```java
/* Generated By:JavaCC: Do not edit this line. JavaCCParser.java */
package jminusminus;

import java.util.ArrayList;

/**
 * Parser generated by JavaCC. It parses a j-- compilation unit
 * (program file), taking tokens from the scanner (also generated by
 * JavaCC), and produces an abstract syntax tree (AST) for it.
 */

class JavaCCParser implements JavaCCParserConstants {
    /** Whether a parser error has been found. */
    private boolean errorHasOccurred;

    /** Name of the file that is parsed. */
    private String fileName;

    /**
     * Pull out the ambiguous part of a name and return it.
     *
     * @param name with an ambiguos part (possibly).
     * @return ambiguous part or null.
     */

    private AmbiguousName ambiguousPart( TypeName name ) {
        String qualifiedName = name.toString();
        int lastDotIndex = qualifiedName.lastIndexOf( '.' );
        return lastDotIndex == -1
            ? null // It was a simple name
            : new AmbiguousName( name.line(),
                qualifiedName.substring( 0, lastDotIndex ) );
    }

    /**
     * Report a syntax error.
     *
     * @param message message identifying the error.
     * @param args related values.
     */

    private void reportParserError( String message, Object... args ) {
        errorHasOccurred = true;
        System.err.printf( "%s:%d: ", fileName, token.beginLine );
        System.err.printf( message, args );
        System.err.println();
    }

    /**
     * Recover from the parser error that occurred by skipping to
     * any of the specified tokens.
     *
     * Current error recovery mechanism is rather simple-minded and is
     * based on skipping all the tokens until a SEMI or an EOF is
     * encountered. This scheme can be enhanced by passing in the
     * FOLLOW-SET of the non-terminal at hand.
     *
     * @param skipTo array of tokens that we could skip to.
     * @param e exception that is raised by JavaCC in the event
     * of a parser error.
     */

    private void recoverFromError( int[] skipTo, ParseException e ) {
        // Get the possible expected tokens
        StringBuffer expected = new StringBuffer();
        for ( int i = 0; i < e.expectedTokenSequences.length; i++ ) {
```

```
            for ( int j = 0; j < e.expectedTokenSequences[ i ].length;
                j++ ) {
                expected.append( "\n" );
                expected.append( "    " );
                expected.append( tokenImage[
                    e.expectedTokenSequences[ i ][ j ] ] );
                expected.append( "..." );
            }
        }

        // Print error message
        if ( e.expectedTokenSequences.length == 1 ) {
            reportParserError( "\"%s\" found where %s sought",
                getToken( 1 ), expected );
        }
        else {
            reportParserError( "\"%s\" found where one of %s sought",
                getToken( 1 ), expected );
        }

        // Recover
        boolean loop = true;
        do {
            token = getNextToken();
            for ( int i = 0; i < skipTo.length; i++ ) {
                if ( token.kind == skipTo[ i ] ) {
                    loop = false;
                    break;
                }
            }
        } while ( loop );
    }

    /**
     * Set the name of the file that is being parsed.
     *
     * @param fileName name of the file.
     */

    public void fileName( String fileName ) {
        this.fileName = fileName;
    }

    /**
     * Has a parser error occurred up to now?
     *
     * @return true or false.
     */

    public boolean errorHasOccurred() {
        return errorHasOccurred;
    }

  final public JCompilationUnit compilationUnit() throws ParseException {
    int line = 0;
    TypeName packageName = null; // Default
    TypeName anImport = null;
    ArrayList<TypeName> imports =
        new ArrayList<TypeName>();
    JAST aTypeDeclaration = null;
    ArrayList<JAST> typeDeclarations = new ArrayList<JAST>();
    try {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case PACKAGE:
        jj_consume_token(PACKAGE);
                        line = token.beginLine;
        packageName = qualifiedIdentifier();
        jj_consume_token(SEMI);
        break;
```

```
      default:
        jj_la1[0] = jj_gen;
        ;
      }
      label_1:
      while (true) {
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case IMPORT:
          ;
          break;
        default:
          jj_la1[1] = jj_gen;
          break label_1;
        }
        jj_consume_token(IMPORT);
                      line = line == 0 ? token.beginLine : line;
        anImport = qualifiedIdentifier();
              imports.add( anImport );
        jj_consume_token(SEMI);
      }
      label_2:
      while (true) {
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case ABSTRACT:
        case CLASS:
        case PRIVATE:
        case PROTECTED:
        case PUBLIC:
        case STATIC:
          ;
          break;
        default:
          jj_la1[2] = jj_gen;
          break label_2;
        }
        aTypeDeclaration = typeDeclaration();
                line = line == 0 ? aTypeDeclaration.line() : line;
                typeDeclarations.add( aTypeDeclaration );
      }
      jj_consume_token(0);
                line = line == 0 ? token.beginLine : line;
    } catch (ParseException e) {
        recoverFromError( new int[] { SEMI, EOF }, e );
    }
        {if (true) return new JCompilationUnit( fileName, line,
            packageName, imports, typeDeclarations );}
    throw new Error("Missing return statement in function");
  }

  final private TypeName qualifiedIdentifier() throws ParseException {
    int line = 0;
    String qualifiedIdentifier = "";
    try {
      jj_consume_token(IDENTIFIER);
            line = token.beginLine;
            qualifiedIdentifier = token.image;
      label_3:
      while (true) {
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case DOT:
          ;
          break;
        default:
          jj_la1[3] = jj_gen;
          break label_3;
        }
        jj_consume_token(DOT);
        jj_consume_token(IDENTIFIER);
              qualifiedIdentifier += "." + token.image;
```

```
205       }
206     } catch (ParseException e) {
207         recoverFromError( new int[] { SEMI, EOF }, e );
208     }
209       {if (true) return
210          new TypeName( line, qualifiedIdentifier );}
211     throw new Error("Missing return statement in function");
212   }
213
214   final private JAST typeDeclaration() throws ParseException {
215     ArrayList<String> mods = null;
216     JAST classDeclaration = null;
217     try {
218       mods = modifiers();
219       classDeclaration = classDeclaration(mods);
220     } catch (ParseException e) {
221         recoverFromError( new int[] { SEMI, EOF }, e );
222     }
223       {if (true) return classDeclaration;}
224     throw new Error("Missing return statement in function");
225   }
226
227   final private ArrayList<String> modifiers() throws ParseException {
228     ArrayList<String> mods = new ArrayList<String>();
229     boolean scannedPUBLIC = false;
230     boolean scannedPROTECTED = false;
231     boolean scannedPRIVATE = false;
232     boolean scannedSTATIC = false;
233     boolean scannedABSTRACT = false;
234     try {
235       label_4:
236       while (true) {
237         switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
238         case ABSTRACT:
239         case PRIVATE:
240         case PROTECTED:
241         case PUBLIC:
242         case STATIC:
243           ;
244           break;
245         default:
246           jj_la1[4] = jj_gen;
247           break label_4;
248         }
249         switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
250         case PUBLIC:
251           jj_consume_token(PUBLIC);
252             mods.add( "public" );
253             if ( scannedPUBLIC ) {
254                 reportParserError( "Repeated modifier: public" );
255             }
256             if ( scannedPROTECTED || scannedPRIVATE ) {
257                 reportParserError( "Access conflict in modifiers" );
258             }
259             scannedPUBLIC = true;
260           break;
261         case PROTECTED:
262           jj_consume_token(PROTECTED);
263             mods.add( "protected" );
264             if ( scannedPROTECTED ) {
265                 reportParserError( "Repeated modifier: protected" );
266             }
267             if ( scannedPUBLIC || scannedPRIVATE ) {
268                 reportParserError( "Access conflict in modifiers" );
269             }
270             scannedPROTECTED = true;
271           break;
272         case PRIVATE:
273           jj_consume_token(PRIVATE);
```

```
274             mods.add( "private" );
275             if ( scannedPRIVATE ) {
276                 reportParserError( "Repeated modifier: private" );
277             }
278             if ( scannedPUBLIC || scannedPROTECTED ) {
279                 reportParserError( "Access conflict in modifiers" );
280             }
281             scannedPRIVATE = true;
282           break;
283         case STATIC:
284           jj_consume_token(STATIC);
285             mods.add( "static" );
286             if ( scannedSTATIC ) {
287                 reportParserError( "Repeated modifier: static" );
288             }
289             scannedSTATIC = true;
290           break;
291         case ABSTRACT:
292           jj_consume_token(ABSTRACT);
293             mods.add( "abstract" );
294             if ( scannedABSTRACT ) {
295                 reportParserError( "Repeated modifier: abstract" );
296             }
297             scannedABSTRACT = true;
298           break;
299         default:
300           jj_la1[5] = jj_gen;
301           jj_consume_token(-1);
302           throw new ParseException();
303         }
304       }
305     } catch (ParseException e) {
306         recoverFromError( new int[] { SEMI, EOF }, e );
307     }
308       {if (true) return mods;}
309     throw new Error("Missing return statement in function");
310   }
311
312   final private JClassDeclaration classDeclaration(ArrayList<String> mods) throws
ParseException {
313     int line = 0;
314     String name = "";
315     Type superClass = Type.OBJECT;
316     ArrayList<JMember> classBody = null;
317     try {
318       jj_consume_token(CLASS);
319                   line = token.beginLine;
320       jj_consume_token(IDENTIFIER);
321                        name = token.image;
322       switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
323       case EXTENDS:
324         jj_consume_token(EXTENDS);
325         superClass = qualifiedIdentifier();
326         break;
327       default:
328         jj_la1[6] = jj_gen;
329         ;
330       }
331       classBody = classBody();
332     } catch (ParseException e) {
333         recoverFromError( new int[] { SEMI, EOF }, e );
334     }
335         {if (true) return new JClassDeclaration( line, mods,
336             name, superClass, classBody );}
337     throw new Error("Missing return statement in function");
338   }
339
340   final private ArrayList<JMember> classBody() throws ParseException {
341     ArrayList<String> mods = null;
```

```
342     JMember aMember = null;
343     ArrayList<JMember> members = new ArrayList<JMember>();
344     try {
345       jj_consume_token(LCURLY);
346       label_5:
347       while (true) {
348         switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
349         case ABSTRACT:
350         case BOOLEAN:
351         case CHAR:
352         case INT:
353         case PRIVATE:
354         case PROTECTED:
355         case PUBLIC:
356         case STATIC:
357         case VOID:
358         case IDENTIFIER:
359           ;
360           break;
361         default:
362           jj_la1[7] = jj_gen;
363           break label_5;
364         }
365         mods = modifiers();
366         aMember = memberDecl(mods);
367                                             members.add( aMember );
368       }
369       jj_consume_token(RCURLY);
370     } catch (ParseException e) {
371       recoverFromError(new int[] {SEMI, EOF}, e);
