



JADE Development Environment

Version 2018

JADE Software Corporation Limited cannot accept any financial or other responsibilities that may be the result of your use of this information or software material, including direct, indirect, special or consequential damages, or loss of profits. There are no warranties extended or granted by this document or software material.

You should be very careful to ensure that the use of this software material and/or information complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of JADE Software Corporation Limited.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Copyright © 2019 JADE Software Corporation Limited.

All rights reserved.

JADE is a trademark of JADE Software Corporation Limited. All trade names referenced are the service mark, trademark, or registered trademark of the respective manufacturer.

Contents

JADE Development Environment	4
Introduction	4
User Preferences	4
Macros and Regular Expressions.....	5
Sending Messages to other Developers	7
Exercise 1 – Customizing your JADE Development Environment.....	8
Exercise 2 – Creating a JADE Macro	9
Exercise 3 – Sending a Message in a Multiuser System.....	13
Show Symbol Command and Source Windows	14
References and Implementors	16
Renaming Identifiers	17
Exercise 4 – Finding and Fixing Bugs.....	19
Exercise 5 – Refactoring Names.....	25
Unused Variables.....	28
Unused Class Entities	30
Exercise 6 – Locating and Removing Unused Variables.....	33
Exercise 7 – Locating and Removing Unused Class Entities	35
Transient Leaks.....	36
Methods Status List	39
Schema Validation	40
Exercise 8 – Finding and Removing Transient Leaks	42
Exercise 9 – Finding Changed Methods	43
Exercise 10 – Validating the Schemas.....	48

JADE Development Environment

Introduction

The JADE development environment (or IDE) is full of features that are useful for navigating, maintaining, and bug-fixing in large JADE systems.

In this module, you will use the various features of the development environment to locate and repair defects in an intentionally-broken version of the **Erewhon** demonstration system. The **Erewhon** system is a demonstration system that showcases some of the features of JADE, and as such, is large enough for the development environment tools to be useful, while small enough to be reasonable to navigate without requiring a great deal of familiarity with the system.

User Preferences

The Preferences dialog is accessed from the **Preferences** command from the Options menu. It allows you to specify a variety of look-and-feel options for the JADE development environment.

Options set in the Preferences dialog persist between sessions. However, they impact the user who set them; any other users of the same database are unaffected.

The Preferences dialog has the following tabs that access sheets containing the types of options you can set.

Sheet	Purpose
Accelerator Keys	Maps accelerator keys that insert text into the JADE editor; for example, Ctrl+Shift+A inserts abortTransaction ;
Browser	Customizes behavior related to how hierarchy browsers (including the Schema Browser and Class Browser) function; for example, the Mdi group box specifies whether they are floating MDI windows or tabs are displayed above the MDI client window
Editor	Customizes syntax highlighting colors; for example, JADE keywords are a dark blue colour by default, but you can change them to a color of your choice
Editor Options	Customizes the way in which the editor pane is displayed; for example, whether line numbers are displayed
Exit	Customizes the behavior when exiting from the development environment; for example, whether to confirm before closing
Lock	Modifies the default lock exception handling
Miscellaneous	Modifies settings that you are unlikely to want to change; default file suffixes (although it is best not to change the schema file suffix of .scm), the base locale, and interface and versioning options

Sheet	Purpose
Relationship	Customizes the display of the relationships, which are covered later in this module
Schema	Changes the default access to classes, references, and attributes in schemas as well as the persistence and type of classes
Shortcut Keys	Changes development environment and keyboard shortcuts, and swaps to the default use of F11 and F12 keys and F5 and F9 keys; that is, JADE by default has the F11 and F12 keys and the F5 and F9 keys swapped relative to Visual Studio (or Visual Studio to JADE, depending on your perspective)
Source Management	Customizes JADE version control functionality; for example, deltas and Git integration
Status list	Allows for the display of all methods and class constants, instead of the default display of only methods and class constants that are in error or have not been compiled (covered later in this module)
Text Templates	Customizes the default text template; for example, so that you can add a header to every method for documentation purposes
Window	Customizes the colors of windows (for example, the background color of the Class Browser) and the default font used in JADE windows

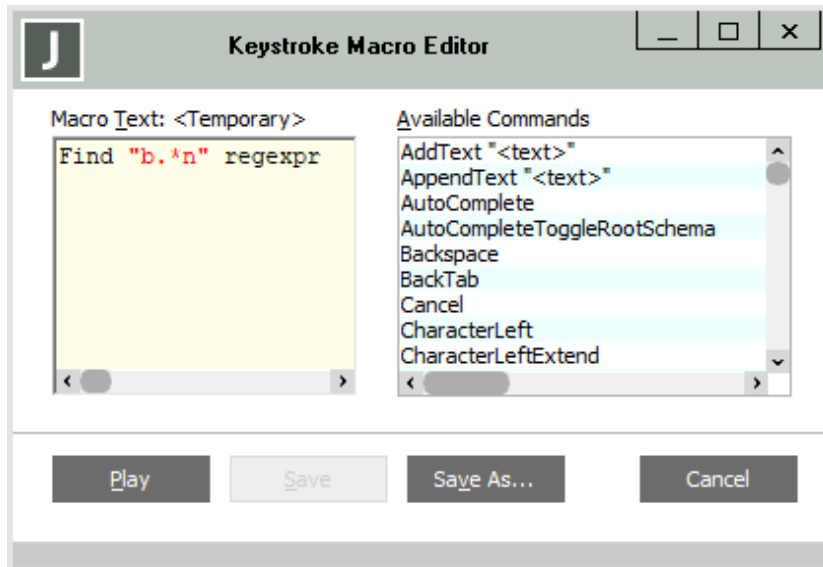
Macros and Regular Expressions

JADE macros enable you to record and replay keystrokes in the editor pane; for example, to comment or uncomment a selection of text, or to delete all text on the current line.

The two ways to create a macro are as follows.

- Use the **Start Macro Record** command from the Macro submenu of the Edit menu to start recording, perform a sequence of keystrokes, and then select the **Stop Macro Record** command from the Macro submenu of the Edit menu.
- Select the **Edit Temp Macro** command from the Macro submenu of the Edit menu to access the Keystroke Macro Editor dialog, select the required actions from the list box at the right (or you can type them in the **Macro Text** text box at the left), and then click **Save**.

When creating macros, you can use regular expressions. The Keystroke Macro Editor supports the use of regular expressions (regex) with the **Find**, **ReplaceFind**, and **ReplaceAll** commands. For example, you can use the **Find** command as follows.



In this example, **regexpr** causes the search string to be treated as a regular expression rather than a string literal. As such, “**b.*n**” translates to a **b**, then any number of any characters, and then an **n**.

You can use the following regular expression characters.

Character	Description
-----------	-------------

.	Matches any character.
\(Marks the start of a region for tagging a match.
\)	Marks the end of a tagged region.
\n	The <i>n</i> parameter is a digit in the range 1 through 9, which refers to the first through ninth tagged region when replacing; for example, if the search string was Fred\([1-9])XXX and the replace string was Sam\1YYY , this would generate Sam2YYY when applied to Fred2XXX .
\<	Matches the start of a word. A word is defined to be a character string beginning or ending, or both beginning and ending, with characters in the ranges A through Z, a through z, 0 through 9, and an underscore. In addition, it must be preceded or followed, or both preceded and followed, by any character outside those mentioned.
\>	Matches the end of a word.
\x	Enables you to use a character <i>x</i> that would otherwise have a special meaning; for example, \[would be interpreted as [and not as the start of a character set.
[...]	Indicates a set of characters; for example, [abc] means any of the characters a, b, or c. You can also use ranges; for example, [a-z] for any lowercase character.
[^...]	Complement of the characters in the set; for example, [^A-Za-z] means any character except an alphabetic character.

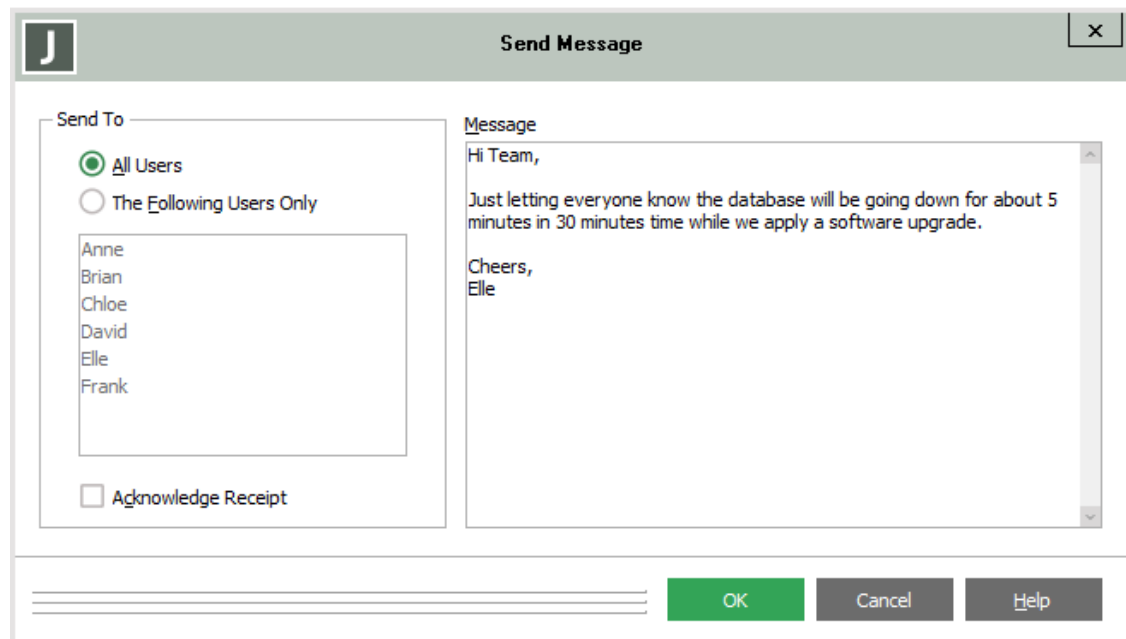
Character	Description
^	Matches the start of a line (unless it is used inside a set).
\$	Matches the end of a line.
*	Matches zero (0) or more times; for example, Sa*m matches Sm , Sam , Saam , Saaam , and so on.
+	Matches one or more times; for example, Sa+m matches Sam , Saam , Saaam , and so on.

When you have saved a macro, you can replay it by selecting the **Play Temp Macro** command from the Macro submenu of the Edit menu. You can also make a temporary macro (whether generated from recording keystrokes or from the Keystroke Macro Editor dialog) permanent, by clicking **Save As** on the Keystroke Macro Editor dialog.

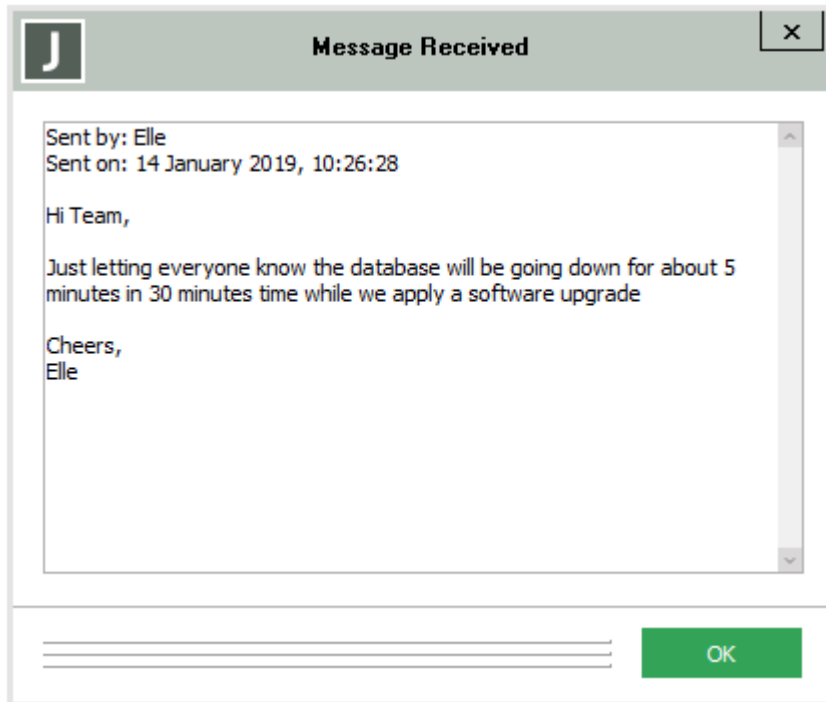
Sending Messages to other Developers

When using the JADE development environment in a multiuser system, it can sometimes be useful to see who else is connected to the JADE server and to send them messages; for example, to let all currently connected developers know when the database is going to be taken down or to check with a developer before terminating their process with the JADE Monitor.

The **Send Message** command of the File menu accesses the Send Message dialog, which displays a list of all users who are currently connected to the database.



You can send messages to all users by selecting the **All Users** option button, or to a specific user or users by selecting the **The Following Users Only** option button and then selecting the users to which the message should be sent.



The sender's name and the timestamp automatically generated for all messages are displayed above the message.

When the Message Received dialog is closed, if you checked the **Acknowledge Receipt** check box on the Send Message dialog, an acknowledgement message is sent back to you.

Exercise 1 – Customizing your JADE Development Environment

In this exercise, you will configure the look and feel of your JADE development environment. Although the steps in this exercise suggest a configuration setting, you can select whatever values you prefer.

1. From the Options menu, select the **Preferences** command.
2. Click on the **Accelerator Keys** tab to display the **Accelerator Keys** sheet and then change the **E** accelerator key from **endforeach;** to **epilog**. This will cause pressing Ctrl+Shift+E to insert **epilog** into the editor, at the caret.
3. On the **Browser** sheet, check the **Show Inherited** check box and then select the **Use Tabs Only** option button in the Mdi group box.
4. On the **Editor** sheet, change the **Background** from light yellow to white.
5. On the **Editor Options** sheet, check the **Insert Parentheses for Method with no parameters** check box in the Auto Complete group box and the **View Line Numbers** check box in the Display Options group box.
6. On the **Exit** sheet, uncheck the **Exit Confirmation** check box and check the **Save Windows** check box.
7. On the **Lock** sheet, change the **Number of times to Retry** to **15**.
8. On the **Miscellaneous** sheet, change the alternative Jade Skin in the Versioning Options group box to **JADE2018 Sumner**.

9. On the **Relationship** sheet, uncheck the **Show Detail** check box and set the **Target Class** color to white.
10. On the **Schema** sheet, check the **Use DDX style (xml format) as Default instead of DDB** check box.
11. On the **Short Cut Keys** sheet, select the **MacroPlayTemp** shortcut, click in the **Key Combination** text box, and then press Shift+Space to add that as the shortcut key combination.
12. On the **Source Management** sheet, check the **Reuse Same Method Source Window For All** check box.
13. On the **Status List** sheet, check the **Compiled Methods** check box and uncheck the **Uncompiled Methods** check box.
14. On the **Text Templates** sheet, add the following to the **Method** folder.

```

// -----
// Method:
//
// Purpose:
//
// Parameters:
//
// Returns:
// -----
begin
end:
    
```

This template will be included in the method source when the method is created

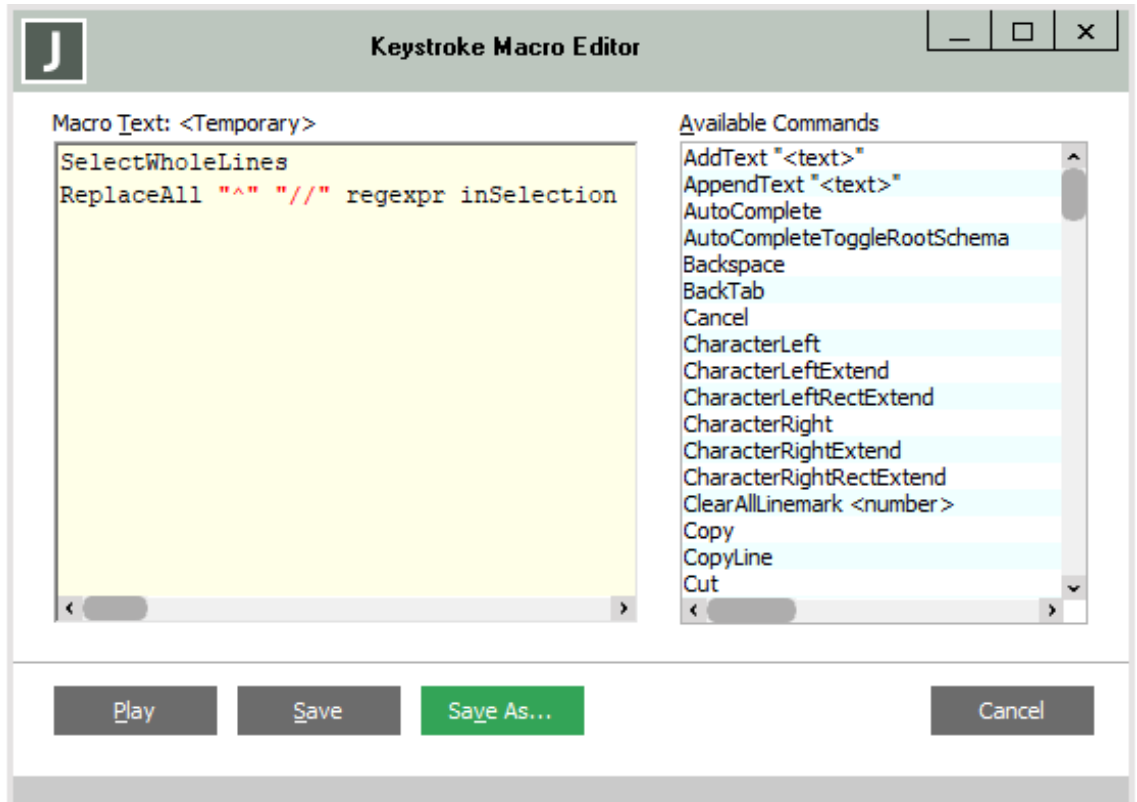
15. On the **Window** sheet, check the **Show Alternating Row BackColor** check box and then select **Jade2018 New Brighton** from the **Select Jade Skin** combo box

Exercise 2 – Creating a JADE Macro

In this exercise, you will create macros to comment on or remove a comment from the selected text by generating or removing // at the beginning of each selected line.

