05.11.2021.

Objektno-orijentisano programiranje 2

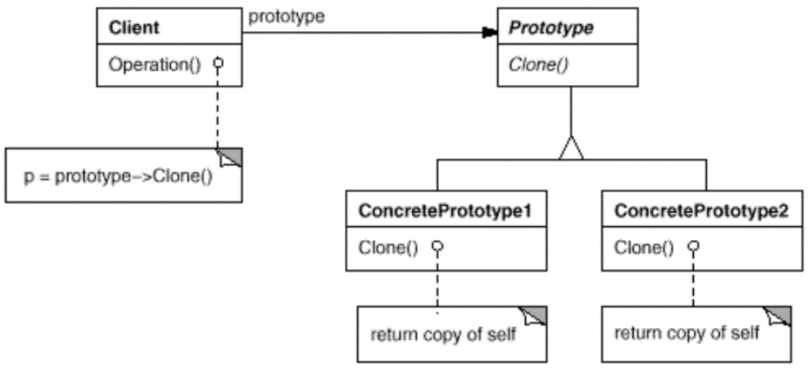
Prototip

Motivacija i osnovna ideja:

Osnovna ideja uzorka je da da omogućimo lako kreiranje objekata ***po uzoru*** na drugi objekat. Zamislimo da imamo objekat za čije je kreiranje potrebna kompleksna logika. Možda je za kreiranje objekta potrebno izvršiti kompleksnu računicu, pozvati neki API čije se korišćenje plaća, izvršiti upite nad udaljenom bazom podataka i sl. Ako su objekti određene klase čije je kreiranje kompleksno slični po tome da svi objekti moraju izvršiti istu kompleksnu računicu (*prilikom koje će doći do **istog** rezultata*) i da se međusobno razlikuju samo po atributima koji se lako zadaju/modifikuju, umesto da svaki objekat date klase kreiramo "od nule" koristeći pritom i kompleksnu logiku, dovoljno je da određenu kompleksnu logiku izvršimo jednom za kreiranje jedne instance date klase, a da ostale objekte klase kreiramo *klonirajući* i naknadno *modifikujući* objekat. Objekat koji kloniramo nazivamo **prototipom.** Kloniranje ostvarujemo metodom koju obično nazivamo *clone()*. U opštem slučaju, imamo više konkretnih prototipova izvedenih iz apstraktne klase Prototip.

Ovaj projektni uzorak naziva se i polimorfnom kopijom. Druga važna primena je kod kopiranja objekta izvedene klase, a na osnovu pokazivača osnovne klase. Naime, zamislimo da dobijemo pokazivač osnovnog tipa (koji pokazuje na objekat nekog izvedenog tipa) i da želimo da izvršimo kopiju datog objekta. Koji konstruktor kopije pozvati ako ne znamo o kom se tačno izvedenom tipu radi? Metoda clone() rešava ovaj problem – svaka izvedena klasa zna kako da se kopira/klonira u vreme izvršavanja.

Učesnici projektnog uzorka Prototip i njihova komunikacija dati su UML dijagramom prikazanim na slici 1.



Slika 1 Projektni uzorak Prototip

Napomena: često se konkretni prototipovi kreiraju fabričkom metodom na osnovu prosleđenog ključa.

Zadaci:

1. Implementirati opšti oblik projektnog uzorka Prototip na osnovu UML dijagrama sa slike 1.
2. Implementirati klase:
    a. IButton koja od atributa ima tekst, boju pozadine (zadaje se kao string oblika #rrggbb) i čisto virtuelnu funkciju opis().
    b. Button koja nasleđuje IButton i od dodatnih atributa ima širinu i visinu.
    c. RadioButton koja nasleđuje IButton i od dodatnih atributa ima id (tipa string) grupe kojoj pripada.
    d. CheckBox koja nasleđuje klasu IButton i od dodatnih atributa ima bool checked.

   Omogućiti da kloniranje ovih klasa i dobijanje novih objekata pozivom metode/funkcije oblika getButton(ButtonType) koja će vratiti kopiju postojeg prototipa dugmeta datog tipa. getButton treba da prvo proveri da li postoji dugme tipa ButtonType i da izvrši kloniranje ukoliko postoji, a ukoliko ne postoji da pre kloniranja napravi prototip datog tipa dugmeta.

