JAVA.IO.STREAMTOKENIZER.PUSHBACK METHOD

 Copyright © tutorialspoint.com

Description

The **java.io.StreamTokenizer.pushBack** method causes the next call to the nextToken method of this tokenizer to return the current value in the ttype field, and not to modify the value in the nval or sval field.

Declaration

Following is the declaration for **java.io.StreamTokenizer.pushBack** method.

```
public void pushBack()
```

Parameters

- **NA**

Return Value

This method does not return a value.

Exception

- **NA**

Example

The following example shows the usage of **java.io.StreamTokenizer.pushBack** method.

```
package com.tutorialspoint;

import java.io.*;

public class StreamTokenizerDemo {

   public static void main(String[] args) {

      String text = "Hello. This is a text \n that will be split "
              + "into tokens. 1+1=2";
      try {
         // create a new file with an ObjectOutputStream
         FileOutputStream out = new FileOutputStream("test.txt");
         ObjectOutputStream oout = new ObjectOutputStream(out);

         // write something in the 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);

         // specify that numbers should be parsed
         st.parseNumbers();

         // 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;
                  }
            }

            // pushback the last token
            st.pushBack();

            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();
      }
   }
}
```

Let us compile and run the above program, this will produce the following result:

```
Word: Hello.
Word: Hello.
Word: This
Word: This
Word: is
Word: is
Word: a
Word: a
Word: text
```

```
Word: text
Word: that
Word: that
Word: will
Word: will
Word: be
Word: be
Word: split
Word: split
Word: into
Word: into
Word: tokens.
Word: tokens.
Number: 1.0
Number: 1.0
+ encountered.
+ encountered.
Number: 1.0
Number: 1.0
= encountered.
= encountered.
Number: 2.0
Number: 2.0
End of File encountered.
End of File encountered.
```

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js