

Utah State University

DigitalCommons@USU

All Graduate Theses and Dissertations

Graduate Studies

5-2014

MULTIVERSEJAVA

Vishal Sharma

Utah State University

Follow this and additional works at: <https://digitalcommons.usu.edu/etd>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Sharma, Vishal, "MULTIVERSEJAVA" (2014). *All Graduate Theses and Dissertations*. 3079.

<https://digitalcommons.usu.edu/etd/3079>

This Thesis is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Theses and Dissertations by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.



MULTIVERSEJAVA<TEMPORAL>

by

Vishal Sharma

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

Dr. Curtis Dyreson
Major Professor

Dr. Nicholas Flann
Committee Member

Dr. Daniel Watson
Committee Member

Dr. Mark R. McLellan
Vice President for Research and
Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2014

Copyright © Vishal Sharma 2014

All Rights Reserved

ABSTRACT

MultiverseJava<*Temporal*>

by

Vishal Sharma, Master of Science

Utah State University, 2014

Major Professor: Dr. Curtis Dyreson
Department: Computer Science

Sequenced semantics, which was first proposed in the context of temporal databases, is the semantics for the evaluation of a program on values annotated with time metadata. The time metadata records when each value is “live” (is valid or has existence). Sequenced semantics stipulates that a computation on values annotated with temporal metadata must be equivalent to, in effect, running the computation at every time point with only the values alive at that time. Sequenced semantics is challenging to program because it is more than just a re-interpretation of the run-time behavior of a program; for instance, a sequenced “if-else” statement may need to evaluate both the “true” and “false” branches, in different time slices of the computation.

This thesis introduces *MultiverseJava*. MultiverseJava supports sequenced semantics for time stamped values in a Java program. Programmers currently have to resort to ad hoc methods to implement sequenced semantics in Java programs; hence, a better approach is needed. We show how MultiverseJava can be implemented using a MultiverseJava to Java translation. The translation layer weaves support for computing with the timestamped values into a Java program. This thesis describes the MultiverseJava architecture, the layer, semantic templates, and experiments to quantify the cost of MultiverseJava.

(109 pages)

PUBLIC ABSTRACT

MultiverseJava<*Temporal*>

Vishal Sharma

MultiverseJava supports sequenced semantics for timestamped values in a Java program. Programmers currently have to resort to ad hoc methods to implement sequenced semantics in Java programs; hence, a better approach is needed. We show how MultiverseJava can be implemented using a MultiverseJava to Java translation. The translation layer weaves support for computing with the timestamped values into a Java program. This thesis describes the MultiverseJava architecture, the layer, semantic templates, and experiments to quantify the cost of MultiverseJava.

To my mother and Jasmin...

ACKNOWLEDGMENTS

This research project has been an excellent learning curve for me, both academically and personally. A good advisor is always a very important factor in determining the success of a graduate student. I have the best one. I would like to thank my advisor and major professor, Dr. Curtis Dyreson, for trusting my abilities and accepting me as a research assistant. His unwavering support and patience with a novice like me, and his willingness to listen, help and answer all my questions, are a few of the things I am really thankful for. I would also like to thank him for supporting me financially.

I would like to thank Dr. Nicholas Flann and Dr. Dan Watson for being a part of my graduate committee and for their help during the course of education at Utah State University. I would also like to thank the CSE Department at USU for the financial support I have received throughout my master's program.

I would like to thank Jasmin for all her support and help. It is because of her, I am able to succeed so far. I owe my success to my mother and Jasmin; thank you for supporting me morally and giving me the most appropriate advice, always. I can never thank them enough for all that they have done for me. It was not possible without them.

Vishal Sharma

CONTENTS

	Page
ABSTRACT	iii
PUBLIC ABSTRACT	iv
ACKNOWLEDGMENTS	vi
LIST OF FIGURES	viii
CHAPTER	
1 INTRODUCTION	1
1.1 Background	2
1.1.1 The Multiverse $\langle X \rangle$ Paradigm	2
1.1.2 Temporal Databases	3
1.1.3 Sequenced Semantics	4
1.1.4 Passing values to Java	5
1.2 What Is MultiverseJava?	5
1.2.1 The <code>@multiverse</code> Annotation	5
1.2.2 Multiverse Computation	7
1.2.3 Multiverse Flow of Control	8
1.2.4 Promoting a Universe to a Multiverse construct	9
1.2.5 Demoting a Multiverse to a Universe construct	9
2 TEMPLATES FOR JAVA STATEMENTS	11
2.1 Denotational Semantics	11
2.2 Example of template translation	18
3 ARCHITECTURE OF MUTIVERSEJAVA	21
3.1 Multiversed Architecture	21
3.2 Classes Description	21
3.3 Parsing with ANTLRWorks	23
4 OBJECT-ORIENTED CHALLENGES	26
5 CONCLUSION AND FUTURE WORK	27
References	28
Appendix	30

LIST OF FIGURES

Figure	Page
1.1 Sequenced semantics	5
1.2 Data annotated with metadata (data about data)	6
1.3 Traditional way of computing	7
1.4 Multiverse way of computing	8
1.5 Flow of control in Traditional IF statement	9
1.6 Flow of control in Multiversed IF statement	9
1.7 Computation between Multiverse and Non-Multiverse	10
1.8 Degrading Multiverse to non-Multiverse	10
2.1 Example Java code	18
2.2 A programmer annotates the code in Figure 2.1 to use the multiverse programming paradigm	19
2.3 Translation of the code in Figure 2.2 to Java.	20
3.1 System Architecture	22
3.2 Multiversed Integer	22
3.3 Stages of Compiler	23
3.4 ANTLRWorks Java grammar modification	25

CHAPTER 1

INTRODUCTION

The context of this research is the fertile nexus between programming languages (PLs) and databases (DBs). Historically, PLs have made a tremendous impact on DBs. The application of object-oriented ideas to DBs in the 1990s led to the development of object-oriented and object-relational DBs [1]. The 1990s were also when research in Datalog peaked [2]. Datalog combined logic programming with DBs and greatly advanced query optimization techniques, especially for recursive queries [3]. Currently, concepts from functional PLs are finding a home in the analysis of Big Data using Map/Reduce; Map and Reduce are well-known higher-order functions [4]. First-class support for functional motifs has also contributed to integrating PLs and DB query languages using systems like LINQ [5].

DBs, in contrast, have not had a significant impact on PLs. PLs have been extended in various ways to interface with databases through embedded languages and APIs, and there was a brief flurry of interest in persistent object-oriented languages. But no new PL paradigm or methodology has emerged from DBs, even though many programs manipulate data from DBs. DBs too often have been perceived by the PL community as a black box, where all that is needed in a PL is a way to put data into and extract data from the box.

Previous research in the PL/DB nexus has overlooked one aspect of data that we envision could have an important impact on PLs: *metadata*. Metadata in a DB context is often simplified to mean the schema or organization of the data [6]. But more broadly, metadata includes data that annotates and extends the meaning of data [7]. It is the broader sense that we will use metadata in this thesis. Many different kinds of metadata exist: temporal, security, privacy, reliability, quality, lineage, and measures of incompleteness (e.g., probability).

When a value is extracted from a DB black box into a PL, it is usually stripped of its metadata, since a PL can only compute with an unannotated value. Consider for instance the following simple

code snippet from a Java program.

```

...
int x;
...
if (x < 3) x = 1;
else x = 0;
...

```

After the evaluation of the `if` the value of variable `x` is either 0 or 1 (exclusive, one or the other but not both). But now assume that initially the value of `x` is pulled out of a temporal database and annotated with time metadata such that at time t_1 it is 2, but at time t_2 it is 9. When annotated with time metadata, the `if` condition is *true* at time t_1 and *false* at time t_2 . (The value is *undefined* or *non-existent* at all times other than t_1 or t_2 .) *Both* branches of the `if` should be evaluated in different time slices, and after evaluation `x` should be both 1 (at time t_1) and 0 (at time t_2) (and *undefined* at other times).

In this thesis we focus on temporal metadata as an example of how a PL can be changed to provide better support for data with metadata, but more generally our technique is applicable to programming with a diverse panoply of “interesting values” that emerge from databases which store data annotated with metadata.

1.1 Background

1.1.1 The Multiverse $\langle X \rangle$ Paradigm

In a *multiverse programming paradigm* a program is evaluated in multiple *program universes*, where each program universe is described by an *intersection* of metadata. The multiverse paradigm is novel, and can best be explained by contrasting the multiverse paradigm with the non-multiverse or *universe programming paradigm*, with which all programmers are familiar since it is how programs are normally evaluated. In the universe paradigm, there is only one program universe, which we will call the *default universe*. The default universe does not have explicit metadata, instead,

it has implicit metadata and all values have exactly the same metadata. Since all values share the same metadata in the default universe, the metadata can be entirely ignored in program evaluation. For example, suppose that values are annotated with temporal metadata. The temporal metadata describes *when* each value exists. In the default universe a value always exists, hence, it implicitly is annotated with “all of time.” (Or it could be assumed to exist only at the current time and be annotated with the time *now*.) The annotation is irrelevant to computation since all the metadata is the same, hence all computation is carried out in a single programming universe. But when values are annotated with different metadata more than one programming universe emerges. For temporal metadata, if two values are annotated with different times, then the metadata describes different program universes. In some universe one or both of the values may not exist.

The multiverse paradigm is parameterized by X , which is a list of metadata *domains*. In general a domain is a set of metadata values and an “intersection” function to compute the metadata common to a set of metadata values. Example domains include temporal, security, and privacy. For a temporal domain a metadata value is a set of times and the intersection function computes temporal intersection. Times can be represented compactly using temporal periods [8], e.g., the set of times from t_s through (including) time t_e can be represented as $[t_s, t_e]$. The temporal *overlaps* constructor computes the intersection of a set of periods. For a security domain, the metadata would be levels of security in a partial order, and the intersection operation would be least upper bound (LUB).

In this thesis we will focus on a single domain: *time*. Our goal is to provide better support for programmers of user-defined functions (UDFs) in temporal databases. Hence this thesis is titled `MultiverseJava<Temporal>`. The next three sections give a background of temporal databases and UDFs.

1.1.2 Temporal Databases

A temporal database is a database that provides special support for *time*. The temporal database community has recognized three primary kinds of time: transaction, valid, and user-defined time [8]. *Transaction time* is the time at which a fact is “live” in a system usually the time between when it

is created and deleted. *Valid time* is the real-world time of a fact. Both valid and transaction time are kinds of *metadata*, that is, they are data *about* a fact. In contrast, *user-defined time* is a time value within a data structure, i.e., user-defined time is field that happens to be a time value. As an example consider the fact that Joe was born in 1978, is employed at the ACME company from 2010 through 2012, and that this fact was created in the DBMS in 2011. One way to model the fact of Joe’s employment is with a user-defined time for his date of birth (1978), a valid time interval of [2010-2012] representing the real-world time of his employment, and a transaction time interval of [2011-*until changed*] (since the fact is current and has not yet been deleted, it lives in the database “until changed” [9]).

1.1.3 Sequenced Semantics

Time data can in general be treated as other kinds of data, but time metadata requires special handling. This is because the time metadata modifies the data, restricting its use in some contexts. In particular, a common semantics for time metadata is *sequenced semantics* [10]. Sequenced semantics defines the correctness criteria for the efficient evaluation of temporal programs and prevents, in effect, facts from different slices of a data collection being incorrectly mixed. An example of sequenced semantics is sketched in Figure 1.1. In the figure, a Java program is to be evaluated on a *temporal relation*. A temporal relation is a bag of tuples where each tuple has some number of “Data” attribute values annotated with “Time” (metadata) timestamps. The timestamp records the lifetime of the value in a temporal dimension. For example a *temporal period* [11] timestamp, $[t_b, t_e]$, records that the lifetime is from time t_b to time t_e (inclusive). The evaluation of a Java program on a temporal relation obeys sequenced semantics if, logically, it yields the same *result* as slicing the relation at each time point, evaluating each slice using the Java program, and then coalescing or recombining the slices into a single temporal relation with time metadata. As relations can be enormous, the slice-based evaluation strategy is infeasible in practice since a large number of slices (one for every time point) would need to be created and evaluated. Sequenced semantics ensures the correct and efficient processing of temporal relations, without computing each slice individually.

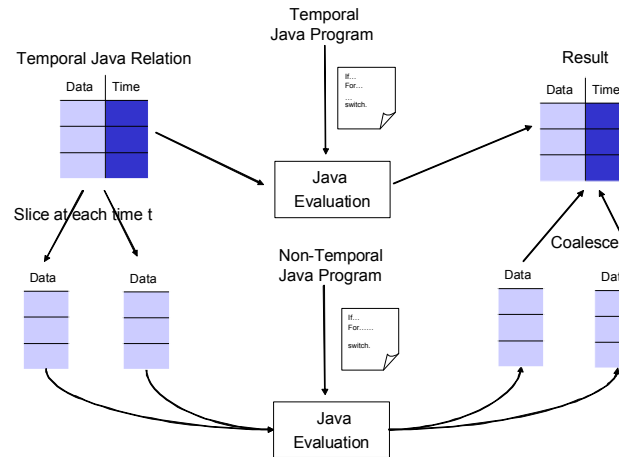


Figure 1.1: Sequenced semantics

1.1.4 Passing values to Java

It is common for a database query to call a *user-defined function (UDF)*. A user-defined function is an arbitrary piece of Java (or other language) code that is invoked during evaluation of a program. It is invoked for every value in a column, and the result of the function is substituted for the value. UDFs are a common part of database query languages, e.g., they can be invoked from SQL and XQuery queries. The problem is that a temporal value has temporal metadata that is ignored by the UDF, that is, the UDF is not evaluated using sequenced semantics. In this thesis, we show how to rewrite a UDF in Java to obey sequenced semantics.

1.2 What Is MultiverseJava?

MultiverseJava is Java with support for the multiverse programming paradigm. We implement MultiverseJava with a MultiverseJava to Java translator and a library of metadata-sensitive tools. In this section we briefly describe MultiverseJava from a programmer's perspective.

1.2.1 The @multiverse Annotation

In MultiverseJava, the annotation `@multiverse` on a language construct (values, expressions or statements) indicates that the construct should be evaluated using the multiverse paradigm. As an example consider two variables `x` and `y` in the following code:

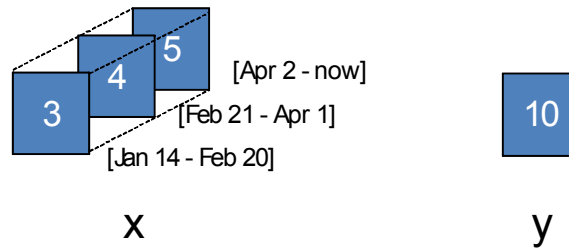


Figure 1.2: Data annotated with metadata (data about data)

```

...
@multiverse
int x;
...
int y;
...

```

The code represents that `x` is a multiverse `int` and `y` is a non-multiverse `int` variable. Because `x` is multiverse it can have different values in different multiverses. As shown in Figure 1.2 `x` has three different values at three different times where data in the shaded box is the value of `x` and the time for which the value is valid is in brackets. `x` has value 3 from Jan 14 - Feb 20, 4 from Feb 21 - Apr 1, and 5 from Apr 2 - now. It can also be viewed as a table with a value and time stamp as shown below:

Value	timestamp(start, end)
3	(Jan 14, Feb 20)
4	(Feb 21, Apr 1)
5	(Apr 2, now)

Figure 1.2 also represents value and metadata of `y` (which is a non-multiverse integer) in a multiverse environment.

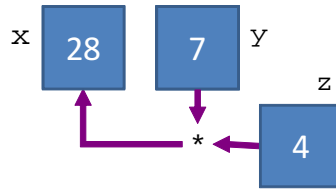


Figure 1.3: Traditional way of computing

1.2.2 Multiverse Computation

Computation is a process of following an algorithm to produce definite outputs. In this section, we will see how the multiverse way of computing is different from the traditional way of computing. Figure 1.3 depicts the traditional way of computing where the given expressions are given below:

```
...
x = y * z;
...
```

Notice that x is calculated by multiplying y , which is 7, and z , which is 4, yielding 28. Figure 1.4 shows the multiverse way of computation. Below is the code representation of Figure 1.4.

```
...
@multiverse
int y
@multiverse
int z
@multiverse
x = y * z;
...
```

Operations on multiverse values are performed on the time they both are “live”. In the above example for calculating the value of x which is the product of y and z we calculate the temporal intersection between y and z and multiply the associated values. In this case Jan 14 – Feb 20 and Apr 2 – now are when x and y intersect. The result is calculated by multiplying the values

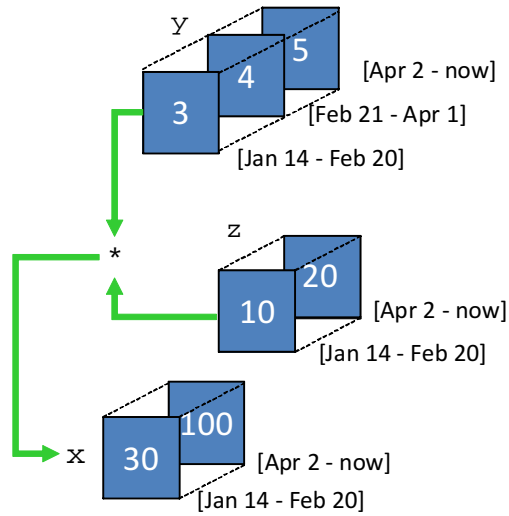


Figure 1.4: Multiverse way of computing

from both y and z , which is $3 * 10$ at time Jan 14 - Feb 20 and $5 * 20$ at time Apr 2 - now. Notice that value 4 from y is omitted in the result since the time associated with it does not intersect with any time from z . There is no multiverse in which y with value 4 is alive with a z value.

1.2.3 Multiverse Flow of Control

Traditionally, computation is in a single universe (single execution flow), but computation on values annotated with metadata has multiple flows. Figure 1.5 and Figure 1.6 show the flow of control of an `if` statement in the universe and multiverse paradigm. Figure 1.5 depicts that control starts from `if` and executes the condition. If the condition evaluates to true it executes the `then` branch otherwise it executes `else` branch. In this case the value of y is assumed to be less than 10 so it executes the `else` branch only. In a MultiverseJava paradigm, both are potentially executed. As shown in Figure 1.6, in some universe the condition may evaluate to `true`. For instance if y is 23 from Jun 3 - Apr 5, then the `then` branch should be executed. But there may be some universe in which the condition evaluates to `false` and the `else` branch is executed. Potentially, both the `true` and `false` branches are executed.

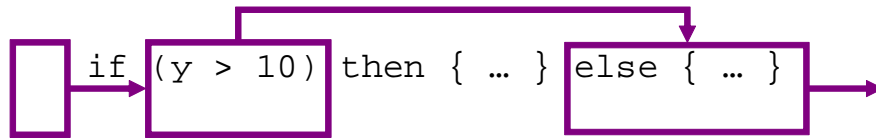


Figure 1.5: Flow of control in Traditional IF statement

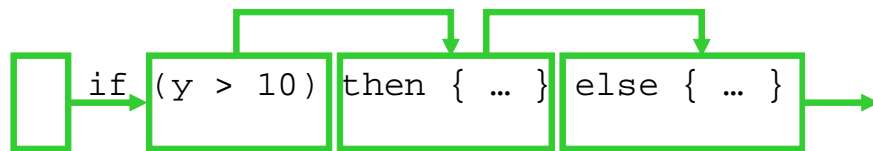


Figure 1.6: Flow of control in Multiversed IF statement

1.2.4 Promoting a Universe to a Multiverse construct

A universe construct is *promoted* to a multiverse construct by adding default metadata. Figure 1.7 describes an operation between a non-multiversed `int` and a multiversed `int`. In this case the non-multiversed value `z` is considered to be “all time” and it is multiplied to all of the values of `y` because each time interval in `y` will intersect all time of `z`. So the result is multiplication of all values of `y` with the value of `z`.

