Projektni uzorci ponašanja (Behavioral design patterns-DP)

Posmatrač (engl. *Observer*) – objektni uzorak ponašanja

Drugi nazivi:

Dependents

Može se uočiti da neke aukcije upravo demonstriraju ovaj pattern. Svaki takmičar u licitaciji poseduje brojne kartice kao učesnik.
Čim startuje aukcija "observira" se podizanje kartice kao potvrda prihvatanja iznosa. Prihvatanje iznosa menja cenu o čemu su obavešteni (notificirani) ostali učesnici aukcije.

Publish-Subscribe

U odnosu izdavaštvo-pretplata Subject je izdavač obaveštenja na koja su Observeri pretplaćeni. Subject šalje obaveštenja do svih Observer-a. Ne postoje uslovi oko broja Observer-a koji se pretplaćuju.

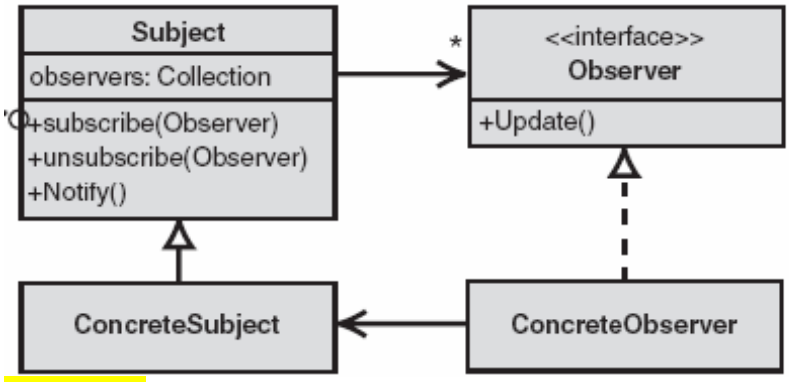
Namena:

-definiše 1:n zavisnost između objekata takvu da kada jedan objekat promeni stanje svi zavisni objekti se obaveste i automatski ažuriraju

Primenljivost

U okruženjima koja razdvajaju sadržaj svog modela od samog prikaza. Naime, sam prikaz modela može biti raznovrstan, te je nužno održavati sve prikaze ažurnim kada se podaci promene kroz bilo koji prikaz.
Na primer, isti podaci u programu za tabelarna izračunavanja mogu da se prikažu tabelarno ili grafički na više načina (histogram, radni list,pie chart, linijski dijagram,...), te je jasno da moraju prikazi biti ažurni pri promeni ma kog podatka kroz ma koji prikaz.

DIJAGRAM PRIKAZA – klasni dijagram



Učesnici:

Subject
Zna za svoje posmatrače. Njega može da posmatra bilo koji broj posmatrača.
Obezbeđuje interfejs za povezivanje i raskidanje veze sa objektima Observera.

Observer
Definiše interfejs za ažuriranje objekta koje treba obavestiti o promeni subjekta.

ConcreteSubject
Čuva stanje od interesa za objekte *ConcreteObserver.*
Šalje obaveštenje posmatračima kada se stanje promeni.

ConcreteObserver
Održava referencu na objekat *ConcreteSubject*.
Čuva stanje koje treba da ostane konzistentno sa stanjem subjekta.
Implementira *Observer* interfejs za ažuriranje da bi stanje ostalo konzistentno sa stanjem subjekta.

Saradnja:
–konkretan subjekat signalizira svojim posmatračima svaku promenu svog stanja
–nakon poziva *Update*, konkretan posmatrač traži od subjekta informaciju o stanju
–posmatrač koristi informaciju o stanju subjekta da ažurira svoje stanje

Prednosti i mane:

Prednosti upotrebe Observer DP-a

- Nezavisna izmena subjekata i observer-a
  I subjekti mogu biti ponovno korišćeni bez svojih observer-a i obrnuto. Sami observer-i mogu da budu pridodati, bez izmene subjekta i ostalih observer-a.
- Nije neophodno navesti primaoca u obaveštenju koje šalje subjekt
  Jer to obaveštenje biva emitovano do svih "zainteresovanih" objekata (tj.objekata koji su se pretplatili na to obaveštenje).
  Subjekat je zadužen da obaveštava svoje observer-e, ali ne i da vodi računa o tome koliko ima zaintersovanih za to obaveštenje.
  Otuda se observer-i mogu dodavati bilo kada ili uklanjati.Sam observer je zadužen da odlluči da li će obaveštenje biti obrađeno, ili ignorisano.

Mana upotrebe Observer DP-a

- kaskadna i neusklađena ažuriranja
  Kako Observer-i ne znaju jedan za drugog, oni ne znaju ni za sve promene koje su se desile u subjektu.
  Zato valja dodati izvesne protokole ažuriranja (kojima bi se utvrdile promene u svojoj celovitosti) kao vid preventive od:
  1. kaskadnog ažuriranja observer-a (npr. neka od operacija nad subjektom može da izazove ažuriranja tog tipa i nad objektima koji zavise od observer-a) i od
  2. neusklađenog ažuriranja (npr. ako kriterijumi zavisnosti među objektima su nedovoljno dobro definisani)

Primeri korišćenja Observer DP-a

Postoji bliskost Observer pattern-a sa Model-View-Controller arhitekturom poznatom još od Smalltalk, jer u opštem slučaju u Observer pattern-u učestvuju sledeće tri uloge:

- Subjekt ili model za čiji sadrzaj su zainteresovani posmatrači.
- Posmatrač ili pogled ili view koji prikazuje sadržaj subjekta, te je zainteresovan i za njegove promene.
- Kontrolor ili controller koji održava vezu posmatrača putem čuvanja njihovog spiska i obaveštavanja o izmenama sadrzaja izvora.

