Modele de analiza si modele de proiectare

ANALIZA

- Diagrame de clase
- Diagrame de cazuri de utilizare
- Diagrame de secvente
- Descrieri ale cazurilor de utilizare
- Diagrame de stari
- Diagrame de activitati

PROIECTARE

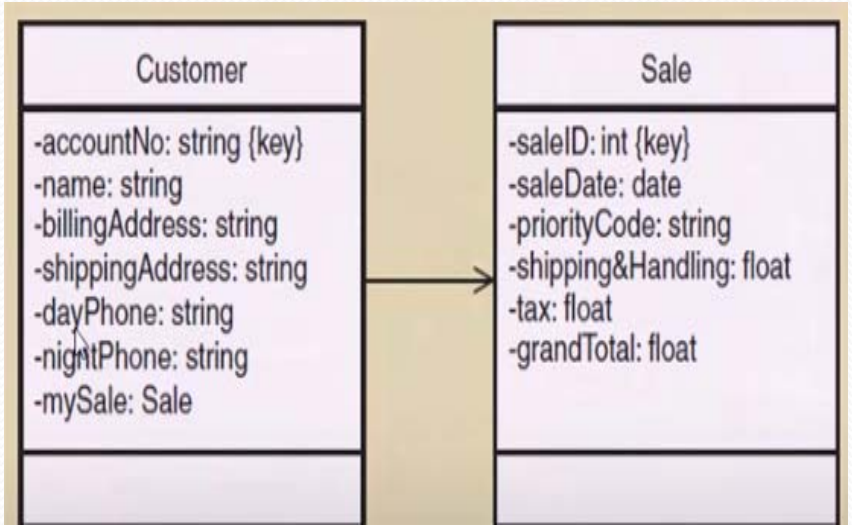
- Diagrame de pachete
- Diagrame de clase - proiectare
- Diagrame de secvente
- Schema bazei de date
- Interfata cu utilizatorul
- Diagrama de componente
- Diagrama de desfasurare
- Securitatea si controlul sistemului

Diagrama detaliata a claselor

- Se parcurg pe rand **cazurile de utilizare**
- Se aleg clasele domeniului care sunt implicate in cazul de utilizare; se verifica **preconditiile** si **postconditiile** pentru completarea acestora
- Se adauga o **clasa controller** care sa fie responsabila *pentru cazul de utilizare*
- Se determina cerintele de **vizibilitate a navigarii**
- Se completeaza **atributele** fiecarei clase cu *vizibilitate* si *tip*
- **Observatie**: de multe ori asocierile si multiplicitatile sunt eliminate din diagrama, pentru a pune accentul pe navigare, dar ele se pot pastra

