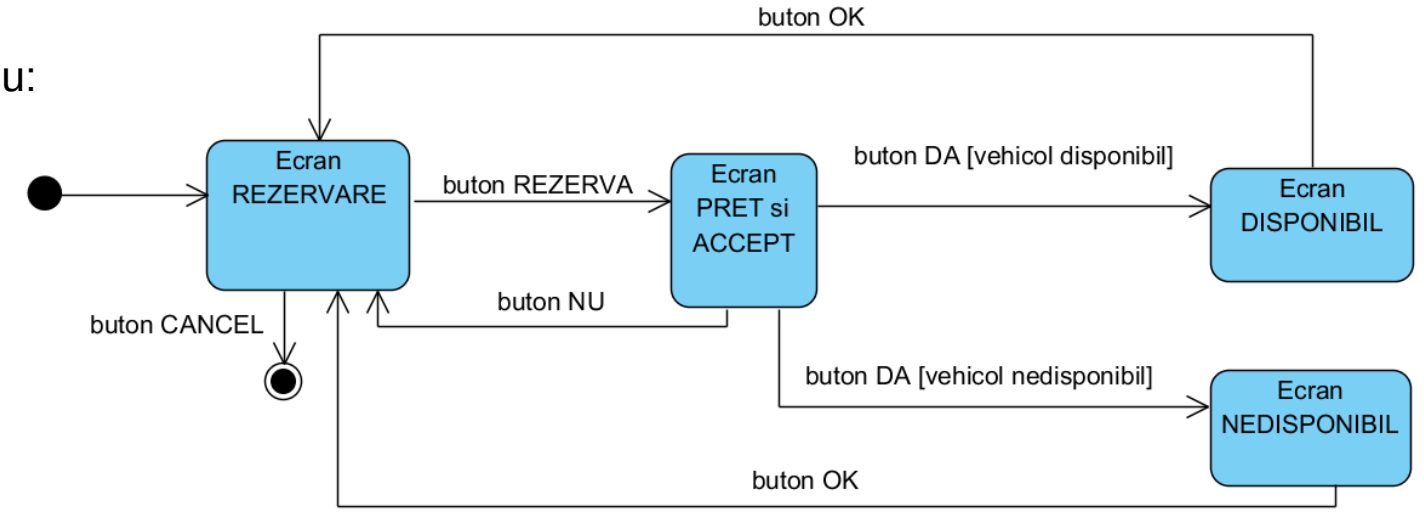
Ce conține prototipul interfeței grafice cu utilizatorul ?

Pentru reprezentarea separată a fluxului ecranelor se recomandă folosirea diagramei de stări și tranziții.

În acest caz :

- Ce se va reprezenta ca stare?
- Ce va reprezenta o tranziție între stări?
- Ce se va reprezenta ca eveniment ce declaşează o tranziţie de la o stare la alta?

Exemplu:



# PLAN CURS

- Analiza sistemelor software: concepte şi abordări
- Modelare date
- Modelare funcții
- Modelare comportament
- Elemente de analiză structurată
- Elemente de analiză OO
- Principii practice ale modelării

# MODELAREA COMPORTAMENTULUI

Reprezentare dinamicii, a *comportamentului* sistemului, ca funcţie de diferite *evenimente* şi de *timp*.

Descrie modul în care software-ul *răspunde* la evenimentele sau stimulii externi și interni.

Modelarea comportamentului se face pe nivele de detaliu:

1. Modelarea comportamentului sistemului în contextul său.
2. Modelarea comportamentului intern al sistemului.

# MODELAREA COMPORTAMENTULUI

PROCEDURĂ:

A. Modelarea comportamentului sistemului în contextul său

1. Evaluarea fiecărui *caz de utilizare* pentru înţelegerea completă a secvenţelor de interacţiuni ale sistemului cu contextul său.
   a. Identificare *evenimentelor* care declanşează secvenţe de interacţiuni şi a relaţiei acestora cu sistemul.
   b. Crearea *diagramei de secvenţe la nivel de sistem* (interacțiuni cu contextul și acțiuni interne) la nivelul sistemului, pentru fiecare caz de utilizare.
2. Crearea *diagramei de stări și tranziții* la nivelul sistemului.

B. Modelarea comportamentului intern al sistemului.

1. Identificare *obiectelor* implicate în fiecare caz de utilizare.
2. Crearea *secvenţei de interacţiuni* între obiectele interne pentru fiecare caz de utilizare.
3. Creare *diagrame de stări și tranziții* pentru fiecare obiect cu comportament relevant în sistem.

C. *Verificarea* acurateţei şi consistenţei modelului.

# MODELAREA COMPORTAMENTULUI
# *IDENTIFICARE* şi *ALOCARE* **EVENIMENTE**

**Caz de utilizare** – funcționalitate sistem obținută cu o secvenţă de activităţi care implică actorul şi sistemul.

**Eveniment** – orice schimbarea de informaţii între actor şi sistem.

PROCEDURĂ:

1. Identificare:
   - actor (*orice* entitate externă aflată în interacțiune cu sistemul)
   - informaţie ce trebuie transferată
   - condiţii
   - constrângeri

