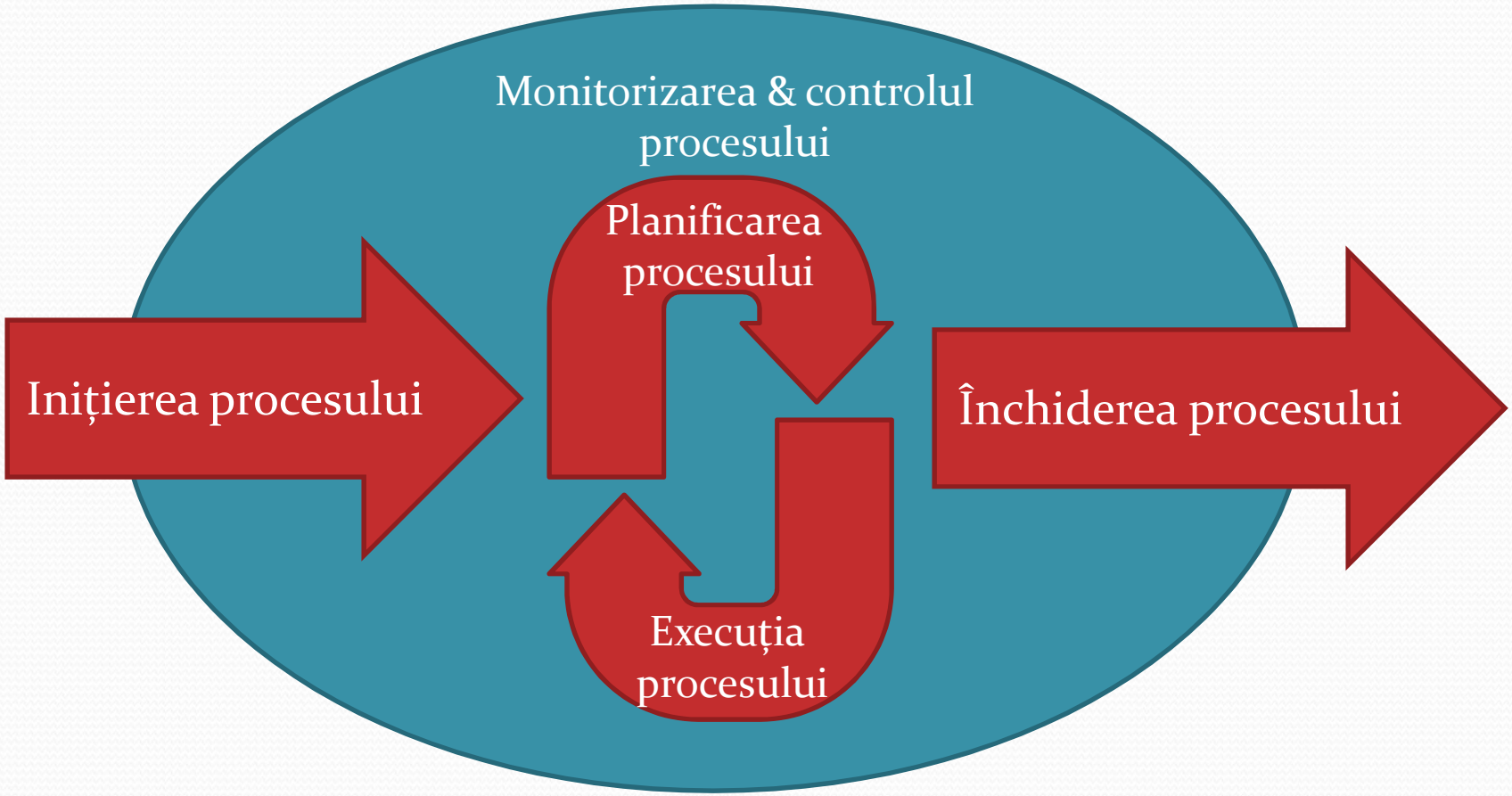
Software Engineering (Ingineria Sistemelor Soft)

Curs 7

Procese de execuție a proiectelor

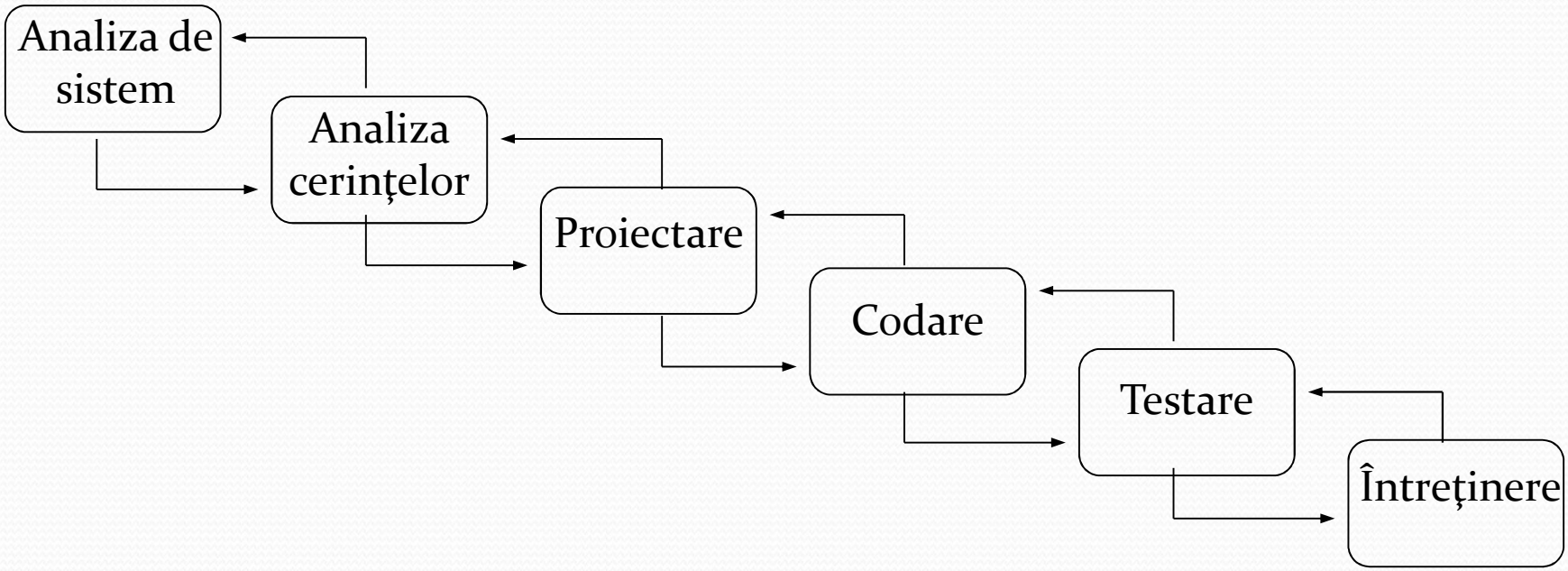
Lector dr. Pop Andreea-Diana


Monitorizarea & controlul procesului
Planificarea procesului
Inițierea procesului
Închiderea procesului
Execuția procesului

Procesul de execuție - etape

- Realizare / dezvoltare produs
- Emiterea ordinelor de schimbare
- Analiza situației proiectului
- Rapoarte asupra proiectului
- Revizuiri asupra bugetelor și planificărilor

Modelul clasic de dezvoltare a softului



Diagrame UML utilizate în faza de proiectare

- Diagrama cazurilor de utilizare
- Diagrama de colaborare
- Diagrama de secvență

