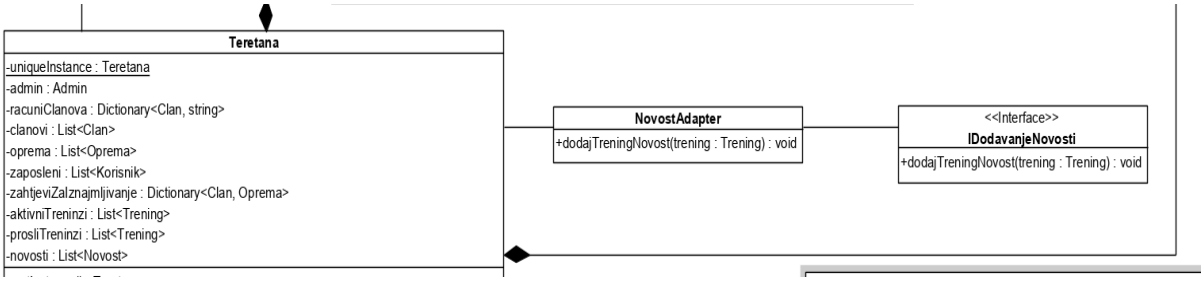
Strukturalni patterni

1. ***Adapter pattern*** – Kada bi naš sistem podržavao funkcionalnost da se stvaranjem novog termina za trening omogući dodavanje obavijesti o tom treningu u listu novosti, bilo bi potrebno da metoda dodajNovost klase Teretana, koja prima parametar tipa Novost, može primiti parametar tipa Trening, što nije moguće budući da klase Novost i Trening nisu kompatibilne. Da bi ovo omogućili, potrebno je kreirati interface IDodavanjeNovosti koji ima metodu dodajTreningNovost koja prima parametar tipa Trening. Zatim je potrebno kreirati adapter klasu NovostAdapter koja će implementirati interface IDodavanjeNovosti. Unutar adaptera je potrebno implementirati metodu dodajTreningNovost, na način da ona kreira objekat tipa Novost koristeći odgovarajuće atribute objekta tipa Trening kojeg je dobila kao parametar (npr. Naziv treninga se koristi kao naziv Novosti, slika trenera koji drži trening se koristi kao slika Novosti itd.). Ovako kreiranu novost ona zatim prosljeđuje kao parametar metodi dodajNovost klase Teretana, koja zatim vrši dodavanje ove novosti u listu novosti. Da bi ovo bilo moguće, potrebno je da klasa Teretana bude singleton klasa (što ćemo posebno obraditi u kreacijskim paternima) odnosno da je moguće kreirati samo jednu instancu te klase. Na taj način će se omogućiti da se ovo dodavanje novosti preko adaptera odvija nad odgovarajućom listom.

