Elektrotehnički fakultet u Sarajevu

Patern izvještaj

Upotreba strukturnih,kreacijskih paterna i paterna ponasanja

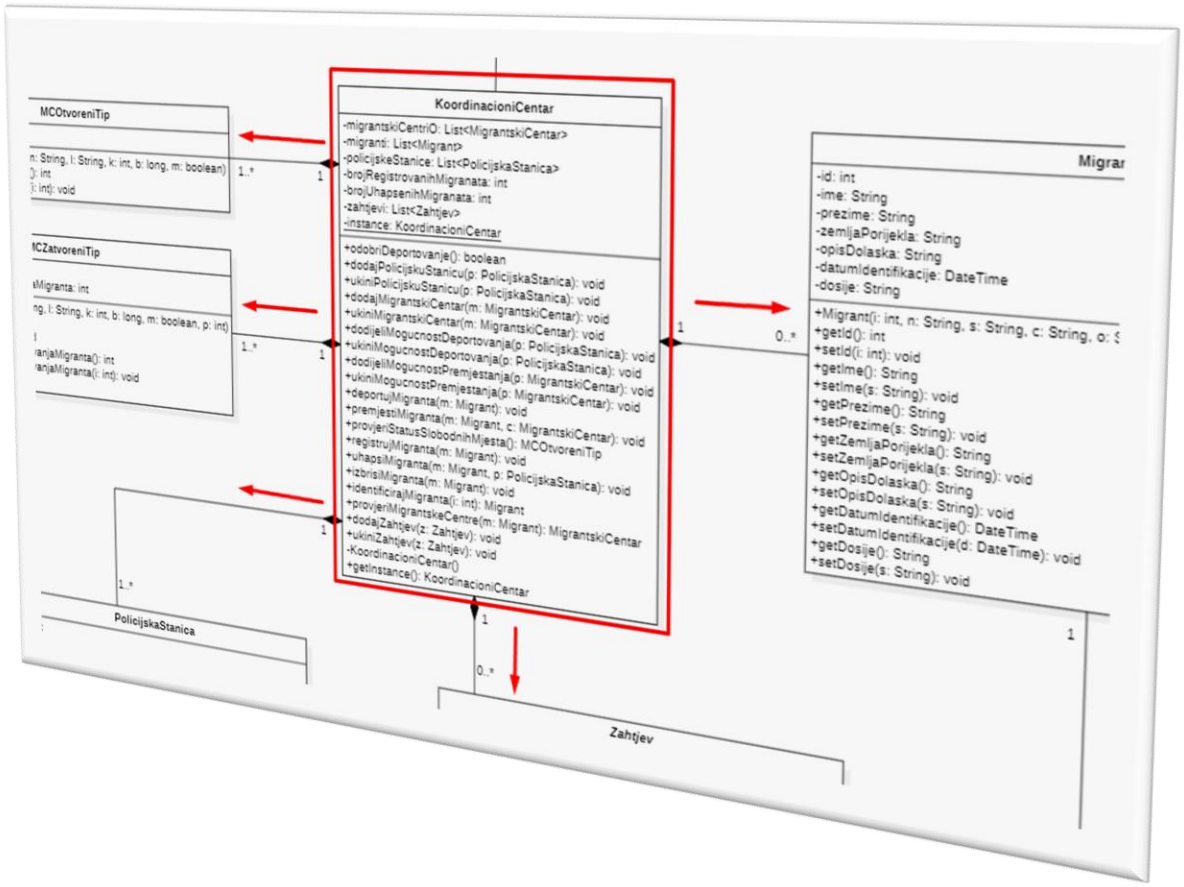
Grupa: Briferi

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

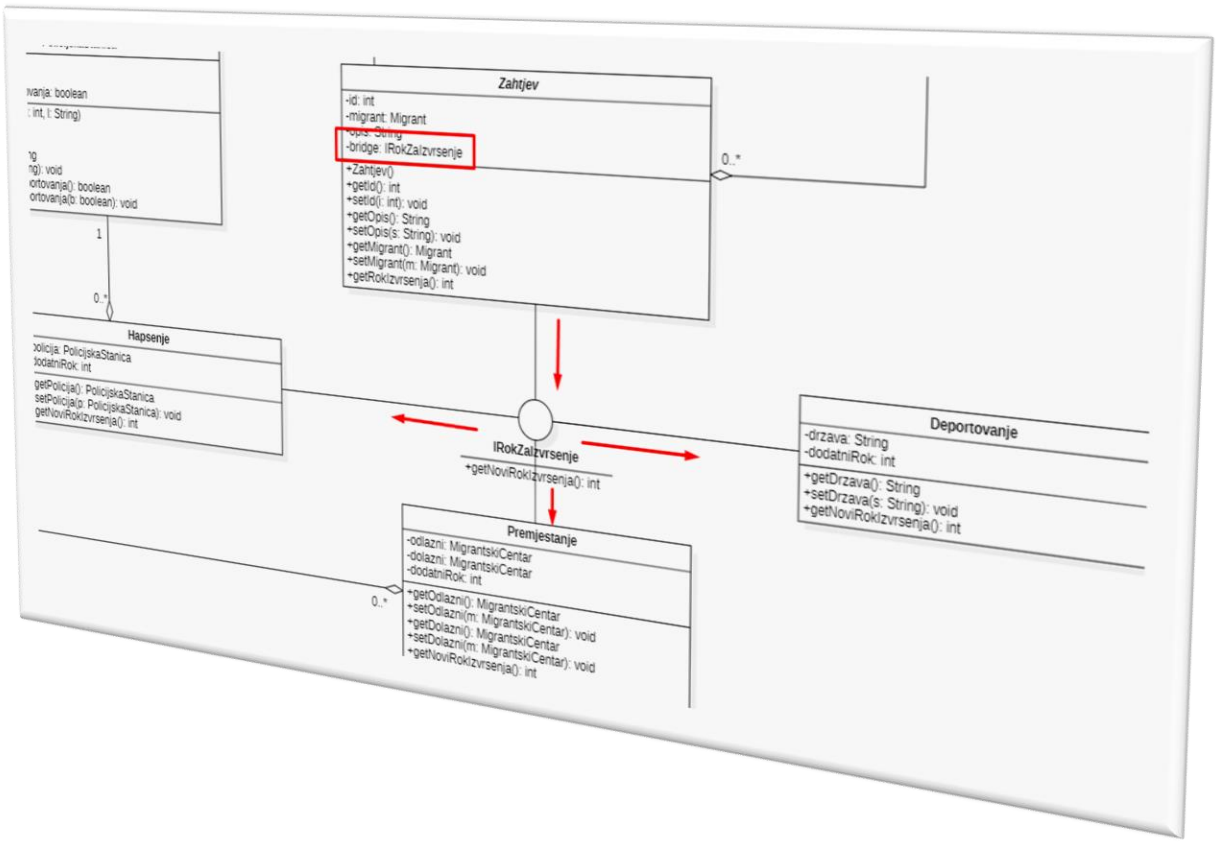


Fasadni patern je implementiran preko klase KoordinacioniCentar(Administrator) buduci da sve kompleksne metode koje koriste razlicite klase (Migrant, PolicijskaStanica, Zahtjev) se nalaze unutar ove klase.
Zbog ovog paterna korisnik sistema ne mora biti upucen kako se pojedine kompleksne metode odvijaju 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 nije implementiran. Ovaj patern bi bilo potrebno implementirati u slučaju da pravimo jedan komplekni zahtjev. Zahtjev bi se mogao sastojati od roka za izvršenje, prirode zahtjeva i lokacija izvršenja. Potrebno bi bilo dodati interfejs IZahtjev, te tri klase Rok, Priroda i Lokacija koje implementiraju prethodno navedeni interfejs. Metoda doradi bi „dorađivala“ zahtjev prolazeći kroz svaku klasu dok ne dođemo do finalnog kompleksnog zahtjeva. Prilikom prolaska kroz svaku od klasa koristio bi se prethodno dorađeni 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 dajSveRokoveIzvrsenja 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. Metoda dajSveRokoveIzvrsenja bi pozvala metodu getRokIzvrsenja za svaki zahtjev i prikazala unutar jednog stringa.