} Diagrame de interacțiune

- Diagrama de clase
- Diagrama de arhitectură

- Modelul Entitate-Relație de modelare a Bazei de date

Diagrama cazurilor de utilizare

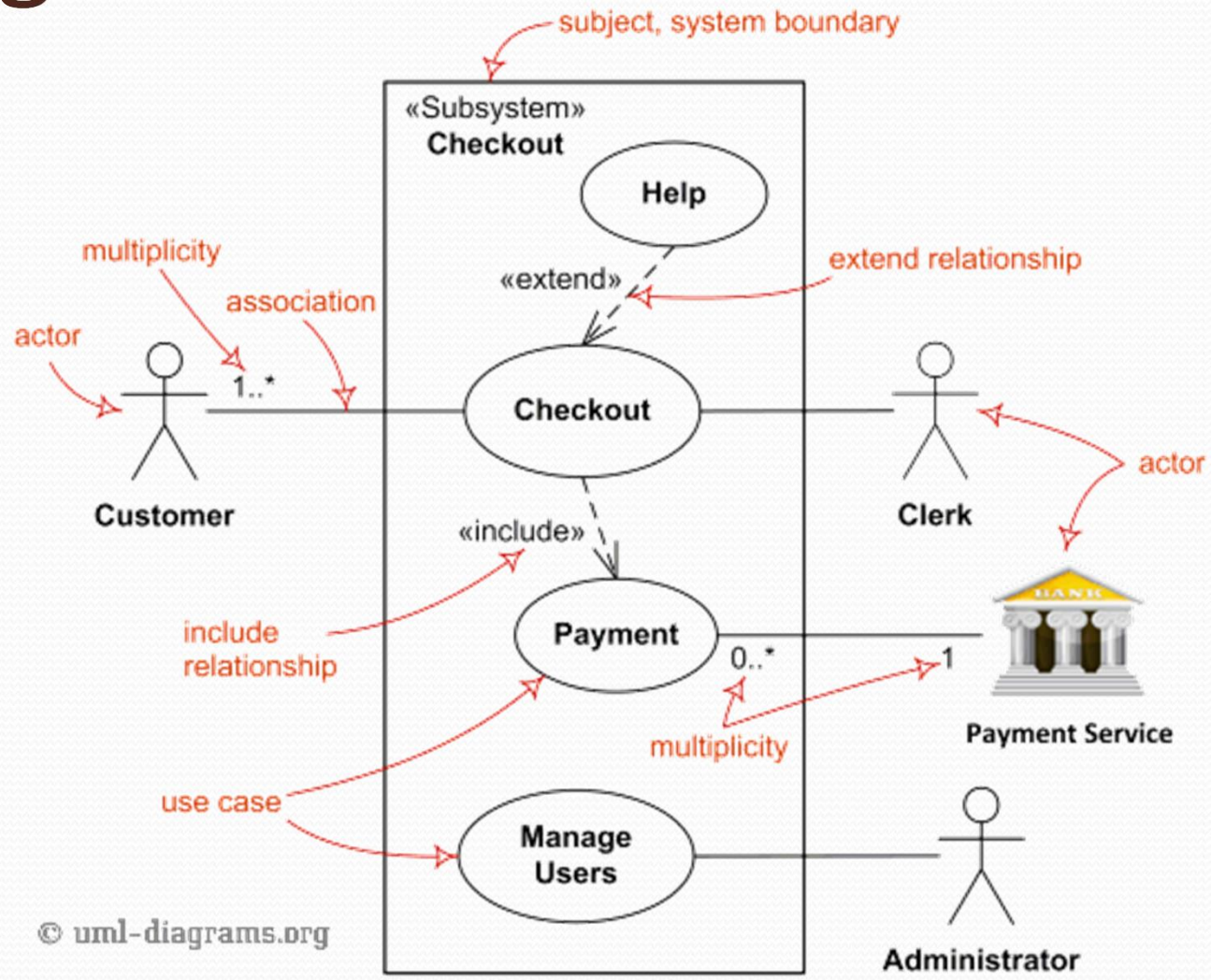


Diagrama de colaborare

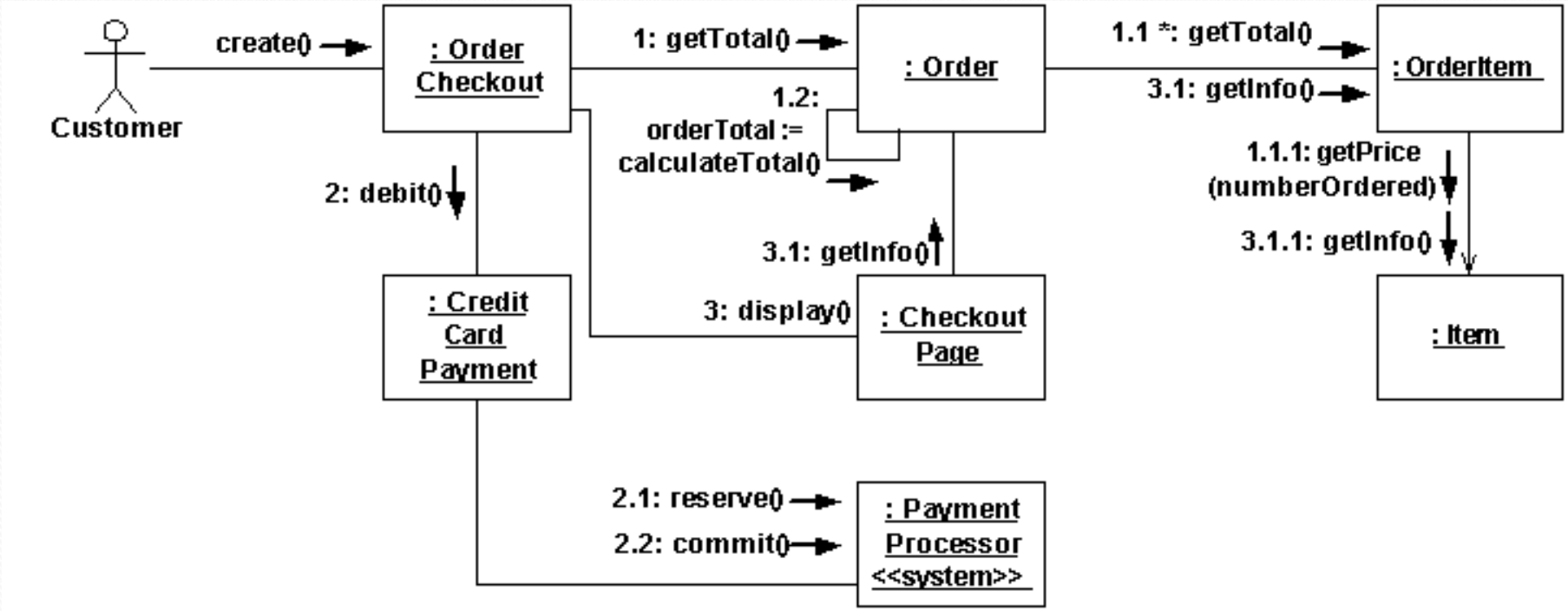


Diagrama de secvență

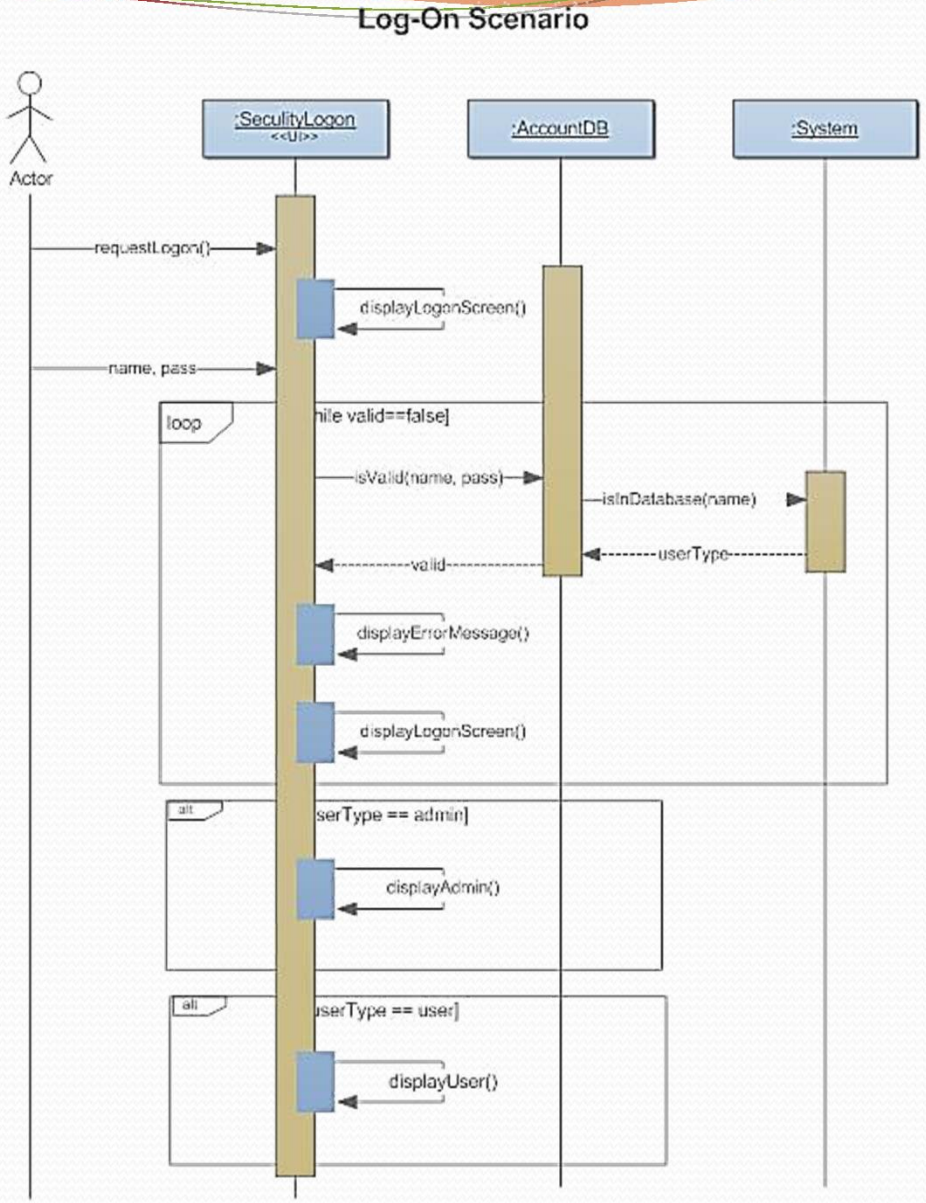
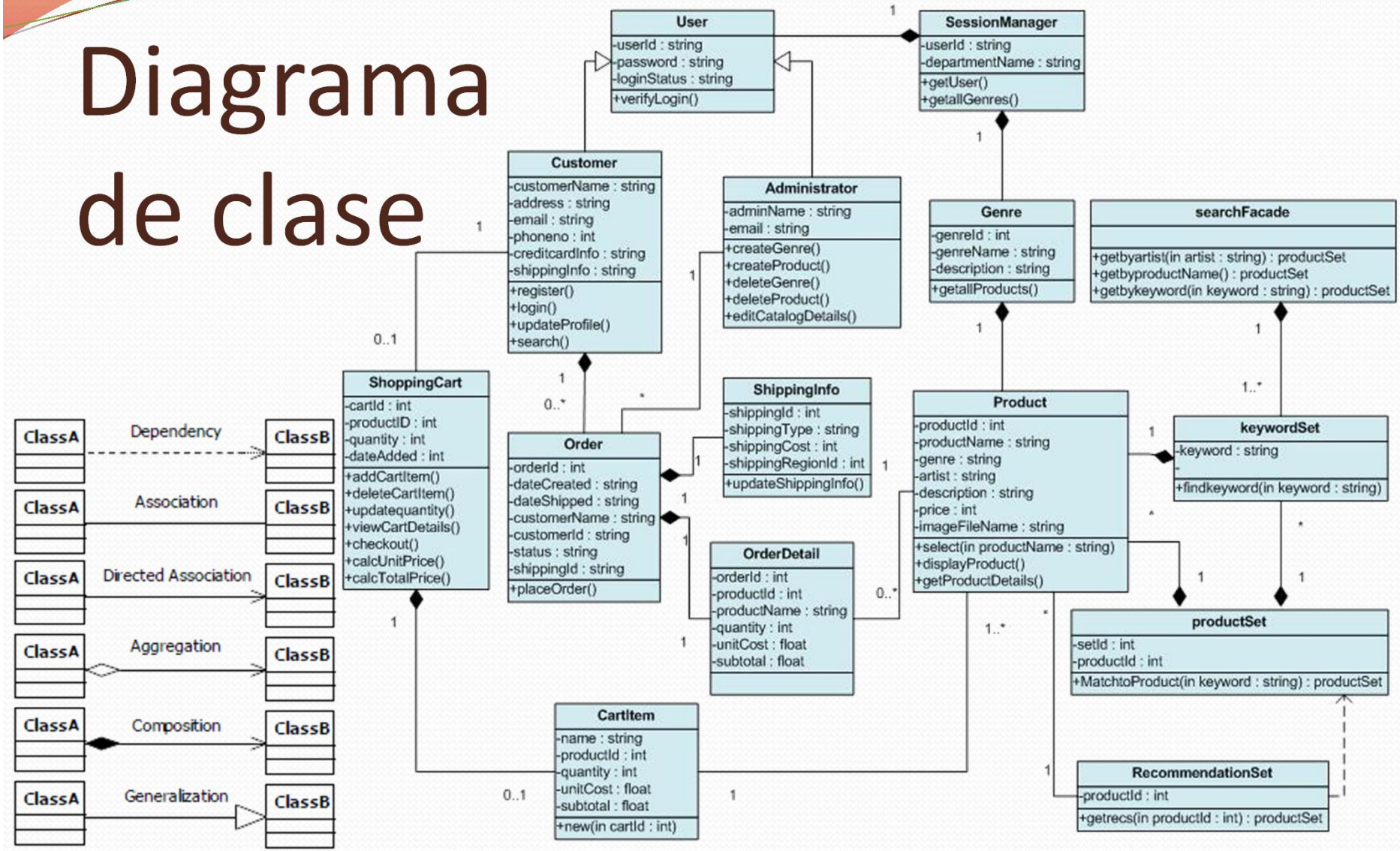


