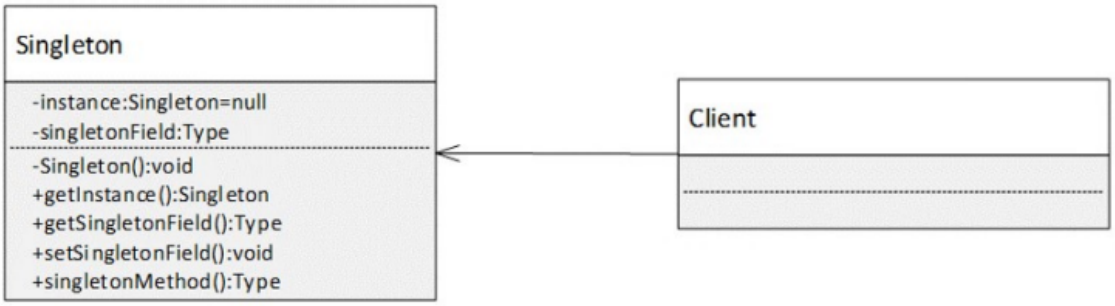
Creaționale	Structurale	Comportamentale
• Factory Method • **Abstract Factory** • Builder • Prototype • Singleton	• Adapter (clasă)/Adapter (obiect) • **Bridge** • Composite • Decorator • Façade • Flyweight • Proxy	• ***Interpreter*** • Template • Chain of Responsibility • Command • ***Iterator*** • ***Mediator*** • Memento • Observer • State • Strategy • ***Visitor***

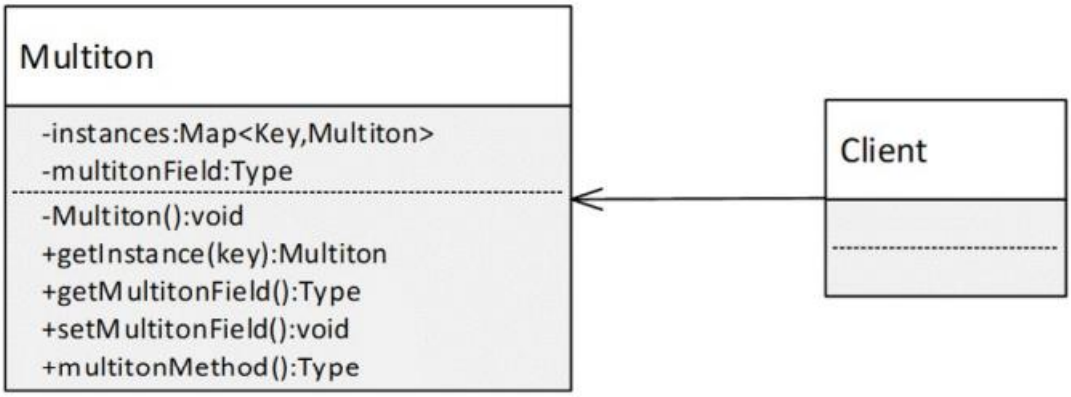
DP Creationale

Singleton



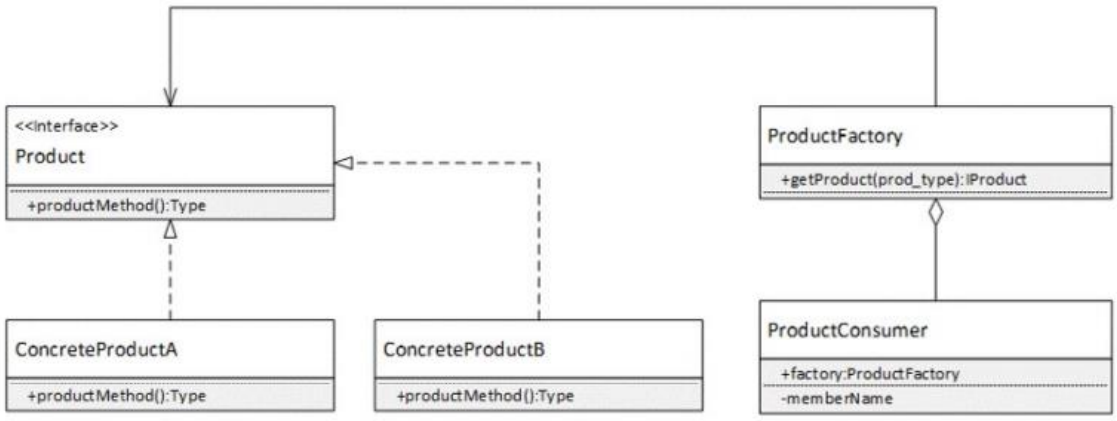
- **Singleton**
  - Definește mecanismul de creare și returnare a unei singure instanțe
  - Include și membrii specifici clasei
- **Client**
  - Utilizatorul obiectele de tip singleton

Multiton



- Multition
  - Definește mecanismul de creare și returnare a unei singure instanțe dintr-o anumită categorie
  - În plus, include și membrii specifici clasei
- **Client**
  - Utilizatorul obiectele generate

Simple Factory



- **Product**
  - Definește interfața obiectelor create prin intermediul clasei de tip ProductFactory
- **ConcreteProductA, ConcreteProductB**
  - Implementează interfața Product
- **ProductFactory**
  - Definește clasa de tip *factory* pentru crearea obiectelor de tip Product
- **ProductConsumer**
  - Utilizează clasa ProductFactory pentru crearea de obiecte de tip Product

Factory Method

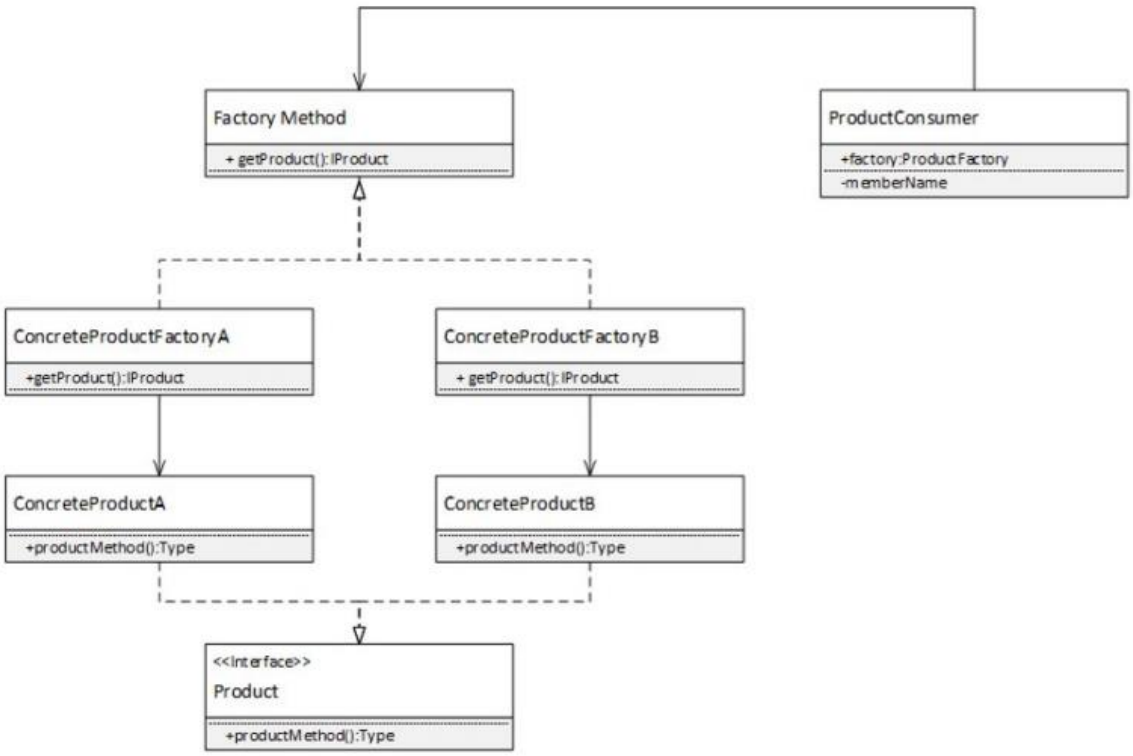
1. Crearea de diferite tipuri de caractere din joc

- in prezent exista 3 tipuri de caractere DISNEY, DC si MARVEL

- jocul fiind in dezvoltare exista riscul ca in viitor sa apara tipuri noi sau cele existente se vor modifica

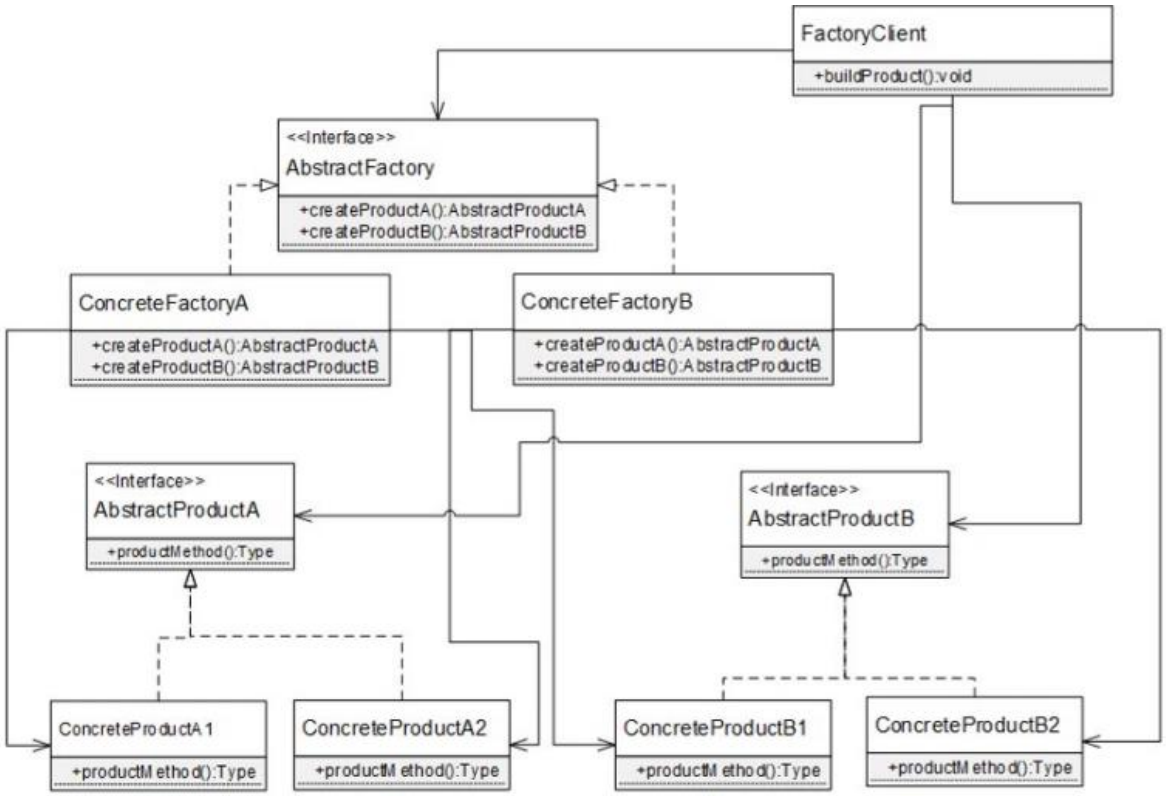
- jocul permite alegerea unei teme care influenteaza colectia de personaje; trebuie utilizata o metoda eficienta care sa pemita jocului

