Univerzitet u Sarajevu
Elektrotehnički fakultet u Sarajevu
Odsjek za računarstvo i informatiku

Kreacijski paterni
Imunizacija '21

Objektno orijentisana analiza i dizajn

Naziv grupe: Schpritzer
Članovi: Muhamed Borovac
Eldar Čivgin
Dženan Nuhić
Benjamin Pašić

1. Singleton patern (primijenjen)

Uloga **Singleton patern**-a je da osigura da se klasa može instancirati samo jednom i da osigura globalni pristup kreiranoj instanci klase. Postoji više objekata koje je potrebno samo jednom instancirati i nad kojim je potrebna jedinstvena kontrola pristupa.
U našem sistemu, mi smo ovaj patern već iskoristili za jednu klasu, koja nam služi da predstavimo statistiku o pandemiji koja će nam biti potrebna u aplikaciji. U pitanju je klasa *StatistikaKSSingleton*. Ovu klasu koristimo kao "globalni resurs" aplikacije, te ostale klase mogu koristiti njene atribute i ažurirati ih.

2. Prototype patern

Prototip patern nam omogućava da kopiramo postojeće objekte bez da ovisimo o njihovim klasama.
Na primjer, želimo kreirati identičnu kopiju nekog objekta. Prvo bismo morali kreirati novi objekat iste klase. Zatim moramo preći kroz sve njegove atribute i kopirati njihove vrijednosti u novi objekat. Međutim, dolazimo na problem - ne mogu se svi objekti kopirati na ovaj način, jer neki atributi su privatni te nisu vidljivi izvan tog samog objekta. Još jedan problem je što kada trebamo kreirati duplikat nekog objekta, moramo znati klasu tog objekta, te naš kod postaje ovisan o toj klasi.
Iz ova dva razloga koristimo Prototip patern.
Koristeći ovaj patern, pravimo jedan interfejs za sve objekte koji podržavaju kloniranje. Ovaj interfejs se obično sadrži od metode *clone()*. Implementacija ove metode je slična u svim klasama - kreira se objekat trenutne klase i prenesu se vrijednosti svih atributa u novi objekat. Na ovaj način, kada nam treba isti objekat kao objekat koji smo konfigurisali, pozovemo metodu *clone()* umjesto da iz početka stvaramo novi objekat.
U našem sistemu, primjena ovog patern-a bi se mogla primijeniti nad klasom *Osoba*. Ako se odlučimo na vakcinaciju cijele porodice, onda nam je lakše klonirati jednu osobu (pošto porodice imaju isto prezime, lokaciju, itd.), a zatim mijenjati neke podatke (poput imena i sl.).

3. Factory Method patern

Uloga **Factory Method patern**-a je da omogući kreiranje objekata na način da podklase odluče koju klasu instancirati.
Factory Method instancira odgovarajuću podklasu(izvedenu klasu) preko posebne metode na osnovu informacije od strane klijenta ili na osnovu tekućeg stanja.
Naš sistem je moguće proširiti koristeći ovaj patern na način tako da na osnovu starosti osoba primi određenu vakcinu.
Za primjer ću uzeti vakcine *Sinovac*(preferebilne za mlado stanovništvo) i *AstraZeneca*(preferabilne za starije stanovništvo) koje bi kreirali kao klase koje bi implementirale neki naš interfejs *IVakcina*. Potrebna nam je klasa *Creator* koja sadrži metodu *FactoryMethod()* koja odlučuje na osnovu zadate starosne grupe koju će vakcinu odabrati, odnosno koju će klasu instancirati.

4. Abstract Factory patern

Abstract Factory patern nam omogućava da se kreiraju familije povezanih objekata. Na osnovu apstraktne familije produkata kreiraju se konkretne fabrike produkata različitih tipova i različitih kombinacija.
U naš sistem moguće je implementirati ovaj patern tako što u klasi *Vakcinacija* zamijenimo tip prve i druge doze, koji je trenutno *Tuple*, sa vlastitim tipovima, odnosno klasama. Ove klase će biti apstraktne i zvat će se *PrvaDoza* i *DrugaDoza*. Nakon toga, iz apstraktne klase *PrvaDoza* naslijedimo klase *PrvaDozaPfizer*, *PrvaDozaModerna*, itd; na isti način naslijedimo klase *DrugaDozaPfizer*, *DrugaDozaModerna*, itd. iz klase *DrugaDoza*. Sada ćemo napraviti interfejs *IDozaFactory* koji će sadržavati metode *getPrvaDoza* i *getDrugaDoza* koje vraćaju instance klasa *PrvaDoza* i *DrugaDoza* respektivno. Sljedeći korak je da kreiramo klase *PfizerDozeFactory*, *ModernaDozeFactory*, itd. koje će implementirati interfejs *IDozaFactory* i njegove metode. Na kraju klasa *Vakcinacija*, koja sada sadrži dva atributa (jedan tipa *PrvaDoza* i jedan tipa *DrugaDoza*), će u konstruktoru primati instancu odgovarajuće factory klase, te na taj način može da instancira svoje privatne atribute prve i druge doze. Ovim ćemo osigurati da neće doći do miješanja nekompatibilnih tipova prvih i drugih doza, npr. *PrvaDozaPfizer* i *PrvaDozaModerna* nisu kompatibilne. Također, pored ovoga, možemo naravno i imati neke metode koje postavljaju ili izmjenjuju doze, a pri tome koriste factory klasu.

5. Builder patern

Uloga **Builder patern**-a je odvajanje specifikacije kompleksnih objekata od njihove stvarne konstrukcije. Isti konstrukcijski proces može kreirati različite reprezentacije.
U našem sistemu ovaj patern bismo mogli realizirati tako što bismo dodali interfejs *IVakcinaBuilder* sa raznim metodama kao npr. *dajDjecijeVakcine*, *dajVakcineZaStarijeOsobe*, *dajPovoljneVakcine*, *dajSkupeVakcine*, itd. (ove metode bi mogle kao parametar primati *brojVakcina* tipa int). Ove metode bi vraćale liste vakcina, te bi uzimale u obzir razne faktore (starosnu dob, cijenu vakcine, efikasnost, ...). Zatim bismo dodali klase *DjecijeVakcineBuilder*, *VakcineZaStarijeBuilder* (i slične po potrebi), koje bi u sebi imale kao atribut listu svih raspoloživih vakcina u sistemu. Ove metode bi mogle u ovisnosti od starosti korisnika (ili drugih faktora) preporučivati mu one vakcine koje su najbolje za njega.