Scanner Implementation

Example: We have 6 different token classes

- `NUM     ["-" | "+"] Digit { Digit }`
- `LT      <`
- `LTEQ    <=`
- `EQ      ==`
- `NEQ     /=`
- `EOF     \uFFFF`

A valid token sequence would be:

```
12<117<===<-3==/==
```

Task: Implement a scanner, which prints out the sequence of token classes for a given input

```
NUM, <, NUM, <=, ==, <, -3, ==, /=
```


Scanner Framework (1)

A token class is represented by an integer constant:

```
interface Tokens {
    int EOF     = 0;
    int NUM     = EOF + 1;
    int LT      = NUM + 1;
    int LTEQ    = LT + 1;
    int EQ      = LTEQ + 1;
    int NEQ     = EQ + 1;
}
```


Scanner Framework (2)

```
public class Scanner implements Tokens {

  public Scanner(InputStream in) {
    this.in = in;
    this.buf = new StringBuffer();
    nextCh();
  }

  public int token;                     // the current token class
  public String chars;                  // token representation
  private InputStream in;               // the input stream
  private char ch;                      // the current character
  private final char EOF_CH = (char)-1;
  private StringBuffer buf;

  public void nextToken() {
    ...
  }

  private void nextCh() {
    try {
      ch = (char)in.read();
    } catch (IOException e) {
      error(e.toString());
    }
  }
```


Scanner Framework (3)

```
  void error(String message) {
    try {
      close();
    } catch (IOException e) {}
      throw new Error(message);
  }

  void close() throws IOException {
    in.close();
  }

  public String representation() {
    return tokenClass(token) +
      ((token == NUM) ? "(" + chars + ")" : "");
  }

  public static String tokenClass(int token) {
    switch (token) {
      case EOF:  return "<eof>";
      case NUM:  return "number";
      case LT:   return "<";
      case LTEQ: return "<=";
      case EQ:   return "==";
      case NEQ:  return "/=";
      default:   return "<unknown>";
    }
  }
}
```

Testbed for the Scanner

```
import java.io.*;

public class ScannerTest implements Tokens {

  static public void main(String[] args) {
    try {
      // create and initialize scanner
      Scanner s = new Scanner(new FileInputStream(args[0]));
      s.nextToken();
      // print out all tokens
      while (s.token != EOF) {
        System.out.println(s.representation());
        s.nextToken();
      }
    } catch (IOException e) {
      System.out.println(e.toString());
    }
  }
}
```


The First Solution

```
void nextToken() {
  switch (ch) {
    case EOF_CH:
      token = EOF; break;
    case '-': case '+': case '0': case '1': case '2': case '3':
    case '4': case '5': case '6': case '7': case '8': case '9':
      buf.setLength(0); buf.append(ch);
      if ((ch == '-') || (ch == '+')) {
        nextCh(); if ((ch < '0') || (ch > '9')) error("missing number");
      } else
        nextCh();
      while ((ch >= '0') && (ch <= '9')) {
        buf.append(ch); nextCh();
      }
      token = NUM; chars = buf.toString(); break;
    case '<':
      nextCh();
      if (ch == '=') {token = LTEQ; nextCh();} else token = LT;
      break;
    case '=':
      nextCh();
      if (ch == '=') {token = EQ; nextCh();} else error("missing =");
      break;
    case '/':
      nextCh();
      if (ch == '='){token = NEQ; nextCh();} else error("missing =");
      break;
    default:
      error("illegal character: " + ch);
  }
}
```

What about whitespaces?

Dealing with Whitespaces

```
void nextToken() {
  while (ch <= ' ')
    nextCh();
  switch (ch) {
    case EOF_CH:
      token = EOF; break;
    case '-': case '+': case '0': case '1': case '2': case '3':
    case '4': case '5': case '6': case '7': case '8': case '9':
      buf.setLength(0); buf.append(ch);
      if ((ch == '-') || (ch == '+')) {
        nextCh(); if ((ch < '0') || (ch > '9')) error("missing number");
      } else
        nextCh();
      while ((ch >= '0') && (ch <= '9')) {
        buf.append(ch); nextCh();
      }
      token = NUM; chars = buf.toString(); break;
    case '<':
      nextCh();
      if (ch == '=') {token = LTEQ; nextCh();} else token = LT;
      break;

    ...

    default:
      error("illegal character: " + ch);
  }
}
```

What about comments?

The Complete Solution

```
void nextToken() {
  while (ch <= ' ')
    nextCh();
  switch (ch) {
    case EOF_CH:
      token = EOF; break;
    ...
    case '/':
      nextCh();
      if (ch == '=') {
        token = NEQ; nextCh();
      } else if (ch == '*') {
        do {
          nextCh();
          while (ch == '*') {
            nextCh();
            if (ch == '/') {
              nextCh(); nextToken(); return;
            }
          }
        } while (ch != EOF_CH);
        error("unclosed comment");
      } else
        error("missing =");
      break;
    default:
      error("illegal character: " + ch);
  }
}
```

What about efficiency?

The Final Solution

```
void nextToken() {
  outer: while (true) {
    while (ch <= ' ')
      nextCh();
    buf.setLength(0);
    switch (ch) {
      case EOF_CH:
        token = EOF;
        return;
      case '-': case '+':
        buf.append(ch); nextCh();
        if ((ch < '0') || (ch > '9')) error("missing number");
      case '0': case '1': case '2': case '3': case '4':
      case '5': case '6': case '7': case '8': case '9':
        buf.append(ch); nextCh();
        while ((ch >= '0') && (ch <= '9')) {
          buf.append(ch); nextCh();
        }
        token = NUM; chars = buf.toString(); return;
      ...
      case '/':
        nextCh();
        if (ch == '=') {
          token = NEQ; nextCh();
        } else if (ch == '*') {
          do {
            nextCh();
            while (ch == '*') {
              nextCh(); if (ch == '/') {nextCh(); continue outer;}
            }
          } while (ch != EOF_CH);
          error("unclosed comment");
        } else
          error("missing =");
        return;
      default:
        error("illegal character: " + ch);
    }
  }
}
```