DirectJava®

Automated Smalltalk to Java Migration Solution

Olivier Picot (CEO, Object'ive)

Fabrice Le Calvez (Sales Manager, Object'ive)

October 24th, London JSIG

Program

1. Why Migrate ?
2. Differences between Java & Smalltalk
3. DirectJava : a 4-Step Translation Process
4. Code Translation Examples
5. Case Study

Who is Object'ive ?

- Founded in January 1999, staff of 12, 1 M€ turnover.
- Main technical skills : Java, Smalltalk, customised solutions.
- Areas of expertise : Migration, B2B & B2C Solutions, Mobility, eLearning
- References: mostly large corporates e.g.,
  - CCR: Caisse Centrale de Réassurance (largest re-insurer in France)
  - EDF: Electricité De France: largest eletricity utility company in France, 4- million customers in 24 countries
  - Veolia Water: largest water supply company in the world, 13 b€ turnover

Factors Driving Migration

SMALLTALK

- Vendor support services (Fear – Uncertainty – Doubt)
- Standardization decisions
- Expert Staffing Shortages
- Run-Time / Maintenance Expenses

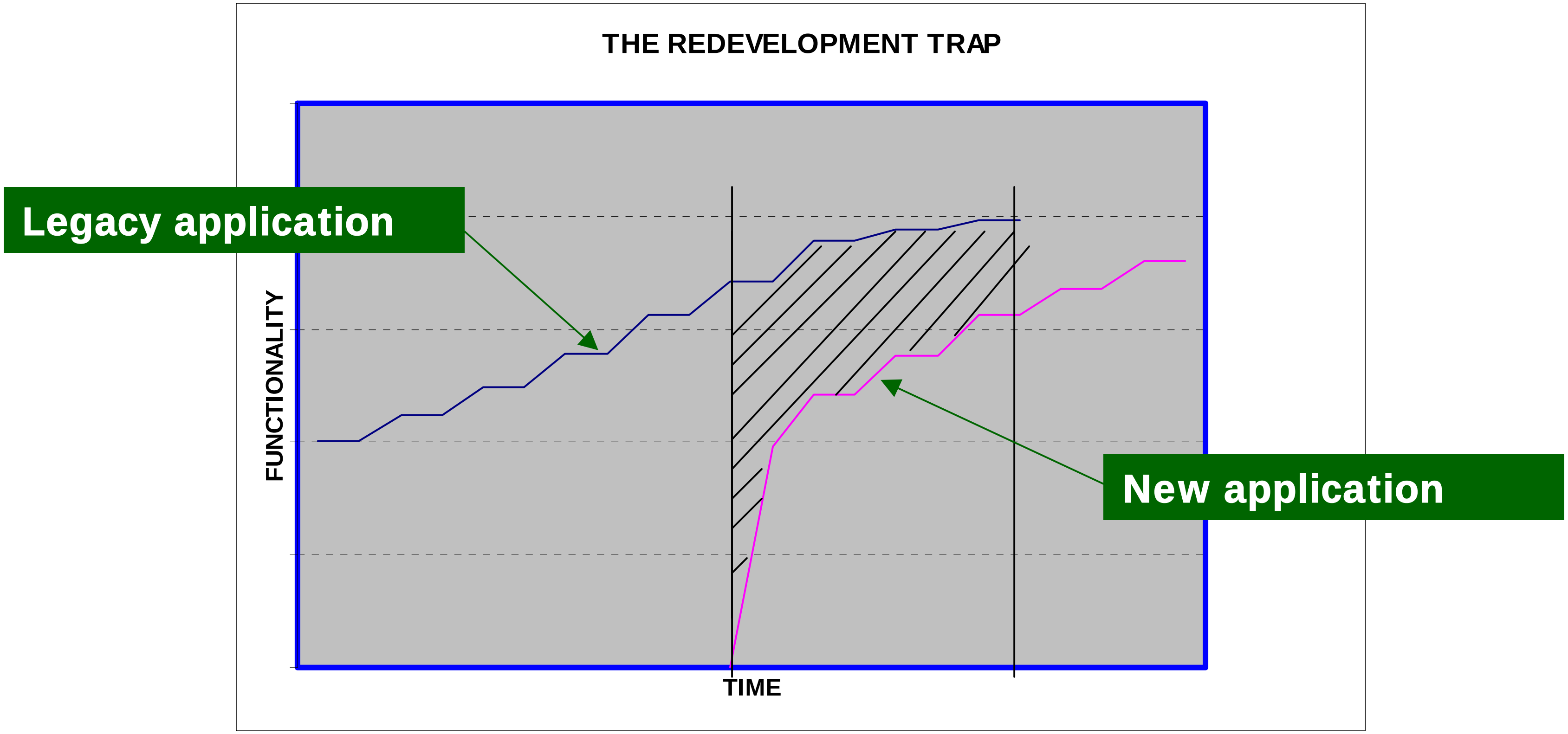
JAVA

- Vendor Independent
- Vibrant Developers’ Community , available software components
- Connectivity, Web-enablement, Mobility
- Distributed applications
- Fast development time
- Cost

Migration Components

- A Migration Project includes several parts such as:
  - Framework Migration
  - Views Migration
  - Functionality Migration
  - Other parts (Communication protocols, security, ...etc)
- Focus today: functionality migration & automation of language translation.
- Architecture and design issues :we will concentrate on these when translation choices, and the use of DirectJava has an impact on them.

The Cost of Re-Development



Legacy application

New application

A (non exhaustive) List of Problems

Dynamic typing in Smalltalk vs Static typing in Java

Multityping is « authorized » in Smalltalk

Java includes primitive types, in Smalltalk everything is object

```
multiTypeSample

" Dynamic typing and multitype sample "

      | i oc |
      i := 'hello'.
      i := 1.
      oc := OrderedCollection new.
      oc add: i.
      oc add: true.
      oc add: 3.
      oc add: 'world'
```

A (non exhaustive) List of Problems

Dynamic typing in Smalltalk vs Static typing in Java

Multityping is « authorized » in Smalltalk

Java includes primitive types, in Smalltalk everything is object

Blocks do not exist in Java

In Smalltalk, a method returns self by default

```
| dic |

dic := Dictionary new.
dic at: #key1 put: #val1.
dic at: #key2 put: #val2.
dic at: #key3 put: #val3.
dic keysAndValuesDo: [:k :v|
    | st |
    st := k printString , v printString
]

callToDefaultSelfReturnType

     | var |

       var := self testBlock.
       Transcript cr; show:
                    'var class = ' ,
                    var class printString,
                    ' even if testBlock does not return
anything'
```

A (non exhaustive) List of Problems

Dynamic typing in Smalltalk vs Static typing in Java

Multityping is « authorized » in Smalltalk

Java includes primitive types, in Smalltalk everything is object

Blocks do not exist in Java

In Smalltalk, a method returns self by default

Indices management starts at 1 in Smalltalk but at 0 in Java

```
indicesSamples: aString

" Sample which shows the indices managing (starting at 0 in
Java and 1 in Smalltalk "

      | oc anIndice obj |

      oc := self sampleReturnCollectionMethodWithYourself.
      ((aString size > 10) and: [aString size < 20])
      ifFalse: [
                  ^'Error'
      ]
      ifTrue: [
                  aString copyFrom: 10 to: 20
      ].

      anIndice := self getIndice.
      obj := oc at: 4.
      ^oc at: anIndice
```

A (non exhaustive) List of Problems

Dynamic typing in Smalltalk vs Static typing in Java

Multityping is « authorized » in Smalltalk

Java includes primitive types, in Smalltalk everything is object

Blocks do not exist in Java

In Smalltalk, a method returns self by default

Indices management starts at 1 in Smalltalk but at 0 in Java

No extending basic classes in Java, delegation must be used

Smalltalk has no constructors concept

Cascading messages do not exist in Java, no `yourself` message

```
sampleReturnCollectionMethodWithYourself

^OrderedCollection new
      add: 'a';
      add: 'b';
      add: 'c';
      add: 'd';
      yourself
```

A (non exhaustive) List of Problems

Dynamic typing in Smalltalk vs Static t

Multityping is « authorized » in Smallta

Java includes primitive types, in Small

Blocks do not exist in Java

In Smalltalk, a method returns self by

Indices management starts at 1 in Sm

No extending basic classes in Java, de

Smalltalk has no constructors concept

Cascading messages do not exist in Ja

No inheritance of static method in Java

No class instance variables in Java

No Pool Dictionaries in Java

```
Object subclass: #Class1
      Class1 class>>#foo1
                  ^ 'foo1'
      Class1>>#fooInst1
                  ^self class foo1

Class1 subclass: #Class2
      Class2>>#fooInst2
                  ^self fooInst1
      Class2 class>>#foo1
                  ^ 'foo2'
```

What about

Class2 new fooInst1 returns 'foo2' in Smalltalk

new Class2().fooInst1(); returns "foo1" in Java ?

A (non exhaustive) List of Problems

Dynamic typing in Smalltalk vs Static typing in Java
Multityping is « authorized » in Smalltalk
Java includes primitive types, in Smalltalk everything is object

Blocks do not exist in Java
In Smalltalk, a method returns self by default

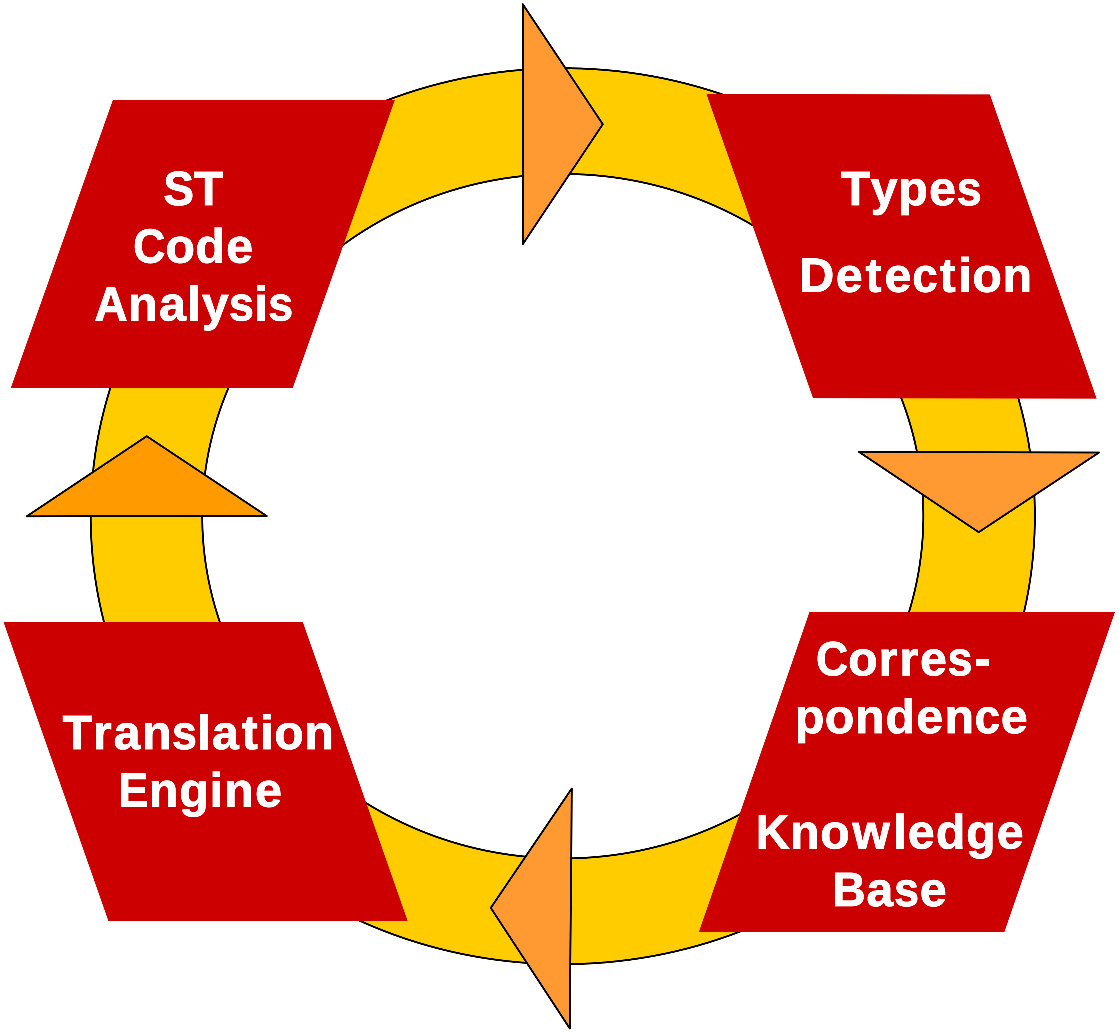
Indices management starts at 1 in Smalltalk but at 0 in Java

