

VDMTools

Java VDM++ ユーザマニュアル
ver.1.0



How to contact:

http://fmvdm.org/	VDM information web site(in Japanese)
http://fmvdm.org/tools/vdmttools	VDMTools web site(in Japanese)
inq@fmvdm.org	Mail

Java *VDM++ ユーザマニュアル 1.0*

— Revised for VDMTools v9.0.6

© COPYRIGHT 2016 by Kyushu University

The software described in this document is furnished under a license agreement.
The software may be used or copied only under the terms of the license agreement.

This document is subject to change without notice

目次

1	導入	1
2	Java クラスをプロジェクトに含める	2
3	翻訳オプション	11
4	制限	13
4.1	Java と VDM++間のスコープの相違	13
4.1.1	クラス名称	13
4.1.2	名称不一致	14
4.1.3	クラス要素のスコープ	15
4.1.4	サブクラスのインスタンス変数へのアクセス	17
4.2	副次的作用をもつ文における制限事項	19
4.2.1	クラスあるいはインスタンス変数の初期化	19
4.2.2	代入文の左辺における条件式	20
4.2.3	JavaLangObject 要素アクセス	20
4.3	言語構成の相違点	21
4.3.1	Java と VDM++ のキーワード	21
4.3.2	関数パラメータへの代入	21
4.3.3	新インスタンスへの代入の式としての利用	22
4.3.4	数値型	23
4.3.5	型変換	24
4.3.6	内部クラス	25
4.3.7	修飾された this の暗黙使用	26
4.3.8	匿名クラス	26
4.3.9	label、break、continue	26
4.3.10	選択肢分岐のない switch	27
4.4	サポートされていない概念	28
4.4.1	同時並行性	29
4.4.2	Unicode 文字	29
5	Java から VDM++ 翻訳の詳細	30
5.1	組込型	30
5.2	リテラル	30
5.3	名称	31
5.4	配列	33

5.5	クラス	34
5.5.1	メソッドと要素	34
5.5.2	継承	35
5.5.3	クラス修飾子	36
5.5.4	アクセス修飾子	36
5.5.5	静的初期化子	37
5.5.6	クラスについての情報取得	39
5.6	インターフェイス	41
5.7	null	42
5.8	式	46
5.9	文	48
5.9.1	If 文	48
5.9.2	Block 文	49
5.9.3	for 文	49
5.9.4	while 文	50
5.9.5	do while 文	51
5.9.6	switch 文	52
5.9.7	try catch 文	53
6	VDM++ 変換群	55
6.1	isMapCompLoop (Block Stmt)	56
6.2	ifTestTrue (Block Stmt)	57
6.3	ifTestFalse (Block Stmt)	58
6.4	isRedundantIfBlock (Block Stmt)	59
6.5	isRedundantIfBlockNoElse (Block Stmt)	61
6.6	ifToAnd (Block Stmt)	61
6.7	isRedundantDcl (Block Stmt)	62
6.8	ifToCases (If Stmt)	63
6.9	ifToEquiv (If Stmt)	65
6.10	nestedIfsNoElses (If Stmt)	66
6.11	whileIfTestTrue (While Loop)	67
6.12	orToNotEquiv (Binary Expr)	68
7	Java API クラス群の VDM++ モデル	69
7.1	java.lang クラス群	70
7.1.1	JavaLangArrayIndexOutOfBoundsException クラス	70
7.1.2	JavaLangBoolean クラス	70

7.1.3	JavaLangCharacter クラス	70
7.1.4	JavaLangClass クラス	71
7.1.5	JavaLangClassCastException クラス	72
7.1.6	JavaLangClassNotFoundException クラス	72
7.1.7	JavaLangComparable クラス	72
7.1.8	JavaLangConversionBufferFullException クラス	72
7.1.9	JavaLangDouble クラス	73
7.1.10	JavaLangException クラス	73
7.1.11	JavaLangIllegalAccessException クラス	74
7.1.12	JavaLangIllegalArgumentException クラス	74
7.1.13	JavaLangIllegalStateException クラス	74
7.1.14	JavaLangIndexOutOfBoundsException クラス	74
7.1.15	JavaLangInstantiationException クラス	75
7.1.16	JavaLangInteger クラス	75
7.1.17	J2VUTIL クラス	76
7.1.18	JavaLangNullPointerException クラス	76
7.1.19	Nullable クラス	76
7.1.20	JavaLangNumber クラス	76
7.1.21	JavaLangNumberFormatException クラス	77
7.1.22	JavaLangObject クラス	77
7.1.23	JavaLangRuntimeException クラス	77
7.1.24	JavaLangString クラス	78
7.1.25	JavaLangStringBuffer クラス	79
7.1.26	JavaLangSystem クラス	79
7.1.27	JavaLangThrowable クラス	80
7.1.28	JavaLangUnsupportedOperationException クラス	80
7.2	java.util クラス群	81
7.2.1	JavaUtilALitr クラス	81
7.2.2	JavaUtilALListItr クラス	81
7.2.3	JavaUtilAbstractCollection クラス	81
7.2.4	JavaUtilAbstractList クラス	81
7.2.5	JavaUtilAbstractMap クラス	82
7.2.6	JavaUtilAbstractSet クラス	83
7.2.7	JavaUtilCollection クラス	83
7.2.8	JavaUtilConcurrentModificationException クラス	83
7.2.9	JavaUtilDate クラス	83

7.2.10	JavaUtilDictionary クラス	84
7.2.11	JavaUtilEmptyEnumerator クラス	85
7.2.12	JavaUtilEmptyIterator クラス	85
7.2.13	JavaUtilEmptyStackException クラス	85
7.2.14	JavaUtilEntry クラス	85
7.2.15	JavaUtilEnumeration クラス	85
7.2.16	JavaUtilHTEntry クラス	86
7.2.17	JavaUtilHTKeySet クラス	86
7.2.18	JavaUtilHashMap クラス	86
7.2.19	JavaUtilHashSet クラス	88
7.2.20	JavaUtilHashtable クラス	88
7.2.21	JavaUtilIterator クラス	89
7.2.22	JavaUtilList クラス	89
7.2.23	JavaUtilListIterator クラス	90
7.2.24	JavaUtilLocale クラス	90
7.2.25	JavaUtilMap クラス	91
7.2.26	JavaUtilMissingResourceException クラス	92
7.2.27	JavaUtilNoSuchElementException クラス	92
7.2.28	JavaUtilObservable クラス	92
7.2.29	JavaUtilObserver クラス	93
7.2.30	JavaUtilProperties クラス	93
7.2.31	JavaUtilResourceBundle クラス	93
7.2.32	JavaUtilSet クラス	94
7.2.33	JavaUtilStack クラス	94
7.2.34	JavaUtilStringTokenizer クラス	94
7.2.35	JavaUtilVector クラス	95
7.3	java.io クラス群	95
7.3.1	JavaIoBufferedInputStream クラス	95
7.3.2	JavaIoBufferedOutputStream クラス	96
7.3.3	JavaIoBufferedReader クラス	96
7.3.4	JavaIoBufferedWriter クラス	97
7.3.5	JavaIoByteArrayInputStream クラス	97
7.3.6	JavaIoCharArrayReader クラス	98
7.3.7	JavaIoFile クラス	98
7.3.8	JavaIoFileDescriptor クラス	100
7.3.9	JavaIoFileInputStream クラス	100

7.3.10	JavaIoFileNotFoundException クラス	100
7.3.11	JavaIoFileOutputStream クラス	101
7.3.12	JavaIoFileReader クラス	101
7.3.13	JavaIoFileSystem クラス	101
7.3.14	JavaIoFileWriter クラス	103
7.3.15	JavaIoFilterInputStream クラス	103
7.3.16	JavaIoFilterOutputStream クラス	103
7.3.17	JavaIoIOException クラス	104
7.3.18	JavaIoInputStream クラス	104
7.3.19	JavaIoInputStreamReader クラス	104
7.3.20	JavaIoOutputStream クラス	105
7.3.21	JavaIoOutputStreamWriter クラス	105
7.3.22	JavaIoPrintStream クラス	105
7.3.23	JavaIoPrintWriter クラス	106
7.3.24	JavaIoReader クラス	106
7.3.25	JavaIoStreamTokenizer クラス	107
7.3.26	JavaIoStringWriter クラス	107
7.3.27	JavaIoUnsupportedEncodingException クラス	108
7.3.28	JavaIoWriter クラス	108
7.4	java.net クラス群	108
7.4.1	JavaNetURL クラス	108
7.5	java.text クラス群	109
7.5.1	JavaTextDateFormat クラス	109
7.5.2	JavaTextDecimalFormat クラス	109
7.5.3	JavaTextDecimalFormatSymbols クラス	110
7.5.4	JavaTextFieldPosition クラス	111
7.5.5	JavaTextFormat クラス	111
7.5.6	JavaTextMessageFormat クラス	111
7.5.7	JavaTextNumberFormat クラス	112
7.5.8	JavaTextParseException クラス	113
7.5.9	JavaTextParsePosition クラス	114
7.5.10	JavaTextSimpleDateFormat クラス	114
7.6	java.sql クラス群	114
7.6.1	JavaSqlConnection クラス	114
7.6.2	JavaSqlDriverManager クラス	115
7.6.3	JavaSqlResultSet クラス	115



7.6.4	JavaSqlSQLException クラス	117
7.6.5	SqlConnection クラス	117
7.6.6	SqlResultSet クラス	118
7.6.7	SqlStatement クラス	120
7.6.8	JavaSqlStatement クラス	120

1 導入

This manual gives an introduction to the Java to VDM++ feature of **VDMTools**. This feature can be used for reverse engineering existing legacy Java applications to VDM++. At the VDM++ level different kinds of analysis may then be conducted and new features specified and forward engineered. 本書では、**VDMTools** の中で Java から VDM++ の機能についての導入を行う。現存する古い仕様の Java アプリケーションを VDM++ へとリバースエンジニアリングを行うために、この機能を用いることが可能である。そのために VDM++ レベルで様々な種類の分析が行われ、新たな機能が指定されて、フォワードエンジニアリングが行われる可能性がある。

The Java to VDM++ translator is an add-on feature to the VDM++ Toolbox. This manual is an extension to the *User Manual for the VDM++ Toolbox* [3]. In general it is intended that the standard `javac` should always be invoked on a collection of Java files before they are included in a project with **VDMTools**. If Java code which cannot be accepted by `javac` is provided **VDMTools** may not behave correctly. Java から VDM++ への翻訳は、VDM++ Toolbox に対してアドオン機能となる。本書は、*User Manual for the VDM++ Toolbox* [3] の拡張版である。一般的には、標準 `javac` が **VDMTools** でプロジェクトに取り込まれる前の Java ファイルが収集される時点で、常に起動されるべきである。`javac` で受け入れることができない Java コードがもちこまれた場合には、**VDMTools** が正しく動作しない可能性がある。

This manual starts by explaining how to include Java classes in a project file in the VDM++ Toolbox. This is followed by an overview of the options for the Java to VDM++ translator, which include the ability to apply a set of transformations to the generated VDM++ which convert certain parts of it to equivalent but more abstract forms. More details of this feature can be found in Section 6. 本書では最初に、VDM++ Toolbox のプロジェクトに Java クラスを組み入れる方法を述べる。次に Java から VDM++ へ翻訳を行うためのオプションを概説するが、生成された VDM++ に対して変換の集合を当てはめること、つまりある部分をそれと同等だがより抽象的な形態へ変換する、ということも可能だ。この機能の詳細は Section 6 で見ることができる。

Section 4 describes the different limitations for the Java to VDM++ translator. This includes all the different situations which should be avoided in order to



automatically produce an equivalent VDM++ model for a collection of Java source files. 第 4 章では、the Java to VDM++ translator に対する制限のいろいろを述べる。これは、Java ソースファイルの集まりに対して同等の VDM++ モデルを自動的に生成するために、避けるべき様々な状況すべてを含めている。

Section 5 gives specific details of the translation process and also describes some of the design decisions made when developing the Java to VDM++ Translator, including the name conventions used. This section should be studied intensively before using the Java to VDM++ translator professionally. 第 5 章では、翻訳プロセスの具体的な詳細について述べ、また、Java to VDM++ Translator を展開する上でなされたデザイン決定をいくつか述べるが、使用される名称の慣習についても含めている。この章は the Java to VDM++ translator を職業的に使用する場合には、集中して学習するべきであろう。

Finally, Section 7 describes the subset of Java API which is available at the VDM++ level. 最後に第 7 章では、VDM++ レベルで役立つ Java API のサブセットを説明する。

2 Java クラスをプロジェクトに含める

Before your Java source files can be automatically translated to VDM++ using the translator described in this manual you need to include all the files you would like to have translated into a **VDMTools** project. 本書に記述した変換装置を用いて、Java のソースファイルが VDM++ へと自動的な翻訳を可能とする前に、まずは翻訳したいすべてのファイルを **VDMTools** プロジェクトに含めておく必要がある。

To do this, first start the VDM++ Toolbox (details of how to do this can be found in the general User Manual for the VDM++ Toolbox [3]), then press the  (Add Files) button on the (Project Operations) toolbar (or if you prefer you can select the action Add File to Project on the Project menu). The dialog box shown in Figure 1 will then appear. これを行うために、最初に VDM++ Toolbox をスタートし (これを行う方法の詳細は VDM++ Toolbox の一般向けユーザマニュアル [3] で見ることができる)、次に (Project Operations) ツールバー上の  (Add Files) ボタンを押す (あるいは、Project メニュー上でアクション Add File to Project を選択することもできる)。すると、図 1 で示されたダイアログが現

れる。

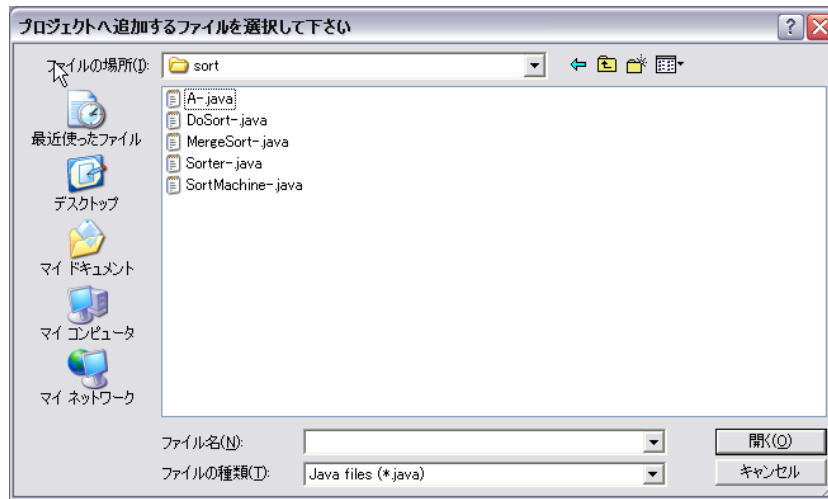


図 1: プロジェクトにファイルを加える

As an example, select the five `.java` files in the `vpphome/java2vdm/examples/sort` directory from the Toolbox distribution by holding down the `Ctrl` key and clicking the left-hand mouse button on each of the files in turn. Then press the “Open” button. The files will then be included in the project and will appear in the Project View of the Manager in the main Toolbox window as shown in Figure

2. You can also add a single file to a project by double clicking the left-hand mouse button on it (but note that this also closes the dialog box so it is not an efficient way of adding a number of files), and you can also mark a list of files at the same time by selecting the first and last files in the list (in either order), holding down the `Shift` key while making the second selection. 例として、Toolbox 配布の `vpphome/j2vexamples/sort` ディレクトリ中の 5 つのファイルを、`Ctrl` キーを押し各々のファイルを順に左側マウスボタンでクリックすることで、選択していく。次に “Open” ボタンを押す。するとファイルはプロジェクトに含まれ、図 2 に見られるように Toolbox メインウィンドウにある Manager の Project View に現れることになる。さらに 1 つのファイル上で左側マウスボタンをダブルクリックし、これをプロジェクトに加えることもできるし (ただしこれによってダイアログが閉じてしまうため、たくさんのファイルを加える場合は効率的な方法ではない)、また最初と最後 (どちらが先でもよい) を選択し後の選択で `Shift` キーを押した状態に保つことで、ファイル一覧の選択を同時に行うこともできる。

It is important here to note that it is also necessary to include some Java API

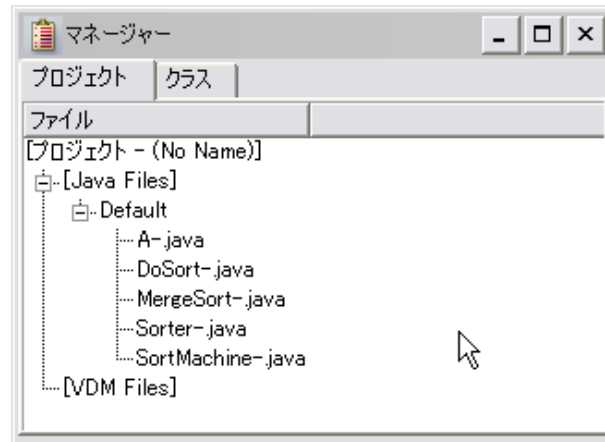


図 2: ファイル追加後の管理

skeletons in the project. These define functionality equivalent to the standard Java API classes except that in general they only include the signatures of methods because this information is all that is required in order to perform the necessary checks on your Java project files. These files are located in the `vpphome/java2vdm/javaapi/java` directory, and the particular files required for this application are shown in Figure 3. Add these to the project in the same way. ここで、プロジェクトに Java API スケルトンをいくつか含める必要があることに、注意を向ける必要がある。これらは一般的にメソッドのシグニチャを含めるだけである以外は、標準 Java API クラスと同等な機能を定義するものだが、対象の Java プロジェクトファイル上で必要なチェックを行うためには、これが必要とされる全情報となるからである。これらファイルは `vpphome/java2vdm/javaapi/java` ディレクトリ中に配置され、このアプリケーションに必要な特定ファイルは図 3 で示されている。同じ方法を用いて、これらをプロジェクトに加えよう。

Next you need to syntax check all your Java files, including the Java API skeleton files. 次に、Java API スケルトンファイルを含め、対象 Java ファイルすべての構文チェックを行う必要がある。



To do this, select all the files in the Project View of the Manager, then press the  (Syntax Check) button on the (Class Operations) toolbar. (Selecting the containing level folders and applying the syntax check operation to those has the same effect – this applies the syntax check operation to each of the files in the folders.) Notice that at this point the Log Window opens automatically (if it is not



図 3: 必要とされた Java API スケルトン

already open) and displays a message informing you of the success or failure of the check for each file. In addition, if syntax errors are discovered the **Error List** is also automatically invoked and the **Source Window** is automatically opened. See the general User Manual for the VDM++ Toolbox [3] for information about the Error List and the Source Window and about using the External Editor to correct errors. これを行うためには、Manager の Project View にある全ファイルを選択してから、(Class Operations) ツールバーの  (Syntax Check) ボタンを押そう。(フォルダレベルで選択を行いそれらに構文チェックを適用することでも、同様の効果を得る –つまりこれでフォルダ中の各ファイルに、構文チェック操作を適用することになる。) この時点で、Log Window が (まだ開いていない場合に) 自動的に開き、各ファイルに対するチェックが成功したか失敗したかを知らせるメッ

セージを表示することに注意しよう。加えて構文エラーが発見された場合には、自動的に Error List も起動され Source Window も開かれる。Error List と Source Window についてや、エラー修正のための External Editor 使用についての情報は、VDM++ ツールボックスの一般向けユーザーマニュアル [3] を参照しよう。





















The next step is to “type check” your files, that is to check that they only use the subset of Java that can be translated to VDM++ (see Section 4 for a description of the current limitations). However, this is not necessary for the Java API skeleton files because these do not need to be translated and syntax checking gives enough context information to allow you to check and translate the Java files from your own application successfully. So just select the folder containing the application files and invoke the type checker by pressing the  (Type Check) button on the (Class Operations) toolbar. 次の段階は対象ファイルを“型検査”することで、VDM++ に翻訳可能な Java のサブセットだけを用いているかどうかをチェックする(現段階の制限の記述は第 4 を参照)。しかし Java API スケルトンファイルに対してはこれは不要で、なぜならばこれらは翻訳の必要がなく、対象アプリケーションの Java ファイルを自身でうまくチェックし翻訳するために十分な文脈情報を、構文チェックで得られるからである。したがって、アプリケーションファイルを含むフォルダを選択し、(Class Operations) ツールバー上の  (Type Check) ボタンを押して、型検査をただ発動することである。Note that after Java files have been successfully syntax checked the names of the classes defined in those files are listed in the Java View in the Class View of the Manager. You can select individual classes here to which you want to apply Toolbox operations (i.e. instead of applying the operations to all the classes in a file as is done in the Project View). You can also see the current status of each of the individual Java classes in the project. Java ファイルの構文チェックがうまくなされると、その後はそれらのファイルで定義されたクラスの名称が、Manager の Class View にある Java View に一覧されることに注意しよう。この段階でツールボックス操作を適用したいクラス個々を選択することができる(つまり Project View で行ったのと同様の、1 ファイル内の全クラスに対して操作を適用するという代わりにである)。さらにプロジェクト内の Java クラス個々の、それぞれの現在状態を見ることもできる。

Figure 4 shows the current state of the Java View. The symbol  in the Syntax column next to each class indicates that it has been syntax checked successfully, and the symbol  in the Type column next to the classes belonging to the example application indicate that these have also been successfully type checked. In case

the syntax or type check was unsuccessful, the symbols  respectively  are shown in the appropriate columns instead. In addition, if a source file is edited the symbol  (this is the symbol  with a red triangle superimposed) is shown in the Syntax column to indicate that there is an inconsistency between the version of the file currently in the Toolbox and the version on the file system. The file must be syntax checked (and type checked) again before proceeding. 図 4 は Java View の現在状態を表している。各クラスの隣の Syntax 列の記号  は、構文チェックがうまくなされたことを示し、例題アプリケーションに含まれているクラスの横の Type 列の記号  は、それらが型チェックもまたうまくなされたことを示す。構文チェックと型チェックとが成功しなかった場合は、記号  や  が代わりにそれぞれ適した列に示される。さらにもしソースファイルが編集されているならば、記号  (これは  に赤い三角形が重なった記号) が構文列に表われて、Toolbox 中の現ファイルバージョンとファイルシステムのバージョンに矛盾があることを示す。このファイルは処理が進む前に再度、構文チェック (および型チェック) がなされなければならない。

Once your files/classes have passed the syntax and type checks, they are ready to be translated to VDM++. Select the five classes belonging to the example application (you do not need to translate the Java API skeleton classes but you will later need to load VDM++ equivalents of these which are also supplied with the Toolbox), then press the  button to translate them to VDM++. The symbol  appears in the VDM column of the Java View next to each of the selected classes to indicate that it was translated successfully (see Figure 5). In case of failure, the symbol  is shown instead. ファイルあるいはクラスを構文チェックと型チェックに渡してしまえば、その時点で VDM++ に翻訳される準備が整ったことになる。例題アプリケーションに属する 5 つのクラスを選択し (Java API スケルトンクラスを翻訳する必要はないが、後でツールボックスで提供されるこれらの VDM++ 同等物を読み込む必要があるだろう)、その後 VDM++ に翻訳するために  ボタンを押そう。記号  は翻訳が成功したことを示し、Java View で VDM 列の選択クラス各々の横に現れる (図 5 を参照)。失敗の場合は、この代わりに記号  が表示される。

This generates five .vpp files, one for each class. These can be seen by returning to the Project View (see Figure 6). それぞれのクラスに対して 1 つで、5 つの .vpp ファイルが生成される。Project View (図 6 参照) に戻れば、これらを見ることができる。

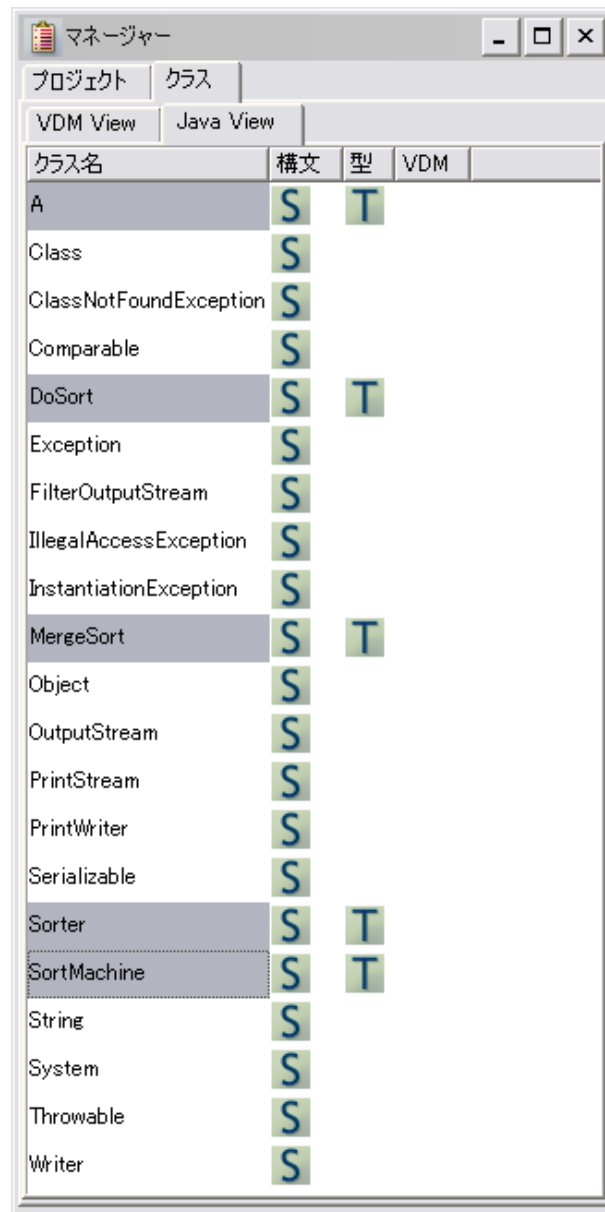




図 4: Java View

You have now finished with the Java files and we recommend that you remove them from the project – select them all and press the  (Remove Files) button on the Project Operations toolbar. Java ファイルについての処理はこれで終わりなので、プロジェクトからの削除を推奨する – これらすべてを選択してから Project Operations ツールバー上の  (Remove Files) ボタンを押すこと。

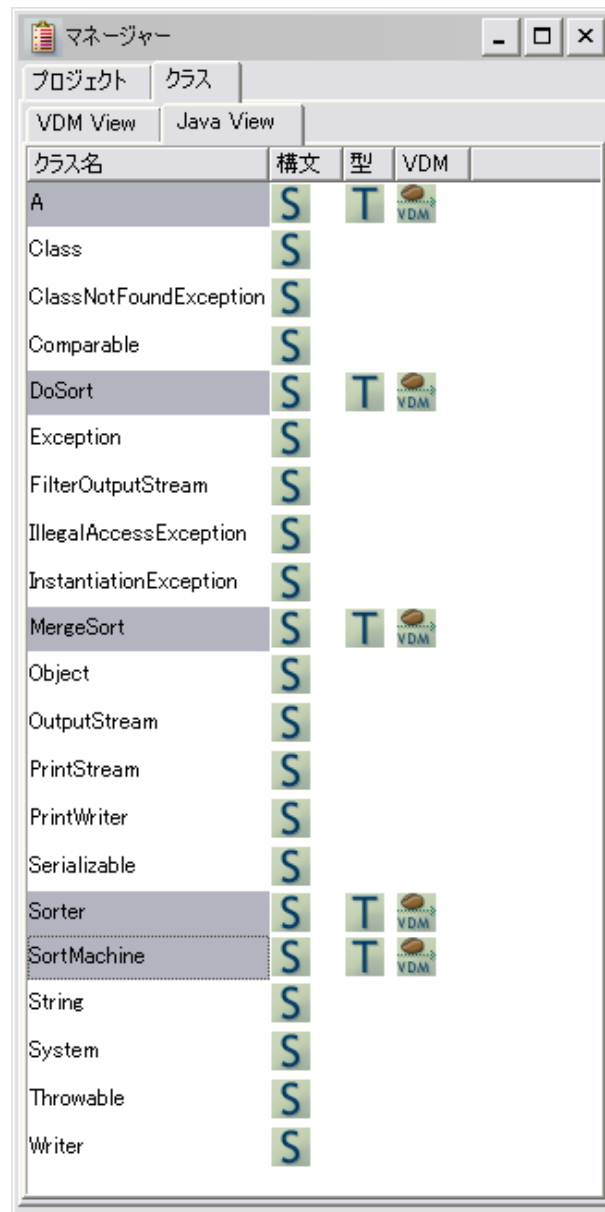


図 5: 翻訳後の Java View

The final step is to add the VDM++ files which correspond to the Java API skeletons used with the Java files. These can be found in the `vpphome/java2vdm/javaapi/vpp` directory, but since a lot of files are required we recommend that to save time you add the contents of all the appropriate subdirectories rather than the individual files. These subdirectories are shown in Figure 7. 最終段階で、Java ファイル

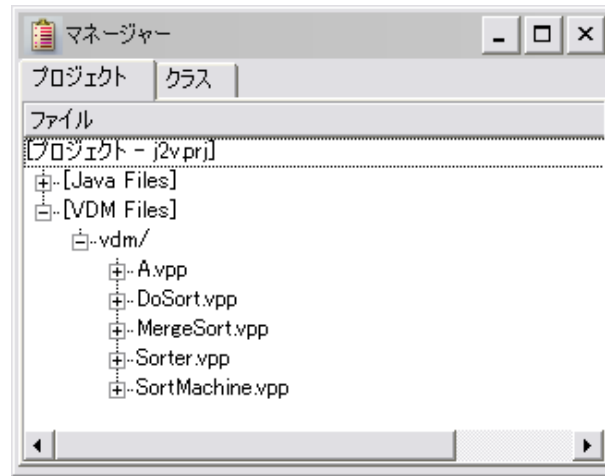


図 6: 生成された VDM++ ファイル

ルと共に用いた Java API スケルトンに相当する VDM++ ファイルを追加する。これらは `vpphome/java2vdm/javaapi/vpp` ディレクトリで見つけることができるが、結構たくさんのファイルが要求されるため、個々のファイルより適当なサブディレクトリの全内容を追加することで、手間を省くことを推奨する。図 7 ではこういったサブディレクトリが示されている。

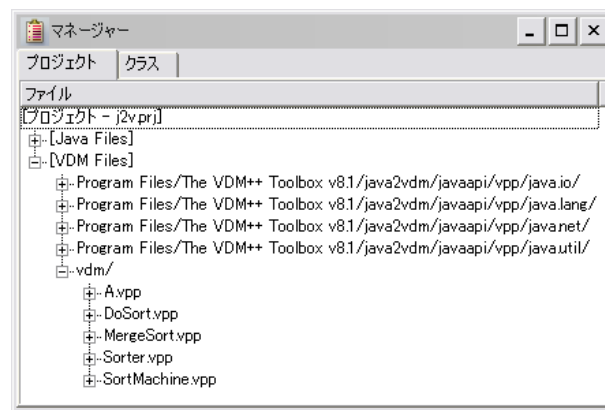




図 7: VDM++ API ファイル

In fact these files also contain only skeletons of the appropriate classes in general, the main functionality being provided through the dynamic link facility of the Toolbox (see [1] for details). Thus the bodies of the methods in the skeleton classes appear in general as not yet specified and the functionality

is linked in as executable C++ code. In order to access this code (so that, for example, you can use the interpreter to debug your specification), you need to make sure the Toolbox knows where to find it. In fact it is contained in the file `vpphome/bin/j2vdl1.so`, so you should either add this full path name to your global path or include the directory `vpphome/java2vdm/javaapi` in the list of directories referenced by the environment variable `VDM.DYNLIB`. (Alternatively you can copy the file `vpphome/bin/j2vdl1.so` to the current directory.) 実際には、主な機能がツールボックスの動的リンク設定を通して提供されるので、通常これらファイルもまた適当なクラスのスケルトンを含むだけである (詳細は [1] 参照)。このように、通常 is not yet specified と表されたスケルトンクラスのメソッド本体とその機能は、実行可能な C++ コードとして組み入れられている。このコードにアクセスするには (それにより、仕様デバッグのためにインタプリタ使用が可能である)、ツールボックスがどこを捜せばよいかを知っていると信じる必要がある。実際にはファイル `vpphome/bin/j2vdl1.so` 中にこれが含まれているので、グローバルパスにこのフルパス名を付け加えるか、あるいは、環境変数 `VDM.DYNLIB` で参照されるディレクトリ一覧にディレクトリ `vpphome/java2vdm/javaapi` を含めるべきである。(代わりに、カレントディレクトリにファイル `vpphome/bin/j2vdl1.so` をコピーすることもできる。)

You now have a set of VDM++ files which correspond to your original Java application and you can interact with these as with any other VDM++ project. See the User Manual for the VDM++ Toolbox [3] for details. これで元の Java アプリケーションに相当する VDM++ ファイル集合を得たわけで、他の VDM++ プロジェクト同様、これらとの情報のやりとりが可能である。詳細は VDM++ ツールボックス [3] ユーザーマニュアルを参照のこと。

3 翻訳オプション

The Java to VDM++ translator has two options which can be set in the Java to VDM++ panel of the Project Options window, which is displayed by pressing the  (Project Options) button on the (Project Operations) toolbar. This is shown in Figure 8. Java から VDM++ 翻訳には、Project Options 画面の Java to VDM++ パネルで設定を行うことができる 2 つのオプションがあり、(Project Operations) ツールバーの  (Project Options) ボタンを押すことで表示される。これは 図 8 に示されている。

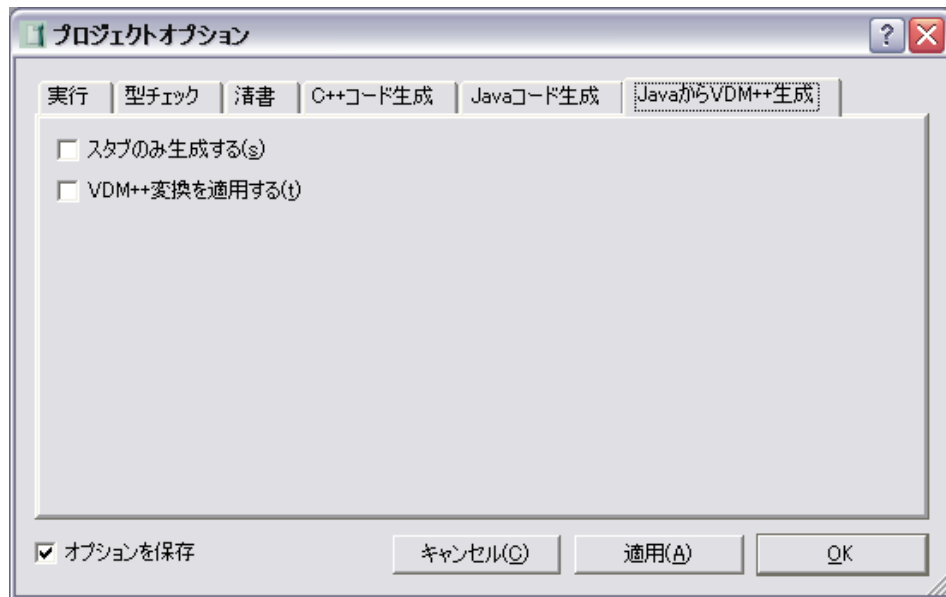


図 8: Java から VDM++ 翻訳でのオプション設定

The possible options are as follows: 可能なオプションは以下の通り:

Generate stubs only: If this option is selected it means that the translator will only produce VDM++ code for class members and method signatures and will use the “is not yet specified” construct for the VDM++ operation bodies;

スタブのみ生成: このオプションが選択された場合の翻訳は、クラス要素およびメソッドシグニチャのみの生成を行い、VDM++ 操作本体に対しては “is not yet specified” 構成を用いることになる;

Apply VDM++ transformations If this option is selected the VDM++ which is generated by the translator is also passed through a series of transformations which convert certain parts of the specification to equivalent but more abstract forms. This feature is described in more detail in Section 6.

VDM++ 変換の適用 このオプションが選択されると、翻訳で生成された VDM++ は、さらに仕様部分をそれと同等だがさらに抽象的な形態への変換にと引き渡される。この機能の詳細は、さらに 第 6 章で述べられる。

4 制限

This section explains the current limitations on the Java to VDM++ translator and also includes recommendations on how your Java source files should be modified before using the translator. この章では、Java から VDM++ 翻訳における現時点での制限について説明を行い、翻訳にかける前にどのように Java ファイルの修正を行うべきかという提案をさらに含める。

4.1 Java と VDM++ 間のスコープの相違

The scope rules for Java and VDM++ are not identical. Thus, there are a number of limitations for the Java to VDM++ translator which are related to this difference. In general it is worth noting here that the default modifier in Java is package. In VDM++ the default modifier is private and no semantic package structure is present. Thus, when the Java to VDM++ translator translates all default modifiers to default modifiers in VDM++ there is a semantic difference. This gives less visibility than in Java, but this reflects the differences in scoping for the two languages. A number of limitations related to scope issues are presented below. Java と VDM++ におけるスコープ規程は同一ではない。したがってこの相違に関連して、Java から VDM++ の翻訳にたくさんの制限がある。一般的にはここで注目されるのが、Java の既定の修飾子がパッケージ化されているということである。VDM++ では既定の修飾子はプライベートのもので、意味上のパッケージ構造は存在しない。このように、Java から VDM++ 翻訳では、すべての既定修飾子を VDM++ の既定修飾子に翻訳する場合に、意味上の相違がある。これは Java よりも可視性を弱めるが、2つの言語間のスコーピングの相違を反映するものだ。スコープ問題に関連した多くの制限を、以下に提示する。

4.1.1 クラス名称

Class names in Java are considered as being local to packages, which means that it is possible in a Java program to have two classes with the same name provided they belong to different packages. In VDM++, however, the notion of packages is purely syntactic: class names are effectively globally visible and there cannot

be two classes with the same name in the same project, even if they belong to different packages. Thus, if two Java classes have the same name, one of these must be renamed before translation to VDM++. Java のクラス名称はパッケージに対して局所的であると考えられ、つまり Java プログラムでは異なるパッケージに属する同じ名称の 2 つのクラスの提供が可能であることを意味している。VDM++ ではしかし、パッケージの観念は純粹に構文的なことであり：クラス名称は効果的に全体が可視されるもので、たとえ異なるパッケージに属していても同じプロジェクトに同じ名称の 2 つのクラスの存在はあり得ない。このように、2 つのクラスが同じ名称をもつ場合、そのうち 1 つは VDM++ 翻訳される前に改名されなければならない。

4.1.2 名称不一致

In Java there are less restrictions on the overloading of names than in VDM++ so that, for example, in Java it is possible to have an instance variable and a method with the same name in the same class whereas in VDM++ this causes an error. Java では VDM++ におけるよりも名称多重化について制限は少なく、例えば Java では同じクラス中で同じ名称のインスタンス変数とメソッドの存在が可能であるが、一方の VDM++ ではこれはエラーを引き起こす。

```
class B
{
    public int b() { return 0 };
}
class A extends B
{
    b: int:=0;
}
```

```
class B

operations
    public b : () ==> int
    b() == ( return 0);

end B

class A is subclass of B
```

```
instance variables
  b: int:=0;      -- Error: "b" is multiple defined in super classes
  b: int:=0;      -- Error: "b" はスーパークラスで多重定義されている

end A
```

In general, the restriction in VDM++ is that there should never be two different constructs with the same name visible in the same scope and the Java should generally respect this. Of course the visibility of constructs can be determined by the access modifiers so that it is possible for a construct in a subclass to have the same name as a private construct in a superclass as in the following example: 一般的に VDM++ における制限は、同一スコープ内に同じ名称をもつ異なる 2 つの構成要素が見えてはならないということであり、通常は Java でこれを考慮すべきである。もちろん構成要素の可視性はアクセス修飾子で決定ができるため、以下の例題に示す通り、サブクラスの構成要素がスーパークラス内の局地的な構成要素と同じ名称をもつことは可能である:

```
class B
{
  private int b() { return 0 };
}
class A extends B
{
  b: int:=0;
}
```

4.1.3 クラス要素のスコープ

In Java, the fact that the default modifier is package means that subclasses can reference declarations from superclasses if no explicit modifiers are included as in the following example where `super.i` references the declaration of `i` in class B: Java において既定修飾子がパッケージになっているということは、もし明示的な修飾子が含まれないならば、サブクラスがスーパークラスから宣言参照が可能であることを意味するが、以下の例題では `super.i` がクラス B における `i` の宣言を参照している:

```
class B
{
    int i=1;
}

class A extends B {
    int j=0;

    void a() { j = super.i; }
}
```

In VDM++, however, declarations are by default assumed to be private, so that the direct analogue of the above in VDM++: **しかしながら VDM++の宣言は既定で局所的であると仮定されるため、上記を VDM++に直接置き換えると以下のようになる:**

```
class B
instance variables
    i : int := 1;

end B

class A is subclass of B
instance variables
    j : int:=0;

operations

a: () ==> ()
a() ==
    j:= B'i; -- Error: Access violation for "B'i"
    j:= B'i; -- Error: "B'i" に対するアクセス違反

end A
```

would yield a compile-time error stating that the instance variable B'i is not in scope in class A. This can be fixed easily by adding a protected or public modifier to the declaration of i in class B and the translator currently issues a warning and asks the user to specify these access modifiers explicitly¹. **結果は、インスタ**

¹The translation could be modified to automatically generate public or protected modifiers by default in such situations.

ンス変数 `B.i` がクラス `A` のスコープ内にはないという、コンパイルエラーを引き起こすことになるだろう。クラス `B` における `i` の宣言では、`protected` 修飾子 または `public` 修飾子を加えることにより簡単に手直しができて、翻訳では現在は警告が発せられてユーザーに明示的にこれらアクセス修飾子を指定するように求めている²。

4.1.4 サブクラスのインスタンス変数へのアクセス

In Java selection of class attributes is determined at compile time, whereas it is determined at run-time in VDM++. This means that there is a slight difference in semantics between Java sources and translated VDM++ descriptions in a few cases. This can be an issue when class attributes are redefined by a subclass and instances of both the superclass and the subclass are intermixed as in the following example in which an instance of the subclass is assigned to a variable representing the superclass: クラス属性の選択は Java ではコンパイル時に決定されるが、VDM++では実行時に決定される。これは、Java ソースと翻訳された VDM++記述との間に多少は意味上の違いが生じている場合があることを意味している。これは、クラス属性があるサブクラスによって再定義されスーパークラスとサブクラスの両インスタンスが混じる場合や、下記の例題のようにサブクラスのインスタンスがスーパークラスを表す変数に代入された場合に、問題となる可能性がある:

```
class S {int i=0;}
class T extends S {int i=1;}
class A {
    void a() {
        T t=new T();
        JavaLangSystem.out.println(t.i);
        S s=new S();
        JavaLangSystem.out.println(s.i);
        s=t;
        JavaLangSystem.out.println(s.i);
    }
}
```

²このような状況において、自動的に `public` または `protected` の修飾子を生成するように、既定で翻訳修飾できるだろう。

The result of evaluating the above will be that the sequence of numbers 1, 0, 0 will be printed out. Note that the the scope rules of Java cause the last number to be 0 since the value of *s* is fixed at compile time. 上記を評価した結果は、数列 1、0、0 が印刷出力されてくるはずである。Java のスコープ規則で、*s* の値はコンパイル時に固定されるために 0 となることには、注意しよう。

Compare this with the following VDM++ specification of the “same” example: “同じ” 例題で、以下の VDM++ 仕様と比べてみよう:

```
class S
instance variables
  public i:int:=0
end S

class T is subclass of S
instance variables
  public i:int:=1
end T

class A
operations
public Test: () ==> seq of int
Test() == (
  dcl t:T:=new T(),
    s:S:=new S(),
    res: seq of int:=[];
    res:=res^[t.i];
    res:=res^[s.i];
    s:=t;
    res:=res^[s.i];
  return res
)

end A
```

Here the result of calling the Test operation is a sequence of 1, 0, 1 because in VDM++ the third value is determined at run-time rather than at compile time as done in Java. Care should therefore be taken to avoid such situations in the Java code³. ここでは Test 操作を呼び出した結果は 1, 0, 1 という数列になる

³A future version of the type checker (applied statically to the Java source code) will detect this kind of problem and produce a warning, but this is not yet implemented.

が、Java でコンパイル時に決定された 3 番目の値が VDM++ では実行時に決定されるからである。したがってこのような状況を避けるための処置は、Java コードにおいてとられるべきである⁴。

4.2 副次的作用をもつ文における制限事項

In Java expressions can have side effects. For example, the expression `i++` returns the value of `i` and as a side-effect also increments `i` by 1. Such an expression therefore effectively corresponds to a sequence of VDM++ statements. However, the syntax of VDM++ does not allow sequences of statements to occur at arbitrary positions within a specification, which imposes certain constraints on the Java to VDM++ translator. We present these below. Java において、式は副次的作用をもつことが可能である。例えば、式 `i++` は `i` の値を返すと共に、副次的作用として `i` を 1 増加させる。したがってこのような式は、実際には VDM++ 文の連続に相当するものとなる。しかしながら VDM++ の構文では、文の連続が仕様中の任意の位置に現れることが許されてはいないため、Java から VDM++ への翻訳には確実に制限が与えられていることになる。以下にこれらを提示する。

4.2.1 クラスあるいはインスタンス変数の初期化

In Java it is possible to use expressions which have side effects on the right-hand side of initialisation expressions in a class declaration as in the following example: Java では下記の例題にあるように、クラス宣言中の初期化式の右辺で、副次的作用をもつ式を用いることができる:

```
class A {  
  int i=1;  
  int j=i++;  
  static int k=1;  
  static int l=k++;  
}
```

⁴型検査 (Java ソースコードに静的に適用されているもの) の将来版では、この種の問題を発見し警告を発することになるだろうが、これはまだ実装されていない。

In the translator, however, we require that only expressions without side effects can be used in initialisation expressions⁵. しかしながら翻訳中の初期化式においては、副次的作用をもたない式だけ利用できることを要求したい⁶.

4.2.2 代入文の左辺における条件式

In Java it is possible to use a conditional expression on the left-hand side of an assignment statement and this is supported by the Java to VDM++ translator. However, in Java the alternatives in the conditional statement can involve expressions with side effects as in the following example: Java では代入文の左辺に条件式を用いることが可能で、Java から VDM++ への翻訳でもこれがサポートされている。しかし Java では以下の例題にあるように、条件文の選択肢で副次的作用をもつ式を含めることができる:

```
(i==0 ? a[i++] : b)[0] = ...
```

These are not supported by the translator, which requires that it must be possible to translate the alternatives to VDM++ expressions not sequences of VDM++ statements⁷. これらは翻訳でサポートされていないため、VDM++文の連続には、VDM++式に対しては選択肢の翻訳が可能でなければならないことを要求している⁸.

4.2.3 JavaLangObject 要素アクセス

JavaLangObject member access expressions which involve expressions with side effects, for example: 副次的作用をもつ式を含む JavaLangObject 要素アクセス式は、例えば次の通りとなる:

⁵In some special cases, including this example in fact, it would actually be possible to make a translation so it may be possible to relax this restriction in the future.

⁶実際はこの例題を含めて、何らかの特別な場合には翻訳を行うことが可能で、将来的にはこの制限が緩められる可能性はある。

⁷Again in some special cases it would be possible to make a translation so it may be possible to relax this restriction in the future.

⁸繰り返すが、特別な場合においては翻訳も可能となるかもしれない、将来的にこの制限を緩めることができる可能性がある。

```
... a[i++].b[i++] ...
```

also cannot in general be translated to VDM++ and are not currently accepted by the translator in any form⁹. これらもまた一般的には VDM++ に翻訳されるのは不可能であり、現時点ではどのような形式の翻訳においても受け入れられていない¹⁰。

4.3 言語構成の相違点

In a number of areas the syntactic and semantic differences between Java and VDM++ cause problems with translation. We discuss these areas below. 多くの領域において、Java と VDM++ の間の構文的かつ意味上の相違が、翻訳において問題を引き起こす。以下にこれらの領域を論じる。

4.3.1 Java と VDM++ のキーワード

Some of the keywords in VDM++ are not keywords in Java (e.g. `len`, `value`, `is_bool`) which means they can be used as the names of classes, instance variables, functions, etc. in Java. These should be renamed before translation. VDM++ のキーワードのいくつかは、Java のキーワードでない(例えば `len`、`値`、`is_bool`)、つまり Java ではそれらをクラス、インスタンス変数、関数、等の名称に用いることができることを意味している。これらは翻訳前に、名称替えがなされるべきである。

4.3.2 関数パラメータへの代入

In Java, it is possible to assign a value to a parameter of a function within the body of the same function, as, for example, in: Java においては、ある関数のパ

⁹Again simple forms could be translated so this restriction could possibly be relaxed in the future.

¹⁰繰り返すが将来的には、簡単な形式のものは翻訳できるようになるであろうし、制限が緩和される可能性もあるだろう。

ラメータへ値を代入することが、同じ関数の本体内において可能であり、例えば次にある通りである:

```
int f(int i) { ...; i=1; ...; }
```

In VDM++, this is not possible so functions which include such assignments cannot be translated. VDM++ではこれは不可能で、このような代入が含まれる関数の翻訳はできない。

4.3.3 新インスタンスへの代入の式としての利用

In Java the assignment statement Java において、次の代入文では

```
a1 = new A();
```

returns the value a1 which represents a new object of class A and this assignment statement can therefore be used to interact directly with that object as in the following example: クラス A の新しいオブジェクトを表す値 a1 を返しているの
で、以下の例題のように、この代入文を用いてオブジェクトと直接的に相互作用
を行うことが可能だ:

```
(a1 = new A()).i = 1;
```

In VDM++ this notation is not allowed (because the assignment does not return a1 as a result so the field reference .i is not being applied to an object). Expressions of this form therefore cannot be translated. They should instead be written as a sequence of statements in which the assignment is factored out, as in: VDM++でこの表記法は許されていない (代入結果は a1 を返さないなので、項目参照の .i がオブジェクトに適用されないからだ)。したがってこの形式の翻訳は不可能である。その代わりに次のように、代入文に分解される連続文として、書かれるべきである:

```
a1 = new A();  
a1.i = 1;
```

4.3.4 数値型

In Java, the numeric types ‘int’, ‘long’ and ‘real’ are considered as being distinguishable, so that, for example, with the following definitions Java では、数値型 ‘int’、‘long’、‘real’ が区別可能と考えられて、例えば以下のような定義で

```
class A
operations

public class A
{
    int o(int i) {return 0;}
    int o(long i) {return 1;}
    int o(real i) {return 2;}
}
```

the evaluation of the expression `o(j)` will yield 0 if `j` is of type ‘int’, 1 if `j` is of type ‘long’, and 2 if `j` is of type ‘real’. In VDM++, however, there is no type ‘long’ (it is effectively equivalent to ‘int’) and the type ‘int’ is a subtype of the type ‘real’ so that if `j` is of type ‘int’ it is also of type ‘real’. In translating the above example, therefore, the second function would override the first (because both ‘int’ and ‘long’ translate to ‘int’) and would also cause a conflict with the third when applied to integer arguments (because the fact that ‘int’ is a subtype of ‘real’ means that the third function can be applied to values of type ‘int’ as well as values of type ‘real’). Overloading functions in Java in such a way that distinguishing between them relies on the distinctions between numeric types should therefore be avoided. 式 `o(j)` の評価では、もし `j` が ‘int’ 型ならば 0、‘long’ 型ならば 1、‘real’ 型ならば 2、となる。しかしながら、VDM++ で ‘long’ 型はなく (事実上 ‘int’ 型と等しい) しかも ‘int’ 型は ‘real’ 型のサブタイプであるために、もし `j` が ‘int’ 型ならば ‘real’ 型でもある。したがって上記例題を翻訳する場合に、2 番目の関数は 1 番目の関数を上書きすることになるだろうし (‘int’ と ‘long’ は両方 ‘int’ に翻訳されるから)、整数引数の場合はこれは 3 番目の関数と矛盾することになる (‘int’ は ‘real’ のサブタイプであるということは、3 番目の関数は ‘real’ 型の値と同様 ‘int’ 型の値にも適用できるということを意味するからだ)。したがって、関数間の違いの認識を数値型間の相違に依存しているこのようなやり方で Java 関数を上書きするのは、避けるべきである。

4.3.5 型変換

In Java, the type definition associated with a particular value can cause a calculated value to change in order to conform to the declared type, whereas in VDM++ such an inconsistency between the actual type and the declared type would give rise to an error. For example, in the following Java code Javaにおいて特定の値に関連する型定義は、宣言された型に一致させるために計算された値の変化を引き起こす可能性があって、一方 VDM++ ではこのような実際の型と宣言された型との間の不一致がエラーを引き起こすことになる。たとえば以下の Java コードにおいて

```
int j = 5;
int i = j/2;
```

the value of *i* becomes 2 and not 2.5 because *i* is declared to be an integer. *i* の値は 2.5 でなく 2 となるが、これは *i* が整数であると宣言されているからだ。

In a similar way, Java also does explicit conversion of the actual parameters of methods. For example, an operation ‘op’ which is defined to take a string as an argument can be called with a character ‘A’ as that parameter: the character ‘A’ is in fact converted to the string “A”. 同様の方法で、Java はメソッドにおける実際のパラメーターの明白な変換もまた行う。例えば、引数に文字列をとるように定義された操作 ‘op’ を、パラメーターとしての文字 ‘A’ と共に呼び出すことが可能で：この時文字 ‘A’ は実際には文字列 “A” に変換される。

```
... op(JavaLangString str) {...}
...

op('A')
```

The Java to VDM++ Translator translates the Java code precisely as written and does not simulate such implicit type conversions, which means that the specification generated would contain type errors (in the first example a run-time type error would arise when trying to assign a real value to an integer variable, whereas the second example would give a static type error because a string should be a sequence of characters and not just a single character). Implicit type conversions should therefore not be used in the Java code. The Java to VDM++ Translator は Java コードを書かれている通りに詳細に翻訳し、このような暗黙の型変換を

模倣しない、つまり生成された仕様には型エラーが含まれることを意味している (最初の例題では、実数値を整数変数に代入しようとする場合に実行時に型エラーが生じ、一方2番目の例題では、文字列は単なる1文字でなく複数文字の列であるべきなので静的な型エラーを起こす)。したがって Java コードでは、暗黙の型変換を用いるべきでない。

4.3.6 内部クラス

In Java it is possible to nest class definitions using inner classes as in the following example: 以下の例題にあるように、Java では内部クラスを用いてクラス定義を行うことができる:

```
class WithDeepNesting{
    boolean toBe;
    WithDeepNesting(boolean b) { toBe = b;}
    class Nested {
        boolean theQuestion;
        class DeeplyNested {
            DeeplyNested(){
                Nested.this.theQuestion
                    = WithDeepNesting.this.toBe || !WithDeepNesting.this.toBe; }}}}

```

This example also includes examples of the use of the qualified `this`, which allows an inner class to reference definitions belonging to one of its containing classes – the keyword `this` is prefixed with the name of the appropriate containing class. この例題には、修飾された `this` を用いる例も含まれ、それを含むクラスの1つに属する定義を参照する内部クラスを許す – キーワード `this` はクラスを含む適切な名称で前修飾される。

VDM++ does not support inner classes, nor the qualified `this`, so neither of these is supported by the Java to VDM++ translator¹¹. VDM++ は内部クラスをサポートしないし、修飾された `this` もサポートされないため、これらどちらも Java から VDM++ の翻訳ではサポートされていない¹²。

¹¹It might be possible to support these at some stage in the future, for instance by converting the inner classes to normal classes and automatically renaming them and their attributes where appropriate, but this requires further investigation.

¹²将来的にある段階で、これらをサポートすることは可能かもしれない、たとえば内部クラスを通常クラスに変換すること、そして適切な場所で自動的にそれらとそれらの属性の名称を変更すること、ただしこれは更なる検討が必要である。

4.3.7 修飾された `this` の暗黙使用

The `this` keyword can also be used implicitly. Semantically the use of a class name in a field access expression is equivalent to an implicit use of the `this` construct. For example an expression of the form `ClassName.super.Identifier` semantically means `((NameOfSuperClass)ClassName.this).Identifier`. This implicit use of `this` is not supported by the Java to VDM++ translator. `this` キーワードを暗黙に用いることも可能だ。意味上では、項目アクセス式においてクラス名称を使用することは、`this` 構成の暗黙使用と同じである。たとえば `ClassName.super.Identifier` という形の式は、意味的には `((NameOfSuperClass)ClassName.this)` を示している。`this` のこのような暗黙の使用は、Java から VDM++ の翻訳ではサポートされない。

4.3.8 匿名クラス

Anonymous classes in Java have no counterpart in VDM++ and are not supported by the Java to VDM++ translator¹³. Java における匿名クラスは、VDM++ において相当するものはない¹⁴。

4.3.9 `label`、`break`、`continue`

Using `label:`, `break label`, `continue label` and `continue` in Java effectively corresponds to using GOTO's – the flow of control is interrupted and transferred to another point. This is not supported in VDM++ so these constructs cannot be translated. Java において `label:`、`break label`、`continue label`、`continue` を用いることは事実上、GOTO を用いるのに相当する – コントロールの流れは妨げられもうひとつの地点に移される。これは VDM++ ではサポートされないため、このような構築を翻訳することはできない。

¹³It would be fairly easy to support these if inner classes can be supported: the translator could simply allocate an arbitrary name to the anonymous class and then treat it in the same way as an inner class.

¹⁴もし内部クラスがサポートされているならば、これらを適正にサポートすることは簡単であろう: 翻訳では単に匿名クラスに任意の名称を割り当てて、内部クラスと同じ方法で扱うことができるであろう。

The use of `break` alone (i.e. with no label) to leave a loop is supported, however, though this is done using exception handling which means that the VDM++ generated looks quite different from the Java source code. We therefore recommend that `break` should not be used to exit from loops. しかし、ループを抜けるための `break` のみの使用 (つまり ラベルは不使用) はサポートされている、とはいえそれは例外操作を用いることによって、つまり生成された VDM++ は Java ソースコードとはまったく異なる様相をとることを意味する。したがって、ループから抜けるために `break` を用いるべきではないことを提言する。

4.3.10 選択肢分岐のない `switch`

In Java it is not necessary to put `break` between alternatives in a `switch` statement, so that the following is valid: Java では必ずしも `switch` 文の選択肢間に `break` をおく必要はないので、次が有効である:

```
oneOrTwo=0;
switch(i) {
  case 1:
    secondAlternative=false;
  case 2:
    secondAlternative=true;
  break;
}
```

However, if `switch(1)` is evaluated `secondAlternative` will have the value `true` – since there is no `break` between the cases the execution will continue to the final `break`. This means that both `switch(1)` and `switch(2)` have the same effect with respect to `secondAlternative`, namely `secondAlternative=true` and, so the line `secondAlternative=false` is entirely redundant and we could achieve the same effect as the above by writing the following instead: しかしながら、`switch(1)` が評価された場合には `secondAlternative` は `true` の値をもつことになる – 各ケース間には分岐がないため、実行は最後の `break` に続いていくことになる。これは `switch(1)` と `switch(2)` の両方が `secondAlternative` に関して同じ効果、すなわち `secondAlternative=true` となることを意味していて、したがって行 `secondAlternative=false` はまったく不必要であり、以下のように書くことで上記と同じ効果が達成できる:

```
oneOrTwo=0;

switch(i) {
  case 1:
  case 2:
    secondAlternative=true;
  break;
}
```

In the translator we insist that this second form is used and we reject switch statements in which there is some code between the case alternatives but no `break` at the end of that code. Switch statements in which the alternatives are separated by breaks are fully supported, however, so that, for example, the following is allowed: 翻訳においてはこの第2の形が用いられることを強調して、ケース選択肢間で書かれたコードの最後に `break` がないものは認めない。選択肢が分岐で分断されている Switch 文は完全にサポートされていて、例えばそのために、以下が許される:

```
one=0;
two=0;

switch(i) {
  case 1:
    one=1;
  break;
  case 2:
    two=1;
  break;
}
```

4.4 サポートされていない概念

In this subsection we present a few concepts where Java and VDM++ are compatible but which have not yet been incorporated into the Java to VDM++ translator. この節では、Java と VDM++ での互換性のある概念だが、Java から VDM++ の翻訳でまだ組み入れられていないものを、いくつか述べる。

4.4.1 同時並行性

The concurrency concepts in Java with the wait and notify mechanisms are similar to the notions for concurrency found in VDM++. However, currently the Java to VDM++ translator does not provide support for any of the concurrency features of Java including the synchronized statement. wait や notify 機構をもった Java の同時並行性の概念は、VDM++で見られる同時並行性に対する観念と似ている。しかし現時点での Java から VDM++ 翻訳では、同期文を含む Java 同時実行性に対するサポートは、全く提供されていない。

4.4.2 Unicode 文字

In Java it is possible to write arbitrary Unicode identifiers. This is not yet supported by the Java parser built into **VDMTools**. Java では任意の Unicode 識別子を書くことが可能である。**VDMTools**に組み入れられた Java 構文解析ツールでは、これはまだサポートされていない。

5 Java から VDM++ 翻訳の詳細

This section explains how individual elements of a Java program are translated to VDM++. この節では、Java プログラムの個々の要素がどのように VDM++ に翻訳されていくかを説明する。

5.1 組込型

The various kinds of integers in Java are all translated to the type ‘int’ in VDM++, and the various kinds of real numbers are all translated to the type ‘real’. Characters and booleans are translated to the types ‘char’ and ‘bool’ respectively. These transformations are summarised in the table in Figure 9. Java におけるさまざまな種類の整数はすべて、VDM++ では ‘int’ 型に翻訳され、実数はすべて ‘real’ 型に翻訳される。文字型とブール型はそれぞれ ‘char’ 型と ‘bool’ 型に翻訳される。これらの変換は 図 9 の表でまとめている。

Java	VDM++
byte	int
short	int
int	int
long	int
float	real
double	real
char	char
boolean	bool

図 9: 組込型変換

5.2 リテラル

Literal values belonging to the boolean, character and numeric types all have exact counterparts in VDM++ so are translated verbatim. JavaLangString literals

are translated to equivalent values belonging to the VDM++ class `JavaLangString` since the `JavaLangString` class in Java belongs to the API (see Section 5.3). Some examples of the translation of literals are given in the table in Figure 11. **11. ブール型、文字型、数値型に属するリテラル値はすべて、一語一語文字通りに VDM++ に翻訳される。** `JavaLangString` リテラルは、VDM++ クラスである `JavaLangString` に属する等価に翻訳されるが、Java の `JavaLangString` クラスは API (第 5.3 章参照) に属するからである。リテラル変換のいくつかの例題は図 11 の表に挙げてある。

Type	Java	VDM++
boolean	true, false	true, false
char	'A', 'B', ...	'A', 'B', ...
long/int/short/byte	255, -12, ...	255, -12, ...
float/double	1.1, 1e-5, ...	1.1, 1e-5, ...
JavaLangString	"abc..."	new JavaLangString("abc...")

図 10: Transformations of Literals

型	Java	VDM++
boolean	true, false	true, false
char	'A', 'B', ...	'A', 'B', ...
long/int/short/byte	255, -12, ...	255, -12, ...
float/double	1.1, 1e-5, ...	1.1, 1e-5, ...
JavaLangString	"abc..."	new JavaLangString("abc...")

図 11: リテラル変換

5.3 名称

Names in Java are translated to the same names in VDM++ with the following two exceptions: Java の名称は、以下の例外を除いて、VDM++ においても同じ名称に翻訳される:

1. where the name coincides with a VDM++ keyword two underscores are appended to the name in VDM++. Thus, for example, a Java method named 'bool' would become a method named 'bool__' in VDM++.
2. where the name represents a class which is part of the Java API, the VDM++ name is prefixed by the (Java) package name in which the class is defined. Thus, for example, the Java class `JavaLangObject` becomes the class `JavaLangObject` in VDM++.

1. 名称が VDM++ キーワードと一致する場合には、VDM++ での名称に 2 つのアンダーバーを付け加える。このことからたとえば、Java メソッド 'bool' ならば VDM++ でのメソッド名称が 'bool__' となる。
2. 名称が Java API の一部となるクラスを表す場合には、VDM++ での名称にクラス定義がなされた (Java) パッケージ名称を前に付け加える。このことからたとえば、Java クラス `JavaLangObject` は VDM++ での `JavaLangObject` クラスとなる。

The reason for the second property is that when we translate a Java class to VDM++ we add some extra functionality, for instance to simulate the `null` value (see Section 5.7). If we then want to translate our VDM++ back to Java we must generate Java code which corresponds to this added functionality because it is not included in the Java API classes, and this would give us two Java classes with the same name, the one already in the Java API and the one generated by translating the extended version of this from VDM++ back to Java. 2 番目の特性が必要な理由は、Java クラスから VDM++ へ翻訳を行う場合、たとえば `null` 値 (第 5.7 章を参照) をシミュレートするといったいくつか余分な機能追加を行うからだ。このときに VDM++ コードを Java に逆翻訳しようとする、この追加機能が Java API クラスに含まれないので、相当する Java コード生成が必要となり、したがって同じ名称の 2 つの Java クラス、1 つは Java API にあるもの、もう 1 つは VDM++ から Java への逆拡張版の翻訳で生成されるもの、を持つこととなる。

5.4 配列

An array in Java is translated to a map in VDM++, and accessing and modifying values in an array are written in terms of map application and map override. Some examples illustrating this are given in the table in Figure 12. Java の配列は VDM++ では写像に翻訳され、配列中の値へのアクセスや修正は、写像アプリケーションや写像上書きに関することとして記載される。これを描く例題いくつか、図 12 の表で与えられている。

Java	VDM++
Type a[]	map int to Type
... = a[0]	... := a(0)
a[0]=...	a:=a++{0 ->...}

図 12: 配列変換

Note that an array can be used as an instance of `java.lang.JavaLangObject` in Java but if this is done the VDM++ generated by the translator will contain a type error due to mismatching of types. Arrays should therefore not be used in this way in Java. 配列は Java では `java.lang.JavaLangObject` のインスタンスとして用いることができるが、この場合は、翻訳で生成される VDM++ に型不一致が原因の型エラーが含まれてしまうことに注意しよう。したがって、Java で配列のこのような使い方はすべきではない。

5.5 クラス

A Java class is translated to a VDM++ class of the same name, and the various elements of the class are translated as explained in the following subsections. Java クラスは同名の VDM++ クラスに翻訳され、クラスの各要素は以下の節に示すように翻訳される。

5.5.1 メソッドと要素

Methods become operations while members become instance variables, with static methods and members becoming static operations and instance variables. Methods whose result type is `void` become operations with no result in VDM++. This is illustrated in the following example: メソッドが操作となる一方で要素はインスタンス変数となるが、それに伴って静的なメソッドと要素が静的な操作とインスタンス変数になる。結果型が `void` であるメソッドは、VDM++ では戻り値のない操作となる。これは以下の例題で表わされている:

```
class A
{
  int i = 0;
  static int s = 1;

  int method(int)
  { return i; }
  static int smethod(int)
  { return i; }
  void op()
  {...}
  static int op1 (int j)
  {...}
}

class A is subclass of JavaLangObject
instance variables
  i: int := 0;
  static s : int := 1;
operations
  method : int ==> int
    ( return i );
  static smethod : int ==> int
    ( return i );
  op : () ==> ()
    ( ... );
  static op1 : int ==> int
    ( ... );
end A
```

Note that in Java a variable which is not explicitly initialised has a value by default whereas in VDM++ it is undefined. This can lead to run-time errors in VDM++ so care should be taken to ensure that instance variables in Java are initialised correctly where appropriate. Java では明白な初期化がなされなかった変数は既定による値を1つもつが、一方で VDM++ ではこれが未定義となる。これは VDM++ において実行エラーを導く可能性があり、Java におけるインスタンス変数は適切な箇所での確実な初期化を保証するように処理されるべきである。

5.5.2 継承

In Java, a class can inherit from other classes by extension or by implementation. VDM++ only supports a single form of inheritance, however, namely subclassing. Both forms of inheritance in Java are therefore translated to subclassing in VDM++. Thus, for example, the inheritance clause `class A extends B implements C,D` becomes `class A is subclass of B, C, D` in VDM++. Java では、1つのクラスは機能拡張あるいは実装によって他のクラスから継承を行うことが可能だ。VDM++ では継承の単純な形式、サブクラス化と名づけているが、これをサポートしているだけである。したがって Java における継承の両形式は、VDM++ ではサブクラス化へと翻訳される。このようにたとえば、VDM++ における継承節 `class A extends B implements C,D` は VDM++ において `class A is subclass of B, C, D` となる。

Java classes which have no explicit inheritance clause are assumed to inherit implicitly from `java.lang.Object`. The translator makes this dependency explicit, so that in VDM++ such a class explicitly inherits from `JavaLangObject`, the VDM++ counterpart of `java.lang.Object`. The example in Section 5.5.1 illustrates this. 明白な継承節をもたない Java 節は、暗黙に `java.lang.Object` から継承すると仮定される。翻訳ではこの依存性を明白にして、そのために VDM++ ではこのようなクラスは明白に `java.lang.Object` の VDM++ の対応部分である `JavaLangObject` から継承を行う。第 5.5.1 章の例題がこれを描写している。

Note that the fact that both extension and implementation in Java are translated to subclasses in VDM++ can cause a problem if the VDM++ is translated naively back to Java since VDM++ subclasses are by default translated to extension in Java. The manual for the VDM++ to Java Coce Generator [2] explains how to avoid this problem. Java の機能拡張と実装の両方が VDM++ でのサブクラスに翻訳されるという実際からは、もし VDM++ が単純に Java へ逆翻訳されたならば、VDM++ のサブクラスは既定として Java の機能拡張に翻訳されるために、問題を起こす可能性があることに注意しよう。VDM++ から Java のコード生成 [2] で、この問題をどのように避けるか説明している。

5.5.3 クラス修飾子

Modifiers of a Java class (e.g. `abstract`, `final`) have no counterpart in VDM++ and are all ignored by the translator. Java クラスの修飾子 (例えば `abstract`, `final`) は VDM++ では同等のものがいないため、翻訳ですべてが無視される。

5.5.4 アクセス修飾子

Access modifiers are translated directly from Java to VDM++ as shown in the table in Figure 14. Note, however, that in Java the absence of a modifier indicates that the construct can be accessed by any class in the same package, whereas in VDM++ a construct with no access modifier is assumed to be private. Omitting access modifiers in Java can thus give rise to access violation errors when type checking the translated VDM++. アクセス修飾子は 図 14 の表で示すよ

うに、Java から VDM++ へそのまま翻訳される。しかし Java で修飾子がない場合に、構造体は同じパッケージ内の任意のクラスからアクセス可能で、一方で VDM++ ではアクセス修飾子をもたない構造体は非公開であると仮定される。Java でアクセス修飾子を省くと、翻訳された VDM++ を型検査するときアクセス違反エラーを引き起こす可能性が生じる。

Java	VDM++
no modifier	no modifier
public	public
protected	protected
private	private

図 13: Transformations of Access Modifiers

Java	VDM++
修飾子なし	修飾子なし
public	public
protected	protected
private	private

図 14: アクセス修飾子変換

5.5.5 静的初期化子

Static initialisers are supported by the translator although they have no direct counterpart in VDM++. To implement these, the Java code for all static initialisers in a class is collected together and translated into the body of a static operation called `j2v_staticInitializer`. This operation takes no inputs and its result is the special VDM++ quote value `<VOID>`. In addition, a static instance variable called `dummy` is added to the VDM++ class, the type of which is this same quote value `<VOID>`, and the operation `j2v_staticInitializer` is defined

as the initialisation of this variable. Thus, when the VDM++ class is initialised, the operation `j2v_staticInitializer` is invoked and simulates the effects of the original static initialisers in the Java class. 静的初期化子は、VDM++には直接対応するものがないが、翻訳においてサポートされている。これらを実装するために、クラス内のすべての静的初期化子に対する Java コードは集められ一緒に翻訳されて、`j2v_staticInitializer` と呼ぶ静的操作の本体をつくる。この操作で入力値はとらず、結果は特殊 VDM++ 引用句である `<VOID>` である。それに加えて、型がこの同じ引用句 `<VOID>` である `dummy` という静的インスタンス変数が VDM++ クラスに加えられ、操作 `j2v_staticInitializer` がこの変数の初期設定として定義される。このように VDM++ クラスが初期化されると、操作 `j2v_staticInitializer` は起動され、Java クラスのもともとの静的初期化子の結果はシミュレートされる。

The following example illustrates this. 以下の例題がこれを示す。

```
                                instance variables
static int i=1;                static i : int  := 1;
static int j=2;                static j : int  := 2;
static int k=3;                static k : int  := 3;

static {
    i=k+j;
}

int l;                          l : int ;

static {
    j=i+k;
}                                static dummy : <VOID> := j2v_staticInitializer()

                                operations
                                static j2v_staticInitializer : () ==> <VOID>
                                j2v_staticInitializer() ==
                                ( i := k+j;
                                  ( j := i+k
                                    ) ;
                                  return <VOID>
                                ) ;
```

5.5.6 クラスについての情報取得

In Java every class inherits from `java.lang.Object`. This includes the method `getClass` which returns an instance of `java.lang.Class`, and this in turn provides functionality for obtaining various information about the class, e.g. the class name, whether the class represents an interface type or a primitive Java type, etc. Java では全クラスが `java.lang.Object` から継承を行う。これは `java.lang.Class` のインスタンスを返すメソッド `getClass` を含むもので、言い換えればクラスについての様々な情報を得るための機能、たとえば、クラス名称や、クラスがインターフェイス型に相当するかあるいは基本的 Java 型であるか、等々、を提供してくれる。

The translator simulates part of this functionality using the class `JavaLangClass` which is defined as follows: 翻訳においては、以下のように定義されたクラス `JavaLangClass` を用いて、この機能の一部をシミュレートする:

```
class JavaLangClass ...
types
  CLASS ::
    name : seq of char
    cori : <CLASS> | <INTERFACE>
    isPrim : bool

instance variables
  private val : JavaLangClass 'CLASS;

operations

public JavaLangClass : seq1 of char *
  (<CLASS> | <INTERFACE>) * bool ==> JavaLangClass
JavaLangClass(name, cori, prim) ==
( val.name := name;
  val.isPrim := prim;
  val.cori := cori;
);

public getName : () ==> JavaLangString
getName() ==
  return new JavaLangString(val.name);

public isArray : () ==> bool
```

```
isArray() ==
  return false;

public toString : () ==> JavaLangString
toString() ==
( dcl
  str: seq of char :=
    if isInterface()
    then "interface "
    else
      if isPrimitive()
      then ""
      else "class ";
  str:=str^getName().toSeqOfChar();
  return new JavaLang(str)
);

public isInterface : () ==> bool
isInterface() ==
  return val.corl=<INTERFACE>;

public isPrimitive : () ==> bool
isPrimitive() ==
  return val.isPrim;

...

end JavaLangClass
```

A constant `CLASS`, whose value is an appropriate instance of this class, together with a (public) method `getClass` which returns the value of this constant, are then added to each generated VDM++ class. This is illustrated by the example below. 定数 `CLASS` は、その値がこのクラスに適合したインスタンスであり、この定数の値を返す (public) メソッド `getClass` と共に、各 VDM++ クラスが生成されたときに付加される。これは以下の例題で表わされる。

```
class A ...
values
  CLASS : JavaLangClass = new JavaLangClass("A", <CLASS>, false)
...
```



```
public getClass : () ==> JavaLangClass
getClass() ==
return CLASS;

end A
```

5.6 インターフェイス

Interfaces in Java are translated to classes in VDM++. However, only the signatures of the methods in a Java interface are given, while in VDM++ a method must always have a body. The translator therefore sets the bodies of methods in Java interfaces to `is not yet specified` in VDM++. Java のインターフェイスは VDM++ のクラスに翻訳される。しかし Java インターフェイスでメソッドのシグニチャのみが与えられる一方で、VDM++ ではメソッドは常に本体をもたなければならない。したがって翻訳では、Java インターフェイスのメソッド本体は VDM++ での `is not yet specified` に設定する。

It might seem more natural to use `is subclass responsibility` here instead of `is not yet specified`. However, this does not work because of the way the value `null` is translated (see Section 5.7). To see this, consider the following Java example in which the value `null` is used in conjunction with objects of an interface type: ここでは、`is not yet specified` の代わりに `is subclass responsibility` を用いる方がより自然であろう。しかし値 `null` が翻訳される方式のせいで、これは機能しない (第 5.7 章参照)。この理解を行うために、1 つのインターフェイス型のオブジェクトと結びついて値 `null` が用いられる以下の Java 例題を考えよう:

```
interface IFace {...}
...
IFace iface = null;
```

In this case the translator would translate `null` to `new IFace(<NIL>)` as explained in Section 5.7, which includes an instantiation of the class `IFace`. This means that the class `IFace` cannot be abstract (otherwise this instantiation does not make sense), and thus that we cannot use `is subclass responsibility` to represent the bodies of interface methods. この場合の翻訳では、第 5.7 章で述

べるように `null` を `new IFace(<NIL>)` に変換するが、`IFace` クラスの実体化を含むものである。これはクラス `IFace` が概念上のものではないはずということ(そうでなければこの実体化は意味をなさない)、したがってインターフェイスメソッドの本体を表すのに `is subclass responsibility` を用いることはできないことを意味する。

Recall also that translating an interface class from Java to VDM++ and then re-translating the VDM++ back to Java can cause problems. See the discussion at the end of Section 5.5.2 for details. インターフェイスクラスを Java から VDM++ へ翻訳し、その後逆に VDM++ から Java に再び翻訳することが、問題の原因となる可能性を思い出そう。詳細は、第 5.5.2 章の最後の記述を参照すること。

5.7 null

In Java it is possible to overload operations and invoke a particular one of these operations with the (polymorphic) value `null` by casting the value to the appropriate input type as in the examples below: Java では下記の例題にあるように、(多様型の) 値 `null` が適切な入力型となるようにその値をキャストすることによって、操作の上書きをし、またそれら操作の特定の 1 つを起動することも可能である:

```
void op(A a)
```

```
void op(B b)
```

```
op((A)null)
```

```
op((B)null)
```

In VDM++ there is no polymorphic constant which belongs to all types and which can be coerced to a specific class by tagging it with the name of that class. (The closest is perhaps the value `nil`, which belongs to all optional types, though this cannot be coerced to any one type.) VDM++ では、すべての型に属しつつ、そのクラス名称のタグ付けで特定クラスに振られるといった、多様型の定数は存

在しない。(最も近い値はたぶん `nil` で、これはすべての任意型に属し、しかもどのような型にも振られることはない。)

Instead, therefore, we make every translated VDM++ class a subclass of a special class `Nullable` (by defining the class `JavaLangObject` to be a subclass of the class `Nullable`). The (boolean valued) instance variable `isNil` in the class `Nullable` is then used to indicate whether or not a particular object belonging to the class corresponds to the Java value `null`. したがってこの代わりに、すべての翻訳済みの VDM++ クラスを特定クラス `Nullable` のサブクラスとする(クラス `JavaLangObject` をクラス `Nullable` のサブクラスと定義することにより)。したがって、クラス `Nullable` の(ブール値である)インスタンス変数 `isNil` は、Java 値である `null` に相当するクラスに属する特定オブジェクトかそうでないかを示すのに用いられる。

```
class JavaLangObject is subclass of Nullable
...

class Nullable
instance variables
  public isNil: bool := false
operations
  public IsNil: () ==> bool
  IsNil() == return isNil
end Nullable
```

In addition, we introduce a new special value `<NIL>` into the VDM++ specification and generate an additional constructor for each VDM++ class which creates instances which represent the `<NIL>` value of that class. This is shown in the following example: 加えて、VDM++ 仕様に新しい特定値 `<NIL>` を導入して、各 VDM++ クラスに対してそのクラスの `<NIL>` 値を表すインスタンスをつくる追加のコンストラクタを生成する。これは以下の例題でみることができる:

```
ClassName: <NIL> ==> ClassName
ClassName() ==
  isNil := true;
```

To simulate the `null` value for arrays we use the empty map. 配列に対する `null` 値をシミュレートするために、空写像を用いる。

The examples in the table in Figure 15 illustrate how `null` is translated in different contexts. 図 15 にある表の例題は、`null` が異なる文脈でどのように翻訳されるかを表している。

Java	VDM++
<code>ClassType o = null;</code>	<code>dcl o: ClassType := new ClassType(<NIL>)</code>
<code>ClassType o[2] = null;</code>	<code>dcl o: map int to ClassType := {new ClassType(<NIL>) i in set {0,...,1}}</code>
<code>type o[] = null;</code>	<code>o := { ->}</code>
<code>op(null);</code>	<code>op(new ClassType(<NIL>))</code>
<code>o == null</code>	<code>o.IsNil()</code>
<code>o != null</code>	<code>not o.IsNil()</code>

図 15: `null` 変換

It is important to note here that this treatment of `null` can lead to endless loops at the VDM++ level. Consider the following example in Java: `null` のこの取扱いは、VDM++ レベルで終わりのないループを導く可能性があることを、注意しておくことが大切だ。Java の以下の例題を考えてみよう:

```
class Database
{
    Database db = null ;
}
```

This would be translated to the following in VDM++: これは VDM++ では以下のように翻訳されるだろう:

```
class Database is subclass of JavaLangObject
instance variables
    db : Database = new Database(<NIL>) ;
```

which contains an infinite loop – each instantiation of the class Database invokes another instantiation in order to set the value of its instance variable. The same problem can also occur indirectly as in the following example where the Database class has an instance variable of type Controller and vice versa: これは無限ループを含んでいる – クラスデータベースの各初期化が、そのインスタンス変数の値を設定するために、もう1つの初期化を起動するのである。Database クラスが Controller 型のインスタンス変数を持ち、またその逆でもあることで、以下の例題のように同じ問題が間接的に起きる可能性もある: 以下の例題では、Database クラスが型 Controller のインスタンス変数をもっているし逆にもたれてもいる:

JAVA:

```
class Controller
{
    Database database;
    Controller()
    {
        database = null;
    }
    static public Controller getController()
    {
        return new Controller();
    }
}

class Database
{
    Controller
        controller = Controller.getController();
}
```

VDM:

```
class Controller is ...
instance variables
    database: Database;
operations
    Controller: () ==> Controller
        database := new Database(<NIL>);

    static public getController: () ==> ()
```

```
    getController() ==  
        return new Controller(<NIL>);  
end Controller  
  
class Database is ...  
instance variables  
    controller: Controller := Controller.getController();  
end Database
```

Care should therefore be taken to avoid such situations in the Java code. したがって Java コード中で、このような状態をさけるための注意が払われるべきである。

5.8 式

In Java it is possible to write an expression with side effects such as $f(n) - mm[n++]$. This feature has no direct counterpart in VDM++, and in general a sequence of VDM++ statements is required in order to obtain something semantically equivalent. We deal with this by introducing a **def** expression and generating local names within this to store the values of the various components of the expression. This is best illustrated by an example. Java では、 $f(n) - mm[n++]$ といった副次的作用をもつ式を書くことができる。この特質は VDM++ でちょうど一致するものではなく、一般的にも意味的に同等のものを得るためには、VDM++ 文の連続が必要となる。式の様々な構成要素の値を保存するために **def** 式を導入して、この中に局所的な名称を生成することで、これを扱う。例題を用いるのが最もわかりやすい。

Consider the following fragment of an expression which has side effects in Java: 以下のような、Java で副次的作用をもつ式の断片を考えよう:

```
... f(n)-mm[n++] ...
```

This expression is part of an enclosing expression, so in the VDM++ we introduce a **def** expression and the pattern name `l_1` to record its value. この式は囲った式の一部であり、VDM++ ではこの値を記録するために、**def** 式とパターン名称 `l_1` を導入する。

The expression itself is a binary expression, so two more pattern names `l_2` and `l_3` are introduced to store the values of its left and right operands respectively and `l_1` is defined appropriately in terms of `l_2` and `l_3`. 式それ自体は2進式で、その左右両辺の値をそれぞれ保存するために、さらに2つのパターン名称 `l_2` と `l_3` が導入され、そして `l_2` と `l_3` の点から見て `l_1` が適当なものとして定義される。

Note that `l_2` is only needed because the right operand has side effects. We can not define `l_1` simply as `f(n) - l_3` because its definition is preceded by the statements generated for the right operand which change `n`. 右辺が副次作用をもつために、`l_2` のみが必要とされていることに注意しよう。`l_1` を単純に `f(n) - l_3` として定義することはできない、というのはこの定義では、`n` を変える右辺のために生成された文が、前に置かれているからである。

The final result is as follows: 最終結果は以下の通り:

```
def l_2 = f(n);
l_4 = n;
l_5 = l_4
in ( n := l_4 + 1;
    def l_3 = mm(l_5);
    l_1 = l_2 - l_3
    in ... l_1 ...
) ;
```

Expressions like `new ClassType(...)`; and `op()`; can also be used as statements in Java, and it is possible to write sequences of such expressions as in the following example: `new ClassType(...)`; や `op()`; といった式は、Java では文としても用いられ得るし、以下の例題のようにこのような式の連続を書くことが可能である:

```
new A();
op1();
op2();
```

In Java, if an expression in such a sequence returns a result that result is ignored and the execution passes to the next expression in the sequence, but the semantics of VDM++ do not match this and instead state that as soon as the

execution encounters an expression that returns a result the execution terminates and that result becomes the result of the whole sequence. When translating such expressions, therefore, we need to avoid this problem. This is done by using a `let` statement to assign the result to a dummy (in fact unnamed) variable and making the body of the `let` statement the identity statement `skip`. The translation of the Java example above is thus: Java では、このような連続の式が結果を返すときは、その結果は無視されて実行は連続の次の式に渡されるが、VDM++ の意味論はこれとは一致せず、かわりに実行が結果を戻す式に出会うやいなや実行は終了し、結果は全連続の結果となる。したがって、このような式を翻訳するときは、この問題をさける必要がある。let 文を用いて結果をダミーに代入することで、対処する。上記 Java 例題の翻訳はこのようになる:

```
let - = new A()
in
  skip ;
let - = op1()
in
  skip ;
let - = op2()
in
  skip ;
```

5.9 文

5.9.1 If 文

The translation of `if` statements is straightforward. The component parts are simply translated as illustrated below: `if` 文の翻訳はそのまま行われる。構成部分は、以下に示されるように素直に翻訳される:

JAVA:

```
if condition
then thenStmt
else elseStmt
```

VDM:


```
if translatedCondition
then translatedThenStmt
else translatedElseStmt
```

5.9.2 Block 文

A block in Java becomes a block statement in VDM++. The Java local variable declarations become `dcl` statements within the block and the body of the block in Java becomes the body of the block statement in VDM++. The following example illustrates this: Java のブロックは、VDM++においてブロック文となる。VDM++において Java の局所変数宣言は、ブロック内での `dcl` 文となり、Java のブロック本体は ブロック文の本体となる。以下の例題ではこれを表わしている:

JAVA:

```
{ int b, c; int d, e=0; ... }
```

VDM:

```
( dcl b : int ,
    c : int ,
    d : int ,
    e : int := 0;
    ...
) ;
```

5.9.3 for 文

A `for` statement in Java is translated to a block statement in VDM++ in which the loop variables are introduced in `dcl` statements and the body is a `while` loop. This is illustrated by the following example: Java の `for` 文は VDM++ のブロック文に翻訳されるが、ここでループ変数が `dcl` 文中に導入され本体は `while` ループとなる。これは以下の例題で表される:

JAVA:

```
for(int i=0, j=10; i<10; i++, j--) {  
    ...  
}
```

VDM:

```
( dcl    i : int    := 0,  
      j : int    := 10;  
  while i < 10 do  
    ( ...  
      i := i+1;  
      j := j-1  
    )  
  ) ;
```

5.9.4 while 文

The translation of while loops is straightforward except where the loop termination expression has side effects in which case it is treated in the way described in Section 5.8 both before the loop is entered and at the end of each iteration. This is illustrated in the following example: while ループの翻訳は、ループ終了式が副次作用をもつ場所では、ループに入る前と各繰り返しの最後との両方で、Section 5.8 に記されている方法で取り扱われるが、それ以外の場所ではそのまま素直な方法で取り扱われる。これは以下の例題に表されている:

JAVA:

```
while(mm[i++]>0) {  
    ...  
}
```

VDM:

```
( dcl    l_7 : bool ;  
  def l_5 = i;  
  l_4 = l_5  
  in ( i := l_5+1;  
      def l_2 = mm(l_4);
```

```
    l_1 = l_2 > 0
    in l_7 := l_1
  ) ;
while l_7 do
( ...
  def l_5 = i;
  l_4 = l_5
  in ( i := l_5+1;
    def l_2 = mm(l_4);
    l_1 = l_2 > 0
    in l_7 := l_1
  )
)
);
```

5.9.5 do while 文

To translate a do while statement of the form 次の形式の do while 文を翻訳するとする

```
do {
    i++; ...
} while(mm[i]>0);
```

we convert it to the equivalent while statement まずこれと同等な while 文に変換する

```
( i := i+1; ...
) ;
while mm(i) > 0 do
( i := i+1; ...
) ;
```

and translate this while statement. そしてこれを while 文に翻訳する。

5.9.6 switch 文

A `break` in Java is translated to an exception throw in VDM++ because it can appear anywhere inside an alternative. The following example, in which there is a `break` in case 1:, illustrates this. Java の `break` は VDM++ の例外処理実行で翻訳されるが、選択肢内どこにでも現れる可能性があるからだ。以下の例題で、case 1: に `break` があるが、これを表している。

JAVA:

```
switch(a) {
  case 0:
    alternative1
  break;
  case 1:
    if(b<5) break;
    alternative2
  break;
  default:
    alternative3
  break;
}
```

VDM:

```
trap j2v_break with
  cases true :
    (isofclass (J2V_Break,j2v_break)) ->
      skip

end

in
  ( cases a:
    (0) ->
      ( translatedAlternative1
        exit new J2V_Break()
      ) ,
    (1) ->
      ( if b<5 then exit new J2V_Break();
        translatedAlternative2
        exit new J2V_Break()
      ) ,
```

```
    others ->
      ( translatedAlternative3
        exit new J2V_Break()
      )
    end
  ) ;
```

5.9.7 try catch 文

A Java try catch statement is translated to a combination of an `always` statement and a `trap` statement as illustrated in the following example: Java の try catch 文は、以下の例題に表されるように `always` 文と `trap` 文 との結合に翻訳される:

JAVA:

```
class B extends JavaLangException { ... }

try { tryBody }
catch(B b) { catchBbody }
catch(JavaLangException e) { catchEbody }
finally{ finallyBody }
```

VDM:

```
class B is subclass of JavaLangException
...
end B

always
  ( translatedFinallyBody
  )
in
  trap j2v_exception : JavaLangException with
    cases true :
      (isofclass(B, j2v_exception) and not j2v_exception.IsNil()) ->
        ( dcl b : B := j2v_exception;
          translatedCatchBbody
        ) ,
      (isofclass(JavaLangException, j2v_exception) or
        j2v_exception.IsNil()) ->
```

```
      ( decl b : JavaLangException := j2v_exception;  
        translatedCatchEbody  
      ) ,  
    others ->  
      exit j2v_exception  
  end  
in  
  translatedTryBody
```

6 VDM++ 変換群

The transformations which are currently supported apply to certain kinds of expressions (binary expressions) and certain kinds of statements (if statements, block statements, and while loop statements). The specific transformations are listed in Table 16, the first column of which shows the name of the transformation and the second the kind of expression or statement to which it applies. A brief explanation and an example of each transformation is given below. 現在サポートされている変換は、特定の式 (2 進式) と特定の文 (if 文、block 文、while ループ文) に適用される。具体的な変換は表 16 に一覧されていて、その最初の欄には変換の名称が示され、第 2 欄は適用される式または文の種類が示されている。短い説明と各変換例は以下に与えられている。

名称	適用
isMapCompLoop	Block Stmt
ifTestTrue	Block Stmt
ifTestFalse	Block Stmt
isRedundantIfBlock	Block Stmt
ifToAnd	Block Stmt
isRedundantIfBlockNoElse	Block Stmt
isRedundantDcl	Block Stmt
ifToCases	If Stmt
ifToEquiv	If Stmt
nestedIfsNoElses	If Stmt
whileIfTestTrue	While Loop
orToNotEquiv	Binary Expr

図 16: 変換一覧

The transformations are attempted in the order in which they appear in the table, so in situations in which two different transformations could in principle be applied to the same construct only the first of these is actually applied. In addition, the transformations are only applied once. This means that there could still be places in the VDM++ output where transformations could be applied, for example when one transformation transforms the specification into something which matches another transformation. 変換は、表に現れる順に試みられるの

で、2つの異なる変換が同じ構築物に適用される状況では、原則的にはその最初の方のみが実際に適用される。さらに、変換の適用は一度だけとなる。これはVDM++の出力にさらに変換が適用され得ることを意味していて、例えばある変換が別の変換と合致するような仕様の変化を行うときである。

6.1 isMapCompLoop (Block Stmt)

Replaces stepwise construction of a map using a while loop with a map comprehension expression. Can deal with increasing and decreasing loop variables, inclusion or exclusion of equality in the test of the while loop, and arbitrary step sizes for the loop variable. Also allows arbitrary specification to appear after the loop. 写像の段階的な構築を while ループを用いて写像包含式に置き換える。ループ変数の増加減少、while ループテストでの相等の包含または排除、ループ変数の任意の刻み幅、を扱うことができる。さらに、ループ後に現れる任意の仕様は許されている。

As an example, the following specification involving a while loop 例題として、以下は while ループを含む仕様である

```
public Test : () ==> map nat to [nat]
Test () ==
  ( dcl m : map nat to [nat] :=
    {l |-> nil | l in set {0,...,12}};
    dcl b : bool := true;
    (dcl i : int := 0;
     while i < 12 do
       ( m := m ++ {i |-> f(i)};
        i := i + 1 )
     );
    return m);
```

would transform to this specification involving a map comprehension 写像包括を含めたこのような仕様への変換がなされるであろう

```
public Test : () ==> map nat to [nat]
```



```
Test () ==
  ( dcl m : map nat to [nat] :=
    { l |-> nil | l in set {0,...,12} },
    b : bool := true;
  ( dcl i : int := 0;
    m := m ++ { i |-> f(i) |
      i in set {0, ..., 12 - 1} }
  );
  return m);
```

6.2 ifTestTrue (Block Stmt)

Transforms a block which begins 次に始まるブロックを変換する

```
( dcl ... , x : bool := true, ... ;
  if x then y else z;
  ...
)
```

into one which begins 次に始まるようなものとなる

```
( dcl ... , x : bool := true, ... ;
  y;
  ...
)
```

For example, the following specification たとえば、以下の仕様は

```
public Test : () ==> nat
Test () ==
  def n : nat = 5 in
    ( dcl x : bool := true;
      if x then return n else return 0 );
```

is transformed to 次のように変換される

```
public Test : () ==> nat
Test () ==
  def n : nat = 5 in
    ( dcl x : bool := true;
      return n );
```

6.3 ifTestFalse (Block Stmt)

Transforms a block which begins 次に始まるブロックがある

```
( dcl ... , x : bool := false, ... ;
  if x then y else z;
  ...
)
```

into one which begins これは次のようになる

```
( dcl ... , x : bool := false, ... ;
  z;
  ...
)
```

For example, the following specification たとえば、以下の仕様がある

```
public Test : () ==> nat
Test () ==
  def n : nat = 5 in
    ( dcl x : bool := false;
      if x then return n else return 0 );
```

is transformed to 変換されると次のようになる

```
public Test : () ==> nat
Test () ==
  def n : nat = 5 in
    ( dcl x : bool := false;
      return 0 );
```

6.4 isRedundantIfBlock (Block Stmt)

Replaces a block statement of the form 次の構造のブロック文を置き換える

```
( dcl ..., b : bool := false, ... ;
  if test
  then
    ( b := true
    )
  else
    ( b := false
    ) ;
  return b
) ;
```

with

```
return test;
```

The block can be inside another block and the boolean type can be an indirect reference as in the following example ブロックはもう1つのブロックの中に置くことができるし、ブール型は以下の例題にあるように間接的な参照となる

```
types T = bool;
```

```
operations
```

```
public Test : () ==> bool
Test() ==
  ( dcl b : seq of char;
    ( dcl b : bool := false;
      ( dcl b : T := true;
        dcl c : bool := b;
        if c
        then (b := true)
        else (b := false);
        ((return b;))
      );
    return b
  );
);
```

which transforms to これは次のように変換される

```
types T = bool;
```

```
operations
```

```
public Test : () ==> bool
Test() ==
  ( dcl b : seq of char;
    ( dcl b : bool := false;
      ( dcl b : T := true,
        c : bool := b;
        return c
      );
    return b
  );
);
```

6.5 isRedundantIfBlockNoElse (Block Stmt)

Analogous to the above transform except that it applies to the case when the if statement has no else clause. Thus, for example, the following specification if 文が else 節を持たない場合に適用されるという以外は、上記変換に類似している。このように、たとえば以下のような仕様がある

```
public Test : () ==> bool
Test () ==
  ( dcl a : bool;
    ( dcl b : bool := false;
      ( dcl c : bool := false;
        if (2 > 1)
          then (b := true)
        );
      return b
    );
  );
```

is transformed to これは次のように変換される

```
public Test : () ==> bool
Test () ==
  ( dcl a : bool;
    ( dcl b : bool := false;
      ( dcl c : bool := false;
        return (2 > 1)
      );
    );
  );
```

6.6 ifToAnd (Block Stmt)

Transforms a consecutive pair of statements of the form 次の形式の 2 文の連続を変換する

```
if a then return b;  
return false;
```

to the single statement **これを 1 文にする**

```
return a and b;
```

Redundant blocks are also removed as in the following example **冗長ブロックもまた、以下の例題にあるように、取り除かれる**

```
public Test : () ==> bool  
Test () ==  
  ( dcl x : nat := 12;  
    ( if x < 3 then return x > 3);  
    ((return false)))  
  
  );
```

which is transformed to **これは次のように変換される**

```
public Test : () ==> bool  
Test () ==  
  ( dcl x : nat := 12;  
    return x < 3 and x > 3  
  );
```

6.7 isRedundantDcl (Block Stmt)

Replaces a block statement of the form **次の形式のブロック文がある**

```
( dcl x : A := y;  
  return x  
);
```

with the return statement **これを return 文で置き換える**

```
return y;
```

Thus, for example, the following specification **たとえば、以下のような仕様がある**

types

```
public T = (nat * nat)
```

operations

```
public Test : () ==> T
Test () ==
  (dcl r : T := def mk_(a,2) = mk_(1,2) in mk_(a,2);
   return r);
```

is transformed to **これは次のように変換される**

types

```
public T = (nat * nat);
```

operations

```
public Test : () ==> T
Test() ==
  return def mk_(a,2) = mk_(1,2) in mk_(a,2);
```

6.8 ifToCases (If Stmt)

Transforms a multiply nested if statement of the form **次の形式の多重ネストされた if 文を変換しよう**

```
if x = a1
then r1
else if x = a2
    then r2
    else if x = r3
        then r3
        else ....
```

to a cases statement of the form 次の形式の cases 文に対して

```
cases x :
  a1 -> r1,
  a2 -> r2,
  a3 -> r3,
  ...
```

At the same time, redundant blocks are removed. Thus, for example, the operation 同時に、冗長なブロックが取り除かれる。たとえば、次のような操作がある

```
public Test : () ==> nat
Test () ==
( dcl x : nat := 5;
  if x = 1
  then return 1
  else if (x = 2)
    then ((return 2))
    else if (((x = 3)))
      then return (3)
      else if x = 4
        then (return 4)
        elseif (x = 5)
          then return 5;
  return x);
```

is transformed to これは次のように変換される


```
public Test : () ==> nat
Test() ==
  ( dcl x : nat := 5;
    cases x :
      1 -> return 1,
      2 -> return 2,
      3 -> return 3,
      4 -> return 4,
      5 -> return 5
    end;
  return x);
```

6.9 ifToEquiv (If Stmt)

Transforms an if statement of the form 次の形式の if 文を変換する

```
if a then (if b then x else y) else (if b then y else x)
```

to the form 次の形式になる

```
if a <=> b then x else y
```

As an example, the following operation 例題として、以下の操作があるとする

```
public Test : nat * nat ==> nat
Test (x, y) ==
  ( if x > 2
    then ( if y < 1 then return 1 else return 2 )
    else ( if y < 1 then return 2 else return 1 )
  );
```

would be transformed to これは次のようになるだろう

```
public Test : nat * nat ==> nat
```

```
Test (x, y) ==  
  ( if x > 2 <=> y < 1  
    then return 1  
    else return 2  
  );
```

6.10 nestedIfsNoElses (If Stmt)

Transforms a double if statement of the form 次の形式の 2 つの if 文を変換する

```
if a then (if b then c)
```

to a single if statement

```
if a and b then c
```

Thus, for example, the operation たとえば、次のような操作がある

```
public Test : int * int ==> bool  
Test (x, y) ==  
  ( for i in [1, 2, 3] do  
    ( if x > i then if i > y then return false );  
    return true;  
  );
```

would be transformed to 次のように変換されるだろう

```
public Test : int * int ==> bool  
Test (x, y) ==  
  ( for i in [1, 2, 3] do  
    ( if x > i and i > y then return false );  
    return true;  
  );
```

6.11 whileIfTestTrue (While Loop)

Simplifies an if statement which occurs at the beginning of a while loop and which has the same test as the while loop. In general, the transformation converts a while loop of the form while ループの最初に現れて、while ループと同じ判定を行う、そのような if 文を単純化する。これは一般的に、次の形式の while ループの変換である

```
while test do
  ( if test then x else y; ... )
```

to one of the form これが次のようになる

```
while test do
  ( x; ... )
```

For example, the following operation たとえば、以下の操作は

```
public Test : nat * int ==> nat
Test (x, y) ==
  ( while x > 5 do
    ( if x > 5 then y := y + 1 else y := y - 1;
      x := x - 1 );
    return y;
  );
```

is transformed to 次のように変換される

```
public Test : nat * int ==> nat
Test (x, y) ==
  ( while x > 5 do
    ( y := y + 1;
      x := x - 1 );
    return y;
  );
```

6.12 orToNotEquiv (Binary Expr)

Transforms an or expression of the form 次の形式の and / or 式を変換する

```
not a and b or a and not b
```

into the logically equivalent form 結果を次のような論理的に同等な形式とする

```
not (a <=> b)
```

At the same time, redundant brackets are removed. Thus, for example, the operation 同時に、冗長な括弧は取り除かれる。このように、たとえば次の操作は

```
public Test : () ==> bool
Test () ==
  ( return ((not (let a = true in a) and (false)) or
            (((((let a = true in a) and not (false))))))
  );
```

is transformed to 次のように変換される

```
public Test : () ==> bool
Test () ==
  return (not ((let a = true in a) <=> false));
```

7 Java API クラス群の VDM++ モデル

The Java to VDM++ Translator is supplied with `*-.java` files containing skeletons of a subset of Java API classes. They might need to be in your project. At least you need `java.lang.JavaLangObject` and all classes it references. The Java to VDM++ Translator は、Java API クラスのサブセットのスケルトンを含む `*-.java` ファイルと共に提供される。これらは、プロジェクト内に入れておく必要があるだろう。少なくとも、`java.lang.JavaLangObject` とそれが参照するすべてのクラスは必要である。

There is no need to translate them to VDM++ because the Java to VDM++ translator is also equipped with their VDM++ counterparts. A Java API class is represented at the VDM++ level as a plain VDM++ class or a dynamically linkable class (`dlclass`). Implementation of `dlclasses` is housed in `j2vd11.so`. それらを VDM++ に翻訳する必要はない、というのは the Java to VDM++ translator もまた VDM++ に一致する部分を備えている。Java API クラスは VDM++ レベルでは、単純 VDM++ クラスまたは動的リンク可能クラス (`dlclass`) として標識される。動的リンク可能クラスの実装は、`j2vd11.so` に格納されている。

Java API is only partially covered at the VDM++ level. Some methods could not be translated into VDM++ because they use Java features that are not available in VDM++. For example, the method `JavaUtilVector.copyInto` has an array as an output parameter which is impossible in VDM++ because an array is a map and a map is passed by value. There are also Java API classes which have overloaded methods which cannot be translated because their signatures will coincide (e.g. `JavaLangString.valueOf(int)` and `JavaLangString.valueOf(long)`). Java API は、部分的に過ぎないが VDM++ レベルで保護されている。いくつかの方法は VDM++ で翻訳可能にはならない、これは VDM++ では利用できない Java 特性を用いているからである。たとえばメソッド `JavaUtilVector.copyInto` は出力パラメータとして配列をもつが、これは VDM++ では不可能であり、配列は写像で写像は値によって渡されるからである。また Java API クラスで、それらのシグニチャが一致してしまうために上書きされてしまうメソッドをもつ、翻訳が不可能な Java API クラスもある (たとえば `textttJavaLangString.valueOf(int)` や `JavaLangString.valueOf(long)`)。

The following is the list of VDM++ classes with their methods which implement a subset of Java API. This documents gives only the list. Actual documentation

can be found in the Java documentation. 以下は、Java API のサブセットを実装するメソッド群をもつ、VDM++ クラスの一覧である。本書では、一覧表のみ示してある。実際の説明は、Java 文書内に見つけることができる。

7.1 java.lang クラス群

7.1.1 JavaLangArrayIndexOutOfBoundsException クラス

```
public  JavaLangArrayIndexOutOfBoundsException :  
        () ==> JavaLangArrayIndexOutOfBoundsException  
public  JavaLangArrayIndexOutOfBoundsException :  
        int ==> JavaLangArrayIndexOutOfBoundsException  
public  JavaLangArrayIndexOutOfBoundsException :  
        <NIL> ==> JavaLangArrayIndexOutOfBoundsException  
public  JavaLangArrayIndexOutOfBoundsException :  
        JavaLangString ==> JavaLangArrayIndexOutOfBoundsException
```

7.1.2 JavaLangBoolean クラス

```
public  static TRUE : JavaLangBoolean := new JavaLangBoolean(true);  
public  static FALSE : JavaLangBoolean := new JavaLangBoolean(false);  
public  static TYPE : JavaLangClass :=  
        new JavaLangClass("boolean", <CLASS>, true);  
public  JavaLangBoolean : bool ==> JavaLangBoolean  
public  hashCode : () ==> int  
public  toString : () ==> JavaLangString  
public  JavaLangBoolean : <NIL> ==> JavaLangBoolean  
public  booleanValue : () ==> bool  
public  equals : JavaLangObject ==> bool  
public  JavaLangBoolean : JavaLangString ==> JavaLangBoolean  
public  static valueOf : JavaLangString ==> JavaLangBoolean  
public  static getBoolean : JavaLangString ==> bool
```

7.1.3 JavaLangCharacter クラス

```
public static MIN_RADIX : int := 2;
```

```
public static MAX_RADIX : int := 36;
public static MIN_VALUE : char:= '0';
public static MAX_VALUE : char:= 'f';
public static getType : char ==> int
public static isSpace : char ==> bool
public static isDefined : char ==> bool
public static forDigit : int * int ==> char
public static isLowerCase : char ==> bool
public static isTitleCase : char ==> bool
public static isUpperCase : char ==> bool
public static toLowerCase : char ==> char
public static toTitleCase : char ==> char
public static toUpperCase : char ==> char
public static isISOControl : char ==> bool
public static isJavaLetter : char ==> bool
public JavaLangCharacter : <NIL> ==> JavaLangCharacter
public static getNumericValue : char ==> int
public static isJavaLetterOrDigit : char ==> bool
public static isJavaIdentifierPart : char ==> bool
public static isIdentifierIgnorable : char ==> bool
public static isJavaIdentifierStart : char ==> bool
public static isUnicodeIdentifierPart : char ==> bool
public static isUnicodeIdentifierStart : char ==> bool
```

7.1.4 JavaLangClass クラス

```
public JavaLangClass : <NIL> ==> JavaLangClass
public JavaLangClass : seq1 of char *
    (<CLASS> | <INTERFACE>) * bool ==> JavaLangClass
public getName : () ==> JavaLangString
public isArray : () ==> bool
public toString : () ==> JavaLangString
public getClasses : () ==> map int to JavaLangClass
public getSigners : () ==> map int to JavaLangObject
public isInterface : () ==> bool
public isPrimitive : () ==> bool
public newInstance : () ==> JavaLangObject
public getInterfaces : () ==> map int to JavaLangClass
public static forName : JavaLangString ==> JavaLangClass
```

7.1.5 JavaLangClassCastException クラス

```
public JavaLangClassCastException :  
    () ==> JavaLangClassCastException  
public JavaLangClassCastException :  
    <NIL> ==> JavaLangClassCastException  
public JavaLangClassCastException :  
    JavaLangString ==> JavaLangClassCastException
```

7.1.6 JavaLangClassNotFoundException クラス

```
public getException : () ==> JavaLangThrowable  
public printStackTrace : () ==> ()  
public JavaLangClassNotFoundException :  
    () ==> JavaLangClassNotFoundException  
public JavaLangClassNotFoundException :  
    <NIL> ==> JavaLangClassNotFoundException  
public printStackTrace : JavaIoPrintStream ==> ()  
public printStackTrace : JavaIoPrintWriter ==> ()  
public JavaLangClassNotFoundException :  
    JavaLangString ==> JavaLangClassNotFoundException  
public JavaLangClassNotFoundException : JavaLangString *  
    JavaLangThrowable ==> JavaLangClassNotFoundException
```

7.1.7 JavaLangComparable クラス

```
public compareTo : JavaLangObject ==> int
```

7.1.8 JavaLangConversionBufferFullException クラス

```
public JavaLangConversionBufferFullException :  
    () ==> JavaLangConversionBufferFullException  
public JavaLangConversionBufferFullException :  
    <NIL> ==> JavaLangConversionBufferFullException  
public JavaLangConversionBufferFullException :  
    JavaLangString ==> JavaLangConversionBufferFullException
```


7.1.9 JavaLangDouble クラス

```
public static POSITIVE_INFINITY : real :=0;
public static NEGATIVE_INFINITY : real :=0;
public static NaN : real :=0;
public static MAX_VALUE : real := 1.79769313486231570e+308;
public static MIN_VALUE : real := 4.94065645841246544e-324;
public static TYPE : JavaLangClass :=
    new JavaLangClass("double", <CLASS>, true);
public isNaN : () ==> bool
public static isNaN : real ==> bool
public JavaLangDouble : real ==> JavaLangDouble
public hashCode : () ==> int
public intValue : () ==> int
public toString : () ==> JavaLangString
public byteValue : () ==> int
public longValue : () ==> int
public static toString : real ==> JavaLangString
public floatValue : () ==> real
public isInfinite : () ==> bool
public shortValue : () ==> int
public JavaLangDouble : <NIL> ==> JavaLangDouble
public doubleValue : () ==> real
public static isInfinite : real ==> bool
public JavaLangDouble : JavaLangString ==> JavaLangDouble
public equals : JavaLangObject ==> bool
public static valueOf : JavaLangString ==> JavaLangDouble
public compareTo : JavaLangObject ==> int
public static doubleToLongBits : real ==> int
public static longBitsToDouble : int ==> real
public static parseDouble : JavaLangString ==> real
public static doubleToRawLongBits : real ==> int
```

7.1.10 JavaLangException クラス

```
public JavaLangException : () ==> JavaLangException
public JavaLangException : <NIL> ==> JavaLangException
public JavaLangException : JavaLangString ==> JavaLangException
```

7.1.11 `JavaLangIllegalAccessException` クラス

```
public JavaLangIllegalAccessErrorException :  
    () ==> JavaLangIllegalAccessErrorException  
public JavaLangIllegalAccessErrorException :  
    <NIL> ==> JavaLangIllegalAccessErrorException  
public JavaLangIllegalAccessErrorException :  
    JavaLangString ==> JavaLangIllegalAccessErrorException
```

7.1.12 `JavaLangIllegalArgumentException` クラス

```
public JavaLangIllegalArgumentException :  
    () ==> JavaLangIllegalArgumentException  
public JavaLangIllegalArgumentException :  
    <NIL> ==> JavaLangIllegalArgumentException  
public JavaLangIllegalArgumentException :  
    JavaLangString ==> JavaLangIllegalArgumentException
```

7.1.13 `JavaLangIllegalStateException` クラス

```
public JavaLangIllegalStateException :  
    () ==> JavaLangIllegalStateException  
public JavaLangIllegalStateException :  
    <NIL> ==> JavaLangIllegalStateException  
public JavaLangIllegalStateException :  
    JavaLangString ==> JavaLangIllegalStateException
```

7.1.14 `JavaLangIndexOutOfBoundsException` クラス

```
public JavaLangIndexOutOfBoundsException :  
    () ==> JavaLangIndexOutOfBoundsException  
public JavaLangIndexOutOfBoundsException :  
    <NIL> ==> JavaLangIndexOutOfBoundsException  
public JavaLangIndexOutOfBoundsException :  
    JavaLangString ==> JavaLangIndexOutOfBoundsException
```

7.1.15 JavaLangInstantiationException クラス

```
public JavaLangInstantiationException :  
    () ==> JavaLangInstantiationException  
public JavaLangInstantiationException :  
    <NIL> ==> JavaLangInstantiationException  
public JavaLangInstantiationException :  
    JavaLangString ==> JavaLangInstantiationException
```

7.1.16 JavaLangInteger クラス

```
public CLASS : JavaLangClass =  
    new JavaLangClass("JavaLangInteger", <CLASS>, false);  
public TYPE : JavaLangClass =  
    new JavaLangClass("int", <CLASS>, true);  
public MIN_VALUE : int == -2147483648;  
public digits : map int to char  
static public MAX_VALUE : int == 2147483647;  
public JavaLangInteger : () ==> JavaLangInteger  
public JavaLangInteger : int ==> JavaLangInteger  
public JavaLangInteger : char ==> JavaLangInteger  
public JavaLangInteger : JavaLangString ==> JavaLangInteger  
public getClass : () ==> JavaLangClass  
public hashCode : () ==> int  
public intValue : () ==> int  
public toString : () ==> JavaLangString  
public byteValue : () ==> int  
public longValue : () ==> int  
public charValue : () ==> char  
static public toString : int ==> JavaLangString  
public floatValue : () ==> real  
public shortValue : () ==> int  
public doubleValue : () ==> real  
public static toString_ : int * int ==> JavaLangString  
public JavaLangInteger : <NIL> ==> JavaLangInteger  
public static toHexString : int ==> JavaLangString  
public static decode : JavaLangString ==> JavaLangInteger  
public equals : JavaLangObject ==> bool  
public static toOctalString : int ==> JavaLangString  
public static valueOf : JavaLangString ==> JavaLangInteger  
public parseInt : JavaLangString ==> int  
public compareTo : JavaLangObject ==> int
```

```
public compareToInt : JavaLangInteger ==> int
public static getInteger : JavaLangString ==> JavaLangInteger
public parseInt : JavaLangString * int ==> int
public static getInteger : JavaLangString * int ==> JavaLangInteger
public static getInteger :
    JavaLangString * JavaLangInteger ==> JavaLangInteger
```

7.1.17 J2VUTIL クラス

```
public BitOp: int * (<AND> | <OR> | <EXCLOR>) * int ==> int
public ConcatStr: JavaLangString * JavaLangString ==> JavaLangString
public toString :
    int | real | char | JavaLangObject ==> JavaLangString
public toChar : int | real | char ==> char
public toInt : int | real | char ==> int
public toFloat : int | real | char ==> real
```

7.1.18 JavaLangNullPointerException クラス

```
public JavaLangNullPointerException :
    () ==> JavaLangNullPointerException
public JavaLangNullPointerException :
    <NIL> ==> JavaLangNullPointerException
public JavaLangNullPointerException :
    JavaLangString ==> JavaLangNullPointerException
```

7.1.19 Nullable クラス

```
public IsNil: () ==> bool
public isNil: bool := false
```

7.1.20 JavaLangNumber クラス

```
public intValue : () ==> int
public byteValue : () ==> int
public longValue : () ==> int
```

```
public floatValue : () ==> real
public shortValue : () ==> int
public JavaLangNumber : <NIL> ==> JavaLangNumber
public doubleValue : () ==> real
```

7.1.21 JavaLangNumberFormatException クラス

```
public JavaLangNumberFormatException :
    () ==> JavaLangNumberFormatException
public JavaLangNumberFormatException :
    <NIL> ==> JavaLangNumberFormatException
public JavaLangNumberFormatException :
    JavaLangString ==> JavaLangNumberFormatException
```

7.1.22 JavaLangObject クラス

```
public JavaLangObject: <NIL> ==> JavaLangObject
public JavaLangObject : () ==> JavaLangObject
public wait : () ==> ()
public clone : () ==> JavaLangObject
public wait : int ==> ()
public notify : () ==> ()
public wait : int * int ==> ()
protected finalize : () ==> ()
public getClass : () ==> JavaLangClass
public hashCode : () ==> int
public toString : () ==> JavaLangString
public notifyAll : () ==> ()
public equals : JavaLangObject ==> bool
```

7.1.23 JavaLangRuntimeException クラス

```
public JavaLangRuntimeException : () ==> JavaLangRuntimeException
public JavaLangRuntimeException : <NIL> ==> JavaLangRuntimeException
public JavaLangRuntimeException :
    JavaLangString ==> JavaLangRuntimeException
```

7.1.24 JavaLangString クラス

```
public CLASS : JavaLangClass =
    new JavaLangClass("JavaLangString", <CLASS>, false);
public JavaLangString : seq of char ==> JavaLangString
public JavaLangString : JavaLangString ==> JavaLangString
public getClass : () ==> JavaLangClass
public toSeqOfChar : () ==> seq of char
public JavaLangString: <NIL> ==> JavaLangString
public trim: () ==> JavaLangString
public JavaLangString : () ==> JavaLangString
public intern : () ==> JavaLangString
public length : () ==> int
public charAt : int ==> char
public getBytes : () ==> map int to int
public hashCode : () ==> int
public indexOf : int | char ==> int
public toString : () ==> JavaLangString
public indexOf : int * int ==> int
public replace : char * char ==> JavaLangString
public substring : int * int ==> JavaLangString
public substring : int ==> JavaLangString
public JavaLangString : map int to char ==> JavaLangString
public JavaLangString : map int to int ==> JavaLangString
public toCharArray : () ==> map int to char
public toLowerCase : () ==> JavaLangString
public toUpperCase : () ==> JavaLangString
public lastIndexOf : int ==> int
public JavaLangString :
    map int to int * int ==> JavaLangString
public lastIndexOf : int * int ==> int
public JavaLangString :
    (map int to char) * int * int ==> JavaLangString
public JavaLangString :
    map int to int * int * int ==> JavaLangString
public concat : JavaLangString ==> JavaLangString
public concat' : seq of char ==> ()
public static copyValueOf : map int to char ==> JavaLangString
public equals: JavaLangObject ==> bool
public JavaLangString :
    map int to int * int * int * int ==> JavaLangString
public endsWith : JavaLangString ==> bool
public getBytes : JavaLangString ==> map int to int
public compareTo : JavaLangObject ==> int
```

```
public  getBytes : int * int * map int to int * int ==> ()
public  getChars : int * int * map int to char * int ==> ()
public  indexOf : JavaLangString ==> int
public  indexOf : JavaLangString * int ==> int
public  static copyValueOf :
    map int to char * int * int ==> JavaLangString
public  startsWith : JavaLangString * int ==> bool
public  startsWith : JavaLangString ==> bool
public  lastIndexOf : JavaLangString ==> int
public  lastIndexOf : JavaLangString * int ==> int
public  JavaLangString :
    map int to int * JavaLangString ==> JavaLangString
public  JavaLangString : map int to int *
    int * int * JavaLangString ==> JavaLangString
public  equalsIgnoreCase : JavaLangString ==> bool
public  compareToIgnoreCase : JavaLangString ==> int
public  regionMatches : int * JavaLangString * int * int ==> bool
public  regionMatches :
    bool * int * JavaLangString * int * int ==> bool
public  static valueOf :
    map int to char * int * int ==> JavaLangString
public  static valueOf : map int to char ==> JavaLangString
```

7.1.25 JavaLangStringBuffer クラス

```
public  CLASS : JavaLangClass =
    new JavaLangClass("JavaLangString", <CLASS>, false);
public  JavaLangStringBuffer : <NIL> ==> JavaLangStringBuffer
public  getClass : () ==> JavaLangClass
public  toString : () ==> JavaLangString
```

7.1.26 JavaLangSystem クラス

```
private static props : JavaUtilProperties :=
    new JavaUtilProperties(<NIL>);
public static out : JavaIoPrintStream :=
    new JavaIoPrintStream(new JavaIoFileOutputStream("stdout", true), true);
public static err : JavaIoPrintStream :=
    new JavaIoPrintStream(new JavaIoFileOutputStream("stderr", true), true);
public static gc : () ==> ()
```

```
public  JavaLangSystem : () ==> JavaLangSystem
public  static  exit__ : int ==> ()
public  static  load : JavaLangString ==> ()
public  static  getProperties : () ==> JavaUtilProperties
public  static  getenv : JavaLangString ==> JavaLangString
public  static  runFinalization : () ==> ()
public  static  currentTimeMillis : () ==> int
public  static  getProperty : JavaLangString ==> JavaLangString
public  static  loadLibrary : JavaLangString ==> ()
public  static  runFinalizersOnExit : bool ==> ()
public  static  mapLibraryName : JavaLangString ==> JavaLangString
public  static  identityHashCode : JavaLangObject ==> int
public  static  getProperty :
    JavaLangString * JavaLangString ==> JavaLangString
public  static  setProperty :
    JavaLangString * JavaLangString ==> JavaLangString
```

7.1.27 JavaLangThrowable クラス

```
public  toString : () ==> JavaLangString
public  JavaLangThrowable : () ==> JavaLangThrowable
public  getMessage : () ==> JavaLangString
public  JavaLangThrowable : <NIL> ==> JavaLangThrowable
public  printStackTrace : () ==> ()
public  fillInStackTrace : () ==> JavaLangThrowable
public  JavaLangThrowable : JavaLangString ==> JavaLangThrowable
public  getLocalizedMessage : () ==> JavaLangString
public  printStackTrace : JavaIoPrintStream ==> ()
public  printStackTrace : JavaIoPrintWriter ==> ()
```

7.1.28 JavaLangUnsupportedOperationException クラス

```
public  JavaLangUnsupportedOperationException :
    () ==> JavaLangUnsupportedOperationException
public  JavaLangUnsupportedOperationException :
    <NIL> ==> JavaLangUnsupportedOperationException
public  JavaLangUnsupportedOperationException :
    JavaLangString ==> JavaLangUnsupportedOperationException
```


7.2 java.util クラス群

7.2.1 JavaUtilALitr クラス

```
protected cursor : int := 0;
protected lastRet : int := -1;
protected expectedModCount : int;
protected al : JavaUtilAbstractList;
public hasNext : () ==> bool
public next : () ==> JavaLangObject
```

7.2.2 JavaUtilALListItr クラス

```
public hasPrevious : () ==> bool
public previous : () ==> JavaLangObject
public nextIndex : () ==> int
public previousIndex : () ==> int
public set__ : JavaLangObject ==> ()
```

7.2.3 JavaUtilAbstractCollection クラス

```
public clear : () ==> ()
public isEmpty : () ==> bool
public toArray : () ==> map int to JavaLangObject
public toString : () ==> JavaLangString
public contains : JavaLangObject ==> bool
public addAll : JavaUtilCollection ==> bool
public removeAll : JavaUtilCollection ==> bool
public retainAll : JavaUtilCollection ==> bool
public JavaUtilAbstractCollection :
    <NIL> ==> JavaUtilAbstractCollection
```

7.2.4 JavaUtilAbstractList クラス

```
public modCount : int := 0
public get : int ==> JavaLangObject
public clear : () ==> ()
```

```
public getClass : () ==> JavaLangClass
public hashCode : () ==> int
public iterator : () ==> JavaUtilIterator
public add : JavaLangObject ==> bool
public listIterator : () ==> JavaUtilListIterator
public listIterator : int ==> JavaUtilListIterator
public equals : JavaLangObject ==> bool
protected removeRange : int * int ==> ()
public indexOf : JavaLangObject ==> int
public set__ : int * JavaLangObject ==> JavaLangObject
public JavaUtilAbstractList : <NIL> ==> JavaUtilAbstractList
public lastIndexOf : JavaLangObject ==> int
protected cursor : int := 0;
protected lastRet : int := -1;
protected expectedModCount : int;
protected al : JavaUtilAbstractList;
public hasNext : () ==> bool
public next : () ==> JavaLangObject
public hasPrevious : () ==> bool
public previous : () ==> JavaLangObject
public nextIndex : () ==> int
public previousIndex : () ==> int
public set__ : JavaLangObject ==> ()
```

7.2.5 JavaUtilAbstractMap クラス

```
public size : () ==> int
public clear : () ==> ()
public keySet : () ==> JavaUtilSet
public isEmpty : () ==> bool
public entrySet : () ==> JavaUtilSet
public hashCode : () ==> int
public toString : () ==> JavaLangString
public values__ : () ==> JavaUtilCollection
protected JavaUtilAbstractMap : () ==> JavaUtilAbstractMap
public get : JavaLangObject ==> JavaLangObject
public putAll : JavaUtilMap ==> ()
public equals : JavaLangObject ==> bool
public remove : JavaLangObject ==> JavaLangObject
public JavaUtilAbstractMap : <NIL> ==> JavaUtilAbstractMap
public containsKey : JavaLangObject ==> bool
public put : JavaLangObject * JavaLangObject ==> JavaLangObject
```

```
public containsValue : JavaLangObject ==> bool
```

7.2.6 JavaUtilAbstractSet クラス

```
public hashCode : () ==> int
public JavaUtilAbstractSet : () ==> JavaUtilAbstractSet
public equals : JavaLangObject ==> bool
public JavaUtilAbstractSet : <NIL> ==> JavaUtilAbstractSet
public removeAll : JavaUtilCollection ==> bool
```

7.2.7 JavaUtilCollection クラス

```
public size : () ==> int
public clear : () ==> ()
public isEmpty : () ==> bool
public hashCode : () ==> int
public iterator : () ==> JavaUtilIterator
public equals : JavaLangObject ==> bool
public contains : JavaLangObject ==> bool
public addAll : JavaUtilCollection ==> bool
public removeAll : JavaUtilCollection ==> bool
public retainAll : JavaUtilCollection ==> bool
```

7.2.8 JavaUtilConcurrentModificationException クラス

```
public JavaUtilConcurrentModificationException :
    () ==> JavaUtilConcurrentModificationException
public JavaUtilConcurrentModificationException :
    <NIL> ==> JavaUtilConcurrentModificationException
public JavaUtilConcurrentModificationException :
    JavaLangString ==> JavaUtilConcurrentModificationException
```

7.2.9 JavaUtilDate クラス

```
public JavaUtilDate : () ==> JavaUtilDate
public JavaUtilDate : int ==> JavaUtilDate
```

```
public clone : () ==> JavaLangObject
public getDay : () ==> int
public getDate : () ==> int
public getTime : () ==> int
public getYear : () ==> int
public getHours : () ==> int
public getMonth : () ==> int
public hashCode : () ==> int
public setDate : int ==> ()
public setTime : int ==> ()
public setYear : int ==> ()
public toString : () ==> JavaLangString
public JavaUtilDate : <NIL> ==> JavaUtilDate
public JavaUtilDate : int * int * int ==> JavaUtilDate
public setHours : int ==> ()
public setMonth : int ==> ()
public getMinutes : () ==> int
public getSeconds : () ==> int
public after : JavaUtilDate ==> bool
public setMinutes : int ==> ()
public setSeconds : int ==> ()
public toGMTString : () ==> JavaLangString
public JavaUtilDate : JavaLangString ==> JavaUtilDate
public before : JavaUtilDate ==> bool
public JavaUtilDate : int * int * int * int * int ==> JavaUtilDate
public static parse : JavaLangString ==> int
public static UTC : int * int * int * int * int * int ==> int
public equals : JavaLangObject ==> bool
public toLocaleString : () ==> JavaLangString
public JavaUtilDate :
    int * int * int * int * int * int ==> JavaUtilDate
public compareTo : JavaLangObject ==> int
public getTimezoneOffset : () ==> int
```

7.2.10 JavaUtilDictionary クラス

```
public isEmpty : () ==> bool
public JavaUtilDictionary : <NIL> ==> JavaUtilDictionary
```

7.2.11 JavaUtilEmptyEnumerator クラス

```
public  nextElement : () ==> JavaLangObject
public  JavaUtilEmptyEnumerator : () ==> JavaUtilEmptyEnumerator
public  hasMoreElements : () ==> bool
```

7.2.12 JavaUtilEmptyIterator クラス

```
public  next : () ==> JavaLangObject
public  remove : () ==> ()
public  hasNext : () ==> bool
public  JavaUtilEmptyIterator : () ==> JavaUtilEmptyIterator
```

7.2.13 JavaUtilEmptyStackException クラス

```
public  JavaUtilEmptyStackException :
    () ==> JavaUtilEmptyStackException
public  JavaUtilEmptyStackException :
    <NIL> ==> JavaUtilEmptyStackException
```

7.2.14 JavaUtilEntry クラス

```
public  getClass : () ==> JavaLangClass
public  hashCode : () ==> int
public  equals : JavaLangObject ==> bool
```

7.2.15 JavaUtilEnumeration クラス

```
public  JavaUtilEnumeration : <NIL> ==> JavaUtilEnumeration
public  nextElement : () ==> JavaLangObject
public  hasMoreElements : () ==> bool
protected  expectedModCount : int;
public  remove : () ==> ()
public  hasNext : () ==> bool
public  nextElement : () ==> JavaLangObject
```

```
public  JavaUtilEnumerator :  
        int * bool * JavaUtilHashtable ==> JavaUtilEnumerator  
public  JavaUtilEnumerator : <NIL> ==> JavaUtilEnumerator  
public  hasMoreElements : () ==> bool
```

7.2.16 JavaUtilHTEntry クラス

```
public  hash : int ;  
public  key : JavaLangObject;  
public  value__ : JavaLangObject;  
public  next : JavaUtilHTEntry  
public  clone : () ==> JavaLangObject  
public  getKey : () ==> JavaLangObject  
public  getValue : () ==> JavaLangObject  
public  hashCode : () ==> int  
public  toString : () ==> JavaLangString  
public  JavaUtilHTEntry : <NIL> ==> JavaUtilHTEntry  
public  equals : JavaLangObject ==> bool  
public  setValue : JavaLangObject ==> JavaLangObject  
public  JavaUtilHTEntry : int * JavaLangObject *  
        JavaLangObject * JavaUtilHTEntry ==> JavaUtilHTEntry
```

7.2.17 JavaUtilHTKeySet クラス

```
public  JavaUtilHTKeySet : JavaUtilHashtable ==> JavaUtilHTKeySet  
public  iterator : () ==> JavaUtilIterator  
public  size : () ==> int  
public  contains : JavaLangObject ==> bool  
public  remove : JavaLangObject ==> bool  
public  clear : () ==> ()
```

7.2.18 JavaUtilHashMap クラス

```
public  hash : int;  
public  key : JavaLangObject;  
public  value__: JavaLangObject;  
public  next : HMEntry;  
public  HMEntry : <NIL> ==> HMEntry
```

```
public clone : () ==> JavaLangObject
public getKey : () ==> JavaLangObject
public getValue : () ==> JavaLangObject
public setValue : JavaLangObject ==> JavaLangObject
public equals : JavaLangObject ==> bool
public hashCode : () ==> int
public toString : () ==> JavaLangString
public table : map int to HMEntry;
public count : int :=0;
public threshold : int;
public loadFactor_ : real;
public modCount : int := 0;
public static KEYS : int := 0;
public static VALUES : int := 1;
public static ENTRIES : int := 2;
public size : () ==> int
public clear : () ==> ()
public clone : () ==> JavaLangObject
public keySet : () ==> JavaUtilSet
public getHashIterator : int ==> JavaUtilIterator
public JavaUtilHashMap : () ==> JavaUtilHashMap
public isEmpty : () ==> bool
public JavaUtilHashMap : int ==> JavaUtilHashMap
public entrySet : () ==> JavaUtilSet
public values__ : () ==> JavaUtilCollection
public JavaUtilHashMap : int * real ==> JavaUtilHashMap
public get : JavaLangObject ==> JavaLangObject
public putAll : JavaUtilMap ==> ()
public JavaUtilHashMap : JavaUtilMap ==> JavaUtilHashMap
public JavaUtilHashMap : <NIL> ==> JavaUtilHashMap
public remove : JavaLangObject ==> JavaLangObject
public containsKey : JavaLangObject ==> bool
public put : JavaLangObject * JavaLangObject ==> JavaLangObject
protected rehash : () ==> ()
public containsValue : JavaLangObject ==> bool
public HMKeySet: JavaUtilHashMap ==> HMKeySet
public iterator : () ==> JavaUtilIterator
public size : () ==> int
public contains : JavaLangObject ==> bool
public remove : JavaLangObject ==> bool
public clear : () ==> ()
public HMEntrySet : JavaUtilHashMap ==> HMEntrySet
public iterator : () ==> JavaUtilIterator
public size : () ==> int
```

```
public contains : JavaLangObject ==> bool
public remove : JavaLangObject ==> bool
public clear : () ==> ()
public next : () ==> JavaLangObject
public remove : () ==> ()
public hasNext : () ==> bool
public EmptyHashIterator : () ==> EmptyHashIterator
public HashIterator: int * JavaUtilHashMap ==> HashIterator
public hasNext : () ==> bool
public next : () ==> JavaLangObject
public remove : () ==> ()
```

7.2.19 JavaUtilHashSet クラス

```
public size : () ==> int
public clear : () ==> ()
public clone : () ==> JavaLangObject
public JavaUtilHashSet : () ==> JavaUtilHashSet
public isEmpty : () ==> bool
public JavaUtilHashSet : int ==> JavaUtilHashSet
public iterator : () ==> JavaUtilIterator
public JavaUtilHashSet : int * real ==> JavaUtilHashSet
public add : JavaLangObject ==> bool
public JavaUtilHashSet : <NIL> ==> JavaUtilHashSet
public remove : JavaLangObject ==> bool
public contains : JavaLangObject ==> bool
public JavaUtilHashSet : JavaUtilCollection ==> JavaUtilHashSet
```

7.2.20 JavaUtilHashtable クラス

```
public static KEYS : int := 0;
public static VALUES : int := 1;
public static ENTRIES : int := 2;
public table : map int to JavaUtilHTEntry;
public count : int := 0;
public modCount : int := 0;
public keys : () ==> JavaUtilEnumeration
public size : () ==> int
public clear : () ==> ()
public clone : () ==> JavaLangObject
```



```
public keySet : () ==> JavaUtilSet
protected rehash : () ==> ()
public isEmpty : () ==> bool
public elements : () ==> JavaUtilEnumeration
public entrySet : () ==> JavaUtilSet
public getClass : () ==> JavaLangClass
public hashCode : () ==> int
public toString : () ==> JavaLangString
public values__ : () ==> JavaUtilCollection
public JavaUtilHashtable : () ==> JavaUtilHashtable
public JavaUtilHashtable : int ==> JavaUtilHashtable
public get : JavaLangObject ==> JavaLangObject
public putAll : JavaUtilMap ==> ()
public JavaUtilHashtable : int * real ==> JavaUtilHashtable
public getIterator : int ==> JavaUtilIterator
public JavaUtilHashtable : <NIL> ==> JavaUtilHashtable
public equals : JavaLangObject ==> bool
public remove : JavaLangObject ==> JavaLangObject
public contains : JavaLangObject ==> bool
public containsKey : JavaLangObject ==> bool
public put : JavaLangObject * JavaLangObject ==> JavaLangObject
public containsValue : JavaLangObject ==> bool
```

7.2.21 JavaUtilIterator クラス

```
public JavaUtilIterator : <NIL> ==> JavaUtilIterator
public next : () ==> JavaLangObject
public remove : () ==> ()
public hasNext : () ==> bool
```

7.2.22 JavaUtilList クラス

```
public get : int ==> JavaLangObject
public size : () ==> int
public clear : () ==> ()
public isEmpty : () ==> bool
public toArray : () ==> map int to JavaLangObject
public hashCode : () ==> int
public iterator : () ==> JavaUtilIterator
public subList : int * int ==> JavaUtilList
```

```
public equals : JavaLangObject ==> bool
public indexOf : JavaLangObject ==> int
public set__ : int * JavaLangObject ==> JavaLangObject
public contains : JavaLangObject ==> bool
public addAll : JavaUtilCollection ==> bool
public lastIndexOf : JavaLangObject ==> int
public toArray : map int to JavaLangObject ==>
                    map int to JavaLangObject
public addAll : int * JavaUtilCollection ==> bool
public removeAll : JavaUtilCollection ==> bool
public retainAll : JavaUtilCollection ==> bool
public containsAll : JavaUtilCollection ==> bool
```

7.2.23 JavaUtilListIterator クラス

```
public next : () ==> JavaLangObject
public remove : () ==> ()
public hasNext : () ==> bool
public previous : () ==> JavaLangObject
public nextIndex : () ==> int
public add : JavaLangObject ==> ()
public hasPrevious : () ==> bool
public previousIndex : () ==> int
public set__ : JavaLangObject ==> ()
```

7.2.24 JavaUtilLocale クラス

```
public static ENGLISH : [JavaUtilLocale] := nil;
public static FRENCH : [JavaUtilLocale] := nil;
public static GERMAN : [JavaUtilLocale] := nil;
public static ITALIAN : [JavaUtilLocale] := nil;
public static JAPANESE : [JavaUtilLocale] := nil;
public static KOREAN : [JavaUtilLocale] := nil;
public static CHINESE : [JavaUtilLocale] := nil;
public static SIMPLIFIED_CHINESE : [JavaUtilLocale] := nil;
public static TRADITIONAL_CHINESE : [JavaUtilLocale] := nil;
public static FRANCE : [JavaUtilLocale] := nil;
public static GERMANY : [JavaUtilLocale] := nil;
public static ITALY : [JavaUtilLocale] := nil;
public static JAPAN : [JavaUtilLocale] := nil;
```

```
public static KOREA : [JavaUtilLocale] := nil;
public static CHINA : [JavaUtilLocale] := nil;
public static PRC : [JavaUtilLocale] := nil;
public static TAIWAN : [JavaUtilLocale] := nil;
public static UK : [JavaUtilLocale] := nil;
public static US : [JavaUtilLocale] := nil;
public static CANADA : [JavaUtilLocale] := nil;
public static CANADA_FRENCH : [JavaUtilLocale] := nil;
public clone : () ==> JavaLangObject
public hashCode : () ==> int
public toString : () ==> JavaLangString
public getCountry : () ==> JavaLangString
public static getDefault : () ==> JavaUtilLocale
public getVariant : () ==> JavaLangString
public JavaUtilLocale : <NIL> ==> JavaUtilLocale
public getLanguage : () ==> JavaLangString
public equals : JavaLangObject ==> bool
public getDisplayName : () ==> JavaLangString
public getISO3Country : () ==> JavaLangString
public getISO3Language : () ==> JavaLangString
public static getISOCountries : () ==> map int to JavaLangString
public static getISOLanguages : () ==> map int to JavaLangString
public getDisplayCountry : () ==> JavaLangString
public getDisplayVariant : () ==> JavaLangString
public getDisplayLanguage : () ==> JavaLangString
public static setDefault : JavaUtilLocale ==> ()
public static getAvailableLocales :
    () ==> map int to JavaUtilLocale
public getDisplayName : JavaUtilLocale ==> JavaLangString
public JavaUtilLocale :
    JavaLangString * JavaLangString ==> JavaUtilLocale
public getDisplayCountry : JavaUtilLocale ==> JavaLangString
public getDisplayVariant : JavaUtilLocale ==> JavaLangString
public getDisplayLanguage : JavaUtilLocale ==> JavaLangString
public JavaUtilLocale : JavaLangString * JavaLangString *
    JavaLangString ==> JavaUtilLocale
```

7.2.25 JavaUtilMap クラス

```
public size : () ==> int
public clear : () ==> ()
public keySet : () ==> JavaUtilSet
```

```
public isEmpty : () ==> bool
public get : JavaLangObject ==> JavaLangObject
public containsKey : JavaLangObject ==> bool
public put : JavaLangObject * JavaLangObject ==> JavaLangObject
public containsValue : JavaLangObject ==> bool
```

7.2.26 JavaUtilMissingResourceException クラス

```
public getKey : () ==> JavaLangString
public getClassName : () ==> JavaLangString
public JavaUtilMissingResourceException :
    <NIL> ==> JavaUtilMissingResourceException
public JavaUtilMissingResourceException :
    JavaLangString * JavaLangString * JavaLangString ==>
        JavaUtilMissingResourceException
```

7.2.27 JavaUtilNoSuchElementException クラス

```
public JavaUtilNoSuchElementException :
    () ==> JavaUtilNoSuchElementException
public JavaUtilNoSuchElementException :
    <NIL> ==> JavaUtilNoSuchElementException
public JavaUtilNoSuchElementException :
    JavaLangString ==> JavaUtilNoSuchElementException
```

7.2.28 JavaUtilObservable クラス

```
public JavaUtilObservable : () ==> JavaUtilObservable
public hasChanged : () ==> bool
protected setChanged : () ==> ()
protected clearChanged : () ==> ()
public countObservers : () ==> int
public JavaUtilObservable : <NIL> ==> JavaUtilObservable
public deleteObservers : () ==> ()
public notifyObservers : () ==> ()
public addObserver : JavaUtilObserver ==> ()
public notifyObservers : JavaLangObject ==> ()
public deleteObserver : JavaUtilObserver ==> ()
```

7.2.29 JavaUtilObserver クラス

```
public JavaUtilObserver : <NIL> ==> JavaUtilObserver
public update : JavaUtilObservable * JavaLangObject ==> ()
```

7.2.30 JavaUtilProperties クラス

```
protected defaults : JavaUtilProperties;
public JavaUtilProperties : () ==> JavaUtilProperties
public propertyNames : () ==> JavaUtilEnumeration
public JavaUtilProperties : <NIL> ==> JavaUtilProperties
public list : JavaIoPrintStream ==> ()
public load : JavaIoInputStream ==> ()
public getProperty : JavaLangString ==> JavaLangString
public JavaUtilProperties : JavaUtilProperties ==> JavaUtilProperties
public save : JavaIoOutputStream * JavaLangString ==> ()
public getProperty : JavaLangString * JavaLangString ==> JavaLangString
public setProperty : JavaLangString * JavaLangString ==> JavaLangObject
public store : JavaIoOutputStream * JavaLangString ==> ()
```

7.2.31 JavaUtilResourceBundle クラス

```
protected parent : JavaUtilResourceBundle
public getKeys : () ==> JavaUtilEnumeration
public getLocale : () ==> JavaUtilLocale
public JavaUtilResourceBundle : () ==> JavaUtilResourceBundle
public static getBundle : JavaLangString ==> JavaUtilResourceBundle
public getObject : JavaLangString ==> JavaLangObject
public getString : JavaLangString ==> JavaLangString
public JavaUtilResourceBundle : <NIL> ==> JavaUtilResourceBundle
public getStringArray : JavaLangString ==> map int to JavaLangString
protected handleGetObject : JavaLangString ==> JavaLangObject
protected setParent : JavaUtilResourceBundle ==> ()
public static getBundle : JavaLangString *
    JavaUtilLocale ==> JavaUtilResourceBundle
```

7.2.32 JavaUtilSet クラス

```
public clear : () ==> ()
public isEmpty : () ==> bool
public toArray : () ==> map int to JavaLangObject
public hashCode : () ==> int
public iterator : () ==> JavaUtilIterator
public add : JavaLangObject ==> bool
public equals : JavaLangObject ==> bool
public contains : JavaLangObject ==> bool
public addAll : JavaUtilCollection ==> bool
public toArray : map int to JavaLangObject ==>
                    map int to JavaLangObject
public removeAll : JavaUtilCollection ==> bool
public retainAll : JavaUtilCollection ==> bool
public containsAll : JavaUtilCollection ==> bool
```

7.2.33 JavaUtilStack クラス

```
public pop : () ==> JavaLangObject
public peek : () ==> JavaLangObject
public JavaUtilStack : () ==> JavaUtilStack
public empty : () ==> bool
public getClass : () ==> JavaLangClass
public JavaUtilStack : <NIL> ==> JavaUtilStack
public push : JavaLangObject ==> JavaLangObject
public search : JavaLangObject ==> int
```

7.2.34 JavaUtilStringTokenizer クラス

```
public nextToken : () ==> JavaLangString
public countTokens : () ==> int
public nextElement : () ==> JavaLangObject
public hasMoreTokens : () ==> bool
public hasMoreElements : () ==> bool
public nextToken : JavaLangString ==> JavaLangString
public JavaUtilStringTokenizer : <NIL> ==> JavaUtilStringTokenizer
public JavaUtilStringTokenizer :
    JavaLangString ==> JavaUtilStringTokenizer
```

```
public  JavaUtilStringTokenizer : J
        avaLangString * JavaLangString ==> JavaUtilStringTokenizer
public  JavaUtilStringTokenizer : JavaLangString *
        JavaLangString * bool ==> JavaUtilStringTokenizer
```

7.2.35 JavaUtilVector クラス

```
public  insertElementAt : JavaLangObject * int ==> ()
public  clear : () ==> ()
public  clone : () ==> JavaLangObject
public  contains : JavaLangObject ==> bool
public  containsAll : JavaUtilCollection ==> bool
public  elementAt : int ==> JavaLangObject
public  firstElement : () ==> JavaLangObject
public  get : int ==> JavaLangObject
public  isEmpty : () ==> bool
public  indexOf : JavaLangObject ==> int
public  indexOfFrom : JavaLangObject * int ==> int
public  lastElement : () ==> JavaLangObject
public  remove : int ==> JavaLangObject
public  remove' : JavaLangObject ==> bool
public  removeElementAt : int ==> ()
public  removeElement : JavaLangObject ==> bool
public  removeAll : JavaUtilCollection ==> bool
public  retainAll : JavaUtilCollection ==> bool
public  subList : int * int ==> JavaUtilList
public  elements : () ==> JavaUtilEnumeration
public  VEnumeration : JavaUtilVector ==> VEnumeration
public  hasMoreElements : () ==> bool
public  nextElement : () ==> JavaLangObject
```

7.3 java.io クラス群

7.3.1 JavaIoBufferedInputStream クラス

```
protected buf : map int to int ;
protected count : int ;
protected pos : int ;
protected markpos : int := -1;
```

```
protected marklimit : int
public read : () ==> int
public close : () ==> ()
public mark : int ==> ()
public reset : () ==> ()
public skip__ : int ==> int
public getClass : () ==> JavaLangClass
public available : () ==> int
public markSupported : () ==> bool
public read : map int to int * int * int ==> int
public readMIIIntInt' : map int to int *
                        int * int ==> int * map int to int
public JavaIoBufferedInputStream :
    <NIL> ==> JavaIoBufferedInputStream
public JavaIoBufferedInputStream :
    JavaIoInputStream ==> JavaIoBufferedInputStream
public JavaIoBufferedInputStream :
    JavaIoInputStream * int ==> JavaIoBufferedInputStream
```

7.3.2 JavaIoBufferedOutputStream クラス

```
protected buf : map int to int ;
protected count : int
public flush : () ==> ()
public write : int ==> ()
public getClass : () ==> JavaLangClass
public write2 : map int to int * int * int ==> ()
public JavaIoBufferedOutputStream :
    <NIL> ==> JavaIoBufferedOutputStream
public JavaIoBufferedOutputStream :
    JavaIoOutputStream ==> JavaIoBufferedOutputStream
public JavaIoBufferedOutputStream :
    JavaIoOutputStream * int ==> JavaIoBufferedOutputStream
```

7.3.3 JavaIoBufferedReader クラス

```
public read : () ==> int
public close : () ==> ()
public mark : int ==> ()
public ready : () ==> bool
```



```
public reset : () ==> ()
public skip__ : int ==> int
public readLine : () ==> JavaLangString
public markSupported : () ==> bool
public read : map int to char * int * int ==> int
public read' : map int to char *
               int * int ==> int * map int to char
public javaIoBufferedReader : <NIL> ==> javaIoBufferedReader
public javaIoBufferedReader : javaIoReader ==> javaIoBufferedReader
public javaIoBufferedReader :
    javaIoReader * int ==> javaIoBufferedReader
```

7.3.4 javaIoBufferedWriter クラス

```
public close : () ==> ()
public flush : () ==> ()
public write : int ==> ()
public newLine : () ==> ()
public flushBuffer : () ==> ()
public write : (map int to char | JavaLangString) *
               int * int ==> ()
public writeICMIntInt : map int to char * int * int ==> ()
public writeStrIntInt : JavaLangString * int * int ==> ()
public javaIoBufferedWriter : <NIL> ==> javaIoBufferedWriter
public javaIoBufferedWriter : javaIoWriter ==> javaIoBufferedWriter
public javaIoBufferedWriter :
    javaIoWriter * int ==> javaIoBufferedWriter
```

7.3.5 javaIoByteArrayInputStream クラス

```
protected buf : map int to int ;
protected pos : int ;
protected mark_ : int := 0;
protected count : int
public read : () ==> int
public close : () ==> ()
public mark : int ==> ()
public reset : () ==> ()
public skip__ : int ==> int
public getClass : () ==> JavaLangClass
```

```
public available : () ==> int
public markSupported : () ==> bool
public read : map int to int * int * int ==> int
public readMIIIntInt' : map int to int * int * int ==>
                                int * map int to int

public JavaIoByteArrayInputStream :
    map int to int ==> JavaIoByteArrayInputStream
public JavaIoByteArrayInputStream : <NIL> ==> JavaIoByteArrayInputStream
public JavaIoByteArrayInputStream :
    map int to int * int * int ==> JavaIoByteArrayInputStream
```

7.3.6 JavaIoCharArrayReader クラス

```
protected buf : map int to char ;
protected pos : int ;
protected markedPos : int := 0;
protected count : int
public read : () ==> int
public close : () ==> ()
public mark : int ==> ()
public ready : () ==> bool
public reset : () ==> ()
public skip__ : int ==> int
public getClass : () ==> JavaLangClass
public markSupported : () ==> bool
public read : map int to char * int * int ==>
                                int * map int to char

public JavaIoCharArrayReader :
    map int to char ==> JavaIoCharArrayReader
public JavaIoCharArrayReader : <NIL> ==> JavaIoCharArrayReader
public JavaIoCharArrayReader :
    map int to char * int * int ==> JavaIoCharArrayReader
```

7.3.7 JavaIoFile クラス

```
public static separatorChar : char := fs.getSeparator();
public static separator : JavaLangString :=
    new JavaLangString([fs.getSeparator()]);
public static pathSeparatorChar : char :=
    fs.getPathSeparator();
```

```
public static pathSeparator : JavaLangString :=
    new JavaLangString([fs.getPathSeparator()]);
public list : () ==> map int to JavaLangString
public mkdir : () ==> bool
public toURL : () ==> JavaNetURL
public delete : () ==> bool
public isFile : () ==> bool
public length : () ==> int
public mkdirs : () ==> bool
public canRead : () ==> bool
public getName : () ==> JavaLangString
public getPath : () ==> JavaLangString
public canWrite : () ==> bool
public exists__ : () ==> bool
public getClass : () ==> JavaLangClass
public hashCode : () ==> int
public isHidden : () ==> bool
public toString : () ==> JavaLangString
public JavaIoFile : <NIL> ==> JavaIoFile
public getParent : () ==> JavaLangString
public listFiles : () ==> map int to JavaIoFile
public static listRoots : () ==> map int to JavaIoFile
public isAbsolute : () ==> bool
public isDirectory : () ==> bool
public setReadOnly : () ==> bool
public JavaIoFile : JavaLangString ==> JavaIoFile
public deleteOnExit : () ==> ()
public lastModified : () ==> int
public createNewFile : () ==> bool
public getParentFile : () ==> JavaIoFile
public JavaIoFile : JavaLangString * int ==> JavaIoFile
public JavaIoFile : JavaIoFile * JavaLangString ==> JavaIoFile
public equals : JavaLangObject ==> bool
public renameTo : JavaIoFile ==> bool
public getAbsoluteFile : () ==> JavaIoFile
public getAbsolutePath : () ==> JavaLangString
public getCanonicalFile : () ==> JavaIoFile
public getCanonicalPath : () ==> JavaLangString
public setLastModified : int ==> bool
public compareTo : JavaLangObject ==> int
public JavaIoFile : JavaLangString * JavaLangString ==> JavaIoFile
public static createTempFile :
    JavaLangString * JavaLangString ==> JavaIoFile
public static createTempFile :
```

```
JavaLangString * JavaLangString * JavaIoFile ==> JavaIoFile
```

7.3.8 JavaIoFileDescriptor クラス

```
public valid : () ==> bool
public sync__ : () ==> ()
public JavaIoFileDescriptor : () ==> JavaIoFileDescriptor
public JavaIoFileDescriptor : <NIL> ==> JavaIoFileDescriptor
public getClass : () ==> JavaLangClass
```

7.3.9 JavaIoFileInputStream クラス

```
public read : () ==> int
public close : () ==> ()
public getFD : () ==> JavaIoFileDescriptor
public skip__ : int ==> int
protected finalize : () ==> ()
public available : () ==> int
public readMIIIntInt' : map int to int * int * int ==>
                        int * map int to int
public JavaIoFileInputStream : JavaIoFile ==> JavaIoFileInputStream
public JavaIoFileInputStream : JavaLangString ==> JavaIoFileInputStream
public JavaIoFileInputStream : seq of char ==> JavaIoFileInputStream
public JavaIoFileInputStream :
    JavaIoFileDescriptor ==> JavaIoFileInputStream
```

7.3.10 JavaIoFileNotFoundException クラス

```
public JavaIoFileNotFoundException : () ==> JavaIoFileNotFoundException
public JavaIoFileNotFoundException :
    <NIL> ==> JavaIoFileNotFoundException
public JavaIoFileNotFoundException :
    JavaLangString ==> JavaIoFileNotFoundException
```

7.3.11 `JavaIoFileOutputStream` クラス

```
public  close : () ==> ()
public  getFD : () ==> JavaIoFileDescriptor
public  write : int ==> ()
protected  finalize : () ==> ()
public  JavaIoFileOutputStream : JavaIoFile ==> JavaIoFileOutputStream
public  JavaIoFileOutputStream : JavaLangString ==> JavaIoFileOutputStream
public  JavaIoFileOutputStream :
    JavaLangString * bool ==> JavaIoFileOutputStream
public  JavaIoFileOutputStream :
    seq of char * bool ==> JavaIoFileOutputStream
public  JavaIoFileOutputStream :
    JavaIoFileDescriptor ==> JavaIoFileOutputStream
```

7.3.12 `JavaIoFileReader` クラス

```
public  JavaIoFileReader : <NIL> ==> JavaIoFileReader
public  JavaIoFileReader : JavaIoFile ==> JavaIoFileReader
public  JavaIoFileReader : JavaLangString ==> JavaIoFileReader
public  JavaIoFileReader : JavaIoFileDescriptor ==> JavaIoFileReader
```

7.3.13 `JavaIoFileSystem` クラス

```
public  static  BA_EXISTS : int :=1;
public  static  BA_REGULAR : int :=2;
public  static  BA_DIRECTORY : int :=4;
public  static  BA_HIDDEN : int :=8
public  listRoots : () ==> map int to JavaIoFile
public  list : JavaIoFile ==> map int to JavaLangString
public  delete : JavaIoFile ==> bool
public  getSeparator : () ==> char
public  static  getFileSystem : () ==> JavaIoFileSystem
public  resolve : JavaIoFile ==> JavaLangString
public  hashCode : JavaIoFile ==> int
public  getLength : JavaIoFile ==> int
public  getDefaultParent : () ==> JavaLangString
public  getPathSeparator : () ==> char
public  isAbsolute : JavaIoFile ==> bool
```

```
public  normalize : JavaLangString ==> JavaLangString
public  setReadOnly : JavaIoFile ==> bool
public  deleteOnExit : JavaIoFile ==> bool
public  checkAccess : JavaIoFile * bool ==> bool
public  rename : JavaIoFile * JavaIoFile ==> bool
public  canonicalize : JavaLangString ==> JavaLangString
public  compare : JavaIoFile * JavaIoFile ==> int
public  prefixLength : JavaLangString ==> int
public  createDirectory : JavaIoFile ==> bool
public  resolve : JavaLangString * JavaLangString ==> JavaLangString
public  getLastModifiedTime : JavaIoFile ==> int
public  getBooleanAttributes : JavaIoFile ==> int
public  setLastModifiedTime : JavaIoFile * int ==> bool
public  createFileExclusively : JavaLangString ==> bool
public  J2VFileSystem : <NIL> ==> J2VFileSystem
public  listRoots : () ==> map int to JavaIoFile
public  list : JavaIoFile ==> map int to JavaLangString
public  delete : JavaIoFile ==> bool
public  getSeparator : () ==> char
public  static  getFileSystem : () ==> JavaIoFileSystem
public  resolve : JavaIoFile ==> JavaLangString
public  hashCode : JavaIoFile ==> int
public  getLength : JavaIoFile ==> int
public  getDefaultParent : () ==> JavaLangString
public  getPathSeparator : () ==> char
public  isAbsolute : JavaIoFile ==> bool
public  normalize : JavaLangString ==> JavaLangString
public  setReadOnly : JavaIoFile ==> bool
public  deleteOnExit : JavaIoFile ==> bool
public  checkAccess : JavaIoFile * bool ==> bool
public  rename : JavaIoFile * JavaIoFile ==> bool
public  canonicalize : JavaLangString ==> JavaLangString
public  compare : JavaIoFile * JavaIoFile ==> int
public  prefixLength : JavaLangString ==> int
public  createDirectory : JavaIoFile ==> bool
public  resolve : JavaLangString * JavaLangString ==> JavaLangString
public  getLastModifiedTime : JavaIoFile ==> int
public  getBooleanAttributes : JavaIoFile ==> int
public  setLastModifiedTime : JavaIoFile * int ==> bool
public  createFileExclusively : JavaLangString ==> bool
```

7.3.14 `JavaIoFileWriter` クラス

```
public JavaIoFileWriter : <NIL> ==> JavaIoFileWriter
public JavaIoFileWriter : JavaLangString ==> JavaIoFileWriter
public JavaIoFileWriter : JavaLangString * bool ==> JavaIoFileWriter
public JavaIoFileWriter : JavaIoFileDescriptor ==> JavaIoFileWriter
public JavaIoFileWriter : JavaIoFile ==> JavaIoFileWriter
```

7.3.15 `JavaIoFilterInputStream` クラス

```
protected in__ : JavaIoInputStream
public read : () ==> int
public close : () ==> ()
public mark : int ==> ()
public reset : () ==> ()
public skip__ : int ==> int
public getClass : () ==> JavaLangClass
public available : () ==> int
public read : map int to int ==> int
public markSupported : () ==> bool
public read : map int to int * int * int ==> int
public JavaIoFilterInputStream : <NIL> ==> JavaIoFilterInputStream
protected JavaIoFilterInputStream :
    JavaIoInputStream ==> JavaIoFilterInputStream
```

7.3.16 `JavaIoFilterOutputStream` クラス

```
protected out : JavaIoOutputStream
public close : () ==> ()
public flush : () ==> ()
public write : int ==> ()
public write1 : map int to int ==> ()
public write2 : map int to int * int * int ==> ()
public JavaIoFilterOutputStream :
    JavaIoOutputStream ==> JavaIoFilterOutputStream
```

7.3.17 JavaIoIOException クラス

```
public JavaIoIOException : () ==> JavaIoIOException
public JavaIoIOException : <NIL> ==> JavaIoIOException
public JavaIoIOException : JavaLangString ==> JavaIoIOException
```

7.3.18 JavaIoInputStream クラス

```
public read : () ==> int
public close : () ==> ()
public mark : int ==> ()
public reset : () ==> ()
public skip__ : int ==> int
public getClass : () ==> JavaLangClass
public available : () ==> int
public read : map int to int ==> int
public markSupported : () ==> bool
public readMIIIntInt : map int to int * int * int ==> int
public readMIIIntInt' : map int to int *
                        int * int ==> int * map int to int
public JavaIoInputStream : <NIL> ==> JavaIoInputStream
```

7.3.19 JavaIoInputStreamReader クラス

```
public read : () ==> int
public close : () ==> ()
public ready : () ==> bool
public getClass : () ==> JavaLangClass
public getEncoding : () ==> JavaLangString
public readMICIntInt : map int to char * int * int ==> int
public readMICIntInt' : map int to char *
                        int * int ==> int * map int to char
public JavaIoInputStreamReader : <NIL> ==> JavaIoInputStreamReader
public JavaIoInputStreamReader :
    JavaIoInputStream ==> JavaIoInputStreamReader
public JavaIoInputStreamReader : JavaIoInputStream *
    JavaLangString ==> JavaIoInputStreamReader
public ByteToCharConverter : <NIL> ==> ByteToCharConverter
```


7.3.20 `JavaOutputStream` クラス

```
public close : () ==> ()
public flush : () ==> ()
public write : int ==> ()
public write1 : map int to int ==> ()
public write2 : map int to int * int * int ==> ()
```

7.3.21 `JavaOutputStreamWriter` クラス

```
public close : () ==> ()
public flush : () ==> ()
public write : int ==> ()
public flushBuffer : () ==> ()
public getEncoding : () ==> JavaLangString
public writeICMIntInt : map int to char * int * int ==> ()
public writeStrIntInt : JavaLangString * int * int ==> ()
public JavaOutputStreamWriter : <NIL> ==> JavaOutputStreamWriter
public JavaOutputStreamWriter :
    JavaOutputStream ==> JavaOutputStreamWriter
public JavaOutputStreamWriter :
    JavaOutputStream * JavaLangString ==> JavaOutputStreamWriter
public CharToByteConverter : <NIL> ==> CharToByteConverter
```

7.3.22 `JavaPrintStream` クラス

```
public close : () ==> ()
public flush : () ==> ()
public print : bool | char | int | real |
    (map int to char) | JavaLangObject ==> ()
public write : int ==> ()
public println : () ==> ()
public println : bool | char | int | real |
    (map int to char) | JavaLangObject ==> ()
protected setError : () ==> ()
public checkError : () ==> bool
public write : map int to int * int * int ==> ()
public JavaPrintStream :
    JavaOutputStream * bool ==> JavaPrintStream
```

7.3.23 JavaIoPrintWriter クラス

```
protected out : JavaIoWriter;
public close : () ==> ()
public flush : () ==> ()
public write : int | map int to char ==> ()
public println : () ==> ()
public println : bool | char | int | real |
    (map int to char) | JavaLangObject ==> ()
public print : bool | char | int | real |
    (map int to char) | JavaLangObject ==> ()
protected setError : () ==> ()
public checkError : () ==> bool
public write : JavaLangString ==> ()
public write : map int to char * int * int ==> ()
public JavaIoPrintWriter : <NIL> ==> JavaIoPrintWriter
public write : JavaLangString * int * int ==> ()
public JavaIoPrintWriter : JavaIoWriter ==> JavaIoPrintWriter
public JavaIoPrintWriter : JavaIoWriter * bool ==> JavaIoPrintWriter
public JavaIoPrintWriter : JavaIoOutputStream ==> JavaIoPrintWriter
public JavaIoPrintWriter :
    JavaIoOutputStream * bool ==> JavaIoPrintWriter
```

7.3.24 JavaIoReader クラス

```
public read : () ==> int
public close : () ==> ()
public mark : int ==> ()
public ready : () ==> bool
public reset : () ==> ()
public JavaIoReader : () ==> JavaIoReader
public skip__ : int ==> int
public getClass : () ==> JavaLangClass
public readMIC : map int to char ==> int
public JavaIoReader : <NIL> ==> JavaIoReader
public markSupported : () ==> bool
public readMICIntInt : map int to char * int * int ==> int
public readMICIntInt' : map int to char *
    int * int ==> int * map int to char
protected JavaIoReader : JavaLangObject ==> JavaIoReader
```

7.3.25 JavaIoStreamTokenizer クラス

```
public ttype : int := TT_NOthing;
public static TT_EOF : int := -1;
public static TT_EOL : int := J2VUTIL.toInt('\n');
public static TT_NUMBER : int := -2;
public static TT_WORD : int := -3;
public sval : JavaLangString;
public nval : real
public lineno : () ==> int
public pushBack : () ==> ()
public toString : () ==> JavaLangString
public nextToken : () ==> int
public quoteChar : int ==> ()
public resetSyntax : () ==> ()
public commentChar : int ==> ()
public parseNumbers : () ==> ()
public wordChars : int * int ==> ()
public ordinaryChar : int ==> ()
public lowerCaseMode : bool ==> ()
public ordinaryChars : int * int ==> ()
public eolIsSignificant : bool ==> ()
public slashStarComments : bool ==> ()
public whitespaceChars : int * int ==> ()
public slashSlashComments : bool ==> ()
public JavaIoStreamTokenizer : <NIL> ==> JavaIoStreamTokenizer
public JavaIoStreamTokenizer : JavaIoReader ==> JavaIoStreamTokenizer
public JavaIoStreamTokenizer :
    JavaIoInputStream ==> JavaIoStreamTokenizer
```

