**Strategy patern**

Strategy patern služi kako bi se različite implementacije istog algoritma izdvojile u različite klase, te kako bi se omogućila brza i jednostavna promjena implementacije koja se želi koristiti u bilo kojem trenutku. Na ovaj način omogućava se i jednostavno brisanje ili dodavanje novog algoritma koji se može koristiti po želji. Ovaj patern ćemo implementirati. Služi nam za sortiranje hotela po različitim parametrima. Moramo dodati apstraktnu klasu Sortiranje sa listom hotela kao privatnim atributom, ona će imati metodu templateMethod(), te ćemo imati i metode koje će biti implementirane u izvedenim klasama tipa kriterijPoredjenja(). Imat ćemo izvedene klase pod nazivima AbecedaSort i FilterSort koje će imati svoju metodu kriterijPoredjenja().

Diagram

Description automatically generated

**Iterator patern**

Iterator patern namijenjen je kako bi se omogućio prolazak kroz listu elemenata bez da je neophodno poznavati implementacijske detalje strukture u kojoj se čuvaju elementi liste. Izvedba liste može biti u obliku stabla, jednostruke liste, niza i sl., no klijentu se omogućava da na jednostavan način dolazi do željenih elemenata. Osim toga, ovaj patern preporučljivo je iskoristiti kada se za iteriranje koristi kompleksna logika koja ovisi o više kriterija. Ovaj patern smo odlučili da implementiramo. Odlučili samo napraviti da pristupamo elementima kolekcije soba u klasi Hotel preko iteratora. Da bi ovo postigli morali bi napraviti interfejs IKreatorIterator, koja kreira iterator, tj posjeduje metodu createIterator(), zatim bismo kreirali interfejs Iterator koji posjeduje dvije metode hasNext() i getNext(). Morali bismo i kreirati klasu SobaIterator, koja kao atribute ima listu soba i atribut koji govori o trenutnom indeksu, ova klasa bi implementirala interfejs Iterator. Također bi u klasu Hotel morali dodati još jedan atribut tipa Iterator, dok nam lista soba u klasi Hotel ne bi bila potrebna.

Diagram

Description automatically generated

**Template Method patern**

Template method patern služi za omogućavanje izmjene ponašanja u jednom ili više dijelova. Najčešće se primjenjuje kada se za neki kompleksni algoritam uvijek trebaju izvršiti isti koraci, no pojedinačne korake moguće je izvršiti na različite načine. Ovaj patern nismo implementirali. Međutim kada bi željeli implementirati ovaj patern to bi bilo kod plaćanja rezervacije, tačnije kada bi omogućili drugačije načine plaćanja rezervisanja sobe. Dakle dodali bi interfejs IPlacanje, koji bi imao jednu metodu placanjeSobe, i napravili bi nove klase koje implementiraju ovaj interfejs, npr. klasa za plaćanje karticom, te klasa za plaćanje u kešu.

**State patern**

State patern omogućava objektu da mijenja svoja stanja, od kojih zavisi njegovo ponašanje. Sa promjenom stanja objekt se počinje ponašati kao da je promijenio klasu. Stanja se ne mijenjaju po želji klijenta, već automatski, kada se za to steknu uslovi. Nismo implementirali ovaj patern. Imamo priliku za implementaciju ovog paterna kod rezervisanja sobe, tj kod interfejsa IStanje, koji bi imao metode rezervisiSobu, i otkaziRezervaciju, te bi dodali IStanje interfejs kao atribut klase Soba. Također bi dodali i dvije klase koje bi implementirale interfejs IStanje, Rezervisana i NijeRezervisana, ove klase bi opisivale moguća stanja za pojedinu Sobu. Funkcionisalo bi na način da kada je soba rezervisana, u klasi Rezervisana bi metodom otkaziRezervaciju mijenjali stanje karte u NijeRezervisana. Na isti način bi metodom rezervisiKartu u klasi NijeRezervisana, mijenjali stanje karte u Rezervisana.

**Observer patern**

Uloga Observer paterna je da uspostavi relaciju između objekata tako kada jedan objekat promijeni stanje drugi zainteresirani objekti se obavještavaju. Ovaj patern bi se mogao iskoristiti kada neka od rezervacija biva odobrena. Klasa Rezervacija bi implementirala Interfejs IRezervacijaObserver, koji bi imao metodu NaOdobrenuRezervaciju i pomoću nje bi dodao rezervaciju u bazu.

**Chain of responsibility patern**

Chain of responsibility patern predstavlja listu objekata, ukoliko objekat ne može da odgovori prosljeđuje zahtjev narednom u nizu. U našem sistemu, ovaj patern bi mogli iskoristiti kada korisnik krene napraviti zahtjev za rezervaciju (kroz klasu ZahtjevZaRezervaciju). Tu bi imali jednu klasu KreiranjeRezervacije sa metodom kreiraj() koja bi prvo pozvala klasu ZahjtevZaRezervaciju koja bi kreirala jedan zahtjev a onda bi se po odobravanju istog zahtjeva, napravila i instanca klase Rezervacija, te bi se konačno ta rezervacija uspješno izvršila.

**Medijator patern**

Medijator patern omogućava reduciranje ovisnosti izmedu objekata. Ovaj patern ograničava direktnu komunikaciju izmedu objekata i forsira ih da kolaboriraju samo preko medijator objekta. Umjesto za direktno povezujemo veliki broj objekata, koristimo medijator, koji je zadužen za njihovu komunikaciju. Kada neki objekat želi poslati poruku drugom objektu, on šalje poruku medijatoru, a medijator zatim prosljeđuje tu poruku drugom objektu. Ovaj patern bi mogli implementirati u vidu komentara ili chat-a za svaki hotel, na način da će s ekorisnici dijeliti na: NeregistrovaniGost, RegistrovaniGost, Radnik i Administrator. Sve poruke se šalju AskRoom klasi koja bi imala metodu showMessage, koja će primati pošiljaoca i poruku koju šalje. Administratori bi mogli brisati poruke ako su one neprimjerenog sadržaja.