Patterni ponašanja

Predmet: Objektno orijentisana analiza i dizajn

Grupa: SeKEmin

Lazović Kemal

Suljević Semir

Šiljak Emin

1. Strategy pattern

Pattern koji izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Pogodan je kada postoje različiti pogodni algoritmi za rješavanje jednog problema. Naš sistem ne implementira ovaj pattern. Međutim ukoliko bismo uveli mogućnost studentu da plaća za neke usluge mogli bismo slijediti ovaj pattern. Prvo možemo uvesti novu klasu StudentFinancije izvedenu iz klase Student u kojoj ćemo imati mogućnost plaćanja za neke usluge ili za boravak u studentskom domu. Dakle naša Context klasa bi bila klasa StudentFinancije. Unutar metoda te klase će se koristiti plaćanje. Znači da potrebni interfejs IPlacanje (ovaj interfejs predstavlja IStrategy interfejs) će imati jednu metodu koja predstavlja plaćanje studenta. Dalje bismo omogućili 3 različite vrste plaćanja: keš, karticom kao i ček. Za svaku od ta tri načina plaćanja se pravi nova klasa KesPlacanje, KarticaPlacanje, CekPlacanje i sve te klase implementiraju interfejs IPlacanje te tako student može da odabere jedan od tri načina plaćanja. Bilo kojim načinom plaćanja student dobija isti rezultat (pod uslovom da ima dovoljno novca za sve načine i da su ispunjeni dodatni uslovi) te smo tako slijedili strategy pattern.

2. State pattern

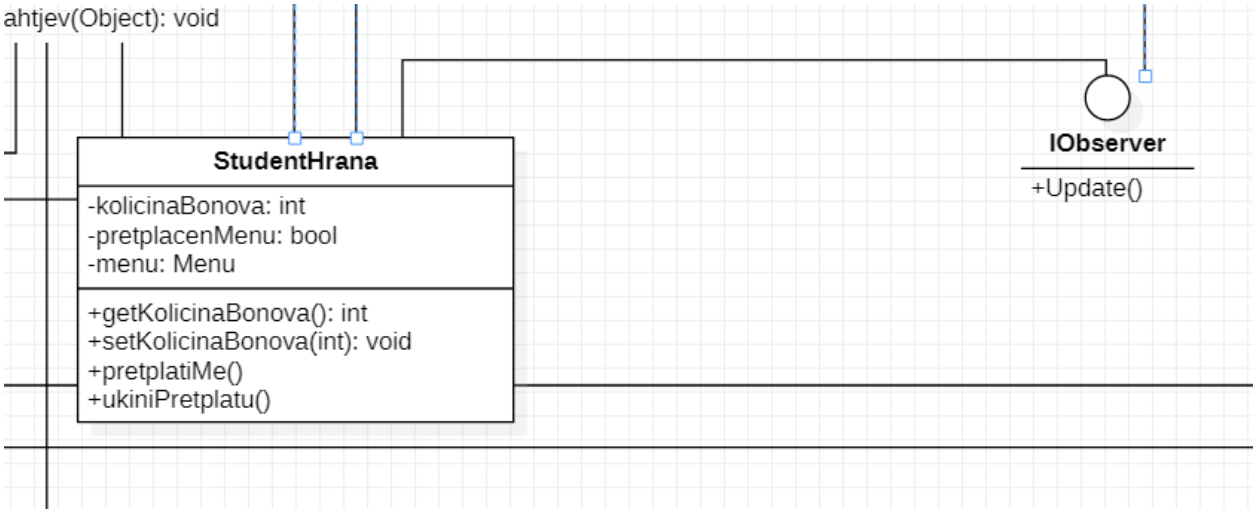
Ovaj pattern je dinamička verzija strategy patterna. Objekat mijenja način ponašanja na osnovu trenutnog stanja. Ovaj pattern nije praćen u našem projektu, ali čisto primjeri radi, mogli bismo ga implementirati na stanju knjiga. Kad se izda neka knjiga ona promijeni svoje stanje u „izdana“ (u suprotnom nije izdana). Kao client klasa koristimo klasu Student. U nju kao atribut dodajemo novu klasu Context koja definira tekući kontekst i interfejs koji je od interesa za klijenta. U interfejsu (npr. IState) imamo metode koje zasebna stanja klase Knjiga trebaju implementirat. U ovome slučju to mogu biti metode izdajKnjigu i vratiKnjigu. Također 2 klase koje implementiraju ovaj interfejs su 2 moguća stanja za knjigu. Neka se te 2 klase zovu Izdata i Neizdata. U klasi Izdata u metodi vratiKnjigu mijenjamo context.State u neizdato knjigu. Samo obrnuto radimo u klasi Neizdata za metodu izdajKnjigu.

