Strategy pattern

Implementiramo Strategy pattern kako bismo izdvojili različite načine obrade kupovine (sa rezervacijom i bez nje) u zasebne klase koje se mogu lako mijenjati i proširivati. Klasa Kupovina je dobar kandidat jer sadrži podatke o kupovini, ali sama logika obrade zavisi od tipa kupovine, što znači da nije njena odgovornost da zna kako se tačno obrada vrši. Na kraju, kreiramo interfejs IKupovinaStrategija sa metodom obradiKupovinu(), a klase KupovinaSaRezervacijom i KupovinaBezRezervacije ga implementiraju. Kontekst (npr. KupovinaObradjivac) dinamički postavlja strategiju i poziva njenu obradu – što čini sistem fleksibilnijim i preglednijim.

State pattern

Da bi se realizovao State pattern, potrebno je uvesti klase poput StanjeKreirano, StanjePlaceno i StanjeOtkazano koje implementiraju zajednički interfejs (npr. KupovinaState). Svako stanje definiše kako Kupovina treba da se ponaša u tom trenutku. Na primjer, dok je Kupovina u stanju "kreirano", možeš je obraditi ili otkazati. Kada je jednom plaćena, prelazi u stanje "plaćeno" i tada više ne možeš otkazati. Dakle, stanje mijenja dozvoljene radnje nad klasom Kupovina, umjesto da Kupovina sama sadrži sve if-else uslove, određivanje ponašanja se prenosi na objekat koji predstavlja njeno trenutno stanje.

Template Method pattern

Što svi tipovi kupovine imaju određene ustaljene korake, poput izračuna cijene i potvrde kupovine, dok se neki koraci razlikuju (npr. provjera dostupnosti i način plaćanja), primjenjujemo Template Method pattern koristeći interfejs. Klasa Kupovina sadrži metodu izvrsiKupovinu() koja definiše kompletan tok kupovine kroz fiksni redoslijed koraka. Promjenjivi koraci se ne implementiraju direktno u toj klasi, već se delegiraju interfejsu (npr. IKupovinaKoraci). Klase kao što su KupovinaSaRezervacijom i KupovinaBezRezervacije implementiraju ovaj interfejs i definišu konkretna ponašanja za te korake. Na taj način, zadržavamo jedinstven šablon toka u klasi Kupovina, ali omogućavamo fleksibilnost u načinu na koji se pojedini koraci izvode, zavisno od vrste kupovine

Observer pattern

Observer pattern se može primijeniti u situaciji kada korisnik označi da mu se određeni događaj dopada. U tom trenutku, događaj postaje centralni objekat koji

obavještava sve registrovane posmatrače da je označen kao interesantan. Jedan od posmatrača može biti sistem za čuvanje bookmarka, koji automatski dodaje taj događaj u korisnikovu listu sačuvanih događaja.

Iterator pattern

Pošto na početnom ekranu aplikacije prikazujemo listu događaja kroz koje se korisnik može kretati (npr. listanje, filtriranje), primjenjujemo Iterator pattern. Umjesto da direktno pristupamo strukturi podataka, koristi se iterator koji omogućava prolazak kroz događaje jedan po jedan. Na taj način se postiže odvajanje logike kretanja od načina pohrane.

Command pattern

Pošto želimo omogućiti da se dodavanje događaja u bookmark izvrši na jasan i odvojen način, primjenjujemo Command pattern. Kreiramo interfejs Komanda sa metodom izvrsi(), a konkretna klasa DodajBookmarkKomanda implementira tu metodu i zna kako da doda događaj u bookmark tako što poziva odgovarajuću metodu nad objektom BookmarkSistem (receiver). Izvršavanje komande preuzima klasa Izvrsilac, koja poziva izvrsi() bez znanja o njenoj unutrašnjoj logici. Na ovaj način se postiže odvajanje zahtjeva za dodavanje u bookmark od same njegove realizacije.