

# VDMTools

---

**The Java to VDM++ User Manual**  
ver.1.0



**How to contact:**

<http://fmvdm.org/>

<http://fmvdm.org/tools/vdmttools>

[inq@fmvdm.org](mailto:inq@fmvdm.org)

VDM information web site(in Japanese)

VDMTools web site(in Japanese)

Mail

*The Java to VDM++ User Manual 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

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Including Java Classes in a Project</b>	<b>1</b>
<b>3</b>	<b>Options to the Translation</b>	<b>8</b>
<b>4</b>	<b>Limitations</b>	<b>9</b>
4.1	Scope Differences between Java and VDM++ . . . . .	9
4.1.1	Class Names . . . . .	10
4.1.2	Name Conflicts . . . . .	10
4.1.3	The scope of a class member . . . . .	11
4.1.4	Access to instance variables of a subclass . . . . .	12
4.2	Restrictions on Statements with Side Effects . . . . .	14
4.2.1	Initialization of class or instance variables . . . . .	14
4.2.2	Conditional Expressions on the Left-hand side of Assignments . . . . .	14
4.2.3	JavaLangObject Member Access . . . . .	15
4.3	Language Construct Differences . . . . .	15
4.3.1	Java vs. VDM++ Keywords . . . . .	15
4.3.2	Assignment to Function Parameters . . . . .	15
4.3.3	Using Assignment to new Instances as an Expression . . . . .	16
4.3.4	Numeric Types . . . . .	16
4.3.5	Type Conversions . . . . .	17
4.3.6	Inner Classes . . . . .	18
4.3.7	Implicit use of Qualified <b>this</b> . . . . .	18
4.3.8	Anonymous Classes . . . . .	18
4.3.9	<b>label</b> , <b>break</b> and <b>continue</b> . . . . .	19
4.3.10	<b>switch</b> without Alternative Breaks . . . . .	19
4.4	Unsupported Concepts . . . . .	20
4.4.1	Concurrency . . . . .	20
4.4.2	Unicode Characters . . . . .	20
<b>5</b>	<b>Details of the Java to VDM++ Translation</b>	<b>21</b>
5.1	Built-in Types . . . . .	21
5.2	Literals . . . . .	21
5.3	Names . . . . .	22
5.4	Arrays . . . . .	22
5.5	Classes . . . . .	23
5.5.1	Methods and Members . . . . .	23
5.5.2	Inheritance . . . . .	24
5.5.3	Modifiers of a Class . . . . .	24

5.5.4	Access modifiers . . . . .	24
5.5.5	Static initialisers . . . . .	25
5.5.6	Getting information about a class . . . . .	26
5.6	Interfaces . . . . .	28
5.7	<code>null</code> . . . . .	28
5.8	Expressions . . . . .	32
5.9	Statements . . . . .	33
5.9.1	If statements . . . . .	33
5.9.2	Block statement . . . . .	34
5.9.3	<code>for</code> statement . . . . .	34
5.9.4	<code>while</code> statement . . . . .	35
5.9.5	<code>do while</code> statement . . . . .	35
5.9.6	<code>switch</code> statement . . . . .	36
5.9.7	<code>try catch</code> statement . . . . .	37
<b>6</b>	<b>VDM++ Transformations</b>	<b>39</b>
6.1	<code>isMapCompLoop</code> (Block Stmt) . . . . .	39
6.2	<code>ifTestTrue</code> (Block Stmt) . . . . .	40
6.3	<code>ifTestFalse</code> (Block Stmt) . . . . .	41
6.4	<code>isRedundantIfBlock</code> (Block Stmt) . . . . .	42
6.5	<code>isRedundantIfBlockNoElse</code> (Block Stmt) . . . . .	44
6.6	<code>ifToAnd</code> (Block Stmt) . . . . .	44
6.7	<code>isRedundantDcl</code> (Block Stmt) . . . . .	45
6.8	<code>ifToCases</code> (If Stmt) . . . . .	46
6.9	<code>ifToEquiv</code> (If Stmt) . . . . .	47
6.10	<code>nestedIfsNoElises</code> (If Stmt) . . . . .	48
6.11	<code>whileIfTestTrue</code> (While Loop) . . . . .	49
6.12	<code>orToNotEquiv</code> (Binary Expr) . . . . .	50
<b>7</b>	<b>VDM++ model of Java API classes</b>	<b>51</b>
7.1	<code>java.lang</code> . . . . .	51
7.1.1	<code>JavaLangArrayIndexOutOfBoundsException</code> . . . . .	51
7.1.2	<code>JavaLangBoolean</code> . . . . .	51
7.1.3	<code>JavaLangCharacter</code> . . . . .	52
7.1.4	<code>JavaLangClass</code> . . . . .	53
7.1.5	<code>JavaLangClassCastException</code> . . . . .	53
7.1.6	<code>JavaLangClassNotFoundException</code> . . . . .	53
7.1.7	<code>JavaLangComparable</code> . . . . .	54
7.1.8	<code>JavaLangConversionBufferFullException</code> . . . . .	54
7.1.9	<code>JavaLangDouble</code> . . . . .	54
7.1.10	<code>JavaLangException</code> . . . . .	55

