PRINCIPIUL SEGREGARII INTERFETELOR - PSI

Clientii nu trebuie fortati sa depinda de interfete pe care nu le folosesc.

Este mai bine sa existe mai multe interfete specializate pe fiecare client decat una de interes general.

Impactul modificarii unei interfete va fi mai mic daca interfata este mai mica.

PRINCIPIUL ECHIVALENTEI REFOLOSIRE/FOLOSIRE - PER

Granularitatea de refolosire este granularitatea de distributie.

Doar componentele ce pot fi usor urmarite pot fi refolosite efficient.

Un element software reutilizabil nu poate fi refolosit cu adevarat daca nu este gestionat de un sistem de distributie.

In practica, nu exista clasa reutilizabila fara garantia notificarii, sigurantei si asistentei.

Trebuie integrat intregul modul.

Fie toate clasele unui pachet sunt reutilizabile, fie niciuna nu este.

PRINCIPIUL REUTILIZARII COMUNE - PRC

Toate clasele unui pachet trebuie refolosite impreuna.

Daca refolosim o clasa din pachet le refolosim pe toate.

Pachetele de component reutilizabile trebuie grupate dupa utilizarea asteptata, nu dupa functionalitate comuna sau alta clasificare arbitrara.

Cand se depinde de un pachet se depinde de fiecare clasa din acesta si in consecinta trebuie sa grupam in pachet clase ce pot fi refolosite impreuna.

PRINCIPIUL INCHIDERII COMUNE - PIC

Clasele unui pachet trebuie sa fie inchise fata de aceleasi tipuri de modificari.

O modificare ce afecteaza pachetul afecteaza toate clasele acelui pachet.

Clasele care se modifica impreuna trebuie grupate la un loc pentru a limita dispersarea modificarilor intre pachete.

Modificarile trebuie sa afecteze un numar minim de pachete.

Clasele unui pachet trebuie sa fie coezive si pentru un anumit tip de notificare fie se modifica toate clasele unei component fie niciuna.

PRINCIPIUL DEPENDENTELOR STABILE - PDS

Dependentele intre component trebuie sa fie in directia stabilitatii.

O componenta trebuie sa depinda de componente mai stabile decat ea.

Stabilitatea este o masura a dificultatii schimbarii unui modul.

Intrucat arhitectura si deciziile de design de nivel inalt nu se modifica frecvent, ar trebui plasate in pachete stabile. Aceste pachete ar trebui sa contina clase ce pot fi extinse, fara sa fie modificate, deci clase abstracte.

PRINCIPIUL ABSTRACTIUNILOR STABILE - PAS

Abstractizarea unui pachet trebuie sa fie proportionlala cu stabilitatea sa.

Pachetele maximal stabile trebuie sa fie maximal abstracte.

Pachetele instabile trebuie sa fie concrete.

Arhitectura ideala ar trebui sa contina pachetele instabile in partea superioara iar pachetele instabile in partea inferioara.

Pachetele stabile trebuie sa fie usor de extins si dificil de modificat.

PRINCIPIUL DESCHIS-INCHIS - PDI

Entitatile software trebuie sa fie deschise la extinderi si inchise la modificari.

Comportamentul modelului poate fi extins.

Codul sursa al modulului nu trebuie modificat.

Inchiderea se obtine prin folosirea abstractizarii.

Toate clasele membre trebuie sa fie private.

Nu se declara variabile globale.

Modificarea datelor publice pot provoca modificari in cascada.

PRINCIPIUL SUBSTITUTIEI LISKOV - PSL

Prin mostenire, orice proprietate adevarata despre obiectele supertipului trebuie sa ramana adevarata despre obiectele subtipului.

Functiile ce folosesc referinte la clasele de baza trebuie sa poata folosi obiecte ale claselor derivate.

Oriunde se face referire la o clasa in cod, orice subclasa a acesteia trebuie sa fie substituibila.

Este illegal ca o clasa derivate sa suprascrie o metoda a clasei de baza printr-o metoda NOP (no operation).