Diagrama de clase



Diagramă UML pentru componente

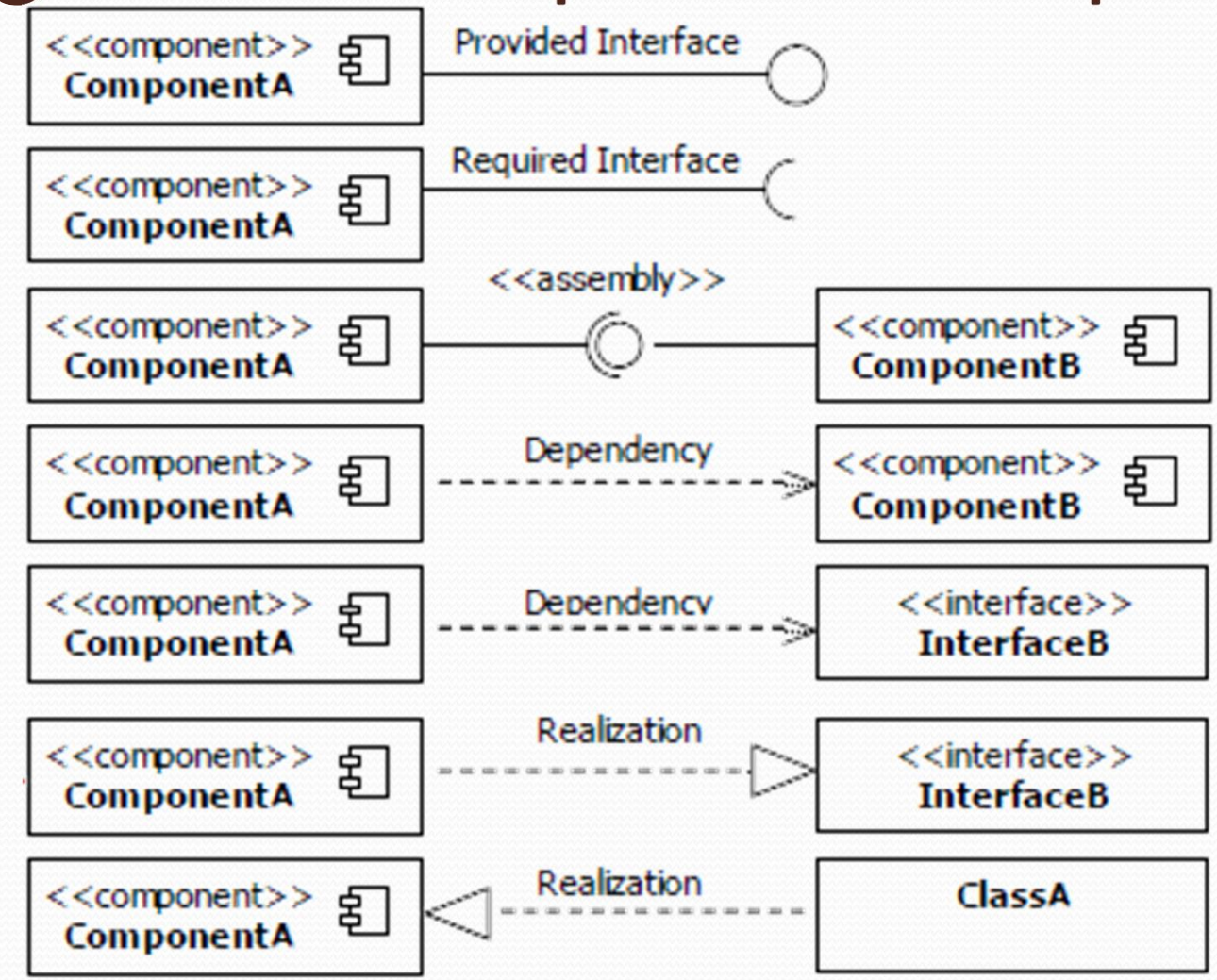
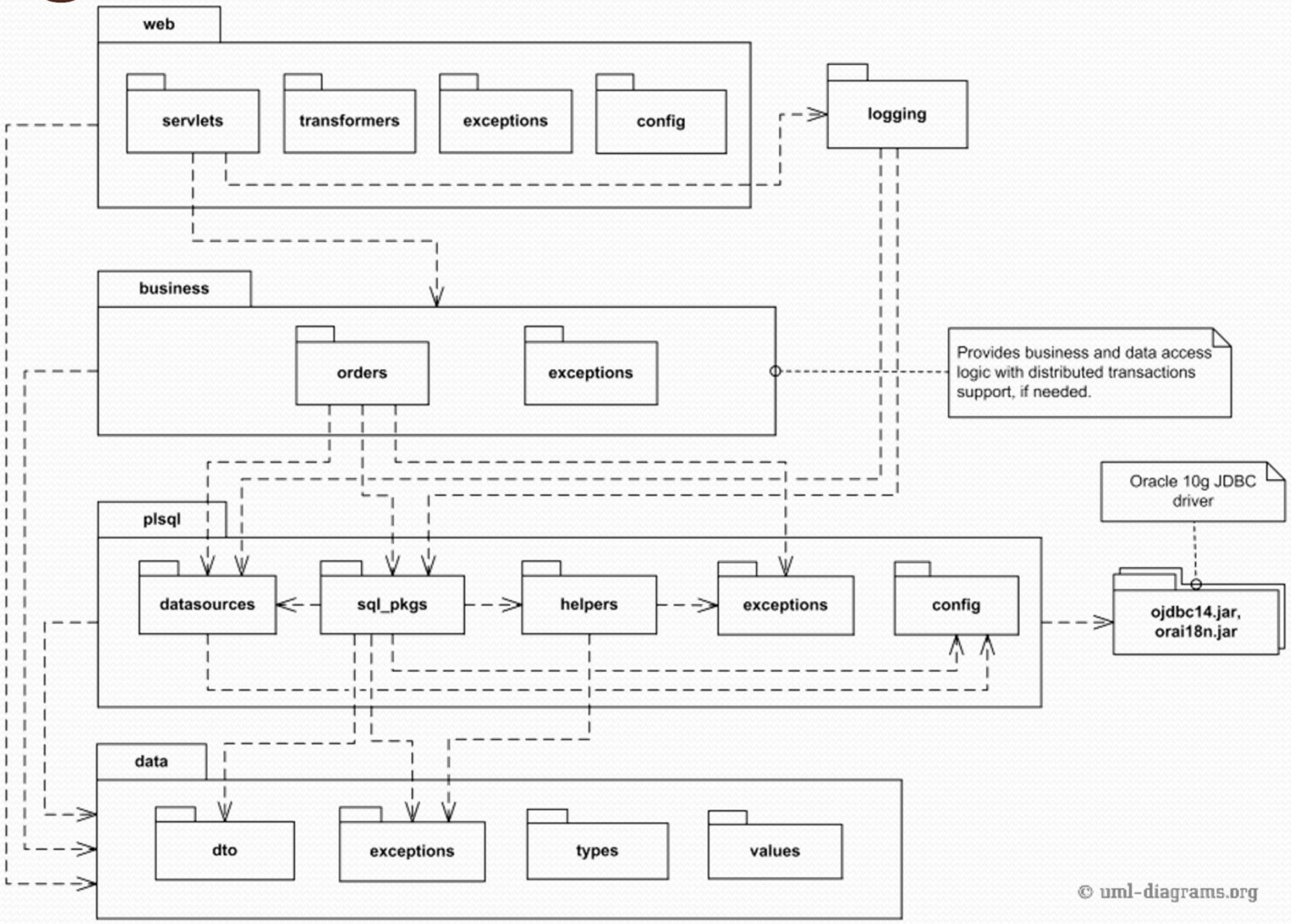
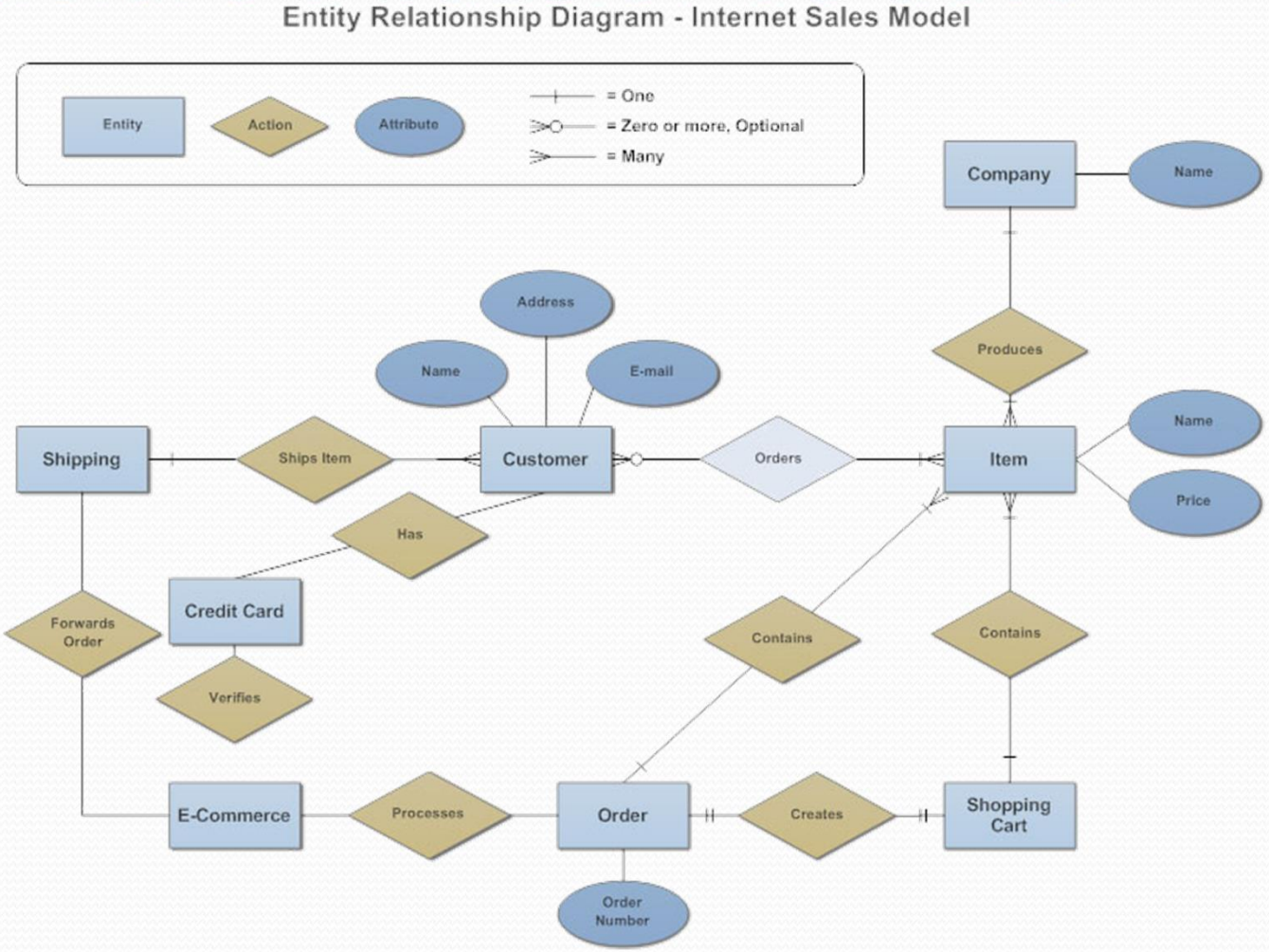


