Elektrotehnički fakultet u Sarajevu

Patern izvještaj

Upotreba strukturnih i kreacijskih paterna

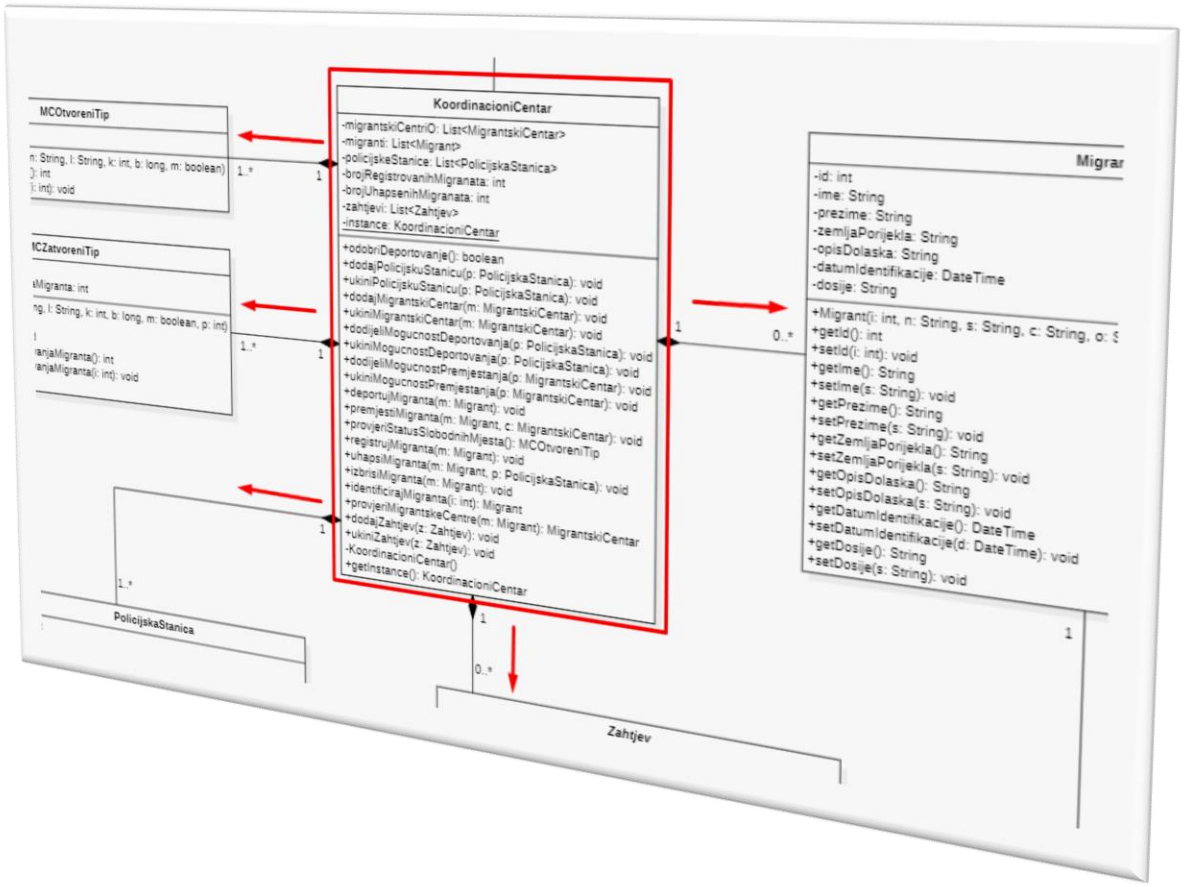
Grupa: Briferi

Ak. godina: 2019/20

Adapter patern

Adapter patern služi da se postojeći objekat prilagodi za korištenje na neki novi način u odnosu na postojeći rad, bez mijenjanja same definicije objekta. U našem projektu adapter pattern nije implementiran. Jedan od slučajeva, kada bismo mogli implementirati ovaj pattern, jeste u slučaju da klasa Administrator ima metodu ucitajMigranteIzXml(XML migranti), koja učitava migrante iz xml formata. U slučaju da je potrebano učitati migrante iz nekog drugog formata, npr. JSON-a, možemo koristiti adapter pattern da omogućimo ovo funkcionalnost bez izmjene postojećeg objekta. Implementaciju bi izvršili kreiranjem interfejsa IUcitavanje sa metodom ucitajMigranteIzJSON(JSON migranti) i klase AdministratorAdapter sa metodom ucitajMigranteIzJSON(JSON migranti), koja će prvo učitati iz JSON-a i pretvoriti u XML, a zatim pozvati postojeću metodu ucitajMigranteIzXml.