utilizarea oricarei teme fara a modifica major functionalitatea



- **Product**
  - Definește interfața obiectelor create prin intermediul metodei de tip *factory*
- **ConcreteProductA, ConcreteProductB**
  - Implementări ale interfeței Product
- **FactoryMethod**
  - Declară metoda de tip *factory* care creează obiecte de tip Product
    - Poate avea și o implementare implicită
- **ConcreteProductFactoryA, ConcreteProductFactoryB**
  - Redefinesc metoda de tip *factory* pentru crearea de obiecte concrete de tip Produs
- **ProductConsumer**
  - Utilizează clasele de tip FactoryMethod pentru crearea de obiecte de tip Product

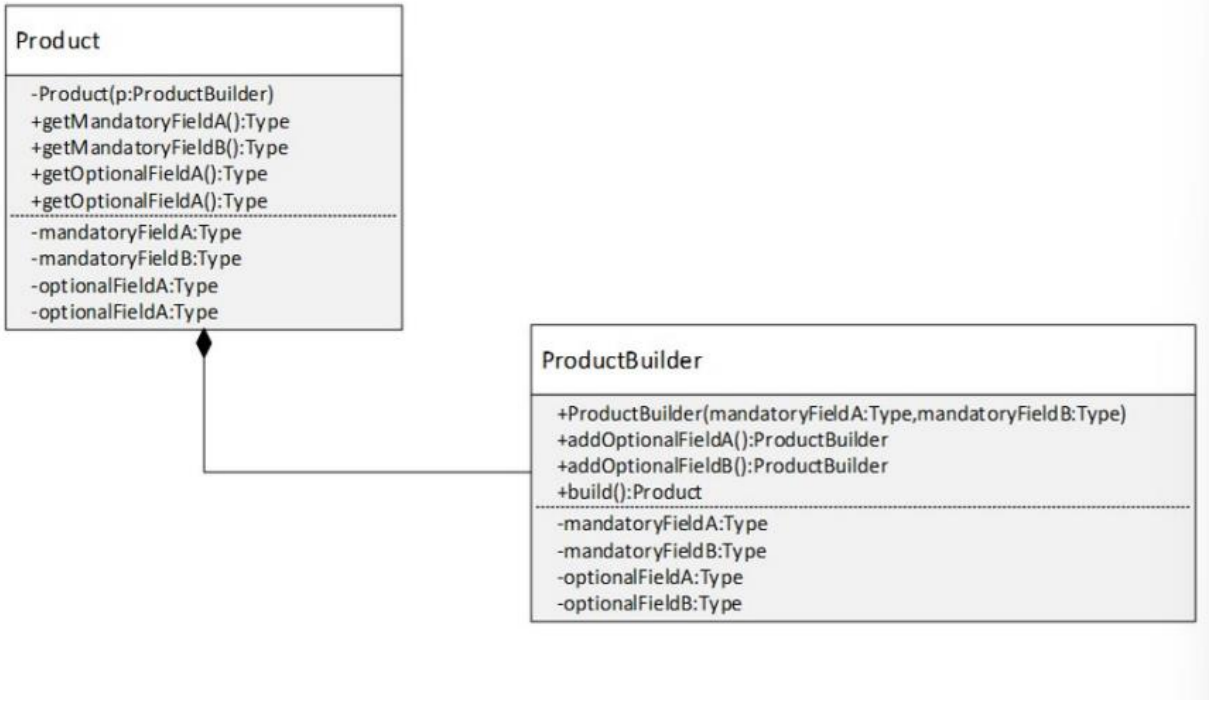
Abstract Factory



Componente

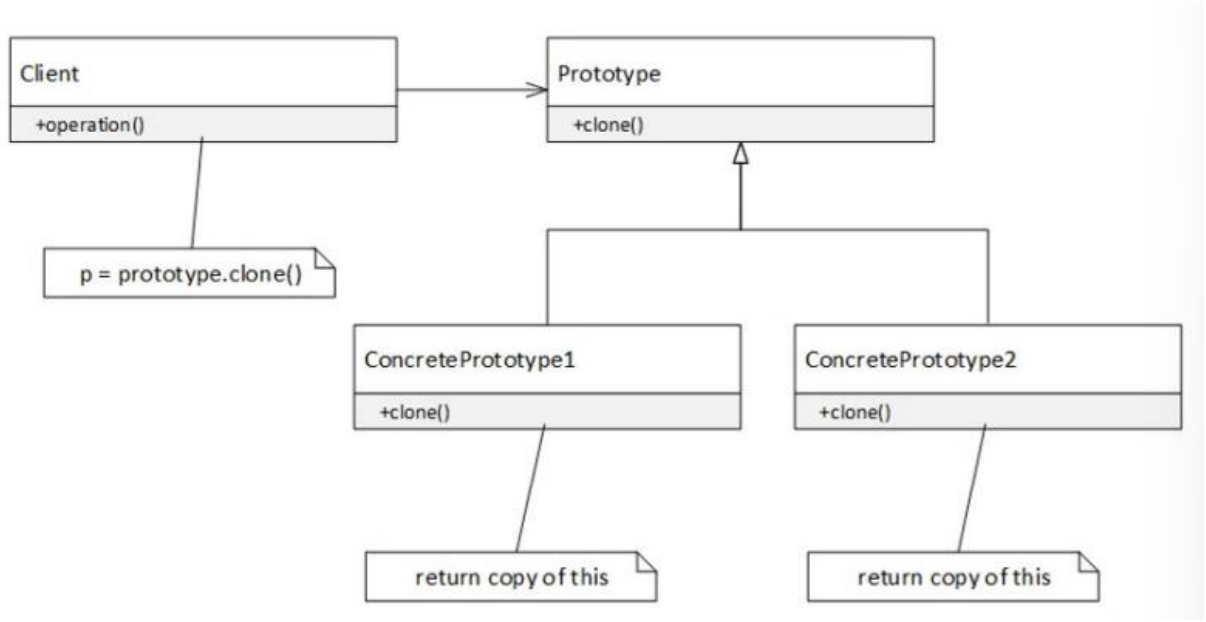
- **AbstractFactory**
  - Declară o interfață pentru crearea de produse abstracte
- **ConcreteFactoryA, ConcreteFactoryB**
  - Implementează operațiile pentru crearea obiectelor concrete
- **AbstractProductA, AbstractProductB**
  - Declară o interfață de tipul produselor
- **ProductA1, ProductA2, ProductB1, ProductB2**
  - Definesc clase concrete pentru produse ce vor fi create prin clasele corespunzătoare
- **FactoryClient**
  - Utilizează interfețele declarate de AbstractFactory și AbstractProduct

Builder



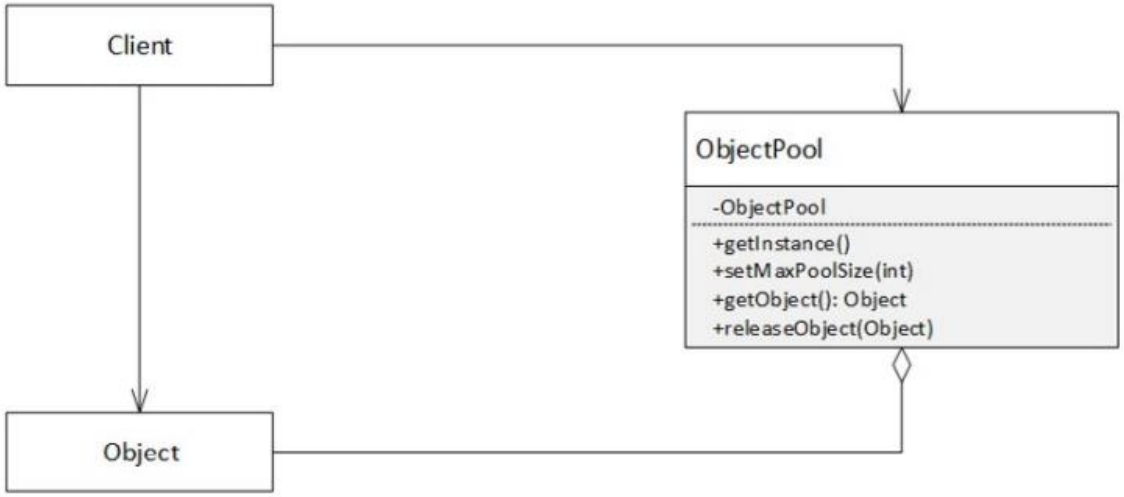
- **ProductBuilder**
  - Clasă pentru crearea obiectelor de tip Product
- **Product**
  - Obiecte create prin intermediul clasei ProductBuilder
- **Client**
  - Creează obiecte Product prin intermediul clasei Builder

Prototype



- **Prototype**
  - Declară o interfață pentru propria clonare
- **ConcretePrototypeA, ConcretePrototypeB**
  - Implementează o operație pentru propria clonare
- **Client**
  - Creează obiecte noi prin cereri de clonare către prototip
  - Doar primul prototip va fi creat prin constructor