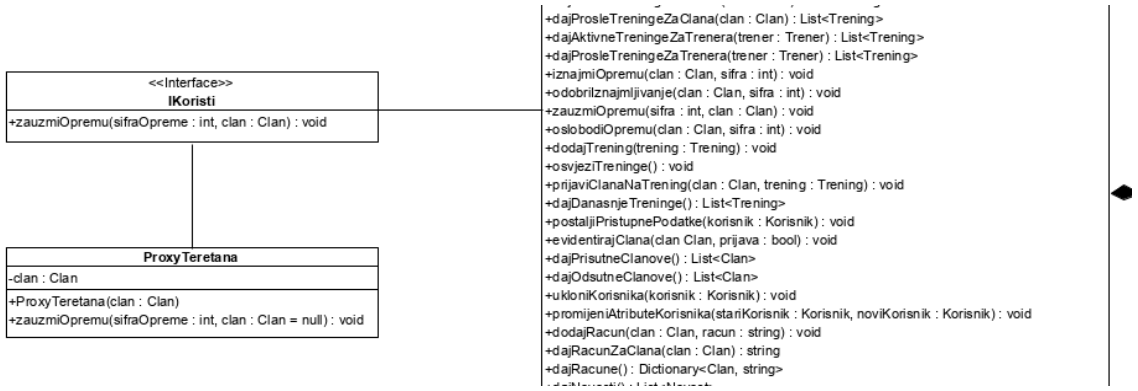


2. ***Facade pattern*** – Unutar našeg sistema omogućen je pregled 3 vrste izvještaja koji se tiču opreme u teretani, jedan izvještaj vezan za slobodnu opremu, jedan vezan za opremu koja se trenutno koristi i jedan za iznajmljenu opremu. Pri tome je za kreiranje jednog izvještaja potrebno najprije pozvati odgovarajuću metodu klase teretana (dajSlobodnuOpremu(), dajZauzetuOpremu() ili dajOpremuKojaSeKoristi()) u zavisnosti od toga o kojem tipu izvještaja se radi, te za svaki konkretni objekat izdvojiti odgovarajuće informacije neophodne za kreiranje izvještaja. Za izvještaj o slobodnoj opremi to je samo šifra opreme, za izvještaj o zauzetoj opremi šifra opreme i naziv korisnika koji je trenutno koristi, a za iznajmljenu opremu šifra opreme, naziv korisnika koji je koristi te krajnji rok za vraćanje opreme. Umjesto da se za svaki izvještaj najprije vrši pozivanje odgovarajuće metode klase Teretana a zatim vrši izdvajanje odgovarajućih atributa, možemo kreirati klasu IzvjestajFasada, koja će kao takva imati jednu metodu napraviIzvjestaj, koja kao paramter prima samo tip opreme za koju se pravi izvještaj (SLOBONO, ZAUZETO, IZNAJMLJENO). Metoda vrši dobavljane odgovarauće liste opreme iz klase teretana, izdvajanje odgovarajućih atributa za svaki element liste te za svaki element liste kreira zaseban string odgovarajućeg formata, a kao rezultat vraća listu tih stringova koja

predstavlja krajnji oblik izvještaja. Time je unutrašnja struktura izvještaja sakrivena od korisnika koji pravi izvještaj.

3. ***Decorator pattern*** – Ukoliko bi našem sistemu dodali funkcionalnost realizovanja neke specijalne vrste novosti, npr. praznične novosti kod kojih se na samom kraju novosti nalazi čestitka "Sretan praznik", to možemo uraditi koristeći ovaj patern. Potreban bi nam bio interfejs INovost koji sadrži metodu getText koja nam služi za dohvatanje teksta za odgovarajuću novost. Naša trenutna klasa Novost bi nasljeđivala ovaj interface te bi njena implementacija metode getText() podrazumijevala samo vraćanje postojećeg teksta. Svaka druga specijalna vrsta novosti koju poželimo dodati bi onda predstavljala dekorator, pa bi tako za naš primjer imali klasu NovostZaPraznik koja također implementira interface INovost. Ova klasa sadrži jedan atribut tipa INovost i sadrži konstruktor koji služi za incijalizaciju tog atributa. Njena metoda getText() bi na kraj osnovnog teksta dobavljenog iz svog atributa dodala na kraj string "Sretan Praznik!". Na isti način bi se uradilo i za dodavanje još nekih specijalnih vrsta novosti, kao što su npr. novosti sa motivacijskom porukom na kraju i slično.