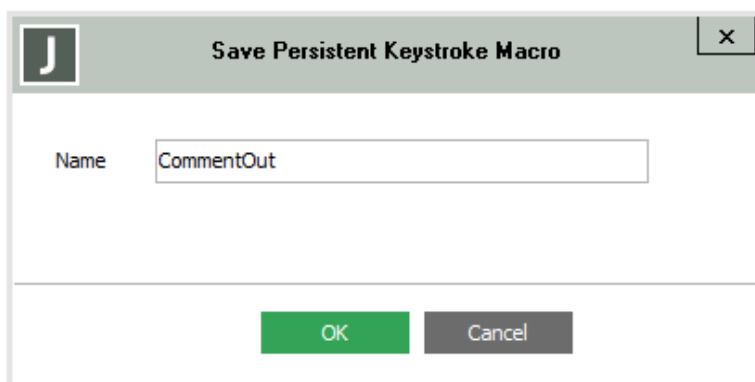
1. From the Macro submenu of the Edit menu, select the **Edit Temp Macro** command.

- Code the macro as follows, and then click **Save As**.

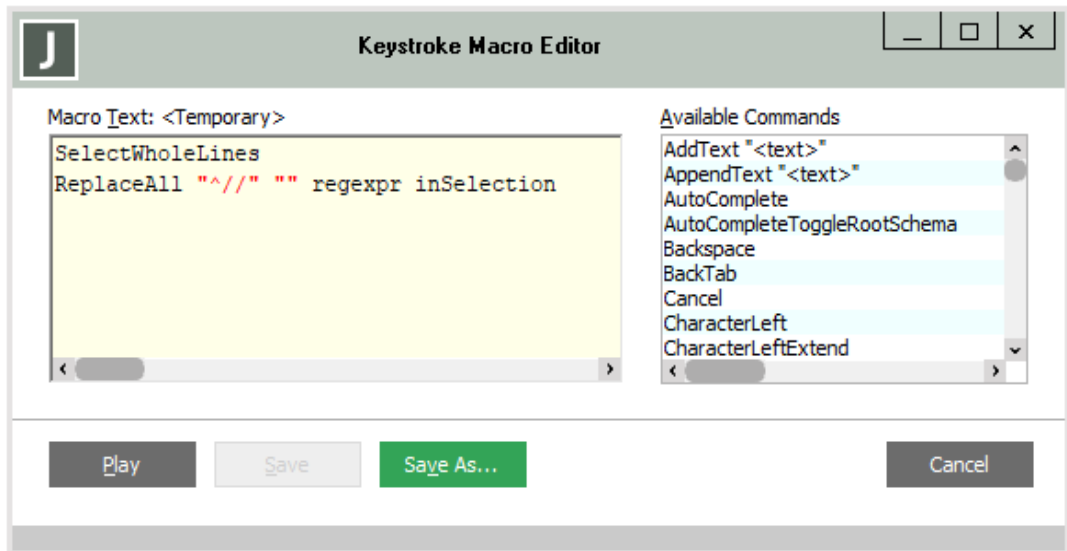


This macro finds the beginning of each selected line with a regular expression and then adds // to the start of each line.

Enter **CommentOut** as the name of the macro and then click **OK**.



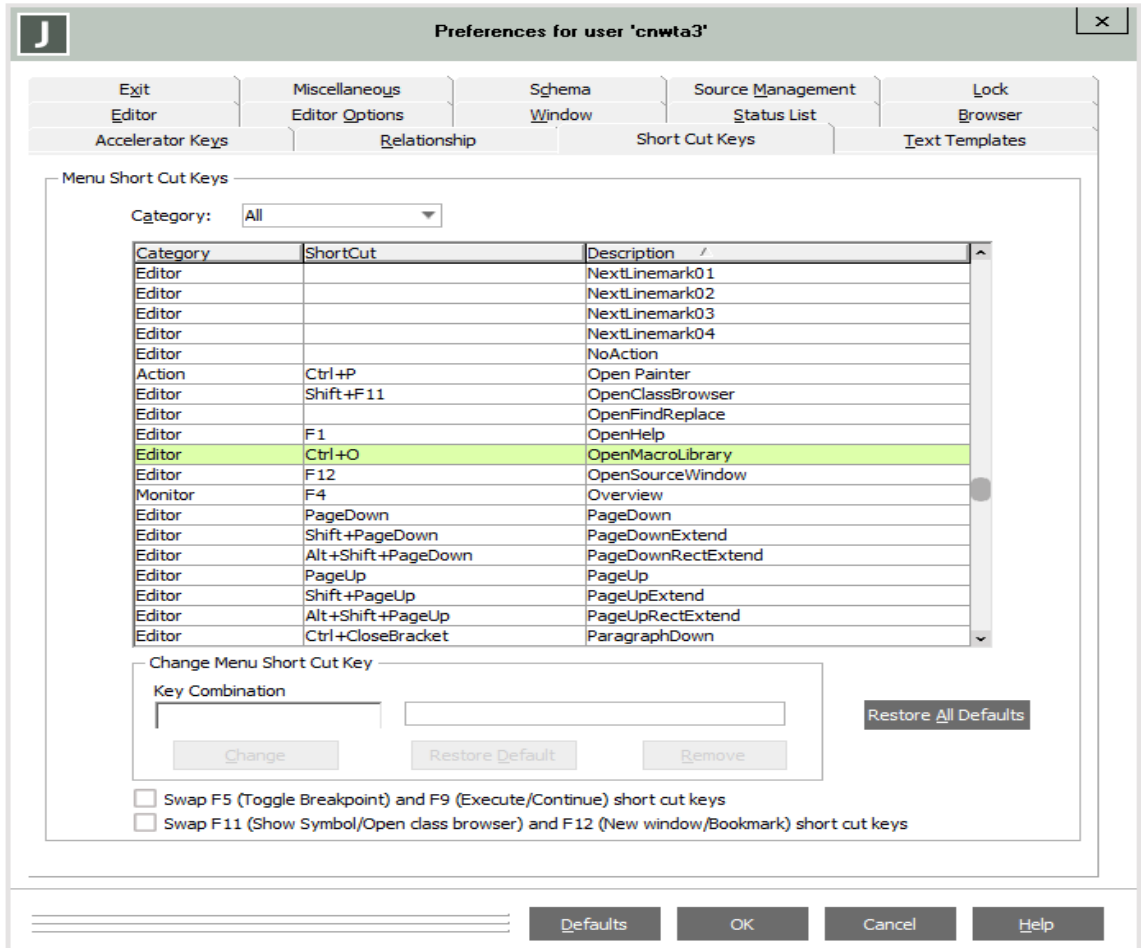
3. Reopen the Keystroke Macro Editor dialog and code a new macro as follows. Save it as **Uncomment**.



This macro finds the beginning of each selected line that begins with // and replaces the // with an empty string, which deletes the // from the line.

4. Open the Preferences window (from the **Preferences** command from the **Options** menu).

5. Add a Ctrl+O shortcut to the **OpenMacroLibrary** command.



6. Create a new JadeScript method called **m1** (in any schema, or create a new schema named **S1**, if needed) and then select the **begin** and **end;** instructions, as follows.

```

1 m1();
2 // -----
3 // Method:
4 //
5 // Purpose:
6 //
7 // Parameters:
8 //
9 // Returns:
10 // -----
11 begin
12
13 end;
14

```

7. Press Ctrl+O to open the Keystroke Macro Editor dialog, select the **CommentOut** command in the list box at the right of the dialog, and then click **Play**.

The **begin** and **end;** instructions should be commented out, along with the blank line between them.

```
1 ml();
2 // -----
3 // Method:
4 //
5 // Purpose:
6 //
7 // Parameters:
8 //
9 // Returns:
10 // -----
11 //begin
12 //
13 //end;
14
```

8. With the **begin** and **end;** instructions still highlighted, press Ctrl+O to reopen the Keystroke Macro Editor dialog and then select the **Uncomment** command in the macro list at the right. When you click **Play**, the highlighted text should no longer be commented out.

```
1 ml();
2 // -----
3 // Method:
4 //
5 // Purpose:
6 //
7 // Parameters:
8 //
9 // Returns:
10 // -----
11 begin
12
13 end;
14
```

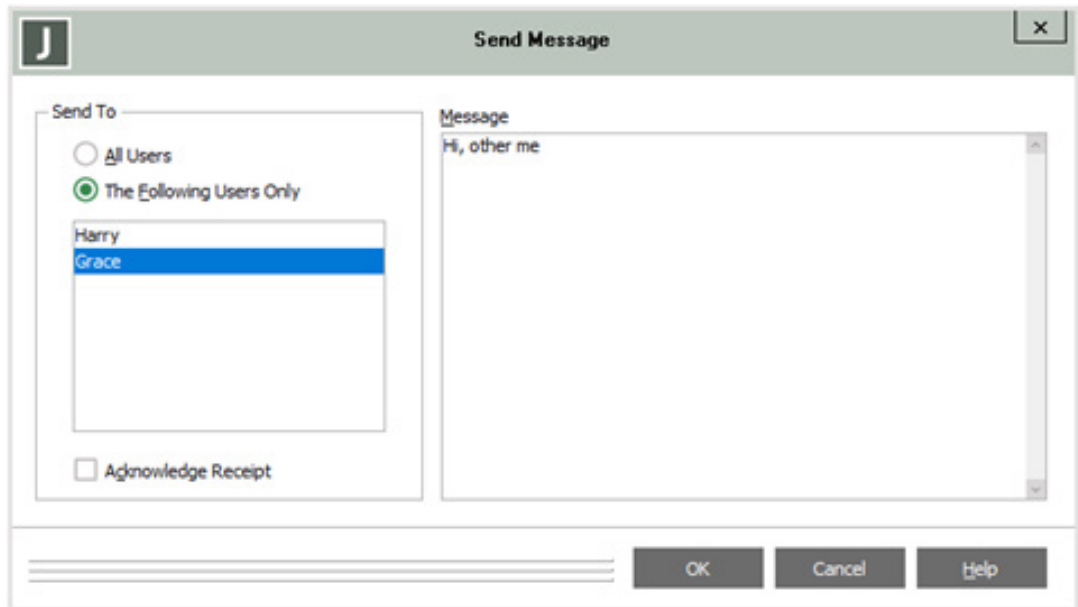
9. Delete the **m1** method (and the **S1** schema, if you created it).

Exercise 3 – Sending a Message in a Multiuser System

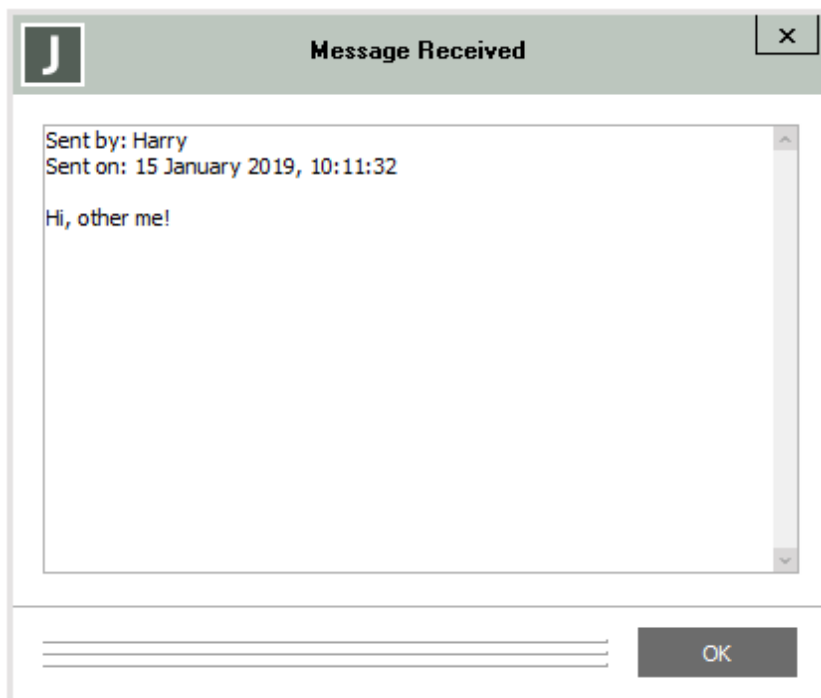
In this exercise, you will open JADE in multiuser mode and then send a message from one client to another.

1. If your JADE system is open, close it.
2. Start the JADE database server for your database.
3. Start two JADE fat (standard) client nodes for your database, signing on to each with a different name.
4. From one of the client nodes, select the **Send Message** command from the File menu.
5. Select the **The Following Users Only** option button and the user name that you selected for the other client node.

- Enter a message in the **Message** text box and then click **OK**.



- The message should appear on the other client node.



Show Symbol Command and Source Windows

When navigating around a larger system, it is often important to be able to see the implementation of an identifier when encountering it in the code base. An identifier can be a method name, a local variable, a property, a class, a constant, or an interface.

It is considered best practice to use descriptive names for these identifiers. However, when maintaining another person's code, you may need to peek at what an implementor describes.

The **Show Symbol** command, accessed by the F11 key, shows the implementation of the identifier under the caret. It will behave slightly differently, depending on the identifier that is under the caret and the values of the **Reuse Same Method Source Window** check boxes on the **Source Management** sheet of the Preferences dialog.

The behavior of the F11 key is as follows.

- For variables, properties, classes, constants, and interfaces, it creates a dialog with a description of the identifier; for example, the following will be displayed for a property.

```

clearAllAgents();
// -----
// Method:      clearAllAgents
//
// Purpose:     Remove all of our agents from this commission rate
// -----
begin
  // Clear all of our agents.
  // Automatic inverse maintenance will handle removing this commission rate from
  // each of our related agent's allCommissionRates dictionary.
  self.allAgents
end;
    
```

Name: allAgents (2)
Class: CommissionRate ()
Type: AgentByNameDict
 ----- AgentByNameDict Collection Class Details -----
Membership: Agent
Member Keys:
 name (String[80]) ascending, case sensitive, sort order: Binary
 Duplicates are not allowed
Access: readonly
SubId: 1
Ordinal: 2
 non-virtual
Inverses:
 allCommissionRates of Agent
Update Mode: Manual/Automatic
Relationship Type: peer-to-peer

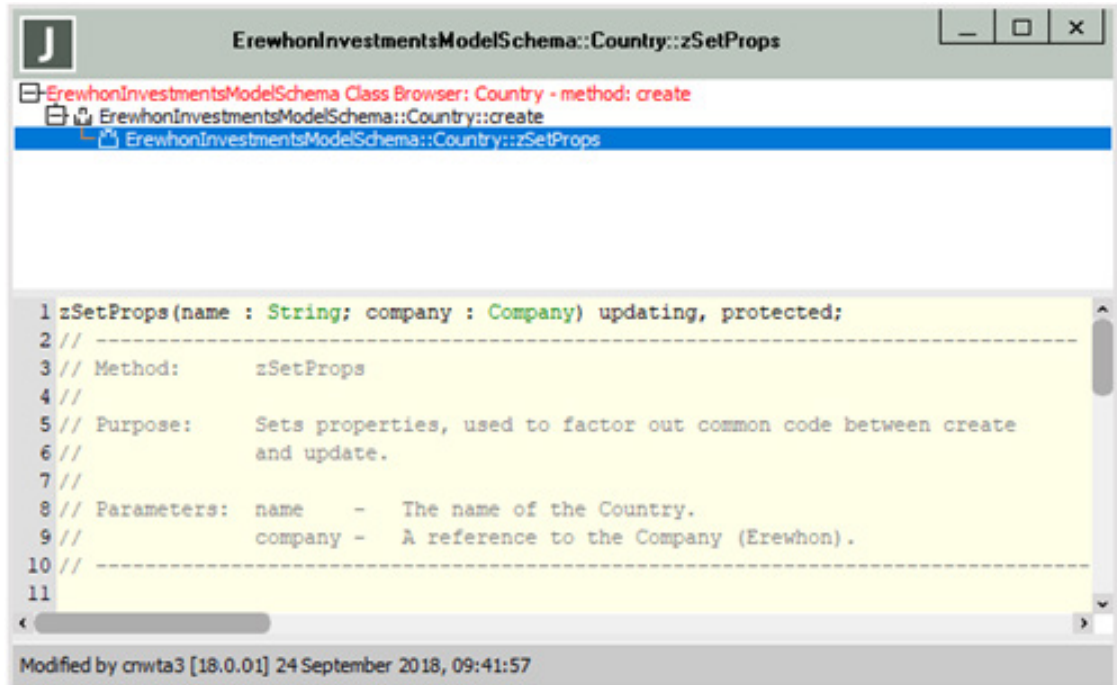
- For methods when the **Reuse Same Method Source Window** check boxes on the **Source Management** sheet are unchecked, the method is opened in a new window.

```

1 raiseModelException(errorNo : Integer);
2 // -----
3 // Method:      raiseModelException
4 //
5 // Purpose:     Raises an exception of type ModelException.
6 //             The errorCode of the exception is set to the supplied error number.
7 // -----
8 vars
9 // exceptionObject : ModelException;
10
11 exceptionObject : Exception;
12
13 begin
14   create exceptionObject transient;
15   exceptionObject.errorCode := errorNo;
16   exceptionObject.errorItem := self.getErrorString(errorNo);
17   raise exceptionObject;
18 end;
    
```

Modified on 08 August 2018, 12:06:06

- For methods when the **Reuse Same Method Source Window For All** check box on the **Source Management** sheet is checked, the method is added to a common method source form, shared amongst all opened methods.



