17.12.2021.

Objektno-orijentisano programiranje 2

Interpreter

Motivacija i osnovna ideja:

Projektni uzorak interpreter može se koristiti u slučaju da je potrebno raditi sa izrazima ili nekim *jezikom* koji je definisan simbolima i pravilima (*gramatikom*). Zadatak interpretera je da u zavisnosti od nekog *konteksta* interpretira iskaze datog jezika. Tipičan primer je jezik iskazne logike u kome od konstanti, simbola promenljivih i logičkih veznika mi gradimo formule. Kontekst formule u iskaznoj logici bio bi dodela vrednosti promenljivama – kažemo da je neka formula u datom kontekstu (konkretno, u logici rekli bismo u valuaciji) tačna/netačna.

Interpreter mora biti u stanju da za dati kontekst interpretira formulu bilo da je ona konjunkcija, disjunkcija, promenljiva, konstanta itd. Uočimo da neke od formula ne mogu da se rastave na podformule (promenljive i konstante). Takvi izrazi nazivaju se **terminalnim**. Ostale formule mogu se rastaviti na podformule (izraze) i njihova vrednost u datom kontekstu (valuaciji, za dati primer) definisana je rekurzivno (na primer, konjunkcija dve formule je tačna akko su obe formule tačne – zahteva se interpretacija podformula). Takve izraze nazivamo **neterminalnim.** Neterminalne izraze u terminologiji jezika mogli bismo nazvati pravilima ili gramatikom datog jezika (na primer, ako su p i q formule i validni iskazi u datom jeziku, i $p \wedge q$ je validan iskaz, tj. "gramatički ispravan").

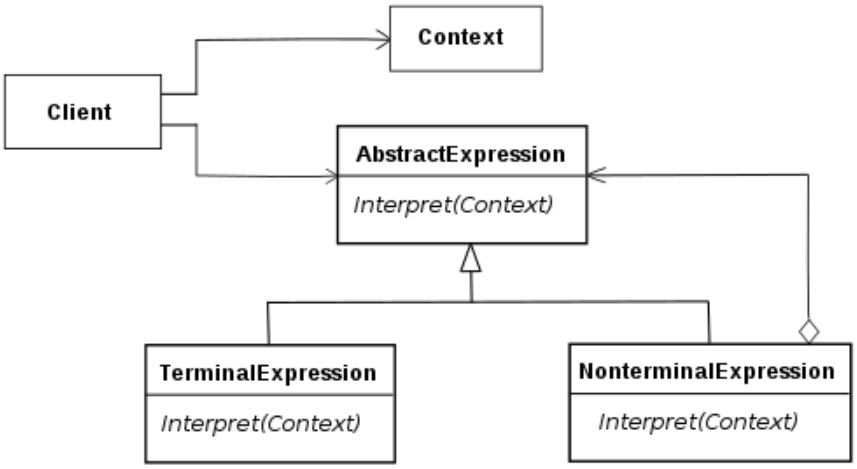
Celokupni izraz koji treba interpretirati može se predstaviti *stablom apstraktne sintakse* koje je sastavljeno od terminalnih i neterminalnih izraza, pri čemu su terminalni izrazi listovi. Projektni uzorak interpreter zahteva postojanje zasebne klase za svako pravilo jezika (logičke veznike u našem primeru) i klase za terminalne izraze (promenljive i konstante). Sve klase treba izvesti iz zajedničke klase apstraktni izraz koji ima metodu interpretiraj(Kontekst). Neterminalni izrazi pozivaju metodu interpretiraj nad svojim podizrazima.

Klijent kreira izraze i obezbeđuje kontekst u kome ih treba interpretirati.

Napomena: ukoliko interpreter koristimo za izračunavanje izraza, interpreter i zastupnik korišćen za odloženo izračunavanje su takoreći isti, s razlikom da odloženo izračunavanje ne koristi kontekst kao objekat nezavistan od izraza. I interpreter i zastupnik kod odloženog izračunavanja su kompoziti.

Napomena 2: interpreter kao projektni uzorak ne definiše kako ćemo da kreiramo izraz, već samo kako da omogućimo njegovo interpretiranje. Izraz često može biti zadat kao string, a zadatak parsiranja je da kreira stablo. Primeri stringova za interpretiranje: SQL upiti, aritmetički/logički izrazi, regularni izrazi za pretragu teksta.

UML dijagram projektnog uzorka Interpreter dat je na slici 1.



Slika 1: Projektni uzorak Interpreter

Zadaci:

1. Implementirati klase koje omogućavaju kreiranje logičkog izraza linijom:
   auto izraz = AND(OR(NOT(x), y), AND(z, NOT(OR(p, q))));
   Dodatno: omogućiti kreiranje izraza na osnovu stringova umesto promenljivih i dodavanjem konstanti. Omogućiti, na primer, kreiranje izraza sledećom linijom:
   auto izraz = AND(OR(NOT(„x“), y), AND(true, NOT(OR(„z“, „x“))));
2. Omogućiti kreiranje istovetnog izraza iz zadatka 1, ali na osnovu stringa. Na primer, izraz bi se kreirao kao:
   Izraz* = Izraz::parsiraj(“AND(OR(NOT(x), y), AND(true, NOT(OR(z, x)))”);
   Napomena: zadatak 2 treba da se oslanja na nasleđivanje i korišćenje virutalnih metoda, a neterminalni izrazi treba da imaju pokazivače na izraze.

   Dodatne ideje: kod zadatka 1 omogućiti kreiranje izraza preklapanjem operatora ||, && i !. Dodati kod oba zadatka metodu ispiši() kojom se ispisuje dobijeni izraz.