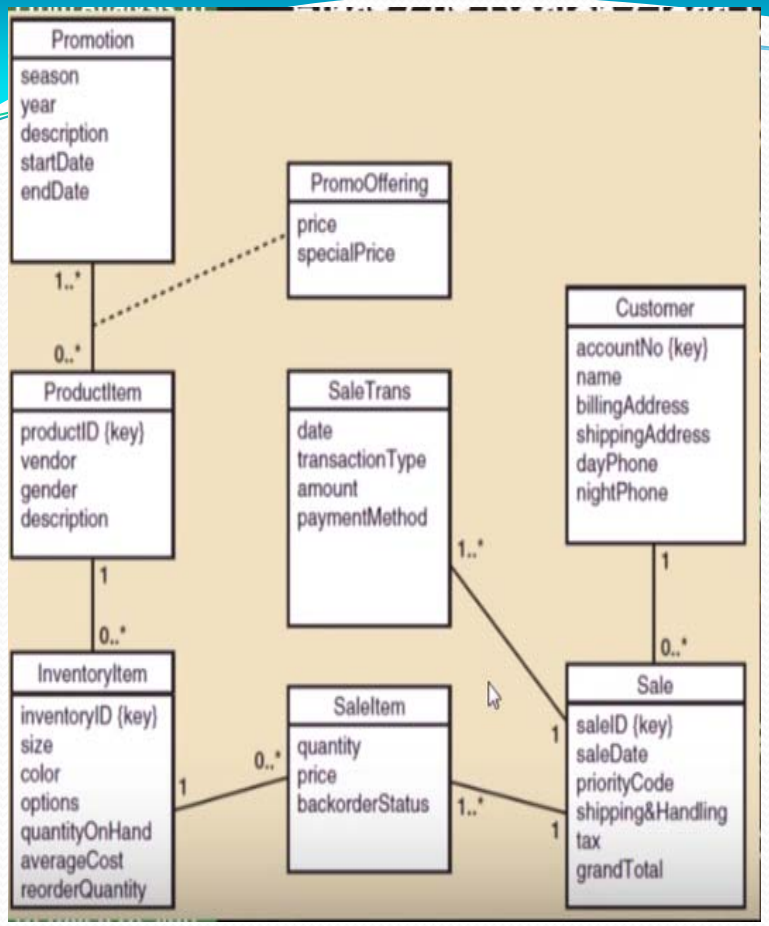
Vizibilitatea navigarii

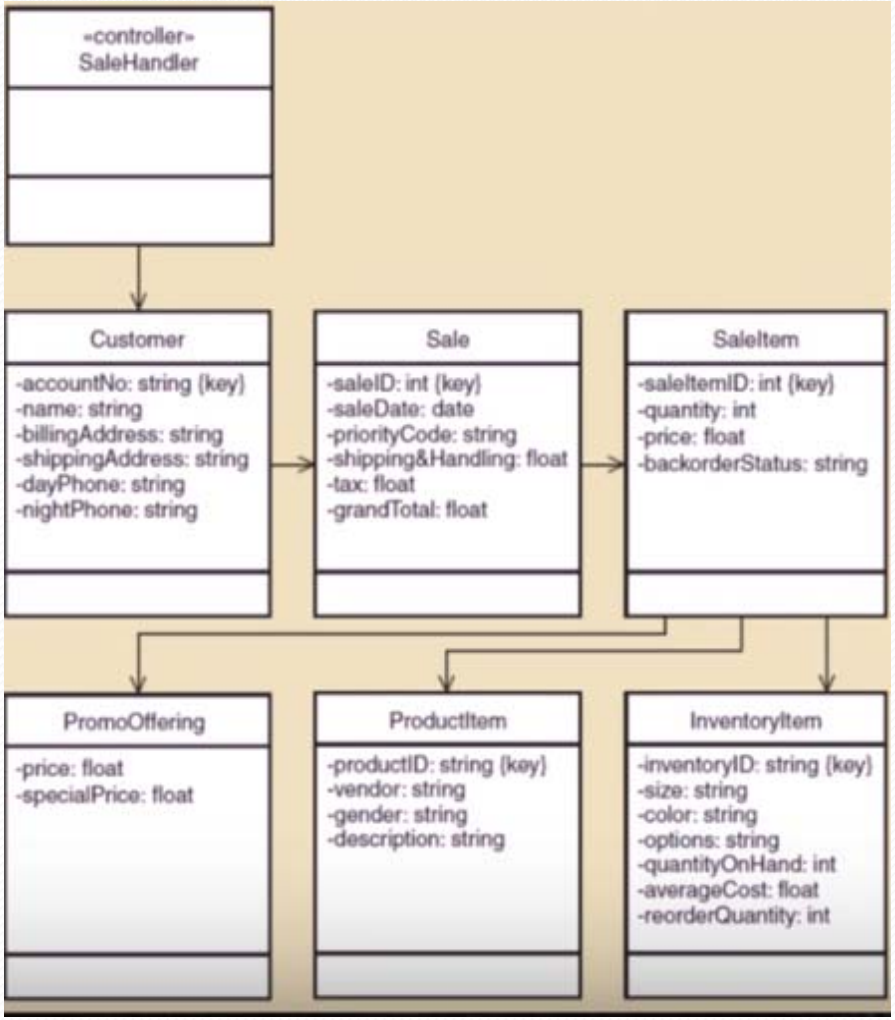
- Abilitatea unui obiect de a vedea si interactiona cu alt obiect
- Realizata prin adaugarea intr-o clasa a unei variabile referinta la obiect
- Apare ca o sageata pe capatul asocierii – Clientul poate vedea si interactiona cu Vanzare



Reguli pentru navigabilitate

- Asocierile **unu la multi** care indica o relatie superior-subordonat asigura navigarea de la superior la subordonati
- Asocierile obligatorii in care obiectele dintr-o clasa **nu pot sa existe** fara obiectele din alta clasa asigura de obicei navigarea de la clasa mai independenta la cea mai dependenta
- Cand un obiect **are nevoie de informatii** de la un alt obiect, ar putea fi nevoie de o sageata de navigare

