Analiza și proiectarea sistemelor software

Curs 4

PLAN CURS

- **Modelul proiect**
- Identificare elemente de proiectare
- Proiectare clase

MODELELE PRIMARE ALE UNUI SISTEM SOFTWARE

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

Modelul domeniului (modelul analiză)

- adevărurile durabile despre domeniu, relevante pt. sistemul de dezvoltat
- detalii ale domeniului independente de implementarea sistemului
- mod de a înţelege elementele ce vor fi esenţiale în procesul de proiectare
- reprezentat cu notaţii simple şi intuitive pentru o interacţiune eficientă cu experţii domeniului
- baza unui limbaj universal pentru domeniul respectiv (DSL – Domain Specific Language ; Domain Patterns)

Modelul proiect

- descrie sistemul ce va fi construit : decizii de proiectare, mecanisme, etc.
- mai multe nivele de abstractizare
- pentru fiecare componentă:
  - interfaţa publică vizibilă
  - detaliile interne

Modelul codului

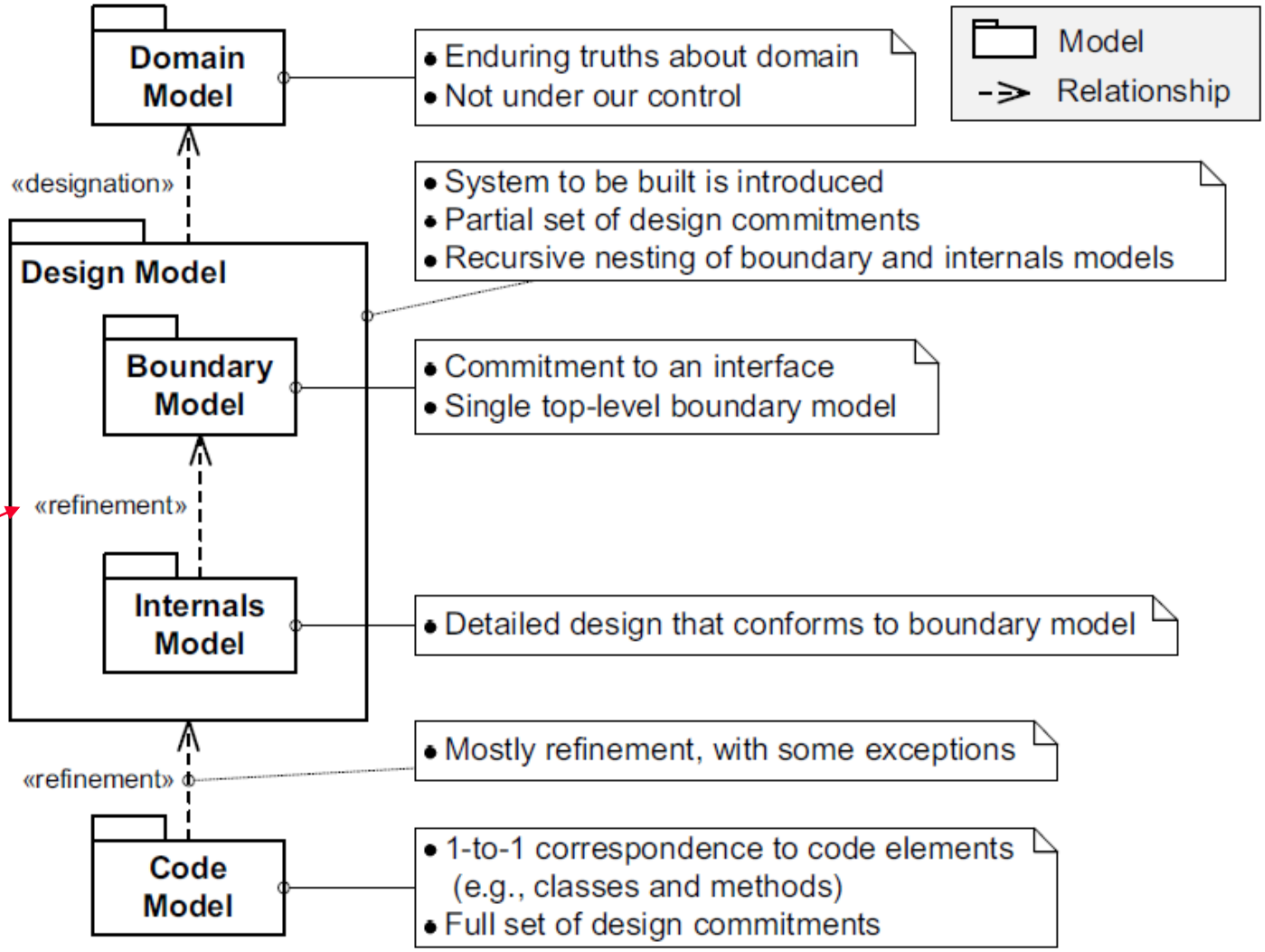
- descrie codul sursă al sistemului cu un model de nivelul limbajului de programare
- suficient pentru a fi executat automat

STRUCTURA CANONICĂ A MODELELELOR PRIMARE

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

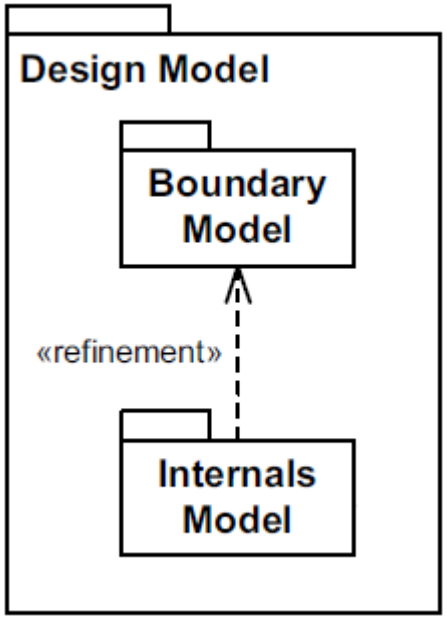
Corespondenţă între conceptele din modelul domeniului şi tipurile definite în modelul proiect.

Relaţie între modele ale aceluiaşi element aflate pe nivele de detaliu diferite.



MODELUL PROIECT

- Descrie sistemul ce va fi construit.
- Încuibare recursivă (structură arborescentă de modele)
  - modele de interfaţă publică vizibilă (boundary)
  - modele interne (internals)



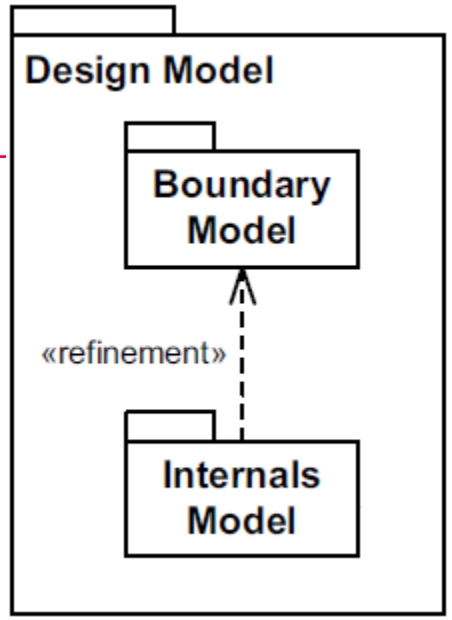
Modelul de interfaţă:

- vedere încapsulată ce ascunde detaliile interne
- reprezentare
  - comportament
  - schimburi de date
- unic la nivelul sistemului
- câte unul pentru fiecare subsistem şi componentă

Modelul intern:

- detaliile de realizare a interfeţei definită într-un model de interfaţă
- rafinarea unui model de interfaţă

MODELUL PROIECT



- Elemente de modelare :
  - scenarii
  - componente
  - conectori
  - porturi – seturi de funcţii înrudite din interfaţa componentei
  - responsabilităţi
  - module
  - clase
  - interfeţe
  - elemente de context

MODELUL PROIECT

DATE

Proiectare date $\Rightarrow$ modelul (arhitectura) datelor.

Dezvoltare incrementală, prin rafinări succesive

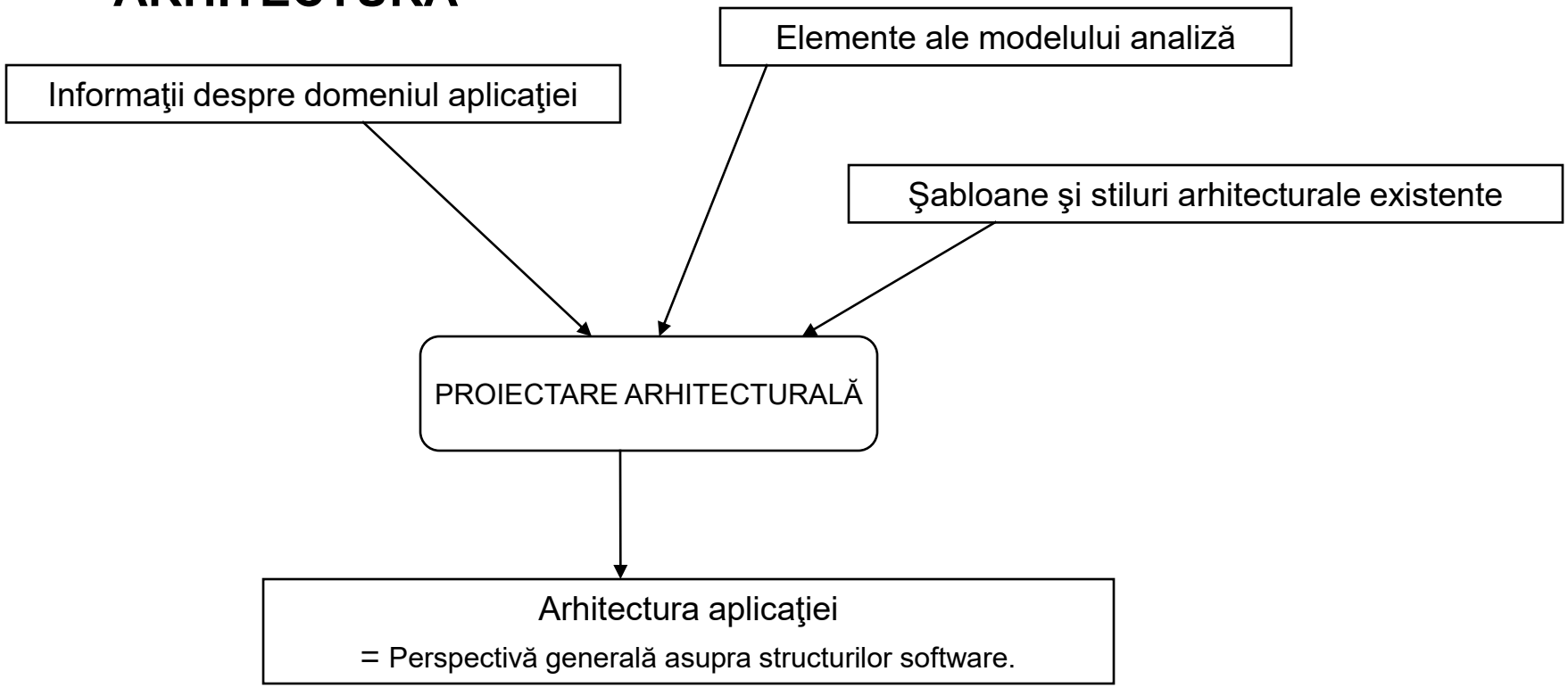
Exemplu:

- La nivel arhitectural : fişiere şi baze de date.
- La nivel de componentă : structurile de date necesare implementării obiectelor de date.

Influenţă semnificativă asupra arhitecturii software-lui ce va procesa datele.

MODELUL PROIECT

ARHITECTURĂ



INTERFEŢE

Descrierea fluxului de informaţii între:

- sistem şi contextul său,
- componentele sistemului.

Categorii de interfeţe:

- interfaţa cu utilizatorul (UI),
- interfeţele cu sisteme externe (cu: alte sisteme, dispozitive, reţea, alţi producători şi consumatori de informaţii),
- interfeţele interne (între componentele sistemului).

MODELUL PROIECT

INTERFEŢE **- Interfaţa cu utilizatorul (UI).**

În general, un singur subsistem al arhitecturii aplicaţiei.

Elemente de **estetică**: culori, stiluri, grafică, etc.

Elemente **ergonomice**: plasarea informaţiilor, metafore, navigare, etc.

Elemente **tehnice**: şabloane pentru UI, componente reutilizabile.

MODELUL PROIECT

INTERFEŢE **- Interfeţele cu sisteme externe.**

- Necesită *informaţii complete şi definitive* despre entitatea cu care se transferă informaţii.
  - colectate în procesul ingineriei cerinţelor
  - verificate la începutul proiectării (actualizarea specificaţiilor)
- Trebuie să incorporeze *verificarea erorilor*.
- Se recomandă adăugarea de *trăsături de securitate* corespunzătoare.

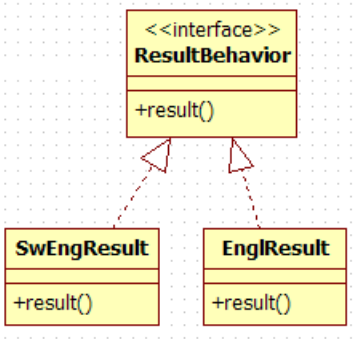
MODELUL PROIECT

INTERFEŢE - **Interfeţele interne**.

