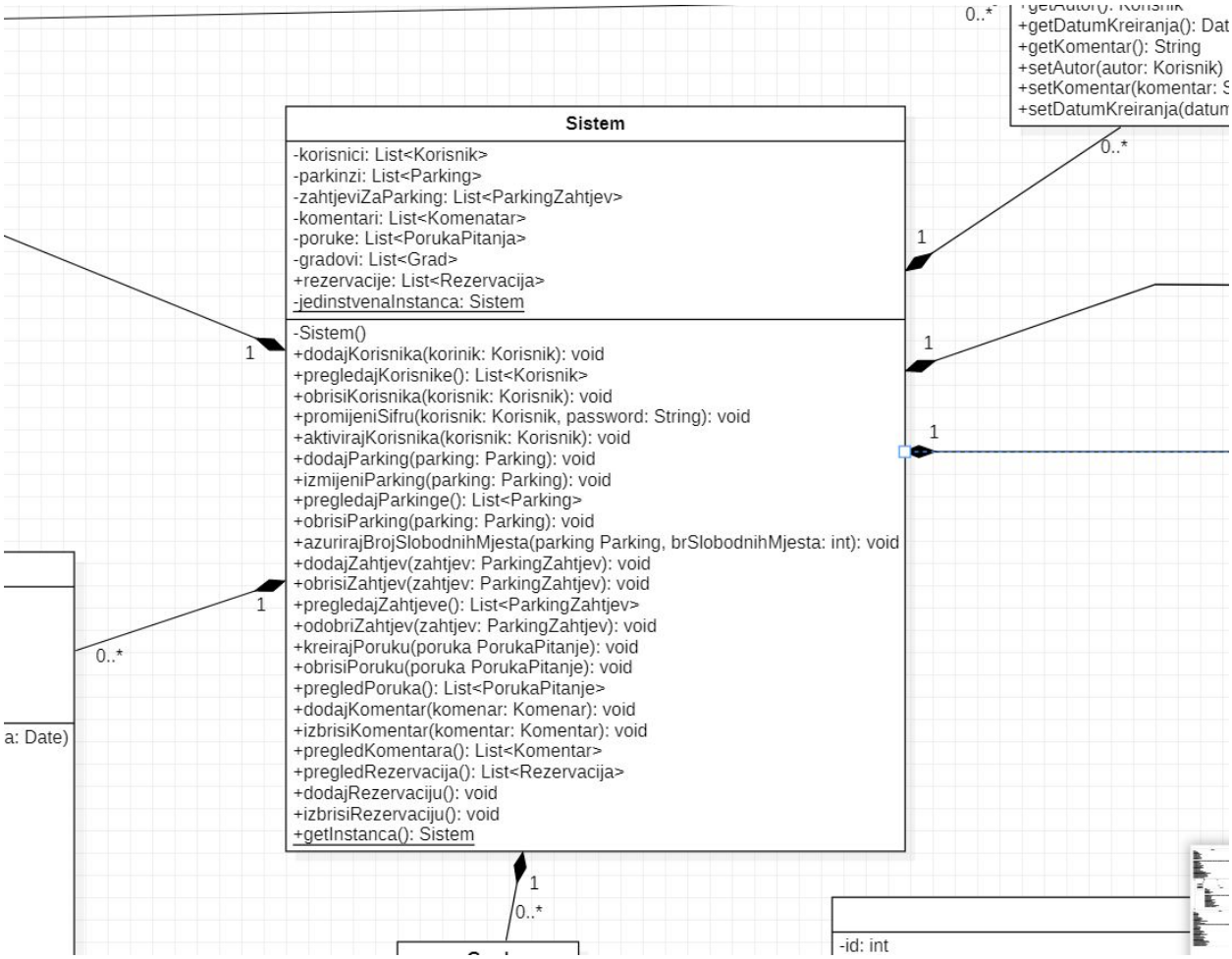
Kreacijski paterni

Singleton patern – implementirana u sistemu

- Uloga Singleton paterna je da osigura da se klasa može instancirati samo jednom i da osigura globalni pristup kreiranoj instanci klase. Instanciranjem više nego jednom navedenih objekata mogu se prouzrokovati problemi kao što su nekorektno ponašanje programa, neadekvatno korištenje resursa ili nekonzistentan rezultat. U našem sistemu Singleton patern je implementiran u klasi Sistem tako što smo dodali private static konstruktor i private static varijablu (jedinstvenaInstanca) koja čuva jednu/jedinstvenu instancu klase. Također smo dodali static metoda (getInstance) preko koje se pristupa Singleton klasi.