372     }
373       {if (true) return members;}
374     throw new Error("Missing return statement in function");
375   }
376
377   final private JMember memberDecl(ArrayList<String>  mods) throws ParseException
{
378     int line = 0;
379     Type type = null;
380     String name = "";
381     ArrayList<JFormalParameter> params = null;
382     JBlock body = null;
383     ArrayList<JVariableDeclarator> variableDeclarators = null;
384     JMember memberDecl = null;
385     try {
386       if (jj_2_1(2147483647)) {
387         jj_consume_token(IDENTIFIER);
388                 line = token.beginLine;
389                 name = token.image;
390         params = formalParameters();
391         body = block();
392                 memberDecl =
393                 new JConstructorDeclaration( line, mods,
394                                                 name, params, body );
395       } else if (jj_2_2(2147483647)) {
396         switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
397         case VOID:
398           jj_consume_token(VOID);
399                         type = Type.VOID;
400           break;
401         case BOOLEAN:
402         case CHAR:
403         case INT:
404         case IDENTIFIER:
405           type = type();
406           break;
407         default:
408           jj_la1[8] = jj_gen;
409           jj_consume_token(-1);
```

```
410           throw new ParseException();
411         }
412               line = token.beginLine;
413         jj_consume_token(IDENTIFIER);
414                             name = token.image;
415         params = formalParameters();
416         switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
417         case LCURLY:
418           body = block();
419           break;
420         case SEMI:
421           jj_consume_token(SEMI);
422           break;
423         default:
424           jj_la1[9] = jj_gen;
425           jj_consume_token(-1);
426           throw new ParseException();
427         }
428                 memberDecl =
429                    new JMethodDeclaration( line, mods, name,
430                                            type, params, body );
431       } else {
432         switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
433         case BOOLEAN:
434         case CHAR:
435         case INT:
436         case IDENTIFIER:
437           type = type();
438                             line = token.beginLine;
439           variableDeclarators = variableDeclarators(type);
440           jj_consume_token(SEMI);
441                 memberDecl = new JFieldDeclaration( line, mods,
442                     variableDeclarators );
443           break;
444         default:
445           jj_la1[10] = jj_gen;
446           jj_consume_token(-1);
447           throw new ParseException();
448         }
449       }
450     } catch (ParseException e) {
451       recoverFromError( new int[] { SEMI, EOF }, e );
452     }
453       {if (true) return memberDecl;}
454     throw new Error("Missing return statement in function");
455   }
456
457   final private JBlock block() throws ParseException {
458     int line = 0;
459     JStatement aStatement = null;
460     ArrayList<JStatement> statements = new ArrayList<JStatement>();
461     try {
462       jj_consume_token(LCURLY);
463                     line = token.beginLine;
464       label_6:
465       while (true) {
466         switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
467         case BOOLEAN:
468         case CHAR:
469         case FALSE:
470         case IF:
471         case INT:
472         case NEW:
473         case NULL:
474         case RETURN:
475         case SUPER:
476         case THIS:
477         case TRUE:
478         case WHILE:
```

```
479         case INC:
480         case LNOT:
481         case MINUS:
482         case LPAREN:
483         case LCURLY:
484         case SEMI:
485         case IDENTIFIER:
486         case INT_LITERAL:
487         case CHAR_LITERAL:
488         case STRING_LITERAL:
489           ;
490           break;
491         default:
492           jj_la1[11] = jj_gen;
493           break label_6;
494         }
495         aStatement = blockStatement();
496               statements.add( aStatement );
497       }
498       jj_consume_token(RCURLY);
499     } catch (ParseException e) {
500         recoverFromError( new int[] { SEMI, EOF }, e );
501     }
502       {if (true) return new JBlock( line, statements );}
503     throw new Error("Missing return statement in function");
504   }
505
506   final private JStatement blockStatement() throws ParseException {
507     JStatement statement = null;
508     try {
509       if ((jj_2_3(2147483647))) {
510         statement = localVariableDeclarationStatement();
511       } else {
512         switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
513         case FALSE:
514         case IF:
515         case NEW:
516         case NULL:
517         case RETURN:
518         case SUPER:
519         case THIS:
520         case TRUE:
521         case WHILE:
522         case INC:
523         case LNOT:
524         case MINUS:
525         case LPAREN:
526         case LCURLY:
527         case SEMI:
528         case IDENTIFIER:
529         case INT_LITERAL:
530         case CHAR_LITERAL:
531         case STRING_LITERAL:
532           statement = statement();
533           break;
534         default:
535           jj_la1[12] = jj_gen;
536           jj_consume_token(-1);
537           throw new ParseException();
538         }
539       }
540     } catch (ParseException e) {
541         recoverFromError( new int[] { SEMI, EOF }, e );
542     }
543       {if (true) return statement;}
544     throw new Error("Missing return statement in function");
545   }
546
547   final private JStatement statement() throws ParseException {
```

```
548     int line = 0;
549     JStatement statement = null;
550     JExpression test = null;
551     JStatement consequent = null;
552     JStatement alternate = null;
553     JStatement body = null;
554     JExpression expr = null;
555     try {
556       switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
557       case LCURLY:
558         statement = block();
559         break;
560       case IF:
561         jj_consume_token(IF);
562                line = token.beginLine;
563         test = parExpression();
564         consequent = statement();
565         if (jj_2_4(2147483647)) {
566           jj_consume_token(ELSE);
567           alternate = statement();
568         } else {
569           ;
570         }
571           statement =
572             new JIfStatement( line, test, consequent, alternate );
573         break;
574       case WHILE:
575         jj_consume_token(WHILE);
576                  line = token.beginLine;
577         test = parExpression();
578         body = statement();
579           statement = new JWhileStatement( line, test, body );
580         break;
581       case RETURN:
582         jj_consume_token(RETURN);
583                  line = token.beginLine;
584         switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
585         case FALSE:
586         case NEW:
587         case NULL:
588         case SUPER:
589         case THIS:
590         case TRUE:
591         case INC:
592         case LNOT:
593         case MINUS:
594         case LPAREN:
595         case IDENTIFIER:
596         case INT_LITERAL:
597         case CHAR_LITERAL:
598         case STRING_LITERAL:
599           expr = expression();
600           break;
601         default:
602           jj_la1[13] = jj_gen;
603           ;
604         }
605         jj_consume_token(SEMI);
606           statement = new JReturnStatement( line, expr );
607         break;
608       case SEMI:
609         jj_consume_token(SEMI);
610           statement = new JEmptyStatement( line );
611         break;
612       case FALSE:
613       case NEW:
614       case NULL:
615       case SUPER:
616       case THIS:
```

```
617       case TRUE:
618       case INC:
619       case LNOT:
620       case MINUS:
621       case LPAREN:
622       case IDENTIFIER:
623       case INT_LITERAL:
624       case CHAR_LITERAL:
625       case STRING_LITERAL:
626         // Must be a statementExpression
627                 statement = statementExpression();
628         jj_consume_token(SEMI);
629         break;
630       default:
631         jj_la1[14] = jj_gen;
632         jj_consume_token(-1);
633         throw new ParseException();
634       }
635     } catch (ParseException e) {
636         recoverFromError( new int[] { SEMI, EOF }, e );
637     }
638       {if (true) return statement;}
639     throw new Error("Missing return statement in function");
640   }
641
642   final private ArrayList<JFormalParameter> formalParameters() throws
ParseException {
643     ArrayList<JFormalParameter> parameters =
644         new ArrayList<JFormalParameter>();
645     JFormalParameter aParameter = null;
646     try {
647       jj_consume_token(LPAREN);
648       switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
649       case BOOLEAN:
650       case CHAR:
651       case INT:
652       case IDENTIFIER:
653         aParameter = formalParameter();
654               parameters.add( aParameter );
655         label_7:
656         while (true) {
657           switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
658           case COMMA:
659             ;
660             break;
661           default:
662             jj_la1[15] = jj_gen;
663             break label_7;
664           }
665           jj_consume_token(COMMA);
666           aParameter = formalParameter();
667                   parameters.add( aParameter );
668         }
669         break;
670       default:
671         jj_la1[16] = jj_gen;
672         ;
673       }
674       jj_consume_token(RPAREN);
675     } catch (ParseException e) {
676         recoverFromError( new int[] { SEMI, EOF }, e );
677     }
678       {if (true) return parameters;}
679     throw new Error("Missing return statement in function");
680   }
681
682   final private JFormalParameter formalParameter() throws ParseException {
683     int line = 0;
684     Type type = null;
```

```
685     String name = "";
686     try {
687       type = type();
688                         line = token.beginLine;
689       jj_consume_token(IDENTIFIER);
690                        name = token.image;
691     } catch (ParseException e) {
692         recoverFromError( new int[] { SEMI, EOF }, e );
693     }
694       {if (true) return new JFormalParameter( line, name, type );}
695     throw new Error("Missing return statement in function");
696   }
697
698   final private JExpression parExpression() throws ParseException {
699     JExpression expr = null;
700     try {
701       jj_consume_token(LPAREN);
702       expr = expression();
703       jj_consume_token(RPAREN);
704     } catch (ParseException e) {
705         recoverFromError( new int[] { SEMI, EOF }, e );
706     }
707       {if (true) return expr;}
708     throw new Error("Missing return statement in function");
709   }
710
711   final private JVariableDeclaration localVariableDeclarationStatement() throws
ParseException {
712     int line = 0;
713     Type type = null;
714     ArrayList<JVariableDeclarator> vdecls = null;
715     ArrayList<String> mods = new ArrayList<String>();
716     try {
717       type = type();
718                   line = token.beginLine;
719       vdecls = variableDeclarators(type);
720       jj_consume_token(SEMI);
721     } catch (ParseException e) {
722         recoverFromError( new int[] { SEMI, EOF }, e );
723     }
724       {if (true) return new JVariableDeclaration( line, mods, vdecls );}
725     throw new Error("Missing return statement in function");
726   }
727
728   final private ArrayList<JVariableDeclarator> variableDeclarators(Type type)
throws ParseException {
729     JVariableDeclarator aVariableDeclarator = null;
730     ArrayList<JVariableDeclarator> variableDeclarators =
731         new ArrayList<JVariableDeclarator>();
732     try {
733       aVariableDeclarator = variableDeclarator(type);
734           variableDeclarators.add( aVariableDeclarator );
