*Nazivi paterna označenih rozom bojom,označavaju paterne implementirane na dijagramu klasa priloženom u folderu pod nazivom “Paterni”.Također,na dijagramu je pokušano da se implementira dosta paterna,što ne znači da će svi biti implementirani kada se zaista krene sa prebacivanjem ovih ideja u kod.Razlog je u tome što možda nisu potrebni svi paterni koji se sada nalaze na dijagramu i što bi možda implementacija svih navedenih dovela do rušenja kompletnosti i pojavljivanja krhkih dijelova u programu.

STRUKTURALNI PATERNI

NAZIV PATERNA	OPIS
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. Na taj način obezbjeđuje se da će se objekti i dalje moći upotrebljavati na način kako su se dosad upotrebljavali, a u isto vrijeme će se omogućiti njihovo prilagođavanje novim uslovima. Ovaj patern bi se u našem slučaju mogao iskoristiti kod plaćanja,jer može se desiti da neko ko živi van države planira doći na neki događaj kod nas,a želi da mu se vrijednost karte iz KM preračuna u eure I da onda izvrši transakciju.Za potrebe ovog slučaja, moraćemo dodati jedan interfejs za obračun rezervacije u EUR,klasu Adapter koja će služiti da implementira metodu iz interfejsa I to će biti povezano sa klasom Transakcija,jer će ta implementirana metoda pozvati metodu za obračunavanje iz klase Transakcija,naravno nakon izvršene konverzije valuta.Isto tako,ovaj adapter nam nudi mogućnost nadogradnje koda,ukoliko se ubace još neke valute u sistem transakcija.
FACADE PATERN	Fasadni patern služi kako bi se klijentima pojednostavilo korištenje kompleksnih sistema. Klijenti vide samo fasadu, odnosno krajnji izgled objekta, dok je njegova unutrašnja struktura skrivena. Na ovaj način smanjuje se mogućnost pojavljivanja grešaka jer klijenti ne moraju dobro poznavati sistem kako bi ga mogli koristiti. Ovaj patern se u našem dijagramu može povezati sa klasom EventManager,koja omogućava izlistavanje podataka o korisnicima,a korisnika u sistemu ima više vrsta,te sa tim u vezi ima više relevantnih atributa koji se moraju prikazati za svakog korisnika,a onaj ko želi da vidi podatke,ne treba da se zamara sa atributima npr.klase FizickoLice I klase Ustanova.

	Ovaj patern se u našem dijagramu može povezati sa klasom Korisnik,jer ona objedinjuje više vrsta korisnika,koji je svaki specifičan na svoj način,a svi posjeduju nekoliko zajedničkih atributa. U našem dijagramu,primjenu ovog paterna vidimo kod klase za obavještavanje korisnika,jer u njoj prosto postoje metode za slanje obavijesti za generički tip zvani Korisnik I korisnik aplikacije ne treba da unosi sve podatke za korisnika kojem želi poslati obavijest već je dovoljan samo ID ili ime.Uz to,ukoliko se iskoristi decorator patern,kada želimo da se pošalje obavijest korisniku,korisnik ne treba da se zamara sa tim kakve vrste obavijesti prima korisnik kojem šalje obavijest-da li je to putem emaila,Faceboka i sl.
DECORATOR PATERN	Decorator patern služi za omogućavanja različitih nadogradnji objektima koji svi u osnovi predstavljaju jednu vrstu objekta (odnosno, koji imaju istu osnovu). Umjesto da se definiše veliki broj izvedenih klasa, dovoljno je omogućiti različito dekoriranje objekata (tj. dodavanje različitih detalja), te se na taj način pojednostavljuje i rukovanje objektima klijentima, i samo implementiranje modela objekata. U našem dijagramu bi se ovaj patern mogao iskoristiti u slučaju klase Obavijest na sljedeći način:Ukoliko želimo da korisnika obavještavamo putem emaila,SMS,Facebooka,moguće je ukoliko imamo interfejs IObavjestavanje koji će sadržati definiciju metode za obavještavanje posaljiObavijest I biće dodane još četiri klase EmailObavijest,SMSObavijest, FacebookObavijest i klasa EMObavjestavanje,koja je vezana za obavjestavanje u našoj aplikaciji.
BRIDGE PATERN	Bridge patern služi kako bi se apstrakcija nekog objekta odvojila od njegove implementacije. Ovaj patern veoma je važan jer omogućava ispunjavanje Open-Closed SOLID principa, odnosno uz poštivanje ovog paterna omogućava se nadogradnja modela klasa u budućnosti te osigurava da se neće morati vršiti određene promjene u postojećim klasama. U našem dijagramu vidimo potencijalno mjesto za iskorištavanje ovog paterna-kod transakcije,pojavljuje se interfejs za plaćanje.Pošto nije isti obračun za fizičko lice sa običnim profilom I sa VIP profilom,zato će trebati

