**Design Patterns**

**-**ajuta la rezolvarea altor probleme similare cu alte probleme pt care au fost identificate rezolvari

-foarte multe probleme se rezolva in acelasi mod sau folosind acelasi numar de pasi

**DP Creationale**

🡺Singleton

🡺Factory

🡺Factory Method

🡺Abstract Factory

🡺Builder

🡺Prototype

**DP Structurale**

🡺Adapter

🡺Bridge

🡺Composite

🡺Facade

🡺Decorator

🡺Flyweight

🡺Proxy

DP Creationale

🡺ajuta la initializarea si configurarea claselor si obiectelor

🡺separa crearea obiectelr de utilizarea lor concreta

🡺ofera flexibilitate:cine creeaza,cand creeaza,cum creeaza

**Singleton**

🡺un singur obiect de tipul clasei

🡺clasa este cea care se ocupa de faptul ca poate fi creat un singur obiect.Nu lasa acest lucru pentru utilizatori sau apelator

🡺instanta unica, o singura instanta

🡺constructorii privati

*Eager Initialization*

🡪initializarea instantei chiar daca nu este folosita

🡪in cazul in care clasa Singleton nu este folosita,instanta tot este creata

🡪aceasta varianta nu este eficienta

*Static Block Initialization*

🡪Aparea posibilitatea de captare a posibilelor exceptii generate de intializarea instante i statice

🡪instantiaza instanta chiar daca nu este folosita

*Lazy Initialization*

-->problema:multithreading

*Thread Safe Singleton*

🡪asigura faptul ca metoda nu o sa fie apelata de pe un alt fir de executie pana cand nu se termina metoda apelata deja pe un fir de executie

*Enum*

🡪nu permite Lazy Initialization

De ce folosim Singleton si nu o clasa Statica?

R:Un obiect Singleton poate fi transmit ca parametru unei functii, dar o clasa statica nu poate fi trimisa ca parametru

Utilizari:

🡪deschiderea unei singure instante ale aplicatiei

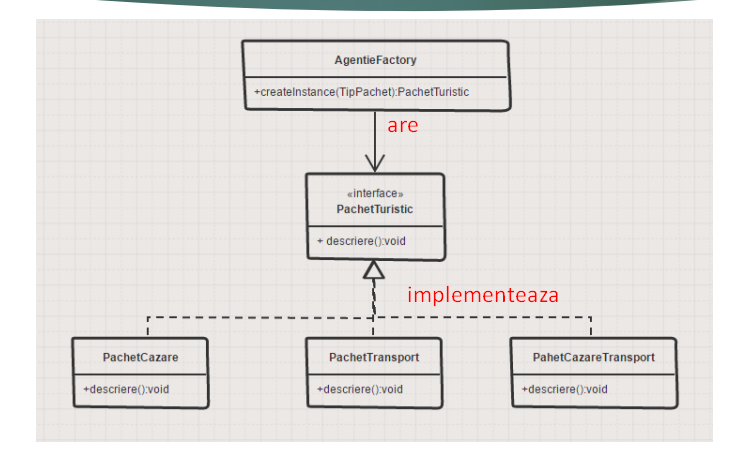
🡪conexiune unica la baza de date

🡪SharedPreferences in Android

***Factory***

🡪creeaza obiecte concrete din acceasi familie fara sa stie exact tipul concret al obiectului

🡪familie de obiecte



Utilizari

🡪creare VIEW-uri pt GUI

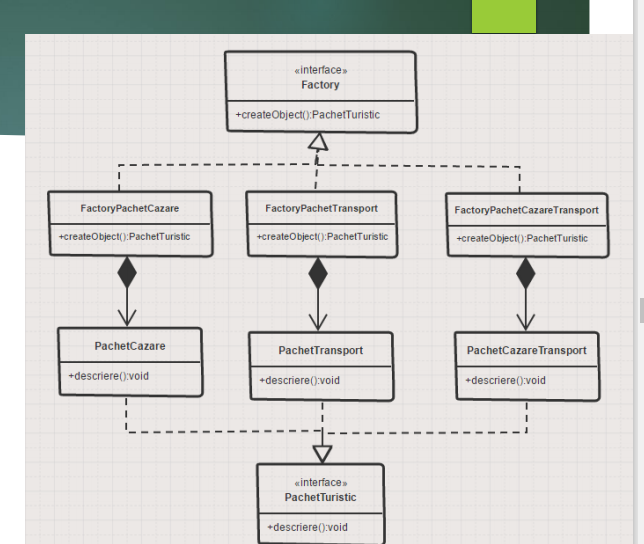
🡪existenta unei familii de obiecte intr o aplicatie

