STRUKTURALNI PATTERNI

1. **Fasadni Pattern**
   Uloga fasadnog patterna je pojednostavljivanje upotrebe korisnicima.
   Ovaj pattern bi mogle iskoristiti kada bi u našem sistemu imale neke podsisteme, recimo podsistem za napredno pretraživanje knjiga, sistemi za rangiranje autora i slično. U tom slučaju, bilo bi korisno da napravimo neku klasu koja predstavlja fasadu pomoću koje bi mogle objediniti sve ove podsisteme i korisniku pružiti sve njihove funkcionalnosti bez previše detalja.

2. **Adapter Pattern**
   Adapter pattern koristimo kada želimo da postojeće nekompatibilne klase rade jedna s drugom, a to postižemo uvodeći novu klasu Adapter koja služi za komunikaciju između njih. U našem sistemu nemamo primjer takvog patterna.Kada bi imale plaćanje na neki drugi način izuzev karticu, mogli bi ga tu iskoristiti.

3. **Proxy Pattern**
   Proxy patern služi za dodatno osiguranje 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.
   U našem sistemu, implementirale smo ovaj pattern na sljedeći način: dodale smo interface IUpravljanjeKorisnicima u kojem se nalaze metode izbrisiKorisnika i dodajKorisnika i klasu Proxy koja na osnovu prava pristupa odobrava ili odbija te zahtjeve.

4. **Flyweight pattern**
   Flyweight patern koristi se kako bi se onemogućilo bespotrebno stvaranje velikog broja instance objekata koji svi u suštini predstavljaju jedan objekat.
   U našem sistemu je to implementirano u klasi Korisnik. S obzirom da korisnik dobija dosta notfikacija, bilo bi korisno da jednu instancu možemo koristiti više puta uz manje izmjene. Ukoliko korisnik dobije notifikaciju koja ima istu svrhu i poruku, a datum je drugačiji, možemo iskorsititi postojeću instance uz izmjenu datuma.

5. **Composite Pattern**
   Composite pattern služi za kreiranje hijerarhije objekata. Koristi se kada svi objekti imaju različite implementacije nekih metoda, a potrebno im je svima pristupati na isti način, te se na taj način pojednostavljuje njihova implementacija. Ovaj pattern bi mogao biti u našem sistemu za neku metodu koju imaju Korisnik i SuperKorisnik, ali su različitno implementirane.

6. **Decorator pattern**

Decorator pattern služi za različite nadogradnje objektima koji svi u osnovi predstavljaju jednu vrstu objekta, imaju istu osnovu. Umjesto da se definiše veliki broj izvedenih klasa, dovoljno je omogućiti različito dekoriranje objekata. U našem sistemu nije prisutan, a mogao bi biti kada bi imali neke posebne popuste za knjige, te bi nam tu bio potreban da bi kupovina bila pravilno izvedena.

7. **Bridge pattern**

   Bridge pattern služi kako bi se apstrakcija nekog objekta odvojila od njegove implementacije. U našem sitemu nije implementiran. Uz poštivanje ovog patterna omogućava se nadogradnja modela klasa u budućnosti te osigurava da se neće morati vršiti određene promjene u postojećim klasama.

   Bridge pattern bismo mogli iskoristiti da u metodi platiKnjigu SuperKorisnik ima dodatni popust u odnosu na običnog korisnika. Dodale bismo interface IKupovina koji bi sadržavao metodu platiKnjigu i njega bi nasljeđivao Korisnik i SuperKorisnik, te bismo dodale i klasu Bridge koja bi imala atribut IKupovina i metodu platiKnjigu.