

Patern izvještaj

Upotreba strukturnih, kreacijskih paterna i paterna ponasanja

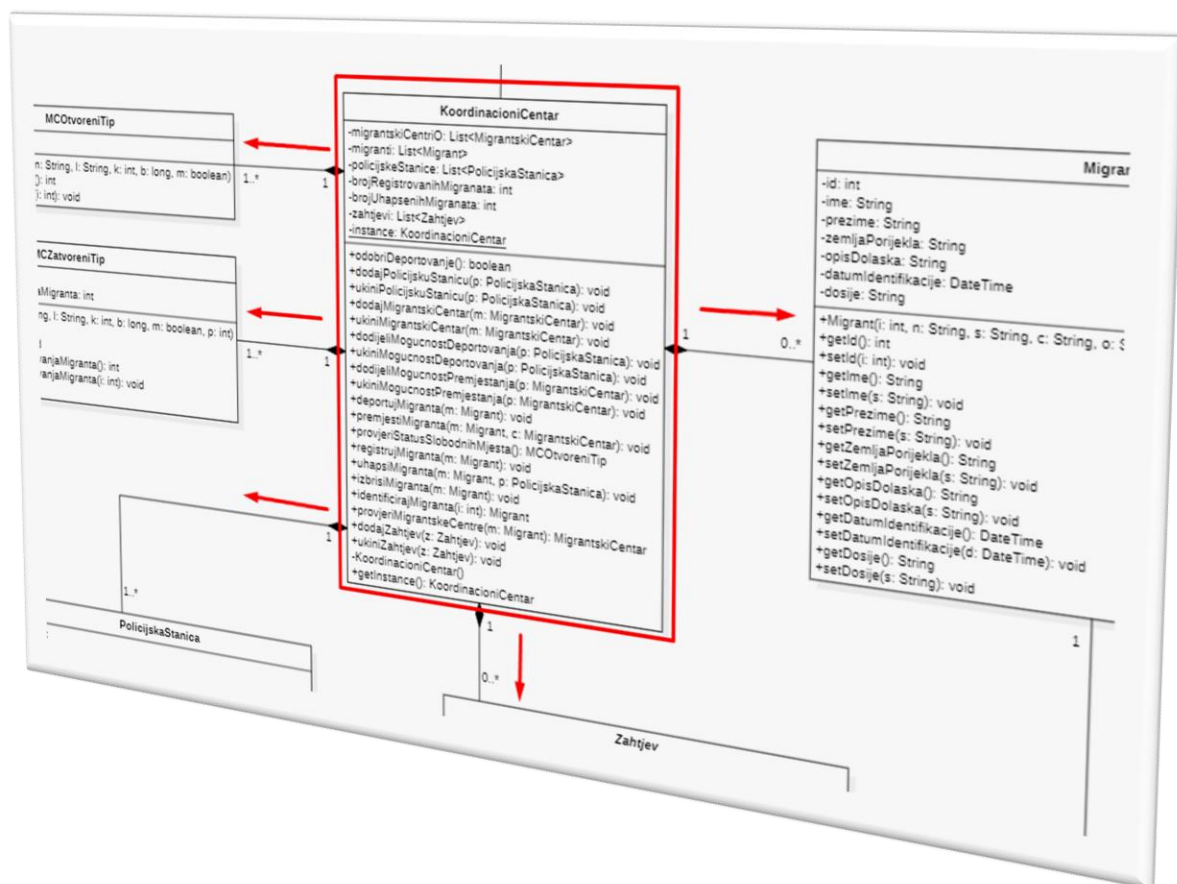
Grupa: Briferi

Ak. godina: 2019/20

Adapter pattern

Adapter pattern 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 učitajMigrantelzXml(XML migranti), koja učitava migrante iz xml formata. U slučaju da je potrebno 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 učitajMigrantelzJSON(JSON migranti) i klase AdministratorAdapter sa metodom učitajMigrantelzJSON(JSON migranti), koja će prvo učitati iz JSON-a i pretvoriti u XML, a zatim pozvati postojeću metodu učitajMigrantelzXml.

