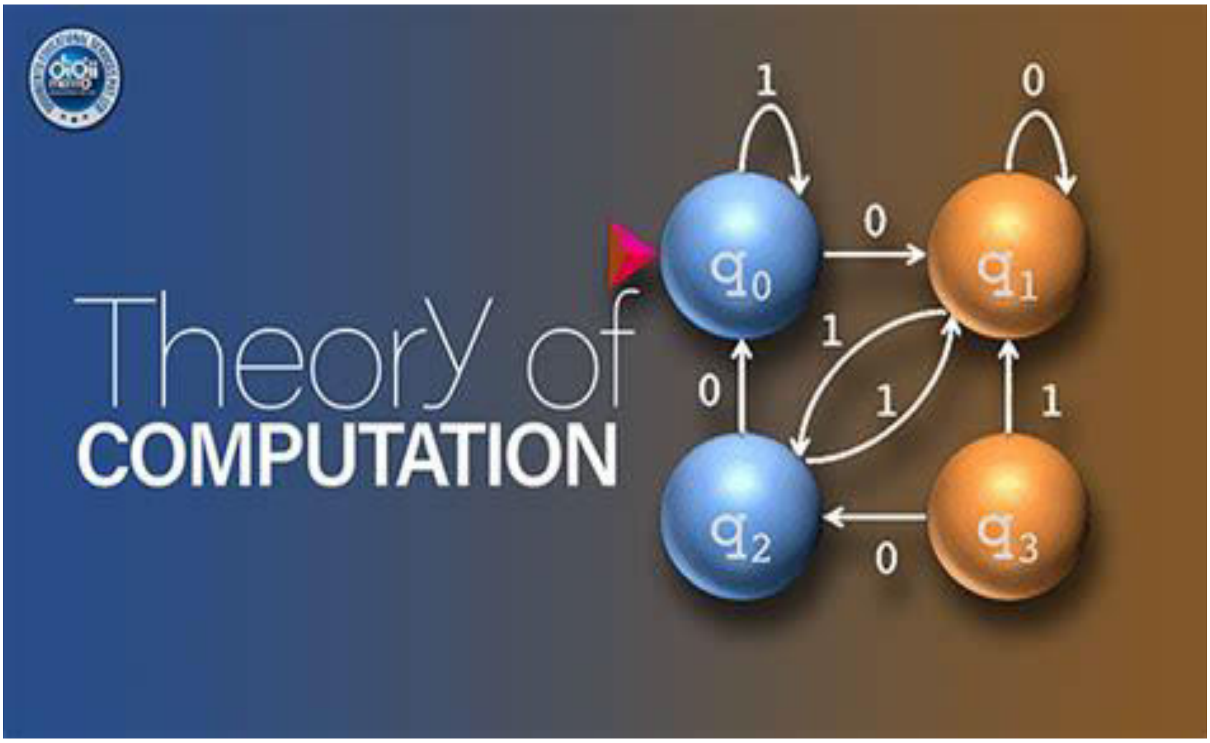
UNIVERSITY OF KWAZULU-NATAL ™

INYUVESI YAKWAZULU-NATALI

COMP314 – Theory of Computation

Assignment 2 – Group Y

218 067 924	**Hakeem Hoopdeo**
218 009 114	Ricardo Aaron Pillay
217 047 781	Kyle Naidoo
217 047 727	Thembelihle Mpofana

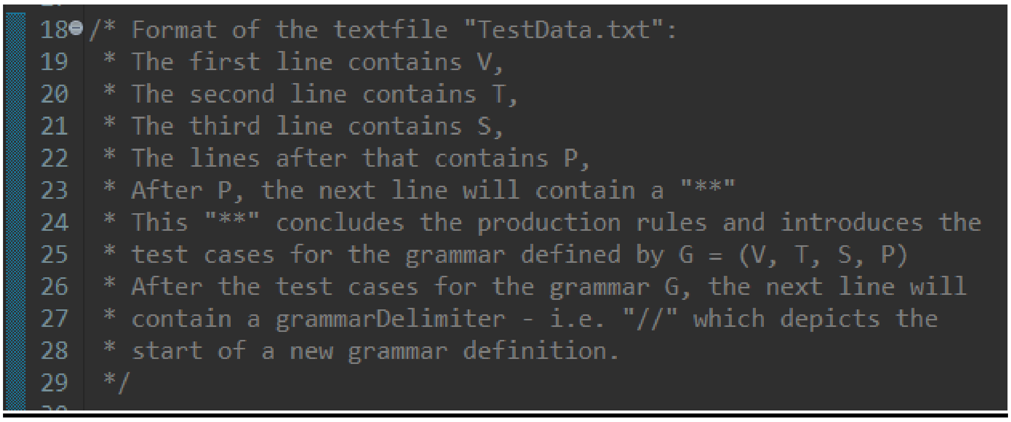


Introduction:

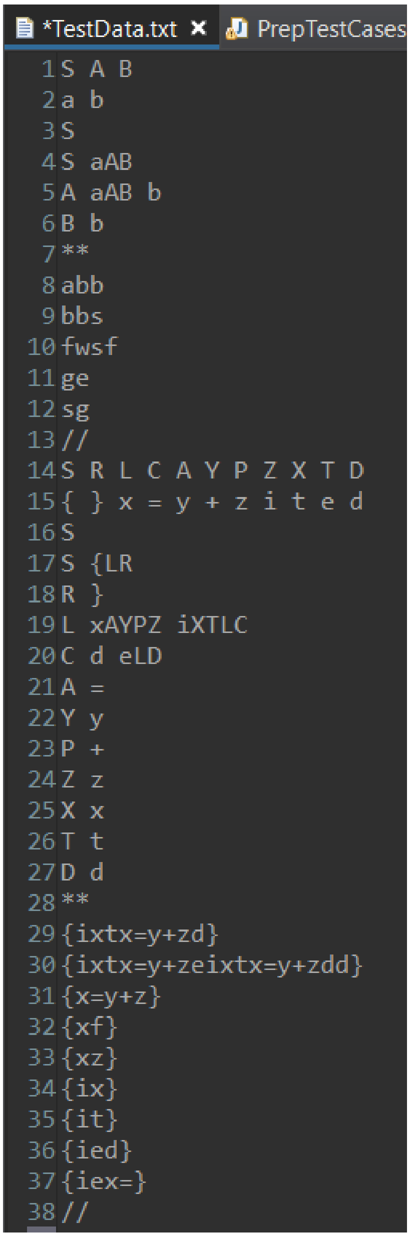
The purpose of this project was to work as a group and write a program that, given a grammar G = (V, T, S, P), & given strings for that grammar, write code to:

1. Read the grammar G from the *testData* file
2. Check if the grammar is a valid *s-grammar*
3. Throws an exception if it is not a valid *s-grammar*
4. Read the strings for that *s-grammar* from the *testData* file if it is a valid *s-grammar*
5. Parse the strings to determine whether the strings are members of the language that is defined by G.

Format of our text file (“testData.txt”):



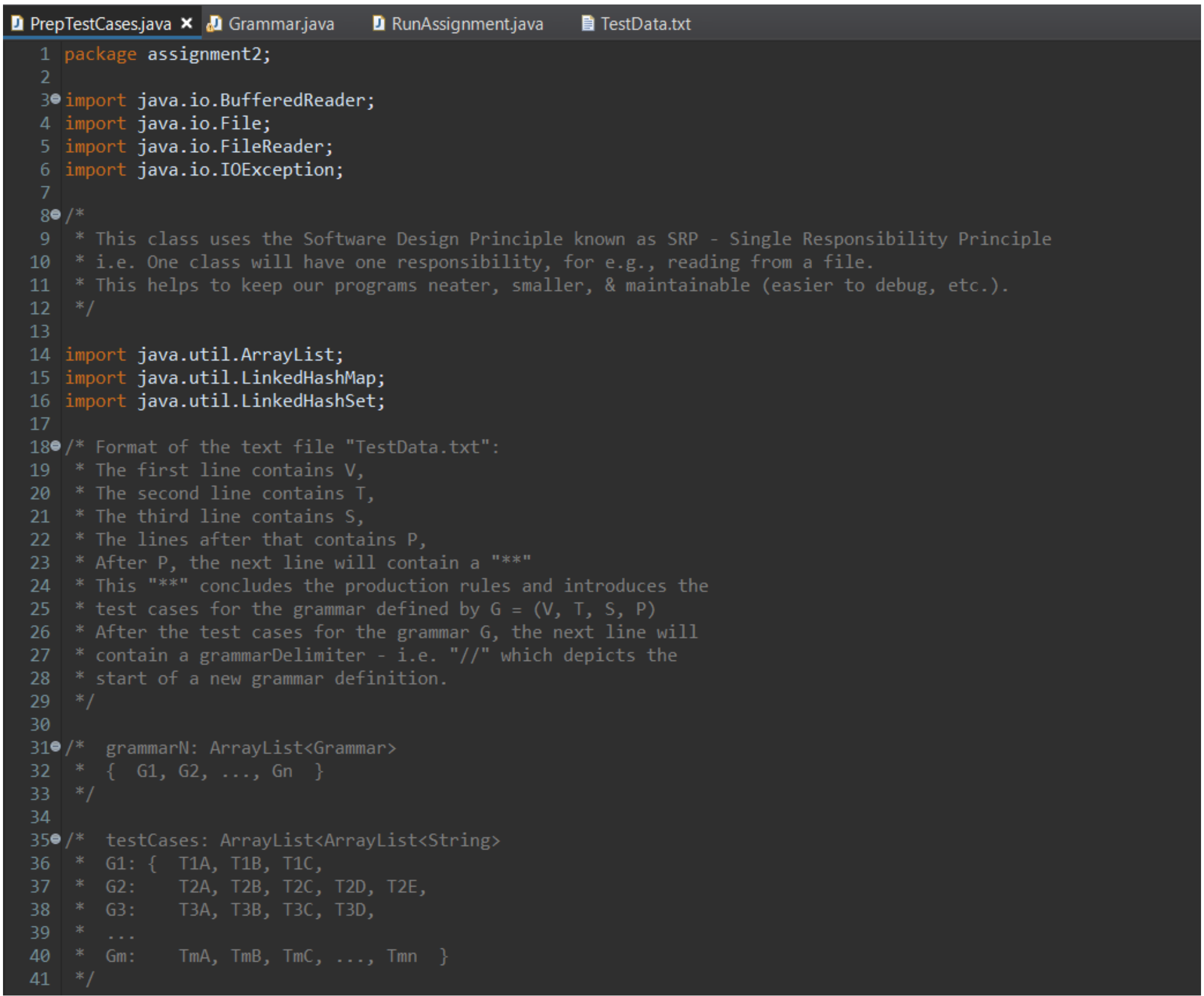
```
/* Format of the textfile "TestData.txt":
 * The first line contains V,
 * The second line contains T,
 * The third line contains S,
 * The lines after that contains P,
 * After P, the next line will contain a "**"
 * This "**" concludes the production rules and introduces the
 * test cases for the grammar defined by G = (V, T, S, P)
 * After the test cases for the grammar G, the next line will
 * contain a grammarDelimiter - i.e. "//" which depicts the
 * start of a new grammar definition.
 */
```



