1 – Extract Lexemes

Tuesday, January 9, 2024 10:10 AM

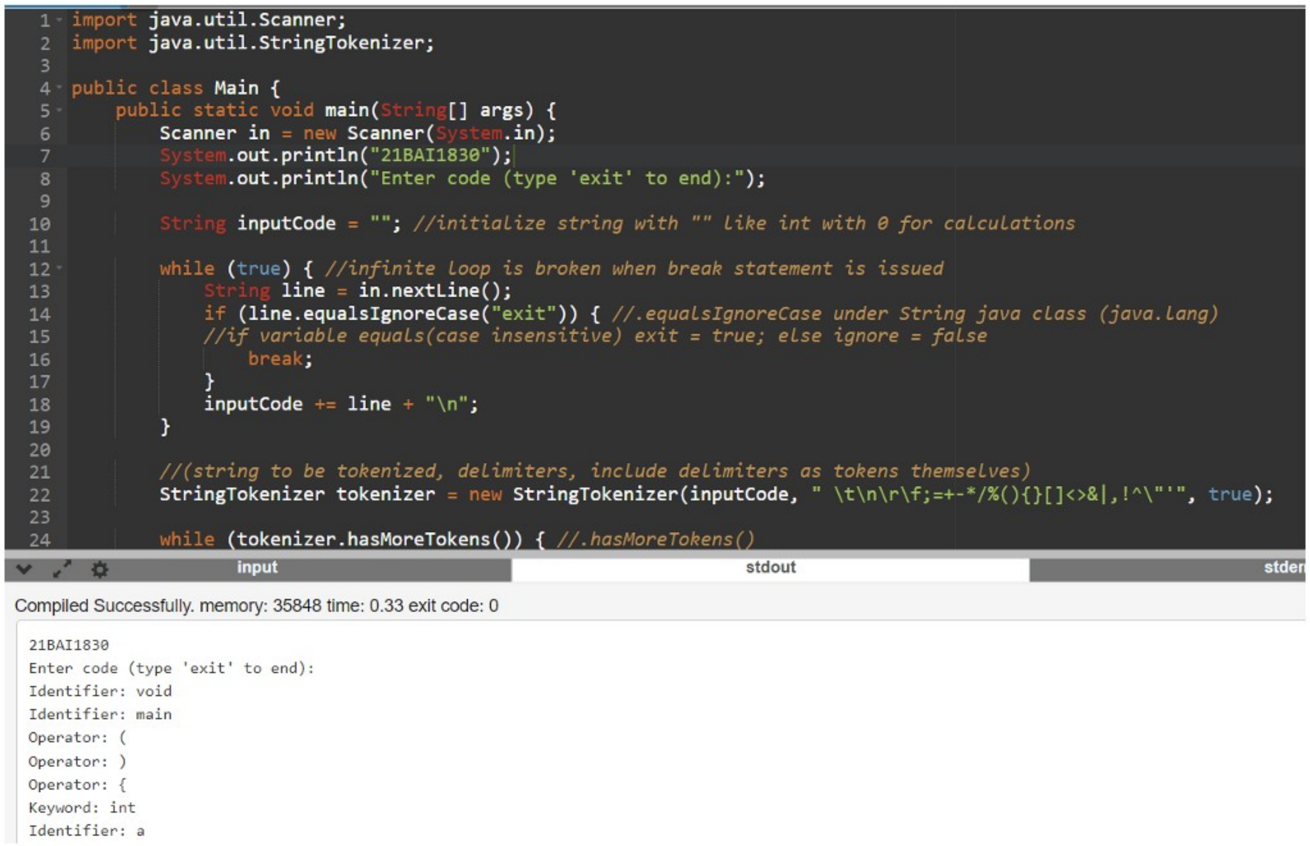
Ashima Fatima Seik Mugibur Raghman, 21BAI1830

AIM: Write a program to extract lexemes from the given input and identify its token type.

ALGORITHM:

1. Prompt user to enter code line by line until they type 'exit'.
2. Use 'StringTokenizer' to break the input code into tokens based on specific delimiters.
3. Iterate through the tokens.
4. For each token, ignore if it is empty. Else, check if it's a keyword, identifier, operator, delimiter, literal or number.
   - for keyword: check if token matches keyword.
   - identifier: if 1st charecter is a letter
   - operator: if token is one of the specified operators.
   - delimiters: if token is one of specified delimiters.
   - literals: check using regular expression if token is a string literal.
   - numbers: check using regular expression if token is a number.