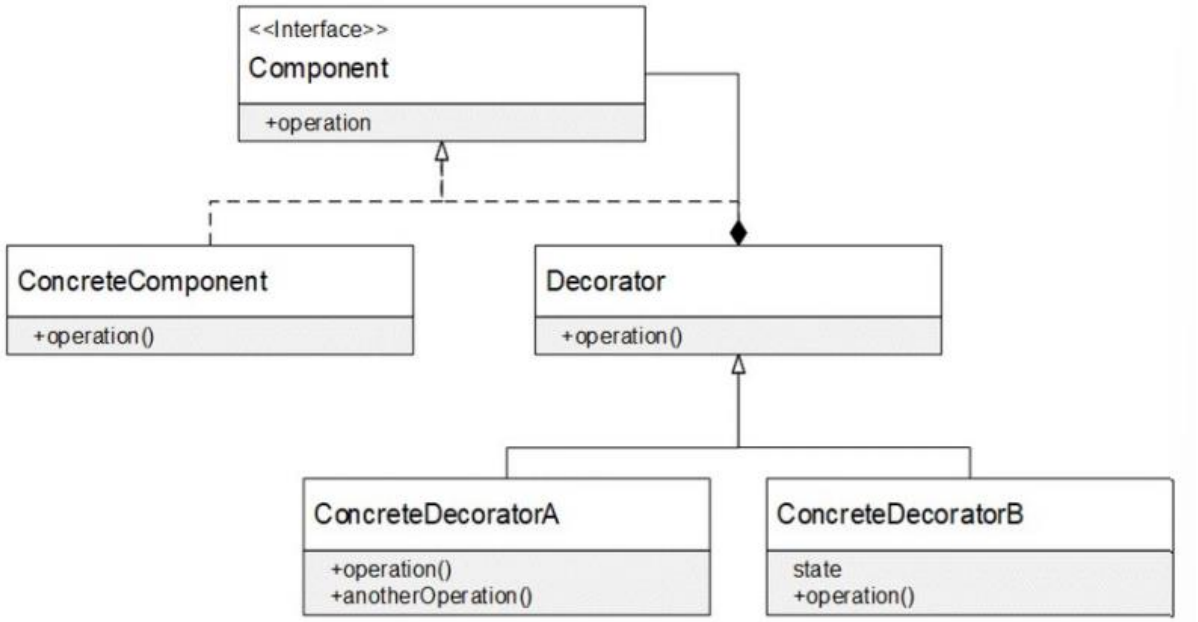
Object Pool



- **Object**
  - Clasa asociată obiectelor reutilizabile
- **ObjectPool**
  - Clasa care gestionează obiectele reutilizabile
- **Client**
  - Clasa care utilizează obiectele reutilizabile

DP Structurale

Decorator



- **Component**
  - Declară interfața obiectelor ce pot fi decorate cu noi funcții
- **ConcreteComponent**
  - Definește obiectele ce pot fi decorate
- **Decorator**
  - Gestionează o referință de tip Component către obiectul decorat
  - Metodele moștenite apelează implementările specifice din clasa obiectului referit
  - Poate defini o interfață comună claselor de tip decorator
- **ConcreteDecoratorA, ConcreteDecoratorB**
  - Clase concrete care adaugă funcții noi obiectului referit

Flyweight

1. Pentru a gestiona eficient (din punct de vedere al memoriei) scenariile in care jucatorii interactioneaza cu mai multe caractere controlate de calculator

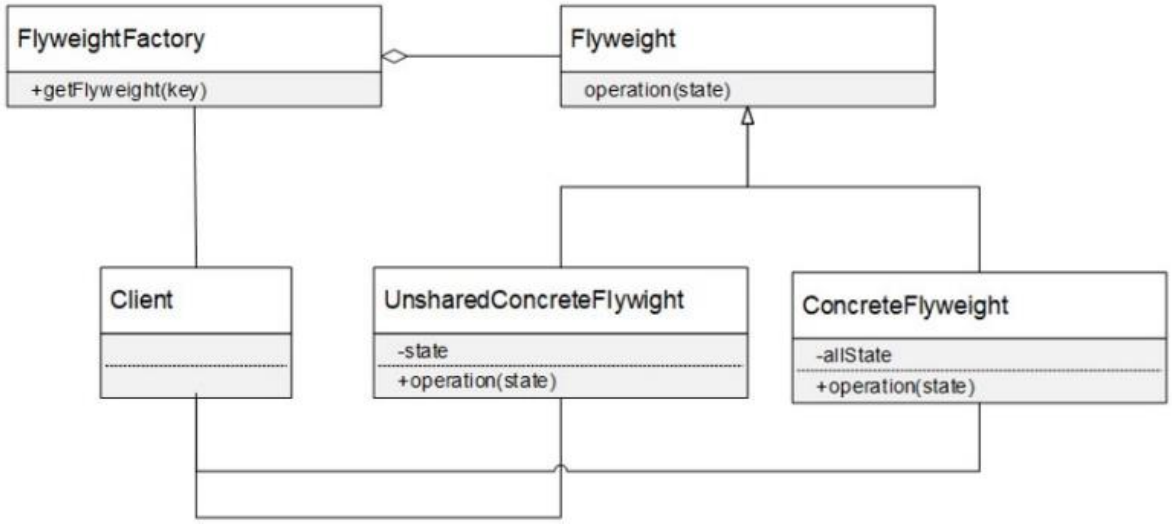
- colectia de caractere coordonate de calculator contine elemente de acelasi fel (aceleasi modele 3D)

- spatiul ocupat de un model 3D este semnificativ

- generarea unui astfel de scenariu necesita un spatiu semnificativ de memorie pentru client

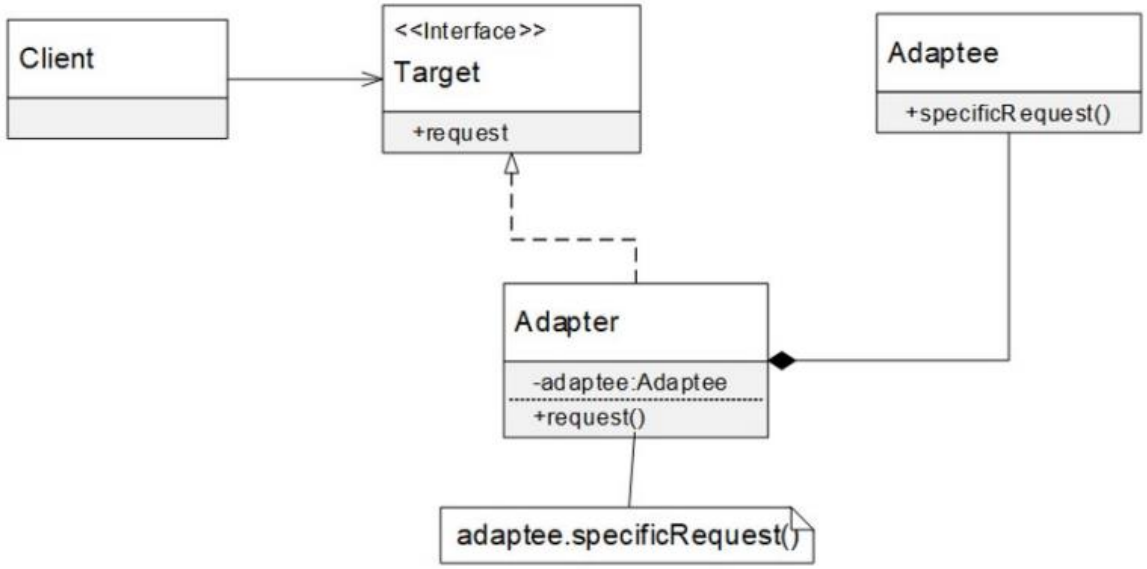
- modelele 3D sunt afisate pe ecran la coordonate diferite si pot avea texturi de culori diferite

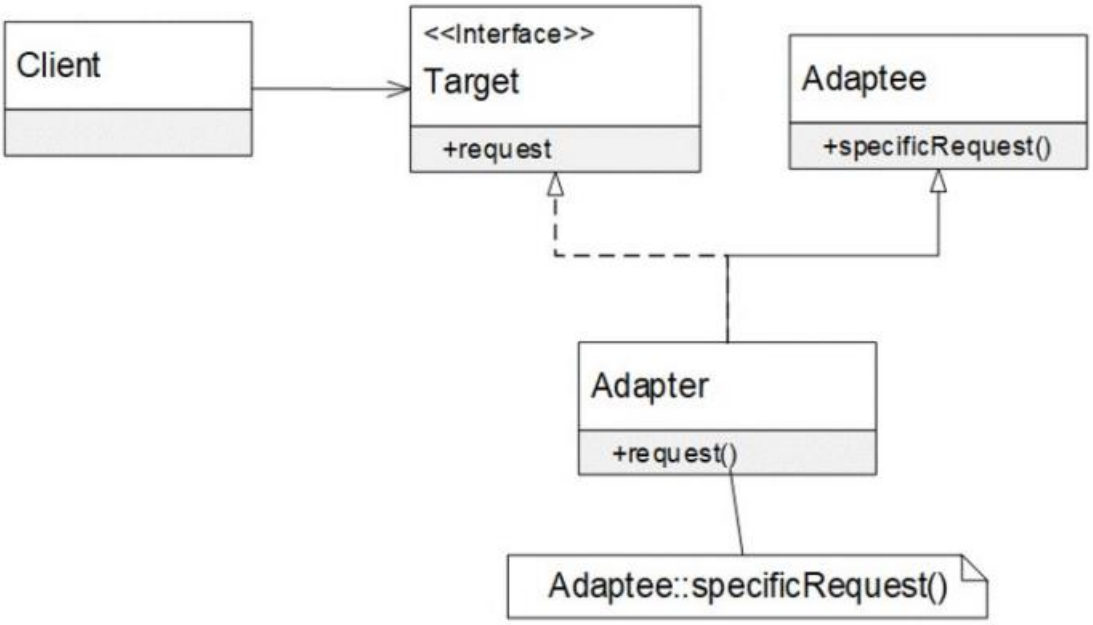
- trebuie gasita o solutie eficienta privind gestiunea acestor situatii



- **Flyweight**
  - Interfață care declară comportamentul prin intermediul căruia obiectele recepționează și reacționează la starea externă
- **ConcreteFlyweight**
  - Implementează interfața Flyweight
  - Stochează starea internă a obiectelor
  - Starea externă este gestionată prin intermediul metodelor din interfață
- **UnsharedConcreteFlyweight**
  - Clasa implementează interfața de tip Flyweight, dar nu permite partajarea stării
  - Pot exista clase derivate din aceasta care vor partaja starea
- **FlyweightFactory**
  - Construiește și gestionează obiecte de tip Flyweight
  - Menține o colecție de obiecte diferite, astfel încât acestea să fie create o singură dată
- **Client**
  - Utilizează obiectele de tip Flyweight
  - Gestionează referințele și starea externă a obiectelor