2. Identificare evenimente.

3. Alocare evenimente identificate la obiectele implicate care fie generează fie recunosc evenimente.

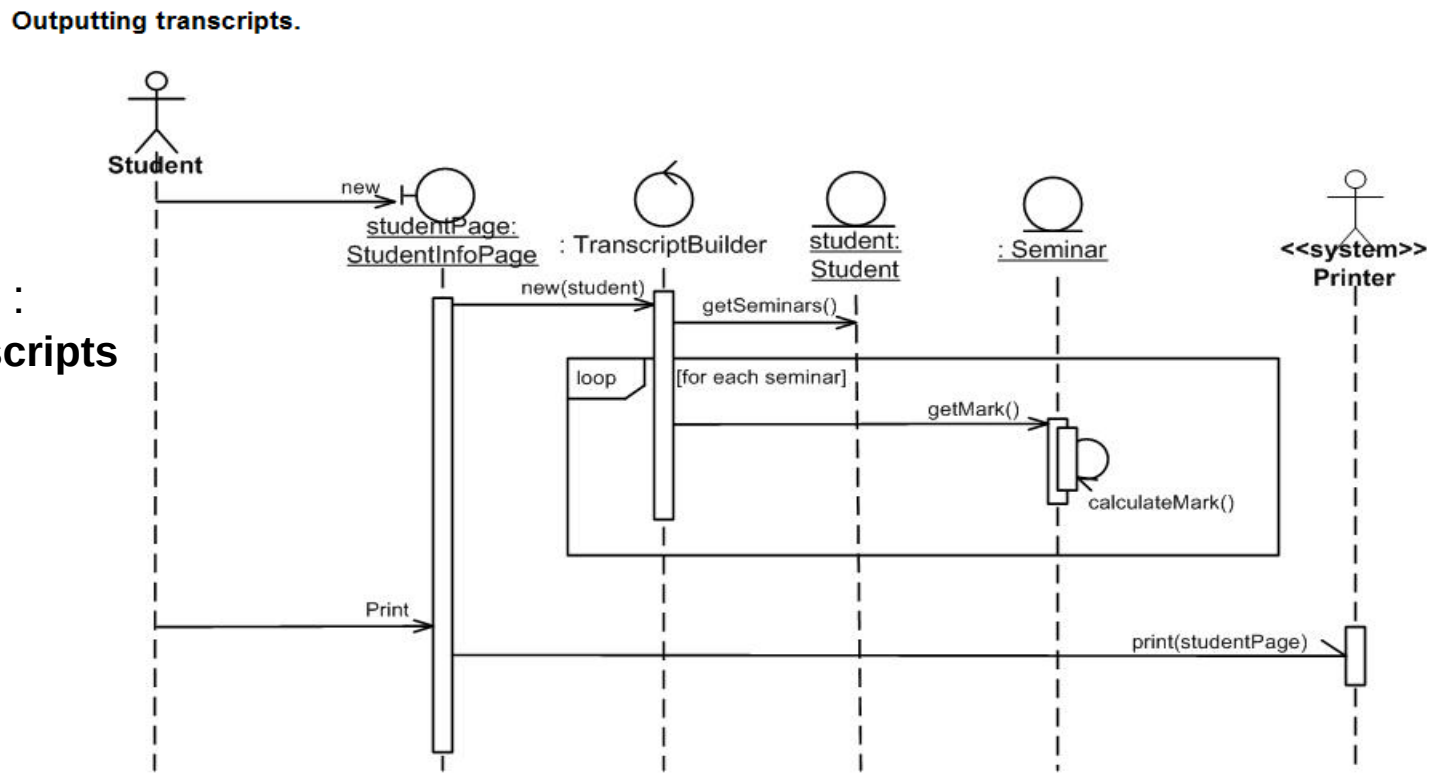
# MODELAREA COMPORTAMENTULUI
# REPREZENTAREA **INTERACŢIUNILOR**

DIAGRAMA DE SECVENŢE: fluxul de *mesaje* de la un obiect la altul, reprezentat ca funcţie de *timp*.

**Utilizare**: determinarea tuturor evenimentelor de intrare şi de ieşire ale fiecărui obiect.

Exemplu.

Caz de utilizare :
**Outputting transcripts**



# MODELAREA COMPORTAMENTULUI
# REPREZENTĂRILE **STĂRILOR**

- Stările sistemului observabile din exterior

- Stările fiecărui obiect în cursul operării sistemului

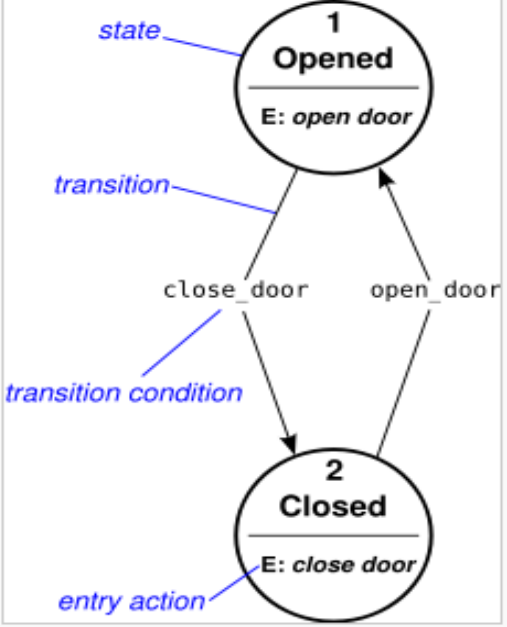
Categorii de stări ale unui *obiect*:

- Stare *pasivă* – valorile curente ale atributelor.
- Stare *activă* – starea curentă în contextul unui proces; rezultat al unei secvențe de tranziţii declanşate evenimente (trigger-e).

# MODELAREA COMPORTAMENTULUI
# REPREZENTAREA **STĂRILOR**

DIAGRAMA DE STĂRI (State Machine Diagram):

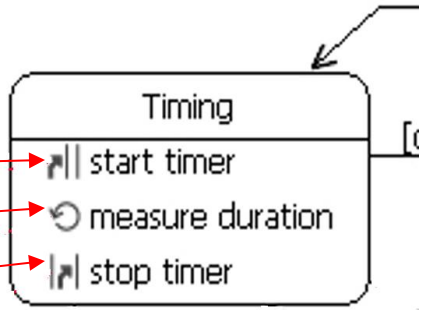
reprezentarea *stărilor active*, a *tranziţiilor* între acestea şi a *evenimentelor* ce declaşează tranziţiile de la o stare la alta.



A State Diagram for a door that can only be opened and closed

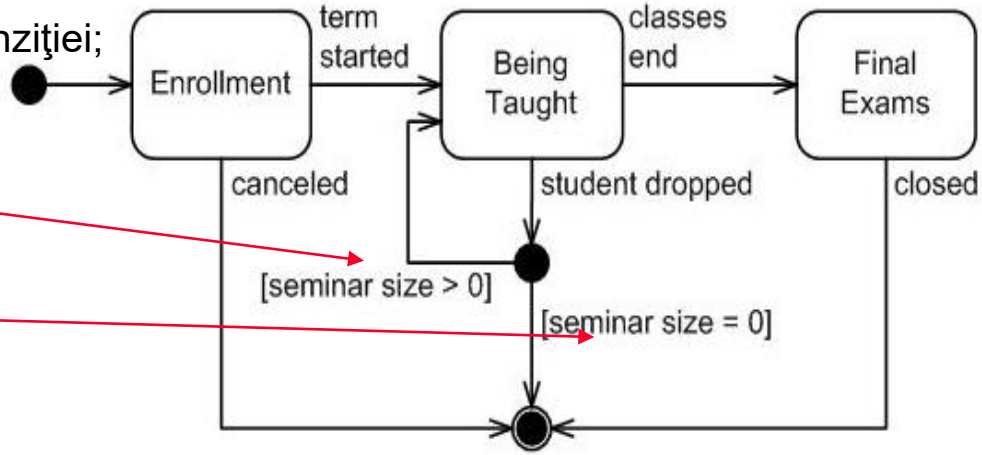
Pe fiecare stare se pot adăuga :

- activitate '*entry*'
- activitate 'do'
- activitate '*exit*'