7.1.11	JavaLangIllegalAccessException	55
7.1.12	JavaLangIllegalArgumentException	55
7.1.13	JavaLangIllegalStateException	55
7.1.14	JavaLangIndexOutOfBoundsException	56
7.1.15	JavaLangInstantiationException	56
7.1.16	JavaLangInteger	56
7.1.17	J2VUTIL	57
7.1.18	JavaLangNullPointerException	57
7.1.19	Nullable	57
7.1.20	JavaLangNumber	58
7.1.21	JavaLangNumberFormatException	58
7.1.22	JavaLangObject	58
7.1.23	JavaLangRuntimeException	58
7.1.24	JavaLangString	59
7.1.25	JavaLangStringBuffer	60
7.1.26	JavaLangSystem	60
7.1.27	JavaLangThrowable	61
7.1.28	JavaLangUnsupportedOperationException	61
7.2	java.util	62
7.2.1	JavaUtilALitr	62
7.2.2	JavaUtilALListItr	62
7.2.3	JavaUtilAbstractCollection	62
7.2.4	JavaUtilAbstractList	62
7.2.5	JavaUtilAbstractMap	63
7.2.6	JavaUtilAbstractSet	64
7.2.7	JavaUtilCollection	64
7.2.8	JavaUtilConcurrentModificationException	64
7.2.9	JavaUtilDate	64
7.2.10	JavaUtilDictionary	65
7.2.11	JavaUtilEmptyEnumerator	65
7.2.12	JavaUtilEmptyIterator	66
7.2.13	JavaUtilEmptyStackException	66
7.2.14	JavaUtilEntry	66
7.2.15	JavaUtilEnumeration	66
7.2.16	JavaUtilHTEntry	67
7.2.17	JavaUtilHTKeySet	67
7.2.18	JavaUtilHashMap	67
7.2.19	JavaUtilHashSet	69
7.2.20	JavaUtilHashtable	69
7.2.21	JavaUtilIterator	70
7.2.22	JavaUtilList	70

7.2.23	JavaUtilListIterator	71
7.2.24	JavaUtilLocale	71
7.2.25	JavaUtilMap	72
7.2.26	JavaUtilMissingResourceException	72
7.2.27	JavaUtilNoSuchElementException	73
7.2.28	JavaUtilObservable	73
7.2.29	JavaUtilObserver	73
7.2.30	JavaUtilProperties	73
7.2.31	JavaUtilResourceBundle	74
7.2.32	JavaUtilSet	74
7.2.33	JavaUtilStack	75
7.2.34	JavaUtilStringTokenizer	75
7.2.35	JavaUtilVector	75
7.3	java.io	76
7.3.1	JavaIoBufferedInputStream	76
7.3.2	JavaIoBufferedOutputStream	77
7.3.3	JavaIoBufferedReader	77
7.3.4	JavaIoBufferedWriter	77
7.3.5	JavaIoByteArrayInputStream	78
7.3.6	JavaIoCharArrayReader	78
7.3.7	JavaIoFile	79
7.3.8	JavaIoFileDescriptor	80
7.3.9	JavaIoFileInputStream	80
7.3.10	JavaIoFileNotFoundException	81
7.3.11	JavaIoFileOutputStream	81
7.3.12	JavaIoFileReader	81
7.3.13	JavaIoFileSystem	81
7.3.14	JavaIoFileWriter	83
7.3.15	JavaIoFilterInputStream	83
7.3.16	JavaIoFilterOutputStream	83
7.3.17	JavaIoIOException	84
7.3.18	JavaIoInputStream	84
7.3.19	JavaIoInputStreamReader	84
7.3.20	JavaIoOutputStream	85
7.3.21	JavaIoOutputStreamWriter	85
7.3.22	JavaIoPrintStream	85
7.3.23	JavaIoPrintWriter	86
7.3.24	JavaIoReader	86
7.3.25	JavaIoStreamTokenizer	87
7.3.26	JavaIoStringWriter	87
7.3.27	JavaIoUnsupportedEncodingException	88