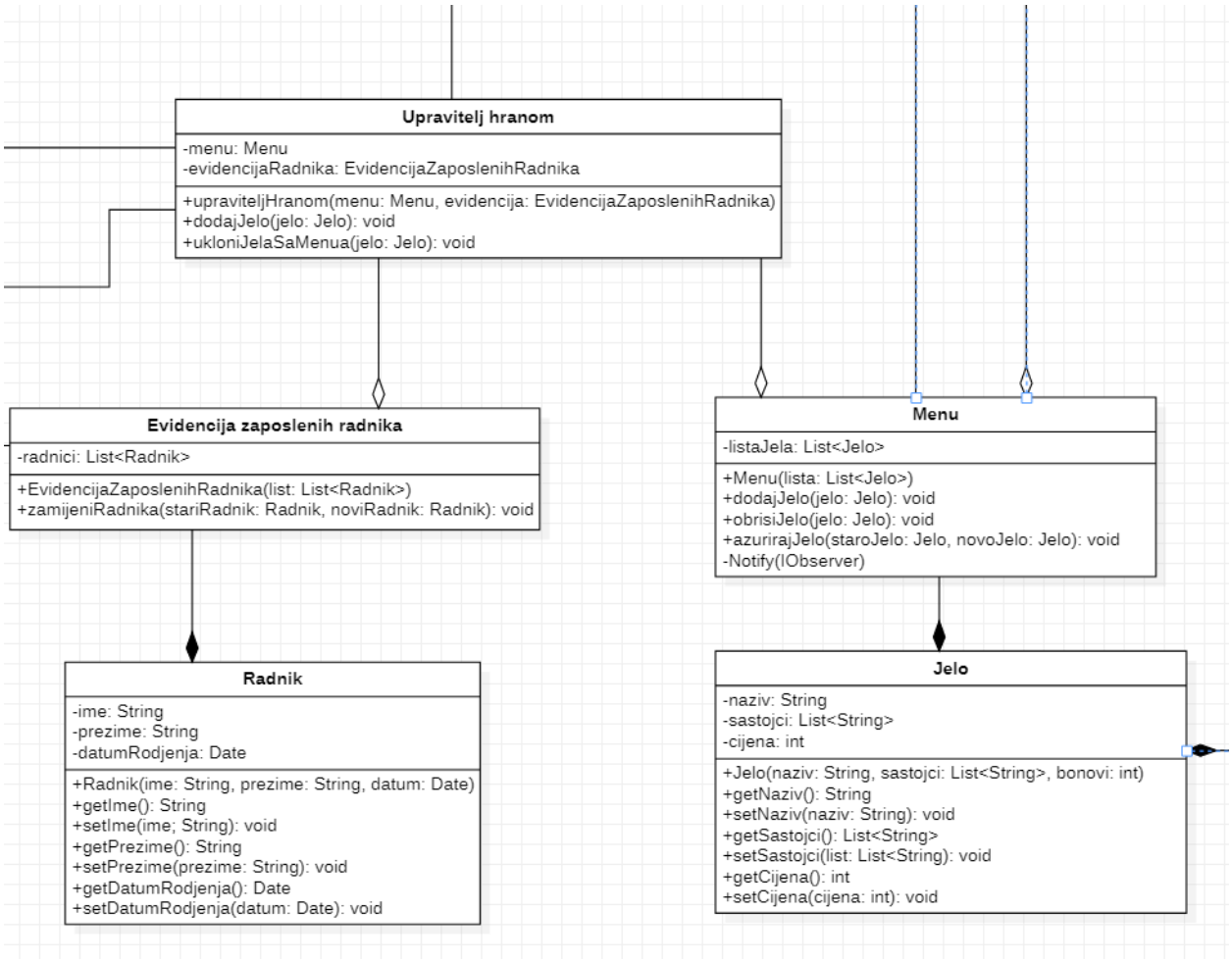
3. TemplateMethod pattern

Ovaj pattern omogućava izdvajanje određenih koraka algoritma u odvojene podklase. Naš sistem ne slijedi ovaj pattern. Međutim ukoliko bismo dodali metodu koja provjerava ispravnost sobe uz određene dodatke bismo mogli slijediti ovaj pattern. Klasa soba bi bila Algorithm klasa, metoda za provjeru ispravnosti sobe bi nam bila TemplateMethod. Dodali bismo interfejs IIspravnostSobe s metodama za provjeru čistoće sobe kao i za provjeravanje načinjene štete u sobi. Te metode bi se pozivale iz ove metode što znači da je IIspravnostSobe naš IPrimitive interfejs. Nedostaje nam još klasa/e koja/e će implementirati taj interfejs. Možemo dodati klase SobaCistoca i SobaSteta koje imaju metode za provjeru čistoće odnosno štete u sobi. Sada template metoda poziva metode iz te dvije klase da se ustanovi ispravnost sobe.

