PATERNI PONAŠANJA

1. Strategy pattern(Primijenjen u sistemu)

- U našem sistemu je moguće upotrijebiti Strategy patern na sljedeći način: S obzirom na to da imamo opciju filtriranja pretrage po recenziji i cijeni, umjesto da dodajemo if-else blok pomoću kojeg ćemo znati koja opcija filtriranja pretrage je izabrana, mi možemo kreirati dvije posebne klase: filterByPrice i filterByReview, koje će implementirati interfejs IFilterOfTools. Taj interfejs ima metodu *filter (T: List) void* koja će biti implementirana u te dvije klase za filtriranje. Također, taj intefejs treba da naslijedi i neka nova klasa FilteredTool koja će u sebi imati listu alata kao i atribut tipa tog interfejsa. Od metoda koje će ona imati su changeWayOfFiltering (i:IFilterOfTools) i filter(). Metodu filter() pozivamo direktno nad objektom FilteredTool. Klasa FilteredTool ce biti povezana sa kontrolerom koji je zaduzen za filtriranje alata, I preko DBContext klase.

2. State pattern(Primijenjen u sistemu)

-U nasem sistemu cemo primijeniti ovaj patern na nacin da cemo za svaki alata razlikovati 2 stanja vec iznajmljen I dostupan.Koristicemo interfejs IState sa metodama toolDetails() I Rent(),koje ce dozvoliti pregled osobina alata I rentanje alata,respektivno.Pregled osobina ce biti omogucen za oba stanja alata,dok ce iznajmlivanje biti omoguceno samo za one alate koji nisu vec iznajmljeni(koji su u avaliable stanju).Ta dva stanja cemo razdvojiti klasama *avaliable* I *rented* sa metodama toolDetails() I Rent(),koje ce se biti razlicito implementirane,tj. Metoda toolDetails na isti nacin,a metoda Rent() sa vec opisanom funkcionalnoscu.

3. Template method

-Ukoliko bismo prosirili funkcionalnost naseg sistema na nacin da se bavi I administrativnim pitanjima kao sto su placanje kurira I administratora,onda bismo imali interfejs uposlenik,koji bi bio naslijedjen od strane klasa courrir i administrator,sa metodama kao sto su brojRadnihSati() I obracunPlate(),sa ociglednim funkcionalnostima.

4. Observer

-Ovaj patern bismo mogli primijeniti u nasem sistemu na nacin da imamo metodu u toolType klasi koja bi se automatski pokretala pri dodavanju novog tipa alata u sistem,a koja bi registrovanog korisnika obavjestavala o novom tipu alata koji do tada nije bio dostupan u radnjama.Na taj nacin bismo omogucili kompaniji da lakse promovise svoje nove proizvode za iznajmljivanje,a I korisniku bismo omogucili da lakse dodje do novih alata u ponudi.

5. Iterator

-Ovaj patern bismo mogli primjeniti u sistemu na nacin da imamo razlicite iteratore za prolazak kroz objekte klase tool.Recimo iteratorAbecedni,iteratorCjenovni,iteratorZaOcjenu,sa ociglednim funkcionalnostima,koji bi se mogli koristiti za “random” prikazivanje alata kada registrovani korsinik pristupi sistemu.U zavisnosti od toga koju stavku je korisnik ranije najvise koristio u filter pretrazi,koristio bi se odgovarajuci tip iteratora.

6. Chain of responsibility

- Što se tiče Chain of responsibility patterna, u našem sistemu mogli bismo ga iskoristiti u slučaju plaćanja iznajmljivanja alata,tj. U slucaj. To bi se uradilo na način da korisnik prvo mora odabrati alat, pa onda rentati, a nakon toga bi mu se prikazale opcije za plaćanje gdje bi mogao izabrati željenu opciju. Nakon odabira načina plaćanja, korisniku će biti ponuđena opcija za plaćanje ili za poništavanje renta.

7. Medijator

- Medijator pattern u našem sistemu bismo mogli iskoristit za komunikaciji klijenta sa support osobljem, a to je u stvari naš administrator. Ukoliko dođe do neke greške pri rentanu alata ili ukoliko je potrebno refundirati novac, a to se nije desilo automatski preko sistema plaćanja ili nešto slično, tada bismo mogli ovo iskoristiti, tj. ostvariti komunikaciju sa support osobljem. To bi se implementiralo na način da imamo interfejs IMedijator koji će biti povezan sa klijentom i administratorom i koji će implementirati metode koje će provjeravati da li se nalazi neki govor mržnje ili pravi zahtjev korisnika za pomoć.