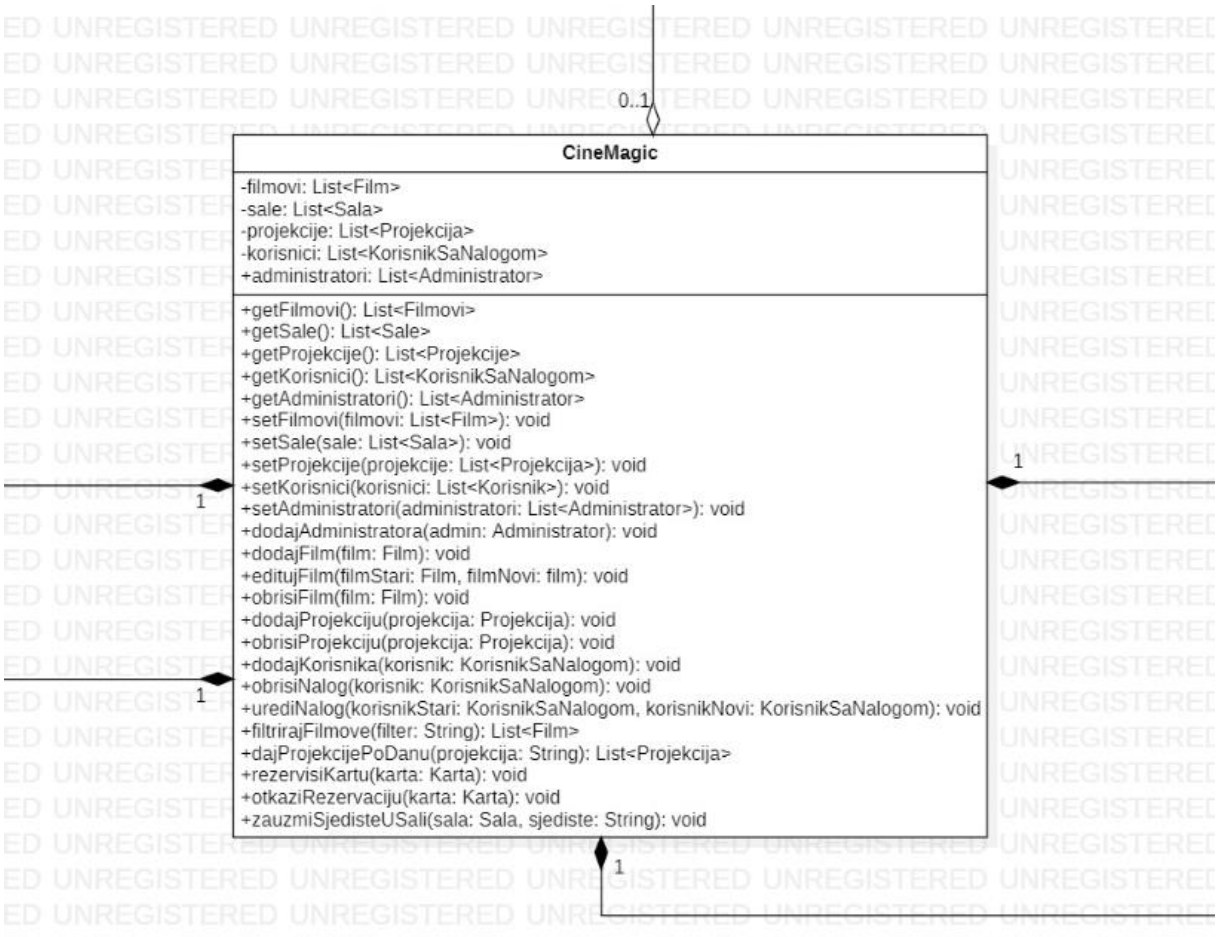
# STRUKTURALNI PATERNI

## 1. Adapter pattern

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. U našem programu nismo implementirali ovaj pattern, ali kada bi se odlučili da proširimo aplikaciju novim funkcionalnostima, to bi bilo u vidu omogućavanja korisniku da ocjeni i komentariše film koji je prethodno rezervisao. Za dodavanje ove funkcionalnosti bilo bi potrebno da napravimo Adapter klasu i da napravimo interfejs koji bi sadržavao metode potrebne za realizaciju ovih funkcionalnosti.

