Introduction to Compiler Construction

Assignment 2 (Scanning) Discussion

Problem 1 (Multiline Comment)

Add support for multiline comments in which all characters between `/*` and `*/` are ignored

In `Scanner.getNextToken()`

- On seeing a `*` after a `/`, set `inComment` to `true` and advance the input
- Repeat as long as the current character is not a newline or `EOF`
    - If it is a `*` and the next character is a `/`, set `inComment` to `false`, advance the input, and break
    - Advance the input
- Report a scanner error if `inComment` is `true`

Testing

```
>_ ~/workspace/j--

$ ant
$ ./bin/j-- -t scanning/MultilineComment.java
3         : "import" = import
3         : <IDENTIFIER> = java
...
19        : "}" = }
21        : <EOF> = <end of file>
```

Compare your output with the reference output in `scanning/MultilineComment.tokens`

Problem 2 (Reserved Words)

Add support for the following reserved words in *j--*:

`break`	`case`	`continue`	`default`
`double`	`for`	`long`	`switch`

For each reserved word

- Define a token in the `TokenInfo.TokenKind` (eg, `BREAK` for `break`)
- Add the token to the table of reserved words in `Scanner`

Testing

```
>_ ~/workspace/j--

$ ant
$ ./bin/j-- -t scanning/Keywords.java
1         : "abstract" = abstract
2         : "boolean" = boolean
...
40        : "while" = while
41        : <EOF> = <end of file>
```

Compare your output with the reference output in `scanning/Keywords.tokens`

Problem 3 (Operators)

Add support for the following operators:

`-=` `*=` `/=` `%=` `!=` `>=` `<` `||`

For each operator

- Define a token in the `TokenInfo.TokenKind` (eg, `MINUS_ASSIGN` for `-=`)
- Scan the token in `Scanner.getNextToken()`

Testing

```
>_ ~/workspace/j--

$ ant
$ ./bin/j-- -t scanning/Operators.java
1         : "=" = =
2         : ":" = :
...
24        : "*=" = *=
25        : <EOF> = <end of file>
```

Compare your output with the reference output in `scanning/Operators.tokens`

Problem 4 (Literals)

Add support for long and double literals

Regular expressions for int, long, and double literals

```
DIGITS         ::= ( "0"..."9" ) { "0"..."9" }
INT_LITERAL    ::= DIGITS
LONG_LITERAL   ::= INT_LITERAL ( "l" | "L" )
EXPONENT       ::= ( "e" | "E" ) [ ( "+" | "-" ) ] DIGITS
SUFFIX         ::= "d" | "D"
DOUBLE_LITERAL ::= DIGITS "." [ DIGITS ] [ EXPONENT ] [ SUFFIX ] // part 1
                 | "." DIGITS [ EXPONENT ] [ SUFFIX ]            // part 2
                 | DIGITS EXPONENT [ SUFFIX ]                    // part 3
                 | DIGITS SUFFIX                                 // part 4
```

Implement the following helper methods

- `private String digits()` that scans and returns a string of digits starting at `ch`, which must be a digit
- `private String exponent()` that scans and returns an exponent starting `ch`, which must be an `'e'` or `'E'`

Problem 4 (Literals)

In `Scanner.getNextToken()`

- Group cases `'0'`, `'1'`, ..., `'9'`, and `'.'`
- Under that group, create a `StringBuilder` object called `buffer`
- If `ch` is a digit
  - Append the digits starting at `ch` to `buffer` (use `digits()`)
  - If `ch` is `'l'` or `'L'`, append it to `buffer`, advance the input, and return a `TokenInfo` object for a long literal
  - If `ch` is not any of `'.'`, `'e'`, `'E'`, `'d'`, or `'D'`, return a `TokenInfo` object for an int literal
  - If `ch` is `'.'`, see "Scanning double literals (part 1)"
  - Otherwise, see "Scanning double literals (parts 2 and 3)"
- Otherwise, see "Scanning double literals (part 4)"

Problem 4 (Literals)

Scanning double literals (part 1)

- Append `ch` to `buffer`
- Advance the input
- If `ch` is a digit, append the digits starting at `ch` to `buffer` (use `digits()`)
- If `ch` is `'e'` or `'E'`, append the exponent starting at `ch` to `buffer` (use `exponent()`)
- If `ch` is `'d'` or `'D'`, append it to `buffer`, and advance the input
- Return a `TokenInfo` object for a double literal

Problem 4 (Literals)

Scanning double literals (parts 2 and 3)

- If `ch` is `'e'` or `'E'`
  - Append the exponent starting at `ch` to `buffer` (use `exponent()`)
  - If `ch` is `'d'` or `'D'`, append it to `buffer`, and advance the input
- Otherwise
  - If `ch` is `'d'` or `'D'`, append it to `buffer`, and advance the input
  - Otherwise, report a "malformed double literal" error
- Otherwise, return a `TokenInfo` object for a double literal

Problem 4 (Literals)

Scanning double literals (part 4)

- Advance the input
- If `ch` is a digit
    - Append `'.'` to `buffer`
    - Append digits starting at `ch` to `buffer` (use `digits()`)
    - If `ch` is `'e'` or `'E'`, append the exponent starting at `ch` to `buffer` (use `exponent()`)
    - If `ch` is `'d'` or `'D'`, append it to `buffer`, and advance the input
    - Return a `TokenInfo` object for a double literal
- Return a `TokenInfo` object the separator `DOT`

Problem 4 (Literals)

Testing int and long literals

```
>_ ~/workspace/j--
```

```
$ ant
$ ./bin/j-- -t scanning/IntLiterals.java
1         : <INT_LITERAL> = 0
2         : <INT_LITERAL> = 9
...
5         : <INT_LITERAL> = 1234567890
6         : <EOF> = <end of file>
$ ./bin/j-- -t scanning/LongLiterals.java
1         : <LONG_LITERAL> = 1l
2         : <LONG_LITERAL> = 9L
...
6         : <LONG_LITERAL> = 1234567890L
7         : <EOF> = <end of file>
```

Compare your output with the reference output in `scanning/IntLiterals.tokens` and `scanning/LongLiterals.tokens`

Problem 4 (Literals)

Testing double literals

```
>_ ~/workspace/j--
```

```
$ ./bin/j-- -t scanning/DoubleLiterals1.java
1         : <DOUBLE_LITERAL> = 0.
2         : <DOUBLE_LITERAL> = 1.
...
74        : <DOUBLE_LITERAL> = 123456789.e-135D
75        : <EOF> = <end of file>
$ ./bin/j-- -t scanning/DoubleLiterals2.java
1         : <DOUBLE_LITERAL> = .0
2         : <DOUBLE_LITERAL> = .1
...
32        : <DOUBLE_LITERAL> = .098765e-135
33        : <EOF> = <end of file>
$ ./bin/j-- -t scanning/DoubleLiterals3.java
1         : <DOUBLE_LITERAL> = 0e2
2         : <DOUBLE_LITERAL> = 9e9
...
21        : <DOUBLE_LITERAL> = 246e-13D
22        : <EOF> = <end of file>
$ ./bin/j-- -t scanning/DoubleLiterals4.java
1         : <DOUBLE_LITERAL> = 0d
2         : <DOUBLE_LITERAL> = 0D
...
6         : <DOUBLE_LITERAL> = 0987654321D
7         : <EOF> = <end of file>
```

Compare your output with the reference output in `scanning/DoubleLiterals*.tokens`