1.2.5 Demoting a Multiverse to a Universe construct

A multiverse construct is *demoted* to a universe construct by stripping its metadata. Figure 1.8 describes an operation between two non-multiversed `int` and the result is stored in a multiversed `int`. In this case the non-multiversed values `z` and `y` are multiplied. So the result is 30 which is multiplication of 3 and 10. Multiversed `int` `x` is converted to a universe construct and assigned the value 30.

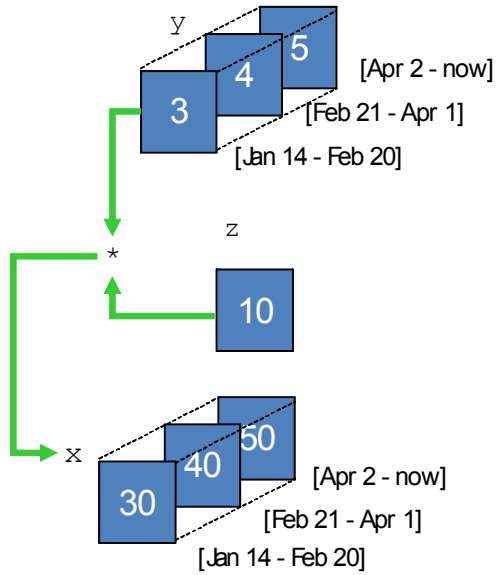


Figure 1.7: Computation between Multiverse and Non-Multiverse

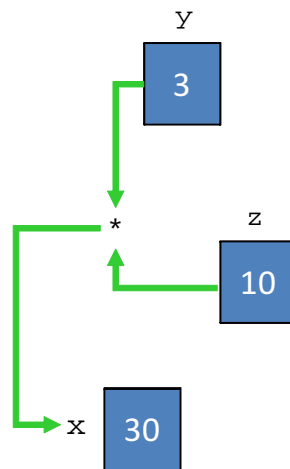


Figure 1.8: Degrading Multiverse to non-Multiverse

CHAPTER 2

TEMPLATES FOR JAVA STATEMENTS

The Multiverse paradigm changes the semantics or meaning of each statements. This section gives the denotational semantics for `MultiverseJava<Temporal>`. After giving the semantics we show how the templates are used with an example.

2.1 Denotational Semantics

Assume that the state, $T = [T_s, T_r]$, consists of variables

- T_s - stack of `Time` variables, and
- T_r - a (sequenced) variable that holds the return value.

We define the following “macros”.

- t_t - Time variable on the top of the stack, *i.e.*, $t_t = T_s.\text{peek}()$
- $T \circ t$ - establish t as the current time variable, *i.e.*, $T_s.\text{push}(t)$ to start parsing a construct and $T_s.\text{pop}()$ after parsing.
- $T \circ \perp$ - establish loop boundary, \perp , when parsing
- $\varphi(t, X)$ - `multiverse.Boolean` constructor, e.g. `new multiverse.Boolean(t, X)`

Non-multiverse

$$\llbracket @\text{nonmultiverse } X \rrbracket(T) \equiv X$$

Note that in the semantics, to reduce clutter, we use the annotation `@nonmultiverse` to switch to a universe programming paradigm, other than the annotation `@multiverse` to turn on the multiverse paradigm. This is syntactic sugar.

Constant

$$\llbracket \text{constant} \rrbracket (T) \equiv \text{constant}$$

Identifier

$$\llbracket \text{var} \rrbracket (T) \equiv \text{var}$$

Assignment

$$\llbracket V=E \rrbracket (T) \equiv \llbracket V \rrbracket (T) . \text{assign}(t_t, \llbracket E \rrbracket (T))$$

Right-associative binary operation

$$\llbracket E_1 \text{ op } E_2 \rrbracket (T) \equiv \llbracket E_1 \rrbracket (T) . \text{op}(t_t, \llbracket E_2 \rrbracket (T))$$

Left-associative binary operation

$$\llbracket E_1 \text{ op } E_2 \rrbracket (T) \equiv \llbracket E_2 \rrbracket (T) . \text{op}(t_t, \llbracket E_1 \rrbracket (T))$$

Unary operation

$$\llbracket \text{op } E \rrbracket (T) \equiv \llbracket E \rrbracket (T) . \text{op}(t_t)$$

Ternary (conditional) operation

$$\llbracket (C) ? E_1 : E_2 \rrbracket (T) \equiv$$

```

// C may have side effects, execute only once
// Allocate variables to collect the time of C
private static Time t1 = null;
private static Time t2 = null;
// Build a function to store the true and false times
private Boolean F() {
    TimePair tc =  $\varphi(t_t, \llbracket C \rrbracket (T))$ .getWhen();

```

```

    t1 = tc.whenTrue ();
    t2 = tc.whenFalse ();
    return true;
}
// Evaluate alternatives at true and false times
// F executed for side effects, expression returns a sequenced type
(F) ?  $\llbracket E_1 \rrbracket (T \circ t_1)$ .temporalUnion ( $\llbracket E_2 \rrbracket (T \circ t_2)$ )
(*) Note that F, t1, and t2 are generated names

```

If

```

 $\llbracket \text{if } (C) S_1 [\text{else } S_2] \rrbracket (T) \equiv$ 
{
    // Capture time of C
    TimePair tc =  $\varphi(t_t, \llbracket C \rrbracket (T))$ .getWhen ();
    // Evaluate true branch at times C is true
    Time t = Q.whenTrue ();
    if (!t.empty ())  $\llbracket S_1 \rrbracket (T \circ t)$ 
    // Evaluate false branch at times C is false
    t = tc.whenFalse ();
    if (!t.empty ())  $\llbracket S_2 \rrbracket (T \circ t)$ 
}
(*) Note that t, and tc are generated names

```

While

```

 $\llbracket \text{while } (C) \text{ do } S \rrbracket (T) \equiv$ 
while (true) do {
    TimePair tc =  $\llbracket C \rrbracket (T)$ .getWhen ();
    Time t = tc.whenTrue ();

```

```

    if (t.empty()) break;
     $\llbracket S \rrbracket(T \circ \perp \circ t)$ 
}

```

Break (from loop)

```

 $\llbracket \text{break} \rrbracket(T) \equiv$ 
    // Remove current time,  $t_t$ , from times in loop on stack
    // For every  $t_i$  on the stack before the first  $\perp$ 
     $t_i.\text{subtractTime}(t_t)$ ;

```

Continue (in loop)

```

 $\llbracket \text{continue} \rrbracket(T) \equiv$ 
    // Set current time (top of TimeStack) to empty
     $t_t = \text{new Time}(\text{null})$ ;

```

Return

```

 $\llbracket \text{return } V \rrbracket(T) \equiv$ 
    // Add to multiversed return value
     $T_r = T_r.\text{merge}(\llbracket V \rrbracket(T))$ ;
    // Remove current time,  $t_t$ , from times in function on stack
    // For every  $t_i$  on the stack
     $t_i.\text{subtractTime}(t_t)$ ;

```

GOTO

```

 $\llbracket \text{GOTO } L; \dots L: \dots \rrbracket(T) \equiv$ 
    {
        Allocate a variable to accumulate the “GOTO” time
         $\text{Time } t = \text{new Time}(\text{null})$ ;

```

```

...
// Replace GOTO by setting current time to null
tt = new Time (null);
...
// Replace L, merge GOTO time with current time
L:
tt.union (t);
t = new Time (null);
...
}

```

Try-catch (only partially multiversed)

```

// Java does not support “resume” exception-handling semantics,
// exceptions resume after end of block, hence can’t be fully sequenced
[[try { S1 ; ... ; Sn } catch E]](T) ≡
  try { [[S1]](T) } catch E
...
  try { [[Sn]](T) } catch E

```

Sequencing

```

[[S1 ; S2]](T) ≡
  [[S1]](T);
  [[S2]](T);

```

Do

```

[[do S while ( C )]](T) ≡
  {
    Time t = tt;

```



```

do {
   $\llbracket S \rrbracket(T \circ \perp \circ t)$ 
  TimePair  $t_c = \llbracket C \rrbracket(T).timePair()$ 
  if ( $t_c.whenTrue().empty()$ ) break;
} while (true);
}

```

For

```

 $\llbracket \text{for } (I; C; K) S \rrbracket(T) \equiv$ 
 $\llbracket I \rrbracket(T)$ 
while (true) do {
  TimePair  $t_c = \llbracket C \rrbracket(T).getWhen();$ 
  Time  $t = t_c.whenTrue();$ 
  if ( $t.empty()$ ) break;
   $\llbracket S \rrbracket(T \circ \perp \circ t)$ 
   $\llbracket K \rrbracket(T \circ \perp \circ t)$ 
}

```

For each

```

 $\llbracket \text{for } (TV: C) S \rrbracket(T) \equiv$ 
multiverse. $TV;$ 
while (true) do {
  TimePair  $t_c = \varphi(t, \llbracket C \rrbracket(T)).getWhen();$ 
  Time  $t = t_c.whenTrue();$ 
  if ( $t.empty()$ ) break;
   $\llbracket S \rrbracket(T \circ \perp \circ t)$ 
   $\llbracket K \rrbracket(T \circ \perp \circ t)$ 
}

```

Method call

$$\llbracket \text{obj.method}(E_1, \dots, E_n) \rrbracket(T) \equiv$$

$$\text{obj.method}(t_t, \llbracket E_1 \rrbracket(T), \dots, \llbracket E_n \rrbracket(T))$$
Method def

$$\llbracket \text{modifiers } N(\dots, Y_i P_i, \dots) \{ S \} \rrbracket(T) \equiv$$

$$\text{modifiers } N(\text{Time } t, \dots, \llbracket Y_i P_i \rrbracket(T \circ t), \dots)$$
Formal param

$$\llbracket Y P \rrbracket(T) \equiv$$

$$\llbracket Y \rrbracket(T) P$$
Type

$$\llbracket Y \rrbracket(T) \equiv \text{multiverse.Y}$$
Class def

$$\llbracket \text{class } N \{ S \} \rrbracket(T) \equiv$$

$$\text{class } N \text{ extends Multiverse } \{ \llbracket S \rrbracket(T) \}$$
Package def

$$\llbracket \text{package } N \rrbracket(T) \equiv \text{package multiverse.N}$$

```
int test=0;
if (test==1)
{
    test+=10;
}
else
{
    while(test>10)
    {
        test++;
        break;
    }
}
```

Figure 2.1: Example Java code

2.2 Example of template translation

This section depicts usage of denotational semantics with an example. The code given in Figure 2.1 is in the universe programming paradigm. The user adds @multiverse annotations to convert the program to the multiverse paradigm.

Note in Figure 2.3: lines with // are clarifying comments. Comments (//) are not part of the actual code translation.

```
/**
 * Input
 **/
@multiverse
int test=0;
@multiverse
if (test==1)
{
    test+=10;
}
else
{
    @multiverse
    while(test>10)
    {
        test++;
        @multiverse
        break;
    }
}
```

Figure 2.2: A programmer annotates the code in Figure 2.1 to use the multiverse programming paradigm

```

/**
 * Ouptut
 **/
// int test=0;
sequenced.Integer test = new sequenced.Integer(timeMap, 0);

// if (test==1)
TimeMapPair tp21 = (test.equal(allTime, 1).timeMapPair());
TimeMap tTrue22 = tp21.trueTimeMap();
if (tTrue22.notEmpty())
{
    // test+=10;
    test.plusAssign(tTrue22,10);
}

// else
TimeMap tFalse23 = tp21.falseTimeMap();
if (tFalse23.notEmpty())
{
    // while(test>10)
    while (true)
    {
        TimeMapPair tp24 = (test.greaterThan(tFalse23, 10).timeMapPair());
        TimeMap tTrue25 = tp24.trueTimeMap();
        if (tTrue25.empty())
        {
            break;
        }
        // test++;
        test.increment(tTrue25, 12);

        // break;
        tp21.subtractTime(tTrue25);
        tpAllTime.subtractTime(tTrue25);
    }
}
}

```

Figure 2.3: Translation of the code in Figure 2.2 to Java.

CHAPTER 3

ARCHITECTURE OF MUTIVERSEJAVA

This section describes architecture of MultiverseJava and the layer built over the Java parser. It also explains how the Java grammar modifications are performed using ANTLRWorks.

3.1 Multiversed Architecture

The Multiverse architecture is shown in Figure 3.1. A sample program in `program.mvjava` is input to our Multiverse Java translator which converts the program to a Java program and the final output is passed to the Java compiler. The architecture of the translation is shown in Figure 3.1.

- Input code is converted to character stream `CharStream`.
- `CharStream` is passed to `JavaLexer`.
- Result is converted to `tokens` of `TokenRewriteStream`. `TokenRewriteStream` is used to modify tokens to produce Multiverse Java version.
- `tokens` are passed to Java parser which generates Parser Tree followed by Abstract Syntax Tree. This is the *modification stage* where we use `TokenRewriteStream` to modify the given input.
- Finally we print out the modified code.

Details about the MultiverseJava translator using ANTLRWorks are given in section 3.3.

3.2 Classes Description

Code generation using ANTRLWorks was the first step, the next step is to build classes for multiverse package. This requires each expression to be rewritten and this is the stage where we provide

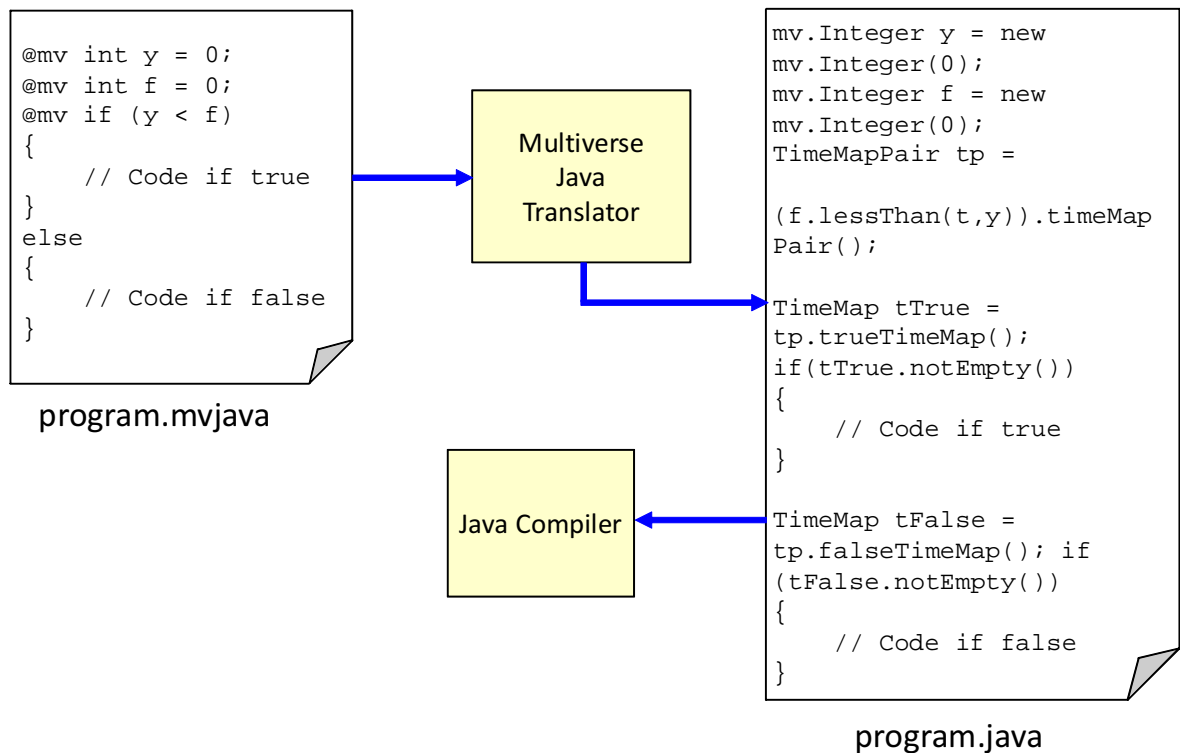


Figure 3.1: System Architecture

<code>y>>f;</code>	<code>-></code>	<code>y.bitShiftRight (timeMap, f);</code>
<code>f=f^y;</code>	<code>-></code>	<code>f.assign (timeMap, y.xor (timeMap, f));</code>
<code>f==y;</code>	<code>-></code>	<code>f.equal (timeMap, y);</code>
<code>x=x+f;</code>	<code>-></code>	<code>x.assign (timeMap, f.add (timeMap, x));</code>

Figure 3.2: Multiversed Integer

temporal support/operation in the code. All the operations will be based on time that annotated a value.

Assuming `y` and `f` are multiverse integers Figure 3.2 is the list of few operations on `int`. The method name written on right are equivalent to the operations used on the left. In the function call `timeMap` is the time variable which represents the time for which this operation will be executed. The evaluation in the methods will be based on `timeMap`, by finding the intersection of times from metadata associated with operands and `timeMap`. The final piece is the second operand of the expression.

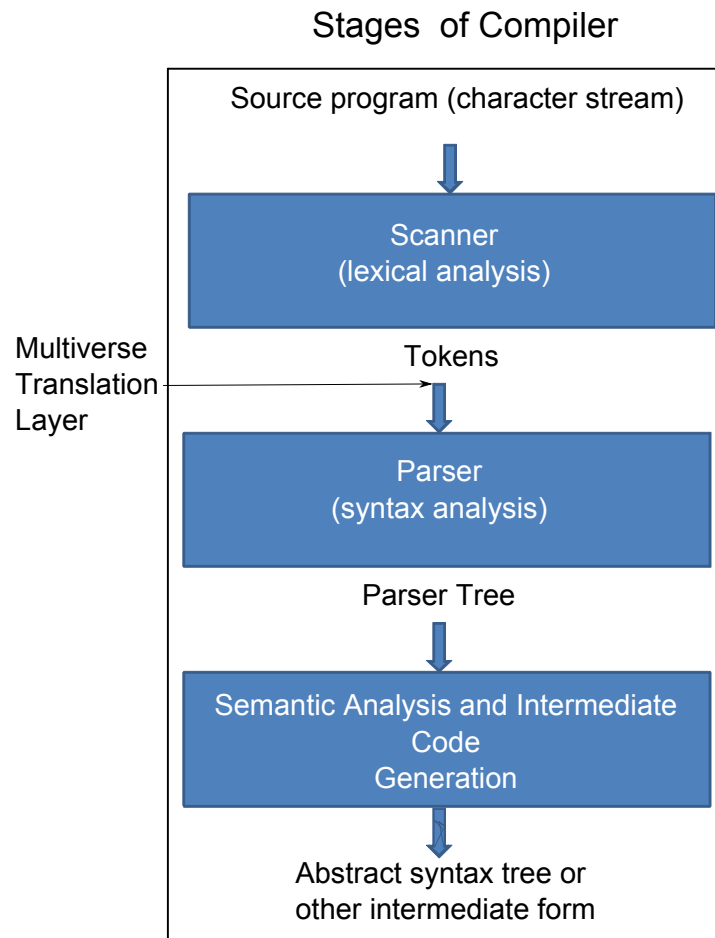


Figure 3.3: Stages of Compiler

3.3 Parsing with ANTLRWorks

In our approach to modify Java we are using annotations in the input and based on the annotations we modify the source to multiverse code. During translation the code annotations are removed as the abstract syntax tree is built. We perform all the code modifications before the abstract syntax tree is built. To perform this task we use ANTLRWorks 1.5.2 and Java grammar which are open source.

