STRUKTURALNI PATERNI

Adapter pattern

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 orginalne klase i željenog interfejsa.

Adapter patern bi mogli iskoristiti kada bi trenutni tip atributa Slika koji je sada string zamijenili tipom JPG ili PNG. Tada bi uz pomoć metoda konverzija Adapter klase omogućili prihvatanje raznih formata slike.

Facade pattern

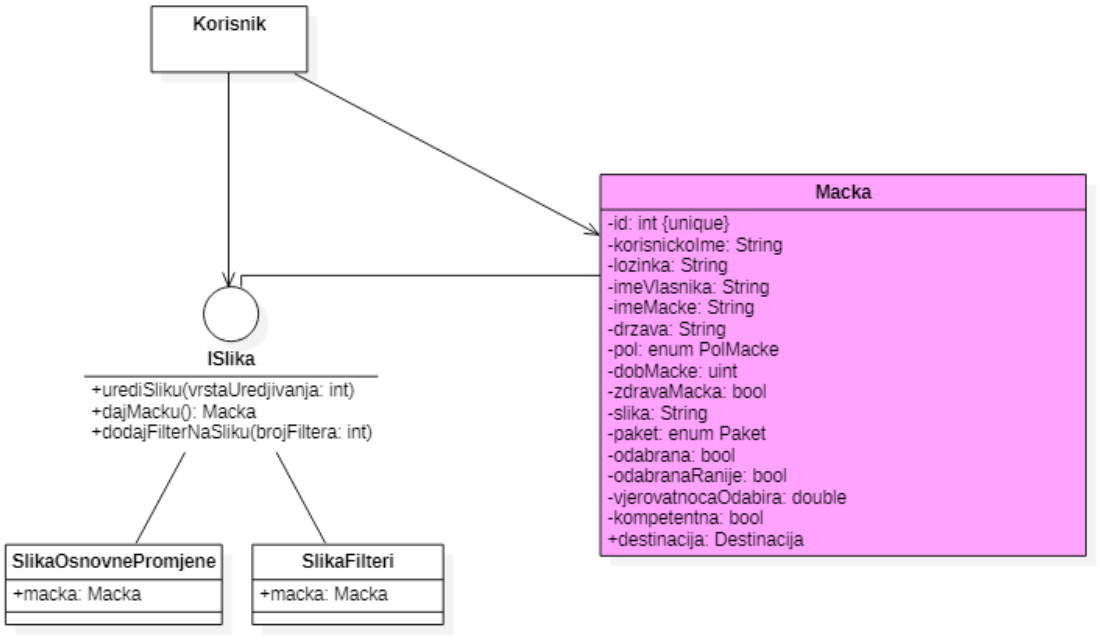
Kako mu i sam naziv govori, ovaj patern se korisiti s ciljem da osigura više pogleda visokog nivoa na podsisteme (implementacija podsistema skrivena od korisnika).

Facade patern bi mogli iskoristiti kada bi kreirali novu klasu PutovanjeFacade koja bi sadržavala metode koje bi od parametara imale samo id Mačke, ime, vjerovatnoću odabira i id putovanja.
Na taj način bismo mogli lakše pozvati metodu za odabir mačaka za putovanje dok bi se u pozadini klase dešavalo puno više.

Decorator pattern

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.

U našoj aplikaciji ćemo dodati mogućnost editovanja slika kao što su rotacija, rezanje i dodavanje filtera(npr. koji postavlja astronautsku kacigu na profilnu sliku mačke).



Bridge pattern

Osnovna namjena Bridge paterna je da omogući odvajanje apstrakcije i implementacije neke klase tako da ta klasa može posjedovati više različitih apstrakcija i više različitih implementacija za pojedine apstrakcije.

Ovaj patern bi mogao biti iskorišten ako bismo htjeli da realizujemo različit način prikazivanja destinacija. Naprimjer ako želimo da se free korisniku prikazuje samo uvećana lista free destinacija uz napomenu da je uz premium moguće odabrati puno bolje destinacije(kod nas je moguć pregled svih raspoređen u dvije liste s tim da su free korisniku premium destinacije zabranjene za odabir).