7.3.26 JavaIoStringWriter クラス

```
public close : () ==> ()
public flush : () ==> ()
public write : int ==> ()
public toString : () ==> JavaLangString
public getBuffer : () ==> JavaLangStringBuffer
public JavaIoStringWriter : () ==> JavaIoStringWriter
public JavaIoStringWriter : int ==> JavaIoStringWriter
public write : JavaLangString ==> ()
public write : map int to char * int * int ==> ()
public JavaIoStringWriter : <NIL> ==> JavaIoStringWriter
```

```
public write : JavaLangString * int * int ==> ()
```

7.3.27 `JavaIoUnsupportedEncodingException` クラス

```
public JavaIoUnsupportedEncodingException :  
    () ==> JavaIoUnsupportedEncodingException  
public JavaIoUnsupportedEncodingException :  
    <NIL> ==> JavaIoUnsupportedEncodingException  
public JavaIoUnsupportedEncodingException :  
    JavaLangString ==> JavaIoUnsupportedEncodingException
```

7.3.28 `JavaIoWriter` クラス

```
public JavaIoWriter : <NIL> ==> JavaIoWriter  
public close : () ==> ()  
public flush : () ==> ()  
protected JavaIoWriter : () ==> JavaIoWriter  
public write : int | map int to char ==> ()  
public writeInt : int ==> ()  
public getClass : () ==> JavaLangClass  
public writeMIC : map int to char ==> ()  
public writeICMIntInt : map int to char * int * int ==> ()  
public writeStr : JavaLangString ==> ()  
protected JavaIoWriter : JavaLangObject ==> JavaIoWriter  
public writeStrIntInt : JavaLangString * int * int ==> ()
```

7.4 `java.net` クラス群

7.4.1 `JavaNetURL` クラス

```
public JavaNetURL : <NIL> ==> JavaNetURL
```

7.5 java.text クラス群

7.5.1 SimpleDateFormat クラス

```
public getClass : () ==> JavaLangClass
protected SimpleDateFormat : () ==> SimpleDateFormat
public format : java.util.Date ==> java.lang.String
public SimpleDateFormat : <NIL> ==> SimpleDateFormat
```

7.5.2 DecimalFormat クラス

```
public static currentSerialVersion : int := 2;
public static DOUBLE_INTEGER_DIGITS : int := 309;
public static DOUBLE_FRACTION_DIGITS : int := 340;
public clone : () ==> java.lang.Object
public getClass : () ==> JavaLangClass
public hashCode : () ==> int
public toPattern : () ==> java.lang.String
public DecimalFormat : () ==> DecimalFormat
public getMultiplier : () ==> int
public equals : java.lang.Object ==> bool
public setMultiplier : int ==> ()
public getGroupingSize : () ==> int
public setGroupingSize : int ==> ()
public getNegativePrefix : () ==> java.lang.String
public getNegativeSuffix : () ==> java.lang.String
public getPositivePrefix : () ==> java.lang.String
public getPositiveSuffix : () ==> java.lang.String
public DecimalFormat : <NIL> ==> DecimalFormat
public toLocalizedPattern : () ==> java.lang.String
public applyPattern : java.lang.String ==> ()
public DecimalFormat :
    java.lang.String ==> DecimalFormat
public getDecimalFormatSymbols :
    () ==> DecimalFormatSymbols
public setMaximumIntegerDigits : int ==> ()
public setMinimumIntegerDigits : int ==> ()
public setMaximumFractionDigits : int ==> ()
public setMinimumFractionDigits : int ==> ()
public setNegativePrefix : java.lang.String ==> ()
public setNegativeSuffix : java.lang.String ==> ()
```

```
public setPositivePrefix : JavaLangString ==> ()
public setPositiveSuffix : JavaLangString ==> ()
public applyLocalizedPattern : JavaLangString ==> ()
public isDecimalSeparatorAlwaysShown : () ==> bool
public parse : JavaLangString * JavaTextParsePosition ==> JavaLangNumber
public setDecimalSeparatorAlwaysShown : bool ==> ()
public format'' : real * JavaLangStringBuffer *
    JavaTextFieldPosition ==> JavaLangStringBuffer
public JavaTextDecimalFormat : JavaLangString *
    JavaTextDecimalFormatSymbols ==> JavaTextDecimalFormat
public setDecimalFormatSymbols : JavaTextDecimalFormatSymbols ==> ()
```

7.5.3 JavaTextDecimalFormatSymbols クラス

```
public clone : () ==> JavaLangObject
public getNaN : () ==> JavaLangString
public getClass : () ==> JavaLangClass
public getDigit : () ==> char
public hashCode : () ==> int
public setDigit : char ==> ()
public getPerMill : () ==> char
public getPercent : () ==> char
public getInfinity : () ==> JavaLangString
public setPerMill : char ==> ()
public setPercent : char ==> ()
public getMinusSign : () ==> char
public getZeroDigit : () ==> char
public setMinusSign : char ==> ()
public setZeroDigit : char ==> ()
public equals : JavaLangObject ==> bool
public setNaN : JavaLangString ==> ()
public getCurrencySymbol : () ==> JavaLangString
public getDecimalSeparator : () ==> char
public getPatternSeparator : () ==> char
public setInfinity : JavaLangString ==> ()
public JavaTextDecimalFormatSymbols : () ==> JavaTextDecimalFormatSymbols
public getGroupingSeparator : () ==> char
public setDecimalSeparator : char ==> ()
public setPatternSeparator : char ==> ()
public setGroupingSeparator : char ==> ()
public JavaTextDecimalFormatSymbols :
    <NIL> ==> JavaTextDecimalFormatSymbols
```

```
public setCurrencySymbol : JavaLangString ==> ()
public getMonetaryDecimalSeparator : () ==> char
public JavaTextDecimalFormatSymbols :
    JavaUtilLocale ==> JavaTextDecimalFormatSymbols
public setMonetaryDecimalSeparator : char ==> ()
public getInternationalCurrencySymbol : () ==> JavaLangString
public setInternationalCurrencySymbol : JavaLangString ==> ()
public getClass : () ==> JavaLangClass
```

7.5.4 JavaTextFieldPosition クラス

```
public getField : () ==> int
public hashCode : () ==> int
public toString : () ==> JavaLangString
public getEndIndex : () ==> int
public setEndIndex : int ==> ()
public getBeginIndex : () ==> int
public JavaTextFieldPosition : int ==> JavaTextFieldPosition
public equals : JavaLangObject ==> bool
public setBeginIndex : int ==> ()
public JavaTextFieldPosition : <NIL> ==> JavaTextFieldPosition
```

7.5.5 JavaTextFormat クラス

```
public clone : () ==> JavaLangObject
public getClass : () ==> JavaLangClass
public JavaTextFormat : <NIL> ==> JavaTextFormat
public formatObject : JavaLangObject ==> JavaLangString
public parseObject : JavaLangString ==> JavaLangObject
public parseObject :
    JavaLangString * JavaTextParsePosition ==> JavaLangObject
public format' : JavaLangObject * JavaLangStringBuffer *
    JavaTextFieldPosition ==> JavaLangStringBuffer
```

7.5.6 JavaTextMessageFormat クラス

```
public clone : () ==> JavaLangObject
public getClass : () ==> JavaLangClass
```

```
public hashCode : () ==> int
public getLocale : () ==> JavaUtilLocale
public toPattern : () ==> JavaLangString
public getFormats : () ==> map int to JavaTextFormat
public parse : JavaLangString ==> map int to JavaLangObject
public equals : JavaLangObject ==> bool
public setLocale : JavaUtilLocale ==> ()
public JavaTextMessageFormat : <NIL> ==> JavaTextMessageFormat
public setFormat : int * JavaTextFormat ==> ()
public applyPattern : JavaLangString ==> ()
public JavaTextMessageFormat : JavaLangString ==> JavaTextMessageFormat
public setFormats : map int to JavaTextFormat ==> ()
public static format :
    JavaLangString * map int to JavaLangObject ==> JavaLangString
public parse : JavaLangString * JavaTextParsePosition ==>
    map int to JavaLangObject
public JavaTextMessageFormat : JavaLangString * JavaUtilLocale ==>
    JavaTextMessageFormat
public parseObject : JavaLangString * JavaTextParsePosition ==>
    JavaLangObject
public format' : JavaLangObject * JavaLangStringBuffer *
    JavaTextFieldPosition ==> JavaLangStringBuffer
public formatObjects : map int to JavaLangObject *
    JavaLangStringBuffer * JavaTextFieldPosition ==> JavaLangStringBuffer
```

7.5.7 JavaTextNumberFormat クラス

```
public static INTEGER_FIELD : int := 0;
public static FRACTION_FIELD : int := 1;
public static cachedLocaleData :
    JavaUtilHashtable := new JavaUtilHashtable(3);
public static NUMBERSTYLE : int := 0;
public static CURRENCYSTYLE : int := 1;
public static PERCENTSTYLE : int := 2;
public static SCIENTIFICSTYLE : int := 3;
public clone : () ==> JavaLangObject
public format : real ==> JavaLangString
public getClass : () ==> JavaLangClass
public hashCode : () ==> int
public static getInstance : () ==> JavaTextNumberFormat
public parse : JavaLangString ==> JavaLangNumber
public equals : JavaLangObject ==> bool
```



```
public isGroupingUsed : () ==> bool
public setGroupingUsed : bool ==> ()
public JavaTextNumberFormat : <NIL> ==> JavaTextNumberFormat
public static getNumberInstance : () ==> JavaTextNumberFormat
public static getPercentInstance : () ==> JavaTextNumberFormat
public isParseIntegerOnly : () ==> bool
public static getAvailableLocales : () ==> map int to JavaUtilLocale
public static getCurrencyInstance : () ==> JavaTextNumberFormat
public static getInstance : JavaUtilLocale ==> JavaTextNumberFormat
public setParseIntegerOnly : bool ==> ()
public static getInstance :
    JavaUtilLocale * int ==> JavaTextNumberFormat
public getMaximumIntegerDigits : () ==> int
public getMinimumIntegerDigits : () ==> int
public getMaximumFractionDigits : () ==> int
public getMinimumFractionDigits : () ==> int
public setMaximumIntegerDigits : int ==> ()
public setMinimumIntegerDigits : int ==> ()
public static getNumberInstance :
    JavaUtilLocale ==> JavaTextNumberFormat
public setMaximumFractionDigits : int ==> ()
public setMinimumFractionDigits : int ==> ()
public static getPercentInstance :
    JavaUtilLocale ==> JavaTextNumberFormat
public static getCurrencyInstance :
    JavaUtilLocale ==> JavaTextNumberFormat
public parse :
    JavaLangString * JavaTextParsePosition ==> JavaLangNumber
public parseObject :
    JavaLangString * JavaTextParsePosition ==> JavaLangObject
public format'' : real * JavaLangStringBuffer *
    JavaTextFieldPosition ==> JavaLangStringBuffer
public format' : JavaLangObject * JavaLangStringBuffer *
    JavaTextFieldPosition ==> JavaLangStringBuffer
```

7.5.8 JavaTextParseException クラス

```
public getClass : () ==> JavaLangClass
public getErrorOffset : () ==> int
public JavaTextParseException : <NIL> ==> JavaTextParseException
public JavaTextParseException :
    JavaLangString * int ==> JavaTextParseException
```

7.5.9 JavaTextParsePosition クラス

```
public index : int := 0;
public errorIndex : int := -1
public getClass : () ==> JavaLangClass
public getIndex : () ==> int
public hashCode : () ==> int
public toString : () ==> JavaLangString
public setIndex : int ==> ()
public getErrorIndex : () ==> int
public JavaTextParsePosition : int ==> JavaTextParsePosition
public equals : JavaLangObject ==> bool
public setErrorIndex : int ==> ()
public JavaTextParsePosition : <NIL> ==> JavaTextParsePosition
```

7.5.10 JavaTextSimpleDateFormat クラス

```
public getClass : () ==> JavaLangClass
public JavaTextSimpleDateFormat : <NIL> ==> JavaTextSimpleDateFormat
public JavaTextSimpleDateFormat :
    JavaLangString ==> JavaTextSimpleDateFormat
```

7.6 java.sql クラス群

7.6.1 JavaSqlConnection クラス

```
public static TRANSACTION_NONE : int :=0;
public static TRANSACTION_READ_UNCOMMITTED : int :=0;
public static TRANSACTION_READ_COMMITTED : int :=0;
public static TRANSACTION_REPEATABLE_READ : int :=0;
public static TRANSACTION_SERIALIZABLE : int :=0
public JavaSqlConnection : <NIL> ==> JavaSqlConnection
public close : () ==> ()
public commit : () ==> ()
public getClass : () ==> JavaLangClass
public isClosed : () ==> bool
public rollback : () ==> ()
public getCatalog : () ==> JavaLangString
public getTypeMap : () ==> JavaUtilMap
```

```
public  isReadOnly : () ==> bool
public  setReadOnly : bool ==> ()
public  clearWarnings : () ==> ()
public  getAutoCommit : () ==> bool
public  setAutoCommit : bool ==> ()
public  createStatement : () ==> JavaSqlStatement
public  nativeSQL : JavaLangString ==> JavaLangString
public  createStatement : int * int ==> JavaSqlStatement
public  setCatalog : JavaLangString ==> ()
public  getTransactionIsolation : () ==> int
public  setTransactionIsolation : int ==> ()
```

7.6.2 JavaSqlDriverManager クラス

```
public  getClass : () ==> JavaLangClass
public  static  getConnection : JavaLangString ==> JavaSqlConnection
public  static  getConnection : JavaLangString * JavaLangString *
                                JavaLangString ==> JavaSqlConnection
```

7.6.3 JavaSqlResultSet クラス

```
public  static  FETCH_FORWARD : int := 1000;
public  static  FETCH_REVERSE : int := 1001;
public  static  FETCH_UNKNOWN : int := 1002;
public  static  TYPE_FORWARD_ONLY : int := 1003;
public  static  TYPE_SCROLL_INSENSITIVE : int := 1004;
public  static  TYPE_SCROLL_SENSITIVE : int := 1005;
public  static  CONCUR_READ_ONLY : int := 1007;
public  static  CONCUR_UPDATABLE : int := 1008
public  last : () ==> bool
public  next : () ==> bool
public  close : () ==> ()
public  first : () ==> bool
public  getRow : () ==> int
public  isLast : () ==> bool
public  getInt : int ==> int
public  getType : () ==> int
public  isFirst : () ==> bool
public  wasNull : () ==> bool
public  getByte : int ==> int
```

```
public getClass : () ==> JavaLangClass
public getLong : int ==> int
public previous : () ==> bool
public absolute : int ==> bool
public afterLast : () ==> ()
public deleteRow : () ==> ()
public getBytes : int ==> map int to int
public getFloat : int ==> real
public getShort : int ==> int
public insertRow : () ==> ()
public relative : int ==> bool
public updateRow : () ==> ()
public getDouble : int ==> real
public getString : int ==> JavaLangString
public refreshRow : () ==> ()
public rowDeleted : () ==> bool
public rowUpdated : () ==> bool
public beforeFirst : () ==> ()
public getBoolean : int ==> bool
public isAfterLast : () ==> bool
public rowInserted : () ==> bool
public updateNull : int ==> ()
public getFetchSize : () ==> int
public getStatement : () ==> JavaSqlStatement
public updateInt : int * int ==> ()
public clearWarnings : () ==> ()
public getCursorName : () ==> JavaLangString
public isBeforeFirst : () ==> bool
public setFetchSize : int ==> ()
public updateByte : int * int ==> ()
public updateLong : int * int ==> ()
public getConcurrency : () ==> int
public getInt : JavaLangString ==> int
public updateFloat : int * real ==> ()
public updateShort : int * int ==> ()
public getByte : JavaLangString ==> int
public getLong : JavaLangString ==> int
public moveToInsertRow : () ==> ()
public updateDouble : int * real ==> ()
public cancelRowUpdates : () ==> ()
public getBytes : JavaLangString ==> map int to int
public getFloat : JavaLangString ==> real
public getShort : JavaLangString ==> int
public moveToCurrentRow : () ==> ()
```

```
public updateBoolean : int * bool ==> ()
public getDouble : JavaLangString ==> real
public getFetchDirection : () ==> int
public getObject : JavaLangString ==> JavaLangObject
public getString : JavaLangString ==> JavaLangString
public findColumn : JavaLangString ==> int
public getBoolean : JavaLangString ==> bool
public setFetchDirection : int ==> ()
public updateBytes : int * map int to int ==> ()
public updateNull : JavaLangString ==> ()
public updateInt : JavaLangString * int ==> ()
public updateByte : JavaLangString * int ==> ()
public updateLong : JavaLangString * int ==> ()
public updateFloat : JavaLangString * real ==> ()
public updateShort : JavaLangString * int ==> ()
public updateDouble : JavaLangString * real ==> ()
public updateObject : int * JavaLangObject ==> ()
public updateString : int * JavaLangString ==> ()
public updateBoolean : JavaLangString * bool ==> ()
public updateBytes : JavaLangString * map int to int ==> ()
public updateString : JavaLangString * JavaLangString ==> ()
```

7.6.4 JavaSqlSQLException クラス

```
public getClass : () ==> JavaLangClass
public getSQLState : () ==> JavaLangString
public JavaSqlSQLException : () ==> JavaSqlSQLException
public getErrorCode : () ==> int
public getNextException : () ==> JavaSqlSQLException
public JavaSqlSQLException : <NIL> ==> JavaSqlSQLException
public JavaSqlSQLException : JavaLangString ==> JavaSqlSQLException
public JavaSqlSQLException :
    JavaLangString * JavaLangString ==> JavaSqlSQLException
public setNextException : JavaSqlSQLException ==> ()
public JavaSqlSQLException : JavaLangString *
    JavaLangString * int ==> JavaSqlSQLException
```

7.6.5 SqlConnection クラス

```
public static TRANSACTION_NONE : int := 0;
```

```
public static TRANSACTION_READ_UNCOMMITTED : int := 0;
public static TRANSACTION_READ_COMMITTED : int := 0;
public static TRANSACTION_REPEATABLE_READ : int := 0;
public static TRANSACTION_SERIALIZABLE : int := 0
public close : () ==> ()
public commit : () ==> ()
public getClass : () ==> JavaLangClass
public isClosed : () ==> bool
public rollback : () ==> ()
public getCatalog : () ==> JavaLangString
public getTypeMap : () ==> JavaUtilMap
public isReadOnly : () ==> bool
public setReadOnly : bool ==> ()
public clearWarnings : () ==> ()
public getAutoCommit : () ==> bool
public setAutoCommit : bool ==> ()
public createStatement : () ==> JavaSqlStatement
public nativeSQL : JavaLangString ==> JavaLangString
public createStatement : int * int ==> JavaSqlStatement
public setCatalog : JavaLangString ==> ()
public getTransactionIsolation : () ==> int
public setTransactionIsolation : int ==> ()
```

7.6.6 SqlResultSet クラス

```
public getColumns : () ==> map seq of char to int * int * int * int
public last : () ==> bool
public next : () ==> bool
public close : () ==> ()
public first : () ==> bool
public getRow : () ==> int
public isLast : () ==> bool
public getInt : int ==> int
public getType : () ==> int
public isFirst : () ==> bool
public wasNull : () ==> bool
public getByte : int ==> int
public getClass : () ==> JavaLangClass
public getLong : int ==> int
public previous : () ==> bool
public absolute : int ==> bool
public afterLast : () ==> ()
```

```
public deleteRow : () ==> ()
public getBytes : int ==> map int to int
public getFloat : int ==> real
public getShort : int ==> int
public insertRow : () ==> ()
public relative : int ==> bool
public updateRow : () ==> ()
public getDouble : int ==> real
public getString : int ==> JavaLangString
public refreshRow : () ==> ()
public rowDeleted : () ==> bool
public rowUpdated : () ==> bool
public beforeFirst : () ==> ()
public getBoolean : int ==> bool
public isAfterLast : () ==> bool
public rowInserted : () ==> bool
public updateNull : int ==> ()
public getFetchSize : () ==> int
public getStatement : () ==> JavaSqlStatement
public updateInt : int * int ==> ()
public clearWarnings : () ==> ()
public getCursorName : () ==> JavaLangString
public isBeforeFirst : () ==> bool
public setFetchSize : int ==> ()
public updateByte : int * int ==> ()
public updateLong : int * int ==> ()
public getConcurrency : () ==> int
public getInt : JavaLangString ==> int
public updateFloat : int * real ==> ()
public updateShort : int * int ==> ()
public getByte : JavaLangString ==> int
public getLong : JavaLangString ==> int
public moveToInsertRow : () ==> ()
public updateDouble : int * real ==> ()
public cancelRowUpdates : () ==> ()
public getBytes : JavaLangString ==> map int to int
public getFloat : JavaLangString ==> real
public getShort : JavaLangString ==> int
public moveToCurrentRow : () ==> ()
public updateBoolean : int * bool ==> ()
public getDouble : JavaLangString ==> real
public getFetchDirection : () ==> int
public getObject : JavaLangString ==> JavaLangObject
public getString : JavaLangString ==> JavaLangString
```

```
public findColumn : JavaLangString ==> int
public getBoolean : JavaLangString ==> bool
public setFetchDirection : int ==> ()
public updateBytes : int * map int to int ==> ()
public updateNull : JavaLangString ==> ()
public updateInt : JavaLangString * int ==> ()
public updateByte : JavaLangString * int ==> ()
public updateLong : JavaLangString * int ==> ()
public updateFloat : JavaLangString * real ==> ()
public updateShort : JavaLangString * int ==> ()
public updateDouble : JavaLangString * real ==> ()
public updateObject : int * JavaLangObject ==> ()
public updateString : int * JavaLangString ==> ()
public updateBoolean : JavaLangString * bool ==> ()
public updateBytes : JavaLangString * map int to int ==> ()
public updateString : JavaLangString * JavaLangString ==> ()
```

7.6.7 SqlStatement クラス

```
public close : () ==> ()
public getClass : () ==> JavaLangClass
public executeQuery : JavaLangString ==> JavaSqlResultSet
public executeUpdate : JavaLangString ==> int
```

7.6.8 JavaSqlStatement クラス

```
public close : () ==> ()
public cancel : () ==> ()
public getClass : () ==> JavaLangClass
public clearBatch : () ==> ()
public getMaxRows : () ==> int
public setMaxRows : int ==> ()
public executeBatch : () ==> map int to int
public getFetchSize : () ==> int
public getResultSet : () ==> JavaSqlResultSet
public clearWarnings : () ==> ()
public getConnection : () ==> JavaSqlConnection
public setFetchSize : int ==> ()
public getMoreResults : () ==> bool
public getUpdateCount : () ==> int
```



```
public execute : JavaLangString ==> bool
public getMaxFieldSize : () ==> int
public getQueryTimeout : () ==> int
public addBatch : JavaLangString ==> ()
public getResultSetType : () ==> int
public setMaxFieldSize : int ==> ()
public setQueryTimeout : int ==> ()
public getFetchDirection : () ==> int
public setFetchDirection : int ==> ()
public executeQuery : JavaLangString ==> java.sql.ResultSet
public setEscapeProcessing : bool ==> ()
public executeUpdate : JavaLangString ==> int
public setCursorName : JavaLangString ==> ()
public getResultSetConcurrency : () ==> int
```

参考文献

- [1] CSK. *The Dynamic Link Facility*. CSK.
- [2] CSK. The vdm++ to java code generator. Tech. rep.
- [3] CSK. *VDM++ Toolbox User Manual*. CSK.