Solutiile sunt: relatia de derivare inversa , extragerea unei clase de baza comune.

PRINCIPIUL INVERSARII DEPENDENTELOR – PID

Modulele de nivel inalt nu trebuie sa depinda de modulele de nivel scazut. Ambele trebuie sa depinda de abstractiuni.

Abstractiunile nu trebuie sa depinda de detalii.

Detaliile trebuie sa depinda de abstractiuni.

O clasa de baza in ierarhie nu trebuie sa-si stie subclasele.

Modulele detaliate trebuie sa depinda de abstractiuni si nimic nu trebuie sa depinda de ele.

Avantajul proiectarii centrate pe interfete este ca tind sa se schimbe mai rar.

Trebuie evitate dependentele tranzititve, adica structurile in care straturi de nivel inalt depind de abstractiuni de nivel scazut.

Dependentele tranzitive pot fi eliminate prin folosirea mostenirii si a claselor stramos abstracte.

SABLONUL MEDIATOR

Defineste un obiect ce incapsuleaza cum interactioneaza o multime de obiecte.

Promoveaza cuplajul redus prin faptul ca obiectele nu fac referinta direct unele la altele.

Se incapsuleaza comportamentul colectiv intr-un obiect mediator separat.

Mediatorul coordoneaza interactiunile grupului de obiecte, previne ca obiectele sa se refere explicit unele la altele.

Se foloseste cand o multime de obiecte comunica intr-un mod bine definit dar complex.

Fiecare coleg comunica cu mediatorul sau oricand ar fi comunicat cu un alt coleg.

Avantaje:

- limiteaza derivarea
- modificarea comportamentului necesita doar derivarea mediatorului
- decupleaza colegii
- abstractizeaza cooperarea

Dezavantaje:

- centralizeaza controlul

Clase:

- Mediator – defineste o interfata pentru comunicarea cu obiectele colleague
- ConcreteMediator – coordoneaza obiectele colleague

SABLONUL ADAPTER

Converteste interfata unei clase intr-o interfata asteptata de client, permitand unor clase cu interfete diferite sa lucreze impreuna.

Un obiect adaptor poate adapta interfata clasei sale parinte.

Clase:

- Target defineste interfata pe care o foloseste clientul.
- Client colaboreaza cu obiecte in conformitate cu interfata Target.
- Adaptee defineste o interfata ce necesita adaptare.
- Adapter adapteaza interfata Adaptee la interfata Target.

Consecinte adaptor clasa:

- introduce un singur obiect si nu este necesara indirectarea suplimentara cu pointeri pentru a obtine obiectul adaptat.

Consecinte adaptor obiect:

- permite unui adapter sa lucreze cu mai multe clase Adaptee.

SABLONUL BUILDER

Separa construirea unui obiect complex de reprezentarea sa.

Acelasi proces de constructie poate genera reprezentari diferite.

In loc de a folosi constructori numerosi, se foloseste un alt obiect care primeste parametrii de initializare pas cu pas si returneaza la final obiectul construit.

Este util in crearea obiectelor ce contin date de tipul atribut-valoare ce nu se pot edita usor si trebuie construite deodata.

Se foloseste cand algoritmul de creare a obiectului trebuie sa fie independent de partile ce compun obiectul si modul de asamblare.

Clase:

- Builder specifica o interfata pentru crearea partilor unui obiect Product.
- ConcreteBuilder construieste partile unui produs prin implementarea interfetei Builder.
- Director construieste un obiect folosind interfata Builder, notifica Builder-ul cand trebuie construita o noua parte a obiectului.
- Product reprezinta obiectul complex care se construieste.

Consecinte:

- permite varierea reprezentarii interne a unui produs
- izoleaza codul pentru constructie si reprezentare.

SABLONUL SINGLETON

Asigura ca o clasa sa aiba o singura instanta si furnizeaza un punct unic de acces global la ea.

Singleton defineste o metoda Instance care devine singura prin care clientii pot accesa instanta unica.