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 upućen kako se pojedine kompleksne metode odvijaju u pozadini jer je sve već implementirano i spremno za korištenje pozivom jedne kompleksnije metode.

Mana ovog pristupa jeste sto su odredjene metode kompleksnije, a prednost je u tome sto je korisniku olaksan poziv metoda i sto je sve upakovano u jednu klasu sa kojom komunicira korisnik.

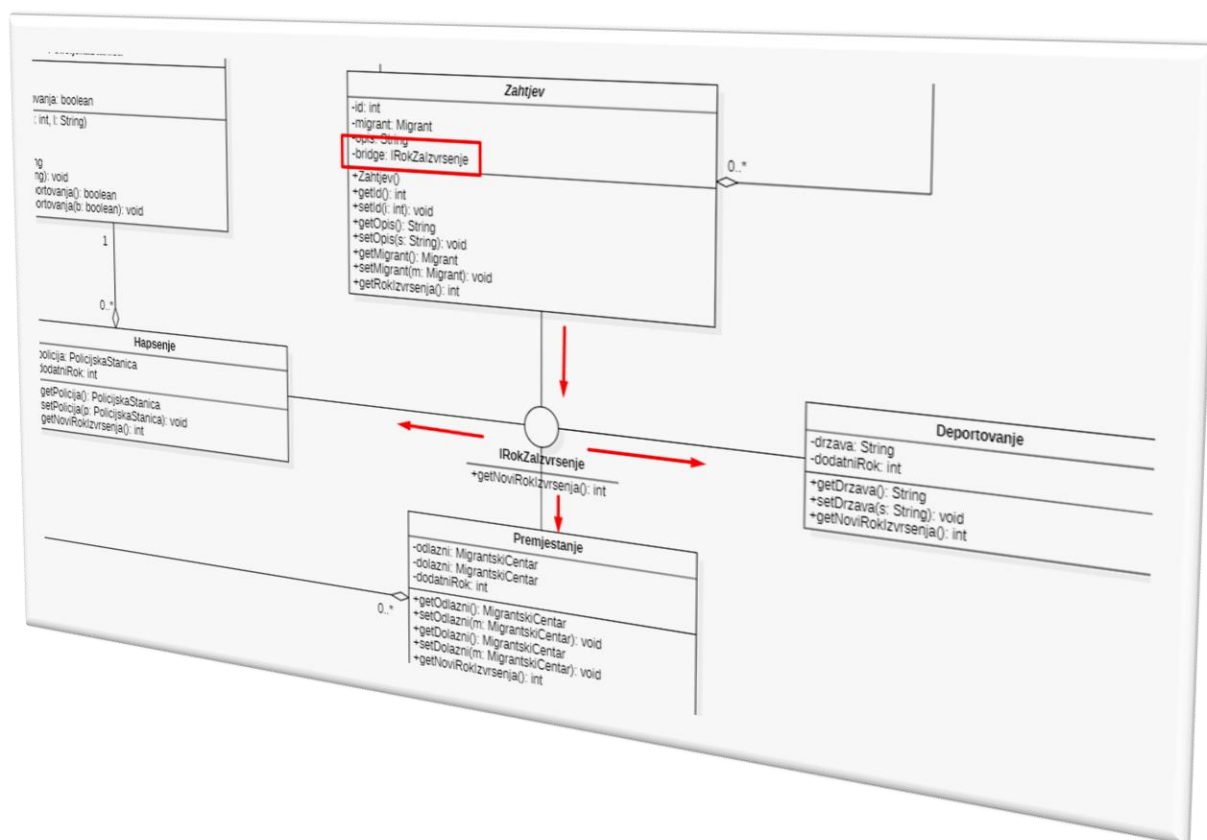
U konkretnom primjeru naseg sistema poziv metode deportujMigranta ili premjestiMigranta ce obaviti vise funkcija sa klasama MigrantskiCentar i njegovim podklasama, te sa klasom PolicijskaStanica koja se mora pripremiti te sprovesti navedenu akciju tj. zapovijed.

