JavaCC
TokenManager Mini-Tutorial

Lexical States

- The JavaCC lexical specification is organized into a set of "lexical states"
- Each lexical state is named with an identifier
- There is a standard lexical state called DEFAULT
- Each lexical state contains an ordered list of regular expressions
- The token manager is in exactly one state at any moment and it only considers the regular expressions defined in this state for matching purposes

A token is matched as follows

- All regular expressions in the current lexical state are considered as **potential match candidates**.
- The token manager consumes the maximum number of characters from the input stream that match one of these regular expressions (the **longest possible match**)
- If there are multiple longest matches (of the same length), the regular expression that is matched is the one with the **earliest order of occurrence** in the grammar file
- After a match, one can specify **an action to be executed** as well as a new **lexical state to move to**.
- If a new lexical state is not specified, the token manager remains in the current state.

Regular expression kind

The regular expression kind specifies what to do when a regular expression has been successfully matched:

- SKIP: simply throw away the matched string (after executing any lexical action)
- MORE: continue (to whatever the next state is) taking the matched string along. This string will be a prefix of the new matched string.
- TOKEN: create a token using the matched string and send it to the parser (or any caller).
- SPECIAL_TOKEN: are like tokens, but they do not have significance during parsing. However, they are still passed on to the parser so that **parser actions can access them**. Special tokens are passed to the parser by linking them to neighbouring real tokens using the field "specialToken" in the Token class.

MORE: an example

```
<DEFAULT>

MORE : { "a" : S1 }

<S1>

MORE :

{

  "b"  : S2

}


<S2> TOKEN :

{

  "cd" : DEFAULT

}
```

SPECIAL TOKEN: an example

```
<DEFAULT>
SPECIAL_TOKEN : { "/*" :WITHIN-COMMENT }
TOKEN: {"ab"}
<WITHIN-COMMENT>
SPECIAL_TOKEN : { "*/" :DEFAULT }
MORE :
{
  <~[]>
}
```

```
/* bla bla */ ab
```

Variables/methods within lexical actions

- **StringBuffer image (READ/WRITE)**: (different from the "image" field of the matched token) is a StringBuffer variable that contains all the characters that have been matched since the last SKIP, TOKEN, or SPECIAL_TOKEN
- **int lengthOfMatch (READ ONLY):** this is the length of the current match (is not cumulative over MORE's)
- **int curLexState (READ ONLY):** This is the index of the current lexical state.
- **InputStream (READ ONLY):** This is the input stream. The stream is currently at the **last character consumed** for this match.
- **Token matchedToken (READ/WRITE):** may only be used in actions associated with TOKEN and SPECIAL_TOKEN regular expressions. This is set to be the token that will be returned to the parser
- **void SwitchTo(int)**

Variable image: an example

```
<DEFAULT>

MORE : { "a" : S1 }

<S1>

MORE :

{

  "b" {

          int i = image.length()-1; ①

          image.setCharAt(i, image.charAt(i).toUpperCase()); ②

       }  : S2

}
```

```
<S2> TOKEN :

{

  "cd" : DEFAULT ③

}
```

Variable matchedToken: an example

```
<DEFAULT>
MORE : { "a" : S1 }
<S1>
MORE :
{
  "b" {
          int i = image.length()-1; ①
          image.setCharAt(i, image.charAt(i).toUpperCase()); ②
      } : S2
}
```

```
<S2> TOKEN :
{
  "cd" {
          matchedToken.image = image.toSstring();
   }: DEFAULT ③
}
```

Mail Processing: lexical specification

```
TOKEN:
{
  <#EOL: "\n" | "\r" | "\r\n">
|
  <#TWOEOLS: <EOL> <EOL> >
|
  <#NOT_EOL: ~["\n","\r"]>
}

<DEFAULT>
SKIP:
{
    < <EOL> "*** EOOH ***" <EOL> > : MAILHEADER
|
   <~[]>
}
```

Mail Processing: lexical specification

```
<MAILHEADER>
SKIP:
{
   <HEADER_EOLS: <TWOEOLS>> : MAILBODY
    // We cannot have just a reference to a regular expression in a
    // lexical specification - we cannot simply have <TWOEOLS>.
|
  "Subject: " : MAILSUBJECT
|
   "From: " : MAILFROM
|
   "Date: " : MAILDATE
|
   <~[]>
}
```

Mail Processing: lexical specification

```
<MAILSUBJECT>
TOKEN:
{
    <SUBJECT: ( <NOT_EOL> )+>
}

<MAILSUBJECT>
SKIP:
{
  <SUB_EOL: <EOL>> : MAILHEADER
}
```

Mail Processing: lexical specification

```
<MAILFROM>
TOKEN:
{
    <FROM: ( <NOT_EOL> )+>
}

<MAILFROM>
SKIP:
{
  <FROM_EOL: <EOL>> : MAILHEADER
}
```

Mail Processing: lexical specification

```
<MAILDATE>
TOKEN:
{
     <DATE: ( <NOT_EOL> )+>
}

<MAILDATE>
SKIP:
{
   <DATE_EOL: <EOL>> : MAILHEADER
}
```

Mail Processing: lexical specification

```
<MAILBODY>
TOKEN:
{
     <BODY: ( ~[ "\n" | "\r" | "\u001f" ])* <EOL>>
   |
    <END: "\u001f"> : DEFAULT
}
```