**Factory Method(Virtual Constructor)**

🡪abstractizeaza nivelul de creare

🡪nu mai avem nevoie de un swtich sau un if-else

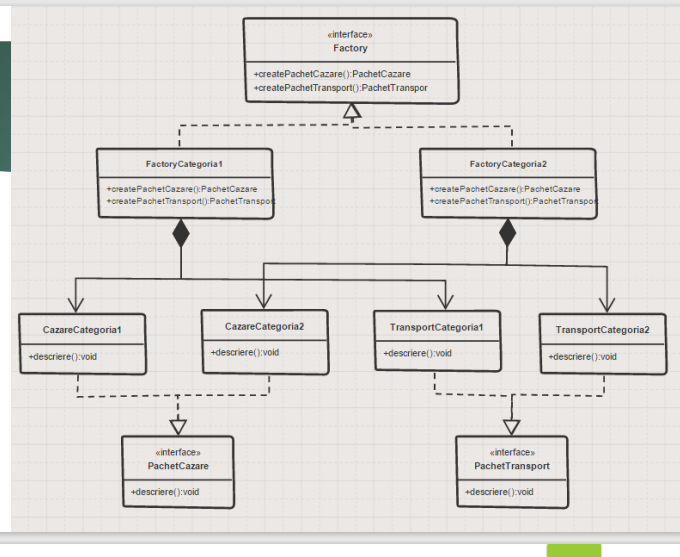
🡪pt apel se foloseste abstractizarile nu obiectel concrete



***Abstract Factory***

🡪Fiecare fabrica de obiecte creeaza obiecte din doua sau mai multe familii de obiecte

🡪fiecare factory va crea doua sau mai multe tipuri de obiecte.Pentru fiecare obiect va exista o metoda

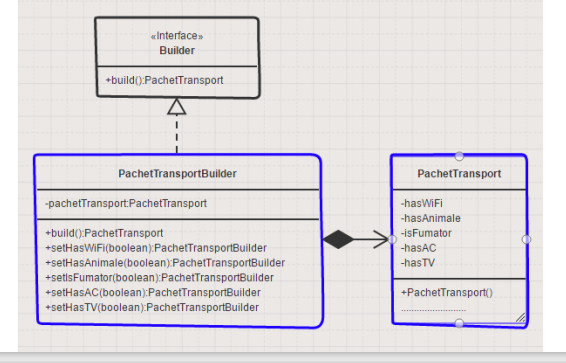


***Builder***

🡪ajuta la creare de obiecte concrete

🡪se construiesc obiecte complexe prin specificarea anumitor proprietati dorite din multitudinea existenta

🡪exista 2 variante de implementare:



Utilizari:

🡪obiecte cu multe atribute

Ex:StringBuilder

***Prototype***

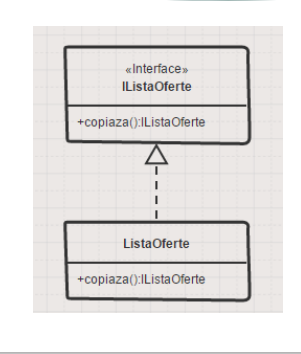
🡪creare de clone a caror construire dureaza f mult sau consuma f multe resurse

🡪metoda clone din Cloneable nu face deepCopy

🡪metoda de clonare trb implementata astfel incat sa realizeze deep copy si sa nu mai fie nevoie de interogarea bazei de date

Utilizari

🡪atunci cand obiectele create seamana intre ele iar creare unui obiect dureaza f mult sau consuma multe resurse



***Structural DP***

🡪ajuta la compunere si configurarea claselor si obiectelor

🡪compozitia claselor

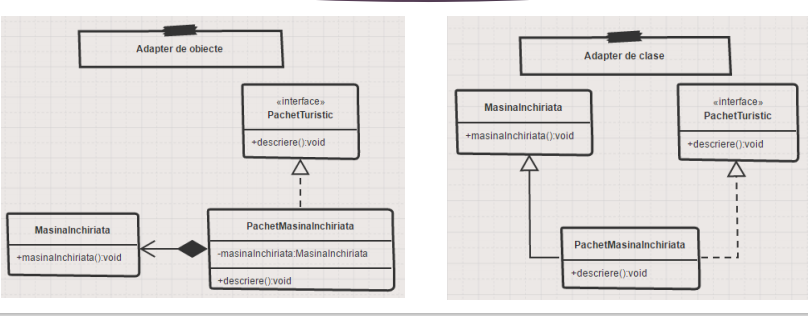
🡪realizeaza decuplarea interfetelor de clase