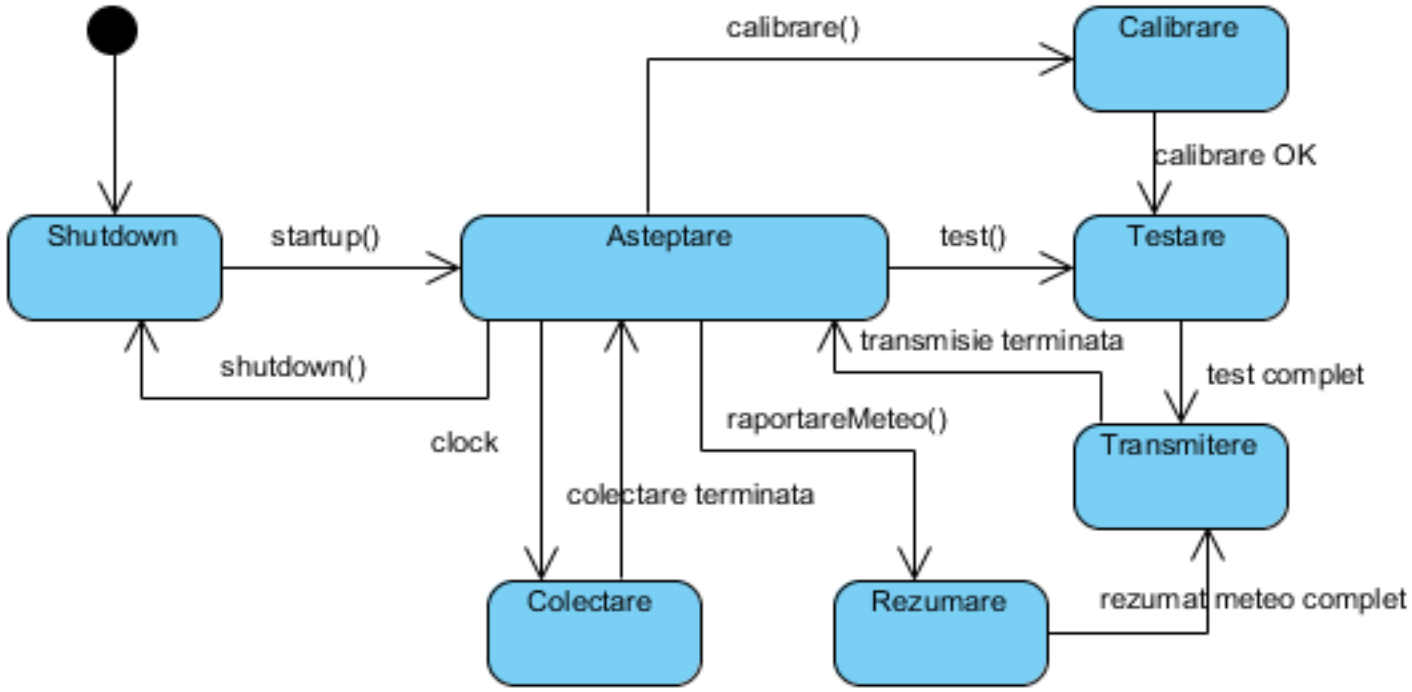

Pe fiecare tranziţie se pot adăuga:

- *gardă* : condiţie de declanşare a tranziţiei; dependentă (în general) de starea pasivă a obiectului (valori ale atributelor)
- *acţiune* : realizată concurent cu tranziţia; implică una sau mai multe operaţii cu obiectul.



# MODELAREA COMPORTAMENTULUI
# REPREZENTAREA **STĂRILOR**

Exemplu de diagramă de stări completată cu descrieri tabelare



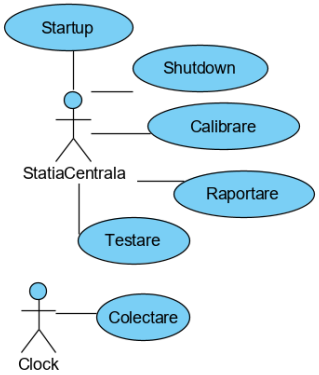

| Stare | Descriere |
|---|---|
| Colectare | Sistemul colectează date de la instrumentele din teren. |
| . . . | |

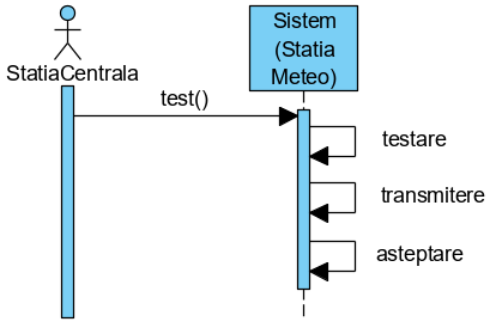
| Stimul | Descriere |
|---|---|
| calibrare( ) | Sistemul client solicită operaţia de calibrare a instrumentelor. |
| . . . | |

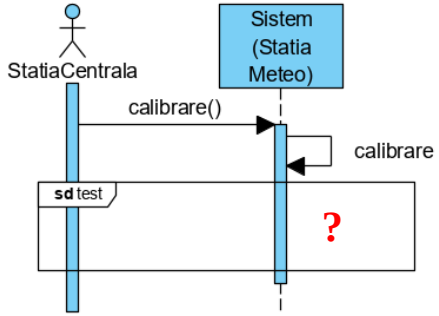
# Exemplu : MODELAREA COMPORTAMENTULUI la nivelul SISTEMULUI

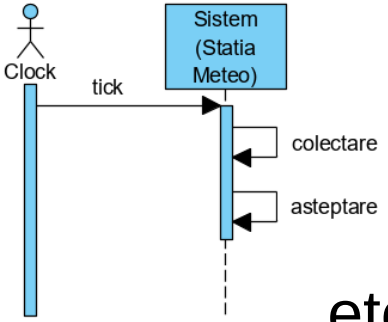
## Exemplu: Stație meteo

Crearea *diagramei de secvenţe la nivel de sistem*, pentru fiecare caz de utilizare.



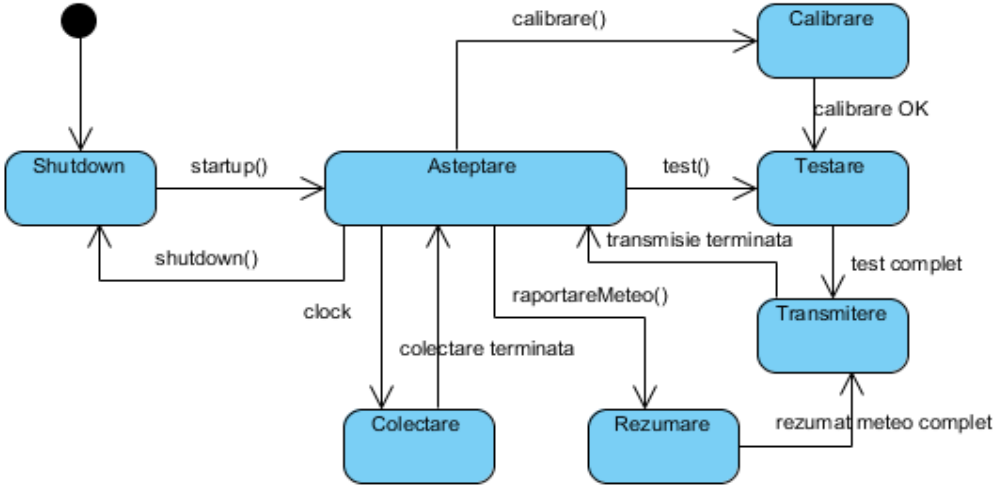




etc.