Fasadni patern

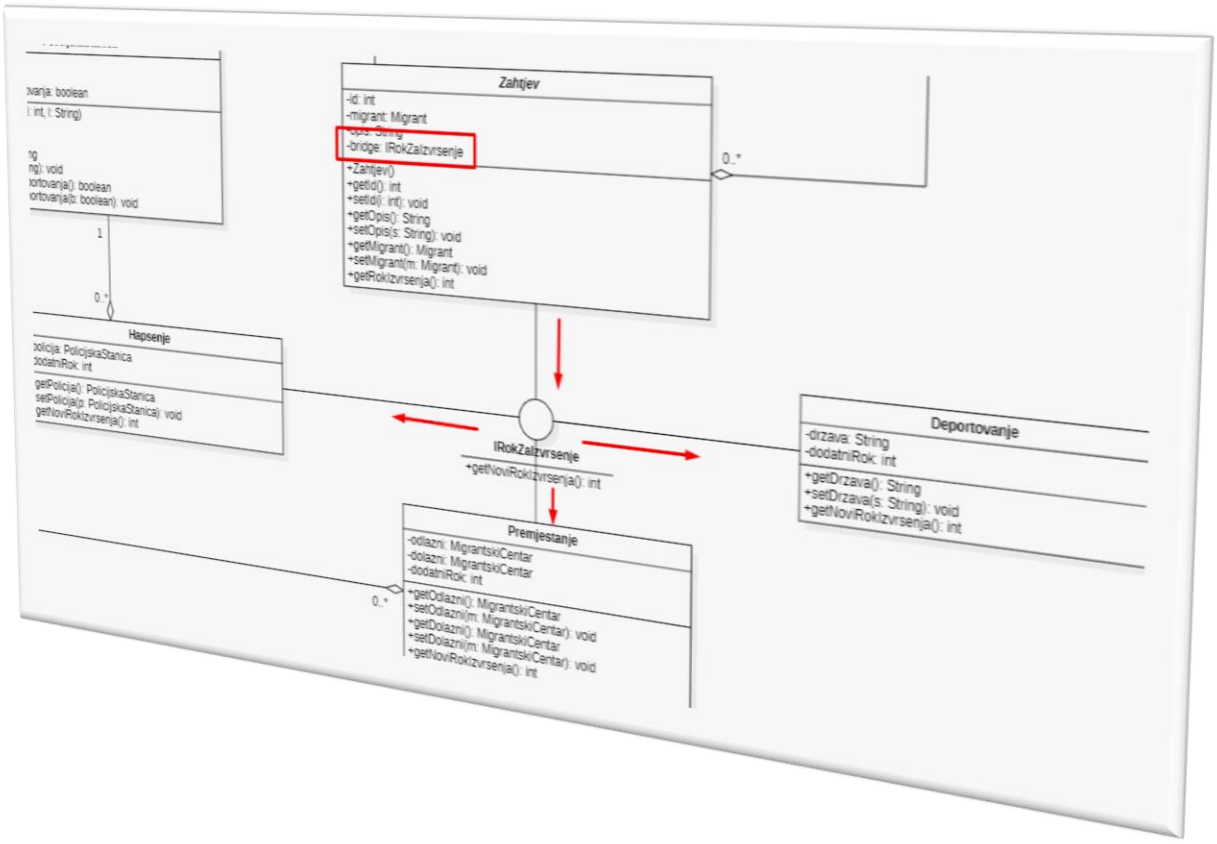


Fasadni patern je implementiran preko klase KoordinacioniCentar(Administrator) buduci da sve kompleksne metode koje koriste razlicite klase (Migrant, PolicijskaStanica, Zahtjev) se nalaze unutar ove klase.
Zbog ovog paterna korisnik sistema ne mora biti upucen kako se pojedine kompleksne metode odvijaju u pozadini jer je sve vec implementirano i spremno za koristenje pozivom jedne kompleksnije metode.
Mana ovog pristupa jeste sto su odredjene metode kompleksnije, a predonost je u tome sto je korisniku olaksan poziv metoda i sto je sve upakovano u jednu klasu sa kojom komunicira korisnik.
U konkretnom primjeru naseg sistema poziv metode deportujMigranta ili premjestiMigranta ce obaviti vise funckija sa klasama MigrantskiCentar i njegovim podklasama, te sa klasom PolicijskaStanica koja se mora pripremiti te sprovesti navedenu akciju tj. zapovijed.
Sa slike vidimo nasu 'fasadnu' klasu koja komunicira sa svim ostalim klasama tkd. se olaksava pristup sistemu.

Dekorator patern

Dekorater patern je implementiran preko klase Zahtjev. U klasi KoordinacioniCentar postoji metoda dodajZahtjev, te ukiniZahtjev. Zahvaljujući klasi Zahtjev kao parametar prethodno navedenim metodama šaljemo upravo klasu Zahtjev, a ne cijeli broj (gdje bi cijeli broj predstavljao prirodu zahtjeva, npr. broj 1 bi govorio da se radi o zahtjevu za hapšenje,...). Samim tim smo izbjegli potrebu da korisnik pamti koji broj predstavlja koji zahtjev.