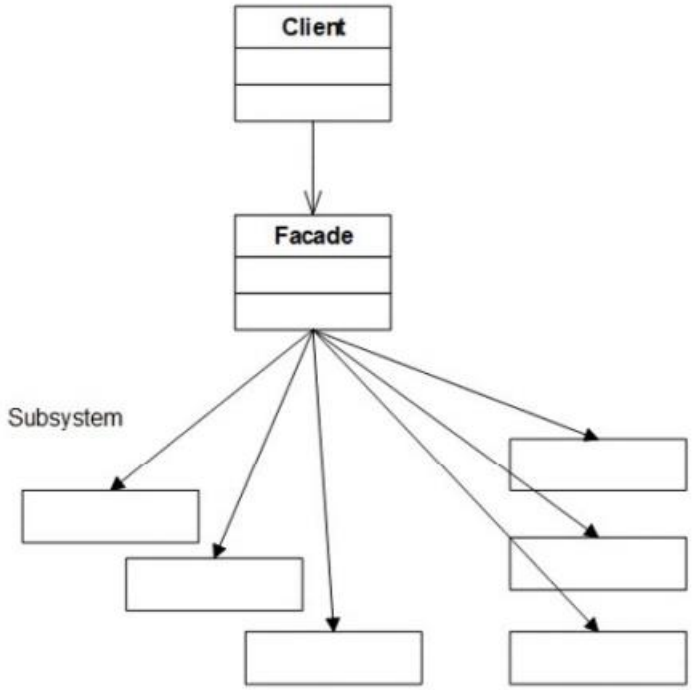
Adapter


Client
<<Interface>>
Target
+request
Adaptee
+specificRequest()
Adapter
-adaptee:Adaptee
+request()
adaptee.specificRequest()


Client
<<Interface>>
Target
+request
Adaptee
+specificRequest()
Adapter
+request()
Adaptee::specificRequest()

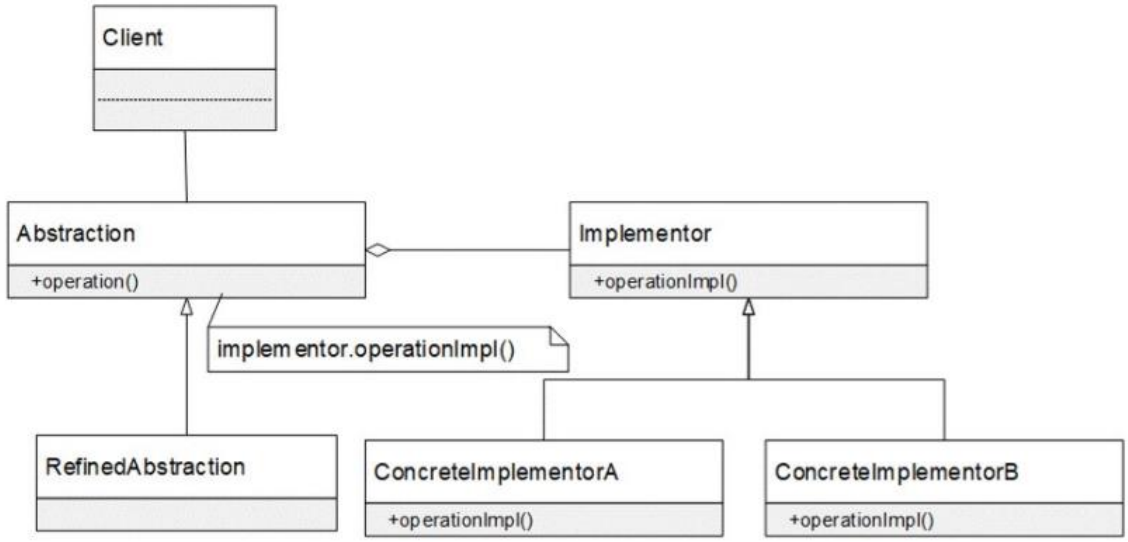
- **Adaptee**
  - Clasa existentă ce trebuie adaptată la o nouă interfață
- **Target**
  - Declară interfața specifică noului domeniu
- **Adaptor**
  - Adaptează interfața clasei existente la cea a clasei din noul context
- **Client**
  - Componenta care utilizează interfața specifică noului domeniu

Facade



- **Subsystem**
  - Structura existentă de componente ce pun la dispoziție diferite interfețe
- **Façade**
  - Declară o interfață simplificată pentru contextul existent
- **Client**
  - Componenta care utilizează interfața specifică noului domeniu

Bridge



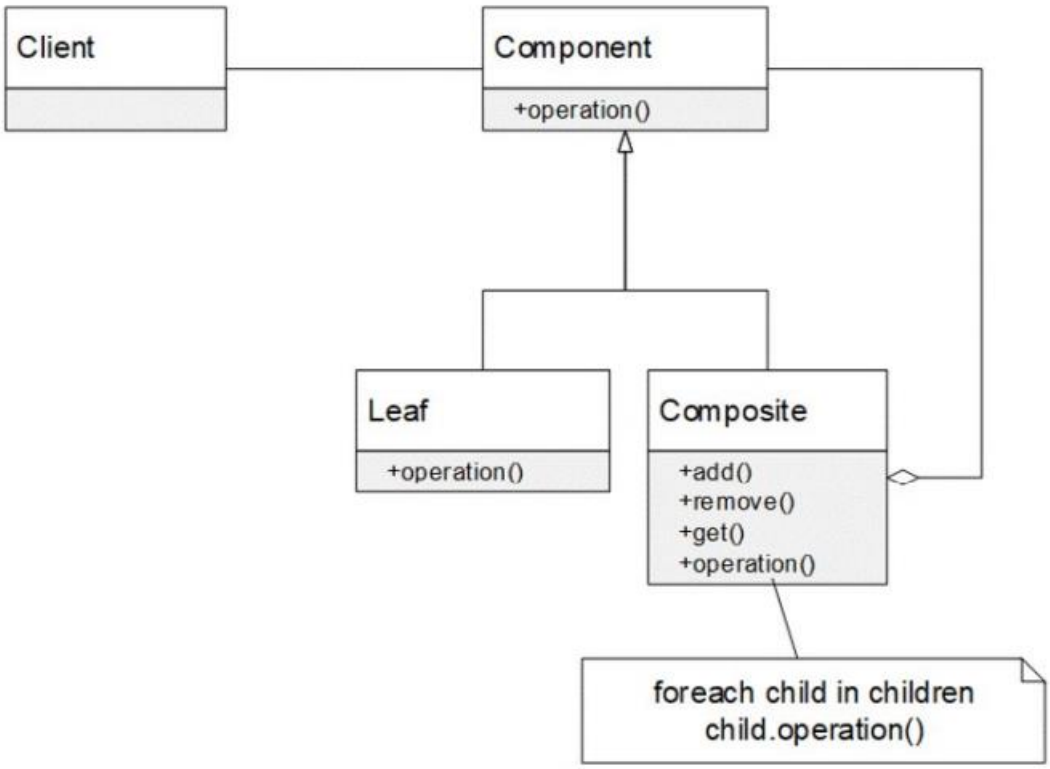
- **Abstraction**
  - Declară o interfață
  - Pune la dispoziție metode de nivel înalt
  - Include o referință a unui obiect de tip *Implementor*
- **RefinedAbstraction**
  - Extinde *Abstraction*
- **Implementor**
  - Declară interfața pentru clasele de implementare
  - Pune la dispoziție metode de nivel scăzut
  - Metodele din *Abstraction* invocă metodele din *Implementor*
- **ConcreteImplementorA, ConcreteImplementorA**
  - Implementează interfața *Implementor*
  - Include comportamentul specific

Composite

Pentru a gestiona o structura ierarhica de caractere (de ex o armata sau un grup de jucatori)

- sa se gaseasca o solutie eficienta care sa permita gruparea jucatorilor

- un grup poate contine caractere controlate de jucatori sau alte grupuri



- **Component**
  - Declară interfața obiectelor aflate in compoziție
- **Leaf**
  - Asociată nodurilor frunză din compoziție
- **Composite**
  - Componenta compusă, include noduri fiu
  - Implementează metode prin care sunt gestionate nodurile fiu
- **Client**
  - Manipulează obiectele din ierarhie

Proxy

2. Adaugarea unui filtru suplimentar pentru login

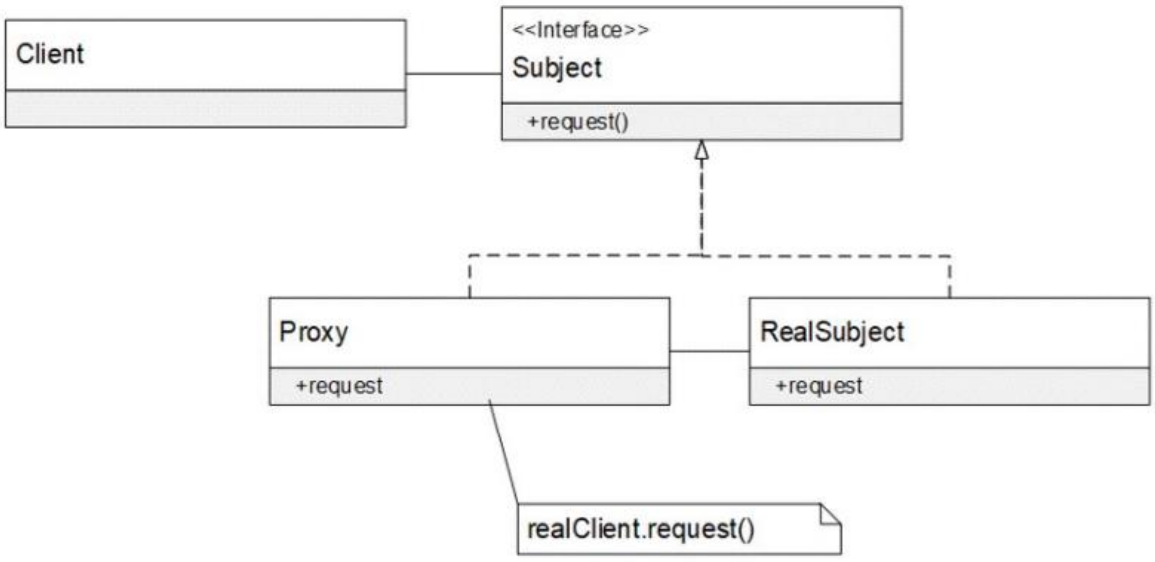
- in acest moment login-ul se face prin username si parola si in cazul in care parola este gresita utilizatorul poate incerca de mai multe ori