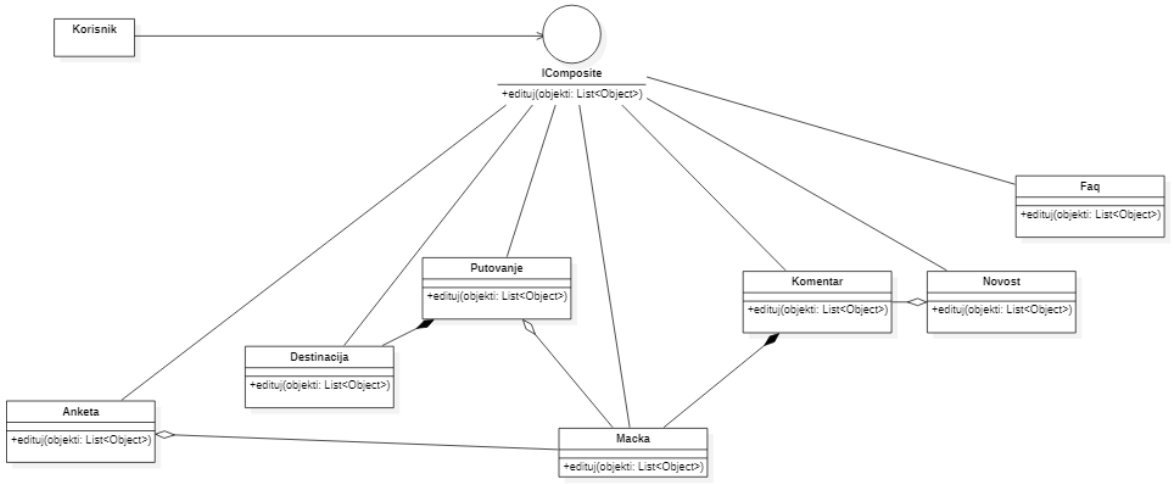
Composite pattern

Composite patern služi za kreiranje hijerarhije objekata. Koristi se kada svi objekti imaju različite implementacije nekih metoda, no potrebno im je svima pristupati na isti način, te se na taj način pojednostavljuje njihova implementacija.

Composite pattern predlaže da kroz zajednicki interfejs radimo sa svim objektima nad kojima vršimo slične akcije.

Umjesto kreiranja više metoda koje bi u bazi podataka zasebo ažurirale podatke za Faq, Macku, Novost i ostale klase, možemo napraviti interfejs IComposite sa metodom azuriraj().
Ta metoda se može koristi nad objektima različitih kompleksnosti.

Da bi implementirali Composite patter cijeli sistem trebamo predstaviti kao stablo. Ovaj uvjet je zadvoljen, jer sve klase u C# imaju zajedničku baznu klasu Object. Listovi stabla su klase koje ne mogu proslijediti naredbu na neku drugu klasu.



Kad se želi editovati vrijednost u nekoj klasi modela primamo kao parametar listu sa svim parametrima koji nam trebaju da bi se edit uspješno završio.

Prilikom pozivanja azuriraj() nad instancom klase Faq,
Faq vrši editovanje jer nema klasa kojima bi proslijedio naredbu.

Novosti mogu sadržavati komentare.
Prilikom editovanja instance klase Novosti sa metodom azuriraj(), Novost neće ažurirati komentar, već će komentaru biti proslijeđena naredba da sam pozove metodu azuriraj().

Proxy pattern

Proxy patern služi za dodatno osiguravanje objekata od pogrešne ili zlonamjerne upotrebe. Primjenom ovog paterna omogućava se kontrola pristupa objektima, te se onemogućava manipulacija objektima ukoliko neki uslov nije ispunjen, odnosno ukoliko korisnik nema prava pristupa traženom objektu.

Možemo napraviti proxy koji će zaštititi sva putovanja.
Proxy bismo mogli iskoristiti ako hoćemo da sami pošaljemo mačku u svemir sa naredbom Macka.posljaljiUSvemir(). Međutim tako bi mogao svaki korisnik pokrenuti putovanje. Da bi se od toga ograničili, mi ćemo napraviti proxy koji će to onemogućiti na način da se mora proslijediti neki password koji samo admin zna ili
sam admin objekat da bi se moglo to putovanje pokrenuti.

Flyweight pattern

Flyweight patern se koristi kako bi se onemogućilo bespotrebno stvaranje velikog broja instanci objekata koji svi u suštini predstavljaju jedan objekat. Samo ukoliko postoji potreba za kreiranjem specifičnog objekta sa jedinstvenim karakteristikama (tzv. specifično stanje), vrši se njegova instantacija, dok se u svim ostalim slučajevima koristi postojeća opća instance objekta (tzv. bezlično stanje).

Sve klase pretežno imaju različite vrijednosti atributa, pa samim tim ne možemo implementirati flyweight pattern.

Kada bi smo npr. u klasi Putovanje imali atribute za raketu, droneship za slijetanje, i količinu goriva koju raketa može da drži, ovi atributi bi uvijek imali istu vrijednost, jer SpaceCat koristi istu raketu za sve letove.

Uštedjeli bi na memoriji kada bi kreirali instancu nove klase Raketa i u njoj čuvali vrijednosti ovih atributa, za razliku od njihovog pohranjivanja u svakoj instanci klase Putovanje. Na ovaj način bi uštedjeli na memorijskim resursima.