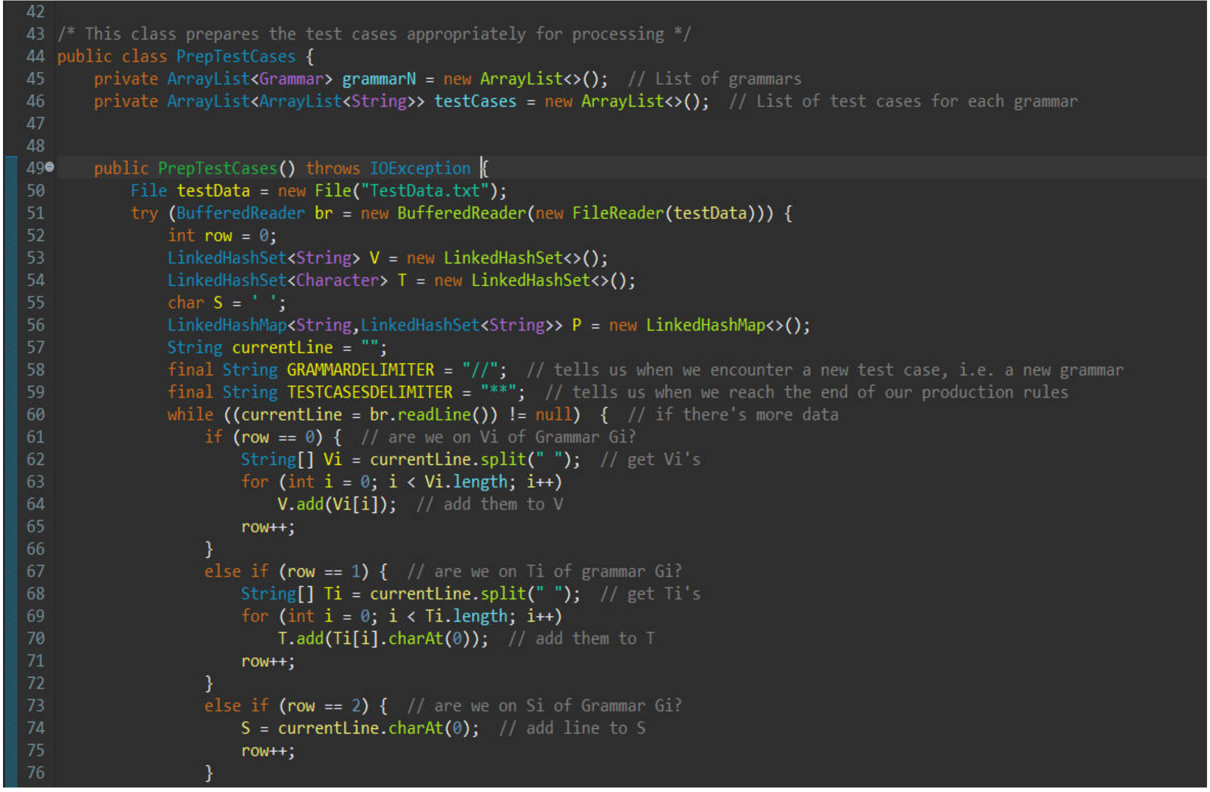
```
S A B
a b
S
S aAB
A aAB b
B b
**
abb
bbs
fwsf
ge
sg
//
S R L C A Y P Z X T D
{ } x = y + z i t e d
S
S {LR
R }
L xAYPZ iXTLC
C d eLD
A =
Y y
P +
Z z
X x
T t
D d
**
{ixtx=y+zd}
{ixtx=y+zeixtx=y+zdd}
{x=y+z}
{xf}
{xz}
{ix}
{it}
{ied}
{iex=}
//
```

Note: Please reference the file “*testData.txt*” for all 7 Grammars along with their test cases – 4 of which are valid *s-grammars*, 3 of which are not.

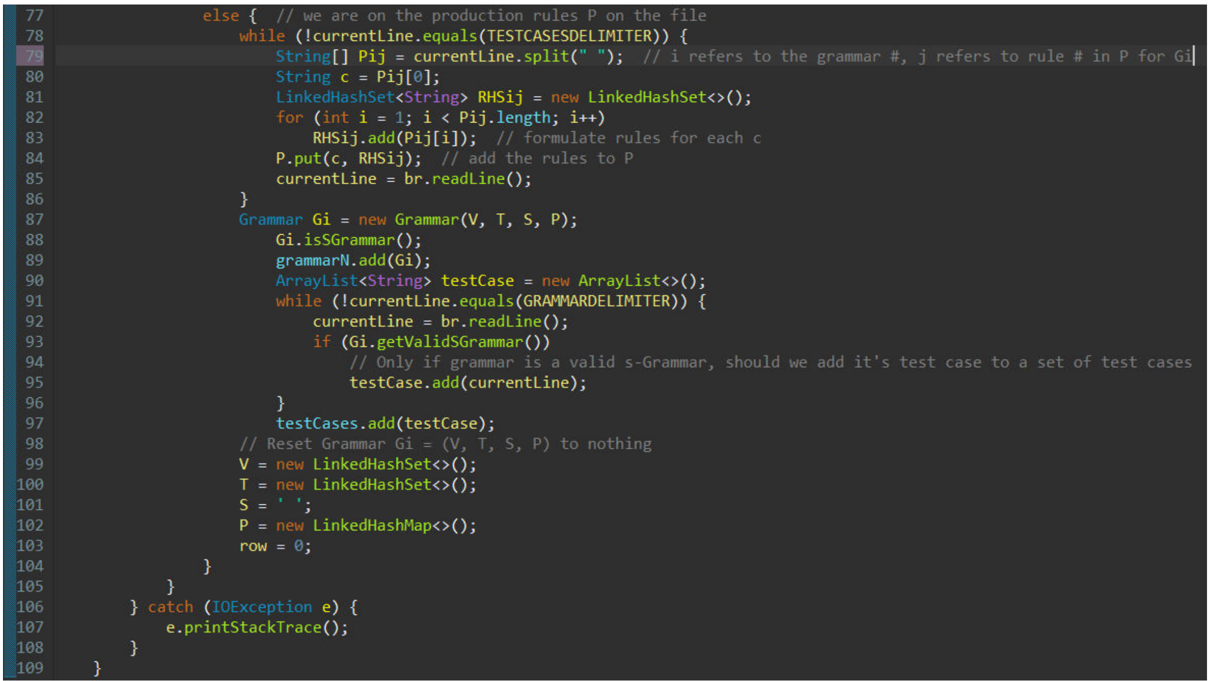
1. PrepTestCases.java



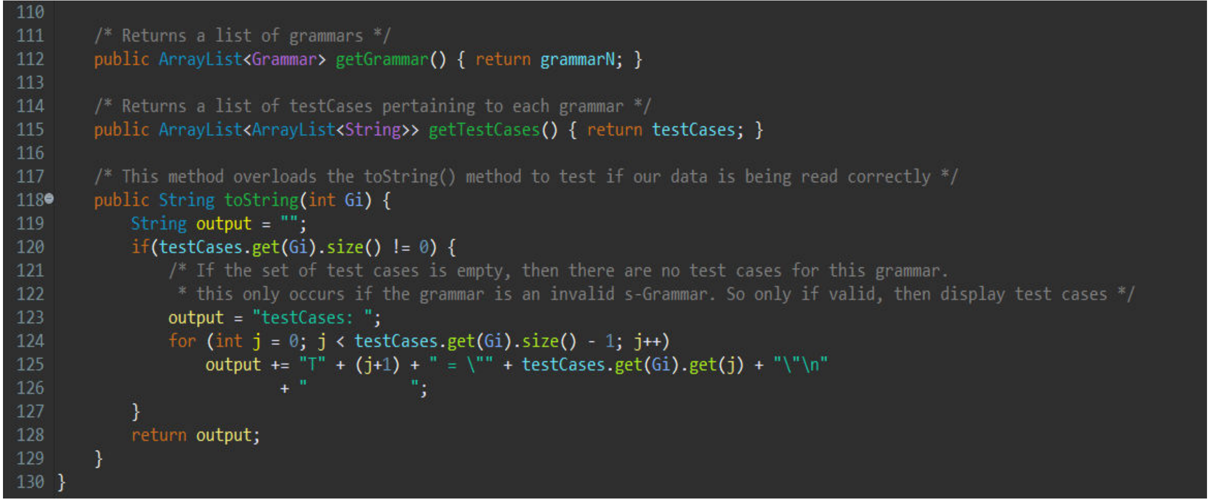
```java
package assignment2;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

/*
 * This class uses the Software Design Principle known as SRP - Single Responsibility Principle
 * i.e. One class will have one responsibility, for e.g., reading from a file.
 * This helps to keep our programs neater, smaller, & maintainable (easier to debug, etc.).
 */

import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.LinkedHashSet;

/* Format of the text file "TestData.txt":
 * The first line contains V,
 * The second line contains T,
 * The third line contains S,
 * The lines after that contains P,
 * After P, the next line will contain a "**"
 * This "**" concludes the production rules and introduces the
 * test cases for the grammar defined by G = (V, T, S, P)
 * After the test cases for the grammar G, the next line will
 * contain a grammarDelimiter - i.e. "//" which depicts the
 * start of a new grammar definition.
 */

/*  grammarN: ArrayList<Grammar>
 *  {  G1, G2, ..., Gn  }
 */

/*  testCases: ArrayList<ArrayList<String>
 *  G1: {  T1A, T1B, T1C,
 *  G2:    T2A, T2B, T2C, T2D, T2E,
 *  G3:    T3A, T3B, T3C, T3D,
 *  ...
 *  Gm:    TmA, TmB, TmC, ..., Tmn  }
 */
```



```java

/* This class prepares the test cases appropriately for processing */
public class PrepTestCases {
    private ArrayList<Grammar> grammarN = new ArrayList<>();  // List of grammars
    private ArrayList<ArrayList<String>> testCases = new ArrayList<>();  // List of test cases for each grammar


    public PrepTestCases() throws IOException {
        File testData = new File("TestData.txt");
        try (BufferedReader br = new BufferedReader(new FileReader(testData))) {
            int row = 0;
            LinkedHashSet<String> V = new LinkedHashSet<>();
            LinkedHashSet<Character> T = new LinkedHashSet<>();
            char S = ' ';
            LinkedHashMap<String,LinkedHashSet<String>> P = new LinkedHashMap<>();
            String currentLine = "";
            final String GRAMMARDELIMITER = "//";  // tells us when we encounter a new test case, i.e. a new grammar
            final String TESTCASESDELIMITER = "**";  // tells us when we reach the end of our production rules
            while ((currentLine = br.readLine()) != null)  {  // if there's more data
                if (row == 0) {  // are we on Vi of Grammar Gi?
                    String[] Vi = currentLine.split(" ");  // get Vi's
                    for (int i = 0; i < Vi.length; i++)
                        V.add(Vi[i]);  // add them to V
                    row++;
                }
                else if (row == 1) {  // are we on Ti of grammar Gi?
                    String[] Ti = currentLine.split(" ");  // get Ti's
                    for (int i = 0; i < Ti.length; i++)
                        T.add(Ti[i].charAt(0));  // add them to T
                    row++;
                }
                else if (row == 2) {  // are we on Si of Grammar Gi?
                    S = currentLine.charAt(0);  // add line to S
                    row++;
                }
```