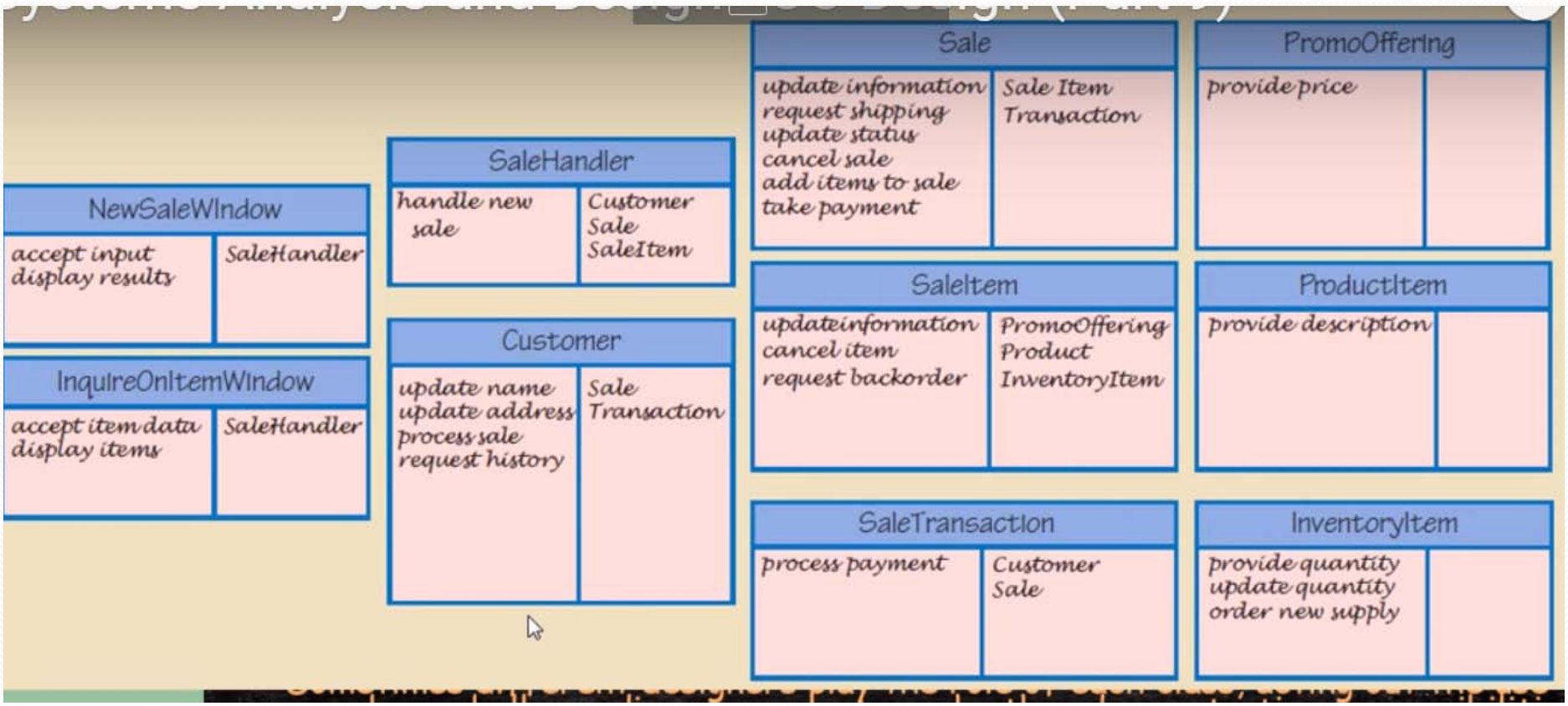

Promotion
season
year
description
startDate
endDate
PromoOffering
price
specialPrice
1..*
0..*
Customer
accountNo {key}
name
billingAddress
shippingAddress
dayPhone
nightPhone
ProductItem
productID {key}
vendor
gender
description
SaleTrans
date
transactionType
amount
paymentMethod
1
0..*
1..*
1
0..*
InventoryItem
inventoryID {key}
size
color
options
quantityOnHand
averageCost
reorderQuantity
SaleItem
quantity
price
backorderStatus
Sale
saleID {key}
saleDate
priorityCode
shipping&Handling
tax
grandTotal
1
0..*
1..*
1