Diagrama de arhitectură



Modelul Entitate-Relație de modelare a bazelor de date



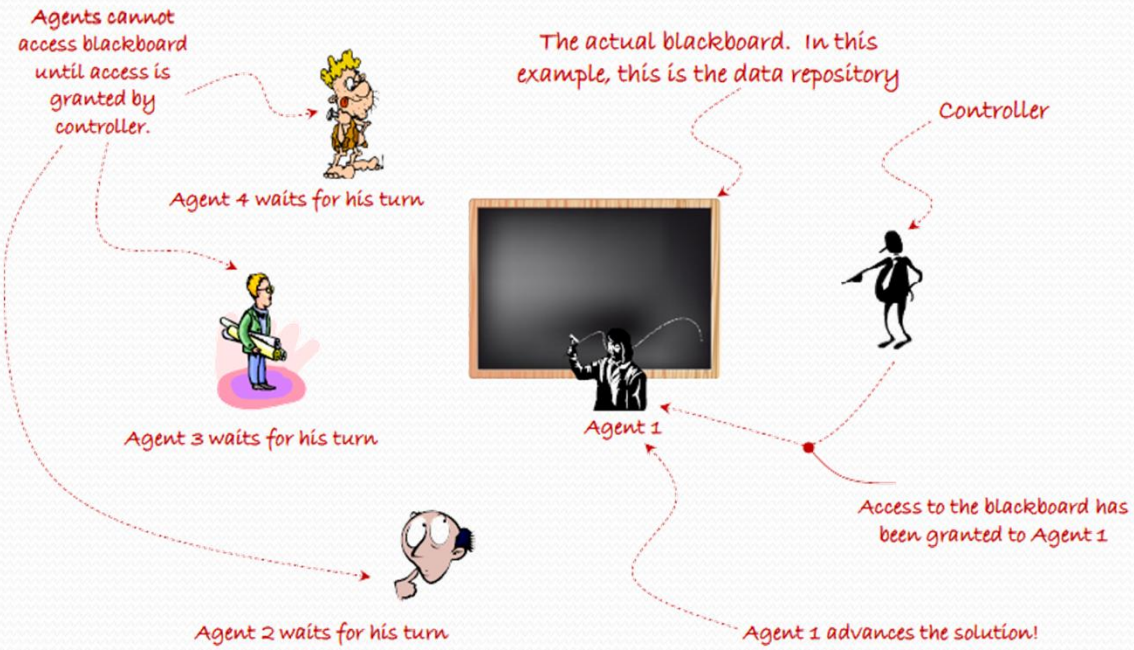
Design patterns – Șabloane de arhitectură

≈ rețete

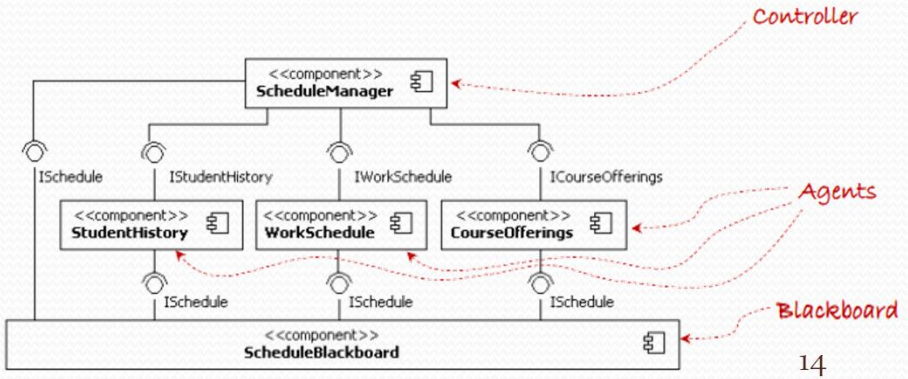
- Blackboard
- Pipes-and-Filters
- Client Server
- Broker

- Model-View-Controller
- Abstract Factory
- Builder
- Prototype
- Singleton
- Adapter
- Composite
- Facade
- Iterator
- Observer