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

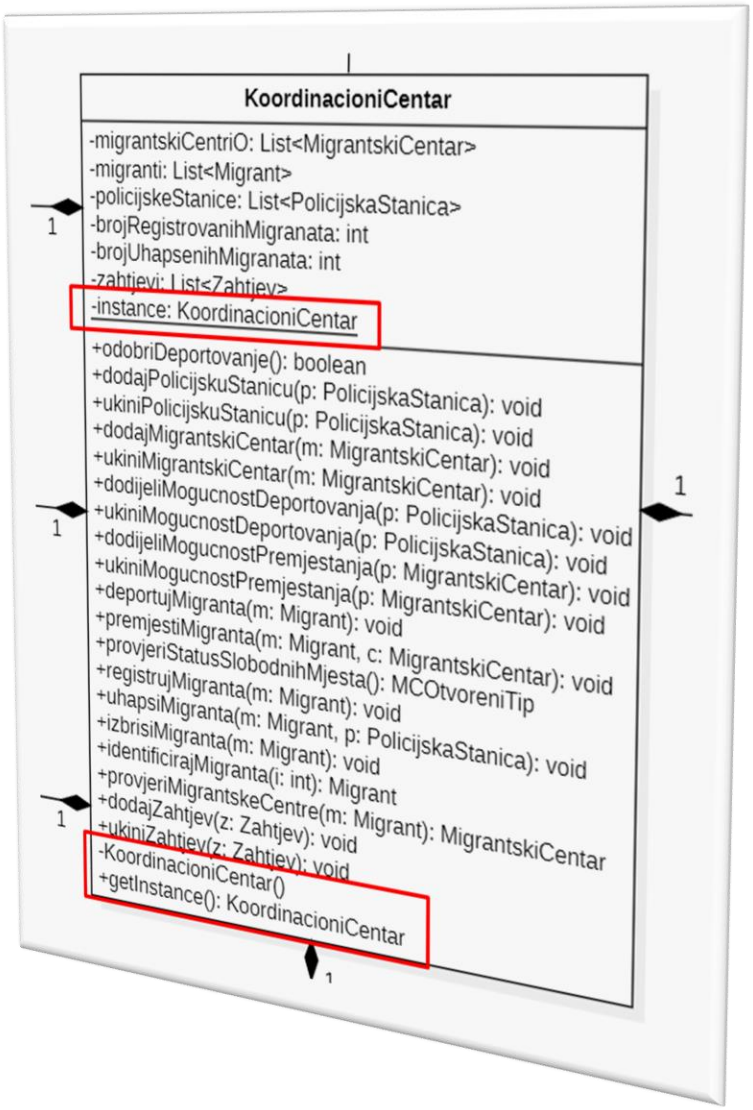
Proxy patern

Proxy pattern omogućava kreiranje surogata originalnog objekta, pomoću kojeg se kontroliše pristup originalnom objektu. U našem projektu ovaj pattern nije implementiran. Proxy pattern možemo implementirati u slučaju da imamo i različite zaposlenike u MigrantskimCentrima, koji nemaju ista prava pristupa informacijama. Ovaj problem se rješava kreiranjem proxy-ja Migranata, koji bi čuvali samo osnovne podatke o migrantu u klasi PrototipMigrant. U slučaju da neko želi dodatne podatke o Migrantu tražila bi se validacija prava pristupa. Implementacija bi se vrsila preko interfejsa IMigrant, koji bi imao metode dajIme(), dajPrezime(), dajId(),dajMigrantskiCentar() i preko klasa Migrant, MigrantPrototip i Proxy. Klase Migrant i MigrantPrototip bi implementirale interfejs IMigrant. U klasi Proxy bi se čuvala lista MigrantPrototipa, imala bi listu zaposlenika sa višim nivoom pristupa i metodu dajDetaljeOMigrantu(MigrantPrototip). Kada zaposlenik zatraži detalje o migrantu vršila bi se provjera da li je u listi zaposlenika sa višim nivoom pristupa. U slučaju da jeste podaci se učitavaju iz baze i prosljeđuju se korisniku u klasi Migrant, a u slučaju da nije vraća se klasa MigrantPrototip.