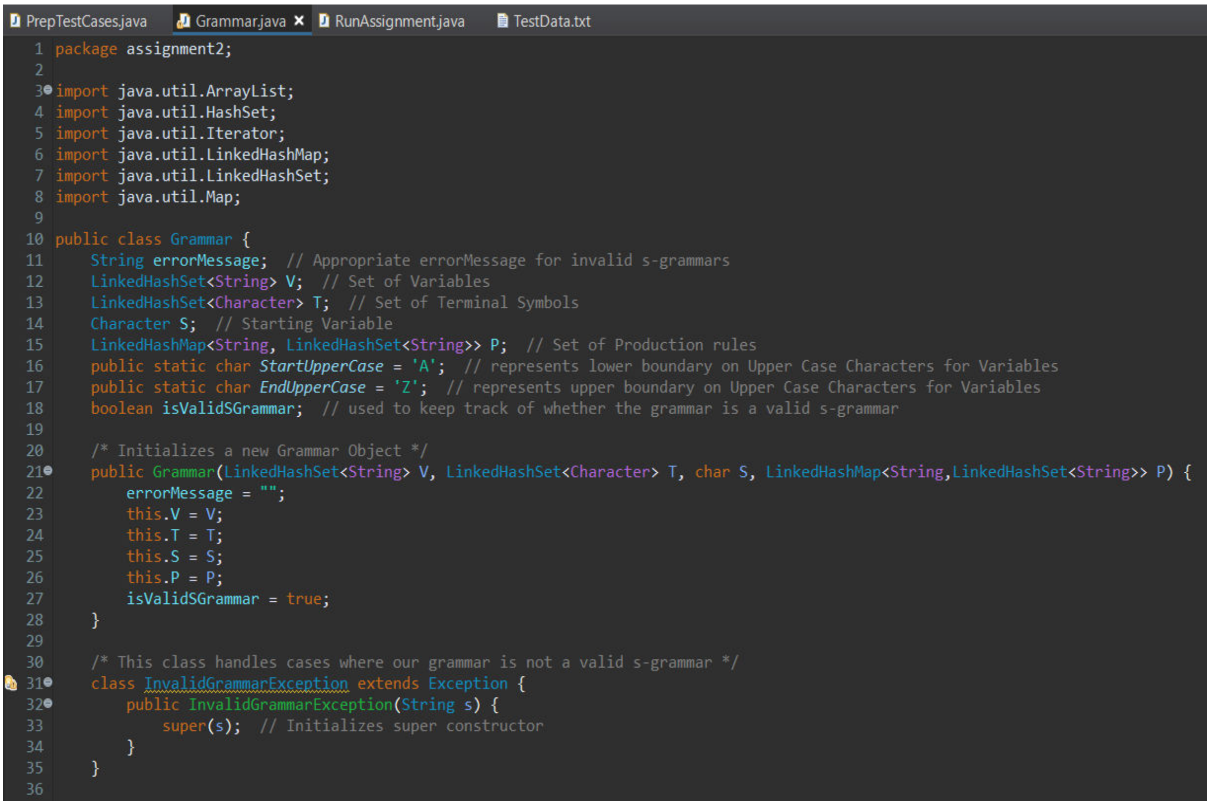


```java
                else {  // we are on the production rules P on the file
                    while (!currentLine.equals(TESTCASESDELIMITER)) {
                        String[] Pij = currentLine.split(" ");  // i refers to the grammar #, j refers to rule # in P for Gi
                        String c = Pij[0];
                        LinkedHashSet<String> RHSij = new LinkedHashSet<>();
                        for (int i = 1; i < Pij.length; i++)
                            RHSij.add(Pij[i]);  // formulate rules for each c
                        P.put(c, RHSij);  // add the rules to P
                        currentLine = br.readLine();
                    }
                    Grammar Gi = new Grammar(V, T, S, P);
                        Gi.isSGrammar();
                        grammarN.add(Gi);
                        ArrayList<String> testCase = new ArrayList<>();
                        while (!currentLine.equals(GRAMMARDELIMITER)) {
                            currentLine = br.readLine();
                            if (Gi.getValidSGrammar())
                                // Only if grammar is a valid s-Grammar, should we add it's test case to a set of test cases
                                testCase.add(currentLine);
                        }
                        testCases.add(testCase);
                    // Reset Grammar Gi = (V, T, S, P) to nothing
                    V = new LinkedHashSet<>();
                    T = new LinkedHashSet<>();
                    S = ' ';
                    P = new LinkedHashMap<>();
                    row = 0;
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

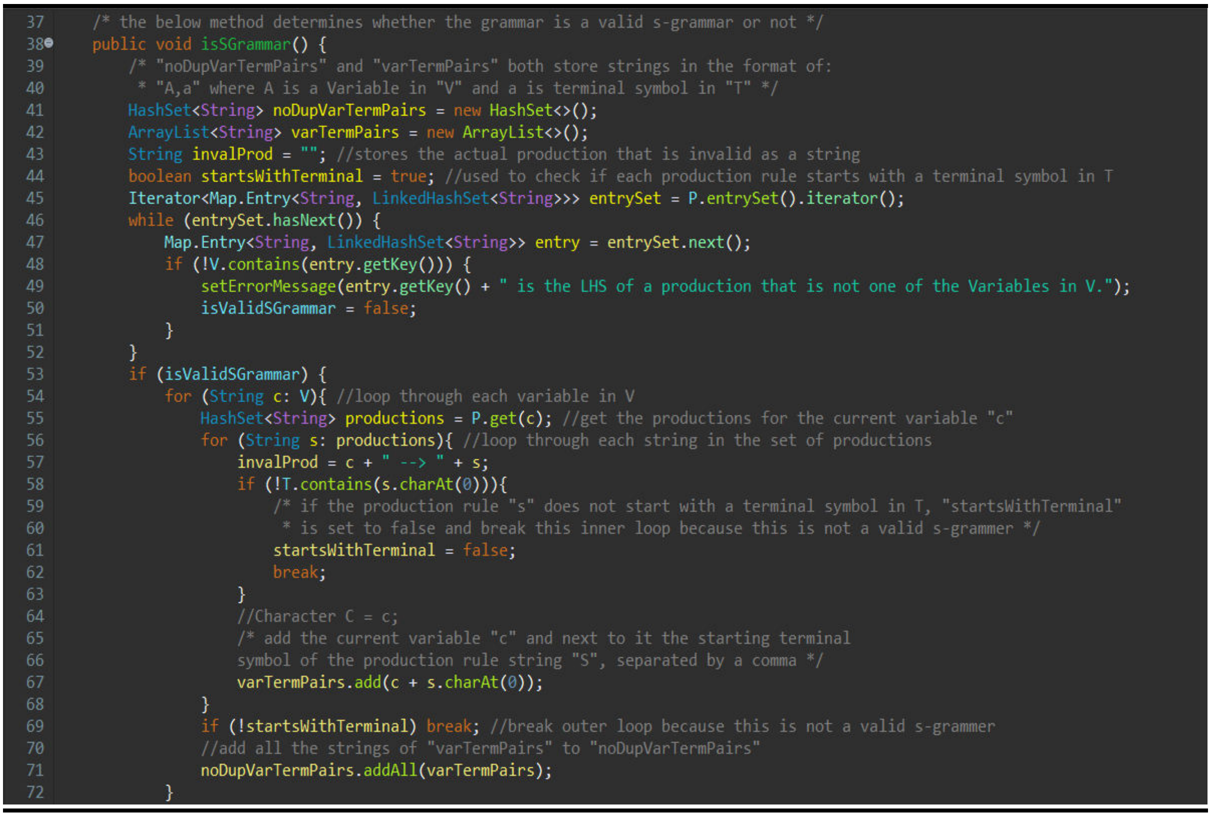


```java

    /* Returns a list of grammars */
    public ArrayList<Grammar> getGrammar() { return grammarN; }

    /* Returns a list of testCases pertaining to each grammar */
    public ArrayList<ArrayList<String>> getTestCases() { return testCases; }

    /* This method overloads the toString() method to test if our data is being read correctly */
    public String toString(int Gi) {
        String output = "";
        if(testCases.get(Gi).size() != 0) {
            /* If the set of test cases is empty, then there are no test cases for this grammar.
             * this only occurs if the grammar is an invalid s-Grammar. So only if valid, then display test cases */
            output = "testCases: ";
            for (int j = 0; j < testCases.get(Gi).size() - 1; j++)
                output += "T" + (j+1) + " = \"" + testCases.get(Gi).get(j) + "\"\n"
                        + "           ";
        }
        return output;
    }
}
```

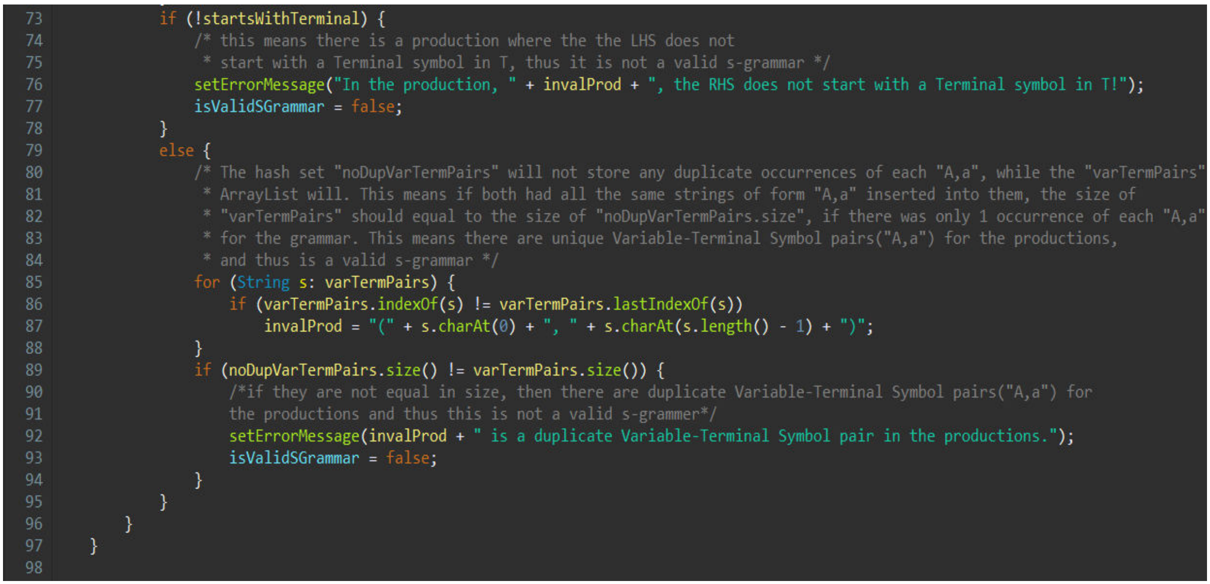
2. Grammar.java



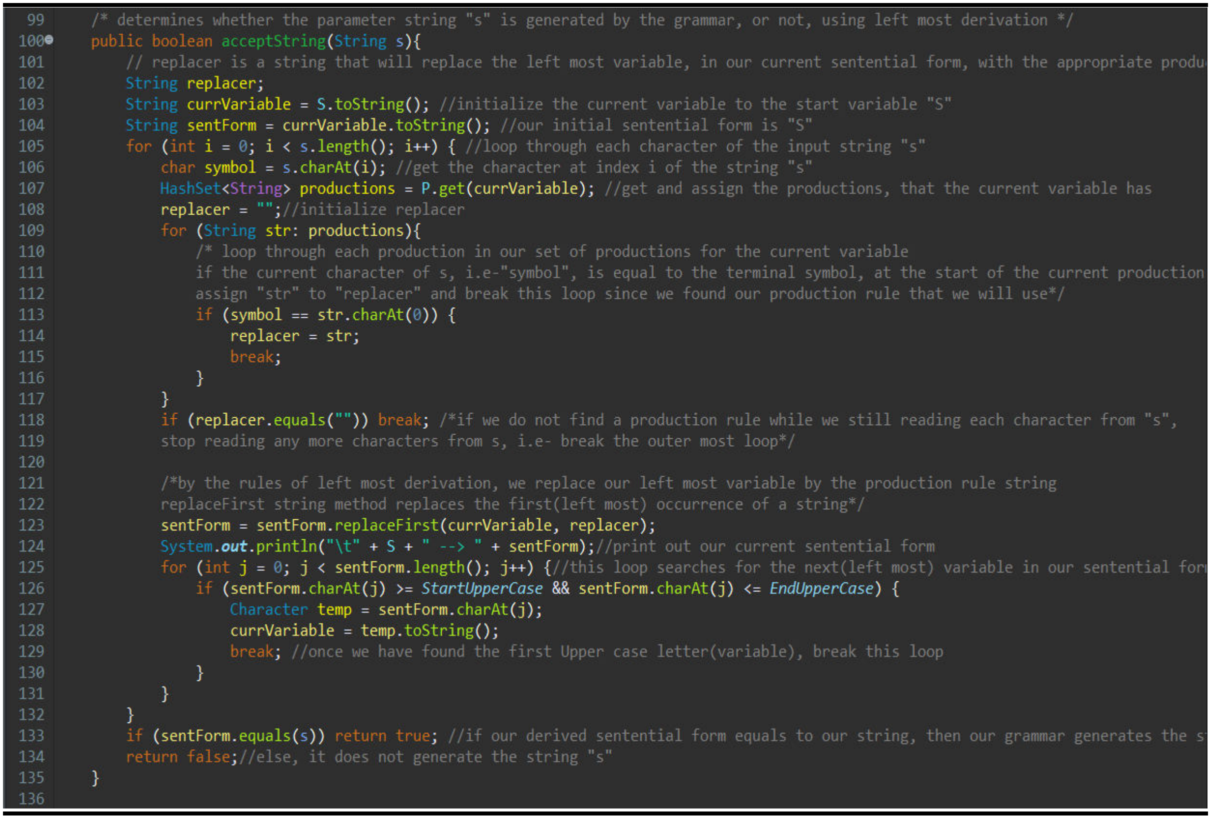
```java
package assignment2;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.LinkedHashSet;
import java.util.Map;