ANTLRWorks is a tool for writing the grammar of a programming languages. The main purpose of ANTLRWorks is to make grammar of programming language more accessible so that it can be improved and easily maintained. ANTLR based `Java.g` contains the Java grammar which

generates Java parser Tree and modified `Java.g` is the translating layer over source/input code.

Parsing of source code starts with conversion of source code to `character stream`. The `character stream` is passed to the Java lexer which converts the stream to `tokens`. The `TokenRewriteStream` is used to store tokens and we perform all the modification required on tokens using the `TokenRewriteStream` as shown in Figure 3.4. The `TokenRewriteStream` is a class library present in ANTLRWorks and it is useful in dumping out the input stream after performing some augmentation or manipulations. We can use this to insert, delete, remove, replace, convert to string etc. on input stream. This is very efficient because all operations are lazily performed which makes sure that the data is not moving around all the time. We can also imagine this as insert, delete, modifying a node in a `LinkedList`, where the data is not moved just the pointers are manipulated. Figure 3.4 represents an operation `result = x + (y * z)` in the universe programming paradigm and its equivalent outcome in the multiverse paradigm. Below code snippet is used to perform actions shown in Figure 3.4

```
tokens.replace($ASSIGN, ".assign(");
tokens.replace($ADD, ".add(");
tokens.replace($STAR, ".multiply(");
```

Individual operations are converted to their respective method call and the result is given below:

```
result.assign(..., x.add(..., y.multiply(..., z)));
```

NOTE: “...” represents time variable as explained in previous section.

Rewritten tokens are then passed to the parser. A Java Parser Tree is generated followed by Abstract Syntax tree and then the final code.

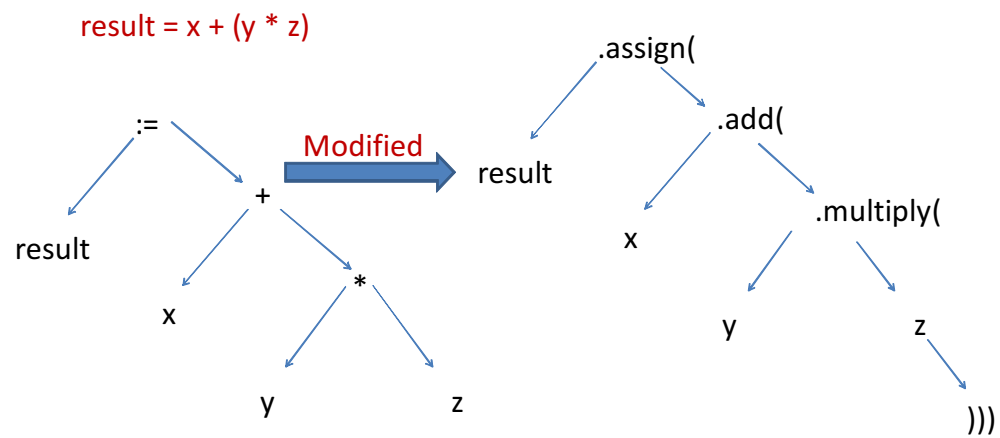


Figure 3.4: ANTLRWorks Java grammar modification

CHAPTER 4

OBJECT-ORIENTED CHALLENGES

This chapter describes several Object Oriented challenges faced during building library for giving support for temporal operations.

1. Number of libraries in Java: Java built-in library is enormous. Building support for all of them is a really challenging task. It will require alot of code to be rewritten. Moreoever it can also degrade the performance.
2. Multiverse Classes and Objects: Currently we have translation layer only for primitive data types (int, byte, long, short, char, boolean, long, float, double) of Java but building support for User-Defined Classes is a challenging task because of OO concepts used in User-Defined classes viz. Polymorphism, Inheritance etc.

CHAPTER 5

CONCLUSION AND FUTURE WORK

This thesis introduces *MultiverseJava*. MultiverseJava supports sequenced semantics for time-stamped values in a Java program. We show how MultiverseJava can be implemented using a MultiverseJava to Java translation. The translation layer weaves support for computing with the time-stamped values into a Java program. This thesis describes the MultiverseJava architecture, the layer, semantic templates, and experiments to quantify the cost of MultiverseJava.

Currently we use *time* as metadata. In future, *Security* and *privacy* can be used as metadata. Building a debudding tool to debug code generated using translation can be very helpful. This paradigm can be adapted in other programming languages. Applying multiverse semantics to Javascript will be lot less challenging compared to Java because of duck typing. It will allow us to overcome several object oriented challenges we are currently facing in Java.

Currently we have to define all statements using annotations. In future, using fewer annotations to infer others.

REFERENCES

- [1] François Bancilhon and Won Kim. “Object-oriented database systems: In transition”. *SIG-MOD Rec.*, Vol. 19,(No. 4):pp 49–53, December 1990.
- [2] Jeffrey D. Ullman. “The database approach to knowledge representation”. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI. Portland, Oregon, 1996, pp 1346-1348.
- [3] Seppo Sippu and Eljas Soisalon-Soininen. “An analysis of magic sets and related optimization strategies for logic queries”. *J. ACM*, Vol. 43,(No. 6):pp. 1046–1088, November 1996.
- [4] Jeffrey Dean and Sanjay Ghemawat. “Mapreduce: Simplified data processing on large clusters”. *Commun. ACM*, Vol. 51,(No. 1):pp. 107–113, January 2008.
- [5] James Cheney, Sam Lindley, and Philip Wadler. “A practical theory of language-integrated query”. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, ICFP ’13, New York, NY, 2013. pp. 403–416.
- [6] Ramez A. Elmasri and Shankrant B. Navathe. *Fundamentals of Database Systems*. 3rd edition, Boston, MA:Addison-Wesley Longman Publishing Co., 1999.
- [7] Curtis E. Dyreson, Omar U. Florez, Akshay Thakre, and Vishal Sharma. “Supporting data aspects in pig latin”. In *Proceedings of the 12th Annual International Conference on Aspect-oriented Software Development*, AOSD ’13, Fukuoka, Japan, 2013. ACM, pp 13-24.
- [8] Curtis Dyreson, Fabio Grandi, Wolfgang Käfer, Nick Kline, Nikos Lorentzos, Yannis Mitsopoulos, Angelo Montanari, Daniel Nonen, Elisa Peressi, Barbara Pernici, John F. Roddick, Nandlal L. Sarda, Maria Rita Scalas, Arie Segev, Richard Thomas Snodgrass, Mike D. Soo,

- Abdullah Tansel, Paolo Tiberio, and Gio Wiederhold. “A consensus glossary of temporal database concepts”. *SIGMOD Rec.*, Vol. 23,(No. 1):pp 52–64, March 1994.
- [9] Curtis E. Dyreson, Christian S. Jensen, and Richard T. Snodgrass. “Now in temporal databases”. In *Encyclopedia of Database Systems*. 2009, pp. 1920-1924.
- [10] Michael H. Böhlen and Christian S. Jensen. “Sequenced semantics”. In *Encyclopedia of Database Systems*, pages pp. 2619–2621. 2009.
- [11] Curtis Dyreson, Fabio Grandi, Wolfgang Käfer, Nick Kline, Nikos Lorentzos, Yannis Mitsopoulos, Angelo Montanari, Daniel Nonen, Elisa Peressi, Barbara Pernici, John F. Roddick, Nandlal L. Sarda, Maria Rita Scalas, Arie Segev, Richard Thomas Snodgrass, Mike D. Soo, Abdullah Tansel, Paolo Tiberio, and Gio Wiederhold. *A Consensus Glossary of Temporal Database Concepts*. *SIGMOD Rec.*, Vol. 23(No. 1):pp. 52–64, March 1994.
- [12] Igor Timko, Curtis E. Dyreson, and Torben Bach Pedersen. “A probabilistic data model and algebra for location-based data warehouses and their implementation”. *GeoInformatica*, Vol. 18,(No. 2):pp 357–403, 2014.

APPENDIX

```
1  /**
2   * An ANTLRv3 capable Java 1.5 grammar for building ASTs.
3   *
4   * Note that there's also the tree grammar 'JavaTreeParser.g' that can
5   * be fed
6   * with this grammer's output.
7   *
8   * Please report any detected errors or even suggestions regarding this
9   * grammar
10  * to
11  *
12  * dieter [D O T] habelitz [A T] habelitz [D O T] com
13  *
14  * with the subject
15  *
16  * jsom grammar: [your subject note]
17  *
18  * To generate a parser based on this grammar you'll need ANTLRv3, which
19  * you can
20  * get from 'http://www.antlr.org'.
21  *
22  * This grammar is published under the ...
23  *
24  * BSD licence
25  *
26  * Copyright (c) 2007–2008 by HABELITZ Software Developments
27  *
28  * All rights reserved.
29  *
30  * http://www.habelitz.com
31  *
32  * Redistribution and use in source and binary forms, with or without
33  * modification, are permitted provided that the following conditions
34  * are met:
35  *
36  * 1. Redistributions of source code must retain the above copyright
37  * notice, this list of conditions and the following disclaimer.
38  * 2. Redistributions in binary form must reproduce the above copyright
39  * notice, this list of conditions and the following disclaimer in
   * the
```



```

73     AND_ASSIGN      = '&='      ;
74     ASSIGN          = '='       ;
75     AT              = '@'       ;
76     BIT_SHIFT_RIGHT = '>>>'    ;
77     BIT_SHIFT_RIGHT_ASSIGN = '>>>=' ;
78     COLON           = ':'       ;
79     COMMA           = ','       ;
80     DEC             = '—'       ;
81     DIV             = '/'       ;
82     DIV_ASSIGN      = '/='      ;
83     DOT             = '.'       ;
84     DOTSTAR         = '.*'      ;
85     ELLIPSIS        = '...'    ;
86     EQUAL           = '=='      ;
87     GREATER_OR_EQUAL = '>='     ;
88     GREATER_THAN   = '>'      ;
89     INC             = '++'      ;
90     LBRACK          = '['       ;
91     LCURLY          = '{'       ;
92     LESS_OR_EQUAL   = '<='     ;
93     LESS_THAN       = '<'      ;
94     LOGICAL_AND     = '&&'     ;
95     LOGICAL_NOT     = '!'       ;
96     LOGICAL_OR      = '||'     ;
97     LPAREN          = '('       ;
98     MINUS           = '-'       ;
99     MINUS_ASSIGN    = '-='     ;
100    MOD              = '%'       ;
101    MOD_ASSIGN       = '%='     ;
102    NOT              = '~'       ;
103    NOT_EQUAL        = '!='     ;
104    OR               = '|'       ;
105    OR_ASSIGN        = '|='     ;
106    PLUS            = '+'       ;
107    PLUS_ASSIGN      = '+='     ;
108    QUESTION         = '?'       ;
109    RBRACK           = ']'       ;
110    RCURLY           = '}'       ;
111    RPAREN           = ')'       ;
112    SEMI             = ';'       ;
113    SHIFT_LEFT       = '<<'     ;
114    SHIFT_LEFT_ASSIGN = '<<='   ;
115    SHIFT_RIGHT      = '>>'     ;
116    SHIFT_RIGHT_ASSIGN = '>>='  ;

```

```
117     STAR                = '*'                ;
118     STAR_ASSIGN         = '*='             ;
119     XOR                  = '^'                ;
120     XOR_ASSIGN          = '^='             ;
121
122     // keywords
123
124     ABSTRACT             = 'abstract'        ;
125     ASSERT               = 'assert'          ;
126     BOOLEAN              = 'boolean'         ;
127     BREAK                = 'break'           ;
128     BYTE                  = 'byte'           ;
129     CASE                  = 'case'           ;
130     CATCH                 = 'catch'          ;
131     CHAR                  = 'char'           ;
132     CLASS                 = 'class'          ;
133     CONTINUE              = 'continue'       ;
134     DEFAULT               = 'default'        ;
135     DO                    = 'do'             ;
136     DOUBLE                = 'double'         ;
137     ELSE                  = 'else'           ;
138     ENUM                  = 'enum'           ;
139     EXTENDS               = 'extends'        ;
140     FALSE                  = 'false'         ;
141     FINAL                 = 'final'          ;
142     FINALLY               = 'finally'        ;
143     FLOAT                  = 'float'          ;
144     FOR                    = 'for'           ;
145     IF                     = 'if'            ;
146     IMPLEMENTS            = 'implements'     ;
147     INSTANCEOF            = 'instanceof'     ;
148     INTERFACE             = 'interface'      ;
149     IMPORT                 = 'import'         ;
150     INT                    = 'int'           ;
151     LONG                   = 'long'          ;
152     NATIVE                 = 'native'         ;
153     NEW                    = 'new'           ;
154     NULL                   = 'null'          ;
155     PACKAGE                = 'package'       ;
156     PRIVATE                = 'private'       ;
157     PROTECTED              = 'protected'     ;
158     PUBLIC                 = 'public'        ;
159     RETURN                 = 'return'        ;
160     SHORT                  = 'short'         ;
```

```
161     STATIC                = 'static'          ;
162     STRICTFP              = 'strictfp'        ;
163     SUPER                  = 'super'          ;
164     SWITCH                  = 'switch'         ;
165     SYNCHRONIZED           = 'synchronized'    ;
166     THIS                    = 'this'          ;
167     THROW                   = 'throw'         ;
168     THROWS                  = 'throws'        ;
169     TRANSIENT               = 'transient'     ;
170     TRUE                     = 'true'         ;
171     TRY                      = 'try'          ;
172     VOID                     = 'void'         ;
173     VOLATILE                 = 'volatile'     ;
174     WHILE                    = 'while'        ;
175
176     // tokens for imaginary nodes
177
178     ANNOTATION_INIT_ARRAY_ELEMENT;
179     ANNOTATION_INIT_BLOCK;
180     ANNOTATION_INIT_DEFAULT_KEY;
181     ANNOTATION_INIT_KEY_LIST;
182     ANNOTATION_LIST;
183     ANNOTATION_METHOD_DECL;
184     ANNOTATION_SCOPE;
185     ANNOTATION_TOP_LEVEL_SCOPE;
186     ARGUMENT_LIST;
187     ARRAY_DECLARATOR;
188     ARRAY_DECLARATOR_LIST;
189     ARRAY_ELEMENT_ACCESS;
190     ARRAY_INITIALIZER;
191     BLOCK_SCOPE;
192     CAST_EXPR;
193     CATCH_CLAUSE_LIST;
194     CLASS_CONSTRUCTOR_CALL;
195     CLASS_INSTANCE_INITIALIZER;
196     CLASS_STATIC_INITIALIZER;
197     CLASS_TOP_LEVEL_SCOPE;
198     CONSTRUCTOR_DECL;
199     ENUM_TOP_LEVEL_SCOPE;
200     EXPR;
201     EXTENDS_BOUND_LIST;
202     EXTENDS_CLAUSE;
203     FOR_CONDITION;
204     FOR_EACH;
```

```
205     FOR_INIT ;
206     FOR_UPDATE ;
207     FORMAL_PARAM_LIST ;
208     FORMAL_PARAM_STD_DECL ;
209     FORMAL_PARAM_VARARG_DECL ;
210     FUNCTION_METHOD_DECL ;
211     GENERIC_TYPE_ARG_LIST ;
212     GENERIC_TYPE_PARAM_LIST ;
213     INTERFACE_TOP_LEVEL_SCOPE ;
214     IMPLEMENTS_CLAUSE ;
215     LABELED_STATEMENT ;
216     LOCAL_MODIFIER_LIST ;
217     JAVA_SOURCE ;
218     METHOD_CALL ;
219     MODIFIER_LIST ;
220     PARENTESIZED_EXPR ;
221     POST_DEC ;
222     POST_INC ;
223     PRE_DEC ;
224     PRE_INC ;
225     QUALIFIED_TYPE_IDENT ;
226     STATIC_ARRAY_CREATOR ;
227     SUPER_CONSTRUCTOR_CALL ;
228     SWITCH_BLOCK_LABEL_LIST ;
229     THIS_CONSTRUCTOR_CALL ;
230     THROWS_CLAUSE ;
231     TYPE ;
232     UNARY_MINUS ;
233     UNARY_PLUS ;
234     VAR_DECLARATION ;
235     VAR_DECLARATOR ;
236     VAR_DECLARATOR_LIST ;
237     VOID_METHOD_DECL ;
238 }
239
240 @header {
241     package sequencedjava . grammar ;
242
243 }
244
245 @members {
246
247     private boolean mMessageCollectionEnabled = false ;
248     private boolean mHasErrors = false ;
```

```

249     private List<String> mMessages;
250
251     TokenRewriteStream tokens;
252
253     Boolean isSequence = false;
254     Boolean temp = false;
255     String ident="";
256     Boolean forStart = false;
257     Boolean forBlock = false;
258     String relationalOperator ="";
259     SymbolTable st = new SymbolTable();
260     String opr = "" ;
261     Boolean doIt;
262     String doDont="";
263
264     public void setTokenStream(TokenRewriteStream input) {
265         //super.setTreeNodeStream(input);
266         tokens = input;
267     }
268
269     Stack<String> timeMapStack = new Stack<String>();
270     Stack<String> timeMapPairStack = new Stack<String>();
271     TimeMap allTime = new TimeMap();
272     Boolean sequencedIf = false;
273     Boolean ifSeqStatement = false;
274     String ifcondition = "";
275     int count = 0;
276     Boolean sequencedElse = false;
277     Boolean seqIf = false;
278
279     String forcondition = "";
280     Boolean whileBlock = false;
281     Boolean doWhile = false;
282     Boolean doPara = false;
283
284     Boolean switchCase = false;
285     Stack<String> cases = new Stack<String>();
286         Stack<String> statements = new Stack<String>();
287         Boolean isBreak = false;
288         Boolean leftCurly = false;
289         Boolean rightCurly = false;
290
291         Boolean breakTrue = false;
292         Boolean methodBlock= false;

