

Experiment-2A

Objective: To tokenize the string using StringTokenizer class in java

Technical Prerequisites

StringTokenizer class

The string tokenizer lets an application to break a string into tokens. The tokenization method is much better than the one used by StreamTokenizer class in terms of simplicity. The StringTokenizer methods don't distinguish among identifiers, numbers and quoted strings, and they don't recognize and skip comments.

Methods used in the program:

i) hasMoreTokens()

Modifier and type: boolean

Description: This method checks if there are more tokens available from this tokenizer's string. If this method returns true, then the nextToken method will be called without an argument and will return a token.

ii) nextToken()

Modifier and type: String

Description: This method returns the next token from the string tokenizer. It throws

NoSuchElementException in case there are no more tokens in this tokenizer's string.

iii) **countTokens()**

Modifier and type: int

Description: This method calculates the number of times a tokenizer's nextToken method can be called before it generates an exception or in simpler words, the number of tokens remaining in the string using current delimiter set.

Program 1:

Source Code:

```
import java.util.StringTokenizer;
```

```
/**
```

```
*
```

```
* @author diva21
```

```
*/
```

```
public class stringTokenizer {
```

```
/**
```

```

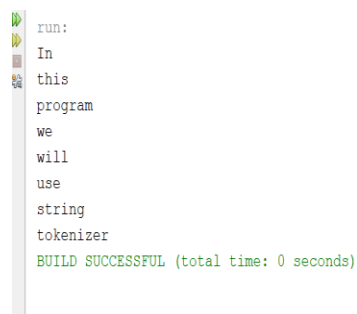
    * @param args the command line arguments
    */
    public static void main(String[] args) {
        // TODO code application logic here

        String msg = "In this program we will use string
tokenizer";

        StringTokenizer st = new StringTokenizer(msg, " ");
        while(st.hasMoreTokens()){
            System.out.println(st.nextToken());
        }
    }
}

```

Output:



```

run:
In
this
program
we
will
use
string
tokenizer
BUILD SUCCESSFUL (total time: 0 seconds)

```

Program 2: This program will take a String with multiple tokens and place each part in an array of Strings. It will then find the index of the largest String in the array.

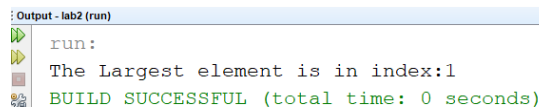
Source Code:

```
import java.util.StringTokenizer;

/**
 *
 * @author diva21
 */
public class StringTokenizer2 {
    public static void main(String args[]){
        String s="Five+Three=Nine-one";
        String arr[];
        StringTokenizer st= new StringTokenizer(s, "+=-");
        arr= new String[st.countTokens()];
        int i =0;
        while(st.hasMoreTokens()){
            arr[i]=st.nextToken();
            i++;
        }
        int indexMax=0;
        for(i =1; i< arr.length;i++){
```

```
if(arr[i].length()>arr[indexMax].length())  
    indexMax=i;  
}  
  
System.out.println("The Largest element is in index:"  
+indexMax);  
  
}  
  
}
```

Output:



```
: Output - lab2 (run)  
run:  
The Largest element is in index:1  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Learning Outcome: Learnt how to tokenize the string using String Tokenizer class in java

Experiment-2B

Objective: To tokenize the string using StreamTokenizer class in java

Technical Prerequisites

StreamTokenizer class

This class takes an input string and parses it into “tokens”, which can be read one at a time. Unlike

StringTokenizer , the stream tokenizer can recognize identifiers, numbers, quotes strings and recognize and skip comments.

Every byte read from the input stream is regarded as a character in the range '\u0000' through '\u00FF'. Then the character value is used to look up five possible attributes of the character: white space, alphabetic, numeric, string quote, and comment character. Each character can have zero or more of these attributes.

Methods used in the program:

ordinaryChar(int ch)

Modifier and type: void

Parameters: ch- the character

Description: This method specifies that the character argument is "ordinary" in this tokenizer. It removes any special significance the character has as a comment character, word component, string delimiter, white space, or number character.

quoteChar(int ch)

Modifier and type: void

Parameters: ch - character

Description: This method specifies that the matching pair of this character delimit string constants in the tokenizer.

The usual escape sequences such as `"\n"` and `"\t"` are recognized and converted to single characters as the string is parsed.

Any other attribute settings for the specified character are cleared.

nextToken()

Modifier and type: String

Description: This method returns the next token from the string tokenizer. It throws `NoSuchElementException` in case there are no more tokens in this tokenizer's string.

TT_EOF

Modifier and type: static int

Description: A constant indicating that the end of the stream has been read.

TT_EOL

Modifier and type: static int

Description: A constant indicating that the end of the line has been read.

TT_WORD

Modifier and type: static int

Description: A constant indicating that a word token has been read.

TT_NUMBER

Modifier and type: static int

Description: A constant indicating that a number token has been read.

FileOutputStream class

Origin/Package details:

FileOutputStream extends the package and exhibits the methods of java.io.OutputStream.

java.io.FileOutputStream

ObjectOutputStream class

Origin/Package details:

ObjectOutputStream extends the package and exhibits the methods of java.io.OutputStream.

java.io.ObjectOutputStream

Methods used in the program

writeUTF(String str)

Modifier and type: void

Parameters: str- string

Description: This method converts primitive data write of this String in modified UTF-8 format.

flush()

Modifier and type: void

Description: This method flushes the stream.

ObjectInputStream class

Origin/Package details:

ObjectInputStream extends and exhibits methods of java.io.InputStream package and implements Closeable, DataInput, ObjectInput, ObjectStreamConstants, AutoCloseable interfaces.

java.io.ObjectInputStream

Reader class

Origin/Package details:

Reader class extends and exhibits methods of java.io.Object package and implements Closeable, AutoCloseable,Readable interfaces.

Its known subclasses are BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, StringReader.

java.io.Reader

Subclasses from the above package used in the program:-

BufferedReader class: This subclass reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.

The buffer size may be specified, or the default size may be used. The default is large enough for most purposes.

InputStreamReader class: This subclass is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified charset. The charset that it uses may be specified by name or may be given explicitly, or the platform's default charset may be accepted.

Each invocation of one of an InputStreamReader's read() methods may cause one or more bytes to be read from the underlying byte-input stream. To enable the efficient conversion of bytes to characters, more bytes may be read ahead from the underlying stream than are necessary to satisfy the current read operation.

Source Code:

```
import java.io.BufferedReader;
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Reader;
import java.io.StreamTokenizer;
```

```
/**
```

```
 *
```

```
 * @author diva21
```

```
 */
```

```
public class streamTokenizer {
```

```
    public static void main(String[] args){
```

```
        String text ="Hello. This is a text  
ekhdskfidsldopasojs";
```

```
        try{
```

```
            //create a new file with an ObjectOutputStream
```

```
            FileOutputStream out = new  
FileOutputStream("test.txt");
```

```
        ObjectOutputStream oout = new
ObjectOutputStream(out);

        //write something in file
        oout.writeUTF(text);
        oout.flush();

        //create an ObjectInputStream for the file we
created before

        ObjectInputStream ois = new
ObjectInputStream(new FileInputStream("test.txt"));

        //create a new tokenizer

        Reader r = new BufferedReader(new
InputStreamReader(ois));

        StreamTokenizer st = new StreamTokenizer(r);

        //set \n as an ordinary char
        st.ordinaryChar('\n');
        st.quoteChar('o');

        //print the stream tokens
        boolean eof = false;
        do{
            int token = st.nextToken();
            switch (token) {
```

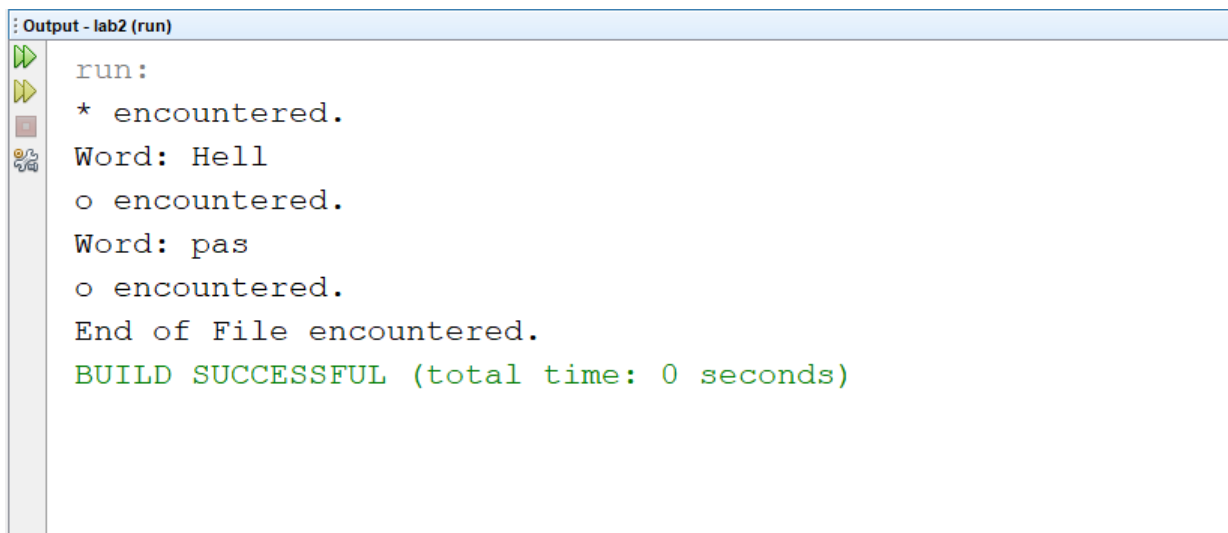
```
        case StreamTokenizer.TT_EOF:
            System.out.println("End of File
encountered.");
            eof = true;
            break;
        case StreamTokenizer.TT_EOL:
            System.out.println("End of Line
encountered.");
            break;
        case StreamTokenizer.TT_WORD:
            System.out.println("Word: " + st.sval);
            break;
        case StreamTokenizer.TT_NUMBER:
            System.out.println("Number: " + st.nval);
            break;
        default:
            System.out.println((char) token + "
encountered.");
            if (token == '!') {
                eof = true;
            }
        }
```