- exista atacuri asupra conturilor jucatorilor iar atacatorul ghiceste parola aferenta contului dupa mai multe incercari

- se doreste modificarea modulului de login prin limitarea numarului de incercari la maxim 3

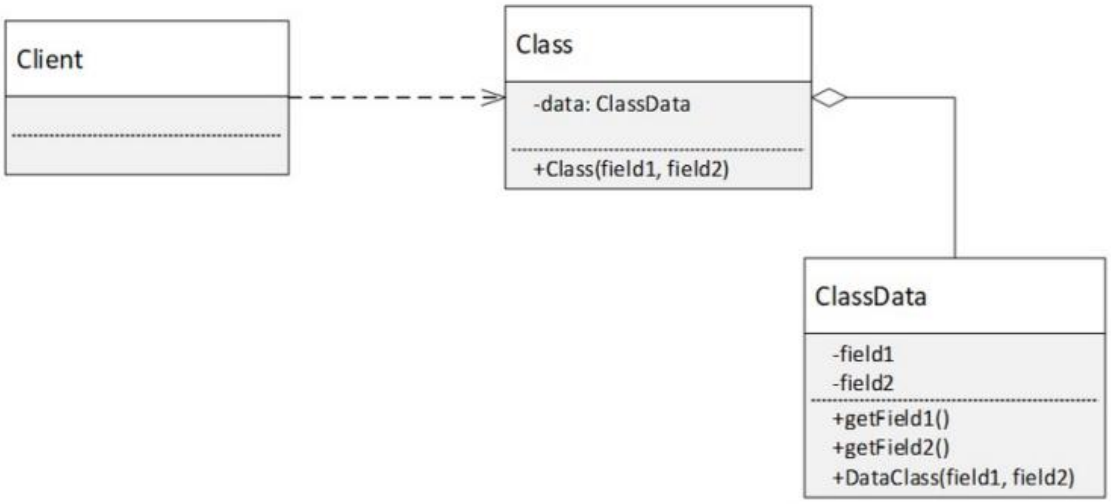
- modulul de login existent trebuie modificat fara a intrerupe jocul

- **Virtual Proxy**
  - Gestionează crearea și inițializarea unor obiecte
    - Procese costisitoare
  - Crearea în momentul accesului sau partajarea unei instanțe între mai mulți clienți
- **Remote Proxy**
  - Asigură o instanță virtuală locală pentru un obiect aflat la distanta (Java RMI, servicii Web etc.)
- **Protection Proxy**
  - Controlează accesul la anumite obiecte
  - Controlează accesul la metodele unui obiect
- **Smart References**
  - Gestionează automat referințele către un obiect și eliberează resursele atunci cînd acesta nu mai este utilizat
- **Cache Proxy**
  - Gestionează eficient apelurile costisitoare ale unui obiect concret
  - Îmbunătățirea performanțelor



- **Subject**
  - Declară interfața obiectului real la care se face conectarea
  - Interfața este implementată și de proxy
- **Proxy**
  - Implementează interfața obiectului real
  - Gestionează referința către obiectul real
  - Controlează accesul la obiectul real
- **RealSubject**
  - Obiectul real către gestionat prin intermediul proxy-ului
- **Client**
  - Utilizează serviciile puse la dispoziție de către RealSubject, prin intermediului Proxy-ului

Private class data



- **Class**
  - Clasa principală
- **ClassData**
  - Clasa asociată datelor din clasa Class
  - Datele sînt private
  - Accesibile prin getter-i
- **Client**

Aggregate

Aggregate este un pattern din domeniul programării orientate pe obiecte, care se referă la o colecție de obiecte care sunt tratate ca o unitate singulară pentru scopul manipulării datelor. Acest pattern este frecvent folosit în programarea bazată pe domenii (Domain-Driven Design - DDD), unde agregatele sunt folosite pentru a delimita limitele contextului în care un obiect sau grup de obiecte operează. Agregatul este format dintr-un obiect rădăcină (aggregate root), prin care se accesează toate celelalte obiecte din agregat, garantând consistența.

Pipes and filters

Design Pattern-ul **Pipes and Filters** este folosit în procesarea și manipularea seturilor de date printr-o serie de pași secvențiali. Fiecare pas este reprezentat de un "filter" care procesează datele într-un mod specific, iar "pipes" sunt conexiunile care transportă datele între filtre. Acest pattern este ideal pentru aplicații care necesită o serie de transformări sau procesări ale datelor, cum ar fi în sistemele de procesare a fluxurilor de date, audio/video encoding, sau chiar în procesarea interogărilor.

DP Comportamentale

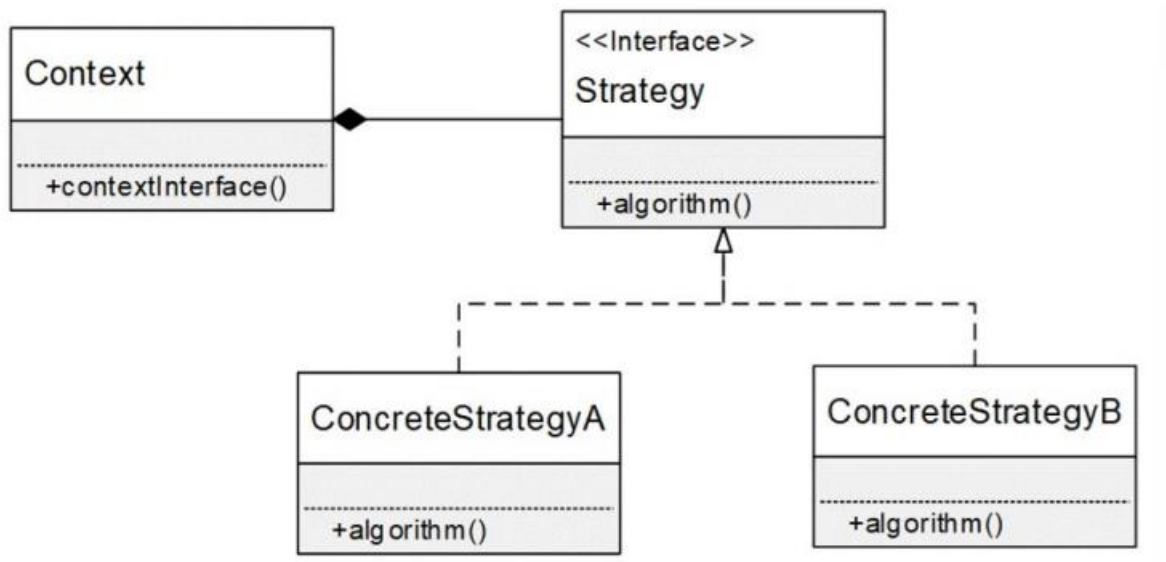
Strategy

3. In functie de implicarea in joc (timp petrecut, activitate in timpul jocului, etc) jucatorii primesc puncte bonus.

Aceasta strategie este stabilita de departamentul de marketing si se modifica in functie de alte campanii, perioada anului, etc.

Sa se gaseasca o solutie care sa permita:

- modificarea strategiei de acordare a bonusului la runtime fara a modifica implementarea jocului



Componente

- **Strategy**
  - Declară interfața pe care o implementează toți algoritmii
- **ConcreteStrategyA, ConcreteStrategyB**
  - Implementează algoritmul folosind interfața *Strategy*
- **Context**
  - Gestionează o referința de tip *Strategy*
  - Gestionează contextual în care au loc prelucrările

Observer

1. In timpul jocului clientul poate pierde conexiunea la server (din diferite motive). Cand acest lucru se intampla,

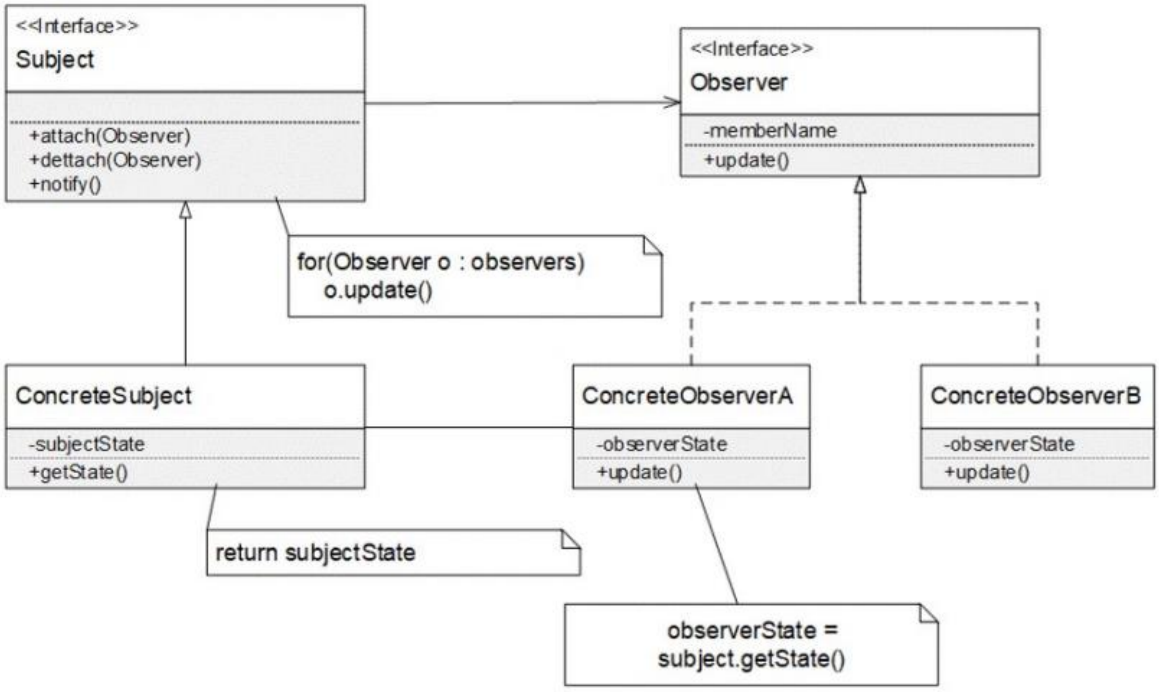
diferitele module din joc trebuie sa reactioneze in consecinta

- trebuie salvata instanta jocului pe masina clientului

- trebuie salvate atributele caracterului

- trebuie notificat clientul

- trebuie reincercata conexiunea



- **Subject**
  - Declară interfața obiectelor observabile
  - Include metode pentru adăugarea și eliminarea obiectelor de tip *Observer*
- **ConcreteSubject**
  - Gestionează starea unui obiect și notifică observatorii în momentul schimbării acesteia
- **Observer**
  - Declară interfața de actualizare a observatorilor, atunci cînd sunt notificați
- **ConcreteObserverA, ConcreteObserverB**
  - Implementează metodele care sînt executate în urma notificării
  - Gestionează starea obiectelor observator

