

Projektovanje softvera

Interpreter



Interpreter (1)

- Ime i klasifikacija:
 - Interpreter – klasni uzorak ponašanja
- Namena:
 - za dati jezik, definiše:
 - reprezentaciju njegove gramatike
 - interpreter koji koristi tu reprezentaciju da interpretira iskaze jezika

Interpreter (2)

- Motivacija:
 - u nekim aplikacijama:
 - pojedini zahtevi se mogu predstaviti iskazima jednostavnog jezika
 - može se napraviti interpreter koji rešava zahteve interpretirajući iskaze
 - primer: pretraga teksta na osnovu zadatog iskaza (uzorka za pretragu)
 - iskaz predstavlja opis skupa mogućih niski koje se traže u tekstu
 - potrebno je u tekstu pronaći bilo koju pojavu niske koja odgovara opisu
 - regularni izrazi su standardan jezik za iskaze koji opisuju skupove niski
 - algoritam pretrage može da pokušava uparivanje dela teksta koji se pretražuje interpretirajući regularan izraz koji opisuje skup niski za uparivanje
 - uzorak *Interpreter* opisuje:
 - kako definisati gramatiku (gramatička pravila) za jednostavne jezike
 - kako predstaviti iskaze u jeziku i
 - kako interpretirati te iskaze

Interpreter (3)

- Motivacija (nastavak):

- sledeća gramatika definiše regularne izraze:

```
izraz = redjanje | izbor | ponavljanje | grupisanje | literal;
```

```
redjanje = izraz, "&", izraz;
```

```
izbor = izraz, "|", izraz;
```

```
ponavljanje = izraz, "*";
```

```
grupisanje = "(", izraz, ")";
```

```
literal = znak, {znak};
```

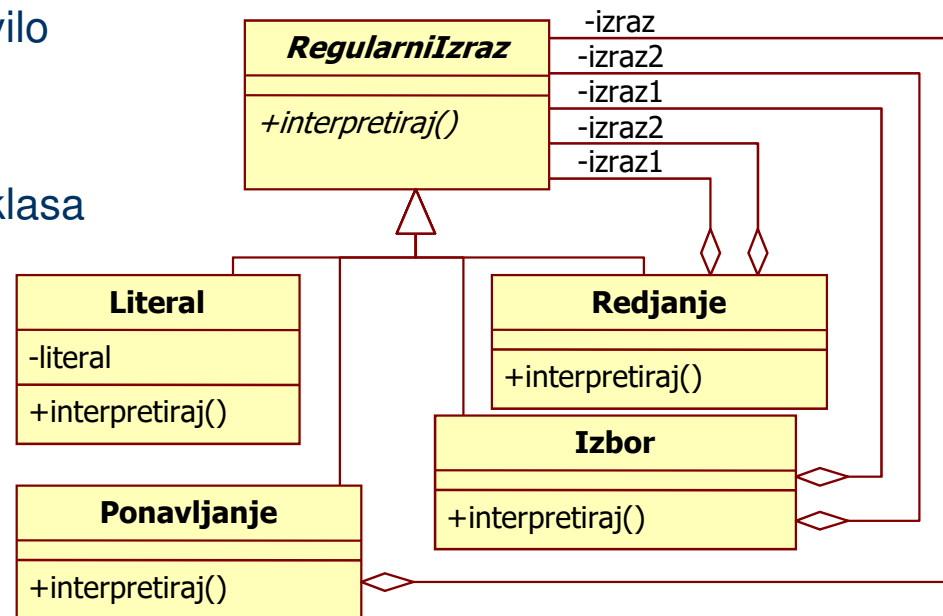
```
znak = "a" | "b" | "c" | ... ;
```

- na primer - zadavanje uzorka niski za pretragu:

```
projektni & uzorci & (stvaranja | strukture | ponašanja)
```