No extending basic classes in Java, delegation must be used
Smalltalk has no constructors concept
Cascading messages do not exist in Java, no `yourself` message

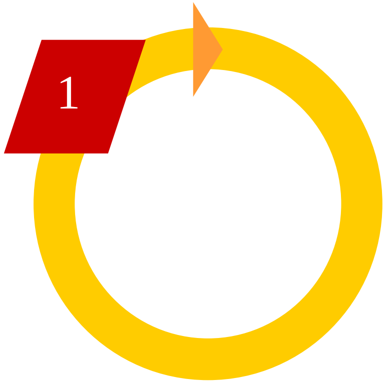
No inheritance of static method in Java
No class instance variables in Java
No Pool Dictionaries in Java

Java does not support become:
No package name concept in Smalltalk
No overloading in Smalltalk

DirectJava : 4-Step Iterative Process

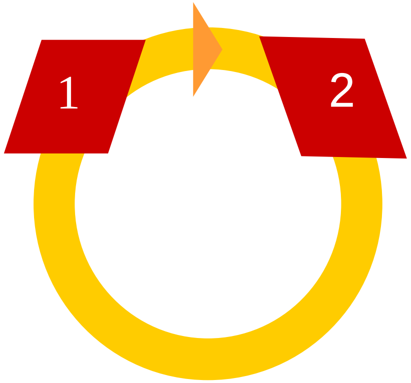


Automated Smalltalk Code Analysis



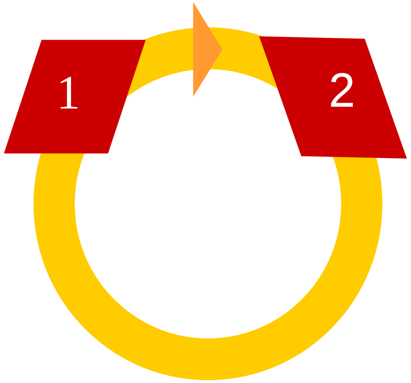
- Code volume analysis
- Class methods and instances methods with similar signatures.
- Quality Analysis
  - –Classes, Methods, Unlisted Variables
  - –Messages sent but not implemented
  - –Certain kinds of multityped Methods (return boolean and non boolean)
  - –Use of Methods without returns (SMT) but as if it was returning self by developpers (;yourself missing) etc…
  - –Variables read before written, written but never read
  - –Checking of temporary variables defined outside a block but affecting this particular block…
- Detection of class instances' variables.
- Detection of specific indices issues
- Duplicated code in subclasses

Types Detection



1. Recording and launching applications scenarios
2. Automatic type inference
3. Manual allocation of types

Types Detection



1. Recording and launching applications scenarios

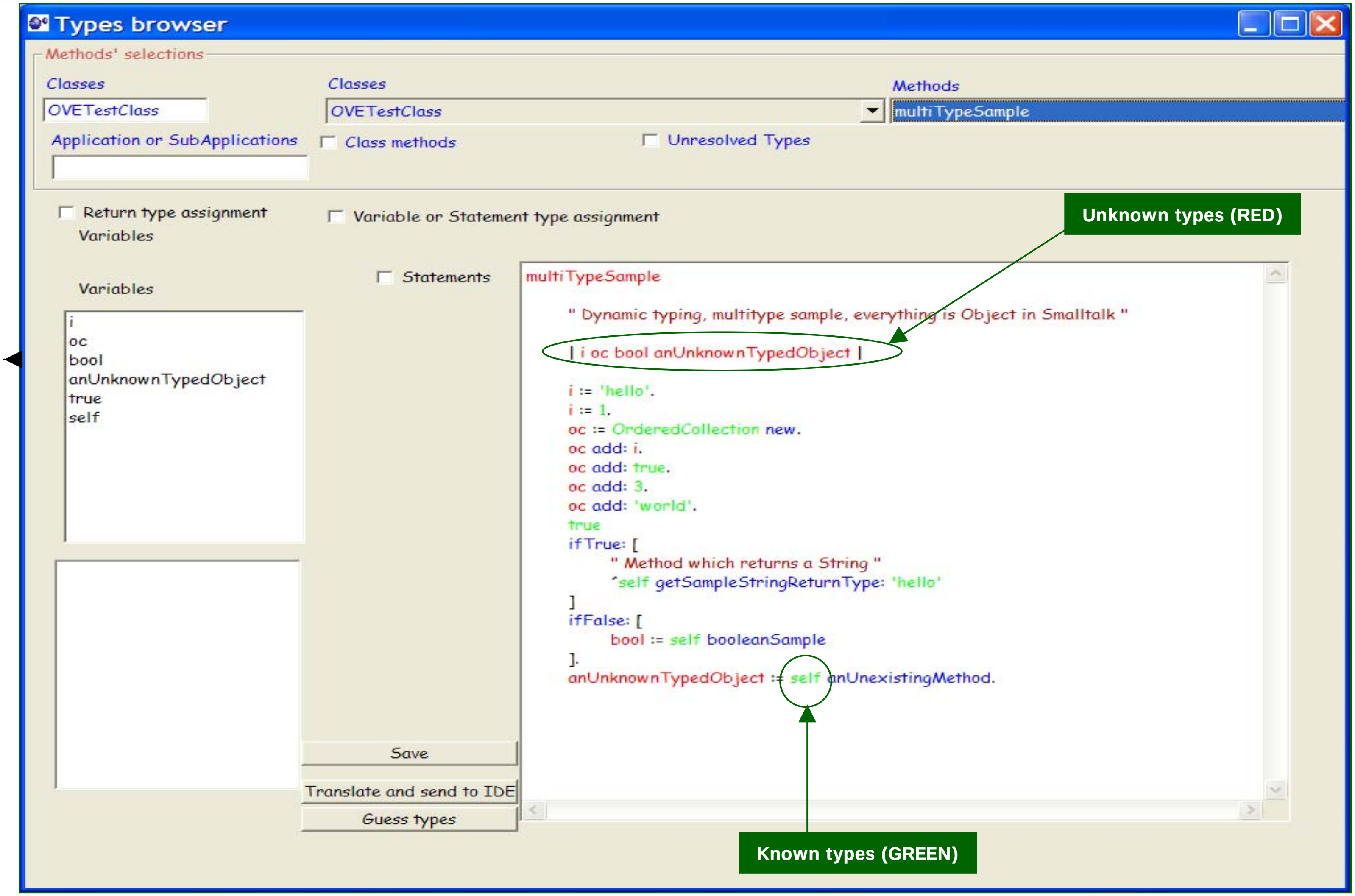
2. Automatic type inferen

3. Manual allocation o

At the end of this analysis, following types will have been identified :

- •Instance variables
- •Class variables
- •Temporary variables
- •Methods’ arguments
- •Methods’ return types
- •Statements types
- •Reporting on unknown or multi-valued types.
- •Methods with source code not fully verified

Before Scenario Launch



Classes: OVETestClass

Classes: OVETestClass

Methods: multiTypeSample

Application or SubApplications:

□ Class methods

□ Unresolved Types

□ Return type assignment

Variables

□ Variable or Statement type assignment

□ Statements

Variables

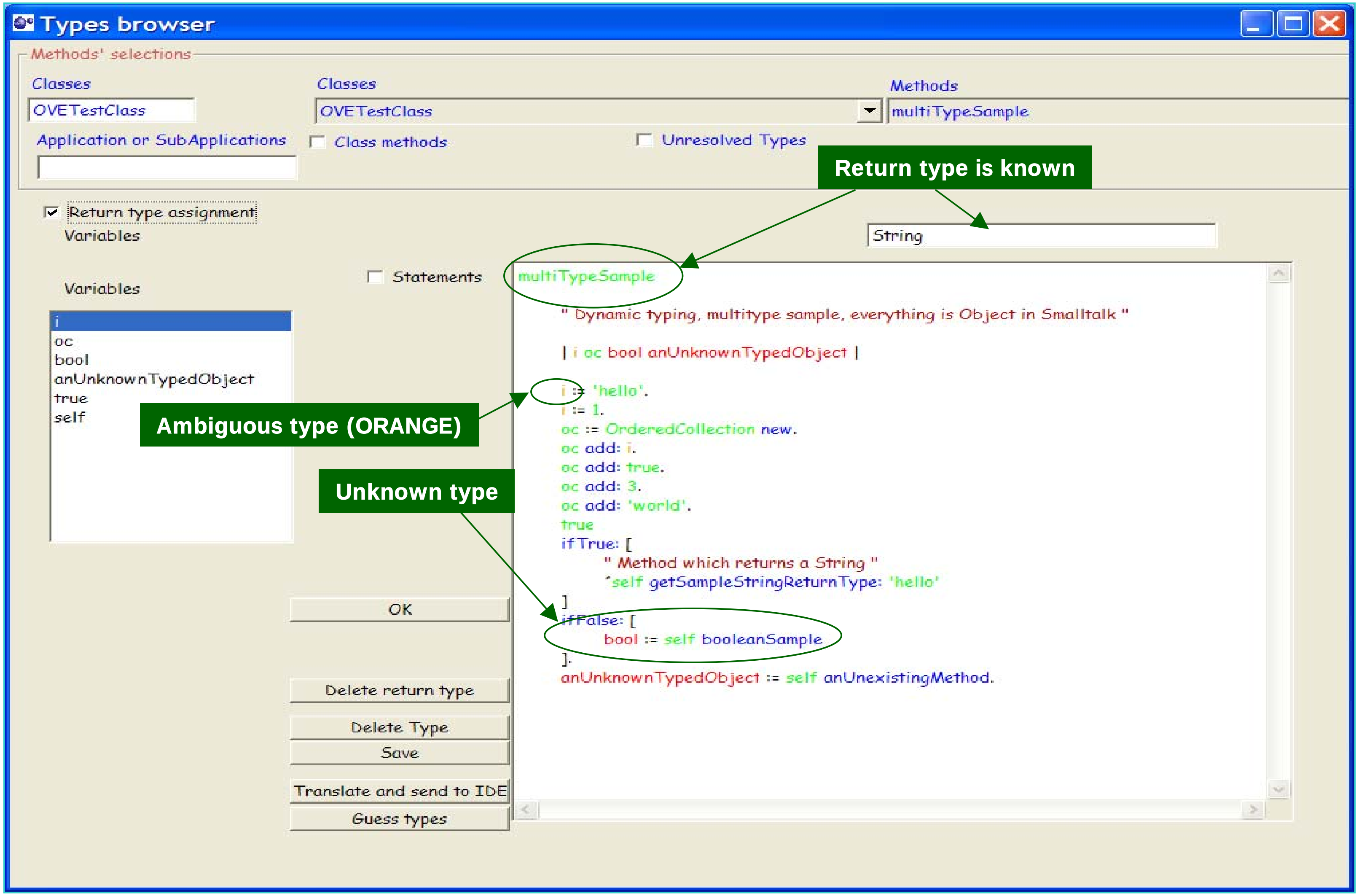
```
i
oc
bool
anUnknownTypedObject
true
self
```

```
multiTypeSample

    " Dynamic typing, multitype sample, everything is Object in Smalltalk "

    | i oc bool anUnknownTypedObject |

    i := 'hello'.
    i := 1.
    oc := OrderedCollection new.
    oc add: i.
    oc add: true.
    oc add: 3.
    oc add: 'world'.
    true
    ifTrue: [
        " Method which returns a String "
        ^self getSampleStringReturnType: 'hello'
    ]
    ifFalse: [
        bool := self booleanSample
    ].
    anUnknownTypedObject := self anUnexistingMethod.
```

