JavaScript is disabled on your browser.

- **Prev Class**
- **Next Class**

- Frames
- No Frames

- All Classes

- Summary:
- Nested |
- Field |
- Constr |
- Method

- Detail:
- Field |
- Constr |
- Method

jminusminus

Class JStringConcatenationOp

- java.lang.Object
  - jminusminus.JAST
    - jminusminus.JStatement
      - jminusminus.JExpression
        - jminusminus.JBinaryExpression
          - jminusminus.JStringConcatenationOp

-

```
 class JStringConcatenationOp
extends JBinaryExpression
```

The AST node for a string concatenation operation. Nodes of this type are not produced by the parser but by analysis of a + operation where the arguments are strings. Such operations are rewritten to be string concatenation operations.

- **Field Summary**

- **Fields inherited from class jminusminus.JBinaryExpression**

  `lhs, operator, rhs`

- **Fields inherited from class jminusminus.JExpression**

  `isStatementExpression, type`

- **Fields inherited from class jminusminus.JAST**

  `compilationUnit, line`

- **Constructor Summary**

Constructors

Constructor and Description

`JStringConcatenationOp(int line, JExpression lhs, JExpression rhs)` Construct an AST node for a string concatenation expression given its line number, and the lhs and rhs operands.

- ## Method Summary

Methods

Modifier and Type	Method and Description
`JExpression`	`analyze(Context context)` Analysis is simple here.
`void`	`codegen(CLEmitter output)` Code generation generates code for creating a StringBuilder atop the runtime stack, appending the operands (which might contain nested concatenations; these are handled by cascadingCodegen()), and then for converting the StringBuilder to a String.
`(package private) void`	`nestedCodegen(CLEmitter output)` Like a codegen() but we needn't (and shouldn't) create a StringBuilder nor convert the result to a String, as that will be done in a parent.

- ### Methods inherited from class jminusminus.JBinaryExpression

  `writeToStdOut`

- ### Methods inherited from class jminusminus.JExpression

  `codegen, isStatementExpression, type`

- ### Methods inherited from class jminusminus.JAST

  `line, partialCodegen`

- ### Methods inherited from class java.lang.Object

  `clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

- ## Constructor Detail

  - ### JStringConcatenationOp

```
publicJStringConcatenationOp(intline,
                              JExpressionlhs,
                              JExpressionrhs)
```

Construct an AST node for a string concatenation expression given its line number, and the lhs and rhs operands. An expression of this sort is created during the analysis of a (overloaded) + operation (and not by the Parser).

Parameters:

`line` - line in which the expression occurs in the source file.
`lhs` - lhs operand.
`rhs` - rhs operand.

- ## Method Detail

  - ### analyze

`publicJExpressionanalyze(Contextcontext)`

Analysis is simple here. The operands have already been analyzed (in JPlusOp) so we simply set the result type.

Specified by:
: `analyze` in class `JExpression`

Parameters:
: `context` - context in which names are resolved.

Returns:
: the analyzed (and possibly rewritten) AST subtree.

  - ### codegen

`publicvoidcodegen(CLEmitteroutput)`

Code generation generates code for creating a StringBuilder atop the runtime stack, appending the operands (which might contain nested concatenations; these are handled by cascadingCodegen()), and then for converting the StringBuilder to a String.

Specified by:
: `codegen` in class `JAST`

Parameters:
: `output` - the code emitter (basically an abstraction for producing the .class file).

  - ### nestedCodegen

`voidnestedCodegen(CLEmitteroutput)`

Like a codegen() but we needn't (and shouldn't) create a StringBuilder nor convert the result to a String, as that will be done in a parent.

Parameters:
: `output` - the code emitter (basically an abstraction for producing the .class file).