KREACIJSKI PATERNI

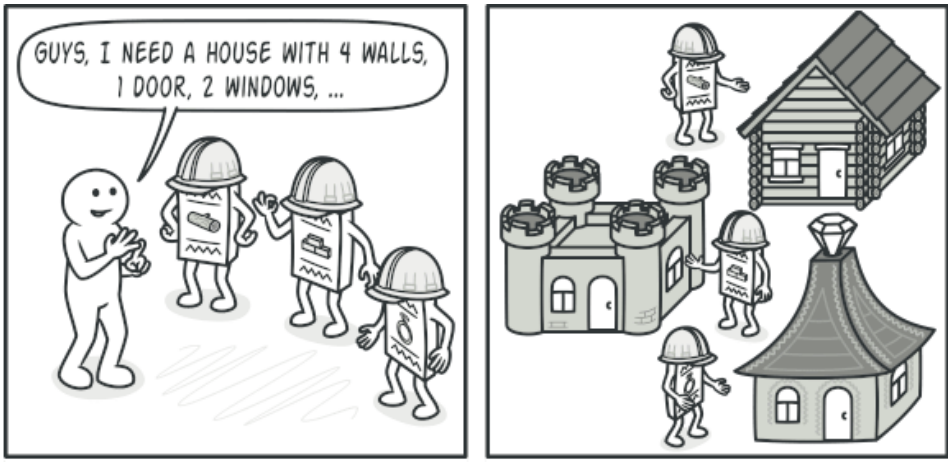
1. FACTORY METHOD

Patern koji omogućava interfejs za kreiranje objekata u superklasi, ali dozvoljavajući podklase da mijenjaju tip objekata koji će se stvoriti.

Ukoliko bi se naš sistem rasprostranio širom svijeta kako bi ga koristile mnogobrojne države za svoje lige ovaj patern bi bio veoma koristan i zaista bi bilo potrebno iskoristiti ga, a evo i zbog čega.
Svaka država ima neku svoju ligu koja se možda razlikuje od drugih liga po nekim pravilima i sadržajima, recimo broja timova, poseban ranking sistem ukoliko dva kluba imaju isti broj bodova pa da li će se recimo gledati međusobni odnos ili pak gol razlika i slično. Uprkos svim tim razlikama opet svaka liga ima istu svrhu i isti cilj kao i veoma velik broj identičnih metoda naravno. Obzirom da svaka liga vodi evidenciju o postignutom broju pogodaka i asistencija igrača, broju kartona, te evidenciju o rezultatima odigranih utakmica kao i raspored prethodećim.

2. BUILDER

Builder patern omogućava konstrukciju složenih objekata korak po korak. On nam dozvoljava izradu različitih tipova i zastupanja objekata koristeći isti konstrukcijski kod.
Ukoliko bi se naš sistem proširio sa dodatkom pregleda stadiona za svaki tim koji je registrovan, te ukoliko bi se sistem organizovao na način da se pravi 3D stadion sa objektima na koje se može kliknuti mišem da se obavi neka metoda, to bi bilo mnogo komplikovana hijerarhija stvaranja takvog stadiona. Jedan jednostavan stadion ima najmanje jednu tribinu, teren, pomoćni toalet za posjetitelje (vanjski eventualno unutrašnji), parking te nužno svlačionice! Najjednostavnije (ali ne i najefikasnije) rješenje bi bilo extendati klasu Stadion i kreirati set podklasa pokrivši razne kombinacije ovih datih parametara, ali šta ukoliko neki stadion ima još neke dodatne zgrade ili možda 3 tribine umjesto 1 i sl. Rješenje je u ovom paternu na način da se naprave opcije za izgradnje pomoćnih objekata koji se lahko mogu imenovati, dakle ukoliko stadion ima hitnu pomoć, prvo odaberemo buildera koji će oblikovati zgradu da izgleda poput hitne pomoći, zatim ćemo odabrati kreiranje zgrade. Slično tome možemo odabrati buildera za tribine (jer jedan builder pravi tribinu sa krovom, drugi bez itd.) zatim ćemo kreirati dovoljan broj tribina.



3. SINGLETON

Još jedan od kreacijskih paterna, on omogućava da osiguramo da klasa ima samo jednu instancu i istovremeno pružajući globalnu pristupnu tačku ovoj instanci.

Tima je izvrstan primjer ovog paterna. Tim može imati samo jedinstven stručni štab. Bez obzira na lični identitet pojedinaca koji čine stručni štab, naziv “Stručni štab Real Madrida” (npr.) je pristupna tačka koja identificira grupu odgovornih ljudi.

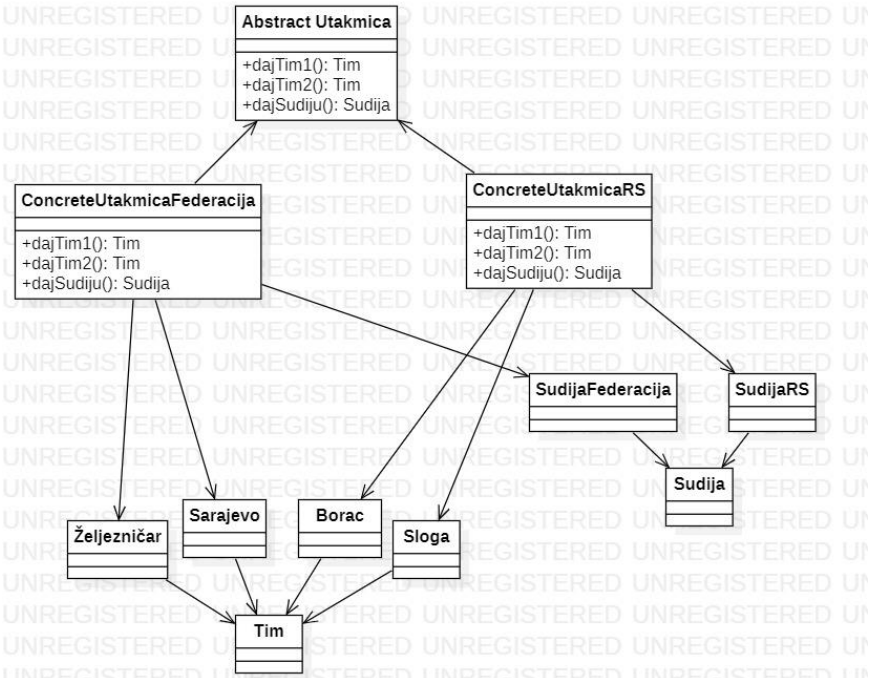
4. ABSTRACT FACTORY

Ovaj patern omogućava da stvorimo familiju srodnih objekata bez navođenja njihovih konkretnih klasa. Abstract Factory i Factory Method su veoma slični, a osnovna razlika između njih je to što Factory Method patern stvara jedan objekat, dok Abstract Factory stvara više objakata. Abstract Factory je veoma moćan i koristan patern jer je u mogućnosti da kreira familiju konkretnih klasa koje se sastoje od više objekata koji ne moraju nužno biti istog tipa.

Primjer upotrebe u našem sistemu:

Recimo, imamo klasu FactoryUtakmica koja će dati dva tima i sudiju i niz klasa ConcreteUtakmica koje će „grupisati“ koja to dva tima i sudija su moguća „kombinacija“ iz nizova klase Tim i klase Sudija. Da bi grupisanje imalo smisla, potrebno da je budu makar dvije naslijeđene (različite) klase Tim, kao i makar dvije različite klase Sudija.

Primjer upotrebe, UML dijagram:



5. PROTOTYPE

Posljednji kreacijski patern koji omogućava kopiranje postojećih objekata bez da kôd zavisi od klase objekata.

Recimo da postoje dvije lige, Premier liga i Liga prvaka. Jedan igrač može igrati obje lige (zašto da ne), ali želimo da razgraničimo njegove podatke u te dvije lige na način da su to kao dvije različite osobe jer zaista igrač igra posebno te dvije lige. To se može obaviti jednostavnim kopiranjem (prvo kreiranjem novog objekta iste klase, a zatim prolaženjem kroz sva originalna polja neophodna za opis igrača kopirajući ih u novokreirani objekat). Zvuči jednostavno ali postoji “caka”. Ne mogu se svi objekti kopirati na ovaj način jer neki atributi mogu biti privatni i njihov pristup može biti nevidljiv van samog objekta. Također, da bi se napravio duplikat, moramo znati tačno koje je klase objekat pa kôd postaje više ovisan o toj klasi. Štaviše ponekad samo znamo interfejs koji taj objekat prati, ali ne i konkretnu klasu, kada npr. parametar metode prihavata objekat koji prati neki interfejs.
Zato je ovaj patern rješenje jer delegira postupak kloniranja na stvarne objekte koji se kloniraju. Deklariše se zajednički interfejs za sve objekte koji podržavaju kloniranje. Ovime se omogućava kloniranje objekta bez povezivanja koda sa klasom tog objekta. Obično takav interfejs sadrži samo jednu metodu kloniranja.