Unknown types (RED)

Known types (GREEN)

After Scenario Launch



Types browser

Methods' selections

Classes: OVETestClass

Application or SubApplications:

Classes: OVETestClass

Methods: multiTypeSample

□ Class methods

□ Unresolved Types

☑ Return type assignment

String

Variables

□ Statements

Variables:

- i
- oc
- bool
- anUnknownTypedObject
- true
- self

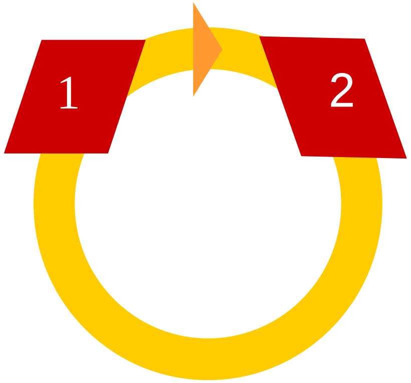
```
multiTypeSample

    " Dynamic typing, multitype sample, everything is Object in Smalltalk "

    | i oc bool anUnknownTypedObject |

    i := 'hello'.
    i := 1.
    oc := OrderedCollection new.
    oc add: i.
    oc add: true.
    oc add: 3.
    oc add: 'world'.
    true
    ifTrue: [
           " Method which returns a String "
           ^self getSampleStringReturnType: 'hello'
    ]
    ifFalse: [
           bool := self booleanSample
    ].
    anUnknownTypedObject := self anUnexistingMethod.
```

Return type is known

Ambiguous type (ORANGE)

Unknown type

Types Detection

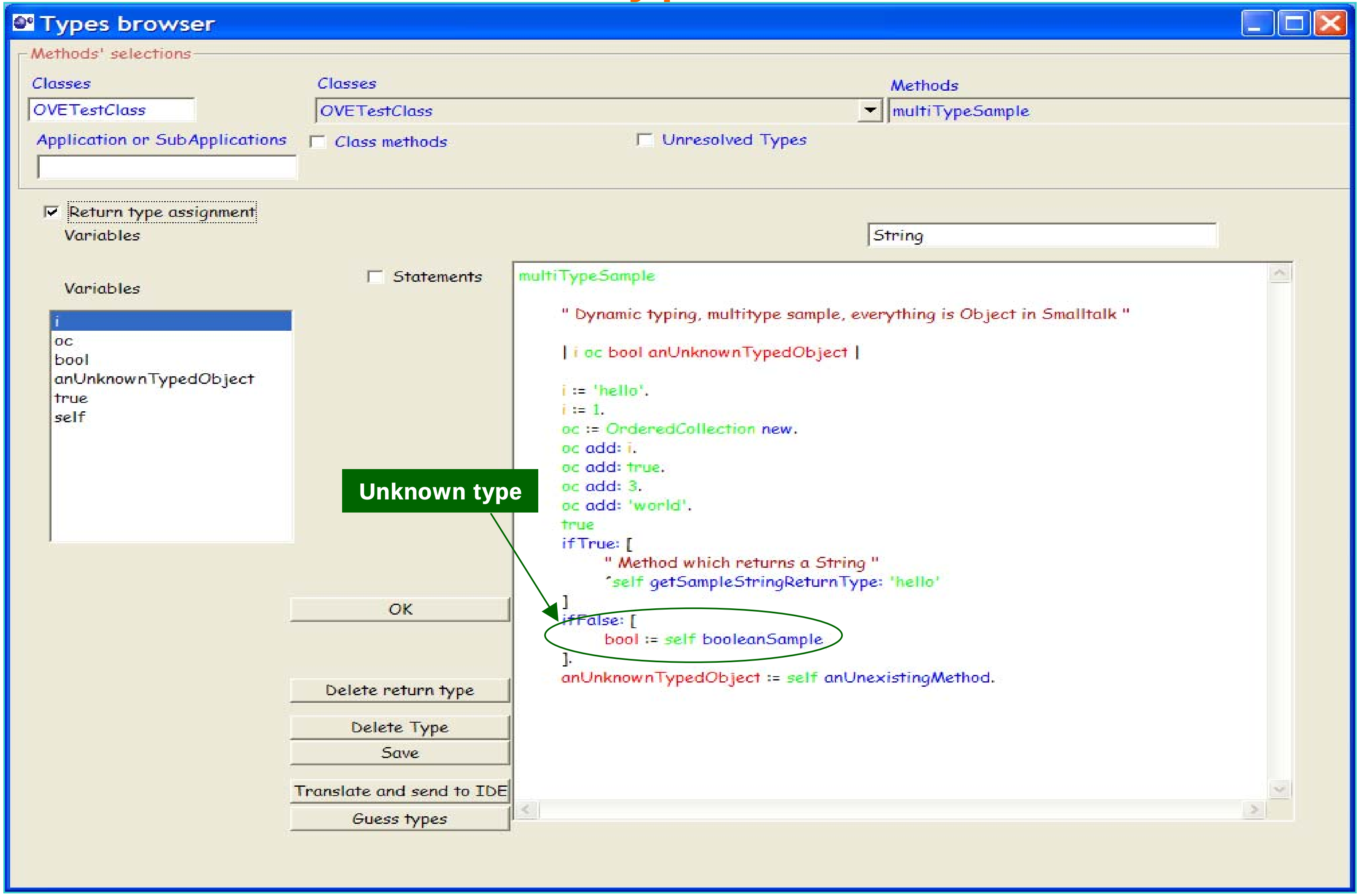


1. Recording and launching applications scenarios
2. **Automatic type inference**
3. Manual allocation of t

DirectJava inference engine is based on several principles such as:

- The constructors concept.
- Knowledge of methods return types called
- The concept of « SMALLEST COMMON ROOT EXCEPT OBJECT» for a group of methods called for the same receiver.

Before Types Inference



Methods' selections

Classes: OVETestClass

Classes: OVETestClass

Methods: multiTypeSample

Application or SubApplications:

□ Class methods

□ Unresolved Types

☑ Return type assignment

Variables: String

□ Statements

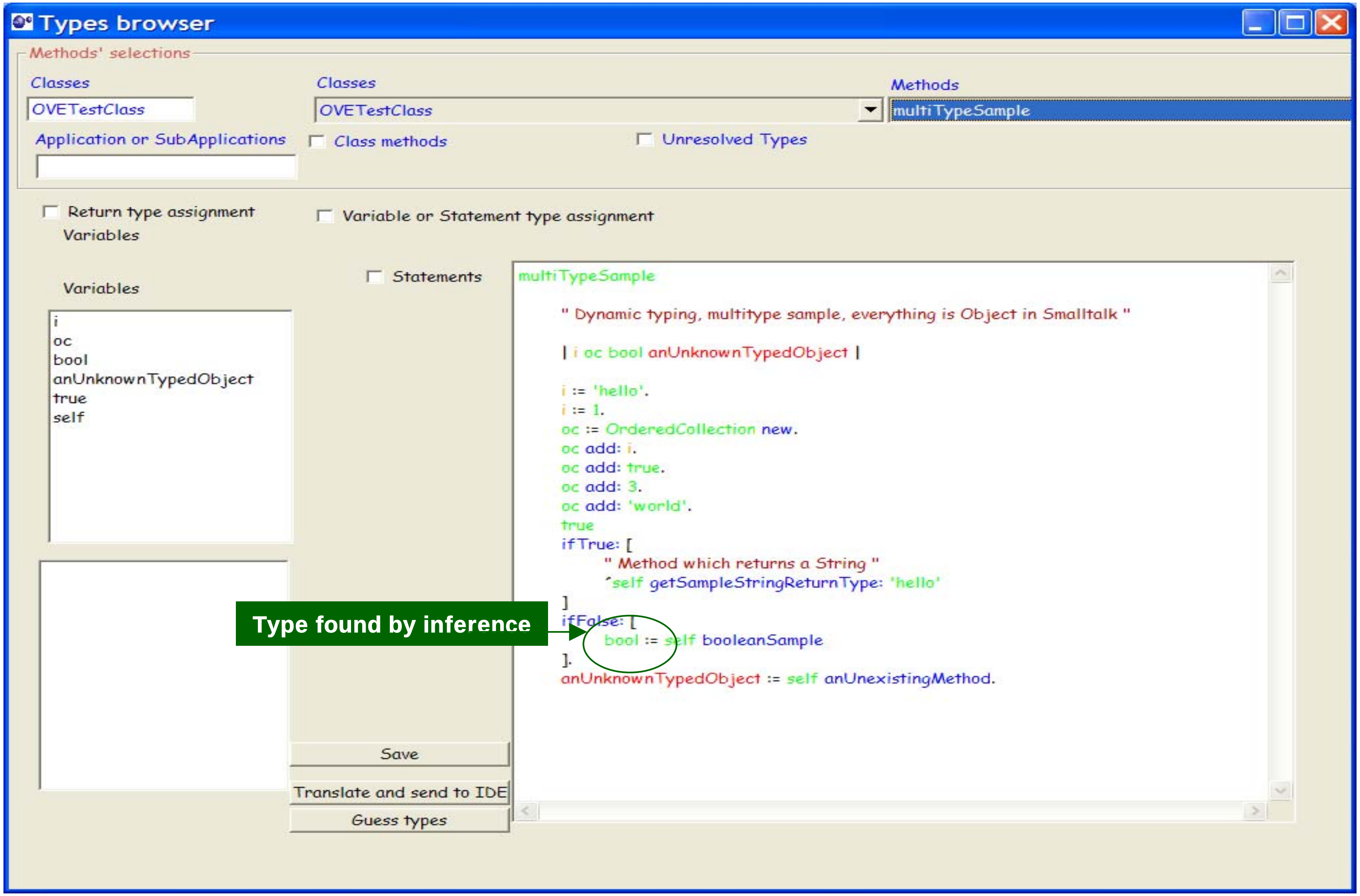
Variables:

- i
- oc
- bool
- anUnknownTypedObject
- true
- self

```
multiTypeSample

	" Dynamic typing, multitype sample, everything is Object in Smalltalk "

	| i oc bool anUnknownTypedObject |

	i := 'hello'.
	i := 1.
	oc := OrderedCollection new.
	oc add: i.
	oc add: true.
	oc add: 3.
	oc add: 'world'.
	true
	ifTrue: [
		" Method which returns a String "
		^self getSampleStringReturnType: 'hello'
	]
	ifFalse: [
		bool := self booleanSample
	].
	anUnknownTypedObject := self anUnexistingMethod.
```

Unknown type

After Types Inference



Methods' selections

Classes: OVETestClass

Classes: OVETestClass

Methods: multiTypeSample

Application or SubApplications:

□ Class methods

□ Unresolved Types

□ Return type assignment

Variables

□ Variable or Statement type assignment

Variables

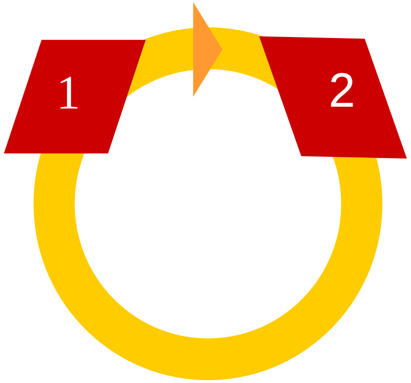
- i
- oc
- bool
- anUnknownTypedObject
- true
- self

