Univerzitet u Sarajevu
Elektrotehnički fakultet u Sarajevu
Odsjek za računarstvo i informatiku

ReadTheWorld

Patterni ponašanja

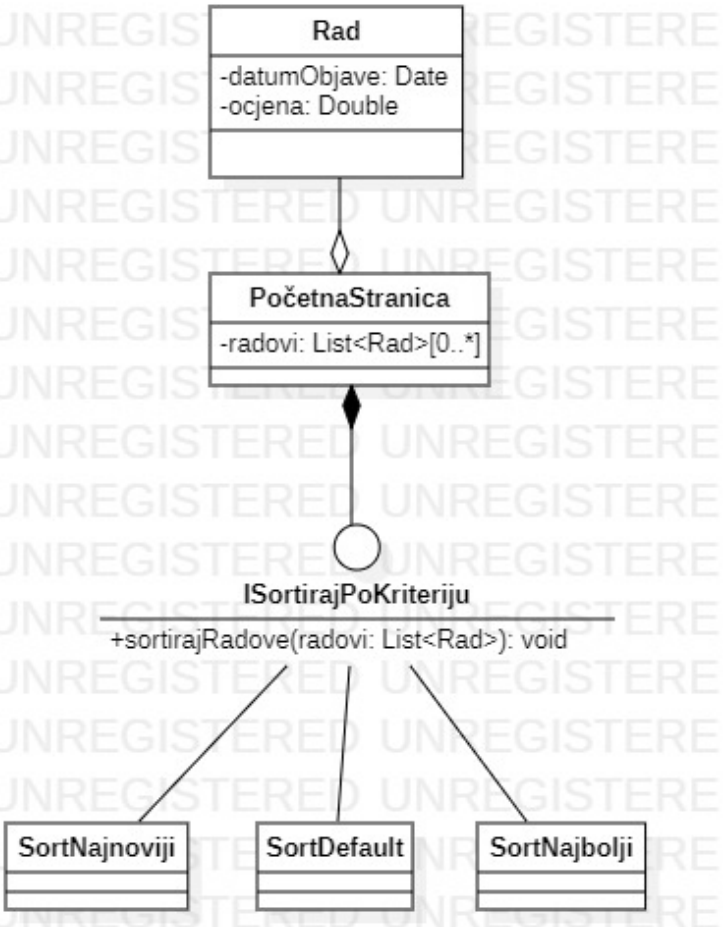
Objektno orijentisana analiza i dizajn

Iris Pjanić
Adna Husičić
Mirza Kadrić

Patterni ponašanja

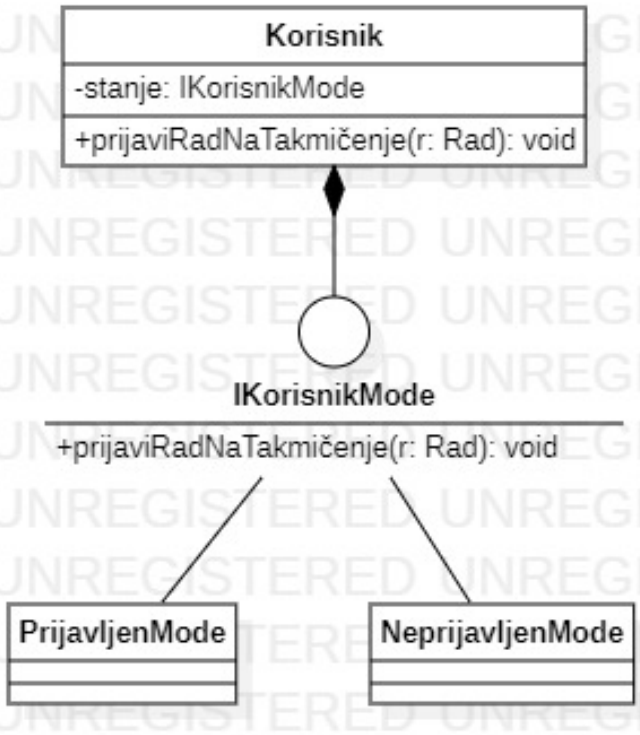
- **Strategy pattern**

  Strategy pattern koristimo kada postoji više primjenjivih algoritama za određeni problem. Ovim patternom izdvajamo algoritam iz matične klase, i uključujemo ga u posebne klase, čime omogućujemo da korisnik odabere koji algoritam želi koristiti. U našem sistemu, ovaj pattern možemo iskoristiti da realiziramo sortiranje radova na početnoj stranici po kriterijima koje korisnik odabere. Iako u svim slučajevima koristimo isti algoritam sortiranja, svaka verzija algoritma koristi različit kriterij poređenja.