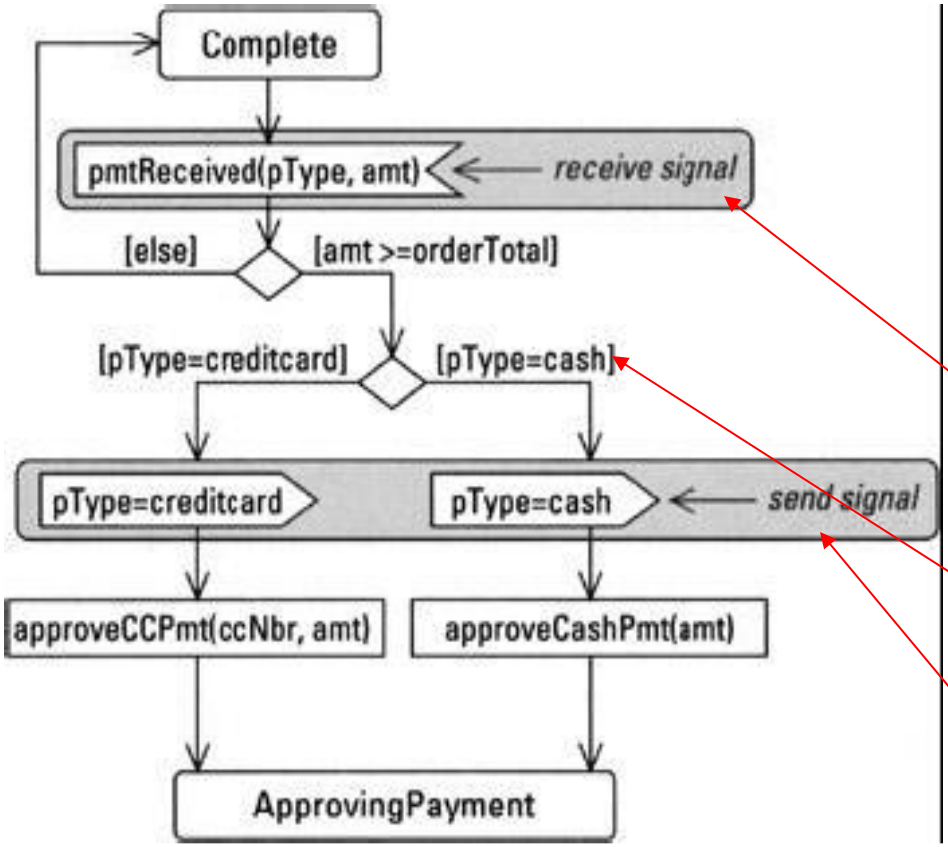
Crearea *diagramei de stări și tranziții* la nivelul sistemului.



# MODELAREA COMPORTAMENTULUI

## Exemplu de utilizare diagramă de activitate

Reprezentare pe bază de model logic




# Include:

Acţiuni

Decizii

Emitere şi recepţionare de evenimente

**Acţiuni** - asociate executării unei tranziţii:

1. Recepţionarea valorilor de la evenimentul ce invocă tranziţia (ex. amt)
2. Manipularea/evaluarea acestor valori
3. Transmiterea valorilor ca intrare pentru acţiunile următoare (ex.pType)

# MODELARE COMPORTAMENT - Evaluare formativă

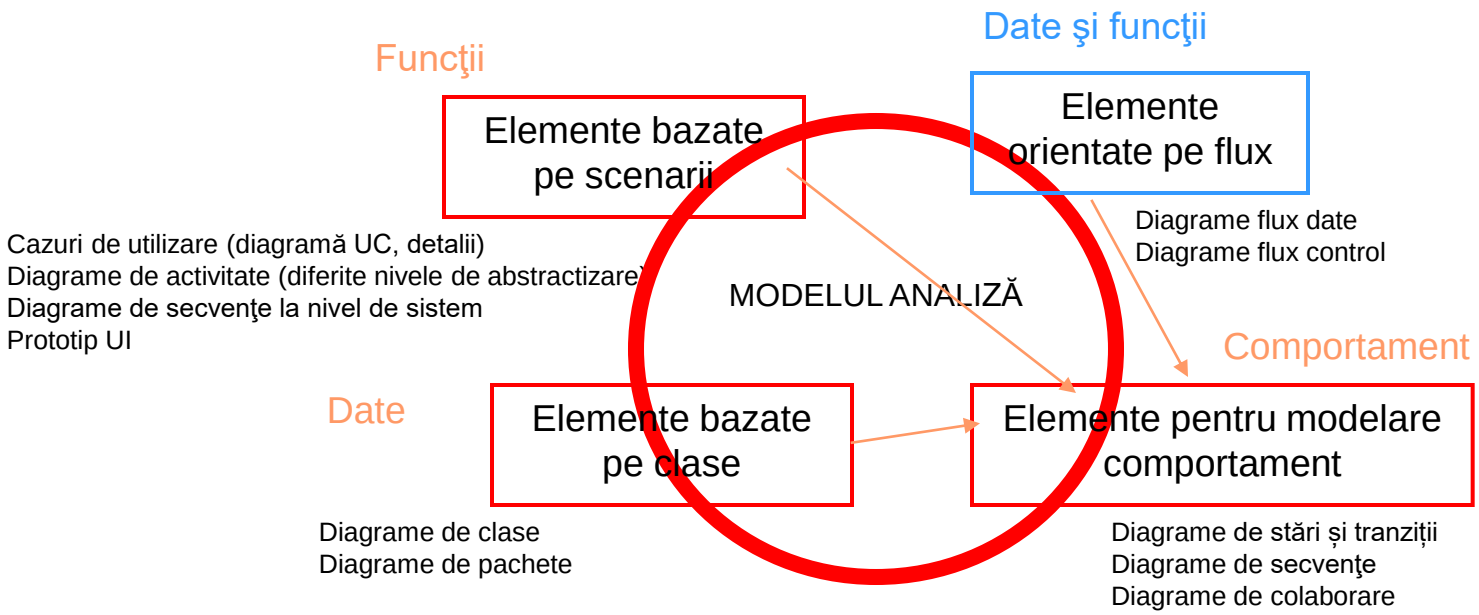
1. Ce diagramă UML se folosește pentru modelarea interacțiunilor dintre obiectele sistemului software?

2. Ce diagramă UML se folosește pentru modelarea comportamentului generat de evenimente interne și externe al sistemului software?

3. Pe ce nivele de detaliu se realizează modelarea comportamentului?

https://forms.gle/8BZpQjd3gE6aP4UYA

# ELEMENTELE MODELULUI ANALIZĂ



# PLAN CURS

- Analiza sistemelor software: concepte şi abordări
- Modelare date
- Modelare funcții
- Modelare comportament
- Elemente de analiză structurată
- Elemente de analiză OO
- Principii practice ale modelării

# MODELARE ORIENTATĂ PE FLUX - date

Modelarea fluxului datelor – activitate de modelare centrală în *analiza structurată*.

Diagramele **DFD** modelează simultan *informaţii* şi *funcţii*.

Scop DFD : consens semantic între utilizatorii şi dezvoltatorii sistemului.