□ Statements

```
multiTypeSample

    " Dynamic typing, multitype sample, everything is Object in Smalltalk "

    | i oc bool anUnknownTypedObject |

    i := 'hello'.
    i := 1.
    oc := OrderedCollection new.
    oc add: i.
    oc add: true.
    oc add: 3.
    oc add: 'world'.
    true
    ifTrue: [
          " Method which returns a String "
          ^self getSampleStringReturnType: 'hello'
    ]
    ifFalse: [
          bool := self booleanSample
    ].
    anUnknownTypedObject := self anUnexistingMethod.
```

Type found by inference

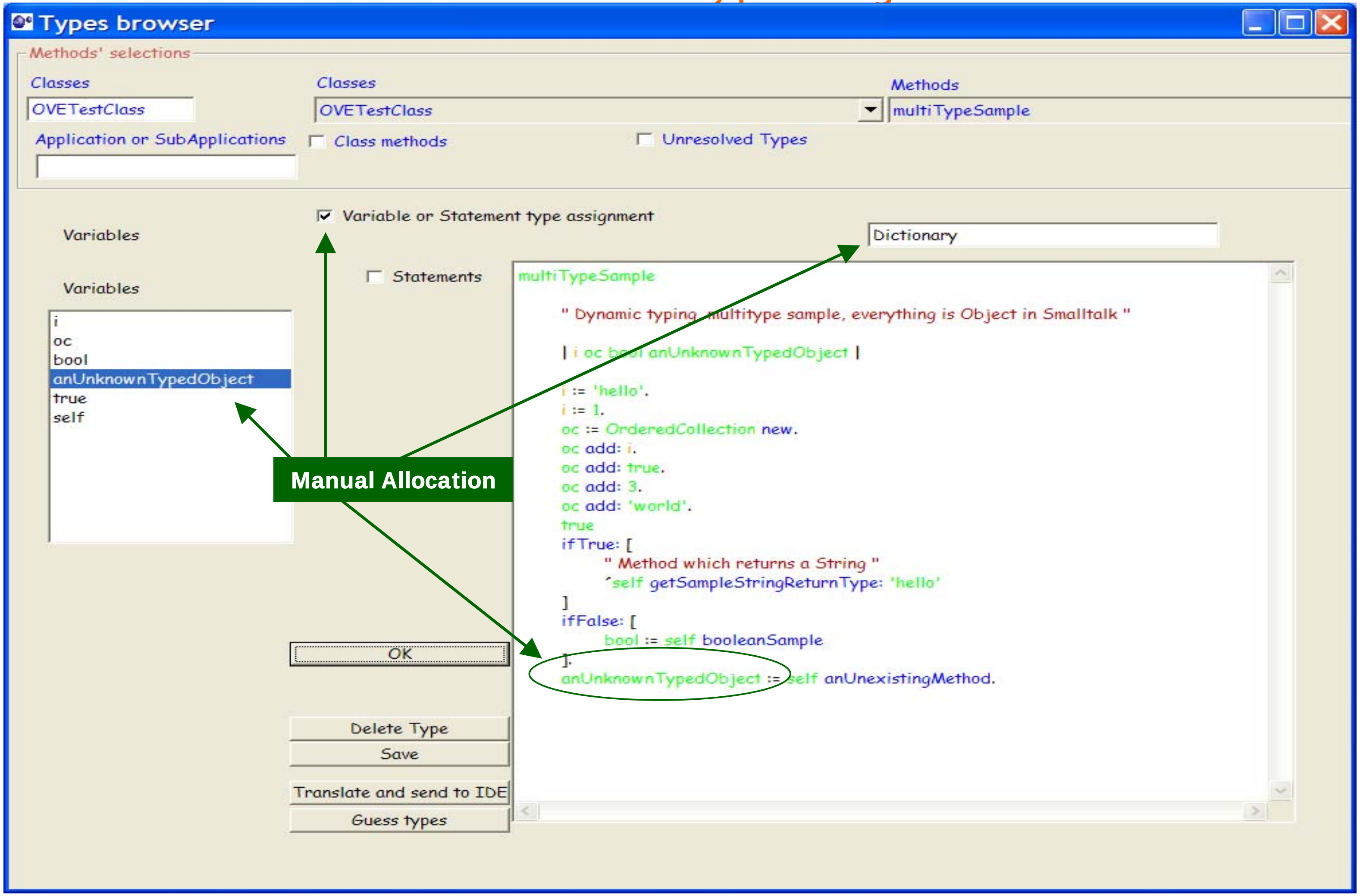
Types Detection



1. Recording and launching applications scenarios
2. Automatic type inference
3. **Manual allocation of types**

   (Saved in Knowledge Base for use in Translation Engine)

After Manual Variable Allocation
anUnknownTypedObject



Classes: OVETestClass

Classes: OVETestClass

Methods: multiTypeSample

□ Class methods

□ Unresolved Types

☑ Variable or Statement type assignment

Dictionary

□ Statements

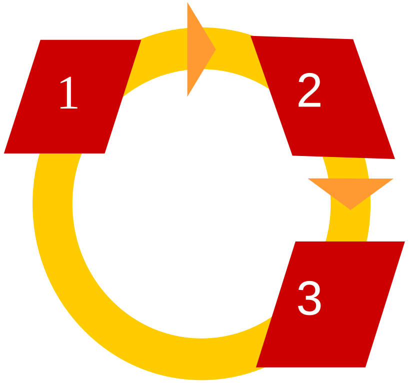
Variables

Variables

- i
- oc
- bool
- anUnknownTypedObject
- true
- self

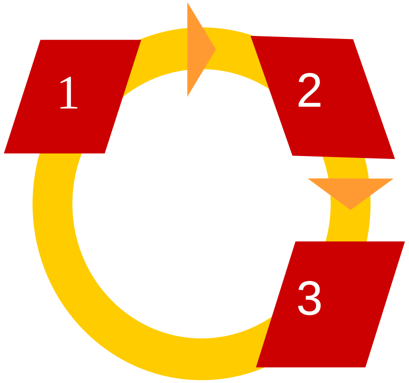
```
multiTypeSample

    " Dynamic typing, multitype sample, everything is Object in Smalltalk "

    | i oc bool anUnknownTypedObject |

    i := 'hello'.
    i := 1.
    oc := OrderedCollection new.
    oc add: i.
    oc add: true.
    oc add: 3.
    oc add: 'world'.
    true
    ifTrue: [
        " Method which returns a String "
        ^self getSampleStringReturnType: 'hello'
    ]
    ifFalse: [
        bool := self booleanSample
    ].
    anUnknownTypedObject := self anUnexistingMethod.
```

ST to Java Correspondance Knowledge Base



- Correspondance of Packages, Classes.
- Correspondance of Methods.
- Classes used for delegation (because of insufficient class libraries in Java, or impossibility of subclassing final classes).
- Variables prefixes.
- Pool Dictionaries.

ST to Java Correspondance Knowledge Base



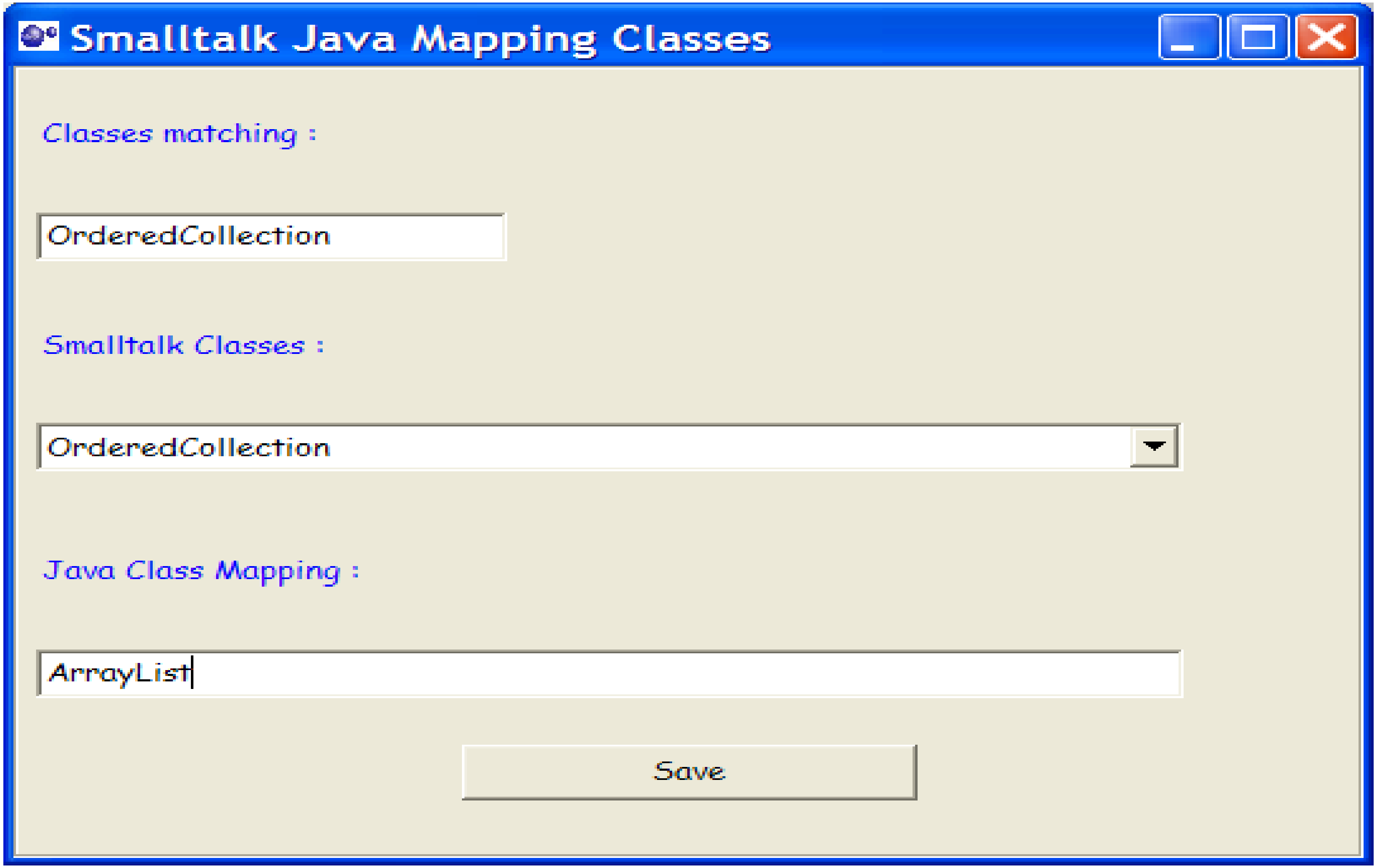
- **Correspondance of Packages, Classes.**
  - Correspondance of Metho
  - Classes used for delega
    Java, or impossibility of su
  - Variables prefixes.
  - Pool Dictionaries.

Specific packages names are defined either by:

- explicit names
- specific patterns

(for instance, the **OVETestApp** can correspond the the explicit package **com.ove.examples** or by specific pattern to **ove.test.app**)

Correspondance of classes Interface



Classes matching :

OrderedCollection

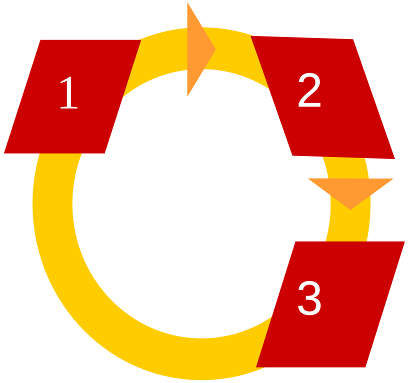
Smalltalk Classes :

OrderedCollection

Java Class Mapping :

ArrayList

ST to Java Correspondance Knowledge Base



- Correspondance of Packages, Classes.
- **Correspondance of Methods.**
- Classes used fo delega libraries in Java, o
- Variables prefixes.
- Pool Dictionaries.