735       label_8:
736       while (true) {
737         switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
738         case COMMA:
739           ;
740           break;
741         default:
742           jj_la1[17] = jj_gen;
743           break label_8;
744         }
745         jj_consume_token(COMMA);
746         aVariableDeclarator = variableDeclarator(type);
747               variableDeclarators.add( aVariableDeclarator );
748       }
749     } catch (ParseException e) {
750         recoverFromError( new int[] { SEMI, EOF }, e );
751     }
```

```
752       {if (true) return variableDeclarators;}
753     throw new Error("Missing return statement in function");
754   }
755
756   final private JVariableDeclarator variableDeclarator(Type type) throws
ParseException {
757     int line = 0;
758     JExpression initial = null;
759     String name = "";
760     try {
761       jj_consume_token(IDENTIFIER);
762                        line = token.beginLine; name = token.image;
763       switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
764       case ASSIGN:
765         jj_consume_token(ASSIGN);
766         initial = variableInitializer(type);
767         break;
768       default:
769         jj_la1[18] = jj_gen;
770         ;
771       }
772     } catch (ParseException e) {
773         recoverFromError( new int[] { SEMI, EOF }, e );
774     }
775       {if (true) return new JVariableDeclarator( line, name, type, initial );}
776     throw new Error("Missing return statement in function");
777   }
778
779   final private JExpression variableInitializer(Type expected) throws
ParseException {
780     JExpression initializer = null;
781     try {
782       switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
783       case LCURLY:
784         initializer = arrayInitializer(expected);
785         break;
786       case FALSE:
787       case NEW:
788       case NULL:
789       case SUPER:
790       case THIS:
791       case TRUE:
792       case INC:
793       case LNOT:
794       case MINUS:
795       case LPAREN:
796       case IDENTIFIER:
797       case INT_LITERAL:
798       case CHAR_LITERAL:
799       case STRING_LITERAL:
800         initializer = expression();
801         break;
802       default:
803         jj_la1[19] = jj_gen;
804         jj_consume_token(-1);
805         throw new ParseException();
806       }
807     } catch (ParseException e) {
808         recoverFromError( new int[] { SEMI, EOF }, e );
809     }
810       {if (true) return initializer;}
811     throw new Error("Missing return statement in function");
812   }
813
814   final private JArrayInitializer arrayInitializer(Type expected) throws
ParseException {
815     int line = 0;
816     ArrayList<JExpression> initials = new ArrayList<JExpression>();
817     JExpression anInitializer = null;
```

```
    try {
      jj_consume_token(LCURLY);
                    line = token.beginLine;
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case FALSE:
      case NEW:
      case NULL:
      case SUPER:
      case THIS:
      case TRUE:
      case INC:
      case LNOT:
      case MINUS:
      case LPAREN:
      case LCURLY:
      case IDENTIFIER:
      case INT_LITERAL:
      case CHAR_LITERAL:
      case STRING_LITERAL:
        anInitializer = variableInitializer(expected.componentType());
              initials.add( anInitializer );
        label_9:
        while (true) {
          switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
          case COMMA:
            ;
            break;
          default:
            jj_la1[20] = jj_gen;
            break label_9;
          }
          jj_consume_token(COMMA);
          anInitializer = variableInitializer(expected.componentType());
                initials.add( anInitializer );
        }
        break;
      default:
        jj_la1[21] = jj_gen;
        ;
      }
      jj_consume_token(RCURLY);
    } catch (ParseException e) {
        recoverFromError( new int[] { SEMI, EOF }, e );
    }
      {if (true) return new JArrayInitializer( line, expected, initials );}
    throw new Error("Missing return statement in function");
  }

  final private ArrayList<JExpression> arguments() throws ParseException {
    ArrayList<JExpression> args = new ArrayList<JExpression>();
    JExpression anExpression = null;
    try {
      jj_consume_token(LPAREN);
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case FALSE:
      case NEW:
      case NULL:
      case SUPER:
      case THIS:
      case TRUE:
      case INC:
      case LNOT:
      case MINUS:
      case LPAREN:
      case IDENTIFIER:
      case INT_LITERAL:
      case CHAR_LITERAL:
      case STRING_LITERAL:
        anExpression = expression();
```

```
                                        args.add( anExpression );
      label_10:
      while (true) {
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case COMMA:
          ;
          break;
        default:
          jj_la1[22] = jj_gen;
          break label_10;
        }
        jj_consume_token(COMMA);
        anExpression = expression();
                args.add( anExpression );
      }
      break;
    default:
      jj_la1[23] = jj_gen;
      ;
    }
    jj_consume_token(RPAREN);
  } catch (ParseException e) {
      recoverFromError( new int[] { SEMI, EOF }, e );
  }
    {if (true) return args;}
  throw new Error("Missing return statement in function");
}

final private Type type() throws ParseException {
  Type type = null;
  try {
    if (jj_2_5(2147483647)) {
      type = referenceType();
    } else {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case BOOLEAN:
      case CHAR:
      case INT:
        type = basicType();
        break;
      default:
        jj_la1[24] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
    }
  } catch (ParseException e) {
     recoverFromError( new int[] { SEMI, EOF }, e );
  }
    {if (true) return type;}
  throw new Error("Missing return statement in function");
}

final private Type basicType() throws ParseException {
  Type type = Type.ANY;
  try {
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case BOOLEAN:
      jj_consume_token(BOOLEAN);
                  type = Type.BOOLEAN;
      break;
    case CHAR:
      jj_consume_token(CHAR);
                type = Type.CHAR;
      break;
    case INT:
      jj_consume_token(INT);
              type = Type.INT;
      break;
```

```
      default:
        jj_la1[25] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
    } catch (ParseException e) {
        recoverFromError( new int[] { SEMI, EOF }, e );
    }
      {if (true) return type;}
    throw new Error("Missing return statement in function");
  }

  final private Type referenceType() throws ParseException {
    Type type = Type.ANY;
    try {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case BOOLEAN:
      case CHAR:
      case INT:
        type = basicType();
        jj_consume_token(LBRACK);
        jj_consume_token(RBRACK);
                              type = new ArrayTypeName( type );
        label_11:
        while (true) {
          switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
          case LBRACK:
            ;
            break;
          default:
            jj_la1[26] = jj_gen;
            break label_11;
          }
          jj_consume_token(LBRACK);
          jj_consume_token(RBRACK);
              type = new ArrayTypeName( type );
        }
        break;
      case IDENTIFIER:
        type = qualifiedIdentifier();
        label_12:
        while (true) {
          switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
          case LBRACK:
            ;
            break;
          default:
            jj_la1[27] = jj_gen;
            break label_12;
          }
          jj_consume_token(LBRACK);
          jj_consume_token(RBRACK);
              type = new ArrayTypeName( type );
        }
        break;
      default:
        jj_la1[28] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
    } catch (ParseException e) {
        recoverFromError( new int[] { SEMI, EOF }, e );
    }
      {if (true) return type;}
    throw new Error("Missing return statement in function");
  }

  final private JStatement statementExpression() throws ParseException {
    int line = 0;
```

```
1025    JExpression expr = null;
1026    try {
1027      expr = expression();
1028            line = expr.line();
1029            if ( expr instanceof JAssignment
1030              || expr instanceof JPreIncrementOp
1031              || expr instanceof JPostDecrementOp
1032              || expr instanceof JMessageExpression
1033              || expr instanceof JSuperConstruction
1034              || expr instanceof JThisConstruction
1035              || expr instanceof JNewOp
1036              || expr instanceof JNewArrayOp ) {
1037                // So as not to save on stack
1038                expr.isStatementExpression = true;
1039            }
1040            else {
1041                reportParserError( "Invalid statement expression; " +
1042                    "it does not have a side-effect" );
1043            }
1044    } catch (ParseException e) {
1045        recoverFromError( new int[] { SEMI, EOF }, e );
1046    }
1047      {if (true) return new JStatementExpression( line, expr );}
1048    throw new Error("Missing return statement in function");
1049  }
1050
1051  final private JExpression expression() throws ParseException {
1052    JExpression expr = null;
1053    try {
1054      expr = assignmentExpression();
1055    } catch (ParseException e) {
1056        recoverFromError( new int[] { SEMI, EOF }, e );
1057    }
1058      {if (true) return expr;}
1059    throw new Error("Missing return statement in function");
1060  }
1061
1062  final private JExpression assignmentExpression() throws ParseException {
1063    int line = 0;
1064    JExpression lhs = null, rhs = null;
1065    try {
1066      lhs = conditionalAndExpression();
1067                                          line = lhs.line();
1068      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
1069      case ASSIGN:
1070      case PLUS_ASSIGN:
1071        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
1072        case ASSIGN:
1073          jj_consume_token(ASSIGN);
1074          rhs = assignmentExpression();
1075              lhs = new JAssignOp( line, lhs, rhs );
1076          break;
1077        case PLUS_ASSIGN:
1078          jj_consume_token(PLUS_ASSIGN);
1079          rhs = assignmentExpression();
1080              lhs = new JPlusAssignOp( line, lhs, rhs );
1081          break;
1082        default:
1083          jj_la1[29] = jj_gen;
1084          jj_consume_token(-1);
1085          throw new ParseException();
1086        }
1087        break;
1088      default:
1089        jj_la1[30] = jj_gen;
1090        ;
1091      }
1092    } catch (ParseException e) {
1093        recoverFromError( new int[] { SEMI, EOF }, e );
```

```
    }
      {if (true) return lhs;}
    throw new Error("Missing return statement in function");
  }

  final private JExpression conditionalAndExpression() throws ParseException {
    int line = 0;
    JExpression lhs = null, rhs = null;
    try {
      lhs = equalityExpression();
                                      line = lhs.line();
      label_13:
      while (true) {
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case LAND:
          ;
          break;
        default:
          jj_la1[31] = jj_gen;
          break label_13;
        }
        jj_consume_token(LAND);
        rhs = equalityExpression();
              lhs = new JLogicalAndOp( line, lhs, rhs );
      }
    } catch (ParseException e) {
        recoverFromError( new int[] { SEMI, EOF }, e );
    }
      {if (true) return lhs;}
    throw new Error("Missing return statement in function");
  }

  final private JExpression equalityExpression() throws ParseException {
    int line = 0;
    JExpression lhs = null, rhs = null;
    try {
      lhs = relationalExpression();
                                      line = lhs.line();
      label_14:
      while (true) {
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case EQUAL:
          ;