```

        } while (!eof);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
}

```

Output:



```

Output - lab2 (run)
run:
* encountered.
Word: Hell
o encountered.
Word: pas
o encountered.
End of File encountered.
BUILD SUCCESSFUL (total time: 0 seconds)

```

Learning Outcome:

Learn how to tokenize the string using
StreamTokenizer class in java

Experiment-2C

Objective: To find match of a string using Matcher and
Pattern classes in java

Technical Overview

Pattern class

A regular expression, in the form of string, must first be compiled into an instance of this class. The resulting pattern can then be used to create a Matcher object that can match arbitrary character sequences against the regular expression. All of the state involved in performing a match resides in the matcher, so many matchers can share the same pattern.

Origin or package details:

Pattern is a class in java.util.regex package.

java.util.regex.Pattern

compile(String regex)

Modifier and type: static Pattern

Parameters: The RE to be compiled

Description: Compiles the given regular expression into a pattern.

matcher(CharSequence input)

Modifier and type: Matcher

Parameters: input - The character sequence to be matched.

Description: Creates a matcher that will match the given input against this pattern.

Matcher class

A matcher is created from a pattern by invoking the pattern's matcher method. Once created, a matcher can be used to perform three different kinds of match operations:

The matches method attempts to match the entire input sequence against the pattern. The lookingAt method attempts to match the input sequence, starting at the beginning, against the pattern.

The find method scans the input sequence looking for the next subsequence that matches the pattern.

Each of these methods returns a boolean indicating success or failure. More information about a successful match can be obtained by querying the state of the matcher.

Origin or package details:

Matcher_ is a class in java.util.regex package extends the methods of java.lang.Object and implements MatchResult.

java.util.regex.Matcher

find()

Modifier and type: boolean

Description: Attempts to find the next subsequence of the input sequence that matches the pattern.

group(int group)

Modifier and type: String

Description: Returns the input subsequence captured by the given group during the previous match operation.

For a matcher `m`, input sequence `s`, and group index `g`, the expressions `m.group(g)` and `s.substring(m.start(g), m.end(g))` are equivalent.

Source Code:

```
package MatcherPattern;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 *
 * @author diva21
 */
```

```

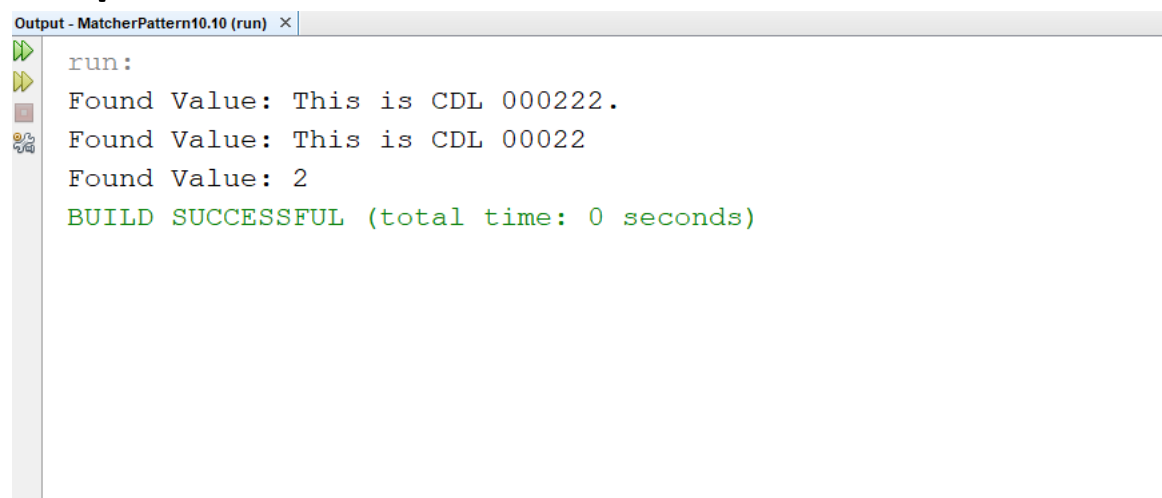
public class MatcherPattern {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // String for scanning and finding pattern
        String string = "This is CDL 000222.";
        String pattern = "(.*)(\\d)(.*)";
        //Pattern object
        Pattern r = Pattern.compile(pattern);
        //Matcher object
        Matcher m = r.matcher(string);
        if(m.find()){
            System.out.println("Found Value:
"+m.group(0));
            System.out.println("Found Value:
"+m.group(1));
            System.out.println("Found Value:
"+m.group(2));
        }
    }
}

```