Perspectiva DFD : **intrare→proces→ieşire**.

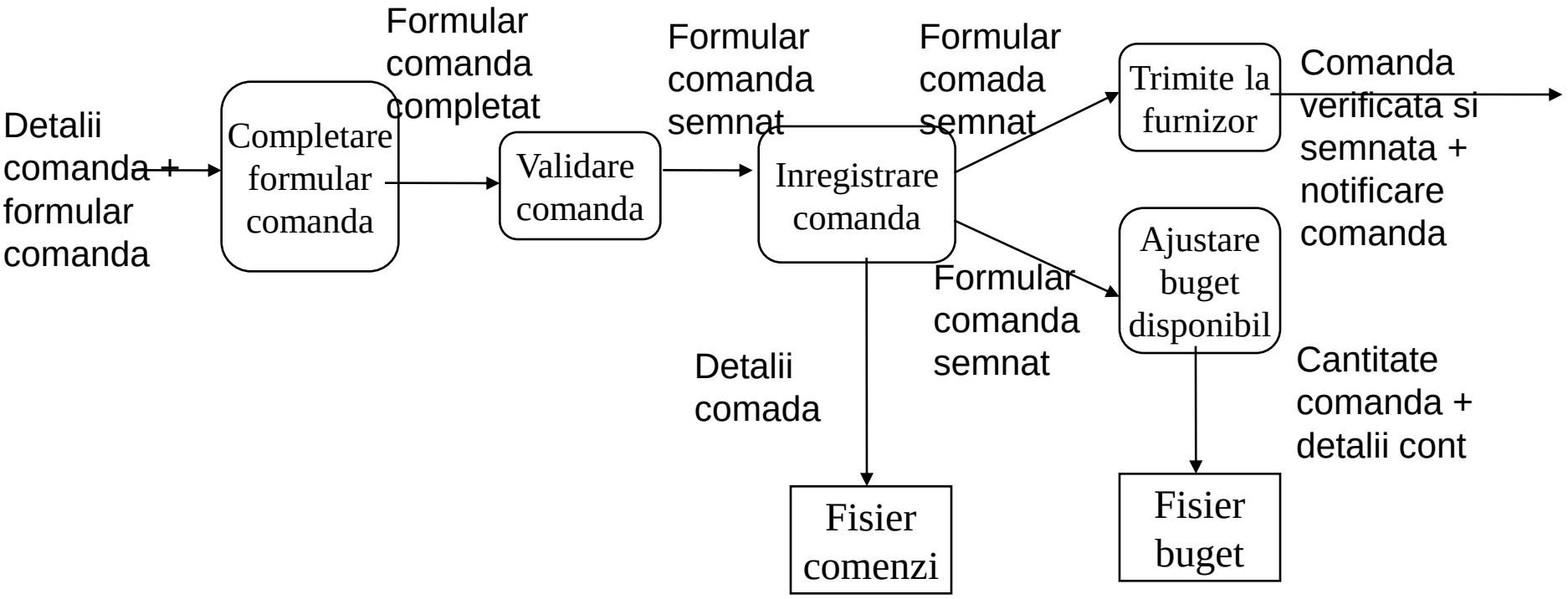
Nivele multiple: nivelul 0 reprezintă diagrama de context; fiecare nivel adaugă detalii.

La rafinarea unui nivel analistul realizează:

- descompunere funcţională explicită,
- rafinarea datelor.

(Diagramele fluxurilor de date (DFD) şi cele asociate lor pot complementa cu succes diagramele UML.)

# MODELARE ORIENTATĂ PE FLUX – Exemplu: Diagrama Fluxului Datelor (DFD)



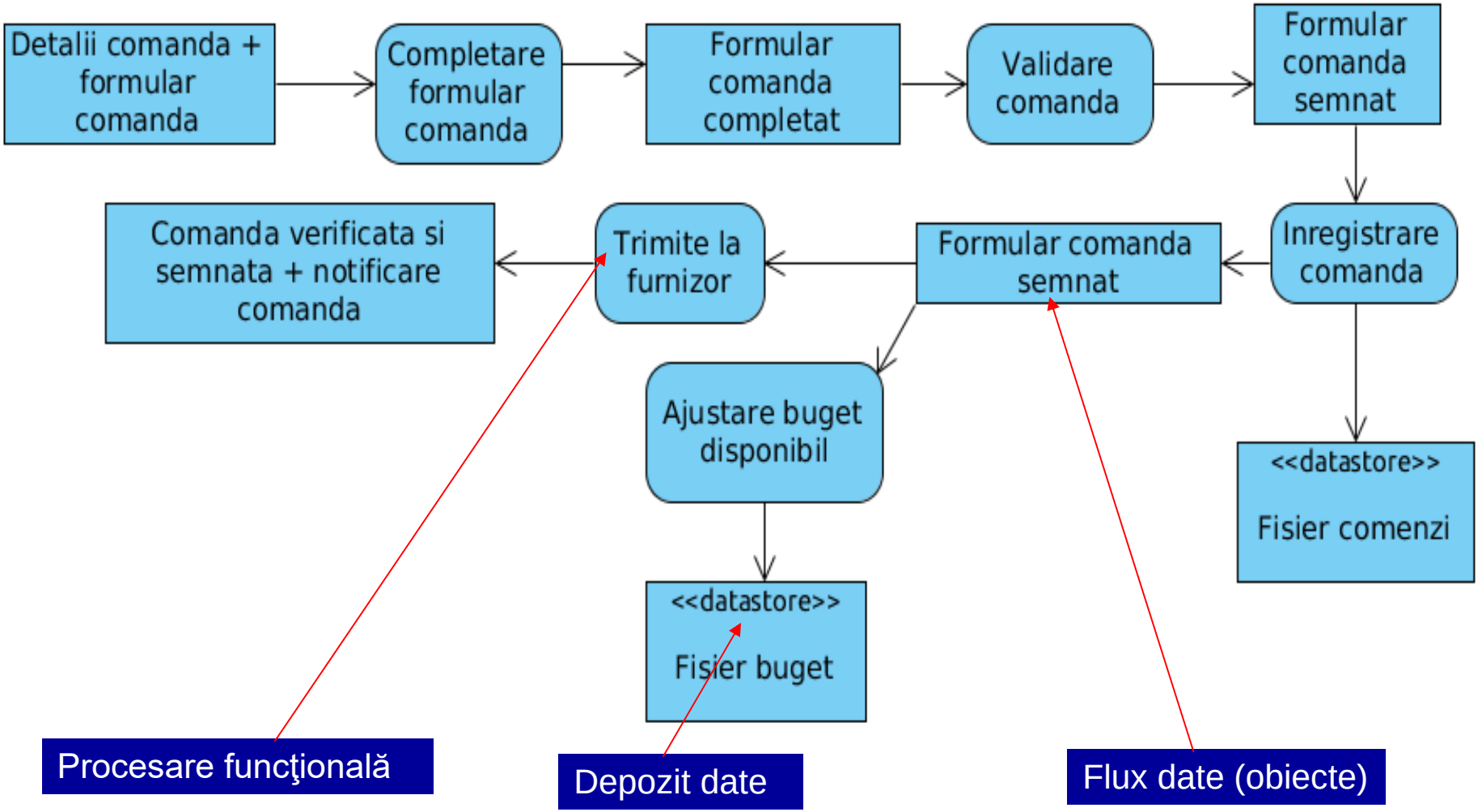

Notaţie:

Dreptunghi rotunjit – procesare funcţională

Dreptunghi – depozit de date

Săgeată etichetată – fluxul de date.

