jminusminus

Class JMessageExpression

- java.lang.Object
  - jminusminus.JAST
    - jminusminus.JStatement
      - jminusminus.JExpression
        - jminusminus.JMessageExpression

.


```
class JMessageExpression
extends JExpression
```

The AST node for a message expression that has a target, optionally an ambiguous part, a message name, and zero or more actual arguments.

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

  `isStatementExpression, type`
- **Fields inherited from class jminusminus.JAST**

  `compilationUnit, line`
- **Constructor Summary**

Constructors

Modifier	Constructor and Description
protected	**JMessageExpression**(int line, JExpression target, AmbiguousName ambiguousPart, String messageName, ArrayList<JExpression> arguments) Construct an AST node for a message expression having an

	ambiguous part.
protected	**JMessageExpression**(int line, JExpression target, String messageName, ArrayList<JExpression> arguments) Construct an AST node for a message expression without an ambiguous part.

- **Method Summary**

Methods

Modifier and Type	Method and Description
JExpression	**analyze**(Context context) Analysis of a message expression involves: (1) reclassifying any ambiguous part, (2) analyzing and computing the types for the actual arguments, (3) determining the type we are currently in (for checking access), (4) analyzing the target and determining its type, (5) finding the appropriate Method, (6) checking accessibility, and (7) determining the result type.
void	**codegen**(CLEmitter output) Code generation for a message expression involves generating code for loading the target onto the stack, generating code to load the actual arguments onto the stack, and then invoking the named Method.
void	**codegen**(CLEmitter output, String targetLabel, boolean onTrue) The semantics of j-- require that we implement short-circuiting branching in implementing message expressions.
void	**writeToStdout**(PrettyPrinter p) Write the information pertaining to this AST to STDOUT.

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

  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**

  - **JMessageExpression**

```
protectedJMessageExpression(intline,
                JExpressiontarget,
                Stringmessage Name,
                ArrayList<JExpression>arguments)
```

Construct an AST node for a message expression without an ambiguous part.

Parameters:

`line` - line in which the expression occurs in the source file.
`target` - the target expression.
`messageName` - the message name.
`arguments` - the ambiguousPart arguments.

- **JMessageExpression**

```
protectedJMessageExpression(intline,
                  JExpressiontarget,
                  AmbiguousNameambiguousPart,
                  StringmessageName,
                  ArrayList<JExpression>arguments)
```

Construct an AST node for a message expression having an ambiguous part.

Parameters:

`line` - line in which the expression occurs in the source file.
`target` - the target expression.
`ambiguousPart` - the ambiguous part.
`messageName` - the message name.
`arguments` - the arguments.

- **Method Detail**

- **analyze**

```
publicJExpressionanalyze(Contextcontext)
```


Analysis of a message expression involves: (1) reclassifying any ambiguous part, (2) analyzing and computing the types for the actual arguments, (3) determining the type we are currently in (for checking access), (4) analyzing the target and determining its type, (5) finding the appropriate Method, (6) checking accessibility, and (7) determining 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 for a message expression involves generating code for loading the target onto the stack, generating code to load the actual arguments onto the stack, and then invoking the named Method. Notice that if this is a statement expression (as marked by a parent JStatementExpression) then we also generate code for popping the stacked value for any non-void invocation.

Specified by:

`codegen` in class `JAST`

Parameters:

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

- **codegen**

```
publicvoidcodegen(CLEmitteroutput,
```

```
                Stringtargetlabel,
                booleanonTrue)
```

The semantics of j-- require that we implement short-circuiting branching in implementing message expressions.

Overrides:

codegen in class JExpression

Parameters:

output - the code emitter (basically an abstraction for producing the .class file).
targetLabel - the label to which we should branch.
onTrue - do we branch on true?

- **writeToStdOut**

```
publicvoidwriteToStdOut(PrettyPrinterp)
```

Description copied from class: JAST

Write the information pertaining to this AST to STDOUT.

Specified by:

writeToStdOut in class JAST

Parameters:

p - for pretty printing with indentation.