Model static: clase ce implementează interfeţe

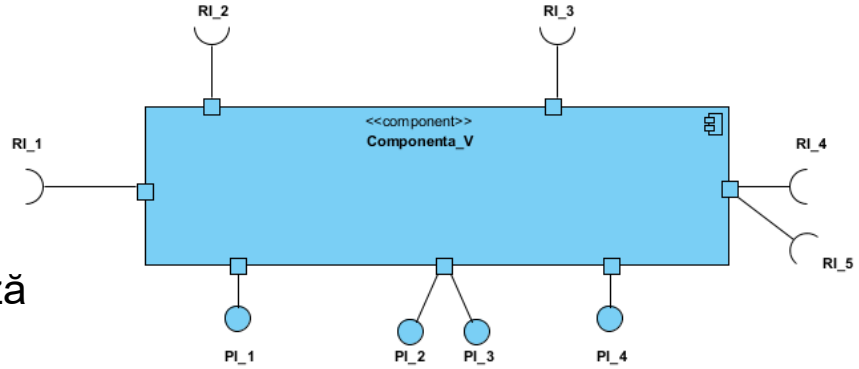
Proiectate odată cu proiectarea componentelor.

Def. (OMG) interfaţa = specificator pentru operaţiile vizibile din exterior (publice) ale unei clase sau componente sau alt clasificator (inclusiv subsisteme), fără specificarea structurii interne.



Interfaţa :

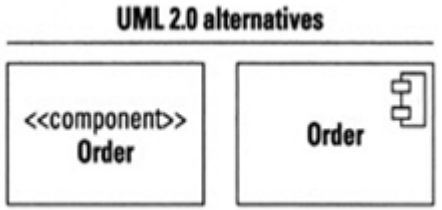
- Defineşte un set de operaţii ce descrie *o parte* a comportamentului unei clase sau componente.
- Oferă acces la aceste operaţii.



Model dinamic: componente cu porturi prin care expun interfeţe oferite (provided) sau se conectează la interfeţe oferite de alte componente (required).

COMPONENTE

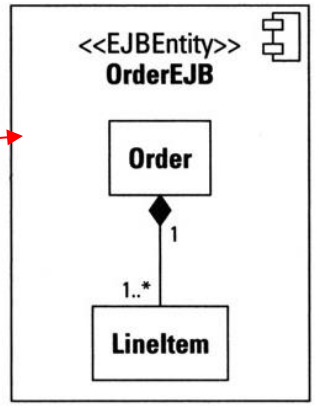
Descriere completă a *detaliilor interne* ale fiecărei componente software:



Structurile de date pentru fiecare obiect de date.

Detalii algoritmice pentru fiecare procesare din componentă.

Interfeţele de acces la toate operaţiile componentei.



Reprezentare *structură internă*: diagramă de clase.

Reprezentare *logică de procesare* : diagramă de activitate, pseudocod, etc.

Reprezentare *interfeţe* : diagramă de componente – porturi şi interfeţe *required* şi *provided.*

MODELUL PROIECT

INSTALARE (repartizare)

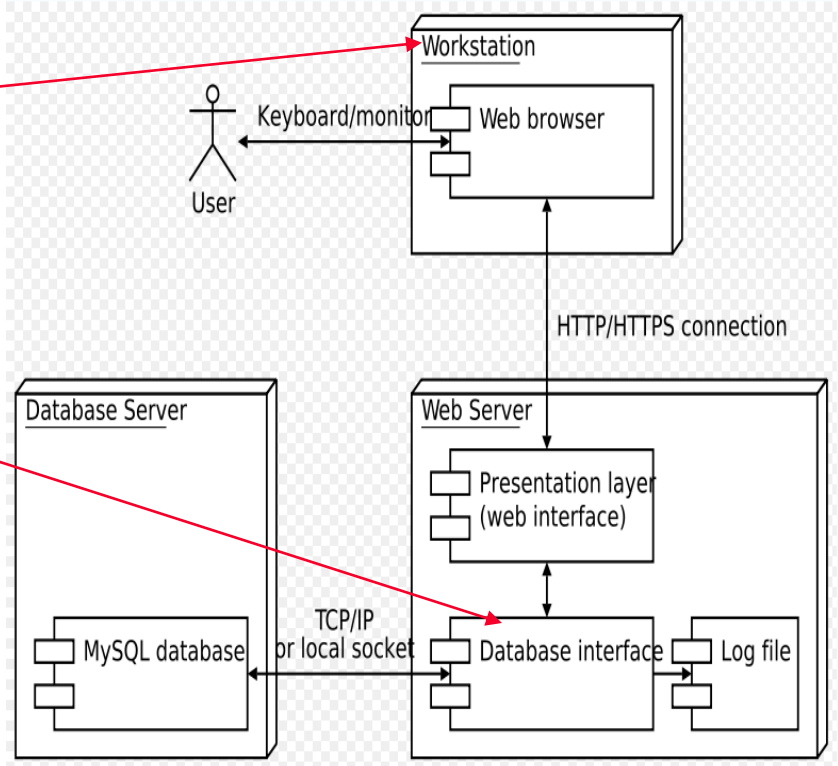
Alocarea funcţionalităţii la contextul de calcul.

Se indică ***infrastructura gazdă*** pentru fiecare subsistem.

Se indică toate ***componentele*** implementate de fiecare subsistem.

Diagramă UML de instalare (deployment diagram) – dezvoltată şi rafinată.

- *format descriptor* : fără detalii de configurare.
- *format instanţă* : cu identificarea configuraţiei hardware.



Evaluare formativă (1)

1. Conform structurii canonice a sistemelor software, enumerați modelele primare ale unui sistem software. Explicați încuibarea recursivă caracteristică unuia dintre aceste modele.

2. Care sunt principalele categorii de interfețe ale unui sistem software ?

3. Ce trebuie să conțină o descriere completă a detaliilor interne ale unei componente software și ce diagrame UML se pot folosi ?

https://forms.gle/svFNfKrL8Wzz6evs5

PLAN CURS

- Modelul proiect
- **Identificare elemente de proiectare**
- Proiectare clase

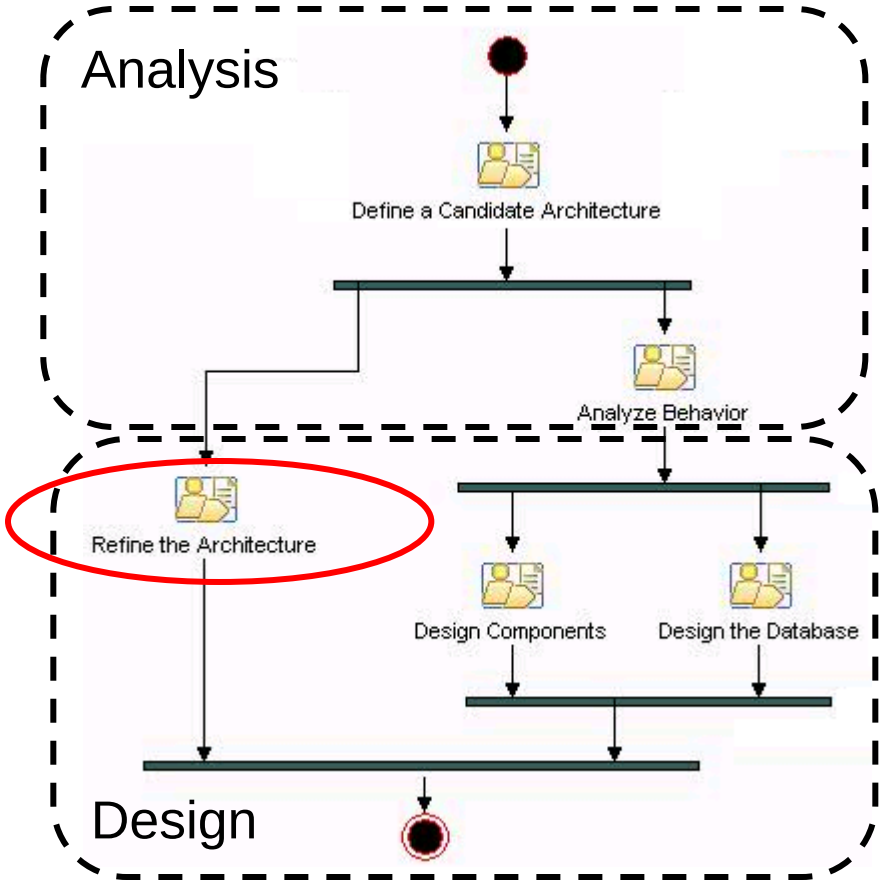
Etape

Analiză

- Analiză arhitecturală (definire arhitectură candidat)
- Analiza UC (analiză comportament)

Proiectare

- Identificare elemente de proiectare (rafinarea arhitecturii)
- Identificare mecanisme de proiectare (rafinarea arhitecturii)
- Proiectare clase (proiectare componente)
- Proiectare subsisteme (proiectare componente)
- Descrierea arhitecturii la execuţie şi a distribuirii (rafinarea arhitecturii)
- Proiectarea BD



IDENTIFICAREA ELEMENTELOR DE PROIECTARE

Scop

- Analizarea interacţiunilor claselor de analiză pentru a identifica elemente ale modelului de proiectare

Rol responsabil

- Arhitectul software

Etape majore

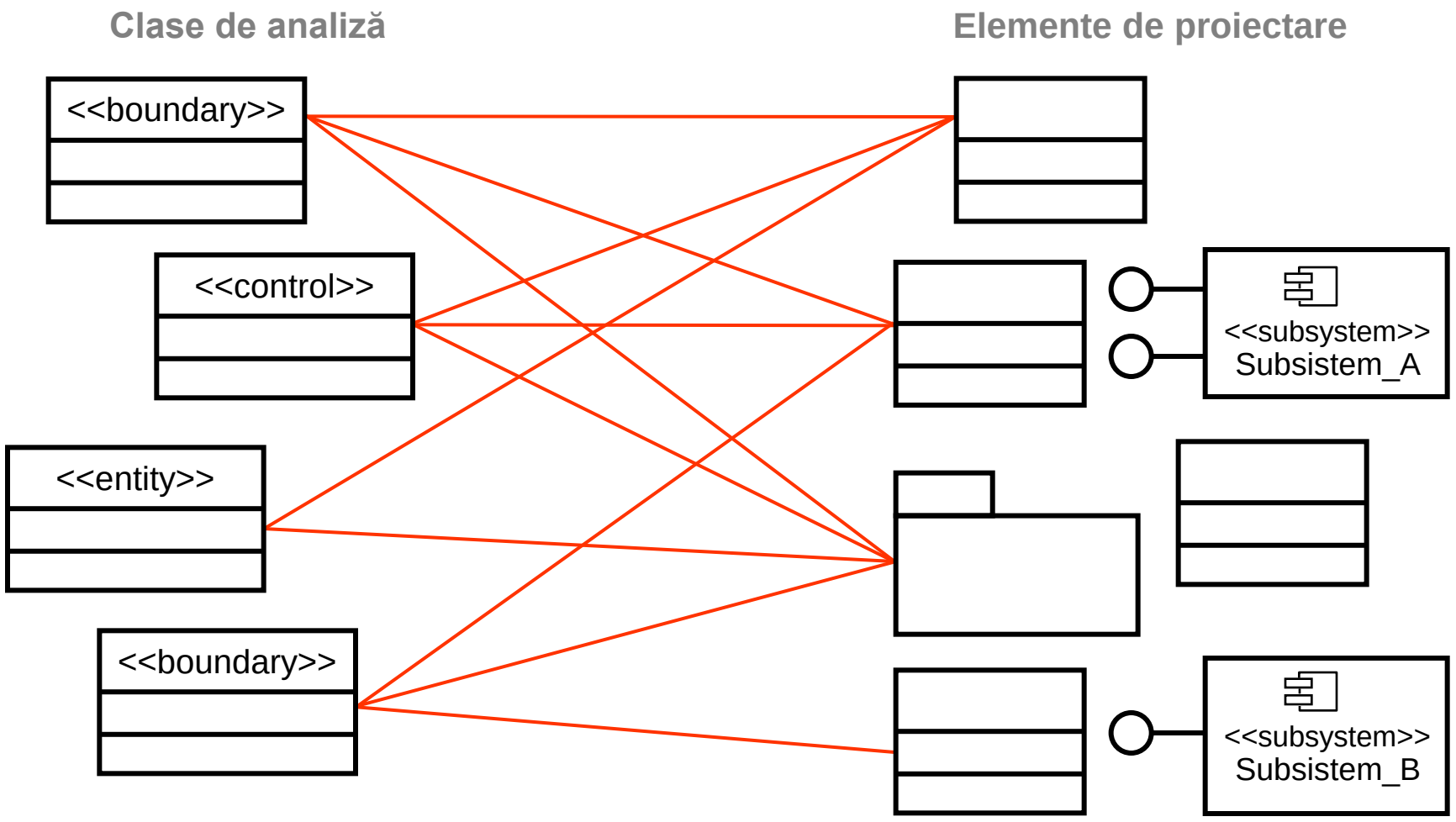
- Punerea în corespondenţă a claselor de analiză cu elemente de proiectare
- Identificarea subsistemelor şi a interfeţelor dintre subsisteme
- Actualizarea organizării modelului

Obs:

Scopul este de a identifica elemente de proiectare, NU de a rafina design-ul (care va avea loc la proiectarea componentelor).