- If the **Reuse Same Method Source Window For Each Origin** check box on the **Source Management** sheet is enabled, the method is added to a method source window that is common to methods opened from the same Class Browser.

References and Implementors

The JADE development environment enables you to list all:

- References (the methods that call a specific method or that read or update a specific property) for a method or property
- Implementors (other methods with the same name) for a method

Find references of a property by right-clicking a property or from the Properties menu while the property is selected, and then selecting any of the following.

- References
Shows all references to the property, and whether they read the property or update it
- Read References
Shows only the references to the property that read the property
- Update References
Shows only the references to the property that update the property

To find the references to a method, you can search for all references or for local references only.

To search for all references across all classes both within the current schema and any subschemas, right-click on the method and then select **References** or select the **References** command from the Methods menu when the method is selected.

Alternatively, you can search for the local references only; that is, references to the method that are within the method’s class or one of its subclasses, and in the current schema only. To search for local references only, right-click on the method and select **Local References** or select the **Local References** command from the Methods menu when the method is selected.

Implementors of a method are most commonly used for finding a method’s reimplementations. As such, all reimplementations of the method within the class hierarchy, but including subschemas, can be found using the **Local Implementors** command. Access this command by right-clicking on a method and then selecting **Local Implementors** or by selecting the **Local Implementors** command from the Methods menu when the method is selected.

Alternatively, you can locate all methods within the schema with the same name as a method; for example, to see which classes have a **toString** method implemented. The **Implementors** command finds all methods that share a name with a method across all classes of the schema, but it does not search subschemas. To search for implementors of a selected method, right-click on the method and then select **Implementors** or select the **Implementors** command from the Methods menu when the method is selected.

Renaming Identifiers

When renaming an identifier, special care must be taken to ensure that the references to the identifier refer to the new name after it changes.

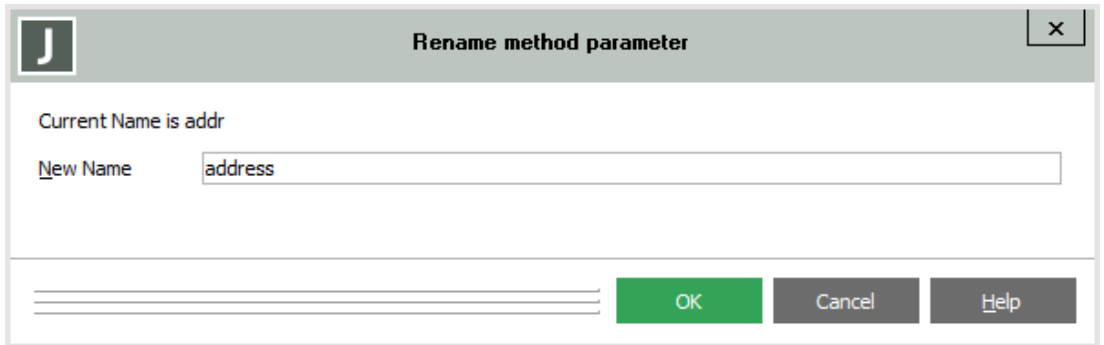
The JADE development environment provides the ability to perform this task automatically, avoiding the risk of human error creating compilation errors.

The steps required to safely rename an identifier vary, based on the type of identifier, as follows.

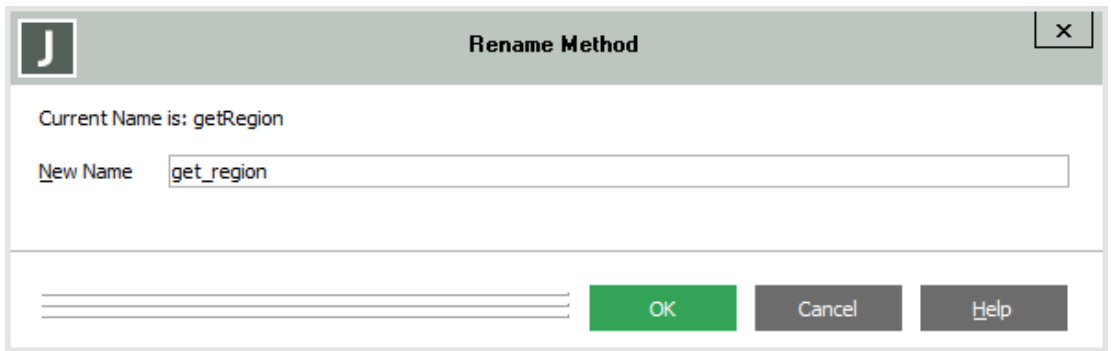
- For classes, properties, and references, right-click on the property and then select **Change**.

Modify the **Name** value as required, so that all references to the new name are updated automatically when you click **OK**.

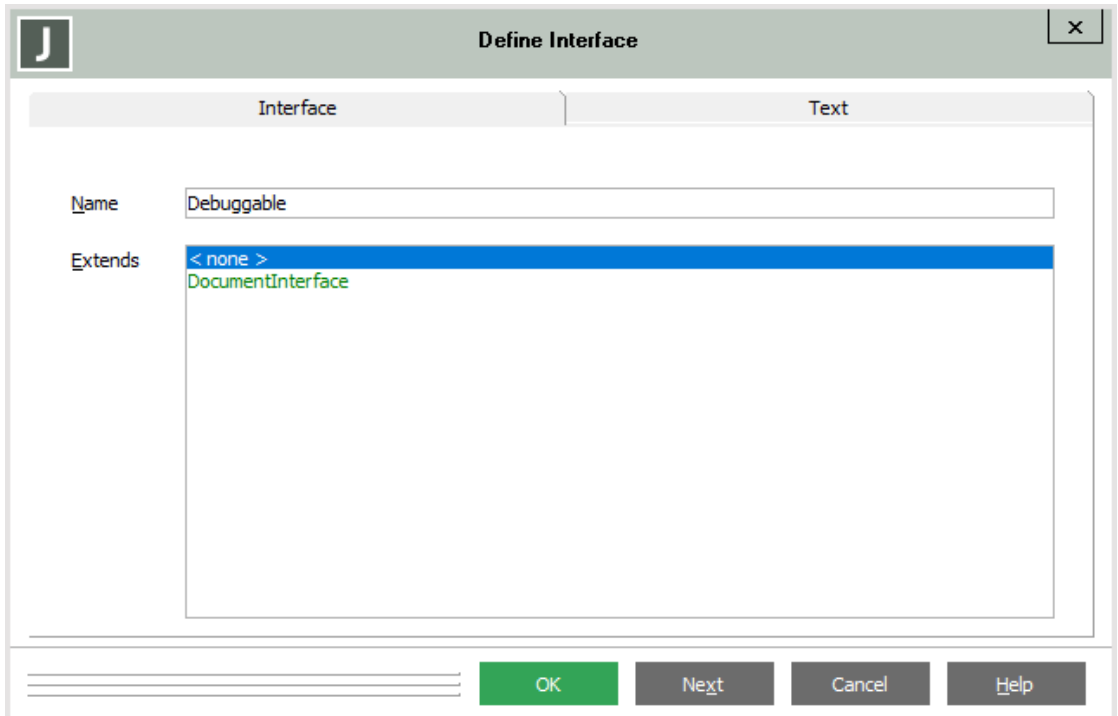
- For method parameters and local variables, right-click on the parameter or variable and then select the **Rename / Change** command from the Refactor submenu.



- For method names, right-click on the method and then select **Rename**, or select the **Rename** command from the Methods menu while the method is selected.



- For interfaces, open the Interfaces Browser (Ctrl+N), right-click on the interface, and then select **Change**. Modify the **Name** value as required, so that all references to the new name are updated automatically when you click **OK**.



Caution Schemas cannot be renamed once they have been created, so pick the name carefully.

Exercise 4 – Finding and Fixing Bugs

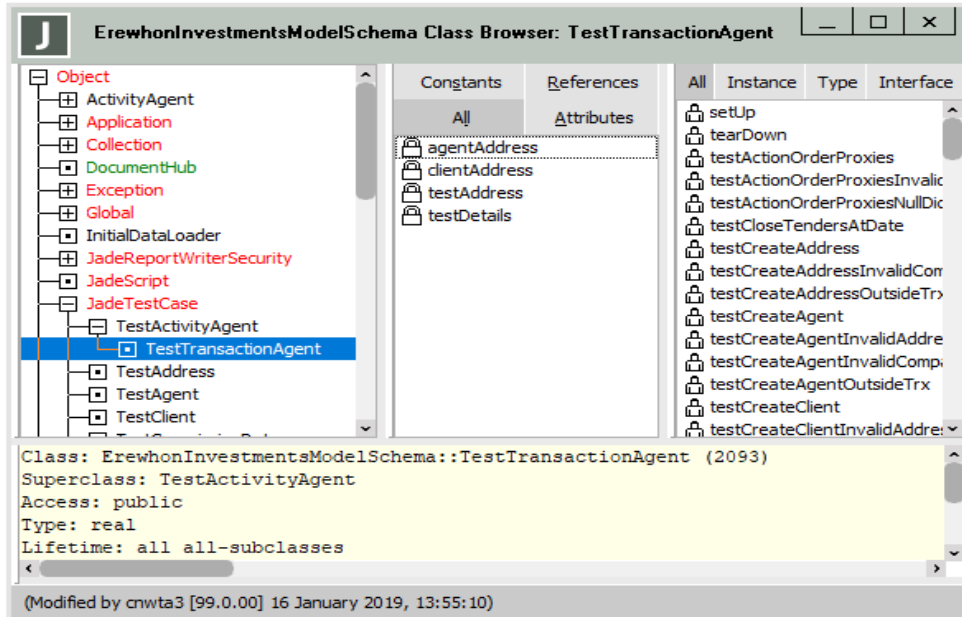
In this exercise, you will locate and resolve several defects that have been intentionally introduced into the Erewhon demonstration system, by using the **Show Symbol**, **References**, and **Implementors** commands to navigate around the system and find the methods that are causing the provided unit tests to fail.

1. Load the Erewhon system provided in the **BrokenErewhon** folder of the flash drive, by selecting the **Load** command of the Schema menu, checking the **Load Multiple Schemas** check box, and then selecting **ErewhonInvestments.mul** as the schema file name.
2. Open **ErewhonInvestmentsModelSchema** in the Class Browser by selecting it in the Schema Browser and then using the Ctrl+B keyboard shortcut.
3. Select the **JadeTestCase** class and then press F9 to run the provided tests. You should see that the following tests fail.
 - TestTransactionAgent::testCreateCommRateOutsideTrx
 - TestTransactionAgent::testCreateCommissionRate
 - TestTransactionAgent::testUpdateCountry
 - TestClient::testGetAllSales
 - TestCountry::testInvalidName
 - TestCountry::testUpdate
 - TestSale::testGetDate
 - TestRetailSale::testGetDate
 - TestRetailSaleItem::testGetDebugString
 - TestTenderSaleItem::testGetDebugString
 - TestSaleItemCategory::testCreateCommissionRate
4. For each of these test failures, find the failing test in the Class Browser and use F11 key to show the implementation of the method that is being tested by the unit test. Find the defect in the tested method and re-run the tests, to check that it is now fixed.

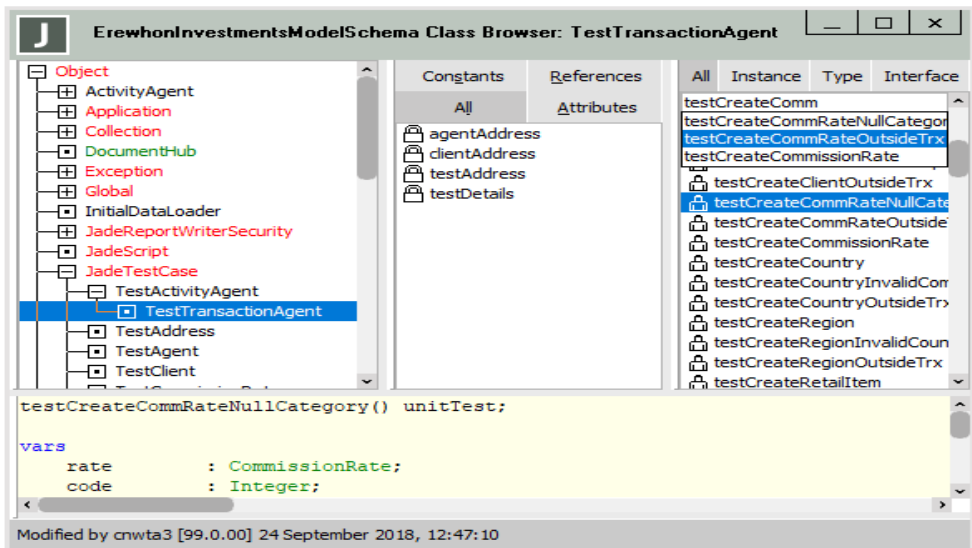
To get you started, the fix for the first failing test is given as an example.

- a. The failing test is **TestTransactionAgent::testCreateCommRateOutsideTrx**, and the test failed with a 1090 error on the assert; that is, the created object was null when we tried to read it. We can therefore deduce that the method completed without error but did not create the object that it was supposed to.

- b. Find the **TestTransactionAgent** class in the Class Browser.

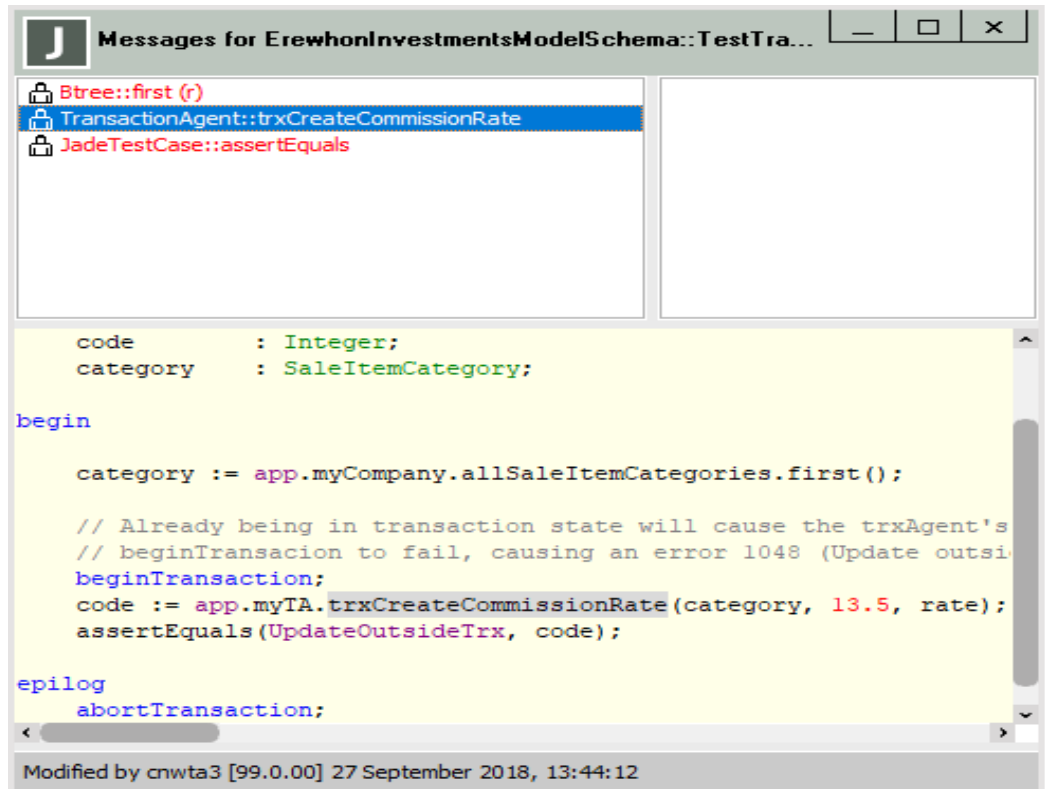


- c. Use the Ctrl+7 key to display the method search text box at the top of the Methods List, and search for **testCreateCommRateOutsideTrx**.



