Strukturalni dizajn paterni

Strukturalni paterni služe za organizaciju i raspodjelu odgovornosti među klasama, na način da se smanji kompleksnost veza, poveća fleksibilnost i olakša buduće proširenje sistema.

U kontekstu našeg sistema postoji više razloga zašto bi implementacija pojedinih dizajn paterna bila korisna. Oni uključuju: integraciju vanjskih servisa, modularno slaganje dodatnih troškova ili popusta nad artiklima, jednostavan poziv cijelog toka narudžbe i slično. Konkretan primjer za svaki dizajn patern je naveden u nastavku.

Adapter

Osnovna namjena ovog paterna je da omogući širu upotrebu već postojećih klasa. Ovo postaje korisno u situacijama kada je potreban drugačiji interfejs već postojeće klase, koju ne želimo mijenjati. Stoga se kreira nova adapter klasa koja služi kao posrednik između originalne klase i željenog interfejsa.

U okviru našeg sistema bi se ovaj patern mogao koristiti da se implementira kartično plaćanje. U slučaju da eksterni sistem za plaćanje koristi drugačiji interfejs od naše klase *Placanje*, bilo bi korisno primijeniti ovaj dizajn patern da se postojeća klasa prilagodi.

Façade

Facade patern pruža jedinstven interfejs na skupu interfejsa u podsistemu. On definiše interfejs na višem nivou koji olakšava upotrebu podsistema. Organiziranje sistema u podsisteme pomaže u smanjenju kompleksnosti; uobičajeni cilj je da se minimizira komunikacija i ovisnosti između podsistema.

Unutar našeg dijagrama klasa bi bilo naročito prikladno iskoristiti ovaj patern za čitav proces narudžbe; od dodavanja u korpu, do plaćanja, procesiranja QR koda, narudžbe i računa. Dakle, bilo bi moguće kreirati novu klasu koja pruža jedinstven interfejs za ovaj cjelokupan proces, i vrši: validaciju korpe, izračun cijene, primjenu popusta, plaćanje, generisanje QR koda, te kreiranje računa i narudžbe.

Decorator

Decorator patern služi da se omogući dinamičko dodavanje novih elemenata i ponašanja postojećim objektima. Pri tome, objekat „ne zna“ da je urađena dekoracija, što doprinosi iskoristivosti i mogućnosti ponovne upotrebe komponenti softverskog sistema. Idealan je ako se želi primijeniti slojevito poboljšanje operacija objekta.

Na ovaj način bi u našem sistemu bilo moguće dinamički „omotati“ objekat sa dodatnim ponašanjima bez promjene same klase. Primjera radi, ukoliko bi se naknadno odlučilo da je neophodno dodati neke artikle s više „ukrasa“ odnosno detalja, poput nekih dodatnih popusta specifičnih za te artikle, moguće bi bilo omotati klasu *Artikal* da se postigne tražena funkcionalnost bez promjene postojeće klase.

Bridge

Osnovna namjena ovog paterna je da se omogući odvajanje apstrakcije i implementacije neke klase, tako da ta klasa može posjedovati više različitih apstrakcija i više različitih implementacija za pojedine apstrakcije. Naročito je pogodan kada se implementira nova verzija softvera, a postojeća mora ostati u funkciji.

Njega bi bilo korisno primijeniti da se dodatno doprinese apstrakciji logike plaćanja od više konkretnih implementacija (kartično plaćanje ili neka druga opcija).

Composite

Svrha Composite paterna je da omogući formiranje strukture drveta pomoću klasa, u kojoj se individualni objekti i kompozicije individualnih objekata jednako tretiraju. Njegov zadatak je da omogući uniformni rad nad pojedinačnim komponentama i nad kompozicijom pojedinačnih elemenata različitih tipova.

U okviru našeg sistema bi bio koristan da se omogući grupisanje više artikala u jedan „paket“. Time bi se omogućilo da se pojedinačni artikli i grupe tretiraju isto. Ovo bi bilo idealno za neke promotivne pakete, kombinacije proizvoda i slično.

Proxy

Proxy je strukturalni patern koji odvaja interfejs od implementacije. Ideja paterna je da proxy objekat radi kao surogat objekt za stvarne objekte. On rješava probleme kada objekt ne može uvijek biti instanciran direktno, što se može desiti u slučaju restrikcije pristupa.

U okviru našeg sistema bi bilo pogodno koristiti Proxy patern za restrikciju pristupa klasi *Artikal*. Naime, objekat ove klase može mijenjati samo administrator ili uposlenik, a ostali korisnici ne mogu. Dakle, pogodno bi bilo odvojiti interfejs od stvarne implementacije.