STRUKTURALNI PATERNI

Facade pattern

Ovaj pattern omogućava korisnicima da koriste sistem bez potrebe da poznaju njegove unutrašnje detalje. Facade djeluje kao posrednik koji objedini više funkcionalnosti u jednu lako razumljivu tačku pristupa. Koristi se za smanjenje zavisnosti i povećanje čitljivosti i održavanja koda.
U našem sistemu, na dijagramu se Facade pattern koristi kroz klasu SortiranjeFasada, koja objedinjuje kompleksnu logiku pretrage pregleda karata (npr. prijasnjih i trenutnih) i pruža jednostavan interfejs metodama pretraziPoPrijasnjim() i pretraziPoTrenutnim().

Decorator pattern

Decorator pattern omogućava dinamičko dodavanje novih funkcionalnosti objektu bez mijenjanja njegove strukture. Koristi se kada želimo proširiti ponašanje objekta na fleksibilan i modularan način, bez korištenja nasljeđivanja.
Dekoratori obavijaju osnovni objekt i implementiraju isti interfejs, omogućavajući lančano dodavanje različitih funkcionalnosti.

U našem sistemu, decorator pattern je implementiran kroz KartaDekorator klasu koja nasljeđuje interfejs IRezervacija i omogućava dinamičko dodavanje dodataka poput kokica (DodatakKokice) i pića (DodatakPice) na kartu.

Adapter pattern

Adapter pattern omogućava da dvije klase koje inače ne mogu međusobno komunicirati rade zajedno. Funkcioniše tako što “prevodi” interfejs jedne klase u interfejs koji druga klasa očekuje. Koristan je kada želimo iskoristiti postojeću klasu, ali joj moramo prilagoditi interfejs da bi odgovarao sistemu.

U našem sistemu, FilmAdapter može pretvarati podatke iz vanjske filmske baze (npr. IMDb) u našu klasu Film, mapirajući atribute poput title u nazivFilma. Time omogućavamo integraciju sa spoljnim servisima bez mijenjanja postojećeg koda.

Bridge pattern

Bridge pattern razdvaja apstrakciju od implementacije kako bi se obje strane mogle nezavisno razvijati. Koristi se kada imamo više varijacija apstrakcije i implementacije.
Struktura uključuje apstraktnu klasu koja sadrži referencu na implementaciju, čime se postiže fleksibilnost.

U našem sistemu, bridge pattern možemo implementirati pomoću apstrakcije PrikazKarte sa implementacijama kao WebPrikaz, MobilniPrikaz, i posebno PDFPrikaz, dok implementacija sjedišta (TipSjedistaImplementacija) ostaje nezavisna.
Bridge pattern bi ti omogućio kombinovanje različitih prikaza sa različitim vrstama sjedišta bez kreiranja brojnih podklasa.

Proxy pattern

Proxy pattern koristi zamjenski objekat koji kontroliše pristup stvarnom objektu.
Može se koristiti za dodavanje dodatne logike poput keširanja, autentifikacije ili odgađanja inicijalizacije. Klijent komunicira s proxy objektom kao da je u pitanju pravi objekat.

U našem sistemu, proxy pattern možemo implementirati u scenariju kada ne želimo da svaki korisnik ima direktan pristup objektu rezervacija, jer bi mogli pokušati manipulisati cijenama, statusima ili rezervisati više karata nego što je dozvoljeno.

Klasa RezervacijaProxy bi implementirala isti interfejs kao i Rezervacija, te u sebi imala stvarni Rezervacija objekat. RezervacijaProxy bi kontrolisala pristup – npr. provjeravala da li korisnik ima pravo da rezerviše, da li ima dovoljno slobodnih mjesta, i sprečavala izmjene cijene karte bez dozvole.

Composite pattern

Composite pattern omogućava tretiranje pojedinačnih objekata i njihovih grupa (kompozicija) na isti način. Koristi se za hijerarhijske strukture gdje listovi i čvorovi dijele isti interfejs.
Kompozicija može sadržavati druge kompozicije, što omogućava rekurzivnu obradu strukture.

Composite pattern u ovom slučaju možemo primijeniti kroz interfejs IRezervacija koji implementiraju osnovna klasa Rezervacija i dekoratori kao KartaDekorator. Dekoratori (DodatakPice, DodatakKokice) omogućavaju dodavanje novih funkcionalnosti karti bez mijenjanja osnovne klase. Na taj način se korisniku može dinamički kreirati karta sa različitim dodacima kombinovanjem objekata.