Clientii acceseaza instantele Singleton doar prin metoda Instance.

Avantaje:

- se obtine acces controlat la instanta unica
- permite rafinarea operatiilor si reprezentarii
- permite un numar variabil de instante
- reduce vizibilitatea globala

Pentru a preveni instantierea clasei mai mult decat o data, constructorul este declarat private.

SABLONUL ITERATOR

Furnizeaza o modalitate de acces secvential la elementele unui agregat fara a expune reprezentarea interna.

Sustine traversari multiple ale obiectelor agregat si furnizeaza o interfata uniforma de traversare.

Clase:

- Iterator defineste o interfata pentru traversarea si accesarea elementelor.
- ConcreteIterator implementeaza interfata Iterator, retine pozitia curenta, tine evidenta obiectului curent din agregat, poate calcula urmatorul obiect din traversare.
- Aggregate defineste o interfata pentru crearea unui obiect Iterator.

- ConcreteAggregate implementeaza interfata de creare iterator, intoarce o instanta de ConcreteIterator.

Consecinte: sprijina variatiile in traversarea unui agregat, pot exista mai multe traversari simultan.

SABLONUL MEMENTO

Capteaza si externalizeaza starea interna a unui obiect, astfel incat se poate restaura starea mai tarziu.

Se foloseste cand starea unui obiect trebuie salvata pentru a readuce mai tarziu obiectul in aceasta stare.

Clase:

- Memento stocheaza starea interna a obiectului Originator.
- Originator reprezinta obiectul a carui stare trebuie salvata, creeaza un Memento continand un extras al starii sale interne, foloseste Memento pentru a-si reface starea interna.
- Caretaker raspunde de pastrarea Memento in siguranta, gestioneaza momentul salvarii, salveaza Memento, foloseste Memento pentru a restabili starea obiectului Originator.

Consecinte: simplifica Originatorul, uneori folosirea este costisitoare daca este multa informatie de copiat.

SABLONUL STRATEGY

Defineste o familie de algoritmi, incapsuland fiecare algoritm si facandu-i interschimbabili.

Permite algoritmului sa varieze indiferent de clientii ce il folosesc.

Se foloseste cand sunt necesare variante diferite ale unui algoritm, un algoritm foloseste date despre care clientii nu trebuie sa stie.

Clase:

- Strategy declara o interfata comuna tuturor algoritmilor.
- ConcreteStrategy implementeaza algoritmul.
- Context apeleaza algoritmul.

Consecinte: este o alternativa la derivare, elimina instructiunile conditionale, se poate face exces de comunicare intre Strategy si Context, clientii trebuie sa fie constienti de diferitele strategii, numar mare de obiecte.

SABLONUL DECORATOR

Adauga responsabilitati unui obiect particular mai degraba decat clasei sale.

Ataseaza dinamic responsabilitati aditionale unui obiect, fiind o alternativa flexibila la derivare.

Se foloseste cand extinderea prin derivare nu este practica.

Clase:

- Component defineste interfata obiectelor carora li se pot adauga dinamic responsabilitati.
- ConcreteComponent reprezinta obiectele la care se pot adauga responsabilitati.
- Decorator mentine o referinta catre un obiect Component.
- ConcreteDecorator adauga responsabilitati componentei.

Avantaje:

- este mai flexibil decat mostenirea
- evita clasele incarcate de prea multe caracteristici in partea de sus a ierarhiei

Dezavantaj:

- un decorator si componenta sa nu sunt identice

SABLONUL OBSERVER

Cand un obiect isi schimba starea, toate dependentele sale sunt anuntate si actualizate automat.

Se foloseste cand modificarea unui obiect solicita modificarea altora, un obiect trebuie sa poata notifica alte obiecte.

Pot exista mai multi observatori si fiecare poate reactiona diferit la aceeasi notificare.

