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.
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

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

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

```java

... 
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 \( t_a \) through (including) time \( t_e \) can be represented as \([t_a, t_e]\). 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.
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:
The code represents that \texttt{x} is a multiverse \texttt{int} and \texttt{y} is a non-multiverse \texttt{int} variable. Because \texttt{x} is multiverse it can have different values in different multiverses. As shown in Figure 1.2 \texttt{x} has three different values at three different times where data in the shaded box is the value of \texttt{x} and the time for which the value is valid is in brackets. \texttt{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:

<table>
<thead>
<tr>
<th>Value</th>
<th>timestamp(start, end)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(Jan 14, Feb 20)</td>
</tr>
<tr>
<td>4</td>
<td>(Feb 21, Apr 1)</td>
</tr>
<tr>
<td>5</td>
<td>(Apr 2, now)</td>
</tr>
</tbody>
</table>

Figure 1.2 also represents value and metadata of \texttt{y} (which is a non-multiverse integer) in a multiverse environment.
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
from both $y$ and $z$, which is $3 \times 10$ at time Jan 14 - Feb 20 and $5 \times 20$ 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.
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.*, $t_t = T_s.peek()$

- $T \circ t$ - establish $t$ as the current time variable, *i.e.*, $T_s.push(t)$ to start parsing a construct and $T_s.pop()$ after parsing.

- $T \circ \bot$ - establish loop boundary, $\bot$, when parsing

- $\varphi(t, X)$ - multiverse.Boolean constructor, *e.g.*, new multiverse.Boolean($t, X$)

Non-multiverse

$[\text{\@nonmultiverse } X](T) \equiv X$

Note that in the semantics, to reduce clutter, we use the annotation \texttt{@nonmultiverse} to switch to a universe programming paradigm, other than the annotation \texttt{@multiverse} to turn on the multiverse paradigm. This is syntactic sugar.
Constant
\[ [\text{constant}] (T) \equiv \text{constant} \]

Identifier
\[ [\text{var}] (T) \equiv \text{var} \]

Assignment
\[ [V=E] (T) \equiv [V] (T) \cdot \text{assign}(t_t, [E] (T)) \]

Right-associative binary operation
\[ [E_1 \text{ op } E_2] (T) \equiv [E_1] (T) \cdot \text{op}(t_t, [E_2] (T)) \]

Left-associative binary operation
\[ [E_1 \text{ op } E_2] (T) \equiv [E_2] (T) \cdot \text{op}(t_t, [E_1] (T)) \]

Unary operation
\[ [\text{op } E] (T) \equiv [E] (T) \cdot \text{op}(t_t) \]

Ternary (conditional) operation
\[ [ ( \text{ C } ) ? E_1 : E_2 ] (T) \equiv \]

// C may have side effects, execute only once

// Allocate variables to collect the time of C
private static Time t_1 = null;
private static Time t_2 = null;

// Build a function to store the true and false times
private Boolean F() {
    TimePair t_c = \varphi(t_t, [C] (T)) \cdot \text{getWhen}();
$t_1 = t_c . \text{whenTrue}();$

$t_2 = t_c . \text{whenFalse}();$

return true;

}  

// Evaluate alternatives at true and false times

// $F$ executed for side effects, expression returns a sequenced type

$(F) ? [E_1](T \circ t_1).\text{temporalUnion}([E_2](T \circ t_2))$

(*) Note that $F$, $t_1$, and $t_2$ are generated names

If

$[\text{if } (C) S_1 \text{ [else } S_2]](T) \equiv$

{

// Capture time of $C$

TimePair $t_c = \varphi(t_t, [C](T)).\text{getWhen}();$

// Evaluate true branch at times $C$ is true

Time $t = Q . \text{whenTrue}();$

if ($!t.\text{empty}()$) $[S_1](T \circ t)$

// Evaluate false branch at times $C$ is false

$t = t_c . \text{whenFalse}();$

if ($!t.\text{empty}()$) $[S_2](T \circ t)$

}

(*) Note that $t$, and $t_c$ are generated names

While

$[\text{while } (C) \text{ do } S](T) \equiv$

while (true) do {

TimePair $t_c = [C](T).\text{getWhen}();$

Time $t = t_c . \text{whenTrue}();$

}
if (t.empty()) break;
[S](T ∩ t)
}

Break (from loop)
[break](T) ≡
   // Remove current time, \( t_t \), from times in loop on stack
   // For every \( t_i \) on the stack before the first \( \bot \)
   t_i.subtractTime(\( t_t \));

Continue (in loop)
[continue](T) ≡
   // Set current time (top of TimeStack) to empty
   \( t_t = \text{new Time} \text{(null)} \);

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

GOTO
[GOTO L; ... L: ...](T) ≡
{
   Allocate a variable to accumulate the “GOTO” time
   \( t = \text{new Time} \text{(null)} \);
...  
// Replace GOTO by setting current time to null

t_{t} = \text{new Time(null)};  
...

// Replace L, merge GOTO time with current time

L:

t_{t}.\text{union}(t);

t = \text{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

[\text{try}\ \{\ S_{1} ; \ldots ; S_{n} \} \ \text{catch } E](T) \equiv
try \ \{\ [S_{1}](T) \} \ \text{catch } E

\ldots

try \ \{\ [S_{n}](T) \} \ \text{catch } E

Sequencing

[S_{1} ; S_{2}](T) \equiv

[S_{1}](T);

[S_{n}](T);

Do

[do \ S \ \text{while } (C)](T) \equiv

\{
    \text{Time } t = t_{t};
\}
do {
    
    $[S](T \circ \perp \circ t)$
    TimePair $t_c = [C](T).timePair()$
    if ($t_c.whenTrue().empty()$) break;
} while (true);
}

For

$[\text{for } (I; C; K) S](T) \equiv$

$[I](T)$

while (true) do {
    TimePair $t_c = [C](T).getWhen()$
    Time $t = t_c.whenTrue()$
    if ($t.empty()$) break;
    $[S](T \circ \perp \circ t)$
    $[K](T \circ \perp \circ t)$
}

For each

$[\text{for } (T V; C) S](T) \equiv$

multiverse.$TV;$

while (true) do {
    TimePair $t_c = \varphi(t, [C](T)).getWhen()$
    Time $t = t_c.whenTrue()$
    if ($t.empty()$) break;
    $[S](T \circ \perp \circ t)$
    $[K](T \circ \perp \circ t)$
}
Method call
\[ [\text{obj.method } (E_1, \ldots, E_n)](T) \equiv \text{obj.method } (t, [E_1](T), \ldots, [E_n](T)) \]

Method def
\[ [\text{modifiers } N (\ldots, Y_i P_i, \ldots) \{ S \}](T) \equiv \text{modifiers } N (\text{Time } t, \ldots, [Y_i P_i](T \circ t), \ldots) \]

Formal param
\[ [Y P](T) \equiv [Y](T) P \]

Type
\[ [Y](T) \equiv \text{multiverse.Y} \]

Class def
\[ [\text{class } N \{ S \}](T) \equiv \text{class } N \text{ extends } \text{Multiverse } \{ [S](T) \} \]

Package def
\[ [\text{package } N](T) \equiv \text{package multiverse.N} \]
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.
```java
/**
 * 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
/**
 * Output
 **/
// 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
@mv int y = 0;
@mv int f = 0;
@mv if (y < f)
{
    // Code if true
} else
{
    // Code if false
}

Multiverse
Java
Translator

mv.Integer y = new
mv.Integer(0);
mv.Integer f = new
mv.Integer(0);
TimeMapPair tp =
(f.lessThan(t,y)).timeMap
Pair();

TimeMap tTrue =
tp.trueTimeMap();
if(tTrue.notEmpty())
{
    // Code if true
}

TimeMap tFalse =
tp.falseTimeMap();
if (tFalse.notEmpty())
{
    // Code if false
}

Java Compiler

Figure 3.1: System Architecture

program.mvjava

program.java

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

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.
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

```java
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.
result = x + (y * z)

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 a lot of code to be rewritten. Moreover 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 timestamped 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 timestamped 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


[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,


APPENDIX
/**
 * An ANTLRv3 capable Java 1.5 grammar for building ASTs.
 * * Note that there's also the tree grammar 'JavaTreeParser.g' that can be fed
 * with this grammar's output.
 * *
 * Please report any detected errors or even suggestions regarding this grammar
 * to
 * with the subject
 * jsom grammar: [your subject note]
 * To generate a parser based on this grammar you'll need ANTLRv3, which you can
 * get from 'http://www.antlr.org'.
 * *
 * This grammar is published under the ...
 * BSD licence
 * Copyright (c) 2007–2008 by HABELITZ Software Developments
 * All rights reserved.
 * http://www.habilitz.com
 *
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the above copyright
* notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
* notice, this list of conditions and the following disclaimer in the
* documentation and/or other materials provided with the
distribution.

* 3. The name of the author may not be used to endorse or promote
products
derived from this software without specific prior written
permission.

* 

* THIS SOFTWARE IS PROVIDED BY HABELITZ SOFTWARE DEVELOPMENTS (‘HSD’)
‘‘AS IS’’
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL ‘HSD’ BE LIABLE FOR ANY DIRECT,
INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA,
OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGligence OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE,
EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*/

grammar Java;

options {
    backtracK = true;
    memoize = true;
    output = AST;
    ASTLabelType = CommonTree;
}

tokens {
    // operators and other special chars

    AND = '&';
<table>
<thead>
<tr>
<th>Line</th>
<th>Keyword</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>117</td>
<td>STAR</td>
<td><code>*</code></td>
</tr>
<tr>
<td>118</td>
<td>STAR_ASSIGN</td>
<td><code>*=</code></td>
</tr>
<tr>
<td>119</td>
<td>XOR</td>
<td><code>^</code></td>
</tr>
<tr>
<td>120</td>
<td>XOR_ASSIGN</td>
<td><code>^=</code></td>
</tr>
<tr>
<td>121</td>
<td></td>
<td></td>
</tr>
<tr>
<td>122</td>
<td>// keywords</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>ABSTRACT</td>
<td><code>'abstract'</code></td>
</tr>
<tr>
<td>124</td>
<td>ASSERT</td>
<td><code>'assert'</code></td>
</tr>
<tr>
<td>125</td>
<td>BOOLEAN</td>
<td><code>'boolean'</code></td>
</tr>
<tr>
<td>126</td>
<td>BREAK</td>
<td><code>'break'</code></td>
</tr>
<tr>
<td>127</td>
<td>BYTE</td>
<td><code>'byte'</code></td>
</tr>
<tr>
<td>128</td>
<td>CASE</td>
<td><code>'case'</code></td>
</tr>
<tr>
<td>129</td>
<td>CATCH</td>
<td><code>'catch'</code></td>
</tr>
<tr>
<td>130</td>
<td>CHAR</td>
<td><code>'char'</code></td>
</tr>
<tr>
<td>131</td>
<td>CLASS</td>
<td><code>'class'</code></td>
</tr>
<tr>
<td>132</td>
<td>CONTINUE</td>
<td><code>'continue'</code></td>
</tr>
<tr>
<td>133</td>
<td>DEFAULT</td>
<td><code>'default'</code></td>
</tr>
<tr>
<td>134</td>
<td>DO</td>
<td><code>'do'</code></td>
</tr>
<tr>
<td>135</td>
<td>DOUBLE</td>
<td><code>'double'</code></td>
</tr>
<tr>
<td>136</td>
<td>ELSE</td>
<td><code>'else'</code></td>
</tr>
<tr>
<td>137</td>
<td>ENUM</td>
<td><code>'enum'</code></td>
</tr>
<tr>
<td>138</td>
<td>EXTENDS</td>
<td><code>'extends'</code></td>
</tr>
<tr>
<td>139</td>
<td>FALSE</td>
<td><code>'false'</code></td>
</tr>
<tr>
<td>140</td>
<td>FINAL</td>
<td><code>'final'</code></td>
</tr>
<tr>
<td>141</td>
<td>FINALLY</td>
<td><code>'finally'</code></td>
</tr>
<tr>
<td>142</td>
<td>FLOAT</td>
<td><code>'float'</code></td>
</tr>
<tr>
<td>143</td>
<td>FOR</td>
<td><code>'for'</code></td>
</tr>
<tr>
<td>144</td>
<td>IF</td>
<td><code>'if'</code></td>
</tr>
<tr>
<td>145</td>
<td>IMPLEMENTS</td>
<td><code>'implements'</code></td>
</tr>
<tr>
<td>146</td>
<td>INSTANCEOF</td>
<td><code>'instanceof'</code></td>
</tr>
<tr>
<td>147</td>
<td>INTERFACE</td>
<td><code>'interface'</code></td>
</tr>
<tr>
<td>148</td>
<td>IMPORT</td>
<td><code>'import'</code></td>
</tr>
<tr>
<td>149</td>
<td>INT</td>
<td><code>'int'</code></td>
</tr>
<tr>
<td>150</td>
<td>LONG</td>
<td><code>'long'</code></td>
</tr>
<tr>
<td>151</td>
<td>NATIVE</td>
<td><code>'native'</code></td>
</tr>
<tr>
<td>152</td>
<td>NEW</td>
<td><code>'new'</code></td>
</tr>
<tr>
<td>153</td>
<td>NULL</td>
<td><code>'null'</code></td>
</tr>
<tr>
<td>154</td>
<td>PACKAGE</td>
<td><code>'package'</code></td>
</tr>
<tr>
<td>155</td>
<td>PRIVATE</td>
<td><code>'private'</code></td>
</tr>
<tr>
<td>156</td>
<td>PROTECTED</td>
<td><code>'protected'</code></td>
</tr>
<tr>
<td>157</td>
<td>PUBLIC</td>
<td><code>'public'</code></td>
</tr>
<tr>
<td>158</td>
<td>RETURN</td>
<td><code>'return'</code></td>
</tr>
<tr>
<td>159</td>
<td>SHORT</td>
<td><code>'short'</code></td>
</tr>
<tr>
<td>160</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
35
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 ARRAYDECLARATOR;
188 ARRAYDECLARATOR_LIST;
189 ARRAYELEMENT_ACCESS;
190 ARRAY_INITIALIZER;
191 BLOCK_SCOPE;
192 CAST_EXPR;
193 CATCH_CLAUSE_LIST;
194 CLASSCONSTRUCTOR_CALL;
195 CLASSINSTANCE_INITIALIZER;
196 CLASSSTATIC_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;
@header {

    package sequencedjava.grammar;

}
private List<String> mMessages;

TokenRewriteStream tokens;

Boolean isSequence = false;
Boolean temp = false;
String ident="";
Boolean forStart = false;
Boolean forBlock = false;
String relationalOperator ="";
SymbolTable st = new SymbolTable();
String opr = "";
Boolean doIt;
String doDont="";

public void setTokenStream(TokenRewriteStream input) {
    //super.setTreeNodeStream(input);
    tokens = input;
}

Stack<String> timeMapStack = new Stack<String>();
Stack<String> timeMapPairStack = new Stack<String>();
TimeMap allTime = new TimeMap();
Boolean sequencedIf = false;
Boolean ifSeqStatement = false;
String ifcondition = "";
int count = 0;
Boolean sequencedElse = false;
Boolean seqIf = false;

String forcondition = "";
Boolean whileBlock = false;
Boolean doWhile = false;
Boolean doPara = false;

Boolean switchCase = false;
Stack<String> cases = new Stack<String>();
    Stack<String> statements = new Stack<String>();
    Boolean isBreak = false;
    Boolean leftCurly = false;
    Boolean rightCurly = false;
    Boolean breakTrue = false;
    Boolean methodBlock= false;
Iterator x;
String returnType = "void";
Boolean sequencedReturn = false;
Boolean methodCall = false;

// variable for operation identification

final int and = 1;
final int and_assign = 2;
final int assign = 3;
final int bit_shift_right = 4;
final int bit_shift_right_assign = 5;
final int dec = 6;
final int div = 7;
final int div_assign = 8;
final int equal = 9;
final int greater_or_equal = 10;
final int greater_than = 11;
final int inc = 12;
final int less_or_equal = 13;
final int less_than = 14;
final int logical_and = 15;
final int logical_not = 16;
final int logical_or = 17;
final int minus = 18;
final int minus_assign = 19;
final int mod = 20;
final int mod_assign = 21;
final int not = 22;
final int not_equal = 23;
final int or = 24;
final int or_assign = 25;
final int plus = 26;
final int plus_assign = 27;
final int shift_left = 28;
final int shift_left_assign = 29;
final int shift_right = 30;
final int shift_right_assign = 31;
final int star = 32;
final int star_assign = 33;
final int xor = 34;
final int xor_assign = 35;

/**
Switches error message collection on or off.
The standard destination for parser error messages is `<code>System.err</code>`.
However, if `<code>true</code>` gets passed to this method this default behaviour will be switched off and all error messages will be collected instead of written to anywhere.
The default value is `<code>false</code>`.

@param pNewState `<code>true</code>` if error messages should be collected.

```java
public void enableErrorMessageCollection(boolean pNewState) {
    mMessageCollectionEnabled = pNewState;
    if (mMessages == null && mMessageCollectionEnabled) {
        mMessages = new ArrayList<String>();
    }
}
```

Collects an error message or passes the error message to `<code>super.emitErrorMessage(...)</code>`.
The actual behaviour depends on whether collecting error messages has been enabled or not.

@override
```java
public void emitErrorMessage(String pMessage) {
    if (mMessageCollectionEnabled) {
        mMessages.add(pMessage);
    } else {
        super.emitErrorMessage(pMessage);
    }
}
```

Returns collected error messages.
* @return A list holding collected error messages or `<code>null</code>` if collecting error messages hasn’t been enabled. Of course, this list may be empty if no error message has been emitted.

```java
public List<String> getMessages() {
    return mMessages;
}
```

/∗∗
* Tells if parsing a Java source has caused any error messages.
* @return `<code>true</code>` if parsing a Java source has caused at least one error message.
*/

```java
public boolean hasErrors() {
    return mHasErrors;
}
```

@lexer::header {
    package sequencedjava.grammar;
}

@lexer::members {
    /∗∗
* Determines if whitespaces and comments should be preserved or thrown away.
* If `<code>true</code>` whitespaces and comments will be preserved within the hidden channel, otherwise the appropriate tokens will be skipped. This is a ‘little bit’ expensive, of course. If only one of the two behaviours is needed forever the lexer part of the grammar should be changed by replacing * the ‘if–else’ stuff within the appropriate lexer grammar actions.
*/

```java
public boolean preserveWhitespacesAndComments = true;
```
}

/*
@rulecatch
{
    catch(RecognitionException re) { throw re; }
} /*

// Starting point for parsing a Java file.
javaSource : compilationUnit
    -> "(JAVA_SOURCE compilationUnit)
;

compilationUnit :
    {
        st.enterScope();
        timeMapStack.push("allTime");
        timeMapPairStack.push("tpAllTime");
    }
annotationList
packageDeclaration?
importDeclaration*
typeDecls*
    {
        //System.out.println("last timeMapStack " + timeMapStack.peek());
        //System.out.println("last timeMapPairStack " + timeMapPairStack.peek());
        timeMapPairStack.pop();
        timeMapStack.pop();
        st.exitScope();
    }
;
typeDecls : typeDeclaration
            | SEMI!
;
packageDeclaration : PACKAGE" qualifiedIdentifier SEMI!
;
importDeclaration
typeDeclaration
    : modifierList!
      ( classTypeDeclaration [ $modifierList . tree ]
        | interfaceTypeDeclaration [ $modifierList . tree ]
        | enumTypeDeclaration [ $modifierList . tree ]
        | annotationTypeDeclaration [ $modifierList . tree ]
      )
    ;

classTypeDeclaration [ CommonTree modifiers ]
    : CLASS IDENT genericTypeParameterList? classExtendsClause?
      implementsClause? classBody
      -> "(CLASS { $modifiers } IDENT genericTypeParameterList?
        classExtendsClause? implementsClause? classBody)
    ;

classExtendsClause
    : EXTENDS type
      -> "(EXTENDS_CLAUSE[$EXTENDS, "EXTENDS_CLAUSE"] type)
    ;

interfaceExtendsClause
    : EXTENDS typeList
      -> "(EXTENDS_CLAUSE[$EXTENDS, "EXTENDS_CLAUSE"] typeList)
    ;

implementsClause
    : IMPLEMENTS typeList
      -> "(IMPLEMENTES_CLAUSE[$IMPLEMENTES, "IMPLEMENTES_CLAUSE"]
        typeList)
    ;

genericTypeParameterList
    : LESS_THAN genericTypeParameter (COMMA genericTypeParameter)*
      genericTypeListClosing
      -> "(GENERIC_TYPE_PARAM_LIST[$LESS_THAN, "GENERIC_TYPE_PARAM_LIST"]
        genericTypeParameter+ )
    ;
genericTypeListClosing  // This 'trick' is fairly dirty - if there's some time a better solution should
    // be found to resolve the problem with nested generic type parameter lists
    // (i.e. <T1 extends AnyType<T2>> for generic type parameters or <T1<T2>> for
    // generic type arguments etc).
        : GREATER_THAN
        | SHIFT_RIGHT
        | BIT_SHIFT_RIGHT
        | // nothing
        :

genericTypeParameter
    : IDENT bound?
        -> ^(IDENT bound?)
    :
    bound
        : EXTENDS type (AND type)*/
        -> ^(EXTENDS_BOUND_LIST[$EXTENDS, "EXTENDS_BOUND_LIST"] type+)
    :

typeDeclaration[CommonTree modifiers]
    : ENUM IDENT implementsClause? enumBody
        -> ^(ENUM {$modifiers} IDENT implementsClause? enumBody)
    :
    enumBody
        : LCURLY {st.enterScope();}enumScopeDeclarations {st.exitScope();}RCURLY
        -> ^(ENUM_TOP_LEVEL_SCOPE[$LCURLY, "ENUM_TOP_LEVEL_SCOPE"]
            enumScopeDeclarations)
    :

enumScopeDeclarations
    : enumConstants (COMMA!)? enumClassScopeDeclarations?
    :

enumClassScopeDeclarations
    : SEMI classScopeDeclarations*
        -> ^(CLASS_TOP_LEVEL_SCOPE[$SEMI, "CLASS_TOP_LEVEL_SCOPE"]
            classScopeDeclarations*)
    :
enumConstants
  :  enumConstant \texttt{(COMMA! enumConstant)}* \\
  ;

enumConstant
  :  annotationList IDENT' arguments? classBody? \\
  ;

interfaceTypeDeclaration[CommonTree modifiers]
  :  INTERFACE IDENT genericTypeParameterList? interfaceExtendsClause \\
     ? interfaceBody \\
     \rightarrow  "(INTERFACE \{modifiers\} IDENT genericTypeParameterList? \\
                  interfaceExtendsClause? interfaceBody)"

  ;

typeList
  :  type \texttt{(COMMA! type)}* \\
  ;

classBody
  :  LCURLY \{st.enterScope();\} classScopeDeclarations* \{st.exitScope ()\}RCURLY \\
     \rightarrow  "(CLASS_TOP_LEVEL_SCOPE[SLCURLY, "CLASS_TOP_LEVEL_SCOPE"] \\
                  classScopeDeclarations *)"

  ;

interfaceBody
  :  LCURLY \{st.enterScope();\} interfaceScopeDeclarations* \{st. 
                 exitScope ()\}RCURLY \\
     \rightarrow  "(INTERFACE_TOP_LEVEL_SCOPE[SLCURLY, "CLASS_TOP_LEVEL_SCOPE" \\
                  ] interfaceScopeDeclarations *)"

  ;

classScopeDeclarations
  :  block  \rightarrow  "(CLASS_INSTANCE_INITIALIZER block) \\
  \texttt{|}  STATIC block  \rightarrow  "(CLASS_STATIC_INITIALIZER[SSTATIC, \\
                           "CLASS_STATIC_INITIALIZER"] block) \\
  \texttt{|}  modifierList \\
          ( genericTypeParameterList? \\
            \{ q=type \{if(isSequence) returnType=\$q.text:} IDENT \\
            formalParameterList arrayDeclaratorList? throwsClause? ( \\
            block \texttt{|} SEMI) \


interfaceScopeDeclarations
  : modifierList
    ( genericTypeParameterList?
      ( type IDENT formalParameterList arrayDeclaratorList? throwsClause? SEMI
        -> "FUNCTION_METHOD_DECL modifierList
genericTypeParameterList? type IDENT
formalParameterList arrayDeclaratorList?
throwsClause? block?)
      VOID IDENT formalParameterList throwsClause? (block | SEMI)
        -> "VOID_METHOD_DECL modifierList
genericTypeParameterList? IDENT formalParameterList
throwsClause? block?
      id=IDENT formalParameterList throwsClause? block
        -> "CONSTRUCTOR_DECL[ $id , "CONSTRUCTORDECL"]
modifierList genericTypeParameterList?
formalParameterList throwsClause? block)
    )
  | type classFieldDeclaratorList SEMI
    -> "VAR_DECLARATION modifierList type
classFieldDeclaratorList"
  )
  | typeDeclaration
  | SEMI!

;
class FieldDeclaratorList
  : q=classFieldDeclarator (COMMA classFieldDeclarator)*
  -> "(VARDECLARATOR_LIST classFieldDeclarator+)
  ;

classFieldDeclarator
  : variableDeclaratorId (ASSIGN variableInitializer)?
  -> "(VARDECLARATOR variableDeclaratorId variableInitializer?)
  ;

interfaceFieldDeclaratorList
  : interfaceFieldDeclarator (COMMA interfaceFieldDeclarator)*
  -> "(VARDECLARATOR_LIST interfaceFieldDeclarator+)
  ;

interfaceFieldDeclarator
  : variableDeclaratorId ASSIGN variableInitializer
  -> "(VARDECLARATOR variableDeclaratorId variableInitializer)
  ;

variableDeclaratorId
  : q=IDENT { 
    if (isSequence)
    {
      st.addId($q.text, true);
      isSequence = false;
    }
  } else
  { 
    st.addId($q.text, false);
  }
  } arrayDeclaratorList? 

variableInitializer
  : arrayInitializer
  | expression
  ;

arrayDeclarator
  : LBRACK RBRACK 
  -> "(ARRAY,DECLARATOR)
  ;
arrayDeclaratorList
  : arrayDeclarator+
  -> "(ARRAY DECLARATOR LIST arrayDeclarator+)
  ;

arrayInitializer
  : LCURLY (variableInitializer (COMMA variableInitializer)* COMMA?) ? RCURLY
  -> "(ARRAY_INITIALIZER [SLCURLY, "ARRAY_INITIALIZER"]
    variableInitializer*)
  ;

throwsClause
  : THROWS qualifiedIdentList
  -> "(THROWS_CLAUSE [STHROWS, "THROWS_CLAUSE"]
    qualifiedIdentList
  )
  ;

modifierList
  : modifier*
  -> "(MODIFIER LIST modifier*)
  ;

modifier
  : PUBLIC
  | PROTECTED
  | PRIVATE
  | STATIC
  | ABSTRACT
  | NATIVE
  | SYNCHRONIZED
  | TRANSIENT
  | VOLATILE
  | STRICTFP
  | localModifier
  ;

localModifierList
  : localModifier*
  -> "(LOCAL_MODIFIER LIST localModifier*)
  ;

localModifier

type
  : simpleType
      | objectType

simpleType // including static arrays of simple type elements
  : primitiveType arrayDeclaratorList?
      -> "(TYPE primitiveType arrayDeclaratorList?)"

objectType // including static arrays of object type reference elements
  : qualifiedTypeIdent arrayDeclaratorList?
      -> "(TYPE qualifiedTypeIdent arrayDeclaratorList?)"

objectTypeSimplified
  : qualifiedTypeIdentSimplified arrayDeclaratorList?
      -> "(TYPE qualifiedTypeIdentSimplified arrayDeclaratorList?)"

qualifiedTypeIdent
  : typeIdent (DOT typeIdent)*
      -> "(QUALIFIED_TYPE_IDENT typeIdent+)

qualifiedTypeIdentSimplified
  : typeIdentSimplified (DOT typeIdentSimplified)*
      -> "(QUALIFIED_TYPE_IDENT typeIdentSimplified+)

typeIdent
  : IDENT^
genericTypeArgumentList?
;

typeIdentSimplified
: IDENT* genericTypeArgumentListSimplified?
;

primitiveType
: BOOLEAN
{
    if (isSequence)
    {
        tokens.replace($BOOLEAN, "sequenced.Boolean");
        ident = "sequenced.Boolean";
    }
}
| CHAR
{
    if (isSequence)
    {
        tokens.replace($CHAR, "sequenced.Char");
        ident = "sequenced.Char";
    }
}
| BYTE
{
    if (isSequence)
    {
        tokens.replace($BYTE, "sequenced.Byte");
        ident = "sequenced.Byte";
    }
}
| SHORT
{
    if (isSequence)
    {
        tokens.replace($SHORT, "sequenced.Short");
        ident = "sequenced.Short";
    }
}
| INT
{
    if (isSequence)
    {
        tokens.replace($INT, "sequenced.Integer");
    }
ident = "sequenced.Integer";

}  

}  

| LONG  

{
  if (isSequence)
  {
    tokens.replace($LONG, "sequenced.Long");
    ident = "sequenced.Long";
  }
  }

| FLOAT  

{
  if (isSequence)
  {
    tokens.replace($FLOAT, "sequenced.Float");
    ident = "sequenced.Float";
  }
  }

| DOUBLE  

{
  if (isSequence)
  {
    tokens.replace($DOUBLE, "sequenced.Double");
    ident = "sequenced.Double";
  }
  }


genericTypeArgumentList  

: LESS_THAN genericTypeArgument (COMMA genericTypeArgument)*
genericTypeListClosing  

-> "($GENERIC_TYPE_ARG_LIST[$LESS_THAN, "$GENERIC_TYPE_ARG_LIST"]
genericTypeArgument+)

;

genericTypeArgument  

: type  

| QUESTION genericWildcardBoundType?

  -> "($QUESTION genericWildcardBoundType?)

  ;

genericWildcardBoundType
(EXTENDS | SUPER)^ type

genericTypeArgumentListSimplified

: LESS_THAN genericTypeArgumentListSimplified (COMMA

genericTypeArgumentListSimplified)* genericTypeListClosing

-> "(GENERIC_TYPE_ARG_LIST [LESS_THAN, "GENERIC_TYPE_ARG_LIST"]
genericTypeArgumentListSimplified+)

;  

genericTypeArgumentListSimplified

: type

| QUESTION

;  

qualifiedIdentList

: qualifiedIdentifier (COMMA! qualifiedIdentifier)*

;  

formalParameterList

: LPAREN

(

| if(isSequence)

{  

tokens.insertAfter(SLPAREN, "TimeMap timeMap.");

isSequence = false;

methodBlock = true;

}

}

// Contains at least one standard argument declaration and
// optionally a variable argument declaration.

formalParameterStandardDecl (COMMA

formalParameterStandardDecl)+ (COMMA

formalParameterVarArgDecl)?

-> "(FORMAL_PARAM_LIST [SLPAREN, "FORMAL_PARAM_LIST"]

formalParameterStandardDecl+ formalParameterVarArgDecl?)

// Contains a variable argument declaration only.

| formalParameterVarArgDecl

-> "(FORMAL_PARAM_LIST [SLPAREN, "FORMAL_PARAM_LIST"]

formalParameterVarArgDecl)

// Contains nothing.

| -> "(FORMAL_PARAM_LIST [SLPAREN, "FORMAL_PARAM_LIST"]

)
RPAREN

formalParameterStandardDecl
  : localModifierList type variableDeclaratorId
  -> "(FORMAL_PARAM_STDDECL localModifierList type variableDeclaratorId)"

formalParameterVarArgDecl
  : localModifierList type ELLIPSIS variableDeclaratorId
  -> "(FORMAL_PARAM_VARARGDECL localModifierList type variableDeclaratorId)"

qualifiedIdentifier
  : ( i=IDENT { 
      if (isSequence)
      {
        st.addId($i.text, true);
        //isSequence=false;
      }
      else
      {
        st.addId($i.text, false);
      }
    } -> IDENT

  )
  ( DOT ident=IDENT -> "(DOT $qualifiedIdentifier $ident)"

  )

// ANNOTATIONS

annotationList
  : annotation*
  -> "(ANNOTATION_LIST annotation*)"

annotation
  : AT quot q=qualifiedIdentifier z=annotationInit?
  {
    //System.out.println("IN ");
    if ($q.text.equals("Sequence"))
{  
isSequence = true;
  temp = true;
  // System.out.println("IN ");
  if ($z.text != null)
    tokens.replace($AT.getTokenIndex(), $z.stop.getTokenIndex(), "");
  else
    tokens.replace($AT.getTokenIndex(), $q.stop.getTokenIndex(), ");
  // System.out.println("OUT ");
}

;  
annotationInit
  : LPAREN annotationInitializers RPAREN
  -> "(ANNOTATION_INIT_BLOCK[SLPAREN, "ANNOTATION_INIT_BLOCK"]
    annotationInitializers)
  ;

annotationInitializers
  : annotationInitializer (COMMA annotationInitializer)*
  -> "(ANNOTATION_INIT_KEY_LIST annotationInitializer+)
    annotationElementValue // implicit initialization of the
    annotation field 'value'
  -> "(ANNOTATION_INIT_DEFAULT_KEY annotationElementValue)
  ;

annotationInitializer
  : IDENT^ ASSIGN! annotationElementValue
  ;

annotationElementValue
  : annotationElementValueExpression
  | annotation
  | annotationElementValueArrayInitializer
  ;

annotationElementValueExpression
  : conditionalExpression
  -> "(EXPR conditionalExpression)"  ;
annotationElementValueArrayInitializer
  :  LCURLY (annotationElementValue (COMMA annotationElementValue)*)?
      (COMMA)? RCURLY
     ->  "(ANNOTATION_INIT_ARRAY_ELEMENT[$LCURLY, " ANNOTATION_ELEM_VALUE_ARRAY_INIT"] annotationElementValue *)"
  ;
annotationTypeDeclaration[CommonTree modifiers]
  :  AT INTERFACE IDENT annotationBody
     ->  "(AT {$modifiers} IDENT annotationBody)
  ;
annotationBody
  :  LCURLY annotationScopeDeclarations* RCURLY
     ->  "(ANNOTATION_TOP_LEVEL_SCOPE[$LCURLY, "CLASS_TOP_LEVEL_SCOPE"] annotationScopeDeclarations *)"
  ;
annotationScopeDeclarations
  :  modifierList type
     ( IDENT LPAREN RPAREN annotationDefaultValue? SEMI
      ->  "(ANNOTATION_METHODDECL modifierList type IDENT annotationDefaultValue?)
      |  classFieldDeclaratorList SEMI
      ->  "(VARDECLARATION modifierList type
classFieldDeclaratorList)
      |
      |  typeDeclaration
     )
  ;
annotationDefaultValue
  :  DEFAULT* annotationElementValue
  ;
// STATEMENTS / BLOCKS
block
  :  LCURLY
     {
      st.enterScope();
     }
     if (leftCurly)
     {

tokens.replace($LCURLY,"" );
}

// code to insert for Sequenced for Loop
if (forBlock)
{
  timeMapPairStack.push("tp"+count);
count++;
timeMapStack.push("tTrue"+count);
count++;

tokens.insertAfter($LCURLY,"\n \t if ("+ timeMapStack.peek()+" .
   empty () ) break; ");
tokens.insertAfter($LCURLY,"\n \t TimeMap"+timeMapStack.peek()
   +" = "+timeMapPairStack.peek()+" .trueTimeMap () ; ");
tokens.insertAfter($LCURLY,"\n \t TimeMapPair "+
   timeMapPairStack.peek()+" = "+forcondition+" .timeMapPair () ; ");

  forBlock = false;
}

// code to insert in sequenced while block
if (whileBlock)
{
  timeMapPairStack.push("tp"+count);
count++;
timeMapStack.push("tTrue"+count);
count++;

tokens.insertAfter($LCURLY,"\n \t if ("+ timeMapStack.peek()+" .
   empty () ) break; ");
tokens.insertAfter($LCURLY,"\n \t TimeMap"+timeMapStack.peek()
   +" = "+timeMapPairStack.peek()+" .trueTimeMap () ; ");
tokens.insertAfter($LCURLY,"\n \t TimeMapPair "+
   timeMapPairStack.peek()+" = "+forcondition+" .timeMapPair () ; ");

  //whileBlock = false;
}

if (methodBlock && !("void" .equals (returnType)))
{
  tokens.insertAfter($LCURLY,"\n"+returnType+" ret1 = new "+
returnType+" ("+timeMapPairStack.peek()+" ); ");
methodBlock = false;
returnType="void";

/* if (doWhile)
{
    timeMapPairStack.push("tp"+count);
    count++;
    timeMapStack.push("tTrue"+count);
    count++;

    tokens.insertAfter(SLCURLY, "\n \t if ("+timeMapStack.peek() +".empty()) break;");
    tokens.insertAfter(SLCURLY, "\n \t TimeMap"+timeMapStack.peek() "+" = "+timeMapPairStack.peek()+".trueTimeMap();");
    tokens.insertAfter(SLCURLY, "\n \t TimeMapPair"+timeMapPairStack.peek() +" = "+forcondition+".timeMapPair();");
*/

}

blockStatement* { st.exitScope(); } RCURLY

{ 
    if (doPara) 
    { 
        tokens.replace(SRCURLY,"");
        doPara = false;
    } 
    
    if (rightCurly) 
    { 
        tokens.replace(SRCURLY,"");
    } 
    
}

-> "(BLOCK_SCOPE[SLCURLY, "BLOCK_SCOPE"] blockStatement*)

; 

blockStatement 
: localVarDeclaration SEMI!
| typeDeclaration
| q=statement
;
localVariableDeclaration
    : localModifierList type classFieldDeclaratorList
    -> "(VAR DECLARATION localModifierList type
classFieldDeclaratorList)"
    ;

statement
    : block
    | annotation statement
    | ASSERT expr1=expression
      ( COLON expr2=expression SEMI
        -> "(ASSERT $expr1 $expr2)"
      )
    | SEMI
    -> "(ASSERT $expr1"
    )
    | IF
    { if (isSequence)
      { sequencedIf = true;
      }
      }
q=parenthesizedExpression
{ ifCondition = Sq.text;
  if (sequencedIf)
  {
    timeMapPairStack.push("tp"+count);
    count++;
    timeMapStack.push("tTrue"+count);
    count++;
    tokens.replace(Sq.start getTokenIndex(),Sq.stop.
    getTokenIndex(),"("+timeMapStack.peek()+"."+notEmpty
    (")")");
    tokens.insertBefore(SIF,"\t TimeMap "+timeMapStack.peek()+
    " = "+timeMapPairStack.peek()+".trueTimeMap();\n")
    tokens.insertBefore(SIF,"TimeMapPair "+ timeMapPairStack.
    peek()+" = "+ifCondition+".timeMapPair();\n")
  }
// sequencedIf = false;
sequencedElse = true;
seqIf = true;
// timeMapStack.pop();
}
}
ifStat = statement { if (sequencedIf) { timeMapStack.pop(); }
( ELSE
{
if (sequencedElse)
{
timeMapStack.push("tFalse"+count);
count++;
tokens.replace($ELSE,"\t if ("+ timeMapStack.peek()
 +".notEmpty())");
tokens.insertBefore($ELSE,"\t TimeMap "+ timeMapStack.
peek() +" = "+timeMapPairStack.peek()+".falseTimeMap
();\n");
sequencedElse = false;
}
} elseStat = statement
{
// remove the timemap and timeMapPair


timeMapStack.pop();
if (seqIf)
{
// System.out.println("In seqIf " + timeMapPairStack.peek() );
timeMapPairStack.pop();
seqIf = false;
sequencedIf = false;
sequencedElse = false;
}
}

-> ^if parenthesizedExpression
   $ifStat $elseStat
|
```java
{  
  if (seqIf)  
  {  
    // System.out.println("In seqIf " + timeMapPairStack.peek());  
    timeMapPairStack.pop();  
    seqIf = false;  
    sequencedElse = false;  
    sequencedIf = false;  
  }  
}  

-> "(IF parenthesizedExpression $ifStat)"

| FOR {  
|   if (isSequence)  
|   {  
|     forBlock = true;  
|     forStart = true;  
|   }  
|   }  
| LPAREN
|   ( forInit SEMI z=forCondition
|   {  
|     if (forStart)  
|     {  
|       // System.out.println("for condition " + $z.text);  
|       forCondition = $z.text;  
|       tokens.replace($z.start.getTokenIndex(),$z.stop.getTokenIndex() + 1,":");  
|       forStart = false;  
|     }  
|   }  
|   SEMI forUpdater RPAREN statement
|   {  
|     // POPPING timeMap and TimeMapPair  
|     timeMapPairStack.pop();  
|     timeMapStack.pop();  
|   }
|   -> "(FOR forInit forCondition forUpdater statement)"
|   | localModifierList type IDENT COLON expression RPAREN
|   | statement
|   -> "(FOR_EACH[$FOR, "FOR_EACH"]
|   | localModifierList type IDENT expression statement)"
```
WHILE
{
    if (isSequence)
    {
        whileBlock = true;
        // forStart = true;
    }
}
q = parenthesizedExpression
{
    if (whileBlock)
    {
        forCondition = $q.text;
        tokens.replace($q.start.getTokenIndex(), $q.stop.
getTokenIndex(), "(true)");
    }
}
statement
{
    // POPPING timeMap and TimeMapPair
    if (whileBlock)
    {
        // System.out.println("In while +timeMapPairStack.peek() ");
        if (breakTrue == false)
        {
            timeMapPairStack.pop();
        }
        timeMapStack.pop();
        whileBlock = false;
        breakTrue = false;
    }
}
-> "(WHILE parenthesizedExpression statement)
DO
{
    if (isSequence)
    {
        doWhile = true;
        doPara = true;
// tokens.insertBefore($DO,)"t TimeMap" + timeMapStack.peek() ++" = " + timeMapPairStack.peek() + ".falseTimeMap();\n);"
)

} statement WHILE q=parenthesizedExpression
{
  if (doWhile)
  {
    // code
    timeMapPairStack.push("tp"+count);
    count++;
    timeMapStack.push("tTrue"+count);
    count++;
    tokens.insertBefore($WHILE,"t TimeMap" + timeMapStack.peek() ++" = " + timeMapPairStack.peek() + ".trueTimeMap();\n);"
    tokens.insertBefore($WHILE,"TimeMapPair" + timeMapPairStack.peek() ++" = " + Sq.text + ".timeMapPair();\n);"
    tokens.replace(Sq.start.getTokenIndex(), Sq.stop getTokenIndex(), "(true)\n);"
    doWhile = false;
  }
}

| (DO statement parenthesizedExpression)
| TRY block (catches finallyClause? | finallyClause?)
| (TRY block catches? finallyClause?)

if (isSequence)
{
  switchCase = true;
  doPara = true;
  leftCurly = true;
  rightCurly = true;
}
tokens.replace($SWITCH,\n);"
q=parenthesizedExpression { tokens.replace($q.start.getTokenIndex(), $q.stop.getTokenIndex(), "") } LCURLY {

tokens.replace($LCURLY, "")

st.enterScope();
}

switchBlockLabels
{
    st.exitScope();
cases.removeAllElements();
statements.removeAllElements();
switchCase = false;
doPara = false;
leftCurly = false;
rightCurly = false;
}
RCURLY { tokens.replace($RCURLY, "") } -> "(SWITCH parenthesizedExpression switchBlockLabels)" || SYNCHRONIZED parenthesizedExpression block

RETURN {
    if(isSequence)
    {
        sequencedReturn = true;
        isSequence = false;
    }
    }
q=expression?
{
    if(sequencedReturn)
    {
        tokens.replace($RETURN, "ret1.merge("+$q.text+";\n");
        String temp = timeMapStack.peak();
tokens.insertAfter($RETURN, temp+" = new Time(null);\n");

        //tokens.replace($RETURN, "")
tokens.replace($q.start.getTokenIndex(), $q.stop.getTokenIndex(), "")

        sequencedReturn = false;
    }
    }
}
SEMIS

→ `(RETURN expression?)

| THROW expression SEMI

→ `(THROW expression)

| {isBreak = true;} BREAK

{ if(isSequence)

| String temp = timeMapStack.peek();

| timeMapPairStack.pop();

| breakTrue = true;

| x = timeMapPairStack.iterator();

| while (x.hasNext())

| {

| tokens.insertAfter($BREAK, x.next()+".subtractTime("+temp+"):\n");

| }

| tokens.replace($BREAK,"\\n");

| isSequence = false;

|

} } IDENT? SEMI

→ `(BREAK IDENT?)

| CONTINUE

{ if(isSequence)

| String temp = timeMapStack.peek();

| tokens.replace($CONTINUE, temp+" = new Time(null) ;");

| isSequence = false;

|

} } IDENT? SEMI

→ `(CONTINUE IDENT?)

| IDENT COLON statement

→ `(LabeledStatement IDENT statement)

| expression SEMI!

| SEMI

;
catches : catchClause+ -> "((CATCH_CLAUSE_LIST catchClause+) ;
catchClause : CATCH LPAREN! formalParameterStandardDecl RPAREN! block ;
finallyClause : FINALLY block -> block ;
switchBlockLabels : switchCaseLabels switchDefaultLabel? switchCaseLabels
  -> "((SWITCH_BLOCK_LABEL_LIST switchCaseLabels
      switchDefaultLabel? switchCaseLabels)
switchCaseLabels : switchCaseLabel* ;
switchCaseLabel : CASE z=expression COLON! q=blockStatement* //\{System.out.
  println(isBreak);\}
  { //System.out.println(switchCase+"in case labels "+isBreak); if (switchCase)
    {
      cases.push($z.text);
      statements.push($q.text);
      //System.out.println("in case labels "+isBreak);
      tokens.replace($z.start.getTokenIndex(),$q.stop.
        getTokenIndex(),"" );
tokens.replace($CASE,"" );
      if (isBreak == true) {
        String code = "" ;
int size = cases.size();

for (int i = 0; i < size; i++) {
    code = statements.pop() + "\n" + code;
    //System.out.println("if(" + cases.pop() + ")\n" + code);
    tokens.insertAfter(SCASE,"if(" + cases.pop() + ")\n" + code+"})\n");
}

isBreak = false;

switchDefaultLabel
    : DEFAULT COLON! blockStatement*

forInit
    :
        localVariableDeclaration -> ^(FOR_INIT
                        localVariableDeclaration)
        | expressionList -> ^(FOR_INIT expressionList)
        | -> ^(FOR_INIT)

;

forCondition
    :

q=expression? /*{System.out.println("forcon "+Sq.text);}*/
    -> ^(FOR_CONDITION expression?)

;

forUpdater
text);}]
    -> ^(FOR_UPDATE expressionList)

;

// EXPRESSIONS

parenthesizedExpression
   : LPAREN expression RPAREN
   -> "(PARENTESIZED_EXPR[SLPAREN, "PARENTESIZED_EXPR"] expression )"
   ;

expressionList
   : q=expression
   
   
   if(isSequence)
   {
   
   tokens.insertBefore($q.start,"TimeMap timeMap,");
   //temp.pop();
   isSequence = false;
   }
   
   (COMMA! expression)*
   
   ;

expression
   : q=assignmentExpression // {System.out.println("expression "+Sq. text);}
   -> "(EXPR assignmentExpression )"
   
   ;

assignmentExpression returns [Boolean isSeq]
   : {$isSeq=false; doDont="";}
   
   q = conditionalExpression {$isSeq=$q.isSeq;} {
   
   // replace i=0 -> 0 to new sequenced.Integer(
   
   timeMap,0)
   
   if(!ident.equals(""))
   {
   
   String t = Sq.text;
   
   tokens.replace(Sq.start.getTokenIndex(),Sq.
   
   stop.getTokenIndex(),"new "+ident+("+
   
   timeMap,+Sq.text+)");
   
   ident = "";
   
   }
   
   }
   
   ( ( ASSIGN
   
   
   }
   
   opr = "ASSIGN"; // assign()
   
   if($q.isSeq)
tokens.replace($ASSIGN.getTokenIndex(),".assign(
"+timeMapStack.peek()+","+assign+",";)

$isSeq = true;
doDont="dontDo";

PLUS_ASSIGN^ {
opr = "PLUS_ASSIGN": // plusAssign()
if ($q.isSeq)
{
tokens.replace($PLUS_ASSIGN.getTokenIndex(),".
     plusAssign("+timeMapStack.peek()+","+
     plus_assign+",";)

$isSeq = true;
doDont="dontDo";

MINUS_ASSIGN^ {
opr = "MINUS_ASSIGN": // minusAssign()
if ($q.isSeq)
{
tokens.replace($MINUS_ASSIGN.getTokenIndex(),".
     minusAssign("+timeMapStack.peek()+","+
     minus_assign+",";)

$isSeq = true;
doDont="dontDo";

STAR_ASSIGN^ {
opr = "STAR_ASSIGN": // starAssign()
if ($q.isSeq)
{
tokens.replace($STAR_ASSIGN.getTokenIndex(),".
     starAssign("+timeMapStack.peek()+","+
     star_assign+",";)

$isSeq = true;
doDont="dontDo";

DIV_ASSIGN^
opr = "DIV_ASSIGN"; // divideAssign()
if ($q.isSeq)
{
    tokens.replace($DIV_ASSIGN.getTokenIndex(), ".
    divideAssign("+timeMapStack.peek()+","+
    div_assign+",")
    $isSeq = true;
doDont="dontDo";
}

AND_ASSIGN
|
opr = "AND_ASSIGN"; // andAssign()
if ($q.isSeq)
{
    tokens.replace($AND_ASSIGN.getTokenIndex(), ".
    andAssign("+timeMapStack.peek()+","+
    and_assign+",")
    $isSeq = true;
doDont="dontDo";
}

| OR_ASSIGN
|
opr = "OR_ASSIGN"; // orAssign()
if ($q.isSeq)
{
    tokens.replace($OR_ASSIGN.getTokenIndex(), ".
    orAssign("+timeMapStack.peek()+","+or_assign
    +",")
    $isSeq = true;
doDont="dontDo";
}

| XOR_ASSIGN
|
opr = "XOR_ASSIGN"; // xorAssign()
if ($q.isSeq)
{
    tokens.replace($XOR_ASSIGN.getTokenIndex(), ".
    xorAssign("+timeMapStack.peek()+","+
    xor_assign+",")
    $isSeq = true;
doDont="dontDo";
}
if (Sq.isSeq)
{
    tokens.replace($MOD_ASSIGN getTokenIndex(), ".modAssign("
               .timeMapStack.peek()+".mod_assign+",");
    $isSeq = true;
doDont="dontDo";
}
opr = "MOD_ASSIGN"; // modAssign()
if (Sq.isSeq)
{
    tokens.replace($SHIFT_LEFT_ASSIGN getTokenIndex(), ".shiftLeftAssign("
               .timeMapStack.peek()+".shift_left_assign+",");
    $isSeq = true;
doDont="dontDo";
}
opr = "SHIFT_LEFT_ASSIGN"; // shiftLeftAssign()
if (Sq.isSeq)
{
    tokens.replace($SHIFT_RIGHT_ASSIGN getTokenIndex(), ".shiftRightAssign("
               .timeMapStack.peek()+".shift_right_assign+",");
    $isSeq = true;
doDont="dontDo";
}
opr = "SHIFT_RIGHT_ASSIGN"; // shiftRightAssign()
if (Sq.isSeq)
{
    tokens.replace($BIT_SHIFT_RIGHT_ASSIGN getTokenIndex(), ".bitShiftRightAssign("
               .timeMapStack.peek()+".bit_shift_right_assign+",");
    $isSeq = true;
doDont="dontDo";
tokens.replace($BIT_SHIFT_RIGHT_ASSIGN,
  getTokenIndex()," . bitShiftRightAssign("+
  timeMapStack.peek()","+
  bit_shift_right_assign+",";

$isSeq = true;
doDont="dontDo";
}
)
}
z=assignmentExpression
{    
  if ($q.isSeq!=null && $z.isSeq!=null)
  {
    if ($z.isSeq && (!doDont.equals("dontDo")))
    {
      if (opr.equals("ASSIGN"))
      {
        String t = $q.text;
        tokens.replace($q.start.getTokenIndex(),$q.
        stop.getTokenIndex(),$z.text);
        tokens.replace($ASSIGN getTokenIndex()," .
        assign("+timeMapStack.peek()","+assign+"
        ");
        //tokens.insertAfter($ASSIGN," ");
        tokens.replace($z.start.getTokenIndex(),$z.
        stop.getTokenIndex(),t);
      }
      if (opr.equals("PLUS_ASSIGN"))
      {
        String t = $q.text;
        tokens.replace($q.start.getTokenIndex(),$q.
        stop.getTokenIndex(),$z.text);
        tokens.replace($PLUS_ASSIGN getTokenIndex(),
        ", .plusAssign("+timeMapStack.peek()"+","+
        plus_assign+";");
        //tokens.insertAfter($PLUS_ASSIGN," ");
        tokens.replace($z.start.getTokenIndex(),$z.
        stop.getTokenIndex(),t);
      }
      if (opr.equals("MINUS_ASSIGN"))
      {
        String t = $q.text;
        tokens.replace($q.start.getTokenIndex(),$q.
        stop.getTokenIndex(),$z.text);
      }   
  
  
}
tokens.replace($MINUS_ASSIGN, getTokenIndex(), 
"\".minusAssign("+timeMapStack.peek()+"," 
+minus_assign+"\"");
tokens.replace($z.start.getTokenIndex(),$z.
stop.getTokenIndex(),t);

if (opr.equals("STAR_ASSIGN"))
{
    String t = $q.text;
    tokens.replace($q.start.getTokenIndex(),$q.
stop.getTokenIndex(),$z.text);
    tokens.replace($STAR_ASSIGN, getTokenIndex(), 
"\".starAssign("+timeMapStack.peek()+"," 
+star_assign+"\"");
    tokens.replace($z.start.getTokenIndex(),$z.
stop.getTokenIndex(),t);
}

if (opr.equals("DIV_ASSIGN"))
{
    String t = $q.text;
    tokens.replace($q.start.getTokenIndex(),$q.
stop.getTokenIndex(),$z.text);
    tokens.replace($DIV_ASSIGN, getTokenIndex(), 
"\".divAssign("+timeMapStack.peek()+"," 
+div_assign+"\"");
    tokens.replace($z.start.getTokenIndex(),$z.
stop.getTokenIndex(),t);
}

if (opr.equals("AND_ASSIGN"))
{
    String t = $q.text;
    tokens.replace($q.start.getTokenIndex(),$q.
stop.getTokenIndex(),$z.text);
    tokens.replace($AND_ASSIGN, getTokenIndex(), 
"\".andAssign("+timeMapStack.peek()+"," 
+and_assign+"\"");
    tokens.replace($z.start.getTokenIndex(),$z.
stop getTokenIndex(),t);
}

if (opr.equals("OR_ASSIGN"))
{
    String t = $q.text;
    tokens.replace($q.start.getTokenIndex(),$q.
stop.getTokenIndex(),$z.text);
tokens.replace($OR_ASSIGN.getTokenIndex(), ",
orAssign("+timeMapStack.peek()+","+ or_assign+");
tokens.replace($z.start.getTokenIndex(),$z.
stop.getTokenIndex(),t);
}
if(opr.equals("XOR_ASSIGN"))
{
  String t = $q.text;
tokens.replace($q.start.getTokenIndex(),$q.
stop.getTokenIndex(),$z.text);
tokens.replace($XOR_ASSIGN.getTokenIndex(),
.xorAssign("+timeMapStack.peek()+","+
xor_assign+");
tokens.replace($z.start.getTokenIndex(),$z.
stop.getTokenIndex(),t);
}
if(opr.equals("MOD_ASSIGN"))
{
  String t = $q.text;
tokens.replace($q.start.getTokenIndex(),$q.
stop.getTokenIndex(),$z.text);
tokens.replace($MOD_ASSIGN.getTokenIndex(),"+
.modAssign("+timeMapStack.peek()+","+
mod_assign+");
tokens.replace($z.start.getTokenIndex(),$z.
stop.getTokenIndex(),t);
}
if(opr.equals("SHIFT_LEFT_ASSIGN"))
{
  String t = $q.text;
tokens.replace($q.start.getTokenIndex(),$q.
stop.getTokenIndex(),$z.text);
tokens.replace($SHIFT_LEFT_ASSIGN.
getTokenIndex(),".shiftLeftAssign("+
timeMapStack.peek()+","+
shift_left_assign+");
tokens.replace($z.start.getTokenIndex(),$z.
stop.getTokenIndex(),t);
}
if(opr.equals("SHIFT_RIGHT_ASSIGN"))
{
  String t = $q.text;
tokens.replace($q.start.getTokenIndex(),$q.stop.getTokenIndex(),$z.text);
tokens.replace($SHIFT_RIGHT_ASSIGN getTokenIndex(),".shiftRightAssign("
+timeMapStack.peek()","+.shift_right_assign+");
tokens.replace($z.start.getTokenIndex(),$z.stop.getTokenIndex(),t);
}
if(opr.equals("BIT_SHIFT_RIGHT_ASSIGN"))
{
    String t = $q.text;
tokens.replace($q.start.getTokenIndex(),$q.stop.getTokenIndex(),$z.text);
tokens.replace($BIT_SHIFT_RIGHT_ASSIGN
    getTokenIndex(),".bitShiftRightAssign("+timeMapStack.peek()","+.bit_shift_right_assign+");
tokens.replace($z.start.getTokenIndex(),$z.stop.getTokenIndex(),t);
}

    $isSeq = true;
}

if($z.isSeq || $q.isSeq)
tokens.insertAfter($z.stop.getTokenIndex(),"");
}

}

conditionalExpression returns [Boolean isSeq]
{
    $isSeq=false; doDont=""; e=logicalOrExpression {$isSeq=$e.isSeq
    ;}(QUESTION^ e1=assignmentExpression COLON! e2=
    conditionalExpression )?
    {
        if($e.isSeq!=null & & $e1.isSeq!=null & & $e2.isSeq!=null)
        {
            if($e.isSeq==true)
            {
                tokens.replace($QUESTION.getTokenIndex(),"");
tokens.replace($COLON.getTokenIndex(),"");
String t = $e.text+".conditional("+timeMapStack.peek
()","+$e1.text","+$e2.text")";
tokens.replace($e.start.getTokenIndex(), $e.stop.getTokenIndex(), $e.text + "\"Se\".text + "\"Se2.text +\")
					getTokenIndex(), $e2.start.getTokenIndex(), $e2.stop.getTokenIndex(), $e2.text + "\"Se2\".text + "\");
				
tokens.replace($e.start.getTokenIndex(), $e.stop.getTokenIndex(), $e.text + "\"Se\".text + "\"Se1.text +\")
					getTokenIndex(), $e1.start.getTokenIndex(), $e1.stop.getTokenIndex(), $e1.text + "\"Se1.text + "\";}

};

// logicalOrExpression returns [Boolean isSeq]
logicalOrExpression returns [Boolean isSeq]
: {isSeq=false; doDont="";} q=logicalAndExpression {isSeq=$q.isSeq;}


opr = "LOGICAL_OR"; // logicalOr()
if ($q.isSeq !=null && $q.isSeq)
{

tokens.replace(SLOGICAL_OR.getTokenIndex(), ","

logicalOr("+timeMapStack.peek()+","+
logical_or+",");


isSeq = true;

doDont="dontDo";

}

}

z=logicalAndExpression

{

if ($q.isSeq !=null && $z.isSeq !=null)
{

if ($z.isSeq && (!doDont.equals("dontDo")))
{

if (opr.equals("LOGICAL_OR"))
{

String t =$q.text;

tokens.replace($q.start.getTokenIndex(), $q.stop.getTokenIndex(), $q.text);

tokens.replace(SLOGICAL_OR.getTokenIndex(), "

.logicalOr("+timeMapStack.peek()+","+
.logical_or+",");

tokens.replace($z.start.getTokenIndex(), $z.stop.getTokenIndex(), $z.text);


isSeq = true;

}


if ($z.isSeq || $q.isSeq)


logicalAndExpression returns [Boolean isSeq]

```java
isSeq=false; doDont="";

q=inclusiveOrExpression {isSeq= Sq.

isSeq;} (LOGICAL_AND

opr = "LOGICAL_AND"; // logicalAnd()

if ($q.isSeq)

{ tokens.replace ($LOGICAL_AND.getTokenIndex () , ".

logicalAnd("+timeMapStack.peek() +" , "+

logical_and+" , ");

isSeq = true;

doDont="dontDo";

}

z=inclusiveOrExpression

{ if ($q.isSeq!=null && $z.isSeq!=null)

{ if ($z.isSeq && (!doDont.equals("dontDo")))

{ if (opr.equals("LOGICAL_AND"))

{}

String t = $q.text;

tokens.replace ($q.start.getTokenIndex () , $q.

stop.getTokenIndex () , $z.text);

tokens.replace ($LOGICAL_AND.getTokenIndex () ,

".logicalAnd("+timeMapStack.peek()+" , "+

logical_and+" , ");

tokens.replace ($z.start.getTokenIndex () , $z.

stop.getTokenIndex () , t);

$z.isSeq = true;

}

$z.isSeq = true;

}

if ($z.isSeq || $q.isSeq)

tokens.insertAfter ($z.stop.getTokenIndex () , "")

};

```
inclusiveOrExpression returns [Boolean isSeq]

    : {isSeq=false; doDont="";} q=exclusiveOrExpression {isSeq= Sq. isSeq;} (OR"

    
    opr = "OR": //. or()

    if (Sq.isSeq)

        
        tokens.replace(SOR.getTokenIndex(),". or("+ timetablestack.peek()","+.or+","";

        isSeq = true;

        doDont="dontDo";

    )

    }

    z=exclusiveOrExpression

    {

        if (Sq.isSeq!=null && $z.isSeq!=null)

            {

                if ($z.isSeq && (!doDont.equals("dontDo")))

                    {

                        if (opr.equals("OR"))

                            {

                                String t = Sq.text;

                                tokens.replace(Sq.start.getTokenIndex(),Sq.

                                stop.getTokenIndex(),$z.text);

                                tokens.replace($OR.getTokenIndex(),". or("+

                                timetablestack.peek()","+.or+","";

                                tokens.replace($z.start.getTokenIndex(),$z.

                                stop.getTokenIndex(),t);

                                isSeq = true;

                            }

            }

        isSeq = true;

        }

    if ($z.isSeq || Sq.isSeq)

        tokens.insertAfter($z.stop.getTokenIndex(),"")

    }

)}

exclusiveOrExpression returns [Boolean isSeq]
: \{\text{isSeq=false; doDont="\";}} q=andExpression \{\text{isSeq=Sq.isSeq;}} (\nO\text{R}^\ast \n)\n    \{\n        \text{opr = "XOR"}; // xor()\n    \}
    \n    \text{if(Sq.isSeq)}\n    \{\n        \text{tokens.replace(SXOR.getTokenIndex(),"xor("+timeMapStack.peek()+","+xor+"","));}\n        \text{isSeq = true;}\n        \text{doDont="dontDo"];}\n    \}
    \n    \text{z=andExpression}\n    \{
        //System.out.println(S$.isSeq+""+$z$.isSeq);\n        \text{if(S$.isSeq!=null && $z$.isSeq!=null)}\n        \{\n            \text{if($z$.isSeq && (!doDont.equals("dontDo")))}\n            \{\n                \text{if(opr.equals("XOR"))}\n                \{\n                    \text{String t = Sq.text;}\n                    \text{tokens.replace(Sq.start.getTokenIndex(),Sq.}
                        \text{stop.getTokenIndex(),$z$.text);}\n                    \text{tokens.replace(SXOR.getTokenIndex(),"xor("+timeMapStack.peek()+","+xor+"","));}\n                    \text{tokens.replace($z$.start.getTokenIndex(),$z$.}
                        \text{stop.getTokenIndex(),t);}\n                    \text{isSeq = true;}\n                \}\n            \}\n            \text{isSeq = true;}\n        \}\n        \text{if($z$.isSeq || $z$.isSeq)}\n        \text{tokens.insertAfter($z$.stop.getTokenIndex(),""});}\n    \}\n    \}
    \text{andExpression returns [Boolean isSeq]}
```java
789 : {isSeq=false; doDont="";} q=equalityExpression{isSeq=q.isSeq
800 }
801 {
802     if(q.isSeq)
803         {
804             opr = "AND"; // .and()
805             tokens.replace($AND.getTokenIndex(),".and("+timeMapStack.peek()+","+and+",");
806             isSeq = true;
807             doDont="dontDo";
808     }
809 }
810 };
811 equalityExpression z=equalityExpression
812 {
813     $isSeq = Sz.isSeq;
814     if($q.isSeq!=null &&$z.isSeq!=null)
815         {
816             if($z.isSeq && (!doDont.equals("dontDo")))
817                 {
818                     if(opr.equals("AND"))
819                         {
820                             String t =$q.text;
821                             tokens.replace($q.start.getTokenIndex(),$q.
822                             stop.getTokenIndex(),$z.text);
823                             tokens.replace($AND.getTokenIndex(),".and("+timeMapStack.peek()+","+and+",");
824                             tokens.replace($z.start.getTokenIndex(),$z.
825                             stop.getTokenIndex(),t);
826                             $isSeq = true;
827                     }
828                 }
829             $isSeq = true;
830         }
831     if($z.isSeq ||$q.isSeq)
832         tokens.insertBefore($z.stop.getTokenIndex(),""");
833     }
834 *
835 equalityExpression returns [Boolean isSeq]
836 : {isSeq=false;} q=instanceOfExpression{isSeq=q.isSeq;
837     (doDont="";)} EQUAL
```
\{ 
  opr = "EQUAL" ; // equal()

  if (Sq.isSeq)
  
    \{
      tokens.replace(SEQUAL, getTokenIndex(), "equal(\"+timeMapStack.peek()\" +\"+equal\" +\")
      $isSeq = true;
      doDont="dontDo";
    
    }

equal

  \{ 
  opr = "NOT_EQUAL" ; // notEqual()

  if (Sq.isSeq)
  
    \{
      tokens.replace($NOT_EQUAL, getTokenIndex(), "notEqual(\"+timeMapStack.peek()\" +\"+not_equal +\" +\")
      $isSeq = true;
      doDont="dontDo";
    
    }

  NOT_EQUAL

  } 

  z=instanceOfExpression

  if (Sq.isSeq != null && Sz.isSeq != null)
  
    \{
      if ($z.isSeq && (!doDont.equals("dontDo")))
      
        \{
          if (opr.equals("EQUAL"))
          
            \{
              String t = Sq.text;
              tokens.replace(Sq.start, getTokenIndex(), Sq.stop, getTokenIndex(), Sz.text);
              tokens.replace(SEQUAL, getTokenIndex(), ".equal(\"+timeMapStack.peek()\" +\"+equal\" +\")
              tokens.replace($z, start, getTokenIndex(), $z.stop, getTokenIndex(), t);
            
            $isSeq = true;
          
        
    
    

    

\}
if (opr.equals("NOT_EQUAL"))
{
    String t = $q.text;
    tokens.replace($q.start.getTokenIndex(),$q.stop.getTokenIndex(),t);
    tokens.replace($NOT_EQUAL.getTokenIndex(),"notEqual("+timeMapStack.peek()+","+not_equal+");
    tokens.replace($z.start.getTokenIndex(),$z.stop.getTokenIndex(),t);
}

$isSeq = true;
if ($z.isSeq || $q.isSeq)
{
tokens.insertAfter($z.stop.getTokenIndex(),");
}

instanceOfExpression returns [Boolean isSeq]
: {isSeq=false;} q=relationalExpression {isSeq=$q.isSeq;} (INSTANCEOFˆ z=type)?
{
    if ($z.text!=null && $q.isSeq)
    tokens.replace($z.start.getTokenIndex(),$z.stop.getTokenIndex(),"sequenced."+$z.text);
}

/\RELATIONAL EXPRESSION FOR for loop
/\ Relational expression for If else statement

relationalExpression returns [Boolean isSeq]
: {isSeq=false;doDont="";} q=shiftExpression {isSeq=$q.isSeq;} (LESS_OR_EQUALˆ
{
    opr = "LESS_OR_EQUAL"; //lessOrEqual()
}
if ($q.isSeq)
{
tokens.replace($LESS_OR_EQUAL.getTokenIndex(), "
  lessOrEqual("+timeMapStack.peek()+","+
  less_or_equal+".");

$isSeq = true;
doDont="dontDo";
}

| GREATER_OR_EQUAL
|
| GREATER_OR_EQUAL* |
opr = "GREATER_OR_EQUAL"; // greaterOrEqual()

if ($q.isSeq)
{
  tokens.replace($GREATER_OR_EQUAL.getTokenIndex(), 
  ", greedyOrEqual("+timeMapStack.peek()+","+
  greater_or_equal+".");

  $isSeq = true;
doDont="dontDo";
}

| LESS_THAN^ |
|
| LESS_THAN |
opr = "LESS_THAN"; // lessThan()

if ($q.isSeq)
{
  tokens.replace($LESS_THAN.getTokenIndex(), 
  ", lessThan("+timeMapStack.peek()+","+less_than 
  +".");

  $isSeq = true;
doDont="dontDo";
}

| GREATER_THAN^ |
|
| GREATER_THAN |
opr = "GREATER_THAN"; // greaterThan()

if ($q.isSeq!=null)
{
  if ($q.isSeq)
tokens.replace(SGREATER_THAN, getTokenIndex(), "
greaterThan("+timeMapStack.peek()+","+
greater_than+",");
$IsSeq = true;
doDont="dontDo";
}
}
}
}
}
)

z=shiftExpression
{
if ($q.isSeq!=null && $z.isSeq!=null)
{
  if ($z.isSeq && (!doDont.equals("dontDo")))
  {
    if (opr.equals("LESS_OR_EQUAL"))
    {
      String t = $q.text;
tokens.replace($q.start.getTokenIndex(),$q.
stop.getTokenIndex(),$z.text);
tokens.replace(SLESS_OR_EQUAL,getTokenIndex
() ,".lessOrEqual("+timeMapStack.peek()+"  ","+less_or_equal+",");
tokens.replace($z.start.getTokenIndex(),$z.
stop.getTokenIndex(),t);
    }
    if (opr.equals("GREATER_OR_EQUAL"))
    {
      String t = $q.text;
tokens.replace($q.start.getTokenIndex(),$q.
stop.getTokenIndex(),$z.text);
tokens.replace(SGREATER_OR_EQUAL,
getTokenIndex(),".greaterOrEqual("+timeMapStack.peek()+"","+greater_or_equal "+","); 
tokens.replace($z.start.getTokenIndex(),$z.
stop.getTokenIndex(),t);
    }
    if (opr.equals("LESS_THAN"))
    {
      String t = $q.text;
tokens.replace($q.start.getTokenIndex(),$q.
stop.getTokenIndex(),$z.text);
    }
}
tokens.replace($LESS_THAN.getTokenIndex(), "lessThan(" + timeMapStack.peek() + "+", "less_than+");

tokens.replace($z.start.getTokenIndex(), $z.stop.getTokenIndex(), t);

if (opr.equals("GREATER_THAN")) {
    String t = Sq.text;
    tokens.replace($q.start.getTokenIndex(), $q.stop.getTokenIndex(), $z.text);
    tokens.replace($GREATER_THAN.getTokenIndex(), ",greaterThan(" + timeMapStack.peek() + "+", "+greater_than+",");
    tokens.replace($z.start.getTokenIndex(), $z.stop.getTokenIndex(), t);
    $isSeq = true;
}

if ($z.isSeq || $q.isSeq)
    tokens.insertAfter($z.stop.getTokenIndex(), "")
if ($z.isSeq)
    tokens.replace($BIT_SHIFT_RIGHT.getTokenIndex(), ",bitShiftRight(" + timeMapStack.peek() + "+", "+bit_shift_right+",");
    $isSeq = true;
    doDont="dontDo";
}

shiftExpression returns [Boolean isSeq]
: { $isSeq=false; doDont=""; } q=additiveExpression { $isSeq=$q.isSeq ;
  ( ( BIT_SHIFT_RIGHT^ //bitShiftRight() 
  
  opr = "BIT_SHIFT_RIGHT";
  if ($q.isSeq != null)
  { if ($q.isSeq)
    
    tokens.replace($BIT_SHIFT_RIGHT.getTokenIndex(), ",bitShiftRight(" + timeMapStack.peek() + "+", "+bit_shift_right+",");
    $isSeq = true;
    doDont="dontDo";
  }
  
}
SHIFT_RIGHT^  // shiftRight()
{
    opr = "SHIFT_RIGHT";
    if ($q.isSeq!=null)
    {
        if ($q.isSeq)
        {
            tokens.replace($SHIFT_RIGHT.getTokenIndex(), ".
                shiftRight("+timeMapStack.peek()+","+
                shift_right+",");
            isSeq = true;
            doDont="dontDo";
        }
    }
}

SHIFT_LEFT^  // shiftLeft()
{
    opr = "SHIFT_LEFT";
    if ($q.isSeq!=null)
    {
        if ($q.isSeq)
        {
            tokens.replace($SHIFT_LEFT.getTokenIndex(), ".
                shiftLeft("+timeMapStack.peek()+","+
                shift_left+",");
            isSeq = true;
            doDont="dontDo";
        }
    }
}
z=additiveExpression
{
    if ($q.isSeq!=null && $z.isSeq!=null)
    {
        if ($z.isSeq && (!doDont.equals("dontDo")))
        {
            if (opr.equals("BIT_SHIFT_RIGHT"))
            {
                String t = $q.text;
                tokens.replace($q.start.getTokenIndex(),$q.
                    stop.getTokenIndex(),$z.text);
                tokens.replace($BIT_SHIFT_RIGHT.
                    getTokenIndex(), ".bitShiftRight("+
                    }
```java
if (opr.equals("SHIFT_RIGHT"))
{
    String t = sq.text;
    tokens.replace(sz.start.getTokenIndex(), sz.
        stop.getTokenIndex(), t);
    tokens.replace($SHIFT_RIGHT.gettokenId(),
        " shiftRight("+timeMapStack.peek()+","+shift_right+");
    tokens.replace(sz.start.getTokenIndex(), sz.
        stop.getTokenIndex(), t);
}

