Analiza și proiectarea sistemelor software

Curs 7

Şabloane de proiectare

Un şablon de proiectare (design pattern) este un mod de a reutiliza *cunoaştere abstractă* despre o *problemă* şi despre *soluţia* sa.

Un şablon este o descriere a *problemei* şi a *esenţei soluţiei*.

Trebuie să fie suficient de *abstract* pentru a fi *reutilizat* în diferite contexte.

Explicați !

Ce proprietăți ale OO stau la baza șabloanelor de proiectare ?

Deseori şabloanele sunt bazate pe caracteristici ale obiectelor cum ar fi *moştenirea* şi *polimorfismul*.

Şabloane de proiectare

Def. Şablon de proiectare = *Model de abordare* a *soluţionării* unui set tipic de *probleme* într-un anumit *context*.

PROPRIETĂȚI

- Numele
- Problema
- Motivaţia
- Contextul
- Forţele
- Soluţia
- Intenţia
- Colaborările
- Consecinţele
- Implementarea
- Utilizări cunoscute
- Şabloane înrudite

"Each pattern is a three-part rule, which expresses a relation between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain software configuration which allows these forces to resolve themselves."

Richard Gabriel, *Patterns of Software: Tales From The Software Community*

"Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution."

http://www.bradapp.com/docs/patterns-intro.html

Elementele unui şablon

- Numele
- Problema
- Motivaţia
- Contextul
- Forţele
- Soluţia
- Intenţia
- Colabărările
- Consecinţele
- Implementarea
- Utilizări cunoscute
- Şabloane înrudite

Nume

Un *identificator sugestiv*.

Descrierea problemei.

Descrierea soluţiei.

Nu un design concret ci un *model de abordare* (şablon, template) pentru o soluţie de proiectare, care poate fi *instanţiat* în diferite moduri.

Consecinţe

Rezultatele şi *compromisurile* (trade-offs) aplicării şablonului.

Exemplu :

Şablonul Observer

Extras din catalog de şabloane de proiectare.

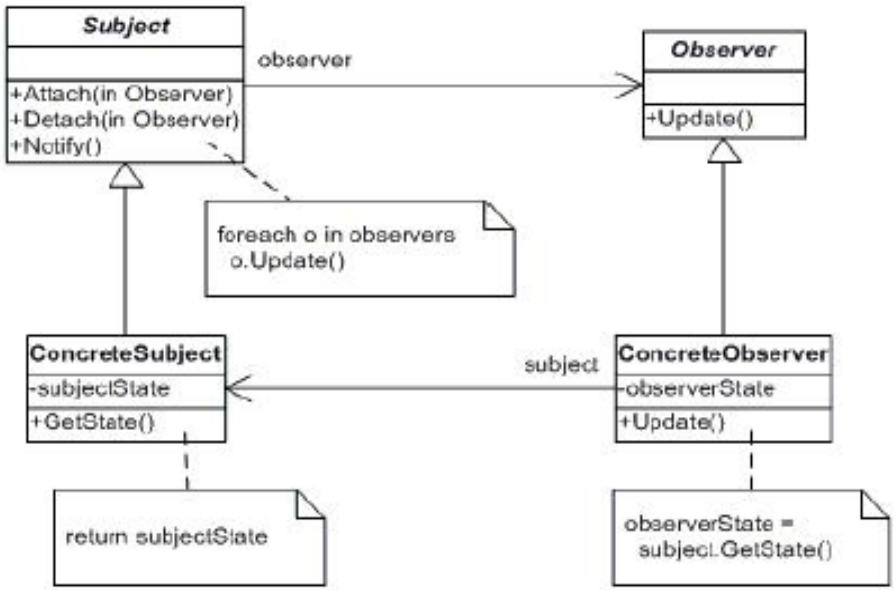
Name:

OBSERVER

Intent:

Defines a one-to-many dependency between objects so that when one object changes its state, all of its dependants are notified and updated automatically.

Structure:



Participants:

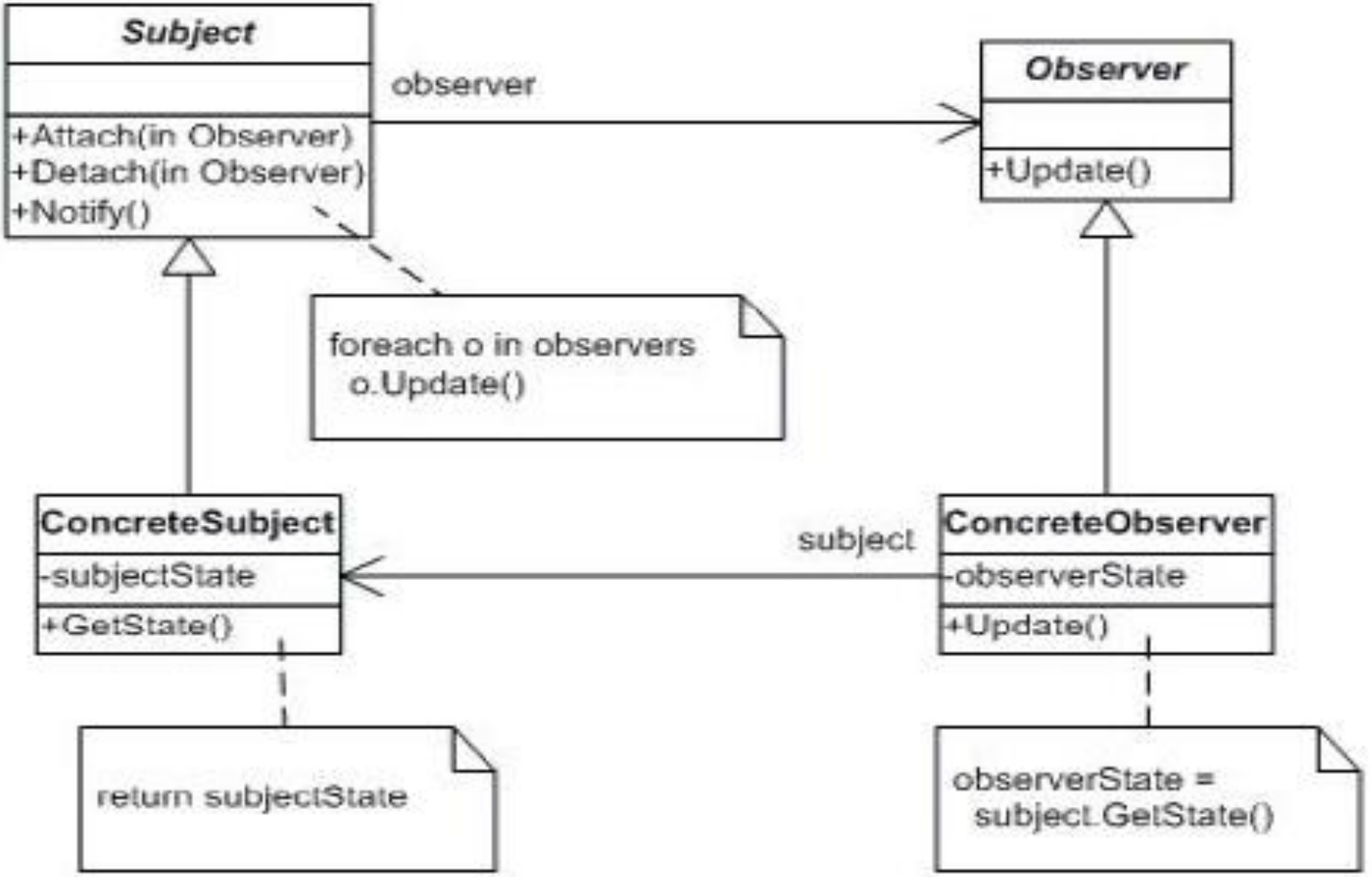
`Subject` (the publisher from the metaphor) – publishes its state.
`Observer` (the subscriber from the metaphor) – dependant, automatically notified and updated when the state of the Subject changes.

When data in `Subject` changes, each `Observer` is notified.
`Observer` has registered (subscribed to) with the `Subject` to receive updates when the data in the `Subject` changes.

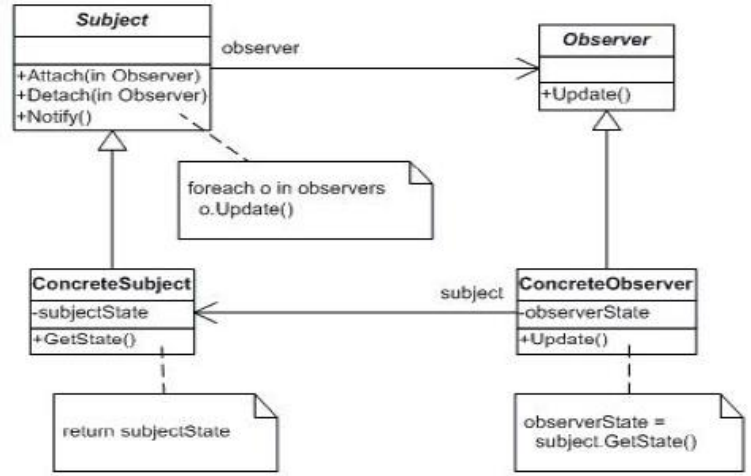
Aplicability when:

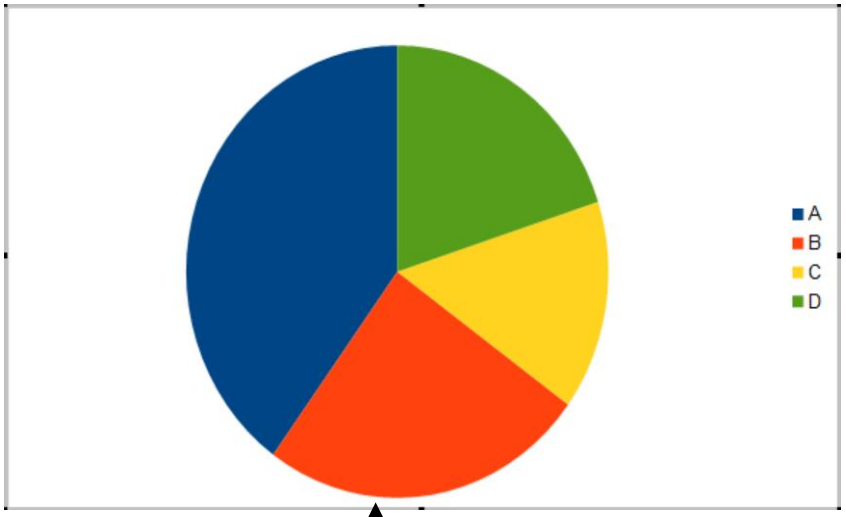
- an abstraction has two aspects, one dependent on the other; encapsulating these aspects in separate objects lets you vary and reuse them independently.
- a change to one object requires changing others, and you do not know how many objects need to be changed.
- An object should be able to notify other objects without making assumptions about who these objects are; in other words, you do not want these objects tightly coupled.

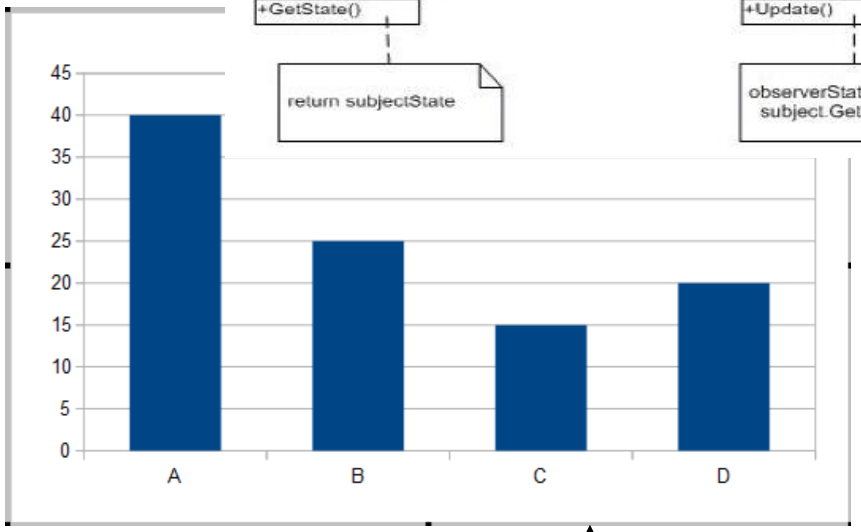
Şablonul Observer



Exemplu utilizare - Afişări multiple







Observer 1

Subject

A=40
B=25
C=15
D=20

Observer 2

Realizați corespondența între obiectele din imagine și șablon.

Utilizare şabloane în proiectarea sistemelor software

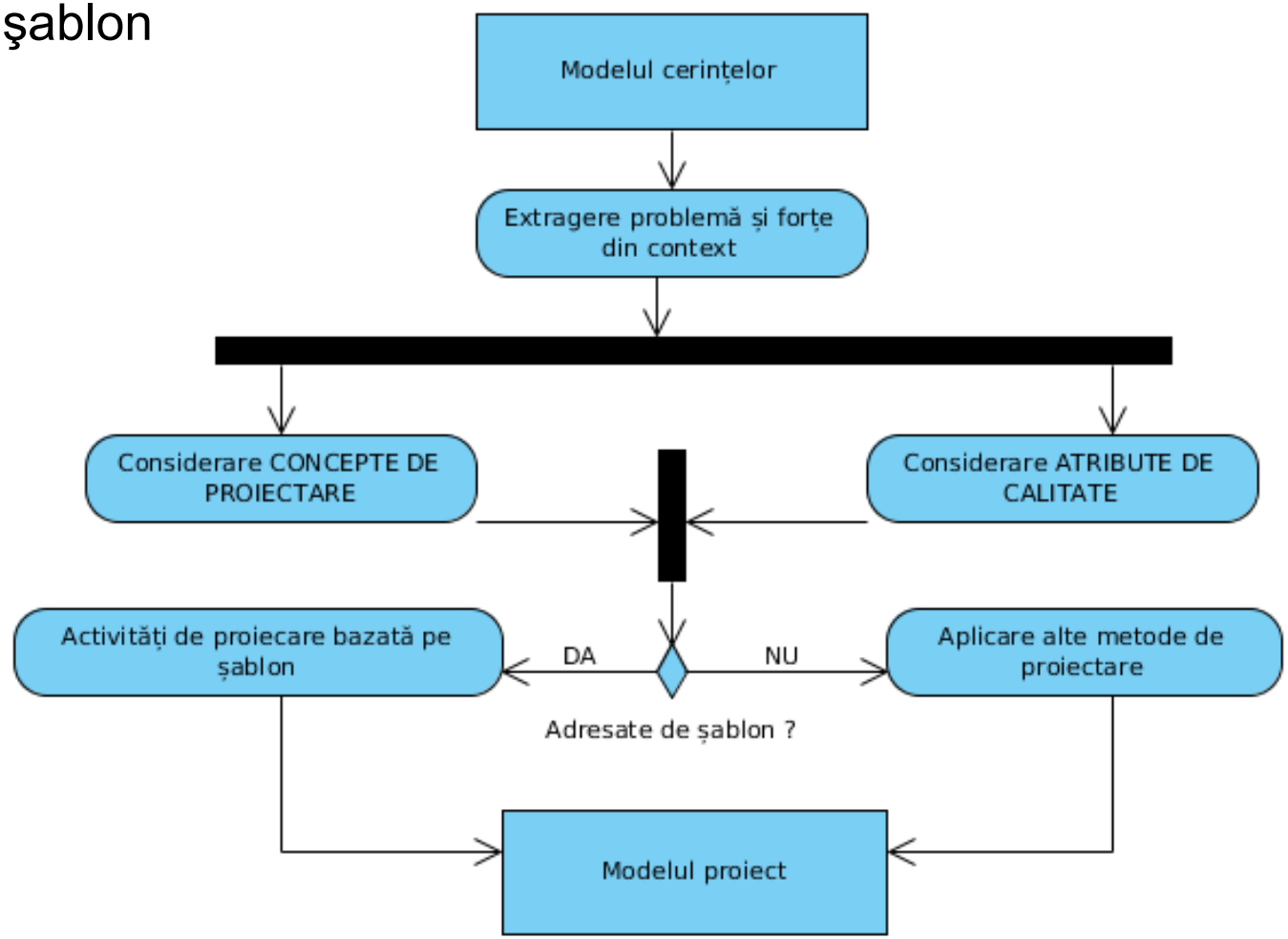
- *Partiţionarea* spaţiului problemei în *subprobleme*.
- *Căutare şabloane* de soluţionare a subproblemelor.
- *Adaptarea şabloanelor* aplicabile la necesităţile specifice ale aplicaţiei de dezvoltat.
- *Creare soluţii specifice* pentru subproblemele la care nu există şablon aplicabil.