DE LA CLASE DE ANALIZĂ LA ELEMENTE DE PROIECTARE

- Punerea în corespondenţă a claselor de analiză cu elemente de proiectare
- Identificarea subsistemelor şi a interfeţelor dintre subsisteme
- Actualizarea organizării modelului



DE LA CLASE DE ANALIZĂ LA ELEMENTE DE PROIECTARE

- •Punerea în corespondenţă a claselor de analiză cu elemente de proiectare
- •Identificarea subsistemelor şi a interfeţelor dintre subsisteme
- •Actualizarea organizării modelului

- Clase de analiză:
  - Manipulează *cerinţe funcţionale*
  - Modelează obiecte din *domeniul problemei*

- Elemente de proiectare:
  - Trebuie să manipuleze *şi cerinţe extra-funcţionale*
  - Modelează obiecte din *domeniul soluţiei*

CRITERII DE IDENTIFICARE A ELEMENTELOR DE PROIECTARE

•Punerea în corespondenţă a claselor de analiză cu elemente de proiectare
•Identificarea subsistemelor şi a interfeţelor dintre subsisteme
•Actualizarea organizării modelului

- Cerinţe extra-funcţionale; exemple:
  - Aplicaţia trebuie distribuită pe mai multe server-e
  - Sisteme de timp real vs. Aplicaţie de comerţ electronic
  - Aplicaţia trebuie să suporte diferite implementări ale memoriei persistente.
- Opţiuni arhitecturale
  - Exemplu: .NET vs. Java Platform
- Opţiuni tehnologice
  - Exemplu: Enterprise Java Beans pot gestiona persistenţa
- Principii de proiectare (identificare în etapele iniţiale ale ciclului de viaţă al proiectului)
  - Utilizarea mecanismelor de proiectare
  - Bune practici (la nivel de industrie, corporaţie, proiect)
  - Strategia de reutilizare

IDENTIFICAREA CLASELOR DE PROIECTARE

•Punerea în corespondenţă a claselor de analiză cu elemente de proiectare
•Identificarea subsistemelor şi a interfeţelor dintre subsisteme
•Actualizarea organizării modelului

- O clasă de analiză poate:
  - corespunde direct unei clase de proiectare

  sau poate :
  - fi divizată în mai multe clase
  - deveni o parte a unei alte clase
  - deveni un pachet
  - deveni un subsistem
  - deveni o relaţie
  - fi realizată parţial în hardware
  - să nu fie modelată deloc
  - orice combinaţie …

- O clasă de analiză corespunde direct unei clase de proiectare dacă:
  - Este o clasă simplă
  - Reprezintă o singură abstractizare logică

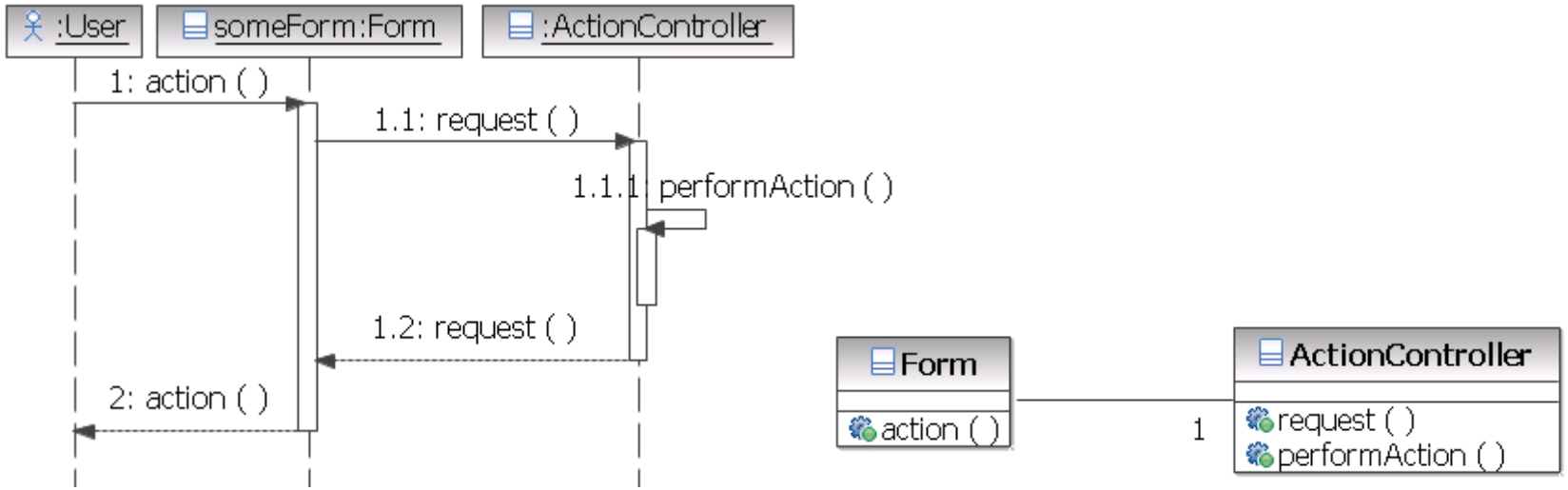
Ex. În mod tipic, clasele entity supravieţuiesc relativ intacte în Modelul Proiect

Exemplu
ANALIZA

- Punerea în corespondenţă a claselor de analiză cu elemente de proiectare
- Identificarea subsistemelor şi a interfeţelor dintre subsisteme
- Actualizarea organizării modelului

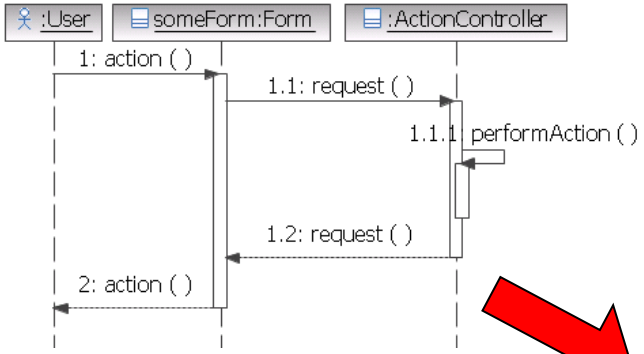
Să presupunem că la finalul analizei am obţinut următorul model (simplu şi generic)

Cerinţele stipulează că este vorba de o aplicaţie tipică JavaEE Web, cu un client “thin” şi un server Web.



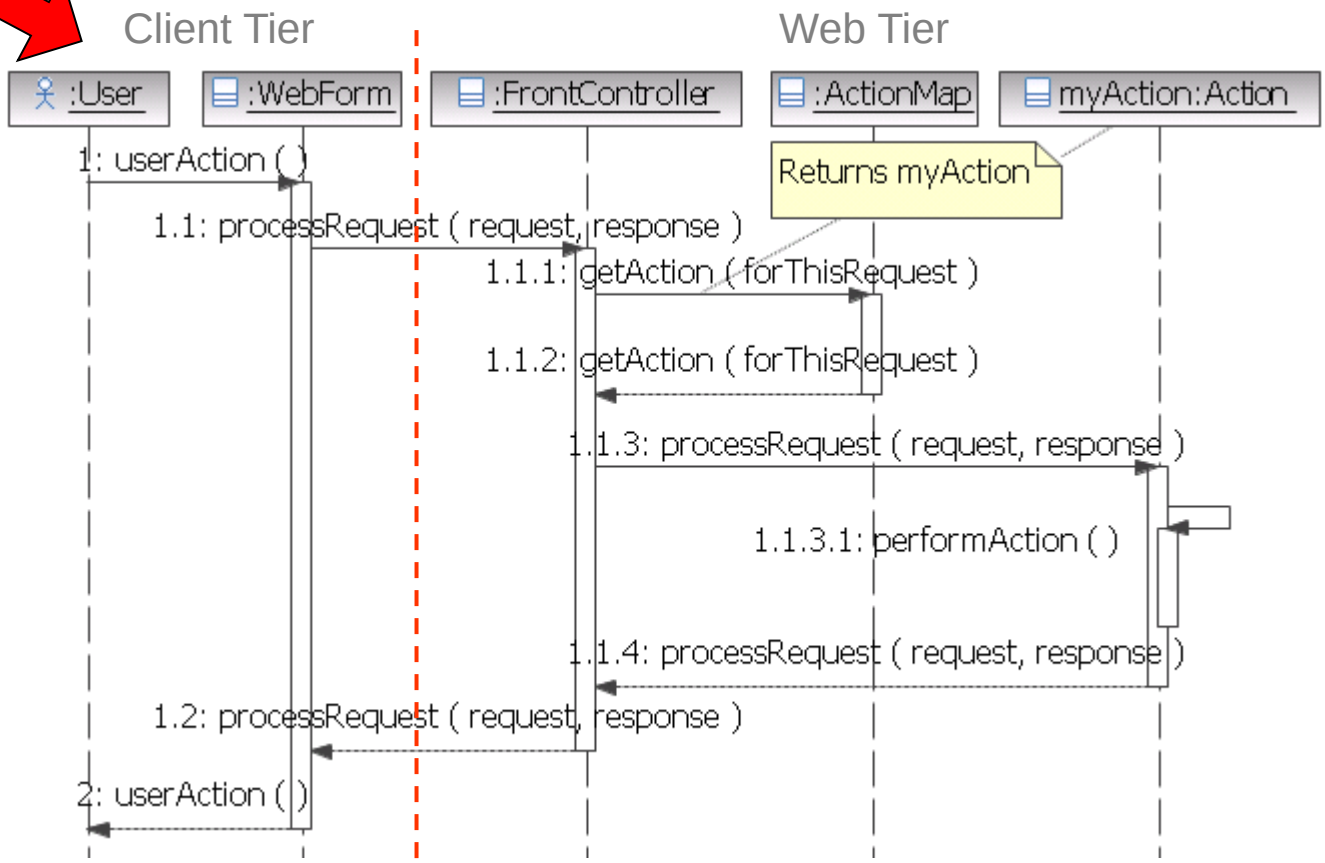
Exemplu
PROIECTAREA

- Punerea în corespondenţă a claselor de analiză cu elemente de proiectare
- Identificarea subsistemelor şi a interfeţelor dintre subsisteme
- Actualizarea organizării modelului



Arhitectul a decis să utilizeze framework-ul Struts, care, printre altele, gestionează elemente de tip *FrontController* şi *ActionMap*.

În acest exemplu, *someForm* devine un obiect de tip JSP şi controlerul este divizat în 2 clase: un servlet *FrontController* (şablon JavaEE) şi o clasă *Action* care realizează activitatea propriu-zisă (*performAction*)



Exemplu
PROIECTAREA

- Punerea în corespondenţă a claselor de analiză cu elemente de proiectare
- Identificarea subsistemelor şi a interfeţelor dintre subsisteme
- Actualizarea organizării modelului

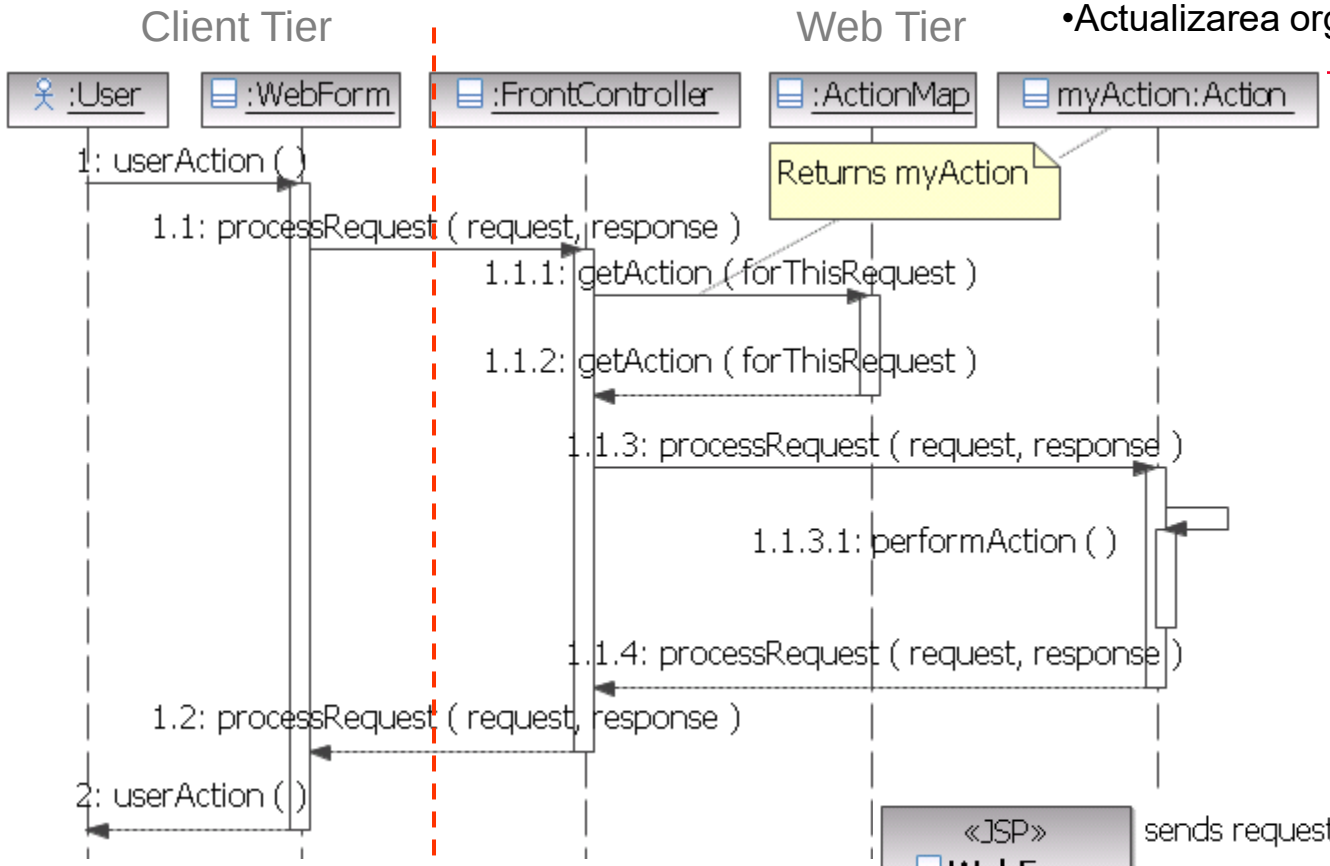
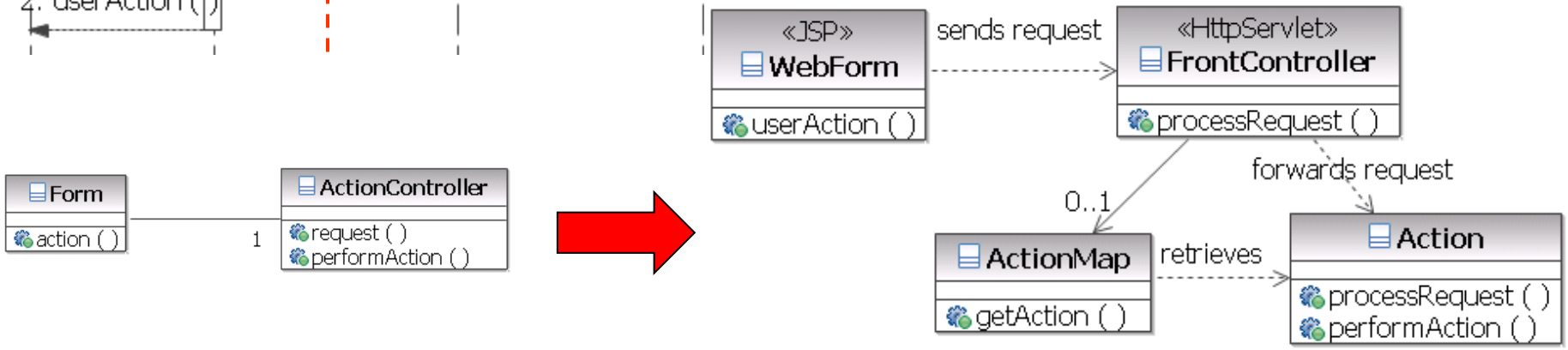


