STRUKTURALNI PATERNI

1. Composite pattern

Composite pattern moguće je iskoristiti na sljedeći način:
-Filtriranje letova:
Naša web aplikacija podržava filtriranje letova prema različitim kriterijima, poput destinacije ili cijene.
-Composite pattern možemo koristiti za implementaciju filtera za letove. Možemo imati apstraktnu klasu "Filter" koja definiše osnovne metode filtriranja. Zatim, možemo imati implementaciju composite-a koja nasljeđuje tu klasu i predstavlja složeni filter. Ova implementacija može sadržavati podfiltere kao svoje komponente. Na primjer, klasa "SlozeniFilter" može naslijediti "Filter" i sadržavati podfiltere poput "DestinacijaFilter", "CijenaFilter". Kada se primijeni "SlozeniFilter", on će izvršiti filtriranje koristeći kombinaciju podfiltera kako bi pružio rezultate koji zadovoljavaju sve specifirane kriterije.

2. Adapter pattern

Adapter se može implementirati na dva načina:

1. klasom adaptera (class adapter) - klasa adaptera koristi nasljeđivanje (koje ćemo mi koristiti u našoj implementaciji adapter patterna ili
2. objektom adaptera (object adapter)- objekt adaptera koristi kompoziciju

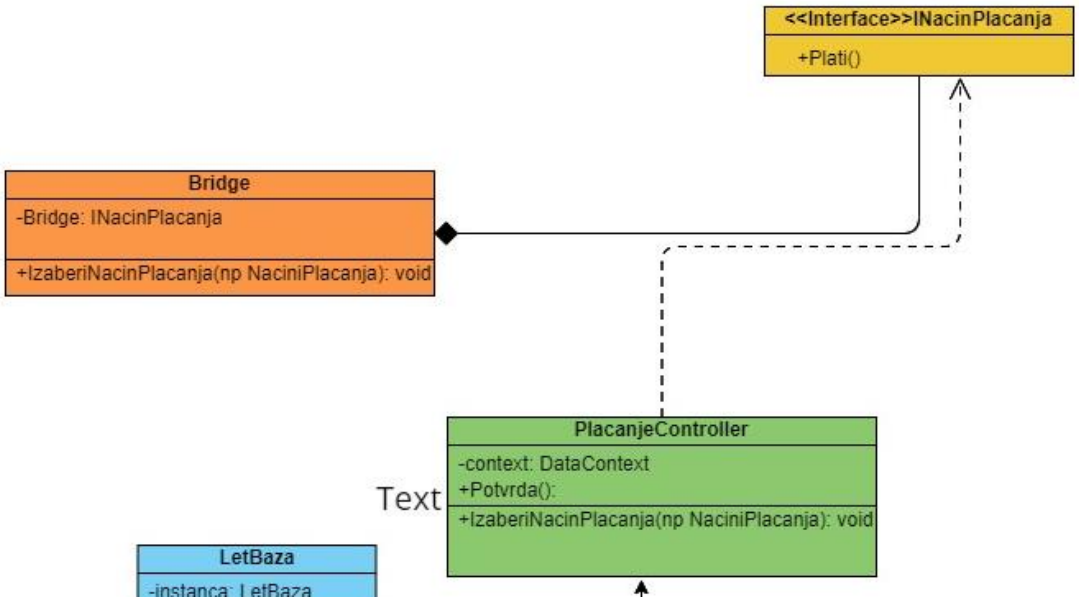
Adapter pattern moguće je iskoristiti na sljedeće načine:
- Integracija sa platnim gatewayima:
Naša web aplikacija podržava više platnih gatewaya (npr. PayPal, kartično, pri dolasku na aerodrom) svaki gateway može imati vlastiti interfejs za obradu plaćanja. Možemo stvoriti adapter klase koje implementiraju zajednički interfejs za plaćanje i prilagoditi specifična interfejs platnih gatewaya tom zajedničkom interfejsu.
- Obrada slika:
Naša web aplikacija može zahtijevati mogućnosti obrade slika(pasoša/lična karta), poput promjene veličine, rezanje(cut) kako bi prilagodili veličinu slike datom formatu. Različite biblioteke za obradu slika mogu imati vlastiti interfejs. Stvaranjem adaptera za obradu slika možemo zapakirati funkcionalnosti određene biblioteke za obradu slika i izložiti dosljedan interfejs koji naša web aplikacija može koristiti, bez obzira na glavnu biblioteku.

3. Bridge pattern

Bridge pattern se obično implementira kroz upotrebu dvije hijerarhije klasa - apstraktne klasa koja predstavlja apstrakciju i implementatorske klase koje predstavljaju implementaciju.

Bridge pattern moguće je iskoristiti na sljedeći način:
-Različiti načini plaćanja: Naša web aplikacija podržava različite načine plaćanja, poput kreditne kartice ili prilikom dolaska na aerodrom.

-Bridge pattern možemo primijeniti kako bi se odvojila apstrakcija načina plaćanja od konkretnih implementacija. Možemo imati različite implementacije bridge-a koje nasljeđuju klasu “NaciniPlacanja” i implementiraju specifične načine plaćanja.