4. ***Bridge pattern*** – Kada bi naš sistem vršio obračunavanje plate uposlenicima, najprije bi kreirali interface IPlata sa jednom metodom izracunajPlatu() koja vraća rezultat tipa double koji predstavlja vrijednost dodatka na platu odgovarajućeg uposlenika. Kako postoje 3 tipa uposlenika : Recepcioner, Trener i Admin te kako se dodatak na platu za svakog od njih računa na različit način, potrebno je da svaka od ovih klasa naslijedi interface IPlata i implementira vlastitu metodu izracunajPlatu(). Metoda izracunajPlatu() koju implementira klasa Recepcioner će dodatak izračunati kao broj radnih sati (Što se može izračunati na osnovu atributa početakRadnogVremena i krajRadnogVremena) pomnožen sa odgovarajućom satnicom i brojem radnih dana u mjesecu. Rezultat metode izracunajPlatu() implementirane unutar klase Trener bit će jednak broju treninga koje je taj trener održao u tekućem mjesecu pomnožen ukupnim brojem članova teretane koji su prisustvovali treninzima u tom mjesecu i brojem 10. Metoda dajPlatu() implementirana unutar klase Admin će vraćati konstantu vrijednost. Nakon toga je potrebno kreirati klasu Bridge koja ima jedan atribut tipa IPlata i jednu metodu dajPlatu(), koja kao rezultat vraća zbir osnovice i dodatka dobijenog pozivom metode izracunajPlatu() interface-a IPlata, što predstavlja vrijednost plate za datog korisnika.
5. ***Proxy pattern*** – Kako član teretane ima mogućnost prijavljivanja na svoj profil bilo kada, potrebno je onemogućiti mu zauzimanje, odnosno korištenje opreme kada se ne nalazi u teretani. Potrebno je kreirati interface IKoristi koji sadrži metodu ZauzmiOpremu koja ima dva parametra od kojih je jedan tipa Oprema, a drugi Clan. Parametar koji je tipa Clan ima podrazumijevanu vrijednost koja je null. Ovaj interface će biti implementiran od strane Teretane te će se unutar metode vršiti odgovarajuće promjene atributa objekta primljenog kao parametar, kao što su postavljanje polja pocetniDatum, krajnjiDatum, korisnikOpreme i tipOpreme. Također, potrebno je kreirati klasu ProxyTeretana koja također implementira ovaj interface, a u konstruktoru prima Člana koji pokušava zauzeti opremu. Unutar metode ZauzmiOpremu (koja u ovom slučaju ne zahtijeva parametar tipa Clan jer je on već primljen putem konstruktora i smješten u atribut) se vrši provjera atributa trenutnoPrisutan

