KREACIJSKI PATERNI

Kreacijski paterni propagiraju dizajn sistema kojim se odvaja korištenje objekata od njihovog kreiranja. Kreacijski paterni enkapsuliraju znanje o tome koje klase sistem koristi, ali skrivaju detalje o tome kako se instance klasa kreiraju. Kreacijski paterni su dizajnirani da prekinu usku povezanost među klasama (npr. nasljeđivanje). U grupu kreacijskih paterna se ubraja sljedećih 5 paterna:

1. Singleton 2. Prototype 3. Factory Method 4. Abstract Factory 5. Builder

1. Singleton pattern

Uloga Singleton paterna je da osigura da se klasa može instancirati samo jednom i da osigura globalni pristup kreiranoj instanci klase. Postoji više objekata koje je potrebno samo jednom instancirati i nad kojim je potrebna jedinstvena kontrola pristupa. Neki od njih su: thread pools (skupina niti), objekti koji upravljaju setovanjem registara, objekti koji se koriste za logiranje, objekti koji se koriste kao drajveri za razne uređaje kao što su printeri i grafičke kartice. Instanciranje više nego jednom navedenih objekata mogu se prouzrokovati problemi kao što su nekorektno ponašanje programa, neadekvantno korištenje resursa ili nekonzistentan rezultat. Najčešće se koristi za uspostavljanje jedinstvene konekcije na bazu podataka, te ubrzava pristup korištenjem keš memorije.

Primjena Singleton paterna u sistemu:

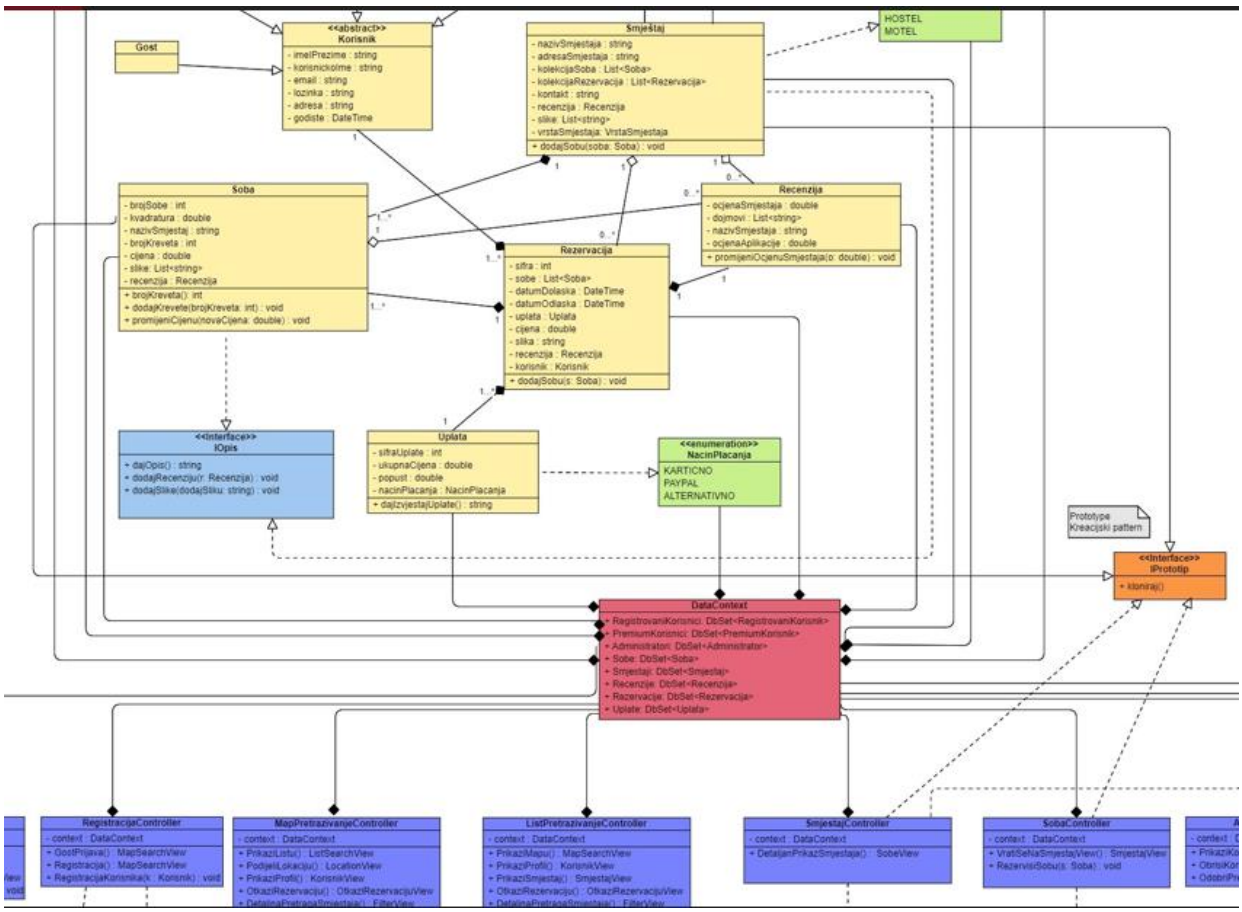
- Naš sistem omogućava korisnicima pretraživanje smještaja i pregled ponuda. Pristup tim ponudama se vrši putem pristupa serverima koji su zaduženi za pohranu i dobavljanje materijala putem internet konekcije. S obzirom da je za brzu, efikasnu i pouzdanu uslugu neophodno koristiti više nezavisnih servera, proširujemo sistem s klasom ServerBalanser koja će biti singleton. Njen zadatak je da upravlja svim raspoloživim serverima i da za određeni zadatak, kao što je dobavljanje slika, izabere optimalno rješenje u smislu brzine dostavljanja podataka i opterećenja sistema.
Da bi se ispunilo singleton pravilo ServerBalanser mora imati privatni statički atribut koji je tipa ServerBalanser, privatni konstruktor i javnu pristupnu metodu odnosno getter.