if (opr.equals("SHIFT_LEFT"))
{
    String t = sq.text;
    tokens.replace(sz.start.getTokenIndex(), sz.
        stop.getTokenIndex(), t);
    tokens.replace($SHIFT_LEFT.gettokenId(),
        " shiftLeft("+timeMapStack.peek()+","+shift_left+");
    tokens.replace(sz.start.getTokenIndex(), sz.
        stop.getTokenIndex(), t);
}

    $isSeq = true;
}

if ($z.isSeq || sq.isSeq)
    tokens.insertAfter(sz.stop.getTokenIndex(), "");
    
    }

additiveExpression returns [Boolean isSeq]
    : { $isSeq=false; doDont=""; doIt=false; } q=multiplicativeExpression
        { $isSeq=eq.isSeq; }
    ( PLUS
    { 
        opr = "PLUS";
        if ($eq.isSeq!=null)
        
        
```
```java
if ($q.isSeq)
{
    tokens.replace($PLUS.getTokenIndex(),".add(" +
    timeMapStack.peek() +" ," +plus +" , ");
    $isSeq = true;
    doDont="dontDo";
}
opr = "MINUS";
if ($q.isSeq)
{
    tokens.replace($MINUS.getTokenIndex(),".subtract(" +
    timeMapStack.peek() +" ," +minus +" , ");
    $isSeq = true;
    doDont="dontDo";
}
opr = "MINUS";
if ($q.isSeq)
{
    tokens.replace($MINUS.getTokenIndex(),".subtract(" +
    timeMapStack.peek() +" ," +minus +" , ");
    $isSeq = true;
    doDont="dontDo";
}
opr = "MINUS";
if ($q.isSeq)
{
    if ($q.text != null && $z.text != null)
    {
        if ($z.isSeq && (!doDont.equals("dontDo")))
        {
            doIt = true;
            if (opr.equals("PLUS"))
            {
                String t = $q.text;
                tokens.replace($q.start.getTokenIndex(),$q.
                stop.getTokenIndex(),$z.text);
                tokens.replace($PLUS.getTokenIndex(),".add(" +
                timeMapStack.peek() +" ," +plus +" , ");
                tokens.replace($z.start.getTokenIndex(),$z.
                stop.getTokenIndex(),t);
                $isSeq = true;
            }
            if (opr.equals("MINUS"))
            {
                String t = $q.text;
                tokens.replace($q.start.getTokenIndex(),$q.
                stop.getTokenIndex(),$z.text);
            }
        }
    }
```
tokens.replace($MINUS . getTokenIndex () , ".
subtract ("+timeMapStack . peek () +" , "+minus +" , ");
tokens.replace ($z . start . getTokenIndex () , $z.
stop . getTokenIndex () , t );
}

sisSeq = true;
}

tokens.insertBefore ($z . stop . getTokenIndex () , ")");
}

}

mul
ti
pli
cativeExpression returns [ Boolean isSeq ]

: { sisSeq=false ; doDont=""

) q=unaryExpression { sisSeq=Q . isSeq ;}

( 

STAR

{
opr = "STAR" ;

if ( Q . isSeq )

{
tokens.replace ($STAR . getTokenIndex () , ".multiply ("+timeMapStack . peek () +" , "+star+" , ");

sisSeq = true ;
doDont="dontDo" ;

}

}

|  DIV

{
opr = "DIV" ;

if ( Q . isSeq )

{
tokens.replace ($DIV . getTokenIndex () , ".divide ("+timeMapStack . peek () +" , "+div+" , ");

sisSeq = true ;
doDont="dontDo" ;

}

}

|  MOD

{
opr = "MOD" ;
if ($q.isSeq)
{
    tokens.replace($MOD.getTokenIndex(), "modulus("+timeMapStack.peek()+","+mod+",")
    $isSeq = true;
    doDont="dontDo";
}

z=unaryExpression
{
    if ($q.isSeq!=null && $z.isSeq!=null)
    {
        if ($z.isSeq && (!doDont.equals("dontDo"))
        {
            if (opr.equals("STAR"))
            {
                String t = $q.text;
                tokens.replace($q.start.getTokenIndex(), $q.
                    stop.getTokenIndex(), $q.text);
                tokens.replace(SSTAR.getTokenIndex(), ".multiply("+timeMapStack.peek()+","+star+
                    ",")
                tokens.replace($z.start.getTokenIndex(), $z.
                    stop.getTokenIndex(), t);
            }
            if (opr.equals("DIV"))
            {
                String t = $q.text;
                tokens.replace($q.start.getTokenIndex(), $q.
                    stop.getTokenIndex(), $q.text);
                tokens.replace(SDIV.getTokenIndex(), ".divide 
                    ("+timeMapStack.peek()+"+"+div+",")
                tokens.replace($z.start.getTokenIndex(), $z.
                    stop.getTokenIndex(), t);
            }
        }
    }
}
if (opr.equals("MOD"))
{
    String t = $q.text;
    tokens.replace($q.start.getTokenIndex(), $q.
        stop.getTokenIndex(), $q.text);
    tokens.replace(SMOD.getTokenIndex(), ".modulus("+timeMapStack.peek()+"+"+mod+",")

tokens.replace($z.start.getTokenIndex(),$z.stop.getTokenIndex(),t);

$z.start = $z.stop = t;

if($z.isSeq || $q.isSeq)
tokens.insertAfter($z.stop.getTokenIndex(),""");

$i isSeq = true;

if($z.isSeq || $q.isSeq)
tokens.insertAfter($z.start.getTokenIndex(),""");

$i isSeq = true;

*/

unaryExpression returns [Boolean isSeq]
{: $isSeq=false; } PLUS p=unaryExpression
{ if($p.isSeq)

  tokens.insertAfter($PLUS.start.getTokenIndex()+1,".add( "+ timeMapStack.peek()" , "+plus+")");
tokens.replace($PLUS.start.getTokenIndex(),""");

  $isSeq=true;

  }

  $isSeq=true;

  } /*(UNARYPLUS[$PLUS, "UNARYPLUS"] unaryExpression)

  MINUS m=unaryExpression
  { if($m.isSeq)

      tokens.insertAfter($MINUS.start.getTokenIndex()+1,".subtract( "+ timeMapStack.peek()" , "+minus+")");
tokens.replace($MINUS.start.getTokenIndex(),""");

      $isSeq=true;

      }

  /*(UNARYMINUS[$MINUS, "UNARYMINUS"] unaryExpression)

  // prefix operations

  INC i=postfixedExpression
  { if($i.isSeq)

      tokens.insertAfter($INC.start.getTokenIndex()+1,".increment( "+ timeMapStack.peek()" , "+inc+")");
tokens.replace($INC.start.getTokenIndex(),""");

      }
\$isSeq = true;
}
}

\$isSeq = (PRE_INC[$INC, "PRE_INC"] postfixedExpression)

// prefix operations

DEC d=postfixedExpression

if ($d.isSeq)
{

tokens.insertAfter($DEC.getTokenIndex()+1, ". decrement("+
timeMapStack.peek() + ", "+dec+)" );
tokens.replace($DEC.getTokenIndex(), "." );
$\$isSeq = true;
}

$\$isSeq = (PRE_DEC[$DEC, "PRE_DEC"] postfixedExpression)

q=unaryExpressionNotPlusMinus

$\$isSeq = $q.isSeq;

unaryExpressionNotPlusMinus returns [Boolean isSeq]

: {\$isSeq=false} \ NOT n=unaryExpression

if ($n.isSeq)
{

tokens.insertAfter($NOT getTokenIndex()+1, ". not("+
timeMapStack.peek() + ", "+not+)" );
tokens.replace($NOT getTokenIndex(), "." );
$\$isSeq = true;

$\$isSeq = (NOT unaryExpression)

LOGICAL_NOT ln=unaryExpression

if ($ln.isSeq)
{

tokens.insertAfter($LOGICAL_NOT getTokenIndex()+1, ". logicalNot("+
timeMapStack.peek() + ", "+logical_not+)" );
tokens.replace($LOGICAL_NOT getTokenIndex(), "." );
$\$isSeq = true;

postfixedExpression returns [Boolean isSeq]
   // At first resolve the primary expression ...
   : ( {$isSeq=false;} q=primaryExpression
   { $isSeq = $q.isSeq;
   } $isSeq = false;
   ) -> primaryExpression
   ;
   
   // ... and than the optional things that may follow a primary
   // expression 0 or more times.
   ( outerDot=DOT
   ( ( genericTypeArgumentListSimplified? // Note: generic
       type arguments are only valid for method calls, i.e. if
       there
       // is an argument
       list.
       IDENT
       -> ^(DOT
       $postfixedExpression IDENT)
   )
   ( arguments
   /*{
   if (isSequence)
   { System.out.println(" abc ");
    methodCall = true;
    isSequence = false;
   }
   }*/

   -> ^(METHOD_CALL
       $postfixedExpression
genericTypeArgumentListSimplified
? arguments)

2310 )?
2311 | THIS
2312 | $postfixedExpression THIS
2313 | $Super=SUPER arguments
2314 | "(SUPER
2315 | CONSTRUCTOR
2316 | CALL[ $Super, "SUPER
2317 | CONSTRUCTOR
2318 | CALL"]
2319 | $postfixedExpression arguments)
2320 | ($ SUPER innerDot=DOT IDENT
2321 | $outerDot $postfixedExpression SUPER) IDENT)
2322 | )
2323 | ( arguments
2324 | /*{
2325 | if (isSequence)
2326 | {
2327 | System.out.println(" abc ");
2328 | methodCall = true;
2329 | isSequence = false;
2330 | }
2331 | }*/
2332 | }?
2333 | innerNewExpression
2334 | $postfixedExpression innerNewExpression
2335 | )
2336 | LBRACK expression RBRACK
2337 | ARRAY_ELEMENT_ACCESS
2338 | $postfixedExpression expression
2339 | )
2340 | */
2341 | // At the end there may follow a post increment/decrement.
2342 | // Post fix increment
2343 | ( INC
2344 | {
2345 | if ($q.isSeq)
2346 | {
2347 | tokens.replace($INC,".increment("+timeMapStack.peek
2348 | ()+", "+inc+")");
2349 | }
2350 | }
2351 | }?
2352 | (POST_INC[$INC, "POST_INC"] $postfixedExpression)
2353 | |
2354 | DEC
2355 | // Post fix decrement
2356 | {
2357 | if ($q.isSeq)
```java
 93