Blackboard

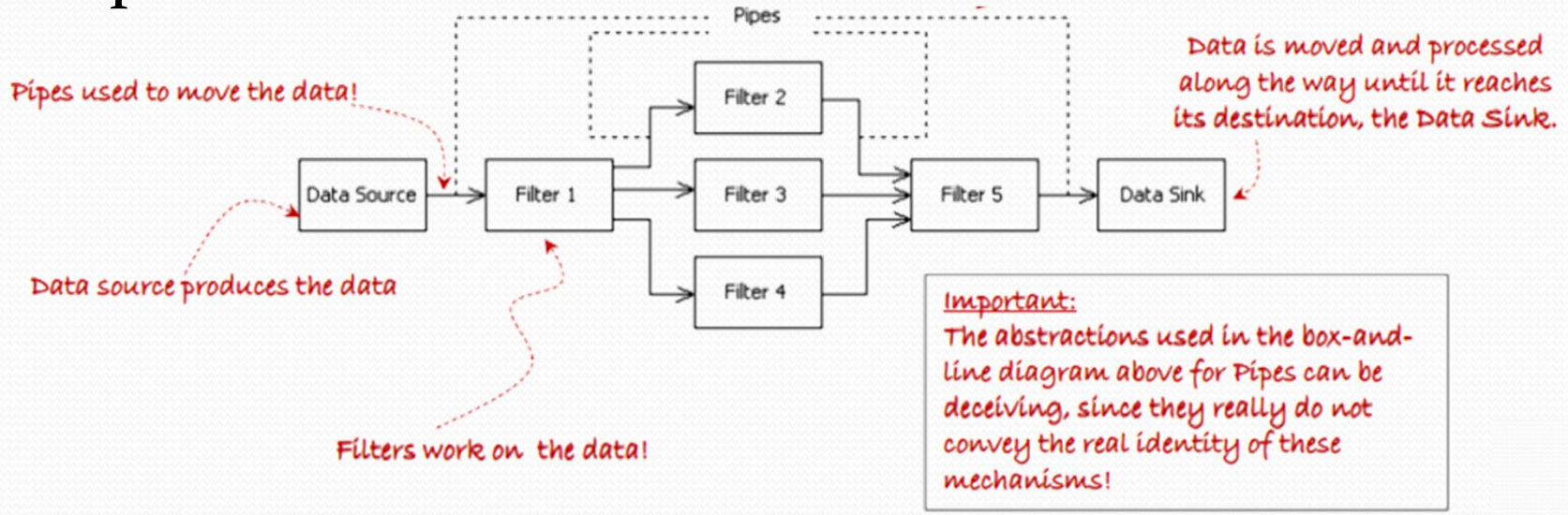


Agenții partajează aceeași resursă sub coordonarea Controller-ului



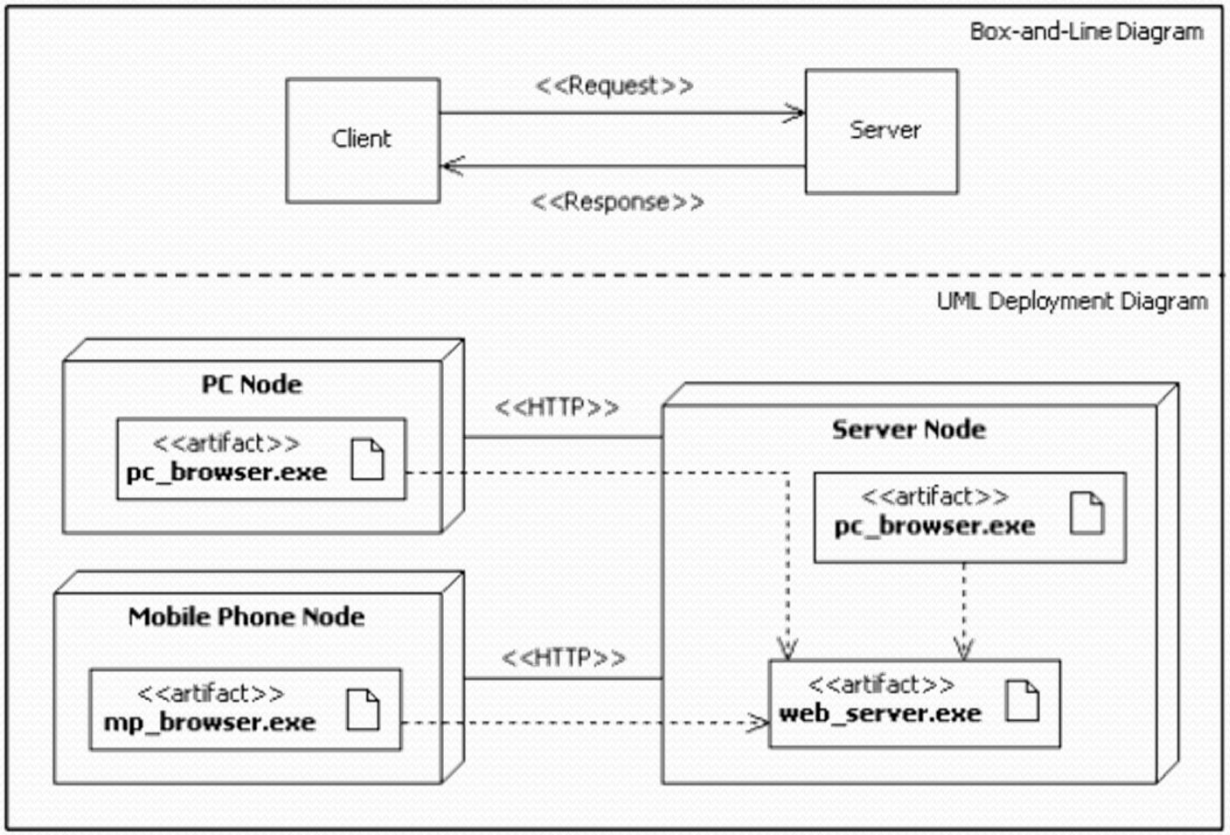
Pipes-and-Filters

Datele sunt mutate prin ”țevi” (pipes) spre senzorul de date ”Data Sink”, și pe drum pot trece prin filtre unde sunt prelucrate.



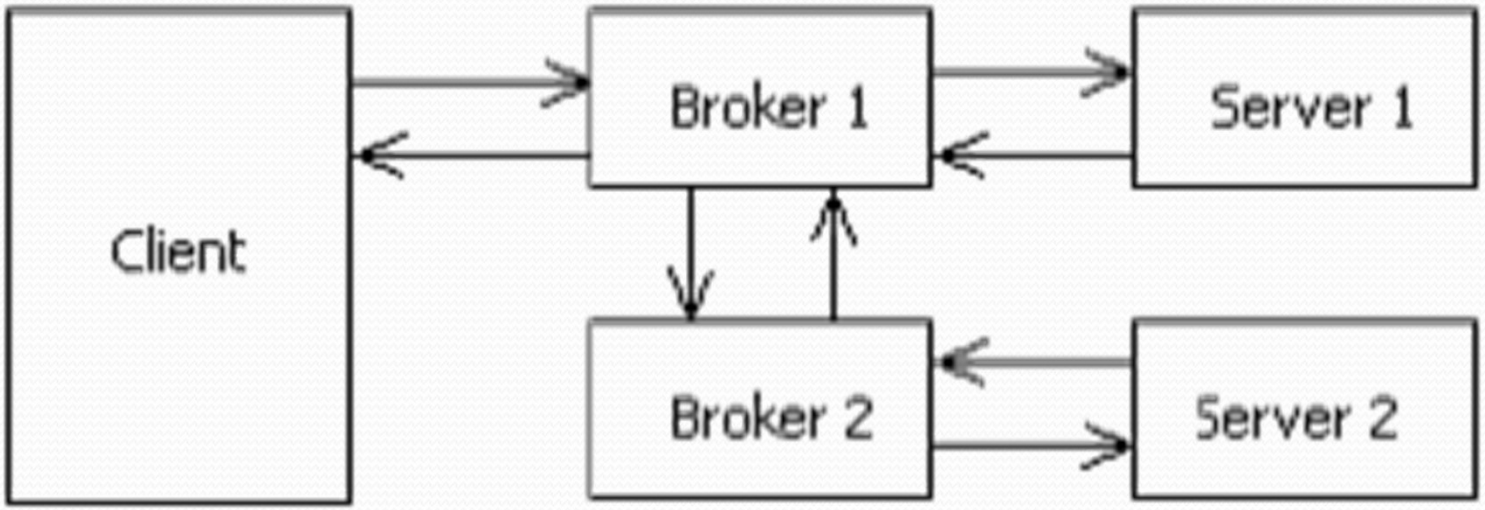
Client Server

Clientul cere un serviciu Server-ului, care îi transmite un răspuns înapoi.

