```
import java.util.ArrayList;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class Driver {

	//*** Simple error handling routine: reports line number and message ***//
	public static void error(String errorMessage, int lineNumber) {
		System.out.print("Error on line " + lineNumber + ": ");
		System.out.println(errorMessage);
		System.exit(1);
	}

	//*** Runtime errors -- no line number ***//
	public static void error(String errorMessage) {
		System.out.println(errorMessage);
		System.exit(1);
	}

	 //*** Run a script: main driver of the interpreting process ***//
	public void run(String inputPath) throws IOException {
		// Step 1: Open then file and read its characters into a buffer
		byte[] bytes = Files.readAllBytes(Paths.get(inputPath));
		String characters = new String(bytes);

		// Step 2: Lexical analysis: Returns an ArrayList<Token>
		Lexer lex = new Lexer(characters);
		ArrayList<Token> tokens = lex.analyze();

		for (Token t : tokens) {
			System.out.println(t);
		}

		//*** Parsing and execution will go here in future versions (project 2) ***//
	 }

	//*** Main: interpret the file name given as a command line argument ***//
	public static void main(String[] args) {
		if (args.length < 1) {
			System.out.println("Usage: java Driver FILENAME");
			System.exit(0);
		}

		Driver Driver = new Driver();

		try {
		   Driver.run(args[0]);
		} catch(IOException e) {
		   Driver.error("Could not open file " + args[0] + ".", 0);
		}
	}
}
```

```
enum Tokens {
  NAME,
  PLUS,
  MINUS,
  TIMES,
  DIVIDE,
  EOF,
  UNKNOWN









}

public class Token {

	Tokens type;

	// Some tokens have associated values:
	// NAME tokens have a string that is the name of the variable
	// STRING tokens have the string
	// NUMBER tokens have the value of the number

	Object value;
	int line;

	public Token(Tokens type, Object value, int line) {
		this.type = type;
		this.value = value;
		this.line = line;
	}

	public Token(Tokens type, int line) {
		this.type = type;
		this.value = null;
		this.line = line;
	}

	public String toString() {
		return "<" + this.type + ", value = " + this.value + ", line = " + this.line + ">";
	}
}
```

```
import java.util.ArrayList;

public class Lexer {
	// String containing the program's text
	String program;

	// Index of the current character being analyzed
	int index;

	// Line counter
	// Used for error reporting and incremented on each newline
	int line;

	//*** Constructor ***//
	public Lexer(String program) {
		this.program = program;
		this.index = 0;
		this.line = 1;
	}

	//*** Back-up one character ***//
	public void unread() {
		this.index--;
	}

	//*** Return the next character and advance the index pointer ***//
	public char nextCharacter() {
		if (this.index >= this.program.length()) {
			this.index++;
			return (char) 0;  // EOF
		}

		char c = this.program.charAt(this.index);
		this.index++;
		return c;
	}

	//*** Build an identifier: returns a NAME or keyword Token ***//
	public Token analyzeIdentifier() {
		 String identifier = "";

		while (true) {
			char c = nextCharacter();

			if (!Character.isLetter(c) && !Character.isDigit(c) && c != '_') {
				unread();
				break;
			} else {
				identifier += c;
			}
		}
		 return new Token(Tokens.NAME, identifier, this.line);
	}
```

```
//*** Integer number literals ***//
public Token analyzeNumber() {









}


//*** Find and return the next Token ***//
public Token nextToken() {

	while (true) {
		char c = nextCharacter();

		if (c == 0) {
			return new Token(Tokens.EOF, this.line);
		}

		// NAME TOKEN
		if (Character.isLetter(c)) {
			unread();
			return analyzeIdentifier();
		}

		// INTEGER TOKEN
		else if (Character.isDigit(c) || c == '.') {
			unread();
			return analyzeNumber();
		}

		// Math symbols
		else if (c == '+') {
			return new Token(Tokens.PLUS, this.line);
		} else if (c == '*') {
			return new Token(Tokens.TIMES, this.line);
		} else if (c == '/') {
			return new Token(Tokens.DIVIDE, this.line);
```

```
                } else if (c == '-') {
                        return new Token(Tokens.MINUS, this.line);
                }

                //Relational operators!









                // Newline
                else if (c == '\n') {
                        this.line++;
                }

                // Default: ignore whitespace
                else if (!Character.isWhitespace(c)) {
                        Driver.error("Unexpected character " + (char) c, this.line);
                }
        }
}


//*** Main lexical analysis routine ***//
//
// Scans the input program and returns an ArrayList containing all of its
// Tokens.
public ArrayList<Token> analyze() {

        ArrayList<Token> tokens = new ArrayList<Token>();

        Token t;
        do {
                t = nextToken();
                tokens.add(t);
        } while (t.type != Tokens.EOF);

        return tokens;
}
}
```