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

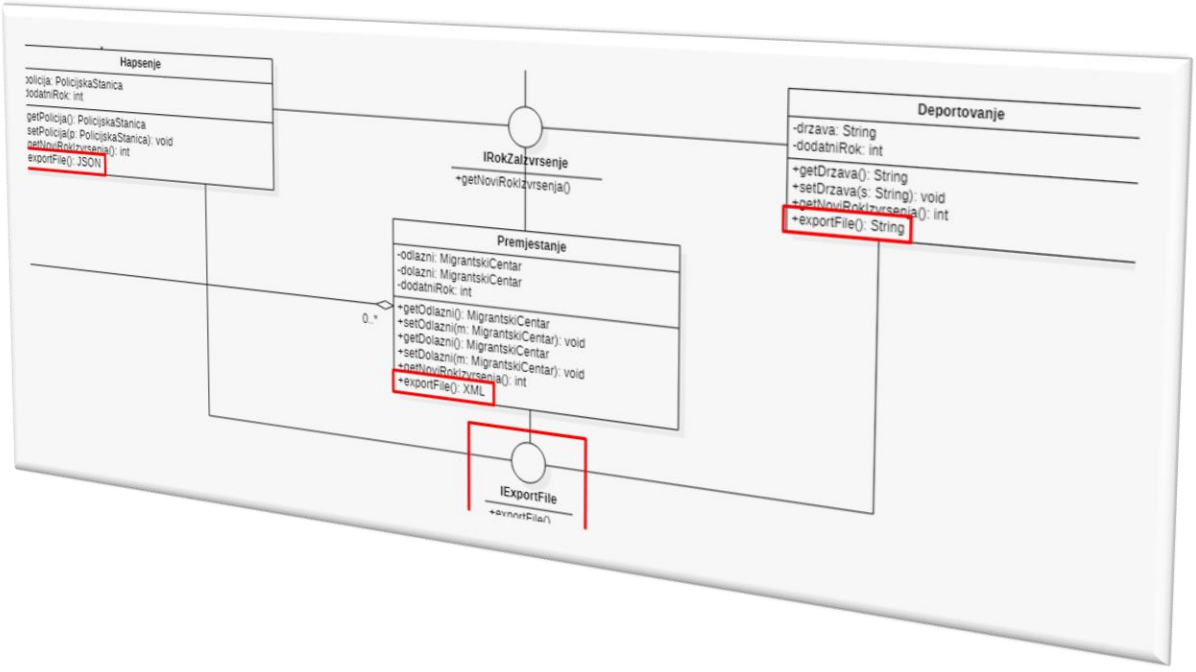


Singlton patern je implementiran u klasu KoordinacioniCentar(Administrator) pomocu static varijable ovog tipa, privatnog konstruktora i metode getInstance. Razlog uvodjenja staticne varijable ovog tipa jeste da bi svaki poziv unutar koda ove instance imao istu referencu na ovu varijablu. Privatni konstruktorom 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 exportFile i interfejsa IExportFile. To je dakle polimorfna metoda koja vraca različite vrste objekata tj. tekstualnih dokumenata koji opisuju atribute i opcenito nase zahtjeve koji ce biti kreirani. Povratni tekstualni objekti se razlikuju u zavisnosti od podklasa zahtjeva (Hapsenje, Deportovanje i Premjestanje).
Nasa klasa Zahtjev predstavlja objekat tj klasu koja je na vecem nivou hijerarhije te ona moze primiti razlicite objekte (zahtjeve). Metode su napravljene da same vracaju razlicite tekstualne objekte medjutim moze se takodjer uvesti i privatni atribut u baznu klasu koji ce se instancirati onim tekstualnim tipom u zavisnosti koja od ove 3 funkcije ce biti pozvana i koji je tip zahtjeva u pitanju tj koja podklasa.
Prednost ove factory metode jeste sto sa jednom polimorfnom funkcijom za razlicite klase dobijamo razlicite rezultantne podklase (JSON, XML, ..).

