Objektno-orijentisano programiranje 2

Lanac odgovornosti

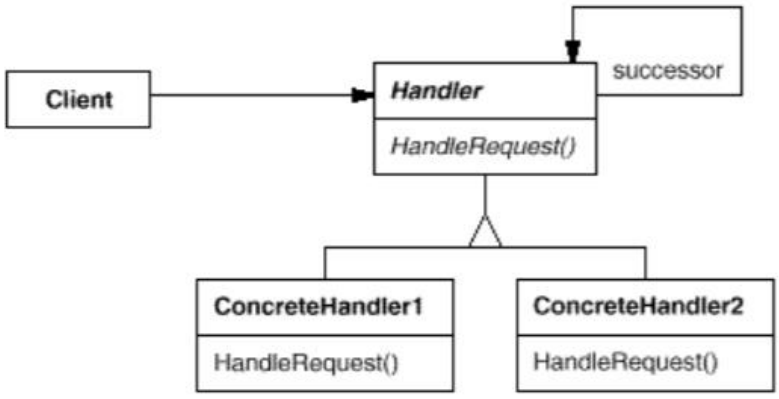
Motivacija i osnovna ideja:

Pretpostavimo da treba da implementiramo funkcionalnost obrade različitih korisničkih zahteva. Način obrade zahteva zavisi od toga kakav je zahtev upućen i u duhu dobre OOP prakse želimo da imamo po jednu klasu za svaki specifični tip zahteva. Ponašanje koje želimo da postignemo je da klijenta oslobodimo brige o tome koja tačno klasa „ume“ da obradi njegov zahtev. Klijent treba da pošalje zahtev, a nekako u pozadini treba se postarati da klasa koja može obraditi dati zahtev taj zahtev i obradi.

Štaviše, možda više klasa može obraditi isti zahtev. Tada bismo možda želeli da damo prednost nekoj klasi pri obradi datog zahteva. Da postignemo željeno ponašanje možemo iskoristiti projektni uzorak Lanac odgovornosti. Kao što smo već opisali, potrebno je imati konkretne klase koje mogu da obrade zahtev (na UML-u nazvane ConcreteHandler1 i ConcreteHandler2), a koje su izvedene iz zajedničke apstraktne klase (na UML-u klasa Handler). Da postignemo željeno ponašanje, ključna ideja je da konkretne *handler*-e organizujemo kao listu, tj. lanac kroz koji naš zahtev treba da “putuje” dok god ga neki objekat ne obradi. Svaki konkretan *handler* treba biti u stanju da proceni da li može da obradi dati zahtev. Ukoliko može, treba i da ga obradi, a u suprotnom treba da prepusti zahtev sledećem *handler*-u u *lancu odgovornosti.* Naravno, može se desiti da zahtev ostane neobrađen ukoliko nijedan *handler* u lancu ne može da ga obradi.

Lanac handler-a organizujemo prosto kao jednostruko povezanu listu. Dovoljno je da svaki handler zna koji je naredni (naredni handler može se smatrati “nadređenim”).

UML dijagram projektnog uzorka Lanac odgovornosti dat je na slici 2.



Slika 1: Projektni uzorak Lanac odgovornosti

Zadaci:

1. Napisati klasu Logger i izvedene klase DebugLogger, InfoLogger, WarningLogger i ErrorLogger. Omogućiti korisniku da izvrši liniju poput:
   logger->log(“poruka”, LogLevel::WARNING);
   logger treba biti objekat tipa DebugLogger koji prosleđuje poruku na obradu InfoLogger-u ukoliko ne može da je obradi, itd.
2. Napisati klasu InternetServiceHandler koja ima metodu WebPage* getPage(string url, string ip). Napisati i klase DownloadPageInternetServiceHandler kojom se preuzima sadržaj sa traženog url-a, RestrictedAccessInternetServiceHandler koja ima listu ip adresa sa kojih je zabranjeno preuzimati sadržaj, kao i klasu CacheInternetServiceHandler koja pamti ranije preuzete sadržaje i vraća ih ukoliko se budu ponovo tražili.
   Kada korisnik zatraži sadržaj stranice treba prvo ispitati ima li pravo da traži dati sadržaj, zatim da li je dati sadržaj već dostupan i tek na kraju preuzeti sadržaj (ukoliko mora).
   Koja je razlika ove primene u odnosnu na Dekoratora?