```

```
293         Iterator x;
294         String returnType = "void";
295         Boolean sequencedReturn = false;
296         Boolean methodCall = false;
297         //variable for operation identification
298
299         final int and = 1;
300         final int and_assign = 2;
301         final int assign = 3;
302         final int bit_shift_right = 4;
303         final int bit_shift_right_assign = 5;
304         final int dec = 6;
305         final int div = 7;
306         final int div_assign = 8;
307         final int equal = 9;
308         final int greater_or_equal = 10;
309         final int greater_than = 11;
310         final int inc = 12;
311         final int less_or_equal = 13;
312         final int less_than = 14;
313         final int logical_and = 15;
314         final int logical_not = 16;
315         final int logical_or = 17;
316         final int minus = 18;
317         final int minus_assign = 19;
318         final int mod = 20;
319         final int mod_assign = 21;
320         final int not = 22;
321         final int not_equal = 23;
322         final int or = 24;
323         final int or_assign = 25;
324         final int plus = 26;
325         final int plus_assign = 27;
326         final int shift_left = 28;
327         final int shift_left_assign = 29;
328         final int shift_right = 30;
329         final int shift_right_assign = 31;
330         final int star = 32;
331         final int star_assign = 33;
332         final int xor = 34;
333         final int xor_assign = 35;
334
335
336         /**
```

```

337      * Switches error message collection on or of.
338      *
339      * The standard destination for parser error messages is <code>
      System.err</code>.
340      * However, if <code>>true</code> gets passed to this method this
      default
341      * behaviour will be switched off and all error messages will be
      collected
342      * instead of written to anywhere.
343      *
344      * The default value is <code>>false</code>.
345      *
346      * @param pNewState <code>>true</code> if error messages should be
      collected.
347      */
348      public void enableErrorMessageCollection(boolean pNewState) {
349          mMessageCollectionEnabled = pNewState;
350          if (mMessages == null && mMessageCollectionEnabled) {
351              mMessages = new ArrayList<String>();
352          }
353      }
354
355      /**
356      * Collects an error message or passes the error message to <code>
      super.emitErrorMessage(...)</code>.
357      *
358      *
359      * The actual behaviour depends on whether collecting error
      messages
360      * has been enabled or not.
361      *
362      * @param pMessage The error message.
363      */
364      @Override
365      public void emitErrorMessage(String pMessage) {
366          if (mMessageCollectionEnabled) {
367              mMessages.add(pMessage);
368          } else {
369              super.emitErrorMessage(pMessage);
370          }
371      }
372
373      /**
374      * Returns collected error messages.
375      *

```



```

376     * @return A list holding collected error messages or <code>null</
      code> if
377     *         collecting error messages hasn't been enabled. Of
      course, this
378     *         list may be empty if no error message has been emitted.
379     */
380     public List<String> getMessages() {
381         return mMessages;
382     }
383
384     /**
385     * Tells if parsing a Java source has caused any error messages.
386     *
387     * @return <code>true</code> if parsing a Java source has caused
      at least one error message.
388     */
389     public boolean hasErrors() {
390         return mHasErrors;
391     }
392 }
393
394 @lexer::header {
395     package sequencedjava.grammar;
396 }
397
398 @lexer::members {
399     /**
400     * Determines if whitespaces and comments should be preserved or thrown
      away.
401     *
402     * If <code>true</code> whitespaces and comments will be preserved
      within the
403     * hidden channel, otherwise the appropriate tokens will be skipped.
      This is
404     * a 'little bit' expensive, of course. If only one of the two
      behaviours is
405     * needed forever the lexer part of the grammar should be changed by
      replacing
406     * the 'if-else' stuff within the appropriate lexer grammar actions.
407     */
408     public boolean preserveWhitespacesAndComments = true;
409 }
410
411 /*

```

```

412     @rulecatch
413     {   catch(RecognitionException re) { throw re; } }
414     */
415
416 // Starting point for parsing a Java file.
417     javaSource
418         :   compilationUnit
419           -> ^(JAVA_SOURCE compilationUnit)
420         ;
421
422 compilationUnit
423     :
424         {
425             st.enterScope();
426             timeMapStack.push("allTime");
427             timeMapPairStack.push("tpAllTime");
428         }
429         annotationList
430         packageDeclaration?
431         importDeclaration*
432         typeDecls*
433         {
434             //System.out.println("last timeMapStack "+timeMapStack.peek())
435             ;
436             //System.out.println("last timeMapPairStack "+timeMapPairStack
437             .peek());
438             timeMapPairStack.pop();
439             timeMapStack.pop();
440         }
441         ;
442
443 typeDecls
444     :   typeDeclaration
445         |   SEMI!
446     ;
447
448 packageDeclaration
449     :   PACKAGE^ qualifiedIdentifier SEMI!
450     ;
451
452 importDeclaration

```

```

453         :   IMPORT^ STATIC? qualifiedIdentifier DOTSTAR? SEMI! //{tokens.
           insertBefore($IMPORT,"import java.util.Iterator;");}
454         ;
455
456     typeDeclaration
457         :   modifierList!
458             (   classTypeDeclaration[$modifierList.tree]
459                 |   interfaceTypeDeclaration[$modifierList.tree]
460                 |   enumTypeDeclaration[$modifierList.tree]
461                 |   annotationTypeDeclaration[$modifierList.tree]
462             )
463         ;
464
465     classTypeDeclaration[CommonTree modifiers]
466         :   CLASS IDENT genericTypeParameterList? classExtendsClause?
           implementsClause? classBody
467         -> ^(CLASS {$modifiers} IDENT genericTypeParameterList?
           classExtendsClause? implementsClause? classBody)
468         ;
469
470     classExtendsClause
471         :   EXTENDS type
472         -> ^(EXTENDS_CLAUSE[$EXTENDS, "EXTENDS_CLAUSE"] type)
473         ;
474
475     interfaceExtendsClause
476         :   EXTENDS typeList
477         -> ^(EXTENDS_CLAUSE[$EXTENDS, "EXTENDS_CLAUSE"] typeList)
478         ;
479
480     implementsClause
481         :   IMPLEMENTS typeList
482         -> ^(IMPLEMENTS_CLAUSE[$IMPLEMENTS, "IMPLEMENTS_CLAUSE"]
           typeList)
483         ;
484
485     genericTypeParameterList
486         :   LESS.THAN genericTypeParameter (COMMA genericTypeParameter)*
           genericTypeListClosing
487         -> ^(GENERIC_TYPE_PARAM_LIST[$LESS.THAN, "
           GENERIC_TYPE_PARAM_LIST"] genericTypeParameter+)
488         ;
489

```

```

490 genericTypeListClosing // This 'trick' is fairly dirty – if there's some
    time a better solution should
491 // be found to resolve the problem with nested
    generic type parameter lists
492 // (i.e. <T1 extends AnyType<T2>> for generic
    type parameters or <T1<T2>> for
493 // generic type arguments etc).
494 : GREATER_THAN
495 | SHIFT_RIGHT
496 | BIT_SHIFT_RIGHT
497 | // nothing
498 ;
499
500 genericTypeParameter
501 : IDENT bound?
502 → ^(IDENT bound?)
503 ;
504
505 bound
506 : EXTENDS type (AND type)*
507 → ^(EXTENDS_BOUND_LIST[$EXTENDS, "EXTENDS_BOUND_LIST"] type+)
508 ;
509
510 enumTypeDeclaration[CommonTree modifiers]
511 : ENUM IDENT implementsClause? enumBody
512 → ^(ENUM {$modifiers} IDENT implementsClause? enumBody)
513 ;
514
515 enumBody
516 : LCURLY {st.enterScope();} enumScopeDeclarations {st.exitScope();}
    RCURLY
517 → ^(ENUM_TOP_LEVEL_SCOPE[$LCURLY, "ENUM_TOP_LEVEL_SCOPE"]
    enumScopeDeclarations)
518 ;
519
520 enumScopeDeclarations
521 : enumConstants (COMMA!)? enumClassScopeDeclarations?
522 ;
523
524 enumClassScopeDeclarations
525 : SEMI classScopeDeclarations*
526 → ^(CLASS_TOP_LEVEL_SCOPE[$SEMI, "CLASS_TOP_LEVEL_SCOPE"]
    classScopeDeclarations*)
527 ;

```

```

528
529 enumConstants
530     :   enumConstant (COMMA! enumConstant)*
531     ;
532
533     enumConstant
534     :   annotationList IDENT^ arguments? classBody?
535     ;
536
537     interfaceTypeDeclaration [CommonTree modifiers]
538     :   INTERFACE IDENT genericTypeParameterList? interfaceExtendsClause
539         ? interfaceBody
540     -> ^(INTERFACE {$modifiers} IDENT genericTypeParameterList?
541         interfaceExtendsClause? interfaceBody)
542     ;
543
544     typeList
545     :   type (COMMA! type)*
546     ;
547
548     classBody
549     :   LCURLY {st.enterScope();} classScopeDeclarations* {st.exitScope
550         ();}RCURLY
551     -> ^(CLASS_TOP_LEVEL_SCOPE[$LCURLY, "CLASS_TOP_LEVEL_SCOPE"]
552         classScopeDeclarations*)
553     ;
554
555     interfaceBody
556     :   LCURLY {st.enterScope();} interfaceScopeDeclarations* {st.
557         exitScope();}RCURLY
558     -> ^(INTERFACE_TOP_LEVEL_SCOPE[$LCURLY, "CLASS_TOP_LEVEL_SCOPE"
559         ] interfaceScopeDeclarations*)
560     ;
561
562     classScopeDeclarations
563     :   block          -> ^(CLASS_INSTANCE_INITIALIZER block)
564     |   STATIC block   -> ^(CLASS_STATIC_INITIALIZER[$STATIC, "
565         CLASS_STATIC_INITIALIZER"] block)
566     |   modifierList
567     |   (   genericTypeParameterList?
568         (   q=type {if(isSequence) returnType=$q.text;} IDENT
569             formalParameterList arrayDeclaratorList? throwsClause? (
570                 block | SEMI)

```

```

562         -> ^(FUNCTION.METHOD.DECL modifierList
              genericTypeParameterList? type IDENT
              formalParameterList arrayDeclaratorList?
              throwsClause? block?)
563     | VOID IDENT formalParameterList throwsClause? (block |
      SEMI)
564     -> ^(VOID.METHOD.DECL modifierList
              genericTypeParameterList? IDENT formalParameterList
              throwsClause? block?)
565     | id=IDENT formalParameterList throwsClause? block
566     -> ^(CONSTRUCTOR.DECL[$id, "CONSTRUCTOR.DECL"]
              modifierList genericTypeParameterList?
              formalParameterList throwsClause? block)
567     )
568     | type classFieldDeclaratorList SEMI
569     -> ^(VAR.DECLARATION modifierList type
          classFieldDeclaratorList)
570     )
571     | typeDeclaration
572     | SEMI!
573     ;
574
575 interfaceScopeDeclarations
576     :   modifierList
577         (   genericTypeParameterList?
578             (   type IDENT formalParameterList arrayDeclaratorList?
                 throwsClause? SEMI
579                 -> ^(FUNCTION.METHOD.DECL modifierList
                     genericTypeParameterList? type IDENT
                     formalParameterList arrayDeclaratorList?
                     throwsClause?)
580             | VOID IDENT formalParameterList throwsClause? SEMI
581                 -> ^(VOID.METHOD.DECL modifierList
                     genericTypeParameterList? IDENT formalParameterList
                     throwsClause?)
582             )
583             | type interfaceFieldDeclaratorList SEMI
584                 -> ^(VAR.DECLARATION modifierList type
                     interfaceFieldDeclaratorList)
585             )
586             | typeDeclaration
587             | SEMI!
588             ;
589

```

```

590 classFieldDeclaratorList
591     :   q=classFieldDeclarator (COMMA classFieldDeclarator)*
592     -> ^(VAR_DECLARATOR_LIST classFieldDeclarator+)
593     ;
594
595 classFieldDeclarator
596     :   variableDeclaratorId (ASSIGN variableInitializer)?
597     -> ^(VAR_DECLARATOR variableDeclaratorId variableInitializer?)
598     ;
599
600 interfaceFieldDeclaratorList
601     :   interfaceFieldDeclarator (COMMA interfaceFieldDeclarator)*
602     -> ^(VAR_DECLARATOR_LIST interfaceFieldDeclarator+)
603     ;
604
605 interfaceFieldDeclarator
606     :   variableDeclaratorId ASSIGN variableInitializer
607     -> ^(VAR_DECLARATOR variableDeclaratorId variableInitializer)
608     ;
609
610 variableDeclaratorId
611     :   q=IDENT^ {
612         if (isSequence)
613         {
614             st.addId($q.text, true);
615             isSequence = false;
616         }
617     }
618     else
619     {
620         st.addId($q.text, false);
621     }
622     } arrayDeclaratorList?
623     ;
624
625 variableInitializer
626     :   arrayInitializer
627     |   expression
628     ;
629
630 arrayDeclarator
631     :   LBRACK RBRACK
632     -> ^(ARRAY_DECLARATOR)
633     ;

```

```

634
635 arrayDeclaratorList
636     :   arrayDeclarator+
637     -> ^(ARRAY_DECLARATOR_LIST arrayDeclarator+)
638     ;
639
640     arrayInitializer
641     :   LCURLY (variableInitializer (COMMA variableInitializer)* COMMA?)
642         ? RCURLY
643     -> ^(ARRAY_INITIALIZER[$LCURLY, "ARRAY_INITIALIZER"]
644         variableInitializer*)
645     ;
646
647 throwsClause
648     :   THROWS qualifiedIdentList
649     -> ^(THROWS_CLAUSE[$THROWS, "THROWS_CLAUSE"] qualifiedIdentList
650         )
651     ;
652
653 modifierList
654     :   modifier*
655     -> ^(MODIFIER_LIST modifier*)
656     ;
657
658 modifier
659     :   PUBLIC
660     |   PROTECTED
661     |   PRIVATE
662     |   STATIC
663     |   ABSTRACT
664     |   NATIVE
665     |   SYNCHRONIZED
666     |   TRANSIENT
667     |   VOLATILE
668     |   STRICTFP
669     |   localModifier
670     ;
671
672 localModifierList
673     :   localModifier*
674     -> ^(LOCAL_MODIFIER_LIST localModifier*)
675     ;
676
677 localModifier

```



```

675         :   FINAL
676         |   annotation
677         ;
678
679 type
680         :   simpleType
681         |   objectType
682         ;
683
684 simpleType // including static arrays of simple type elements
685         :   primitiveType arrayDeclaratorList?
686         -> ^(TYPE primitiveType arrayDeclaratorList?)
687         ;
688
689         objectType // including static arrays of object type reference elements
690         :   qualifiedTypeIdEnt arrayDeclaratorList?
691         -> ^(TYPE qualifiedTypeIdEnt arrayDeclaratorList?)
692         ;
693
694 objectTypeSimplified
695         :   qualifiedTypeIdEntSimplified arrayDeclaratorList?
696         -> ^(TYPE qualifiedTypeIdEntSimplified arrayDeclaratorList?)
697         ;
698
699 qualifiedTypeIdEnt
700         :   typeIdEnt (DOT typeIdEnt)*
701         -> ^(QUALIFIED.TYPE_IDENT typeIdEnt+)
702         ;
703
704 qualifiedTypeIdEntSimplified
705         :   typeIdEntSimplified (DOT typeIdEntSimplified)*
706         -> ^(QUALIFIED.TYPE_IDENT typeIdEntSimplified+)
707         ;
708
709 typeIdEnt
710         :   IDENT^
711         {
712             if (isSequence)
713             {
714                 tokens.replace($IDENT, "sequenced."+$IDENT.text);
715                 isSequence = false;
716                 ident = "sequenced."+$IDENT.text;
717             }
718         }

```

```

719         genericTypeArgumentList?
720         ;
721
722 typeIdentSimplified
723     :   IDENT^ genericTypeArgumentListSimplified?
724     ;
725
726 primitiveType
727     :   BOOLEAN
728         {
729             if (isSequence)
730             {
731                 tokens.replace($BOOLEAN, "sequenced.Boolean");
732                 ident = "sequenced.Boolean";
733             }
734         }
735     |   CHAR
736         {
737             if (isSequence)
738             {
739                 tokens.replace($CHAR, "sequenced.Char");
740                 ident = "sequenced.Char";
741             }
742         }
743     |   BYTE
744         {
745             if (isSequence)
746             {
747                 tokens.replace($BYTE, "sequenced.Byte");
748                 ident = "sequenced.Byte";
749             }
750         }
751     |   SHORT
752         {
753             if (isSequence)
754             {
755                 tokens.replace($SHORT, "sequenced.Short");
756                 ident = "sequenced.Short";
757             }
758         }
759     |   INT   {
760             if (isSequence)
761             {
762                 tokens.replace($INT, "sequenced.Integer");

```

```

763             ident = "sequenced.Integer";
764         }
765     }
766 |   LONG
767     {
768         if (isSequence)
769         {
770             tokens.replace($LONG, "sequenced.Long");
771             ident = "sequenced.Long";
772         }
773     }
774 |   FLOAT
775     {
776         if (isSequence)
777         {
778             tokens.replace($FLOAT, "sequenced.Float");
779             ident = "sequenced.Float";
780         }
781     }
782 |   DOUBLE
783     {
784         if (isSequence)
785         {
786             tokens.replace($DOUBLE, "sequenced.Double");
787             ident = "sequenced.Double";
788         }
789     }
790 ;
791
792
793 genericTypeArgumentList
794     :   LESS.THAN genericTypeArgument (COMMA genericTypeArgument)*
795         genericTypeListClosing
796     ->  ^(GENERIC_TYPE_ARG_LIST[$LESS.THAN, "GENERIC_TYPE_ARG_LIST"]
797         genericTypeArgument+)
798 ;
799
800 genericTypeArgument
801     :   type
802     |   QUESTION genericWildcardBoundType?
803     ->  ^(QUESTION genericWildcardBoundType?)
804 ;
805
806 genericWildcardBoundType

```

```

805         :   (EXTENDS | SUPER)^ type
806         ;
807
808 genericTypeArgumentListSimplified
809         :   LESS.THAN genericTypeArgumentSimplified (COMMA
810             genericTypeArgumentSimplified)* genericTypeListClosing
811         ->  ^(GENERIC_TYPE_ARG_LIST[$LESS.THAN, "GENERIC_TYPE_ARG_LIST"]
812             genericTypeArgumentSimplified+)
813         ;
814
815 genericTypeArgumentSimplified
816         :   type
817         |   QUESTION
818         ;
819
820 qualifiedIdentList
821         :   qualifiedIdentifier (COMMA! qualifiedIdentifier)*
822         ;
823
824 formalParameterList
825         :   LPAREN
826         (
827             {
828                 if(isSequence)
829                 {
830                     tokens.insertAfter($LPAREN, "TimeMap timeMap,");
831                     isSequence = false;
832                     methodBlock = true;
833                 }
834             }
835             // Contains at least one standard argument declaration and
836             // optionally a variable argument declaration.
837             formalParameterStandardDecl (COMMA
838                 formalParameterStandardDecl)* (COMMA
839                 formalParameterVarArgDecl)?
840             ->  ^(FORMAL_PARAM_LIST[$LPAREN, "FORMAL_PARAM_LIST"]
841                 formalParameterStandardDecl+ formalParameterVarArgDecl?)
842             // Contains a variable argument declaration only.
843             |   formalParameterVarArgDecl
844             ->  ^(FORMAL_PARAM_LIST[$LPAREN, "FORMAL_PARAM_LIST"]
845                 formalParameterVarArgDecl)
846             // Contains nothing.
847             |   ->  ^(FORMAL_PARAM_LIST[$LPAREN, "FORMAL_PARAM_LIST"] )
848         )