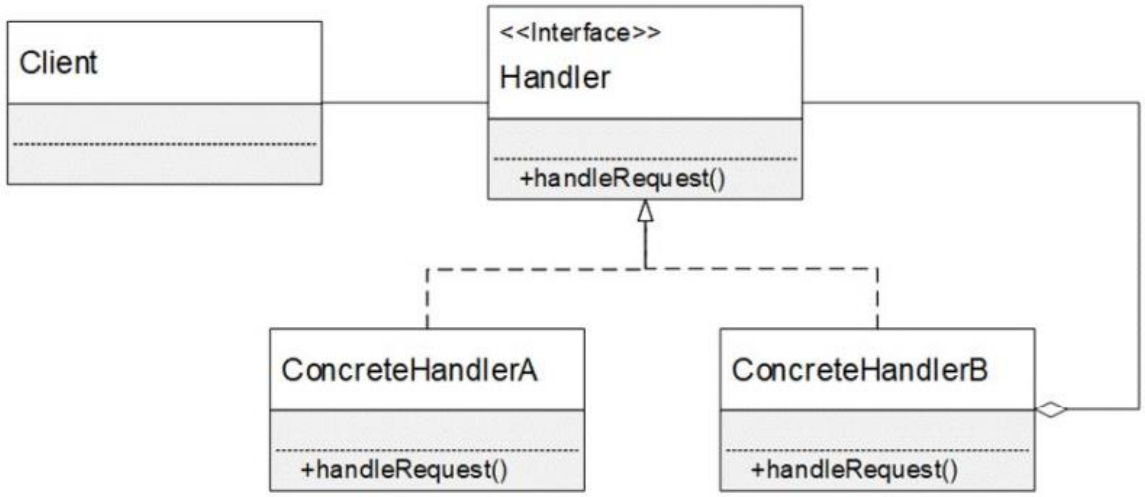
Chain of responsability

3. Jocul are un chat intern

- vrem sa filtram comentariile / mesajele care nu sunt potrivite

- mesajele destinate unui anumit player vor fi private

- difuzare mesaje destinate tuturor; destinatarul este @everyone



- **Handler**
  - Declară interfața pentru gestiunea cererilor care urmează să fie procesate
- **ConcreteHandlerA, ConcreteHandlerB**
  - Preia și procesează cererile adresate
  - Dacă nu poate procesa cererea, aceasta va fi transmisă următorului obiect din listă
- **Client**
  - Inițiază cererea către primul obiect din lista de obiecte

Command

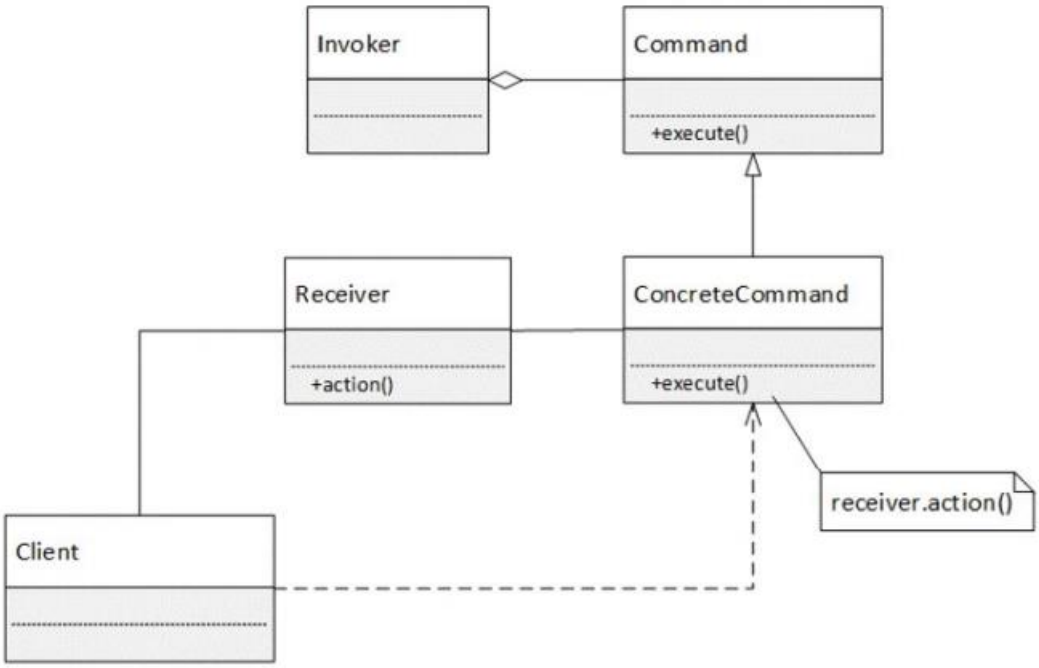
1. Clientul jocului trebuie sa efectueze sarcini in fundal fara a interfera cu derularea jocului (fara a-l bloca sau incetini)

- backup de date client

- primeste actualizari pentru forum

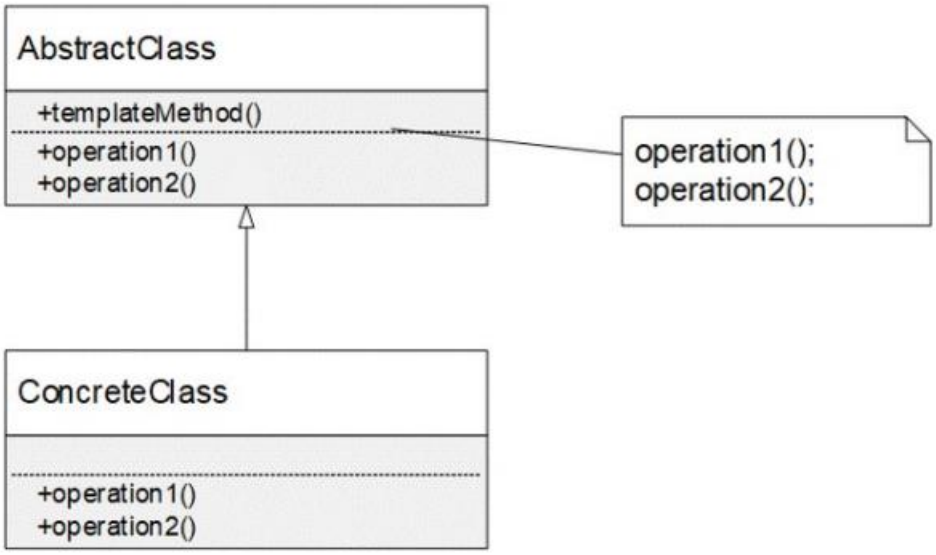
- actualizati modele 3d

- trebuie gasita o solutie prin care aceste task-uri sa fie executate fara a afecta executia jocului



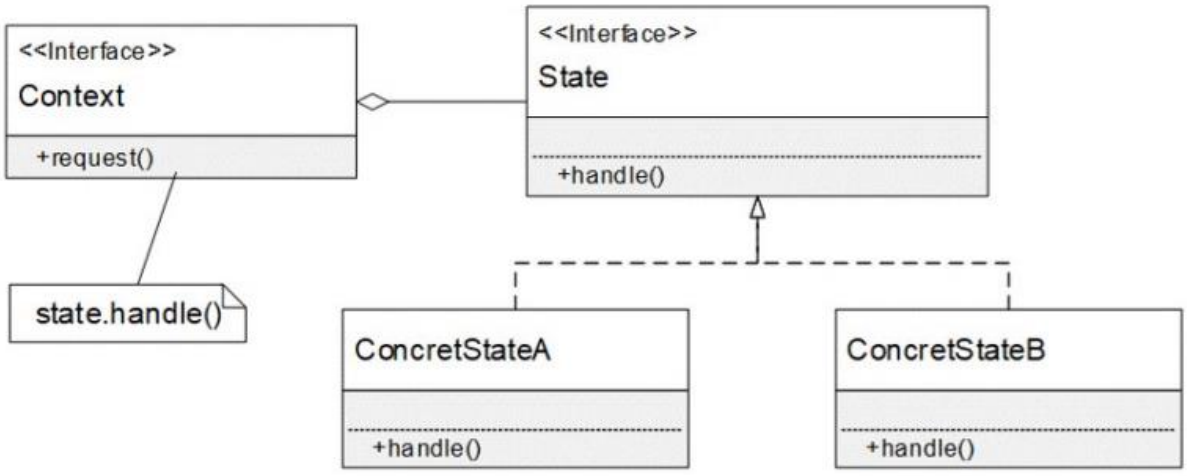
- **Command**
  - Declară o interfață pentru execuția unei operații
- **Receiver**
  - Obiectul receptor
  - Include logica sau datele necesare unei anumite comenzi
  - Cunoaște modul de execuție a unei operații asociate unei cereri
- **ConcreteCommand**
  - Definește o legătură între obiectul receptor și o acțiune
  - Implementează execuția comenzii prin apelul operației corespunzătoare din receptor
- **Invoker**
  - Gestionează obiectele de tip *Command*
  - Invocă o comandă pentru execuția unei acțiuni
- **Client**
  - Creează o comandă concretă și îi asociază un receptor
  - Comenzile se transmit obiectelor de tip *Invoker*