Procedură generală:

1. înţelegerea contextului
2. alegerea şabloanelor din categoria nivelului curent de abstractizare
3. căutare şabloane pentru nivelul inferior de abstractizare
4. repetare 1-3 până la o soluţie de proiectare completă
5. Rafinare model proiect la specificul software-lui de construit

Utilizare şabloane în proiectarea sistemelor software

Aplicarea unui şablon



Surse de şabloane pentru proiectarea sistemelor software

Colecții de şabloane de proiectare:

http://hillside.net/patterns/

http://c2.com/ppr/index.html

http://c2.com/cgi/wiki?PatternIndex

Colecţii de şabloane pentru UI:

www.hcipatterns.org/patterns

Şabloane de proiectare specializate

www.objectarchitects.de/arcus/cookbook - Business Information Systems

Categorii de şabloane
Clasificare 2

- *arhitecturale* – descriu probleme generale de proiectare soluţionate folosind o abordare structurală

- *date* – soluţii de modelare pentru probleme recurente orientate pe date

- *componente* – modele de soluţii pentru probleme asociate cu dezvoltare de subsisteme/componente

- *interfaţă* – soluţii pentru probleme comune ale interfeţei cu utilizatorul în contextul caracteristicilor specifice utilizatorului final

Tipuri de şabloane pentru proiectare OO

- *creaţionale* – soluţii pentru probleme de creare, compunere şi reprezentare a obiectelor

- *structurale* – soluţii pentru probleme de organizare şi integrare a claselor şi obiectelor

- *comportamentale* – soluţii pentru probleme de asignare de responsabilităţi şi de comunicare între obiecte

Cum sunt clasificate șabloanele de proiectare OO în GoF (**Design Patterns**: Elements of Reusable Object-Oriented Software) ?

Alte exemple

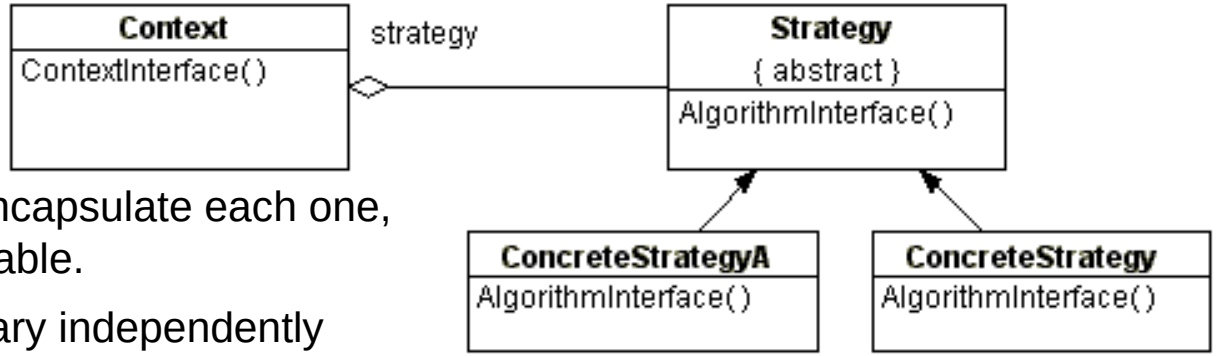
Name:

STRATEGY (POLICY)

Intent:

Design a family of algorithms, encapsulate each one, and make them interchangeable.

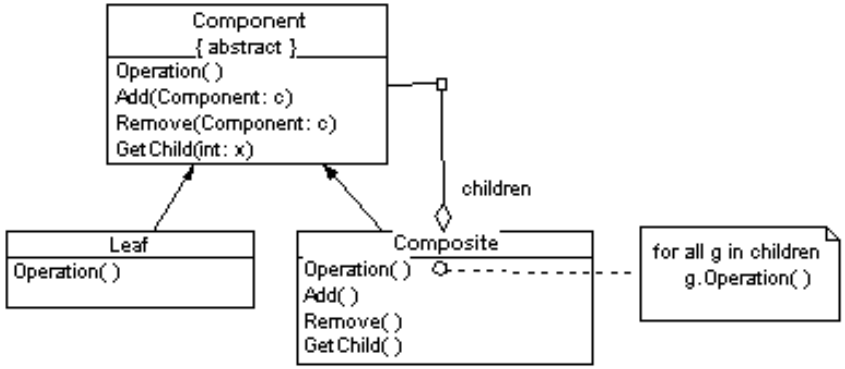
STRATEGY lets the algorithm vary independently from clients that use it.



Name:

COMPOSITE

Intent:



Allows to compose objects into tree structures to represent part-whole hierarchies.

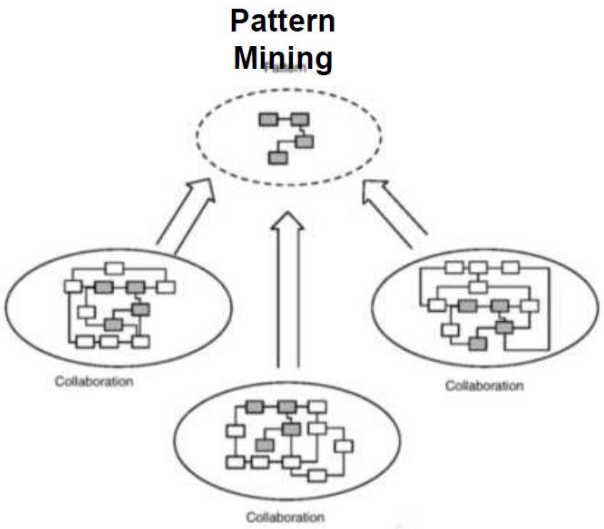
COMPOSITE lets clients treat individual objects and compositions of objects uniformly.

As a consequence, the same operation can be applied over composites and individual objects.

PLAN CURS

- **Utilizare șabloane în proiectarea software-lui**
- Identificarea mecanismelor de proiectare
- Proiectare subsisteme

UTILIZARE ȘABLOANE ÎN PROIECTAREA SOFTWARE-lui



Proiectantul (inginer software)

- Dezvoltă abilitatea de a vedea
  - şabloane ce caracterizează problema
  - şabloane corespunzătoare ce pot fi combinate

  pentru construirea soluţiei.

- Va căuta toate oportunităţile de reutilizare a şabloanelor (de proiectare) existente, înainte de a încerca crearea unora noi.

- Modelul proiect (design) - optimizare a modelului analiză.
- Șabloanele de proiectare (design patterns) - concept general al modului de optimizare a modelului analiză într-un mod particular și cu efecte particulare.

UTILIZARE ȘABLOANE ÎN PROIECTAREA SOFTWARE-lui

Tipuri :

funcţie de:
- nivelul de abstractizare
- gradul în care direcţionează activitatea de codificare

Şabloan **arhitectural** :
- *structura* de ansamblu a software-lui
- *relaţiile* dintre subsistemele şi componentele software
- *regulile* de specificare a relaţiilor dintre elementele arhitecturii (clase, pachete, componente, subsisteme)

Şabloan **de proiectare** :
- adresează un anumit *element al design-ului* (ex. agregarea unor componente, mecanisme de comunicare între componente, relaţii între componente)**.**

Idiom (şabloan **de codificare**, *specific limbajului*) :
- implementarea unui element algoritmic al unei componente,
- protocol specific de interfaţă,
- mecanism de comunicare între componente.

UTILIZARE ȘABLOANE ÎN PROIECTAREA SOFTWARE-lui

CADRU (FRAMEWORK)

Infrastructură scheletică pentru implementare.

Miniarhitectură (nu e şablon arhitectural !) reutilizabilă:

- *structură şi comportament generice* pentru o familie de abstractizări software,
- un *context* care specifică *colaborările* şi utilizarea în cadrul unui anumit domeniu.
- Include, de obicei, șabloane de proiectare

Proiectantul integrează clase şi funcţionalitate specifice problemei.

Structură scheletică ce conţine puncte de racordare (plug points, hooks, slots) ce permit adaptarea la o anumită problemă a domeniului.

Framework (în context OO) = colecţie de clase cooperante.

UTILIZARE ȘABLOANE ÎN PROIECTAREA SOFTWARE-lui

ŞABLON DE PROIECTARE (design pattern)

Def. Şablon de proiectare = soluţie generală reutilizabilă la o problemă comună de proiectare de software.

Def. Limbaj de şabloane = colecţie organizată de şabloane specifice unui anumit domeniu.

Nu este o soluţie finală ce poate fi direct transformată în cod, ci este *descrierea unui model de rezolvare a problemei*.

Descrie o structură comună, recurentă, de componente care comunică pentru a rezolva o problemă generală de proiectare într-un context particular.

Şablon de proiectare - soluţionează o problemă de proiectare.

vs.

Algoritm – soluţionează o problemă computaţională.

ŞABLOANE DE PROIECTARE
Exemple

Popularizate de Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (the “Gang of Four”) în Design Patterns, Elements of Reusable Object-Oriented Software, Addison Wesley, 1994

Exemple:

Şablon	Exemple de aplicare
Command (şablon comportamental)	Lansează o cerere către un obiect fără a cunoaşte ceva despre operaţia cerută sau despre receptorul cererii (ex. răspunsul la o opţiune de menu, o cerere de tip undo, procesarea expirării unui timer).
Abstract factory (şablon creaţional)	Crează obiecte GUI (butoane, bare de derulare, ferestre, etc.) independent de sistemul de operare OS: ca o consecinţă, aplicaţia poate fi portată uşor pe diferite medii.
Proxy (şablon structural)	Gestionează obiecte distribuite în manieră transparentă pentru obiectele client (*remote proxy*) Încarcă un obiect grafic de dimensiuni mari sau orice entitate “costisitoare” pentru a o crea/initializa doar când este necesară (*la cerere*) şi în mod transparent (*virtual proxy*)
Observer (şablon comportamental)	La modificarea stării unui obiect, obiectele dependente sunt notificate. Obiectul modificat este independent de obiectele observatori.

ŞABLOANE DE PROIECTARE

- Şabloanele de proiectare sunt şabloane de dimensiuni mici şi medii, mai mici decât şabloanele arhitecturale, independente de limbajul de programare.
- Datorită nivelului lor de abstractizare, şabloanele de proiectare tind să aibă aplicabilitate independentă de domeniul aplicaţiei.
- Șabloanele de proiectare sunt utilizate la terminarea procesului de analiză, atunci când s-a obținut o reprezentare detaliată a problemei şi a constrângerilor impuse de aceasta.
- Atunci când un şablon de proiectare este conectat la o aplicaţie, el formează o parte a modelului concret de proiectare.
- Un şablon de proiectarea aplicat poate reprezenta o porţiune a unui mecanism de proiectare.

DESCRIERE ŞABLON DE PROIECTARE

Nume : expresiv, descrie esenţa şablonului.

Intenţie : ce face şablonul.

Sinonime : listă.

Motivaţie : exemplu de problemă.

Aplicabilitate : situaţii specifice de proiectare în care se poate aplica.

Structură : clasele implicate în implementare.

Participanţi : responsabilităţile claselor implicate.

Colaborări : modul în care colaborează clasele participante pentru îndeplinirea responsabilităţilor.

Consecinţe : “forţele de proiectare” ce afectează şablonul şi potenţialele compromisuri ce trebuie considerate la implementarea lui.

Şabloanele cu care se află în relaţie.

ŞABLOANE DE PROIECTARE

Exemplu: Construirea unei componente GUI generice

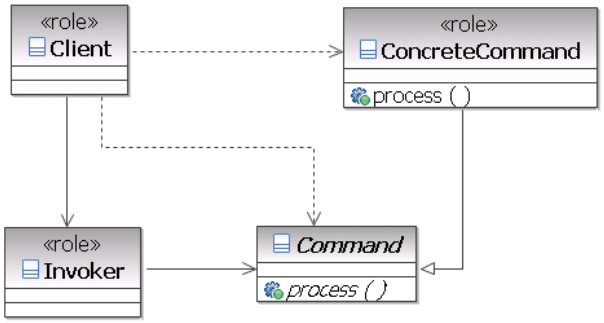
Problema

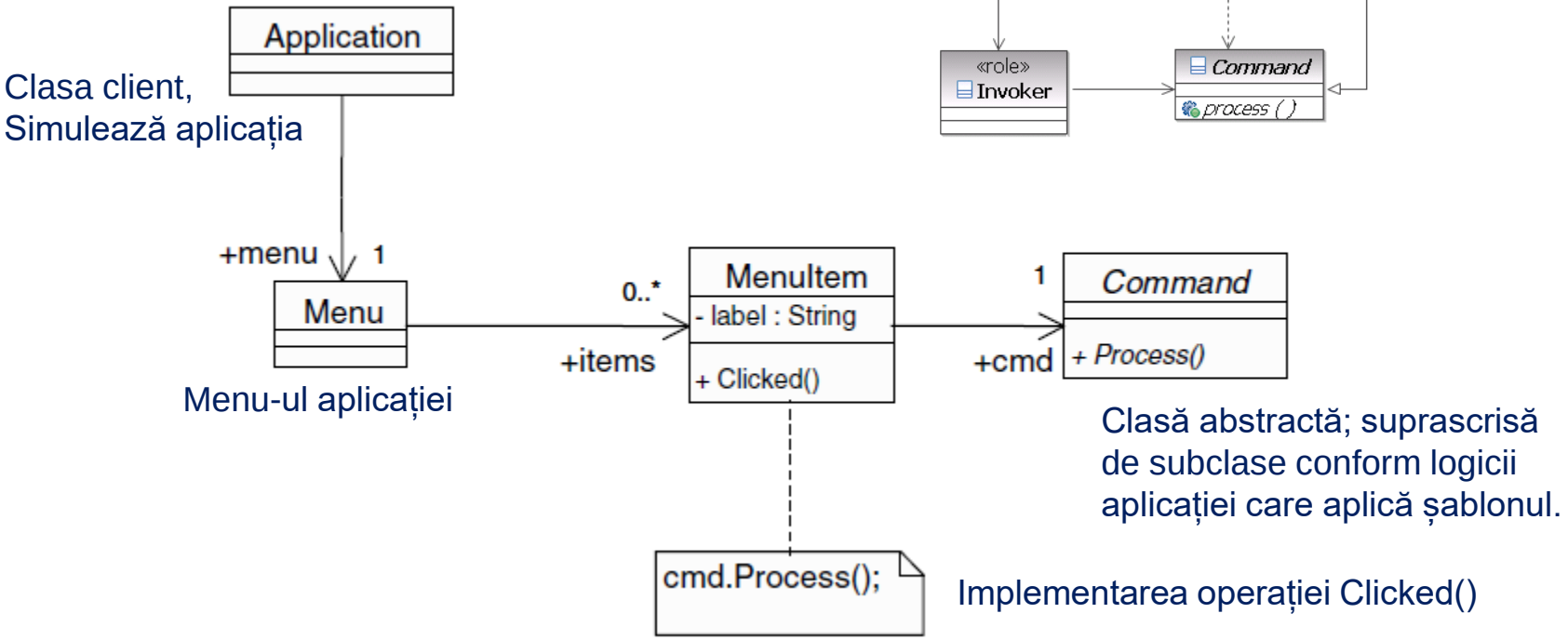
- Presupunem că vrem să construim o componentă GUI reutilizabilă

Pentru simplitate ne vom limita la implementarea unui menu generic într-un sistem bazat pe ferestre, astfel încât să fie posibilă adăugarea de noi opțiuni fără a modifica componenta GUI.