## 2. Facade pattern

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 pattern smo implementirali pomoću klasu CineMagic, na način da u sebi sadrži druge klase kao atribute. Pozivom jedne metode iz ove klase se poziva više različitih metoda iz različitih klasa, te se time izvršava kompleksniji proces u koji korisnik nema uvid.

Primjer ovoga bi mogla biti metoda *rezervisiKartu* u kojoj se izvršava i plaćanje karte, zauzimanje sjedišta, te još neke dodatne metode potrebne za rezervisanje karte.

## 3. Decorator pattern

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.

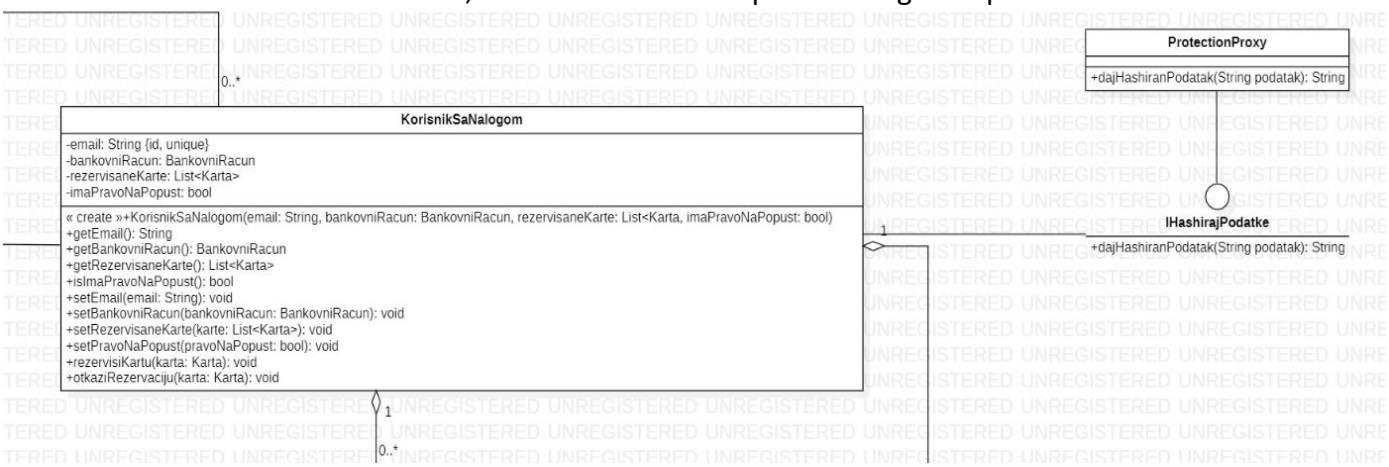
U našem sistemu trenutno nemamo potrebu za implementaciju ovog paterna. Ako bi dodali da cijena karte i izgled početne stranice zavise od određenih praznika, posebnih dana (godišnjica rada i sl.), iskoristili bi ovaj patern. Primjer za ovo je da bi za vrijeme novogodišnjih praznika cijene bile snižene za određeni procenat, te bi početna stranica imala temu prilagođenu ovim praznicima.

## 4. Bridge pattern

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 programu ovaj patern nije primijenjen, te nemamo mogućnost primijene istog.

## 5. Proxy pattern

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. Iskoristit ćemo ovaj pattern tako što ćemo dodati interfejs *IHashirajPodatke* koji će sve osjetljive podatke heširati te ih tako ubaciti u bazu, kako bi se sačuvali ti podaci i osigurala privatnost korisnika.



## 6. Composite pattern

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. Ovaj patern nećemo implementirati, ali kada bi se odlučili da proširimo aplikaciju, omogućili bi posebnu vrstu korisnika *VIPKorisnik*, koji je uplatio godišnju kartu, te ima posebne pogodnosti. Iako bi ova dva korisnika bila na različitim nivoima, pristupalo bi im se na isti način i implementacija bi bila pojednostavljena.

## 7. Flyweight pattern

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. Nismo implementirali ovaj pattern, međutim mogli bi primijeniti ovaj patern kada bi kreirali klasu *Gost*. Kada bi se *Gost* prijavljivao na sistem, ne bi svaki put kreirali novu instancu, već bi se instancirao samo jedan objekat ovog tipa, te bi se uštedila memorija.