# MODELARE ORIENTATĂ PE FLUX – Exemplu: Diagrama de activitate



# MODELARE ORIENTATĂ PE FLUX -date

## Construirea DFD:

Nivelul 0 va reprezenta sistemul software ca procesare funcţională unică.

Se completează intrările şi ieşirile globale.

Rafinare procese, obiecte de date şi depozite de date.

## Recomandări

Utilizare de nume cu semnificaţie.

Păstrarea continuităţii fluxului atunci când se trece de la un nivel de modelare la altul.

Rafinarea unei singure procesări funcţionale în fiecare etapă de rafinare.

# MODELARE ORIENTATĂ PE FLUX - control

Modelul datelor şi DFD – suficiente pentru modelarea structurată a aplicaţiilor simple.

**Modelul fluxului de control** – necesar pentru aplicaţii ce procesează *evenimente* şi produc informaţii de *control*.

Reprezentarea *comportamentului* sistemului în termeni de *stări* şi de *tranziţii* între acestea declanşate de *evenimente* interne sau externe.

# PLAN CURS

- Analiza sistemelor software: concepte şi abordări
- Modelare date
- Modelare funcții
- Modelare comportament
- Elemente de analiză structurată
- Elemente de analiză OO
- Principii practice ale modelării

# ANALIZA ORIENTATĂ OBIECT (OOA)

**Scop**:

- Definirea *claselor* relevante
- Definirea *relaţiilor* între aceste clase
- Definirea *comportamentului* claselor

Fundamentată pe conceptele OO:

- ***Clasă*** – încapsulare date şi abstractizări procedurale; descrierea unei colecţii de obiecte similare.
- ***Atribut*** – colecţie de valori de date ce descriu o proprietate a obiectelor clasei
- ***Obiect*** – instanţă a unei clase
- ***Operaţie*** (metodă, serviciu) – reprezentarea unui comportament al clasei
- ***Subclasă*** – specializarea unei clase
- ***Superclasă*** (clasă de bază) – generalizarea unui set de clase

# ANALIZA ORIENTATĂ OBIECT (OOA)

## OOA: Activităţi necesare:

1. Comunicarea, de la client la inginerul software, a *cerinţelor* utilizator de bază.
2. Identificarea *claselor*, definirea *atributelor* şi *operaţiilor*.
3. Definirea *ierarhiei* claselor.
4. Reprezentarea *relaţiilor* între clase (instanțiabile ca relații obiect-obiect).
5. Modelarea *comportamentului* obiectelor.

Reluare paşi 1-5 până la completarea modelului.

# MODELARE BAZATĂ PE CLASE

## IDENTIFICAREA **CLASELOR** MODELULUI ANALIZĂ (**clasele de analiză**)

Analiză gramaticală pe textul use-case cu selectarea *substantivelor/construcţiilor substantivale*.

Spaţiul *problemei* = clasele necesare *descrierii* soluţiei
Spaţiul *soluţiilor* = clasele necesare *implementării* soluţiei

Reguli:

- Clasa nu trebuie să aibă un nume procedural imperativ:

Ex.
Greşit – SelectareCurs
Corect – Curs, cu operaţia Selectare()

- Intenţia orientării obiect este încapsularea, *dar şi păstrarea distincţiei între date şi operaţii asupra lor.*

# MODELARE BAZATĂ PE CLASE

## CRITERII PENTRU SELECŢIE CLASE.

- Conţin informaţii ce trebuie reţinute (persistente).
- Oferă servicii necesare: i.e. se pot identifica operaţii care modifică valoarea atributelor.
- Se poate defini un set comun de atribute pentru toate instanţele clasei.
- Au mai multe atribute.
- Se poate defini un set comun de operaţii pentru toate instanţele clasei.
- Reprezintă entităţi externe ce consumă/produc informaţii esenţiale pentru operarea sistemului.

# MODELARE BAZATĂ PE CLASE

## CLASIFICĂRI PENTRU CLASELE MODELULUI ANALIZĂ.

**A.**

- Entităţi externe: produc/consumă informaţii sistem
- Lucruri: parte a domemiului informaţional al problemei (ex. rapoarte, afişaje, semnale)
- Evenimente: petrecute în contextul operării sistemului
- Roluri: ale persoanelor ce interacţionează cu sistemul
- Unităţi organizaţionale: relevante pentru aplicaţie (ex. divizie, grup, echipă)
- Locaţii: stabilesc contextul problemei şi funcţionarea de ansamblu a sistemului
- Structuri: clase de obiecte sau clase de obiecte aflate în relaţie.

**B.**

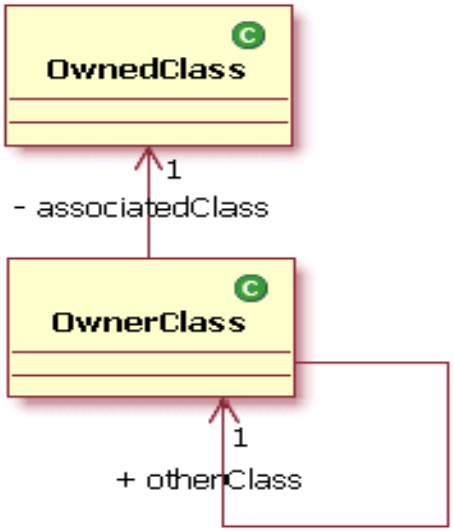
- Producători (sources): surse de date
- Consumatori (sinks): consumatori de date
- Manageri de date
- Observatori (viewer sau observer)
- Suport (helper)

**C.**

- Entităţi (entity, model, business): persistente, memorate în baze de date
- Clase de interfaţă (boundary): gestionează modul în care obiectele entitate sunt reprezentate în intefaţa utilizator
- Clase de control (controller): gestionează ciclul de viaţă pentru o “unitate de lucru” (ex. creare/actualizare entităţi, comunicare complexă între seturi de obiecte, instanţierea obiectelor de interfaţă, validarea datelor comunicate între obiecte.

# MODELARE BAZATĂ PE CLASE

## DEFINIRE **RELAŢII** ÎNTRE CLASE.




***Asociere***:

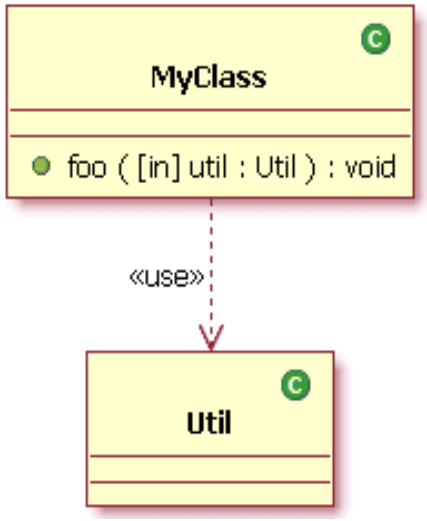
- defineşte o relaţie de conectare şi navigare între clase

Multiplicitate relaţie: numărul de obiecte dintr-o clasă ce se pot afla în relaţie cu un obiect din cealaltă clasă.