- d. Click the **testCreateCommRateOutsideTrx** method and then select the **Messages** command from the Methods menu. In this window, the two messages in red are calls to methods of system classes, which are not going to be the subject of user-defined unit tests.

As such, we can see that the one user method, **trxCreateCommissionRate**, must be the one being tested. Clicking on this method in the messages list will highlight it in the method source.



The screenshot shows a window titled "Messages for ErewhonInvestmentsModelSchema::TestTra...". The messages list on the left contains three entries: "Btree::first (r)", "TransactionAgent::trxCreateCommissionRate" (highlighted in blue), and "JadeTestCase::assertEquals". The code editor on the right shows the following code:

```
code      : Integer;
category  : SaleItemCategory;

begin

    category := app.myCompany.allSaleItemCategories.first();

    // Already being in transaction state will cause the trxAgent's
    // beginTransaction to fail, causing an error 1048 (Update outside
beginTransaction;
code := app.myTA.trxCreateCommissionRate(category, 13.5, rate);
assertEquals(UpdateOutsideTrx, code);

epilog
    abortTransaction;
```

Modified by cnwta3 [99.0.00] 27 September 2018, 13:44:12

- e. Clicking on the highlighted method and then pressing F11 will navigate to the implementation of the method, which is as follows.



```

J ErewhonInvestmentsModelSchema:TransactionAgent:trxCreateCommissionRate
trxCreateCommissionRate (
    saleItemCategory : SaleItemCategory;
    percentage       : Decimal;
    commissionRate   : CommissionRate output) : Integer;
-----
// Method:      trxCreateCommissionRate
//
// Purpose:     Transaction method to create a CommissionRate
//
// Parameters:  Parameters correspond to each of the properties of a CommissionRate.
//              The newly created rate is returned in the output parameter.
//
// Returns:     0 if transaction successful, otherwise a model error number (refer
//              to the global constants in this schema for a list of error numbers)
//
-----
begin
    // Always initialize the activity first
    self.zInitializeActivity;

    // Arm our base exception handlers
    on Exception do self.zExceptionHandler(exception);
    on LockException do self.zLockExceptionHandler(exception);

    if saleItemCategory = null then
        return self.zSetErrorCode(InvalidSaleItemCategory);
    endif;

    // If saleItemCategory is invalid, our handler will return InvalidSaleItemCategory
    self.zRegisterObjectAndErrorCode(saleItemCategory, InvalidSaleItemCategory);

    // Start the transaction
    beginTransaction;

    // Perform the operation
    commissionRate := saleItemCategory.createCommissionRate (percentage);

    // If "createCommissionRate" raises an exception, our handler will resume to here
    if app.noErrors then
        // No errors, so commit
        commitTransaction;
    else
        // We got an error so abort the transaction. Although TransactionAgent
        // exception handlers abort the transaction before resuming, we do so
        // here as well for completeness and symmetry with the beginTransaction.
        // An abortTransaction when not in transaction state does nothing.
        abortTransaction;
    endif;
end;

epilog
    // Always return the error code to the caller (which will be zero if no errors)
    return app.getLastError;
end;
-----
Modified on 08 August 2018, 12:06:06

```

- f. This method appears to be a wrapper method for another method, **createCommissionRate**.

To see the **createCommissionRate** method, select **createCommissionRate** and then press F11.

```

ErewhonInvestmentsModelSchema::SaleItemCategory::createCommissionRate
-----
createCommissionRate(percentage : Decimal) : CommissionRate;
// -----
// Method:      createCommissionRate
//
// Purpose:     Creates a new CommissionRate for the receiver category
//
// Parameters:  percentage - what percent of the sale price the agent takes as
//              commission.
//
// Returns:     The newly created CommissionRate
// -----
vars
    commRate : CommissionRate;

begin
    return commRate;|
    commRate := create CommissionRate(self, percentage);
end;
-----
Modified by cnwta3 [99.0.00] 16 January 2019, 14:04:40
    
```

- g. In this method, we can see that the **CommissionRate** is being returned before it is created.

To fix this, we simply reverse the order of the two statements and recompile (using the F8 key).

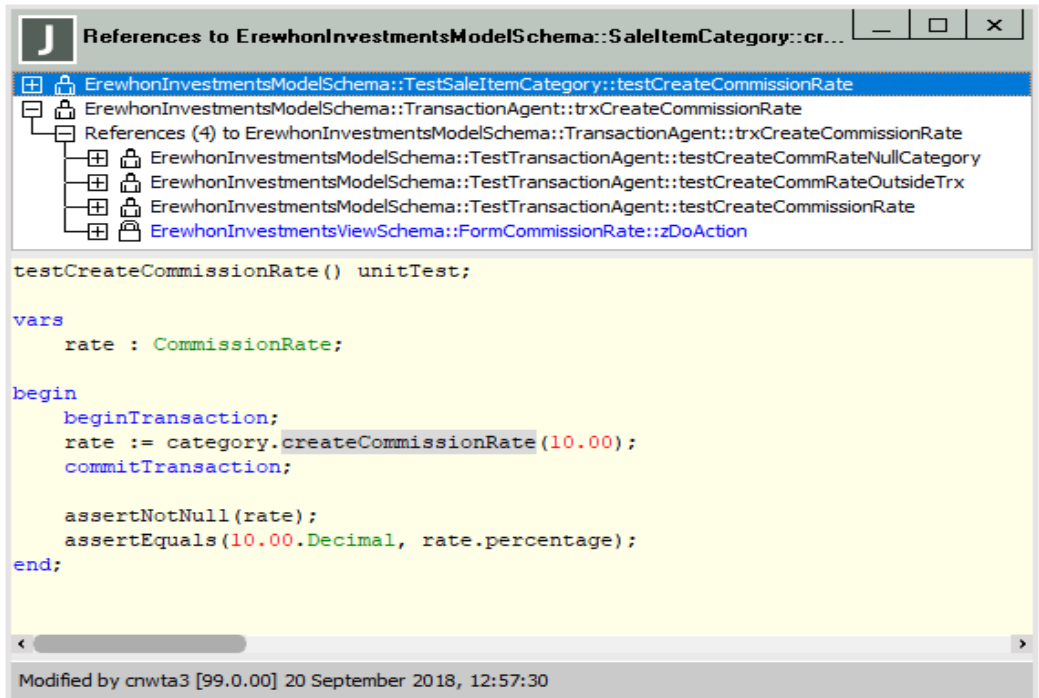
```

ErewhonInvestmentsModelSchema::SaleItemCategory::createCommissionRate
-----
createCommissionRate(percentage : Decimal) : CommissionRate;
// -----
// Method:      createCommissionRate
//
// Purpose:     Creates a new CommissionRate for the receiver category
//
// Parameters:  percentage - what percent of the sale price the agent takes as
//              commission.
//
// Returns:     The newly created CommissionRate
// -----
vars
    commRate : CommissionRate;

begin
    commRate := create CommissionRate(self, percentage);
    return commRate;|
end;
-----
Compilation complete - no errors
    
```

- h. If we re-run the tests, we see that the following tests are no longer failing.
 - TestTransactionAgent::testCreateCommRateOutsideTrx
 - TestTransactionAgent::testCreateCommissionRate
 - TestSaleItemCategory::testCreateCommissionRate

- i. To see why these are no longer failing, we can view the references to the **createCommissionRate** method. With focus on the **createCommissionRate** method, select the **References** command from the Methods menu.



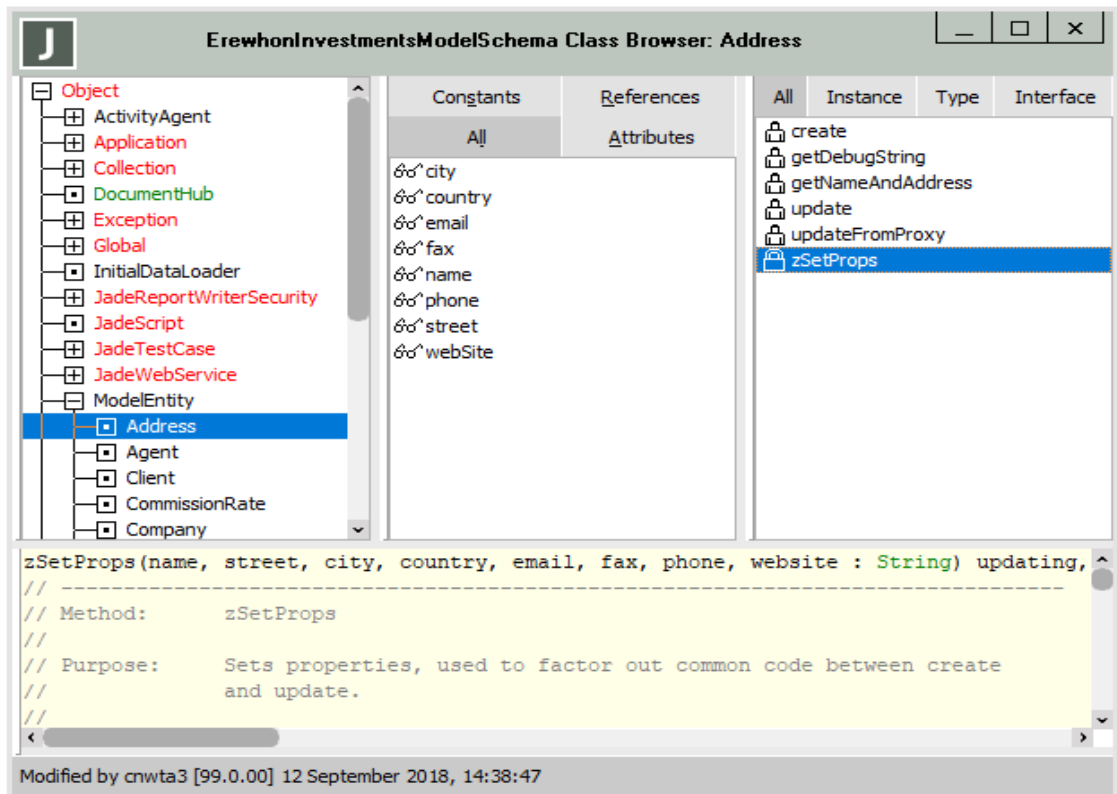
- j. There are four test methods included in the references: the three that are no longer failing and the **TestTransactionAgent::testCreateCommissionRateNullCategory** method, which is an exception-flow test that verifies that the method rejects invalid inputs.
5. The following failing tests remain. Identify and remove the errors so that all tests pass.
- TestTransactionAgent::testUpdateCountry
 - TestClient::testGetAllSales
 - TestCountry::testInvalidName
 - TestCountry::testUpdate
 - TestSale::testGetDate
 - TestRetailSale::testGetDate
 - TestRetailSaleItem::testGetDebugString
 - TestTenderSaleItem::testGetDebugString

Tip Some fixes will solve multiple failing tests.

Exercise 5 – Refactoring Names

In this exercise, you will rename various entities and use the **References** and **Implementors** commands to verify that the renaming has correctly modified the entities' references and implementors.

1. In the Class Browser for the **ErewhonInvestmentsModelSchema**, navigate to the **Address** class, which is a subclass of **ModelEntity**, and find the **zSetProps** method.



Note The **z** prefix on private methods is an old naming convention that was used to distinguish between public and private methods before the functionality to add a visual aid to methods in the Class Browser was implemented.

2. With the **zSetProps** method highlighted, select the **Implementors** command from the Methods menu to see all classes in the current schema that implement this method.
3. For each method found, perform the following actions.
 - a. With the method highlighted, select the **References** command from the Methods menu to see which methods refer to that specific **zSetProps** method.
 - b. Select the **Rename** command from the Methods menu and rename the method to **setProperties**.
 - c. Check the references list for the method to ensure that all references have been updated to **setProperties**.

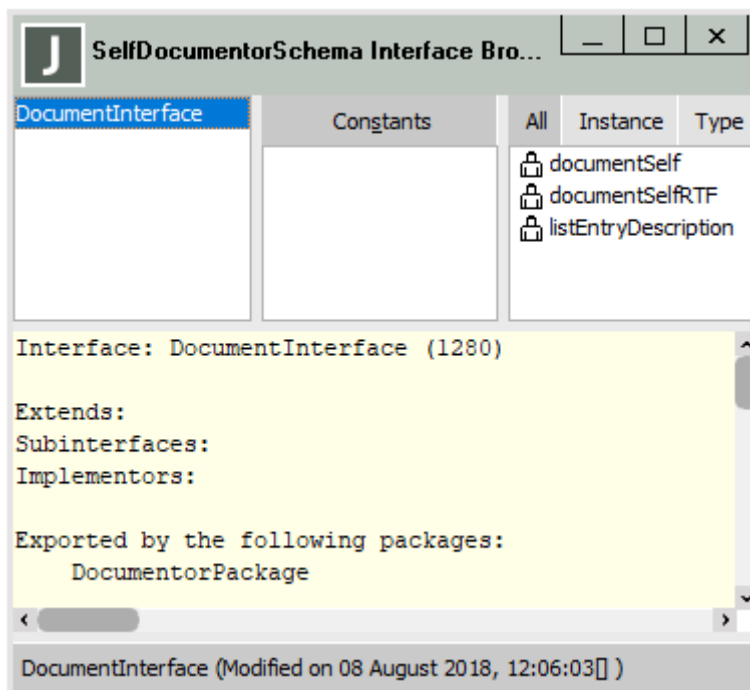
Note You will see that the method header comment at the top of the changed method still includes **Method: zSetProps**. Comments are unaffected by the renaming of identifiers, which is one disadvantage to the use of method headers, as they can become out of date with the actual names and behaviors within the method.

4. In the Class Browser for **ErewhonInvestmentsModelSchema**, navigate to the **InitialDataLoader** class and then find the **zGetNextToken** method.
5. Rename the following parameters and variables by right-clicking on them and then selecting the **Rename / Change** command from the **Refactor** submenu.

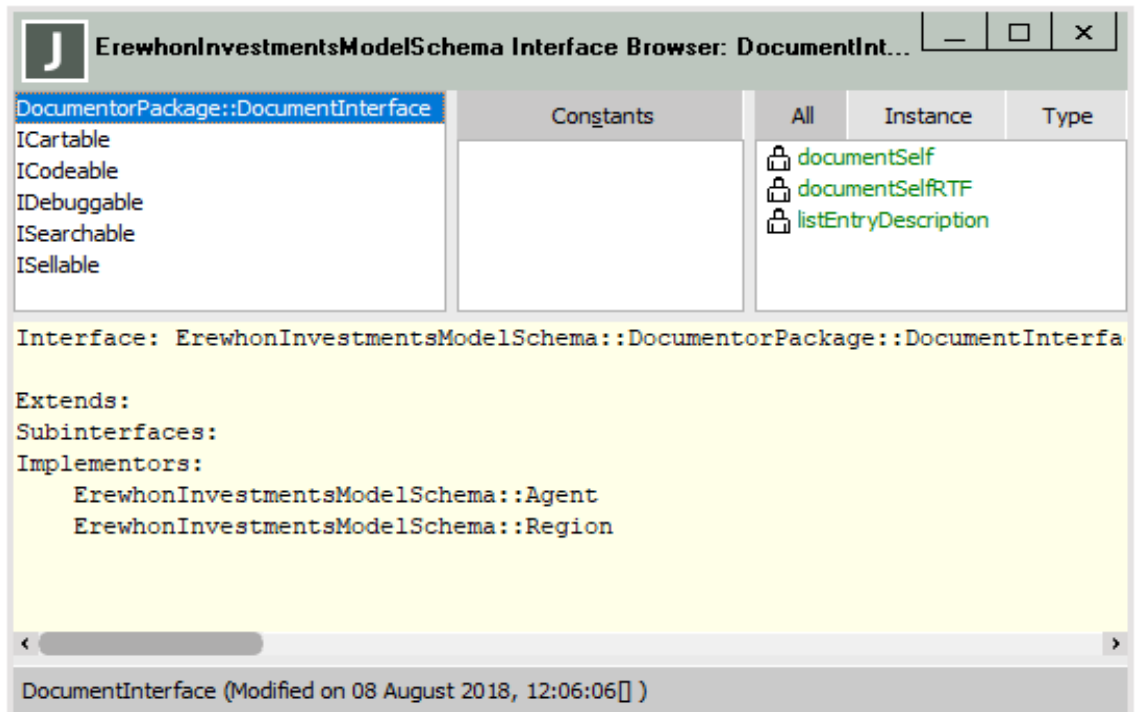
Old Name	New Name	Identifier Type
str	targetString	Parameter
pos	currentPosition	Parameter
len	stringLength	Variable
idx	currentIndex	Variable

Tip You can remove the end-of-line comments for the variables, as it is better practice to use descriptive variable names rather than abbreviations with comments describing what they represent.

6. Select **SelfDocumentorSchema** in the Schema Browser and then press Ctrl+N to open the Interface Browser. You will see that this schema contains one interface: **DocumentInterface**.