```
//      System.out.println("Found Value:
"+m.group(3));
    }
    else{
        System.out.println("No Match");
    }
}
}
```

Output:



```
Output - MatcherPattern10.10 (run) ×
run:
Found Value: This is CDL 000222.
Found Value: This is CDL 00022
Found Value: 2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Learning Outcome: Learnt how to find match of a string using Matcher and Pattern classes in java

ASSIGNMENT-1

Name: Diva Lade
Program/Sem: B.Tech CSE,5
Reg. No.: 190101015

Compiler Design Lab Assignment – 1

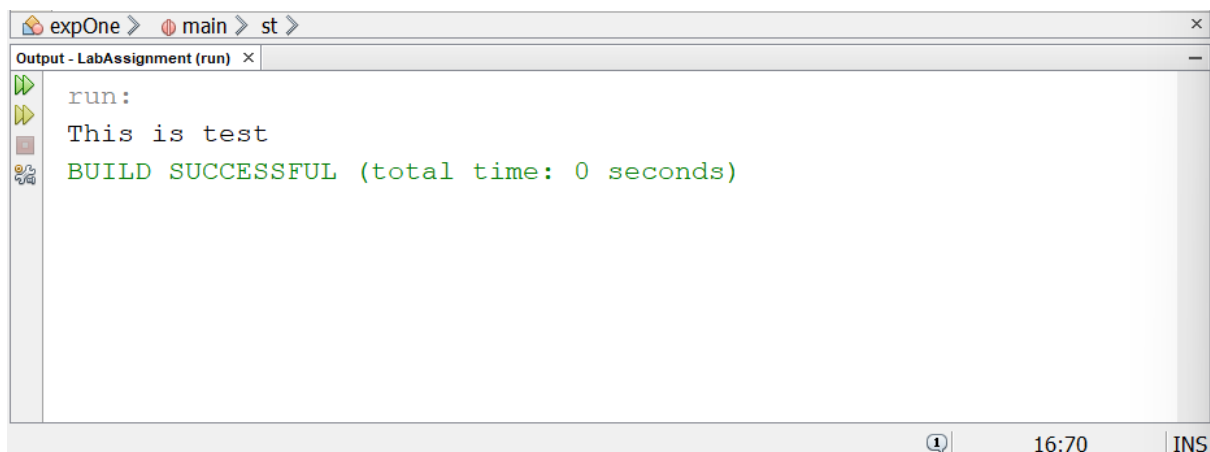
1. Write a Program in Java to tokenize string This is Test use ":" as delimiter

CODE:

```
import java.util.StringTokenizer;

/**
 *
 * @author diva21
 */
public class expOne {
    public static void main(String[] args){
        StringTokenizer st = new StringTokenizer("This is test", ":");
        while(st.hasMoreTokens()){
            System.out.println(st.nextToken());
        }
    }
}
```

OUTPUT:



The screenshot shows an IDE window titled 'expOne' with tabs for 'main' and 'st'. Below the tabs is an 'Output - LabAssignment (run)' window. The output text is as follows:

```
run:
This is test
BUILD SUCCESSFUL (total time: 0 seconds)
```

The status bar at the bottom of the IDE window shows '16:70' and 'INS'.

2. Write a Program in Java RE to tokenize the string 789 ,test 458

CODE:

```

/**
 *
 * @author diva21
 */
public class expTwo {

    public static void main(String args[]){

        String[] str ="789".split("");

        for(int x=0; x<str.length; x++){

            System.out.println(str[x]);

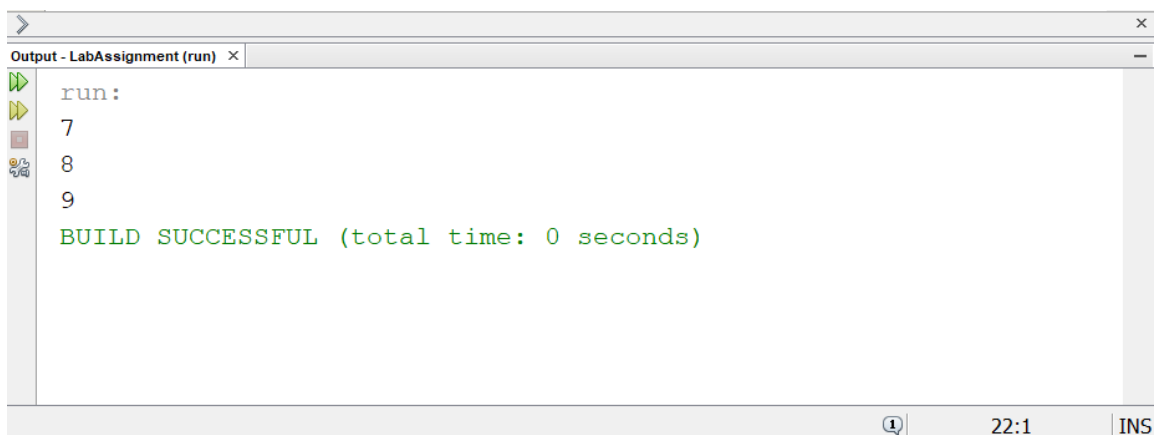
        }

    }

}

