KREACIJSKI PATERNI

1. Singleton Pattern

Singleton pattern osigurava da se neka klasa može instancirati samo jednom, te da se pritom omogući globalni pristup toj instanci iz bilo kojeg dijela aplikacije.

Ovaj pattern nam je potreban kada:

a) Postoji više servera koji mogu uslužiti zahtjev korisnika,

b) Serveri se mogu uključivati/isključivati,

c) Potreban je jedan objekat koji zna o stanju cjelokupne mreže i koji će vršiti usmjeravanje saobraćaja.

U našem sistemu, najpogodniji kandidat na Singleton je DataContext klasa. Ona predstavlja predstavlja centralno skladište podataka, i gotovo svi kontorleri u sistemu imaju atribut tipa DataContext. Kada bi svaki kontroler imao vlastitu instancu, to bi dovelo do nedosljednosti — promjene u jednoj instanci ne bi bile vidljive u drugima. Korištenjem Singleton paterna ovdje, osigurali bismo da svi dijelovi aplikacije pristupaju istoj instanci DataContext klase, čime se izbjegava duplikacija i osigurava konzistentnost podataka.

2. Prototype Pattern

Prototype dizajn patern dozvoljava da se kreiraju prilagođeni objekti bez poznavanja njihove klase ili detalja kako je objekat kreiran. Ovaj pattern obezbjeđuje interfejs za konstrukciju objekata kloniranjem jedne od postojećih prototip instanci i modifikacijom te kopije.

Tipično se koristi kada:

a) novi objekat treba imati **isti početni sadržaj** kao postojeći,

b) inicijalizacija objekta je **kompleksna ili skupa**

Klasa vožnja može biti dobar kandidat za primjenu ovog paterna jer je moguće imati različite varijante ili kopije koje se mogu brzo kreirati s minimalnim naporom. Na primjer, mogli bismo imati osnovnu vožnju s početnom i odredišnom lokacijom te rutom, a zatim klonirati ovaj objekat i prilagoditi vozača, putnika i kasnije ocjenu. Ovaj pristup smanjuje potrebu za ponovnim kreiranjem objekta od nule i omogućava efikasno generisanje varijanti vožnji.

3. Factory method pattern

Ovaj patern omogućava kreiranje objekata na način da podklase odlučuju koju klasu instancirati. Ovaj patern odvaja zahtijevanje objekata od njihovog kreiranja, omogućavajući klijentima da ne moraju znati koji tip objekta je stvoren. Kreiranje objekta se vrši putem posebne metode, koja može zavisiti od informacija od klijenta ili trenutnog stanja.

Kada bismo napravili klasu Notifikacija u našem sistemu apstraktnom baznom, i iz nje naslijedili različite vrste notifikacija (npr email, sms), tada bi imalo smisla uvođenje ovog patterna, iz sljedećih razloga:

a) Različiti tipovi notifikacija imaju različitu logiku, ali svi implementiraju istu metodu posalji().

b) klasa Korisnik ne zna detalje implementacija – samo poziva posalji().

c) Dodavanje novog tipa notifikacije ne bi zahtjevalo promjenu Korisnik klase.

4. Abstract factory pattern

Abstract Factory patern omogućava da se kreiraju familije povezanih objekata/produkata bez specificiranja konkretnih klasa. Na osnovu apstraktne familije produkata kreiraju se konkretne fabrike (factories) produkata različitih tipova i različitih kombinacija.

Uvođenje ovog paterna za klasu Korsinik sa ulogama *Vozac*, *Putnik* i *Administrator*, *TehnickaPodrska* imalo bi smisla kada bismo htjeli odvojiti ponašanje svake uloge u posebne klase. Svaka uloga se mogla implementirati kroz interfejs korisnickaUloga, a zatim za svaku ulogu kreirati zasebnu fabriku koja vraća odgovarajući objekat. Klasa Korisnik bi tada koristila apstraktnu ulogu bez znanja o konkretnoj implementaciji. Ovaj pristup olakšava dodavanje novih uloga i čini kod fleksibilnijim, ali kako su uloge u sistemu prilicno fiksne i jendostavne, mogao bi biti nepotrebna složenost.

5. Builder Pattern

Uloga Builder paterna je odvajanje specifikacije kompleksnih objekata od njihove stvarne konstrukcije. Isti konstrukcijski proces može kreirati različite reprezentacije. Koristi se ako se objekti mogu podijeliti u skupove objekata koji se razlikuje samo po permutaciji njihovih dijelova.

U kontekstu klase Voznja u našem sistemu, Builder pattern ima smisla jer proces kreiranja vožnje uključuje više koraka koji zavise jedan od drugog, kao što su dodeljivanje putnika, pronalaženje vozača, izračunavanje rute i postavljanje ocene. Ovaj postupak može biti postepen i fleksibilan, a Builder pattern omogućava lakše upravljanje tim koracima bez potrebe za velikim brojem parametara u konstruktoru.