Tema 0 : COOL Lists ;)

Autor: Neculai Balaban
Responsabil : Ionuț Mihalache

1 Introducere

COOL (Classroom Object Oriented Language) este un limbaj de programare didactic. Compilatoarele scrise de la zero sunt proiecte enorme, iar standardele pot fi foarte stufoase. Cool este un limbaj îndeajuns de simplu încât se poate scrie un compilator pentru acesta pe parcursul unui semestru. Fiind un limbaj didactic, comunitatea în cadrul limbajului este practic inexistentă: nu există biblioteci standard (sau de orice tip), nici interes pentru a extinde limbajul.

În această temă veți fi puși în locul unui dezvoltator de bibliotecă. Neavând la dispoziție decât elementele de bază ale limbajului, va trebui să dezvoltați o funcționalitate cât mai eficientă și extensibilă cu putință. Tema aceasta are ca scop familiarizarea cu mecanismele limbajului COOL pentru ca, în momentul în care se pune problema arhitecturării compilatorului, să aveți o vedere mai de ansamblu asupra limbajului, și astfel, să puteți structura codul mai bine.

2 Cerința

Să se implementeze o listă generică, capabilă să dețină elemente de orice tip. Aceasta va trebui să suporte operații comune, de uz general, ce vor fi detaliate în secțiunile următoare.

Se va implementa un program care citește date de la STDIN și populează lista cu obiecte din acele date. Obiectele sunt definite în fișierul "things.cl", și vor fi, pe lângă cele implicite din limbaj, singurele care vor trebui tratate în cadrul acestei teme. După ce toate obiectele sunt citite, programul **va intra într-un prompt interactiv** în care utilizatorul va putea:

- încărca altă listă
- aplica operații pe oricare listă
- aplica operații între 2 liste
- ieși din aplicație

Listele de obiecte citite vor fi stocate la rândul lor într-o listă.

3 Intrarea

Cool nu posedă biblioteci de citire din fișiere. Din acest motiv, lista va fi populată cu obiecte primite la STDIN. Obiectele se regăsesc într-un fișier text, și sunt declarate pe mai multe linii, după următorul format:

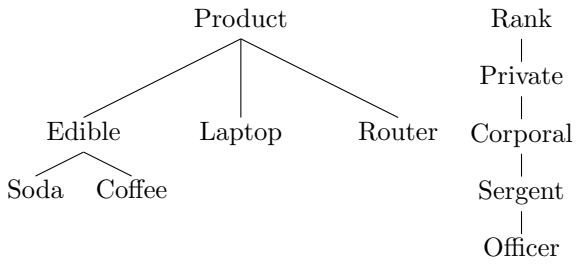
```
tip_clasă atribut1 atribut2 ...
....
tip_clasă atribut1 atribut2 ...
END
```

Fiecare linie conține tipul obiectului, urmat de parametrii funcției de inițializare (aidoma unui constructor). În cazul obiectelor predefinite în COOL, acestea vor fi însoțite de un singur parametru (excepție făcând clasa IO care nu va avea niciun parametru).

La finalul fiecărui fișier de intrare, se va regăsi pe ultima linie cuvântul "END". Introducerea de obiecte inexistente va rezulta în apelul funcției *abort()* și încheierea execuției programului.

4 Obiectele

Obiectele prevăzute pentru această temă se împart în 2 categorii: cea de produse (moștenesc clasa "Product"), și cea de ranguri militare (moștenesc clasa "Rank").



Intuitiv clasele Product, Edible și Rank au rol de clase abstracte, însă în COOL nu există acest concept.

Obiectele sunt definite în scheletul de cod pentru temă, cu tot cu funcțiile de inițializare.

5 Funcționalități

Acest paragraf va enumera comenzile **din promptul interactiv** ce trebuiesc implementate, cât și funcționalitatea lor. Această secțiune **NU** detaliază toate funcțiile din spate pe care va trebui să le implementați. Aici aveți puțină libertate, însă pentru punctaj maxim va trebui să vă gândiți bine cum scrieți codul pentru a fi cât mai modular și extensibil.

În cazul în care o comandă din cele detaliate mai jos primește un parametru invalid (e.g. o opțiune inexistentă sau un index care nu există), se va apela funcția *abort()* și se va încheia execuția programului.

5.1 help

O comandă care să listeze operațiile posibile. Nu va fi testată pentru punctaj.

5.2 load

Această comandă va intra într-un mod în care se așteaptă obiecte noi de la STDIN. Formatul este identic cu cel prezentat în categoria "Intrarea". Noua listă va fi separată de cele existente, și va fi stocată la finalul listei de liste.

5.3 print

Va afișa toate listele încărcate în memorie, în ordinea în care apar în lista de liste. Fiecare listă va fi prefixată de indicele din cadrul listei, iar elementele vor fi între paranteze pătrate.

```
1: [ Laptop(SNSV;AX23K), Coffee(DasKaffee;500g), Soda(NukaCola;0.33) ]
2: [ Router(ChinkTik;2CIP4U) ]
3: [ Colonel(Mike), Corporal(John) ]
4: [ String(Hayo), Int(43), Bool(false), IO() ]
```

Reprezentarea sub format string a obiectelor va fi compusă din: numele clasei, urmat de primele 2 atribute ale clasei, separate de caracterul ';'. În cazul în care clasa are un singur atribut (e.g String), se va afișa între paranteze doar acel atribut.