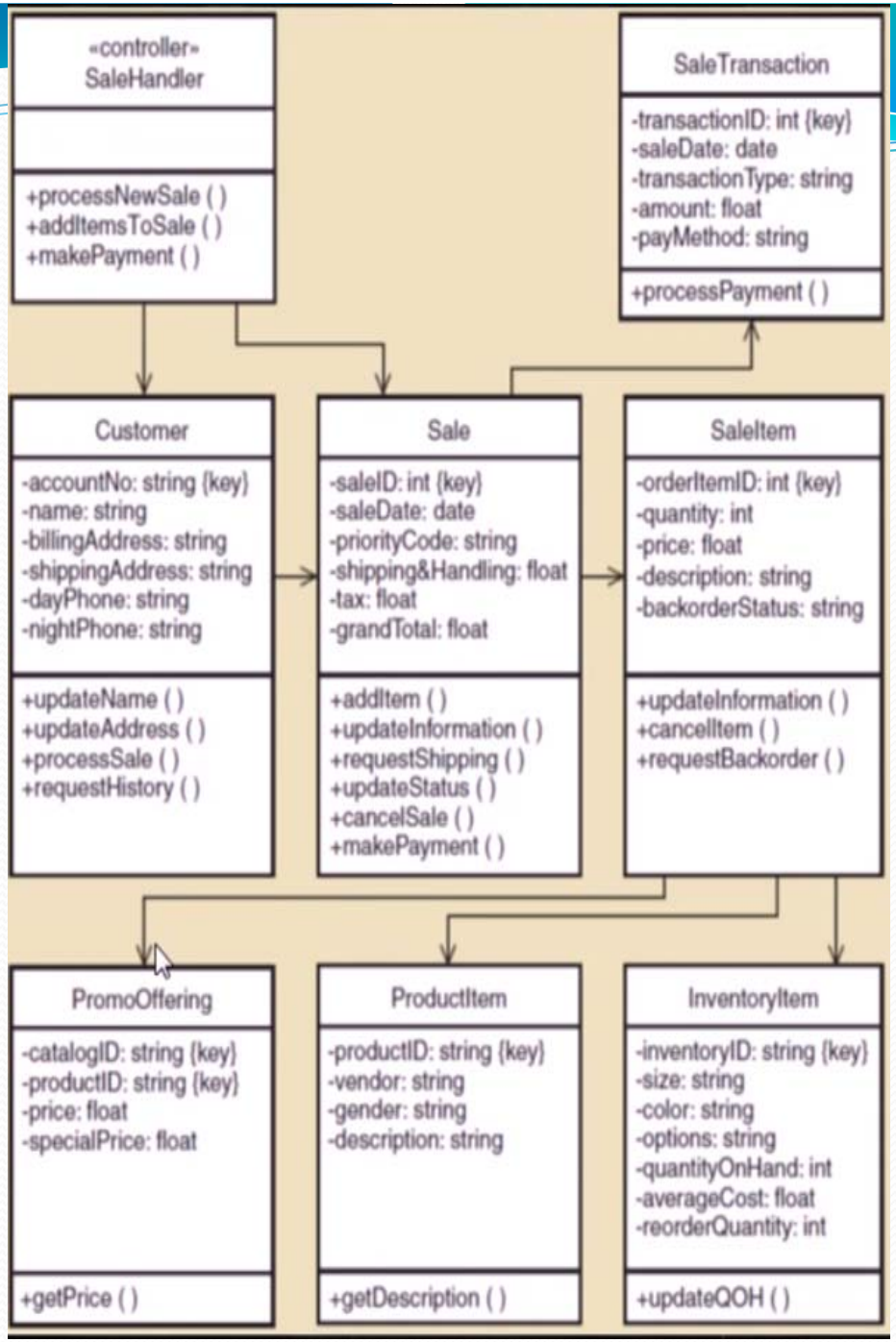

«controller»
SaleHandler
Customer
-accountNo: string {key}
-name: string
-billingAddress: string
-shippingAddress: string
-dayPhone: string
-nightPhone: string
Sale
-saleID: int {key}
-saleDate: date
-priorityCode: string
-shipping&Handling: float
-tax: float
-grandTotal: float
SaleItem
-saleItemID: int {key}
-quantity: int
-price: float
-backorderStatus: string
PromoOffering
-price: float
-specialPrice: float
ProductItem
-productID: string {key}
-vendor: string
-gender: string
-description: string
InventoryItem
-inventoryID: string {key}
-size: string
-color: string
-options: string
-quantityOnHand: int
-averageCost: float
-reorderQuantity: int