Soluţia

- Bazată pe şablonul Command

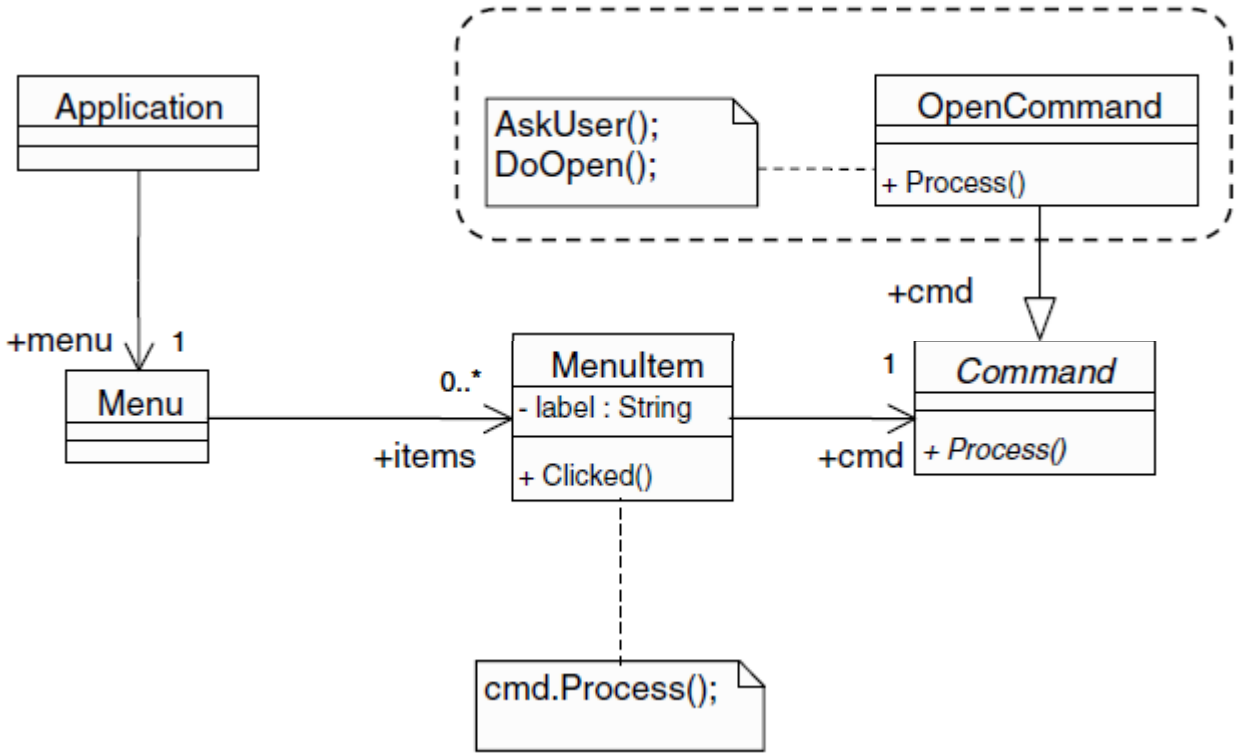




ŞABLOANE DE PROIECTARE

Exemplu: Construirea unei componente GUI generice

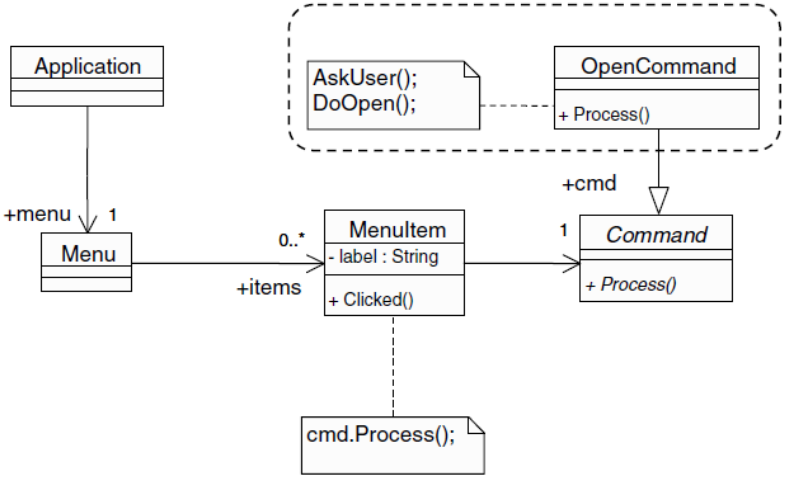
Adăugarea unei opțiuni de deschidere fișier.



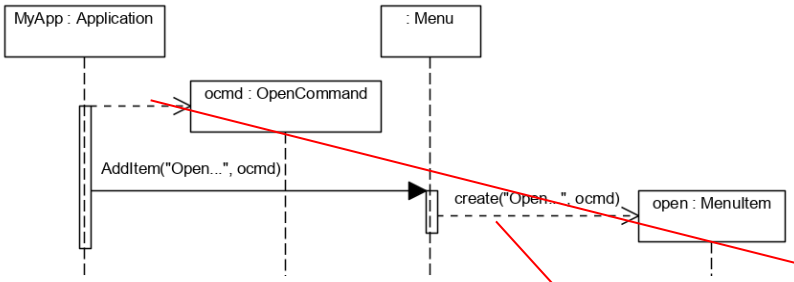
ŞABLOANE DE PROIECTARE

Exemplu:

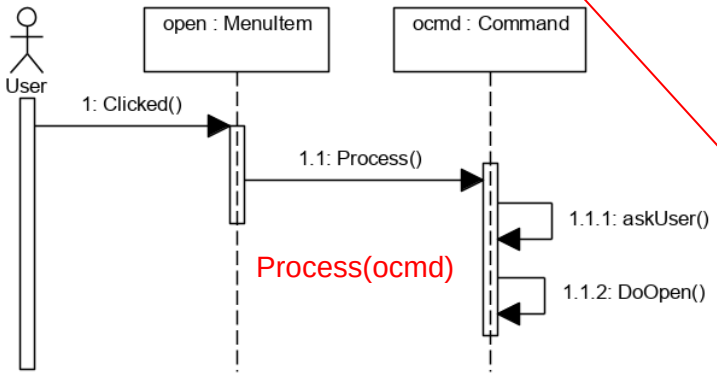
Construirea unei componente GUI generice



Inițializare prealabilă a opțiunii de menu Open



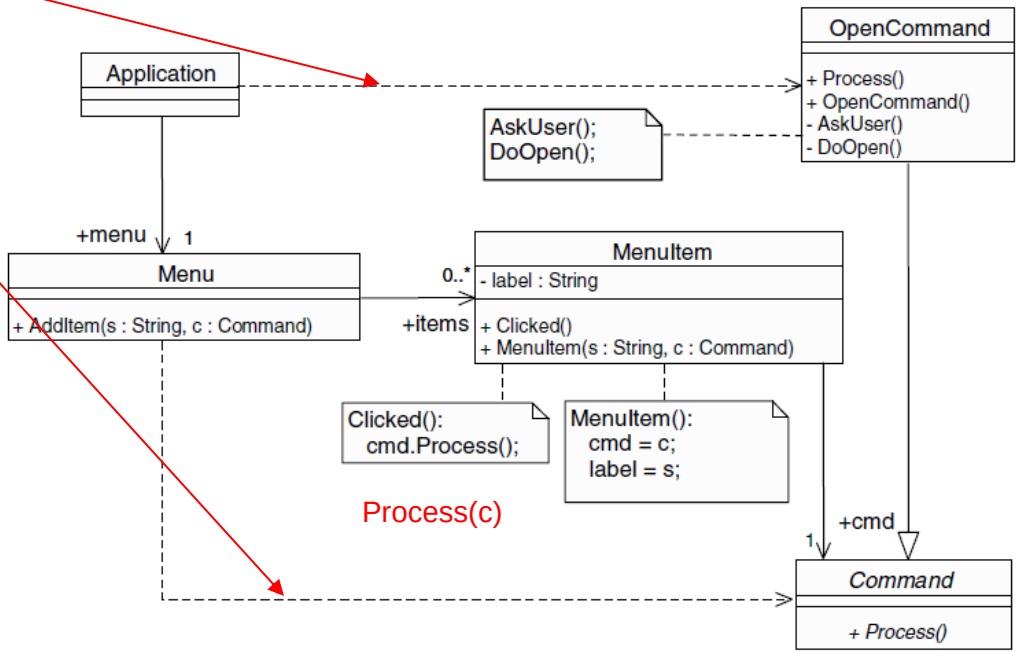
Utilizarea opțiunii de menu Open



Opțiunea de menu `open` este inițializată cu `ocmd` de tip **`Command`**, pe care știe să apeleze `Process()`.

Apelul operației `Process()` se face pe obiectul transmis ca parametru, instanțiat dintr-o subclasă a clasei `Command`.

Diagrama de clase actualizată



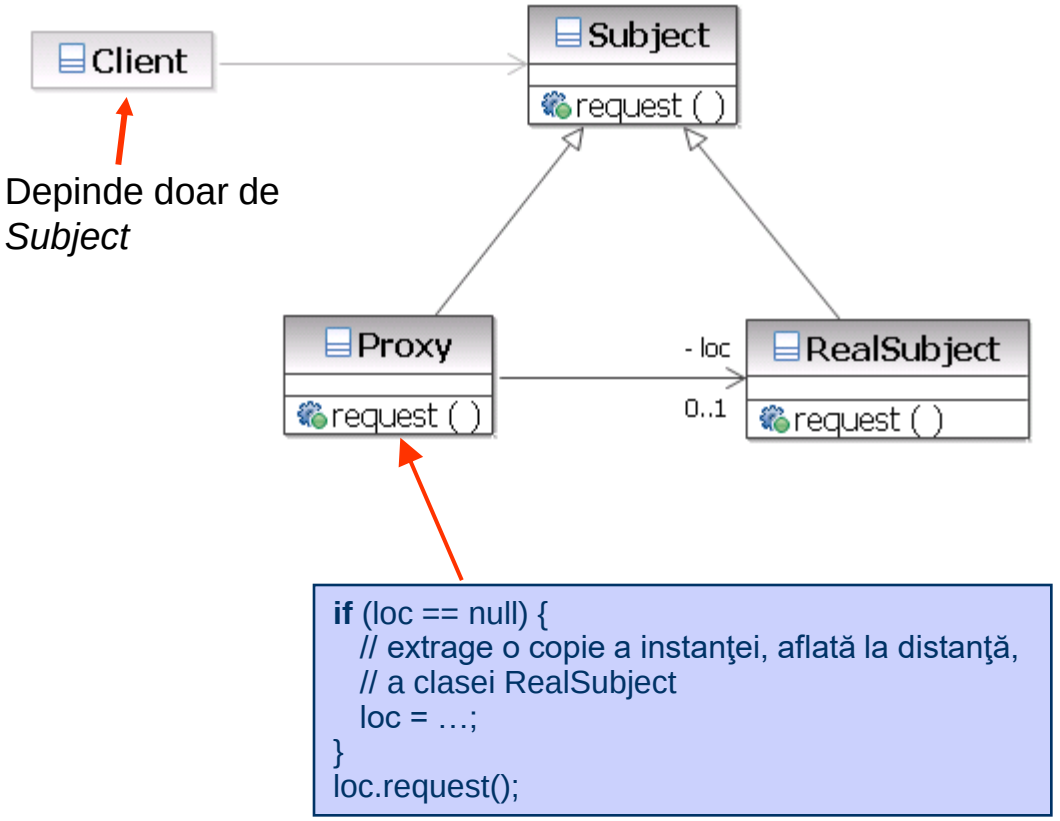
Cum se adaugă noi opțiuni de menu ?

ŞABLOANE DE PROIECTARE
Exemplu

- Un “proxy” este un înlocuitor pentru un alt obiect, cu rolul de a controla accesul la obiectul respectiv.

- Applicabilitate
  - Remote proxy
  - Virtual proxy (crează obiecte “costisitoare” doar la cerere)
  - etc.