```

```

842         RPAREN
843     ;
844
845     formalParameterStandardDecl
846     :   localModifierList type variableDeclaratorId
847     -> ^(FORMAL_PARAM_STD_DECL localModifierList type
           variableDeclaratorId)
848     ;
849
850     formalParameterVarArgDecl
851     :   localModifierList type ELLIPSIS variableDeclaratorId
852     -> ^(FORMAL_PARAM_VARARG_DECL localModifierList type
           variableDeclaratorId)
853     ;
854
855     qualifiedIdentifier
856     :   (   i=IDENT   {
857             if(isSequence)
858             {
859                 st.addId($i.text, true);
860                 //isSequence=false;
861             }
862             else
863             {
864                 st.addId($i.text, false);
865             }
866             }           -> IDENT
867     )
868     (   DOT ident=IDENT   -> ^(DOT $qualifiedIdentifier $ident)
869     )*
870     ;
871
872     // ANNOTATIONS
873
874     annotationList
875     :   annotation*
876     -> ^(ANNOTATION_LIST annotation*)
877     ;
878
879     annotation
880     :   AT^ q=qualifiedIdentifier z=annotationInit?
881     {
882         //System.out.println("IN ");
883         if($q.text.equals("Sequence"))

```

```

884         {
885             isSequence = true;
886             temp = true;
887             //System.out.println("IN ");
888             if ($z.text != null)
889                 tokens.replace($AT.getTokenIndex(),
890                               $z.stop.getTokenIndex(), "");
891             else
892                 tokens.replace($AT.getTokenIndex(),
893                               $q.stop.getTokenIndex(), "");
894             //System.out.println("OUT ");
895         }
896     }
897     ;
898     annotationInit
899     :   LPAREN annotationInitializers RPAREN
900     →   ^(ANNOTATION_INIT_BLOCK[$LPAREN, "ANNOTATION_INIT_BLOCK"]
901           annotationInitializers)
902     ;
903     annotationInitializers
904     :   annotationInitializer (COMMA annotationInitializer)*
905     →   ^(ANNOTATION_INIT_KEY_LIST annotationInitializer+)
906     |   annotationElementValue // implicite initialization of the
907     |   annotation field 'value'
908     →   ^(ANNOTATION_INIT_DEFAULT_KEY annotationElementValue)
909     ;
910     annotationInitializer
911     :   IDENT^ ASSIGN! annotationElementValue
912     ;
913     annotationElementValue
914     :   annotationElementValueExpression
915     |   annotation
916     |   annotationElementValueArrayInitializer
917     ;
918     annotationElementValueExpression
919     :   conditionalExpression
920     →   ^(EXPR conditionalExpression)
921     ;
922     ;
923

```

```

924 annotationElementValueArrayInitializer
925     :   LCURLY (annotationElementValue (COMMA annotationElementValue)*)?
          (COMMA)? RCURLY
926     -> ^(ANNOTATION_INIT_ARRAY_ELEMENT[$LCURLY, "
          ANNOTATION_ELEM.VALUE_ARRAY_INIT"] annotationElementValue*)
927     ;
928
929 annotationTypeDeclaration[CommonTree modifiers]
930     :   AT INTERFACE IDENT annotationBody
931         -> ^(AT {$modifiers} IDENT annotationBody)
932     ;
933
934 annotationBody
935     :   LCURLY annotationScopeDeclarations* RCURLY
936     -> ^(ANNOTATION_TOP_LEVEL_SCOPE[$LCURLY, "CLASS_TOP_LEVEL_SCOPE
          "] annotationScopeDeclarations*)
937     ;
938
939 annotationScopeDeclarations
940     :   modifierList type
941         (   IDENT LPAREN RPAREN annotationDefaultValue? SEMI
942         -> ^(ANNOTATION_METHOD_DECL modifierList type IDENT
          annotationDefaultValue?)
943         |   classFieldDeclaratorList SEMI
944         -> ^(VAR_DECLARATION modifierList type
          classFieldDeclaratorList)
945         )
946     |   typeDeclaration
947     ;
948
949 annotationDefaultValue
950     :   DEFAULT^ annotationElementValue
951     ;
952
953 // STATEMENTS / BLOCKS
954
955 block
956     :   LCURLY
957         {
958         st.enterScope();
959
960
961         if(leftCurly)
962         {

```

```

963         tokens.replace($LCURLY, "");
964     }
965
966     // code to insert for Sequenced for Loop
967     if (forBlock)
968     {
969         timeMapPairStack.push("tp"+count);
970         count++;
971         timeMapStack.push("tTrue"+count);
972         count++;
973
974         tokens.insertAfter($LCURLY, "\n \t if (" + timeMapStack.peek() + ".
                empty()) break;");
975         tokens.insertAfter($LCURLY, "\n \t TimeMap "+timeMapStack.peek()
                + " = "+timeMapPairStack.peek()+".trueTimeMap();");
976         tokens.insertAfter($LCURLY, "\n \t TimeMapPair "+
                timeMapPairStack.peek()+ " = "+forcondition+".timeMapPair();
                ");
977
978         forBlock = false;
979     }
980
981     // code to insert in sequenced while block
982     if (whileBlock)
983     {
984         timeMapPairStack.push("tp"+count);
985         count++;
986         timeMapStack.push("tTrue"+count);
987         count++;
988
989         tokens.insertAfter($LCURLY, "\n \t if (" + timeMapStack.peek() + ".
                empty()) break;");
990         tokens.insertAfter($LCURLY, "\n \t TimeMap "+timeMapStack.peek()
                + " = "+timeMapPairStack.peek()+".trueTimeMap();");
991         tokens.insertAfter($LCURLY, "\n \t TimeMapPair "+
                timeMapPairStack.peek()+ " = "+forcondition+".timeMapPair();
                ");
992         //whileBlock = false;
993     }
994
995     if (methodBlock && !("void".equals(returnType)))
996     {
997         tokens.insertAfter($LCURLY, "\n"+returnType+" ret1 = new "+
                returnType+"("+timeMapPairStack.peek()+");");

```



```

998     methodBlock = false;
999     returnType="void";
1000     }
1001
1002     /* if( doWhile )
1003     {
1004         timeMapPairStack.push("tp"+count);
1005         count++;
1006         timeMapStack.push("tTrue"+count);
1007         count++;
1008
1009         tokens.insertAfter($LCURLY, "\n \t if (" + timeMapStack.peek() + ".
                empty()) break;");
1010         tokens.insertAfter($LCURLY, "\n \t TimeMap "+timeMapStack.peek()
                + " = "+timeMapPairStack.peek()+".trueTimeMap();");
1011         tokens.insertAfter($LCURLY, "\n \t TimeMapPair "+
                timeMapPairStack.peek()+ " = "+forcondition+".timeMapPair()
                ;");
1012     }*/
1013     }
1014     blockStatement* {st.exitScope();} RCURLY
1015
1016     {
1017         if(doPara)
1018         {
1019             tokens.replace($RCURLY, "");
1020             doPara = false;
1021         }
1022
1023         if(rightCurly)
1024         {
1025             tokens.replace($RCURLY, "");
1026         }
1027     }
1028
1029     -> ^(BLOCK_SCOPE[$LCURLY, "BLOCK_SCOPE"] blockStatement*)
1030
1031     ;
1032
1033 blockStatement
1034     :   localVariableDeclaration SEMI!
1035     |   typeDeclaration
1036     |   q=statement
1037     ;

```

```

1038
1039     localVariableDeclaration
1040         :   localModifierList type classFieldDeclaratorList
1041             -> ^(VAR.DECLARATION localModifierList type
                  classFieldDeclaratorList)
1042         ;
1043
1044
1045     statement
1046         :   block
1047             |   annotation statement
1048             |   ASSERT expr1=expression
1049             (   COLON expr2=expression SEMI
1050                 -> ^(ASSERT $expr1
1051                     $expr2)
1052                 |   SEMI
1053                 -> ^(ASSERT $expr1)
1054                 )
1055             |   IF
1056             {
1057                 if (isSequence)
1058                 {
1059                     sequencedIf = true;
1060                 }
1061             }
1062     q=parenthesizedExpression
1063     {
1064         ifcondition = $q.text;
1065         if (sequencedIf)
1066         {
1067             timeMapPairStack.push("tp"+count);
1068             count++;
1069             timeMapStack.push("tTrue"+count);
1070             count++;
1071
1072             tokens.replace($q.start.getTokenIndex(),$q.stop.
1073                 getTokenIndex(),"("+ timeMapStack.peek() +"
1074                 .notEmpty
1075                 ())");
1076             tokens.insertBefore($IF,"\t TimeMap "+ timeMapStack.peek() +
1077                 " = "+timeMapPairStack.peek()+" . trueTimeMap ();\n");
1078             tokens.insertBefore($IF,"TimeMapPair "+ timeMapPairStack.
1079                 peek() +" = "+ifcondition+" . timeMapPair ();\n");
1080         }
1081     }

```

```

1073         //sequencedIf = false;
1074         sequencedElse = true;
1075         seqIf = true;
1076         //timeMapStack.pop();
1077     }
1078 }
1079 ifStat=statement {if(sequencedIf) {timeMapStack.pop();}}
1080 ( ELSE
1081 {
1082     if(sequencedElse)
1083     {
1084         timeMapStack.push("tFalse"+count);
1085         count++;
1086         tokens.replace($ELSE,"\t if (" + timeMapStack.peek()
1087             +".notEmpty())");
1088         tokens.insertBefore($ELSE,"\t TimeMap "+ timeMapStack.
1089             peek() +" = "+timeMapPairStack.peek()+" .falseTimeMap
1090             ();\n");
1091         sequencedElse = false;
1092     }
1093 }
1094 elseStat=statement
1095 {
1096     // remove the timemap and timeMapPair
1097     timeMapStack.pop();
1098
1099     if(seqIf)
1100     {
1101         //System.out.println("In seqIf "+timeMapPairStack.peek()
1102             );
1103         timeMapPairStack.pop();
1104         seqIf = false;
1105         sequencedIf = false;
1106         sequencedElse = false;
1107     }
1108 }
1109     -> ^(IF parenthesizedExpression
1110         $ifStat $elseStat)
1111 |

```

```

1112     {
1113         if(seqIf)
1114         {
1115             //System.out.println("In seqIf "+timeMapPairStack.peek()
1116                 );
1117             timeMapPairStack.pop();
1118             seqIf = false;
1119             sequencedElse = false;
1120             sequencedIf = false;
1121         }
1122     }
1123     -> ^(IF parenthesizedExpression SifStat)
1124 )
1125 | FOR {
1126     if(isSequence)
1127     {
1128         forBlock=true;
1129         forStart = true;
1130     }
1131 }
1132 LPAREN
1133 ( forInit SEMI z=forCondition
1134 {
1135     if(forStart)
1136     {
1137         //System.out.println("for condition "+$z.text);
1138         forcondition = $z.text;
1139         tokens.replace($z.start.getTokenIndex(),$z.stop.
1140             getTokenIndex()+1,";");
1141         forStart = false;
1142     }
1143 }
1144 SEMI forUpdater RPAREN statement
1145 {
1146     // POPPING timeMap and TimeMapPair
1147     timeMapPairStack.pop();
1148     timeMapStack.pop();
1149 }
1150 -> ^(FOR forInit forCondition forUpdater statement)
1151 | localModifierList type IDENT COLON expression RPAREN
1152 statement -> ^(FOR.EACH[$FOR, "FOR.EACH"]
1153 localModifierList type IDENT expression statement)
1154
1155

```

```

1152     )
1153 | WHILE
1154     {
1155     if(isSequence)
1156     {
1157         whileBlock=true;
1158         //forStart = true;
1159     }
1160     }
1161     q=parenthesizedExpression
1162 {
1163     if(whileBlock)
1164     {
1165         forcondition = $q.text;
1166         tokens.replace($q.start.getTokenIndex(),$q.stop.
            getTokenIndex(),"(true)");
1167     }
1168 }
1169 statement
1170 {
1171     // POPPING timeMap and TimeMapPair
1172     if(whileBlock)
1173     {
1174         //System.out.println("In while "+timeMapPairStack.peek()
            );
1175         if(breakTrue == false)
1176         {
1177             timeMapPairStack.pop();
1178         }
1179
1180         timeMapStack.pop();
1181         whileBlock = false;
1182         breakTrue = false;
1183     }
1184 }
1185
1186 -> ^(WHILE parenthesizedExpression statement)
1187 | DO
1188 {
1189     if(isSequence)
1190     {
1191         doWhile = true;
1192         doPara = true;

```

```

1193         //tokens.insertBefore($DO,"\t TimeMap "+ timeMapStack.peek()
           + " = "+timeMapPairStack.peek()+".falseTimeMap();\n");
1194     }
1195 }
1196 statement WHILE q=parenthesizedExpression
1197 {
1198     if(doWhile)
1199     {
1200
1201         //code
1202         timeMapPairStack.push("tp"+count);
1203         count++;
1204         timeMapStack.push("tTrue"+count);
1205         count++;
1206
1207         tokens.insertBefore($WHILE,"\t TimeMap "+ timeMapStack.peek() + "
           = "+timeMapPairStack.peek()+".trueTimeMap();\n }\n");
1208         tokens.insertBefore($WHILE,"TimeMapPair "+ timeMapPairStack.peek()
           + " = "+$q.text+".timeMapPair();\n");
1209
1210         tokens.replace($q.start.getTokenIndex(),$q.stop.
           getTokenIndex(),"(true)");
1211         doWhile = false;
1212     }
1213 }
1214
1215 SEMI {timeMapStack.pop();
1216       timeMapPairStack.pop();}
1217
1218         -> ^(DO statement parenthesizedExpression)
1219 | TRY block (catches finallyClause? | finallyClause)
           -> ^(TRY block catches? finallyClause?)
1220 | SWITCH
1221 {
1222 if(isSequence)
1223     {
1224         switchCase = true;
1225         doPara = true;
1226         leftCurly = true;
1227         rightCurly = true;
1228     }
1229     tokens.replace($SWITCH,"");
1230 }

```

```

1231     q=parenthesizedExpression {tokens.replace($q.start.getTokenIndex(),
1232                               $q.stop.getTokenIndex(),"");} LCURLY {
1233         tokens.replace($LCURLY,"");
1234         st.enterScope();
1235     }
1236
1237     switchBlockLabels
1238     {
1239         st.exitScope();
1240         cases.removeAllElements();
1241         statements.removeAllElements();
1242         switchCase = false;
1243         doPara = false;
1244         leftCurly = false;
1245         rightCurly = false;
1246     }
1247     RCURLY {tokens.replace($RCURLY,"");}    -> ^(SWITCH
1248         parenthesizedExpression switchBlockLabels)
1249     | SYNCHRONIZED parenthesizedExpression block
1250         -> ^(SYNCHRONIZED
1251             parenthesizedExpression block)
1252     | RETURN {
1253     if(isSequence)
1254     {
1255         sequencedReturn = true;
1256     isSequence = false;
1257     }
1258     }
1259     q=expression?
1260     {
1261         if(sequencedReturn)
1262         {
1263             tokens.replace($RETURN,"ret1.merge(""+$q.text+"\n");
1264             String temp = timeMapStack.peek();
1265             tokens.insertAfter($RETURN,temp+" = new Time(null);");
1266             //tokens.replace($RETURN,"");
1267             tokens.replace($q.start.getTokenIndex(),$q.stop.getTokenIndex(),
1268                             "");
1269             sequencedReturn = false;
1270         }
1271     }

```

```

1268 SEMI
      -> ^(RETURN expression?)
1269 | THROW expression SEMI
      -> ^(THROW
      expression)
1270 | {isBreak = true;} BREAK
1271 {
1272   if (isSequence)
1273   {
1274     String temp = timeMapStack.peak();
1275     timeMapPairStack.pop();
1276     breakTrue = true;
1277
1278     x = timeMapPairStack.iterator();
1279
1280     while (x.hasNext())
1281     {
1282       tokens.insertAfter($BREAK, x.next()+" . subtractTime (" + temp
1283         + ");\n");
1284     }
1285     tokens.replace($BREAK, "");
1286     isSequence = false;
1287   }
1288 IDENT? SEMI
      -> ^(BREAK
      IDENT?)
1289 | CONTINUE
1290 {
1291   if (isSequence)
1292   {
1293     String temp = timeMapStack.peak();
1294     tokens.replace($CONTINUE, temp+" = new Time(null);");
1295     isSequence = false;
1296   }
1297 }
1298 IDENT? SEMI
      -> ^(
      CONTINUE IDENT?)
1299 | IDENT COLON statement
      -> ^(
      LABELED_STATEMENT IDENT statement)
1300 | expression SEMI!
1301 | SEMI
1302 ;

```



```

1303
1304 catches
1305     :   catchClause+
1306     →  ^(CATCH_CLAUSE_LIST catchClause+)
1307     ;
1308
1309 catchClause
1310     :   CATCH^ LPAREN! formalParameterStandardDecl RPAREN! block
1311     ;
1312
1313 finallyClause
1314     :   FINALLY block
1315     →  block
1316     ;
1317
1318 switchBlockLabels
1319     :   switchCaseLabels switchDefaultLabel? switchCaseLabels
1320     →  ^(SWITCH_BLOCK_LABEL_LIST switchCaseLabels
1321         switchDefaultLabel? switchCaseLabels)
1321     ;
1322
1323 switchCaseLabels
1324     :   switchCaseLabel*
1325     ;
1326
1327 switchCaseLabel
1328     :   CASE^ z=expression COLON! q=blockStatement* //{System.out.
1329         println(isBreak);}
1329     {
1330     //System.out.println(switchCase+"in case labels "+isBreak);
1331         if(switchCase)
1332         {
1333
1334             cases.push($z.text);
1335             statements.push($q.text);
1336
1337             //System.out.println("in case labels "+isBreak);
1338
1339             tokens.replace($z.start.getTokenIndex(),$q.stop.
1340                 getTokenIndex(),"");
1341             tokens.replace($CASE,"");
1342
1343             if (isBreak == true) {
1344                 String code = "";

```

```

1344         int size = cases.size();
1345
1346         for (int i = 0; i < size; i++) {
1347             code = statements.pop() + "\n" + code;
1348             //System.out.println("if(" + cases.pop() + ")\n" +
1349                 code );
1350             tokens.insertAfter($CASE,"if(" + cases.pop() + ")\n{
1351                 " + code+"}\n");
1352         }
1353         isBreak = false;
1354     }
1355 }
1356
1357 switchDefaultLabel
1358     :   DEFAULT^ COLON! blockStatement*
1359     ;
1360
1361 forInit
1362     :
1363         localVariableDeclaration    ->  ^(FOR_INIT
1364             localVariableDeclaration)
1365         |   expressionList          ->  ^(FOR_INIT expressionList)
1366         |
1367         ->  ^(FOR_INIT)
1368     ;
1369
1370 forCondition
1371     :
1372
1373     q=expression?    /*{System.out.println("forcon "+$q.text);}*/
1374     ->  ^(FOR_CONDITION expression?)
1375     ;
1376
1377 forUpdater
1378     :   q=expressionList? //{System.out.println("expressionList "+$q.
1379         text);}
1380     ->  ^(FOR_UPDATE expressionList?)
1381     ;
1382 // EXPRESSIONS
1383

