STRUKTURALNI PATERNI

1. Adapter Pattern

Može se koristiti za adaptiranje interfejsa klase Vozilo kada treba integrirati sa vanjskim sistemima koji koriste drugačije interfejse. Npr. ako vanjski sistem očekuje podatke o vozilu u JSON formatu, a naša klasa Vozilo nudi podatke u XML formatu, adapter može poslužiti kao posrednik.

2. Decorator Pattern

Može se koristiti za dinamičko dodavanje novih funkcionalnosti klasama Korisnik ili Vozilo bez mijenjanja njihove osnovne strukture. Npr. kreirati VoziloDecorator apstraktnu klasu i konkretne dekoratore kao što su OsiguranjeDecorator, GPSDecorator itd.

3. Facade Pattern

Može se kreirati fasada koje će pojednostaviti kompleksne operacije koje uključuju više klasa (npr. proces rezervacije koji uključuje Korisnik, Vozilo i Rezervacija). Kreiranjem RentACarFacade klase koja će obuhvatiti sve operacije vezane za iznajmljivanje vozila, uključujući rezervacije, recenzije i upravljanje korisnicima obuhvatio bi se ovaj pattern.

4. Composite Pattern

Korisno kada treba tretirati pojedinačne objekte i kompozicije objekata na isti način. Npr. kreiranje paketa usluga za iznajmljivanje koji se sastoje od više komponenti (osnovni paket: vozilo + osiguranje, premium paket: vozilo + osiguranje + GPS + dječja sjedalica). Svaki paket može se tretirati kao pojedinačna usluga

5. Proxy Pattern

Može se koristiti za kontrolisanje pristupa objektu klase Vozilo (npr. lazy loading slika vozila). Također može poslužiti kao zaštitni proxy za klasu Korisnik pri autentifikaciji i autorizaciji.

6. Bridge Pattern

Korisno za odvajanje apstrakcije (npr. korisničkog interfejsa) od implementacije (npr. načina čuvanja podataka). Npr. odvajanje načina prikaza informacija o vozilu od samih podataka o vozilu.

7. Flyweight Pattern

Može se koristiti za optimizaciju memorije kada ima mnogo instanci sa sličnim podacima. Npr. ako ima mnogo vozila istog modela, zajedničke karakteristike (kao što je slika modela) mogu se dijeliti umjesto da se čuvaju za svako vozilo posebno.