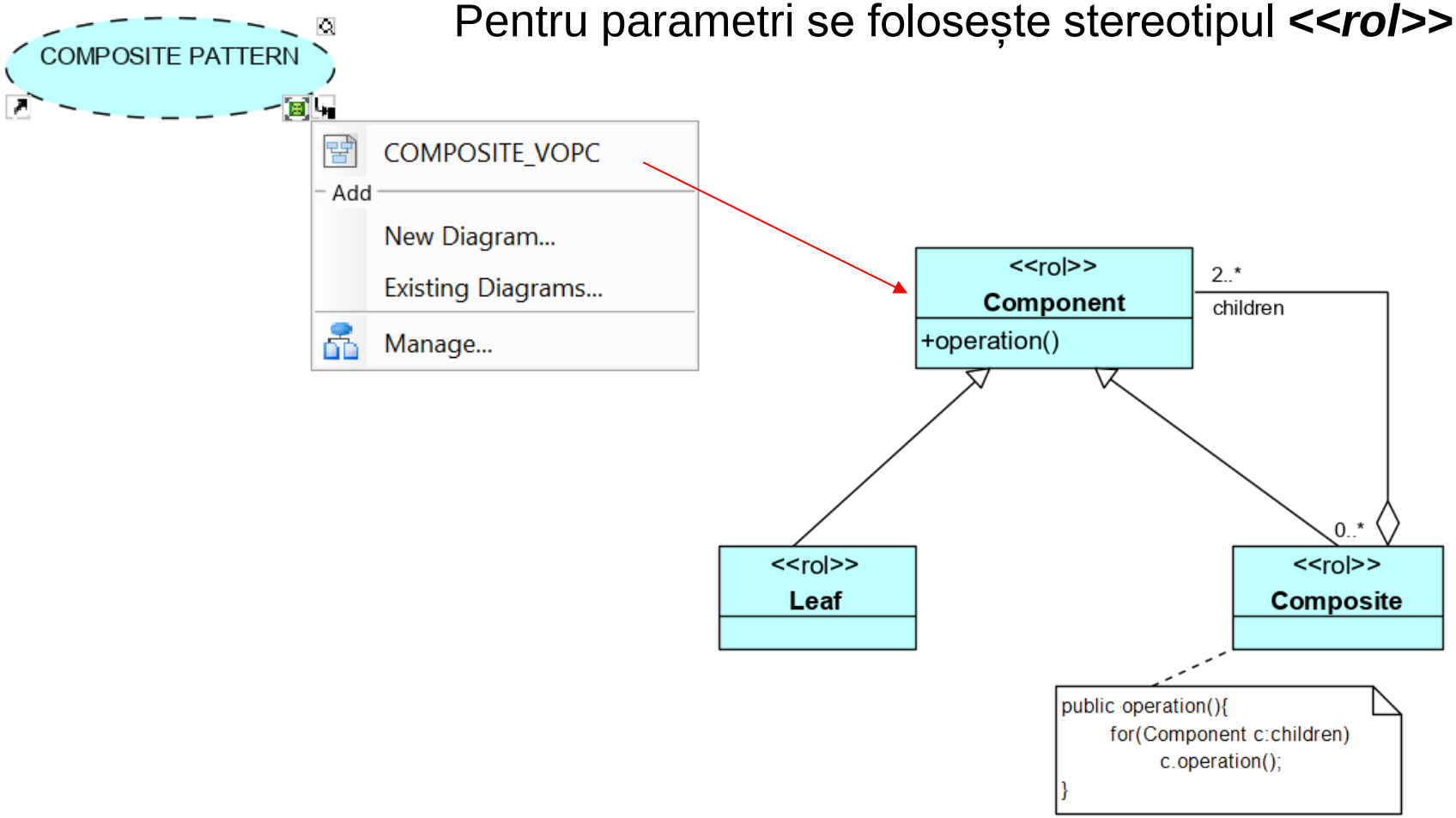
Exemplu : Șablonul Proxy



REPREZENTARE ȘABLON
Exemplu: COMPOSITE

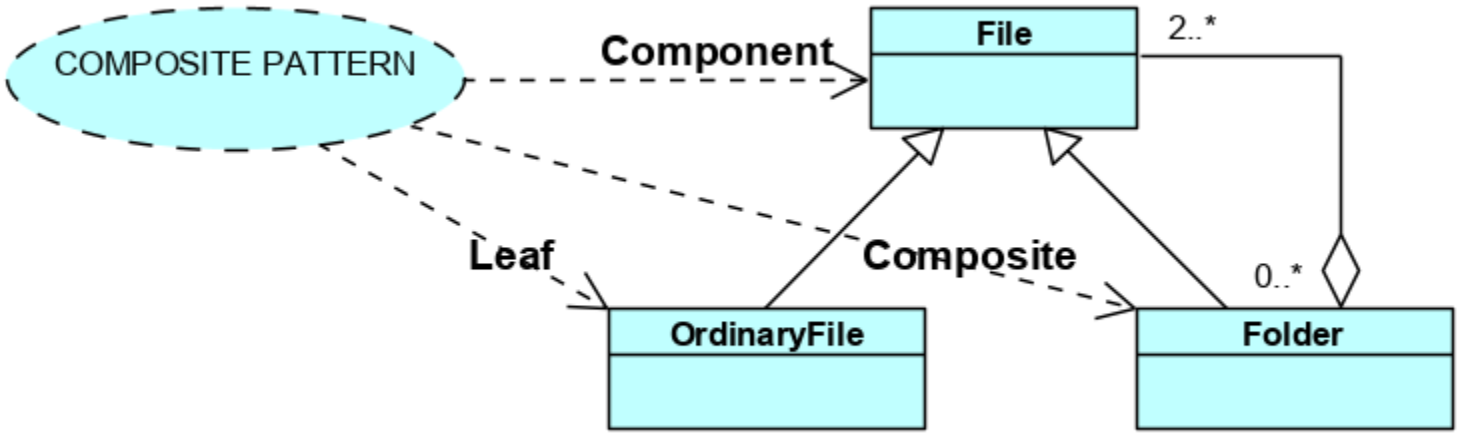
Şablonul de proiectare se reprezintă sub formă de ***colaborare parametrizată***.

Pentru parametri se folosește stereotipul ***<<rol>>*** .



REPREZENTARE INSTANȚIERE ȘABLON
Exemplu: COMPOSITE

Instanțele unui șablonul de proiectare se reprezintă definind ***corespondența*** dintre ***roluri*** și ***clasele concrete*** ale instanței.



Exemple de aplicare a șablonului Composite:

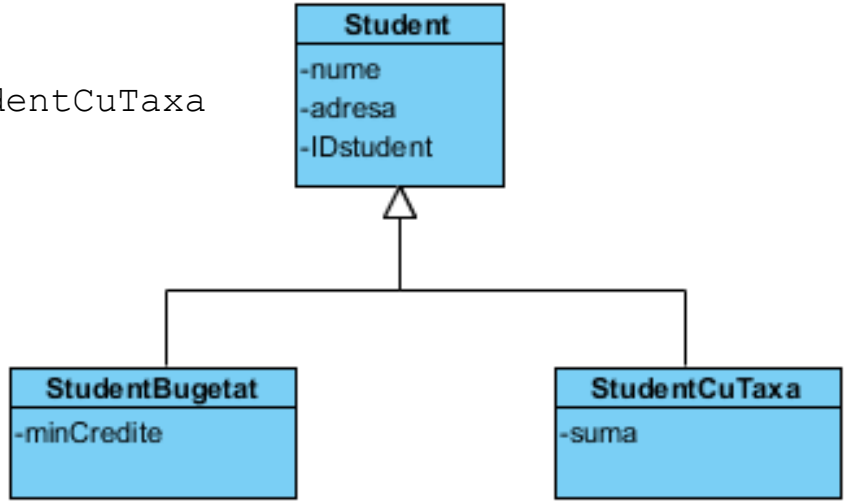
- Sistem de fişiere compus din fişiere şi folder-e
- Grafic compus din forme elementare şi forme asamblate

ŞABLOANE DE PROIECTARE
Exemplu

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`

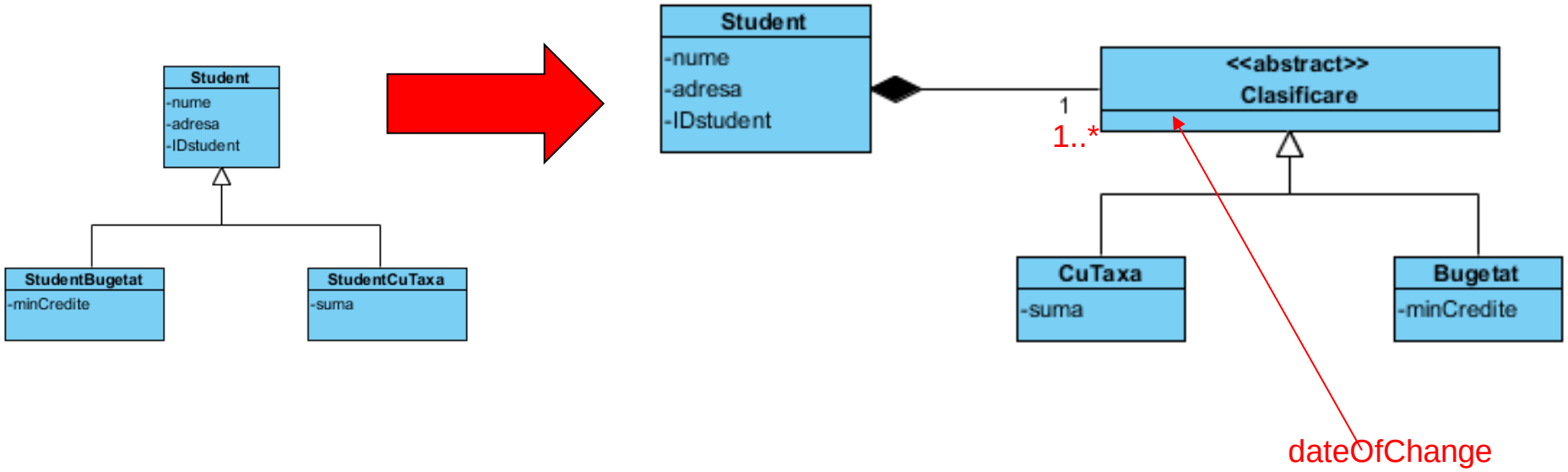


Problema se complică dacă se doreşte păstrarea unui istoric pentru fiecare student.

ŞABLOANE DE PROIECTARE
Exemplu

Soluţia reprezentată în diagrama de mai jos simplifică trecerea de la `StudentCuTaxa` la `StudentBugetat` şi o face mai eficientă (nu este necesară copiere date sau notificare).

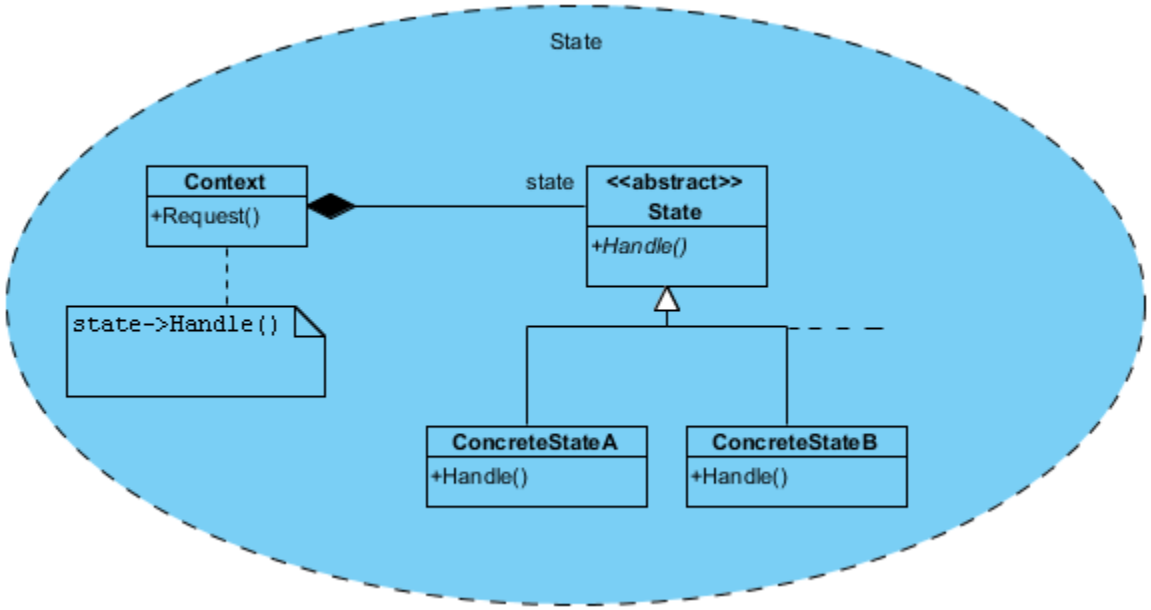
Este posibilă menţinerea unui istoric prin simpla modificare a multiplicităţii compoziţiei în 1..*. Dacă se adaugă un atribut `dateOfChange` la `Clasificare` atunci istoricul poate fi ordonat după dată.



ŞABLOANE DE PROIECTARE
Exemplu

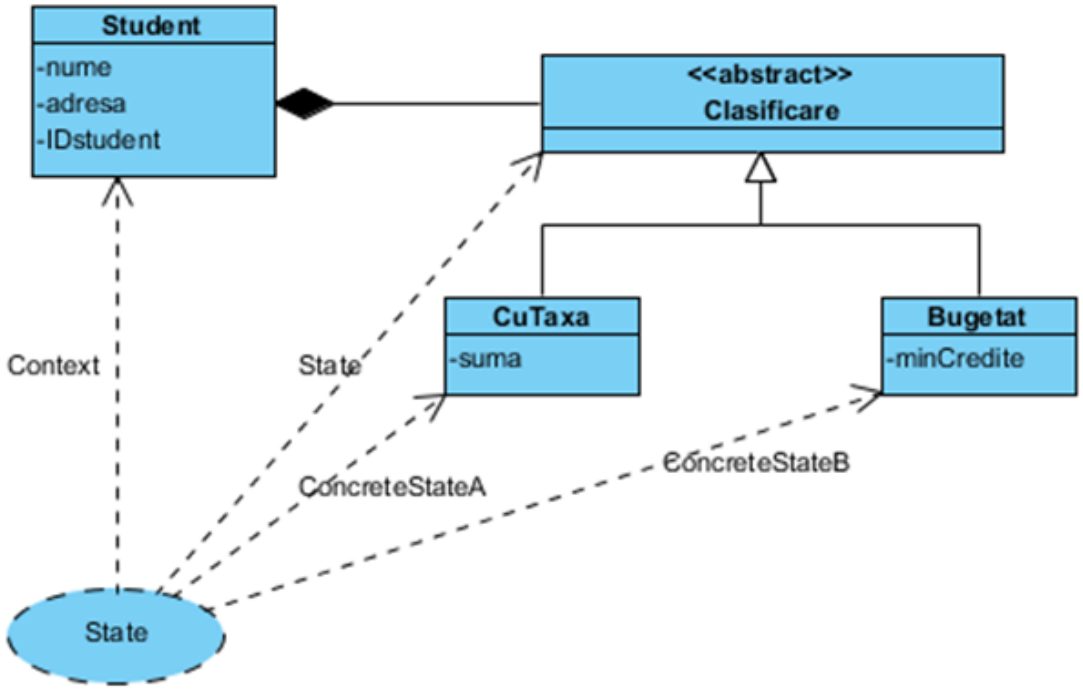
Pentru a obţine soluţia am utilizat de fapt şablonul **State**.

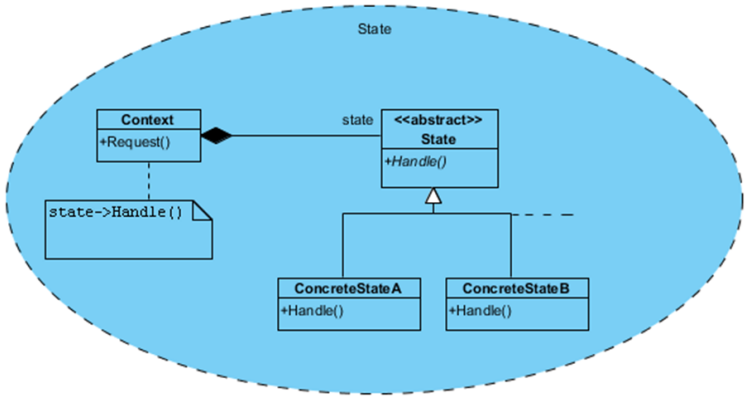
Agregatul (clasa `Context`) poate invoca operaţii fără a cunoaşte starea curentă. La schimbarea stării, agregatul primeşte un nou obiect de tip `State`, instanţă, corespunzătoare noii stări, a unei subclase a clasei `State`. La recepţionarea unei cereri, agregatul invocă pur şi simplu operaţia corectă a obiectului de tip `State`, aşa cum este implementată în subclasa corespunzătoare stării curente.



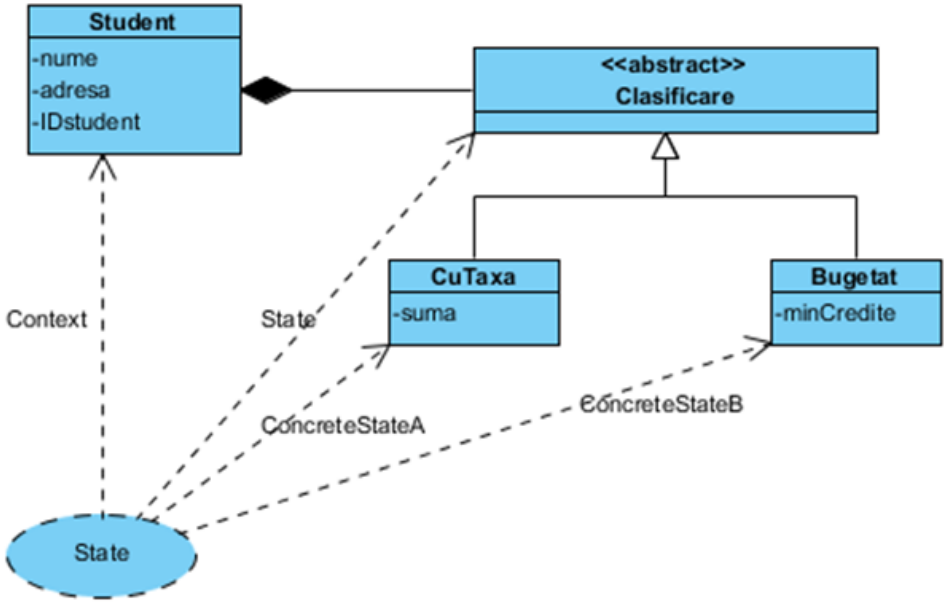
ŞABLOANE DE PROIECTARE
Exemplu: STATE

Instanţierea şablonului **State** ce defineşte soluţia prezentată.