```

```

1384 parenthesizedExpression
1385     : LPAREN expression RPAREN
1386     -> ^(PARENTESIZED_EXPR[$LPAREN, "PARENTESIZED_EXPR"] expression
1387         )
1388
1389     ;
1390
1391 expressionList
1392     : q=expression
1393     {
1394         if(isSequence)
1395         {
1396             tokens.insertBefore($q.start, "TimeMap timeMap,");
1397             //temp.pop();
1398             isSequence = false;
1399         }
1400     }
1401     (COMMA! expression)*
1402
1403     ;
1404
1405 expression
1406     : q=assignmentExpression // {System.out.println("expression "+$q.
1407         text);}
1408     -> ^(EXPR assignmentExpression )
1409
1410     ;
1411
1412 assignmentExpression returns [Boolean isSeq]
1413     : {$isSeq=false; doDont="";}
1414     q = conditionalExpression {$isSeq=$q.isSeq;} {
1415         // replace i=0 -> 0 to new sequenced.Integer(
1416         timeMap,0)
1417         if(!ident.equals(""))
1418         {
1419             String t = $q.text;
1420             tokens.replace($q.start.getTokenIndex(),$q.
1421                 stop.getTokenIndex(),"new "+ident+"(
1422                 timeMap,"+$q.text+"");
1423
1424             ident = "";
1425         }
1426     }
1427
1428     ( ( ASSIGN^
1429     {
1430         opr = "ASSIGN"; // .assign()
1431         if($q.isSeq)

```

```

1423         {
1424         tokens.replace($ASSIGN.getTokenIndex(), ". assign (
           "+timeMapStack.peak()+", "+assign+", ");
1425         $isSeq = true;
1426         doDont="dontDo";
1427         }
1428     }
1429     | PLUS_ASSIGN^
1430     {
1431     opr = "PLUS_ASSIGN"; // .plusAssign()
1432     if($q.isSeq)
1433     {
1434     tokens.replace($PLUS_ASSIGN.getTokenIndex(), ".
           plusAssign("+timeMapStack.peak()+", "+
           plus_assign+", ");
1435     $isSeq = true;
1436     doDont="dontDo";
1437     }
1438     }
1439     | MINUS_ASSIGN^
1440     {
1441     opr = "MINUS_ASSIGN"; // .minusAssign()
1442     if($q.isSeq)
1443     {
1444     tokens.replace($MINUS_ASSIGN.getTokenIndex(), ".
           minusAssign("+timeMapStack.peak()+", "+
           minus_assign+", ");
1445     $isSeq = true;
1446     doDont="dontDo";
1447     }
1448     }
1449     | STAR_ASSIGN^
1450     {
1451     opr = "STAR_ASSIGN"; // .starAssign()
1452     if($q.isSeq)
1453     {
1454     tokens.replace($STAR_ASSIGN.getTokenIndex(), ".
           starAssign("+timeMapStack.peak()+", "+
           star_assign+", ");
1455     $isSeq = true;
1456     doDont="dontDo";
1457     }
1458     }
1459     | DIV_ASSIGN^

```

```

1460     {
1461     opr = "DIV_ASSIGN"; // .divideAssign()
1462     if ($q.isSeq)
1463     {
1464         tokens.replace($DIV_ASSIGN.getTokenIndex(), ".
            divideAssign("+timeMapStack.peek()+", "+
            div_assign+",");
1465         $isSeq = true;
1466         doDont="dontDo";
1467     }
1468 }
1469 | AND_ASSIGN^
1470 {
1471 opr = "AND_ASSIGN"; // .andAssign()
1472 if ($q.isSeq)
1473 {
1474     tokens.replace($AND_ASSIGN.getTokenIndex(), ".
            andAssign("+timeMapStack.peek()+", "+
            and_assign+",");
1475     $isSeq = true;
1476     doDont="dontDo";
1477 }
1478 }
1479 | OR_ASSIGN^
1480 {
1481 opr = "OR_ASSIGN"; // .orAssign()
1482 if ($q.isSeq)
1483 {
1484     tokens.replace($OR_ASSIGN.getTokenIndex(), ".
            orAssign("+timeMapStack.peek()+", "+or_assign
            +",");
1485     $isSeq = true;
1486     doDont="dontDo";
1487 }
1488 }
1489 | XOR_ASSIGN^
1490 {
1491 opr = "XOR_ASSIGN"; // .xorAssign()
1492 if ($q.isSeq)
1493 {
1494     tokens.replace($XOR_ASSIGN.getTokenIndex(), ".
            xorAssign("+timeMapStack.peek()+", "+
            xor_assign+",");
1495     $isSeq = true;

```

```

1496             doDont="dontDo";
1497         }
1498     }
1499     | MOD_ASSIGN^
1500     {
1501         opr = "MOD_ASSIGN"; // .modAssign()
1502         if ($q.isSeq)
1503             {
1504                 tokens.replace($MOD_ASSIGN.getTokenIndex(),".
                    modAssign("+timeMapStack.peek()+" "+
                    mod_assign+"");
1505                 $isSeq = true;
1506                 doDont="dontDo";
1507             }
1508     }
1509     | SHIFT_LEFT_ASSIGN^
1510     {
1511         opr = "SHIFT_LEFT_ASSIGN"; // .shiftLeftAssign()
1512         if ($q.isSeq)
1513             {
1514                 tokens.replace($SHIFT_LEFT_ASSIGN.getTokenIndex
                    ()," .shiftLeftAssign("+timeMapStack.peek()+
                    "+shift_left_assign+"");
1515                 $isSeq = true;
1516                 doDont="dontDo";
1517             }
1518     }
1519     | SHIFT_RIGHT_ASSIGN^
1520     {
1521         opr = "SHIFT_RIGHT_ASSIGN"; // .shiftRightAssign()
1522         if ($q.isSeq)
1523             {
1524                 tokens.replace($SHIFT_RIGHT_ASSIGN.getTokenIndex
                    ()," .shiftRightAssign("+timeMapStack.peek()+
                    "+shift_right_assign+"");
1525                 $isSeq = true;
1526                 doDont="dontDo";
1527             }
1528     }
1529     | BIT_SHIFT_RIGHT_ASSIGN^
1530     {
1531         opr = "BIT_SHIFT_RIGHT_ASSIGN"; // .bitShiftRightAssign()
1532         if ($q.isSeq)
1533             {

```

```

1534         tokens . replace ($BIT_SHIFT_RIGHT_ASSIGN .
                getTokenIndex () , ". bitShiftRightAssign (" +
                timeMapStack . peek () + " , " +
                bit_shift_right_assign + " , " );
1535         $isSeq = true ;
1536         doDont = "dontDo" ;
1537     }
1538 }
1539 )
1540 z = assignmentExpression
1541 {
1542     if ($q . isSeq != null && $z . isSeq != null)
1543     {
1544         if ($z . isSeq && (!doDont . equals ("dontDo")))
1545         {
1546             if (opr . equals ("ASSIGN"))
1547             {
1548                 String t = $q . text ;
1549                 tokens . replace ($q . start . getTokenIndex () , $q .
                        stop . getTokenIndex () , $z . text ) ;
1550                 tokens . replace ($ASSIGN . getTokenIndex () , ".
                        assign (" + timeMapStack . peek () + " , " + assign +
                        " , " );
1551                 // tokens . insertAfter ($ASSIGN , " ") ;
1552                 tokens . replace ($z . start . getTokenIndex () , $z .
                        stop . getTokenIndex () , t ) ;
1553             }
1554             if (opr . equals ("PLUS_ASSIGN"))
1555             {
1556                 String t = $q . text ;
1557                 tokens . replace ($q . start . getTokenIndex () , $q .
                        stop . getTokenIndex () , $z . text ) ;
1558                 tokens . replace ($PLUS_ASSIGN . getTokenIndex () ,
                        ". plusAssign (" + timeMapStack . peek () + " , " +
                        plus_assign + " , " );
1559                 // tokens . insertAfter ($PLUS_ASSIGN , " ") ;
1560                 tokens . replace ($z . start . getTokenIndex () , $z .
                        stop . getTokenIndex () , t ) ;
1561             }
1562             if (opr . equals ("MINUS_ASSIGN"))
1563             {
1564                 String t = $q . text ;
1565                 tokens . replace ($q . start . getTokenIndex () , $q .
                        stop . getTokenIndex () , $z . text ) ;

```

```

1566         tokens.replace($MINUS_ASSIGN.getTokenIndex()
1567             ,".minusAssign("+timeMapStack.peek()+","
1568                 +minus_assign+",");
1569     tokens.replace($z.start.getTokenIndex(),$z.
1570         stop.getTokenIndex(),t);
1571     }
1572     if(opr.equals("STAR_ASSIGN"))
1573     {
1574         String t = $q.text;
1575         tokens.replace($q.start.getTokenIndex(),$q.
1576             stop.getTokenIndex(),$z.text);
1577         tokens.replace($STAR_ASSIGN.getTokenIndex(),
1578             ".starAssign("+timeMapStack.peek()+","
1579                 +star_assign+",");
1580         tokens.replace($z.start.getTokenIndex(),$z.
1581             stop.getTokenIndex(),t);
1582     }
1583     if(opr.equals("DIV_ASSIGN"))
1584     {
1585         String t = $q.text;
1586         tokens.replace($q.start.getTokenIndex(),$q.
1587             stop.getTokenIndex(),$z.text);
1588         tokens.replace($DIV_ASSIGN.getTokenIndex(),
1589             ".divAssign("+timeMapStack.peek()+","
1590                 +div_assign+",");
1591         tokens.replace($z.start.getTokenIndex(),$z.
1592             stop.getTokenIndex(),t);
1593     }
1594     if(opr.equals("AND_ASSIGN"))
1595     {
1596         String t = $q.text;
1597         tokens.replace($q.start.getTokenIndex(),$q.
1598             stop.getTokenIndex(),$z.text);
1599         tokens.replace($AND_ASSIGN.getTokenIndex(),
1600             ".andAssign("+timeMapStack.peek()+","
1601                 +and_assign+",");
1602         tokens.replace($z.start.getTokenIndex(),$z.
1603             stop.getTokenIndex(),t);
1604     }
1605     if(opr.equals("OR_ASSIGN"))
1606     {
1607         String t = $q.text;
1608         tokens.replace($q.start.getTokenIndex(),$q.
1609             stop.getTokenIndex(),$z.text);

```



```

1594         tokens . replace ($OR_ASSIGN . getTokenIndex () , " .
            orAssign (" + timeMapStack . peek () + " , " +
            or_assign + " , " ) ;
1595         tokens . replace ($z . start . getTokenIndex () , $z .
            stop . getTokenIndex () , t ) ;
1596     }
1597     if ( opr . equals ( "XOR_ASSIGN" ) )
1598     {
1599         String t = $q . text ;
1600         tokens . replace ($q . start . getTokenIndex () , $q .
            stop . getTokenIndex () , $z . text ) ;
1601         tokens . replace ($XOR_ASSIGN . getTokenIndex () , "
            . xorAssign (" + timeMapStack . peek () + " , " +
            xor_assign + " , " ) ;
1602         tokens . replace ($z . start . getTokenIndex () , $z .
            stop . getTokenIndex () , t ) ;
1603     }
1604     if ( opr . equals ( "MOD_ASSIGN" ) )
1605     {
1606         String t = $q . text ;
1607         tokens . replace ($q . start . getTokenIndex () , $q .
            stop . getTokenIndex () , $z . text ) ;
1608         tokens . replace ($MOD_ASSIGN . getTokenIndex () , "
            . modAssign (" + timeMapStack . peek () + " , " +
            mod_assign + " , " ) ;
1609         tokens . replace ($z . start . getTokenIndex () , $z .
            stop . getTokenIndex () , t ) ;
1610     }
1611     if ( opr . equals ( "SHIFT_LEFT_ASSIGN" ) )
1612     {
1613         String t = $q . text ;
1614         tokens . replace ($q . start . getTokenIndex () , $q .
            stop . getTokenIndex () , $z . text ) ;
1615         tokens . replace ($SHIFT_LEFT_ASSIGN .
            getTokenIndex () , " . shiftLeftAssign (" +
            timeMapStack . peek () + " , " +
            shift_left_assign + " , " ) ;
1616         tokens . replace ($z . start . getTokenIndex () , $z .
            stop . getTokenIndex () , t ) ;
1617     }
1618     if ( opr . equals ( "SHIFT_RIGHT_ASSIGN" ) )
1619     {
1620         String t = $q . text ;

```

```

1621         tokens.replace($q.start.getTokenIndex(),$q.
1622             stop.getTokenIndex(),$z.text);
1623         tokens.replace($SHIFT_RIGHT_ASSIGN.
1624             getTokenIndex(),".shiftRightAssign("+
1625             timeMapStack.peek()+","+
1626             shift_right_assign+",");
1627         tokens.replace($z.start.getTokenIndex(),$z.
1628             stop.getTokenIndex(),t);
1629     }
1630     if(opr.equals("BIT_SHIFT_RIGHT_ASSIGN"))
1631     {
1632         String t = $q.text;
1633         tokens.replace($q.start.getTokenIndex(),$q.
1634             stop.getTokenIndex(),$z.text);
1635         tokens.replace($BIT_SHIFT_RIGHT_ASSIGN.
1636             getTokenIndex(),".bitShiftRightAssign("+
1637             timeMapStack.peek()+","+
1638             bit_shift_right_assign+",");
1639         tokens.replace($z.start.getTokenIndex(),$z.
1640             stop.getTokenIndex(),t);
1641     }
1642     $isSeq = true;
1643 }
1644 if($z.isSeq || $q.isSeq)
1645     tokens.insertAfter($z.stop.getTokenIndex(),"");
1646 }
1647 }
1648 )?
1649 ;
1650
1651 conditionalExpression returns [Boolean isSeq]
1652 : { $isSeq=false; doDont=""; } e=logicalOrExpression { $isSeq=$e.isSeq
1653 ;}(QUESTION^ e1=assignmentExpression COLON! e2=
1654 conditionalExpression )?
1655 {
1656     if($e.isSeq!=null && $e1.isSeq!=null && $e2.isSeq!=null)
1657     {
1658         if($e.isSeq==true)
1659         {
1660             tokens.replace($QUESTION.getTokenIndex(),"");
1661             tokens.replace($COLON.getTokenIndex(),"");
1662             String t = $e.text+".conditional("+timeMapStack.peek
1663                 (+","+ $e1.text+", "+ $e2.text+")";

```

```

1651 //          tokens.replace($e.start.getTokenIndex(), $e.stop.
           getTokenIndex(), $e.text+".conditional("+ $e1.text+" , "+ $e2.text+" )");
1652          tokens.replace($e.start.getTokenIndex(), $e2.stop.
           getTokenIndex(), t);
1653      }
1654  }
1655  }
1656  ;
1657
1658 logicalOrExpression returns [Boolean isSeq]
1659     : { $isSeq=false; doDont="";} q=logicalAndExpression { $isSeq=$q.
           isSeq;} (LOGICAL_OR^
1660     {
1661         opr = "LOGICAL_OR"; // .logicalOr()
1662         if($q.isSeq!=null && $q.isSeq)
1663         {
1664             tokens.replace($LOGICAL_OR.getTokenIndex(), ".
           logicalOr("+timeMapStack.peek()+", "+
           logical_or+",");
1665             $isSeq = true;
1666             doDont="dontDo";
1667         }
1668     }
1669     z=logicalAndExpression
1670     {
1671         if($q.isSeq!=null && $z.isSeq!=null)
1672         {
1673             if($z.isSeq && (!doDont.equals("dontDo")))
1674             {
1675                 if(opr.equals("LOGICAL_OR"))
1676                 {
1677                     String t = $q.text;
1678                     tokens.replace($q.start.getTokenIndex(), $q.
           stop.getTokenIndex(), $z.text);
1679                     tokens.replace($LOGICAL_OR.getTokenIndex(), ".
           logicalOr("+timeMapStack.peek()+", "+
           logical_or+",");
1680                     tokens.replace($z.start.getTokenIndex(), $z.
           stop.getTokenIndex(), t);
1681                     $isSeq = true;
1682                 }
1683                 $isSeq = true;
1684             }
1685             if($z.isSeq || $q.isSeq)

```

```

1686             tokens.insertAfter($z.stop.getTokenIndex(),");");
1687         }
1688     }
1689     )*
1690     ;
1691
1692 logicalAndExpression returns [Boolean isSeq]
1693     : { $isSeq=false; doDont=""; } q=inclusiveOrExpression { $isSeq=$q.
1694         isSeq; } (LOGICALAND^
1695         {
1696             opr = "LOGICALAND"; //.logicalAnd()
1697             if($q.isSeq)
1698                 {
1699                     tokens.replace($LOGICALAND.getTokenIndex(),".
1700                         logicalAnd("+timeMapStack.peek()+",""+
1701                         logical_and+",");
1702                     $isSeq = true;
1703                     doDont="dontDo";
1704                 }
1705         }
1706     z=inclusiveOrExpression
1707     {
1708         if($q.isSeq!=null && $z.isSeq!=null)
1709             {
1710                 if($z.isSeq && (!doDont.equals("dontDo")))
1711                     {
1712                         if(opr.equals("LOGICALAND"))
1713                             {
1714                                 String t = $q.text;
1715                                 tokens.replace($q.start.getTokenIndex(),$q.
1716                                     stop.getTokenIndex(),$z.text);
1717                                 tokens.replace($LOGICALAND.getTokenIndex(),
1718                                     ".logicalAnd("+timeMapStack.peek()+",""+
1719                                     logical_and+",");
1720                                 tokens.replace($z.start.getTokenIndex(),$z.
1721                                     stop.getTokenIndex(),t);
1722                                 $isSeq = true;
1723                             }
1724                         }
1725                     $isSeq = true;
1726                 }
1727             if($z.isSeq || $q.isSeq)
1728                 tokens.insertAfter($z.stop.getTokenIndex(),");");
1729         }
1730     }

```

```

1723     )*
1724     ;
1725
1726 inclusiveOrExpression returns [Boolean isSeq]
1727     : { $isSeq=false; doDont="";} q=exclusiveOrExpression { $isSeq=$q.
      isSeq;} (OR^
1728     {
1729         opr = "OR"; // .or()
1730         if($q.isSeq)
1731         {
1732             tokens.replace($OR.getTokenIndex(),".or("+
              timeMapStack.peek()+","+or+",");
1733             $isSeq = true;
1734             doDont="dontDo";
1735         }
1736     }
1737     z=exclusiveOrExpression
1738     {
1739         if($q.isSeq!=null && $z.isSeq!=null)
1740         {
1741             if($z.isSeq && (!doDont.equals("dontDo")))
1742             {
1743                 if(opr.equals("OR"))
1744                 {
1745                     String t = $q.text;
1746                     tokens.replace($q.start.getTokenIndex(),$q.
                      stop.getTokenIndex(),$z.text);
1747                     tokens.replace($OR.getTokenIndex(),".or("+
                      timeMapStack.peek()+","+or+",");
1748                     tokens.replace($z.start.getTokenIndex(),$z.
                      stop.getTokenIndex(),t);
1749                     $isSeq = true;
1750                 }
1751                 $isSeq = true;
1752             }
1753             if($z.isSeq || $q.isSeq)
1754                 tokens.insertAfter($z.stop.getTokenIndex(),"");
1755         }
1756     }
1757     )*
1758     ;
1759
1760 exclusiveOrExpression returns [Boolean isSeq]

