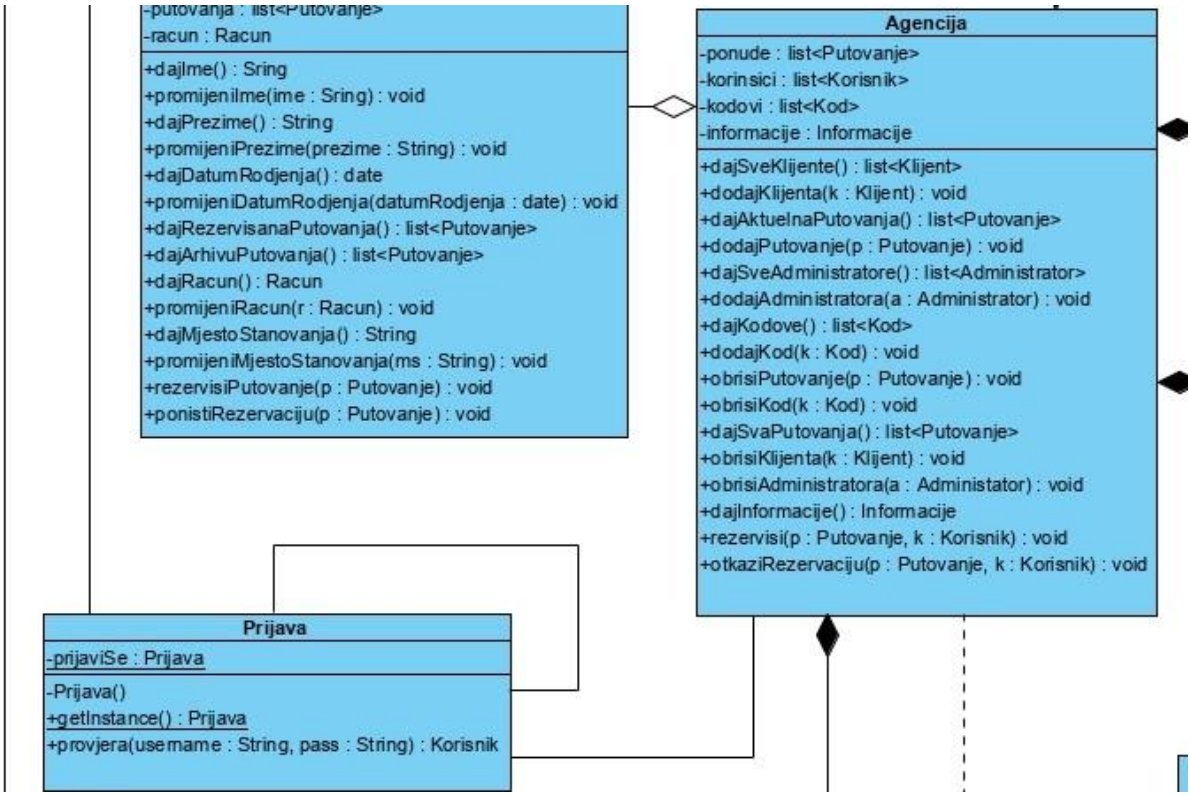
Kreacijski paterni

Singleton patern – je kreacijski patern koji nam omogućava da klasa ima samo jednu instancu, dok nam pruža globalni pristup toj instanci.

Iskoristili smo ovaj patern prilikom logiranja korisnika, jer nam je potrebna jedna instanca koja je dostupna svim klijentima i da bismo imali imali jedinstveno upravljanje bazom u tom slučaju. Također se ovaj patern može iskoristiti i kasnije u projektu kada budemo radili sa bazom naprimjer pri registraciji korisnika da postoji jedinstvena klasa sa metodama za pristup bazi koja bi omogućila da se registriraju korisnici bez da svaki put se moramo posebno konektovati na bazu, i slično.

Implementira se na način da imamo privatni staticki atribut gdje cemo drzati instancu u klasi Prijava, privatni konstruktor kojem će moći prostupiti samo ta(Singleton) klasa, i public metoda getInstance koja omogućava pristup instanci klase, koja će prvi put da pozove konstruktor ukoliko instanca nije kreirana, a svaki naredni put vraća instancu, na ovaj način smo i predvidjeli ovo korištenje u class diagramu.

Osigurali smo da se ova klasa ima jedinstvenu (single) instancu i globalni pristup toj instanci i jedinstveno pristupanje bazi podataka, te singleton objekt je inicijaliziran samo kada se ukazala potreba za njim prvi put.