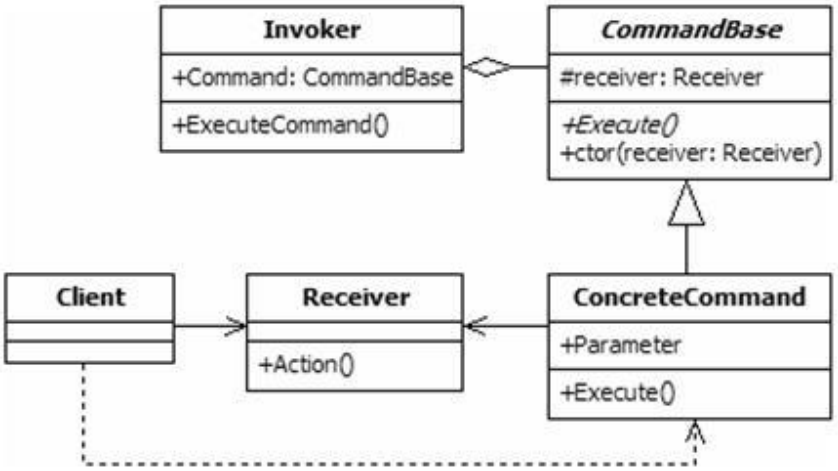
Komanda

Motivacija i osnovna ideja:

Zadatak projektnog uzorka Komanda je da razdvoji pozive metode nad nekim objektom od samog tog objekta i omogući njihovo planiranje i upravljanje. Na primer, možda želimo da napravimo “red čekanja” izvršenja nekih metoda, a da samo izvršenje obavimo kasnije ili možda želimo da pamtimo istoriju poziva metoda ili da omogućimo poništenje efekta neke metode.

Projektni uzorak Komanda omogućuje ove funkcionalnosti pomenutim razdvajanjem klijenta od samog poziva metoda. Izvršenje metoda prepuštamo zasebnim klasama iz hijerarhije klasa *komanda* (na UML dijagramu CommandBase). Komanda treba da “zna” nad kojim objektom treba izvršiti određenu metodu, koju metodu treba da izvrši (obično je hardkodovano) i sa kojim argumentima. Poziv željene metode vršimo preko metode klase komanda koju tipično nazivamo *execute()*. Dodatno razdvajanje postižemo time što klijentu samo prepuštamo kreiranje komandi. Nije na klijentu da direktno zahteva njihovo izvršenje. Komande izvršava posebna klasa *Pokretač* (engl. *Invoker*). Pokretač može u sebi imati logiku za pamćenje istorije komandi, a svaka komanda osim execute() može imati i undo(). Pamćenjem istorije komandi mogli bismo onda proizvoljan put (dok god je to moguće) pozivati operacije *undo(), redo()*. Klasu čija se metoda izvršava komandom nazivamo *Primalac* (engl. *Receiver*).

Učesnici projektnog uzorka Komanda i njihova komunikacija dati su UML dijagramom prikazanim na slici 2.



Slika 2: Projektni uzorak Komanda

Zadaci:

3. Implementirati opšti oblik projektnog uzorka Komanda u skladu sa UML dijagramom na slici 2.
4. Implementirati klasu Dokument koja od atributa ima sadržaj i ime fajla. Od metoda dodati metode save(), addChar(), insertChar(). Kreirati odgovarajuće komande koje osim execute() imaju i undo() metodu. Implementirati klasu Invoker kojom je moguće dodati nove komande u red, pokrenuti sve komande u redu, proslediti komandu koju treba odmah izvršiti i metode undo() i redo().