Kreacijski paterni

1. Singleton patern

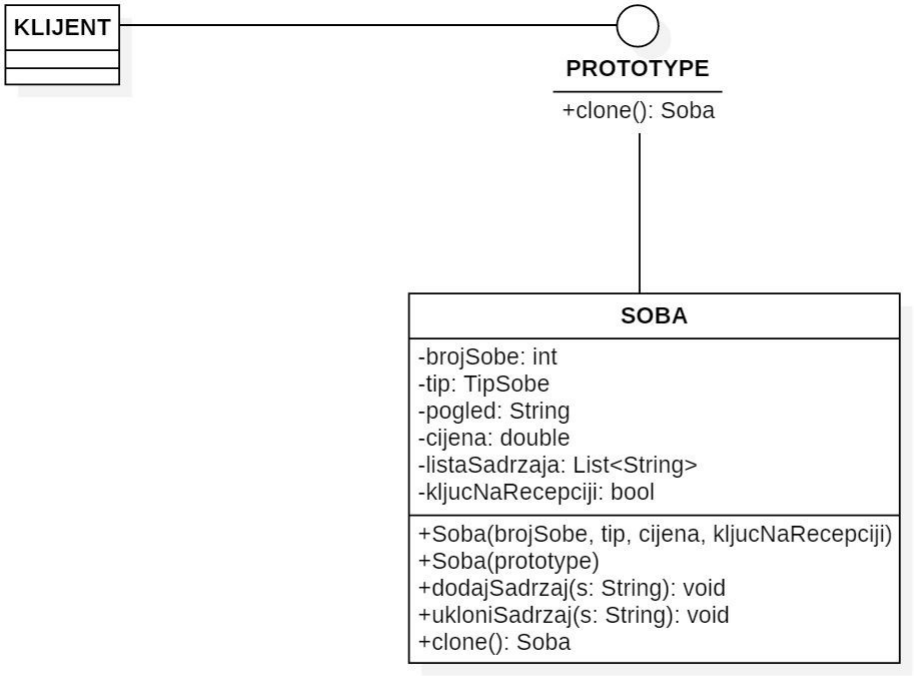
Singleton patern osigurava da se klasa može instancirati samo jednom, te osigurava globalni pristup toj instanci.

Ovaj patern bi se, uz dodatne izmjene, u našem sistemu mogao vezati za klasu *Hotel,* glavnu kontejnersku klasu. Ono što bismo trebali uraditi je napraviti MVC model klasa. Dakle, trebali bismo dopuniti naš sistem sa klasom kontrolera u kojoj bi se kreirao singleton objekat, te vraćao taj isti ako bi se pokušao ponovo kreirati.

2. Prototype patern

Uloga Prototype paterna je da kreira nove objekte klonirajući jednu od postojećih prototip instanci. Patern podrazumijeva i zajednički interfejs za sve objekte koji podržavaju kloniranje.

U našem sistemu pogodna klasa za ovaj pattern je klasa Soba te smo ga iz tog razloga i primijenili. Klasa Soba ima atribute: brojSobe, tip, pogled, cijena, listaSadržaja i ključNaRecepciji. U našem sistemu je sasvim moguće koristiti sobe koje imaju isti pogled, cijenu, tip, itd., a različit broj. Uz pomoć Prototype paterna moguće je klonirati sve instance klase Soba po istim atributima, a zatim naknadno promijeniti atribute koji bi se eventualno razlikovali.