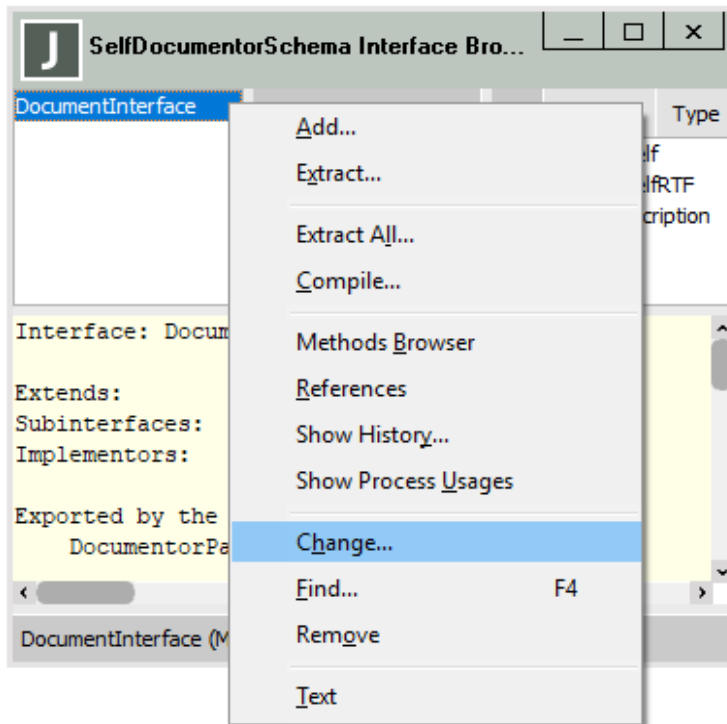
7. With the Interface Browser still open, select **ErewhonInvestmentsModelSchema** in the Schema Browser and then press Ctrl+N to open another Interface Browser.



In this Interface Browser, you can see that all Erewhon interfaces begin with a capital **I** prefix and are of the form *Iverbable*. This is one of the two common interface naming conventions, along with the form *responsibilityF*, which is used in the RootSchema interfaces.

However, **DocumentInterface** does not conform to either style, so we will rename it.

- From the **SelfDocumentorSchema** Interface Browser, right-click **DocumentInterface** and then select the **Change** command.



- Rename the interface to **IDocumentable** and then click **OK**, to save the change.

Unused Variables

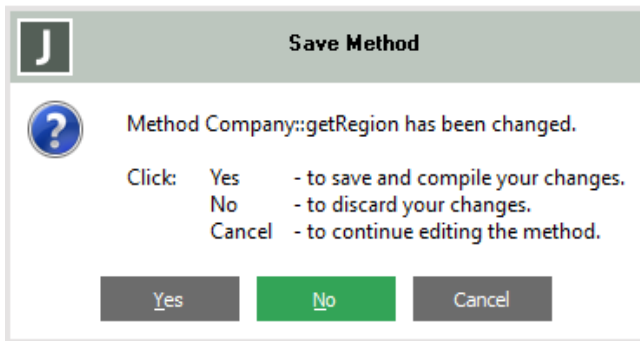
In the JADE language, you must always declare a variable before you use it; that is, attempting to use an undeclared variable results in compiler error 6027 (*Unknown identifier*).

Although no error is generated when declaring a variable and never using it in the method, it is undesirable behavior because as unused variables make the code more cluttered and less-readable.

The JADE development environment includes the ability to find and optionally remove these unused variables. To remove unused variables for a specific method, right-click on the method and then select the **Unused Local Variables** command. The Find Unused Variables dialog is then displayed, advising you that the specified local variable is unused. Buttons enable you to click:

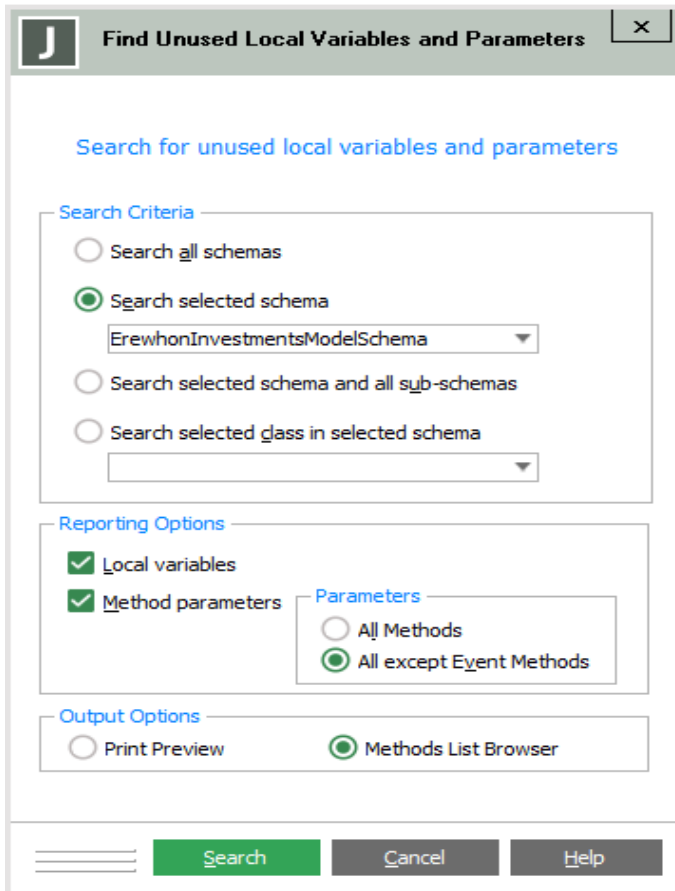
- **Find Next**, to skip the current unused variable and go to the next one
- **Remove**, to remove the current unused variable
- **Remove All**, to remove all unused variables from the method, including those that were previously skipped
- **Cancel**, to close the window as well as removing any previously removed unused variables

After removing unused variables, the method will not automatically save or compile. If you want to compile it, locking in the removal, press F8 or select the **Compile** command from the Methods menu.



Alternatively, if you navigate away from the method without saving or compiling it and then click **No** on to the Save Method dialog, the unused variables are returned to the method.

You can also search entire for unused variables in classes, schemas, or even the whole database by selecting the **Find Unused Local Variables/Parameters** command from the Schema menu when the Schema Browser has focus.



In the Find Unused Local Variables and Parameters dialog, specify the scope of the search by selecting one of the option buttons in the Search Criteria group box. To search:

- All schemas in the database, select **Search all schemas**
- Within a specific schema, select **Search selected schema** and then select the required schema from the associated combo box

- Within a specific schema and any subschemas of that schema, select **Search selected schema and all sub-schemas** and then select the required schema from the associated combo box

Note The **Search selected schema**, **Search selected schema and all sub-schemas**, and **Search selected class in selected schema** options share the schema selection combo box (that is, the first one in the Search Criteria group box).

- Within a specific class of a specific schema, select the **Search selected class in selected schema** option and then select the schema from the first combo box and the class from the second combo box

You can also specify the reporting options, by selecting options within the Reporting Options group box as follows.

- To skip local variables and search only for method parameters, uncheck the **Local variables** check box.
- To skip method parameters and search only for local variables, uncheck the **Method parameters** check box.

Note You must select at least one of these check boxes.

- If the **Method parameters** check box is checked, you can select the **All Methods** or **All except Event Methods** option button in the Parameters group box. Many event methods will have an unused parameter, so typically the **All except Event Methods** default option button should be selected. However, you can select the **All Methods** option button to include them.

In the Output Options group box, the default **Methods List Browser** option button causes the results of the search to display in a methods list to make them easier to resolve. However, if you want to print them instead, select the **Print Preview** option button.

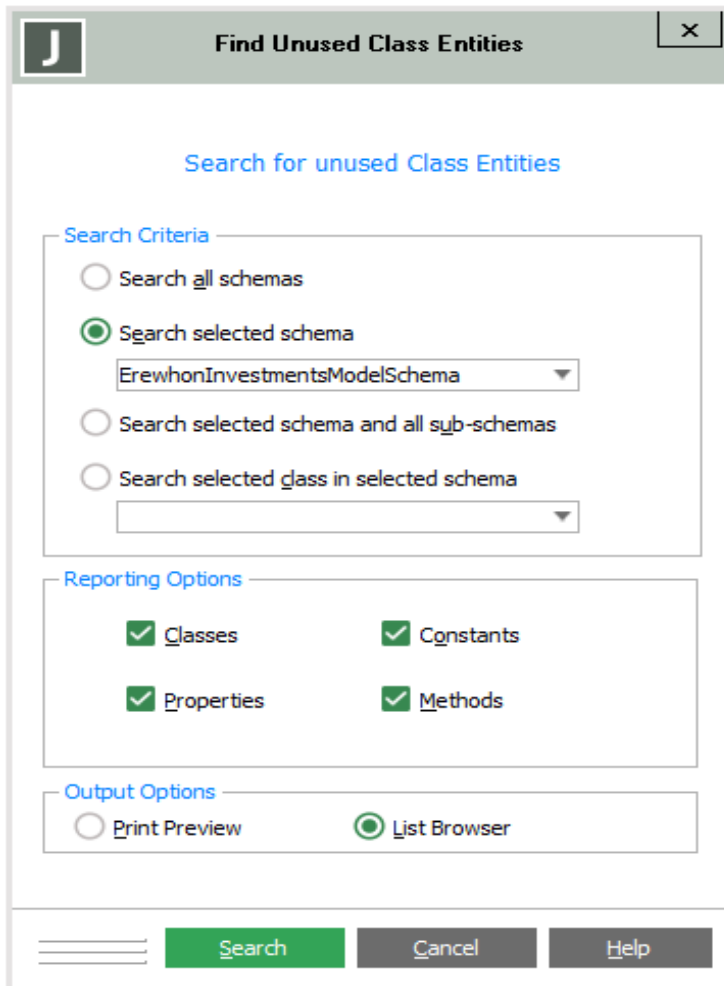
Unused Class Entities

As a code base grows and is maintained, it is common for dead code to arise; that is, code that is never called and therefore provides no value to the system but makes the code base larger and therefore harder to maintain.

The JADE development environment provides the ability to locate and delete unused class entities to combat the accumulation of dead code.

While the Find Unused Class Entities dialog is a fast and powerful way to find and remove dead code, it will err on the side of avoiding the removal of possibly useful code so there may still be dead code remaining after its use.

To open the Find Unused Class Entities dialog, select the **Find Unused Class Entities** command from the Schema menu in the Schema Browser.



In this dialog, you can specify the scope of the search by selecting one of the option buttons in the Search Criteria group box. To search:

- All schemas in the database, select **Search all schemas**
- Within a specific schema, select **Search selected schema** and then select the required schema from the associated combo box
- Within a specific schema and any subschemas of that schema, select **Search selected schema and all sub-schemas** and then select the required schema from the associated combo box

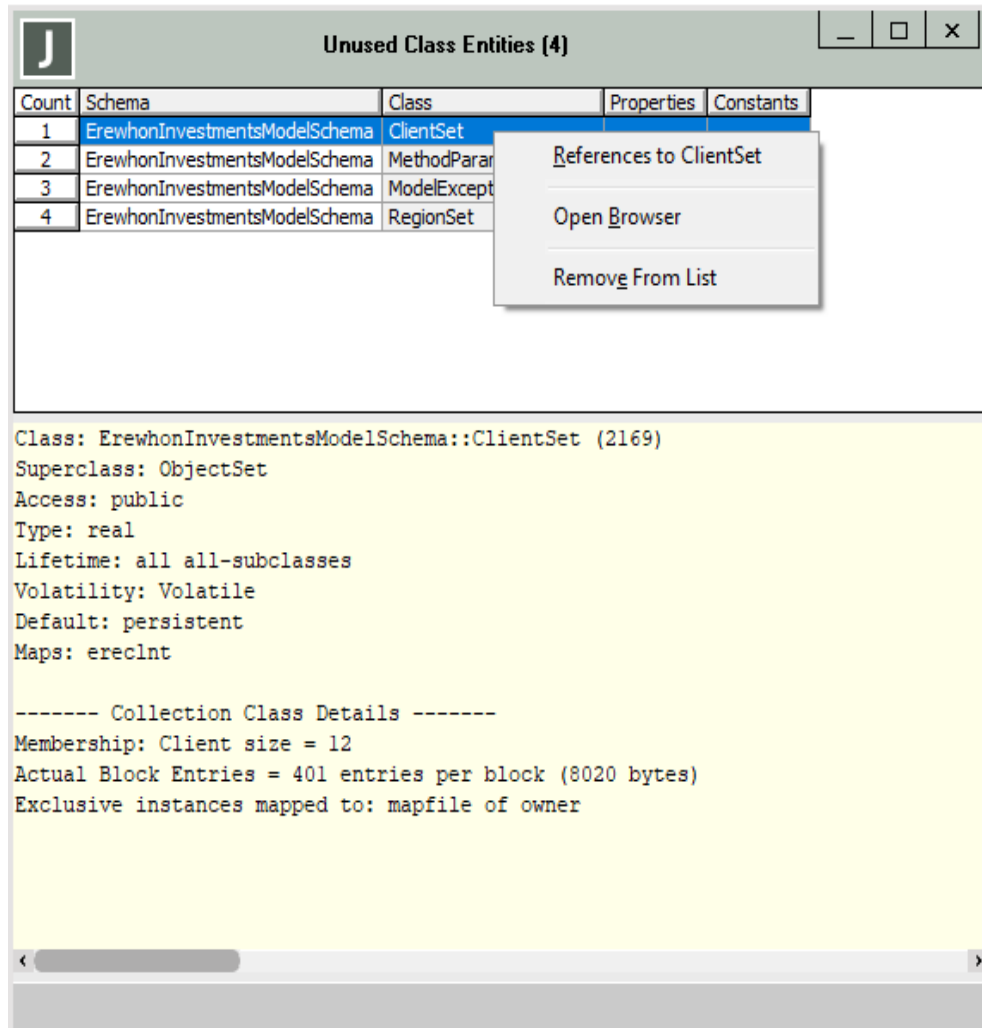
Note The **Search selected schema**, **Search selected schema and all sub-schemas**, and **Search selected class in selected schema** options share the schema selection combo box (that is, the first one in the Search Criteria group box).

- Within a specific class of a specific schema, select the **Search selected class in selected schema** option and then select the schema from the first combo box and the class from the second combo box

In the Reporting Options group box, check the check box of each type of class entity you want find.

In the Output Options group box, the default **List Browser** option button causes the results of the search to display in a methods list to make them easier to resolve. However, if you want to print them instead, select the **Print Preview** option button.

After clicking **Search**, the Unused Class Entities dialog lists any unused entities that match your selected criteria.



Select an unused entity in the table and then right-click to access a popup (context) menu with the following commands.

- **References to *entity-name*** displays any references to the entity. While this will usually be none, a class can have a method referencing itself while still being unused.
- **Open Browser** displays the entity in a new Class Browser.
- **Remove From List** removes the entity from the current Unused Class Entities dialog but it does not remove it from its schema or prevent it from being located the next time an unused class entities search is performed.

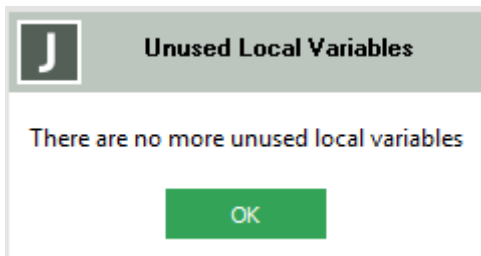
Exercise 6 – Locating and Removing Unused Variables

In this exercise, you will locate and remove unused variables, first in a specific method and then in an entire schema.

1. In the **ErewhonInvestmentsModelSchema**, navigate to the **InitialDataLoader** class **zLoadAgents** method. You will see that there are many declared variables in the method.

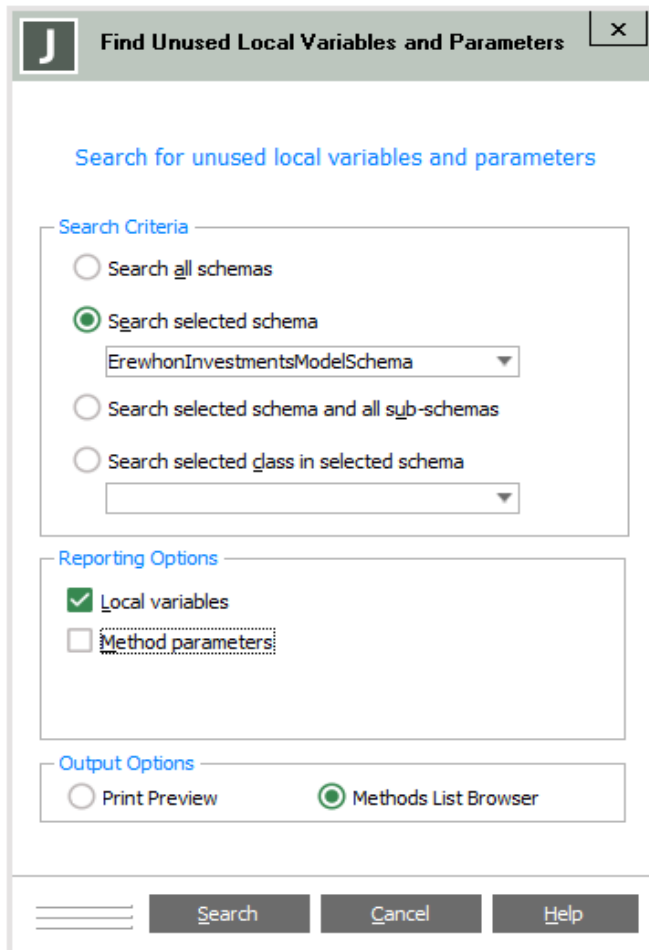
```
vars
  inputFile      : File;
  agent          : Agent;
  agentAddress   : Address;
  fileName       : String;
  line           : String;
  agentName      : String;
  street         : String;
  address        : String;
  address2       : String;
  address3       : String;
  email          : String;
  phone          : String;
  fax            : String;
  web            : String;
  pos            : Integer;
  startClock     : Integer;
```

2. Find the three unused variables by selecting the **Unused Local Variables** command from the Methods menu and then clicking **Find Next** until the following message box is displayed.