ISSUES :

•No correspondance of methods names

•Smalltalk method ⬄ successive calls of several Java methods

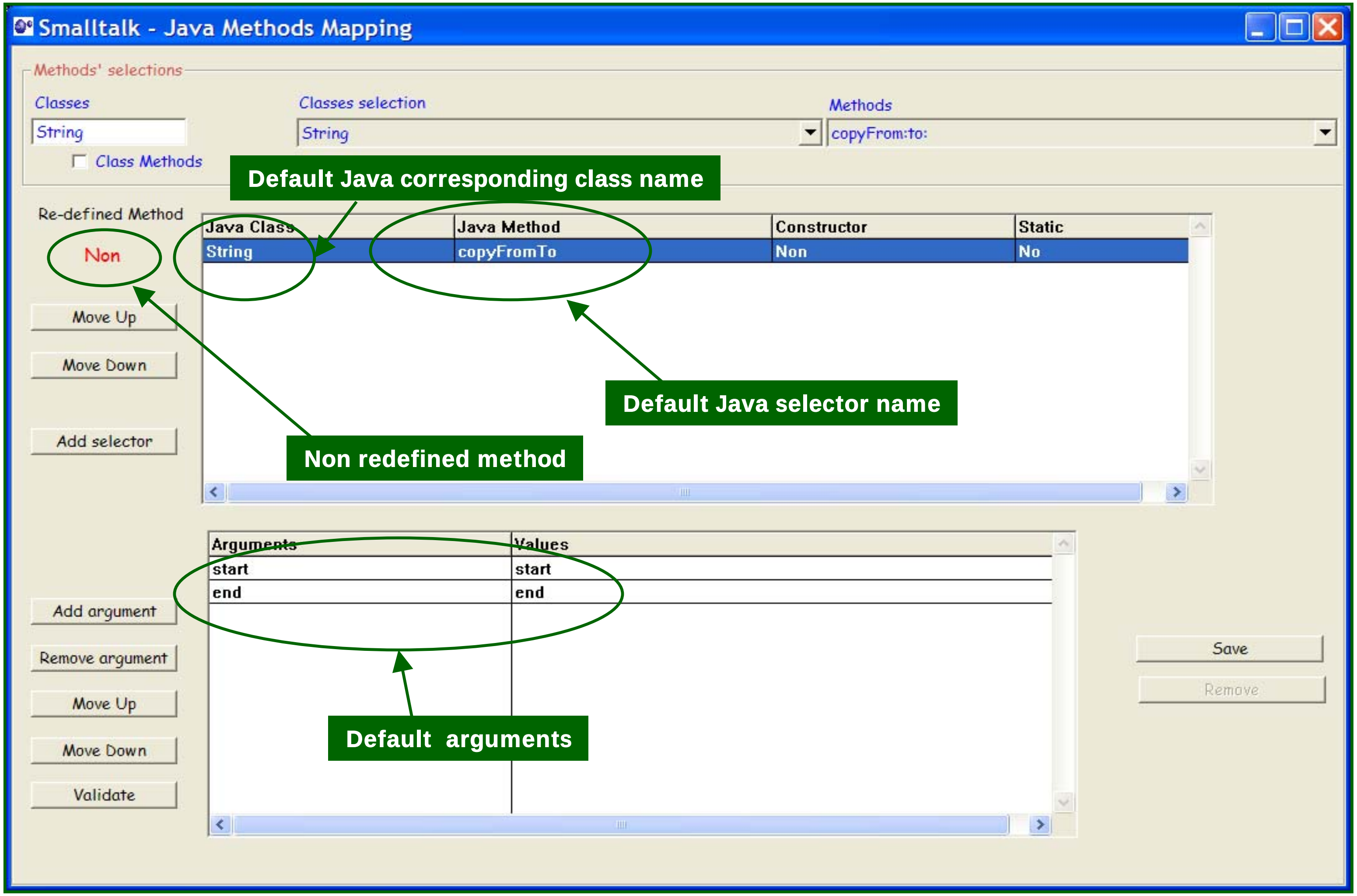
•Delegation concept ⬄ Services inexistant in Java

•Smalltalk Class method⬄Java Instance method

•Number of method's argument can vary

•The order of similar method's argument can vary

Default Correspondance of methods



Smalltalk - Java Methods Mapping

Methods' selections

Classes: String

☐ Class Methods

Classes selection: String

Methods: copyFrom:to:

Re-defined Method: Non

Java Class	Java Method	Constructor	Static
String	copyFromTo	Non	No

Arguments	Values
start	start
end	end

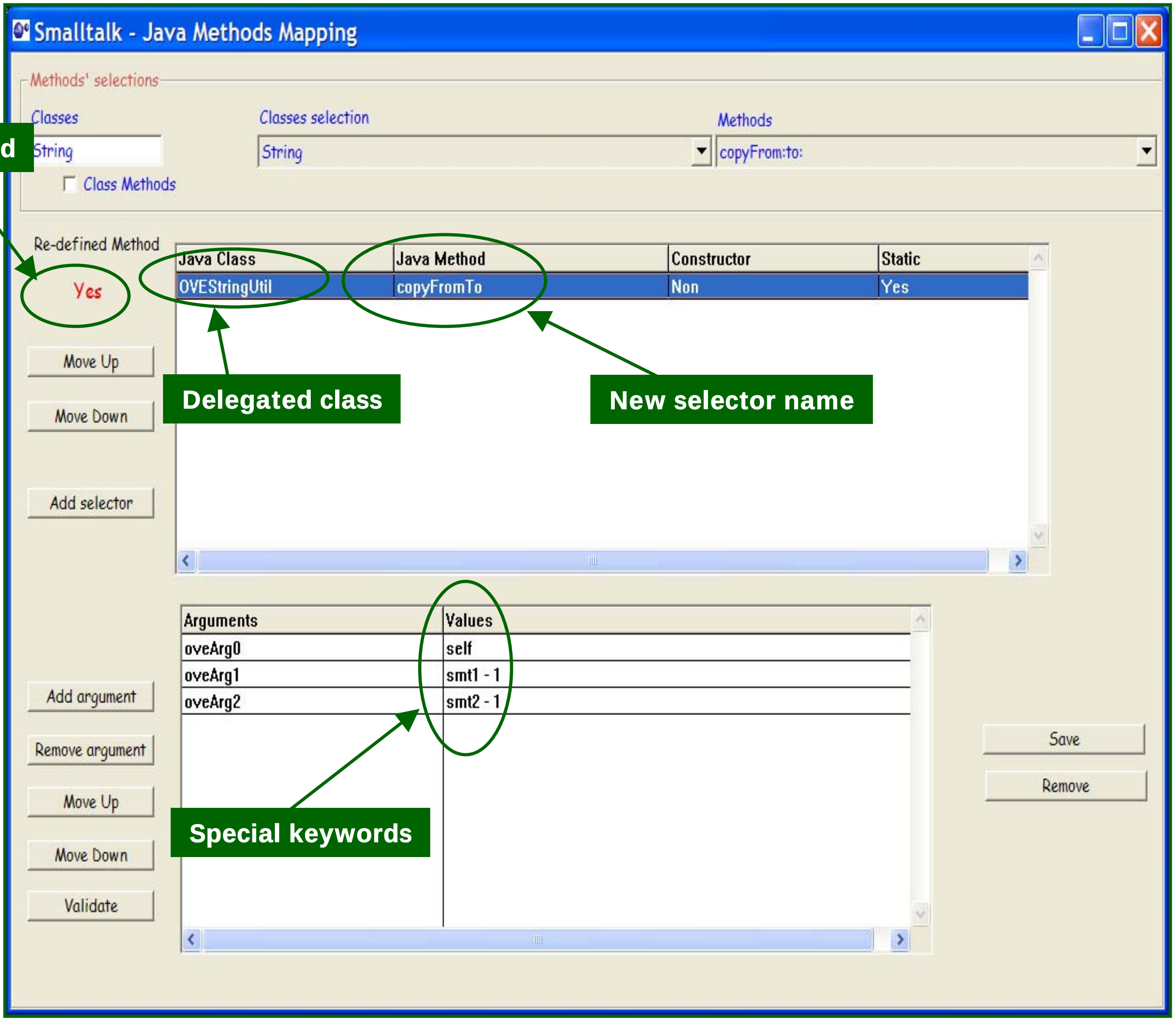
Default Java corresponding class name

Default Java selector name

Non redefined method

Default arguments

Methods Correspondance Customisation



Smalltalk - Java Methods Mapping

Methods' selections

Classes: String

☐ Class Methods

Classes selection: String

Methods: copyFrom:to:

Re-defined Method: Yes

Java Class	Java Method	Constructor	Static
OVEStringUtil	copyFromTo	Non	Yes

Arguments	Values
oveArg0	self
oveArg1	smt1 - 1
oveArg2	smt2 - 1

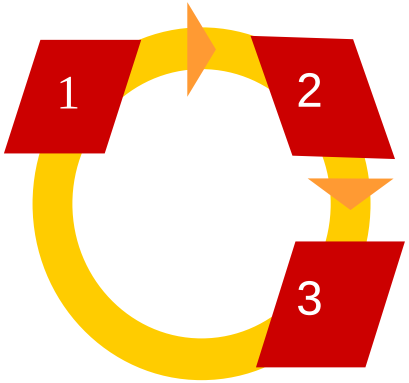
Redefined method

Delegated class

New selector name

Special keywords

ST to Java Correspondance Knowledge Base

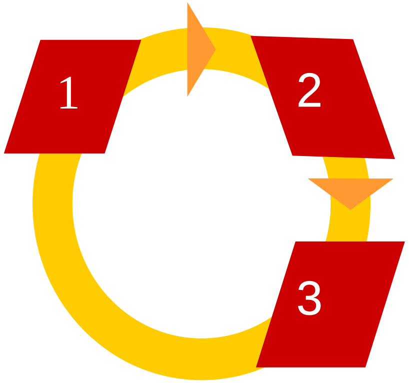


- Correspondance of Packages, Classes.
- Correspondance of Methods.
- **Classes used for delegation (because of insufficient class libraries in Java, or impossibility of subclassing final classes).**
- Variables prefixes.
- Pool Dictionaries.