---

7.3.28	JavaIoWriter	88
7.4	java.net	88
7.4.1	JavaNetURL	88
7.5	java.text	89
7.5.1	JavaTextDateFormat	89
7.5.2	JavaTextDecimalFormat	89
7.5.3	JavaTextDecimalFormatSymbols	90
7.5.4	JavaTextFieldPosition	91
7.5.5	JavaTextFormat	91
7.5.6	JavaTextMessageFormat	91
7.5.7	JavaTextNumberFormat	92
7.5.8	JavaTextParseException	93
7.5.9	JavaTextParsePosition	94
7.5.10	JavaTextSimpleDateFormat	94
7.6	java.sql	94
7.6.1	JavaSqlConnection	94
7.6.2	JavaSqlDriverManager	95
7.6.3	JavaSqlResultSet	95
7.6.4	JavaSqlSQLException	97
7.6.5	SqlConnection	97
7.6.6	SqlResultSet	98
7.6.7	SqlStatement	100
7.6.8	JavaSqlStatement	100

---





# 1 Introduction

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.

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.

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.

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.


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.

Finally, Section 7 describes the subset of Java API which is available at the VDM++ level.

# 2 Including Java Classes in a Project

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.

To do this, first start up 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.

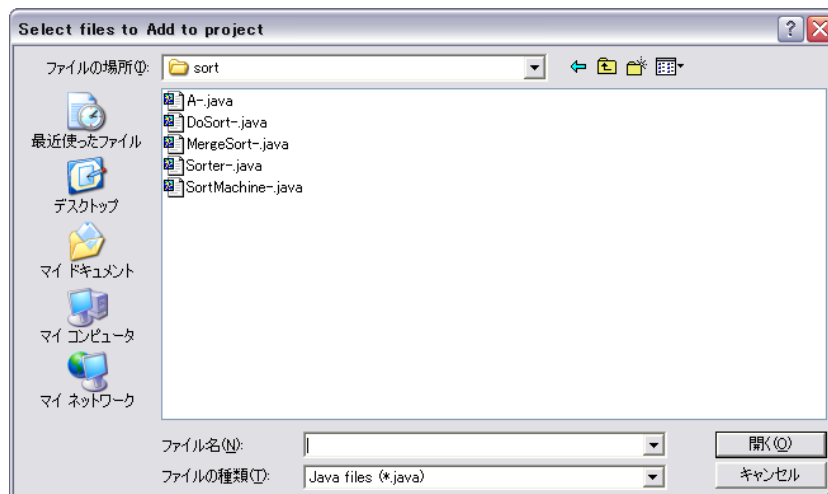


Figure 1: Adding Files to a Project

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.

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

Next you need to syntax check all your Java files, including the Java API skeleton files.

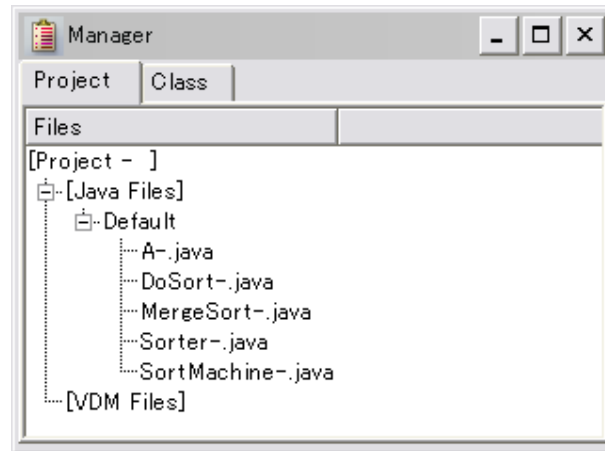




Figure 2: Manager after Addition of Files

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 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.

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.

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



Figure 3: The Java API Skeletons Needed

the individual Java classes in the project.

Figure 4 shows the current state of the **Java View**. The symbol **S** in the **Syntax** column next to each class indicates that it has been syntax checked successfully, and the symbol **T** 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 **S** respectively **T** are shown in the appropriate columns instead. In addition, if a source file is edited the symbol **S** (this is the symbol **S** 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.