```

OUTPUT:

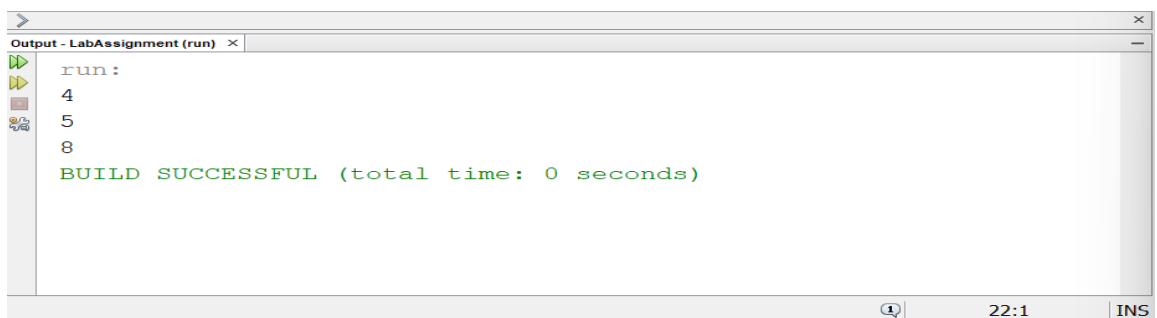


```

run:
7
8
9
BUILD SUCCESSFUL (total time: 0 seconds)

```

Test 458



```

run:
4
5
8
BUILD SUCCESSFUL (total time: 0 seconds)

```

3. Write a Program in Java RE to tokenize the string "http://10.879.86.67:85/";

CODE:

```
import java.util.StringTokenizer;
```

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author diva21
 */
public class expThree {

    public static void main(String args[]){

        StringTokenizer st = new StringTokenizer("
        \http://10.879.86.67:85/\";", "/\":[/][.][:][\]\\"]\"");

        while(st.hasMoreTokens()){

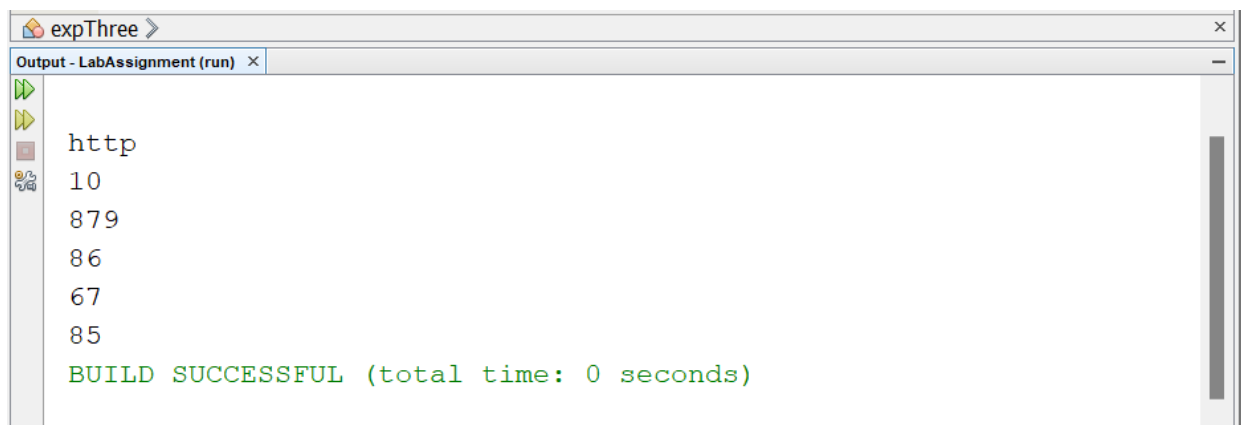
            System.out.println(st.nextToken());

        }

    }
}

```

OUTPUT:



```

expThree >
Output - LabAssignment (run) x
http
\
10
\
879
\
86
\
67
\
85
BUILD SUCCESSFUL (total time: 0 seconds)

```

Experiment-3A

Objective: To understand and implement the concept of Shift Reduce Parser in java.

Technical Prerequisites

class BufferedReader

Reads text from a character-input stream, buffering characters to provide for the efficient reading of characters, arrays, and lines.

The buffer size may be specified, or the default size may be used. The default is large enough for most purposes.

class InputStreamReader

An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified charset. The charset that it uses may be specified by name or may be given explicitly, or the platform's default charset may be accepted.

Methods used in the program

Integer.parseInt()

Modifier and type: String

Description: This method parses the String argument as a signed decimal integer object. It returns the

integer value which is represented by the argument in a decimal integer.

readLine()

Modifier and type: String

Description: Reads a line of text.

substring()

Modifier and type: String

Description: returns a part of the string.

indexOf()

Modifier and type: Integer

Description: to retrieve the index position at which a particular character or substring appears in another string.

equals()

Modifier and type: boolean

Description: compares the two given strings based on the content of the string. If any character is not matched, it returns false. If all characters are matched, it returns true.

Source Code:

```
import java.io.BufferedReader;  
import java.io.InputStreamReader;
```



```

/**
 *
 * @author diva21
 */
class ShiftRedParser{
public static void main(String[] args) throws Exception{
    int no,i,j,loc=0;
    String str,stack="",temp;
    String[][] productions;
    BufferedReader in= new BufferedReader(new
InputStreamReader(System.in));
    System.out.println("Enter no of productions");
    no = Integer.parseInt(in.readLine());
    productions = new String[no][2];
    System.out.println("Enter the productions");
    for(i=0; i<no; i++){
        System.out.println("LHS for production
" +(i+1) + ":");
        productions[i][0]=in.readLine();
    }
}
}

```

```

        System.out.println("RHS for production
" + (i+1) + " :");
        productions[i][1] = in.readLine();
    }
    System.out.println("The productions are: ");
    for(i=0; i<no; i++){
        System.out.println(productions[i][0] + " -> " +
productions[i][1]);
    }
    System.out.println("Enter a string: ");
    str = in.readLine();
    while(loc<str.length()-1){
        temp = str.substring(loc, str.indexOf(' ', loc));
        loc = str.indexOf(' ', loc) + 1;
        for(i=0; i<no; i++){
            if(temp.equals(productions[i][1])){
                temp = productions[i][0];
                break;
            }
        }
        stack = stack + temp;
    }

```

```
System.out.println("Stack contents: "+stack);
for(i=0;i<no;i++){
    if(stack.equals(productions[i][1])){
        stack = productions[i][0];
        break;
    }
}
}
System.out.println("Stack contents: "+stack);
if(stack.equals(productions[0][0]))
    System.out.println("Accepted");
else
    System.out.println("Rejected");
}
}
```

Output:

```
Output - Lab3ABCD_31.08.2021 (run) #3 X
RUN:
Enter no of productions
2
Enter the productions
LHS for production 1:
E
RHS for production 1:
E+E
LHS for production 2:
E
RHS for production 2:
id
The productions are:
E -> E+E
E -> id
Enter a string:
id + id
Stack contents: E
Stack contents: E+
Stack contents: E+E
Stack contents: E
Accepted
BUILD SUCCESSFUL (total time: 29 seconds)
```

Learning Outcome:

Learn how to implement the concept of Shift Reduce Parser in java.

Experiment-3B

Objective: To understand and implement the concept of parsing an XML file with Java with JavaScript DOM methods.

Technical Prerequisites

File class

Utility

An abstract representation of file and directory pathnames. User interfaces and operating systems use an abstract, system-independent view of hierarchical pathnames.

Origin/Package details:

java.io.File

DocumentBuilderFactory class

Utility

Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents.

Origin/Package details:

javax.xml.parsers.DocumentBuilderFactory

Methods used in the program

newInstance()

Modifier and type: static DocumentBuilderFactory

Description: Obtain a new instance of a DocumentBuilderFactory.

newDocumentBuilder()

Modifier and type: abstract DocumentBuilder

Description: Creates a new instance of a DocumentBuilder using the currently configured parameters.

DocumentBuilder class

Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents.

Origin/Package details:

org.w3c.dom.Document

Methods used in the program

parse()

Modifier and type: File

Description: Parse the content of the given file as an XML document and return a new DOM Document object. An `IllegalArgumentException` is thrown if the File is null.

NodeList class

The `NodeList` interface provides the abstraction of an ordered collection of nodes, without defining or constraining how this collection is implemented. `NodeList` objects in the DOM are live.

The items in the `NodeList` are accessible via an integral index, starting from 0.

Origin/Package details:

org.w3c.dom.NodeList

Methods used in the program

.item(int)

Modifier and type: Node

Description: Returns the `index`th item in the collection.

Node class

The Node interface is the primary datatype for the entire Document Object Model. It represents a single node in the document tree. While all objects implementing the Node interface expose methods for dealing with children, not all objects implementing the Node interface may have children. For example, Text nodes may not have children, and adding children to such nodes results in a DOMException being raised.

Origin/Package details:

org.w3c.dom.Node

Methods used in the program

.getNodeTypes()

Modifier and type: short

Description: A code representing the type of the underlying object, as defined above.

.ELEMENT_NODE

Modifier and type: static short

Description: The node is an Element.

Element (Interface)

The Element interface represents an element in an HTML or XML document.

Origin/Package details:

org.w3c.dom.Element

Js DOM

.getDocumentElement()

Modifier and Type: Element

Description: This is a convenience attribute that allows direct access to the child node that is the document element of the document.

.getNodeName()

Modifier and Type: String

Description: The value of this node, depending on its type; see the table above. When it is defined to be null, setting it has no effect, including if the node is read-only.

.getElementByTagName()

Modifier and Type: NodeList

Description: Returns a NodeList of all descendant Elements with a given tag name, in document order.

.getAttribute()

Modifier and Type: String

Description: Retrieves an attribute value by name.

Source Code:

XML File