Prototype patern

- **Uloga Prototype paterna je da kreira nove objekte klonirajući jednu od postojeći prototip instanci.** Ovaj patern bi mogli implementirati u naš sistem tako što bi stavili da klase onih podataka koje čuvamo u bazi podataka, a kojima se frekventno pritupa, implementiraju neki interfejs IPrototype, koji bi nam omogućio da pravimo duboke kopije. Time ne bi morali stalno pristupati bazi kada želimo vršiti modifikaciju nad podacima koji su već učitani u kolekcije

unutar klase Sistem. Dovoljno bi bilo samo izvšiti duboku kopiju onih podataka koji su nam potrebni.

Factory Method patern

Abstract Factory patern

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.**

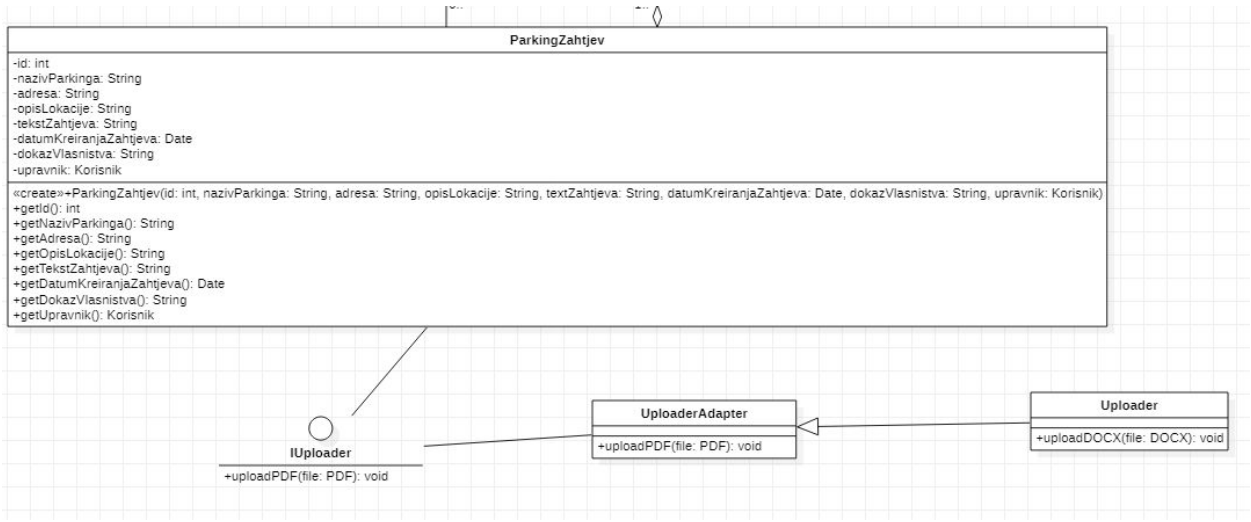
  Mi u našem sistemu nismo uspjeli naći niti jedan način da implementiramo ovaj patern (niti u trenutnoj implementaciji, niti u "hipotetičkoj" implementaciji sa brojnim proširenjima).

Strukturalni paterni

Adapter patern

- ***Osnovna namjena Adapter paterna je da omogući širu upotrebu već postojećih klasa. Adapter patern služi da se postojeći objekat prilagodi za korištenje na neki novi način u odnosu na postojeći rad, bez mijenjanja same definicije objekta.***

