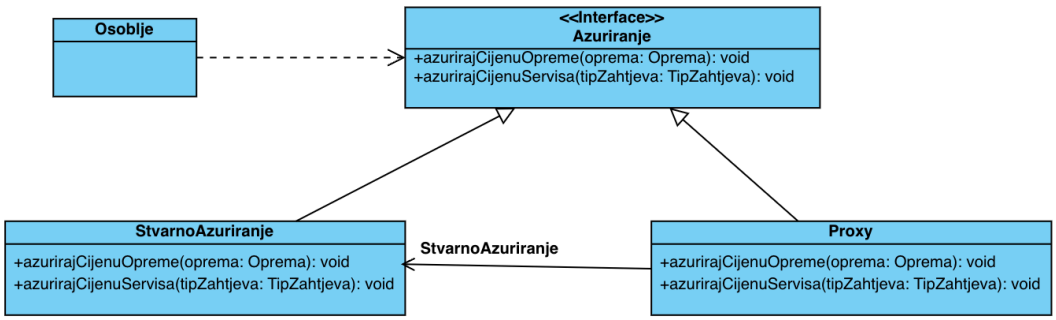
# Strukturalni design patterni

Za implementaciju odabrani su Proxy i Bridge design patterni.

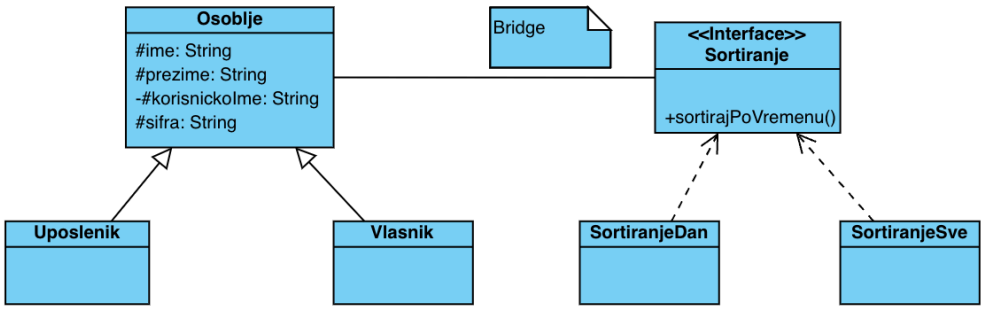
## 1. Proxy

Objekt klase *Oprema*, tačnije atribut cijena ovog objekta, može mijenjati samo vlasnik restorana, dok ostali korisnici ne mogu. Da bi se napravila restrikcija pristupa koristili bi Proxy pattern. Proxy objekt će različitim korisnicima omogućiti različit nivo pristupa štiteći sigurnost aplikacije.




## 2. Brigde

Osnovna namjena Bridge paterna je da omogući odvajanje apstrakcije i implementacije neke klase tako da ta klasa može posjedovati više različitih apstrakcija i više različitih implementacija za pojedine apstrakcije. U našem slučaju Implementor odnosno *Sortiranje* zaduženo je za konkretne implementacije. Klase koje implementiraju taj interfejs sadržavaće specifični kod implementacije, u zavisnosti da li je korisnik vlasnik ili uposlenik, imaće različit pregled zahtjeva.



## 3. Composite

Osnovna primjena Composite paterna jeste pravljenje hijerarhije klasa i omogućavanje pozivanja iste metode nad različitim objektima sa različitim implementacijama. U našem sistemu, ovaj pattern mogao bi se primijeniti ukoliko klasa *Oprema* bude Composite klasa koja ima svoje elemente (Listove) a klase koje su izvedene od *Oprema*, budu komponente odnosno Leaf klase. One će implementirati neki IComposite interfejs u kome će biti definisano defaultno ponašanje koje će biti zajedničko za pojedinačnu komponentu i njihovu definiciju.

## 4. Adapter

Adapter patern je design pattern čija je osnovna namjena da interfejs jedne klase pretvori u neki željeni interfejs kako bi se ona mogla koristiti u situaciji u kojoj bi inače problem predstavljali nekompatibilni interfejsi. Primjenom Adapter paterna dobija se željena funkcionalnost bez izmjena na originalnoj klasi i bez ugrožavanja integriteta cijele aplikacije. U našem sistemu možemo iskoristiti ovaj pattern tako što bi omogućili da *Osoblje* može mijenjati zahtjeve Klijenata. Onda bi napravili Adapter koji će mapirati interfejs Klijenta i prilagoditi ga za Osoblje.

## 5. Facade

Facade patern se koristi kada sistem ima više identificiranih podsistema pri čemu su apstrakcije i implementacije podsistema usko povezane. Osnovna namjena Facade paterna je da osigura više pogleda visokog nivoa na podsisteme. Ovaj patern sakriva implementaciju podsistema od korisnika i pruža pojednostavljeni interfejs putem kojeg korisnik pristupa sistemu. U našem sistemu ovaj pattern možemo iskoristiti ukoliko bi napravili fasadnu klasu za *Uposlenika* ili *Vlasnika* tako da sadrži listu svih klijenata, zahtjeva, odvojeno za različite usluge i tako im pružiti pogled na sve podsisteme.

## 6. Decorator

Osnovna namjena Decorator paterna je da omogući dinamičko dodavanje novih elemenata i ponašanja (funkcionalnosti) postojećim objektima. U našem sistemu ovaj pattern možemo iskoristiti ukoliko bi htjeli da dodamo neke vrste skija, pancerica i sl. Tada bi Decorator klase predstavljale proširenje izbora *Opreme.*

## 7. Flyweight

Flyweight patern koristi se kako bi se onemogućilo bespotrebno stvaranje velikog broja instanci objekata koji u suštini predstavljaju isti objekat. U našem sistemu ovaj pattern možemo iskoristiti ukoliko bi htjeli da sve objekte tipa *Zahtjev* pohranimo u neku listu koja bi mogla memorisati (cache) instance i omogućiti njihovo ponovno korištenje. Određeni korisnik će onda preko FlyweightFactory zahtijevati objekte.