Example of How to Write a Recursive Descent Parsing Method, and How Such a Method Creates a Parse Tree

- We will consider how to write a parsing method

  ```
  varDecl()
  ```

  for the nonterminal defined by this EBNF rule:

```
<varDecl> ::= int <singleVarDecl> { ,  <singleVarDecl> } ;
            | Scanner IDENTIFIER = new Scanner '(' System . in ')' ;
```

- We will also look at how the method reads a syntactically valid varDecl and outputs a sideways parse tree of its sequence of tokens.

```
<varDecl> ::= int <singleVarDecl> { ,  <singleVarDecl>} ;
            | Scanner IDENTIFIER = new Scanner '('  System . in  ')' ;
```

```
private static void varDecl() throws SourceFileErrorException
{
  TJ.output.printSymbol(NTvarDecl); TJ.output.incTreeDepth();

  if (getCurrentToken() == INT) {
    nextToken();
    singleVarDecl();
    while (getCurrentToken() == COMMA) {
      nextToken();
      singleVarDecl();
    }
    accept(SEMICOLON);
  }
  else if (getCurrentToken() == SCANNER) {
    nextToken();

    if (getCurrentToken() == IDENT) nextToken();
    else throw new SourceFileErrorException("Scanner name expected");

    accept(BECOMES); accept(NEW); accept(SCANNER);
    accept(LPAREN); accept(SYSTEM); accept(DOT);
    accept(IN);  accept(RPAREN); accept(SEMICOLON);
  }
  else throw new SourceFileErrorException("\"int\" or \"Scanner\" expected");

  TJ.output.decTreeDepth();
}
```

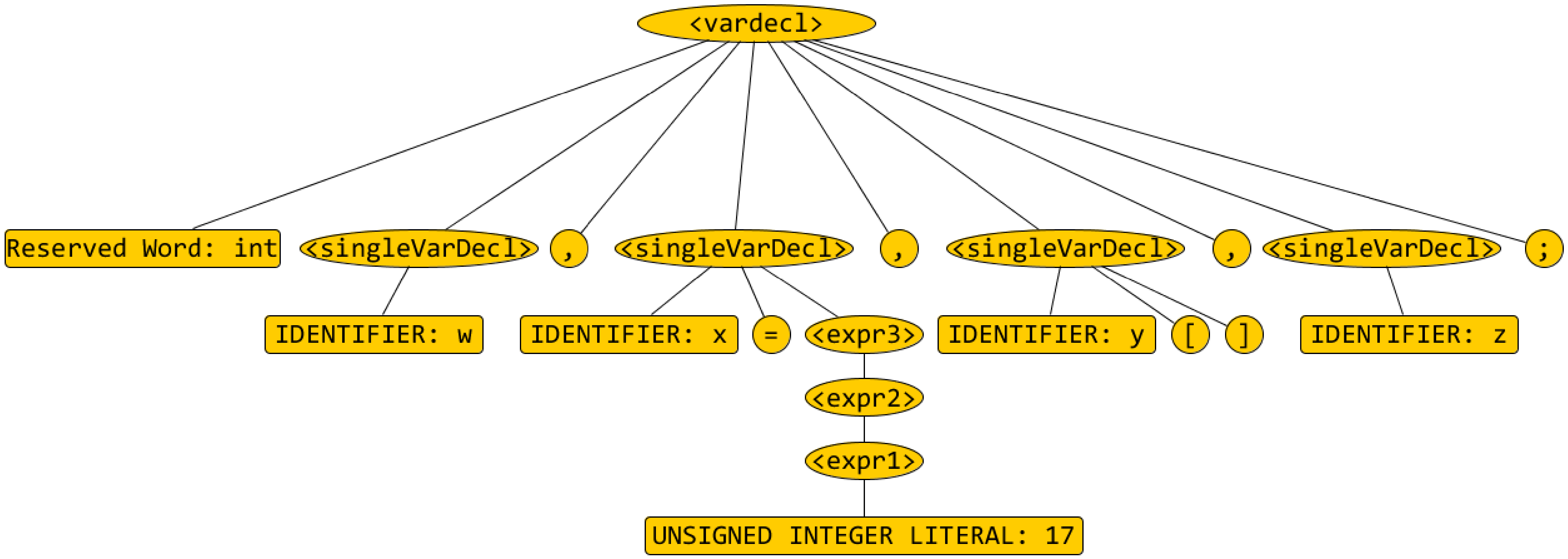
This is the parsing method for <varDecl>.