Bridge patern



Bridge patern je implementiran preko klase Zahtjev jer rok za izvrsenje zahtjeva nije isti za sve tipove zahtjeva. Medjutim razlog implementacije bas ovog paterna jeste sto je primarni rok za sve isti i on je 2 dana, ali svaki tip posebno ima dodatne dane za izvrsenje zahtjeva u skladu sa kompleksnoscu zahtjeva tj. recimo cijeli proces deportovanja ce puno vise trajati nego obicno lociranje i hapsenje migranta.
Kreiran je interfejs IrokZaIzvrsenje koji sadrzi metodu getNoviRokIzvrsenja koju klase Hapsenje, Deportovanje i Premjestanje polimorfno nasljedjuju. Ta metoda sluzi za kalkulaciju glavnoj metodi zahtjeva gerRokIzvrsenja na nacin koji je vec naveden iznad u tekstu.
Prednost ovog paterna je sto smo iskoristili vrijednost primarnog roka za svaki tip zahtjeva i na njega dodali dodatne dane roka.

Kompozitni patern

Kompozitni patern bi bio implementiran dodavanjem nove metode unutar klase KoordionacioniCentar. Uvođenjem ovog paterna korisnik bi imao jasan pregled svih rokova izvršenja određenih zahtjeva. Ključni dio zbog kojeg je lako uvesti ovu metodu je postojanje interfejsa IRokZaIzvrsenje.

Prednost ovog paterna je što bismo u metodi koju trebamo dodati pristupali na isti način za svaki zahtjev.