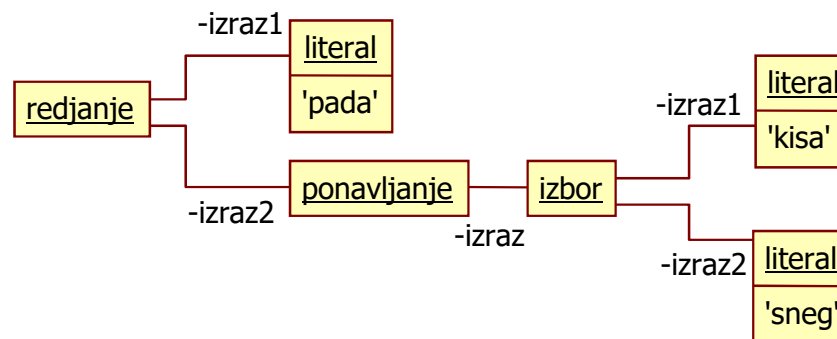
Interpreter (4)

- Motivacija (nastavak):
 - uzorak *Interpreter* koristi po klasu da reprezentuje gramatičko pravilo
 - simboli na levoj strani pravila određuju klase interpretera
 - data gramatika se opisuje sa 5 klasa
 - neterminalni simboli na desnoj strani pravila odgovaraju objektima klasa koje se koriste u opisu pravila
 - napomena: na dijagramu nisu predstavljene sve moguće klase interpretera (Izraz, Grupisanje, Znak)



Interpreter (5)

- Motivacija (nastavak):
 - desna strana pravila određuje sastavljanje objektne strukture
 - svaki regularni izraz definisan na datoj gramatici može da se predstavi jednim stablom apstraktne sintakse (SAS) sastavljenim od objekata datih klasa
 - primer:
 - regularni izraz: `pada & (kisa | sneg) *`
 - odgovarajuće SAS:



Interpreter (6)

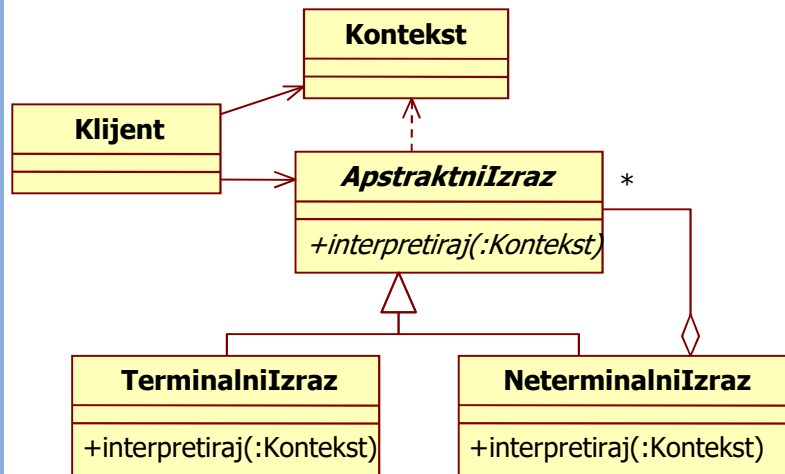
- Motivacija (nastavak):
 - može se napraviti interpreter za ovakve regularne izraze
 - definiše se `interpretiraj()` operacija za svaku potklasu `RegularniIzraz`
 - `interpretiraj()` dobija kao argument kontekst u kojem interpretira izraz
 - kontekst sadrži ulazni tekst i informaciju o tome koji njegov deo je već obrađen
 - svaka potklasa `RegularniIzraz` implementira `interpretiraj()` da upari sledeći deo ulaznog teksta u datom kontekstu
 - na primer:
 - `Literal` proverava da li ulaz odgovara literalu koji definiše
 - `Redjanje` proverava da li ulaz odgovara sledu njegovih izraza
 - `Izbor` proverava da li ulaz odgovara jednom od njegovih izraza
 - `Ponavljjanje` proverava da li ulaz ima ponavljanja njegovog izraza

Interpreter (7)

- **Primenljivost:**
 - uzorak treba primeniti kada
 - postoji jednostavan jezik koji treba interpretirati
 - iskazi jezika se mogu predstaviti kao stabla apstraktne sintakse (SAS)
 - čvorovi u SAS su objekti klasa koje predstavljaju gramatička pravila jezika
 - najbolji rezultati se dobijaju pod sledećim uslovima
 - gramatika je jednostavna
 - za kompleksne gramatike broj klasa u hijerarhiji postaje preveliki za održavanje
 - efikasnost nije kritična
 - efikasni interpreteri se ne implementiraju interpretiranjem stabala parsiranja direktno, već se najpre ona transformišu u neku drugu formu
 - na primer – regularni izrazi se često transformišu u automate stanja, ali i tada se translator može implementirati pomoću uzorka Interpreter