2. Prototype pattern (primijenjen u sistemu)

Uloga Prototype paterna je da kreira nove objekte klonirajući jednu od postojećih prototip instanci (postojeći objekat). Ako je trošak kreiranja novog objekta velik i kreiranje objekta je resursno zahtjevno tada se vrši kloniranje već postojećeg objekata. Prototype dizajn patern dozvoljava da se kreiraju prilagođeni objekti bez poznavanja njihove klase ili detalja kako je objekat kreiran.Prototype patern se koristi kada je potrebno da se sakriju konkretne klase od klijenta, dodaju ili izbrišu nove klase za vrijeme izvršavanja, da se broj klasa u sistemu održi na minimumu, kada je potrebna promjena strukture podataka za vrijeme izvršavanja. Composite i Decorator paterni često imaju prednosti od Prototype paterna. Prototype patern je često koristan i prilikom višestrukog korištenja podataka iz baze. Ako je potrebno uraditi i druge analize nad istim skupom podataka nije dobra ideja ponovo pristupati bazi podataka, čitati podatke i kreirati objekat za njihovo smještanje.

Primjena Prototype paterna u sistemu:

- Prilikom analiziranja ponuda i karakteristika određenih smještaja i soba dolazi do znatnog usporavanja aplikacije izazvanim umanjenjem performansi pri dobivanju informacija iz baze podataka. Navedeni problem ćemo riješiti primjenom prototip paterna koji će omogućiti da već učitane podatke “recikliramo”, tačnije kloniranjem ponovno iskoristimo.
U sistemu bi mogli iskoristiti Prototip patern nad klasama Smještaj i Soba s obzirom da je većina atributa koji se nalaze u njima jednaki, pa samim tim i kloniranje ima smisla. Kreirat ćemo interfejs IPrototip koji će omogućavati kloniranje instanci klasa Smještaj i Soba (upravljat će korisnik).