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 kompleksni 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 izvršenje zahtjeva nije isti za sve tipove zahtjeva. Medjutim razlog implementacije bas ovog patern jest sto je primarni rok za sve isti i on je 2 dana, ali svaki tip posebno ima dodatne dane za izvršenje zahtjeva u skladu sa kompleksnoscu zahtjeva tj. recimo cijeli proces deportovanja ce puno vise trajati nego obicno lociranje i hapsenje migranta.

Kreiran je interfejs IRokZalzvrsenje koji sadrzi metodu getNoviRokizvrsenja koju klase Hapsenje, Deportovanje i Premjestanje polimorfno nasljedjuju. Ta metoda služi za kalkulaciju glavnoj metodi zahtjeva gerRokizvrsenja na nacin koji je vec naveden iznad u tekstu.

Prednost ovog patern jest 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 `dajSveRokoveIzvršenja` unutar klase `KoordinacioniCentar`. Uvođenjem ovog paternu korisnik bi imao jasan pregled svih rokova izvršenja određenih zahtjeva. Ključni dio zbog kojeg je lako uvesti ovu metodu je postojanje interfejsa `IRokZalZvršenje`. Metoda `dajSveRokoveIzvršenja` bi pozvala metodu `getRokIzvršenja` za svaki zahtjev i prikazala unutar jednog stringa.

Prednost ovog paternu je što bismo u metodi `dajSveRokoveIzvršenja` pristupali na isti način za svaki zahtjev.