Diagrama claselor participante (VOPC) se va modifica corespunzător.



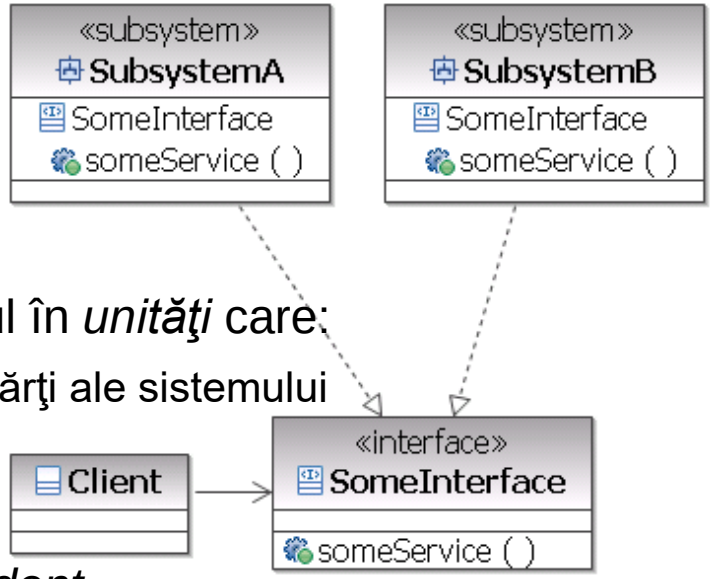
SUBSISTEME CA ELEMENTE DE PROIECTARE ÎNLOCUIBILE

- Punerea în corespondenţă a claselor de analiză cu elemente de proiectare
- Identificarea subsistemelor şi a interfeţelor dintre subsisteme
- Actualizarea organizării modelului

O clasă de analiză complexă poate deveni un subsistem

- Subsistemele sunt componente care oferă *servicii* clienţilor doar prin *interfeţe* publice
  - Orice două subsisteme care *realizează aceeaşi interfaţă* sunt *interschimbabile*
  - În spatele acestei interfețe, subsistemele

suportă *variante multiple de implementare.*



- Subsistemele pot fi utilizate pentru a împărţi sistemul în *unităţi* care:
  - pot fi *schimbate independent*, fără a afecta alte părţi ale sistemului
  - pot fi *dezvoltate independent*

  (atât timp cât interfaţa rămâne nemodificată),
  - pot fi *ordonate, configurate* sau *livrate independent*.

⇓

Subsistemele sunt ideale pentru modelarea componentelor – unităţile înlocuibile ale ansamblului în dezvoltarea bazată pe componente.

SUBSISTEME CANDIDAT

- Punerea în corespondenţă a claselor de analiză cu elemente de proiectare
- Identificarea subsistemelor şi a interfeţelor dintre subsisteme
- Actualizarea organizării modelului

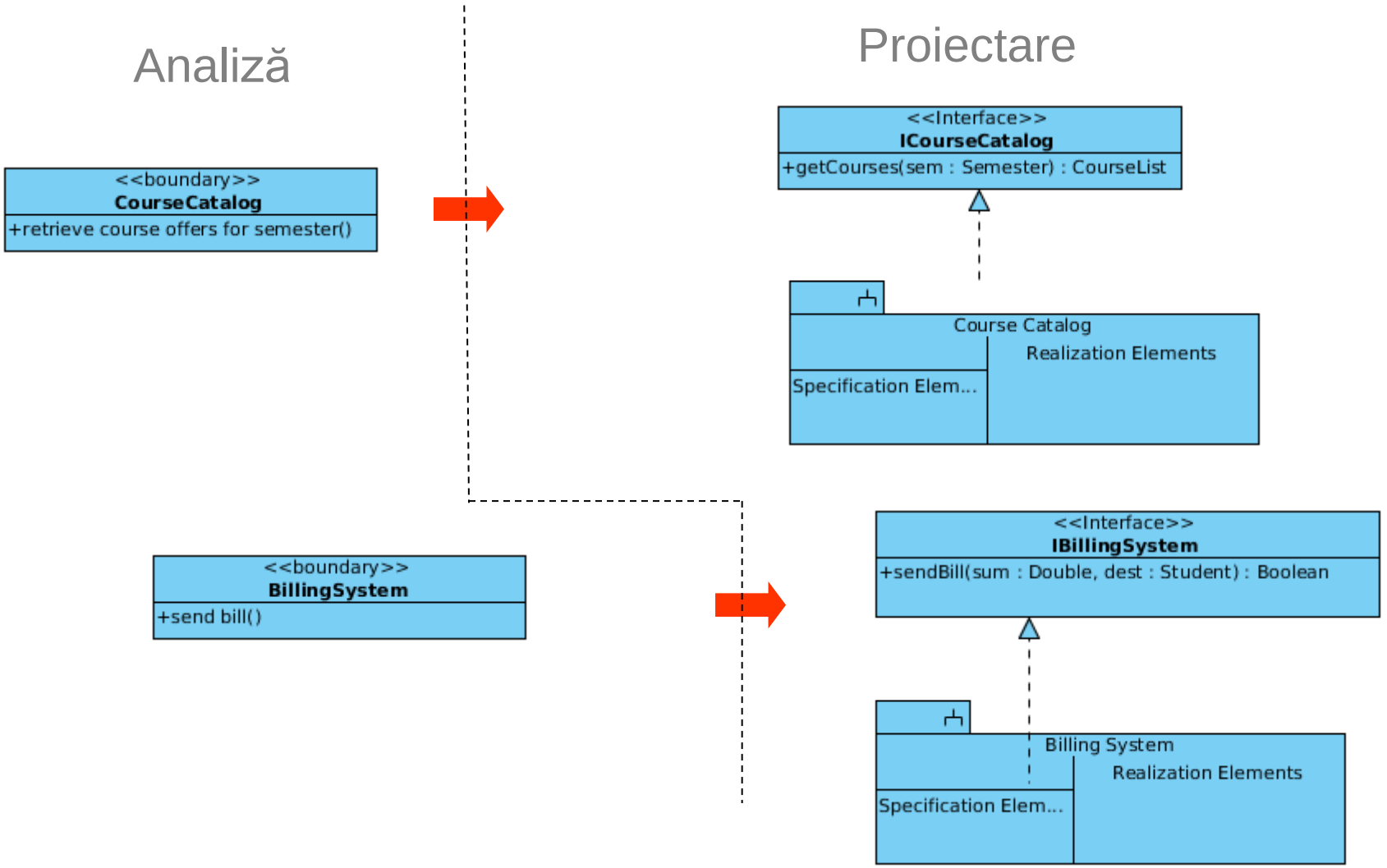
Principalele elemente de analiză care sunt candidate să devină subsisteme :

- Clasele de analiză care oferă servicii şi/sau utilităţi complexe
  - Exemplu: servicii de autorizare pentru securitate
- Clasele <<boundary>>
  - Interfeţele cu utilizatorul – formează în mod tipic un singur subsistem GUI
  - Clasele <<boundary>> de acces la sisteme şi/sau dispozitive externe
- Clasele care oferă comportament opţional sau variante diferite ale aceluiaşi serviciu
- Elemente puternic cuplate
- Produse existente care exportă interfeţe (software de comunicare, suport pentru acces la baze de date, etc.)

Exemplu
SISTEM PENTRU ÎNSCRIERE LA CURSURI

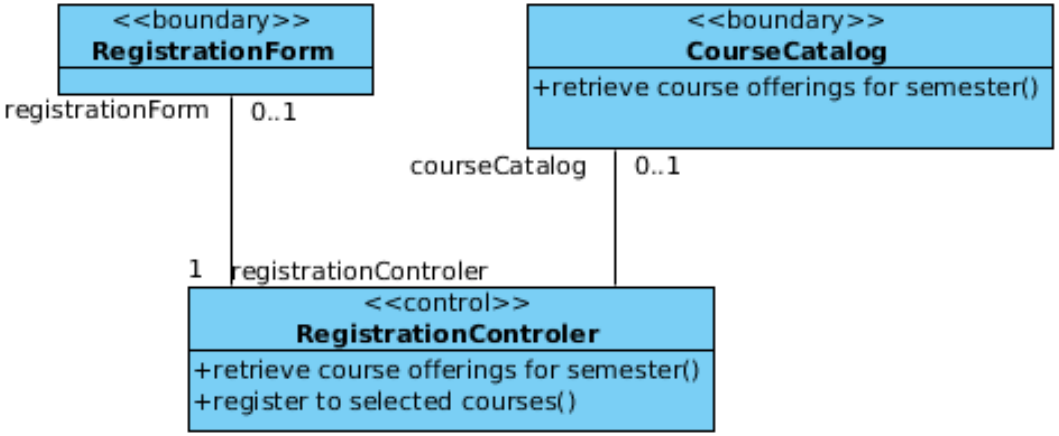
- Punerea în corespondenţă a claselor de analiză cu elemente de proiectare
- Identificarea subsistemelor şi a interfeţelor dintre subsisteme
- Actualizarea organizării modelului

Clasele <<boundary>> de acces la sisteme şi/sau dispozitive externe.



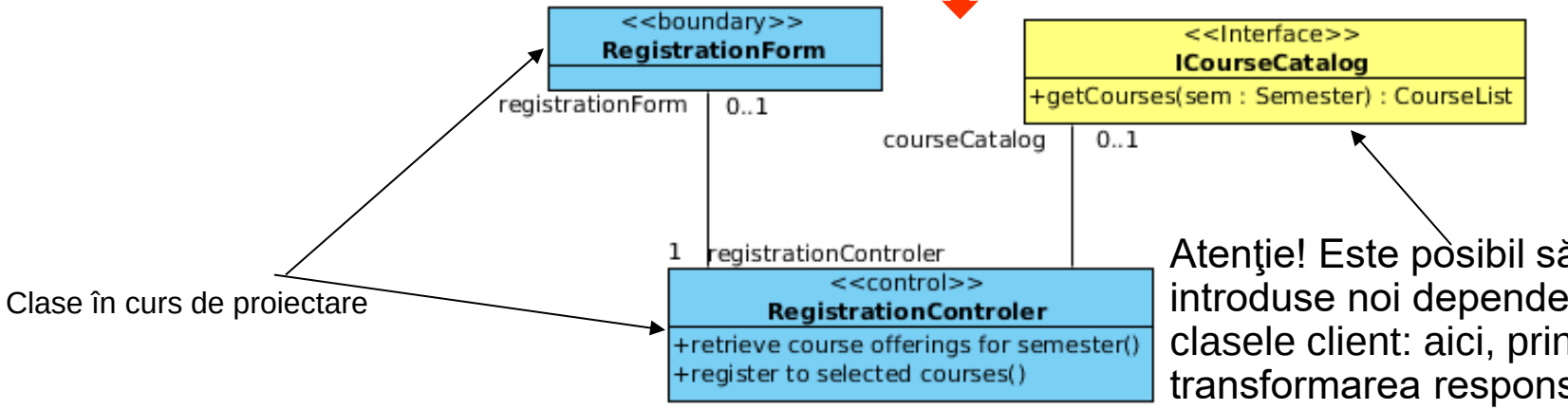
Exemplu
INCORPORARE INTERFEȚE ÎN DIAGRAMA DE CLASE

- Punerea în corespondenţă a claselor de analiză cu elemente de proiectare
- Identificarea subsistemelor şi a interfeţelor dintre subsisteme
- Actualizarea organizării modelului



Clasa <<boundary>> de analiză *CourseCatalog* este înlocuită cu interfața subsistemului corespunzător.