Clasa `Student` poate invoca operaţii fără a cunoaşte starea curentă, adică daca studentul este bugetat sau cu taxă.



Presupunem că studentul a fost iniţial cu taxă şi a devenit bugetat.

La schimbarea stării studentului în bugetat, clasa `Student` primeşte un nou obiect de tip `Clasificare`, instanţă a subclasei `Bugetat`.

La recepţionarea unei cereri pentru o operaţie (de exemplu `calculXXX`), clasa `Student` invocă pur şi simplu operaţia corectă a obiectului de tip `State`, aşa cum este implementată în subclasa `Bugetat`.

State vs Strategy ?

https://stackoverflow.com/questions/1658192/what-is-the-difference-between-strategy-design-pattern-and-state-design-pattern

https://refactoring.guru/design-patterns/state

ŞABLOANE DE PROIECTARE
Instrumente

Exemple:

IBM Rational Software Modeler are incorporate șabloane de proiectare ce pot fi utilizate; accesibile în Pattern Explorer view.
https://www.ibm.com/support/knowledgecenter/SSCLKU_7.5.5/com.ibm.xtools.patterns.apply.tutorial.doc/topics/abstract_apply.html

De asemenea, se pot crea noi șabloane folosind instrumentul “pattern authoring tool” oferit de facilitatea Rational's RAS (Reusable Asset Specifications).

https://www.ibm.com/support/knowledgecenter/SSCLKU_7.5.5/com.ibm.xtools.patterns.create.tutorial.doc/topics/abstract_create.html

Visual Paradigm permite creare de șabloane de proiectare pentru a fi utilizate ulterior.

https://www.visual-paradigm.com/tutorials/mementodesignpattern.jsp

Evaluare formativă

1. Câte opțiuni are menu-ul din diagrama de clase actualizată reprezentată în slide-ul 12 ? Explicați cum se pot adăuga noi opțiuni la acesta.

2. Cum se reprezintă un șablon de proiectare și cum se reprezintă o instanță a sa în UML ?

https://forms.gle/8GoocWhoVmyfu6LK9

PLAN CURS

- Utilizare șabloane în proiectarea software-lui
- **Identificarea mecanismelor de proiectare**
- Proiectare subsisteme

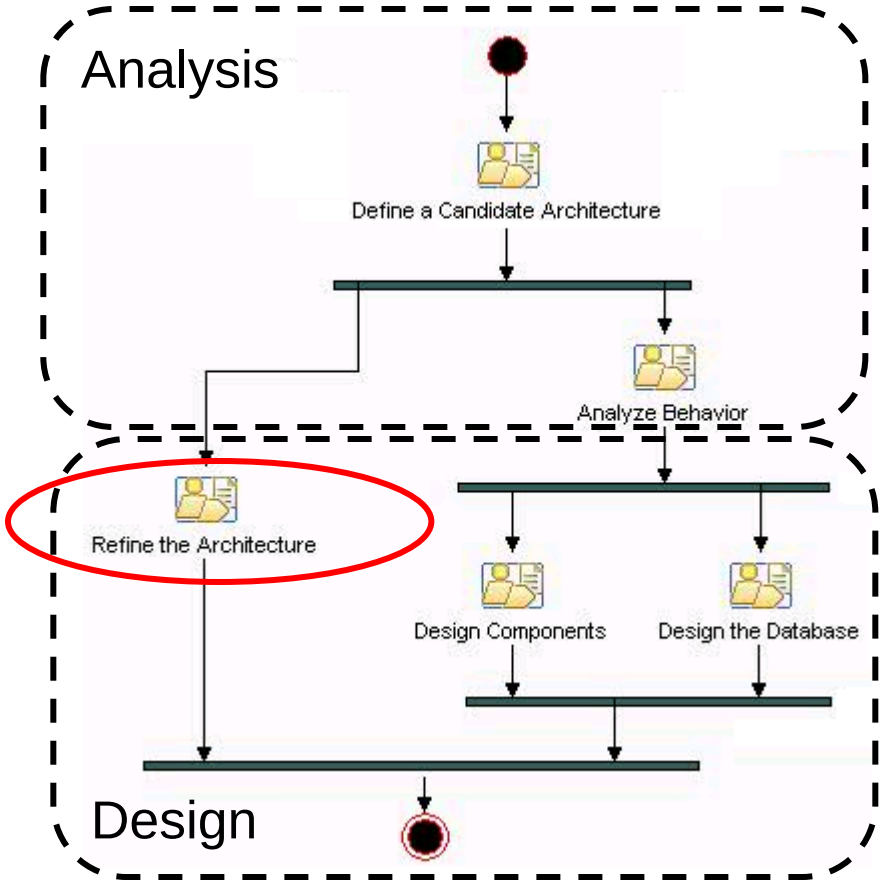
Etape

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 MECANISMELOR DE PROIECTARE

Scop

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

Rol responsabil

Arhitectul software

Etape majore

Identificarea mecanismelor de proiectare şi de implementare

Documentarea mecanismelor

MECANISME DE PROIECTARE

Def. *Mecanism de proiectare* = o rafinarea a unui mecanism de analiză corespondent

- Adaugă detalii concrete la mecanismul de analiza conceptual, până la limita particularităţilor tehnologice
- Poate instanţia unul sau mai multe şabloane (arhitecturale sau de proiectare)

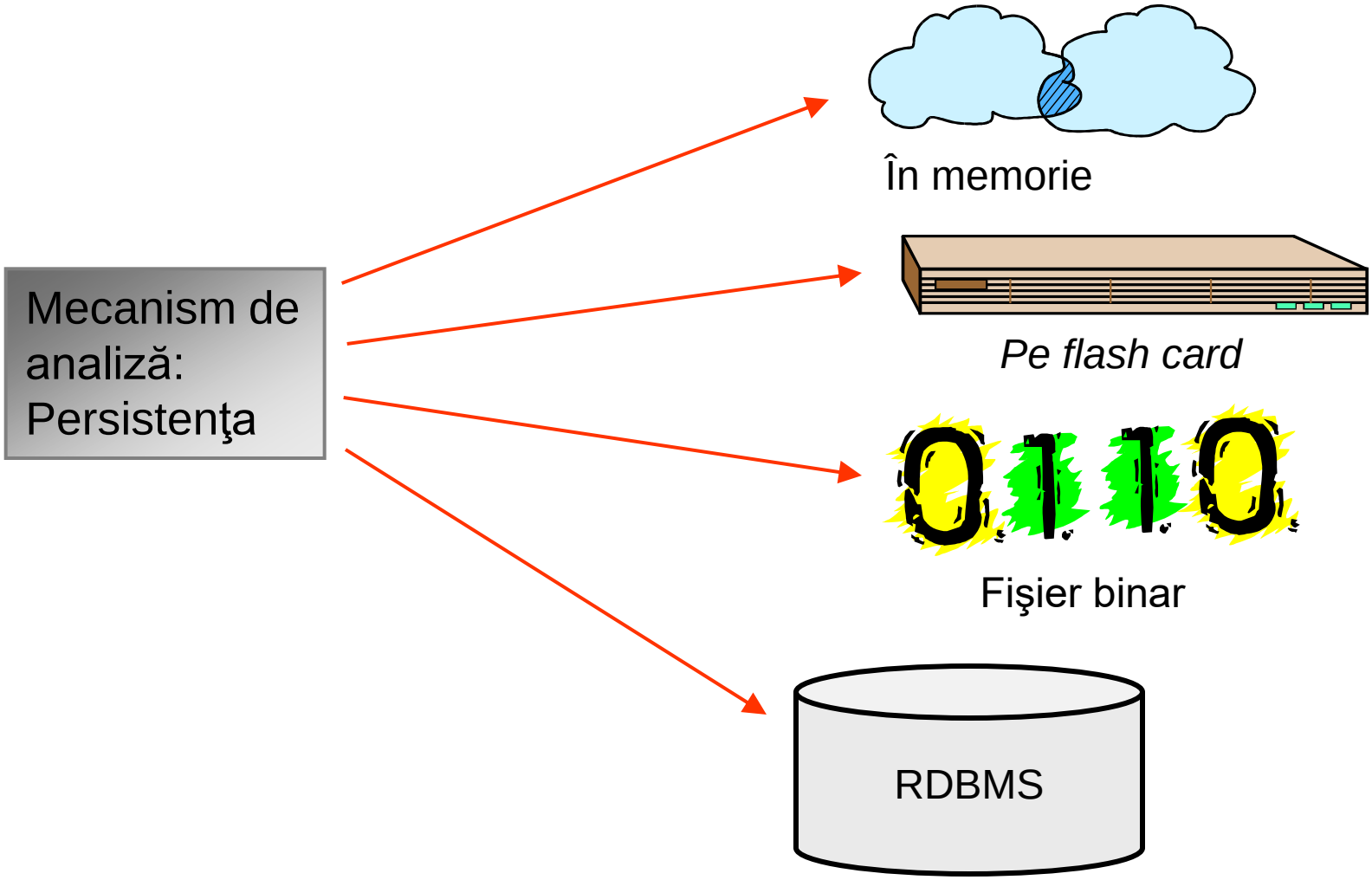
- Identificarea mecanismelor de proiectare din mecanismele de analiză:
  - Identificarea clienţilor fiecărui mecanism de analiză
  - Identificarea de profiluri caracteristice pentru fiecare mecanism de analiză
  - Gruparea clienţilor conform utilizării profilurilor caracteristice
  - Abordare bottom-up şi realizarea unui inventar al mecanismelor de proiectare aflate la dispoziţie.

MECANISME DE ANALIZĂ
Exemple

- **Persistenţă**: metodă de a face ca un obiect să existe şi după terminarea aplicaţiei care l-a generat.
- **Distribuire**: metodă de a distribui un element pe nodurile existente într-un sistem.
- **Securitate**: mijloc de control al accesului la un element.
- **Interfaţă legacy**: mijloc de a accesa un sistem legacy prin intermediul unei interfeţe existentă.

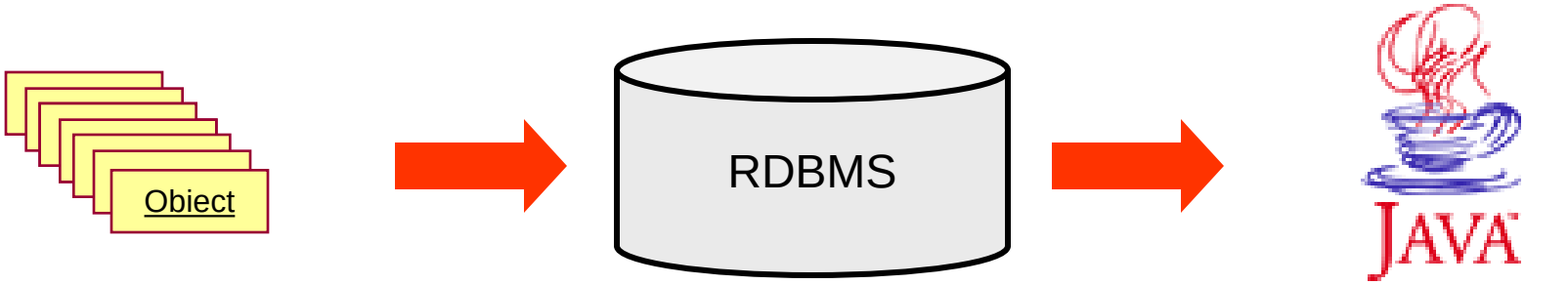
MECANISME DE PROIECTARE

Exemplu



MECANISME DE IMPLEMENTARE
Exemplu

Def. *Mecanism de implementare* = rafinare a unui mecanism de proiectare corespondent.



Mecanism de analiză:
Persistenţa

Mecanism de proiectare:
RDBMS

Mecanism de implementare:
JDBC

DOCUMENTAREA MECANISMELOR

Un mecanism de proiectare/implementare reprezintă un şablon care constituie o *soluţie comună* la o *problemă comună.*

- Scopul final este de a asigura consistenţă în implementarea sistemului, simultan cu îmbunătăţirea productivităţii.

Arhitectul software defineşte *CE* mecanism de implementare trebuie utilizat de toate clasele client cu aceleaşi caracteristici de profil şi *CUM* va fi folosit acesta.

- Rezultatul final este o ***colaborare*** ce va fi documentată ca orice altă colaborare: utilizând diagrame de ***secvenţe*** şi diagrama cu *clasele participante*.

DOCUMENTAREA MECANISMELOR
Exemplu : Mecanismul JDBC pentru persistență

Fiecare clasă persistentă are o clasă *BDClasaPersistentă* corespondentă.

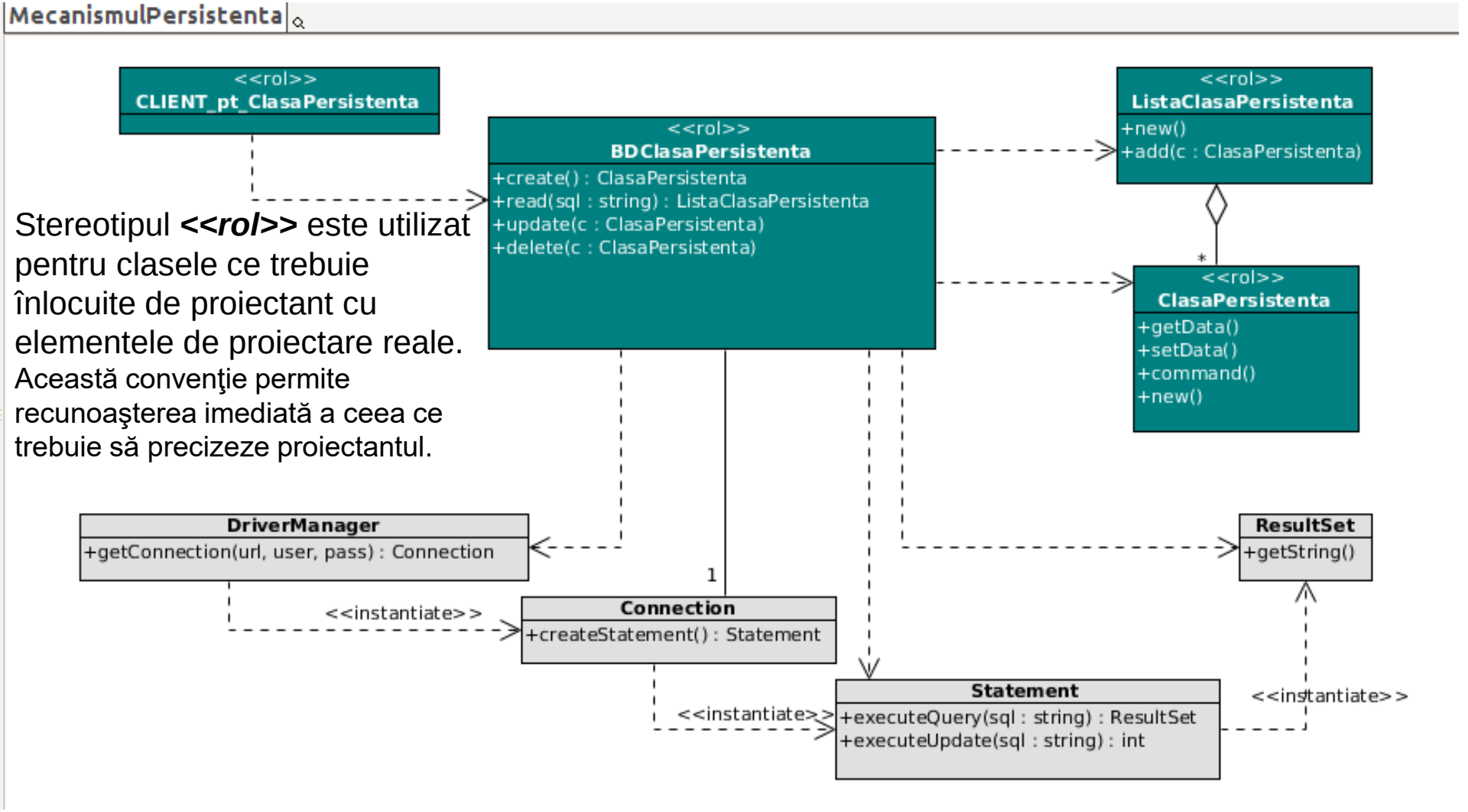
- Un client interacţionează cu o clasă *BDClasaPersistenta* pentru a a accesa date persistente, de tipul clasei persistente, din baza de date relaţională.
- Clasa *BDClasaPersistentă* asigură pentru client interfaţa cu baza de date. Aceasta citeşte înregistrări din baza de date şi construieşte obiecte, sau “aplatizează” obiecte persistente şi le scrie în baza de date.
- Clasa *BDClasaPersistenta* este responsabilă cu accesarea bazei de date utilizând clasa *DriverManager* ce deschide o conexiune la baza de date printr-un driver corespunzător.
- După deschiderea conexiunii, clasa *BDClasaPersistentă* poate crea instrucţiuni SQL ce vor fi trimise bazei de date din RDBMS pentru a fi executate. Obiectul *Statement* este cel care dialoghează cu baza de date.
- Rezultatul unei interogări SQL este returnat într-un obiect *ResultSet*.