3. Reopen the Find Unused Variables dialog and then click **Remove All**. The three unused variables are then removed from the method.
4. Press F8 to compile the method.

5. Select **ErewhonInvestmentsModelSchema** in the Schema Browser and then select the **Find Unused Local Variables/Parameters** command from the Schema menu.



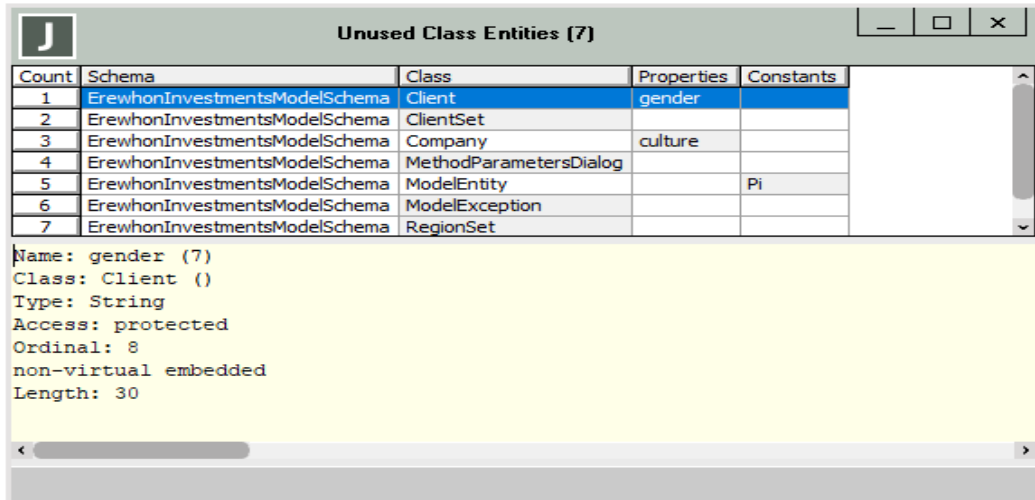
6. Uncheck the **Method parameters** check box and then and click **Search**. This narrows the search to unused local variables only.
7. For each unused variable that is found, double-click it in the Unused Local Variable and Parameters dialog. The unused variable is then removed from the method, the method is automatically compiled, and the entry is removed from the list.

Note You can use the double-click shortcut on unused variables only. If a method has an unused parameter, more care must be taken as the removal of that parameter will change the method signature, which may affect other methods.

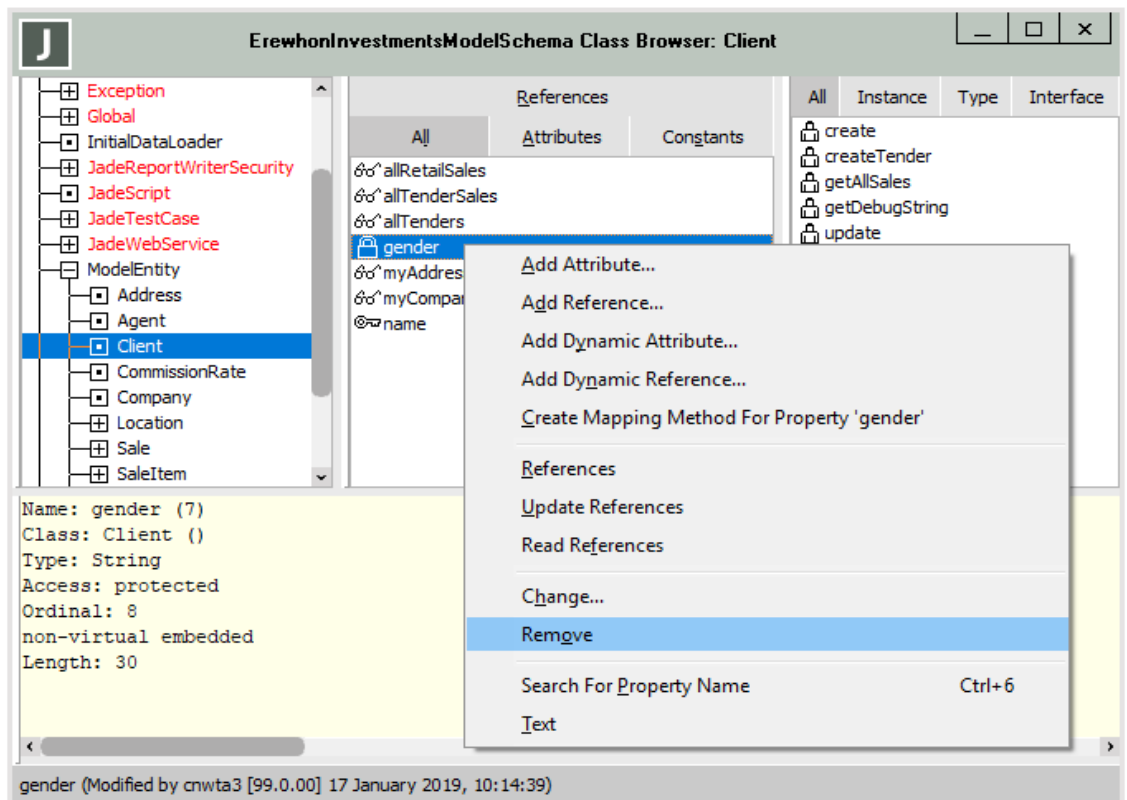
Exercise 7 – Locating and Removing Unused Class Entities

In this exercise, you will locate and remove several unused classes, two unused properties, and an unused constant by using the **Find Unused Class Entities** command.

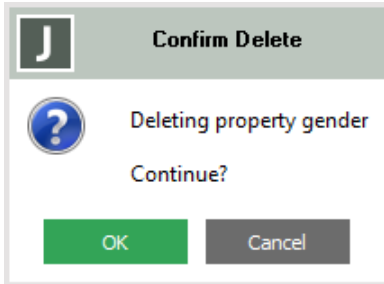
1. Select **ErewhonInvestmentsModelSchema** in the Schema Browser and then select the **Find Unused Class Entities** command from the Schema menu.



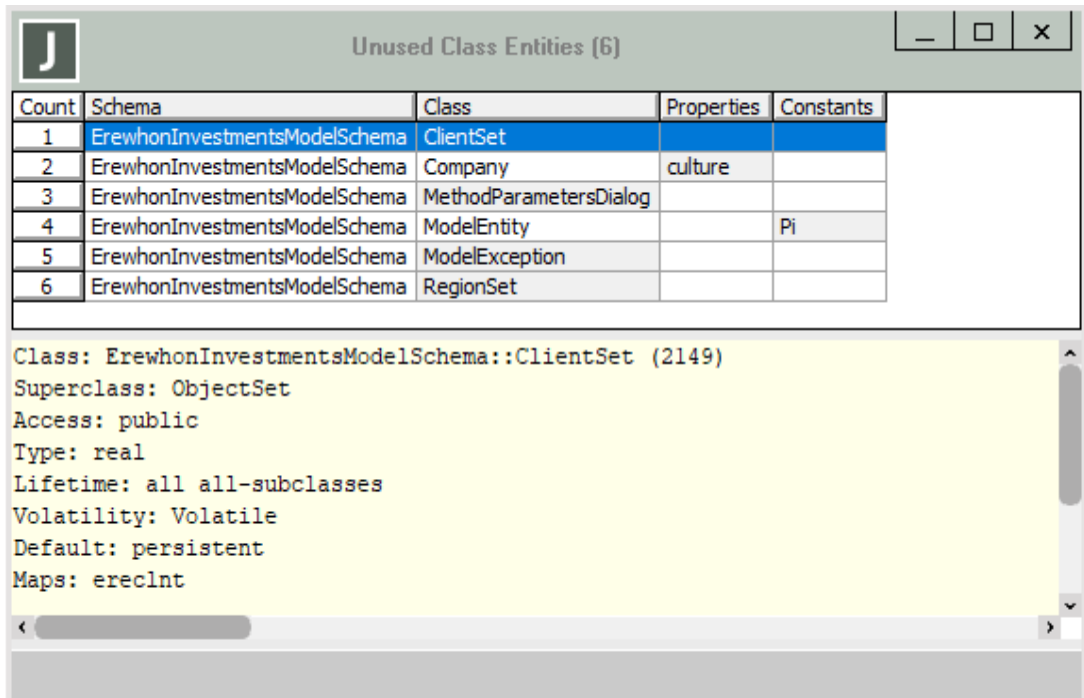
2. For each of the found entities, select the entity and then right-click it and select the **Open Browser** command.
3. A new Class Browser is opened, with the selected entity highlighted. From this Class Browser, right-click the entity and select the **Remove** command.



- In the Confirm Delete message box, click **OK**.



- The entity is then automatically removed from the Unused Class Entities list. Repeat these steps for each unused class entity until the list is empty.



Transient Leaks

A transient leak is when a transient object is created but not deleted after use. In large, long-running applications, transient leaks can have a significant impact on performance as the transient cache has finite capacity and once filled, transients must be saved to a transient database, which is much slower than the transient cache.

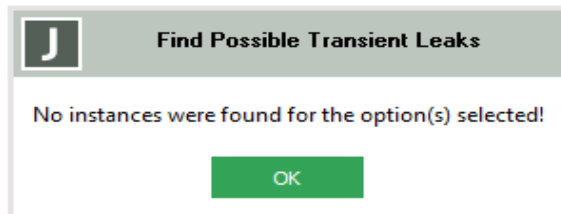
The JADE development environment provides the ability to identify methods in which a transient is created but not deleted and it is a useful tool to identify transient leaks; however, the identified transients may not necessarily be leaks if they are deleted elsewhere in the code.

When creating a transient object that you are confident will be deleted but that may generate a false positive for a transient leak, you can add **[ExcludeFromTransientLeakReport]** into an end-of-line comment after the **create** instruction, to ignore it. For example, the following JadeScript method will leak the transient **Company** named **leak**.

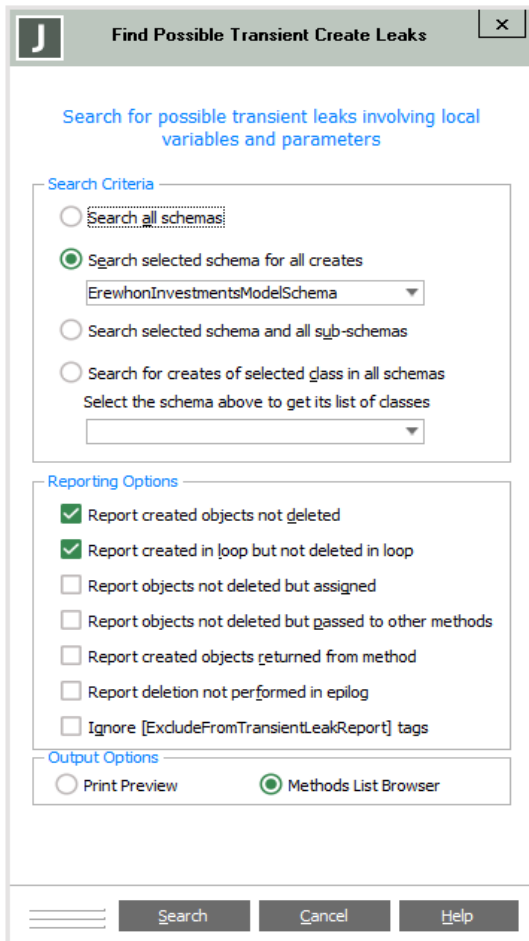
```
leakExample();

vars
    leak : Company;
begin
    create leak transient; // [ExcludeFromTransientLeakReport]
end;
```

However, by including **[ExcludeFromTransientLeakReport]**, the method will not appear in a transient leak search.



To open the Find Possible Transient Create Leaks dialog, select the **Find Possible Transient Leaks** command from the **Schema** menu.



From this dialog, you can specify the scope of the search by selecting one of the options in the Search Criteria group box. To search:

- All schemas in the database, select **Search all schemas**.
- Within a specific schema, select **Search selected schema** and then select the required schema from the associated combo box.
- Within a specific schema and any subschemas of that schema, select **Search selected schema and all sub-schemas** and then select the required schema from the associated combo box.

Note The **Search selected schema**, **Search selected schema and all sub-schemas**, and **Search selected class in selected schema** options share the schema selection combo box (that is, the first one in the Search Criteria group box).

- Within a specific class of all schemas, select **Search selected class in all schemas** and then select the class from the second combo box. The selected class is then searched for across all schemas in which it exists.

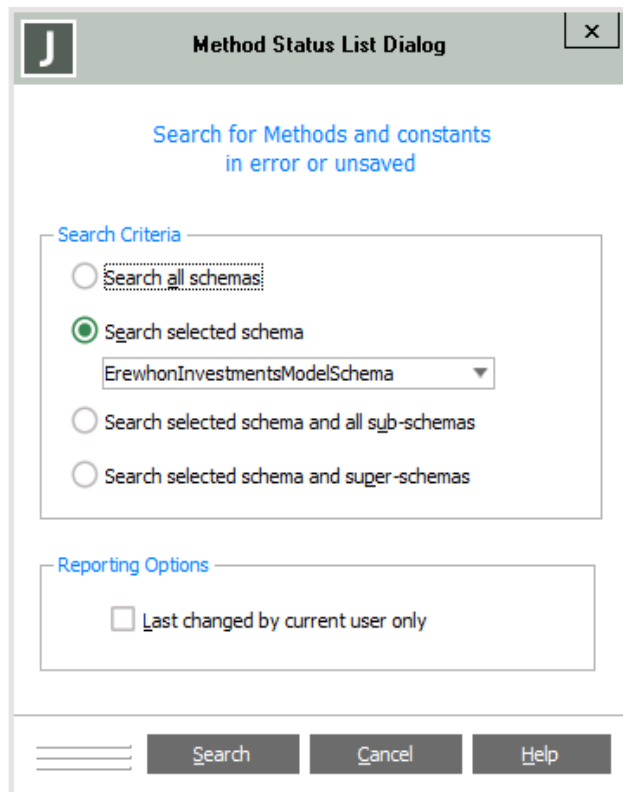
The Reporting Options group box is used to specify the possible leak conditions to be searched for, as follows.

- The **Report created objects not deleted** check box finds transients that are created but not deleted within the method, nor passed to another method, nor returned in a **return** instruction. These objects will usually result in leaks.
- The **Report created in loop but not deleted in loop** check box finds transients that are created in a loop but not deleted inside that loop, but may be deleted outside the loop.
- The **Report objects not deleted but assigned** check box finds transients that are created and not deleted, but are assigned to another property or variable that may or may not be deleted later.
- The **Report objects not deleted but passed to other methods** check box finds transients that are created but not deleted within the method, but are passed to another method as a parameter where they may or may not be deleted.
- The **Report created objects returned from method** check box finds transients that are created but not deleted within the method, but are returned in the **return** instruction of the method. These may or may not be leaks, depending on what the calling method does with the object.
- The **Report deletion not performed in epilog** check box finds transients that are not deleted in the epilog. Such objects may leak if an exception occurs in the method between when the object is created and when it is deleted, returned, or passed to another method.
- The **Ignore [ExcludeFromTransientLeakReport] tags** check box finds transients that could generate a leak according to the other options selected, even if they are marked with a **[ExcludeFromTransientLeakReport]** tag.

In the Output Options group box, the default **Methods List Browser** option button causes the results of the search to display in a methods list to make them easier to resolve. However, if you want to print them instead, select the **Print Preview** option button.

Methods Status List

The JADE development environment provides the ability to quickly find all methods that are unsaved or have errors in a specific schema or schemas. Open the Method Status List dialog by selecting the **Status List** command from the **Browse** menu or with the Ctrl+Shift+C shortcut keys.