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. Imali bismo tri factory klase FactoryHapsenje, FactoryPremjestanje i FactoryDeportovanje. Jednostavno bi se izvršila agregacija između određenih Factory klasa i rokova. Pošto imamo interfejs IRokZaIzvrsenje ostatak uslova za implementaciju Abstract Factory paterna bi bio zadovoljen. Takođe potrebna je polimorfna metoda dajZahtjev koja vraća određene instance Factory klasa.

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). Builderi bi se sastojali od sljedećih metoda: postaviOsnovno(int id, String naziv, long brojTelefona), postaviLokaciju(String lokacija, int kapacitet) I postaviInfoMigrant(bool mogucnostPremjestanja, int standardniPeriodZadrzavanja). Vrijednosti automatskog buildera bi bile: id = prethodni + 1, naziv = “”, lokacija = “”, kapacitet = 100, mogucnostPremjestanja = true, standardniPeriodZadrzavanja = 30. Vrijednosti manuelnog buildera bi unosio korisnik Sistema.

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

Strategy patern

Strategy pattern izdvaja algoritam iz matične klase i uključuje ga u zasebne klase. Ovaj pattern nismo implementirali u svom projektu. Pattern bi mogli implementirati u slučaju da imamo dva načina deportacije migranata. Pattern bi se implementirao pomocu klase Context(o našem slučaju to može biti klasa Administrator), interfejsa IDeportacija koji bi imao metodu deportujMigranta() i klasa Deportacija1 i Deportacija2, koje bi implementirale dati interfejs. Klasa Context bi imala atribut IDeportacija u kojem bi se cuvala trenutna klasa, koja sadrži metodu deportujMigranta(). Također imala bi i setter metodu da se promijeni algoritam deportacije.