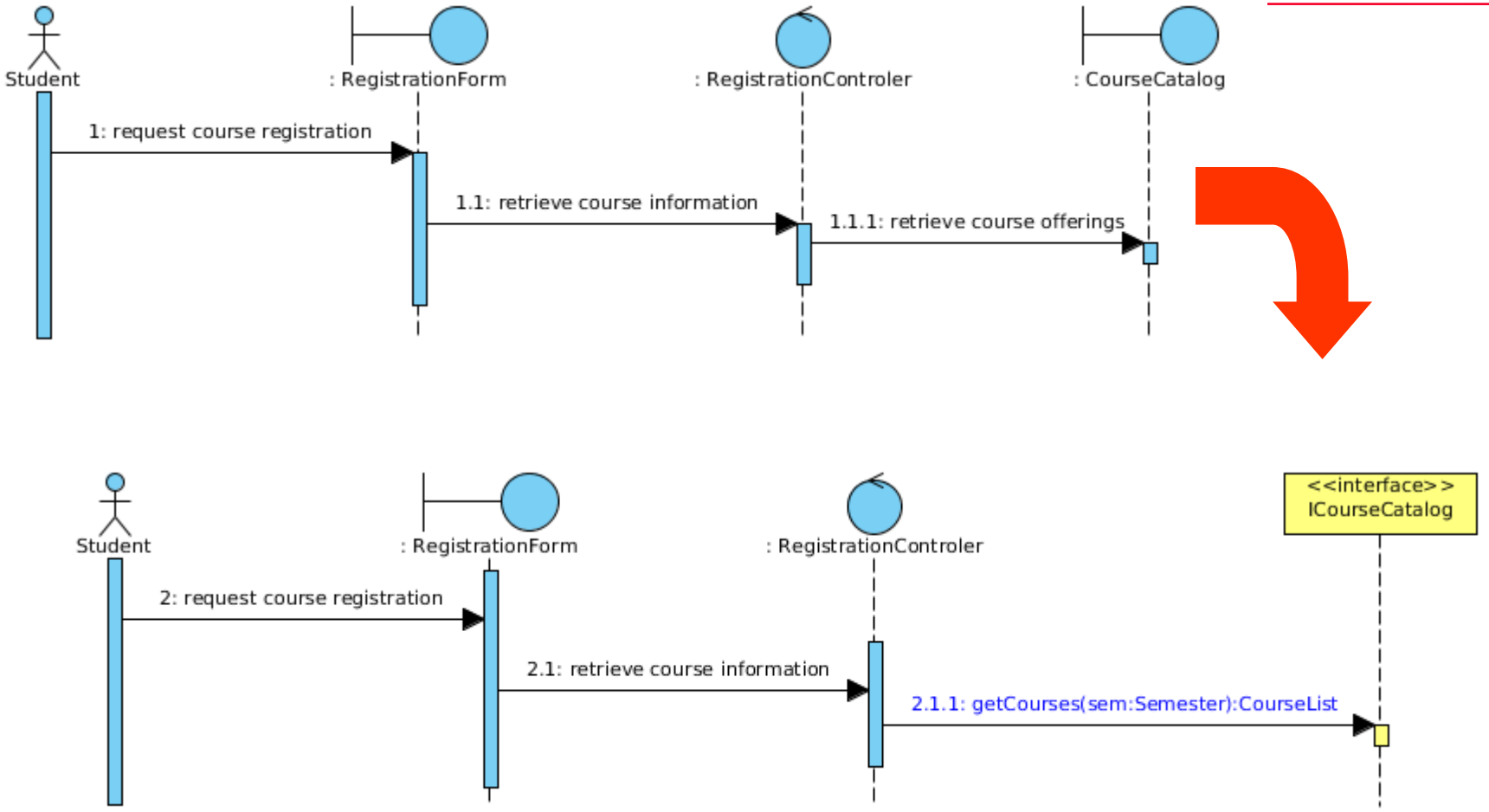
Fiecare relaţie la clasa de analiză iniţială trebuie înlocuită cu o relaţie echivalentă la interfaţa subsistemului.



Atenţie! Este posibil să fie introduse noi dependenţe pentru clasele client: aici, prin transformarea responsabilităţii în operaţie, *RegistrationController* depinde acum şi de *Semester* şi de *CourseList.*

Exemplu
INCORPORARE INTERFEȚE ÎN DIAGRAMA DE SECVENȚE

- Punerea în corespondenţă a claselor de analiză cu elemente de proiectare
- Identificarea subsistemelor şi a interfeţelor dintre subsisteme
- Actualizarea organizării modelului

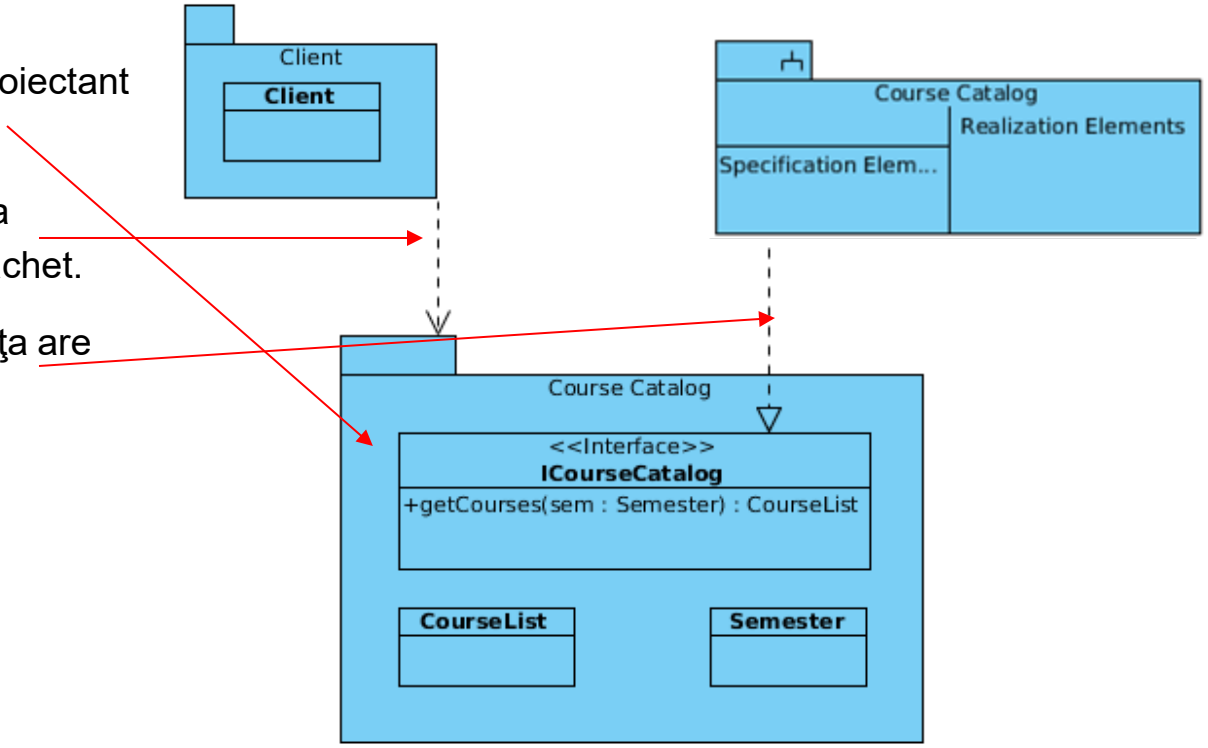


DEPENDENȚE ÎNTRE SUBSISTEME

- Punerea în corespondenţă a claselor de analiză cu elemente de proiectare
- Identificarea subsistemelor şi a interfeţelor dintre subsisteme
- Actualizarea organizării modelului

- Atenţie! Interfeţele oferite (şi/sau solicitate) de un subsistem sunt **în exteriorul** subsistemului.
- Deseori serviciile descrise de o interfaţă vor implica tipuri definite de proiectant (ex. *Semester* şi *CourseList*).

- Interfeţele şi tipurile definite de proiectant pot fi grupate într-un singur pachet.
- Pachetele care utilizează interfața au relaţii de dependenţă cu acest pachet.
- Subsistemul ce realizează interfaţa are relație de realizare cu acest pachet.



Evaluare formativă (2)

1. Ce relație există între clasele de analiză și clasele de proiectare ?

2. Indicații categorii de clase de analiză candidate să devină subsisteme.

3. Care sunt cele 2 zone de descriere a unui subsistem ? De ce este necesară separarea acestora ?

https://forms.gle/hN4MX15mMyw8oUSLA

PLAN CURS

- Modelul proiect
- Identificare elemente de proiectare
- **Proiectare clase**

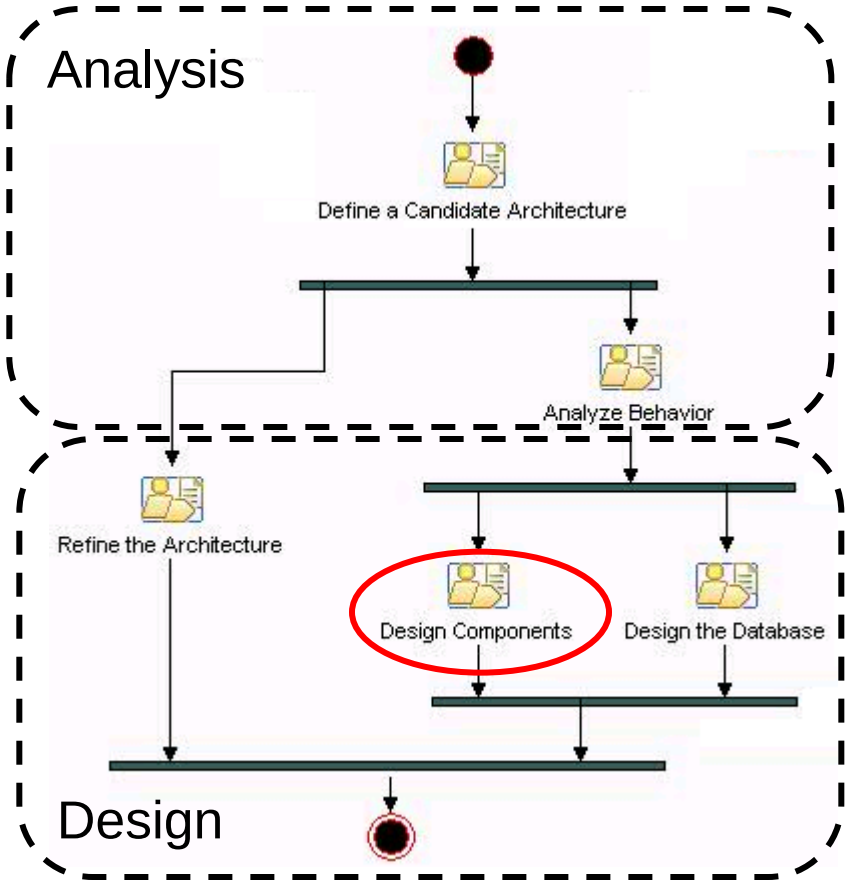
Etape

Analiză

- Analiză arhitecturală (definire arhitectură candidat)
- Analiza UC (analiză comportament)

Proiectare

- Identificare elemente de proiectare (rafinarea arhitecturii)
- Identificare mecanisme de proiectare (rafinarea arhitecturii)
- Proiectare clase (proiectare componente)
- Proiectare subsisteme (proiectare componente)
- Descrierea arhitecturii la execuţie şi a distribuirii (rafinarea arhitecturii)
- Proiectarea BD



PROIECTARE CLASE

Scop

- Rafinarea clasei în vederea asigurării comportamentului necesar realizărilor cazurilor de utilizare
- Definirea de informaţii suficiente pentru implementarea neambiguă a clasei
- Realizarea cerinţelor extra-funcţionale referitoare la clasă
- Incorporarea mecanismelor de proiectare utilizate de clasă

Rol responsabil

- Proiectantul

Etape majore

- Crearea claselor de proiectare
- Rafinarea claselor de proiectare

PROIECTARE CLASE

•Crearea claselor de proiectare
•Rafinarea claselor de proiectare

Strategii de proiectare specifice stereotipului de analiză* (<<boundary>>, <<control>>, <<entity>>)

- Clase <<boundary>>
  - Consideraţi utilizarea de subsisteme
  - Pentru GUI sunt disponibile şabloane bazate pe browser web
- Clase <<control>>
  - Sunt impactate direct de probleme de concurenţă şi distribuire.
  - Unele vor deveni operații ale altor clase
- Clase <<entity>>
  - De obicei sunt clase persistente.

- Considerarea modului în care şabloanele de proiectare (design patterns) pot fi utilizate pentru a ajuta la problemele de implementare.

- Considerarea modului în care mecanismele arhitecturale vor fi realizate în termeni de clase definite la proiectare.

*Obs. Stereotipurile de analiză nu sunt păstrate în design; ele au fost utile pentru a analiza rolurile jucate de obiecte, pentru a separa corect comportamentul.