***Adapter***

🡪rezolva problema adaptarii utilizarii anumitor clase din framework-uri diferite, care nu au o interfata comuna

🡪clasele existente nu se vor modifica, ci se va adauga noi clase

🡪nu se adauga functionalitate!!!!.Functionalitatea este realizata de clasele existente



🡪nu se modifica codul existent

🡪ascunde clasa existenta la fel ca si PROXY

🡪la fel ca si FACADE,ambele sunt wrapere

***FACADE***

🡪usureaza lucrul cu framework-uri foarte complexe

🡪realizeaza o fata pt aceste framework-uri iar cine doreste sa utilizeze acele framework-uri,poate folosi aceasta fata fara a fi necesara cunoasterea tuturor claselor ,metodelor atributelor din cadrul framework-ului

Utilizari:framework-uri open source disponibile

***Decorator***

🡪modificarea functionalitatii la run-time

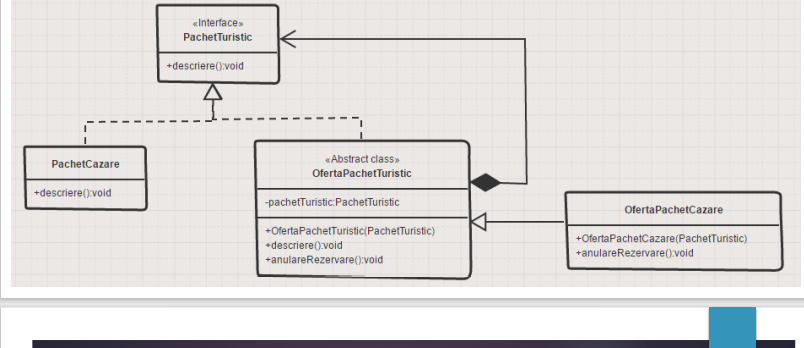
🡪adaugare de noio functionalitati la Run-Time

🡪se pot face decorari multiple prin mostenire continua

🡪adauga noi functionalitati claselor existente

🡪pt imbunatatirea codului existent fara a il modifica si fara a face extindere sau mostenire

🡪modifica intreaga clasa(strategy modifica doar comportamentul)



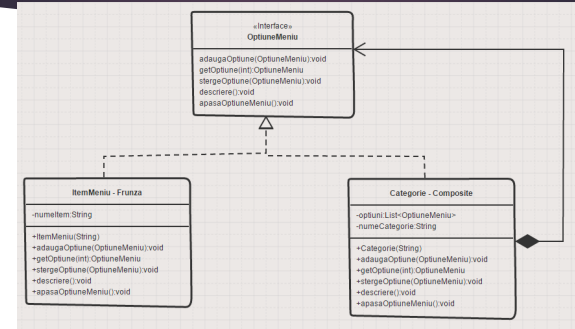
***Composite***

🡪creare de structura ierarhica sau structura arborescenta prin compunere

Utilizari:

🡪meniurile aplicatiilor

🡪meniurile de la restaurant



***Flyweight***

🡪atunci cand trb sa construim f multe obiecte sau instante ale unei clase, insa majoritatea obiectelor au o parte comuna sau permanenta

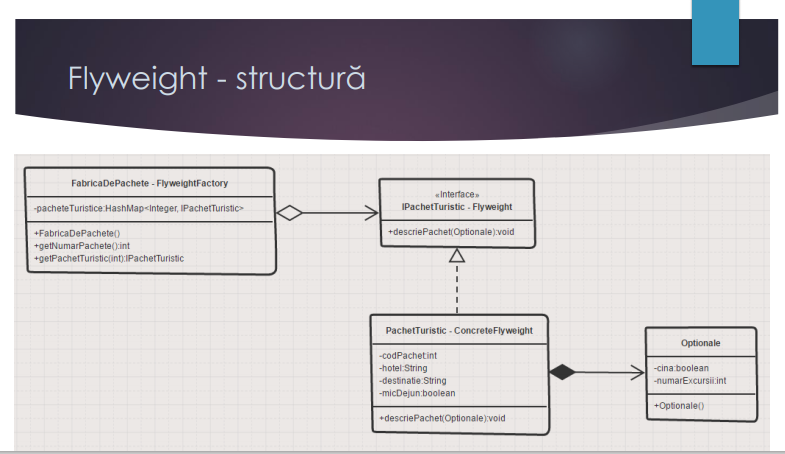
🡪se reduce consumul de memorie,pastrandu se intr o singura instanta partea comuna