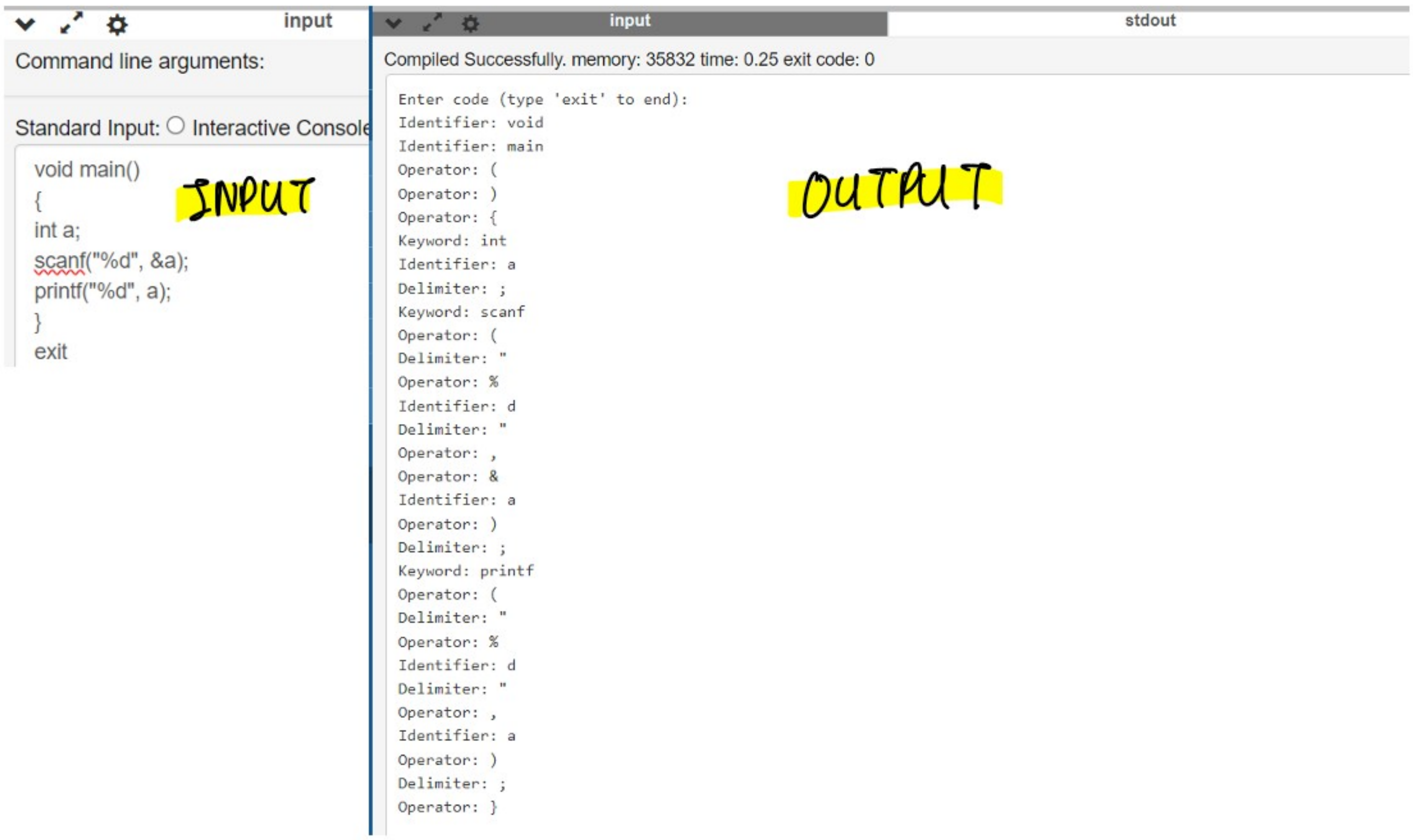
STIMULATION:



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

public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.println("21BAI1830");
        System.out.println("Enter code (type 'exit' to end):");

        String inputCode = ""; //initialize string with "" like int with 0 for calculations

        while (true) { //infinite loop is broken when break statement is issued
            String line = in.nextLine();
            if (line.equalsIgnoreCase("exit")) { //.equalsIgnoreCase under String java class (java.lang)
            //if variable equals(case insensitive) exit = true; else ignore = false
                break;
            }
            inputCode += line + "\n";
        }