public class Grammar {
    String errorMessage;  // Appropriate errorMessage for invalid s-grammars
    LinkedHashSet<String> V;  // Set of Variables
    LinkedHashSet<Character> T;  // Set of Terminal Symbols
    Character S;  // Starting Variable
    LinkedHashMap<String, LinkedHashSet<String>> P;  // Set of Production rules
    public static char StartUpperCase = 'A';  // represents lower boundary on Upper Case Characters for Variables
    public static char EndUpperCase = 'Z';  // represents upper boundary on Upper Case Characters for Variables
    boolean isValidSGrammar;  // used to keep track of whether the grammar is a valid s-grammar

    /* Initializes a new Grammar Object */
    public Grammar(LinkedHashSet<String> V, LinkedHashSet<Character> T, char S, LinkedHashMap<String,LinkedHashSet<String>> P) {
        errorMessage = "";
        this.V = V;
        this.T = T;
        this.S = S;
        this.P = P;
        isValidSGrammar = true;
    }

    /* This class handles cases where our grammar is not a valid s-grammar */
    class InvalidGrammarException extends Exception {
        public InvalidGrammarException(String s) {
            super(s);  // Initializes super constructor
        }
    }

```



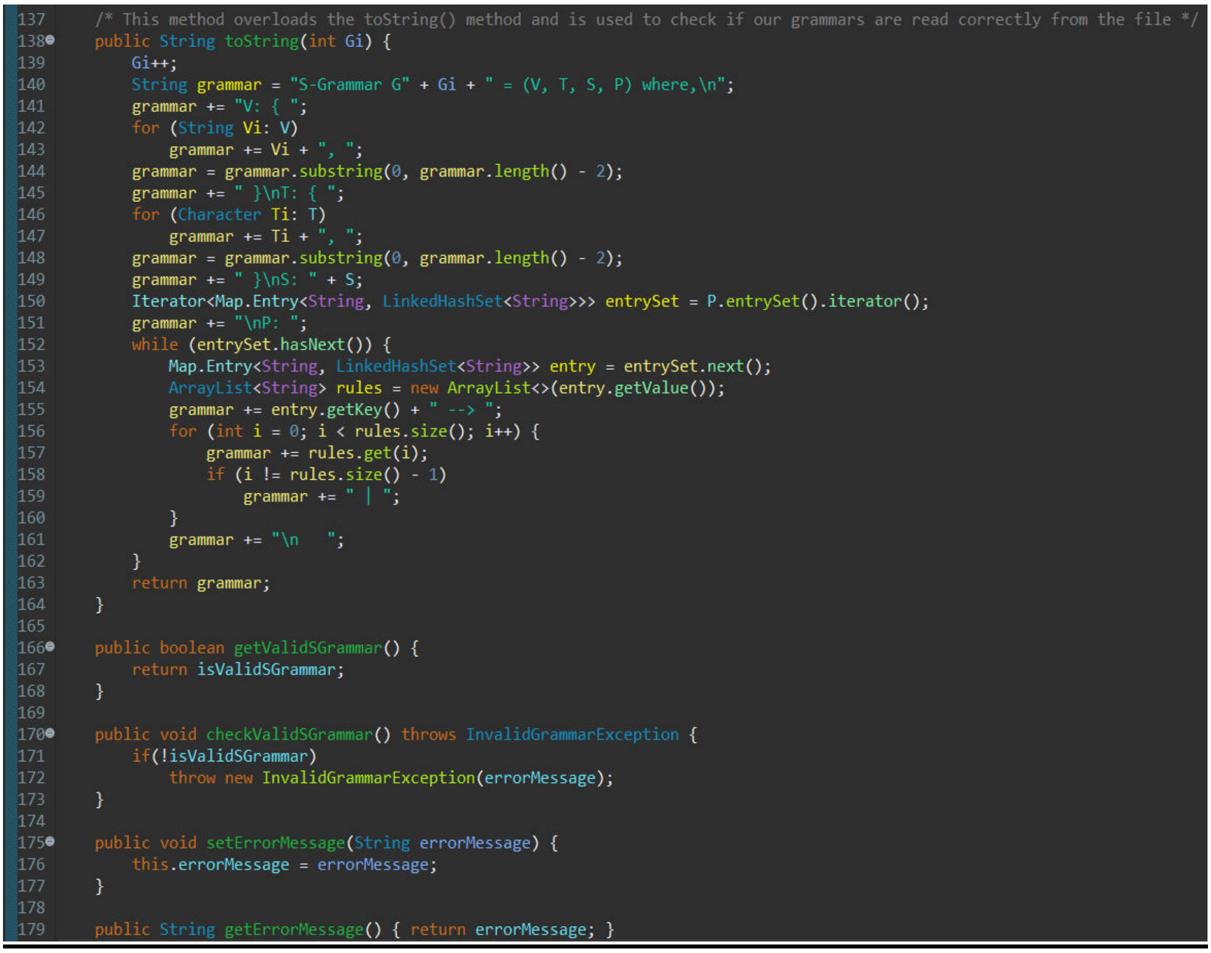
```java
    /* the below method determines whether the grammar is a valid s-grammar or not */
    public void isSGrammar() {
        /* "noDupVarTermPairs" and "varTermPairs" both store strings in the format of:
         * "A,a" where A is a Variable in "V" and a is terminal symbol in "T" */
        HashSet<String> noDupVarTermPairs = new HashSet<>();
        ArrayList<String> varTermPairs = new ArrayList<>();
        String invalProd = ""; //stores the actual production that is invalid as a string
        boolean startsWithTerminal = true; //used to check if each production rule starts with a terminal symbol in T
        Iterator<Map.Entry<String, LinkedHashSet<String>>> entrySet = P.entrySet().iterator();
        while (entrySet.hasNext()) {
            Map.Entry<String, LinkedHashSet<String>> entry = entrySet.next();
            if (!V.contains(entry.getKey())) {
                setErrorMessage(entry.getKey() + " is the LHS of a production that is not one of the Variables in V.");
                isValidSGrammar = false;
            }
        }
        if (isValidSGrammar) {
            for (String c: V){ //loop through each variable in V
                HashSet<String> productions = P.get(c); //get the productions for the current variable "c"
                for (String s: productions){ //loop through each string in the set of productions
                    invalProd = c + " --> " + s;
                    if (!T.contains(s.charAt(0))){
                        /* if the production rule "s" does not start with a terminal symbol in T, "startsWithTerminal"
                         * is set to false and break this inner loop because this is not a valid s-grammer */
                        startsWithTerminal = false;
                        break;
                    }
                    //Character C = c;
                    /* add the current variable "c" and next to it the starting terminal
                    symbol of the production rule string "S", separated by a comma */
                    varTermPairs.add(c + s.charAt(0));
                }
                if (!startsWithTerminal) break; //break outer loop because this is not a valid s-grammer
                //add all the strings of "varTermPairs" to "noDupVarTermPairs"
                noDupVarTermPairs.addAll(varTermPairs);
            }
```