The following slides will show how this code can be derived ***from the EBNF rule that defines*** <varDecl>!

```
<varDecl> ::= int <singleVarDecl> { ,  <singleVarDecl>} ;
            |  Scanner IDENTIFIER = new Scanner '('  System . in  ')' ;

private static void varDecl() throws SourceFileErrorException
{
  TJ.output.printSymbol(NTvarDecl); TJ.output.incTreeDepth();

  if (getCurrentToken() == INT) {
    nextToken();
    singleVarDecl();
    while (getCurrentToken() == COMMA) {
      nextToken();
      singleVarDecl();
    }
    accept(SEMICOLON);
  }
  else if (getCurrentToken() == SCANNER) {
    nextToken();

    if (getCurrentToken() == IDENT) nextToken(); // better than accept(IDENT);
    else throw new SourceFileErrorException("Scanner name expected");

    accept(BECOMES); accept(NEW); accept(SCANNER);
    accept(LPAREN); accept(SYSTEM); accept(DOT);
    accept(IN);  accept(RPAREN); accept(SEMICOLON);
  }
  else throw new SourceFileErrorException("\"int\" or \"Scanner\" expected");

  TJ.output.decTreeDepth();
}
```

Parse tree of `int w, x = 17, y[], z;`
with root <varDecl>, based on the following EBNF rule:

```
<varDecl> ::= int <singleVarDecl> { ,  <singleVarDecl>} ;
            | Scanner IDENTIFIER = new Scanner '('  System . in  ')' ;
```



Creation of Sideways Parse Tree of `int w, x = 17, y[], z;` **with root** `<varDecl>`
Based on `<varDecl> ::=` **int** `<singleVarDecl> { , <singleVarDecl>} ;`
`|` **Scanner** `IDENTIFIER =` **new Scanner** `'('` **System** `.` **in** `')' ;`

```
<varDecl>
 Reserved Word: int
 <singleVarDecl>
  IDENTIFIER: w
  ... node has no more children
 ,
 <singleVarDecl>
  IDENTIFIER: x
  =
  <expr3>
   <expr2>
    <expr1>
     UNSIGNED INTEGER LITERAL: 17
     ... node has no more children
    ... node has no more children
   ... node has no more children
  ... node has no more children
 ,
 <singleVarDecl>
  IDENTIFIER: y
  [
  ]
  ... node has no more children
 ,
 <singleVarDecl>
  IDENTIFIER: z
  ... node has no more children
 ;
 ... node has no more children
```

On the left is the sideways parse tree, with root `<varDecl>`, of:

`int w, x = 17, y[], z;`

The following slides will show just ***how this tree is produced by execution of the method*** `varDecl` that was presented above!

Creation of Sideways Parse Tree of `int w, x = 17, y[], z;` **with root** `<varDecl>`

```
Based on  <varDecl> ::= int <singleVarDecl> { ,  <singleVarDecl>} ;
                      |  Scanner IDENTIFIER = new Scanner '(' System . in ')' ;
```

```
<varDecl>
 Reserved Word: int
 <singleVarDecl>
  IDENTIFIER: w
  ... node has no more children
 ,
 <singleVarDecl>
  IDENTIFIER: x
  =
  <expr3>
   <expr2>
    <expr1>
     UNSIGNED INTEGER LITERAL: 17
     ... node has no more children
    ... node has no more children
   ... node has no more children
  ... node has no more children
 ,
 <singleVarDecl>
  IDENTIFIER: y
  [
  ]
  ... node has no more children
 ,
 <singleVarDecl>
  IDENTIFIER: z
  ... node has no more children
 ;
 ... node has no more children
```

```java
private static void varDecl()
       throws SourceFileErrorException
{
  TJ.output.printSymbol(NTvarDecl);
  TJ.output.incTreeDepth();

  if (getCurrentToken() == INT) {
    nextToken();
    singleVarDecl();
    while (getCurrentToken() == COMMA) {
      nextToken();
      singleVarDecl();
    }
    accept(SEMICOLON);
  }
  else if (getCurrentToken()==SCANNER) {
                ...
  }
  else
    throw new SourceFileErrorException
      ("\"int\" or \"Scanner\" needed");

  TJ.output.decTreeDepth();
}
```

```
<varDecl> ::= int <singleVarDecl> { ,  <singleVarDecl>} ;
           |  Scanner IDENTIFIER = new Scanner '('  System . in  ')' ;

private static void varDecl() throws SourceFileErrorException
{
  TJ.output.printSymbol(NTvarDecl); TJ.output.incTreeDepth();

  if (getCurrentToken() == INT) {
    nextToken();
    singleVarDecl();
    while (getCurrentToken() == COMMA) {
      nextToken();
      singleVarDecl();
    }
    accept(SEMICOLON);
  }
  else if (getCurrentToken() == SCANNER) {
    nextToken();

    if (getCurrentToken() == IDENT) nextToken();
    else throw new SourceFileErrorException("Scanner name expected");

    accept(BECOMES); accept(NEW); accept(SCANNER);
    accept(LPAREN); accept(SYSTEM); accept(DOT);
    accept(IN);  accept(RPAREN); accept(SEMICOLON);
  }
  else throw new SourceFileErrorException("\"int\" or \"Scanner\" expected");

  TJ.output.decTreeDepth();
}
```