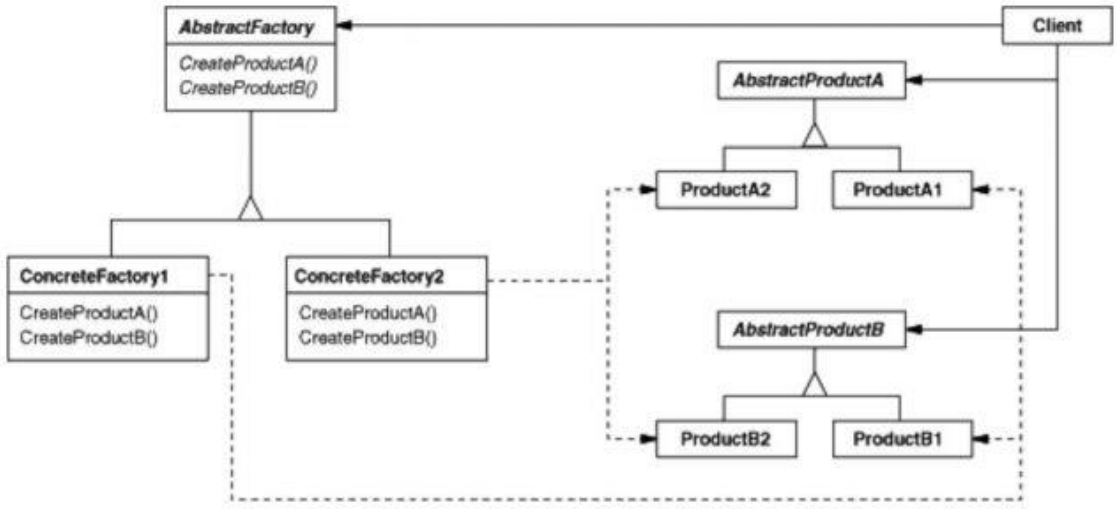
3. Napraviti klasu Paragraf koja od atributa ima tekst, veličinu fonta, stil fonta, poravnanje i informaciju o tome da li treba uvući prvi red paragrafa (može se iskoristiti kod sa prethodnih vežbi). Paragraf može da se kreira pozivom konstruktora (uz navođenje svih potrebnih atributa), a treba omogućiti registrovanje novog stila paragrafa i kasnije lako kreiranje paragrafa istog stila. Na primer, mogli bismo kreirati paragraf p kao
   p = new Paragraph(…);
   i navesti:
   registerParagraph(„moj stil“, p);
   Ukoliko nam kasnije treba paragraf sa istim podešavanjima koje smo postavili paragrafu p mogli bismo prosto navesti:
   Paragraph* p2 = getParagraph(“moj stil”);
   p2->setText(“Tekst novog paragrafa”);

Apstraktna fabrika

Motivacija i osnovna ideja:

Kao i kod ostalih stvaralačkih projektnih uzoraka želimo da izbegnemo direktno kreiranje korišćenjem konstruktora. Pretpostavimo da imamo više klasa raznoraznih proizvoda koje možemo da grupišemo u različite *familije* proizvoda. Na primer, u našem kodu kreiramo klase koje odgovaraju različitim elementima grafičkog interfejsa, a razlikuju se za različite operativne sisteme. Na primer, možda imamo klase IButton iz koje su izvedene klase WindowsButton, LinuxButton, MacOSButton i IWindow iz koje su izvedene klase WindowsWindow, LinuxWindow i MacOSWindow. Tada bismo smatrali da imamo 3 familije proizvoda (za svaki operativni sistem po jednu). U našem kodu ne želimo da mešamo proizvode različitih familija. U ovakvim slučajevima, kreiranje proizvoda možemo prepustiti takozvanim **fabrikama**. Apstraktna fabrika pružila bi interfejs za kreiranje objekata tipa IButton i IWindow, a izvedene **konkretne fabrike** mogle bi se koristiti za kreiranje konkretnih proizvoda samo jedne familije. Tako bismo imali fabriku Linux komponenti, fabriku Windows komponenti i fabriku Mac OS komponenti. Korisnik komunicira sa fabrikom i ne može da kreira raznorodne objekte. Štaviše, ne mora da zna ni kako se konkretne klase zovu – interesuje ga samo osnovni tip proizvoda i iz koje je familije proizvoda.

Učesnici projektnog uzorka Apstraktna fabrika i njihova komunikacija dati su UML dijagramom prikazanim na slici 2.



Slika 2 Projektni uzorak Apstraktna fabrika

Zadaci:

4. Implementirati opšti oblik projektnog uzorka Apstraktna fabrika na osnovu UML dijagrama sa slike 2.
5. Kreirati apstraktne klase Stolica, Sto i Polica (sa čisto virtuelnom metodom opis()) i konkretne klase DrvenaStolica, DrveniSto, DrvenaPolica, ModernaStolica, ModerniSto, ModernaPolica kao i fabrike za proizvode ove dve familije proizvoda.
6. Napraviti klasu Soba koja ima stolicu, sto i policu. Sobu treba kreirati objekat klase Dizajner na način:
   Soba s = dizajner.kreirajSobu(stilSobe);
   Možemo imati dva graditelja koja će kreirati sobu postavljajući nameštaj iz određene fabrike (jedan graditelj koristi jednu fabriku, drugi drugu). Dizajner bi igrao ulogu upravljača/direktora. Ukoliko konkretne graditelje kreiramo na osnovu stila sobe (enum ili string) onda imamo takođe i primenu fabričkog metoda koji će kreirati graditelja.

   Pitanje: da li bismo mogli da kloniramo sobu? Postoje li neke prepreke za to?