Paterni ponašanja

1. Strategy pattern

Strategy pattern moguće je iskoristiti na sljedeće načine:

Strategy za preporuku proizvoda:
Ukoliko bi naša web aplikacija pružala funkcionalnost preporuke proizvoda korisnicima, Strategy pattern može biti koristan. Možemo imati interfejs PreporukaStrategija s metodama poput dajPreporuceneProizvode() ili filtrirajPreporuceneProizvode(). Zatim možemo implementirati konkretne strategije kao PreporukaNaOsnovuHistorije, PreporukaNaOsnovuInteresa ili PreporukaNaOsnovuSlicnosti, koje će implementirati metode preporuke na odgovarajući način. Naša web aplikacija može dinamički odabrati i primijeniti odgovarajuću strategiju preporuke proizvoda na temelju podataka o korisniku ili drugih faktora.
Strategy za sortiranje proizvoda:
Naša web aplikacija omogućava korisnicima da sortiraju proizvode prema različitim kriterijima (npr. po cijeni, popularnosti, ocjeni), možemo koristiti Strategy pattern. Možemo definisati interfejs SortiranjeStrategija s metodom poput sortirajProizvode() koja će sortirati listu proizvoda na temelju odabranog kriterija. Zatim možemo implementirati konkretne strategije kao SortiranjePoCijeni, SortiranjePoPopularnosti ili SortiranjePoOcjeni, koje će implementirati metodu sortirajProizvode() na odgovarajući način. Naša web aplikacija može dinamički odabrati i primijeniti odgovarajuću strategiju sortiranja proizvoda na temelju korisničkog odabira.

2. State pattern

State pattern moguće je iskoristiti na sljedeće načine:

Stanja narudžbe:
Naša web aplikacija pruža informaciju o stanju narudžbe kao što su "u pripremi", "u dostavi" ili "isporučeno", te možemo koristiti State pattern. Možemo definisati interfejs StanjeNarudzbe koje će imati metode poput potvrdi(), prekini() ili dostavi(). Zatim možemo implementirati konkretne klase za svako stanje kao NarudzbaUPripremi, NarudzbaUDostavi ili NarudzbaIsporucena, koje će implementirati metode stanja na odgovarajući način. Objekat narudžbe će imati referencu na trenutno stanje, a metode stanja će mijenjati njegovo ponašanje ovisno o stanju narudžbe.
Stanje korpe:
Naša web aplikacija pruža informaciju o stanju korpe kao što su "prazna", "popunjena" ili "potvrđena", te možemo koristiti State pattern. Možemo definirati interfejs StanjeKorpe s metodama poput dodajProizvod(), ukloniProizvod() ili potvrdiKorpu(). Zatim možemo implementirati konkretne klase za svako stanje kao PraznaKosarica, PopunjenaKorpa ili PotvrdenaKorpa, koje će implementirati metode stanja na odgovarajući način. Objekat korpe će imati referencu na trenutno stanje, a metode stanja će ograničavati ili omogućavati manipulaciju s proizvodima u korpi ovisno o stanju.

3. Template method

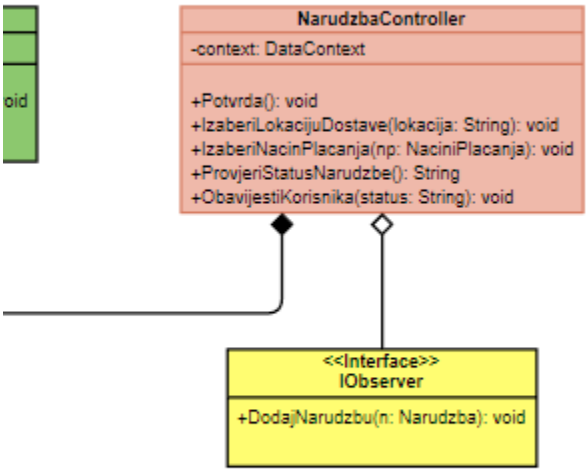
Template method moguće je iskoristiti na sljedeće načine:

Template za prikaz proizvoda:
Naša web aplikacije za različite vrste proizvoda (npr. majice, hlače, cipele) koje trebamo prikazati, te možemo koristiti Template method. Možemo definisati apstraktnu klasu ProizvodTemplate koja će sadržavati osnovnu strukturu i metode za prikaz proizvoda, poput prikaziSliku(), prikaziNaziv() ili prikaziCijenu(). Zatim možemo implementirati konkretne klase za svaku vrstu proizvoda koje će nasljeđivati ProizvodTemplate i pružiti specifične implementacije metoda prema vrsti proizvoda. Na taj način ćemo imati zajednički skellet za prikaz proizvoda, ali će se specifični detalji prikaza razlikovati za svaku vrstu proizvoda.
Template za generiranje e-mail poruka:
Ukoliko bi naša web aplikacija trebala slati različite vrste e-mail poruka (npr. potvrda narudžbe, obavijest o dostavi, reklamne poruke), možemo koristiti Template method. Možemo definisati apstraktnu klasu EmailTemplate koja će sadržavati osnovnu strukturu i metode za generiranje e-mail poruka, poput generirajNaslov(), generirajSadržaj() ili generirajPotpis(). Zatim možemo implementirati konkretne klase za svaku vrstu e-mail poruke koje će nasljeđivati EmailTemplate i pružiti specifične implementacije metoda prema vrsti poruke. Na taj način ćemo imati zajednički template za generiranje e-mail poruka, ali će se specifični detalji poruke razlikovati za svaku vrstu.

4. Observer

Observer je moguće iskoristiti na sljedeće načine:

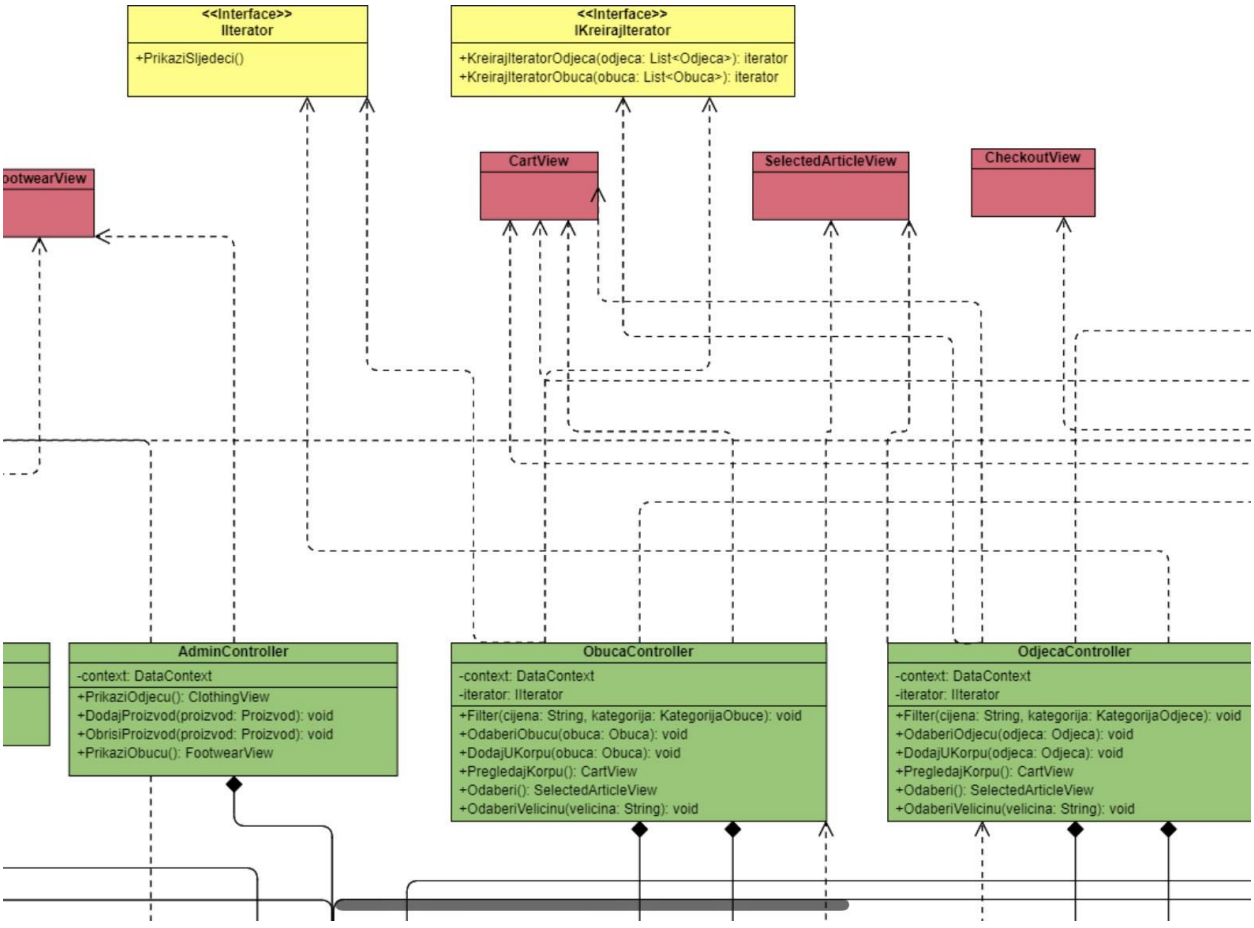
Obavještenje o raspoloživosti proizvoda:
Kada se stanje raspoloživosti određenog proizvoda promijeni, možemo koristiti Observer pattern kako bi obavijestio sve korisnike koji su se pretplatili na obavijesti o tom proizvodu. Svaki korisnik može biti promatrač koji je pretplaćen na subjekt (proizvod) i kada se raspoloživost promijeni, subjekt će automatski obavijestiti sve promatrače. Korisnici će tako dobiti ažuriranje o promjeni raspoloživosti proizvoda.
Narudžba i dostava:
Kada korisnik napravi narudžbu i odabere opciju dostave, možemo koristiti Observer pattern kako bi obavijestio korisnika o statusu njegove narudžbe i ažuriranju dostave. Korisnik može biti promatrač koji je pretplaćen na subjekt (narudžba) i kada se status narudžbe ili dostave promijeni, subjekt će automatski obavijestiti promatrača. Korisnik će tako biti obaviješten o trenutnom statusu svoje narudžbe i mogućim promjenama u dostavi.
Prijavljivanje na novosti:
Ukoliko želimo omogućiti korisnicima da se pretplate na primanje novosti i najnovijih vijesti iz svijeta mode, mogli bismo koristiti Observer pattern. Svaki korisnik može biti promatrač koji je pretplaćen na subjekt (novosti) i kada se pojave nove vijesti, subjekt će automatski obavijestiti sve promatrače. Korisnici će tako biti obaviješteni o najnovijim trendovima, kolekcijama ili događanjima iz svijeta mode.