          break;
        default:
          jj_la1[32] = jj_gen;
          break label_14;
        }
        jj_consume_token(EQUAL);
        rhs = relationalExpression();
              lhs = new JEqualOp( line, lhs, rhs );
      }
    } catch (ParseException e) {
        recoverFromError( new int[] { SEMI, EOF }, e );
    }
      {if (true) return lhs;}
    throw new Error("Missing return statement in function");
  }

  final private JExpression relationalExpression() throws ParseException {
    int line = 0;
    JExpression lhs = null, rhs = null;
    Type type = null;
    try {
      lhs = additiveExpression();
                                      line = lhs.line();
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case INSTANCEOF:
      case GT:
```

```
1163      case LE:
1164        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
1165        case GT:
1166          jj_consume_token(GT);
1167          rhs = additiveExpression();
1168              lhs = new JGreaterThanOp( line, lhs, rhs );
1169          break;
1170        case LE:
1171          jj_consume_token(LE);
1172          rhs = additiveExpression();
1173              lhs = new JLessEqualOp( line, lhs, rhs );
1174          break;
1175        case INSTANCEOF:
1176          jj_consume_token(INSTANCEOF);
1177          type = referenceType();
1178              lhs = new JInstanceOfOp( line, lhs, type );
1179          break;
1180        default:
1181          jj_la1[33] = jj_gen;
1182          jj_consume_token(-1);
1183          throw new ParseException();
1184        }
1185        break;
1186      default:
1187        jj_la1[34] = jj_gen;
1188        ;
1189      }
1190    } catch (ParseException e) {
1191      recoverFromError( new int[] { SEMI, EOF }, e );
1192    }
1193      {if (true) return lhs;}
1194    throw new Error("Missing return statement in function");
1195  }
1196
1197  final private JExpression additiveExpression() throws ParseException {
1198    int line = 0;
1199    JExpression lhs = null, rhs = null;
1200    try {
1201      lhs = multiplicativeExpression();
1202                                       line = lhs.line();
1203      label_15:
1204      while (true) {
1205        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
1206        case PLUS:
1207        case MINUS:
1208          ;
1209          break;
1210        default:
1211          jj_la1[35] = jj_gen;
1212          break label_15;
1213        }
1214        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
1215        case PLUS:
1216          jj_consume_token(PLUS);
1217          rhs = multiplicativeExpression();
1218              lhs = new JPlusOp( line, lhs, rhs );
1219          break;
1220        case MINUS:
1221          jj_consume_token(MINUS);
1222          rhs = multiplicativeExpression();
1223              lhs = new JSubtractOp( line, lhs, rhs );
1224          break;
1225        default:
1226          jj_la1[36] = jj_gen;
1227          jj_consume_token(-1);
1228          throw new ParseException();
1229        }
1230      }
1231    } catch (ParseException e) {
```

```
      recoverFromError( new int[] { SEMI, EOF }, e );
    }
      {if (true) return lhs;}
    throw new Error("Missing return statement in function");
  }

  final private JExpression multiplicativeExpression() throws ParseException {
    int line = 0;
    JExpression lhs = null, rhs = null;
    try {
      lhs = unaryExpression();
                                  line = lhs.line();
      label_16:
      while (true) {
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case STAR:
          ;
          break;
        default:
          jj_la1[37] = jj_gen;
          break label_16;
        }
        jj_consume_token(STAR);
        rhs = unaryExpression();
              lhs = new JMultiplyOp( line, lhs, rhs );
      }
    } catch (ParseException e) {
      recoverFromError( new int[] { SEMI, EOF }, e );
    }
      {if (true) return lhs;}
    throw new Error("Missing return statement in function");
  }

  final private JExpression unaryExpression() throws ParseException {
    int line = 0;
    JExpression expr = null, unaryExpr = null;
    try {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case INC:
        jj_consume_token(INC);
                line = token.beginLine;
        unaryExpr = unaryExpression();
          expr = new JPreIncrementOp( line, unaryExpr );
        break;
      case MINUS:
        jj_consume_token(MINUS);
                  line = token.beginLine;
        unaryExpr = unaryExpression();
          expr = new JNegateOp( line, unaryExpr );
        break;
      case FALSE:
      case NEW:
      case NULL:
      case SUPER:
      case THIS:
      case TRUE:
      case LNOT:
      case LPAREN:
      case IDENTIFIER:
      case INT_LITERAL:
      case CHAR_LITERAL:
      case STRING_LITERAL:
        expr = simpleUnaryExpression();
        break;
      default:
        jj_la1[38] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
```

```
1301    } catch (ParseException e) {
1302      recoverFromError( new int[] { SEMI, EOF }, e );
1303    }
1304      {if (true) return expr;}
1305    throw new Error("Missing return statement in function");
1306  }
1307
1308  final private JExpression simpleUnaryExpression() throws ParseException {
1309    int line = 0;
1310    Type type = null;
1311    JExpression expr = null, unaryExpr = null, simpleUnaryExpr = null;
1312    try {
1313      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
1314      case LNOT:
1315        jj_consume_token(LNOT);
1316                 line = token.beginLine;
1317        unaryExpr = unaryExpression();
1318          expr = new JLogicalNotOp( line, unaryExpr );
1319        break;
1320      default:
1321        jj_la1[39] = jj_gen;
1322        if (jj_2_6(2147483647)) {
1323          jj_consume_token(LPAREN);
1324                   line = token.beginLine;
1325          type = basicType();
1326          jj_consume_token(RPAREN);
1327          unaryExpr = unaryExpression();
1328          expr = new JCastOp( line, type, unaryExpr );
1329        } else if (jj_2_7(2147483647)) {
1330          jj_consume_token(LPAREN);
1331                   line = token.beginLine;
1332          type = referenceType();
1333          jj_consume_token(RPAREN);
1334          simpleUnaryExpr = simpleUnaryExpression();
1335          expr = new JCastOp( line, type, simpleUnaryExpr );
1336        } else {
1337          switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
1338          case FALSE:
1339          case NEW:
1340          case NULL:
1341          case SUPER:
1342          case THIS:
1343          case TRUE:
1344          case LPAREN:
1345          case IDENTIFIER:
1346          case INT_LITERAL:
1347          case CHAR_LITERAL:
1348          case STRING_LITERAL:
1349            expr = postfixExpression();
1350            break;
1351          default:
1352            jj_la1[40] = jj_gen;
1353            jj_consume_token(-1);
1354            throw new ParseException();
1355          }
1356        }
1357      }
1358    } catch (ParseException e) {
1359      recoverFromError( new int[] { SEMI, EOF }, e );
1360    }
1361      {if (true) return expr ;}
1362    throw new Error("Missing return statement in function");
1363  }
1364
1365  final private JExpression postfixExpression() throws ParseException {
1366    int line = 0;
1367    JExpression primaryExpr = null;
1368    try {
1369      primaryExpr = primary();
```

```
                                    line = primaryExpr.line();
    label_17:
    while (true) {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case LBRACK:
      case DOT:
        ;
        break;
      default:
        jj_la1[41] = jj_gen;
        break label_17;
      }
      primaryExpr = selector(primaryExpr);
    }
    label_18:
    while (true) {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case DEC:
        ;
        break;
      default:
        jj_la1[42] = jj_gen;
        break label_18;
      }
      jj_consume_token(DEC);
            primaryExpr =
              new JPostDecrementOp( line, primaryExpr );
    }
  } catch (ParseException e) {
    recoverFromError( new int[] { SEMI, EOF }, e );
  }
    {if (true) return primaryExpr;}
  throw new Error("Missing return statement in function");
}

final private JExpression selector(JExpression target) throws ParseException {
  int line = 0;
  ArrayList<JExpression> args = null;
  TypeName id = null;
  JExpression expr = null;
  try {
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case DOT:
      jj_consume_token(DOT);
              line = token.beginLine;
      id = qualifiedIdentifier();
        expr =
            new JFieldSelection( line, ambiguousPart( id ),
                                 target, id.simpleName() );
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case LPAREN:
        args = arguments();
            expr = new JMessageExpression( line, target,
                ambiguousPart( id ), id.simpleName(), args );
        break;
      default:
        jj_la1[43] = jj_gen;
        ;
      }
      break;
    case LBRACK:
      jj_consume_token(LBRACK);
                line = token.beginLine;
        expr = new JArrayExpression( line, target, expression() );
      jj_consume_token(RBRACK);
      break;
    default:
      jj_la1[44] = jj_gen;
      jj_consume_token(-1);
```

```
1439        throw new ParseException();
1440      }
1441    } catch (ParseException e) {
1442        recoverFromError( new int[] { SEMI, EOF }, e );
1443    }
1444      {if (true) return expr;}
1445    throw new Error("Missing return statement in function");
1446  }
1447
1448  final private JExpression primary() throws ParseException {
1449    int line = 0;
1450    JExpression expr = null;
1451    JExpression newTarget = null;
1452    ArrayList<JExpression> args = null;
1453    TypeName id = null;
1454    try {
1455      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
1456      case LPAREN:
1457        expr = parExpression();
1458        break;
1459      case THIS:
1460        jj_consume_token(THIS);
1461                 line = token.beginLine; expr = new JThis( line );
1462        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
1463        case LPAREN:
1464          args = arguments();
1465              expr = new JThisConstruction( line, args );
1466          break;
1467        default:
1468          jj_la1[45] = jj_gen;
1469          ;
1470        }
1471        break;
1472      case SUPER:
1473        jj_consume_token(SUPER);
1474                 line = token.beginLine;
1475        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
1476        case LPAREN:
1477          args = arguments();
1478              expr = new JSuperConstruction( line, args );
1479          break;
1480        case DOT:
1481          jj_consume_token(DOT);
1482          jj_consume_token(IDENTIFIER);
1483                newTarget = new JSuper( line );
1484                expr = new JFieldSelection( line, newTarget,
1485                                            token.image );
1486          switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
1487          case LPAREN:
1488            args = arguments();
1489                  expr = new JMessageExpression( line, newTarget,
1490                      null, token.image, args );
1491            break;
1492          default:
1493            jj_la1[46] = jj_gen;
1494            ;
1495          }
1496          break;
1497        default:
1498          jj_la1[47] = jj_gen;
1499          jj_consume_token(-1);
1500          throw new ParseException();
1501        }
1502        break;
1503      case FALSE:
1504      case NULL:
1505      case TRUE:
1506      case INT_LITERAL:
1507      case CHAR_LITERAL:
```

```
1508      case STRING_LITERAL:
1509        expr = literal();
1510        break;
1511      case NEW:
1512        jj_consume_token(NEW);
1513        expr = creator();
1514        break;
1515      case IDENTIFIER:
1516        // Language is ambiguous here. JavaCC warns about not being
1517                // able to choose between qualifiedIdentifier and selector.
1518                // Semantic analysis will sort it out.