        //(string to be tokenized, delimiters, include delimiters as tokens themselves)
        StringTokenizer tokenizer = new StringTokenizer(inputCode, " \t\n\r\f;=+-*/%(){}[]<>&|,!^\"'", true);

        while (tokenizer.hasMoreTokens()) { //.hasMoreTokens()
```

Compiled Successfully. memory: 35848 time: 0.33 exit code: 0

```
21BAI1830
Enter code (type 'exit' to end):
Identifier: void
Identifier: main
Operator: (
Operator: )
Operator: {
Keyword: int
Identifier: a
```



Command line arguments:

Standard Input: ○ Interactive Console

```
void main()
{
int a;
scanf("%d", &a);
printf("%d", a);
}
exit
```

Compiled Successfully. memory: 35832 time: 0.25 exit code: 0

```
Enter code (type 'exit' to end):
Identifier: void
Identifier: main
Operator: (
Operator: )
Operator: {
Keyword: int
Identifier: a
Delimiter: ;
Keyword: scanf
Operator: (
Delimiter: "
Operator: %
Identifier: d
Delimiter: "
Operator: ,
Operator: &
Identifier: a
Operator: )
Delimiter: ;
Keyword: printf
Operator: (
Delimiter: "
Operator: %
Identifier: d
Delimiter: "
Operator: ,
Identifier: a
Operator: )
Delimiter: ;
Operator: }
```

CODE:

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

```
public class Main {
   public static void main(String[] args) {
      Scanner in = new Scanner(System.in);
      System.out.println("Enter code (type 'exit' to end):");

      String inputCode = ""; //initialize string with "" like int with 0 for calculations

      while (true) { //infinite loop is broken when break statement is issued
         String line = in.nextLine();
         if (line.equalsIgnoreCase("exit")) { //.equalsIgnoreCase under String java class (java.lang)
         //if variable equals(case insensitive) exit = true; else ignore = false
            break;
         }
         inputCode += line + "\n";
      }

      //(string to be tokenized, delimiters, include delimiters as tokens themselves)
      StringTokenizer tokenizer = new StringTokenizer(inputCode, " \t\n\r\f;=+-*/%(){}[]<>&|,!^\"'", true);

      while (tokenizer.hasMoreTokens()) { //.hasMoreTokens()
         String token = tokenizer.nextToken(); //.nextToken() : used to retrieve the next token from the
string being tokenized

         if (token.trim().isEmpty()) { //.trim() : This expression returns a new string with leading and
trailing whitespaces
         //removed from the original token
         //.isEmpty for when the token contains only whitespaces
         } else if (isKeyword(token)) {
            System.out.println("Keyword: " + token);
         } else if (isIdentifier(token)) {
            System.out.println("Identifier: " + token);
         } else if (isOperator(token)) {
            System.out.println("Operator: " + token);
         } else if (isDelimiter(token)) {
            System.out.println("Delimiter: " + token); //delimiter: a charecter to specify boundry of another
set of charecters
         } else if (isLiteral(token)) {
            System.out.println("Literal: " + token);
         } else if (isNumber(token)) {
            System.out.println("Number: " + token);
         } else {
            System.out.println("Unknown token: " + token);
         }
      }
   }

   private static boolean isKeyword(String token) { //private: accessible only within that class
   //static method is a class-level method that can be called without creating an instance of the class
      return token.matches("^(int|float|double|if|else|while|for|scanf|printf)$"); //.matches()
   }

   private static boolean isIdentifier(String token) {
      //checks if the character at the first position (charAt(0)) of the string token is a letter
      return Character.isLetter(token.charAt(0));
   }
```

```
    private static boolean isOperator(String token) {
        return "+-*/%(){}[]<>&|,!^".contains(token); //.contains()
    }

    private static boolean isDelimiter(String token) {
        return ";=(){}[]<>&|,^\"'".contains(token);
    }

    private static boolean isLiteral(String token) {
        // Check for string literals enclosed in double quotes
        return token.matches("^\"[^\"]*\"$|^%[a-zA-Z]$"); //.matches() : uses regular expression to check
for
    }

    private static boolean isNumber(String token) {
        // Check for numeric literals
        return token.matches("^\\d+(\\.\\d+)?$");
    }
    //refer notes about literals
}
```