5. Iterator

Iterator je moguće iskoristiti na sljedeće načine:

Pretraživanje proizvoda:
Ako korisnik ima potrebu pretraživati proizvode na temelju određenih kriterija, Iterator pattern može biti koristan za iteriranje kroz rezultate pretrage. Možemo imati interfejs Iterator koji definira metode za provjeru ima li sljedeći element i pristup sljedećem elementu, a konkretna implementacija Iteratora može raditi s rezultatima pretrage i pružiti metode za pristup i iteriranje kroz te rezultate. Na taj način možemo omogućiti korisniku pretraživanje proizvoda i prikaz rezultata jedan po jedan.
Prikaz korisnika:
Možemo koristiti Iterator pattern za iteriranje kroz kolekciju korisnika u našoj web aplikaciji, kao i prikaz svih korisnika ili specifičnih grupa korisnika. Na primjer, možemo imati kolekciju korisnika koji su se pretplatili na newsletter, kolekciju korisnika s određenim privilegijama ili kolekciju korisnika s posebnim preferencijama. Iterator pattern omogućuje da se korisnicima pristupi jedan po jedan na jednostavan i fleksibilan način.



6. Chain of responsibility

Chain of responsibility je moguće iskoristiti na sljedeći način:

Obrada narudžbi:
Kada korisnik napravi narudžbu, možemo koristiti Chain of Responsibility pattern za obradu te narudžbe kroz različite korake. Svaki korak u lancu može predstavljati određenu vrstu obrade, poput provjere dostupnosti proizvoda, provjere količine na stanju, provjere podataka o plaćanju i sl. Svaki objekat u lancu ima svoju logiku za obradu određenog koraka, a ako ne može obraditi narudžbu, prenosi je sljedećem objektu u lancu. Na taj način možemo uspostaviti fleksibilan i konfigurabilan sistem za obradu narudžbi, koji se može lako prilagoditi promjenama ili dodavanju novih koraka.

7. Mediator

Mediator je moguće iskoristiti na sljedeće načine:

Obrada narudžbi:
Kada korisnik napravi narudžbu, Mediator pattern može poslužiti kao objekat posrednika koji će koordinirati obradu te narudžbe između različitih komponenti ili podsistema u tvom sistemu. Na primjer, posrednik može obavijestiti skladište o novoj narudžbi, obračunati plaćanje, poslati obavijest kupcu itd. Ovo omogućuje centraliziranu kontrolu i koordinaciju obrade narudžbi.
Promjena stanja proizvoda:
Naša web aplikacija omogućava korisnicima promjenu stanja proizvoda, odnosno dodavanja u korpu. Mediator pattern može poslužiti kao objekat posrednika koji će upravljati tim promjenama. Na primjer, kada korisnik doda proizvod u korpu, posrednik će obavijestiti relevantne komponente ili objekte u sistemu, kao što su preporuke ili personalizacija. Ovo omogućuje centraliziranu kontrolu i koordinaciju promjena stanja proizvoda.