Metodele claselor

- Se poate folosi tehnica **CRC – Class, Responsibility, Collaboration cards**
  - Care sunt responsabilitatile unei clase si cum colaboreaza cu alte clase pentru a realiza cazul de utilizare
  - Se obtin prin brainstorming
- Se pot folosi **diagramele de secventa detaliate** – ***fiecare mesaj receptionat de un obiect*** al unei clase trebuie sa aiba in corespondenta ***o metoda*** in clasa respectiva

Exemplu de carduri CRC




«controller»
SaleHandler
+processNewSale ( )
+addItemsToSale ( )
+makePayment ( )
SaleTransaction
-transactionID: int {key}
-saleDate: date
-transactionType: string
-amount: float
-payMethod: string
+processPayment ( )
Customer
-accountNo: string {key}
-name: string
-billingAddress: string
-shippingAddress: string
-dayPhone: string
-nightPhone: string
+updateName ( )
+updateAddress ( )
+processSale ( )
+requestHistory ( )
Sale
-saleID: int {key}
-saleDate: date
-priorityCode: string
-shipping&Handling: float
-tax: float
-grandTotal: float
+addItem ( )
+updateInformation ( )
+requestShipping ( )
+updateStatus ( )
+cancelSale ( )
+makePayment ( )
SaleItem
-orderItemID: int {key}
-quantity: int
-price: float
-description: string
-backorderStatus: string
+updateInformation ( )
+cancelItem ( )
+requestBackorder ( )
PromoOffering
-catalogID: string {key}
-productID: string {key}
-price: float
-specialPrice: float
+getPrice ( )
ProductItem
-productID: string {key}
-vendor: string
-gender: string
-description: string
+getDescription ( )
InventoryItem
-inventoryID: string {key}
-size: string
-color: string
-options: string
-quantityOnHand: int
-averageCost: float
-reorderQuantity: int
+updateQOH ( )

Protectia in fata schimbarii

- Un principiu al proiectarii este de a separa partile care sunt **stabile** de partile care sufera numeroase schimbari
- Se separa **formularele** si **paginile din interfata cu utilizatorul** care au probabilitate mare de a se modifica de logica aplicatiei
- **Conexiunea la baza de date si logica SQL** care probabil se vor modifica se pastreaza in **clase separate** de logica aplicatiei
- Se utilizeaza **clase adaptor** care se pot schimba pentru interactiunea cu alte sisteme
- Daca se alege intre doua variante de proiectare, se va alege cea care ofera o protectie mai mare in fata schimbarii

Proiectarea bazei de date

- Se pleaca de la **modelul claselor domeniului**
- Se alege **structura** bazei de date
  - De obicei, se lucreaza cu baze de date relaționale, dar pot există platforme care lucrează cu baze de date orientate obiect
- Se proiectează **arhitectura** BD (distribuită, etc)
- Se proiectează schema bazei de date
  - Tabelele și coloanele în relațional
- Se proiectează restricțiile de integritate referențială
  - Referințe prin chei externe

Proiectarea interfețelor

- După **finalizarea** diagramelor de cazuri de utilizare și a unor prime versiuni stabile ale diagramelor de interacțiune și de clase se recomandă implementarea unui **prototip al sistemului informatic**. Acest prototip se numește *prototip de interfață* deoarece are **rolul** de a:
  - rafina relațiile dintre actori și clasele de interfață;
  - obține *feedback* din partea beneficiarului (clientului) asupra aspectului vizual al aplicației.

Proiectarea interfețelor

- Primul pas - investigarea *așteptării actorilor asupra interfeței* prin completarea unor **chestionare** specifice formate din următoarele întrebări:
  - ce nivel de pregătire (informatică) necesită actorul pentru a realiza o anumită funcționalitate?
  - actorul are experiență de lucru în medii bazate pe ferestre?
  - actorul are experiență în utilizarea altor sisteme de automatizare a procesului modelat?
  - este necesară consultarea unor documente/cataloage în paralel cu utilizarea aplicației?
  - actorul dorește implementarea unor facilități de tip 'salvare/restaurare'?