În cazul în care se primește un număr, se va afișa doar lista de la acel index. Dacă indexul nu există, se va apela funcția *abort()*.

5.4 merge

Va fi apelată din promptul interactiv sub următoarea formă:

```
merge index1 index2
```

Lista ce se află la index2 va fi concatenată la finalul celei aflate la index1. Cele două liste vor fi eliminate, iar noua listă va fi inserată, la final. De exemplu, execuția **merge 2 3** peste lista de obiecte din 5.3 va rezulta în:

```
1: [ Laptop(SNSV;AX23K), Coffee(DasKaffee;500g), Soda(NukaCola;0.33) ]
2: [ String(Hayo), Int(43), Bool(false), IO() ]
3: [ Router(ChinkTik;2CIP4U), Colonel(Mike), Corporal(John) ]
```

5.5 filterBy

Această metodă va fi definită în cadrul clasei List. Aceasta va primi o funcție, sub formă de obiect cu o metodă. Se vor implementa două clase pentru filtrare:

- ProductFilter - elimină din listă obiecte ce nu moștenesc Product
- RankFilter - elimină din listă obiecte ce nu moștenesc Rank

Pe lângă aceste două clase, se va implementa și filtrul **SamePriceFilter**, care va păstra în listă doar obiectele ale căror preț calculat (care include taxe și TVA) este egal cu prețul calculat dacă produsul ar fi fost unul generic (doar TVA). Cu alte cuvinte, filtrul păstrează obiectele ale căror metodă *get_price()* returnează o valoare identică cu cea pe care ar returna-o aceeași metodă dacă obiectul ar fi castat la clasa Product.

Modul de apel din promptul interactiv este următorul:

```
filterBy index {ProductFilter,RankFilter,SamePriceFilter}
```

unde index indică care listă va fi filtrată.

5.6 sortBy

Va sorta lista după o anumită funcție. Aceasta va fi definită în cadrul unei clase, iar această clasă va fi pasată către metoda *sort()* a clasei List. Modalitatea este similară cu clasa *Comparator*[1] din Java.

Va trebui să implementați următoarele clase comparator:

- PriceComparator - sortează produse după prețul de bază al acestora
- RankComparator - sortează obiectele după ierarhia militară (de la Private la Colonel)
- AlphabeticComparator - sortează obiectele de tip String alfabetic.

Dacă oricare dintre comparatoarele de mai sus întâlnește un "obiect străin" (i.e. nu aparține acelei ierarhii), programul se va termina prin apelul funcției *abort()*.

În promptul interactiv, comanda va avea forma:

```
sortBy index {PriceComparator,RankComparator,AlphabeticComparator} {ascendent
,descendent}
```

pentru un total de 6 interpretări posibile. Index indică care listă va fi sortată.

6 Precizări și recomandări

Indexarea listelor va începe de la poziția 0. În promptul interactiv se va face afișarea începând de la poziția 1.

Un prim lucru de care vă veți lovi în implementarea funcției load, va fi acela că *in_string()* va citi linii întregi. Astfel, e nevoie să implementați o funcționalitate de a despărți string-uri după spații. Recomandăm să vă implementați o clasă similară cu *StringTokenizer*[2]. din java.

Un alt lucru de care veți avea nevoie este o implementare *atoi()*. Puteți să împrumutați această funcție din exemple.

Scheletul poate fi modificat în totalitate: puteți reorganiza sursele, crea clase intermediare, etc. În momentul testării, niciuna dintre surse nu va fi suprascrisă.

6.1 Rulare

Interpretorul este capabil să proceseze doar un singur fișier *.cl*. De aceea, scheletul de cod vine cu un script de rulare care soluționează această problemă.

Scriptul de rulare se invocă în următorul fel:

```
./run.sh input1
```

Acesta poate primi și mai multe fișiere de intrare, ce conțin fie obiecte, fie comenzi. Trebuie să vă asigurați că concatenarea acestora rezultă într-o secvență care are sens. De exemplu, dacă avem în cmd1.txt o instrucțiune *load* și două fișiere de intrare, putem invoca scriptul în felul următor

```
./run.sh input1.txt cmd1.txt input2.txt
```

[1] www.geeksforgeeks.org/comparator-interface-java/
[2] www.geeksforgeeks.org/stringtokenizer-class-java-example-set-1-constructors/

7 Punctare

Tema va fi testată automat pe platforma VMchecker. Punctajul va fi acordat după următoarea schemă:

- 30% - load
- 10% - print
- 10% - merge
- 20% - filterBy
- 30% - sortBy

Se vor acorda **penalizări pentru conding style** până în 20% din punctajul temei. Din moment ce nu există un standard de-facto de aranjare a codului, nu vom depuncta indentări mai puțin lizibile, și ne vom limita predominant la **hardcodări** și **abateri de la cerință**. Prin "*Hardcodare*" ne referim la secvențe lungi de cod care deși sunt funcționale, există o altă modalitate de scriere a lor care ar fi fie mai scurtă, mai scalabilă sau mai ușor extensibilă.