Kreacijski paterni

Prototype - Uloga Prototype paterna je da kreira nove objekte klonirajući jednu od postojećih objekata. Praktičnije je klonirati objekte koji zauzimaju mnogo resursa. Prototype dizajn patern dozvoljava da se kreiraju prilagođeni objekti bez poznavanja njihove klase ili detalja kako je objekat kreiran. U našem projektu Prototype patern možemo iskoristiti kada se zauzeto mjesto oslobodi, tj kada istekne vrijeme rezervacije. Sve attribute klase Mjesto bismo klonirali, osim 'zauzeto' koji postavljamo na false. Mogli bismo napraviti interfejs IMjesto, koji bi nam omogućio da kloniramo instancu klase Mjesto, te bismo nakon toga mogli modifikovati ili dodati atribut 'zauzeto'.

Abstract Factory – Ovaj patern omogućava da se kreiraju familije povezanih objekata/produkata. Patern odvaja definiciju produkata od klijenta. Zbog toga se familije produkata mogu jednostavno izmjenjivati ili ažurirati bez narušavanja strukture klijenta.
Naš sistem se trenutno odnosi samo na Sarajevo. Ukoliko bismo proširili naš projekat sa mogućnošću traženja I rezervisanja parkinga u različitim gradovima, od velike koristi bi bio ovaj pattern.
Tada bismo implementirali interfejse za sve gradove.
Tako bi npr. za Tuzlu imali TuzlaFactory, za Sarajevo SarajevoFactory, itd., TuzlaFactory1, SarajevoFactory1 bi bile klase koje implementiraju operacije kreiranja za pojedinačne fabrike, pojedinačni parkinzi bi bili produkti (apstraktni interfejsi za pojedinačne grupe produkata), a također bi postojale I klase koje implementiraju interfejse i definiraju objekte produkata koji se kreiraju za odgovarajuću fabriku.

Builder pattern – Uloga Builder paterna je odvajanje specifikacije kompleksnih objekata od njihove stvarne
konstrukcije. Isti konstrukcijski proces može kreirati različite reprezentacije. Ovaj pattern je veoma koristan kada imamo klasu sklonu nadograđivanju i dodavanju parametara u konstruktor. U našem sistemu trenutno imamo samo mogućnost pregleda rezervacije za jednog korisnika. Ukoliko želimo omogućiti da admin (zaposlenik) ima uvid u sve rezervacije, morali bismo dodati nove atribute. Kako ne bi došlo do komplikacija, može se iskoristiti Builder pattern.
Potrebno je napraviti interfejs IBuilder, koji definiše pojedinačne dijelove, koji se koriste za izgradnju produkta, zatim klasu Director, koja sadrži neophodnu sekvencu operacija za izgradnju produkta, klasu Builder, koju poziva Director da bi se izgradio product i klasu Product, koja bi održavala listu dijelova, koja je rezultat konstrukcijskog procesa. To bi u našem slučaju bila lista rezervacija.

Singleton - Ovaj pattern je primjenjen u obliku klase Track koja služi za praćenje vremena, upravljenje vremenski zavisnih taskova, slanje mail-ova itd. Zbog svega navedenog, dovoljna je jedna instanca ove klase.

Factory Method - Ovaj pattern je primjenjen kod popusta na način da se koristi klasa OdabiracPopusta da na osnovu funkcije ProvjeriKorisnika() odluči koju klasu popusta (StalniGostUzastopno, StalniGostMjesecno, OsobaSInvaliditetom) instancira, a navedena klasa se poziva iz RezervacijaControllera.