Kreacijski paterni i njihova primjena u sistemu za iznajmljivanje vozila

U objektno-orijentisanom programiranju, kreacioni paterni predstavljaju skup rješenja za efikasno i kontrolisano kreiranje objekata. Njihova glavna uloga je da razdvoje procese instanciranja od ostatka sistema, što rezultira većom fleksibilnošću, čitljivošću i lakšim održavanjem koda. U okviru ovog rada biće predstavljeno pet osnovnih kreacionih paterna: Singleton, Factory Method, Builder, Prototype i Abstract Factory, uz objašnjenje njihove moguće primjene u postojećem klasnom dijagramu sistema za iznajmljivanje vozila.

1. Singleton Pattern

Singleton patern obezbjeđuje da od određene klase postoji tačno jedna instanca u sistemu. Koristi se kada je potrebno da klasa bude centralizovana, odnosno da svi korisnici sistema pristupaju istom objektu, npr. log menadžer, konfiguracioni fajl, ili korisnička podrška. Ovaj patern sprečava kreiranje dodatnih instanci pomoću privatnog konstruktora i statičke metode za pristup jedinom objektu.

Primjena u sistemu:

U klasnom dijagramu rent-a-car sistema, Singleton je idealan za implementaciju klase Podrška. Ova klasa predstavlja servis koji pruža pomoć korisnicima, i poželjno je da postoji jedna zajednička instanca kako bi svi zahtjevi korisnika bili usmjereni kroz centralizovani kanal. Time se izbjegavaju duplikati podataka i omogućava dosljedno upravljanje zahtjevima. Osoba i Admin klase pozivaju statičku metodu klase Podrška kako bi pristupile podršci.

2. Factory Method Pattern

Factory Method patern omogućava kreiranje objekata bez direktnog korištenja ključne riječi new, već putem metode koja odlučuje koju konkretnu klasu da instancira. Ovaj patern omogućava enkapsulaciju procesa kreiranja objekata, što olakšava održavanje i proširivanje koda.

Primjena u sistemu:

U klasnom dijagramu, Factory Method patern je primijenjen za instanciranje objekata klase Vozilo. Kreirana je posebna klasa VoziloFactory koja sadrži metodu kreirajVozilo, koja prihvata sve neophodne parametre i vraća instancu klase Vozilo. Na ovaj način izbjegava se ponavljanje koda i obezbjeđuje dosljedno i kontrolisano kreiranje objekata, bilo da vozilo dodaje korisnik, administrator ili sistemski modul.

3. Builder Pattern

Builder patern se koristi kada se objekat gradi kroz više koraka i kada konstruktor sadrži mnogo parametara. Umjesto jednog složenog konstruktora, Builder omogućava postepeno konstruisanje objekta, pri čemu se parametri mogu postavljati redom kojim korisniku odgovara.

Moguća primjena u sistemu:

U sistemu za iznajmljivanje vozila, Rezervacija je klasa sa više atributa: datum početka, datum kraja, korisnik, vozilo, status, cijena itd. Umjesto korištenja dugog konstruktora, može se kreirati klasa RezervacijaBuilder koja omogućava postepeno postavljanje svakog atributa, nakon čega se poziva metoda build() da bi se kreirao konačan objekat. Ovaj pristup poboljšava čitljivost i olakšava dodavanje opcionih atributa u budućnosti.

4. Prototype Pattern

Prototype patern se koristi kada je potrebno brzo kreirati novi objekat koji je kopija već postojećeg. Umjesto ponovnog instanciranja i postavljanja istih vrijednosti, koristi se metoda za kloniranje postojećeg objekta.

Moguća primjena u sistemu:

U rent-a-car sistemu, korisnik može često rezervisati isto vozilo za sličan period. Umjesto da svaki put kreira novu rezervaciju od nule, sistem može omogućiti kloniranje prethodne rezervacije pomoću Prototype paterna. Također, vozila koja imaju iste karakteristike (npr. flota identičnih automobila) mogu se brže duplicirati pomoću kloniranja jedne osnovne instance.

5. Abstract Factory Pattern

Abstract Factory je patern koji omogućava kreiranje čitavih porodica povezanih objekata, bez potrebe da klijent zna koje tačno klase koristi. Koristi se kada je sistem zasnovan na više varijacija međusobno povezanih objekata koje treba instancirati zajedno.

Moguća primjena u sistemu:

U budućim verzijama rent-a-car sistema, može se pojaviti potreba za podrškom različitih tipova korisnika – npr. privatni korisnik i kompanija. Svaki tip korisnika bi mogao koristiti različite verzije rezervacija, vozila, ili procesa plaćanja. Abstract Factory bi omogućio kreiranje odgovarajućih objekata u zavisnosti od tipa korisnika, čime bi se dodatno modularizovao i proširio sistem.