Clasa *ListaClasaPersistenta* este utilizată pentru returnarea unui set de obiecte persistente ca rezultat al unei interogări pe baza de date.

DOCUMENTAREA MECANISMELOR

Exemplu : Mecanismul JDBC pentru persistență

ClasaPersistenta este un ***nume generic*** şi va fi ***înlocuit*** cu numele clasei particulare în fiecare caz.



Stereotipul ***<<rol>>*** este utilizat pentru clasele ce trebuie înlocuite de proiectant cu elementele de proiectare reale.

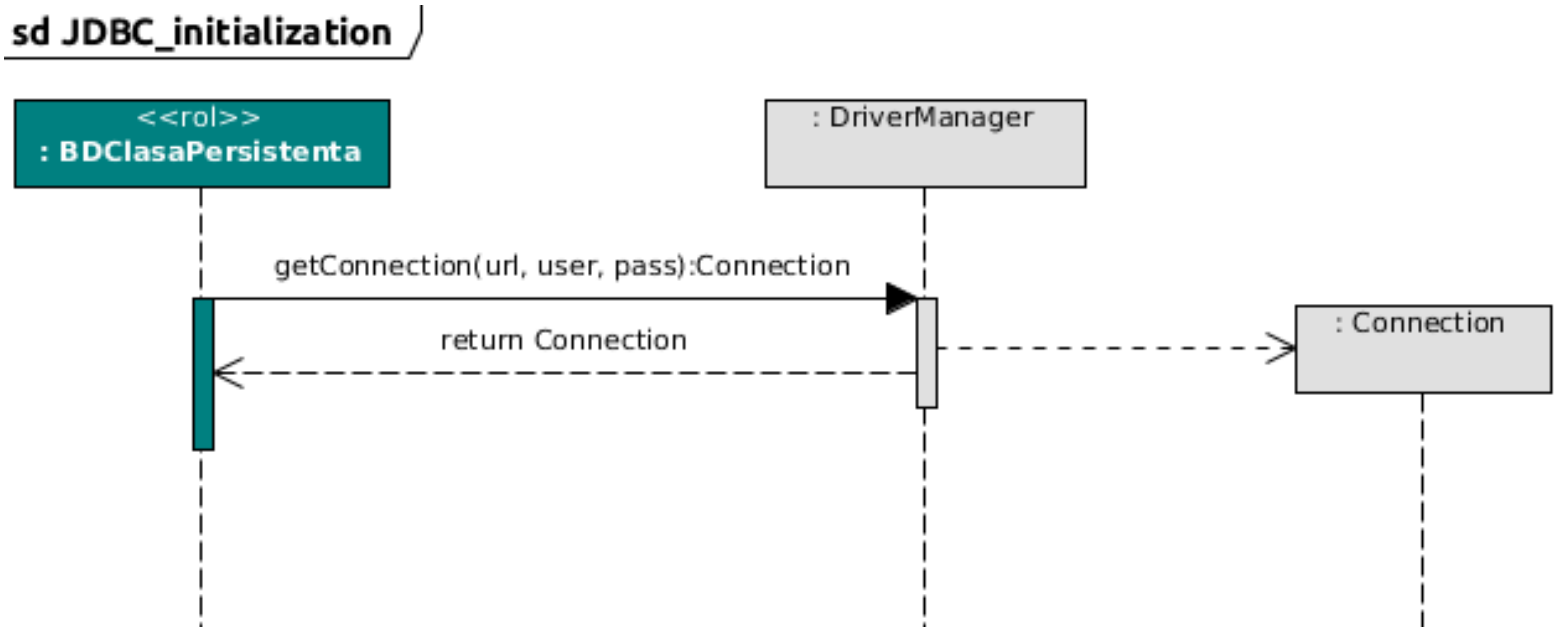
Această convenţie permite recunoaşterea imediată a ceea ce trebuie să precizeze proiectantul.

DOCUMENTAREA MECANISMELOR
Mecanismul JDBC : inițializarea conexiunii

Accesul la baza de date se face prin intermediul unei conexiuni creată de **DriverManager** la solicitarea lansată de **BDClasaPersistenta**.

După crearea conexiunii, **DriverManager** returnează un obiect **Connection** care permite utilizarea acesteia pentru realizarea de operaţii cu baza de date.

ClasaPersistenta este un nume generic şi va fi înlocuit cu numele clasei particulare în fiecare caz.



DOCUMENTAREA MECANISMELOR
Mecanismul JDBC : interogare

ClasaPersistenta este un nume generic şi va fi înlocuit cu numele clasei particulare în fiecare caz.

BDClasaPersistenta realizează următoarele:

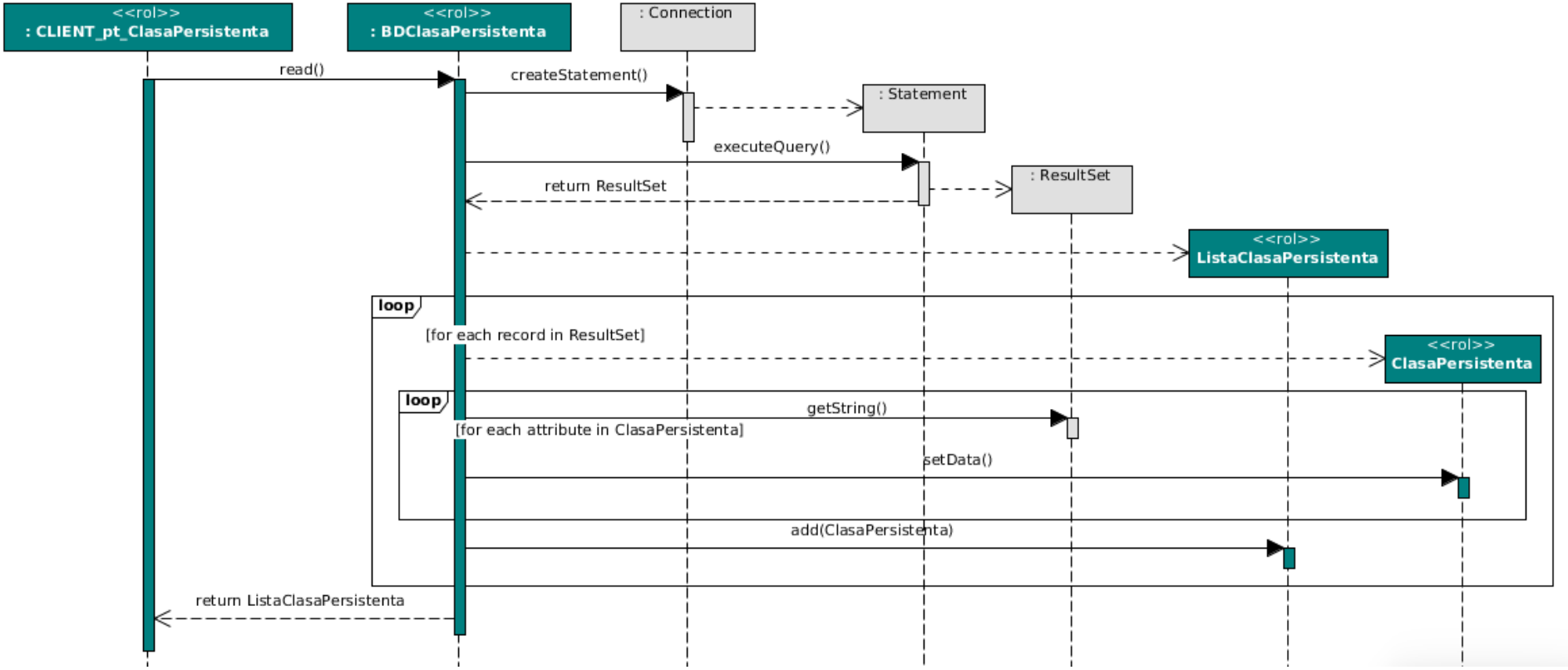
1. Pe obiectul **Connection** solicită crearea unui obiect **Statement**.
2. Pe obiectul **Statement** solicită executarea unei interogări (query).

Ca urmare a executării interogării se crează (automat) un obiect **ResultSet** a cărui referinţă este primită de **BDClasaPersistenta** care continuă astfel:

3. Crează obiectul **ListaClasaPersistenta**.
4. Preia fiecare înregistrare din obiectul **ResultSet** şi construieşte câte un obiect **ClasaPersistenta**.
5. După construirea fiecărui astfel de obiect este adăugată o referinţă către acesta în clasa **ListaClasaPersistenta**.

DOCUMENTAREA MECANISMELOR

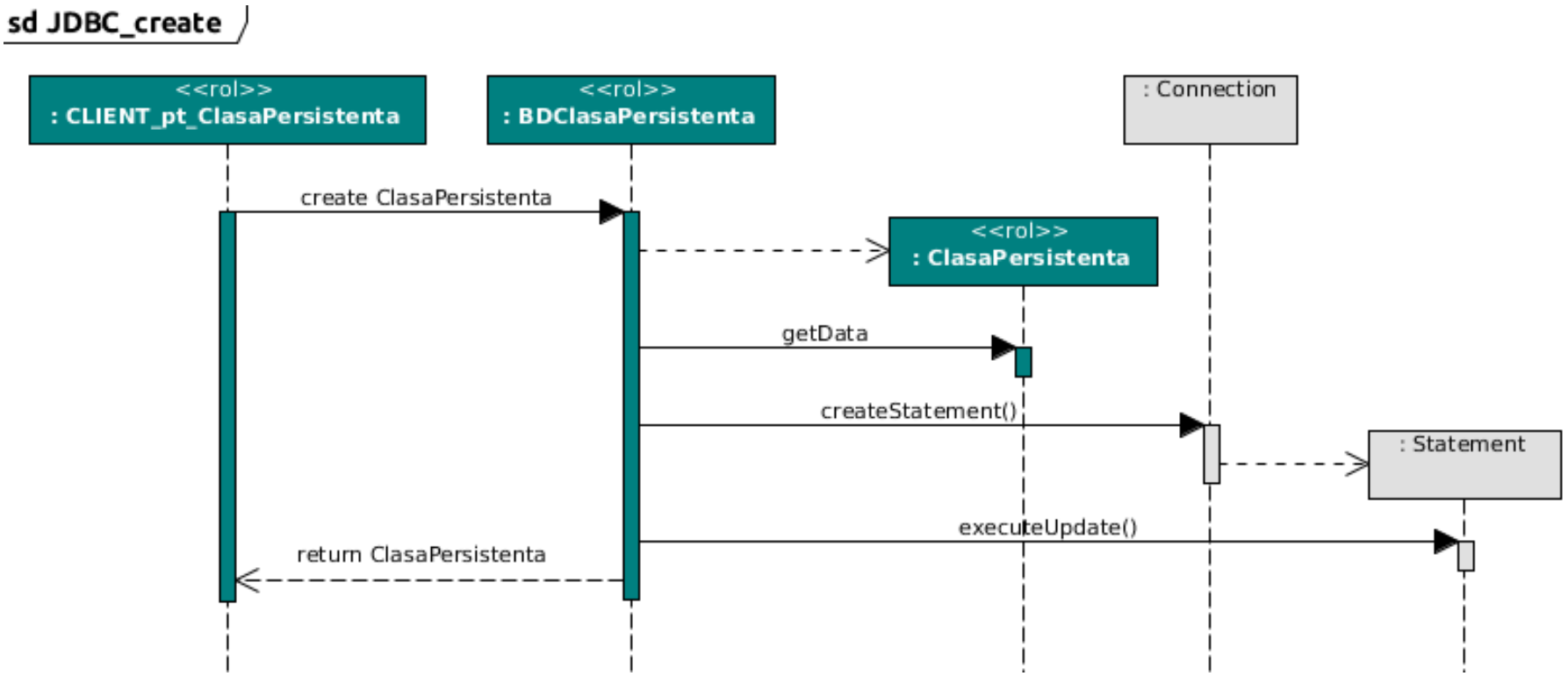
Mecanismul JDBC : interogare



ClasaPersistenta este un nume generic şi va fi înlocuit cu numele clasei particulare în fiecare caz.

DOCUMENTAREA MECANISMELOR

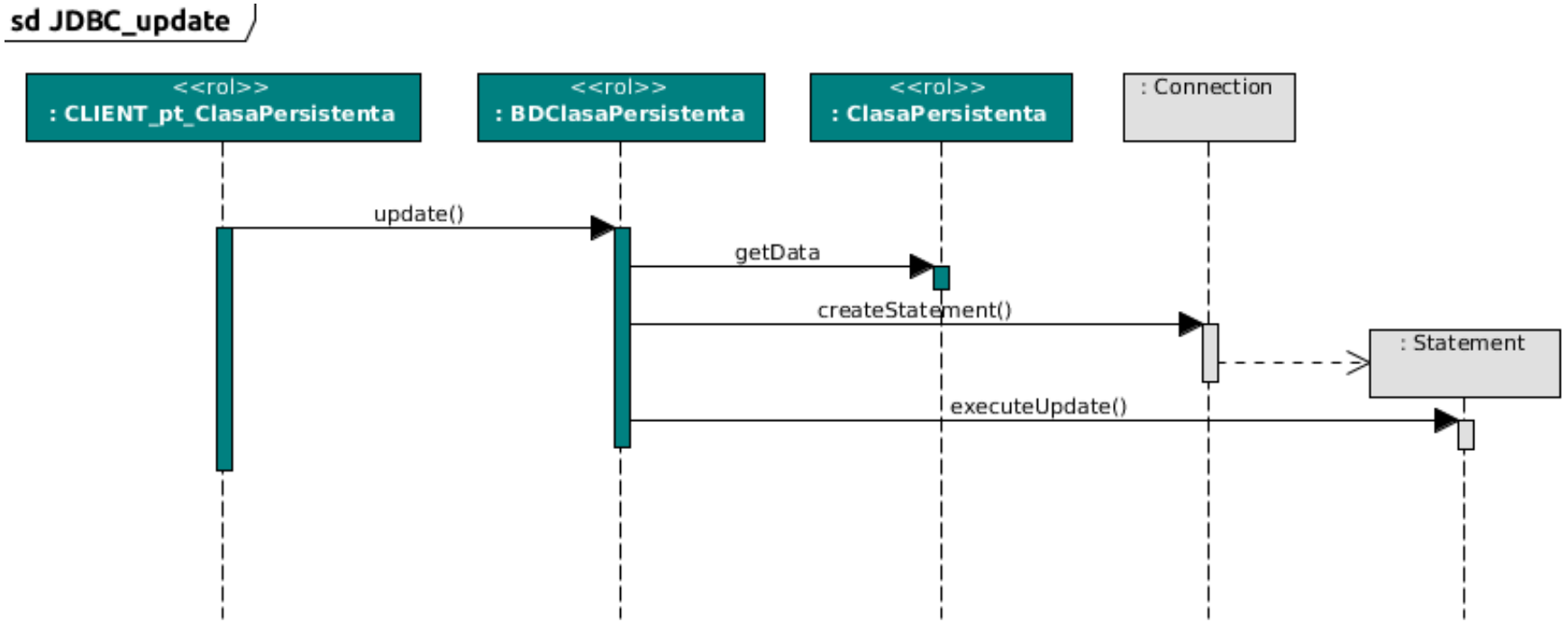
Mecanismul JDBC : creare obiect persistent



ClasaPersistenta este un nume generic şi va fi înlocuit cu numele clasei particulare în fiecare caz.