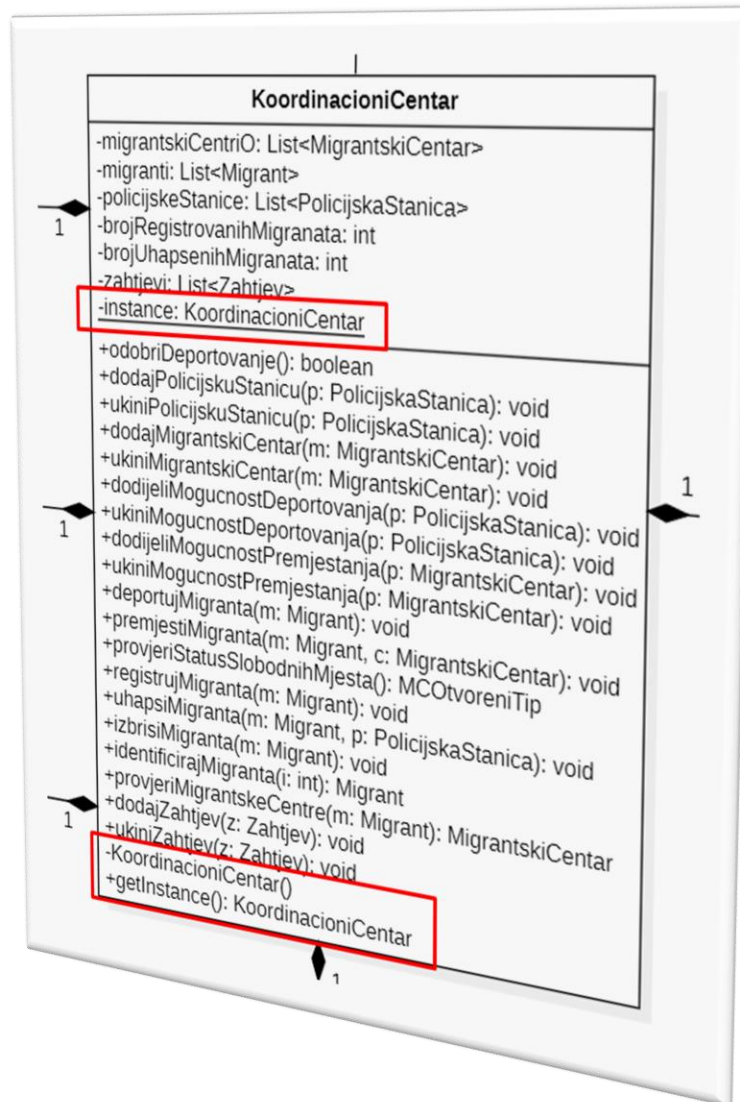
Proxy patern

Proxy patern omogućava kreiranje surogata originalnog objekta, pomoću kojeg se kontroliše pristup originalnom objektu. U našem projektu ovaj patern nije implementiran. Proxy patern 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 klase `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 patern 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 cuvala 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 različite jezike. Za implementaciju ovog paternu potreban je interfejs `IInformacije` sa metodom `dajJezik()`. Klasa `Informacije` bi implementirala ovaj interfejs, a lista sa instancama klase `Informacije` bi se nalazila u klasi `Administrator`.

Singleton patern



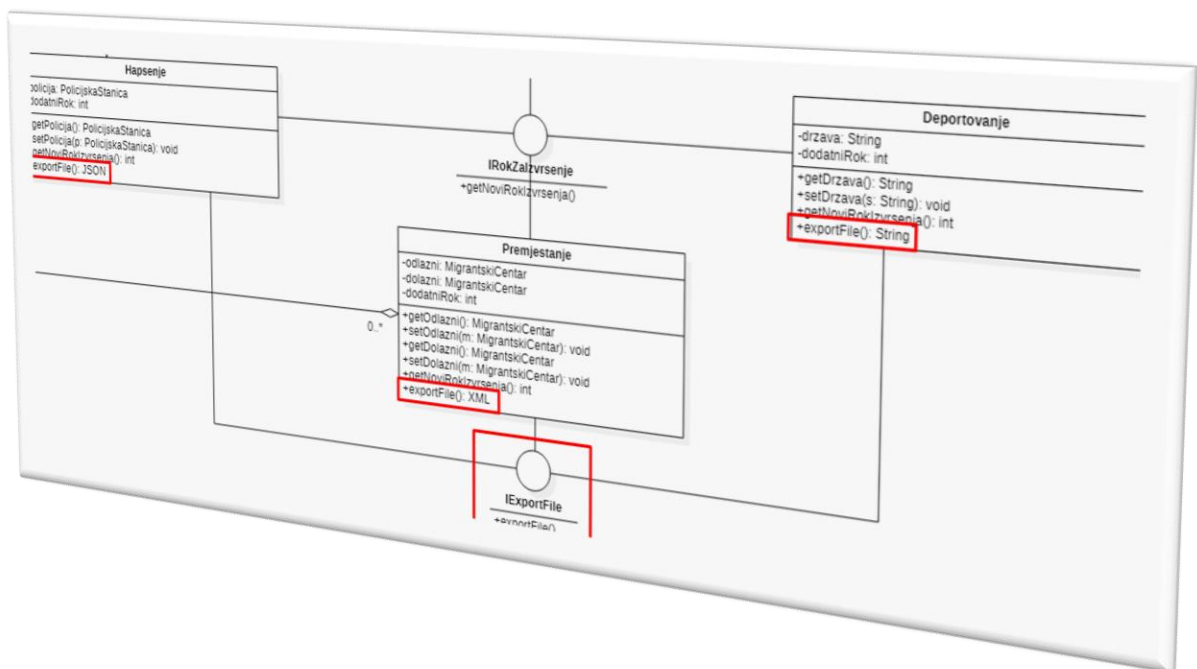
Singleton patern je implementiran u klasu **KoordinacioniCentar** (Administrator) pomocu static varijable ovog tipa, privatnog konstruktora i metode **getInstance**. Razlog uvođenja staticne varijable ovog tipa jeste da bi svaki poziv unutar koda ove instance imao istu referencu na ovu varijablu. Privatni 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 singletona 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 singleton patern predvidjen od samog starta implementacije sistema.

Prototype patern

Uloga Prototype patern 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 metode exportFile i interfejsa IExportFile. To je dakle polimorfna metoda koja vraća različite vrste objekata tj. tekstualnih dokumenata koji opisuju attribute i općenito naše zahtjeve koji će biti kreirani. Povratni tekstualni objekti se razlikuju u zavisnosti od podklase zahtjeva (Hapsenje, Deportovanje i Premjestanje).