🡪partea care difera de la un obiect la altul este salvata intr-o alta clasa si este adaugat dupa construirea obiectelor

🡪memorie in mod optim

🡪utilizari:in jocuri, atunci cand f multe modele seamna insa difera prin culoare sau prin pozitie

Corelatii:-Composite:Nodurile frunza pot reprezenta aceeasi instanta



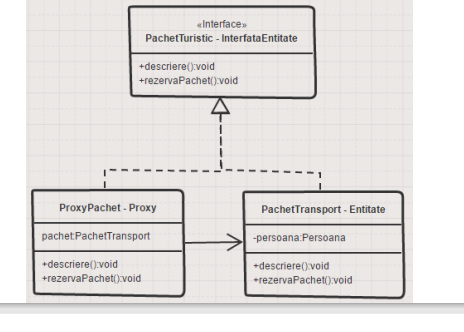
Proxy

🡪se doreste pastrarea functionalitatii unei clase, inca aceasta se va realiza doar in anumite conditii

🡪se controleaza comportamentul si accesul la un obiect

🡪ca si FACADE:reprezinta tot o interfata pt obiectul Real

🡪ca si Adapterul, ambele ascund clasa existenta



***DP Comportamentale***

🡪solutii pt o mai buna interactiune intre obiecte si clase

🡪controleaza relatii coomplexe intre clase

🡪distribuie responsabilitatea pe clase si descrie interactiunea dintre clase si obiecte

***Strategy***

🡪mai multi algoritmi pt rezolvarea unei probleme, alegerea se face la run-time

🡪fiecare comportament este dat de o clasa

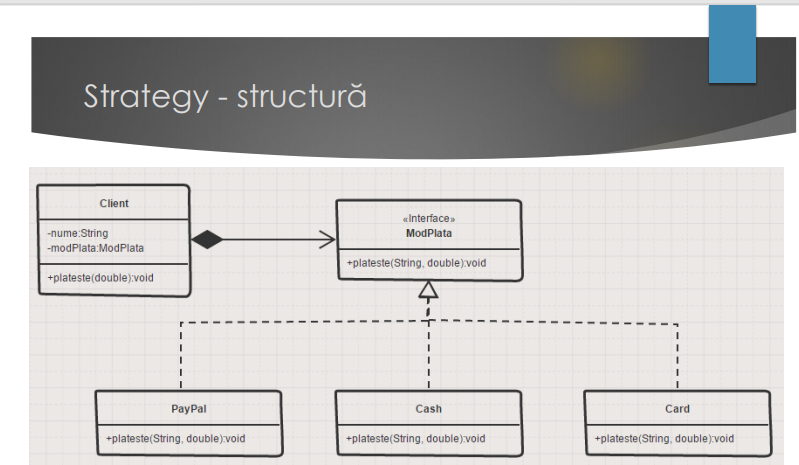
🡪defineste strategia adoptata la run-time

Utilizari:

🡪furnizarea metodei de comparare pt metodele de sortare

🡪utilizarea validatoarelor pt anumite controale

🡪strategia este data ca parametru

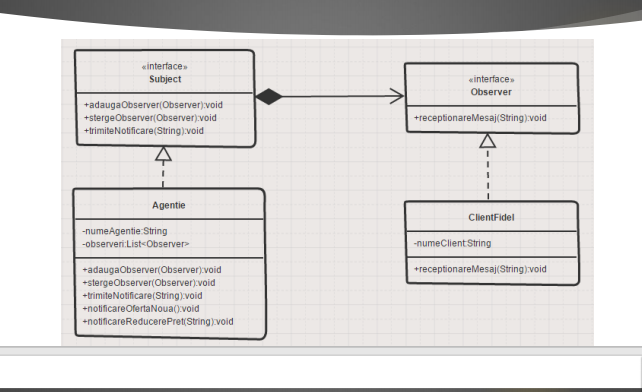


***Observer***

🡪defineste o relatie 1:n in care un subiect notifica mai multi observeri

🡪folosit atunci cand anumite elemente trb sa fie anuntate de schimbarea starii altor obiecte