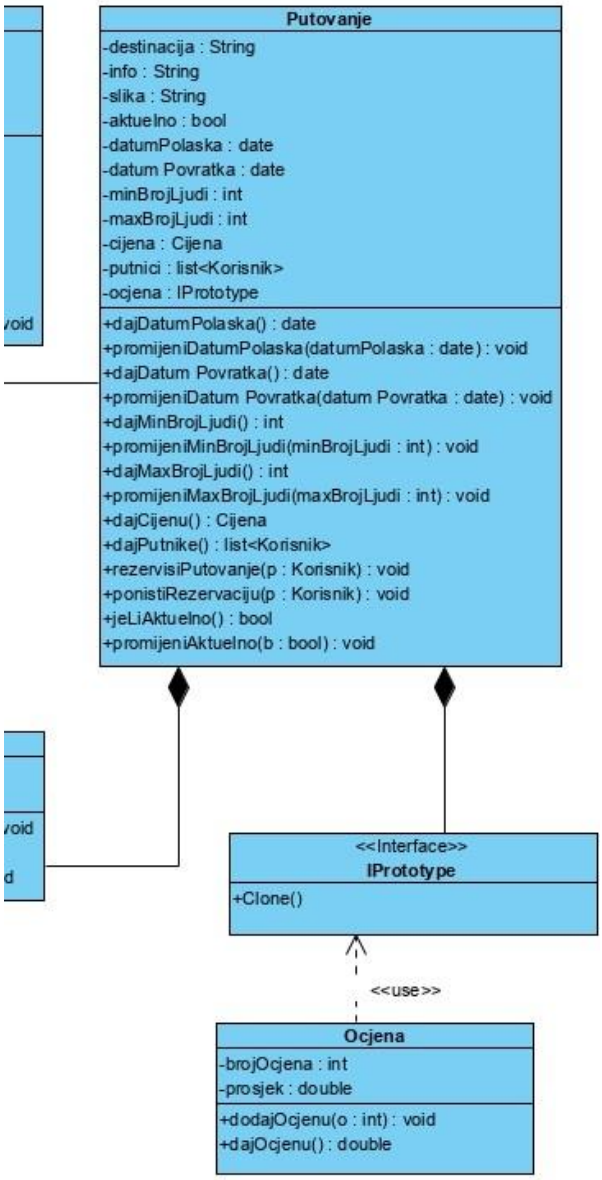


Prototype patern – je kreacijski patern koji nam omogućava da kopiramo postojeće objekte bez da kod bude ovisan o klasama. Pošto moramo znati klasu objekta da bismo mogli kreirati kopiju, kod nam postaje ovisan o toj klasi, da bismo ovo izbjegli koristimo prototype patern. Također kada je trošak kreiranja novog objekra velik i kreiranje objekta je resursno zahtjevno, vrši se kloniranje postojećeg objekta.

Implementira se na način da klasa Client zahtijeva klonranje postojećeg objekta preko interfejsa Iprototype, to je interfejs kojim se omogućava kloniranje postojećih konkretnih objekata.

Ovaj patern ćemo iskoristiti u našem projektu za Ocjene jer većina korisnika daje iste ocjene, te bi se klonirale postojeće ocjene. Imamo interface Iprototype koji nam omogućava kloniranje postojećih Ocjena. Klasa koja zahtjev kloniranje postojećih objekata preko ovog interfejsa je Putovanje.

Upotrebom ovog paterna smo izbjegli ponovno instancijranje klase, kloniranjem su prepisane sve vrijednosti klase Ocjena te se samo može promijeniti vrijednost, također ovaj patern omogućava da se izbjegne bespotrebni dio inicijalizacijskog koda, da možemo povoljnije koristiti kompleksne objekte.



Factory Method patern – uloga ovog 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 preko posebne moetode na osnovu informacije od strane klijenta ili na osnovu tekućeg stanja. Koristimo ga kada na početku ne znamo kojeg je tačno tipa objekat sa kojim radimo, jer odvaja dio koda konstrukcije objekta od dijela u kojem zapravo koristimo taj objekat. Također koristimo ovaj patern kada želimo da sačuvamo resurse sistema, ponovnim korištenjem postojećih objekata umjesto da ih „rebuild“ svaki put.

Mogli bismo ga iskoristiti u našem projektu da smo imali više Agencija(dvije), te da se na osnovu grada u kojem je klijent odluči koja će se agencija instancirati, što nam je potrebno ukoliko korisnik izabere direktno plaćanje da zna u koju poslovnicu treba otići.