```java
            if (!startsWithTerminal) {
                /* this means there is a production where the the LHS does not
                 * start with a Terminal symbol in T, thus it is not a valid s-grammar */
                setErrorMessage("In the production, " + invalProd + ", the RHS does not start with a Terminal symbol in T!");
                isValidSGrammar = false;
            }
            else {
                /* The hash set "noDupVarTermPairs" will not store any duplicate occurrences of each "A,a", while the "varTermPairs"
                 * ArrayList will. This means if both had all the same strings of form "A,a" inserted into them, the size of
                 * "varTermPairs" should equal to the size of "noDupVarTermPairs.size", if there was only 1 occurrence of each "A,a"
                 * for the grammar. This means there are unique Variable-Terminal Symbol pairs("A,a") for the productions,
                 * and thus is a valid s-grammar */
                for (String s: varTermPairs) {
                    if (varTermPairs.indexOf(s) != varTermPairs.lastIndexOf(s))
                        invalProd = "(" + s.charAt(0) + ", " + s.charAt(s.length() - 1) + ")";
                }
                if (noDupVarTermPairs.size() != varTermPairs.size()) {
                    /*if they are not equal in size, then there are duplicate Variable-Terminal Symbol pairs("A,a") for
                    the productions and thus this is not a valid s-grammer*/
                    setErrorMessage(invalProd + " is a duplicate Variable-Terminal Symbol pair in the productions.");
                    isValidSGrammar = false;
                }
            }
        }
    }

```

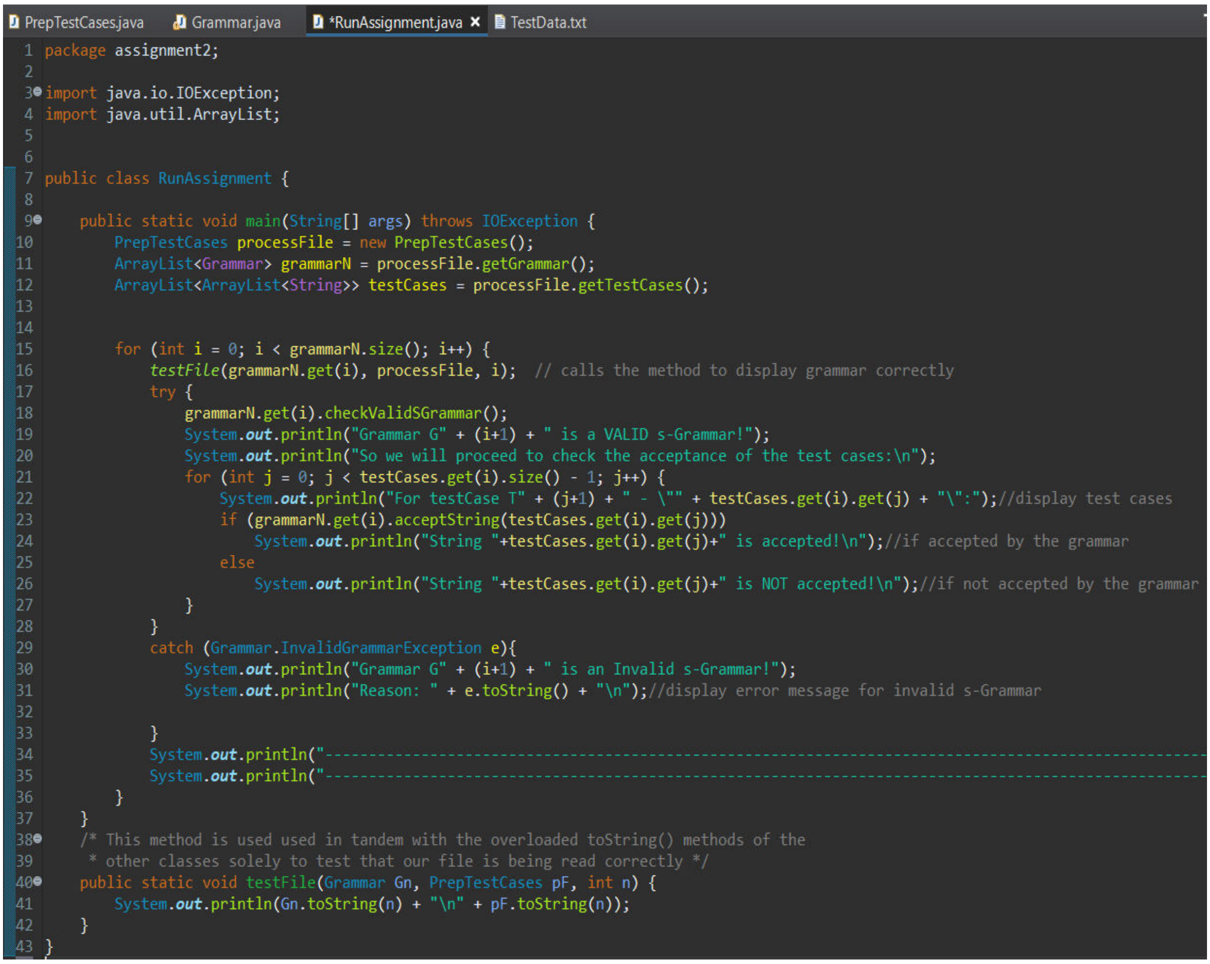


```java
    /* determines whether the parameter string "s" is generated by the grammar, or not, using left most derivation */
    public boolean acceptString(String s){
        // replacer is a string that will replace the left most variable, in our current sentential form, with the appropriate produ
        String replacer;
        String currVariable = S.toString(); //initialize the current variable to the start variable "S"
        String sentForm = currVariable.toString(); //our initial sentential form is "S"
        for (int i = 0; i < s.length(); i++) { //loop through each character of the input string "s"
            char symbol = s.charAt(i); //get the character at index i of the string "s"
            HashSet<String> productions = P.get(currVariable); //get and assign the productions, that the current variable has
            replacer = "";//initialize replacer
            for (String str: productions){
                /* loop through each production in our set of productions for the current variable
                if the current character of s, i.e-"symbol", is equal to the terminal symbol, at the start of the current production
                assign "str" to "replacer" and break this loop since we found our production rule that we will use*/
                if (symbol == str.charAt(0)) {
                    replacer = str;
                    break;
                }
            }
            if (replacer.equals("")) break; /*if we do not find a production rule while we still reading each character from "s",
            stop reading any more characters from s, i.e- break the outer most loop*/

            /*by the rules of left most derivation, we replace our left most variable by the production rule string
            replaceFirst string method replaces the first(left most) occurrence of a string*/
            sentForm = sentForm.replaceFirst(currVariable, replacer);
            System.out.println("\t" + S + " --> " + sentForm);//print out our current sentential form
            for (int j = 0; j < sentForm.length(); j++) {//this loop searches for the next(left most) variable in our sentential for
                if (sentForm.charAt(j) >= StartUpperCase && sentForm.charAt(j) <= EndUpperCase) {
                    Character temp = sentForm.charAt(j);
                    currVariable = temp.toString();
                    break; //once we have found the first Upper case letter(variable), break this loop
                }
            }
        }
        if (sentForm.equals(s)) return true; //if our derived sentential form equals to our string, then our grammar generates the s
        return false;//else, it does not generate the string "s"
    }

```



```java
    /* This method overloads the toString() method and is used to check if our grammars are read correctly from the file */
    public String toString(int Gi) {
        Gi++;
        String grammar = "S-Grammar G" + Gi + " = (V, T, S, P) where,\n";
        grammar += "V: { ";
        for (String Vi: V)
            grammar += Vi + ", ";
        grammar = grammar.substring(0, grammar.length() - 2);
        grammar += " }\nT: { ";
        for (Character Ti: T)
            grammar += Ti + ", ";
        grammar = grammar.substring(0, grammar.length() - 2);
        grammar += " }\nS: " + S;
        Iterator<Map.Entry<String, LinkedHashSet<String>>> entrySet = P.entrySet().iterator();
        grammar += "\nP: ";
        while (entrySet.hasNext()) {
            Map.Entry<String, LinkedHashSet<String>> entry = entrySet.next();
            ArrayList<String> rules = new ArrayList<>(entry.getValue());
            grammar += entry.getKey() + " --> ";
            for (int i = 0; i < rules.size(); i++) {
                grammar += rules.get(i);
                if (i != rules.size() - 1)
                    grammar += " | ";
            }
            grammar += "\n   ";
        }
        return grammar;
    }

    public boolean getValidSGrammar() {
        return isValidSGrammar;
    }

    public void checkValidSGrammar() throws InvalidGrammarException {
        if(!isValidSGrammar)
            throw new InvalidGrammarException(errorMessage);
    }

    public void setErrorMessage(String errorMessage) {
        this.errorMessage = errorMessage;
    }

    public String getErrorMessage() { return errorMessage; }
```

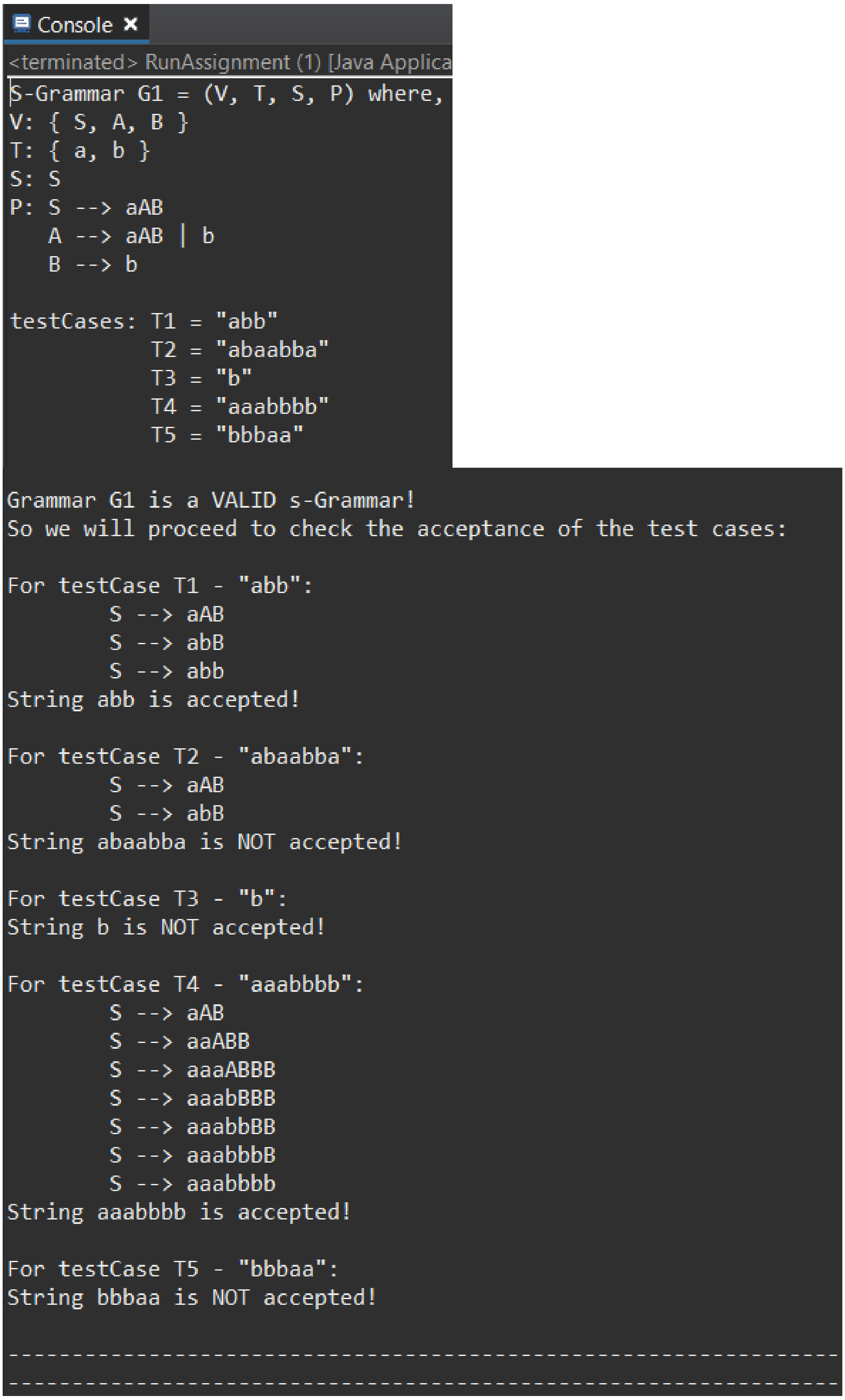
3. RunAssignment.java



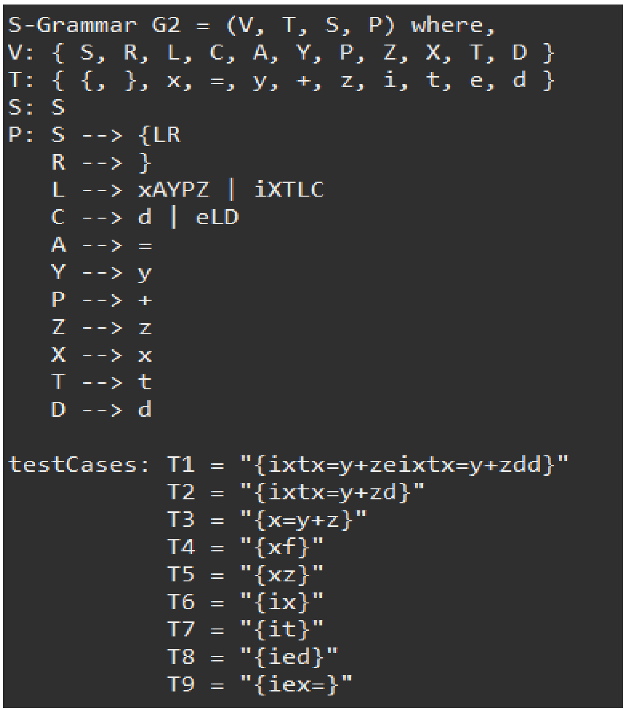
```java
package assignment2;