PROIECTARE CLASE

•Crearea claselor de proiectare
•Rafinarea claselor de proiectare

Multe clase, simple, înseamnă că fiecare clasă:

- Încapsulează mai puţin din logica totală a sistemului
- Este mai reutilizabilă
- Este mai uşor de implementat

Puţine clase, complexe, înseamnă că fiecare clasă:

- Încapsulează o parte importantă din logica sistemului
- Este mai puţin probabil să fie reutilizabilă
- Este mai dificil de implementat

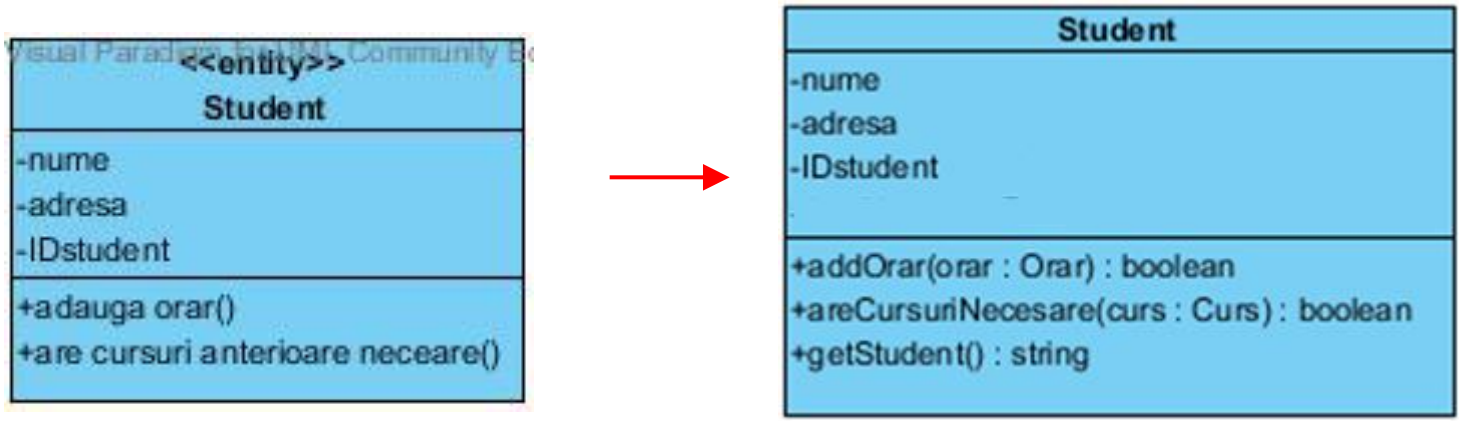
SRP (Single Responsibility Principle)

O clasă ar trebui centrată în jurul unui singur obiectiv!

O clasă trebuie să facă un singur lucru, pe care trebuie să-l facă bine!

DEFINIREA OPERAȚIILOR

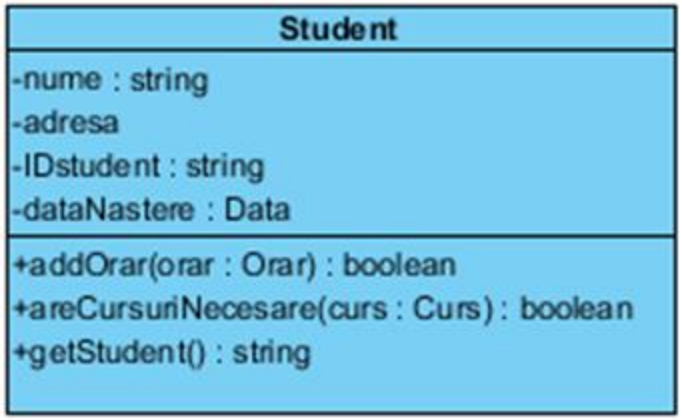
- Operaţiile derivate din responsabilităţile definite la analiză
  - Specificarea numelui operaţiei şi a signaturii complete (parameterii şi valoarea de return).
- Operaţiile adiţionale
  - Operaţii nedefinite explicit la analiză (ex. `getters/setters`)
  - Funcţii manager (ex. constructori, destructori)
  - Funcţii pentru copiere obiecte, pentru testare egalitate, pentru testare relaţii opţionale (ex. `eAsignat(Profesor, Curs)`), etc.
  - Funcţii Helper (de obicei `private` sau `protected`)



DEFINIREA ATRIBUTELOR

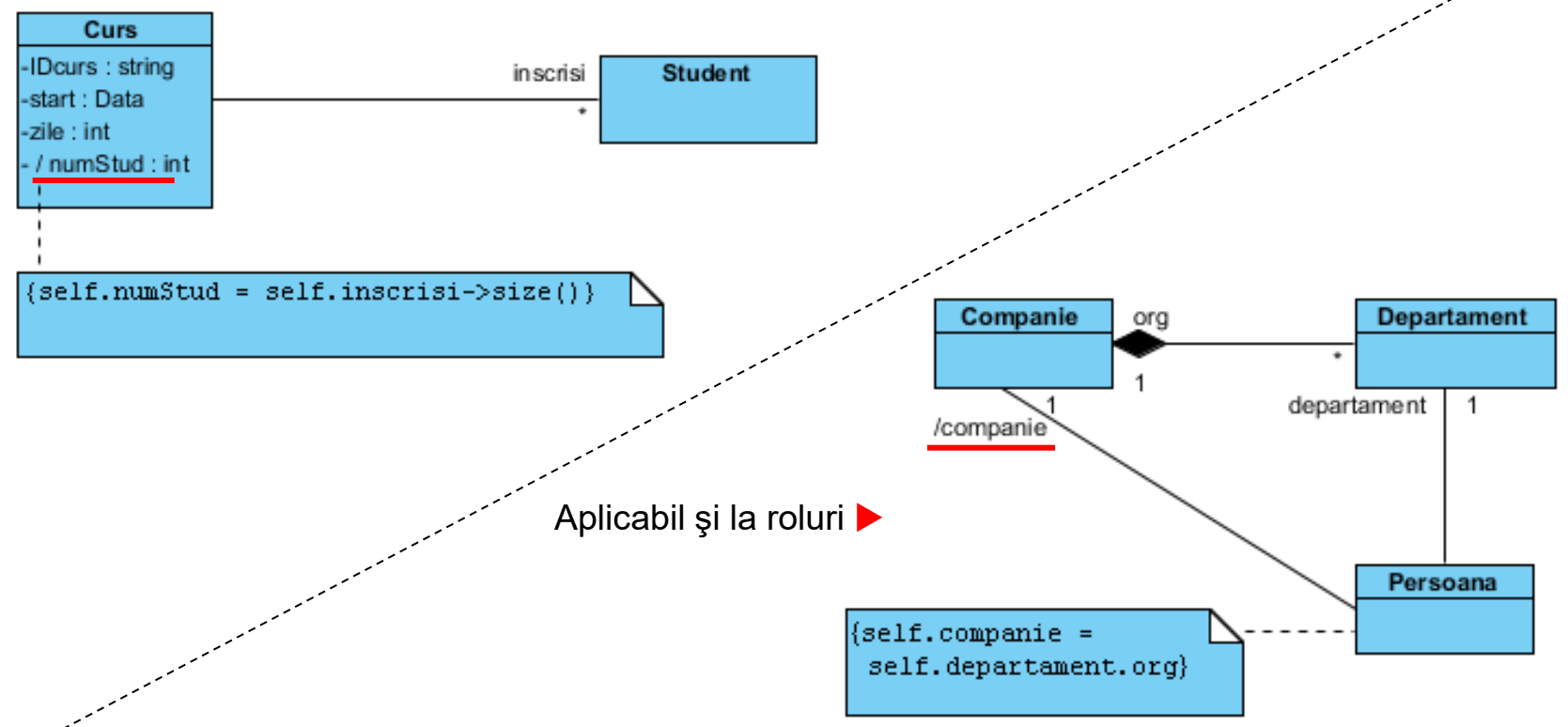
- Atribute derivate din atributele identificate la analiză
  - Specificare nume, tip şi, opţional, valoare implicită
  - Vizibilitate `private` în majoritatea cazurilor
- Tipurile de date pot fi built-in (UML2 sau altele), definite de utilizator, sau clase definite de utilizator
  - Se recomandă să nu se utilizeze tipuri de date specifice limbajului de implementare.

Atributul `adresa` *poate fi tipat ca* `String` sau ca o nouă clasă `Adresa`



ATRIBUTE DERIVATE

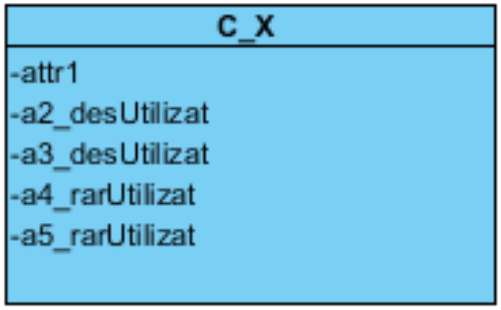
- Calculate pe baza valorilor altor atribute, introduse în mod tipic din motive de performanţă
- Identificate prin “/”



Exemplu
RAFINAREA CLASELOR

Analiză

Clasă asociată mecanismului de persistenţă.

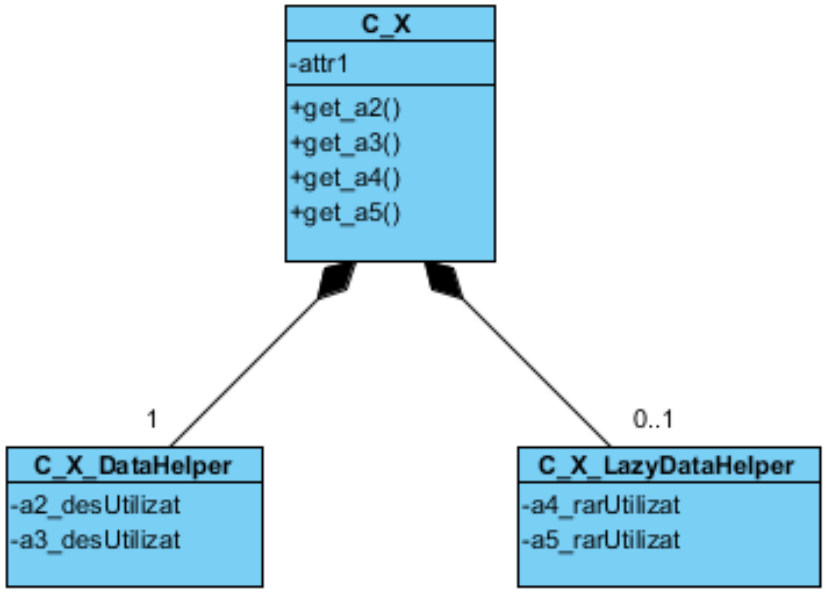


Din examinrea cazurilor de utilizare s-a constatat că:

•Atributul `attr1` – nu e persistent ci e utilizat la runtime pentru evidenţă.

•Doar 2 atribute sunt utilizate frecvent.

Proiectare



La proiectare s-a decis ca atributele utilizate frecvent să fie extrase imediat din baza de date, iar cele rar utilizate să fie extrase la cerere.

Clientul are acces la clasa `C_X` care funcţionează ca proxy pentru două clase persistente în mod real.

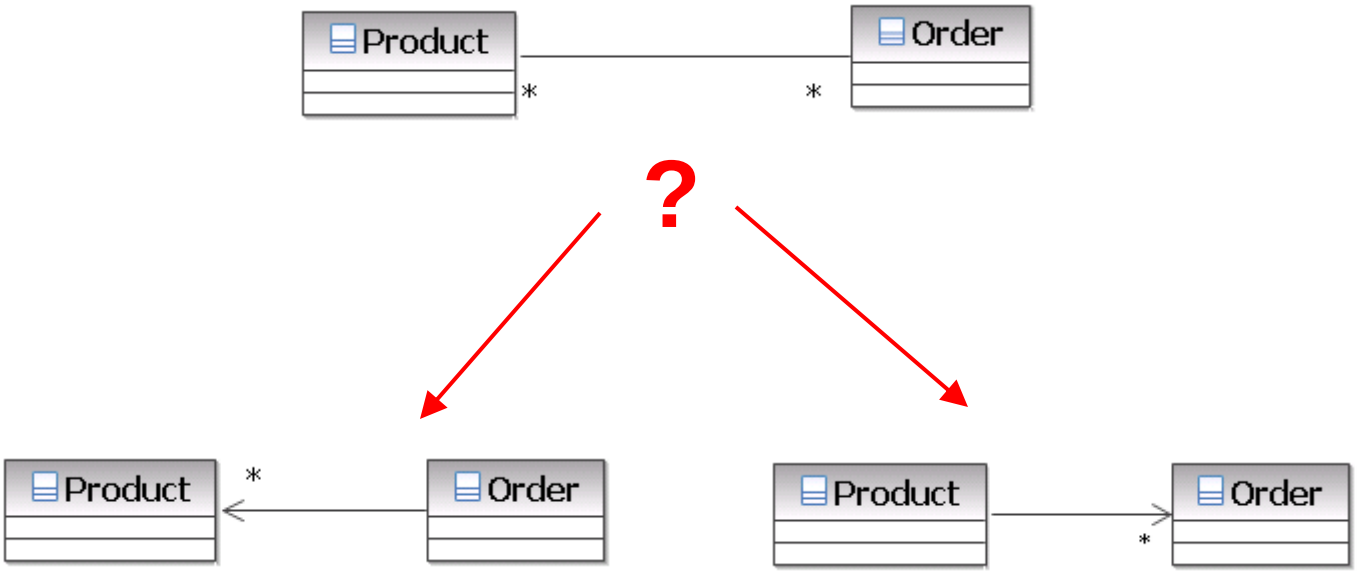
Aceasta va extrage `C_X_DataHelper` din baza de date la primul acces. Clasa `C_X_LazyDataHelper` va fi extrasă doar în cazurile (rare) în care clientul solicită acele atribute.

RAFINARE RELAȚII

- Navigabilitate
- Multiplicitate
- Generalizare vs. agregare
- Factorizare şi delegare
- Refactorizare

RELAȚII
NAVIGABILITATE

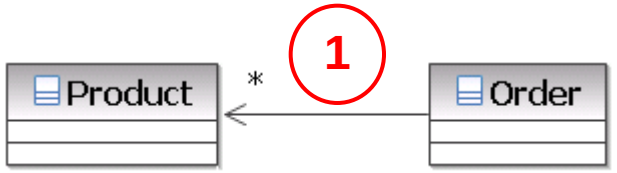
Restricţionarea navigabilităţii reduce dependenţele şi creşte reutilizarea.



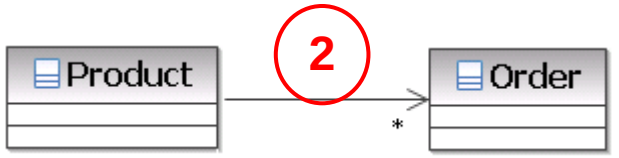
RELAȚII
NAVIGABILITATE

Variante

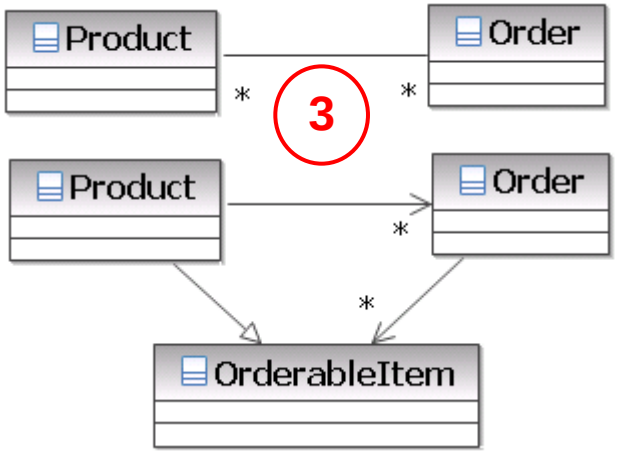
1. Numărul total de ordine este mic sau o listă de ordine care referă un anumit produs este rareori necesară.



2. Numărul total de produse este mic sau o listă de produse incluse într-un ordin dat este rareori necesară.



3. Numărul de produse și de ordine nu este mic şi este necesară navigarea în ambele direcţii.

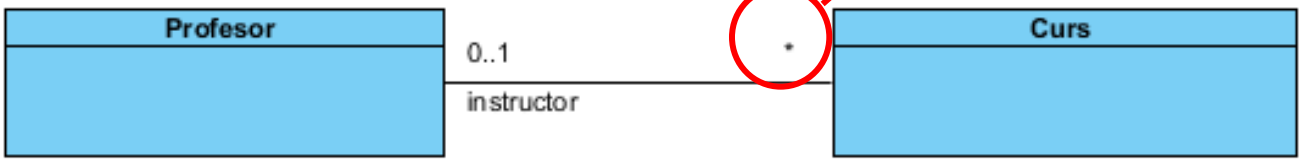


RELAȚII
MULTIPLICITATE

- Multiplicitate = 1 sau multiplicitate = 0..1
  - Implementată direct în clasa asociată (`Curs`), ca valoare simplă sau ca pointer
  - Nu este necesară proiectare ulterioară

- Multiplicitate > 1
  - Nu se poate utiliza o valoare simplă sau un pointer în clasa asociată (`Profesor`)
  - Este necesară proiectare ulterioară

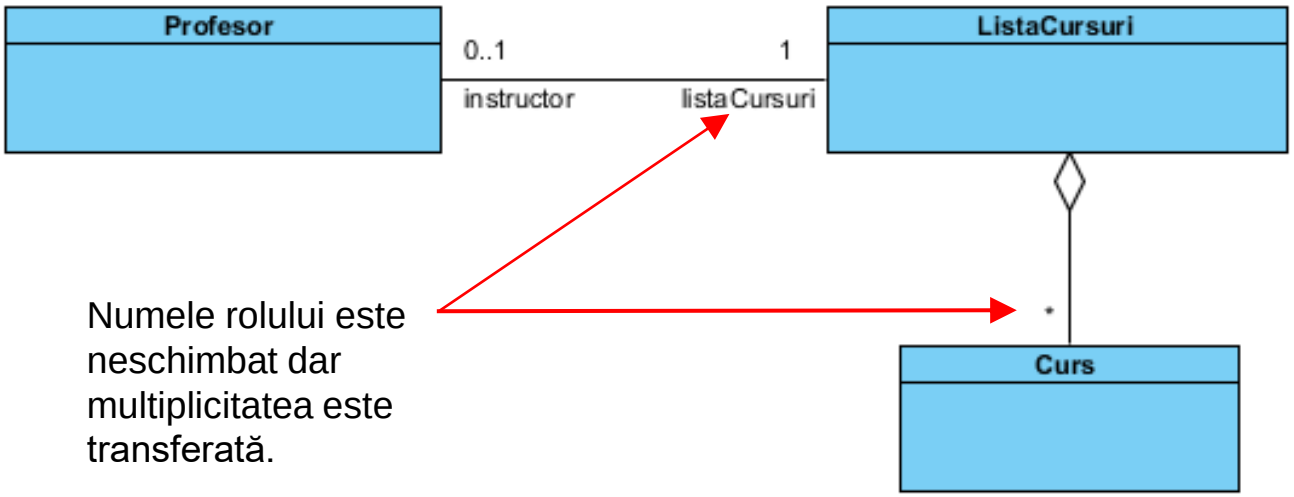
Necesită un container pentru obiecte de tip Curs în clasa asociată `Profesor`.



MODELARE CLASĂ CONTAINER

Multiplicitatea (n > 1) poate fi modelată explicit...

...SAU poate implica utilizarea unei clase container.

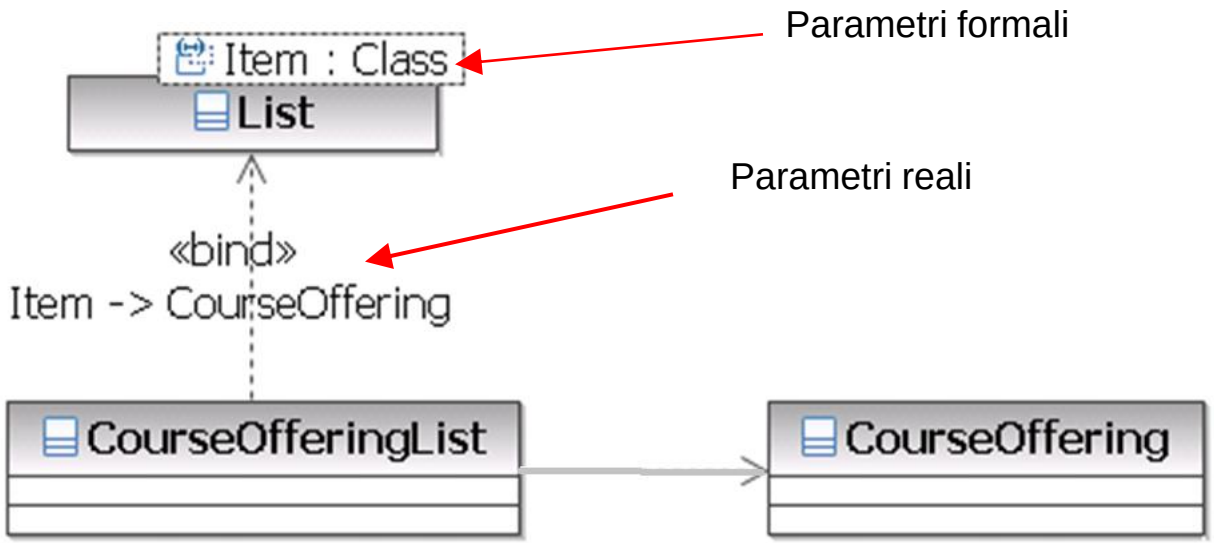


Numele rolului este neschimbat dar multiplicitatea este transferată.

CLASE PARAMETRIZATE

Clasă parametrizată = definiţie de clasă ce defineşte alte clase.

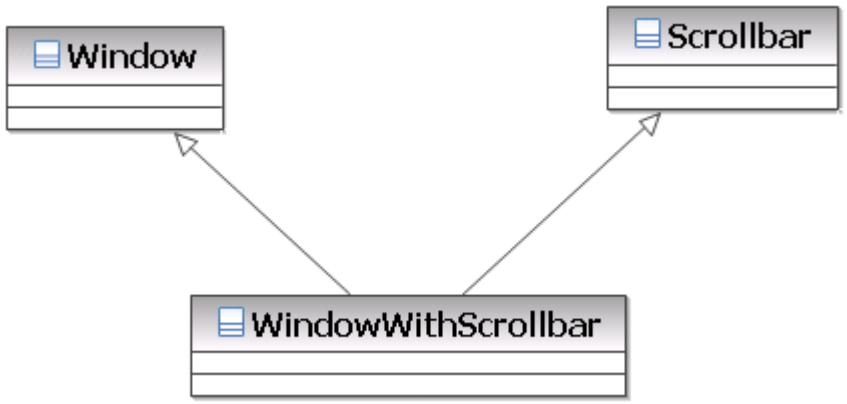
- Denumire UML : “template”
- Utilizare frecventă ca clase container
  - Seturi, liste, dicţionare, stive, cozi
- Concept disponibil la nivel de limbaj în C++, Java, ...



RELAȚII
GENERALIZARE vs AGREGARE

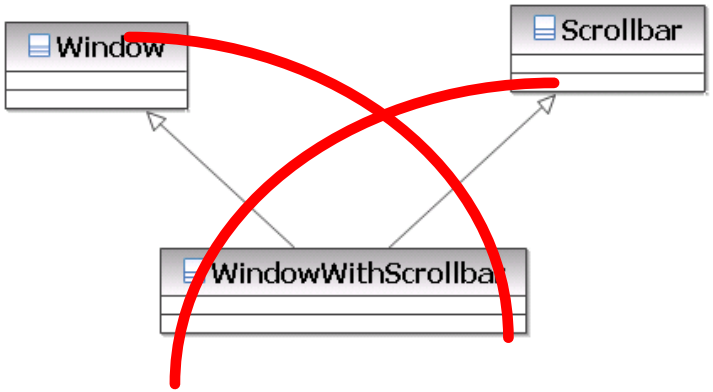
Generalizarea şi agregarea sunt deseori confundate !!!

- Generalizarea reprezintă o relaţie "***is a***" sau "***kind-of***"
- Agregarea reprezintă o relaţie "***part-of***"



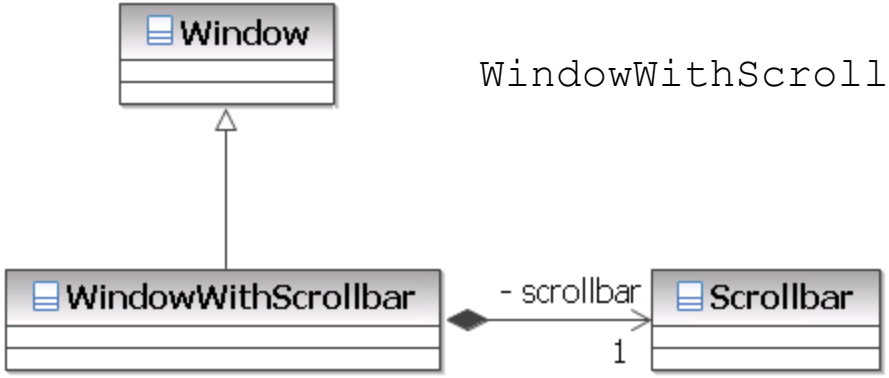
Este corect?

RELAȚII
GENERALIZARE vs AGREGARE



`WindowWithScrollbar` “is a” `Window`

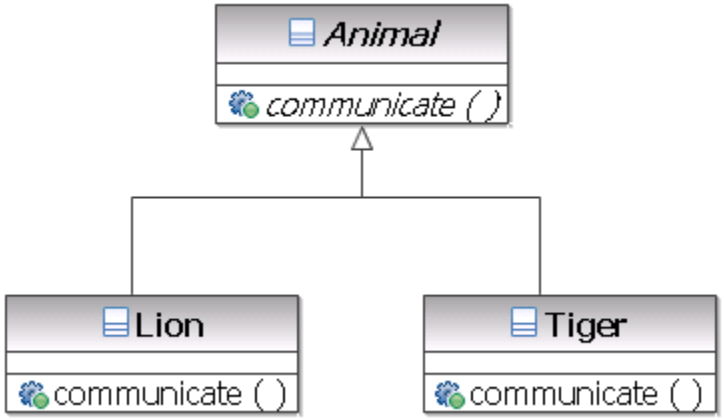
`WindowWithScrollbar` “contains a” `Scrollbar`

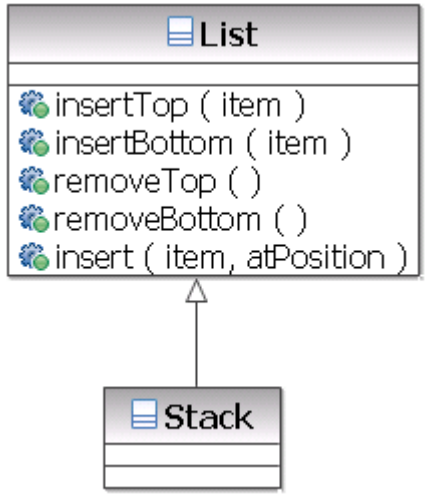


RELAȚII
GENERALIZARE

Principiul Liskov de substituţie: (?)