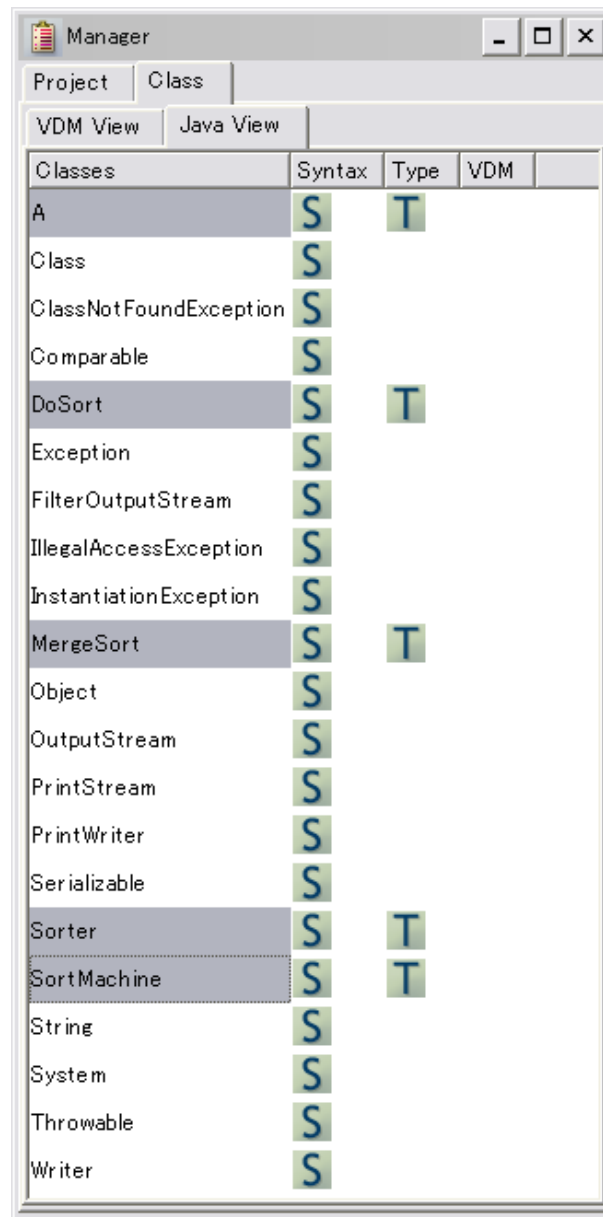





Figure 4: The Java View

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.

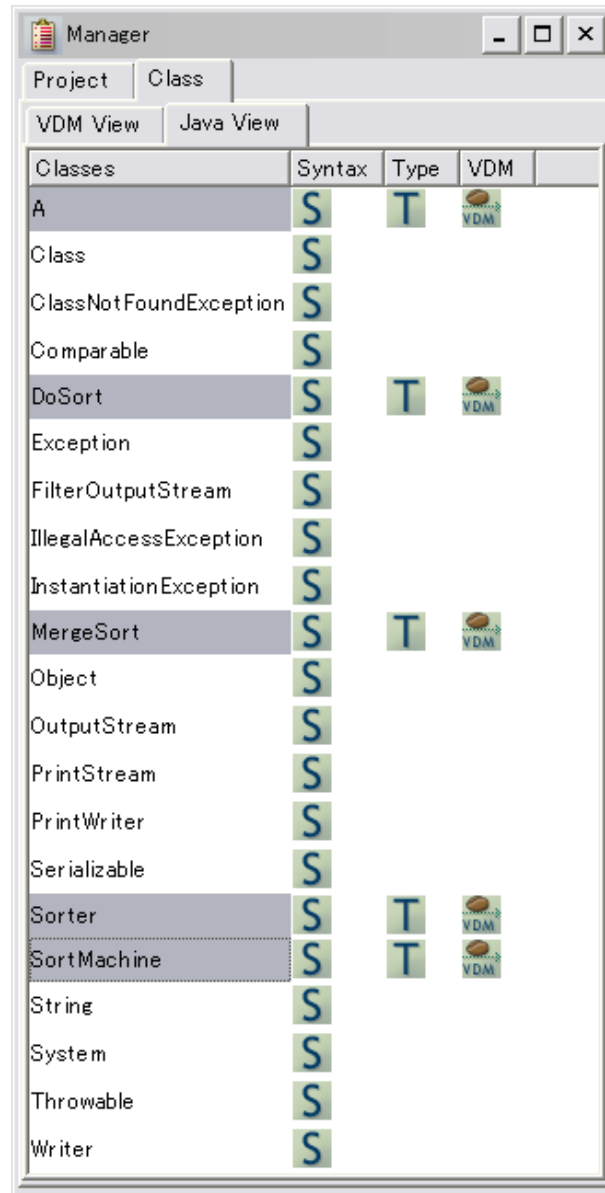


Figure 5: The Java View after Translation

This generates five .vpp files, one for each class. These can be seen by returning to the Project View (see Figure 6).