Implementirali bi na način da imamo interfejs IProduct, zatim konkretne klase AgencijaA i AgencijaB koje implementiraju interfejs. Klasu Sistem koja posjeduje FactoryMethod() metodu koja odlučuje koju klasu instancirati na osnovu adrese Korisnika.

Ovaj patern nam omogućava da izbjegnemo usku povezanost između kreatora i pojedinih produkata i slijedi single responsibility princip, jer možemo kreiranje „proizvoda“ premjestiti na jedno mjesto u programu. Također i open/closed princip jer se mogu dodavati novi tipovu „proizvoda“ u program bez da se mijenja postojeći kod klijenta.

Abstract Factory patern – omogućava da se kreiraju familije povezanih objekata/produkata bez da specificiramo njihovu konkretnu klasu. Na osnovu apstraktne familije produkata kreiraju se konkretne fabrike (factories) produkata različitih tipova. Patern odvaja definiciju klase od klijenta. Familije produkata se mogu jednostavno izmjenjivati ii ažurirati bez narušavanja strukture klijenta. Koristimo ovaj patern kada kod treba da tadi sa različitim familijama povezanih produkata, ali ne želimo da ovisi o konkretnoj klasi tih produkata. Također i za generiranje različitih izgleda i korisničkih interfejsa i za pisanje portabilnog koda za različite operativne sisteme za iste operacije.

Implementira se na način da imamo Ifactory apstraktni interfejs sa Create operacijama za svaku fabriku, zatim klase Factory koje implementiraju operacije kreiranja za pojedinačne fabrike, apstraktne interfejse za pojedinačne grupe produkata, te klase koje implementiraju interfejse produkata i definiraju obekte koji se kreiraju za odgovarajuću fabriku. Tu je i Client klasa koja preko interfejsa koristi objekte(produkte) fabrike.

U našem projektu bismo ga mogli iskoristiti da imamo omogućeno putovanje različitim vrstama prijevoznih sredstava, i da imamo izbor let avionom i vožnju autobusom, apstraktne interfejse koji bi predstavljali ove dvije opcije. Te da imamo Factory-je Aviokompaniju i neku kompaniju koja pruža usluge vožnje autobusom, te za svako putovanje bi imali pojdinačne vožnje avionom i autobusom. Dakle imali bi interfejse ILet i IBus, njih bi implementirale klase Avion koji pripada aviokompaniji i Autobus koji pripada firmi za vožnju autobusom. IFactory interface koji implementiraju Aviokompanije i Buskompanija klase.

Ovaj patern omogućava da budemo sigurni da proizvodi koje dobivamo iz factory su kompaktibilni jedni sa drugima, omogućava nam da izbjegnemo usku vezu između konkretnih objekata(produkata) i klijent koda. i slijedi single responsibility princip, jer možemo kreiranje „proizvoda“ premjestiti na jedno mjesto u programu. Također i open/closed princip jer se mogu dodavati novi tipovu „proizvoda“ u program bez da se mijenja postojeći kod klijenta.

Builder – omogućava nam da kreiramo kompleksne objekte korak po korak. Ovaj patern nam omogućava da proizvedemo različite tipove i reprezentacije objekata koristeći isti konstrukcijski kod.

Osnovni elementi ovog paterna su Ibuilder interfejs koji definira pojedinačne dijelove koji se koriste za izgradnju produkta, Director kllasa koja sadrži neophodnu sekvencu operacija za izgradnju produkata. Builder klasu koja se poziva od strane direktora da se izgradi produkt, te Product klasa na osnovu koje se kreira objekat koji se gradi preko dijelova.

U našem projektu nismo koristili ovaj patern, ali da smo imali funkcionalnost da klijent odabere da li želi puni pansion tokom putovanja, sa doručkom, ručkom i večerom. Mogli bismo iskoristiti ovaj patern da se naprave ti obroci od nekih dijelova tj namirnica, te bismo imali klasu Kuhar koja bi implementirala interfejs IBuilder. Neku klasu Šef koja bi omogućavala da se konstruira objekat iz više djelova i sadržavala metod Construct. Dijelovi bi bili namirnice, a sam proizvod(produkt) taj obrok. Imali bi klase namirnica kao sto su Mlijeko, Brasno itd. I klasu Obrok koja bi predstavljala proizvod sastavljen od dijelova(namirnica).

Ovaj patern nam omogućava da ponovo koristimo kod(reuse) kada gradimo različite reprezentacije produkata, također slijedi princip Single Responsibility, možemo odvojiti kompleksni konstrukcijski kod od biznis logike produkta(objekta).