Template -> dont call us, we'll call you



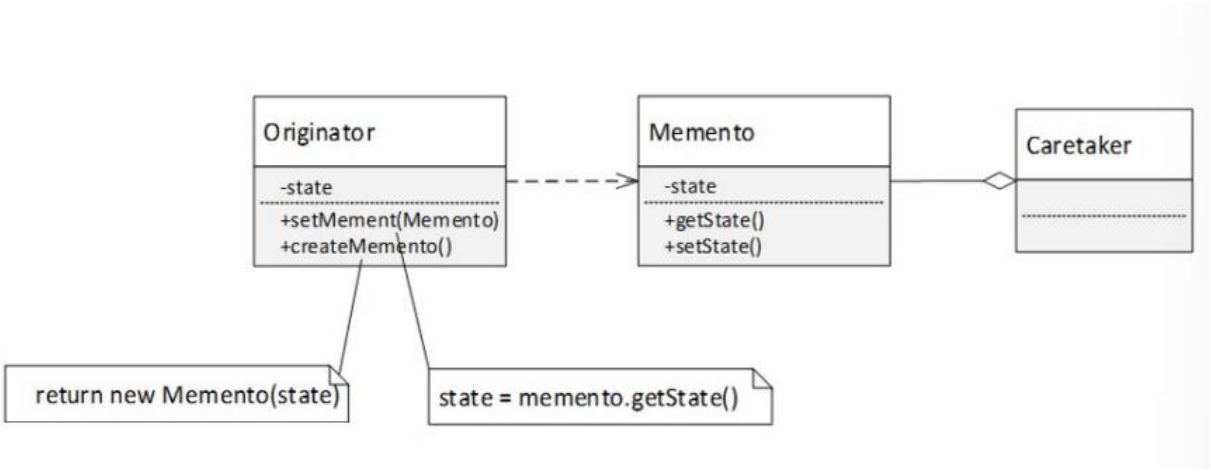
- **AbstractClass**
  - Definește metodele abstracte asociate pașilor unui algoritm
  - Metodele vor fi implementate în clasele derivate
  - Implementează o metodă template care definește structura unui algoritm
    - Apelează metodele asociate pașilor algoritmului
- **ConcreteClass**
  - Implementează metodele asociate pașilor algoritmului
  - Permite rafinarea anumitor pași ai algoritmului

State



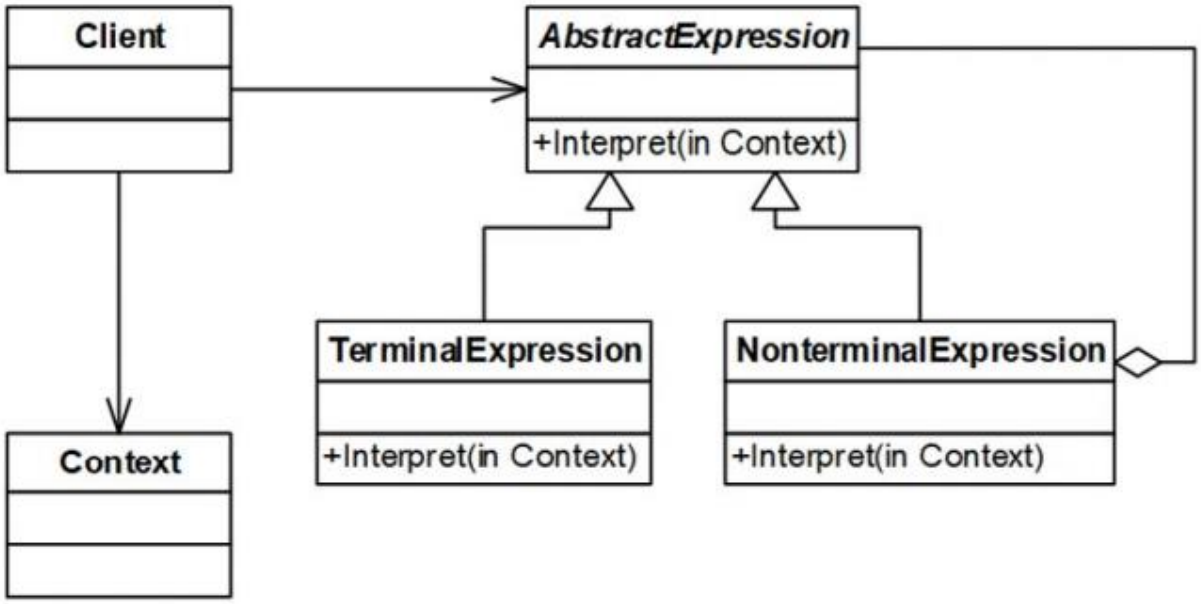
- **Context**
  - Definește interfața utilizată de clienți
  - Gestionează starea curentă
- **State**
  - Definește o interfață pentru încapsularea comportamentului asociat unei anumite stări a contextului
- **ConcreteStateA, ConcreteStateB**
  - Implementează comportamentul asociat unei anumite stări a contextului

Memento



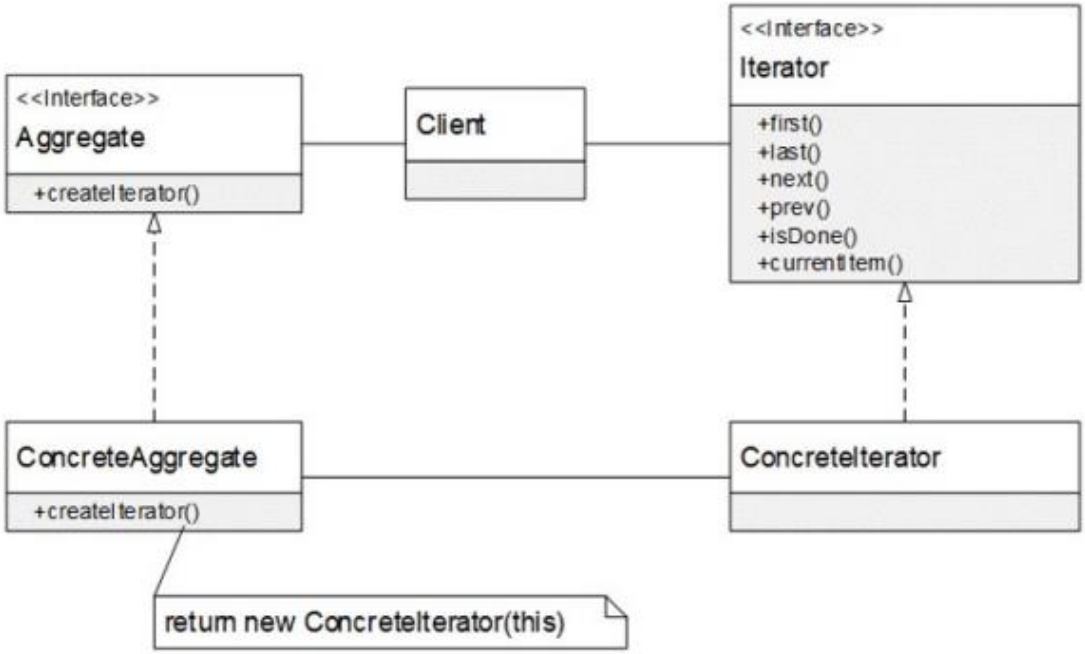
- **Memento**
  - Gestionează starea internă a unui obiect de tip *Originator*
  - Este gestionat de *Caretaker*
- **Originator**
  - Creează un obiect de tip *Memento* pentru salvarea stării interne
  - Utilizează *Memento* pentru restaurarea stării sale interne
- **Caretaker**
  - Gestionează obiectul de tip *Memento*
  - Nu acționează asupra stării acestuia

Interpreter



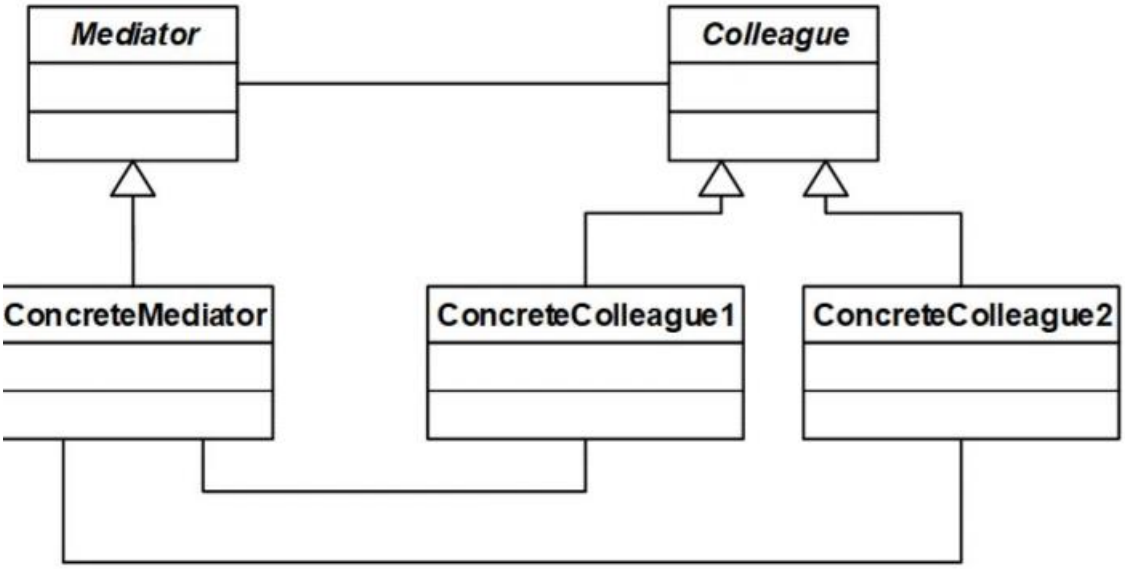
- **AbstractExpression**
  - Declară o interfață pentru executarea unei operații
- **TerminaExpression**
  - Implementează o operație asociată simbolurilor terminale din gramatică
- **NonterminalExpression**
  - Implementează o operație asociată simbolurilor non-terminale din gramatică
  - Instanțele sînt definite pentru orice regulă de compunere
- **Context**
  - Include informațiile globale utilizate de interpretor
- **Client**
  - Preia o expresie exprimată în limbajul în care este definită gramatica
  - Generează arborele de sintaxă asociat expresiei
  - Invocă operațiile de interpretare

Iterator



- **Iterator**
  - Definește o interfață pentru accesarea și traversarea elementelor unei colecții
- **ConcreteIterator**
  - Implementează interfața Iterator
  - Gestionează poziția curentă în cadrul parcurgerii
- **Aggregate**
  - Definește o interfață asociată colecției care poate fi parcursă prin intermediul unui Iterator
  - Definește o interfață pentru crearea obiectelor de tip Iterator
- **ConcreteAggregate**
  - Implementează interfața Aggregate
- **Client**
  - Utilizează obiectele de tip Iterator pentru accesarea elementelor colecțiilor