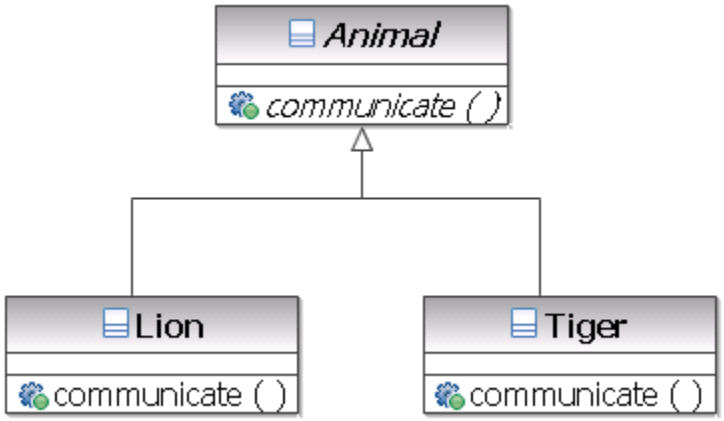
- *Un obiect de tip T se poate înlocui cu orice instanţă a oricărui subtip al lui T*.

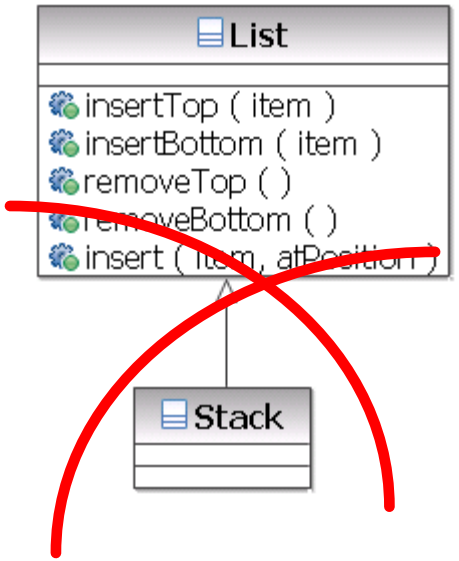




Sunt corecte ?

RELAȚII
GENERALIZARE





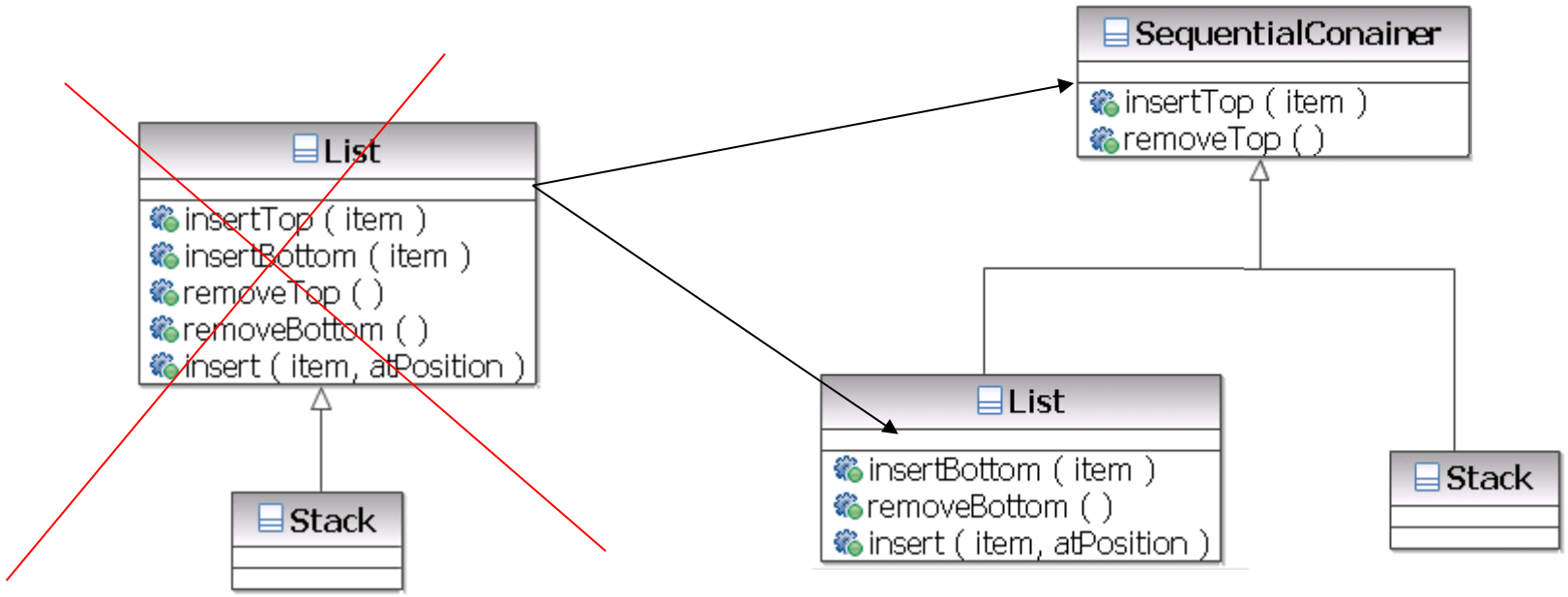
PARTAJAREA IMPLEMENTĂRII
FACTORIZARE

Reutilizarea implementării unei alte clase, prin factorizare:

Clasa este factorizată în 2 clase :

- superclasă care va conține implementarea reutilizată
- subclasă care va conține restul operațiilor

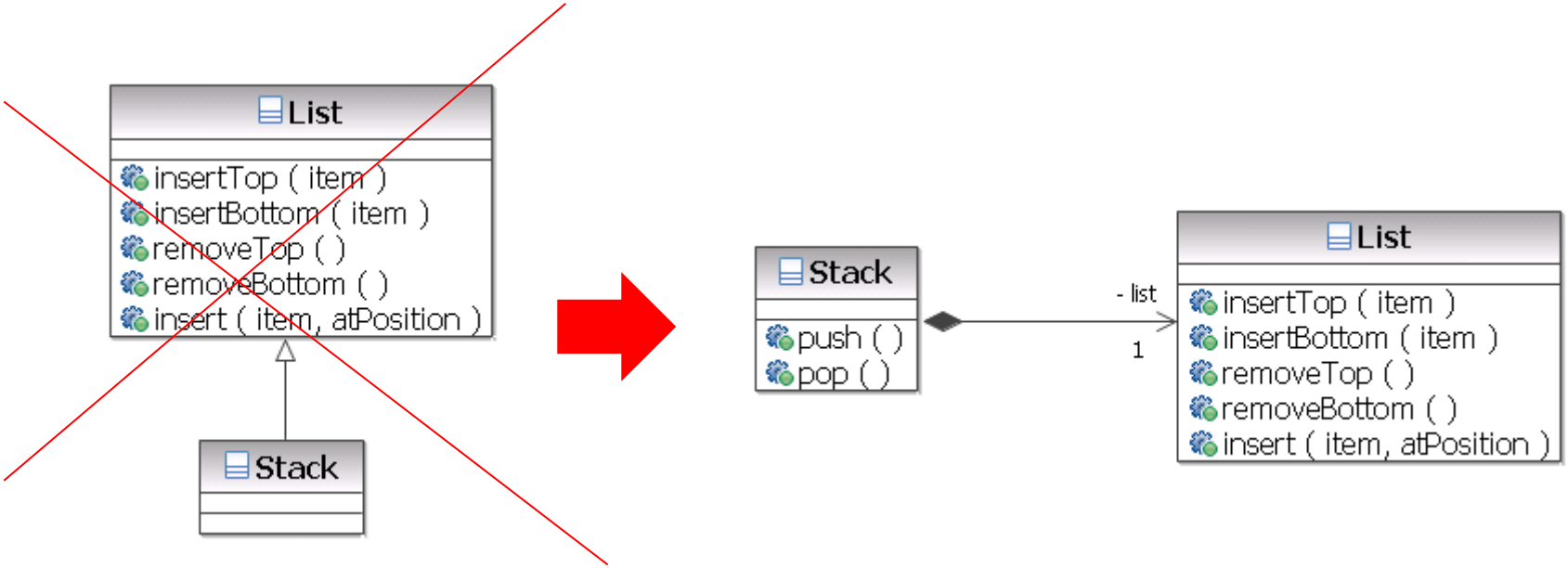
OBS. Factorizarea nu poate fi aplicată dacă clasa “reutilizată” nu poate fi modificată.



PARTAJAREA IMPLEMENTĂRII DELEGARE

- Reutilizarea implementării unei alte clase, prin delegare

Poate fi aplicată şi dacă clasa “reutilizată” nu poate fi modificată



Se foloseşte o relaţie de compoziţie pentru a “reutiliza” funcţionalitatea dorită.

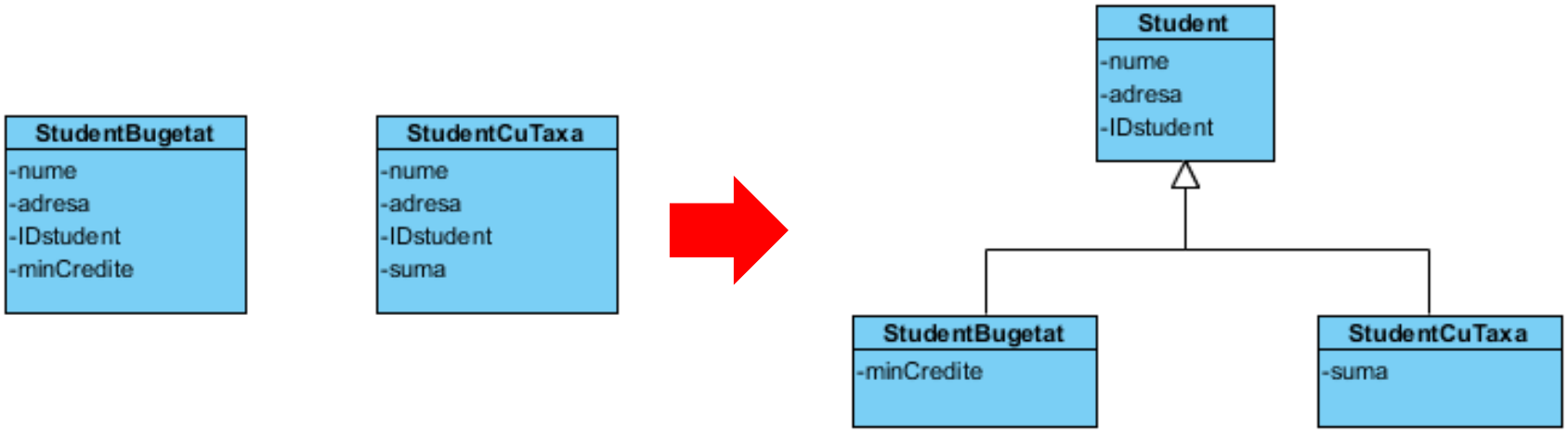
Toate operaţiile ce necesită serviciul “reutilizat” sunt transferate instanţei clasei reutilizată.

În exemplul de mai sus clasa `List` este clasa conţinută în clasa `Stack`.

Exemplu
RAFINAREA RELAȚIILOR

- În universitate exită studenţi bugetaţi şi studenţi cu taxă
  - Pentru studenţii cu taxă trebuie memorată suma de plată.
  - Studenţii bugetaţi trebuie să aibă un număr minim de credite.

- Se poate crea o generalizare pentru a factoriza datele comune
  - Dar ce se întâmplă dacă un student cu taxă devine student bugetat?

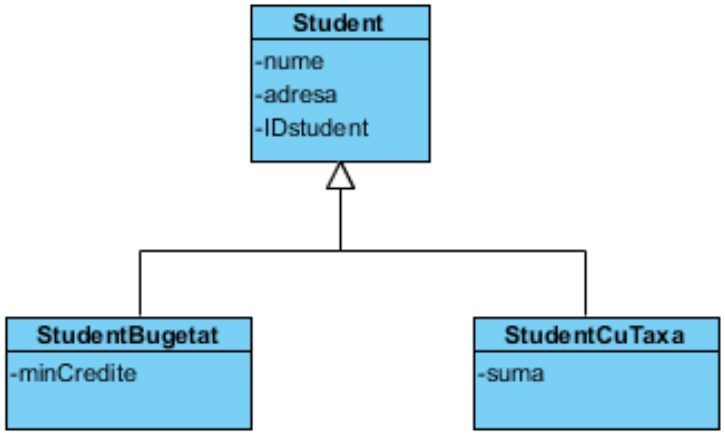


Exemplu
RAFINAREA RELAȚIILOR

Ce se întâmplă dacă un student cu taxă devine student bugetat?

Paşii transformării unui student cu taxă în student bugetat:

1. Crearea unui obiect de tip `StudentBugetat`
2. Copierea datelor comune din obiectul de tip `StudentCuTaxa` existent în noul obiect de tip `StudentBugetat`
3. Notificarea tuturor clienţilor obiectului de tip `StudentCuTaxa`
4. Distrugerea obiectului de tip `StudentCuTaxa`



Problema se complică dacă se doreşte păstrarea unui istoric pentru fiecare student.

Evaluare formativă (3)

1. Ce presupune rafinarea claselor de analiză referitor la atribute și operații ?

2. Ce înțelegeți prin clasă container ?

3. Sub ce forme se poate proiecta reutilizarea unei părți din implementarea unei clase ?

https://forms.gle/RbhHwFThyrjMm78B7

Bibliografie

Roger S. Pressman, **Software Engineering. A Practitioner's Approach**, ed.7, McGraw-Hill, 2010.

capitolele 6 - 14

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