import java.io.IOException;
import java.util.ArrayList;


public class RunAssignment {

    public static void main(String[] args) throws IOException {
        PrepTestCases processFile = new PrepTestCases();
        ArrayList<Grammar> grammarN = processFile.getGrammar();
        ArrayList<ArrayList<String>> testCases = processFile.getTestCases();


        for (int i = 0; i < grammarN.size(); i++) {
            testFile(grammarN.get(i), processFile, i);  // calls the method to display grammar correctly
            try {
                grammarN.get(i).checkValidSGrammar();
                System.out.println("Grammar G" + (i+1) + " is a VALID s-Grammar!");
                System.out.println("So we will proceed to check the acceptance of the test cases:\n");
                for (int j = 0; j < testCases.get(i).size() - 1; j++) {
                    System.out.println("For testCase T" + (j+1) + " - \"" + testCases.get(i).get(j) + "\":");//display test cases
                    if (grammarN.get(i).acceptString(testCases.get(i).get(j)))
                        System.out.println("String "+testCases.get(i).get(j)+" is accepted!\n");//if accepted by the grammar
                    else
                        System.out.println("String "+testCases.get(i).get(j)+" is NOT accepted!\n");//if not accepted by the grammar
                }
            }
            catch (Grammar.InvalidGrammarException e){
                System.out.println("Grammar G" + (i+1) + " is an Invalid s-Grammar!");
                System.out.println("Reason: " + e.toString() + "\n");//display error message for invalid s-Grammar

            }
            System.out.println("----------------------------------------------------------------------------------------------------
            System.out.println("----------------------------------------------------------------------------------------------------
        }
    }
    /* This method is used used in tandem with the overloaded toString() methods of the
     * other classes solely to test that our file is being read correctly */
    public static void testFile(Grammar Gn, PrepTestCases pF, int n) {
        System.out.println(Gn.toString(n) + "\n" + pF.toString(n));
    }
}
```

Testing Record depicting Code Correctness:

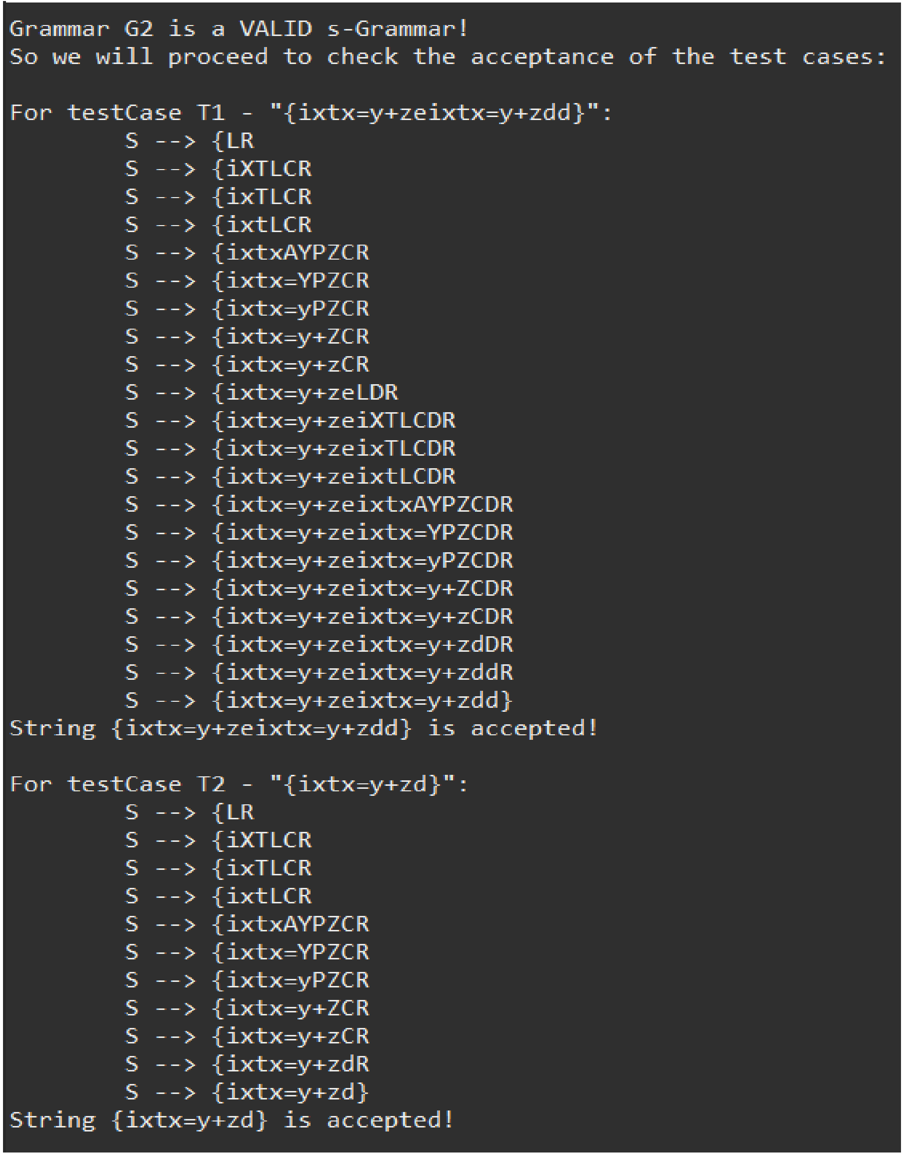
1. The output of the complete program for valid *s-grammars* is:



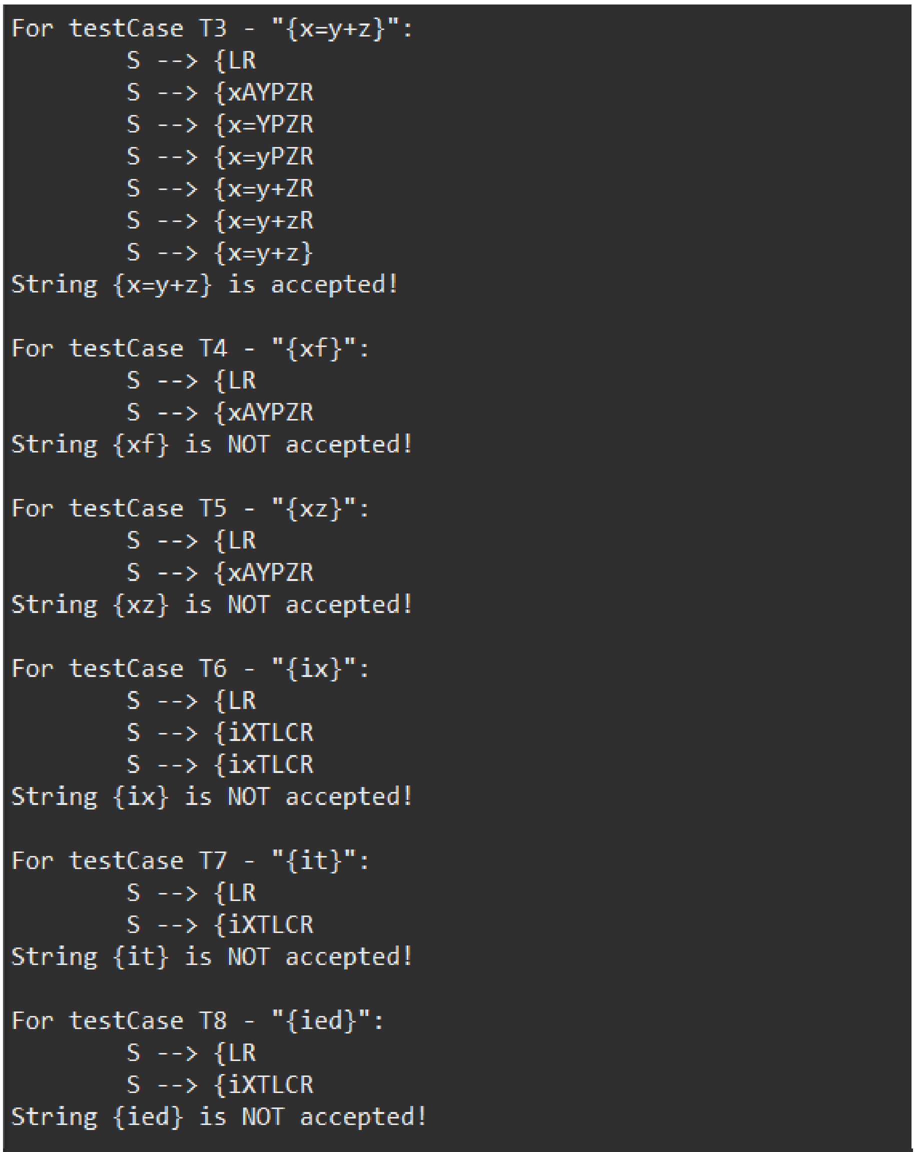
```
S-Grammar G1 = (V, T, S, P) where,
V: { S, A, B }
T: { a, b }
S: S
P: S --> aAB
   A --> aAB | b
   B --> b

testCases: T1 = "abb"
           T2 = "abaabba"
           T3 = "b"
           T4 = "aaabbbb"
           T5 = "bbbaa"

Grammar G1 is a VALID s-Grammar!
So we will proceed to check the acceptance of the test cases:

For testCase T1 - "abb":
        S --> aAB
        S --> abB
        S --> abb
String abb is accepted!

For testCase T2 - "abaabba":
        S --> aAB
        S --> abB
String abaabba is NOT accepted!

For testCase T3 - "b":
String b is NOT accepted!

For testCase T4 - "aaabbbb":
        S --> aAB
        S --> aaABB
        S --> aaaABBB
        S --> aaabBBB
        S --> aaabbBB
        S --> aaabbbB
        S --> aaabbbb
String aaabbbb is accepted!

For testCase T5 - "bbbaa":
String bbbaa is NOT accepted!