tog člana, te se poziva metoda Teretane ZauzmiOpremu ukoliko je taj atribut true, a ukoliko je false ispisuje se upozorenje.

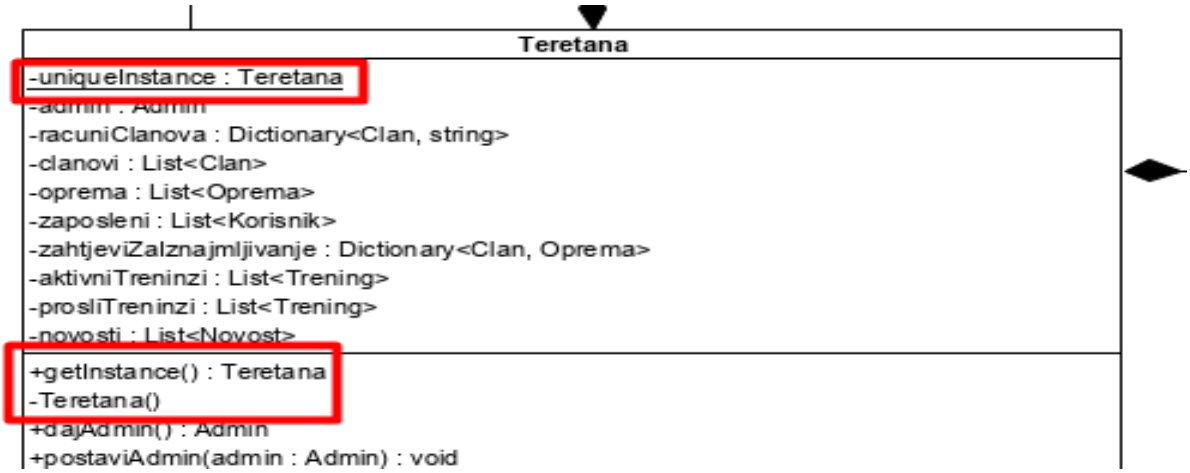


6. ***Composite pattern*** – Kada bi naš sistem vršio izračunavanje vremena dnevno provedenog u teretani za svakog korisnika iste, najprije bi kreirali interface IKorisnik sa jednom metodom dajVrijeme() koja vraća rezultat tipa double koji predstavlja broj sati koje je korisnik proveo u teretani taj dan. Kako postoje 4 tipa korisnika: Recepcioner, Trener, Admin i Član, kako se vrijeme provedeno u teretani za svakog od njih računa na različit način, potrebno je da svaka od ovih klasa naslijedi interface IKorisnik i implementira vlastitu metodu dajVrijeme(). Metoda dajVrijeme() koju implementira klasa Recepcioner će vrijeme izračunati kao broj radnih sati (što se može izračunati na osnovu atributa pocetakRadnogVremena i krajRadnogVremena). Rezultat metode dajVrijeme() implementirane unutar klase Trener bit će jednak zbiru trajanja svih treninga koje je on održao tog dana. Metoda dajVrijeme() implementirana unutar klase Admin će vraćati konstantu vrijednost. Ukoliko želimo da pozivom metode dajVrijeme() unutar klase Teretana dobijemo prosječno vrijeme koje su u teretani proveli svi njeni korisnici tog dana, potrebno je da klasa Teretana implementira interface IKorisnik, a njena implementacija metode dajVrijeme() podrazumijeva prolazak kroz listu svih korisnika (koji imaju vlastite implementacije ove metode) i računanje sume vremena koje su oni taj dan proveli u teretani I dijeljenje dobijenog rezultata sa brojem korisnika teretane. Pri tome je potrebno da liste članova I uposlenika sadrže elemente tipa IKorisnik, a ne Korisnik kako je to prethodno bilo.
7. ***Flyweight pattern*** – Ukoliko opremu teretane ne bi razlikovali po id-u, već samo po tipu (npr. Tegovi, traka I slično), pri čemu je svaki tip opreme predstavljen jednom klasom, tada bi se upotrebom flyweight patterna moglo izbjeći kreiranje prevelikog broja instanci opreme odnosno posebne instance za svaku pojedinačnu opremu. Da bi to realizirali, najprije je potrebno napraviti interface IOprema, sa jednom metodom dajTip() koja kao rezultat vraća string koji govori o kojem tipu opreme se radi. Zatim je potrebno da svaka klasa koja predstavlja jedan tip opreme implementira interface IOprema odnosno njegovu metodu dajTip(), pri čemu će ta metoda za klasu Tegovi vraćati string “tegovi”, za klasu Traka “traka” Itd. Pri tome, svaka od ovih klasa pored standardnih atributa kao što su naziv I opis, ima I 3 atributa tipa int koji predstavljaju ukupan broj raspoložive opreme, broj iznajmljene opreme te broj zauzete opreme tog tipa, pomoću kojih će se voditi evidencija

o iznajmljenoj, zauzetoj I slobodnoj opremi. Inicijalne vrijednosti ovih atributa predstavljaju “bezlično stanje” ove opreme. Ukoliko sada korisnik zatraži zauzimanje, iznajmljivanje ili oslobađanje opreme određenog tipa, pronašla bi se odgovarajuća instanca opreme na osnovu metode dajTip() a zatim vratila korisniku bez potrebe da se vrši novo instanciranje. Samo bi se izvršila promjena odgovarajućih atributa.

Kreacijski patterni

1. ***Singleton pattern*** – Klasa Teretana zahtijeva globalni pristup kreiranog instanci te da se ona može samo jednom instancirati. Stoga je potrebno klasu Teretana realizovati kao Singleton klasu. Da bi klasa Teretana bila Singleton klasa ona mora imati jedan statički atribut tipa Teretana uniqueInstance koja čuva jedinstvenu instancu klase, static metodu getInstance koja vraća tu instancu I privatni konstruktor kako bi se onemogućilo da se bilo gdje drugo kreira objekat tipa Teretana.



