Strukturalni paterni

1. Adapter patern

Ovaj patern nam služi da se postojeći objekat prilagodi za korištenje na neki novi način u odnosu na postojeći rad, bez mijenjanja same definicije objekta. Faktički trebamo izvršiti konverziju objekta bez da Klijent to zna. Hipotetički da u klasa Korisnik ima u sebi atribut slika klase Slika koji mora biti .jpg eksenzija, a Korisnik uploaduje sliku npr u .png ekstenziji. Taj problem mogli bismo bezbolno riješiti tako što bi napravili interfejs ISlika kojeg bi implementirala klasa AdapterSlika iz koje bi bila naslijeđena klasa Slika te bi u AdapterSlika vršili konverziju iz .png formata u .jpg format.

2. Facade patern

Fasadni patern u našem projektu primjenili smo tako što smo napravili klasu SmrtovnicaFasada koja u sebi ima metode katolickaSmrtovnica, pravoslavnaSmrtovnica, islamskaSmrtovnica, ateističkaSmrtovnica, a iz koje su naslijeđene klase Okvir koja ima atribute boja i obiljezje i standardne gettere i settere i klasa TextSlika koja u sebi ima atribute slika i tekst te standardne gettere i settere. Metode klase SmrtovnicaFasad koristit će atribute klasa Okvir i TextSlika te će biti svojevrsna fasada za klijenta kojeg neće interesovati od kojih se dijelova „pravi“ smrtovnica već će je samo kreirati pozivom nad instancom klase SmrtovnicaFasad. Pored gore navedenog ovim smo riješili i problem pojavljivaja novog zahtjeva za izgledom smrtovnice jer u takvom slučaju samo dodajemo metodu u klasu SmrtovnicaFasad koja koristi instance klasa Okvir i TextSlika.

3. Decorator patern

Decorator patern služi za omogućavanja različitih nadogradnji objektima koji svi u osnovi predstavljaju jednu vrstu objekta (odnosno, koji imaju istu osnovu). U našem sistemu nismo mogli primjeniti ovaj patern pa ćemo pokušati da zamislimo da imamo opcije koje se ne sadrže u našem sistemu da bi ga objasnili. Zamislimo da u klasi GrobnoMjesto imamo atribut izgled nadgrobnog spomenika tipa Spomenik i da Korisnik želi vršiti neke promjene nad spomenikom. Da bi iskoristili ovaj patern napravili bismo klasu Spomenik, SpomenikOsnovnePromjene, SpomenikTip i SpomenikDetalji i interfejs ISpomenik. Klase SpomenikOsnovnePromjene, SpomenikTip i SpomenikDetalji bi implementirale interfejs ISlika te bi nam bilo omogućeno da kaskadno vršimo izmjene nadgrobnog spomenika bez da u jednoj klasi pravimo glomazne metode koje će biti nerazumljive.

4. Bridge patern

Bridge patern služi kako bi se apstrakcija nekog objekta odvojila od njegove implementacije. Nismo našli način da je implementiramo u našem sistemu pa ćemo hipotetički zamisliti scenario u kojem bismo mogli iskoristiti ovaj patern. Zamislimo da iz klase Radnik imamo izvedene klase Vozač i Grobar koje u sebi imaju koeficijent plate a Vozač ima i dodatak

za prekovremene sate. Kada bi na ovaj slučaj primjenili ovaj patern uklonili bismo klasu Radnik (sve veze i metode koje je ona imala prepravili bismo da ih imaju Vozač i Grobar) i napravili novu klasu MostPlata i interfejs IPlata kojeg bi implementirali Vozač i Grobar te bi mogli kroz klasu MostPlata dobivati platu Vozača i Grobara preko jedne instance klase MostPlata.

5. Composite patern

Composite patern služi za kreiranje hijerarhije objekata. Koristi se kada svi objekti imaju različite implementacije nekih metoda, no potrebno im je svima pristupati na isti način, te se na taj način pojednostavljuje njihova implementacija. U našem projektu izvršili smo primjenu ovog paterna tako što smo napravili interfejs IZaposlenici, a u klasi Vlasnik čuvamo listu objekata tipa IZaposlenici. Interfejs IZaposlenici implementiraju tj. nasljeđuju klase Radnik, Administrator, Sef koje imaju metodu izracunajPlatu implementiranu na različit način. Ipak ono što Vlasnika interesuje jeste visina njihove plate a ne način na koji se ona izračunava te smo primjenom ovog paterna skratili dužinu koda koja će trebati da se riješi ovaj problem te olakšali eventualna nadograđivanja sistema dodavanjem nove vrste zaposlenika koji će samo trebati implementirati interfejs IZaposlenici te će vlasnik i za njega dobivati informacije o primanjima.

6. Proxy patern

Proxy patern služi za dodatno osiguravanje objekata od pogrešne ili zlonamjerne upotrebe.

Ovaj patern nismo primjenili, ali možemo simulirati slučaj da zaštitimo metode getListaUsluga i setListaUsluga tako da korisnik sistema može pregledati samo usluge koje je on koristio bez mogućnosti da sazna usluge koje je koristio neko drugi.

Napravit ćemo klasu ProxyIzvještaj koja ima atribut izvještaj tipa IUsluga koji je interfejs koji u sebi ima metodu pregledajUsluge koju implementira klasa Usluga i ProxyIzvještaj. Zaštitu ćemo izvršiti tako što ćemo u klasi ProxyIzvještaj u metodi pregledajUslugu definisati da se iz nje vraćaju samo one usluge koje koristi Korisnik koji je i tražio pregledj usluga.