```

```

1761      : { $isSeq=false; doDont="";} q=andExpression { $isSeq=$q.isSeq;} (
      XOR^
1762      {
1763          opr = "XOR"; // .xor()
1764
1765          if ($q.isSeq)
1766              {
1767                  tokens.replace($XOR.getTokenIndex(), ".xor("+
                  timeMapStack.peek()+", "+xor+",");
1768                  $isSeq = true;
1769                  doDont="dontDo";
1770              }
1771
1772      }
1773
1774      z=andExpression
1775      {
1776          //System.out.println($q.isSeq+" "+$z.isSeq);
1777          if ($q.isSeq!=null && $z.isSeq!=null)
1778              {
1779                  if ($z.isSeq && (!doDont.equals("dontDo")))
1780                      {
1781                          if (opr.equals("XOR"))
1782                              {
1783                                  String t = $q.text;
1784                                  tokens.replace($q.start.getTokenIndex(), $q.
                                  stop.getTokenIndex(), $z.text);
1785                                  tokens.replace($XOR.getTokenIndex(), ".xor("+
                                  timeMapStack.peek()+", "+xor+",");
1786                                  tokens.replace($z.start.getTokenIndex(), $z.
                                  stop.getTokenIndex(), t);
1787                                  $isSeq = true;
1788                              }
1789                                  $isSeq = true;
1790                              }
1791                  if ($z.isSeq || $q.isSeq)
1792                      tokens.insertAfter($z.stop.getTokenIndex(), "");
1793              }
1794      }
1795      )*
1796      ;
1797
1798      andExpression returns [Boolean isSeq]

```

```

1799         : { $isSeq=false; doDont="";}    q=equalityExpression { $isSeq=$q.isSeq
        ;} (AND^
1800             {
1801                 opr = "AND"; // .and()
1802
1803                 if ($q.isSeq)
1804                     {
1805                         tokens.replace($AND.getTokenIndex(), ".and("+
                            timeMapStack.peek()+", "+and+",");
1806                         $isSeq = true;
1807                         doDont="dontDo";
1808                     }
1809
1810             }
1811     z=equalityExpression
1812         {
1813             $isSeq = $z.isSeq;
1814             if ($q.isSeq != null && $z.isSeq != null)
1815                 {
1816                     if ($z.isSeq && (!doDont.equals("dontDo")))
1817                         {
1818                             if (opr.equals("AND"))
1819                                 {
1820                                     String t = $q.text;
1821                                     tokens.replace($q.start.getTokenIndex(), $q.
                            stop.getTokenIndex(), $z.text);
1822                                     tokens.replace($AND.getTokenIndex(), ".and("+
                            timeMapStack.peek()+", "+and+",");
1823                                     tokens.replace($z.start.getTokenIndex(), $z.
                            stop.getTokenIndex(), t);
1824                                     $isSeq = true;
1825                                 }
1826                             $isSeq = true;
1827                         }
1828                     if ($z.isSeq || $q.isSeq)
1829                         tokens.insertAfter($z.stop.getTokenIndex(), "");
1830                 }
1831         }
1832     )*
1833 ;
1834
1835 equalityExpression returns [Boolean isSeq]
1836     : { $isSeq=false;} q=instanceOfExpression { $isSeq=$q.isSeq;}
1837     ( ( {doDont="";} EQUAL^

```

```

1838     {
1839         opr = "EQUAL"; // .equal()
1840
1841     if ($q.isSeq)
1842     {
1843         tokens.replace($EQUAL.getTokenIndex(), ".equal("+
            timeMapStack.peek()+","+equal+",");
1844         $isSeq = true;
1845         doDont="dontDo";
1846     }
1847
1848 }
1849 | NOT_EQUAL^
1850 {
1851     opr = "NOT_EQUAL"; // .notEqual()
1852
1853     if ($q.isSeq)
1854     {
1855         tokens.replace($NOT_EQUAL.getTokenIndex(), ".
            notEqual("+timeMapStack.peek()+","+not_equal
            +",");
1856         $isSeq = true;
1857         doDont="dontDo";
1858     }
1859
1860 }
1861 )
1862 z=instanceOfExpression
1863 {
1864     if ($q.isSeq != null && $z.isSeq != null)
1865     {
1866         if ($z.isSeq && (!doDont.equals("dontDo")))
1867         {
1868             if (opr.equals("EQUAL"))
1869             {
1870                 String t = $q.text;
1871                 tokens.replace($q.start.getTokenIndex(), $q.
                    stop.getTokenIndex(), $z.text);
1872                 tokens.replace($EQUAL.getTokenIndex(), ".
                    equal("+timeMapStack.peek()+","+equal+",
                    ");
1873                 tokens.replace($z.start.getTokenIndex(), $z.
                    stop.getTokenIndex(), t);
1874                 $isSeq = true;

```



```

1875         }
1876         if (opr.equals("NOTEQUAL"))
1877         {
1878             String t = $q.text;
1879             tokens.replace($q.start.getTokenIndex(), $q.
1880                 stop.getTokenIndex(), $z.text);
1881             tokens.replace($NOT_EQUAL.getTokenIndex(), ".
1882                 notEqual("+timeMapStack.peek()+" "+
1883                 not_equal+"");
1884             tokens.replace($z.start.getTokenIndex(), $z.
1885                 stop.getTokenIndex(), t);
1886         }
1887         $isSeq = true;
1888     }
1889     if ($z.isSeq || $q.isSeq)
1890         tokens.insertAfter($z.stop.getTokenIndex(), "");
1891 }
1892 )*
1893 ;
1894
1895 instanceofExpression returns [Boolean isSeq]
1896 : { $isSeq=false; } q=relationalExpression { $isSeq=$q.isSeq; } (
1897     INSTANCEOF^ z=type)?
1898 {
1899     if ($z.text!=null && $q.isSeq)
1900         tokens.replace($z.start.getTokenIndex(), $z.stop.
1901             getTokenIndex(), "sequenced."+$z.text);
1902 }
1903 ;
1904
1905 //RELATIONAL EXPRESSION FOR for loop
1906 // Relational expression for If else statement
1907
1908 relationalExpression returns [Boolean isSeq]
1909 : { $isSeq=false; doDont=""; } q=shiftExpression { $isSeq=$q.isSeq; }
1910 ( ( LESS_OR_EQUAL^
1911     {
1912         opr = "LESS_OR_EQUAL"; // .lessOrEqual()
1913
1914         if ($q.isSeq)
1915             {

```

```

1912         tokens.replace($LESS_OR_EQUAL.getTokenIndex(), ".
           lessOrEqual("+timeMapStack.peak()+", "+
           less_or_equal+",");
1913         $isSeq = true;
1914         doDont="dontDo";
1915     }
1916
1917 }
1918 | GREATER_OR_EQUAL^
1919 {
1920     opr = "GREATER_OR_EQUAL"; // .greaterOrEqual()
1921
1922     if($q.isSeq)
1923     {
1924         tokens.replace($GREATER_OR_EQUAL.getTokenIndex()
           , ". greaterOrEqual("+timeMapStack.peak()+", "+
           greater_or_equal+",");
1925         $isSeq = true;
1926         doDont="dontDo";
1927     }
1928
1929 }
1930 | LESS_THAN^
1931 {
1932     opr = "LESS_THAN"; // .lessThan()
1933
1934     if($q.isSeq)
1935     {
1936         tokens.replace($LESS_THAN.getTokenIndex(), ".
           lessThan("+timeMapStack.peak()+", "+less_than
           +",");
1937         $isSeq = true;
1938         doDont="dontDo";
1939     }
1940
1941 }
1942 | GREATER_THAN^
1943 {
1944     opr = "GREATER_THAN"; // .greaterThan()
1945     if($q.isSeq!=null)
1946     {
1947         if($q.isSeq)
1948         {
1949

```

```

1950         tokens . replace ($GREATER_THAN . getTokenIndex () , ".
              greaterThan (" + timeMapStack . peek () + " , " +
              greater_than + " , ") ;
1951         $isSeq = true ;
1952         doDont = "dontDo" ;
1953     }
1954 }
1955 }
1956 )
1957 z = shiftExpression
1958 {
1959     if ($q . isSeq != null && $z . isSeq != null)
1960     {
1961         if ($z . isSeq && (! doDont . equals ("dontDo")))
1962         {
1963             if (opr . equals ("LESS_OR_EQUAL"))
1964             {
1965                 String t = $q . text ;
1966                 tokens . replace ($q . start . getTokenIndex () , $q .
              stop . getTokenIndex () , $z . text ) ;
1967                 tokens . replace ($LESS_OR_EQUAL . getTokenIndex
              () , ". lessOrEqual (" + timeMapStack . peek () + "
              , " + less_or_equal + " , ") ;
1968                 tokens . replace ($z . start . getTokenIndex () , $z .
              stop . getTokenIndex () , t ) ;
1969             }
1970             if (opr . equals ("GREATER_OR_EQUAL"))
1971             {
1972                 String t = $q . text ;
1973                 tokens . replace ($q . start . getTokenIndex () , $q .
              stop . getTokenIndex () , $z . text ) ;
1974                 tokens . replace ($GREATER_OR_EQUAL .
              getTokenIndex () , ". greaterOrEqual (" +
              timeMapStack . peek () + " , " + greater_or_equal
              + " , ") ;
1975                 tokens . replace ($z . start . getTokenIndex () , $z .
              stop . getTokenIndex () , t ) ;
1976             }
1977             if (opr . equals ("LESS_THAN"))
1978             {
1979                 String t = $q . text ;
1980                 tokens . replace ($q . start . getTokenIndex () , $q .
              stop . getTokenIndex () , $z . text ) ;

```

```

1981         tokens . replace ($LESS_THAN . getTokenIndex () , " .
                lessThan (" + timeMapStack . peek () + " , " +
                less_than + " , " ) ;
1982         tokens . replace ($z . start . getTokenIndex () , $z .
                stop . getTokenIndex () , t ) ;
1983     }
1984     if ( opr . equals ( " GREATER_THAN " ) )
1985     {
1986         String t = $q . text ;
1987         tokens . replace ($q . start . getTokenIndex () , $q .
                stop . getTokenIndex () , $z . text ) ;
1988         tokens . replace ($GREATER_THAN . getTokenIndex ()
                , " . greaterThan (" + timeMapStack . peek () + " , "
                + greater_than + " , " ) ;
1989         tokens . replace ($z . start . getTokenIndex () , $z .
                stop . getTokenIndex () , t ) ;
1990     }
1991     $isSeq = true ;
1992 }
1993 if ($z . isSeq || $q . isSeq)
1994     tokens . insertAfter ($z . stop . getTokenIndex () , " ) " ;
1995 }
1996 }
1997 ) *
1998 ;
1999
2000 shiftExpression returns [ Boolean isSeq ]
2001 : { $isSeq = false ; doDont = " " ; } q = additiveExpression { $isSeq = $q . isSeq
                ; }
2002 ( ( BIT_SHIFT_RIGHT ^ // . bitShiftRight ()
2003 {
2004     opr = " BIT_SHIFT_RIGHT " ;
2005     if ($q . isSeq != null )
2006     {
2007         if ($q . isSeq )
2008         {
2009             tokens . replace ($BIT_SHIFT_RIGHT . getTokenIndex () ,
                " . bitShiftRight (" + timeMapStack . peek () + " , " +
                bit_shift_right + " , " ) ;
2010             $isSeq = true ;
2011             doDont = " dontDo " ;
2012         }
2013     }
2014 }

```

```

2015 |   SHIFT_RIGHT^ // . shiftRight ()
2016 |   {
2017 |     opr = "SHIFT_RIGHT";
2018 |   if ($q.isSeq != null)
2019 |   {
2020 |     if ($q.isSeq)
2021 |     {
2022 |       tokens.replace ($SHIFT_RIGHT.getTokenIndex (), ".
                shiftRight (" +timeMapStack. peek ()+" , "+"
                shift_right+" ,");
2023 |       $isSeq = true;
2024 |       doDont="dontDo";
2025 |     }
2026 |   }
2027 | }
2028 |   SHIFT_LEFT^ // . shiftLeft ()
2029 |   {
2030 |     opr = "SHIFT_LEFT";
2031 |   if ($q.isSeq != null)
2032 |   {
2033 |     if ($q.isSeq)
2034 |     {
2035 |       tokens.replace ($SHIFT_LEFT.getTokenIndex (), ".
                shiftLeft (" +timeMapStack. peek ()+" , "+"
                shift_left+" ,");
2036 |       $isSeq = true;
2037 |       doDont="dontDo";
2038 |     }
2039 |   }
2040 | }
2041 | )
2042 | z=additiveExpression
2043 | {
2044 |   if ($q.isSeq != null && $z.isSeq != null)
2045 |   {
2046 |     if ($z.isSeq && (!doDont.equals ("dontDo")))
2047 |     {
2048 |       if (opr.equals ("BIT_SHIFT_RIGHT"))
2049 |       {
2050 |         String t = $q.text;
2051 |         tokens.replace ($q.start.getTokenIndex (), $q.
                stop.getTokenIndex (), $z.text);
2052 |         tokens.replace ($BIT_SHIFT_RIGHT.
                getTokenIndex (), ". bitShiftRight (" +

```

```

timeMapStack.peek()+"", "+bit_shift_right+
",");
2053     tokens.replace($z.start.getTokenIndex(), $z.
        stop.getTokenIndex(), t);
2054     }
2055     if (opr.equals("SHIFT_RIGHT"))
2056     {
2057         String t = $q.text;
2058         tokens.replace($q.start.getTokenIndex(), $q.
            stop.getTokenIndex(), $z.text);
2059         tokens.replace($SHIFT_RIGHT.getTokenIndex(),
            ".shiftRight("+timeMapStack.peek()+"", "+
            shift_right+",");
2060         tokens.replace($z.start.getTokenIndex(), $z.
            stop.getTokenIndex(), t);
2061     }
2062     if (opr.equals("SHIFT_LEFT"))
2063     {
2064         String t = $q.text;
2065         tokens.replace($q.start.getTokenIndex(), $q.
            stop.getTokenIndex(), $z.text);
2066         tokens.replace($SHIFT_LEFT.getTokenIndex(), "
            .shiftLeft("+timeMapStack.peek()+"", "+
            shift_left+",");
2067         tokens.replace($z.start.getTokenIndex(), $z.
            stop.getTokenIndex(), t);
2068     }
2069     $isSeq = true;
2070     }
2071     if ($z.isSeq || $q.isSeq)
2072         tokens.insertAfter($z.stop.getTokenIndex(), "");
2073     }
2074     }
2075     )*
2076     ;
2077
2078 additiveExpression returns [Boolean isSeq]
2079     : { $isSeq=false; doDont="" ; doIt=false; } q=multiplicativeExpression
        { $isSeq=$q.isSeq; }
2080     ( ( PLUS^
2081     {
2082     opr = "PLUS";
2083     if ($q.isSeq != null)
2084     {

```

```

2085         if ($q.isSeq)
2086             {
2087                 tokens.replace($PLUS.getTokenIndex(), ".add("+
                    timeMapStack.peek()+", "+plus+",");
2088                 $isSeq = true;
2089                 doDont="dontDo";
2090             }
2091         }
2092     }
2093 |   MINUS^
2094     {
2095     opr = "MINUS";
2096     if ($q.isSeq)
2097         {
2098             tokens.replace($MINUS.getTokenIndex(), ".subtract("+
                    timeMapStack.peek()+", "+minus+",");
2099             $isSeq = true;
2100             doDont="dontDo";
2101         }
2102     }
2103 )
2104 z=multiplicativeExpression
2105 {
2106     if ($q.isSeq != null && $z.isSeq != null)
2107     {
2108         if ($z.isSeq && (!doDont.equals("dontDo")))
2109         {
2110             doIt = true;
2111             if (opr.equals("PLUS"))
2112             {
2113                 String t = $q.text;
2114                 tokens.replace($q.start.getTokenIndex(), $q.
                    stop.getTokenIndex(), $z.text);
2115                 tokens.replace($PLUS.getTokenIndex(), ".add("
                    +timeMapStack.peek()+", "+plus+",");
2116                 tokens.replace($z.start.getTokenIndex(), $z.
                    stop.getTokenIndex(), t);
2117                 $isSeq = true;
2118             }
2119             if (opr.equals("MINUS"))
2120             {
2121                 String t = $q.text;
2122                 tokens.replace($q.start.getTokenIndex(), $q.
                    stop.getTokenIndex(), $z.text);

```

```

2123         tokens.replace($MINUS.getTokenIndex(),".
                subtract("+timeMapStack.peak()+",""+minus
                    +",");
2124         tokens.replace($z.start.getTokenIndex(),$z.
                stop.getTokenIndex(),t);
2125     }
2126         $isSeq = true;
2127     }
2128
2129         tokens.insertAfter($z.stop.getTokenIndex(),")");
2130     }
2131
2132     }
2133     )*
2134 ;
2135
2136 multiplicativeExpression returns [Boolean isSeq]
2137     : { $isSeq=false;doDont="";} q=unaryExpression { $isSeq=$q.isSeq;}
2138     (
2139     ( STAR^
2140     {
2141         opr = "STAR";
2142         if ($q.isSeq)
2143         {
2144             tokens.replace($STAR.getTokenIndex(),".multiply("+
                timeMapStack.peak()+",""+star+",");
2145             $isSeq = true;
2146             doDont="dontDo";
2147         }
2148     }
2149     | DIV^
2150     {
2151         opr = "DIV";
2152         if ($q.isSeq)
2153         {
2154             tokens.replace($DIV.getTokenIndex(),".divide("+
                timeMapStack.peak()+",""+div+",");
2155             $isSeq = true;
2156             doDont="dontDo";
2157         }
2158     }
2159     | MOD^
2160     {
2161         opr = "MOD";

```



```

2162         if ($q.isSeq)
2163             {
2164                 tokens.replace($MOD.getTokenIndex(), ". modulus (" +
                    timeMapStack.peek() + ", " + mod + ", ");
2165                 $isSeq = true;
2166                 doDont="dontDo";
2167             }
2168         }
2169     )
2170     z=unaryExpression
2171     {
2172         if ($q.isSeq != null && $z.isSeq != null)
2173         {
2174             if ($z.isSeq && (!doDont.equals("dontDo")))
2175             {
2176                 if (opr.equals("STAR"))
2177                 {
2178                     String t = $q.text;
2179                     tokens.replace($q.start.getTokenIndex(), $q.
                        stop.getTokenIndex(), $z.text);
2180                     tokens.replace($STAR.getTokenIndex(), ".
                        multiply (" + timeMapStack.peek() + ", " + star +
                        ", ");
2181                     tokens.replace($z.start.getTokenIndex(), $z.
                        stop.getTokenIndex(), t);
2182                 }
2183                 if (opr.equals("DIV"))
2184                 {
2185                     String t = $q.text;
2186                     tokens.replace($q.start.getTokenIndex(), $q.
                        stop.getTokenIndex(), $z.text);
2187                     tokens.replace($DIV.getTokenIndex(), ". divide
                        (" + timeMapStack.peek() + ", " + div + ", ");
2188                     tokens.replace($z.start.getTokenIndex(), $z.
                        stop.getTokenIndex(), t);
2189                 }
2190                 if (opr.equals("MOD"))
2191                 {
2192                     String t = $q.text;
2193                     tokens.replace($q.start.getTokenIndex(), $q.
                        stop.getTokenIndex(), $z.text);
2194                     tokens.replace($MOD.getTokenIndex(), ".
                        modulus (" + timeMapStack.peek() + ", " + mod + ",
                        ");