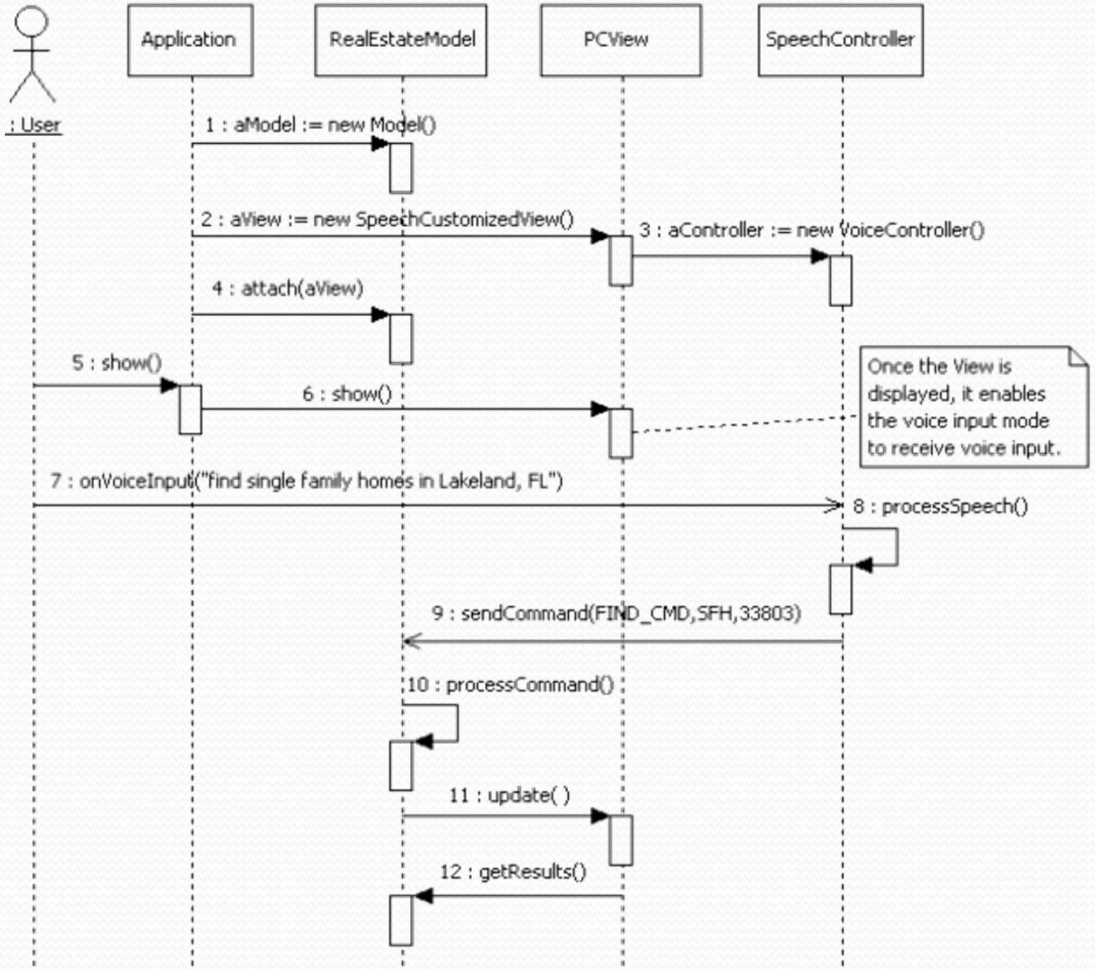


Broker

Provine din șablonul Client-Server, doar că Broker-ul mediază între ele și astfel un client poate să se conecteze la mai multe servere.

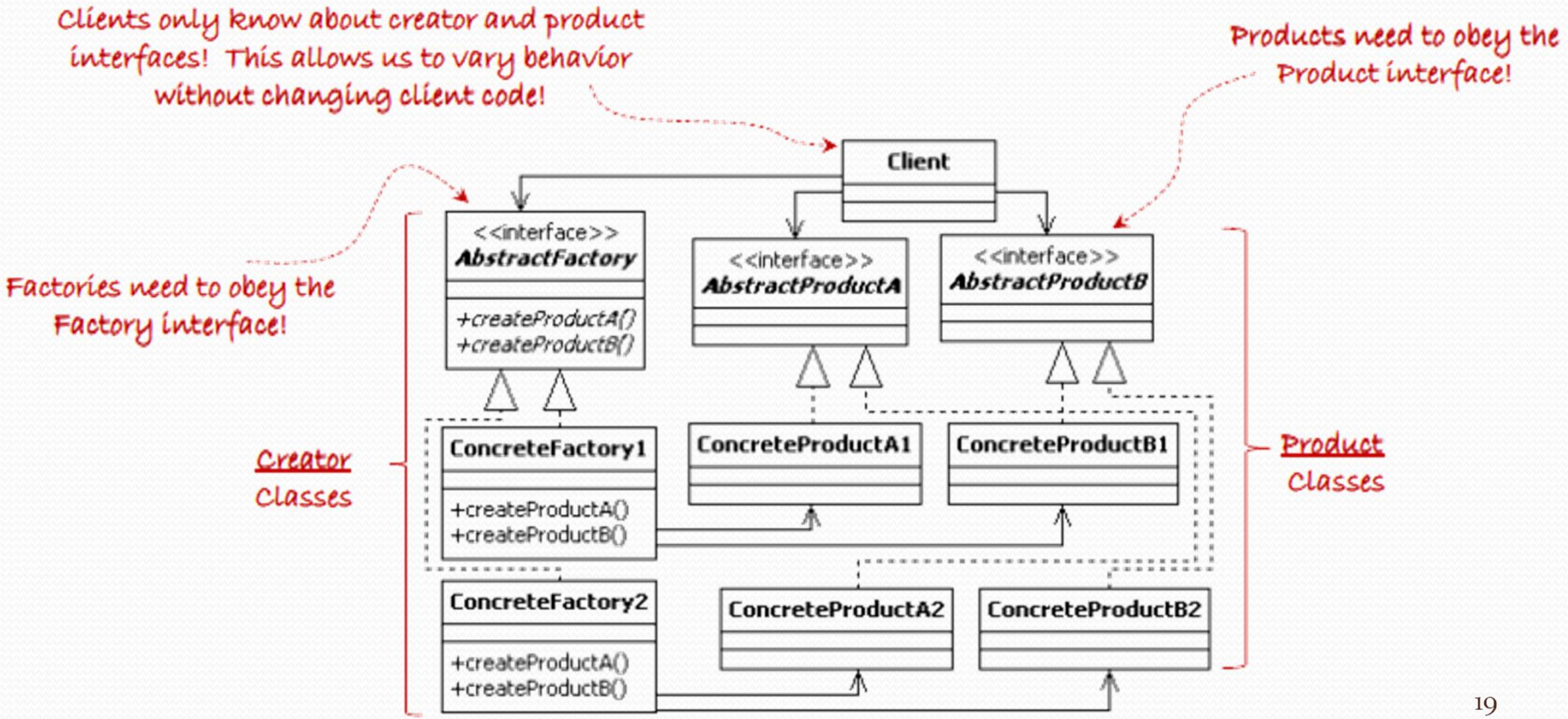


Model-View-Controller

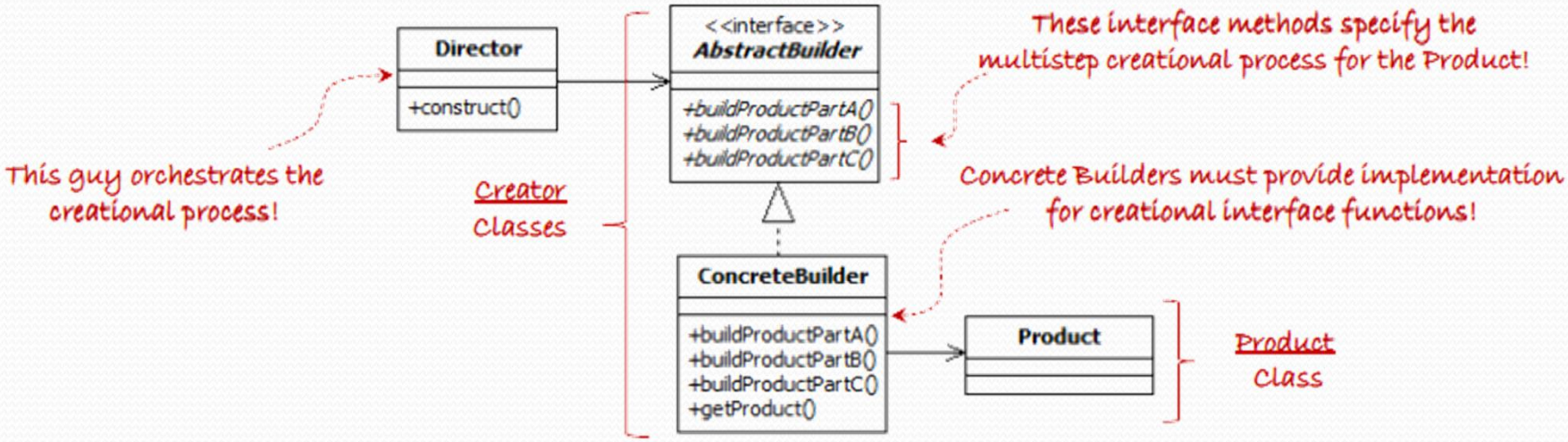


Abstract Factory

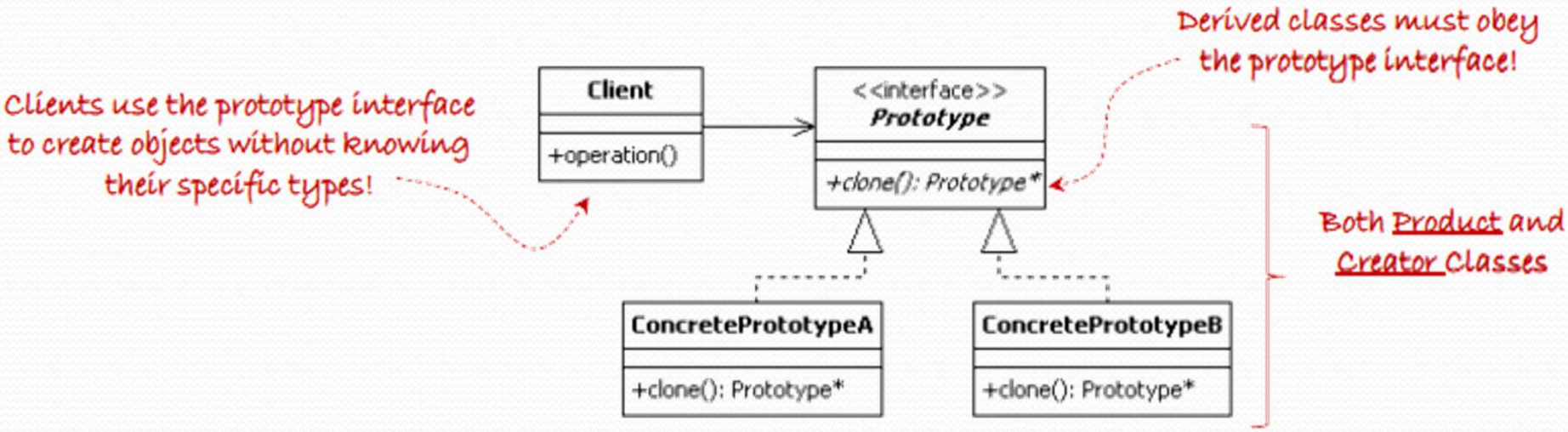
Clientul se basează pe interfața Fabricii Abstracte, lucrând cu Produse abstracte ale sale. La execuție se vor utiliza clase și obiecte (produce) derivate din cea abstractă.



