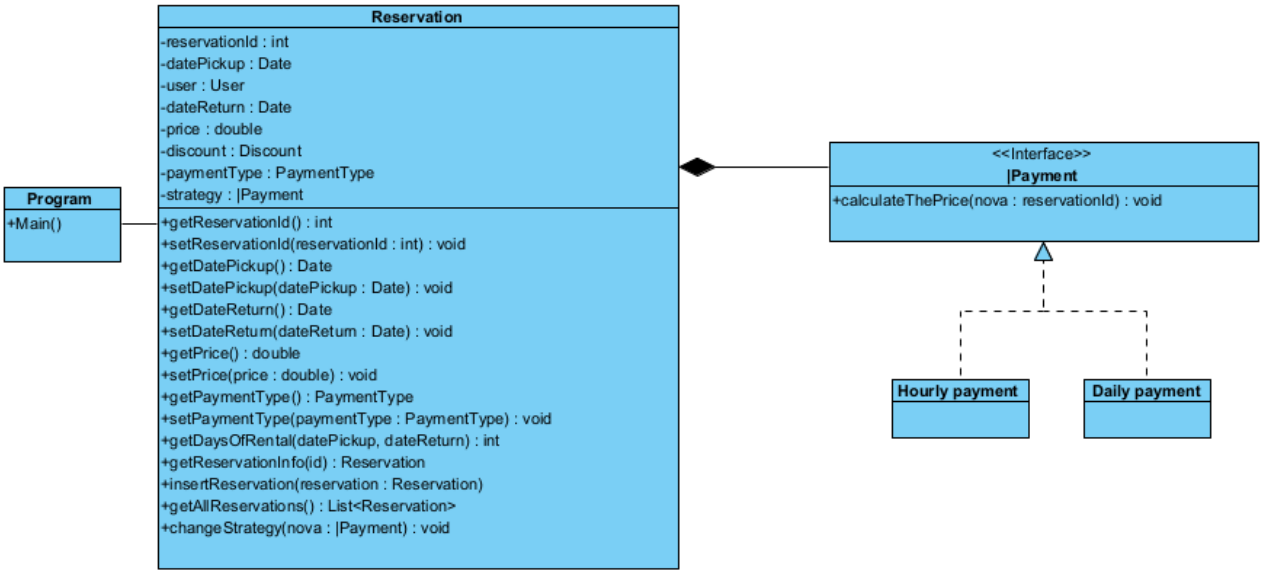
Paterni ponašanja

1. Strategy pattern

Strategy pattern omogućuje definiranje različitih algoritama ili strategija unutar objekta i omogućuje fleksibilno mijenjanje strategije tokom izvođenja programa. Ovaj pattern promoviše visok nivo fleksibilnosti, modularnosti i zamjenjivosti kodiranja.

Strategy pattern omogućuje prilagodljiv pristup obračunu cijene rezervacije vozila koristeći različite strategije. Umjesto da hardkodiramo jedan način obračuna cijene ,omogućit ćemo korisnisnicima da odaberu strategiju obračuna cijene koja najbolje odgovara njihovim potrebama. Kada korisnik rezervise vozilo, moguće je odabrati strategiju obračuna cijene koja se primjenjuje na tu rezervaciju. Na primjer, može postojati strategija koja se temelji na fiksnom iznosu, strategija koja se temelji na broju sati ili dana rezervacije, ili čak strategija koja se temelji na sezonskim faktorima ili promocijama. Kada je strategija odabrana, koristit će se za izračun cijene rezervacije. Strategija obračuna cijene može uzeti u obzir različite faktore, kao što su vremenski period rezervacije, model vozila ili posebne uslove korisnika, kako bi se odredila konačna cijena.

Ovaj design pattern ćemo implementirati u našu aplikaciju.



2. State pattern

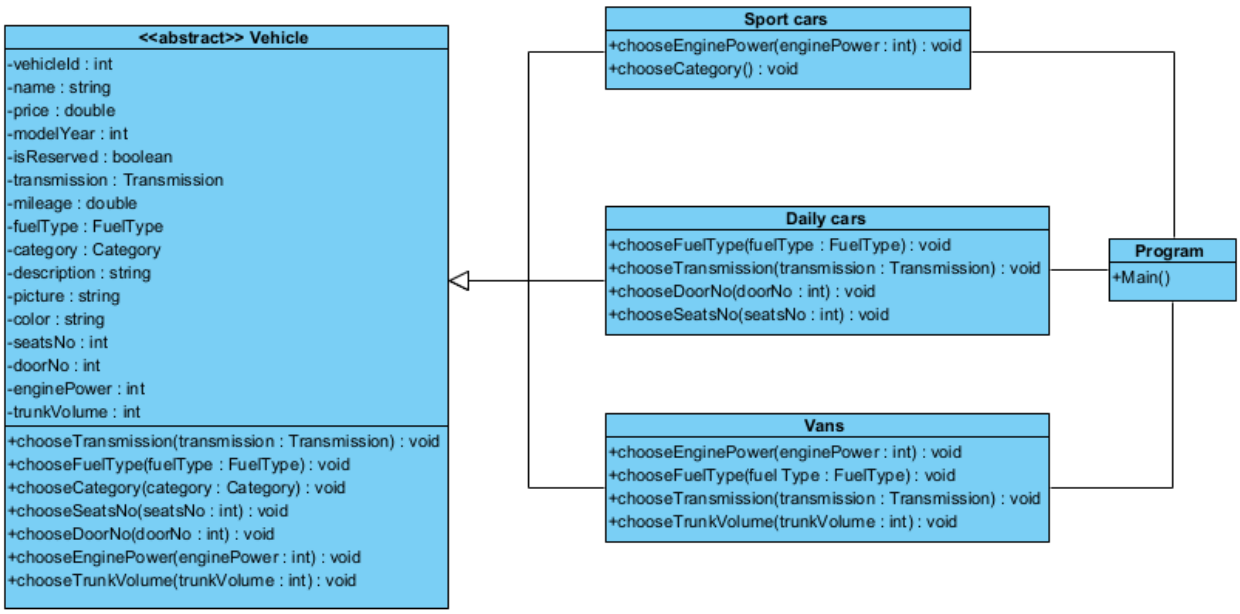
State pattern omogućava promjenu ponašanja objekta ovisno o njegovom unutrašnjem stanju. Ovaj pattern omogućava da se objekt ponaša drugačije u različitim stanjima, pri čemu svako stanje ima svoju vlastitu logiku i ponašanje.