-----------------------------------------------------------------
-----------------------------------------------------------------
```



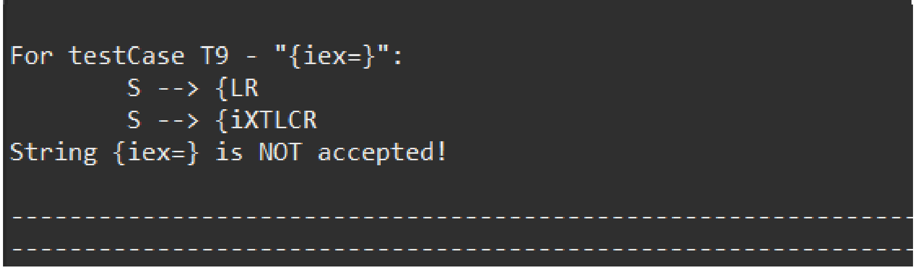
```
S-Grammar G2 = (V, T, S, P) where,
V: { S, R, L, C, A, Y, P, Z, X, T, D }
T: { {, }, x, =, y, +, z, i, t, e, d }
S: S
P: S --> {LR
   R --> }
   L --> xAYPZ | iXTLC
   C --> d | eLD
   A --> =
   Y --> y
   P --> +
   Z --> z
   X --> x
   T --> t
   D --> d

testCases: T1 = "{ixtx=y+zeixtx=y+zdd}"
           T2 = "{ixtx=y+zd}"
           T3 = "{x=y+z}"
           T4 = "{xf}"
           T5 = "{xz}"
           T6 = "{ix}"
           T7 = "{it}"
           T8 = "{ied}"
           T9 = "{iex=}"
```



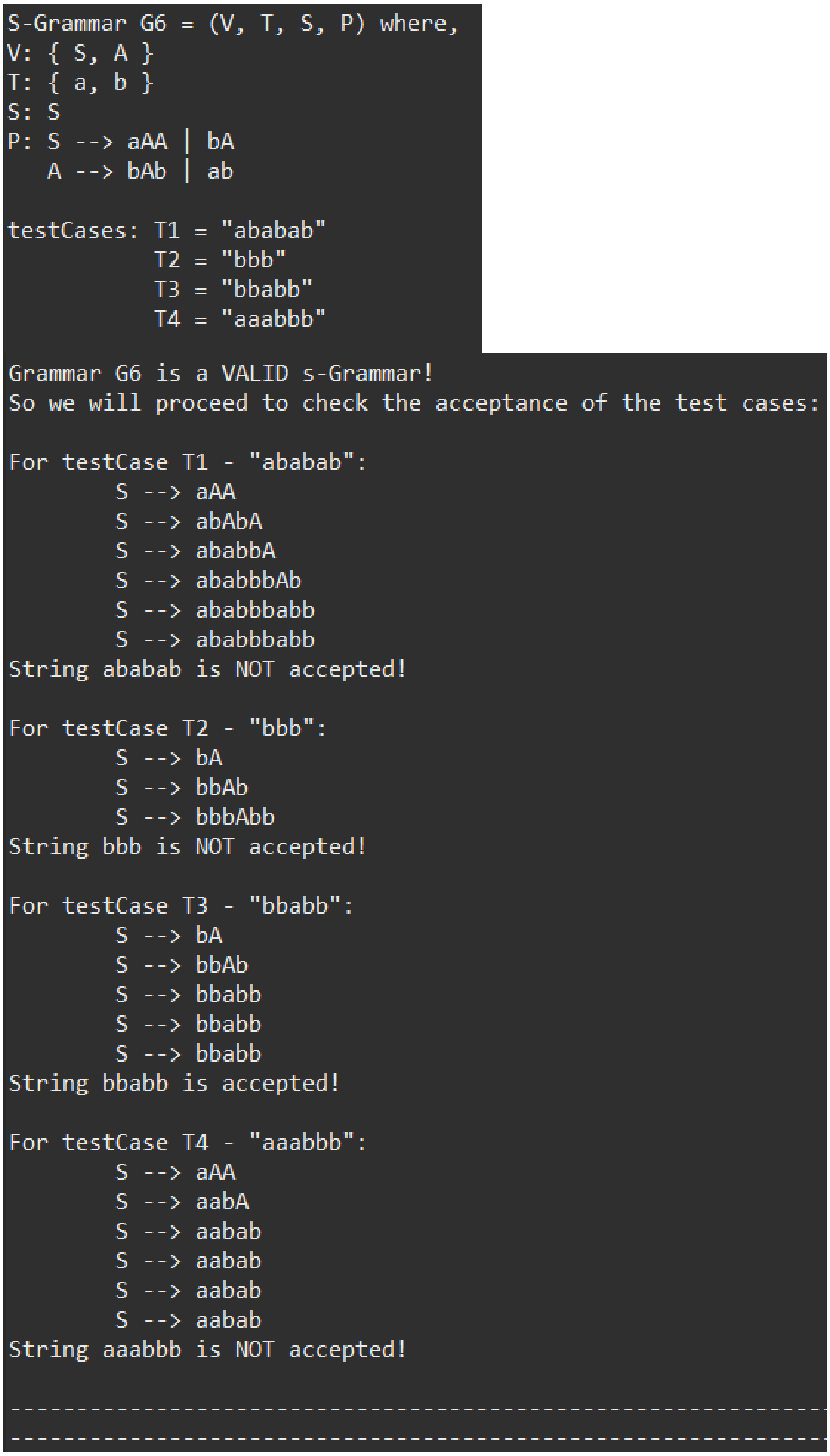
```
Grammar G2 is a VALID s-Grammar!
So we will proceed to check the acceptance of the test cases:

For testCase T1 - "{ixtx=y+zeixtx=y+zdd}":
        S --> {LR
        S --> {iXTLCR
        S --> {ixTLCR
        S --> {ixtLCR
        S --> {ixtxAYPZCR
        S --> {ixtx=YPZCR
        S --> {ixtx=yPZCR
        S --> {ixtx=y+ZCR
        S --> {ixtx=y+zCR
        S --> {ixtx=y+zeLDR
        S --> {ixtx=y+zeiXTLCDR
        S --> {ixtx=y+zeixTLCDR
        S --> {ixtx=y+zeixtLCDR
        S --> {ixtx=y+zeixtxAYPZCDR
        S --> {ixtx=y+zeixtx=YPZCDR
        S --> {ixtx=y+zeixtx=yPZCDR
        S --> {ixtx=y+zeixtx=y+ZCDR
        S --> {ixtx=y+zeixtx=y+zCDR
        S --> {ixtx=y+zeixtx=y+zdDR
        S --> {ixtx=y+zeixtx=y+zddR
        S --> {ixtx=y+zeixtx=y+zdd}
String {ixtx=y+zeixtx=y+zdd} is accepted!

For testCase T2 - "{ixtx=y+zd}":
        S --> {LR
        S --> {iXTLCR
        S --> {ixTLCR
        S --> {ixtLCR
        S --> {ixtxAYPZCR
        S --> {ixtx=YPZCR
        S --> {ixtx=yPZCR
        S --> {ixtx=y+ZCR
        S --> {ixtx=y+zCR
        S --> {ixtx=y+zdR
        S --> {ixtx=y+zd}
String {ixtx=y+zd} is accepted!
```



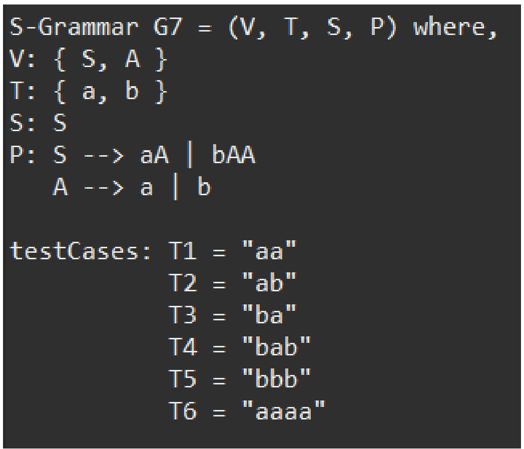
```
For testCase T3 - "{x=y+z}":
        S --> {LR
        S --> {xAYPZR
        S --> {x=YPZR
        S --> {x=yPZR
        S --> {x=y+ZR
        S --> {x=y+zR
        S --> {x=y+z}
String {x=y+z} is accepted!

For testCase T4 - "{xf}":
        S --> {LR
        S --> {xAYPZR
String {xf} is NOT accepted!

For testCase T5 - "{xz}":
        S --> {LR
        S --> {xAYPZR
String {xz} is NOT accepted!

For testCase T6 - "{ix}":
        S --> {LR
        S --> {iXTLCR
        S --> {ixTLCR
String {ix} is NOT accepted!

For testCase T7 - "{it}":
        S --> {LR
        S --> {iXTLCR
String {it} is NOT accepted!

For testCase T8 - "{ied}":
        S --> {LR
        S --> {iXTLCR
String {ied} is NOT accepted!
```



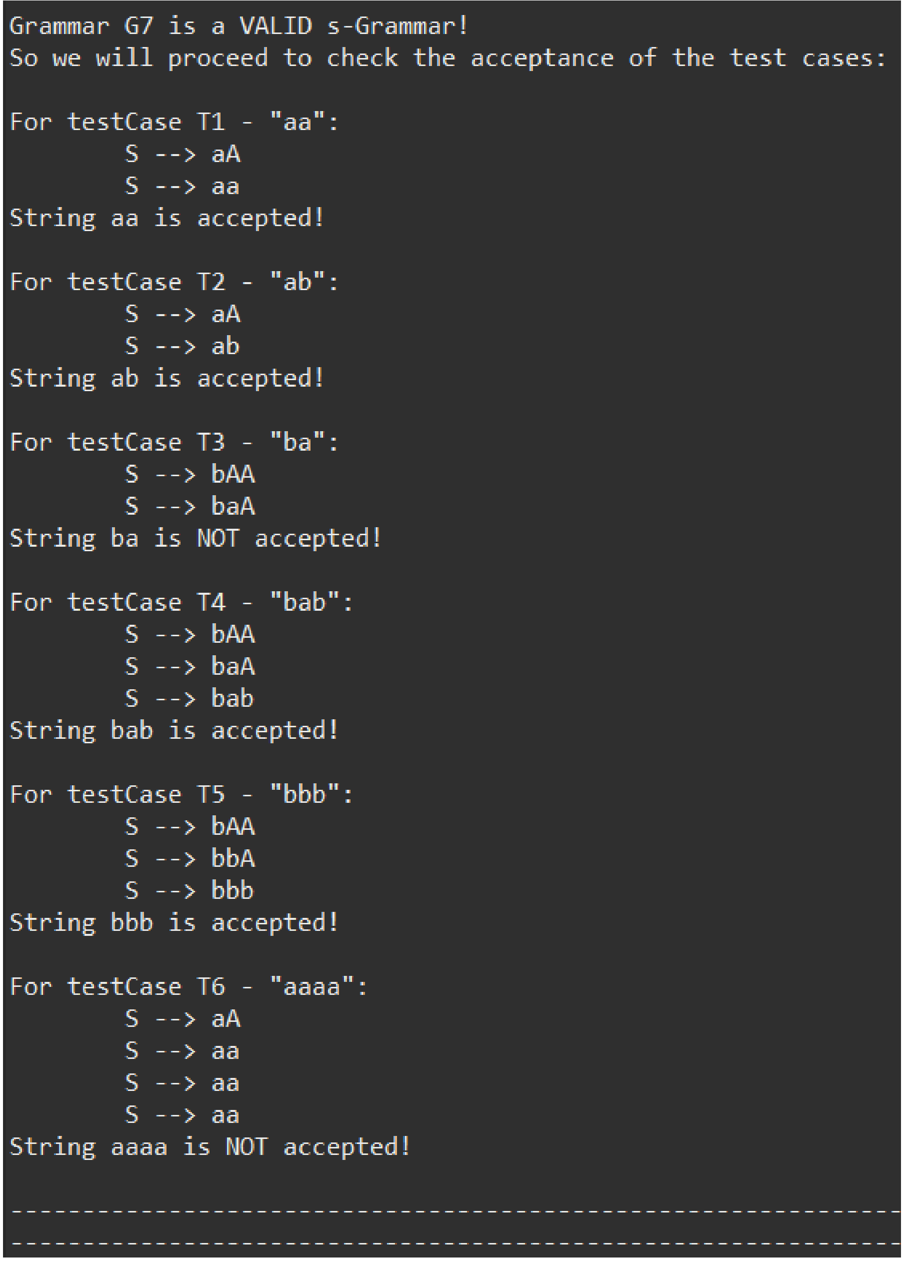
```
For testCase T9 - "{iex=}":
        S --> {LR
        S --> {iXTLCR
String {iex=} is NOT accepted!