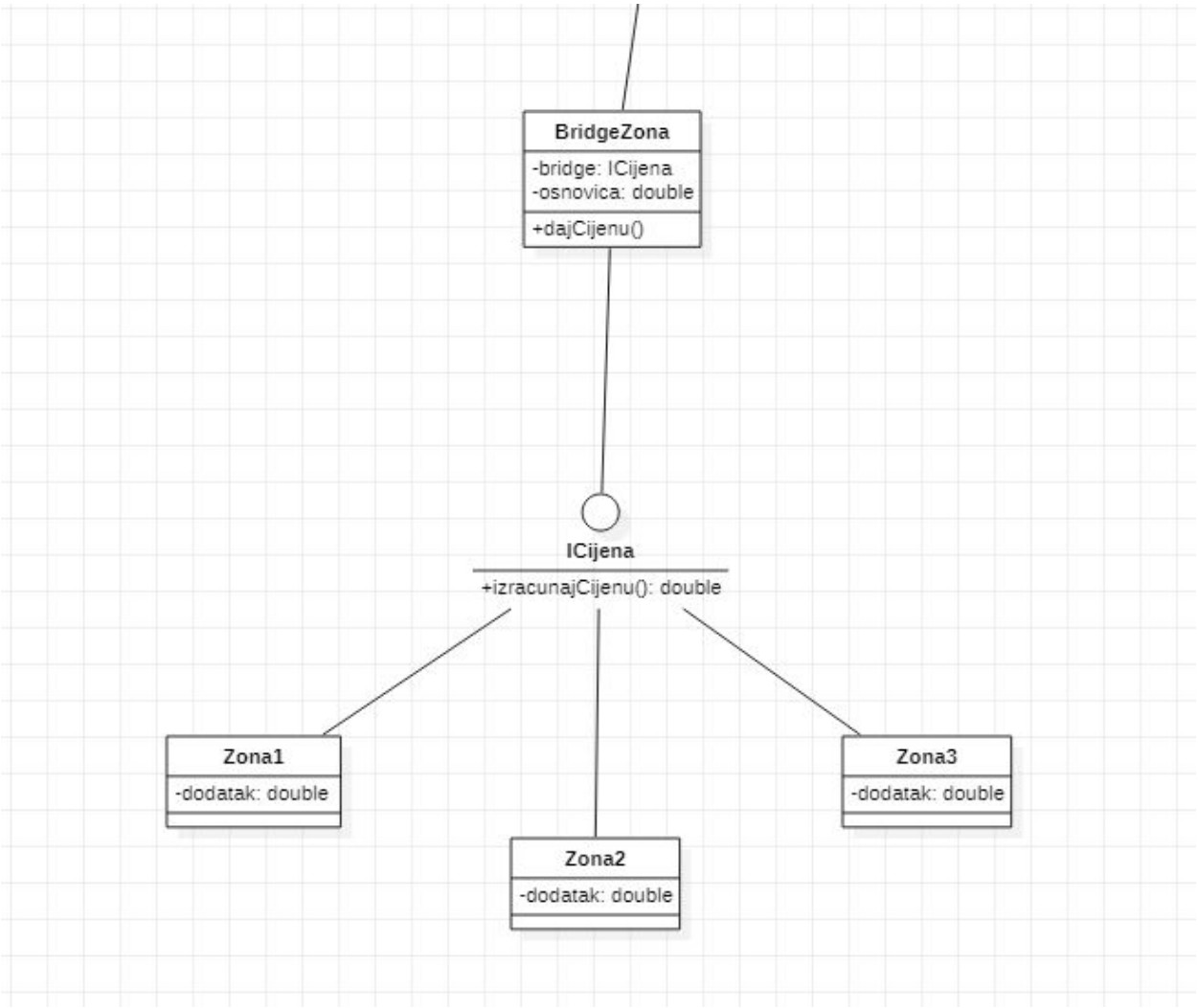
  U našem sistemu bi mogli implementirati adapter patern tako što kreiramo interfejs IUploader sa metodom uploadPDF i klasu UploaderAdapter. Tako omogućavamo korisniku da dokaz o vlasništvu parkinga ne mora biti striktno u pdf formatu, moguće je učitati i iz docx formata. Adapter patern nam omogućava da ovu funkcionalnost možemo ostvarili bez izmjene postojećeg objekta.



Bridge patern

- ***Bridge patern služi kako bi se apstrakcija nekog objekta odvojila od njegove implementacije***.

Kako bi implementriali ovaj patern u naš sistem smo dodali interfejs ICijena i BridgeZona klasu koja implementira interfejs. Interfejs sadrži metodu izracunajCijenu. Ovim smo iskoristili činjenicu da je osnovica jednaka za sve zone i da na osnovicu dodajemo dodatak za specifičnu zonu.



Composite patern

- ***Composite patern omogućava upravljanje jednostavnim objektima i njihovom kompozicijom. Ako objekti imaju različite implementacije nekih metoda, ali se svima pristupa na isti način, tada da bismo pojednostavili implementaciju koristimo composite patern.***

  Ovaj patern nismo implementirali u našem sistemu, ali da postoji potreba da računamo iznos cijene parkinga po zonama parkinga, a da pri tome nije ista osnovica koristili bi composite patern umjesto bridge paterna. Implementirali bi isto kao i Bridge patern s tim da ne bi postojala osnovica koja je jednaka za sve zone parkinga. Naš sistem ne bi mogao naći upotrebu composite paterna na drugom mjestu, pa smo razmatrali lokaciju gdje je prethodno iskoristen Bridge patern.