2. ***Prototype pattern*** – Jedina klasa našeg sistema na koju je moguće primjeniti prototype pattern je klasa Novost (sve ostale klase imaju atribute koji moraju biti specifični za svaku pojedinačnu instancu klase, kao što je to npr atribut korisnickoIme za sve klase izvedene iz klase korisnik, ili sifraOpreme za klasu Oprema itd.). U tu svrhu potrebno je napraviti interface INovost koji ima jednu metodu clone(), koja kao rezultat vraća objekat tipa INovost. Klasa Novost će implementirati interface INovost. Za realizaciju ovog patterna zamislimo da klasa Novost ima i atribut trenutnoVrijeme tipa DateTime koji predstavlja datum i vrijeme kada je ta Novost dodana. Također, recimo da je atribut slika koja sadrži putanju do slike za novost ima neku defaultnu vrijednost koja se po želji i potrebi može mijenjati. Metoda clone() klase Novost će vraćati duboku kopiju objekta nad kojim je pozvana sa svim atributima istim osim trenutnoVrijeme koji se određuje za svaku instancu posebno. Nakon dobavljanja kopije ovog objekta, može se izvršiti potrebna izmjena atributa koji predstavljaju sam naziv i tekst novosti.

3. ***Factory Method pattern*** – Kako naš sistem omogućava pregled 7 vrsta izvještaja, njih možemo realizirati kao 7 podklasa apstraktne klase Izvjestaj, koja bi kao takva imala jedan atribut tipa lista stringova, pri čemu svaki string odgovara jednom redu izvještaja. Za instanciranje ovih podklasa moguće je iskoristiti factory method pattern. U tu svrhu, potrebno je kreirati interface IIzvjestaj koji će imati jednu metodu tipa void, napraviIzvjestaj(). Sada će svaka od 7 podklasa klase Izvjestaj naslijediti interface IIzvjestaj, I implementirati svoju verziju metode napraviIzvjestaj(), unutar koje će se na osnovu odgovarajućih listi dobavljenih iz klase teretana formirati odgovarajuće liste stringova koje predstavljaju izvještaj za tu podklasu, te će se kao takve dodijeliti atributu klase Izvjestaj.
4. ***Abstract Factory pattern*** – Ukoliko bi naš sistem imao 2 vrste članova koji su predstavljeni klasama obicniClan I vipClan izvedenim iz postojeće klase Clan, pri čemu VIP član pored nekih dodatnih privilegija ima I privilegiju prijavljivanja na individualne treninge, dok se obični član može prijaviti samo na grupne treninge (pri tome su grupni I individualni treninzi odvojeni u dvije podklase grupniTrening I individualniTrening izvedene iz postojeće klase Trening), tada se za prikaz aktivnih treninga za svakog člana može iskoristiti abstract factory pattern. Najprije je potrebno kreirati interface IFactory koji ima jednu metodu dajTreninge() koja kao rezultat vraća listu treninga. Potrebno je kreirati I dvije klase obicniClanFactory I vipClanFactory pri čemu svaka ima jedan atribut tipa lista treninga, gdje se za običnog člana u toj listi nalaze samo individualni, a za VIP člana I individualni I grupni treninzi, te svaka implementira interface IFactory, koja kao rezultat vraća ovu listu treninga.
5. ***Builder pattern*** – Builder pattern se koristi kada se više objekata na različit način sastavlja od istih dijelova. Trenutno naša klasa Oprema modelira pojedinačne sprave, međutim mogli smo ju zamisliti i kao klasu koja modelira skupinu nekih sprava na način da ima sljedeće atribute: brojTegova i maksimalnaTezinaTegova koji predstavljaju respektivno broj tegova i maksimalnu težinu tih tegova koji se nalaze u sklopu te opreme, zatim atributi tipa bool spravaZaTrcanje, spravaZaZgibove, razboj i sl. Zatim, potrebno bi bilo imati interface IBuilder sa metodama kao sto su dodajBrojITezinuTegova, dodajSpravuZaTrcanje, dodajSpravuZaZgibove, dodajRazboj... Ovaj interface bi bio implementiran od strane klasa AutomatskiBuilder i SpecijalniBuilder. Metode klase Automatski (kod koje bi parametri metoda imali podrazumijevane vrijednosti) bi atributima dodjeljivale neke unaprijed određene vrijednosti, npr. 10 tegova kojima je maksimalna težina 50g, sprava za trčanje ali ne i sprava za zgibove niti razboj, dok bi se unutar metoda klase SpecijalniBuilder atributima dodjeljivale vrijednosti primljene putem parametara, te bi ovaj builder korsitili oni članovi koji žele sami sebi da odaberu koliko i koje opreme će koristiti.