DirectJava Classes Library

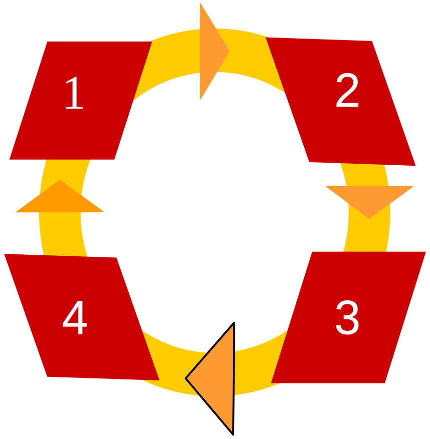
- Overview Package Class **Tree** Deprecated Index Help PREV NEXTFRAMES NO FRAMES**Hierarchy For All Packages**
- **Package Hierarchies:**
  - ove.components.base.collection, ove.components.base.date, ove.components.base.lang, ove.components.base.number, ove.tool
- **Class Hierarchy**
- class java.lang.Object
  - class ove.components.base.lang.**OVEBasicStringUtil**
    - **class ove.components.base.lang.**OVEStringUtil
  - class ove.components.base.lang.**OVEBeanPropertiesUtility**
  - class ove.components.base.lang.**OVEClassUtil**
  - class ove.components.base.lang.**OVECloneUtil**
  - class ove.components.base.collection.**OVECollectionUtil**
  - class ove.tool.**OVEComparatorUtil**
  - class ove.tool.**OVEComparatorUtilities**
  - class ove.components.base.date.**OVEDateUtil**
  - class ove.components.base.lang.**OVEFilterName** (implements java.io.FilenameFilter, java.io.Serializable)
  - class ove.components.base.lang.**OVEInstanceUtil**
  - class ove.components.base.number.**OVEInterval** (implements java.util.Iterator)
  - class ove.components.base.number.**OVEMathUtil**
  - class ove.components.base.lang.**OVEMessage**
  - class ove.components.base.lang.**OVESerializationUtility**
  - class ove.components.base.collection.**OVESortedList** (implements java.util.List)
  - class ove.components.base.date.**OVETimeUtil**
  - class java.lang.Throwable (implements java.io.Serializable)
    - **class java.lang.Exception**
      - class java.lang.RuntimeException
        - » class ove.components.base.collection.**OVEBlockReturnException**
- **Interface Hierarchy**
- interface ove.components.base.collection.**OVEClosure**
- interface ove.components.base.collection.**OVEOneArgClosure**
- interface ove.components.base.collection.**OVEOneArgPredicate**
- interface ove.components.base.collection.**OVEPredicate**
- interface ove.components.base.collection.**OVETransformer**
- interface ove.components.base.collection.**OVETwoArgsClosure**
- interface ove.components.base.collection.**OVETwoArgsPredicate**
- interface ove.components.base.collection.**OVETwoArgsTransformer**
- Overview Package Class **Tree** Deprecated Index Help PREV NEXTFRAMES NO FRAMES

ST to Java Correspondance Knowledge Base



- Correspondance of Packages, Classes.
- Correspondance of Methods.
- Classes used for delegation (because of insufficient class libraries in Java, or impossibility of subclassing final classes).
- **Variables prefixes.**
- **Pool Dictionaries.**

Translation Engine



- Translation by Batches, Classes, Methods
- Direct integration in target environment (VA for Java)
- Translation by Deltas (translating only differences between 2 versions of sub-application, of class, etc…)
- GUI
- Java Overriding

Technical issues

- Static Methods (Java) vs Class Methods (ST)
- Most common cases

Static Methods (Java) vs Class Methods (ST)

- Smalltalk

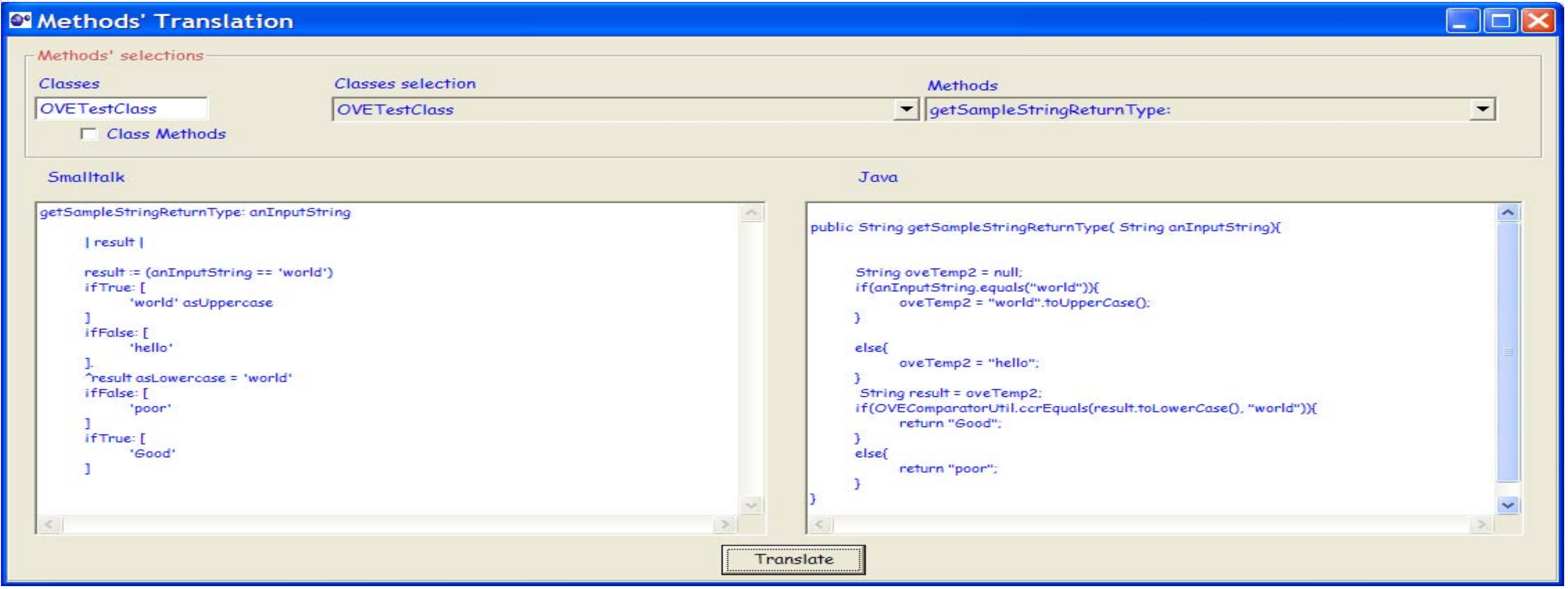
```
Object subclass: #Class1
    Class1 class>>#foo1
      ^ ‘foo1’
    Class1>>#fooInst1
      ^self class foo1

Class1 subclass: #Class2
    Class2>>#fooInst2
      ^self fooInst1
    Class2 class>>#foo1
      ^ ‘foo2’
```

- Java

```
Class1
public static String foo1(Class aClass) {
        return (String)new OVEMessage().perform(aClass,“foo1",new Object[]{});
}
public String fooInst1(){
     return foo1(getClass());
}
Class2
public String fooInst2() {
     return fooInst1();
}

public static String foo1() {
     return “foo2”;
}
```

Simple Samples (1)

Simple String Sample



Methods' Translation

Methods' selections

Classes: OVETestClass

□ Class Methods

Classes selection: OVETestClass

Methods: getSampleStringReturnType:

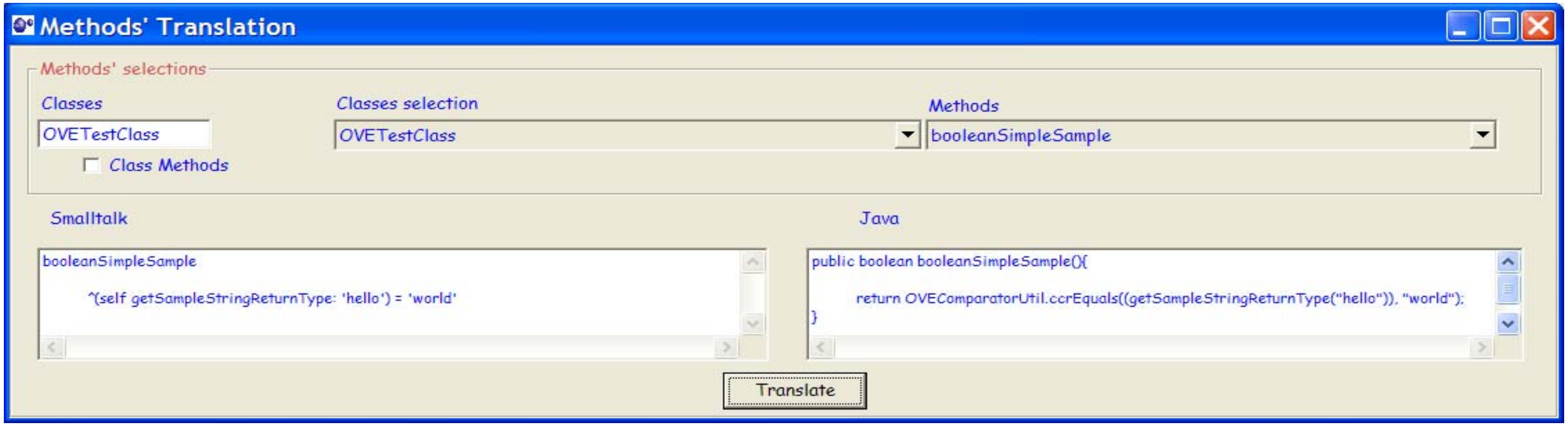
Smalltalk

```
getSampleStringReturnType: anInputString

        | result |

        result := (anInputString == 'world')
        ifTrue: [
                'world' asUppercase
        ]
        ifFalse: [
                'hello'
        ].
        ^result asLowercase = 'world'
        ifFalse: [
                'poor'
        ]
        ifTrue: [
                'Good'
        ]
```

Java

```
public String getSampleStringReturnType( String anInputString){


        String oveTemp2 = null;
        if(anInputString.equals("world")){
                oveTemp2 = "world".toUpperCase();
        }

        else{
                oveTemp2 = "hello";
        }
         String result = oveTemp2;
        if(OVEComparatorUtil.ccrEquals(result.toLowerCase(), "world")){
                return "Good";
        }
        else{
                return "poor";
        }
}
```

Simple boolean sample



Methods' Translation

Methods' selections

Classes: OVETestClass

□ Class Methods

Classes selection: OVETestClass

Methods: booleanSimpleSample

Smalltalk

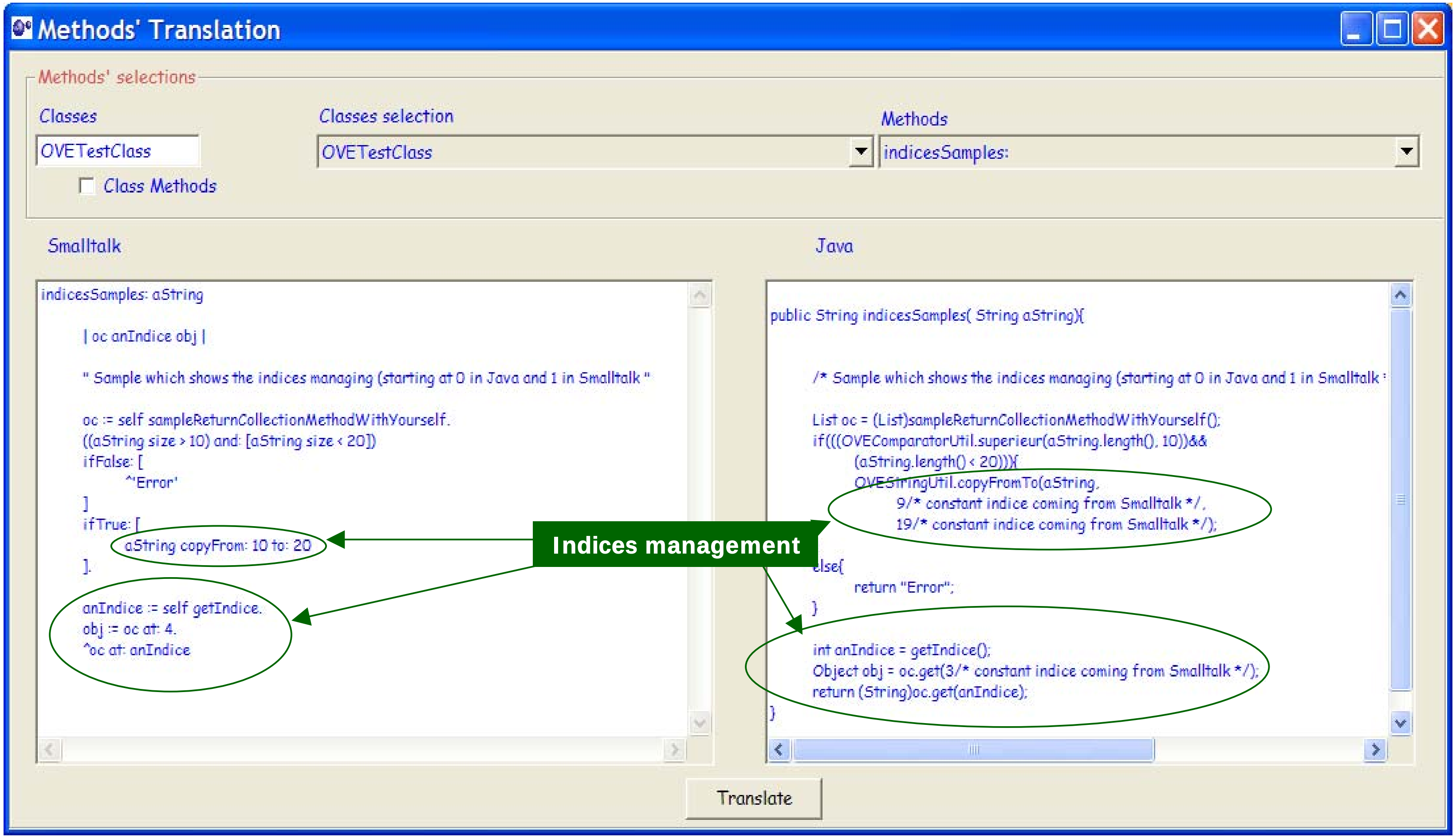
```
booleanSimpleSample

        ^(self getSampleStringReturnType: 'hello') = 'world'
```

