QUIZ IV

Subject: Java Collections and Recursions
TA: Selim Yilmaz

Due Date: 05/19/2019 23:59:59

1 Introduction

In this quiz work, you are expected to handle Backus-Naur Form (or BNF) using Java Collections (i.e., maps, sets, list, and the like), which enables you to store and manipulate a group of objects. This work comprises of two consecutive stages *i*) construction of collection and *ii*) generation of all possible words through a **recursive** manner. In the following, BNF notations as well as all the remarks that you should care to complete this quiz work has been explained in detail.

2 Backus-Naur Form (BNF)

In computer science, Backus–Naur Form is one of the main notation techniques often used to describe the syntax of the languages, such as *computer programming languages*, *document formats*, *instruction sets*, *communication protocols*, and the like.

A BNF consists of:

- a set of terminal symbols
- a set of non-terminal symbols
- a set of production rules

Left hand side of a production rule represents a non-terminal symbols where right-hand side is a sequence of symbols. A BNF grammar example for a phone number like (123)456-7890 (or like 123-4567) is given below:

```
<phone-number> -> <area-code><7-dig>|<7-dig>
<area-code>    -> "("<digit><digit><digit>")"
<7-dig>        -> <digit><digit><digit>"-"<digit><digit><digit><digit>
<digit>        -> "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
```

where all items between angle brackets are the non-terminal symbols and others stated within "." are the terminals representing the end items. Note that all non-terminal symbols should be expanded until they are parsed into the terminal symbols, which is a task of yours within the 2nd stage of this work.

3 The Scope of the Work

With the scope of this quiz work, you are given a file containing a structured BNF grammar, then expected to construct a collection framework so that it stores all the symbols in the BNF.

To do that, you need to employ a reasonable/appropriate collection class. In addition, you should also define the type of the data that is stored within the collection (i.e., a list should be defined as `ArrayList<String> cars = new ArrayList<String>()` for `String` data). To simplify the complexity of the problem, do not mind the case where recursion may occur in the BNF grammar like below (where the symbol `item` might call itself again and again):

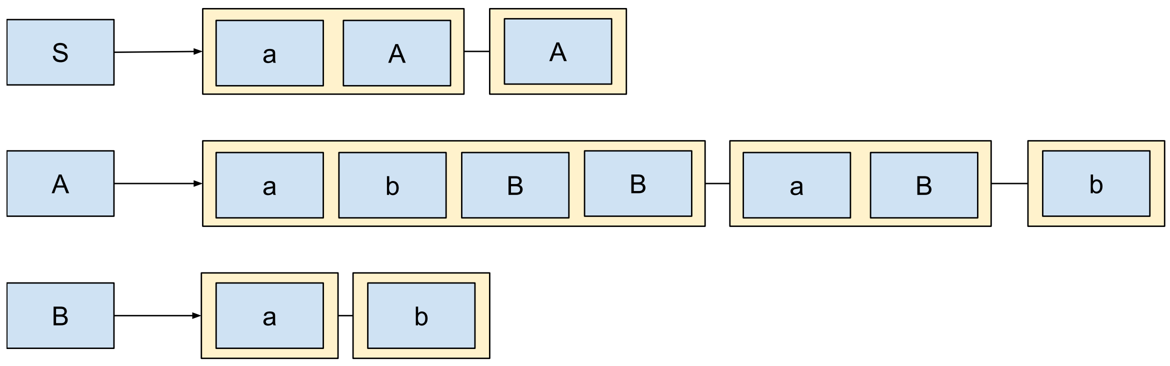
```
<item> -> <item>"X"|"X"
```

Instead of angle brackets, letter case has been used to distinguish terminal and non-terminal symbols for the sake of simplification. The pipe symbols '|' are used to indicate multiple items that can be expanded by a corresponding non-terminal item (see the example below). The starting non-terminal symbol should be labeled with 'S' indicating the item to be first expanded.

```
S->aA|A
A->abBB|aB|b
B->a|b
```

As stated earlier, you are expected to first construct an appropriate collection with respect to a given BNF grammar then recursively generate all possible words in a defined grammar.

A demonstration to data structure that should be generated from the BNF grammar above is given below:



The following will be the only production of your implementation representing the expansion form with respect to the given BNF given above. It is worth to mention again that it should be generated through a **recursion** considering collection object(s) that you defined in your implementation. The step by step generation of this expansion form is explained thereafter.

```
(a(ab(a|b)(a|b)|a(a|b)|b)|(ab(a|b)(a|b)|a(a|b)|b))
```

```
-----------------------------------------------------------------
| Step | Expansion Form                                         |
-----------------------------------------------------------------
|   1. | S                                                      |
|   2. | (aA|A)                                                 |
|   3. | (a(abBB|aB|b)|A)                                       |
|   4. | (a(ab(a|b)B|aB|b)|A)                                   |
|   5. | (a(ab(a|b)(a|b)|aB|b)|A)                               |
|   6. | (a(ab(a|b)(a|b)|a(a|b)|b)|A)                           |
|   7. | (a(ab(a|b)(a|b)|a(a|b)|b)|(abBB|aB|b))                 |
|   8. | (a(ab(a|b)(a|b)|a(a|b)|b)|(ab(a|b)B|aB|b))             |
|   9. | (a(ab(a|b)(a|b)|a(a|b)|b)|(ab(a|b)(a|b)|aB|b))         |
|  10. | (a(ab(a|b)(a|b)|a(a|b)|b)|(ab(a|b)(a|b)|a(a|b)|b))     |
-----------------------------------------------------------------
```

4 A Complete Example

- The content of BNF file:

```
S->abaAC|Ca
A->BaC|B|cC
B->a|b|C|c
C->a|c
```

- Use the following command with the argument to start your implementation:

```
      java Main /path/to/input/file
e.g., java Main /home/user/quiz4/BNF.dat
```

- The only thing that your implementation should produce is: (note that: no whitespace should be printed among letters or symbols)

```
(aba((a|b|(a|c)|c)a(a|c)|(a|b|(a|c)|c)|c(a|c))(a|c)|(a|c)a)
```

5 Submission Notes

- You are given a template source code file with this quiz paper. Read all the instructions stated within that file carefully!
- The quiz must be original, individual work. All the duplicate or Internet works (even if a citation is provided) are both going to be considered as cheating.
- You will submit your work from our department's submission system with the file hierarchy as below: This file hierarchy must be zipped before submission (Not .rar, only .zip files are supported by the system)

→ <student id.zip>
    → Main.java[1] <FILE>

[1]a template file that is given to you along with this paper. Read the comments carefully within it!