- **State pattern**

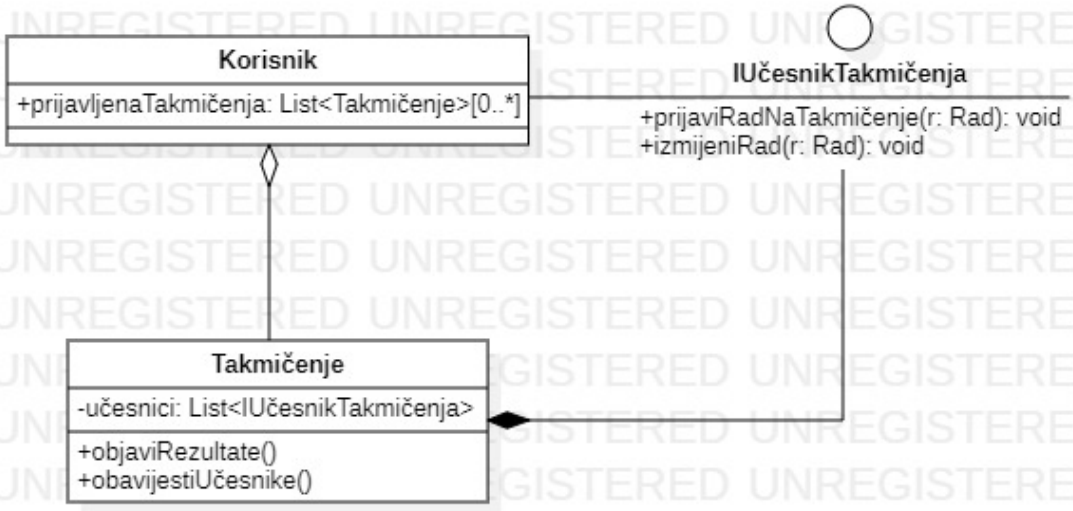
  State pattern nam dopušta da dinamički mijenjamo ponašanje neke klase, ovisno o stanju u kojem se nalazi. Moguća primjena ovog patterna u našem sistemu je zabrana prijavljivanja rada na takmičenje korisniku koji je već prijavljen na neko takmičenje. Sada klasa Korisnik može da bude u dva stanja: Prijavljen i Neprijavljen. Za slučaj kada korisnik nije prijavljen na takmičenje, metoda radi kao i obično, a ako jeste, šalje se upozorenje korisniku da ne može da se istovremeno prijavi na više takmičenja.



- **Observer pattern**

  Observer pattern omogućuje uspostavljanje relacije između objekata, tako da se pri promjeni stanja jednog objekta šalje obavijest svim objektima koji su vezani za njega. U našem sistemu ovaj pattern možemo iskoristiti da svim

korisnicima koji su se prijavili na takmičenje šaljemo notifikacije o datumu kraja takmičenja (za slučaj da žele da prave izmjene poslanog rada), te o rezultatima takmičenja.



- **Iterator pattern**

  Iterator pattern se koristi za iteriranje kroz kolekcije, pri čemu korisnik ne mora imati informacije o unutanjoj strukturi kolekcije. Jedna moguća primjena ovog patterna u našem sistemu je prolazak kroz radove koji su prijavljeni na takmičenje, pri čemu možemo odabrati da prolazimo kroz radove u opadajućem redoslijedu po njihovoj ocjeni, ili da radovi budu sortirani po žanru.

- **Memento pattern**

  Memento pattern koristimo kada trebamo da sačuvamo stanje objekta u određenom trenutku. U našem sistemu možemo iskoristiti ovaj pattern da omogućimo korisniku da pri unosu svog rada može da poništi posljednju promjenu, odnosno, da koristi ”undo” opciju. Ovo bismo mogli implementirati dodavanjem klase TextEditor našem

sistemu, koja bi čuvala prethodno stanje prije svake promjene, i trenutno stanje unosa. Kada korisnik odabere da objavi rad, tekst koji se čuva u trenutnom stanju TextEditor klase proslijedi se konstruktoru klase Rad.

- **Template method pattern**

  Template method pattern se zasniva u odvajanju pojedinih koraka neke metode ili algoritma u podklase. Time omogućujemo da taj korak ima različitu implementaciju za različite podklase. Moguća primjena ovog patterna u našem sistemu je preuređivanje sistema tako da neprijavljeni korisnici mogu da dobiju samo preview objavljenih radova, odnosno, ne bi im bilo omogućeno čitanje cijelog rada. Da bismo ovo postigli, metodu čitajRad() u klasi Korisnik implementiramo tako da prikazuje autora, naziv rada, i dio sadržaja, te da sadrži korak prikažiOstatakSadržaja(). Ovaj korak u klasi RegistriraniKorisnik implementiramo tako da prikaže ostatak sadržaja, dok u klasi GuestKorisnik obavještava korisnika da se mora prijaviti ili registrirati da pročita ostatak rada. Naravno, u našem sistemu ne postoji metoda čitajRad(), budući da se čitanje vrši prikazivanjem određenog view-a, međutim, posmatrali smo pojednostavljeni model sistema radi ilustracije primjera.

- **Mediator pattern**

  Mediator pattern enkapsulira protokol za komunikaciju među objektima, dozvoljavajući da objekti komuniciraju bez međusobnog poznavanja interne strukture objekta. Kada bi u našem sistemu dozvolili da korisnici šalju poruke jedni drugima, kao što je planirano u ranim verzijama sistema, tada bismo mogli iskoristiti ovaj pattern. Svaka

komunikacija između korisnika vršila bi se preko mediator klase, pa bismo mogli prije prikazivanja poruka primaocu provjeriti da li se u poruci nalaze neželjeni sadržaji, te ih blagovremeno ukloniti.