Java

```
public boolean booleanSimpleSample(){

        return OVEComparatorUtil.ccrEquals((getSampleStringReturnType("hello")), "world");
}
```

Simple Samples (2)

Indice sample



Classes: OVETestClass

□ Class Methods

Classes selection: OVETestClass

Methods: indicesSamples:

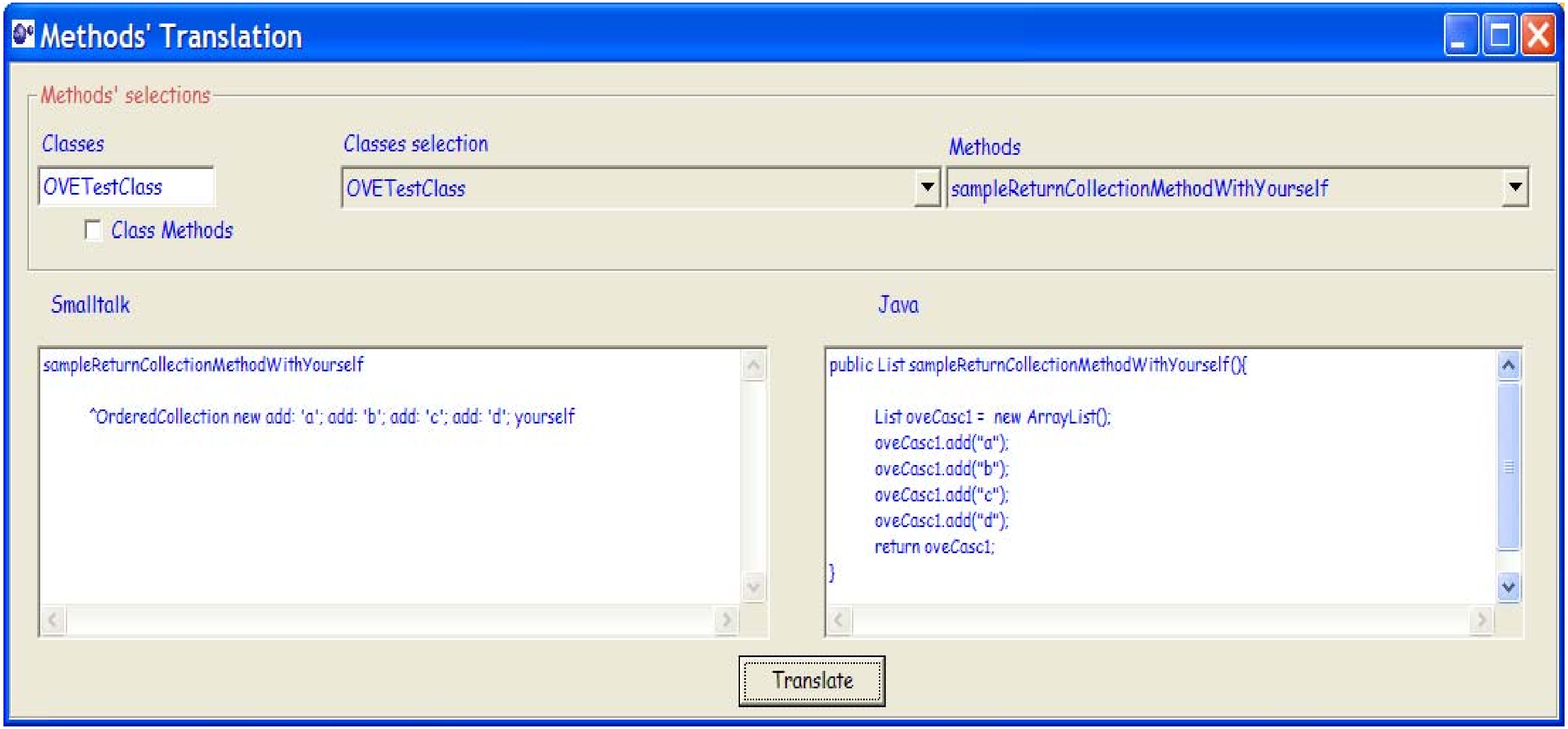
Smalltalk

```
indicesSamples: aString

	| oc anIndice obj |

	" Sample which shows the indices managing (starting at 0 in Java and 1 in Smalltalk "

	oc := self sampleReturnCollectionMethodWithYourself.
	((aString size > 10) and: [aString size < 20])
	ifFalse: [
		^'Error'
	]
	ifTrue: [
		aString copyFrom: 10 to: 20
	].

	anIndice := self getIndice.
	obj := oc at: 4.
	^oc at: anIndice
```

Java

```
public String indicesSamples( String aString){


	/* Sample which shows the indices managing (starting at 0 in Java and 1 in Smalltalk 

	List oc = (List)sampleReturnCollectionMethodWithYourself();
	if(((OVEComparatorUtil.superieur(aString.length(), 10))&&
		(aString.length() < 20))){
		OVEStringUtil.copyFromTo(aString,
			9/* constant indice coming from Smalltalk */,
			19/* constant indice coming from Smalltalk */);

	else{
		return "Error";
	}

	int anIndice = getIndice();
	Object obj = oc.get(3/* constant indice coming from Smalltalk */);
	return (String)oc.get(anIndice);
}
```

Cascading messages with yourself



Classes: OVETestClass

□ Class Methods

Classes selection: OVETestClass

Methods: sampleReturnCollectionMethodWithYourself

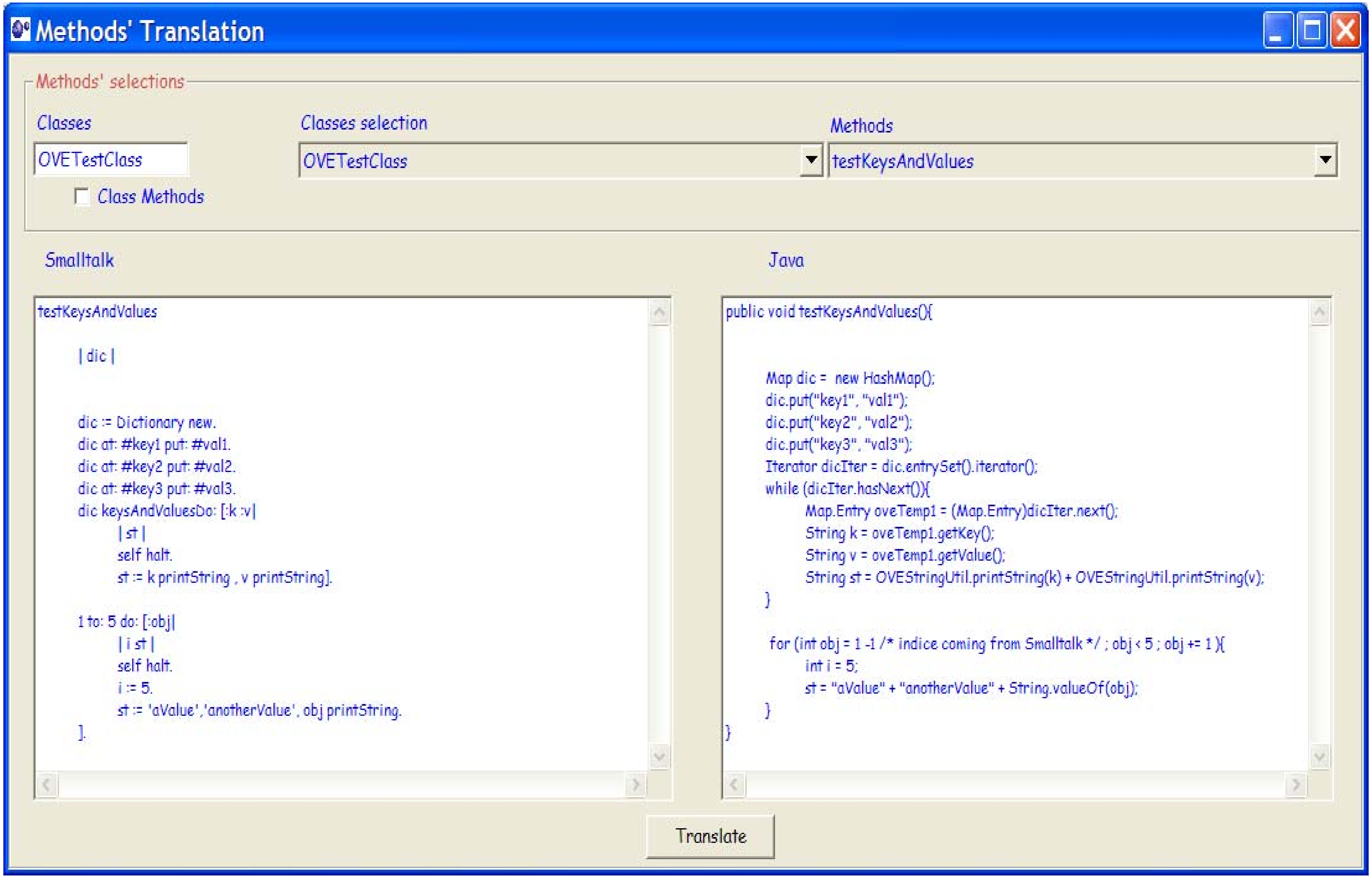
Smalltalk:

```
sampleReturnCollectionMethodWithYourself

	^OrderedCollection new add: 'a'; add: 'b'; add: 'c'; add: 'd'; yourself
```

Java:

```
public List sampleReturnCollectionMethodWithYourself(){

	List oveCasc1 =  new ArrayList();
	oveCasc1.add("a");
	oveCasc1.add("b");
	oveCasc1.add("c");
	oveCasc1.add("d");
	return oveCasc1;
}
```

Block without inner class (special patterns)