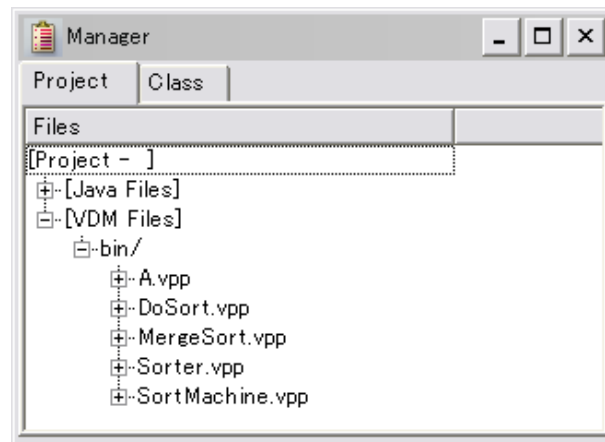



Figure 6: The Generated VDM++ Files

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.

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.

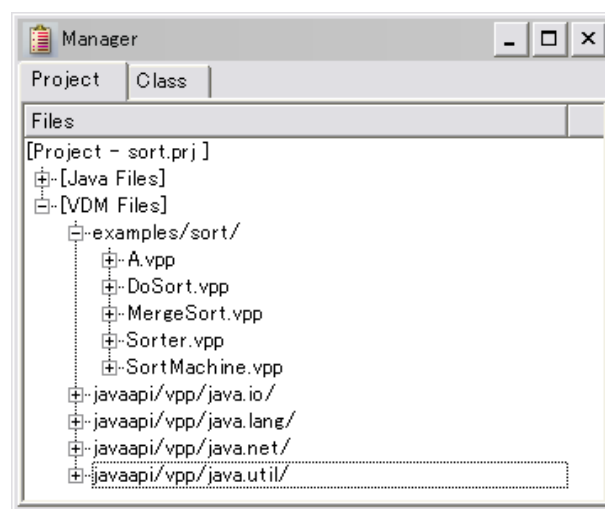



Figure 7: The VDM++ API Files

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 **is 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.)

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.

### 3 Options to the Translation

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.

The possible options are as follows:

**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;

**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.



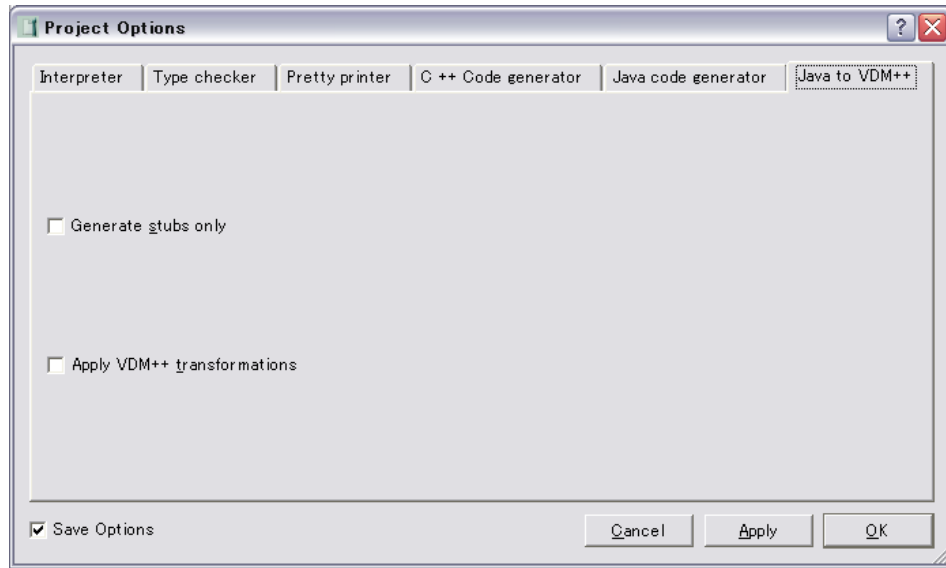


Figure 8: Setting Options for the Java to VDM++ Translator

## 4 Limitations

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.