	izmijeniti stanje u ovom dijelu dijagrama na način da se doda interfejs za obračun,koji će sadržavati definiciju metode za obračun rezervacije.Dodati novu klasu Bridge I njoj će jedino moći pristupiti klasa Transakcija.
COMPOSITE PATERN	Composite patern služi za kreiranje hijerarhije objekata. Koristi se kada svi objekti imaju različite implementacije nekih metoda, no potrebno im je svima pristupati na isti način, te se na taj način pojednostavljuje njihova implementacija.U našem dijagramu se može pronaći potencijalno mjesto za korištenje ovog paterna-kod klase EventManager,ova klasa ima mogućnost da pravi izvještaje.Isto tako,mogućnost pravljenja izvještaja imaju klase Ustanova,Rezervacija,Transakcija,VIPKorisnik.Pošto bi to trebali biti generički izvještaji za svaku ovu klasu,tj.izvještaj o instanci te klase za njen život unutzar aplikacije i najosnovnije podatke,moguće je sve ove interfejse spojiti u jedan interfejs IIzvjestaj,te povezati ga sa svim ovim klasama.Tako ako bi neko htio da vidi sve izvještaje o radu aktera u aplikaciji,to bi bilo moguće nakon realizacije ovog paterna.
PROXY PATERN	Proxy patern služi za dodatno osiguravanje objekata od pogrešne ili zlonamjerne upotrebe. Primjenom ovog paterna omogućava se kontrola pristupa objektima, te se onemogućava manipulacija objektima ukoliko neki uslov nije ispunjen, odnosno ukoliko korisnik nema prava pristupa traženom objektu. Ovaj patern će biti iskorišten kod ispitivanja prava pristupa,konkretno,admin može pristupiti svim userima I svim njihovim podacima koji su unešeni u bazu podataka,dok korisnik ima pravo da pregleda sve svoje informacije,a samo osnovne podatke od drugih korisnika.Zato će nam trebati klasa Proxy,koja će vršiti autentifikaciju putem metode pristup() I ova klasa će naslijediti metode iz interfejsa IProxy.Isto tako,ovaj patern se može iskoristiti kod komentara-gost može da čita komentare na događaje,dok logovani korisnik,bilo FizickoLice ili Ustanova ili Admin,mogu I da čitaju I da pišu komentare.Tako da će biti potreban interfejs IProxyCom,a njegove metode će da implementira klasa ProxyCom,koja će vršiti autentifikaciju putem metode pristup() za pristupu komentarima.