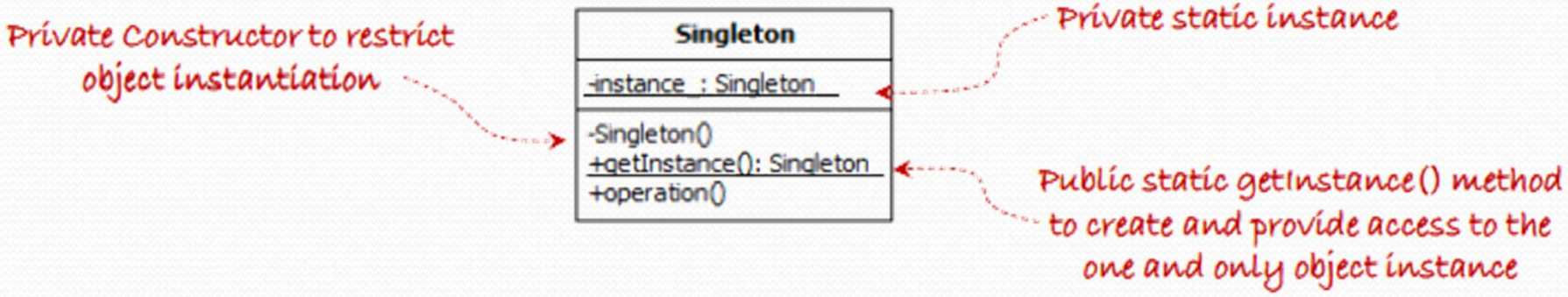
Builder


Director
+construct()
<<interface>>
AbstractBuilder
+buildProductPartA()
+buildProductPartB()
+buildProductPartC()
ConcreteBuilder
+buildProductPartA()
+buildProductPartB()
+buildProductPartC()
+getProduct()
Product
This guy orchestrates the creational process!
Creator Classes
These interface methods specify the multistep creational process for the Product!
Concrete Builders must provide implementation for creational interface functions!
Product Class

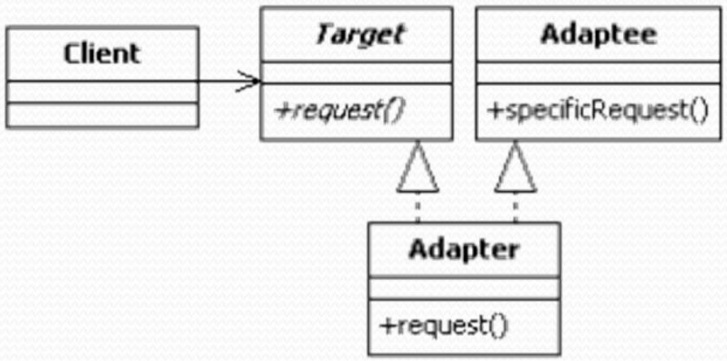
Prototype

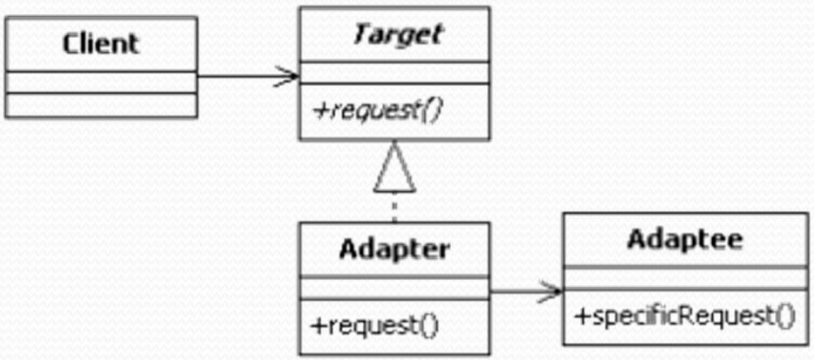

Clients use the prototype interface to create objects without knowing their specific types!
Client
+operation()
<<interface>>
Prototype
+clone(): Prototype*
Derived classes must obey the prototype interface!
ConcretePrototypeA
+clone(): Prototype*
ConcretePrototypeB
+clone(): Prototype*
Both Product and Creator Classes

Singleton

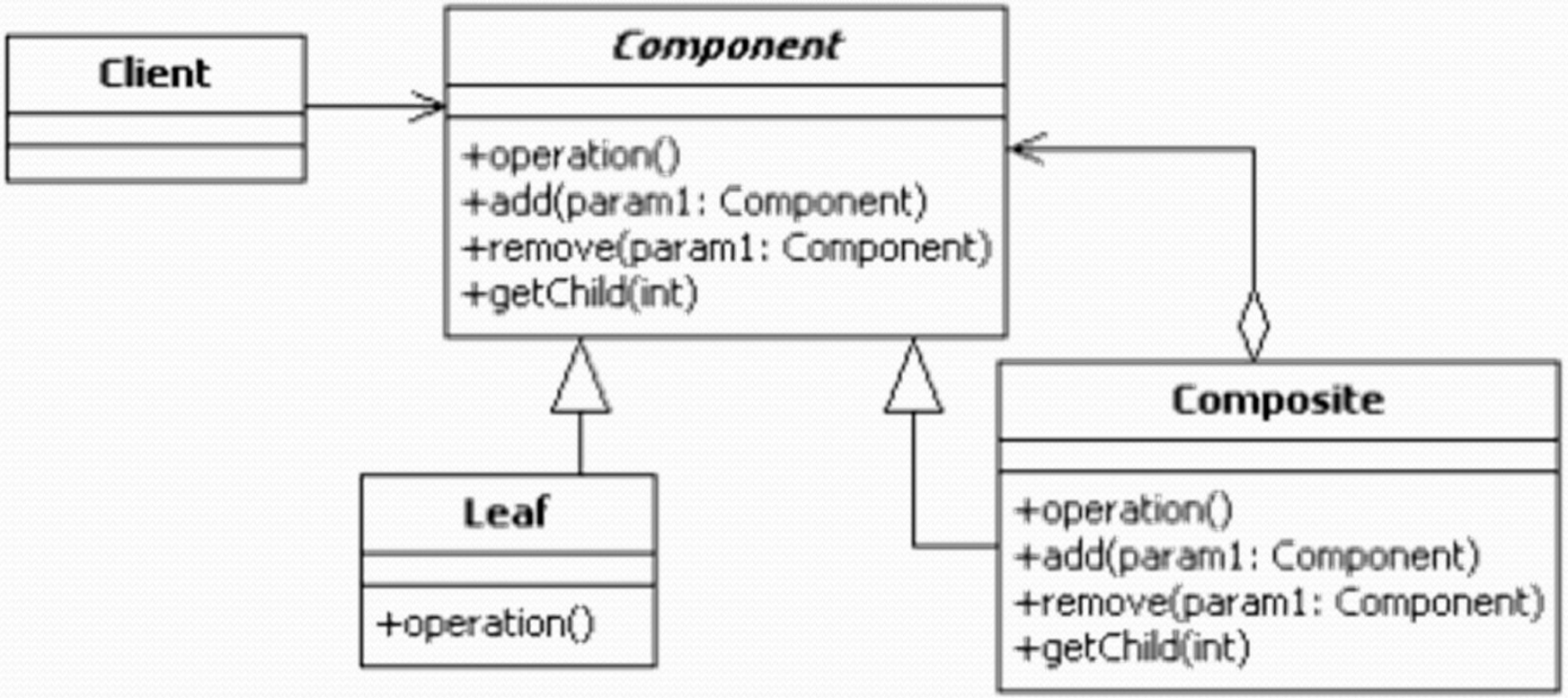

Private Constructor to restrict object instantiation
Singleton
-instance_ : Singleton
-Singleton()
+getInstance(): Singleton
+operation()
Private static instance
Public static getInstance() method to create and provide access to the one and only object instance

