STRUKTURALNI PATERNI

Strukturalni paterni bave se kompozicijom i obično predstavljaju različite načine za definisanje odnosa među objektima. Oni obezbeđuju da kada je neophodna promjena u jednom dijelu sistema, ostatak sistema ne mora da se mijenja. Također, pomažu da svaki dio sistema radi ono čemu je najbolje prilagođen.

U paterne strukture se ubrajaju: Adapter , Bridge, Composite, Decorator, Façade, Proxy.

Adapter patern

Osnovna namjena Adapter paterna je da omogući širu upotrebu već postojećih klasa. U situacijama kada je potreban drugačiji interfejs već postojeće klase, a ne želimo mijenjati postojeću klasu koristi se Adapter patern. Adapter patern kreira novu adapter klasu koja služi kao posrednik između originalne klase i željenog interfejsa. Tim postupkom se dobija željena funkcionalnost bez izmjena na originalnoj klasi i bez ugrožavanja integriteta cijele aplikacije. U slučaju da originalna klasa sadrži samo mali dio funkcionalnosti onda je bolji pristup kreirati novu klasu i zaobići korištenje Adapter paterna. Adapter patern ustvari mapira interfejs jedne klase u drugu tako da mogu raditi zajedno. Nekompatibilne klase mogu doći iz različitih biblioteka.

Naša primjena na dijagram:

Ukoliko bismo nadogradili naš sistem, na način da omogućimo korisniku da pretražuje rute na osnovu neke ključne riječi, u našem slučaju dužine rute, iskoristili bismo Adapter patern. Dakle, kada korisnik odabere ključnu riječ za pretragu, nudi mu se mogućnost da unese dužinu koja mu odgovara, pa će se uz pretraživanje po ključnoj riječi, ponuđene rute filtrirati, to jeste prilagoditi se zahtjevu korisnika (ovo će se desiti samo onda kada se za tim javi potreba, tj. zahtjev korisnika, ali inače ukoliko se ovaj zahtjev ne pojavi korisnik će vidjeti sve lokacije određene tom ključnom riječi).

Koraci:

1. Definisati interfejs IPretraga s metodom pretraziPoDuzini()

2. Definisati klasu RutaAdapter, koja implementira interfejs IPretraga, koja poziva metodu PretražiPoKljucnojRijeci klase Ruta, pri tome prilagođavajući ovu metodu da vraća filtrirano po dužini.

Ova nadogradnja (korištenje Adapter paterna) bi omogućila da se kasnije sistem vrlo lahko nadogradi i za druge vrste pretraga, npr. ukoliko se javi potreba ida se rute pretražuju po trajanju, ocjeni, itd.

Decorator patern

Osnovna namjena Decorator paterna je da omogući dinamičko dodavanje novih elemenata i ponašanja (funkcionalnosti) postojećim objektima. Objekat pri tome ne zna da je urađena dekoracija što je veoma korisno za ponovnu upotrebu komponenti softverskog sistema.

Ovaj patern ćemo primijeniti na naš dijagram klasa na način da ćemo omogućiti uređivanje informacija o rutama.

Koraci:

1. Dodati interfejs IRuta koji će sadržati definicije metoda izmijeni() i getRuta(), koje su prethodno definisane u klasi Ruta;

2. Dodati nekoliko novih klasa koje će se odnositi na pojedine vrste izmjene ruta (promijeniOpis...), koje će naslijediti interfejs IRuta.

Composite pattern

Osnovna namjena *Composite* patterna je da omogući formiranje strukture stabla pomoću klasa, u kojoj se individualni objekti (listovi stabla) i kompozicije individualnih objekata (korijeni stabla) jednako tretiraju. Ovaj pattern omogućava da svaki list stabla izvrši neki zadatak ukoliko mi zatražimo da to učini.

Koraci:

1. Dodati interfejs ClanTima koji će sadržavati metodu dajInformacije() koja služi za identifikaciju pojedinca (da li je osoba korisnik ili vodič).

2. Dodati klasu Tim koja će služiti za kreiranje jednog tima (tim čine vodič i korisnici koji su njega izabrali).