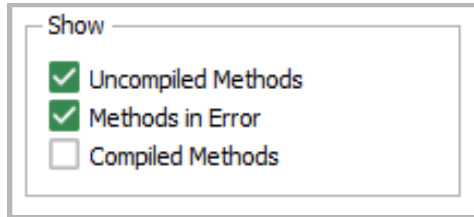


From this dialog, you can specify the scope of the search by selecting one of the options in the Search Criteria group box. To search:

- All schemas in the database, select **Search all schemas**.
- Within a specific schema, select **Search selected schema** and then select the required schema from the associated combo box.
- Within a specific schema and any subschemas of that schema, select **Search selected schema and all sub-schemas** and then select the required schema from the associated combo box.
- Within a specific schema and any superschemas of that schema, select **Search selected schema and all super-schemas** and then select the required schema from the associated combo box.

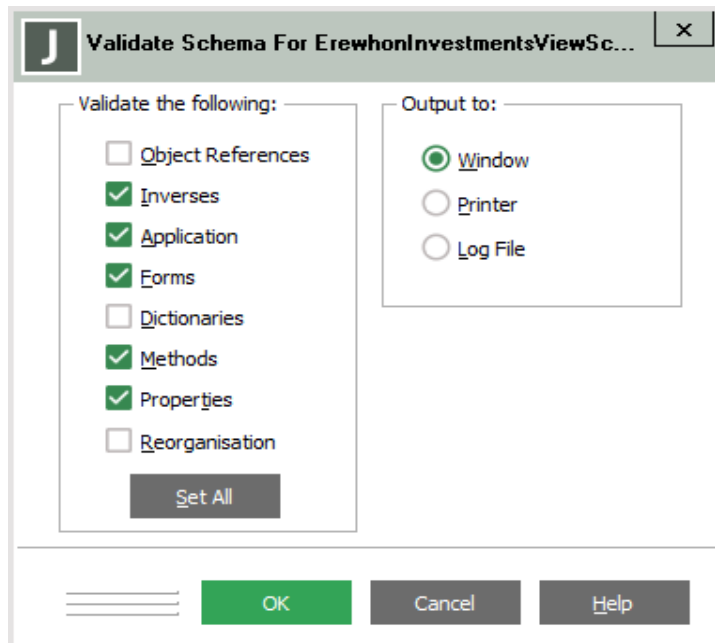
If you want to see only methods that were last changed by you, select the **Last changed by current user only** check box in the Reporting Options group box.

Once you have selected your required options, click **Search** to perform the search. By default, the Method Status List finds methods in error and uncompiled methods. However, this can be customized if needed, by selecting options in the Show group box on the **Status List** sheet of the Preferences dialog.



Schema Validation

The Validate Schema dialog provides validation for a variety of elements in the current schema. Access it by selecting the **Validate** command from the Schema list of the Schema Browser.



From the Validate the following group box, check the check boxes of the elements to be validated, as follows.

- The **Object References** option finds and lists any objects that are referenced but no longer exist. This should normally never occur, so these errors suggest that the schema is corrupted.
- The **Inverses** option finds any invalid inverse reference definitions.
- The **Application** option finds any invalid application definitions.
- The **Forms** option finds any invalid form definitions; for example, a form that contains a control with no corresponding property.
- The **Dictionaries** option lists any invalid dictionary definitions. This should normally never occur, so these errors suggest that the schema is corrupted.
- The **Methods** option outputs the same list as that generated by the Method Status List dialog.
- The **Properties** option outputs a list of all properties in a class that have duplicate feature numbers.

- The **Reorganisation** option validates and reports on the following, both of which suggest corruption of the schema.
 - If a schema has been marked as versioned but there are no classes versioned.
 - If there are classes versioned but the schema has not been marked as versioned.

The Validate Schema dialog provides the following output options that are selected from the Output to group box.

- **Window** (the default), which displays the results of the validation to the Jade Interpreter Output Viewer.

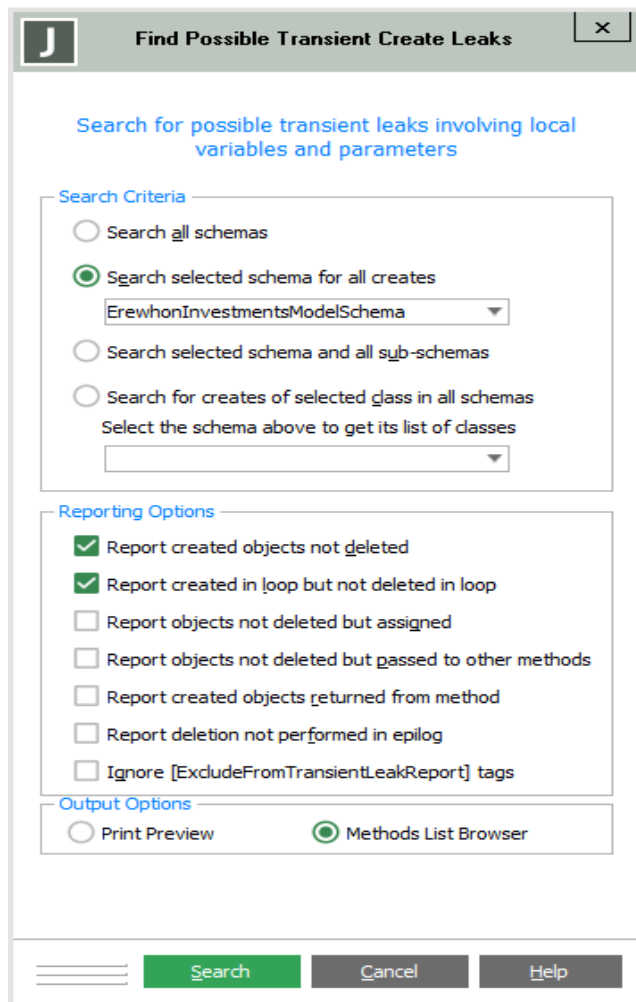


- **Printer**, which saves the results to a PDF file called **Jade.pdf** in your **Documents** directory of your JADE installation.
- **Log File**, which saves the results to the **valscm.log** file in the **bin** directory of your JADE installation.

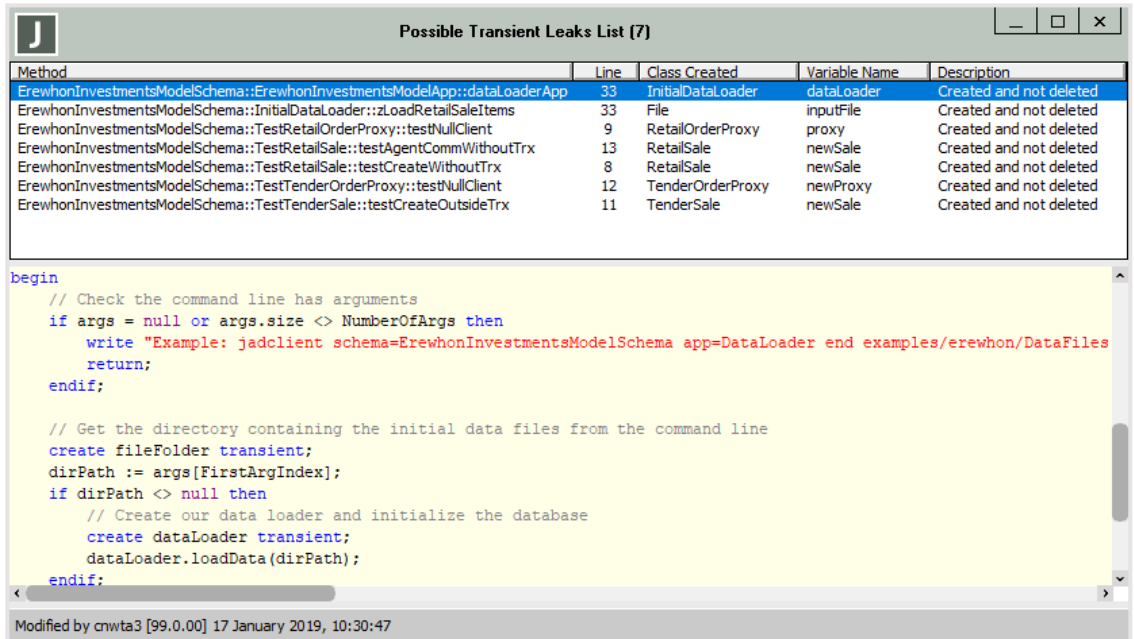
Exercise 8 – Finding and Removing Transient Leaks

In this exercise, you will locate and remove transient leaks from **ErewhonInvestmentsModelSchema**, using the **Find Possible Transient Leaks** command.

1. With **ErewhonInvestmentsModelSchema** selected in the Schema Browser, select the **Find Possible Transient Leaks** command from the Schema menu.



- Click **Search**, to display the list of possible transient leaks.



You should see that the following methods have potential transient leaks.

- ErewhonInvestmentsModelApp::dataLoaderApp
 - InitialDataLoader::zLoadRetailSaleItems
 - TestRetailOrderProxy::testNullClient
 - TestRetailSale::testAgentCommWithoutTrx
 - TestRetailSale::testCreateWithoutTrx
 - TestTenderOrderProxy::testNullClient
 - TestTenderSale::testCreateOutsideTrx
- For each method, delete the created transient in the epilog section of the method. For example, for the **ErewhonInvestmentsModelApp** class **dataLoaderApp** method, the epilog section should be as follows.

```

epilog
  delete fileFolder; // does nothing if fileFolder is null
  delete dataLoader;
  terminate;
end;
    
```

- When you have added the **delete** instructions to the epilog, use the **Find Possible Transient Leaks** command again, to verify that the transient leaks have been resolved.

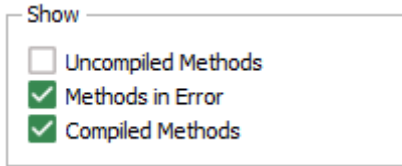
Exercise 9 – Finding Changed Methods

In this exercise, you will explore the use of the Method Status List dialog.

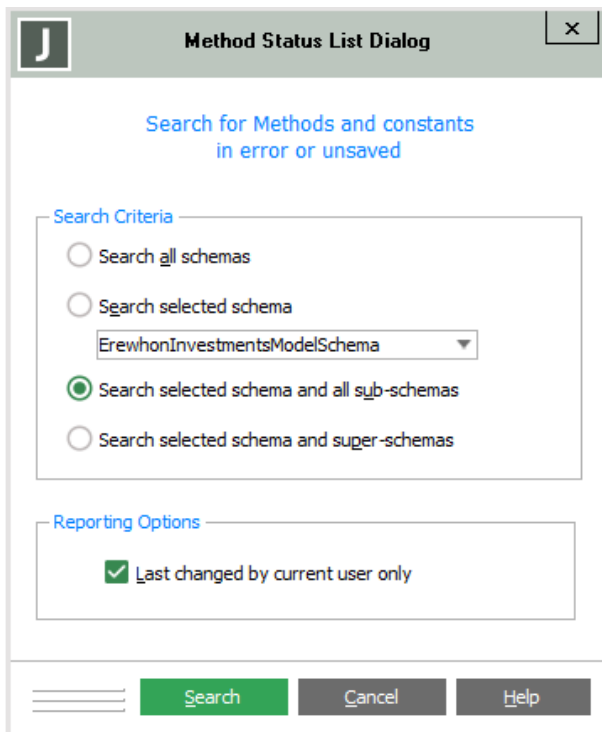
You will first use the Method Status List dialog to find all methods that you have changed throughout this module.

You will then use the Method Status List Browser to find and resolve compiler errors by first intentionally breaking some methods and then using the Method Status List Browser to repair the changes.

1. In the first exercise in this module, you configured some user preferences. Open the Preferences dialog and ensure that the **Status List** sheet has the following settings in the Show group box.

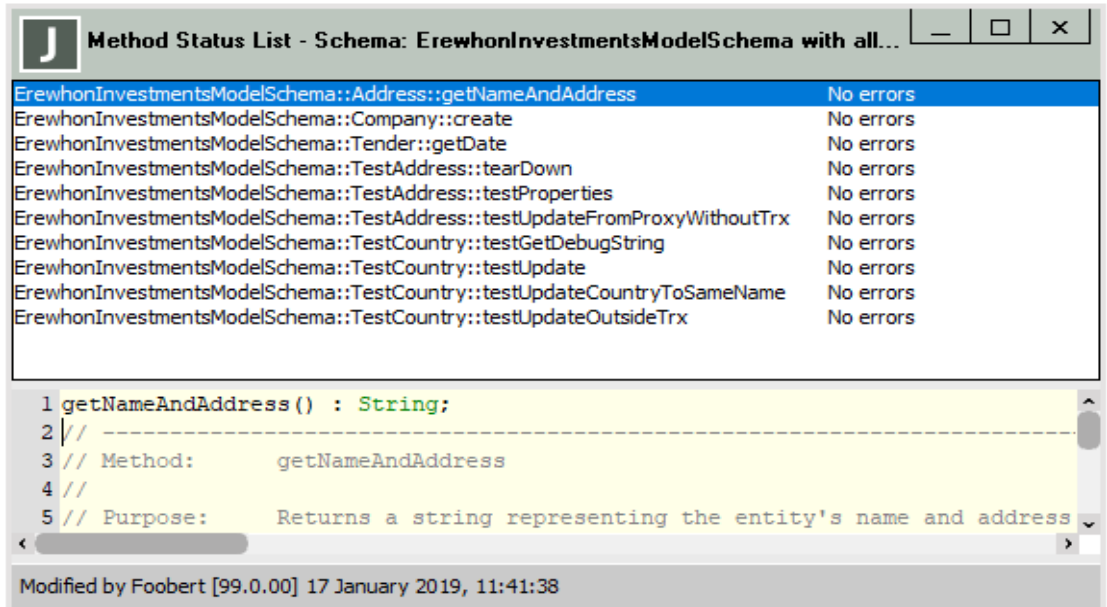


2. Select **ErewhonInvestmentsModelSchema** in the Schema Browser and then open the Method Status List dialog using the Ctrl+Shift+C shortcut keys.
3. In the Search Criteria group box, select **Search selected schema and all sub-schemas** option button and in the Reporting Options group box, check the **Last changed by current user only** check box.



4. Click **Search**.

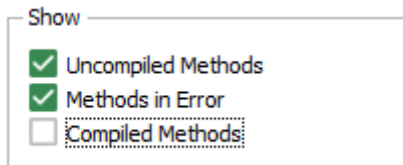
The Method Status List Browser then displays a list of all methods that you have changed.



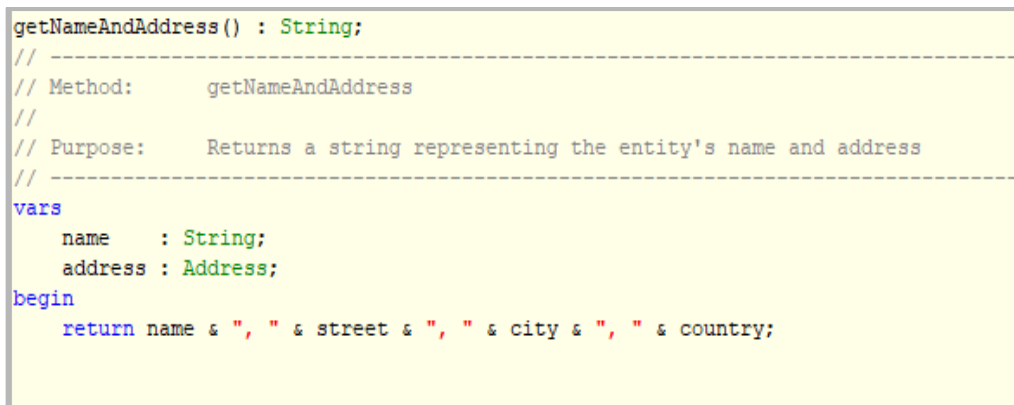
Note Your list will show different methods to this example.

You should see that none of the methods are in error. You will now intentionally break some methods so that the Method Status List has something to find.

- Open the Preferences dialog and set the following preferences on the **Status List** sheet.



- Navigate to the Address class **getNameAndAddress** method, which is a subclass of **ModelEntity**.
- Delete the **end;** instruction on line 12, as follows.



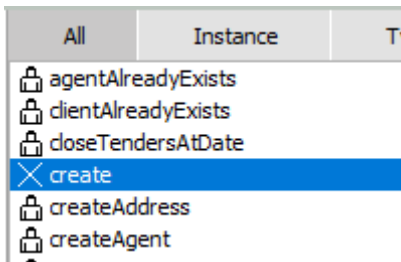
- Compile the method (press F8). Compiler error **7107 - Expecting: end** will be raised.
- Navigate to the **Tender** class **getDate** method, which is also a subclass of **ModelEntity**.

- Comment out the return statement on line 8 by adding // to the beginning of the line, as follows.