DOCUMENTAREA MECANISMELOR

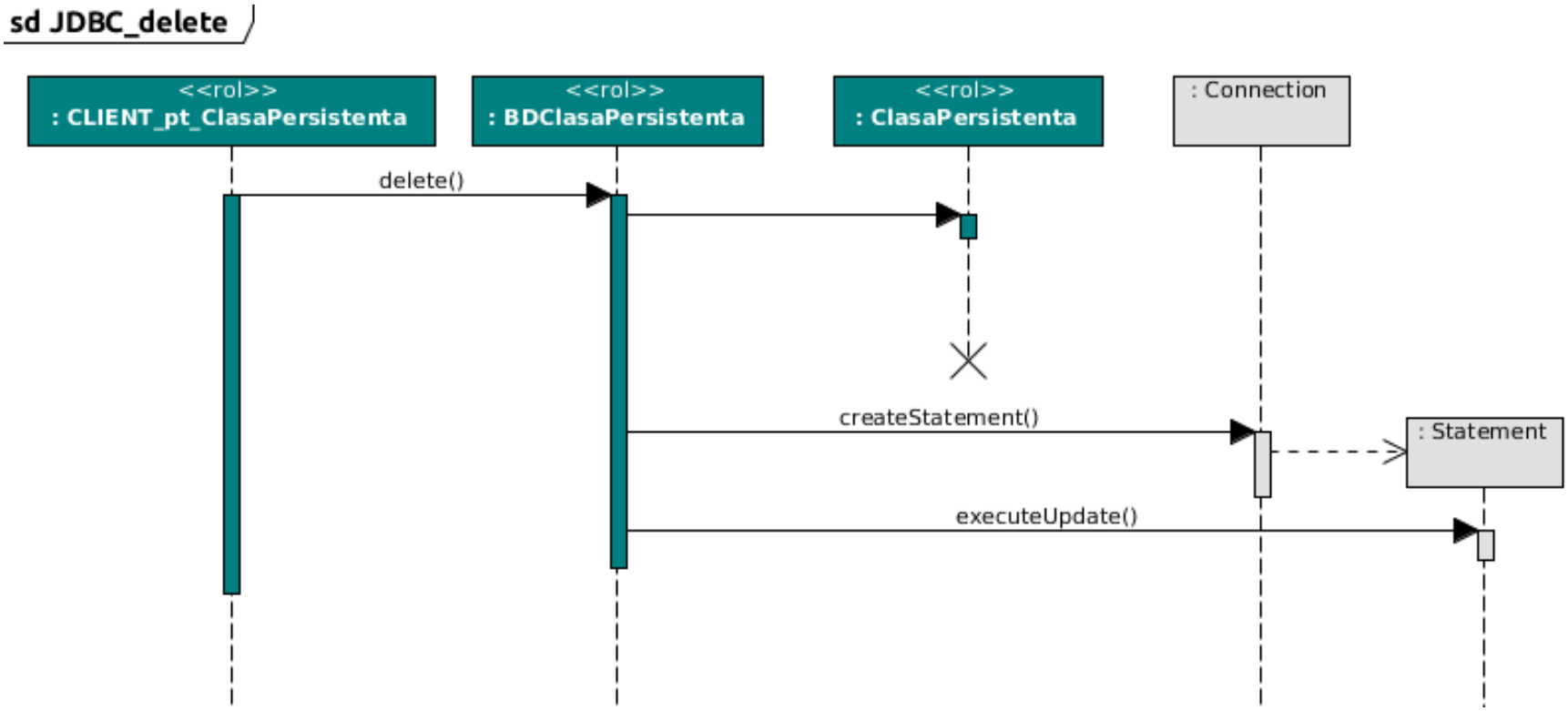
Mecanismul JDBC : actualizare în baza de date



ClasaPersistenta este un nume generic şi va fi înlocuit cu numele clasei particulare în fiecare caz.

DOCUMENTAREA MECANISMELOR

Mecanismul JDBC : ștergere obiect persistent



ClasaPersistenta este un nume generic şi va fi înlocuit cu numele clasei particulare în fiecare caz.

Evaluare formativă

1. Ce conține documentația realizată de arhitectul software pentru un mecanism de implementare?

2. Pentru a aplica mecanismul JDBC, ce clase trebuie adăugate, în proiectul vostru din stadiul curent, pentru clasa persistentă `Student/Employee`.

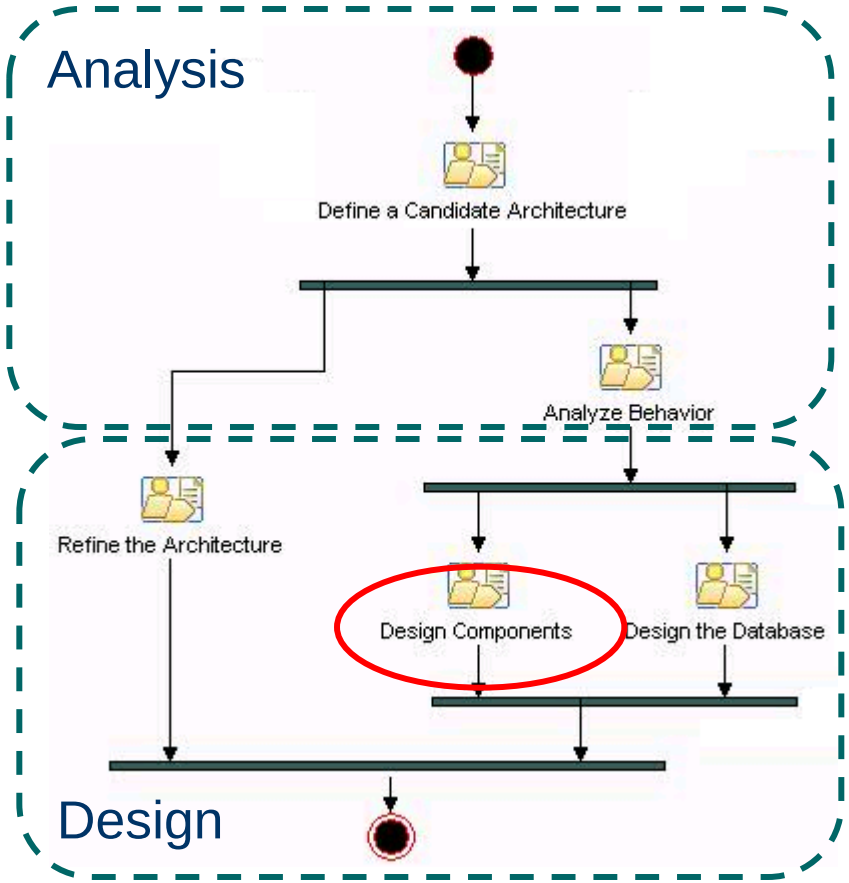
https://forms.gle/inzxbYf29gDfMuvS9

PLAN CURS

- Utilizare șabloane în proiectarea software-lui
- Identificarea mecanismelor de proiectare
- **Proiectare subsisteme**

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 SUBSISTEME

- Scop
  - Încorporarea subsistemelor în modelul proiect şi documentarea comportamentului acestora.
- Rol responsabil
  - Proiectantul
- Etape majore
  - Încorporarea subsistemelor în modelul proiect.
  - Specificarea comportamentului intern al subsistemelor.

INCORPORARE INTERFEȚE ÎN MODELUL PROIECT

Subsistemele sunt componente ce oferă servicii clienţilor lor *doar prin interfeţe publice.*

- Orice două subsisteme care realizează *aceeaşi **interfaţă** publică* sunt interschimbabile.

Subsistemele şi interfeţele acestora au fost identificate în lab. 3. (v. `InstructiuniGenerale_lab3.ppt` şi `Lab3_APSSw.pdf`)

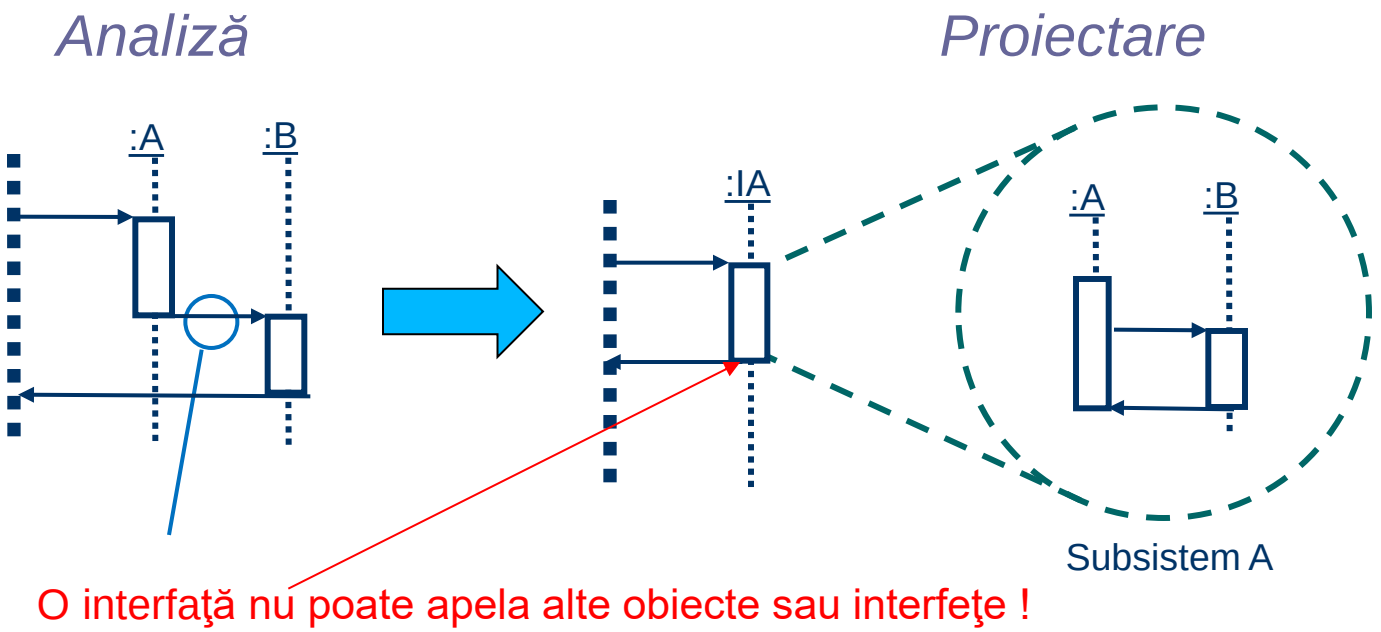
PROCEDURA:

- Transformare responsabilităţilor specificate la analiză în operaţii.
- Separarea interfeţei de implementare.
- Incorporarea interfeţei în diagrama de clase
  - Înlocuirea clasei cu interfaţa.
  - Transferarea tuturor relaţiilor de la clasă la interfaţă.
- Incorporarea intefeţei în diagramele de secvenţe
  - Înlocuirea clasei cu interfaţa.

ÎNCAPSULAREA INTERACȚIUNILOR din SUBSISTEME

- Interacţiunile din interiorul subsistemelor trebuie descrise în diagrame de secvenţe proprii fiecărui subsistem.

Exemplu: Fie o interacţiune definită la analiză care implică clasa de analiză A ce a fost convertită în interfaţa IA



Dacă A este înlocuită la proiectare cu o interfaţă, atunci apelul la :B trebuie descris doar într-o interacţiune internă subsistemului format din A și B.

COMPORTAMENTUL INTERN AL SUBSISTEMELOR

- *Interfeţele* modelează *vederea din exterior* asupra subsistemelor.
- *Comportamentul intern* al subsistemelor este similar cu orice *colaborare*.

Comportamentul intern va fi reprezentat prin:

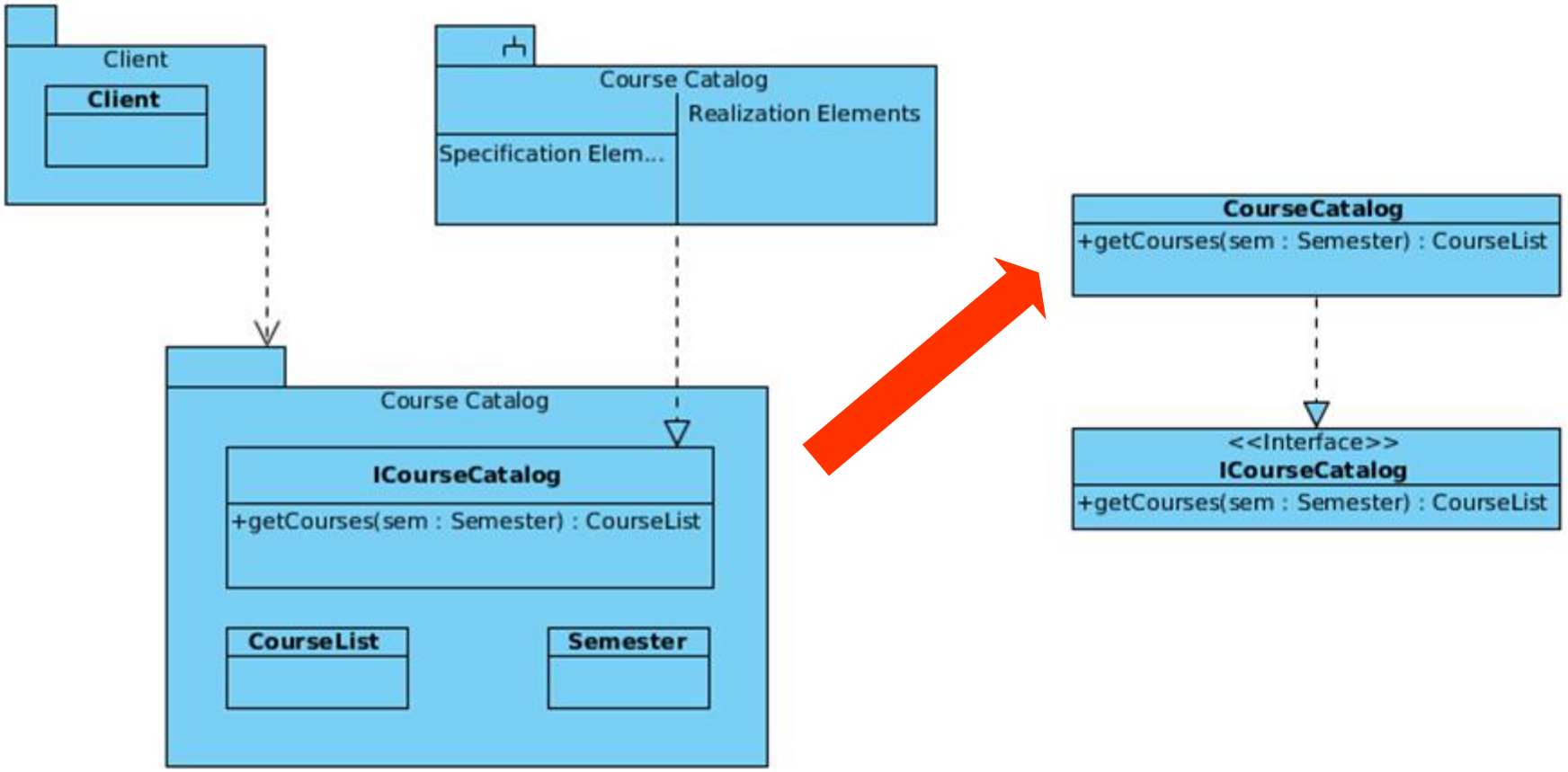
- Una (sau mai multe) diagrame de interacţiune pentru *fiecare* serviciu oferit de subsistem.
- Una (sau mai multe) diagrame de clase care conţin clasele implicate în implementarea serviciilor.