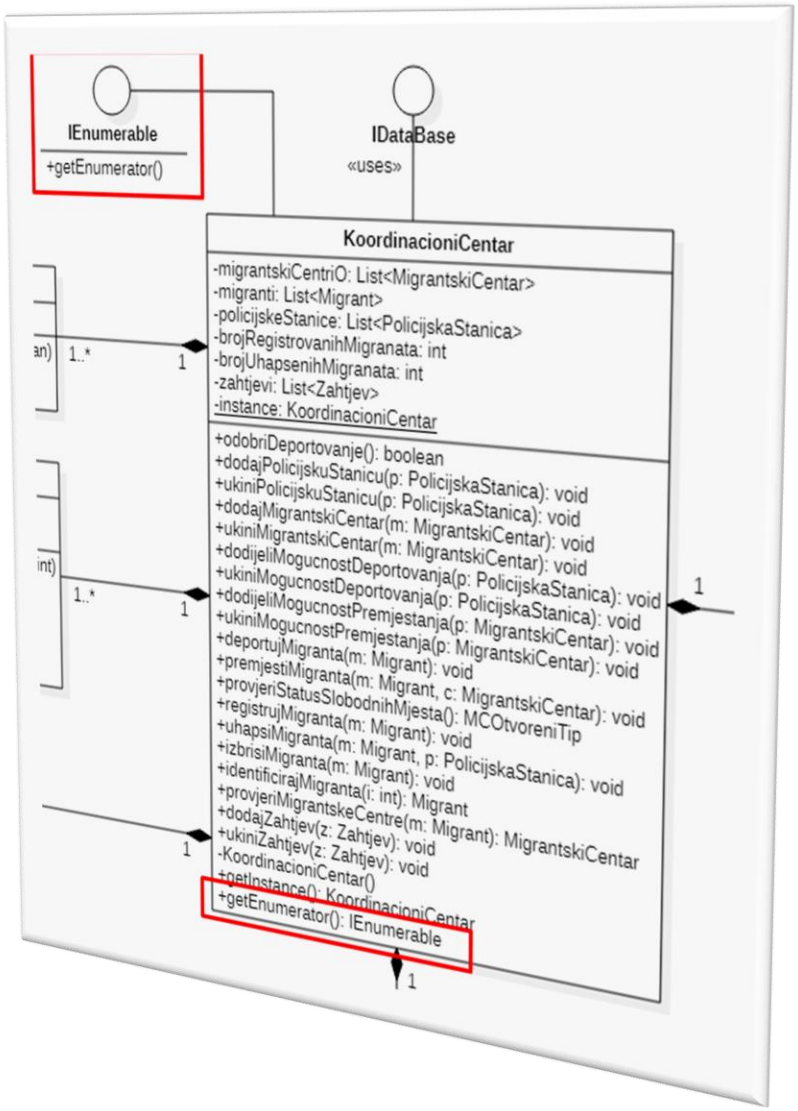
State patern

State pattern je dinamicka verzija Strategy patterna. Objekat mijenja način ponašanja na osnovu trenutnog ponašanja. Ovaj pattern nismo implementirali. Možemo ga implemenitrati ako uzmemo u obzir da migrant može biti u dva stanja, uhapšen ili zatvoren. Zavisno od stanja, metode uhapsi i premjesti bi imale drugačiju implementaciju. Implementacija bi se vršila pomoću klase Context(u našem slučaju to može biti klasa Migrant), interfejsa IMigrant koji bi imao metode uhapsiMigranta() i premjestiMigranta() i klasa UhapseniMigrant i ZatvoreniMigrant, koje implementiraju dati interfejs. Klasa Context bi imala atribut IMigrant, koji bi cuvao trenutno stanje i odgovarajuću sett i get metodu za stanje. Klase UhapseniMigrant i Zatvoreni migrant bi imale različite implementacije metoda uhapsiMigranta() i premjestiMigranta(). Zavisno od toga u kojem je stanju klasa Context, pozivala bi se odgovarajuća metoda.

TemplateMethod patern

Omogućava izdvajanje određenih koraka algoritma u odvojene podklase. Struktura algoritma se ne mijenja - mali dijelovi operacija se izdvajaju i ti se dijelovi mogu implementirati različito. Ovaj pattern nismo implementirali. Ovaj pattern bi se mogao primijeniti kod dodavanja novog migranta, tj. dobijanja informacija o njemu infromacija o njemu. Ovaj proces bi se razlikovao, zavisno od toga da li migrant ima dokumente ili nema. Zbog toga međuprocesi registracije migranata bi bili različiti u ova dva slučaja. Ovo je idealno za TemplateMethod pattern. Implementacija bi se vršila preko klase Algoritam(u našem slučaju je to klasa Administrator, jer posjeduje metodu dodajMigranta()), interfejsa IRegistracijaMigranata koji bi imao par metoda, kao na primjer Operacija1(), Operacija2(), koje bi vršile različite provjere, i klasa MigrantSaDokumentima i MigrantBezDokumenata, koje implementiraju dati interfejs. U klasi Algoritam/Administrator bi se nalazila Template metoda. U našem slučaju to bi bila metoda registrujMigranta(). Metoda bi primala parametar tipa IRegistracijaMigranta, i u njoj bi bio raspored izvrsavanja Operacija iz klasa koje implementiraju interfejs IRegistracijaMigranata. Zavisno od potreba u metodu bi se prosljeđivala odgovarajuća klasa.