 2344  
 2345  
 2346  
 2347  
 2348  
 2349  
 2350  
 2351  
 2352  
 2353  
 2354  
 2355  
 2356  
 2357  
 2358  
 2359  
 2360  
 2361  
 2362  
 2363  
 2364  
 2365  
 2366  
 2367  
 2368  
 2369  
 2370  
 2371  
 2372  
 2373  
 2374  
 2375  
 2376  
 2377  
 2378  
 2379
```
( arguments -> "(
    THIS_CONSTRUCTOR_CALL[$THIS, "THIS_CONSTRUCTOR_CALL"]
    arguments)
)
| SUPER arguments -> "(
    SUPER_CONSTRUCTOR_CALL[$SUPER, "SUPER_CONSTRUCTOR_CALL"]
    arguments)
| ( SUPER DOT IDENT
    )
| ( arguments
    )
/*{
    if(isSequence)
    {
        System.out.println(" abc ");
        methodCall = true;
        isSequence = false;
    }
} */
-> "(METHOD_CALL "(DOT SUPER IDENT) arguments)
| -> "(DOT SUPER IDENT)
| ( primitiveType
    )
| primitiveType
| ( arrayDeclarator
    )
| arrayDeclarator $primaryExpression) *
| DOT CLASS
| $primaryExpression CLASS
| VOID DOT CLASS
| -> "(DOT VOID CLASS)
| ;

qualifiedIdentExpression returns [Boolean isSeq]
// The qualified identifier itself is the starting point for this rule.
: ( {isSeq = false;} q=qualifiedIdentifier { boolean t = st.probe($q.text);
    //System.out.println(" sequenced check "+$q.text + "+t");
    if(t)
    {
    }
$isSeq = true;
}

} -> qualifiedIdentifier

) // And now comes the stuff that may follow the qualified identifier.

( arrayDeclarator -> ^( arrayDeclarator $qualifiedIdentExpression)
) +

( DOT CLASS -> ^(DOT $qualifiedIdentExpression CLASS)
)

| {}

if(isSequence)
{
    System.out.println(" abc ");
    methodCall = true;
    isSequence = false;
}

}) arguments

-> ^(METHOD_CALL qualifiedIdentifier arguments)

| outerDot=DOT

( CLASS -> ^(DOT qualifiedIdentifier CLASS)
)

| genericTypeArgumentListSimplified

( Super=SUPER arguments -> ^( SUPER,CONSTRUCTOR_CALL[$Super , " SUPER,CONSTRUCTOR_CALL"] qualifiedIdentifier
genericTypeArgumentListSimplified arguments)
)

| SUPER innerDot=DOT IDENT arguments

| IDENT arguments

/* if(isSequence)
{
 System.out.println(" abc ");
 methodCall = true;
 isSequence = false;
}}/*
2446  
2447 )
2448  
2449  
2450  
2451 ) {isSequence = false:}
2452
2453 ;
2454
2455 newExpression
2456  : NEW
2457  ( primitiveType newArrayConstruction // new static array
2458  of primitive type elements
2459  
2460  ( genericTypeArgumentListSimplified?
2461  
2462  | arguments classBody? // new object type
2463  via constructor invocation
2464  
2465
2466
2467 innerNewExpression // something like 'InnerType innerType = outer.new
2468   InnerType();'
2469  : NEW genericTypeArgumentListSimplified? IDENT arguments classBody

(CLASS_CONSTRUCTOR_CALL[$NEW, "STATIC_ARRAY_CREATOR"]
genericTypeArgumentListSimplified? IDENT arguments classBody ?)

; 

newArrayConstruction
arrayDeclaratorList arrayInitializer
| LBRACK! expression RBRACK! (LBRACK! expression RBRACK!)*
arrayDeclaratorList?
;

arguments
LPAREN
{
If (methodCall)
{
System.out.println("method call");
tokens.insertAfter($LPAREN, timeMapStack.peek() + ", ");
methodCall = false;
}
}
expressionList? RPAREN

-> "(ARGUMENT_LIST[$LPAREN, "ARGUMENT_LIST"] expressionList?)"
;

literal returns [Boolean isSeq]

{$isSeq=false;} q= HEX_LITERAL
{
boolean t = st.probe($q.text);
if (t)
$isSeq = true;
}

| q= OCTAL_LITERAL
{
boolean t = st.probe($q.text);
if (t)
$isSeq = true;
}

| q= DECIMAL_LITERAL
{
boolean t = st.probe($q.text);
if (t)
$isSeq = true;
}

| q= FLOATING_POINT_LITERAL
{
boolean t = st.probe($q.text);
if (t)
$isSeq = true;

}  
| q= CHARACTER_LITERAL  
  
  boolean t = st.probe($q.text);
  
  if(t)
  
  $isSeq = true;

}  
| q= STRING_LITERAL 
  
  boolean t = st.probe($q.text);
  
  if(t)
  
  $isSeq = true;

}  
| q= TRUE 
  
  boolean t = st.probe($q.text);
  
  if(t)
  
  $isSeq = true;

}  
| q= FALSE 
  
  boolean t = st.probe($q.text);
  
  if(t)
  
  $isSeq = true;

}  
| q= NULL 
  
  boolean t = st.probe($q.text);
  
  if(t)
  
  $isSeq = true;

}  

// LEXER

HEX_LITERAL : '0' ('x' | 'X') HEX_DIGIT+ INTEGER_TYPE_SUFFIX? ;

DECIMAL_LITERAL : ('0' | '1'..'9' '0'..'9') INTEGER_TYPE_SUFFIX? ;

OCTAL_LITERAL : '0' ('0'..'7')+ INTEGER_TYPE_SUFFIX? ;

fragment

HEX_DIGIT : ('0'..'9'|'a'..'f'|'A'..'F') ;

fragment

INTEGER_TYPE_SUFFIX : ('l'|'L') ;

FLOATING_POINT_LITERAL
\[\begin{align*}
&\text{IDENT} : \text{JAVA_ID_START} (\text{JAVA_ID_PART})^* \\
&\text{EXponent} : (\text{'e'} | \text{'E'}) (\text{'+'} | \text{'−'})? (\text{'0'..'9'})^+ \\
&\text{FLOAT_TYPE_SUFFIX} : (\text{'f'} | \text{'F'} | \text{'d'} | \text{'D'}) \\
&\text{ESCAPE_SEQUENCE} : \text{\textbackslash''} (\text{ESCAPE_SEQUENCE} | (\text{\textbackslash''} | \text{'\'})) \text{\textbackslash''} \\
&\text{OCTAL_ESCAPE} : \text{\textbackslash'} (\text{'0'..'3'}) (\text{'0'..'7'}) (\text{'0'..'7'}) \\
&\text{UNICODE_ESCAPE} : \text{\textbackslash'' u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT} \\
\end{align*}\]
<table>
<thead>
<tr>
<th>line_number</th>
<th>line_content</th>
</tr>
</thead>
<tbody>
<tr>
<td>2599</td>
<td>;</td>
</tr>
<tr>
<td>2600</td>
<td>fragment</td>
</tr>
<tr>
<td>2601</td>
<td>JAVA_ID_START</td>
</tr>
<tr>
<td>2602</td>
<td></td>
</tr>
<tr>
<td>2603</td>
<td></td>
</tr>
<tr>
<td>2604</td>
<td></td>
</tr>
<tr>
<td>2605</td>
<td></td>
</tr>
<tr>
<td>2606</td>
<td></td>
</tr>
<tr>
<td>2607</td>
<td></td>
</tr>
<tr>
<td>2608</td>
<td></td>
</tr>
<tr>
<td>2609</td>
<td></td>
</tr>
<tr>
<td>2610</td>
<td></td>
</tr>
<tr>
<td>2611</td>
<td></td>
</tr>
<tr>
<td>2612</td>
<td></td>
</tr>
<tr>
<td>2613</td>
<td></td>
</tr>
<tr>
<td>2614</td>
<td></td>
</tr>
<tr>
<td>2615</td>
<td>;</td>
</tr>
<tr>
<td>2616</td>
<td>fragment</td>
</tr>
<tr>
<td>2617</td>
<td>JAVA_ID_PART</td>
</tr>
<tr>
<td>2618</td>
<td></td>
</tr>
<tr>
<td>2619</td>
<td></td>
</tr>
<tr>
<td>2620</td>
<td></td>
</tr>
<tr>
<td>2621</td>
<td>WS : ( ' '</td>
</tr>
<tr>
<td>2622</td>
<td>{</td>
</tr>
<tr>
<td>2623</td>
<td>if (!preserveWhitespacesAndComments) {</td>
</tr>
<tr>
<td>2624</td>
<td>skip();</td>
</tr>
<tr>
<td>2625</td>
<td>} else {</td>
</tr>
<tr>
<td>2626</td>
<td>$channel = HIDDEN;</td>
</tr>
<tr>
<td>2627</td>
<td>}</td>
</tr>
<tr>
<td>2628</td>
<td>}</td>
</tr>
<tr>
<td>2629</td>
<td>COMMENT</td>
</tr>
<tr>
<td>2630</td>
<td></td>
</tr>
<tr>
<td>2631</td>
<td></td>
</tr>
<tr>
<td>2632</td>
<td>if (!preserveWhitespacesAndComments) {</td>
</tr>
<tr>
<td>2633</td>
<td>skip();</td>
</tr>
<tr>
<td>2634</td>
<td>} else {</td>
</tr>
<tr>
<td>2635</td>
<td>$channel = HIDDEN;</td>
</tr>
<tr>
<td>2636</td>
<td>}</td>
</tr>
<tr>
<td>2637</td>
<td>}</td>
</tr>
</tbody>
</table>
LINE_COMMENT :
    
    '//' "(('\n'|'\r')* '\r'? '\n')" |

    if (!preserveWhitespacesAndComments) {
      skip();
    } else {
      $channel = HIDDEN;
    }

    ;