Variante : ***agregare*** și ***compoziție***.

***Dependenţă***:

- defineşte o relaţie în care o modificare la o clasa induce modificare la clasa ce depinde de ea, nu şi reciproc.
- numită folosind un stereotip (capturează o semantică definită de utilizator)
- defineşte o relaţie de tip client-server.

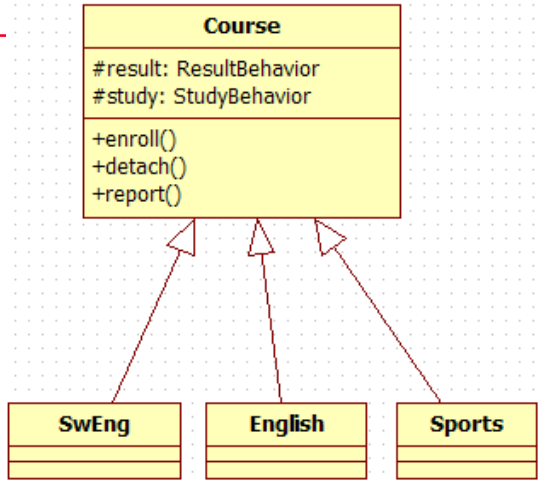


# MODELARE BAZATĂ PE CLASE

DEFINIRE **RELAŢII** ÎNTRE CLASE.

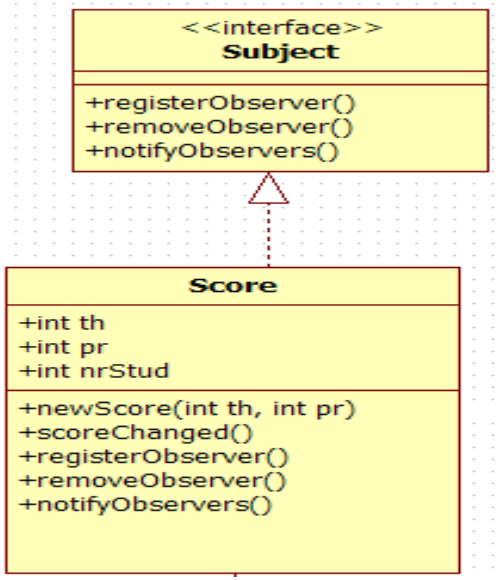
***Generalizare***:

- defineşte o relaţie de tip “isA” în care o clasă (subclasă) este specializarea unei alte clase (superclasă).




***Realizare (implementare):***

- O clasă realizează (implementează) comportamentul specificat de altă clasă/interfaţă



# MODELARE BAZATĂ PE CLASE

DEFINIRE **COMPORTAMENT** (operații, servicii).

Categorii de operaţii (clasificare generală):

- manipulare date
- calcul
- interogarea stării obiectului
- monitorizare obiect pentru apariţia unui eveniment de control (listener /event handler)

Operează pe *atribute* şi/sau pe *asocieri*.

Exemple de proceduri de identificare:

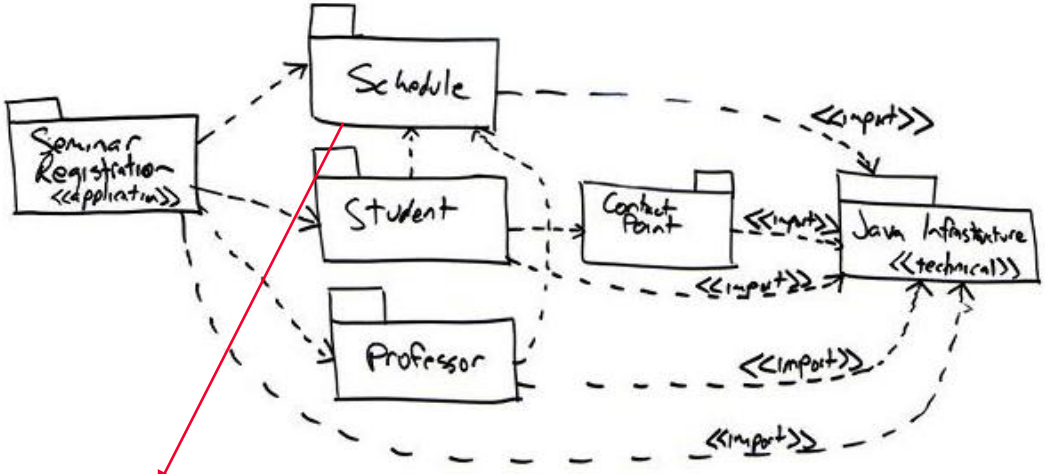
- Analiză gramaticală pe textul cazurilor de utilizare cu *selectare verbe*.
- Procedură bazată pe modelarea scenariilor cu *diagrame de secvențe*.

# MODELARE BAZATĂ PE CLASE

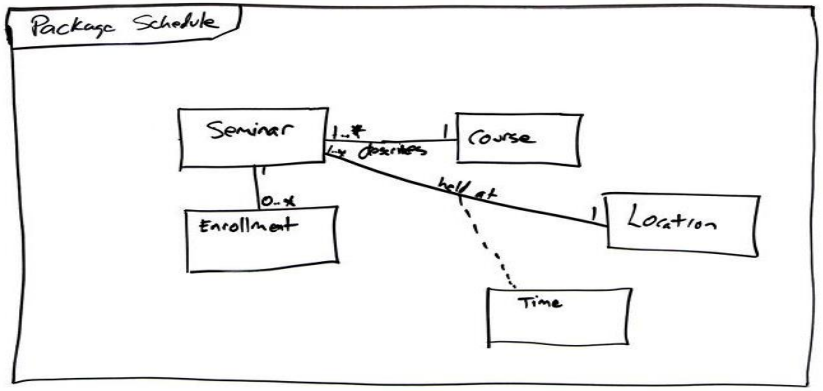
*Organizare clase în pachete*.

PACHETE. Construcţii UML utilizate pentru organizarea elementelor unui model în grupuri.