Clase:

- Subject isi cunoaste observatorii, defineste o interfata pentru atasarea si inlaturarea de obiecte Observer.
- Observer defineste o interfata de actualizare pentru obiectele ce trebuie notificate la modificarile din subiect.
- ConcreteSubject isi notifica observatorii cand se modifica starea.
- ConcreteObserver stocheaza starea ce trebuie sa ramana consistenta cu a subiectului.

Avantaje:

- cuplajul minim intre subiect si observatori
- suport pentru comunicare de tip difuzare

Dezavantaj:

- observatorii nu stiu unul de existenta celuilalt si pot aparea modificari in cascada.

SABLONUL PROXY

Furnizeaza un substitut pentru alt obiect, pentru a controla accesul la acesta.

Se foloseste cand e nevoie de o referinta la un obiect mai flexibila decat un pointer.

Clase:

- Proxy mentine o referinta ce permite sa acceseze subiectul real, controleaza accesul la subiectul real.
- Subject defineste interfata comuna pentru RealSubject si Proxy.
- RealSubject defineste obiectul real.

Consecinte: introduce un nivel de indirectare, face copiere in timpul scrierii.

SABLONUL BRIDGE

Decupleaza o abstractiune de implementarea sa, permitand implementarii sa varieze.

Abstractiunea defineste o interfata.

O abstractiune poate folosi implementari diferite.

O implementare poate fi folosita in diferite abstractiuni.

Se foloseste cand trebuie evitata legarea permanenta intre o abstractiune si implementarea sa, se ascunde implementarea unei abstractiuni complet de clienti.

Clase:

- Abstraction defineste interfata abstractiei, mentine o referinta la un obiect Implementor.

- Implementor defineste interfata pentru clasele de implementare, contine operatii primitive pe care se bazeaza cele de nivel inalt din Abstraction.
- RefinedAbstraction extinde interfata Abstraction.
- ConcreteImplementor defineste implementarea concreta pentru interfata Implementor.

Consecinte: decupleaza interfata si implementarea, reduce dependentele de la momentul compilarii, extensibilitatea este imbunatatita, ascunde detaliile de implementare de clienti.

SABLONUL COMMAND

Incapsuleaza cereri ca obiecte si permite inregistrarea si inlantuirea cererilor.

Clientul apeleaza la o interfata, nu este nevoit sa cunoasca actiunea pe care o cere.

Actiunea se poate modifica fara sa fie modificat codul client.

Clase:

- Command declara interfata pentru executarea operatiei.
- ConcreteCommand asociaza o actiune concreta unei cereri.
- Invoker solicita executarea cererii.
- Receiver executa operatii asociate cu indeplinirea unei cereri.
- Client creeaza o comanda concreta si seteaza primitorul sau.

Avantaje:

- comenzile sunt obiecte ale unor clase si pot fi extinse
- este usor de adaugat comenzi noi

Dezavantaj:

- potential exces de clase de tip Command

SABLONUL STATE

Permite unui obiect sa isi modifice comportamentul cand I se modifica starea interna.

Se foloseste cand comportamentul obiectului depinde de starea sa.

Inlocuieste operatiile cu instructiuni conditionale depinzand de starea obiectului.

Clase:

- Context defineste interfata de interes pentru clienti, mentine o instanta a unei subclase ConcreteState.
- State defineste o interfata pentru un comportament asociat cu o stare particulara.
- ConcreteState implementeaza un comportament asociat cu o stare.

Consecinte: localizeaza comportamentul specific starilor, partitioneaza comportamentul pe stari diferite, face explicita tranzitia intre stari.

SABLONUL COMPOSITE

Trateaza uniform obiectele individuale si compunerile lor.

Se foloseste cand trebuie reprezentate ierarhii de tipul parte-intreg, permitand ignorarea diferentei dintre compuneri de obiecte si obiecte individuale.

Clase:

- Component declara interfata pentru obiectele din compunere.
- Composite defineste comportamentul pentru componentele cu copii.
- Leaf defineste comportamentul pentru obiectele primitive din compunere.
- Client manipuleaza obiectele prin interfata Component.