---------------------------------------------------------------
---------------------------------------------------------------
```



```
S-Grammar G6 = (V, T, S, P) where,
V: { S, A }
T: { a, b }
S: S
P: S --> aAA | bA
   A --> bAb | ab

testCases: T1 = "ababab"
           T2 = "bbb"
           T3 = "bbabb"
           T4 = "aaabbb"

Grammar G6 is a VALID s-Grammar!
So we will proceed to check the acceptance of the test cases:

For testCase T1 - "ababab":
        S --> aAA
        S --> abAbA
        S --> ababbA
        S --> ababbbAb
        S --> ababbbabb
        S --> ababbbabb
String ababab is NOT accepted!

For testCase T2 - "bbb":
        S --> bA
        S --> bbAb
        S --> bbbAbb
String bbb is NOT accepted!

For testCase T3 - "bbabb":
        S --> bA
        S --> bbAb
        S --> bbabb
        S --> bbabb
        S --> bbabb
String bbabb is accepted!

For testCase T4 - "aaabbb":
        S --> aAA
        S --> aabA
        S --> aabab
        S --> aabab
        S --> aabab
        S --> aabab
String aaabbb is NOT accepted!

-------------------------------------------------------------
-------------------------------------------------------------
```



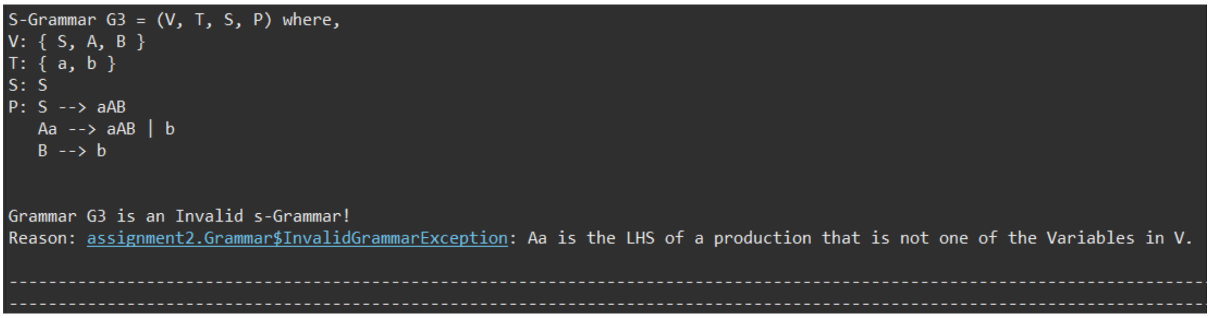
```
S-Grammar G7 = (V, T, S, P) where,
V: { S, A }
T: { a, b }
S: S
P: S --> aA | bAA
   A --> a | b

testCases: T1 = "aa"
           T2 = "ab"
           T3 = "ba"
           T4 = "bab"
           T5 = "bbb"
           T6 = "aaaa"
```



```
Grammar G7 is a VALID s-Grammar!
So we will proceed to check the acceptance of the test cases:

For testCase T1 - "aa":
        S --> aA
        S --> aa
String aa is accepted!

For testCase T2 - "ab":
        S --> aA
        S --> ab
String ab is accepted!

For testCase T3 - "ba":
        S --> bAA
        S --> baA
String ba is NOT accepted!

For testCase T4 - "bab":
        S --> bAA
        S --> baA
        S --> bab
String bab is accepted!

For testCase T5 - "bbb":
        S --> bAA
        S --> bbA
        S --> bbb
String bbb is accepted!

For testCase T6 - "aaaa":
        S --> aA
        S --> aa
        S --> aa
        S --> aa
String aaaa is NOT accepted!

--------------------------------------------------------------
--------------------------------------------------------------
```

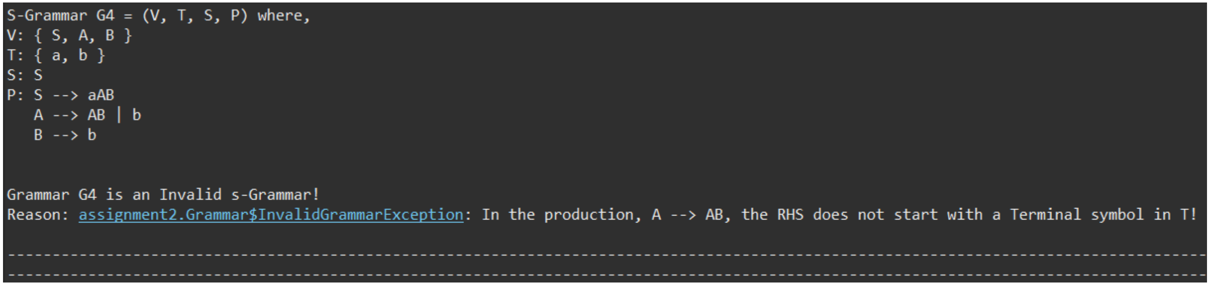
This proves that the program does the acceptance correctly for valid *s-grammars*.

2. <u>The output of the complete program for invalid *s-grammars*</u>:



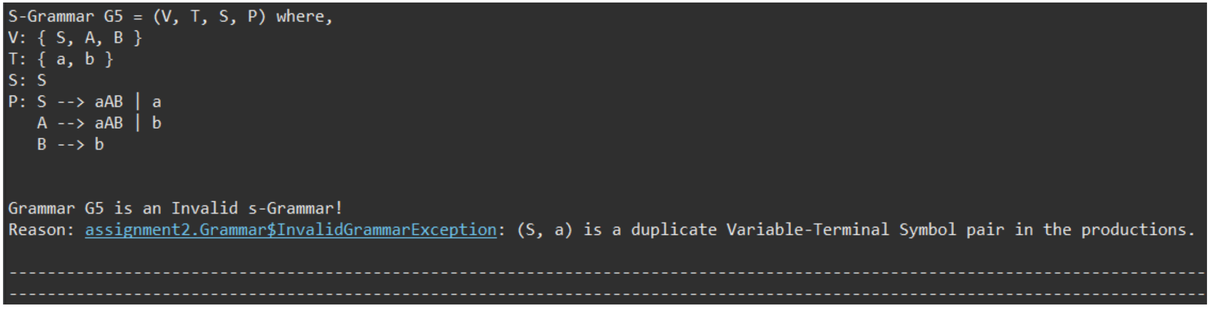
```
S-Grammar G3 = (V, T, S, P) where,
V: { S, A, B }
T: { a, b }
S: S
P: S --> aAB
   Aa --> aAB | b
   B --> b


Grammar G3 is an Invalid s-Grammar!
Reason: assignment2.Grammar$InvalidGrammarException: Aa is the LHS of a production that is not one of the Variables in V.

--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
```

This occurs when a grammar in the file, in this case, G3 is not a valid *s-grammar* due to *the existence of a production rule where the LHS is not one of the variables in V*.



```
S-Grammar G4 = (V, T, S, P) where,
V: { S, A, B }
T: { a, b }
S: S
P: S --> aAB
   A --> AB | b
   B --> b


Grammar G4 is an Invalid s-Grammar!
Reason: assignment2.Grammar$InvalidGrammarException: In the production, A --> AB, the RHS does not start with a Terminal symbol in T!

--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
```

This occurs when a grammar in the file, in this case, G4 is not a valid *s-grammar* due to *the existence of a production rule where the LHS does not start with a Terminal symbol in T*.



```
S-Grammar G5 = (V, T, S, P) where,
V: { S, A, B }
T: { a, b }
S: S
P: S --> aAB | a
   A --> aAB | b
   B --> b


Grammar G5 is an Invalid s-Grammar!
Reason: assignment2.Grammar$InvalidGrammarException: (S, a) is a duplicate Variable-Terminal Symbol pair in the productions.

--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
```

This occurs when a grammar in the file, in this case, G5 is not a valid *s-grammar* due to *duplicate Variable-Terminal Symbol pairs in productions*.

This proves that the program successfully catches our custom-made *InvalidGrammarException* and also prints a meaningful message when encountering invalid *s-grammars* and does not terminate the program until the entire file is processed – as required.

<u>Contributions:</u>

Before anything, I must say that I am very pleased with the work that my group has produced. Not only am I extremely proud of my group members performance, but I am also grateful that I had the opportunity to work with these guys. Without their hard work, dedication, and overall superb team effort, we would not have accomplished *perfection.*

NB: As Group Leader of this project, I had my hands in all parts of this project.

Classes Written:

1. Hakeem: *PrepTestCases.java*, *RunAssignment.java*, *Grammar.java*: *toString()*
2. Ricardo: *Grammar.java*

Bug fixes:

1. Hakeem: *PrepTestCases.java*, *RunAssignment.java*, *Grammar.java*: *toString(), isSGrammar()*
2. Ricardo: *PrepTestCases.java*, *Grammar.java*

Testing:

1. Kyle: Extensively tested the program by providing 5 more grammars (4 valid *s-grammars* and 1 invalid grammar), each with 5 test strings – 3 of which are accepted, 2 of which are rejected
2. Thembelihle: Extensively tested the program by providing 5 more grammars (4 valid *s-grammars* and 1 invalid grammar), each with 5 test strings – 3 of which are accepted, 2 of which are rejected
3. Hakeem: Tested the overall program – *RunAssignment.java*

Using Kyle's and Thembelihle's grammars, we have fully proven that our program works *flawlessly*.

Final Takings on Contributions:

I believe that Ricardo, Kyle, Thembelihle and myself deserve the same marks since we worked together, partitioned the workload, scheduled zoom meetings for discussions and fixes and overall have shown excellent team synergy. This assignment has also gone through extensive testing and works for way more than just the test cases provided in the file.

Conclusion:

My group and I found this assignment to be very enjoyable. It was interesting deciding how we wanted to structure our text file correctly. We had fun trying to figure out how to construct and represent grammars and thoroughly enjoyed working with the data structures rich with object-oriented programming design. It gave use a reminder of the good old days of second year and helped us brush up on our skills. It also allowed us to see that these concepts we learned in second year have limitless applications. This assignment also gave us a more in-depth understanding of grammars which will definitely help for test 2 and potentially the future where applicable. We therefore collectively thank you for this assignment and the knowledge, experience and insight gained from it!