🡪Utilizari:MVC



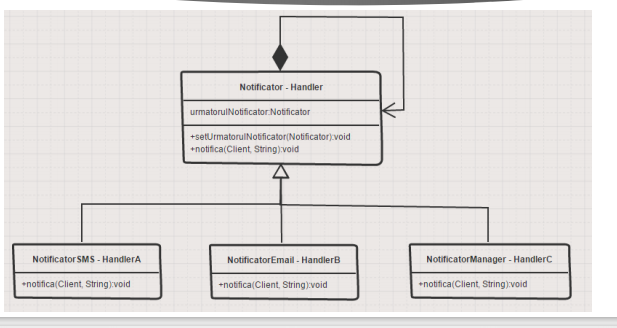
***Chain of Responsability***

🡪exista o lista de posibile obiecte ce pot rezolva problema

🡪obiectele se ordoneaza in lant, apoi cel care are problema de rezolvat apeleaza pentru prima za din lant

Utilizari:DNS Resolver

Ca si Composite:Parintele poate actiona ca si succesorul

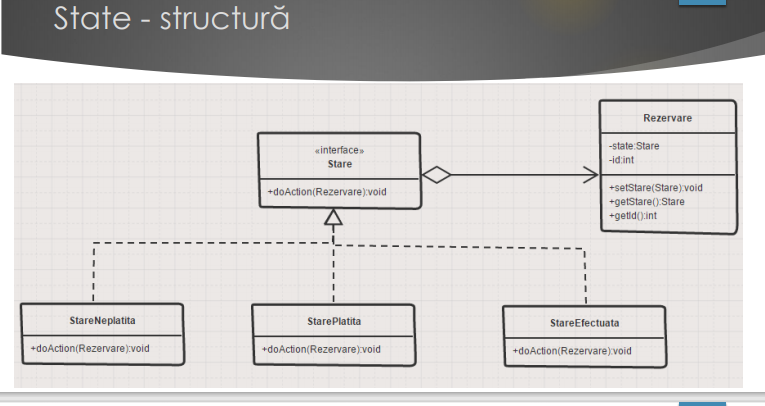


***State***

🡪atunci cand un obiect isi schimba comportamentul pe baza starii in care se afla

🡪Utilizari:starile rezervarilor sau a comenzilor din orice aplicatie

🡪evitarea structurii de switch sau if-else



***Command***

🡪**implementare lose coupling**

🡪ascunde aplicarea de comenzi, fara se stie concret ce presupune acea comanda

🡪clientul este decuplat de cel care executa actiunea

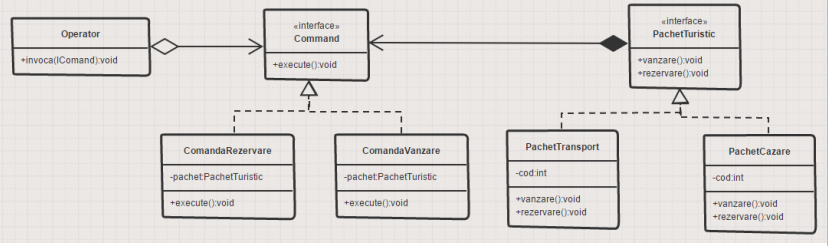
🡪pt implementare exista 2 situatii:

--invokerul sa fie folosit doar pt a invoca comenzile

--invokerul poate salva comenzile invocate si astfel se poate face undo pe comenzile invocate sau exitate

Utilizari:macro-urile,lucru cu fisiere,oriunde se doreste revenirea la o stare anterioara prin intermdiul comenzilor

La fel ca la adapter:se foloseste deja functionalitatea existenta



***Template***

🡪folosit atunci cand un algoritm este cunoscut si urmeaza anumiti pasi

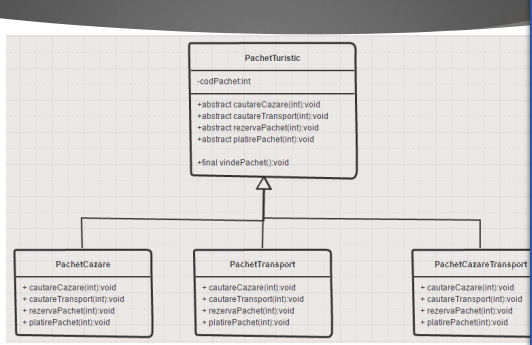
🡪fiecare pas este realizat de cate o metoda

🡪exista o metoda care implementeaza algoritmul si apeleaza celalte metode

🡪metoda care implementeaza algoritmul trebuie sa fie finala ca sa nu poata fi suprascrisa

🡪in cadrul claselor concrete sunt implementate doar metodele folosite in metoda template

Utilizari:modul de procesare cu nr finist si cunoscut de pasi,Backtracking



***Memento***

🡪se doreste salvarea anumitor stari pt obiectele unei clase

🡪permite salvare si revenirea la starile salvate ori de cate ori acest lucru este dorit

🡪backup,salvarea fisierelor