Adapter

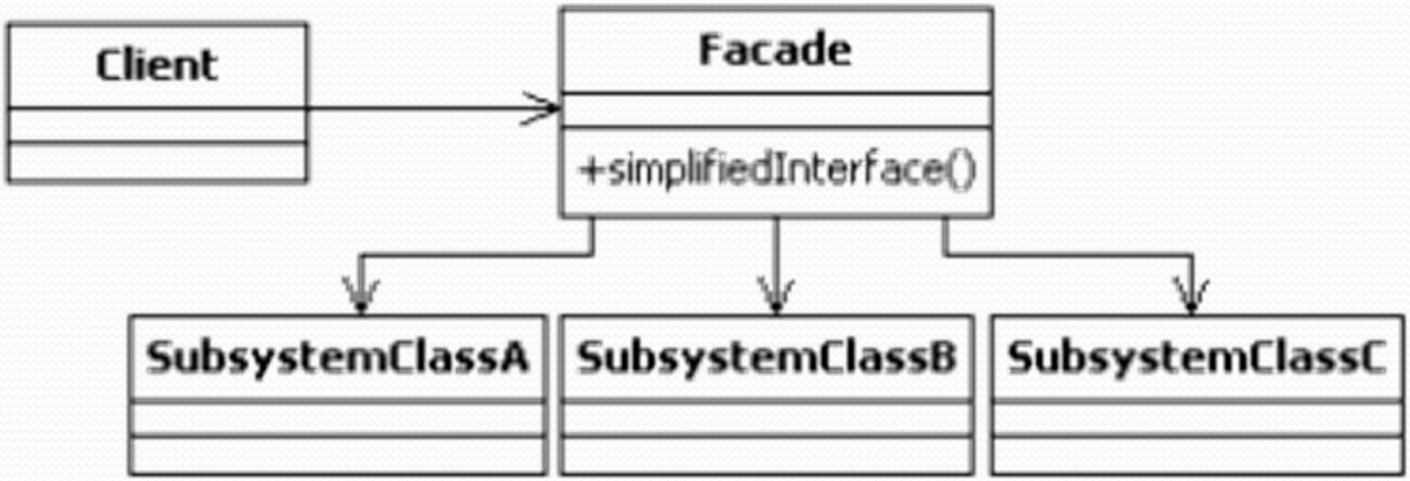

Client
Target
+request()
Adaptee
+specificRequest()
Adapter
+request()


Client
Target
+request()
Adapter
+request()
Adaptee
+specificRequest()

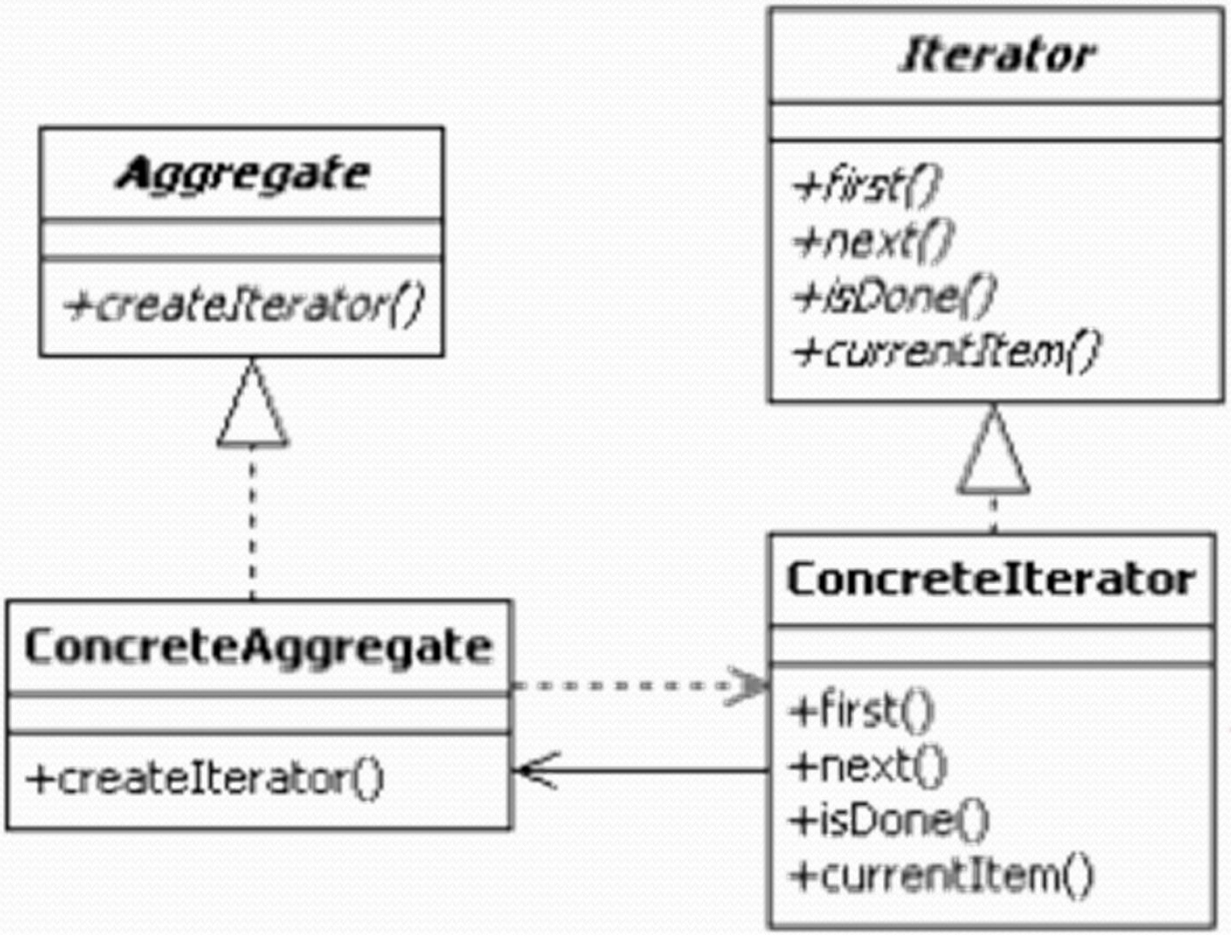
Composite


Client
Component
+operation()
+add(param1: Component)
+remove(param1: Component)
+getChild(int)
Leaf
+operation()
Composite
+operation()
+add(param1: Component)
+remove(param1: Component)
+getChild(int)

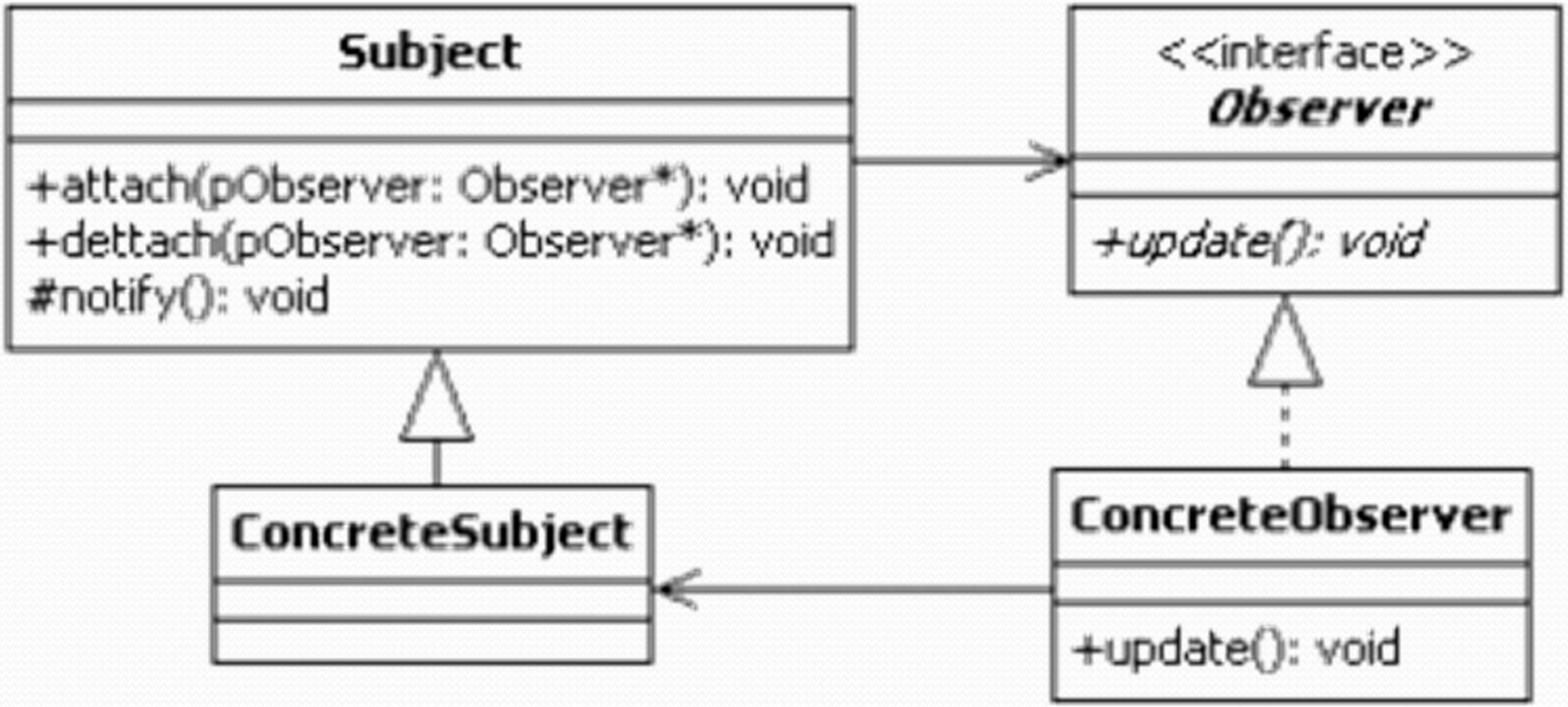
Facade


Client
Facade
+simplifiedInterface()
SubsystemClassA
SubsystemClassB
SubsystemClassC

Iterator


Iterator
+first()
+next()
+isDone()
+currentItem()
Aggregate
+createIterator()
ConcreteIterator
+first()
+next()
+isDone()
+currentItem()
ConcreteAggregate
+createIterator()

Observer


Subject
+attach(pObserver: Observer*): void
+dettach(pObserver: Observer*): void
#notify(): void
<<interface>>
Observer
+update(): void
ConcreteSubject
ConcreteObserver
+update(): void