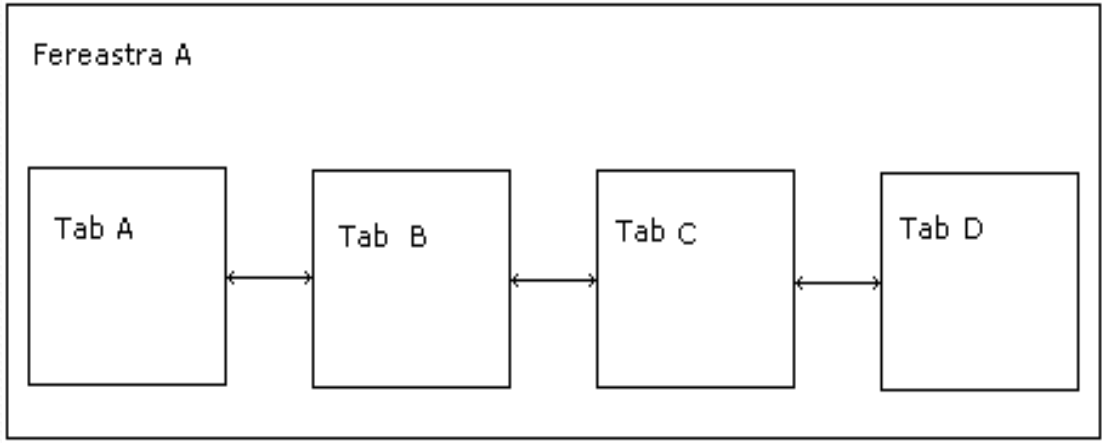
Proiectarea interfețelor

- **Scopurile prototipului** sunt:
  - stabilirea unor *cerințe ale interfeței* pentru funcționalități cheie ale aplicației;
  - se demonstrează clientului (într-o formă vizuală) că cerințele proiectului au fost bine înțelese și sunt realizabile;
  - începerea etapei de *dezvoltare a elementelor standard ale interfeței*.

Proiectarea interfețelor

- Se utilizează **hărți (diagrame) de structură a ecranului** în care se descrie fluxul aplicației urmând căile principale ale cazurilor de utilizare.
- Mod de reprezentare:
  - **forme pătrate** pentru reprezentarea ferestrelor *modale* (necesită un răspuns dat de utilizator pentru a se putea continua o activitate).
  - **forme pătrate cu colțurile rotunjite** Pentru reprezentarea ferestrelor *ne-modale*
- Direcția de traversare arată calea de navigare a ferestrelor.

Ferestre care conțin tab-uri



Diagramă de structură a ecranului

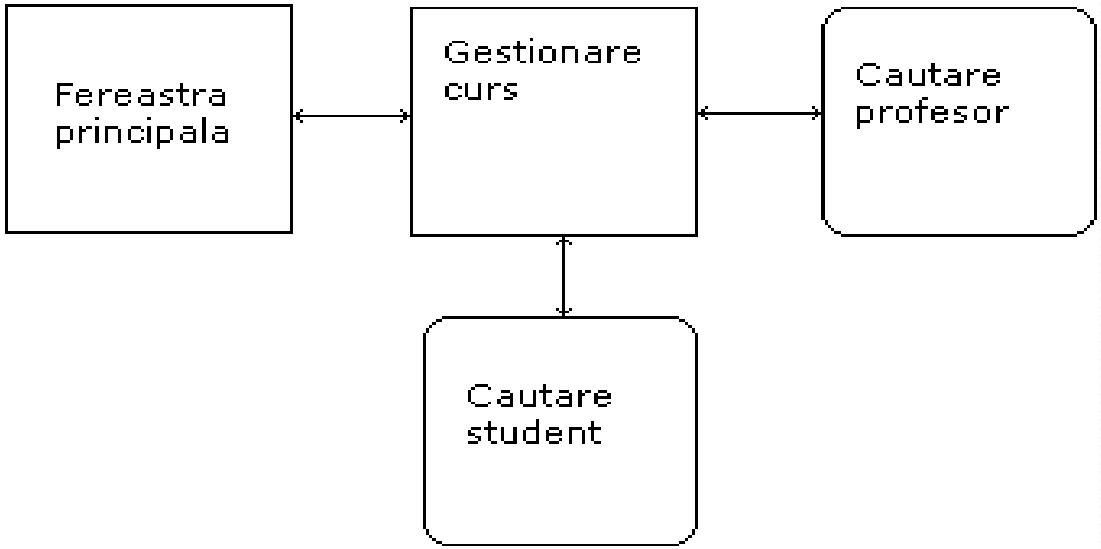
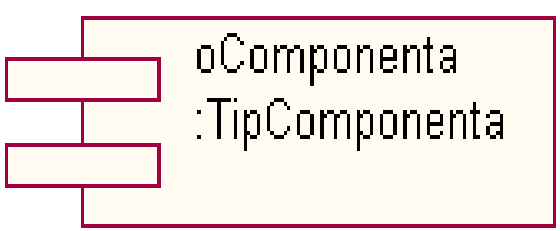