FLYWEIGHT PATERN	Flyweight patern koristi se kako bi se onemogućilo bespotrebno stvaranje velikog broja instanci objekata koji svi u suštini predstavljaju jedan objekat. Samo ukoliko postoji potreba za kreiranjem specifičnog objekta sa jedinstvenim karakteristikama (tzv. specifično stanje), vrši se njegova instantacija, dok se u svim ostalim slučajevima koristi postojeća opća instanca objekta (tzv. bezlično stanje). Korištenje ovog paterna veoma je korisno u slučajevima kada je potrebno vršiti uštedu memorije. U našem primjeru kada bi se enumeracije za TipKarte,TipFizickogLica,VrstePlaćanja pretvorile u klase,njihovi elementi bi bile klase sa istim osobinama.Tako bi nam ovdje trebao interfejs,recimo za vrste karata,IKarte I koja će imati definiciju metode za razlikovanje vrste karata,a u klasi Rezervacija samo dodati metodu koja će se pozvati kod biranja vrste karte U našem dijagramu,ovaj patern ima potencijalno mjesto za svoju implementaciju-kod klase Admin,postoji interfejs za odobravanje zahtjeva koje dobije admin.Pošto admin može davati više vrsta odobrenja-odobrenje za kreiranje događaja,računa,odobrenje za izvršenje uplate,transakcije,rezervacije,...,ovaj interfejs se može povezati sa još par klasa koje bi se trebale implementirati,npr.klasa DogadjajOdobrenje,RezervacijaOdobrenje,Racun Odobrenje,TransakcijaOdobrenje,a klasa Admin bi se trebala povezati sa klasom Odobrenja(ovaj slučaj nije implementiran).Takvo slično potencijalno mjesto je kod klase EventManager,jer je ona povezana sa interfejsom za davanje aplikativnih odobrenja.Tako bi se mogla klasa EventManager povezati sa klasom Odobrenje,ta klasa sa ovim interfejsom koji već postoji,te kreirati klase OdobriZahtjevAplikativnoDogadjaj, OdobriZahtjevAplikativnoRezervacija, OdobriZahtjevAplikativnoKorisnik.

KREACIJSKI PATERNI

NAZIV PATERNA	OPIS
SINGLETON PATERN	Singleton patern služi kako bi se neka klasa mogla instancirati samo jednom. Na ovaj način može se omogućiti i tzv. lazy initialization, odnosno instantacija klase tek onda kada se to prvi put traži. Osim toga, osigurava se i globalni pristup jedinstvenoj instanci - svaki put kada joj se pokuša pristupiti, dobiti će se ista instanca klase. Ovo olakšava i kontrolu pristupa u slučaju kada je neophodno da postoji samo jedan objekat određenog tipa. U našoj aplikaciji bi se ovaj patern mogao iskoristiti kod klase Admin,ali kod nas je moguće da postoji više admina,tako da se neće dobiti jedinstvena instanca. Mjesto koje bi moglo da ispuni ovaj patern je klasa BazaPodataka,jer je to jedinstvena baza za cijelu aplikaciju i ova klasa bi se instancirala samo jednom.Zbog toga,u ovu klasu će se dodati statički atribut BazaSingleton,statičku metodu koja će vršiti vraćanje statičkog atributa,te u ovoj klasi će postojati metode za pristup bazi.
PROTOTYPE PATERN	Prototype patern omogućava smanjenje kompleksnosti kreiranja novog objekta tako što se uvodi operacija kloniranja. Na taj način prave se prototipi objekata koje je moguće replicirati više puta a zatim naknadno promijeniti jednu ili više karakteristika, bez potrebe za kreiranjem novog objekta nanovo od početka. Ovime se osigurava pojednostavljenje procesa kreiranja novih instanci, posebno kada objekti sadrže veliki broj atributa koji su za većinu instanci isti. U našem slučaju moguće je kreirati isti događaj,u isto vrijeme,ali da imaju drukčiju publiku:npr.kino kreira događaj za premijeru filma.Premijera se dešava na datum xx.yy.zzzz,a kino kreira dva takva događaja,za isti film i za isti datum.Jedina razlika između ta dva događaja jeste ta što je jedan događaj vezan za salu A,gdje je premijera za posebne zvanice i drugi događaj je vezan za salu B,gdje je premijera za sve ljude koji žele doći na premijeru.Nakon što se kreiraju ta dva događaja,moguće je za premijeru koja je u sali