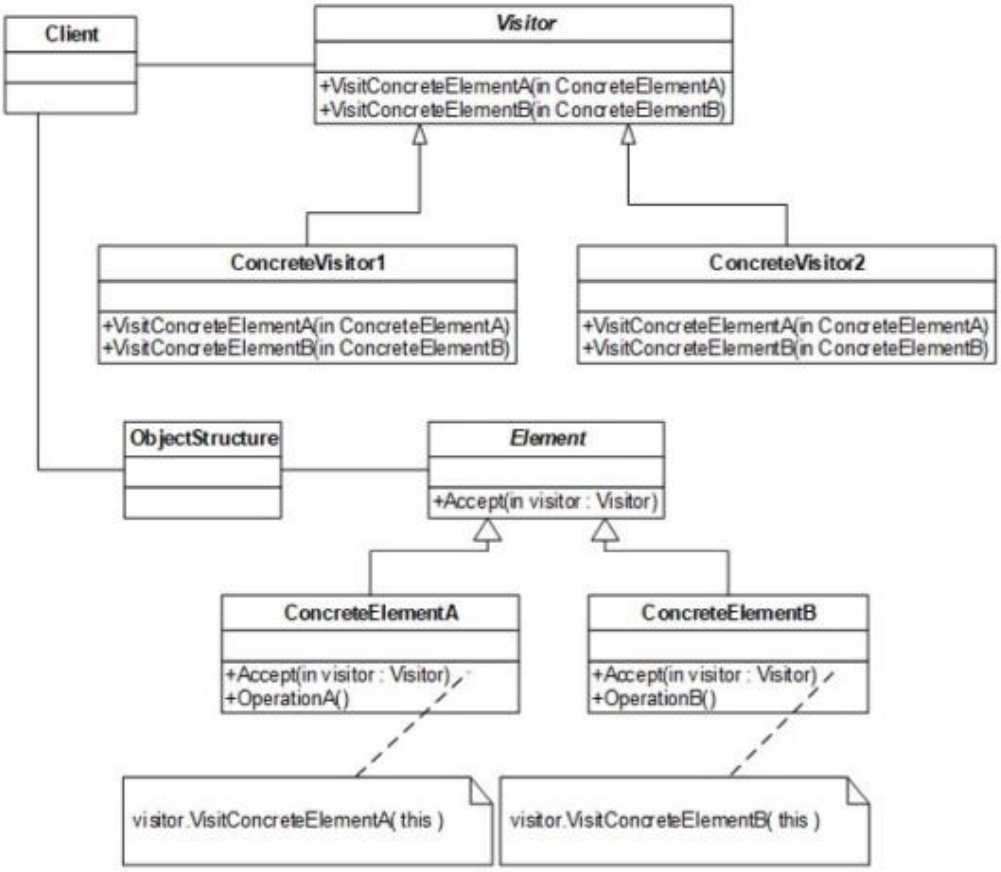
Mediator



- **Mediator**
  - Definește o interfață pentru comunicarea cu obiecte de tip *Colleague*
- **ConcreteMediator**
  - Implementează modalitatea de cooperarea dintre obiectele de tip *Colleague*
  - Gestionează obiectele de tip *Colleague*
- **Colleague**
  - Are acces la o clasă de tip *Mediator*
  - Interacționează cu colegii prin intermediul mediatorului
- **ConcreteCollegue1, ConcreteCollegue2**
  - Implementări concrete pentru *Colleague*

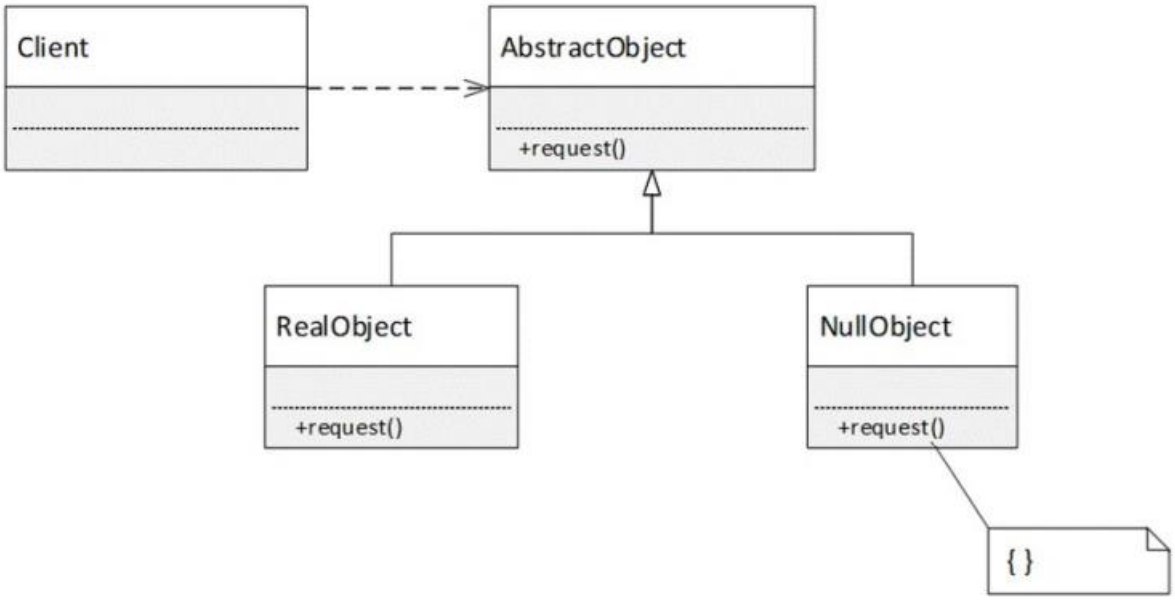
Visitor

Diagrama de clase



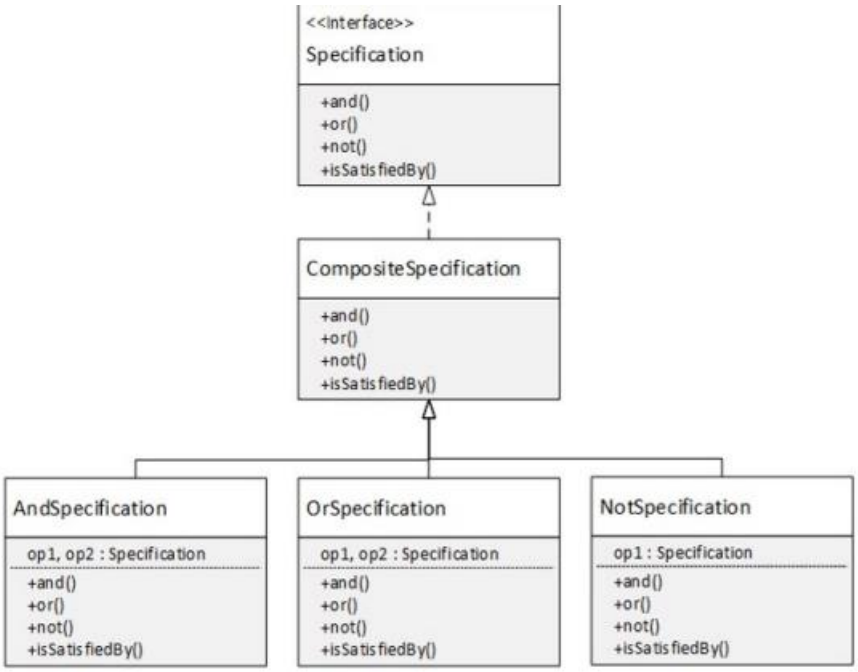
- **Visitor**
  - Declară o operație de vizitare pentru fiecare element concret din structură
- **ConcreteVisitor1, ConcreteVisitor2**
  - Implementează operațiile definite de *Visitor*
- **ObjectStructure**
  - Permite enumerarea elementelor
  - Definește o interfață care accesul vizitatorilor la elemente
- **Element**
  - Definește o operație de acceptare, asociată unui obiect de tip *Visitor*
- **ConcreteElementA, ConcreteElementB**
  - Implementează operația de acceptare
- **Client**

Null Object



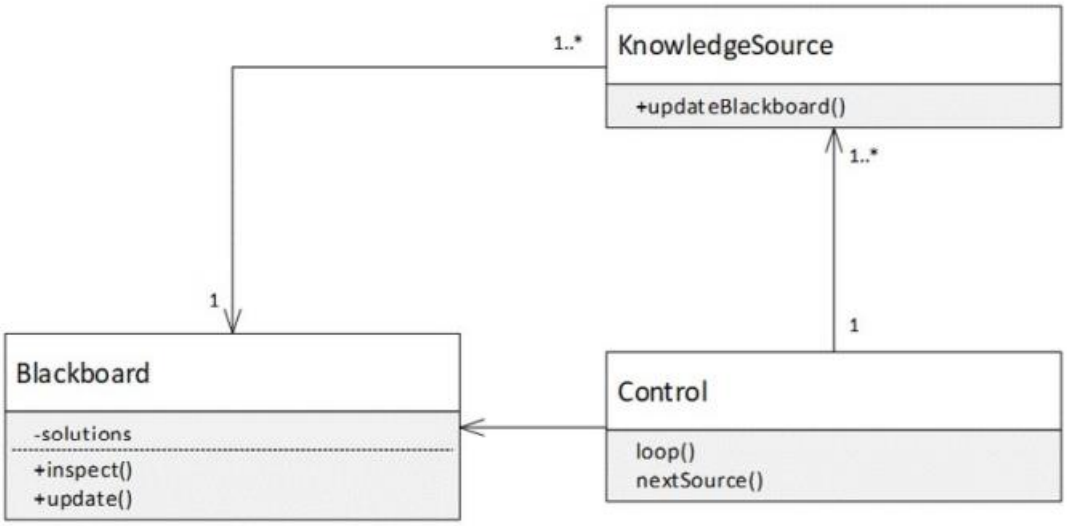
- **AbstractObject**
  - Declară interfața pentru client
  - Definește acțiunile care trebuie implementate
- **RealObject**
  - Implementează comportamentul normal, așteptat de client
- **NullObject**
  - Definește obiectele nule, care pot substitui obiectele reale
  - Acțiunea este implementată fără nici o operație
- **Client**

Specification



- **Specification**
  - Declară metodele pe care le au obiectele de tip specificații
  - Metoda principală este *isSpecifiedBy()*
  - Metode de compunere *and()*, *or()* și *not()*
- **CompositeSpecification**
  - Definește modalitatea de compunere a specificațiilor
  - Implementează metodele e compunere *and()*, *or()* și *not()*
- **AndSpecification**
  - Definește modalitatea de compunere a specificațiilor prin operatorul *and*
- **OrSpecification**
  - Definește modalitatea de compunere a specificațiilor prin operatorul *or*
- **NotSpecification**
  - Definește modalitatea de negare a unei specificații prin operatorul *not*

Blackboard



- **Blackboard**
  - Include obiectele din spațiul soluțiilor
- **KnowledgeSource**
  - Module specializate cu reprezentări specifice
- **Control**
  - Responsabil cu selectarea, configurarea și execuția modulelor