Exemplu: Subsistemul *CourseCatalog*

- Arhitectul software a decis utilizarea mecanismului JDBC pentru interacţiunea cu baza de date legacy.
- Proiectantul este responsabil cu aplicarea comportamentului general al mecanismului la aplicaţia curentă.

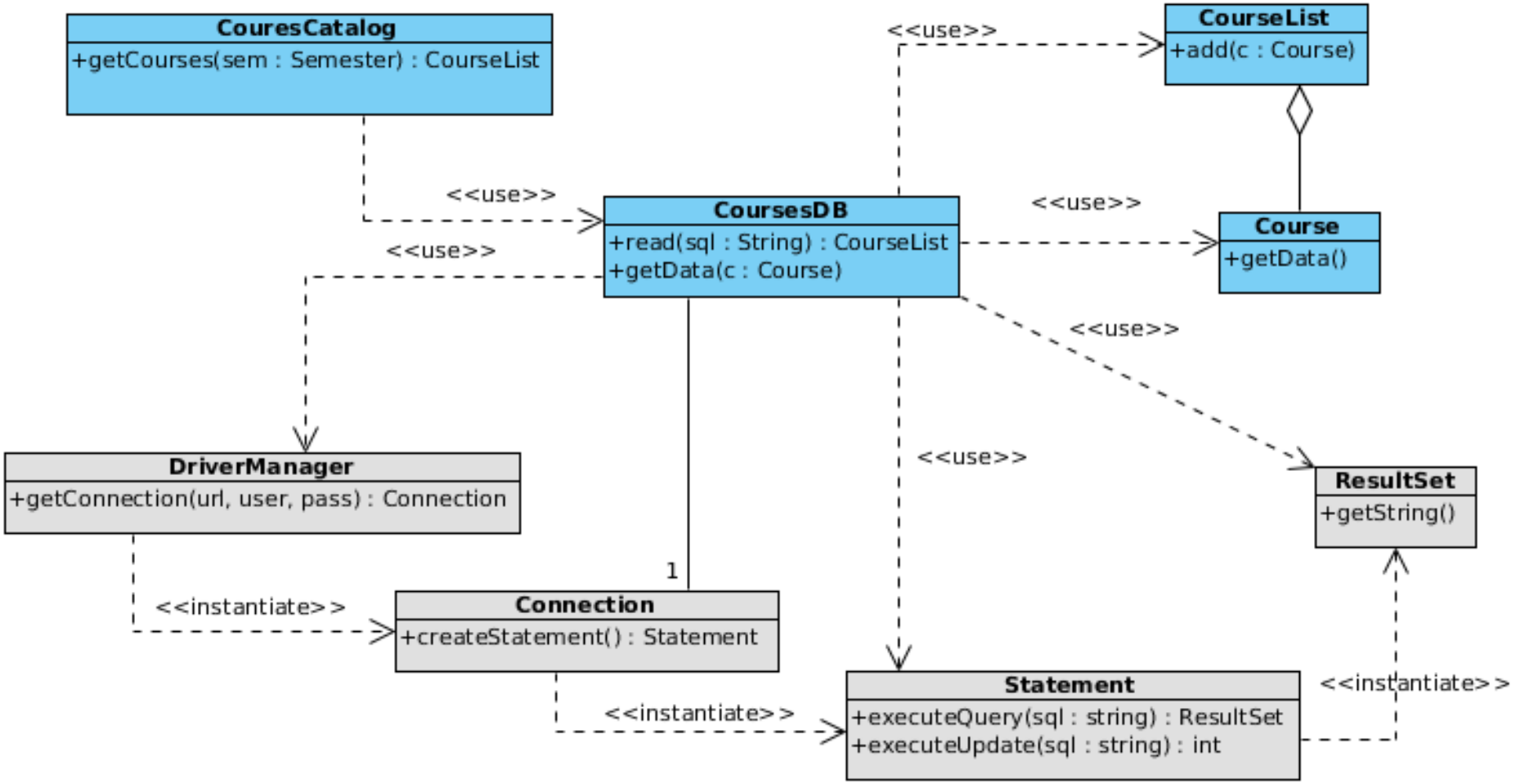
COMPORTAMENTUL INTERN AL SUBSISTEMELOR

Pas 1. Crearea clasei care realizează interfaţa.



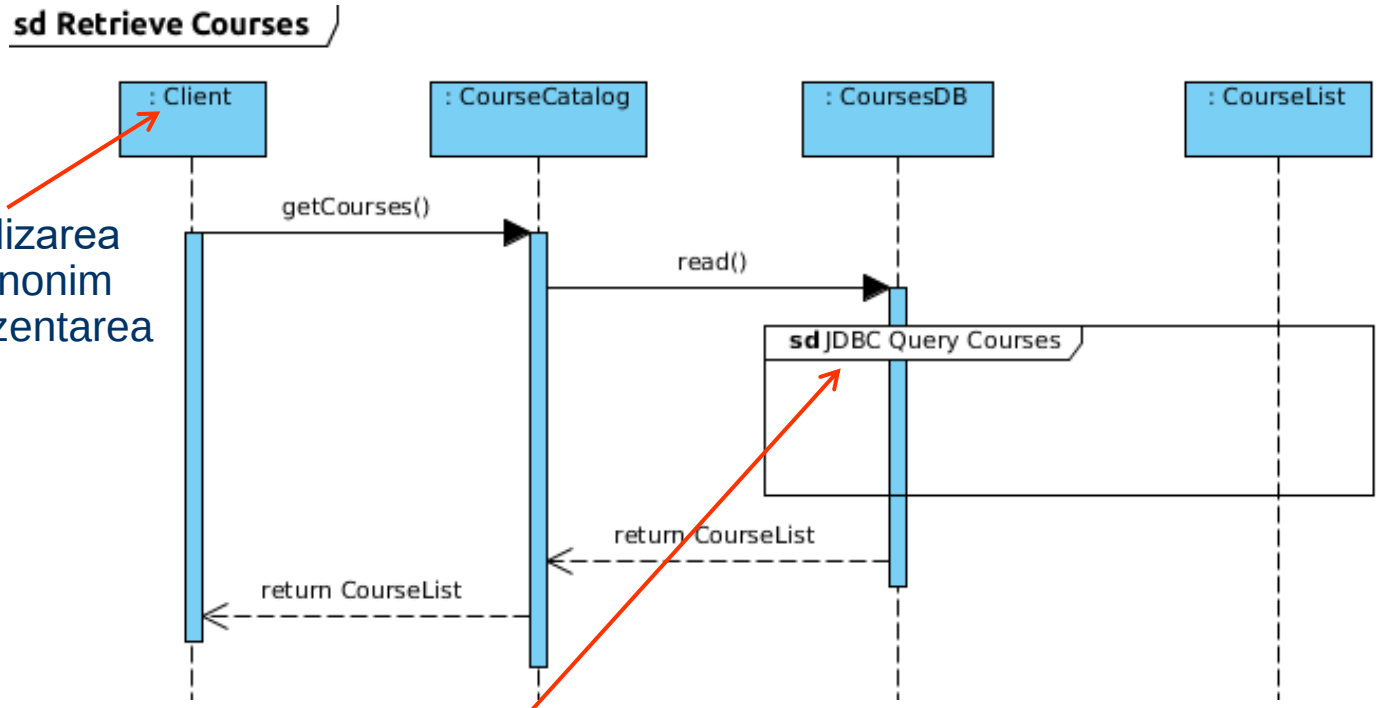
COMPORTAMENTUL INTERN AL SUBSISTEMELOR

Pas 2. Incorporare mecanism de proiectare (JDBC)



COMPORTAMENTUL INTERN AL SUBSISTEMELOR

Pas 3. Descrierea comportamentului cu diagrama de secvențe.



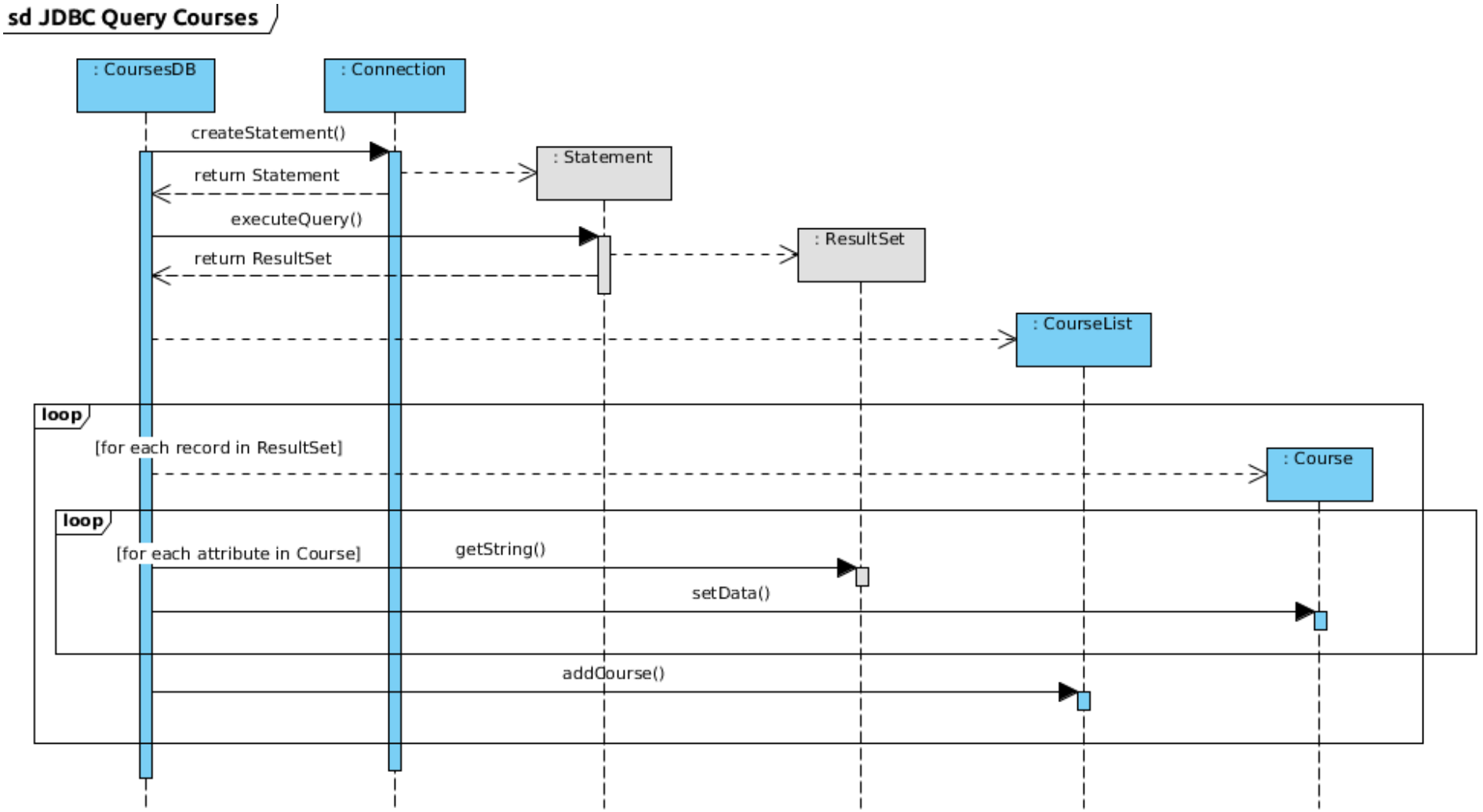
Observaţi utilizarea unui obiect anonim pentru reprezentarea clientului.

Pentru a evita duplicarea şi/sau pentru simplificarea reprezentărilor, putem utiliza un frame pentru înlocuirea unei secvenţe complexe reprezentată separat.

COMPORTAMENTUL INTERN AL SUBSISTEMELOR

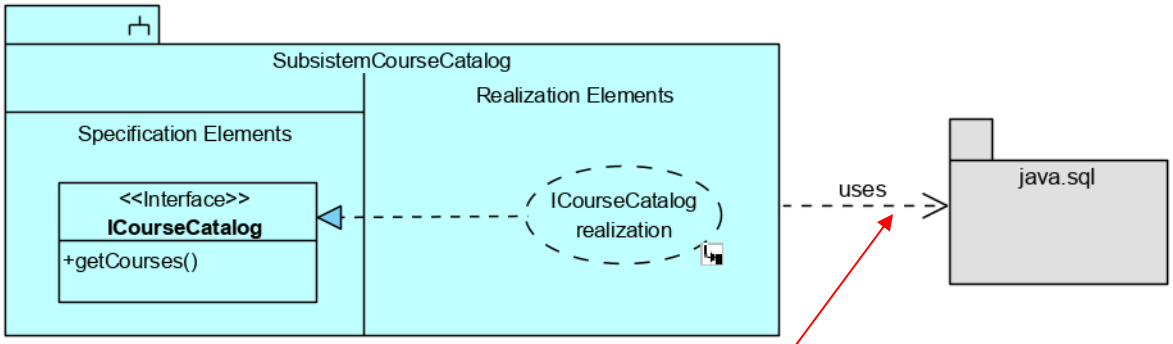
Pas 3. Descrierea comportamentului cu diagrama de secvențe.

Detalierea frame-lui **“JDBCQuery Courses”.**



COMPORTAMENTUL INTERN AL SUBSISTEMELOR
Pas 4. Controlul dependențelor.

- Controlul dependenţelor este critic la nivelul arhitecturii
- Dependenţele rezultă din:
  - Relaţiile dintre elemente,
  - Referinţele la alte elemente în tipurile parametrilor operaţiei sau valorii returnată,
  - Referinţele la alte elemente în tipurile de atribute.



- În cazul subsistemului din exemplu a fost simplu de determinat dependenţele de alte componente.
- Pentru sisteme complexe acest proces este greu de gestionat.

Instrumentele software (ex. Rational Software Architect de la IBM) oferă facilităţi automate de detectare a relaţiilor şi a conflictelor la nivelul arhitecturii.

COMPORTAMENTUL INTERN AL SUBSISTEMELOR

Oportunități de reutilizare

1. Reutilizare șabloane cunoscute

Exemple: șabloane de proiectare, JDBC, etc

2. Uneori se pot descoperi șabloane de interacțiune care se repetă în realizările interfețelor diferitelor subsisteme.

Arhitectul trebuie să analizeze modelele privind în interiorul subsistemelor și peste limitele acestora pentru a descoperi aceste tipare.

Un șablon astfel descoprit va fi extras și documentat separat, pentru ca apoi să poată fi reutilizat.

Evaluare formativă

1. În ce condiții sunt interschimbabile două subsisteme ?

2. Ce diagrame UML se folosesc pentru modelarea comportamentului intern al subsistemelor ?

https://forms.gle/i5bgxenc5xFKD81K8