3. Factory Method patern

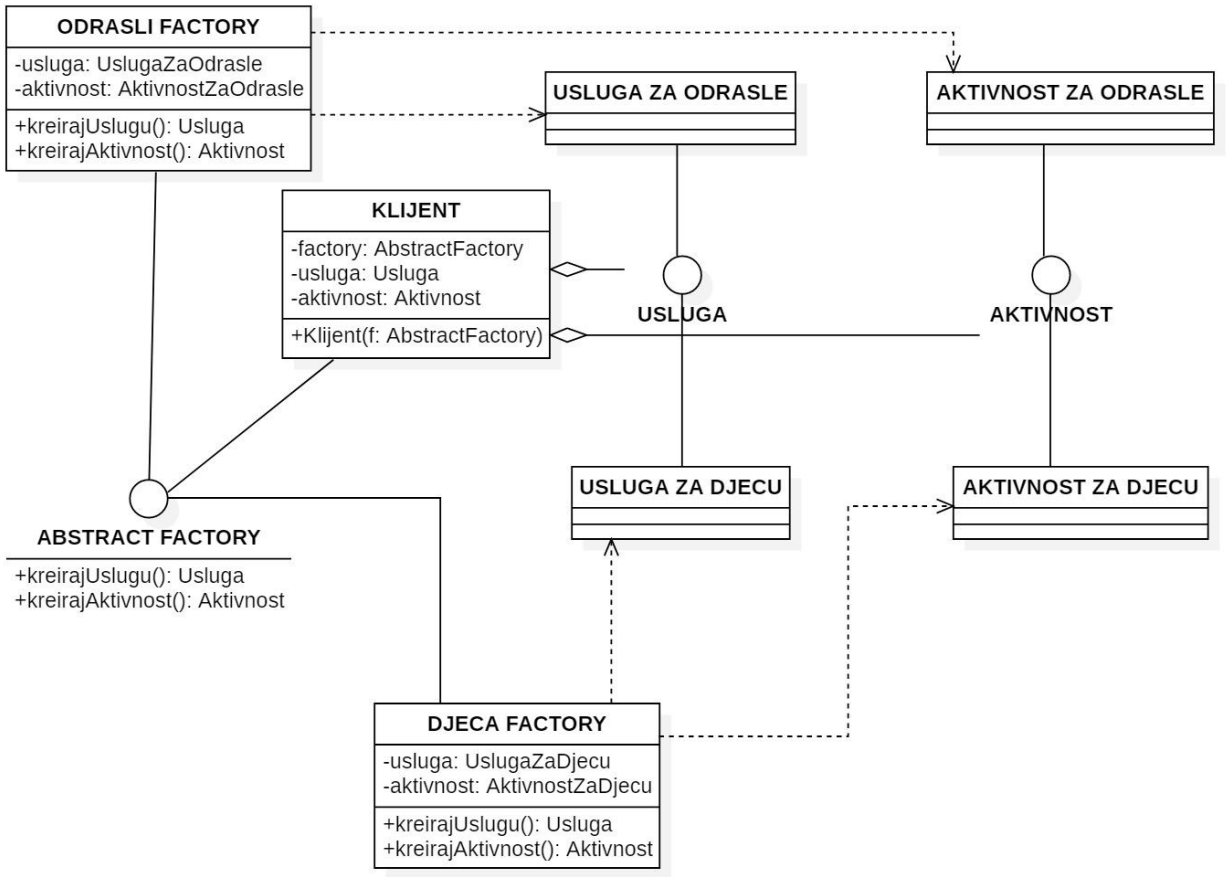
Uloga Factory Method paterna je da se u toku izvršavanja programa odluči koja će se klasa instancirati.

Ovaj patern nismo iskorisitili u našem projektu. Bilo bi ga moguće primijeniti kada bismo imali više nasljeđivanja u sistemu, npr. kada bismo različite tipove soba predstavljali preko izvedenih klasa. U tom slučaju, svaka od podklasa koja može instancirati neku od klasa za tipove sobe, implementirala bi factory metodu koja bi zavisno od toga koju opciju je izabrao korisnik, u toku izvršavanja aplikacije, instancirala odgovarajuću klasu.

4. Abstract Factory patern

Abstract factory patern se koristi kada radimo sa različitim familjama povezanih objekata.

Ovaj patern smo iskoristili u našem projektu. Cijena aktivnosti i usluga koje gosti biraju u toku boravka u hotelu zavisi od uzrasta gosta, pa smo aktivnosti podijelili na aktivnosti za djecu i odrasle, te isto to uradili i sa uslugama. Uveli smo dva nova interfejsa, i četiri nove klase. Interfejs *Usluga* implementiraju klase *UslugaZaOdrasle* i *UslugaZaDjecu*, a interfejs *Aktivnost* implementiraju klase, *AktivnostZaOdrasle* i *AktivnostZaDjecu*. Na osnovu uzrasta gosta, kreirane su i fabrike *DjecaFactory* i *OdrasliFactory* koje objedinjuju "proizvode" *UslugaZaDjecu* i *AktivnostZaDjecu*, odnosno *UslugaZaOdrasle* i *AktivnostZaOdrasle.*



5. Builder patern

Builder patern omogućava da se konstrukcija kompleksnog objekta odvoji od reprezentacije objekta.

Ovaj patern nismo iskorisitili u našem projektu. Uz određene izmjene, mogao bi se primijeniti na klasu *Soba*. Svaka soba ima različite sadržaje koje smo predstavili pomoću liste stringova. Međutim, ukoliko bismo elemente liste razbili u posebne Boolean atribute, te ukoliko bismo u sistemu upravljali i procesom izgradnje sobe, patern bi se mogao koristiti. Tada bismo imali interfejs *IBuilder* kojeg bi implementirala dva buildera, builder za početnu izgradnju sobe i builder za dodavanje sadržaja. Interfejs *IBuilder* bi imao metode kao što su *dodajBalkon()*, *dodajKrevet()*, *dodajProzor()* (koje bi pozivao builder za početnu izgradnju), metode kao što su *dodajJacuzzi()*, *dodajMiniBar()* (koje bi pozivao drugi builder), te metodu *dajSobu()* koja bi vraćala izgrađenu sobu. Oba buildera bi imala atribut tipa *Soba*.