1519                id = qualifiedIdentifier();
1520            line = id.line();
1521            if ( ambiguousPart( id ) == null ) {
1522                expr = new JVariable( line, id.simpleName() );
1523            }
1524            else {
1525                expr = new JFieldSelection( line, ambiguousPart( id ),
1526                                            null, id.simpleName() );
1527            }
1528        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
1529        case LPAREN:
1530          args = arguments();
1531              expr = new JMessageExpression( line, null,
1532                 ambiguousPart( id ), id.simpleName(), args );
1533          break;
1534        default:
1535          jj_la1[48] = jj_gen;
1536          ;
1537        }
1538        break;
1539      default:
1540        jj_la1[49] = jj_gen;
1541        jj_consume_token(-1);
1542        throw new ParseException();
1543      }
1544    } catch (ParseException e) {
1545        recoverFromError( new int[] { SEMI, EOF }, e );
1546    }
1547      {if (true) return expr;}
1548    throw new Error("Missing return statement in function");
1549  }
1550
1551  final private JExpression creator() throws ParseException {
1552    int line = 0;
1553    Type type = null;
1554    ArrayList<JExpression> args = null;
1555    ArrayList<JExpression> dims = null;
1556    JArrayInitializer init =  null;
1557    JExpression expr = null;
1558    Type expected = null;
1559    try {
1560      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
1561      case BOOLEAN:
1562      case CHAR:
1563      case INT:
1564        type = basicType();
1565        break;
1566      case IDENTIFIER:
1567        type = qualifiedIdentifier();
1568        break;
1569      default:
1570        jj_la1[50] = jj_gen;
1571        jj_consume_token(-1);
1572        throw new ParseException();
1573      }
1574          line = token.beginLine; expected = type;
1575      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
1576      case LPAREN:
```

```
1577        args = arguments();
1578              expr = new JNewOp( line, type, args );
1579        break;
1580      default:
1581        jj_la1[52] = jj_gen;
1582        if (jj_2_9(2147483647)) {
1583          expr = newArrayDeclarator(type);
1584        } else {
1585          switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
1586          case LBRACK:
1587            jj_consume_token(LBRACK);
1588            jj_consume_token(RBRACK);
1589                                  expected = new ArrayTypeName( expected
1590    );
1591            label_19:
1592            while (true) {
1593              if (jj_2_8(2147483647)) {
1594                ;
1595              } else {
1596                break label_19;
1597              }
1598              jj_consume_token(LBRACK);
1599              jj_consume_token(RBRACK);
1600                  expected = new ArrayTypeName( expected );
1601            }
1602            switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
1603            case LCURLY:
1604              expr = arrayInitializer(expected);
1605              break;
1606            default:
1607              jj_la1[51] = jj_gen;
1608              ;
1609            }
1610            break;
1611          default:
1612            jj_la1[53] = jj_gen;
1613            jj_consume_token(-1);
1614            throw new ParseException();
1615          }
1616        }
1617      }
1618    } catch (ParseException e) {
1619        expr = new JWildExpression( token.beginLine );
1620        recoverFromError( new int[] { SEMI, EOF }, e );
1621    }
1622      {if (true) return expr;}
1623    throw new Error("Missing return statement in function");
1624  }
1625
1626  final private JNewArrayOp newArrayDeclarator(Type type) throws ParseException {
1627    int line = 0;
1628    ArrayList<JExpression> dimensions = new ArrayList<JExpression>();
1629    JExpression expr = null;
1630    try {
1631      jj_consume_token(LBRACK);
1632                    line = token.beginLine;
1633      expr = expression();
1634          dimensions.add( expr ); type = new ArrayTypeName( type );
1635      jj_consume_token(RBRACK);
1636      label_20:
1637      while (true) {
1638        if (jj_2_10(2147483647)) {
1639          ;
1640        } else {
1641          break label_20;
1642        }
1643        jj_consume_token(LBRACK);
1644        expr = expression();
1645              dimensions.add( expr); type = new ArrayTypeName( type );
```

```
1646        jj_consume_token(RBRACK);
1647      }
1648      label_21:
1649      while (true) {
1650        if (jj_2_11(2147483647)) {
1651          ;
1652        } else {
1653          break label_21;
1654        }
1655        jj_consume_token(LBRACK);
1656        jj_consume_token(RBRACK);
1657                                  type = new ArrayTypeName( type );
1658      }
1659    } catch (ParseException e) {
1660        recoverFromError( new int[] { SEMI, EOF }, e );
1661    }
1662      {if (true) return new JNewArrayOp( line, type, dimensions );}
1663    throw new Error("Missing return statement in function");
1664  }
1665
1666  final private JExpression literal() throws ParseException {
1667    JExpression expr = null;
1668    try {
1669      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
1670      case INT_LITERAL:
1671        jj_consume_token(INT_LITERAL);
1672          expr = new JLiteralInt( token.beginLine, token.image );
1673        break;
1674      case CHAR_LITERAL:
1675        jj_consume_token(CHAR_LITERAL);
1676          expr = new JLiteralChar( token.beginLine, token.image );
1677        break;
1678      case STRING_LITERAL:
1679        jj_consume_token(STRING_LITERAL);
1680          expr =
1681              new JLiteralString( token.beginLine, token.image );
1682        break;
1683      case TRUE:
1684        jj_consume_token(TRUE);
1685          expr = new JLiteralTrue( token.beginLine );
1686        break;
1687      case FALSE:
1688        jj_consume_token(FALSE);
1689          expr = new JLiteralFalse( token.beginLine );
1690        break;
1691      case NULL:
1692        jj_consume_token(NULL);
1693          expr = new JLiteralNull( token.beginLine );
1694        break;
1695      default:
1696        jj_la1[54] = jj_gen;
1697        jj_consume_token(-1);
1698        throw new ParseException();
1699      }
1700    } catch (ParseException e) {
1701        expr = new JWildExpression( token.beginLine );
1702        recoverFromError( new int[] { SEMI, EOF }, e );
1703    }
1704      {if (true) return expr;}
1705    throw new Error("Missing return statement in function");
1706  }
1707
1708  final private boolean jj_2_1(int xla) {
1709    jj_la = xla; jj_lastpos = jj_scanpos = token;
1710    try { return !jj_3_1(); }
1711    catch(LookaheadSuccess ls) { return true; }
1712    finally { jj_save(0, xla); }
1713  }
1714
```

```
1715  final private boolean jj_2_2(int xla) {
1716    jj_la = xla; jj_lastpos = jj_scanpos = token;
1717    try { return !jj_3_2(); }
1718    catch(LookaheadSuccess ls) { return true; }
1719    finally { jj_save(1, xla); }
1720  }
1721
1722  final private boolean jj_2_3(int xla) {
1723    jj_la = xla; jj_lastpos = jj_scanpos = token;
1724    try { return !jj_3_3(); }
1725    catch(LookaheadSuccess ls) { return true; }
1726    finally { jj_save(2, xla); }
1727  }
1728
1729  final private boolean jj_2_4(int xla) {
1730    jj_la = xla; jj_lastpos = jj_scanpos = token;
1731    try { return !jj_3_4(); }
1732    catch(LookaheadSuccess ls) { return true; }
1733    finally { jj_save(3, xla); }
1734  }
1735
1736  final private boolean jj_2_5(int xla) {
1737    jj_la = xla; jj_lastpos = jj_scanpos = token;
1738    try { return !jj_3_5(); }
1739    catch(LookaheadSuccess ls) { return true; }
1740    finally { jj_save(4, xla); }
1741  }
1742
1743  final private boolean jj_2_6(int xla) {
1744    jj_la = xla; jj_lastpos = jj_scanpos = token;
1745    try { return !jj_3_6(); }
1746    catch(LookaheadSuccess ls) { return true; }
1747    finally { jj_save(5, xla); }
1748  }
1749
1750  final private boolean jj_2_7(int xla) {
1751    jj_la = xla; jj_lastpos = jj_scanpos = token;
1752    try { return !jj_3_7(); }
1753    catch(LookaheadSuccess ls) { return true; }
1754    finally { jj_save(6, xla); }
1755  }
1756
1757  final private boolean jj_2_8(int xla) {
1758    jj_la = xla; jj_lastpos = jj_scanpos = token;
1759    try { return !jj_3_8(); }
1760    catch(LookaheadSuccess ls) { return true; }
1761    finally { jj_save(7, xla); }
1762  }
1763
1764  final private boolean jj_2_9(int xla) {
1765    jj_la = xla; jj_lastpos = jj_scanpos = token;
1766    try { return !jj_3_9(); }
1767    catch(LookaheadSuccess ls) { return true; }
1768    finally { jj_save(8, xla); }
1769  }
1770
1771  final private boolean jj_2_10(int xla) {
1772    jj_la = xla; jj_lastpos = jj_scanpos = token;
1773    try { return !jj_3_10(); }
1774    catch(LookaheadSuccess ls) { return true; }
1775    finally { jj_save(9, xla); }
1776  }
1777
1778  final private boolean jj_2_11(int xla) {
1779    jj_la = xla; jj_lastpos = jj_scanpos = token;
1780    try { return !jj_3_11(); }
1781    catch(LookaheadSuccess ls) { return true; }
1782    finally { jj_save(10, xla); }
1783  }
```

```
1784
1785  final private boolean jj_3R_81() {
1786    if (jj_scan_token(DOT)) return true;
1787    if (jj_scan_token(IDENTIFIER)) return true;
1788    Token xsp;
1789    xsp = jj_scanpos;
1790    if (jj_3R_88()) jj_scanpos = xsp;
1791    return false;
1792  }
1793
1794  final private boolean jj_3R_80() {
1795    if (jj_3R_87()) return true;
1796    return false;
1797  }
1798
1799  final private boolean jj_3R_111() {
1800    if (jj_scan_token(COMMA)) return true;
1801    if (jj_3R_110()) return true;
1802    return false;
1803  }
1804