	B,povećati ili smanjiti kapacitet karata za prodaju ili vrijeme događaja (tipa prologirati za pola sata i sl.),bez da to utiče na događaj za ljude koji idu u salu A.
FACTORY METHOD PATERN	Factory method patern služi za omogućavanje instanciranje različitih vrsta podklasa koristeći factory metodu koja odlučuje koja će se podklasa instancirati i koja programska logika izvršiti. Na ovaj način osigurava se ispunjavanje O SOLID principa, jer se kod za kreiranje objekata različitih naslijeđenih klasa ne smješta samo u jednu zajedničku metodu, već svaka podklasa ima svoju logiku za instanciranje željenih klasa, a samo instanciranje kontroliše factory metoda koju različite klase implementiraju na različit način. Ovaj patern je u našem dijagramu već potencijalno primijenjen kod klase Korisnik,ona sadrži metode koje nasljeđuje svaka naslijeđena klasa,samo što bi trebalo ku klasu Korisnik dodati factory metodu,kreirati interfejs IKorisnikFactory,koji će zamijeniti metode za kreiranje računa,a taj interfejs naslijediti klase FizickoLice,Ustanova,Admin.
ABSTRACT FACTORY METHOD PATERN	Abstract factory patern služi kako bi se izbjeglo korištenje velikog broja if-else uslova pri kreiranju različitih hijerarhija objekata. Ukoliko postoji više tipova istih objekata te različite klase koriste različite podtipove, te klase postaju fabrike za kreiranje objekata zadanog podtipa bez potrebe za specificiranjem pojedinačnih objekata. Na ovaj način se, korištenjem nasljeđivanja, ukida potreba za postojanjem if-else uslova jer određeni tip fabrike sadrži određene tipove objekata i zna se tačno koju podklasu će instancirati. Ovaj patern bi se mogao u našem slučaju iskoristiti kod rezervacija,jer postoji više vrsta karata u zavisnosti od vrste fizičkog lica-obična,studentska,VIP,penzionerska.Stoga,ne bi bilo loše posjedovati interfejs IFactoryVrste,koji će omogućiti implementiranje metoda koje su zajedničke za sve fabrika-klase.Zajedničke metode di bile dajPopust(),dajOznaku().Isto tako,trebale bi se kreirati klase npr.StudentFactory,koja bi odjedinjavala atribute i metode vezane za studenta,a to su studentska karta i student kao tip fizičkog lica.Pošto se u već

	nekom od prijašnjih paterna iskoristila slična osobina,ovaj patern se neće implementirati na dijagramu.Sa druge strane,uvođenje još klasa bi možda još više zakomplikovalo implementaciju i možda narušilo kompletnost sistema.
BUILDER PATERN	Builder patern služi za apstrakciju procesa konstrukcije objekta, kako bi se kao rezultat mogle dobiti različite specifikacije objekta koristeći isti proces konstrukcije. Ovaj patern koristi se kako bi se izbjeglo kreiranje kompleksne hijerarhije klasa te kako bi se izbjegao kompleksni programski kod konstruktora jedne klase koja može imati različite konfiguracije atributa. Različiti dijelovi konstrukcije objekta izdvajaju se u posebne metode koje se zatim pozivaju različitim redoslijedom ili se poziv nekih dijelova izostavlja, kako bi se dobili željeni različiti podtipovi objekta bez potrebe za kreiranjem velikog broja podklasa. Ovaj patern bi se u našem slučaju mogao iskorititi kod filtriranja događaja u novostima korisnika-korisnik može da postavi zahtjeve kako želi da mu se izlistavaju aktuelni događaji,dok može da postavi da sistem automatski izbacuje događaje na dashboard.Za ovaj slučaj,biće potrebno kreirati interfejs IBuilderFilter,te kreirati dvije nove klase ManuelniFilter i AutomatskiFilter,te ih povezati sa interfejsom,a interfejs sa klasom EventManager,zbog atributa lista dogadjaja.