4. Observer pattern

Uloga observer patterna je da uspostavi relaciju između objekata tako kada jedan objekat promijeni stanje drugi zainteresovani objekti se obavještavaju. Ovaj pattern smo slijedili u našem sistemu. Naime, za svaku promjenu menija (dodavanje novog jela na menu ili uklanjanje postojećeg jela sa menija) studenti koji su pretplaćeni na obavještavanje ovih promjena će biti obaviješteni (obaviješteni u vidu – *jelo* je dodano na menu ili *jelo* je uklonjeno sa menija). U ovom slučaju Menu nam predstavlja Subject klasu. U Menu klasu smo dodali metodu Notify koja se poziva nakon svake izmjene na meniju. Dodali smo i interfejs IObserver sa metodom update. Taj interfejs je implementiran od strane svih klasa koje treba obavijestiti o promjenama u meniju. U ovom slučaju jedina klasa koja implementira taj interfejs je StudentHrana gdje smo dodali atribut tipa bool koji predstavlja vrijednost da li je student pretplacen te atribut tipa menu da bismo slijedili ovaj pattern. Sliku svih izmjena donosimo u nastavku

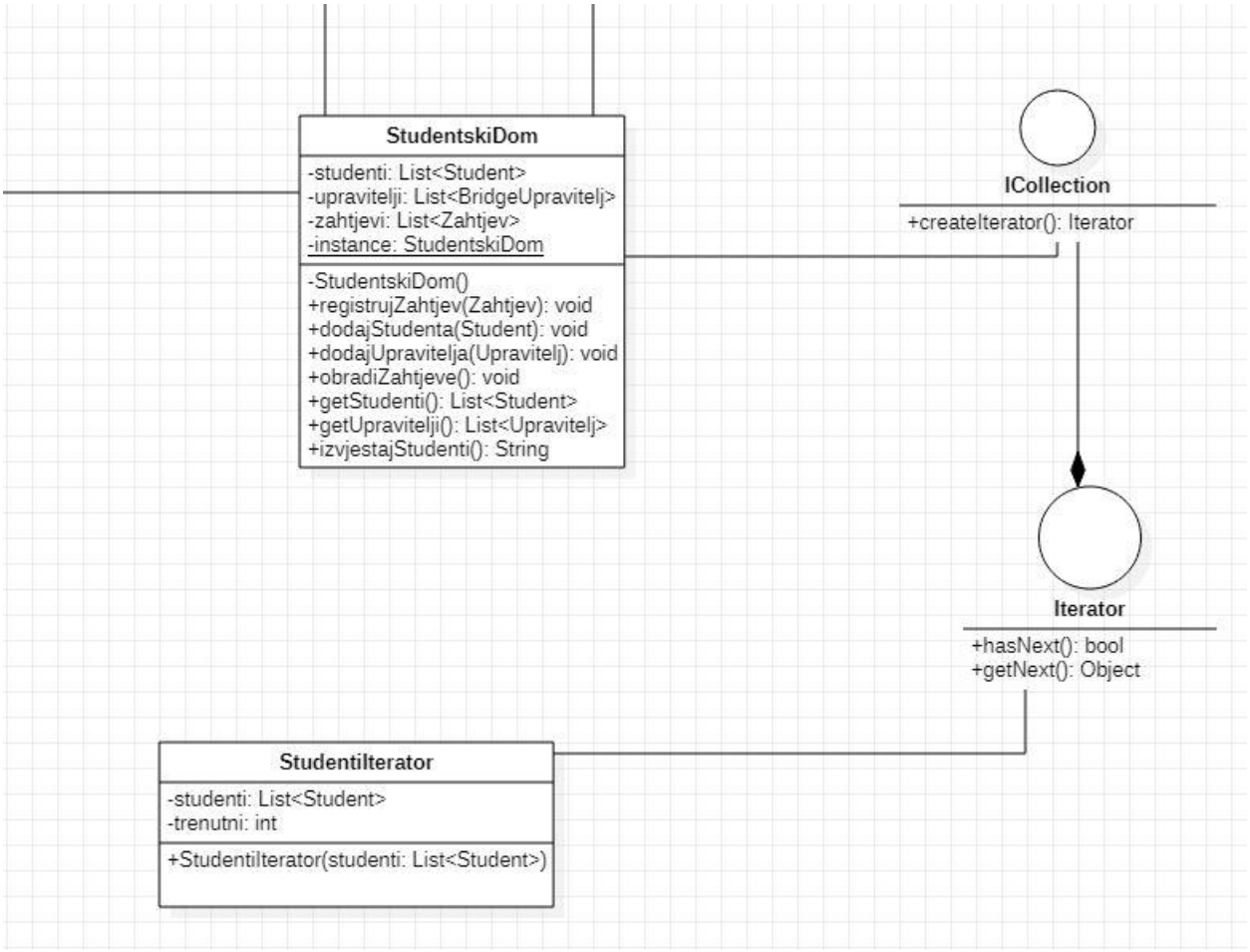




5. Iterator pattern

Pattern omogućava sekvencijalni pristup elementima kolekcije bez poznavanja kako je kolekcija struktuirana.

Iterator pattern biti će primijenjen za pristup svih studenata u kontejnerskoj klasi StudentskiDom.



Klasa StudentskiDom sadrži listu studenata koji su upisani u bazu podataka. StudentiIterator klasa kao privatni atribut ima listu studenata kroz koju će se iterirati. Ta lista je readonly.

Implementira interfejs Iterator sa metodama hasNext() i getNext() koje govore da li ima još objekata i koje daju sljedeći objekat, respektivno.

StudentskiDom kreira Iterator kroz koji se iterira u klasi StudentiIterator.