Avantaje:

- defineste ierarhii uniforme de clase
- face clientii simpli

Dezavantaj:

- designul prea general

SABLONUL ABSTRACT FACTORY

Ofera o interfata pentru crearea de familii de obiecte inrudite sau dependente fara specificarea claselor lor concrete.

Clase:

- AbstractFactory declara o interfata pentru creare de produse.
- ConcreteFactory implementeaza operatiile de creare a produselor.
- AbstractProduct declara o interfata pentru un tip de obiecte produs.
- ConcreteProduct implementeaza o interfata pentru un tip de obiecte produs.
- Client foloseste interfetele.

Consecinte: se izoleaza clasele concrete, faciliteaza schimbul de familii de produse, este dificila sustinerea de tipuri noi de produse.

SABLONUL PROTOTYPE

Este util cand instantierea unei clase este consumatoare de timp sau complexa.

In loc de a crea mai multe instante, se creeaza obiecte noi prin copierea unei instante originale si aceste copii se modifica dupa necesitati.

Clase:

- Prototype declara o interfata pentru clonare
- ConcretePrototype implementeaza o interfata pentru clonare.
- Client creeaza un obiect nou cerand prototipului sa se cloneze.

Avantaje:

- clasele de instantiat pot fi specificate dinamic
- mai putina derivare
- specificarea de obiecte noi prin varierea valorilor sau structurii prototipurilor
- codul client este stabil

Dezavantaj:

- fiecare subclasa a Prototype trebuie sa implementeze clone.

SABLONUL FLYWEIGHT

Foloseste partajarea pentru a sustine implementarea eficienta a unui numar mare de obiecte cu structura complexa.

Flyweight este un obiect partajat care poate fi folosit simultan in contexte diferite.

Numarul de instante se reduce semnificativ prin recunoasterea faptului ca de fapt clasele sunt la fel cu exceptia catorva parametri.

Se distinge intre stare intrinseca, independenta de context, si stare extrinseca ce depinde de context.

Se foloseste cand aplicatia are numar mare de obiecte si nu depinde de identitatea obiectelor.

Clase:

- Flyweight declara o interfata prin care obiectele flyweight primesc starea extrinseca si pot actiona asupra ei.
- ConcreteFlyweight implementeaza interfata Flyweight si adauga stocarea starii intrinseci.
- FlyweightFactory creeaza si administreaza obiectele flyweight, se asigura ca sunt partajate corect.
- Cand un Client necesita un flyweight, FlyweightFactory furnizeaza o instanta existenta sau creeaza una daca o astfel de instanta nu exista.

SABLONUL FACTORY

Se foloseste cand o clasa nu poate anticipa clasa obiectelor pe care trebuie sa le creeze.

Se defineste o interfata pentru crearea unui obiect iar de instantiere se ocupa subclasele.

Clase:

- Product defineste interfata obiectelor ce vor fi create
- ConcreteProduct implementeaza interfata
- Creator declara metoda Factory care returneaza un Product
- ConcreteFactory suprascrie metoda Factory pentru a furniza o instanta ConcreteProduct.

Avantaje:

- faciliteaza derivarea
- poate conecta ierarhii paralele de clase.

SABLONUL CHAIN OF RESPONSABILITY

Decupleaza transmitatorul cererii de primitorul acesteia, oferind mai multor obiecte sansa de a trata cererea.

Primitorii sunt ordonati intr-un lant si cererea este transmisa de-a lungul lantului pana ce un obiect o trateaza.

Se foloseste cand mai mult de un obiect poate trata o cerere, multimea obiectelor ce pot trata cererea trebuie sa fie specificabila dinamic.

Clase:

- Handler defineste interfata pentru tratarea cererilor
- ConcreteHandler fie trateaza cererea, fie o trimite succesorului.

- Client initiaza transmiterea cererii catre un obiect ConcreteHandler.

Avantaje:

- cuplajul este redus
- flexibilitate in asignarea responsabilitatii

Dezavantaj:

- cererile pot ramane netratate daca lantul nu a fost configurat corect.