Kreacijski paterni

1. Factory Method

Opis:
Omogućava da se kreiranje objekata prepusti specijalizovanoj metodi, koja odlučuje koju konkretnu instancu treba vratiti. Ovaj obrazac omogućava zamjenu konkretnog tipa objekta bez promjena u kodu koji ga koristi.

Zašto je koristan?
U *vozi.ba*, imamo različite entitete poput Vozilo, Korisnik, Rezervacija, koji se često instanciraju na više mjesta u aplikaciji. Ako bi se u budućnosti dodala nova vrsta korisnika (npr. pravno lice) ili specifična klasa vozila (npr. električno vozilo), morali bismo mijenjati mnogo klasa u kojima se objekti trenutno ručno instanciraju. Korištenjem Factory Method paterna, promjene se vrše samo u jednoj klasi – u "tvornici", a ostatak sistema ostaje netaknut.
Također, ovaj obrazac omogućava uvođenje dodatne logike prilikom instanciranja objekata – kao što je validacija ulaznih podataka, zadavanje podrazumijevanih vrijednosti ili logovanje kreiranja.

2. Abstract Factory

Opis:
Omogućava kreiranje porodice međusobno povezanih objekata koji treba da rade zajedno, bez direktnog navođenja njihovih konkretnih klasa.

Zašto je koristan?
U aplikaciji *vozi.ba*, korisnik može odabrati različite „pakete“ prilikom kreiranja oglasa, kao što su „Osnovni“, „Standard“ ili „Premium“. Svaki od ovih paketa može podrazumijevati kombinaciju više usluga i objekata: dodatno osiguranje, GPS uređaj, prioritetno prikazivanje oglasa itd. Abstract Factory omogućava da se svi ti elementi kreiraju u skladu s jednim zajedničkim paketom, čime se izbjegava neusklađenost između pojedinačnih komponenti.
Bez ovog paterna, lako može doći do problema – npr. da korisnik ima GPS uređaj, ali nema odgovarajuće osiguranje ili vizualni identitet paketa nije dosljedan. Ovaj obrazac osigurava da sve komponente jedne ponude budu pravilno povezane i kompatibilne.

3. Builder patern

Opis:
Omogućava postepeno i kontrolisano kreiranje složenih objekata, naročito kada ti objekti imaju puno opcionalnih polja ili različite varijacije konfiguracije.

Zašto je koristan?
Kreiranje rezervacije u *vozi.ba* uključuje mnoštvo detalja: korisnik, vozilo, datum preuzimanja i vraćanja, lokacija, dodatne usluge poput osiguranja ili GPS-a, popusti, cijena itd. Korištenje klasičnog konstruktora sa desetak parametara dovodi do nepreglednog i greškama sklonog

koda.
Builder obrazac omogućava da se svaki od tih elemenata doda korak po korak, što je daleko čitljivije i lakše za održavanje. Takođe, korisne su tzv. fluentne metode koje omogućavaju lančano pozivanje (.dodajKorisnika(...).dodajVozilo(...).dodajLokaciju(...)).
Uz to, ovaj obrazac olakšava dinamičko dodavanje opcionalnih funkcionalnosti – npr. korisnik može ili ne mora uključiti dodatno osiguranje, ali Builder to elegantno podržava bez dodatne logike i grana u kodu.

---

4. Singleton

Opis:
Ograničava kreiranje klase na samo jednu instancu tokom trajanja aplikacije i omogućava globalni pristup toj instanci.

Zašto je koristan?
Neke komponente aplikacije *vozi.ba* imaju smisla samo kao jedinstvene instance. Na primjer:

- **Logger**: sistem koji bilježi aktivnosti korisnika (logiranje, kreiranje oglasa, greške) treba da postoji samo jedanput.

- **Servis za slanje notifikacija**: da bi se izbjeglo višestruko slanje iste poruke ili sukobi u komunikaciji s email/SMS API-jem.

- **Konfiguracija sistema** (npr. baza podataka, email postavke): treba biti jedinstvena i dostupna svuda bez višestrukog učitavanja.

Singleton obrazac omogućava da se ove komponente jednostavno pozivaju iz bilo kojeg dijela aplikacije bez potrebe za ponovnim instanciranjem. Takođe se time štedi memorija, a sistem postaje jednostavniji za testiranje i debugiranje.