7. Flyweight patern

Flyweight patern koristi se kako bi se onemogućilo bespotrebno stvaranje velikog broja instanci objekata koji svi u suštini predstavljaju jedan objekat. Ovaj patern nismo primjenili u našem projektu međutim hipotetički ćemo razmatrati kako bi se eventualno mogao koristiti. Zamislimo slučaj da na jednom grobnom mjestu može biti sahranjeno više osoba što je realan slučaj kod grobnica i želimo spriječiti da se više puta kreira instanca istog grobnog mjesta. Pomoću ovog paterna riješit ćemo problem stvaranja bespotrebnih instanci tako što ćemo napraviti interfejs IGrobnoMjesto koju će implementirati klasa GrobnoMjesto. Interfejs će imati metodu postaviGrobnoMjesto koja će provjeravati da li već postoji to grobno mjesto i da li je ono u biti grobnica koja može imati više pokojnika te će nju koristiti za daljni rad bez kreiranja nove instance.

Kreacijski paterni

1. Singleton patern

Singleton patern služi kako bi se neka klasa mogla instancirati samo jednom. Ovo objašnjenje nameće nam zaključak da ovaj patern možemo iskoristiti da u klasi Vlasnik koju ćemo preimenovati u VlasnikSingleton te u njoj napraviti instancu ove klase static VlasnikSingleton vlasnik. Setter ćemo promijeniti i onemogućiti kreiranje više od jednog objekta, a getter će vraćati instancu ove klase koju smo kreirali. Ovim smo onemogućili kreiranje više od jednog Vlasnika čime smo sačuvali eventualne zloupotrebe u našem sistemu.

2. Prototype patern

Prototype patern omogućava smanjenje kompleksnosti kreiranja novog objekta tako što se uvodi operacija kloniranja. Znamo da su u osnovi sva grobna mjesta ista prije nadogradnji koje klijent traži od pogrebnog duštva. Napravili bismo interfejs IGrobnoMjesto te bi listu ovog tipa čuvali u klasi Groblje. Interfejs bi imao jednu metodu kloniraj, njega bi implementirala klasa GrobnoMjesto. U klasi Groblje dodali bismo metodu klonirajGrobnoMjesto koja bi u slučaju da imamo grobna mjesta sa istim osobinama ili imamo grobnicu na grobnom mjestu izvršila poziv funkcije kloniraj iz interfejsa a u klasi GrobnoMjesto gdje je i realizirana funkcija kloniraj bila bi izvršena duboka kopija već postojećeg objekta uz neke promjene koji bismo eventualno tražili.

3. Factory Method patern

Factory method patern služi za omogućavanje instanciranje različitih vrsta podklasa koristeći *factory* metodu koja odlučuje koja će se podklasa instancirati i koja programska logika izvršiti. Ovaj patern nismo implementirali u naš sistem, međutim razmotrit ćemo ga hipotetički. Zamislimo da imamo neke Izvjestaj-e za koje uzimamo podataka koje ima klasa Clanarina i da imamo dva načina obračunavanja za izvještaj koji predstavljaju klase SaPDVom i BezPDVa i interfejs IIzvjestaji koji ima metodu napraviIzvjestaj kojeg implementiraju prethodne dvije spomenute klase. Sada ako budemo htjeli dodati još neku vrstu izvještaja samo trebamo kreirati novu klasu za taj izvjestaj koja implementira ili nasljeđuje interfejs.

4. Abstract Factory patern

Na osnovu apstraktne familije produkata kreiraju se konkretne fabrike produkata razlicitih tipova i razlicitih kombinacija.

Sami patern na dijagramu klasa nije iskorišten ali bi se mogao implementirati tako što bi usluge koje nudi naše pogrebno društo bile razvrstane po nekom kriteriju.

Na primjer, usluge bi se mogle razvrstati po kriteriju koji tipovi radnika su zaduženi za tu uslugu.

Tada bi sve kategorije usluga funkcionisale kao familija povezanih objekata i na osnovu apstraktne familije usluga kreirale bi

se konkretne fabrike produkata razlicitih tipova i kombinacija.

Koraci :

- IFactory interfejs - apstraktni interfejs sa Create operacijama za svaku fabriku proizvoda

- Factory1, Factory2, ... klase - implementiraju operacije kreiranja za pojedinacne fabrike

- IUslugaA, IUslugaB, ...apstraktni intefejsi za pojedinacne grupe usluga (A,B,...)

- UslugaA1, UslugaB2, ... klase koje implementiraju prethodno navedene interfejse i definiraju objekte produkata koji se kreiraju za odgovarajucu fabriku

U sastavu paterna je i Client klasa - preko interfejsa IFactory i IUslugaA, IUslugaB koristi objekte fabrike.

5. Builder patern

Builder patern služi za apstrakciju procesa konstrukcije objekta, kako bi se kao rezultat mogle dobiti različite specifikacije objekta koristeći isti proces konstrukcije. Nismo ga primjenili u našem projektu pa ćemo simulirati postojanje nekih klasa i mogućnosti da bi mogli objasniti njegovu primjenu. Ako bismo imali mogućnost da kreiramo GrobnoMjesto koje bi se sastojalo od nadgrobnog spomenika, teksta na spomeniku, vrste spomenika, cvijeća, ukrasa i još nekih detalja vidjeli bismo da bi nam trebao ogroman konstuktor. Pomoću Build paterna ovaj problem možemo riješiti tako da dodamo klase ManuelnoUredi i AutomatskiUredi koje će u sebi imati atribute tipa GrobnoMjesto i interfejs IMjesto. Interfejs će omogućiti implementaciju različitih metoda za različite načine uređivanja grobnom jesta, klase Manuelnouredi i AutomatkiUredi će implementirati metode za uređivanje grobnom mjesta. Klasu koja bi koristila ove opcije tj. klasu Klijent povezat ćemo sa interfejsom jer će na indirektan način vršiti izradu grobnog mjesta.