```

```

2195             tokens.replace($z.start.getTokenIndex(), $z.
                stop.getTokenIndex(), t);
2196         }
2197         $isSeq = true;
2198     }
2199     if ($z.isSeq || $q.isSeq)
2200         tokens.insertAfter($z.stop.getTokenIndex(), "");
2201     }
2202 }
2203 )*
2204 ;
2205
2206 unaryExpression returns [Boolean isSeq]
2207 : { $isSeq=false; } PLUS p=unaryExpression
2208     {
2209         if ($p.isSeq)
2210         {
2211             tokens.insertAfter($PLUS.getTokenIndex()+1, ".add(" +
                timeMapStack.peek() + ", "+plus+"");
2212             tokens.replace($PLUS.getTokenIndex(), "");
2213             $isSeq = true;
2214         }
2215     }
2216     -> ^(UNARY_PLUS[$PLUS, "UNARY_PLUS"] unaryExpression)
2217 | MINUS m=unaryExpression
2218     {
2219         if ($m.isSeq)
2220         {
2221             tokens.insertAfter($MINUS.getTokenIndex()+1, ".subtract(" +
                +timeMapStack.peek() + ", "+minus+"");
2222             tokens.replace($MINUS.getTokenIndex(), "");
2223             $isSeq = true;
2224         }
2225     }
2226     -> ^(UNARY_MINUS[$MINUS, "UNARY_MINUS"] unaryExpression)
2227
2228     // pre fix operations
2229 | INC i=postfixedExpression
2230     {
2231         if ($i.isSeq)
2232         {
2233             tokens.insertAfter($INC.getTokenIndex()+1, ".increment(" +
                timeMapStack.peek() + ", "+inc+"");
2234             tokens.replace($INC.getTokenIndex(), "");

```

```

2235         $isSeq = true;
2236     }
2237 }
2238 -> ^(PRE_INC[$INC, "PRE_INC"] postfixExpression)
2239
2240     // pre fix operations
2241 | DEC d=postfixExpression
2242 {
2243     if($d.isSeq)
2244     {
2245         tokens.insertAfter($DEC.getTokenIndex()+1, ".decrement("+
                timeMapStack.peek()+", "+dec+")");
2246         tokens.replace($DEC.getTokenIndex(), "");
2247         $isSeq = true;
2248     }
2249 }
2250 -> ^(PRE_DEC[$DEC, "PRE_DEC"] postfixExpression)
2251 | q=unaryExpressionNotPlusMinus
2252 {
2253     $isSeq = $q.isSeq;
2254 }
2255 ;
2256
2257 unaryExpressionNotPlusMinus returns [Boolean isSeq]
2258 : { $isSeq=false; } NOT n=unaryExpression
2259 {
2260     if($n.isSeq)
2261     {
2262         tokens.insertAfter($NOT.getTokenIndex()+1, ".not("+
                timeMapStack.peek()+", "+not+")");
2263         tokens.replace($NOT.getTokenIndex(), "");
2264         $isSeq = true;
2265     }
2266 }
2267 -> ^(NOT unaryExpression)
2268 | LOGICAL_NOT ln=unaryExpression
2269 {
2270     if($ln.isSeq)
2271     {
2272         tokens.insertAfter($LOGICAL_NOT.getTokenIndex()+1, ".
                logicalNot("+timeMapStack.peek()+", "+logical_not+"
                "");
2273         tokens.replace($LOGICAL_NOT.getTokenIndex(), "");
2274         $isSeq = true;

```

```

2275     }
2276   }
2277           -> ^(LOGICAL_NOT unaryExpression)
2278   | LPAREN type RPAREN unaryExpression          -> ^(
2279   |   CAST_EXPR[$LPAREN, "CAST_EXPR"] type unaryExpression)
2280   |   q=postfixedExpression
2281   {
2282     $isSeq = $q.isSeq;
2283   }
2284   ;
2285   postfixExpression returns [Boolean isSeq]
2286     // At first resolve the primary expression ...
2287     : ( {$isSeq=false;} q=primaryExpression
2288         {
2289           $isSeq = $q.isSeq;
2290         }
2291         -> primaryExpression
2292     )
2293     // ... and than the optional things that may follow a primary
2294     // expression 0 or more times.
2295     ( outerDot=DOT
2296       ( ( genericTypeArgumentListSimplified? // Note: generic
2297         // type arguments are only valid for method calls, i.e. if
2298         // there
2299         // is an
2300         // argument
2301         // list.
2302         IDENT -> ^(DOT
2303           $postfixedExpression IDENT)
2304         )
2305         ( arguments
2306           /*{
2307             if(isSequence)
2308             {
2309               System.out.println(" abc ");
2310               methodCall = true;
2311               isSequence = false;
2312             }
2313           }*/
2314         )
2315       )
2316     )
2317     -> ^(METHOD_CALL
2318       $postfixedExpression

```

```

                                genericTypeArgumentListSimplified
                                ? arguments)
2310     )?
2311     | THIS                                -> ^(DOT
        $postfixedExpression THIS)
2312     | Super=SUPER arguments              -> ^(
        SUPER.CONSTRUCTOR_CALL[$Super, "SUPER.CONSTRUCTOR_CALL"]
        $postfixedExpression arguments)
2313     | ( SUPER innerDot=DOT IDENT        -> ^($innerDot ^(
        $outerDot $postfixedExpression SUPER) IDENT)
2314     )
2315     ( arguments
2316     /*{
2317     if(isSequence)
2318     {
2319         System.out.println(" abc ");
2320         methodCall = true;
2321         isSequence = false;
2322     }
2323     */
2324     -> ^(METHOD_CALL $postfixedExpression
        arguments)
2325     )?
2326     | innerNewExpression                  -> ^(DOT
        $postfixedExpression innerNewExpression)
2327     )
2328     | LBRACK expression RBRACK           -> ^(
        ARRAY_ELEMENT_ACCESS $postfixedExpression expression)
2329     )*
2330     // At the end there may follow a post increment/decrement.
2331     // Post fix increment
2332     ( INC
2333     {
2334         if($q.isSeq)
2335         {
2336             tokens.replace($INC, ". increment("+timeMapStack.peek
                (+", "+inc+"));
2337         }
2338     }
2339     -> ^(POST_INC[$INC, "POST_INC"] $postfixedExpression)
2340     | DEC
2341     // Post fix decrement
2342     {
2343         if($q.isSeq)

```

```

2344         {
2345             tokens.replace($DEC,".decrement("+timeMapStack.peek
                (+)", "+dec+)");
2346         }
2347     }
2348     -> ^(POST_DEC[$DEC, "POST_DEC"] $postfixedExpression)
2349     )?
2350 ;
2351
2352 primaryExpression returns [Boolean isSeq]
2353 : { $isSeq = false;} parenthesizedExpression
2354 | z=literal { $isSeq = $z.isSeq;}
2355 | newExpression
2356 | q=qualifiedIdentExpression
2357 {
2358     $isSeq = $q.isSeq;
2359 }
2360 | genericTypeArgumentListSimplified
2361 ( SUPER
2362     ( arguments -> ^(
                SUPER.CONSTRUCTOR_CALL[$SUPER, "SUPER.CONSTRUCTOR_CALL"]
                genericTypeArgumentListSimplified arguments)
2363     | DOT IDENT arguments -> ^(
                METHOD_CALL ^(DOT SUPER IDENT)
                genericTypeArgumentListSimplified arguments)
2364     )
2365     | q1=IDENT
2366     {
2367         System.out.println($q1.text);
2368         if(isSequence)
2369         {
2370             System.out.println(" abc ");
2371             methodCall = true;
2372             isSequence = false;
2373         }
2374     }
2375     arguments -> ^(METHOD_CALL IDENT
                genericTypeArgumentListSimplified arguments)
2376     | THIS arguments -> ^(
                THIS.CONSTRUCTOR_CALL[$THIS, "THIS.CONSTRUCTOR_CALL"]
                genericTypeArgumentListSimplified arguments)
2377     )
2378     | ( THIS -> THIS
2379     )

```

```

2380      ( arguments                                -> ^(
      THIS.CONSTRUCTOR_CALL[$THIS, "THIS.CONSTRUCTOR_CALL"]
      arguments )
2381    )?
2382  | SUPER arguments                                -> ^(
      SUPER.CONSTRUCTOR_CALL[$SUPER, "SUPER.CONSTRUCTOR_CALL"]
      arguments )
2383  | ( SUPER DOT IDENT
2384    )
2385    ( arguments
2386    /*{
2387      if(isSequence)
2388      {
2389        System.out.println(" abc ");
2390        methodCall = true;
2391        isSequence = false;
2392      }
2393    } */
2394                                -> ^(METHOD_CALL ^(DOT SUPER
      IDENT) arguments)
2395  |                                -> ^(DOT SUPER
      IDENT)
2396  )
2397  | ( primitiveType                                ->
      primitiveType
2398  )
2399  | ( arrayDeclarator                                -> ^(
      arrayDeclarator $primaryExpression)
2400  )*
2401  DOT CLASS                                        -> ^(DOT
      $primaryExpression CLASS)
2402  | VOID DOT CLASS                                -> ^(DOT VOID
      CLASS)
2403  ;
2404
2405  qualifiedIdentExpression returns [Boolean isSeq]
2406    // The qualified identifier itself is the starting point for
      this rule.
2407  : ( {$isSeq = false;} q=qualifiedIdentifier {
2408      boolean t = st.probe($q.text);
2409      //System.out.println(" sequenced check "+$q.text
      + " "+t);
2410      if(t)
2411      {

```

```

2412             $isSeq = true;
2413         }
2414
2415     }                                     ->
                qualifiedIdentifier
2416 )
2417 // And now comes the stuff that may follow the qualified
                identifier.
2418 ( ( arrayDeclarator                       -> ^(
                arrayDeclarator $qualifiedIdentExpression)
2419 )+
2420 ( DOT CLASS                               -> ^(DOT
                $qualifiedIdentExpression CLASS)
2421 )
2422 | {{
2423     if (isSequence)
2424     {
2425         System.out.println(" abc ");
2426         methodCall = true;
2427         isSequence = false;
2428     }
2429 }} arguments
2430
2431                                     -> ^(METHOD_CALL qualifiedIdentifier
                arguments)
2432 | outerDot=DOT
2433 ( CLASS                                   -> ^(DOT
                qualifiedIdentifier CLASS)
2434 | genericTypeArgumentListSimplified
2435 ( Super=SUPER arguments                 -> ^(
                SUPER.CONSTRUCTOR_CALL[$Super, "
                SUPER.CONSTRUCTOR_CALL"] qualifiedIdentifier
                genericTypeArgumentListSimplified arguments)
2436 | SUPER innerDot=DOT IDENT arguments
2437
2438 -> ^(METHOD_CALL ^($innerDot ^($outerDot
                qualifiedIdentifier SUPER) IDENT)
                genericTypeArgumentListSimplified arguments)
2439 | IDENT arguments
2440 /*{ if (isSequence)
2441 {
2442     System.out.println(" abc ");
2443     methodCall = true;
2444     isSequence = false;

```



```

2445     }}*/
2446     -> ^(METHOD_CALL ^(DOT
        qualifiedIdentifier IDENT)
        genericTypeArgumentListSimplified
        arguments)
2447     )
2448     | THIS -> ^(DOT
        qualifiedIdentifier THIS)
2449     | Super=SUPER arguments -> ^(
        SUPER.CONSTRUCTOR_CALL[$Super, "SUPER.CONSTRUCTOR_CALL"]
        qualifiedIdentifier arguments)
2450     | innerNewExpression -> ^(DOT
        qualifiedIdentifier innerNewExpression)
2451     ) {isSequence = false;}
2452     )?
2453     ;
2454
2455 newExpression
2456     : NEW
2457     ( primitiveType newArrayConstruction // new static array
        of primitive type elements
2458     -> ^(STATIC_ARRAY_CREATOR[$NEW, "STATIC_ARRAY_CREATOR"]
        primitiveType newArrayConstruction)
2459     | genericTypeArgumentListSimplified?
        qualifiedTypeIdentSimplified
2460     ( newArrayConstruction // new static array
        of object type reference elements
2461     -> ^(STATIC_ARRAY_CREATOR[$NEW, "STATIC_ARRAY_CREATOR"]
        genericTypeArgumentListSimplified?
        qualifiedTypeIdentSimplified newArrayConstruction)
2462     | arguments classBody? // new object type
        via constructor invocation
2463     -> ^(CLASS_CONSTRUCTOR_CALL[$NEW, "STATIC_ARRAY_CREATOR
        "] genericTypeArgumentListSimplified?
        qualifiedTypeIdentSimplified arguments classBody?)
2464     )
2465     )
2466     ;
2467
2468     innerNewExpression // something like 'InnerType innerType = outer.new
        InnerType ();'
2469     : NEW genericTypeArgumentListSimplified? IDENT arguments classBody
        ?

```

```

2470         -> ^(CLASS_CONSTRUCTOR_CALL[$NEW, "STATIC_ARRAY_CREATOR"]
              genericTypeArgumentListSimplified? IDENT arguments classBody
              ?)
2471         ;
2472
2473 newArrayConstruction
2474         :   arrayDeclaratorList arrayInitializer
2475         |   LBRACK! expression RBRACK! (LBRACK! expression RBRACK!)*
              arrayDeclaratorList?
2476         ;
2477
2478 arguments
2479         :   LPAREN
2480         {
2481             if (methodCall)
2482             {
2483                 System.out.println("method call");
2484                 tokens.insertAfter($LPAREN, timeMapStack.peek()+",");
2485                 methodCall = false;
2486             }
2487         }
2488         expressionList? RPAREN
2489         -> ^(ARGUMENT_LIST[$LPAREN, "ARGUMENT_LIST"] expressionList?)
2490         ;
2491
2492 literal returns [Boolean isSeq]
2493         : { $isSeq=false; } q= HEX_LITERAL          {
2494             boolean t = st.probe($q.text);
2495             if (t)
2496                 $isSeq = true;
2497         }
2498         | q= OCTAL_LITERAL          {
2499             boolean t = st.probe($q.text);
2500             if (t)
2501                 $isSeq = true;
2502         }
2503         | q= DECIMAL_LITERAL        {
2504             boolean t = st.probe($q.text);
2505             if (t)
2506                 $isSeq = true;
2507         }
2508         | q= FLOATING_POINT_LITERAL {
2509             boolean t = st.probe($q.text);
2510             if (t)

```

```

2511             $isSeq = true;
2512         }
2513     | q= CHARACTER_LITERAL    {
2514             boolean t = st.probe($q.text);
2515             if(t)
2516                 $isSeq = true;
2517         }
2518     | q= STRING_LITERAL       {
2519             boolean t = st.probe($q.text);
2520             if(t)
2521                 $isSeq = true;
2522         }
2523     | q= TRUE                 {
2524             boolean t = st.probe($q.text);
2525             if(t)
2526                 $isSeq = true;
2527         }
2528     | q= FALSE               {
2529             boolean t = st.probe($q.text);
2530             if(t)
2531                 $isSeq = true;
2532         }
2533     | q= NULL                {
2534             boolean t = st.probe($q.text);
2535             if(t)
2536                 $isSeq = true;
2537         }
2538     ;
2539
2540 // LEXER
2541
2542 HEX_LITERAL : '0' ('x'|'X') HEX_DIGIT+ INTEGER_TYPE_SUFFIX? ;
2543
2544 DECIMAL_LITERAL : ('0' | '1'..'9' '0'..'9'*) INTEGER_TYPE_SUFFIX? ;
2545
2546 OCTAL_LITERAL : '0' ('0'..'7')+ INTEGER_TYPE_SUFFIX? ;
2547
2548 fragment
2549     HEX_DIGIT : ('0'..'9'|'a'..'f'|'A'..'F') ;
2550
2551 fragment
2552     INTEGER_TYPE_SUFFIX : ('l'|'L') ;
2553
2554 FLOATING_POINT_LITERAL

```

```

2555      :   ('0'..'9')+
2556      (
2557          DOT ('0'..'9')* EXPONENT? FLOAT_TYPE_SUFFIX?
2558          |   EXPONENT FLOAT_TYPE_SUFFIX?
2559          |   FLOAT_TYPE_SUFFIX
2560      )
2561      |   DOT ('0'..'9')+ EXPONENT? FLOAT_TYPE_SUFFIX?
2562      ;
2563
2564 fragment
2565     EXPONENT : ('e'|'E') ('+'|'-')? ('0'..'9')+ ;
2566
2567 fragment
2568     FLOAT_TYPE_SUFFIX : ('f'|'F'|'d'|'D') ;
2569
2570 CHARACTER_LITERAL
2571     :   '\'' ( ESCAPE_SEQUENCE | ~('\''|'\\') ) '\''
2572     ;
2573
2574 STRING_LITERAL
2575     :   '"' ( ESCAPE_SEQUENCE | ~('\\"'|'"""') ) * '"'
2576     ;
2577
2578 fragment
2579     ESCAPE_SEQUENCE
2580     :   '\\\'' ('b'|'t'|'n'|'f'|'r'|'\\"'|'\''|'\\\'')
2581     |   UNICODE_ESCAPE
2582     |   OCTAL_ESCAPE
2583     ;
2584
2585 fragment
2586     OCTAL_ESCAPE
2587     :   '\\\'' ('0'..'3') ('0'..'7') ('0'..'7')
2588     |   '\\\'' ('0'..'7') ('0'..'7')
2589     |   '\\\'' ('0'..'7')
2590     ;
2591
2592 fragment
2593     UNICODE_ESCAPE
2594     :   '\\\'' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
2595     ;
2596
2597 IDENT
2598     :   JAVA_ID_START (JAVA_ID_PART)*

```

```

2599         ;
2600
2601 fragment
2602     JAVA_ID_START
2603         :   '\u0024'
2604         |   '\u0041' .. '\u005a'
2605         |   '\u005f'
2606         |   '\u0061' .. '\u007a'
2607         |   '\u00c0' .. '\u00d6'
2608         |   '\u00d8' .. '\u00f6'
2609         |   '\u00f8' .. '\u00ff'
2610         |   '\u0100' .. '\u1fff'
2611         |   '\u3040' .. '\u318f'
2612         |   '\u3300' .. '\u337f'
2613         |   '\u3400' .. '\u3d2d'
2614         |   '\u4e00' .. '\u9fff'
2615         |   '\uf900' .. '\uffff'
2616         ;
2617
2618 fragment
2619     JAVA_ID_PART
2620         :   JAVA_ID_START
2621         |   '\u0030' .. '\u0039'
2622         ;
2623
2624 WS : ( ' ' | '\r' | '\t' | '\u000C' | '\n' )
2625     {
2626         if (!preserveWhitespacesAndComments) {
2627             skip();
2628         } else {
2629             $channel = HIDDEN;
2630         }
2631     }
2632     ;
2633
2634 COMMENT
2635     :   '/*' ( options {greedy=false;} : . )* '*/'
2636     {
2637         if (!preserveWhitespacesAndComments) {
2638             skip();
2639         } else {
2640             $channel = HIDDEN;
2641         }
2642     }

```

```
2643         ;
2644
2645 LINE_COMMENT
2646     : '//' ~('\n'|\r)* '\r'? '\n'
2647     {
2648         if (!preserveWhitespacesAndComments) {
2649             skip();
2650         } else {
2651             $channel = HIDDEN;
2652         }
2653     }
2654     ;
```
