Strukturalni patterni

Proxy:
Ovaj pattern se primjenjuje pri pretraživanju store-ova: Kada korisnik pretražuje merch store preko koda, tada on treba da dobije samo artikle koji se nalaze u datom store-u, dok kada administrator pretražuje, pored artikala, treba se ispisati i korisničko ime korisnika koji posjeduje store i adresa na koju je store povezan.

Facade:
Ovaj pattern se primjenjuje kod kreiranja merch store-a. Naime kada korisnik kreira merch store, on samo treba upisati adresu, dok se u pozadini vrši generisanje koda, validiranje adrese i dodavanje store-a u bazu.

Adapter:
Pretpostavimo da želimo dodati API koji će omogućiti validaciju adresa. Međutim taj api se neće moći direktno iskoristiti na objekte tipa Adresa. U toj situaciji bi mogli dodati adapter klasu sa kojom bi mogli omogućiti validiranje adrese pomoću dodanog API-ja.

Dekorater:
Ukoliko pretpostavimo da garderoba, umjesto da se dodaje slika za svaku boju zasebno, mijenja boju algoritamskim putem, onda bi pri dizajniranju garderobe mogli dodati dekorator jer bi se na istu garderobu morale primjeniti dvije promjene: promjena boje i dodavanje slike.

Bridge:
Pretpostavimo da dodamo novu klasu zvanu Moderator, koji će moći mijenjati status narudžbi ali neće moći terminirati korisnike i storeove. Tada možemo primjeniti ovaj pattern jer i administrator i moderator koriste istu metodu.

Composite:
Pretpostavimo da smo dodali novu klasu nazvanu VIPStore u kojem se obračunavaju popusti na svaki tip garderobe, te dodali novu mogućnost adminstratoru koji dozvoljava da se ispiše neto vrijednosti svih store-ova. Tada bi mogli iskoristiti Composite, tako da administrator jednom metodom može izračunati neto vrijednost svakog store-a bez obzira na tip store-a.

Flyweight:
Primjena kod klase Garderoba. Pri prikazivanju ove klase na korisničkom interfejsu, umjesto da kreiramo novu instancu ove klase za svaku boju garderobe, možemo samo mijenjati slike, a ostale atribute ne mijenjati.