Strukturalni paterni

U osnovne strukturalne paterne ubrajamo:

Adapter, Bridge, Composite, Decorator, Facade, Proxy i Flyweight patern. U nastavku ćemo razmotriti svaki pojedinačno.

Adapter patern

Osnovna namjena ovog paterna je da omogući širu upotrebu već postojećih klasa. Ovaj patern se koristi kada ne želimo mijenjati već postojeću klasu, ali potreban nam je drugačiji interfejs. Konkretno u našem primjeru, ovaj patern nigdje ne možemo primijeniti s obzirom da nema potrebe za ikakvom konverzijom jedne klase u neku drugu. Svaka je precizno definisana i ima svoju ulogu.

Bridge patern

Bridge patern se koristi da omogući odvajanje implementacije i apstrakcije neke klase kako bi ta klasa mogla posjedovati više različitih apstrakcija i više različitih implementacija za pojedine apstrakcije.

U našoj aplikaciji Bridge patern nema potrebe da se koristi s obzirom da kod naslijeđenih klasa iz „Korisnik“, tj. kod „KorisnikSaFakultetskimMailom“, „KorisnikBezFakultetskogMaila“ i „Administrator“ nema potrebe za ovom vrstom paterna jer se svi korisnici upisuju na kurs, ostavljaju ocjenu, započinju kviz na isti način.

Composite patern

Koristi se kada svi objekti imaju različite implementacije nekih metoda, ali im je svima potrebno pristupiti na isti način, te se na taj način pojednostavljuje njihova implementacija. Composite patern služi za kreiranje hijerarhije objekata. Vrlo slično Bridge paternu, ovaj šablon nema potrebe da se koristi s obzirom da svi korisnici obavljaju metode na identičan način.

Decorator patern

Osnovna namjena ovog paterna je da omogući dinamičko dodavanje novih elemenata i ponašanja postojećim objektima. Umjesto definisanja velikog broja izvedenih klasa, dovoljno je omogućiti različito dekoriranje objekata. Kod nas, pošto korisnik nema opciju da uređuje objekte, nema potrebe za ovim paternom.

Facade patern

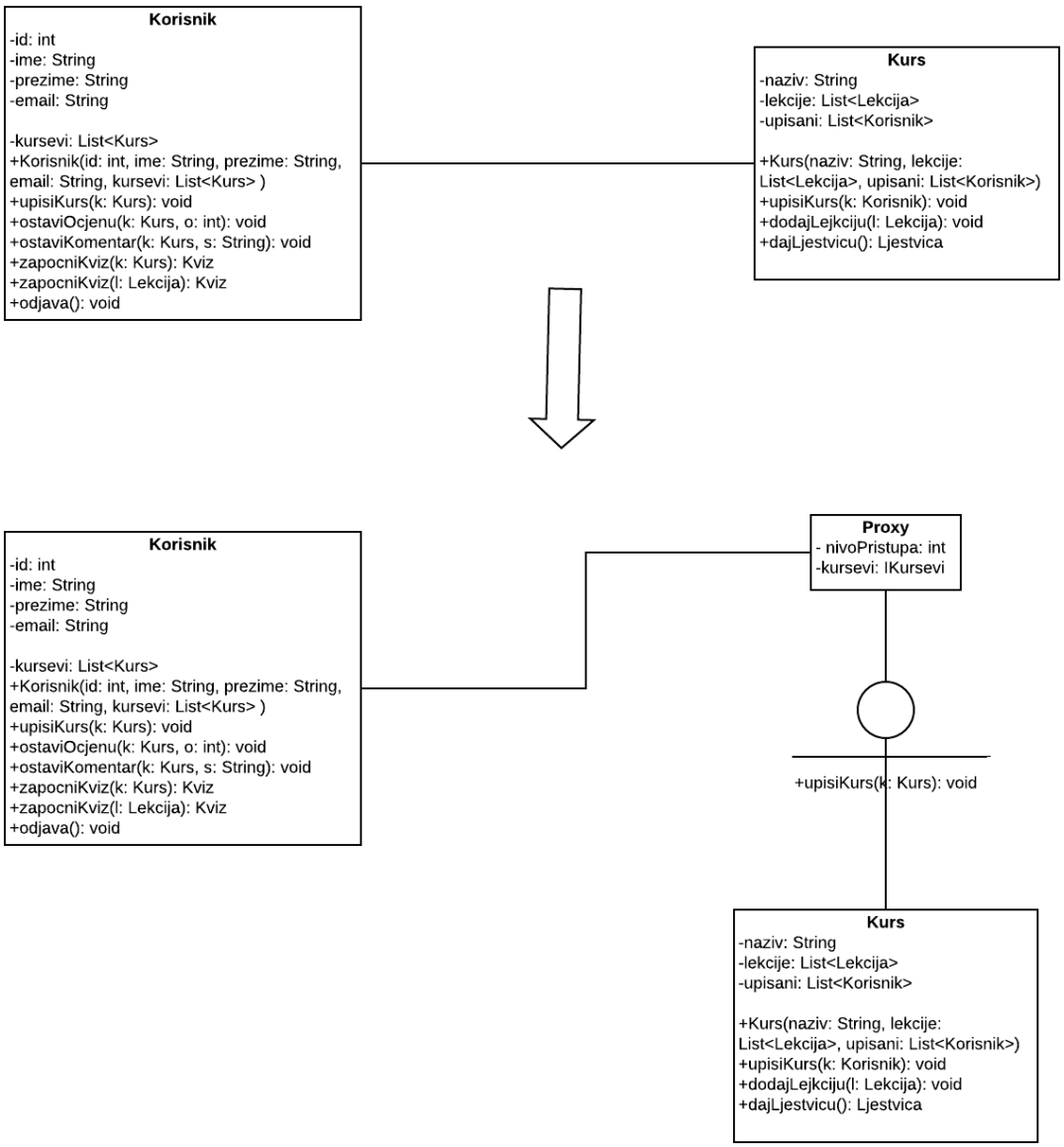
Ovaj patern se koristi kada sistem ima više podsistema pri čemu su apstrakcije i implementacije usko povezane. Facade patern služi kako bi pojednostavio klijentima korištenje kompleksnih sistema.