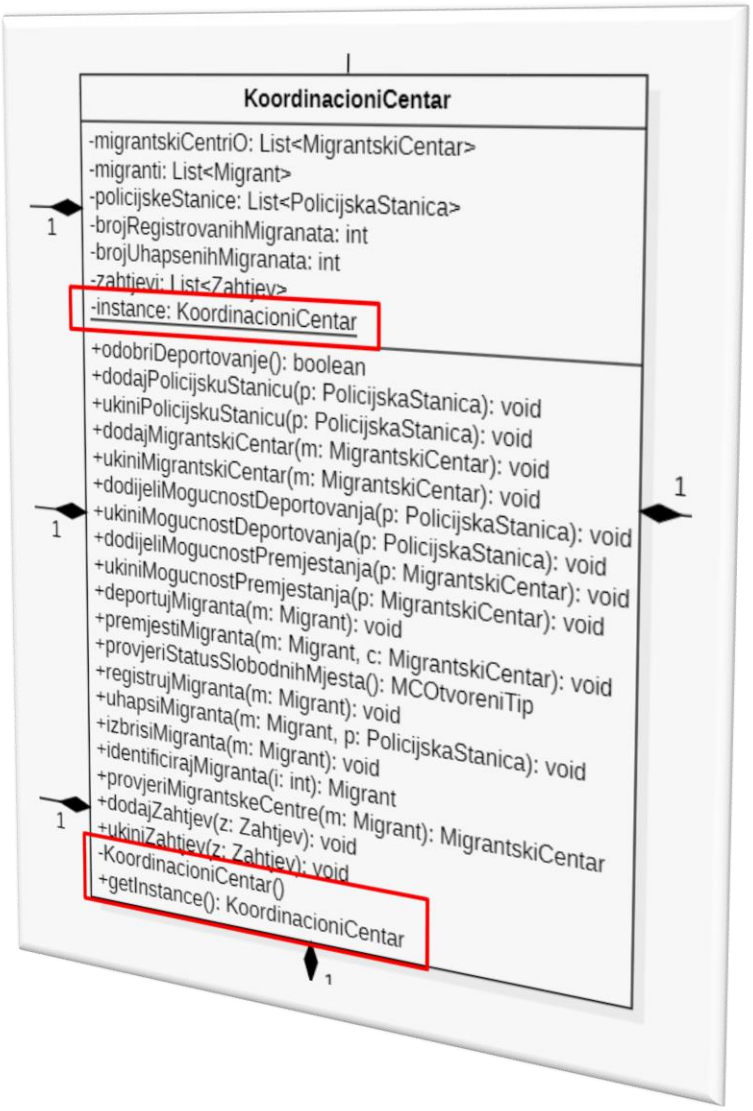
Proxy patern

Proxy pattern omogućava kreiranje surogata originalnog objekta, pomoću kojeg se kontroliše pristup originalnom objektu. U našem projektu ovaj pattern nije implementiran. Proxy pattern bi bilo moguće implementirati u slučaju da objekti tipa Migrant zauzimaju previše memorije(razni izvještaji, slike, ...), pa bi ucitavanje svih migranata iz baze u memoriju računara stvaralo problem. Ovaj problem se rješava kreiranjem proxy-ja Migranata, koji bi čuvali samo osnovne podatke o migrantu. U slučaju da su potrebni svi podaci o migrantu oni bi se učitali iz baze samo za odgovarajućeg migranta. Implementacija bi se vrsila preko interfejsa IMigrant, koji bi imao metode dajIme(), dajPrezime(), dajId(),dajMigrantskiCentar() i preko klasa Migrant i MigrantProxy. Obje klase bi implementirale interfejs IMigrant. Prilikom ucitavanja iz baze kreirali bi se smo objekti MigrantProxy sa atributima ime,prezime i id. Kada bi korisnik zatrazio detaljne podatke o migrantu, one bi se učitale iz baze u objekat tipa Migrant.

Flyweight patern

Flyweight patern koristi se kako bi se onemogućilo bespotrebno stvaranje velikog broja instanci objekata koji svi u suštini predstavljaju jedan objekat. Ovaj pattern nismo implementirali u svom projektu. Mogli bismo ga implementirati u slučaju kada bi migranti imali pristup sistemu, da se upoznaju sa najnovijim informacijama. Migranti bi se prijavljivali u sistem pomoću svog ID-a(da se provjeri da li su prijavljeni) i jezika. ID bi služio samo za validaciju i ne bi uticao na ostale aktivnosti sistema. Informacija bi se cuvale u klasi Informacije, sa atributima jezik i informacije. Informacije su iste za sve govornike jednog jezika. Da se ne bi za svakog migranta kreirala nova instanca klase informacije, najbolje je imati listu kreiranih instanci za razlicite jezike. Za implementaciju ovo patterna potreban je interfejs IInformacije sa metodom dajJezik(). Klasa Infromacije bi implementirala ovaj interfejs, a lista sa instancama klase Informacije bi se nalazila u klasi Administrator.