```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!--
```

To change this license header, choose License Headers in Project Properties.

To change this template file, choose Tools | Templates and open the template in the editor.

```
-->
```

```
<company>
```

```
  <staff id="1">
```

```
    <firstname>John</firstname>
```

```
    <lastname>Kim</lastname>
```

```
    <nickname>JK</nickname>
```

```
    <salary>10000</salary>
```

```
  </staff>
```

```
  <staff id="2">
```

```
    <firstname>Brian</firstname>
```

```
    <lastname>Adams</lastname>
```

```
    <nickname>BA</nickname>
```

```
    <salary>20000</salary>
```

```
  </staff>
```

</company>

Java File

```
package lab3abcd_31_10_2021;
```

```
import java.io.File;
```

```
import javax.xml.parsers.DocumentBuilder;
```

```
import javax.xml.parsers.DocumentBuilderFactory;
```

```
import org.w3c.dom.Document;
```

```
import org.w3c.dom.Element;
```

```
import org.w3c.dom.Node;
```

```
import org.w3c.dom.NodeList;
```

```
/**
```

```
 *
```

```
 * @author diva21
```

```
 */
```

```
public class XMLJava {
```

```
    public static void main(String args[]){
```

```
        try{
```

```
File fXml = new  
File("C:\\Users\\dival\\Documents\\NetBeansProjects\\  
\\Lab3ABCD_31.10.2021\\src\\lab3abcd_31_10_2021\\  
parser.xml");
```

```
DocumentBuilderFactory dbF =  
DocumentBuilderFactory.newInstance();
```

```
DocumentBuilder dB =  
dbF.newDocumentBuilder();
```

```
Document doc = dB.parse(fXml); //heart of this  
application as allows parsing through java
```

```
doc.getDocumentElement().normalize();
```

```
System.out.println("Root  
element:"+doc.getNodeName());
```

```
NodeList nList =  
doc.getElementsByTagName("staff");
```

```
System.out.println("-----");
```

```
for (int temp = 0; temp < nList.getLength();  
temp++)
```

```
{
```

```
Node nNode = nList.item(temp);
```

```
System.out.println("\nCurrent Element : " +  
nNode.getNodeName());
```

```

        if (nNode.getNodeType() ==
Node.ELEMENT_NODE) {

            Element eElement = (Element) nNode;

            System.out.println("Staff id : " +
eElement.getAttribute("id"));

            System.out.println("First Name : " +
eElement.getElementsByTagName("firstname").item(0)
).getTextContent());

            System.out.println("Last Name : "+
eElement.getElementsByTagName("lastname").item(0)
.getTextContent());

            System.out.println("Nick Name : "+
eElement.getElementsByTagName("nickname").item(0)
.getTextContent());

            System.out.println("Salary : " +
eElement.getElementsByTagName("salary").item(0).ge
tTextContent());

        }

    }

}catch(Exception e){

    e.printStackTrace();
}

```

```
    }  
  }  
}
```

Output:

Learning Outcome:

Learn how to implement the concept of parsing an XML file with Java with JavaScript DOM methods.

Experiment-3C

Objective:

Technical Prerequisites

Methods used in the program

Modifier and type:

Description:

Origin/Package details:

Subclasses from the above package used in the program:-

Source Code:

Output:

Learning Outcome:

Learn how to

Experiment-3D

Objective: To parse a json file using java class FileWriter and jar file org.json.simple.

Technical Prerequisites

class JSONObject

A JSONObject is an unordered collection of key and value pairs, resembling Java's native Map implementations. Keys are unique Strings that cannot be null.

Values can be anything from a Boolean, Number, String, or JSONArray to even a JSONObject. NULL object.

A JSONObject can be represented by a String enclosed within curly braces with keys and values separated by a colon, and pairs separated by a comma.

It has several constructors with which to construct a JSONObject.

Methods used in the program

put(String key, Object value)

Modifier and type: JSONObject

Description: inserts or replaces a key-value pair in current JSONObject.

The put() method is an overloaded method that accepts a key of type String and multiple types for the value.

Origin/Package details:

org.simple.JSONObject

class JSONArray

A JSONArray is an ordered collection of values, resembling Java's native Vector implementation.

Values can be anything from a Number, String, Boolean, JSONArray, or JSONObject to even a JSONObject.NULL object.

It's represented by a String wrapped within square brackets and consists of a collection of values separated by commas.

Like JSONObject, it has a constructor that accepts a source String and parses it to construct a JSONArray.

It has several constructors with which to construct a JSONObject.

Methods used in the program

add()

Modifier and type: boolean

Description: appends the element at the end of the list. Since List supports Generics, the type of elements that can be added is determined when the list is created.

Source Code:

```
package lab3abcd_31_10_2021;

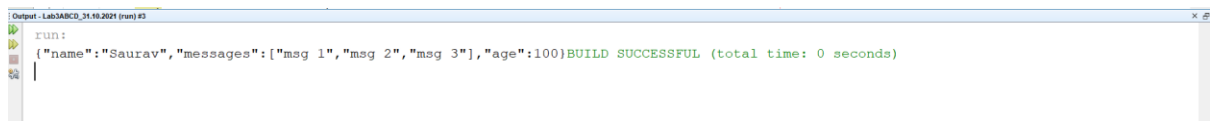
import java.io.FileWriter;
import java.io.IOException;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;

public class JsonSimpleParser{
    public static void main(String[] args){
        JSONObject obj = new JSONObject();
        obj.put("name","Saurav");
```



```
obj.put("age", new Integer(100));
JSONArray list = new JSONArray();
list.add("msg 1");
list.add("msg 2");
list.add("msg 3");
obj.put("messages", list);
try{
    FileWriter file = new
FileWriter("C:\\Users\\dival\\Documents\\NetBea
nsProjects\\Lab3ABCD_31.10.2021\\src\\file.json
");
        file.write(obj.toJSONString());
        file.flush();
        file.close();
    } catch(IOException e){
e.printStackTrace();
    }
    System.out.print(obj);
}}
```

Output:

A screenshot of an IDE's output window. The title bar reads "Output - Lab3ABCD_21.10.2021 (run) #3". The output text is:

```
run:
{"name":"Saurav","messages":["msg 1","msg 2","msg 3"],"age":100}BUILD SUCCESSFUL (total time: 0 seconds)
```

Learning Outcome:

Learn how to

class FileWriter

Convenience class for writing character files. The constructors of this class assume that the default character encoding and the default byte-buffer size are acceptable.

Methods used in the program

write()

Modifier and type: void

Description: Writes a portion of a string.

flush()

Modifier and type: void

Description: Flushes the stream.

close()

Modifier and type: void

Description: Closes the stream, flushing it first.

class JSONParser

Represents a parser for JSON-encoded content.

Methods used in the program

get()

Modifier and type:

Description:

parse()

Modifier and type:

Description:

Origin/Package details:

org.json.simple.parser.JSONParser

class Iterator

Methods used in the program

Modifier and type:

Description:

Modifier and type:

Description:

Origin/Package details: *java.util.Iterator*

Source Code:

```
import java.io.FileNotFoundException;
```

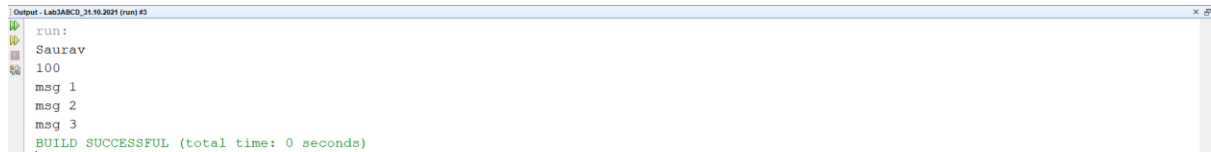
```
import java.io.FileReader;
import java.io.IOException;
import java.util.Iterator;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

public class JsonSimpleExample {
    public static void main(String[] args) {
        JSONParser parser = new JSONParser();
        try {
            Object obj = parser.parse(new
            FileReader("C:\\Users\\dival\\Documents\\NetBeansPr
            ojects\\Lab3ABCD_31.10.2021\\src\\file.json"));
            JSONObject jsonObject = (JSONObject) obj;
            String name = (String)
            jsonObject.get("name");
            System.out.println(name);
            long age = (Long) jsonObject.get("age");
            System.out.println(age);
        }
    }
}
```

```
// loop array
JSONArray msg = (JSONArray)
jsonObject.get("messages");
Iterator<String> iterator =
msg.iterator();
while (iterator.hasNext()) {

System.out.println(iterator.next());
}
} catch (FileNotFoundException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
} catch (ParseException e) {
e.printStackTrace();
}
}
}
```

Output:

A screenshot of a terminal window titled "Output - Lab3ARCD_31.05.2021 (run) #3". The terminal displays the following text: "run:", "Saurav", "100", "msg 1", "msg 2", "msg 3", and ".BUILD SUCCESSFUL (total time: 0 seconds)".

```
run:
Saurav
100
msg 1
msg 2
msg 3
.BUILD SUCCESSFUL (total time: 0 seconds)
```

Learning Outcome: