KREACIJSKI PATERNI

1. Singleton patern

Njegova uloga je da osigura da se klasa može instancirati samo jednom i da osigura globalni pristup kreiranoj instanci klase.

Singleton patern bi se mogao iskoristiti u našem sistemu ukoliko bi se zahtijevalo da postoji samo jedan administrator. U tom slučaju klasa *Administrator* bi postala *Singleton klasa*, te bi se u bilo kojem pristupanju ovoj klasi uvijek dobivala ista instanca kao rezultat.

2. Factory Method patern

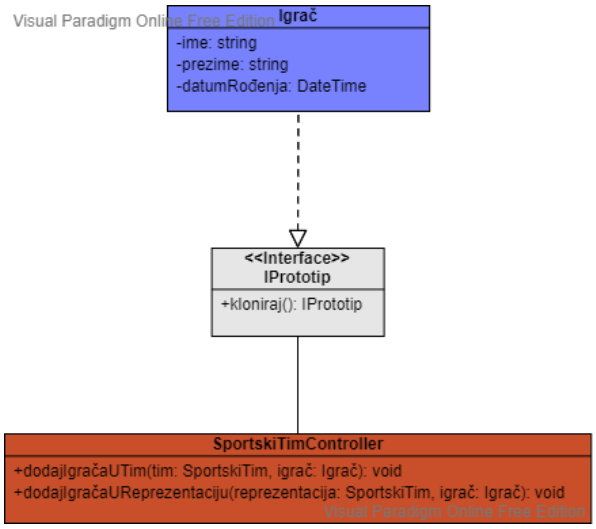
Ovaj patern omogućava interfejs za kreiranje objekata u superklasi, ali dozvoljavajući da podklase mijenjaju tip objekata koji će se stvoriti.

Naš sistem bi mogao implementirati ovaj patern. Naime, kako se naš sistem bavi online plaćanjem karata za sportske događaje, bitno je naglasiti da nema svaka država (ako bi naš sistem postao globalan) isti kriterij vrednovanja utakmica, formiranja tabela za lige i slično. Međutim, većina metoda je identična neovisno o kriterijima.

3. Prototype patern

Uloga *Prototype paterna* je da kreira nove objekte klonirajući jednu od postojećih prototip instanci (postojeći objekat). *Prototype patern* dozvoljava da se kreiraju prilagođeni objekti bez poznavanja njihove klase ili detalja kako je objekat kreiran. Ovaj patern omogućava kopiranje postojećih objekata bez da kod zavisi od klase objekata.

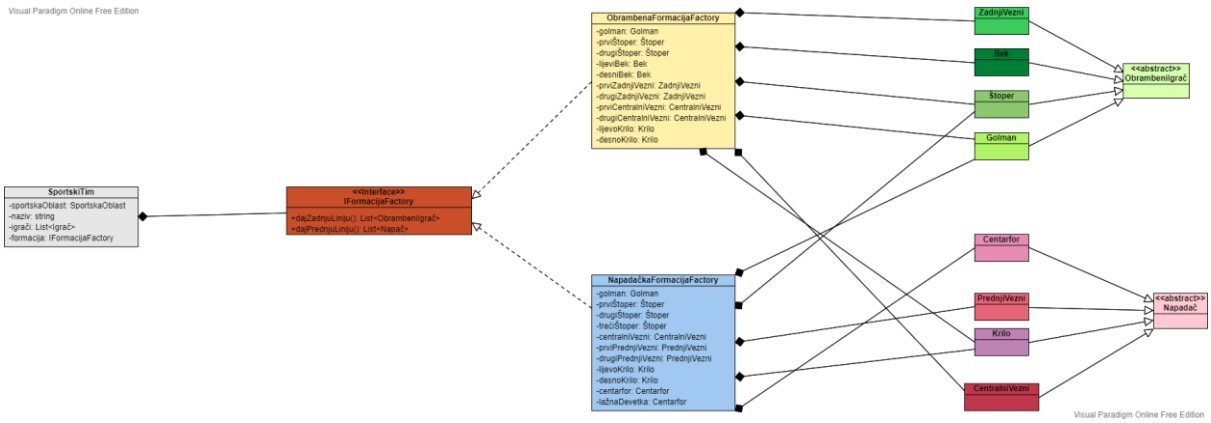
Recimo da u našem sistemu postoje dva različita tima (a ima ih više). Jedan igrač može igrati za oba tima (reprezentacija i klub) ali želimo da razgraničimo njegove podatke u ta dva tima na način da su to kao dvije različite osobe jer zaista igrač igra posebno u ta dva tima. Obično kopiranje nije rješenje ako uzmemo u obzir da su neki atributi privatni i zaštićeni. 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.



4. Abstract Factory patern

Ovaj patern omogućava da stvorimo familiju srodnih objekata bez navođenja njihovih konkretnih klasa.

Abstract Factory patern bi se mogao uključiti u naš sistem ukoliko bi se vodilo računa o tome koje igrače posjeduje koji tim s aspekta pozicija na kojima igraju ti igrači. Recimo, naprimjer, da razmatramo samo nogomet kao sport. Naime, sve igrače bi mogli razdijeliti u dvije skupine koje bi predstavljale dvije apstraktne klase. Jedna skupina bi mogla predstavljati obrambene igrače, gdje bi iz apstraktne klase *ObrambeniIgrač* mogle biti izvedene konkretne klase *Golman, Štoper, Bek* i slično. Druga skupina bi mogla predstavljati napadače, gdje bi iz apstraktne klase *Napadač* mogle biti izvedene konkretne klase *Krilo, Centarfor* i slično. Kako u nogometu postoje razne formacije u kojima jedna ekipa može igrati, a koje pri tome ne obuhvataju igrače istih pozicija, onda bi se kao *factory klase* mogle navesti klase koje reprezentuju razne formacije. Npr. jedna formacija bi imala golmana, dva štopera, dva krila i slično, a druga bi mogla imati golmana, tri štopera, centarfora i slično. Na slici je predstavljen isječak iz dijagrama klasa koji predstavlja implementaciju ovog paterna u našem sistemu. Treba samo istaći da naknadno mogu biti dodane neke *factory* klase ukoliko se javi potreba za definisanjem nove nogometne formacija, ali to svakako ne bi zahtijevalo nikakve izmjene u ovako formiranom sistemu (samo dodatke).



5. Builder patern

Builder patern omogućava konstrukciju složenih objekata korak po korak. On nam dozvoljava izradu različitih tipova i zastupanja objekata koristeći isti konstrukcijski kod.

Builder patern bi se mogao uključiti u naš sistem. Naime, prilikom vršenja plaćanja mogu se dobiti potvrde o uspješnoj uplati na kojima će stajati određene informacije kao što su ime kupca, plaćena cijena, izvor sa kojeg je plaćeno i slično. S obzirom da su omogućena dva načina plaćanja, i to kartično i telefonsko plaćanje, potvrde mogu imati nešto drugačiji sadržaj za ta dva različita tipa uplate, međutim, u osnovi način konstrukcije im je jako sličan. Svaka potvrda će imati sve iste podatke osim izvora sa kojeg je plaćeno, tj. kod kartičnog plaćanja to će biti broj računa, a kod telefonskog broj telefona. U tom smislu, potvrde će se malo razlikovati u konačnom izgledu, ali načini konstrukcije ne bi trebali imati suštinsku međusobnu razliku. Na slici ispod prikazan je isječak iz dijagrama klasa koji objašnjava način implementacije ovog paterna u našem sistemu.