State pattern se može iskoristiti za implementaciju različitih vrsta discounta u sistemu za rezervaciju vozila. Svako stanje predstavlja određenu vrstu popusta, a objekt rezervacije koristi trenutno aktivno stanje za izračunavanje cijene rezervacije.

3. Template Method pattern

Template Method pattern (Obrazac predloška metode) je dizajnerski pattern koji omogućava definiranje okvira ili temeljnog obrasca za izvršavanje određene operacije. Ovaj obrazac definira šablonu za algoritam, s fiksnim koracima, ali omogućava konkretnim podklasama da prilagode određene korake prema svojim potrebama

Ovaj pattern ćemo koristiti za implementaciju odabira automobila prema korisničkim zahtjevima. Apstraktna klasa definiše korake algoritma kao metode, na primjer može postojati metoda „odaberiModel“ za odabir modela automobila, metota „odaberiGodište“, i tako dalje. Konkretne podklase nasljeđuju apstraktnu klasu i implementiraju specifične korake algoritma. Na primjer, jedna podklasa može implementirati odabir sportskih modela automobila iz liste dostupnih modela, dok druga podklasa može implementirati odabir daily automobila.Kada se izvrši algoritam odabira automobila, apstraktna klasa koristi template metode koji poziva definirane korake. Ti koraci se izvršavaju koristeći implementacije iz konkretne podklase za svaki korak. Na kraju, rezultat je odabrani automobil prema korisničkim zahtjevima.
Ovaj design pattern ćemo implementirati u našu aplikaciju.



4. Observer pattern

Observer pattern, ili Oblik promatrača, je dizajnerski pattern koji se koristi za uspostavljanje odnosa jedan-na-više između objekata. Ovaj obrazac omogućuje da jedan objekt automatski obavještava sve svoje promatrače o promjenama stanja ili događajima.

Ovaj pattern se može koristiti za upravljanje rezervacijama. Možemo postaviti korisnika kao subjekt i rezervacije kao promatrače (observers). Kada se dogodi promjena u rezervaciji, korisnik će biti obaviješten o toj promjeni. Kada korisnik stvara rezervaciju, rezervacija se registrira kao promatrač korisnika. Na taj način, kada dođe do promjene u rezervaciji, korisnik će biti automatski obaviješten o toj promjeni putem Observer patterna.

5. Iterator pattern

Iterator pattern se koristi za pristupanje i prolazak kroz elemente kolekcije na jednostavan način, bez otkrivanja detalja implementacije kolekcije. Ovaj obrazac omogućuje ujednačen pristup elementima kolekcije bez obzira na to koja je konkretan tip kolekcije i kako je implementirana

Kada korisnik želi odabrati automobil, može se koristiti iterator kako bi se omogućilo prolazak kroz kolekciju automobila jedan po jedan. Korisnik može pristupiti detaljima svakog automobila, kao što su model, godina, vrsta mjenjača i druge karakteristike. Na temelju tih informacija, korisnik može donijeti odluku o odabiru automobila koji najbolje odgovara njegovim zahtjevima.

6. Chain of responsibility pattern

Chain of Responsibility pattern je dizajnerski pattern koji omogućuje slanje zahtjeva kroz niz objekata koji ga mogu obraditi. Svaki objekt u lancu ima mogućnost obrade zahtjeva, ali ako ne može obraditi zahtjev, prosljeđuje ga sljedećem objektu u lancu.

Chain of Responsibility pattern može se iskoristiti za popunjavanje podataka o vozilu ako su dostupne samo osnovne informacije. U ovom slučaju, možemo imati lanac objekata koji se bave popunjavanjem specifičnih informacija o vozilu.

7. Medijator pattern

Medijator pattern je dizajnerski pattern koji promoviše slanje i primanje poruka između objekata putem posrednika, poznatog kao medijator. U ovom patternu, objekti ne komuniciraju izravno jedni s drugima, već koriste medijator za razmjenu informacija.

Ovaj pattern može se koristiti za rezervaciju vozila. Medijator pattern omogućava da objekti korisnika i vozila komuniciraju putem medijatora. Objekti korisnika šalju zahtjeve za rezervaciju vozila medijatoru, a medijator provjerava dostupnost vozila, rezervira ih i obavještava korisnike o rezervaciji.