Diagrama de componente

- O diagramă de componente prezintă dependențele existente între diverse componente software ce compun un sistem informatic.
- Aceste **dependențe** sunt:
  - **statice** - au loc in etapele de compilare sau link-editare
  - **dinamice** - au loc in timpul execuției
- **O componentă** este un modul soft (cod sursa, cod binar, dll, executabil etc) cu o interfață bine definită.

Diagrama de componente

- În general numele unei componente este **numele fișierului** reprezentat de componentă.
- Obiectele implementate de o instanță a componentei se reprezintă grafic în interiorul simbolului instanței componentei.



Reprezentarea grafică a componentelor în UML

Diagrama de componente

- Diagrama de componente este un graf de componente între care există relaţii de **dependenţă** sau de **compunere** (componente incluse fizic în alte componente).
- **Dependenţele** între componente se reprezintă grafic prin linii întrerupte între o componentă client şi o componentă furnizor de servicii, orientate spre componenta furnizor.
- Relaţia de dependenţă semnifica faptul că *clasele incluse în componenta client pot moşteni, instanţia sau utiliza clase incluse în componenta furnizor*.
- De asemenea, pot exista **relaţii de dependenţă între componente şi interfeţe** ale altor componente, relaţii care semnifică faptul că un client utilizează operaţii ale componentei furnizor

Diagrama de componente

- Exemple de stereotipuri predefinite pentru componente:
  - programe principale (<<Main Program>>)
  - subprograme (<<SubProgram>>)
  - pachete (<<Package>>)
  - librarii cu legare dinamica (<<DLL>>)
  - procese (<<Task>>)
  - executabile (<<EXE>>