Interpreter (8)

- **Struktura:**



- **Učesnici:**

- ApstraktniIzraz (klasa RegularniIzraz)
 - deklarira apstraktnu operaciju `interpretiraj()` koja je zajednička za sve čvorove u SAS
- TerminalniIzraz (klasa Literal)
 - implementira `interpretiraj()` operaciju za terminalne simbole u gramatici
 - po primerak se zahteva za svaki terminalni simbol u iskazu
- NeterminalniIzraz (klase Redjanje, Izbor, Ponavljanje)
 - zahteva se po jedna klasa za svako gramatičko pravilo $R=R_1 R_2 \dots R_n$
 - sadrži objekte tipa `ApstraktniIzraz` za svaki od simbola $R_1 \dots R_n$
 - implementira op. `interpretiraj()` za neterminalne simbole tako što se ova poziva rekurzivno za objekte simbola $R_1 \dots R_n$
- Kontekst (klasa Tekst)
 - sadrži informaciju koja je globalna za interpreter
- Klient
 - gradi (ili je već dato) SAS koje predstavlja pojedini iskaz na jeziku koji definiše gramatika
 - SAS je sastavljeno od objekata klasa `NeterminalniIzraz` i `TerminalniIzraz`
 - poziva `interpretiraj()` operaciju korena SAS

Interpreter (9)

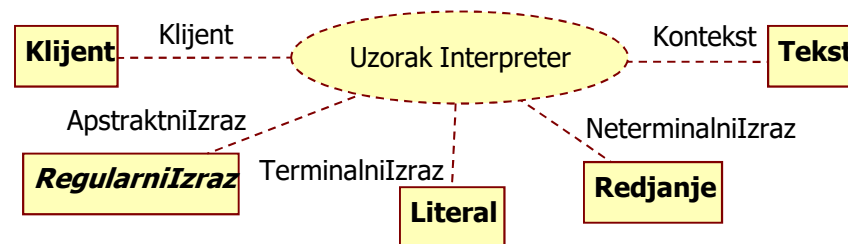
- Saradnja:
 - klijent gradi (ili je već dat) iskaz kao SAS od objekata terminalnih i neterminalnih izraza
 - zatim klijent inicijalizuje kontekst i pokreće operaciju `interpretiraj()` korena SAS
 - svaki čvor neterminalnog izraza definiše operaciju `interpretiraj()` tako što poziva operaciju `interpretiraj()` svojih podizraza
 - operacija `interpretiraj()` svakog terminalnog izraza definiše krajnju tačku u rekurziji
 - operacija `interpretiraj()` u svakom čvoru koristi kontekst da smesti stanje i pristupi stanju interpretacije

Interpreter (10)

- Posledice:
 - lako je menjati i proširivati gramatiku
 - klase reprezentuju pravila
 - ova se mogu menjati i dodavati nova kroz izvođenje
 - lako je implementirati gramatiku
 - klase koje definišu tipove čvorova SAS imaju slične implementacije
 - generisanje ovih klasa može čak da bude automatizovano
 - kompleksne gramatike je teško održavati
 - za svako gramatičko pravilo se definiše barem jedna klasa
 - gramatike sa mnogo pravila rezultuju u velikom broju klasa
 - treba primenjivati druge tehnike (generatori parsera/prevodioca)
 - dodavanje novih načina za interpretaciju izraza
 - dodavanje novog načina za interpretaciju izraza (npr. `proveraTipa()`) zahteva dodavanje nove operacije u sve klase izraza
 - ako se ovo radi više puta, treba razmotriti uzorak *Posetilac*

Interpreter (11)

- UML notacija:



- Povezani uzorci:
 - SAS je primerak projektnog uzorka *Kompozicija*
 - Muva* omogućava da se efikasno dele terminalni simboli u SAS
 - Interpreter* može da koristi *Iterator* za obilazak SAS
 - ako se operacija `interpretiraj()` realizuje u posebnoj klasi, onda je moguće lako dodavati druge načine interpretacije - *Posetilac*