3. Factory method pattern

Uloga Factory Method paterna je da omogući kreiranje objekata na način da podklase odluče koju klasu instancirati. Različite podklase mogu na različite načine implementirati interfejs. Factory Method instancira odgovarajuću podklasu (izvedenu klasu) preko posebne metode na osnovu informacije od strane klijenta ili na osnovu tekućeg stanja

Primjena Factory method paterna u sistemu:

U našem sistemu Factory method pattern možemo najlakše primijeniti kod plaćanja rezervacije pošto smo u sistemu omogućili plaćanje na više načina:
- KARTIČNO
- PAYPAL
- ALTERNATIVNO

S obzirom na to mi bismo trebali posjedovati klasu koja će imati metodu koja će odlučivati koju klasu instancirati. Svaka metoda plaćanja isto tako bi trebala imati svoju vlastitu klasu koja će predstavljati taj metod plaćanja i još dodatno interfejs koji će biti povezan sa controller klasom za rezervaciju smještaja.

4. Abstract factory pattern

Abstract Factory patern omogućava da se kreiraju familije povezanih objekata/produkata. Na osnovu apstraktne familije produkata kreiraju se konkretne fabrike (factories) produkata različitih tipova i različitih kombinacija. Patern odvaja definiciju (klase) produkata od klijenta. Zbog toga se familije produkata mogu jednostavno mijenjati ili ažurirati bez narušavanja strukture klijenta. Važan aspekt Abstract Factory paterna je da se cijela familija (fabrika) produkata može promijeniti za vrijeme izvršavanja aplikacije zbog toga što produkti implementiraju isti apstraktni interfejs. Interfejsi operacija za pojedine fabrike su isti ali njihova implementacija se razlikuje. To omogućava da sistem bude neovisan od toga kako su produkti kreirani, sastavljeni i implementirani

Primjena Abstract factory paterna u sistemu:

U našem sistemu Abstract factory pattern bismo mogli implementirati u vezi sa klasom Uplata gdje bismo imali nasljeđivanja u smislu da bi nam klase koje nasljeđuju klasu Uplata predstavljale načine plaćanja, a pored toga bismo imali dvije factory klase koje bi se zvale PremiumPlaćanjeFactory i RegistrovaniPlaćanjeFactory. Ideja je da registrovani korisnik ima mogućnost plaćanja karticom gdje bi prilikom svakog unosa morao unositi podatke sa svoje kartice, a Premium korisnik bi mogao plaćati preko PayPal-a ili alternativnom metodom koja također ne traži nikakav dodatni unos podataka, nego je to izvršeno prije. Imat će iste metode plaćanja samo će se razlikovati u implementaciji i zahtjevu podataka. Također, primjenom ovog patterna bismo ukinuli enumeration koji nam predstavlja plaćanja i ubacili bismo konkretne klase.

5. Builder pattern (primijenjen u sistemu)

Uloga Builder paterna je odvajanje specifikacije kompleksnih objekata od njihove stvarne konstrukcije. Isti konstrukcijski proces može kreirati različite reprezentacije. Upotreba Builder paterna se često može naći u aplikacijama u kojima se kreiraju kompleksne strukture. Koristi se kada je neovisan algoritam za kreiranje pojedinačnih dijelova, kada je potrebna kontrola procesa konstrukcije, kada se više objekata na različit način sastavlja od istih dijelova. Omogućuje konstrukciju istog objekta primjenom razlicitih postupaka.

Primjena Builder paterna u sistemu:

U sistemu primijenili smo Builder patern na način da omogućimo alternativno dodavanje smještaja u bazu podataka. Jedan način dodavanja bi bio korištenjem web servisa, tj. API-ja pa samim tim bi se podaci prikupljali automatski sa web servisa. Drugi način bi bio manuelno dodavanje smještaja ukoliko nije moguće naći bilo kakve informacije o smještaju pošto naš sistem ne sadrži samo hotele, nego i motele, hostele, apartmane i dr.