### 4.1 Scope Differences between Java and VDM++

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.

#### 4.1.1 Class Names

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++.

#### 4.1.2 Name Conflicts

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.

```
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
```

```
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:

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

#### 4.1.3 The scope of a class member

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`:

```
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++:

```
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"

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 explicitly<sup>1</sup>.

#### 4.1.4 Access to instance variables of a subclass

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:

```
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);
  }
}
```

---

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

```
}  
}
```

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.

Compare this with the following VDM++ specification of the “same” example:

```
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 code<sup>2</sup>.

---

<sup>2</sup>A 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.

## 4.2 Restrictions on Statements with Side Effects

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.

### 4.2.1 Initialization of class or instance variables

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:

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

In the translator, however, we require that only expressions without side effects can be used in initialisation expressions<sup>3</sup>.

### 4.2.2 Conditional Expressions on the Left-hand side of Assignments

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:

```
(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++

---

<sup>3</sup>In 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.

statements<sup>4</sup>.

### 4.2.3 JavaLangObject Member Access

JavaLangObject member access expressions which involve expressions with side effects, for example:

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

also cannot in general be translated to VDM++ and are not currently accepted by the translator in any form<sup>5</sup>.

## 4.3 Language Construct Differences

In a number of areas the syntactic and semantic differences between Java and VDM++ cause problems with translation. We discuss these areas below.

### 4.3.1 Java vs. VDM++ Keywords

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.

### 4.3.2 Assignment to Function Parameters

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:

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

In VDM++, this is not possible so functions which include such assignments cannot be translated.

---

<sup>4</sup>Again in some special cases it would be possible to make a translation so it may be possible to relax this restriction in the future.

<sup>5</sup>Again simple forms could be translated so this restriction could possibly be relaxed in the future.

### 4.3.3 Using Assignment to new Instances as an Expression

In Java the assignment statement

```
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:

```
(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:

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

### 4.3.4 Numeric Types

In Java, the numeric types ‘int’, ‘long’ and ‘real’ are considered as being distinguishable, so that, for example, with the following definitions

```
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.

#### 4.3.5 Type Conversions

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

```
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.

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”.

```
... 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.

### 4.3.6 Inner Classes

In Java it is possible to nest class definitions using inner classes as in the following example:

```
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.

VDM++ does not support inner classes, nor the qualified `this`, so neither of these is supported by the Java to VDM++ translator<sup>6</sup>.

### 4.3.7 Implicit use of Qualified `this`

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.

### 4.3.8 Anonymous Classes

Anonymous classes in Java have no counterpart in VDM++ and are not supported by the Java to VDM++ translator<sup>7</sup>.

---

<sup>6</sup>It 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.

<sup>7</sup>It 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.

#### 4.3.9 label, break and 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.

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.

#### 4.3.10 switch without Alternative Breaks

In Java it is not necessary to put `break` between alternatives in a `switch` statement, so that the following is valid:

```
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:

```
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:

```
one=0;
two=0;

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

## 4.4 Unsupported Concepts

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.

### 4.4.1 Concurrency

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.

### 4.4.2 Unicode Characters

In Java it is possible to write arbitrary Unicode identifiers. This is not yet supported by the Java parser built into **VDMTools**.

## 5 Details of the Java to VDM++ Translation

This section explains how individual elements of a Java program are translated to VDM++.

### 5.1 Built-in Types

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

Figure 9: Transformations of Built-in Types

### 5.2 Literals

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 10.

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...")

Figure 10: Transformations of Literals

### 5.3 Names

Names in Java are translated to the same names in VDM++ with the following two exceptions:

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++.

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.

### 5.4 Arrays

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 11.

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

Figure 11: Transformations of Arrays

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.

## 5.5 Classes

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.

### 5.5.1 Methods and Members

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:

<pre> 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)   {...} </pre>	<pre> class A is subclass of JavaLangObject instance variables   i: int := 0;   static s : int := 1; operations   method : int ==&gt; int     ( return i );   static smethod : int ==&gt; int     ( return i );   op : () ==&gt; ()     ( ... );   static op1 : int ==&gt; int     ( ... ); </pre>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

```
}                                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.

### 5.5.2 Inheritance

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 classes which have no explicit inheritance clause are assumed to inherit implicitly from `java.lang.JavaLangObject`. The translator makes this dependency explicit, so that in VDM++ such a class explicitly inherits from `JavaLangObject`, the VDM++ counterpart of `java.lang.JavaLangObject`. The example in Section 5.5.1 illustrates this.

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.

### 5.5.3 Modifiers of a Class

Modifiers of a Java class (e.g. `abstract`, `final`) have no counterpart in VDM++ and are all ignored by the translator.

### 5.5.4 Access modifiers

Access modifiers are translated directly from Java to VDM++ as shown in the table in Figure 12. 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++.

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

Figure 12: Transformations of Access Modifiers

### 5.5.5 Static initialisers

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.

The following example illustrates this.