Singlton patern



Singlton patern je implementiran u klasu KoordinacioniCentar(Administrator) pomocu static varijable ovog tipa, privatnog konstruktora i metode getInstance. Razlog uvodjenja staticne varijable ovog tipa jeste da bi svaki poziv unutar koda ove instance imao istu referencu na ovu varijablu. Privatni konstruktorom zabranjujemo ponovno kreiranje ove klase sto nema nikakvog smisla, a konstruktor se jedino poziva privatno unutar klase i to je u nasoj metodi getInstance, koja ukoliko nije kreirana niti jedna instanca, kreira je i vrati a ukoliko jeste samo vrati instancu koja ce biti jedina tokom zivota programa. Prednost singltona jeste sto ne moze doci do konflikta da se vise centara (administratora) kreira koji bi mogli napraviti probleme kada bi u isto vrijeme mijenjali podatke, takodjer svakako je zamisljeno da postoji samo jedan KoordinacioniCentar koji ce nadgledati i upravljati sistemom tako da je singlton patern predvidjen od samog starta implementacije sistema.

Prototype patern

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. U našem projektu nismo implementirali ovaj pattern. Slučaj kada bismo mogli implementirati ovaj pattern jeste kreiranje Policijskih Stanica. Ako bi PolicijskeStanice imale istu vecinu atributa, kreiranje novih stanica bi bilo olakšano kloniranjem već postojećih. Implementacija bi se vršila preko interfejsa IPolicijskeStanice, koji bi imao metodu clone(). Klasa PolicijskaStanica bi implementirala ovaj interfejs. Prilikom

kreiranja novih PolicijskihStanica pozivala bi se metoda clone() već postojećeg objekta, a atributi koji se razlikuju izmijenili bi se odgovarajućim setterima.

Factory Method patern

Factory method patern je implementiran preko factory methode koja se nalazi u nasoj fasadnoj klasi koja sve obavlja (KoordinacioniCentar) i ona je kreirajZahtjev. To je dakle polimorfna metoda koja vraca različite vrste objekata tj. zahtjeva u zavisnosti od prosljedjenog parametra tipa int pomocu kojeg se definise povratni tip zahtjeva. Nasa klasa zahtjev predstavlja objekat tj klasu koja je na vecem nivou hijerarhije te ona moze primiti razlicite objekte (zahtjeve).

Prednost ove metode jeste sto smo skratili kreiranje novih zahtjeva, od tri odvojene metode koje bi imale funkciju posebnog kreiranja svakog zahtjeva dosli smo do jedne metode koja sve to objedinjuje i vraca zahtjev tj objekat odredjenog podtipa u zavisnosti od navedenog parametra funkcije tipa int.

Abstract Factory patern

Abstract Factory patern bi bilo potrebno implementirati ukoliko bismo imali klasu Rok odakle su izvedene tri klase Rok1Dan, Rok2Dana i Rok3Dana gdje bi za hapšenje dodatni rok bio jedan dan, za premještanje bi dodatni rok bio dva dana, te za deportovanje dodatni rok bi bio tri dana. Jednostavno bi se izvršila agregacija između određenih zahtjeva i

rokova. Pošto imamo interfejs IRokZaIzvrsenje ostatak uslova za implementaciju Abstract Factory paterna bi bio zadovoljen.

Prednost Abstract Factory paterna je izbjegavanje if-else blokova.

Builder patern

Builder patern se ne može implementirati. Ukoliko bismo na jednoj lokaciji imali više identičnih migrantskih centara zatvorenog tipa, te ako bi se broj migranata povećavao s čime bi došlo do potrebe za pravljenjem novih migrantskih centara builder patern bi se mogao implementirati. Dodali bismo interfejs iBuilder kojeg bi implementirale sljedeće dvije klase AutomatskiBuilder i ManuelniBuilder. AutomatskiBuilder bi se koristio za pravljenje identičnih migrantskih centara, dok ManuelniBuilder bi se koristio za pravljenje novih migrantskih centara nastalih zbog nedostatka kapaciteta u već postojećim migrantskim centrima (zbog nespremnosti sistema novi migrantski centri ne bi bili isti kao i već postojeći jer bi gradnja oduzela mnogo vremena, tako da bi se koristili neki pomoćni objekti).

Uvođenjem Builder paterna ne bi bilo potrebe za korištenjem konstruktora, već bi se to riješilo korištenjem jednostavnih metoda unutar buildera.