1805  final private boolean jj_3_2() {
1806    Token xsp;
1807    xsp = jj_scanpos;
1808    if (jj_scan_token(29)) {
1809    jj_scanpos = xsp;
1810    if (jj_3R_22()) return true;
1811    }
1812    if (jj_scan_token(IDENTIFIER)) return true;
1813    if (jj_scan_token(LPAREN)) return true;
1814    return false;
1815  }
1816
1817  final private boolean jj_3R_56() {
1818    if (jj_scan_token(STAR)) return true;
1819    if (jj_3R_55()) return true;
1820    return false;
1821  }
1822
1823  final private boolean jj_3R_79() {
1824    if (jj_3R_87()) return true;
1825    return false;
1826  }
1827
1828  final private boolean jj_3R_73() {
1829    if (jj_scan_token(SUPER)) return true;
1830    Token xsp;
1831    xsp = jj_scanpos;
1832    if (jj_3R_80()) {
1833    jj_scanpos = xsp;
1834    if (jj_3R_81()) return true;
1835    }
1836    return false;
1837  }
1838
1839  final private boolean jj_3R_109() {
1840    if (jj_3R_110()) return true;
1841    Token xsp;
1842    while (true) {
1843      xsp = jj_scanpos;
1844      if (jj_3R_111()) { jj_scanpos = xsp; break; }
1845    }
1846    return false;
1847  }
1848
1849  final private boolean jj_3R_72() {
1850    if (jj_scan_token(THIS)) return true;
1851    Token xsp;
1852    xsp = jj_scanpos;
```

```
1853    if (jj_3R_79()) jj_scanpos = xsp;
1854    return false;
1855  }
1856
1857  final private boolean jj_3R_71() {
1858    if (jj_3R_78()) return true;
1859    return false;
1860  }
1861
1862  final private boolean jj_3R_50() {
1863    if (jj_3R_55()) return true;
1864    Token xsp;
1865    while (true) {
1866      xsp = jj_scanpos;
1867      if (jj_3R_56()) { jj_scanpos = xsp; break; }
1868    }
1869    return false;
1870  }
1871
1872  final private boolean jj_3R_68() {
1873    Token xsp;
1874    xsp = jj_scanpos;
1875    if (jj_3R_71()) {
1876    jj_scanpos = xsp;
1877    if (jj_3R_72()) {
1878    jj_scanpos = xsp;
1879    if (jj_3R_73()) {
1880    jj_scanpos = xsp;
1881    if (jj_3R_74()) {
1882    jj_scanpos = xsp;
1883    if (jj_3R_75()) {
1884    jj_scanpos = xsp;
1885    if (jj_3R_76()) return true;
1886    }
1887    }
1888    }
1889    }
1890    }
1891    return false;
1892  }
1893
1894  final private boolean jj_3_1() {
1895    if (jj_scan_token(IDENTIFIER)) return true;
1896    if (jj_scan_token(LPAREN)) return true;
1897    return false;
1898  }
1899
1900  final private boolean jj_3R_27() {
1901    if (jj_3R_35()) return true;
1902    return false;
1903  }
1904
1905  final private boolean jj_3R_108() {
1906    if (jj_scan_token(LCURLY)) return true;
1907    Token xsp;
1908    xsp = jj_scanpos;
1909    if (jj_3R_109()) jj_scanpos = xsp;
1910    if (jj_scan_token(RCURLY)) return true;
1911    return false;
1912  }
1913
1914  final private boolean jj_3R_94() {
1915    if (jj_scan_token(NULL)) return true;
1916    return false;
1917  }
1918
1919  final private boolean jj_3R_93() {
1920    if (jj_scan_token(FALSE)) return true;
1921    return false;
```

```
}

final private boolean jj_3R_92() {
  if (jj_scan_token(TRUE)) return true;
  return false;
}

final private boolean jj_3R_58() {
  if (jj_scan_token(MINUS)) return true;
  if (jj_3R_50()) return true;
  return false;
}

final private boolean jj_3R_91() {
  if (jj_scan_token(STRING_LITERAL)) return true;
  return false;
}

final private boolean jj_3R_90() {
  if (jj_scan_token(CHAR_LITERAL)) return true;
  return false;
}

final private boolean jj_3R_51() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_57()) {
  jj_scanpos = xsp;
  if (jj_3R_58()) return true;
  }
  return false;
}

final private boolean jj_3R_57() {
  if (jj_scan_token(PLUS)) return true;
  if (jj_3R_50()) return true;
  return false;
}

final private boolean jj_3R_89() {
  if (jj_scan_token(INT_LITERAL)) return true;
  return false;
}

final private boolean jj_3R_113() {
  if (jj_3R_27()) return true;
  return false;
}

final private boolean jj_3R_82() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_89()) {
  jj_scanpos = xsp;
  if (jj_3R_90()) {
  jj_scanpos = xsp;
  if (jj_3R_91()) {
  jj_scanpos = xsp;
  if (jj_3R_92()) {
  jj_scanpos = xsp;
  if (jj_3R_93()) {
  jj_scanpos = xsp;
  if (jj_3R_94()) return true;
  }
  }
  }
  }
  }
  return false;
```

```
}

final private boolean jj_3R_112() {
  if (jj_3R_108()) return true;
  return false;
}

final private boolean jj_3R_86() {
  if (jj_scan_token(LBRACK)) return true;
  if (jj_scan_token(RBRACK)) return true;
  return false;
}

final private boolean jj_3R_100() {
  if (jj_3R_87()) return true;
  return false;
}

final private boolean jj_3R_48() {
  if (jj_3R_50()) return true;
  Token xsp;
  while (true) {
    xsp = jj_scanpos;
    if (jj_3R_51()) { jj_scanpos = xsp; break; }
  }
  return false;
}

final private boolean jj_3R_110() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_112()) {
  jj_scanpos = xsp;
  if (jj_3R_113()) return true;
  }
  return false;
}

final private boolean jj_3R_51() {
  if (jj_scan_token(DOT)) return true;
  if (jj_scan_token(IDENTIFIER)) return true;
  return false;
}

final private boolean jj_3_11() {
  if (jj_scan_token(LBRACK)) return true;
  if (jj_scan_token(RBRACK)) return true;
  return false;
}

final private boolean jj_3_10() {
  if (jj_scan_token(LBRACK)) return true;
  if (jj_3R_27()) return true;
  if (jj_scan_token(RBRACK)) return true;
  return false;
}

final private boolean jj_3R_85() {
  if (jj_scan_token(DOT)) return true;
  if (jj_3R_37()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_100()) jj_scanpos = xsp;
  return false;
}

final private boolean jj_3R_107() {
  if (jj_scan_token(LBRACK)) return true;
  if (jj_scan_token(RBRACK)) return true;
```

```
2060    return false;
2061  }
2062
2063  final private boolean jj_3_4() {
2064    if (jj_scan_token(ELSE)) return true;
2065    return false;
2066  }
2067
2068  final private boolean jj_3R_77() {
2069    Token xsp;
2070    xsp = jj_scanpos;
2071    if (jj_3R_85()) {
2072    jj_scanpos = xsp;
2073    if (jj_3R_86()) return true;
2074    }
2075    return false;
2076  }
2077
2078  final private boolean jj_3R_37() {
2079    if (jj_scan_token(IDENTIFIER)) return true;
2080    Token xsp;
2081    while (true) {
2082      xsp = jj_scanpos;
2083      if (jj_3R_41()) { jj_scanpos = xsp; break; }
2084    }
2085    return false;
2086  }
2087
2088  final private boolean jj_3R_54() {
2089    if (jj_scan_token(INSTANCEOF)) return true;
2090    if (jj_3R_26()) return true;
2091    return false;
2092  }
2093
2094  final private boolean jj_3R_106() {
2095    if (jj_scan_token(LBRACK)) return true;
2096    if (jj_3R_27()) return true;
2097    if (jj_scan_token(RBRACK)) return true;
2098    return false;
2099  }
2100
2101  final private boolean jj_3R_53() {
2102    if (jj_scan_token(LE)) return true;
2103    if (jj_3R_48()) return true;
2104    return false;
2105  }
2106
2107  final private boolean jj_3R_49() {
2108    Token xsp;
2109    xsp = jj_scanpos;
2110    if (jj_3R_52()) {
2111    jj_scanpos = xsp;
2112    if (jj_3R_53()) {
2113    jj_scanpos = xsp;
2114    if (jj_3R_54()) return true;
2115    }
2116    }
2117    return false;
2118  }
2119
2120  final private boolean jj_3R_52() {
2121    if (jj_scan_token(GT)) return true;
2122    if (jj_3R_48()) return true;
2123    return false;
2124  }
2125
2126  final private boolean jj_3R_38() {
2127    if (jj_scan_token(LBRACK)) return true;
2128    if (jj_scan_token(RBRACK)) return true;
```

```
2129    return false;
2130  }
2131
2132  final private boolean jj_3R_36() {
2133    if (jj_scan_token(LBRACK)) return true;
2134    if (jj_scan_token(RBRACK)) return true;
2135    return false;
2136  }
2137
2138  final private boolean jj_3R_102() {
2139    if (jj_scan_token(LBRACK)) return true;
2140    if (jj_3R_27()) return true;
2141    if (jj_scan_token(RBRACK)) return true;
2142    Token xsp;
2143    while (true) {
2144      xsp = jj_scanpos;
2145      if (jj_3R_106()) { jj_scanpos = xsp; break; }
2146    }
2147    while (true) {
2148      xsp = jj_scanpos;
2149      if (jj_3R_107()) { jj_scanpos = xsp; break; }
2150    }
2151    return false;
2152  }
2153
2154  final private boolean jj_3R_34() {
2155    if (jj_3R_37()) return true;
2156    Token xsp;
2157    while (true) {
2158      xsp = jj_scanpos;
2159      if (jj_3R_38()) { jj_scanpos = xsp; break; }
2160    }
2161    return false;
2162  }
2163
2164  final private boolean jj_3R_46() {
2165    if (jj_3R_48()) return true;
2166    Token xsp;
2167    xsp = jj_scanpos;
2168    if (jj_3R_49()) jj_scanpos = xsp;
2169    return false;
2170  }
2171
2172  final private boolean jj_3R_70() {
2173    if (jj_scan_token(DEC)) return true;
2174    return false;
2175  }
2176
2177  final private boolean jj_3R_33() {
2178    if (jj_3R_25()) return true;
2179    if (jj_scan_token(LBRACK)) return true;
2180    if (jj_scan_token(RBRACK)) return true;
2181    Token xsp;
2182    while (true) {
2183      xsp = jj_scanpos;
2184      if (jj_3R_36()) { jj_scanpos = xsp; break; }
2185    }
2186    return false;
2187  }
2188
2189  final private boolean jj_3R_69() {
2190    if (jj_3R_77()) return true;
2191    return false;
2192  }
2193
2194  final private boolean jj_3R_26() {
2195    Token xsp;
2196    xsp = jj_scanpos;
2197    if (jj_3R_33()) {
```

```
2198    jj_scanpos = xsp;
2199    if (jj_3R_34()) return true;
2200    }
2201    return false;
2202  }
2203
2204  final private boolean jj_3_8() {
2205    if (jj_scan_token(LBRACK)) return true;
2206    if (jj_scan_token(RBRACK)) return true;
2207    return false;
2208  }
2209
2210  final private boolean jj_3R_67() {
2211    if (jj_3R_68()) return true;
2212    Token xsp;
2213    while (true) {
2214      xsp = jj_scanpos;
2215      if (jj_3R_69()) { jj_scanpos = xsp; break; }
2216    }
2217    while (true) {
2218      xsp = jj_scanpos;
2219      if (jj_3R_70()) { jj_scanpos = xsp; break; }