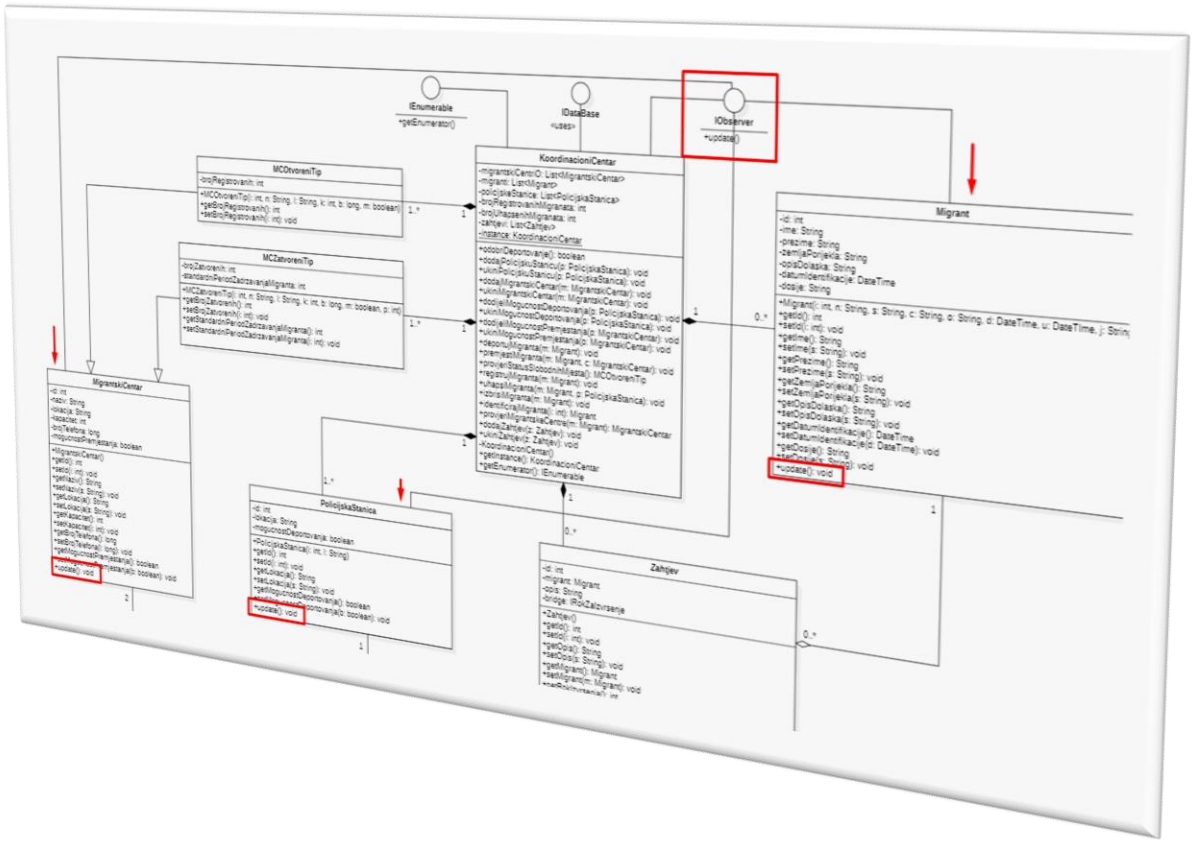
Iterator patern



Iterator patern je implementiran pomocu interfejsa IEnumerable koji sadrzi funkciju getEnumerator. Ovaj interfejs implementira nasa kontejnerska klasa KoordinacioniCentar, koja je idealno mjesto za paterne ponasanja buduci da sadrzi sve attribute i liste, i u njoj se override-a vec spomenuta metoda getEnumerator.
Prednost ovog paterna jeste sto mozemo implementirati vlastitu getEnumerator metodu koja ce vrsiti iteriranje kroz kontejnere tj. liste nase klase po logici koju mi implementiramo.

Stoga prilikom poziva for each petlje nad klasom KoordinacioniCentar sa stringom bice implementirano da se vrati lista migranata koja ce biti sortirana po drzavi porijekla migranata gdje ce pored migrantovih podataka biti ispisan i migrantski centar u kojem je isti lociran.

Observer patern



Observer patern je implementiran pomocu interfejsa IObserver koji sadrzi funkciju tj. metodu update(). Nasa kontejnerska klasa KoordinacioniCentar implementira ovaj interfejs iz razloga promjene atributa liste, jer bilo kakva promjena nad listama sistema se desava bas u ovoj klasi.
Prednost observer paterna jeste sto svaki put kada izvrsimo bilo kakvu modifikaciju nad listama klasa Migrant, PolicijskaStanica ili MigrantskiCentar, pored glavne funkcionalosti - modifikacije, pojavit ce se manje pop-up obavijesti zbog implementiranog interfejsa IObserver u navedenim klasama tj. zbog preklopljene metode update(). Tako da npr. kada dodajemo novu policijsku stanicu ili zahtjev ili vrsimo deportovanje migranta, prilikom uspjesno obavljene akcije izaci ce obavijest o ispunjenju iste jer ce se nakon svake operacije pozivati override-ana metoda.