Decorator patern

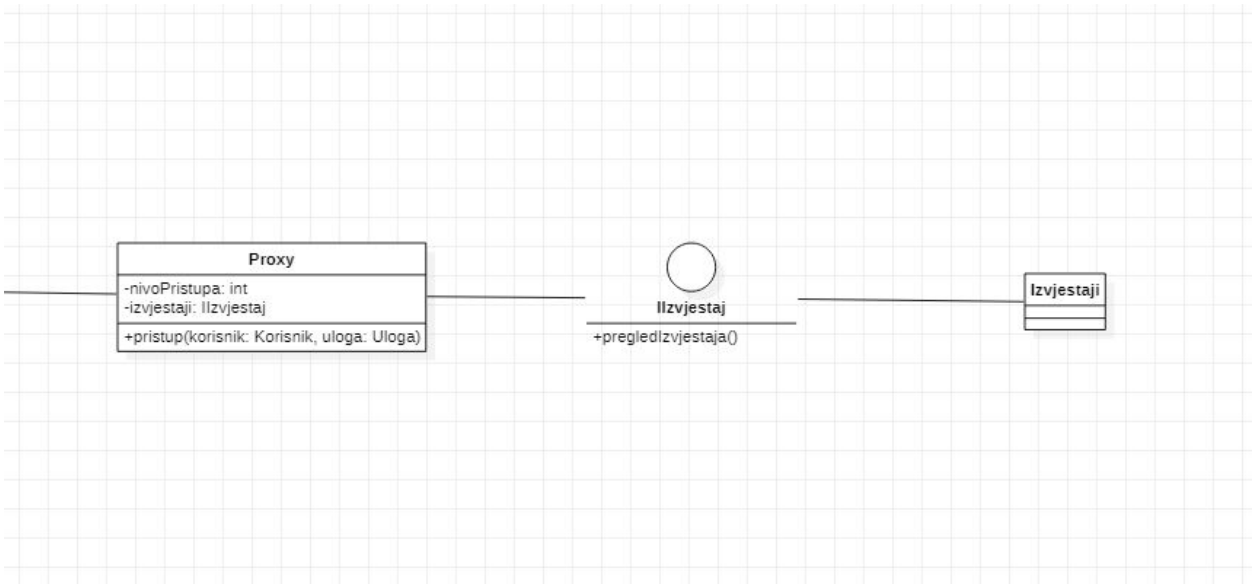
- **Decorator patern služi za omogućavanja različitih nadogradnji objektima koji svi u osnovi predstavljaju jednu vrstu objekta (odnosno, koji imaju istu osnovu). Umjesto da se definiše veliki broj izvedenih klasa, dovoljno je omogućiti različito dekoriranje objekata (tj. dodavanje različitih detalja), te se na taj način pojednostavljuje i rukovanje objektima klijentima, i samo implementiranje modela objekata**
  Trenutno nemamo potrebe za korištenjem ovog patterna, međutim kao što je objašnjeno na tutorijalu na primjeru slike, mogli smo upotrijebiti isti princip kod pisanja Poruke ili Komentara u samom editoru gdje bi se mogla editovati slika koju korisnik želi postaviti uz tekst poruke međutim to nije fokus naše aplikacije i to bi samo dovelo do takozvanog 'software bloat-a'.

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.***
  U našem sistemu ćemo iskoristiti ovaj pattern na više mjesta, konkretno najbolji primjer upotrebe je prikaz mape sa označenim parkinzima. Dakle, klijent nakon što unese lokaciju na kojoj želi naći slobodan parking, sistem će pozvati dio koda koji će pokupiti, analizirati i prikazati podatke sa nekog servisa (GoogleMaps, OpenLayers i sl.), a pri tome klijent ne treba poznavati o kojem se servisu radi niti kako se analiziraju podaci tj. dovoljno je da unese lokaciju.

Proxy patern

- ***Namjena Proxy paterna je da omogući pristup i kontrolu pristupa stvarnim objektima. On dodatno osigurava objekte od pogrešne i zlonamjerne upotrebe.***
  U našem sistemu smo implementriali ovaj metod tako što smo dodali funkcionalnost pregleda izvještaja upravniku parkinga. Definisali smo interfejs IIzvjestaj u okviru kojeg je diefinisana metoda za pregled izvještaja. Definisali smo klasu Proxy koja će implementirati interfejs Iizvjestaj i metodu pristup u kojoj postavljamo nivuPristupa na 1 ako je korisnik upravnik parkinga. U suprotnom nivoPristupa je 0 i korisnik nema pravo pregleda izvještaja.



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 sistemu trenutno nemamo upotrebu ovog patterna jer su sve instance svih klasa unikatne osim klase Sistem, međutim ona je pokrivena Singleton patternom. Mogli bismo upotrijebiti ovaj pattern ako npr trebamo kreirati notifikaciju/poruku za sve korisnike našeg sistema gdje je pošiljalac administrator a primaoci su svi korisnici, dakle nema potrebe kreirati posebnu notifikaciju za svakog korisnika posebno nego jednu notifikaciju koju će moći vidjeti svi korisnici.