2220    }
2221    return false;
2222  }
2223
2224  final private boolean jj_3R_104() {
2225    if (jj_3R_108()) return true;
2226    return false;
2227  }
2228
2229  final private boolean jj_3_9() {
2230    if (jj_scan_token(LBRACK)) return true;
2231    if (jj_3R_27()) return true;
2232    if (jj_scan_token(RBRACK)) return true;
2233    return false;
2234  }
2235
2236  final private boolean jj_3R_47() {
2237    if (jj_scan_token(EQUAL)) return true;
2238    if (jj_3R_46()) return true;
2239    return false;
2240  }
2241
2242  final private boolean jj_3R_103() {
2243    if (jj_scan_token(LBRACK)) return true;
2244    if (jj_scan_token(RBRACK)) return true;
2245    return false;
2246  }
2247
2248  final private boolean jj_3_3() {
2249    if (jj_3R_23()) return true;
2250    if (jj_scan_token(IDENTIFIER)) return true;
2251    return false;
2252  }
2253
2254  final private boolean jj_3R_32() {
2255    if (jj_scan_token(INT)) return true;
2256    return false;
2257  }
2258
2259  final private boolean jj_3R_99() {
2260    if (jj_scan_token(LBRACK)) return true;
2261    if (jj_scan_token(RBRACK)) return true;
2262    Token xsp;
2263    while (true) {
2264      xsp = jj_scanpos;
2265      if (jj_3R_103()) { jj_scanpos = xsp; break; }
2266    }
```

```
2267    xsp = jj_scanpos;
2268    if (jj_3R_104()) jj_scanpos = xsp;
2269    return false;
2270  }
2271
2272  final private boolean jj_3R_31() {
2273    if (jj_scan_token(CHAR)) return true;
2274    return false;
2275  }
2276
2277  final private boolean jj_3R_24() {
2278    if (jj_3R_25()) return true;
2279    if (jj_scan_token(LBRACK)) return true;
2280    if (jj_scan_token(RBRACK)) return true;
2281    return false;
2282  }
2283
2284  final private boolean jj_3_7() {
2285    if (jj_scan_token(LPAREN)) return true;
2286    if (jj_3R_26()) return true;
2287    if (jj_scan_token(RPAREN)) return true;
2288    return false;
2289  }
2290
2291  final private boolean jj_3R_30() {
2292    if (jj_scan_token(BOOLEAN)) return true;
2293    return false;
2294  }
2295
2296  final private boolean jj_3R_28() {
2297    if (jj_3R_102()) return true;
2298    return false;
2299  }
2300
2301  final private boolean jj_3R_42() {
2302    if (jj_3R_46()) return true;
2303    Token xsp;
2304    while (true) {
2305      xsp = jj_scanpos;
2306      if (jj_3R_47()) { jj_scanpos = xsp; break; }
2307    }
2308    return false;
2309  }
2310
2311  final private boolean jj_3R_97() {
2312    if (jj_3R_87()) return true;
2313    return false;
2314  }
2315
2316  final private boolean jj_3R_66() {
2317    if (jj_3R_67()) return true;
2318    return false;
2319  }
2320
2321  final private boolean jj_3R_25() {
2322    Token xsp;
2323    xsp = jj_scanpos;
2324    if (jj_3R_30()) {
2325    jj_scanpos = xsp;
2326    if (jj_3R_31()) {
2327    jj_scanpos = xsp;
2328    if (jj_3R_32()) return true;
2329    }
2330    }
2331    return false;
2332  }
2333
2334  final private boolean jj_3_6() {
2335    if (jj_scan_token(LPAREN)) return true;
```

```
2336    if (jj_3R_25()) return true;
2337    if (jj_scan_token(RPAREN)) return true;
2338    return false;
2339  }
2340
2341  final private boolean jj_3R_96() {
2342    if (jj_3R_37()) return true;
2343    return false;
2344  }
2345
2346  final private boolean jj_3R_95() {
2347    if (jj_3R_25()) return true;
2348    return false;
2349  }
2350
2351  final private boolean jj_3R_65() {
2352    if (jj_scan_token(LPAREN)) return true;
2353    if (jj_3R_26()) return true;
2354    if (jj_scan_token(RPAREN)) return true;
2355    if (jj_3R_62()) return true;
2356    return false;
2357  }
2358
2359  final private boolean jj_3_5() {
2360    Token xsp;
2361    xsp = jj_scanpos;
2362    if (jj_scan_token(52)) {
2363    jj_scanpos = xsp;
2364    if (jj_3R_24()) return true;
2365    }
2366    return false;
2367  }
2368
2369  final private boolean jj_3R_83() {
2370    Token xsp;
2371    xsp = jj_scanpos;
2372    if (jj_3R_95()) {
2373    jj_scanpos = xsp;
2374    if (jj_3R_96()) return true;
2375    }
2376    xsp = jj_scanpos;
2377    if (jj_3R_97()) {
2378    jj_scanpos = xsp;
2379    if (jj_3R_98()) {
2380    jj_scanpos = xsp;
2381    if (jj_3R_99()) return true;
2382    }
2383    }
2384    return false;
2385  }
2386
2387  final private boolean jj_3R_64() {
2388    if (jj_scan_token(LPAREN)) return true;
2389    if (jj_3R_25()) return true;
2390    if (jj_scan_token(RPAREN)) return true;
2391    if (jj_3R_55()) return true;
2392    return false;
2393  }
2394
2395  final private boolean jj_3R_63() {
2396    if (jj_scan_token(LNOT)) return true;
2397    if (jj_3R_55()) return true;
2398    return false;
2399  }
2400
2401  final private boolean jj_3R_43() {
2402    if (jj_scan_token(LAND)) return true;
2403    if (jj_3R_42()) return true;
2404    return false;
```

```
2405  }
2406
2407  final private boolean jj_3R_29() {
2408    if (jj_3R_25()) return true;
2409    return false;
2410  }
2411
2412  final private boolean jj_3R_62() {
2413    Token xsp;
2414    xsp = jj_scanpos;
2415    if (jj_3R_63()) {
2416    jj_scanpos = xsp;
2417    if (jj_3R_64()) {
2418    jj_scanpos = xsp;
2419    if (jj_3R_65()) {
2420    jj_scanpos = xsp;
2421    if (jj_3R_66()) return true;
2422    }
2423    }
2424    }
2425    return false;
2426  }
2427
2428  final private boolean jj_3R_28() {
2429    if (jj_3R_26()) return true;
2430    return false;
2431  }
2432
2433  final private boolean jj_3R_23() {
2434    Token xsp;
2435    xsp = jj_scanpos;
2436    if (jj_3R_28()) {
2437    jj_scanpos = xsp;
2438    if (jj_3R_29()) return true;
2439    }
2440    return false;
2441  }
2442
2443  final private boolean jj_3R_39() {
2444    if (jj_3R_42()) return true;
2445    Token xsp;
2446    while (true) {
2447      xsp = jj_scanpos;
2448      if (jj_3R_43()) { jj_scanpos = xsp; break; }
2449    }
2450    return false;
2451  }
2452
2453  final private boolean jj_3R_84() {
2454    if (jj_3R_87()) return true;
2455    return false;
2456  }
2457
2458  final private boolean jj_3R_105() {
2459    if (jj_scan_token(COMMA)) return true;
2460    if (jj_3R_27()) return true;
2461    return false;
2462  }
2463
2464  final private boolean jj_3R_61() {
2465    if (jj_3R_62()) return true;
2466    return false;
2467  }
2468
2469  final private boolean jj_3R_101() {
2470    if (jj_3R_27()) return true;
2471    Token xsp;
2472    while (true) {
2473      xsp = jj_scanpos;
```

```
2474      if (jj_3R_105()) { jj_scanpos = xsp; break; }
2475    }
2476    return false;
2477  }
2478
2479  final private boolean jj_3R_45() {
2480    if (jj_scan_token(PLUS_ASSIGN)) return true;
2481    if (jj_3R_35()) return true;
2482    return false;
2483  }
2484
2485  final private boolean jj_3R_78() {
2486    if (jj_scan_token(LPAREN)) return true;
2487    if (jj_3R_27()) return true;
2488    if (jj_scan_token(RPAREN)) return true;
2489    return false;
2490  }
2491
2492  final private boolean jj_3R_60() {
2493    if (jj_scan_token(MINUS)) return true;
2494    if (jj_3R_55()) return true;
2495    return false;
2496  }
2497
2498  final private boolean jj_3R_44() {
2499    if (jj_scan_token(ASSIGN)) return true;
2500    if (jj_3R_35()) return true;
2501    return false;
2502  }
2503
2504  final private boolean jj_3R_40() {
2505    Token xsp;
2506    xsp = jj_scanpos;
2507    if (jj_3R_44()) {
2508    jj_scanpos = xsp;
2509    if (jj_3R_45()) return true;
2510    }
2511    return false;
2512  }
2513
2514  final private boolean jj_3R_59() {
2515    if (jj_scan_token(INC)) return true;
2516    if (jj_3R_55()) return true;
2517    return false;
2518  }
2519
2520  final private boolean jj_3R_76() {
2521    if (jj_3R_37()) return true;
2522    Token xsp;
2523    xsp = jj_scanpos;
2524    if (jj_3R_84()) jj_scanpos = xsp;
2525    return false;
2526  }
2527
2528  final private boolean jj_3R_55() {
2529    Token xsp;
2530    xsp = jj_scanpos;
2531    if (jj_3R_59()) {
2532    jj_scanpos = xsp;
2533    if (jj_3R_60()) {
2534    jj_scanpos = xsp;
2535    if (jj_3R_61()) return true;
2536    }
2537    }
2538    return false;
2539  }
2540
2541  final private boolean jj_3R_87() {
2542    if (jj_scan_token(LPAREN)) return true;
```

```
2543    Token xsp;
2544    xsp = jj_scanpos;
2545    if (jj_3R_101()) jj_scanpos = xsp;
2546    if (jj_scan_token(RPAREN)) return true;
2547    return false;
2548  }
2549
2550  final private boolean jj_3R_88() {
2551    if (jj_3R_87()) return true;
2552    return false;
2553  }
2554
2555  final private boolean jj_3R_35() {
2556    if (jj_3R_39()) return true;
2557    Token xsp;
2558    xsp = jj_scanpos;
2559    if (jj_3R_40()) jj_scanpos = xsp;
2560    return false;
2561  }
2562
2563  final private boolean jj_3R_75() {
2564    if (jj_scan_token(NEW)) return true;
2565    if (jj_3R_83()) return true;
2566    return false;
2567  }
2568
2569  final private boolean jj_3R_74() {
2570    if (jj_3R_82()) return true;
2571    return false;
2572  }
2573
2574  final private boolean jj_3R_22() {
2575    if (jj_3R_23()) return true;
2576    return false;
2577  }
2578
2579  public JavaCCParserTokenManager token_source;
2580  SimpleCharStream jj_input_stream;
2581  public Token token, jj_nt;
2582  private int jj_ntk;
2583  private Token jj_scanpos, jj_lastpos;
2584  private int jj_la;
2585  public boolean lookingAhead = false;
2586  private boolean jj_semLA;
2587  private int jj_gen;
2588  final private int[] jj_la1 = new int[55];
2589  static private int[] jj_la1_0;
2590  static private int[] jj_la1_1;
2591  static {
2592      jj_la1_0();
2593      jj_la1_1();
2594   }
2595   private static void jj_la1_0() {
2596      jj_la1_0 = new int[]
{0x100000,0x8000,0x2e00480,0x0,0x2e00080,0x2e00080,0x1000,0x22e20380,0x20020300,0x0,0
x20300,0x5d0e6300,0x5d0c6000,0x1c0c2000,0x5d0c6000,0x0,0x20300,0x0,0x0,0x1c0c2000,0x0