Exemplu de diagramă de componente

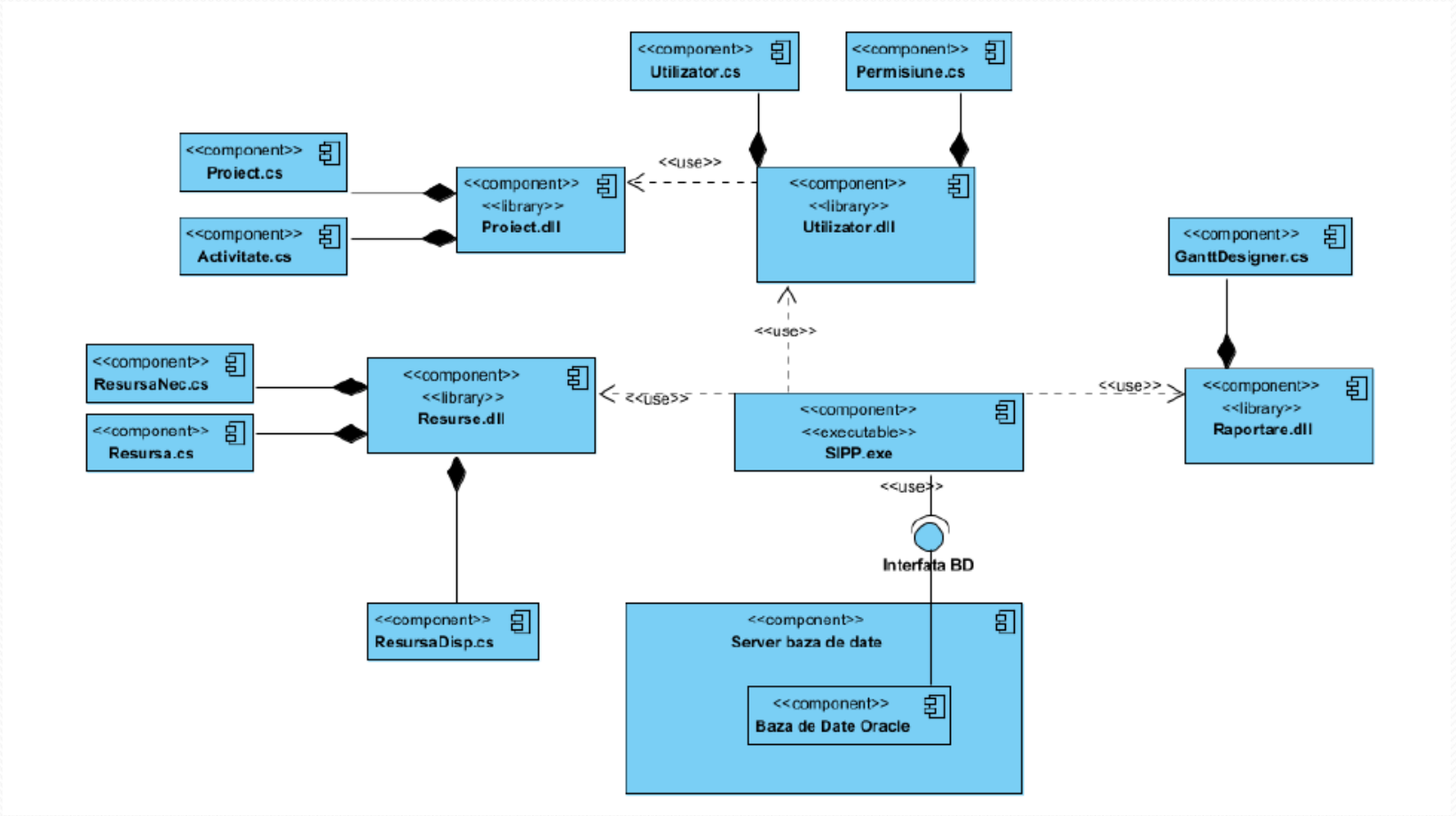


Diagrama de desfăşurare

- Diagramele de desfășurare prezintă configurația elementelor de procesare din timpul execuției și componentele, procesele și obiectele care le conțin.
- O diagramă de desfășurare este un graf de **noduri** conectate prin asocieri de **comunicare**.
- **Un nod** este o entitate fizică ce reprezintă o resursă de procesare, având o memorie și anumite capabilități de procesare (dispozitive de calcul, resurse umane, resurse de procesare mecanică).
- Un nod este reprezentat grafic prin intemediul unui paralelipiped. Un tip de nod are asociat un nume, iar o instanță a unui nod are asociate (opțional) un nume de instanță și un nume de tip (*nume instanță* : *nume tip*). **O asociere între două noduri** indică existența unei căi de comunicare între noduri.

Diagrama de desfăşurare

- Diagramele de desfăşurare pot fi utilizate pentru reprezentarea componentelor ce pot aparţine anumitor noduri prin imbricarea grafică a simbolului componentei în cadrul simbolului ce reprezintă nodul.
- Între componente pot exista şi relaţii de dependenţă.

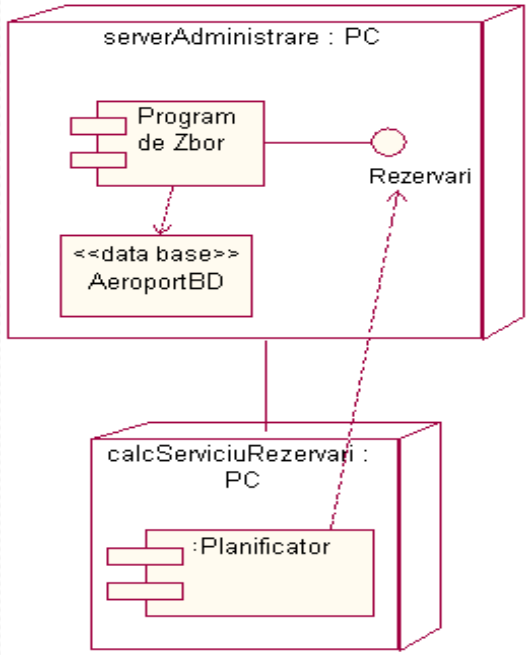


Diagrama de desfăşurare

- Diagramele de desfășurare conţin două tipuri de noduri: **medii de execuţie** si **dispozitive**.
  - **Mediile de execuţie** reprezintă componente hardware capabile să execute programe.
  - ***Dispozitivele*** reprezintă componente hardware fără putere de calcul. Numele asociat unui dispozitiv este in general unul generic (ex. imprimanta, modem, terminal etc)
- O conexiune reprezintă o legatură hardware (în general bidirecţională) între două dispozitive sau procesoare.

Exemplu de diagramă de desfăşurare