```

                                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 {                        static dummy : <VOID> := j2v_staticInitializer()
```

```

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

```

### 5.5.6 Getting information about a class

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.

The translator simulates part of this functionality using the class `JavaLangClass` which is defined as follows:

```

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.

```

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

```

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

end A
```

## 5.6 Interfaces

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++.

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:

```
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.

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.

## 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:

```
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.)

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`.

```
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:

```
ClassName: <NIL> ==> ClassName
```

```
ClassName() ==
```

```
    isNil := true;
```

To simulate the `null` value for arrays we use the empty map.

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

Figure 13: Transformations of `null`

The examples in the table in Figure 13 illustrate how `null` is translated in different contexts.

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:

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

This would be translated to the following in VDM++:

```
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:

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.

## 5.8 Expressions

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.

Consider the following fragment of an expression which has side effects in Java:

```
... 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.

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`.

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 preceeded by the statements generated for the right operand which change `n`.

The final result is as follows:

```
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 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:

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

## 5.9 Statements

### 5.9.1 If statements

The translation of **if** statements is straightforward. The component parts are simply translated as illustrated below:

JAVA:

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

VDM:

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

### 5.9.2 Block statement

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:

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

VDM:

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

### 5.9.3 for statement

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:

```
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 statement

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:

JAVA:

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

VDM:

```
( decl  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 statement

To translate a `do while` statement of the form

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

we convert it to the equivalent **while** statement

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

and translate this **while** statement.

### 5.9.6 switch statement

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:

```
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 statement

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:

```
class B extends java.lang.Exception { ... }

try { tryBody }
catch(B b) { catchBbody }
catch(java.lang.Exception e) { catchEbody }
finally{ finallyBody }
```

VDM:

```
class B is subclass of java.lang.Exception
...
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()) ->
      ( dcl b : JavaLangException := j2v_exception;
        translatedCatchEbody
      ) ,
    others ->
      exit j2v_exception
  end
in
translatedTryBody
```

## 6 VDM++ Transformations

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 14, 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.

Name	Applies To
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

Figure 14: List of Transformations

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.

### 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.

As an example, the following specification involving a while loop

```
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

```
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

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



into one which begins

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

For example, the following specification

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

is transformed to

```
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

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

into one which begins

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

For example, the following specification

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

is transformed to

```
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

```
( 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

```
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

```
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

```
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

```
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

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

to the single statement

```
return a and b;
```

Redundant blocks are also removed as in the following example

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

  );
```

which is transformed to

```
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

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

with the return statement

```
return y;
```

Thus, for example, the following specification

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

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

```
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

```
cases x :
```

```
a1 -> r1,  
a2 -> r2,  
a3 -> r3,  
...
```

At the same time, redundant blocks are removed. Thus, for example, the operation

```
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

```
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

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

to the form

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

As an example, the following operation

```
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

```
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

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

to a single if statement

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

Thus, for example, the operation



```
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

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

to one of the form

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

For example, the following operation

```
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

```
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

not a and b or a and not b

into the logically equivalent form

not (a <=> b)

At the same time, redundant brackets are removed. Thus, for example, the operation

```
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

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

## 7 VDM++ model of Java API classes

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.

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`.

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)`).

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.

### 7.1 java.lang

#### 7.1.1 JavaLangArrayIndexOutOfBoundsException

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

#### 7.1.2 JavaLangBoolean

```
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

```
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

```

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

```

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

```

#### 7.1.6 JavaLangClassNotFoundException

```

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

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

### 7.1.8 JavaLangConversionBufferFullException

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

### 7.1.9 JavaLangDouble

```
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

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

#### 7.1.11 JavaLangIllegalAccessException

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

#### 7.1.12 JavaLangIllegalArgumentException

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

#### 7.1.13 JavaLangIllegalStateException

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

#### 7.1.14 JavaLangIndexOutOfBoundsException

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

#### 7.1.15 JavaLangInstantiationException

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

#### 7.1.16 JavaLangInteger