,0x1c0c2000,0x0,0x1c0c2000,0x20300,0x20300,0x0,0x0,0x20300,0x0,0x0,0x0,0x0,0x10000,0x
10000,0x80000000,0x80000000,0x0,0x1c0c2000,0x0,0x1c0c2000,0x0,0x0,0x0,0x0,0x0,0x0,0x0
,0x0,0x1c0c2000,0x20300,0x0,0x0,0x0,0x10082000,};
2597    }
2598    private static void jj_la1_1() {
2599      jj_la1_1 = new int[]
{0x0,0x0,0x0,0x80000,0x0,0x0,0x0,0x100000,0x100000,0x22000,0x100000,0x7122990,0x71229
90,0x7100990,0x7122990,0x40000,0x100000,0x40000,0x1,0x7102990,0x40000,0x7102990,0x400
00,0x7100990,0x0,0x0,0x8000,0x8000,0x100000,0x201,0x201,0x20,0x4,0x48,0x48,0x100,0x10
0,0x400,0x7100990,0x80,0x7100800,0x88000,0x2,0x800,0x88000,0x800,0x800,0x80800,0x800,
0x7100800,0x100000,0x2000,0x800,0x8000,0x7000000,};
2600    }
2601  final private JJCalls[] jj_2_rtns = new JJCalls[11];
```

```
2602  private boolean jj_rescan = false;
2603  private int jj_gc = 0;
2604
2605  public JavaCCParser(java.io.InputStream stream) {
2606     this(stream, null);
2607  }
2608  public JavaCCParser(java.io.InputStream stream, String encoding) {
2609    try { jj_input_stream = new SimpleCharStream(stream, encoding, 1, 1); }
catch(java.io.UnsupportedEncodingException e) { throw new RuntimeException(e); }
2610    token_source = new JavaCCParserTokenManager(jj_input_stream);
2611    token = new Token();
2612    jj_ntk = -1;
2613    jj_gen = 0;
2614    for (int i = 0; i < 55; i++) jj_la1[i] = -1;
2615    for (int i = 0; i < jj_2_rtns.length; i++) jj_2_rtns[i] = new JJCalls();
2616  }
2617
2618  public void ReInit(java.io.InputStream stream) {
2619     ReInit(stream, null);
2620  }
2621  public void ReInit(java.io.InputStream stream, String encoding) {
2622    try { jj_input_stream.ReInit(stream, encoding, 1, 1); }
catch(java.io.UnsupportedEncodingException e) { throw new RuntimeException(e); }
2623    token_source.ReInit(jj_input_stream);
2624    token = new Token();
2625    jj_ntk = -1;
2626    jj_gen = 0;
2627    for (int i = 0; i < 55; i++) jj_la1[i] = -1;
2628    for (int i = 0; i < jj_2_rtns.length; i++) jj_2_rtns[i] = new JJCalls();
2629  }
2630
2631  public JavaCCParser(java.io.Reader stream) {
2632    jj_input_stream = new SimpleCharStream(stream, 1, 1);
2633    token_source = new JavaCCParserTokenManager(jj_input_stream);
2634    token = new Token();
2635    jj_ntk = -1;
2636    jj_gen = 0;
2637    for (int i = 0; i < 55; i++) jj_la1[i] = -1;
2638    for (int i = 0; i < jj_2_rtns.length; i++) jj_2_rtns[i] = new JJCalls();
2639  }
2640
2641  public void ReInit(java.io.Reader stream) {
2642    jj_input_stream.ReInit(stream, 1, 1);
2643    token_source.ReInit(jj_input_stream);
2644    token = new Token();
2645    jj_ntk = -1;
2646    jj_gen = 0;
2647    for (int i = 0; i < 55; i++) jj_la1[i] = -1;
2648    for (int i = 0; i < jj_2_rtns.length; i++) jj_2_rtns[i] = new JJCalls();
2649  }
2650
2651  public JavaCCParser(JavaCCParserTokenManager tm) {
2652    token_source = tm;
2653    token = new Token();
2654    jj_ntk = -1;
2655    jj_gen = 0;
2656    for (int i = 0; i < 55; i++) jj_la1[i] = -1;
2657    for (int i = 0; i < jj_2_rtns.length; i++) jj_2_rtns[i] = new JJCalls();
2658  }
2659
2660  public void ReInit(JavaCCParserTokenManager tm) {
2661    token_source = tm;
2662    token = new Token();
2663    jj_ntk = -1;
2664    jj_gen = 0;
2665    for (int i = 0; i < 55; i++) jj_la1[i] = -1;
2666    for (int i = 0; i < jj_2_rtns.length; i++) jj_2_rtns[i] = new JJCalls();
2667  }
2668
```

```
2669  final private Token jj_consume_token(int kind) throws ParseException {
2670    Token oldToken;
2671    if ((oldToken = token).next != null) token = token.next;
2672    else token = token.next = token_source.getNextToken();
2673    jj_ntk = -1;
2674    if (token.kind == kind) {
2675      jj_gen++;
2676      if (++jj_gc > 100) {
2677        jj_gc = 0;
2678        for (int i = 0; i < jj_2_rtns.length; i++) {
2679          JJCalls c = jj_2_rtns[i];
2680          while (c != null) {
2681            if (c.gen < jj_gen) c.first = null;
2682            c = c.next;
2683          }
2684        }
2685      }
2686      return token;
2687    }
2688    token = oldToken;
2689    jj_kind = kind;
2690    throw generateParseException();
2691  }
2692
2693  static private final class LookaheadSuccess extends java.lang.Error { }
2694  final private LookaheadSuccess jj_ls = new LookaheadSuccess();
2695  final private boolean jj_scan_token(int kind) {
2696    if (jj_scanpos == jj_lastpos) {
2697      jj_la--;
2698      if (jj_scanpos.next == null) {
2699        jj_lastpos = jj_scanpos = jj_scanpos.next = token_source.getNextToken();
2700      } else {
2701        jj_lastpos = jj_scanpos = jj_scanpos.next;
2702      }
2703    } else {
2704      jj_scanpos = jj_scanpos.next;
2705    }
2706    if (jj_rescan) {
2707      int i = 0; Token tok = token;
2708      while (tok != null && tok != jj_scanpos) { i++; tok = tok.next; }
2709      if (tok != null) jj_add_error_token(kind, i);
2710    }
2711    if (jj_scanpos.kind != kind) return true;
2712    if (jj_la == 0 && jj_scanpos == jj_lastpos) throw jj_ls;
2713    return false;
2714  }
2715
2716  final public Token getNextToken() {
2717    if (token.next != null) token = token.next;
2718    else token = token.next = token_source.getNextToken();
2719    jj_ntk = -1;
2720    jj_gen++;
2721    return token;
2722  }
2723
2724  final public Token getToken(int index) {
2725    Token t = lookingAhead ? jj_scanpos : token;
2726    for (int i = 0; i < index; i++) {
2727      if (t.next != null) t = t.next;
2728      else t = t.next = token_source.getNextToken();
2729    }
2730    return t;
2731  }
2732
2733  final private int jj_ntk() {
2734    if ((jj_nt=token.next) == null)
2735      return (jj_ntk = (token.next=token_source.getNextToken()).kind);
2736    else
2737      return (jj_ntk = jj_nt.kind);
```

```
2738  }
2739
2740  private java.util.Vector jj_expentries = new java.util.Vector();
2741  private int[] jj_expentry;
2742  private int jj_kind = -1;
2743  private int[] jj_lasttokens = new int[100];
2744  private int jj_endpos;
2745
2746  private void jj_add_error_token(int kind, int pos) {
2747    if (pos >= 100) return;
2748    if (pos == jj_endpos + 1) {
2749      jj_lasttokens[jj_endpos++] = kind;
2750    } else if (jj_endpos != 0) {
2751      jj_expentry = new int[jj_endpos];
2752      for (int i = 0; i < jj_endpos; i++) {
2753        jj_expentry[i] = jj_lasttokens[i];
2754      }
2755      boolean exists = false;
2756      for (java.util.Enumeration e = jj_expentries.elements();
e.hasMoreElements();) {
2757        int[] oldentry = (int[])(e.nextElement());
2758        if (oldentry.length == jj_expentry.length) {
2759          exists = true;
2760          for (int i = 0; i < jj_expentry.length; i++) {
2761            if (oldentry[i] != jj_expentry[i]) {
2762              exists = false;
2763              break;
2764            }
2765          }
2766          if (exists) break;
2767        }
2768      }
2769      if (!exists) jj_expentries.addElement(jj_expentry);
2770      if (pos != 0) jj_lasttokens[(jj_endpos = pos) - 1] = kind;
2771    }
2772  }
2773
2774  public ParseException generateParseException() {
2775    jj_expentries.removeAllElements();
2776    boolean[] la1tokens = new boolean[61];
2777    for (int i = 0; i < 61; i++) {
2778      la1tokens[i] = false;
2779    }
2780    if (jj_kind >= 0) {
2781      la1tokens[jj_kind] = true;
2782      jj_kind = -1;
2783    }
2784    for (int i = 0; i < 55; i++) {
2785      if (jj_la1[i] == jj_gen) {
2786        for (int j = 0; j < 32; j++) {
2787          if ((jj_la1_0[i] & (1<<j)) != 0) {
2788            la1tokens[j] = true;
2789          }
2790          if ((jj_la1_1[i] & (1<<j)) != 0) {
2791            la1tokens[32+j] = true;
2792          }
2793        }
2794      }
2795    }
2796    for (int i = 0; i < 61; i++) {
2797      if (la1tokens[i]) {
2798        jj_expentry = new int[1];
2799        jj_expentry[0] = i;
2800        jj_expentries.addElement(jj_expentry);
2801      }
2802    }
2803    jj_endpos = 0;
2804    jj_rescan_token();
2805    jj_add_error_token(0, 0);
```

```
2806    int[][] exptokseq = new int[jj_expentries.size()][];
2807    for (int i = 0; i < jj_expentries.size(); i++) {
2808      exptokseq[i] = (int[])jj_expentries.elementAt(i);
2809    }
2810    return new ParseException(token, exptokseq, tokenImage);
2811  }
2812
2813  final public void enable_tracing() {
2814  }
2815
2816  final public void disable_tracing() {
2817  }
2818
2819  final private void jj_rescan_token() {
2820    jj_rescan = true;
2821    for (int i = 0; i < 11; i++) {
2822    try {
2823      JJCalls p = jj_2_rtns[i];
2824      do {
2825        if (p.gen > jj_gen) {
2826          jj_la = p.arg; jj_lastpos = jj_scanpos = p.first;
2827          switch (i) {
2828            case 0: jj_3_1(); break;
2829            case 1: jj_3_2(); break;
2830            case 2: jj_3_3(); break;
2831            case 3: jj_3_4(); break;
2832            case 4: jj_3_5(); break;
2833            case 5: jj_3_6(); break;
2834            case 6: jj_3_7(); break;
2835            case 7: jj_3_8(); break;
2836            case 8: jj_3_9(); break;
2837            case 9: jj_3_10(); break;
2838            case 10: jj_3_11(); break;
2839          }
2840        }
2841        p = p.next;
2842      } while (p != null);
2843      } catch(LookaheadSuccess ls) { }
2844    }
2845    jj_rescan = false;
2846  }
2847
2848  final private void jj_save(int index, int xla) {
2849    JJCalls p = jj_2_rtns[index];
2850    while (p.gen > jj_gen) {
2851      if (p.next == null) { p = p.next = new JJCalls(); break; }
2852      p = p.next;
2853    }
2854    p.gen = jj_gen + xla - jj_la; p.first = token; p.arg = xla;
2855  }
2856
2857  static final class JJCalls {
2858    int gen;
2859    Token first;
2860    int arg;
2861    JJCalls next;
2862  }
2863
2864}
2865
```