Naša klasa Zahtjev predstavlja objekat tj. klasu koja je na većem nivou hijerarhije te ona može primiti različite objekte (zahtjeve). Metode su napravljene da same vraćaju različite tekstualne objekte međutim može se također uvesti i privatni atribut u baznu klasu koji će se instancirati onim tekstualnim tipom u zavisnosti koja od ove 3 funkcije će biti pozvana i koji je tip zahtjeva u pitanju tj. koja podklasa.

Prednost ove factory metode jeste što sa jednom polimornom funkcijom za različite klase dobijamo različite rezultatne 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 IRokZalzvrsenje ostatak uslova za implementaciju Abstract Factory patern bi bio zadovoljen. Takođe potrebna je polimorfna metoda dajZahtjev koja vraća određene instance Factory klasa.

Prednost Abstract Factory patern 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.

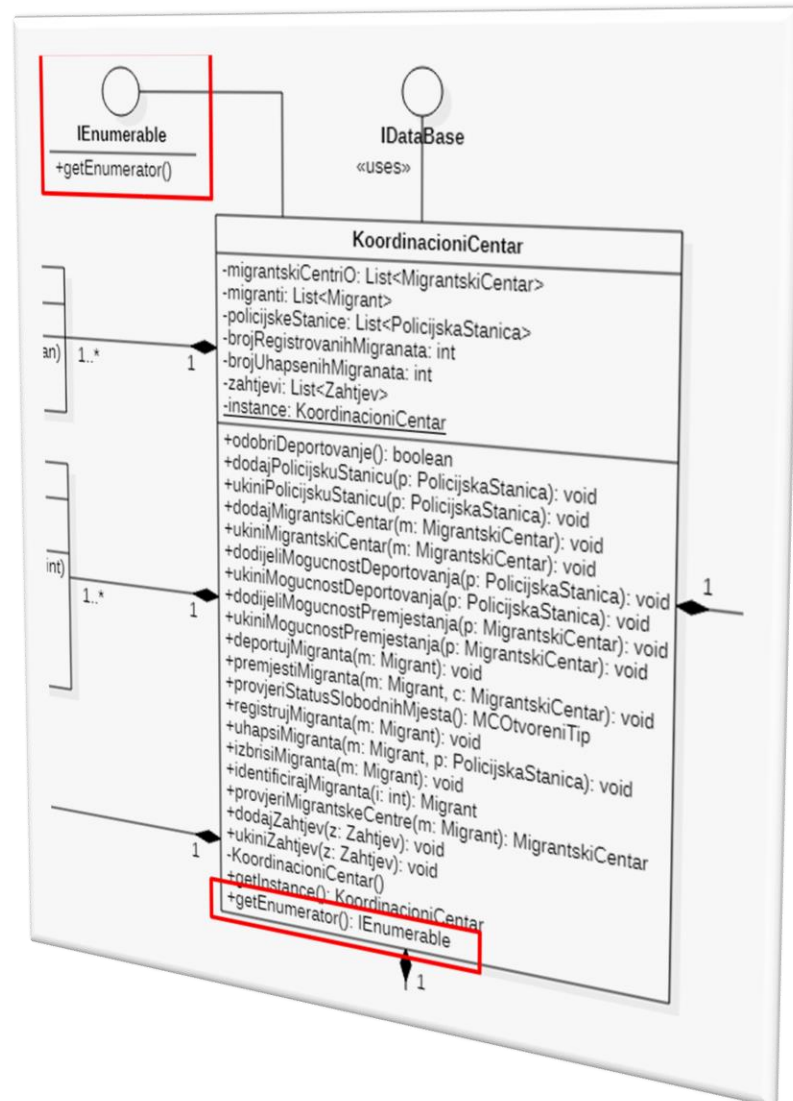
Uvođenjem Builder patern ne bi bilo potrebe za korištenjem konstruktora, već bi se to riješilo korištenjem jednostavnih metoda unutar buildera.