Classes: OVETestClass

☐ Class Methods

Classes selection: OVETestClass

Methods: testKeysAndValues

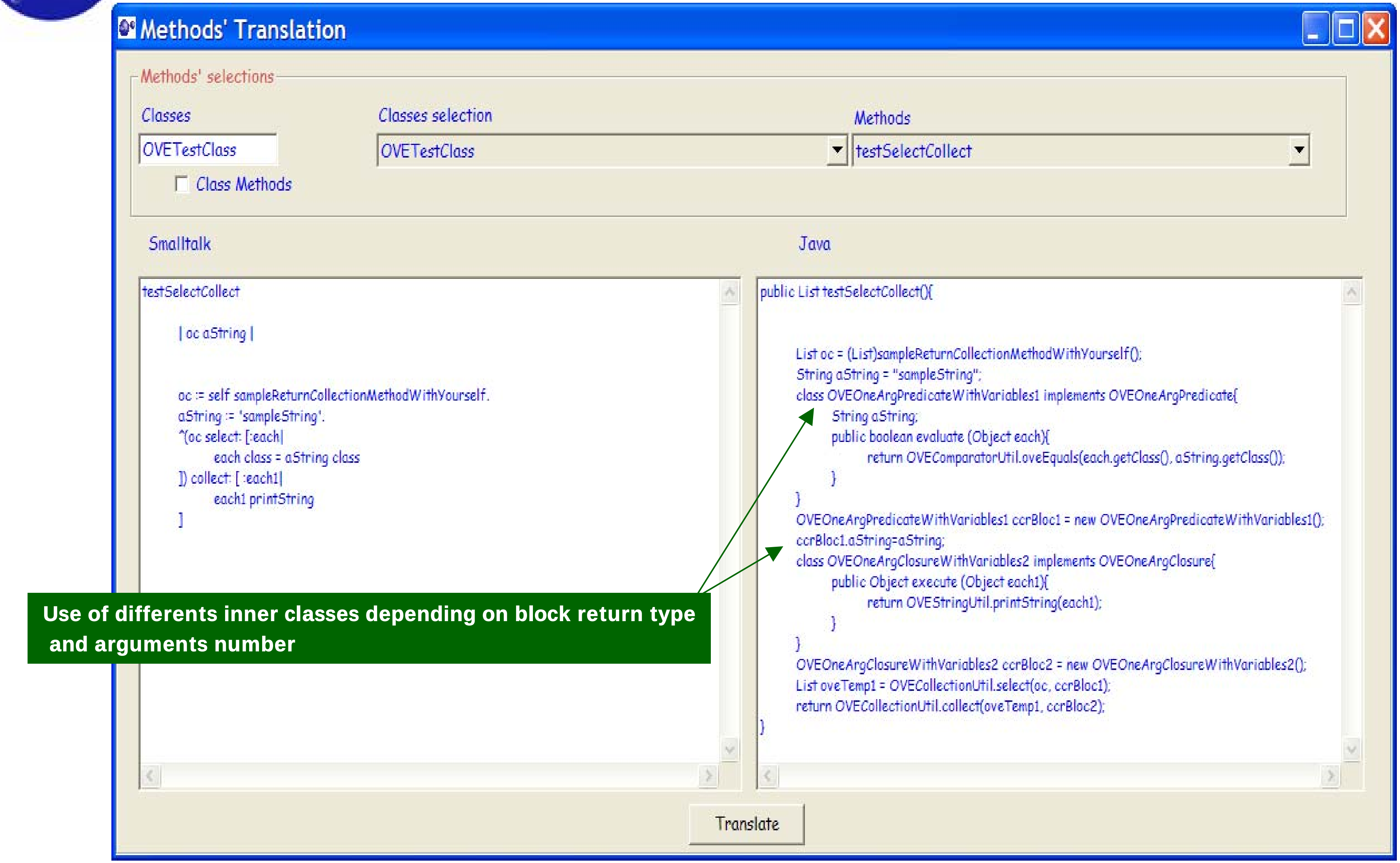
Smalltalk

```
testKeysAndValues

        | dic |


        dic := Dictionary new.
        dic at: #key1 put: #val1.
        dic at: #key2 put: #val2.
        dic at: #key3 put: #val3.
        dic keysAndValuesDo: [:k :v|
                | st |
                self halt.
                st := k printString , v printString].

        1 to: 5 do: [:obj|
                | i st |
                self halt.
                i := 5.
                st := 'aValue','anotherValue', obj printString.
        ].
```

Java

```
public void testKeysAndValues(){


        Map dic =  new HashMap();
        dic.put("key1", "val1");
        dic.put("key2", "val2");
        dic.put("key3", "val3");
        Iterator dicIter = dic.entrySet().iterator();
        while (dicIter.hasNext()){
                Map.Entry oveTemp1 = (Map.Entry)dicIter.next();
                String k = oveTemp1.getKey();
                String v = oveTemp1.getValue();
                String st = OVEStringUtil.printString(k) + OVEStringUtil.printString(v);
        }

         for (int obj = 1 -1 /* indice coming from Smalltalk */ ; obj < 5 ; obj += 1 ){
                int i = 5;
                st = "aValue" + "anotherValue" + String.valueOf(obj);
        }
}
```

Blocks with inner classes



Methods' selections

Classes: OVETestClass

□ Class Methods

Classes selection: OVETestClass

Methods: testSelectCollect

Smalltalk

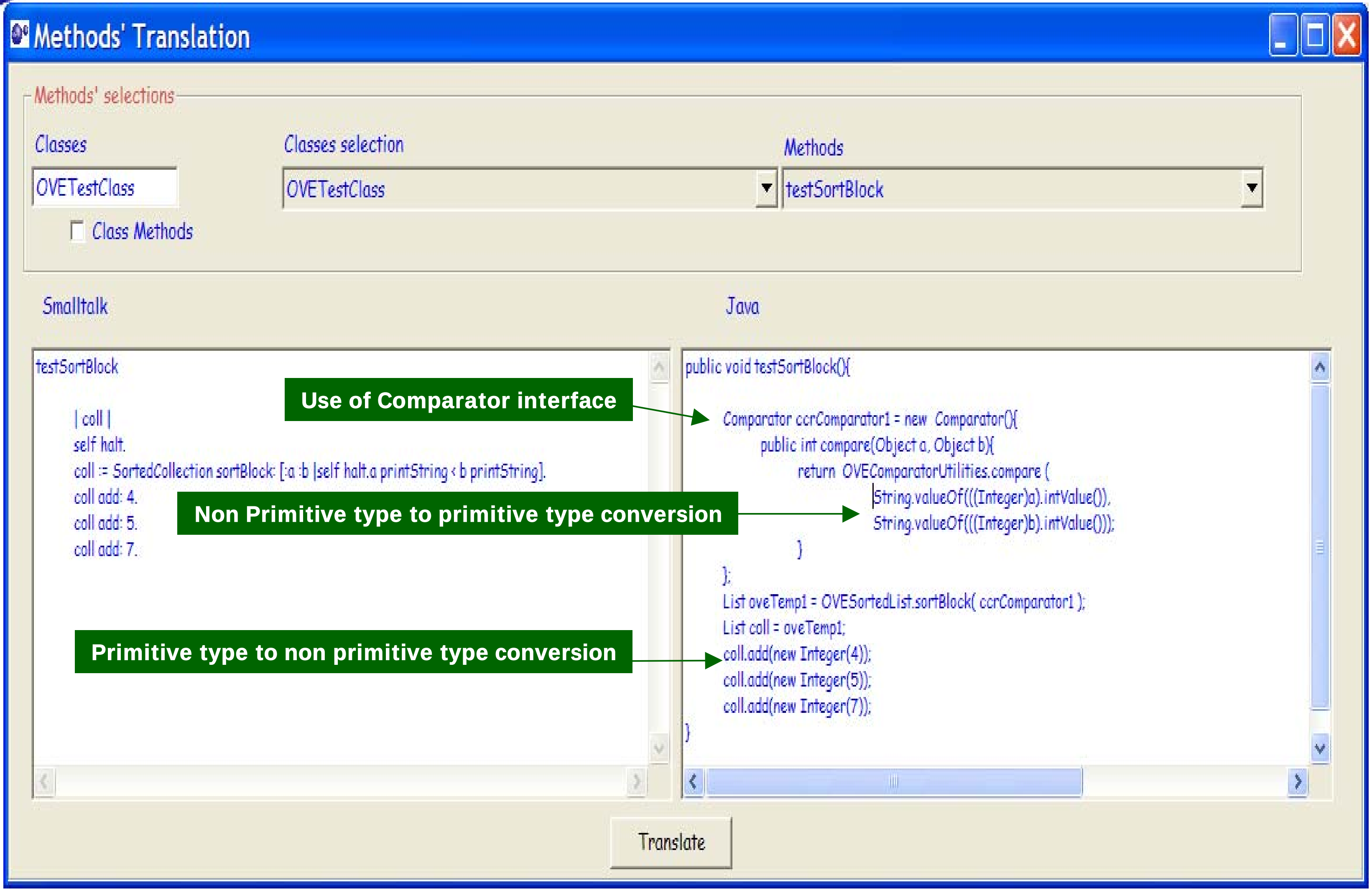
```
testSelectCollect

        | oc aString |


        oc := self sampleReturnCollectionMethodWithYourself.
        aString := 'sampleString'.
        ^(oc select: [:each|
                each class = aString class
        ]) collect: [ :each1|
                each1 printString
        ]
```

Java

```
public List testSelectCollect(){


        List oc = (List)sampleReturnCollectionMethodWithYourself();
        String aString = "sampleString";
        class OVEOneArgPredicateWithVariables1 implements OVEOneArgPredicate{
                String aString;
                public boolean evaluate (Object each){
                        return OVEComparatorUtil.oveEquals(each.getClass(), aString.getClass());
                }
        }
        OVEOneArgPredicateWithVariables1 ccrBloc1 = new OVEOneArgPredicateWithVariables1();
        ccrBloc1.aString=aString;
        class OVEOneArgClosureWithVariables2 implements OVEOneArgClosure{
                public Object execute (Object each1){
                        return OVEStringUtil.printString(each1);
                }
        }
        OVEOneArgClosureWithVariables2 ccrBloc2 = new OVEOneArgClosureWithVariables2();
        List oveTemp1 = OVECollectionUtil.select(oc, ccrBloc1);
        return OVECollectionUtil.collect(oveTemp1, ccrBloc2);
}
```

Use of differents inner classes depending on block return type and arguments number

SortedCollection with a sort block



Classes: OVETestClass

□ Class Methods

Classes selection: OVETestClass

Methods: testSortBlock

Smalltalk

```
testSortBlock

        | coll |
        self halt.
        coll := SortedCollection sortBlock: [:a :b |self halt.a printString < b printString].
        coll add: 4.
        coll add: 5.
        coll add: 7.
```

Java

```
public void testSortBlock(){

    Comparator ccrComparator1 = new  Comparator(){
            public int compare(Object a, Object b){
                    return  OVEComparatorUtilities.compare (
                                    String.valueOf(((Integer)a).intValue()),
                                    String.valueOf(((Integer)b).intValue()));
                    }
    };
    List oveTemp1 = OVESortedList.sortBlock( ccrComparator1 );
    List coll = oveTemp1;
    coll.add(new Integer(4));
    coll.add(new Integer(5));
    coll.add(new Integer(7));
}
```

Use of Comparator interface

Non Primitive type to primitive type conversion

Primitive type to non primitive type conversion

Case Study: CCR
(Caisse Centrale de Réassurance)

- Migration of core mission critical applications (over 1 million lines of code, ERP, Sales Management, Portfolio Management) i.e. 4,449 Classes ; 78,624 Methods
- Reasons for Migration: maintaining an team of Smalltalk experts, Connectivity

- DirectJava benefits : originally estimated as a 100 men/year project,

  migration will turn out to be a 3 men/year project. (now 30 months into it)
- Between 80 and 95% of code has been translated automatically.

  Automatic translation has also helped testing and architecture choices

  Example: persistance framework re-organized in order to use EJBs

  Massive increase in volume of code reviewed helped by automated translation

- Business Benefits: New market developments (esp Online Brokerage) thanks to Java
- Better integration with existing tools in the environment (Domino/Notes, Excell, Word)

Thank You