```
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

```

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

```

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

```

#### 7.1.19 Nullable

```

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

```

### 7.1.20 JavaLangNumber

```
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

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

### 7.1.22 JavaLangObject

```
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

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

```
public JavaLangRuntimeException :
    JavaLangString ==> JavaLangRuntimeException
```

#### 7.1.24 JavaLangString

```
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

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

#### 7.1.26 JavaLangSystem

```
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

```

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

```

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

```

JavaLangString ==> JavaLangUnsupportedOperationException

## 7.2 java.util

### 7.2.1 JavaUtilALitr

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

### 7.2.2 JavaUtilALListItr

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

### 7.2.3 JavaUtilAbstractCollection

```
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

```
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

```

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

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

### 7.2.7 JavaUtilCollection

```
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

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

### 7.2.9 JavaUtilDate

```
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

```

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

```

### 7.2.11 JavaUtilEmptyEnumerator

```

public  nextElement : () ==> JavaLangObject

```

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

#### 7.2.12 JavaUtilEmptyIterator

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

#### 7.2.13 JavaUtilEmptyStackException

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

#### 7.2.14 JavaUtilEntry

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

#### 7.2.15 JavaUtilEnumeration

```
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

```
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

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

### 7.2.18 JavaUtilHashMap

```
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

```
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

```
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

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

#### 7.2.22 JavaUtilList

```
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

```
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

```
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

```
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

```
public  getKey : () ==> JavaLangString
public  getClassName : () ==> JavaLangString
```



```

public  JavaUtilMissingResourceException :
        <NIL> ==> JavaUtilMissingResourceException
public  JavaUtilMissingResourceException :
        JavaLangString * JavaLangString * JavaLangString ==>
                JavaUtilMissingResourceException

```

### 7.2.27 JavaUtilNoSuchElementException

```

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

```

### 7.2.28 JavaUtilObservable

```

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

```

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

```

### 7.2.30 JavaUtilProperties

```

protected  defaults : JavaUtilProperties;
public  JavaUtilProperties : () ==> JavaUtilProperties

```

```
public  propertyName : () ==> 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

```
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

```
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

```
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

```
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

```
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

### 7.3.1 JavaIoBufferedInputStream

```
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`

```

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`

```

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`

```

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

```
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

```
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

```

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

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

### 7.3.9 JavaIoFileInputStream

```
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

```

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

```

### 7.3.11 JavaIoFileOutputStream

```

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

```

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

```

### 7.3.13 JavaIoFileSystem

```

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

```

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

```

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

```

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

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

### 7.3.18 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 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

```
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 JavaIoOutputStream

```

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

```

### 7.3.21 JavaIoOutputStreamWriter

```

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  JavaIoOutputStreamWriter : <NIL> ==> JavaIoOutputStreamWriter
public  JavaIoOutputStreamWriter :
        JavaIoOutputStream ==> JavaIoOutputStreamWriter
public  JavaIoOutputStreamWriter :
        JavaIoOutputStream * JavaLangString ==> JavaIoOutputStreamWriter
public  CharToByteConverter : <NIL> ==> CharToByteConverter

```

### 7.3.22 JavaIoPrintStream

```

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  JavaIoPrintStream :
    JavaIoOutputStream * bool ==> JavaIoPrintStream
```

### 7.3.23 JavaIoPrintWriter

```
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

```
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

```

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

```

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

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

### 7.3.28 JavaIoWriter

```
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

### 7.4.1 JavaNetURL

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



## 7.5 java.text

### 7.5.1 SimpleDateFormat

```
public getClass : () ==> JavaLangClass
protected SimpleDateFormat : () ==> SimpleDateFormat
public format : JavaUtilDate ==> JavaLangString
public SimpleDateFormat : <NIL> ==> SimpleDateFormat
```

### 7.5.2 DecimalFormat

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

```
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

```

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

```

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

```

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

```
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

```

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

```

### 7.5.9 JavaTextParsePosition

```
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

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

## 7.6 java.sql

### 7.6.1 JavaSqlConnection

```
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

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

### 7.6.3 JavaSqlResultSet

```
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

```

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

```

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

```
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

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

#### 7.6.8 JavaSqlStatement

```
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 ==> JavaSqlResultSet
public  setEscapeProcessing : bool ==> ()
public  executeUpdate : JavaLangString ==> int
public  setCursorName : JavaLangString ==> ()
public  getResultSetConcurrency : () ==> int
```



## References

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