[Strategy patern](#)

[State patern](#)

[TemplateMethod patern](#)

[Iterator patern](#)

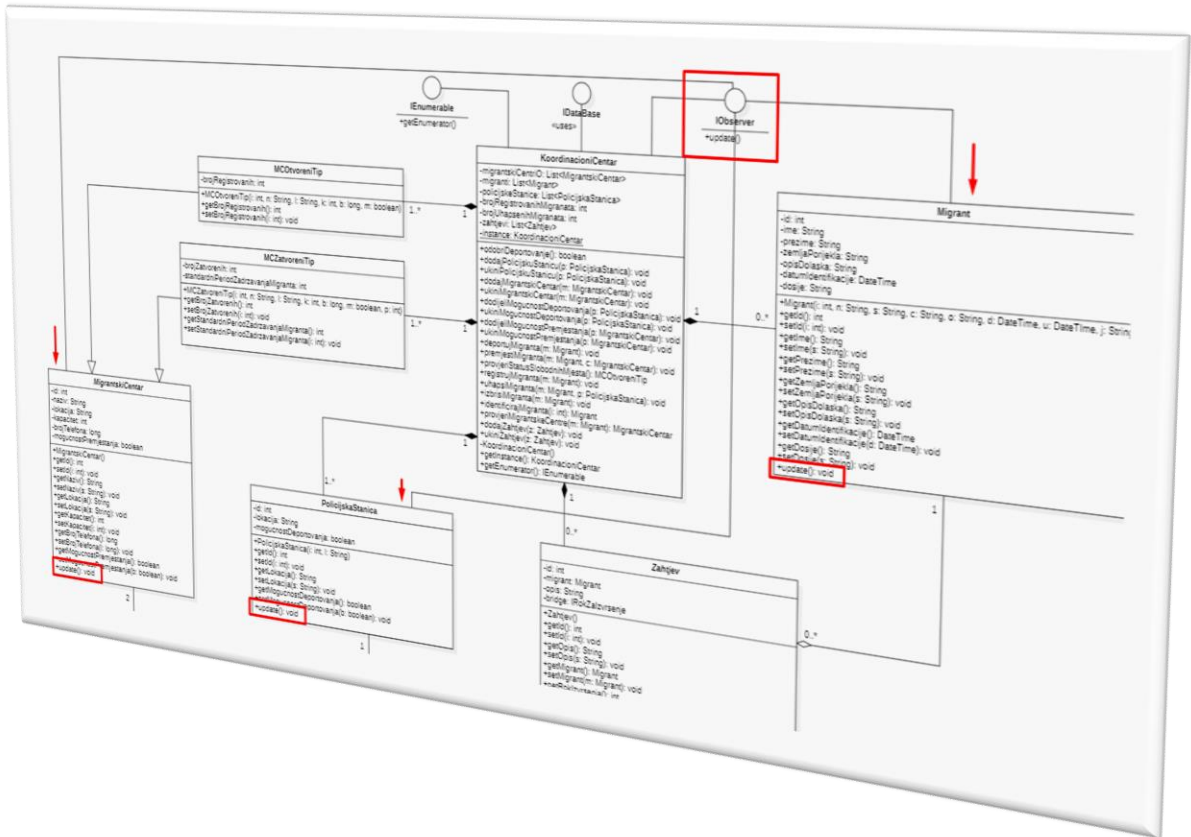


Iterator patern je implementiran pomocu interfejsa IEnumerabile koji sadrzi funkciju GetEnumerator. Ovaj interfejs implementira nasa kontejnerska klasa KoordinacioniCentar, koja je idealno mjesto za paterne ponasanja budući da sadrži sve attribute i liste, i u njoj se override-a već spomenuta metoda GetEnumerator.

Prednost ovog paternu jeste što možemo implementirati vlastitu GetEnumerator metodu koja će vrsiti iteriranje kroz kontejnere tj. liste naše 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 će biti sortirana po državi porijekla migranata gdje će pored migrantovih podataka biti ispisan i migrantski centar u kojem je isti lociran.

Observer pattern



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 paternu jeste sto svaki put kada izvorsimo bilo kakvu modifikaciju nad listama klasa `Migrant`, `PolicajskaStanica` 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.