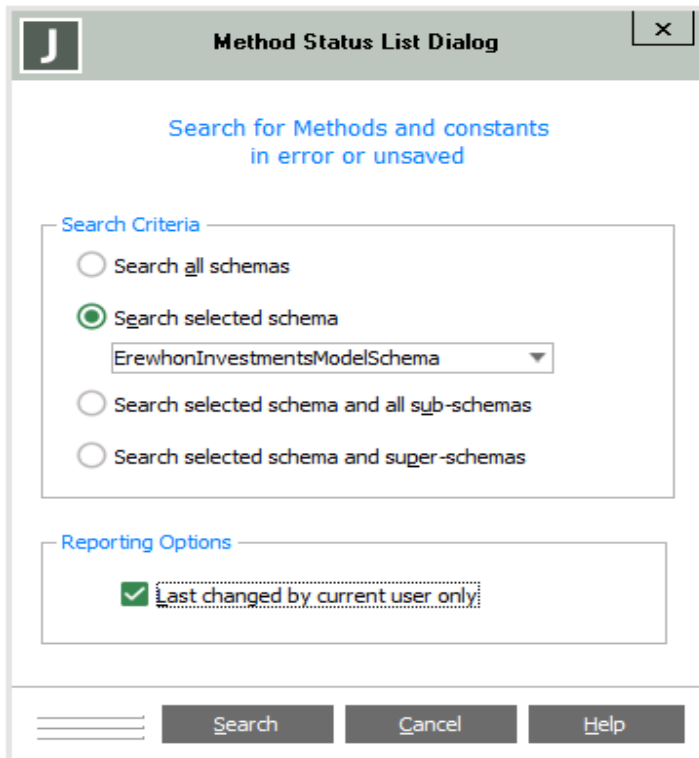
```

getDate() : Date;
// -----
// Method:      getDate
//
// Returns:     The offer date of the tender
// -----
begin
// return timeStamp.date;
end;
    
```

- Compile the method. Compiler error **6114 - Method does not return a value** will be raised.
- Navigate to the **Company** class **create** method.
- Delete the blank line on line 11 (immediately after the **begin** instruction).
- Press F2 to save the method without compiling it. You should see that it now has an **X** to the left of the method name in the Class Browser.



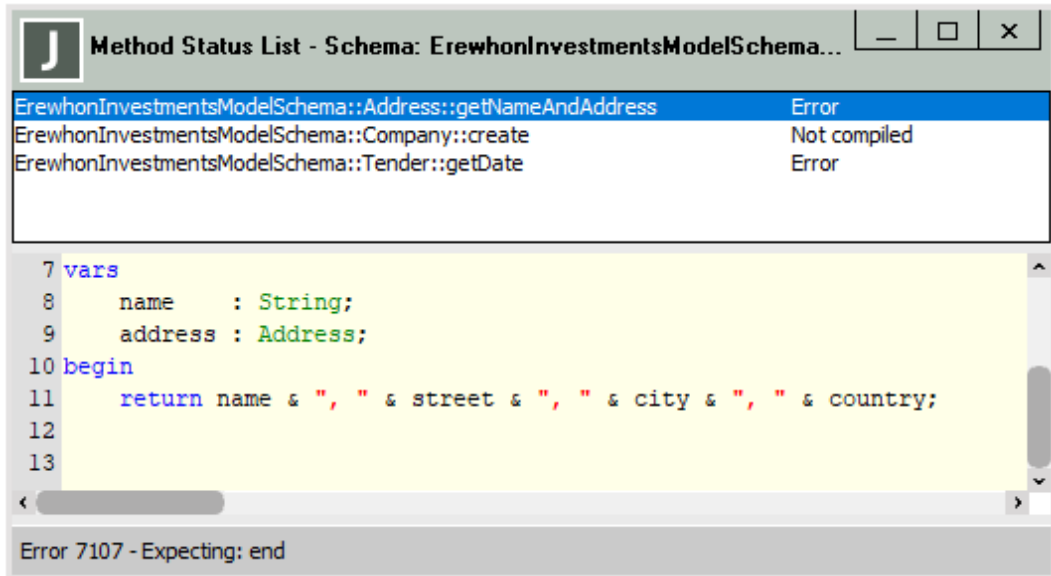
- Select **ErewhonInvestmentsModelSchema** in the Schema Browser and then press Ctrl+Shift+C, to open the Method Status List dialog.



16. Check the **Last changed by current user only** check box and then click **Search**.

Tip In a single user system, the **Last changed by current user only** has no effect, as there is only one user changing methods. However, when working collaboratively, it can be useful to see only those methods that *you* have changed.

The Method Status List Browser will then display the three methods you changed that have errors or are unsaved.



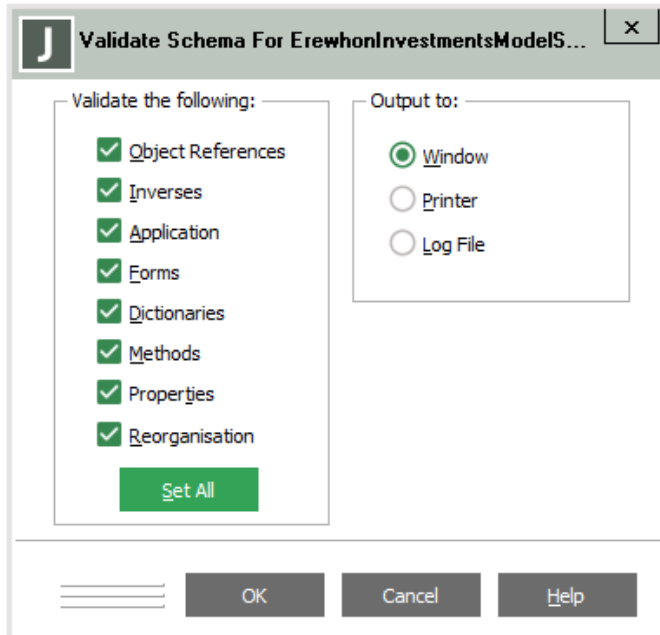
17. For each method, fix the error and then compile the method.

You can do this directly from the Method Status List Browser, and each method will automatically be removed from the list as it is completed.

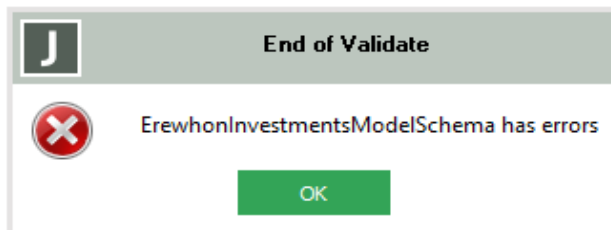
Exercise 10 – Validating the Schemas

In this exercise, you will use the schema validator to find inconsistencies in the **ErewhonInvestmentsModelSchema** and then resolve these inconsistencies.

1. With **ErewhonInvestmentsModelSchema** selected in the Schema Browser, select the **Validate** command from the Schema menu.



2. Click **Set All** to select all possible validations and then click **OK** to run the validator.
3. The following message box should be displayed. Click **OK**.



4. The results of the validation are written to the Jade Interpreter Output Viewer. The following errors should be identified.

```

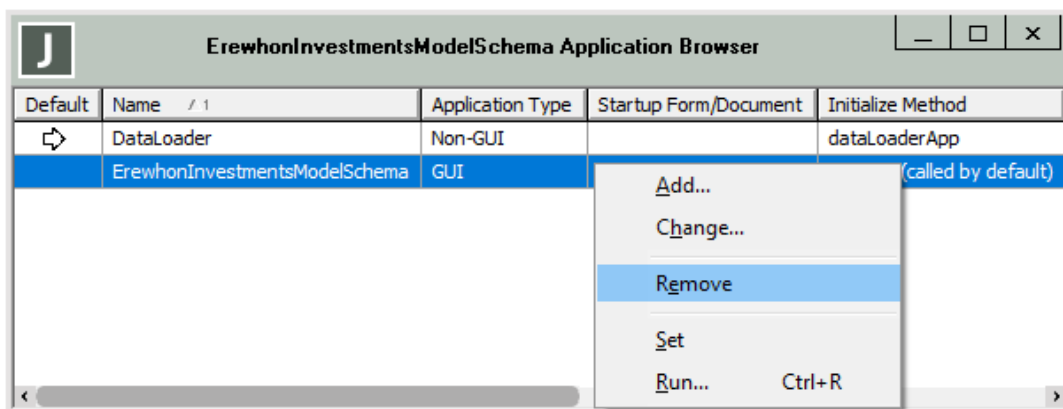
=====
Validating Applications
=====
Application ErewhonInvestmentsModelSchema does not have a startup form
    
```

```

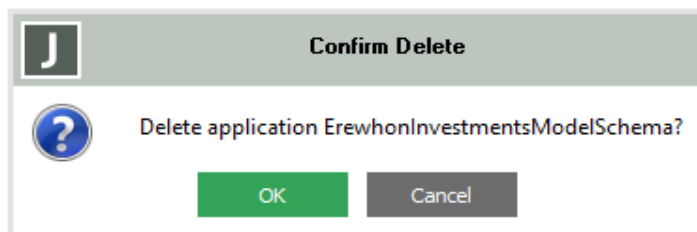
=====
Validating Dictionaries
=====
Key path does not have inverse SaleItem::codePrefix - OrderProxy::mySaleItem.SaleItem::codePrefix
Key path does not have inverse SaleItem::codeNumber - OrderProxy::mySaleItem.SaleItem::codeNumber
    
```

5. Select **ErewhonInvestmentsModelSchema** in the Schema Browser and then use the **Ctrl+L** shortcut keys to open the Application Browser.

6. Select the **ErewhonInvestmentsModelSchema** application in the Application Browser, right-click on it, and then select the **Remove** command.



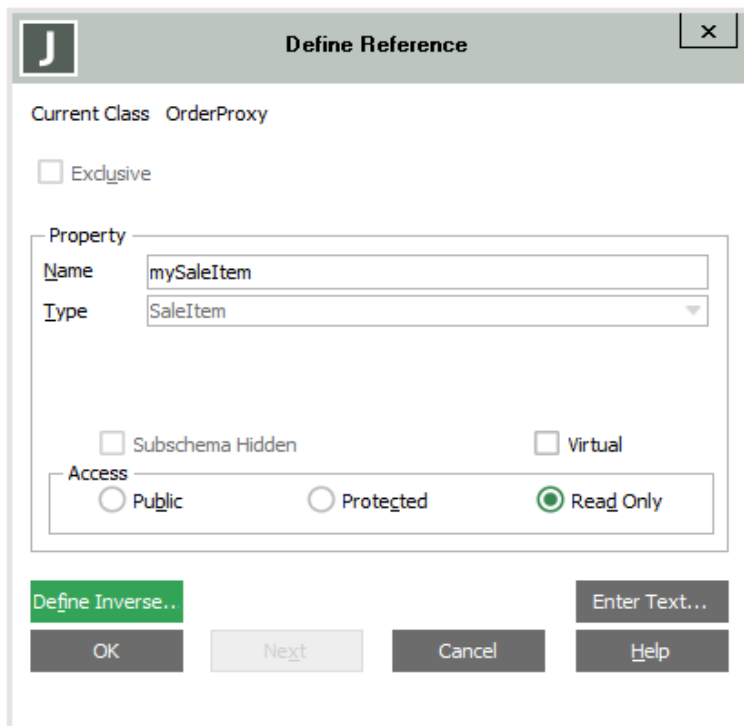
7. Click **OK** in the Confirm Delete message box.



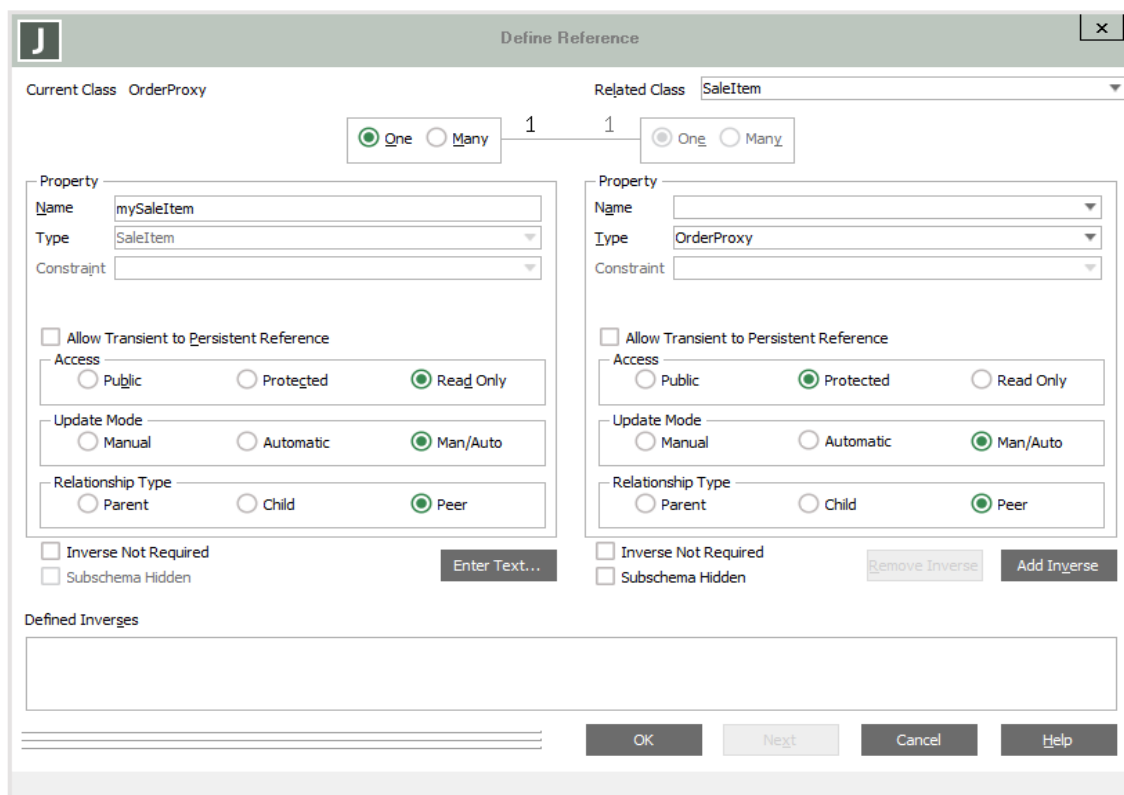
Note The **ErewhonInvestmentsModelSchema** application is the default application that was automatically generated when the **ErewhonInvestmentsModelSchema** schema was created. As it is not used for anything, it can be deleted.

8. Open the **ErewhonInvestmentsModelSchema** schema in a Class Browser. (Press Ctrl+B from the Schema Browser.)
9. Navigate to the **OrderProxy** class (which is a subclass of **ModelTransient**).

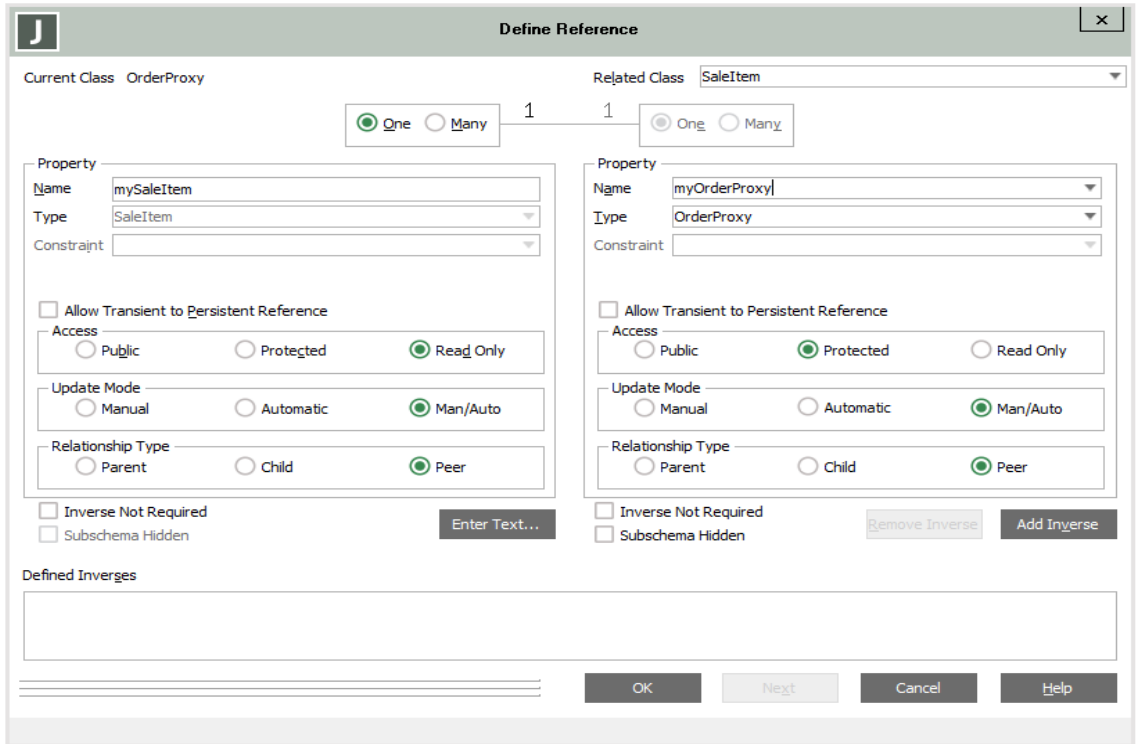
- Right-click on the **mySaleItem** reference and then select the **Change** command.



- Click **Define Inverse**.



12. To the **SaleItem** class (in the Property group box at the right of the dialog), set the **Name** value to **myOrderProxy**.



13. Click **OK**.
14. Re-run the Validate Schema command on **ErewhonInvestmentsModelSchema**.
You should see that there are no longer any errors.