Diagramă de pachete:



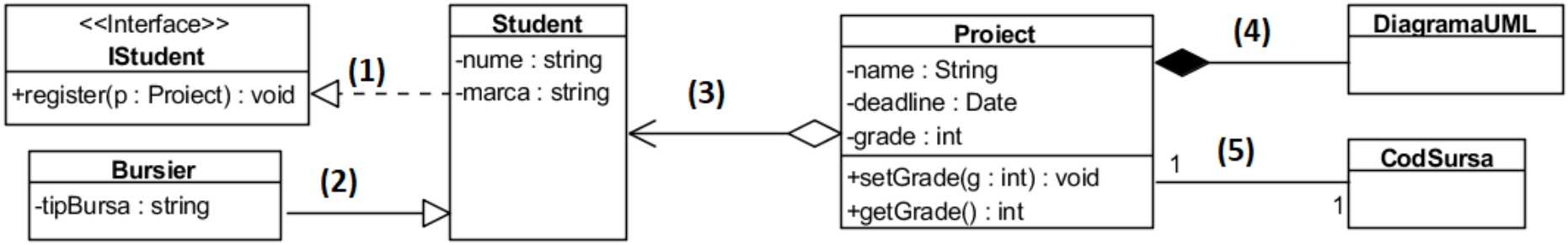

Conţinut pachet:



# ANALIZĂ ORIENTATĂ OBIECT – Evaluare formativă

1. Realizați corespondența corectă între concept și definiția sa.

2. Identificați și explicați relațiile din următoarea diagramă de clase




(1) _____

(2) _____

(3) _____

(4) _____

(5) _____

https://forms.gle/o9a6s952FFWko8c9A

# PLAN CURS

- Analiza sistemelor software: concepte şi abordări
- Modelare date
- Modelare funcții
- Modelare comportament
- Elemente de analiză structurată
- Elemente de analiză OO
- Principii practice ale modelării

# PRINCIPII PRACTICE ALE MODELĂRII

#1 Reprezentarea şi înţelegerea domeniului informaţional al problemei.

- Datele care intră în sistem – de la:
  - » utilizatori finali
  - » alte sisteme
  - » dispozitive externe
- Datele ce ies din sistem – către:
  - » interfaţă utilizator
  - » interfaţă reţea
  - » imprimare: rapoarte, grafice
- Datele persistente

# PRINCIPII PRACTICE ALE MODELĂRII

#2 Definirea funcţiilor realizate de sistem.

- Funcţii de *interfaţă*
- Funcţii *interne* (suport pentru funcţiile de interfaţă)
- Funcţii de *transformare a datelor*
- Funcţii de *control*

Pot fi descrise pe nivele de abstractizare diferite:

declaraţii iniţiale de intenţie **...↔...** detalii ale elementelor de procesare ce trebuie invocate

# PRINCIPII PRACTICE ALE MODELĂRII

#3 Reprezentarea comportamentului (consecinţă a evenimentelor externe) software-lui.

Comportamentul – declanşat de interacţiunea cu mediul extern.

Exemple de factori declanşatori:

- intrări oferite de utilizatorii externi,
- date de control oferite de un sistem extern,
- monitorizarea datelor colectate din reţea.

# PRINCIPII PRACTICE ALE MODELĂRII

#4 Modelele care descriu informaţiile, funcţiile şi comportamentul trebuie *partiţionate* $\Rightarrow$ detaliile sunt descoperite într-o manieră ierarhică.

Utilizarea unei strategii de tip “divide and conquer”, numită *partiţionare*.

- Problema complexă este divizată în subprobleme, care pot fi divizate la rândul lor în subprobleme.
- Procesul încetează când subproblema este relativ simplu de înţeles.

# PRINCIPII PRACTICE ALE MODELĂRII

#5 Activitatea de analiză trebuie să avanseze de la informaţii esenţiale către detalii.

Descrierea problemei din punctul de vedere al utilizatorului: esenţa problemei.

Detaliile de implementare a soluţiei cad în sarcina proiectării.

# PRINCIPII PRACTICE ALE MODELĂRII

SET GENERIC DE ACTIVITĂŢI :

- *Revizuirea cerinţelor*
- Expandarea şi rafinarea *scenariilor*
- Modelarea *informaţiilor*
- Modelarea *funcțiilor*
- Modelarea *comportamentelor*
- Analiza şi modelarea *interfeţei utilizator*
- *Revizuirea* completitudinii şi consistenţei tuturor modelelor.

# PRINCIPII PRACTICE ALE MODELĂRII

SET GENERIC DE ACTIVITĂŢI (1):

- *Revizuirea cerinţelor* business, *caracteristicilor şi nevoilor* utilizatorilor, *ieşirilor* vizibile utilizatorilor, *constrângerilor* business şi a altor *cerinţe tehnice* determinate în etapele anterioare.

- Expandarea şi rafinarea *scenariilor*:
  - Definirea tuturor *actorilor*
  - Reprezentarea *interacţiunilor* actorilor cu software-ul
  - Extragerea *funcţiilor* şi caracteristicilor din scenariile utilizator
  - *Revizuirea* completitudinii şi acurateţei scenariilor utilizator

- Modelarea *informaţiilor*
  - Reprezentarea tuturor *obiectelor* de informaţie majore
  - Definirea *atributelor* pentru fiecare obiect de informaţie
  - Reprezentarea *relaţiilor* dintre obiectele de informaţie

# PRINCIPII PRACTICE ALE MODELĂRII

SET GENERIC DE ACTIVITĂŢI (2):

- Modelarea *funcţiilor*
  - Definirea modului în care funcţiile *modifică datele*.
  - *Rafinarea* funcţiilor pentru a oferi detalii de elaborare.
  - Scrierea unui *text narativ / model* care descrie fiecare funcţie şi subfuncţie.
  - *Revizuirea* modelului funcţional.

- Modelarea *comportamentelor*
  - Identificarea *evenimentelor* externe care *declanşează* modificări comportamentale în sistem.
  - Identificarea *stărilor* care reprezintă fiecare mod de comportare observabil din exterior.
  - Precizarea modului în care un *eveniment determină tranziţia* unui sistem de la o stare la alta.
  - *Revizuirea* modelelor comportamentului.

# PRINCIPII PRACTICE ALE MODELĂRII

SET GENERIC DE ACTIVITĂŢI (3):

- Analiza şi modelarea *interfeţei utilizator*
  - Conducerea activităţii de *analiză a utilizatorilor*.
  - Crearea de *prototipuri* pentru imaginile de pe ecran.

- *Revizuirea* completitudinii şi consistenţei tuturor modelelor.

# Evaluare formativă

1. Deoarece în realitate desfășurarea activităților din setul generic se poate suprapune, propuneți o ordine de lansare a acestora.

https://forms.gle/enooVTX6uavaEq33A

# BIBLIOGRAFIE


Roger S. Pressman, **Software Engineering. A Practitioner's Approach** ed.7, McGraw-Hill International Edition, 2010, cap. 6 – 14.

Tom Pender, **UML Bible**, Ed. John Wiley, 2003

Booch G., Maksimchuk R.A., Engle M.W., Zoung B.J., Conallen J., Houston K.A, **Object-Oriented Analysis and Design with Applications, Third Edition,** Addison-Weslez Professional, 2007

Fairbanks, G.H., **Just Enough Software Architecture: A Risk-Driven Approach,** Marshall & Brainerd, 2010

http://www.cs.sjsu.edu/faculty/pearce/modules/lectures/ooa/

http://www.cs.sjsu.edu/faculty/pearce/modules/lectures/ooa/requirements