Klijenti vide samo krajnji izgled objekta. Pošto su sve funkcionalnosti koje klijent radi zasebne, i svaka ima svoju svrhu, fasadu ovdje nije potrebno postavljati.

Proxy patern

Proxy patern služi za dodatno osiguravanje objekata od pogrešne ili zlonamjerne upotrebe. Namjena ovog paterna je da omogući pristup i kontrolu pristupa stvarnim objektima. Obično je mali javni surogat objekat. U našoj aplikaciji, prije svega je potrebno provjeriti da li neki korisnik ima pristup kursu, i ako nema taj kurs mu se treba onemogućiti, tako da se ovaj patern može primijeniti.

Na slici smo prikazali klase „Korisnik“ i „Kurs“, kao i njihovu vezu prije i poslije upotrebe proxy paterna:

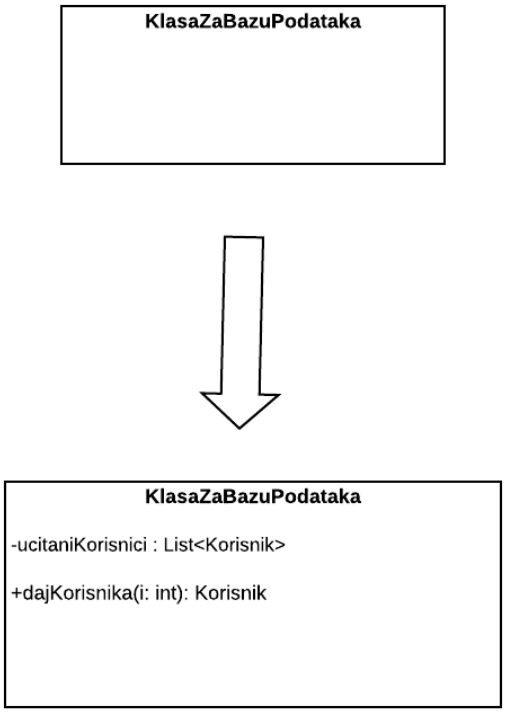


Slika 1: implementacija Proxy paterna

Flyweight patern

Flyweight patern se koristi kako bi omogućio da više različitih objekata dijele isto glavno stanje, a imaju različito sporedno stanje.

U bazi podataka smo dodali listu: **ucitaniKorisnici: List<Korisnik>** kao atribut, a **dajKorisnika(i: int): Korisnik** je dodana metoda. Ta metoda provjerava da li je korisnik ranije učitan iz baze podataka. Ako jeste učitan, tada postoji objekat tog korisnika i onda daje taj objekat, a ako nije, onda ga učitava, stavlja u listu i vraća kao rezultat.



Slika 2: primjena Flyweight paterna