Interfejs java.util.Observer određuje metod za ažuriranje objekta klase java.util.Observable .
U Observer DP-u klasa Observable predstavlja objekat čija promena stanja je od interesa za objekat Observer.
AWT Listeners su Observer varijanta. (java.awt.Component , java.awt.event.XXListener ,...)

Bliski DP: za kapsuliranje kompleksne semantike ažuriranja, *Mediator DP* može da posreduje između subjekata i posmatrača

Strategija (engl. Strategy)

Namena:

Definiše familiju algoritama, vrši njihovu enkapsulaciju i omogućava jednostavnu zamenu algoritama nezavisnu od klijenta. Na primer, moguće je, u zavisnosti od ulaza i stanja aplikacije izabrati algoritam u vreme izvršavanja. Klijent nije "svestan" da se zamena dogodila.

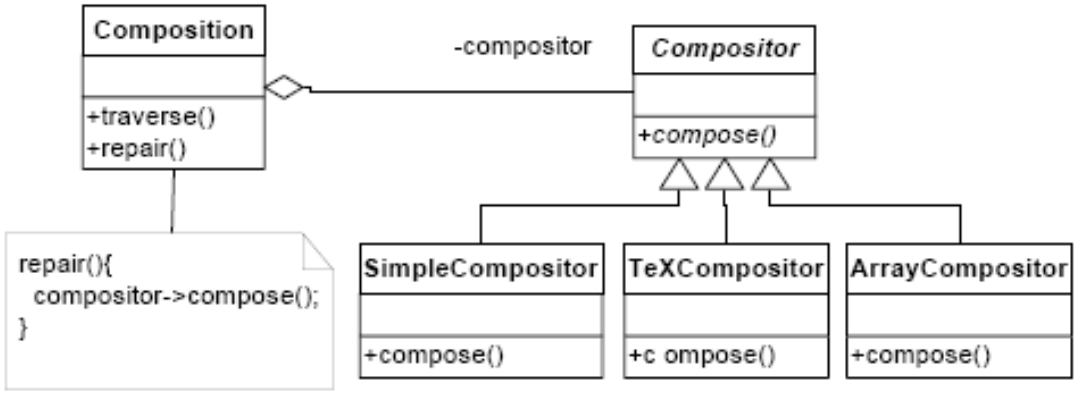
Drugo ime:

Politika (Policy)

Motivacija:

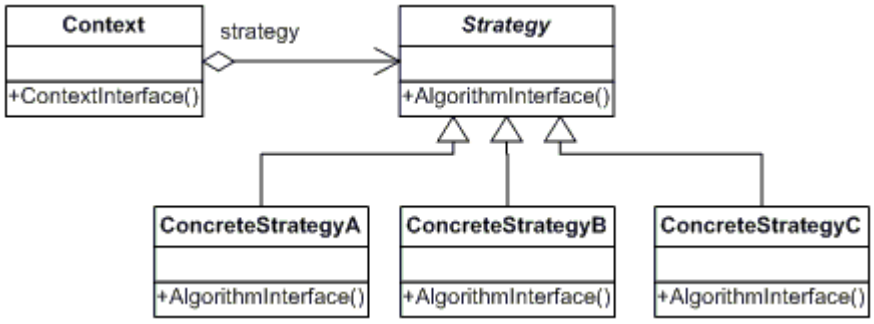
–postoji više algoritama za prelom teksta u linije
–ugrađivanje tih algoritama u klase koje ih zahtevaju nije dobro iz više razloga:

- klijenti kojima je potreban prelom postaju kompleksniji i teži za održavanje
- različiti algoritmi su odgovarajući u različitim trenucima, a teško ih je menjati
- teško je dodati novi algoritam i varirati postojeći



Rešenje: definisati klase koje enkapsuliraju različite algoritme prelamanja
–klasa Composition je odgovorna za održavanje i ažuriranje preloma prikazanog teksta
–strategije preloma nisu implementirane u Composition, već u potklasama Compositor
–klasa Composition sadrži referencu na objekat tipa Compositor
–kada Composition reformatira tekst – ona prosleđuje tu odgovornost Compositor objektu

Dijagram klasa



Strategy (Compositor)
deklariše zajednički interfejs za sve podržane algoritme. Objekti klase kontekst koriste ovaj interfejs da pozovu algoritme definisane ConcreteStrategy familijom klasa.
ConcreteStrategy (TeXCompositor, SimpleCompositor, ArrayCompositor)
implementira algoritam koristeći interfejs klase Strategy
Context (Composition)
konfiguriše se pomoću ConcreteStrategy objekata
čuva referencu na objekat klase Strategy i može da definiše interfejs koji objektu klase Strategy dozvoljava pristup podacima

Primenljivost:

- kada se više srodnih klasa razlikuju samo po ponašanju, Strategija DP omogućava konfigurisanje klase jednim od više ponašanja
- kada su potrebne različite varijante nekog algoritma
- kada algoritam koristi podatke o kojima klijenti ne treba ništa da znaju; izbegava se eksponiranje kompleksnih struktura podataka specifičnih za algoritam (potrebno kapsuliranje algoritma i podataka)
- kada klasa definiše više ponašanja koja se pojavljuju kao grane uslovne naredbe u operaciji; grane uslovne naredbe treba kapsulirati u njima odgovarajuće *Strategy* klase

Prednosti i mane:

–Strategije eliminišu potrebu za uslovnim naredbama u klijentskom kodu
–Izbor implementacija –klijent bira da postigne performanse u vremenu/prostoru
–Klijenti moraju biti svesni različitih strategija –nedostatak

Bliski DP:

Strategy objekti često predstavljaju dobre *Flyweight* objekte