4. Facade pattern

Facade pattern moguće je iskoristiti na sljedeći način:

-Obrada rezervacije letova:

Naša web aplikacija uključuje složen proces obrade rezervacije koji uključuje zadatke poput upravljanja raspoloživim letovima, obrade plaćanja i slanja e-mail-a kao potvrde o uspješnoj rezervaciji. Umjesto da izlažemo detalje i složenosti tih podsistema, možemo stvoriti fasadu koja obuhvata cijelu logiku obrade rezervacije letova. Fasada će pružiti pojednostavljeni interfejs koji klijenti mogu koristiti za moguću rezervaciju, a unutar sebe će upravljati svim potrebnim koracima obrade koordinacijom uključenih podsistema.

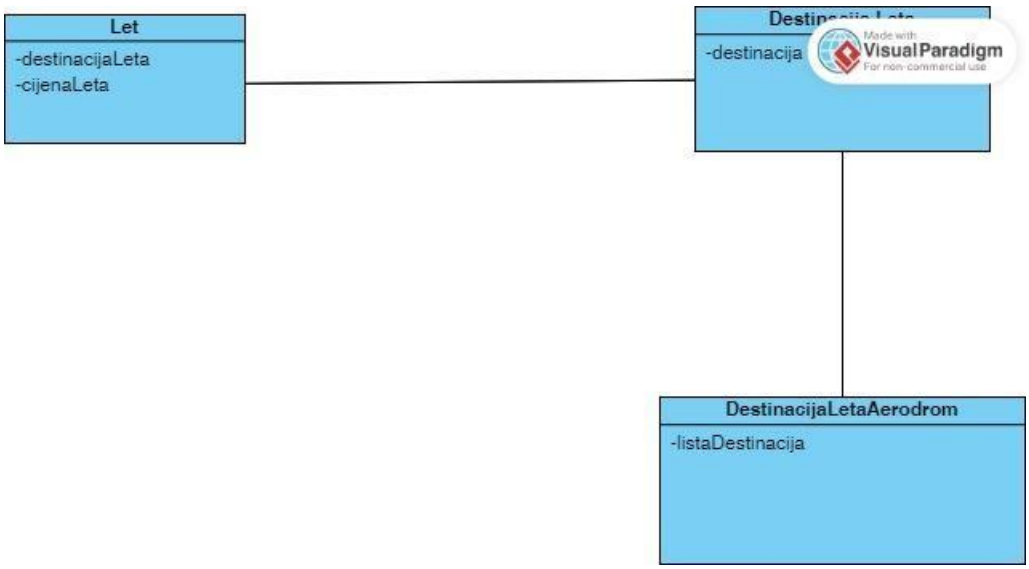
Klijenti vide samo fasadu, odnosno krajnji izgled objekta, dok je njegova unutrašnja struktura skrivena. Na ovaj način smanjuje se mogućnost pojavljivanja grešaka. U projektu, facade pattern se mogu koristiti kako bi se pojednostavila interakcija sa složenim podsistemom koji obuhvata rezervaciju karata, kupovinu karata itd. Na ovaj način, klijenti mogu koristiti jednostavne metode koje pruža fasada "*RezervisiKartu*" ,"*KupiKartu*" kako bi imali interakciju sa složenim podsistemom. Ovaj pattern se brine o detaljima implementacije da bi olakšao korištenje funkcionalnosti.

5. Flyweight pattern

Flyweight pattern moguće je iskoristiti na sljedeći način:

-Upravljanje destinacijama leta:

Naša web aplikacija ima različite destinacije leta. Flyweight pattern možemo koristiti kako bismo optimizirali upravljanje cijenama leta. Možemo stvoriti Flyweight objekte koji predstavljaju jedinstvene cijene, destinacije i njihove atribute. Ovi objekti se mogu dijeliti između različitih letova koji koriste istu destinaciju. Na taj način, umjesto da se za svaki let stvara nova instanca leta, koriste se već postojeći Flyweight objekt što smanjuje potrošnju memorije i olakšava upravljanje letovima.



6. Decorator pattern

Decorator pattern moguće je iskoristiti na sljedeće načine:
-Decorator pattern možemo koristiti kako bismo dodali dodatne karakteristike ili opcije za letove u našoj web aplikaciji. Možemo stvoriti decoratore za letove koji dodaju personalizirane opcije poput odabira težine prtljaga, prioritetno ukrcavanje kao i klasu u kojoj planira putovati. Dekoratori bi osnovnim objektima leta proširili njihove mogućnosti. -Prikaz dodatnih informacija: Decorator pattern možemo koristiti kako bismo stvorili decoratore koji pružaju informacije o destinaciji putovanja, njegovim karakteristikama ili preporukama (recenzijama) leta tj. uslugama koje se pružaju.

7. Proxy pattern

Proxy pattern je poželjan u aplikacijama koje rade s objektima koji su skupi za stvaranje ili koji se ne kreiraju često.v Također se može koristiti u aplikacijama koje trebaju uspostaviti sigurnost, (za bankovne transakcije ili za pristup bazi podataka).

Proxy pattern moguće je iskoristiti na sljedeći način:

-Zaštita pristupa:
Proxy pattern možemo koristiti kako bismo implementirali mehanizam zaštite pristupa određenim resursima. Administratorske funkcionalnosti su dostupne samo administratoru, Proxy može kontrolisati pristup tim funkcionalnostima. Proxy će provjeriti autentifikaciju i

ovlasti korisnika prije nego što omogući pristup administratorskim funkcijama. Ovo pomaže u zaštiti od neovlaštenog pristupa i održavanju sigurnosti u aplikaciji. Proxy object uspostavlja vezu između klijenta I stvarnog objekta.