



# Métodos Formais em Engenharia de Software

## VDMTool Tutorial

Ana Paiva

[apaiva@fe.up.pt](mailto:apaiva@fe.up.pt)   [www.fe.up.pt/~apaiva](http://www.fe.up.pt/~apaiva)



Universidade do Porto  
Faculdade de Engenharia  
**FEUP**

# Agenda

- ◆ Install
- ◆ Start
- ◆ Create a project
- ◆ Write a specification
- ◆ Add a file to a project
- ◆ Check syntax and types
- ◆ Check for errors
- ◆ Interpreter
- ◆ Check invariants, pre and post conditions
- ◆ Test cases
- ◆ Test case coverage analysis
- ◆ Build a report



# Install

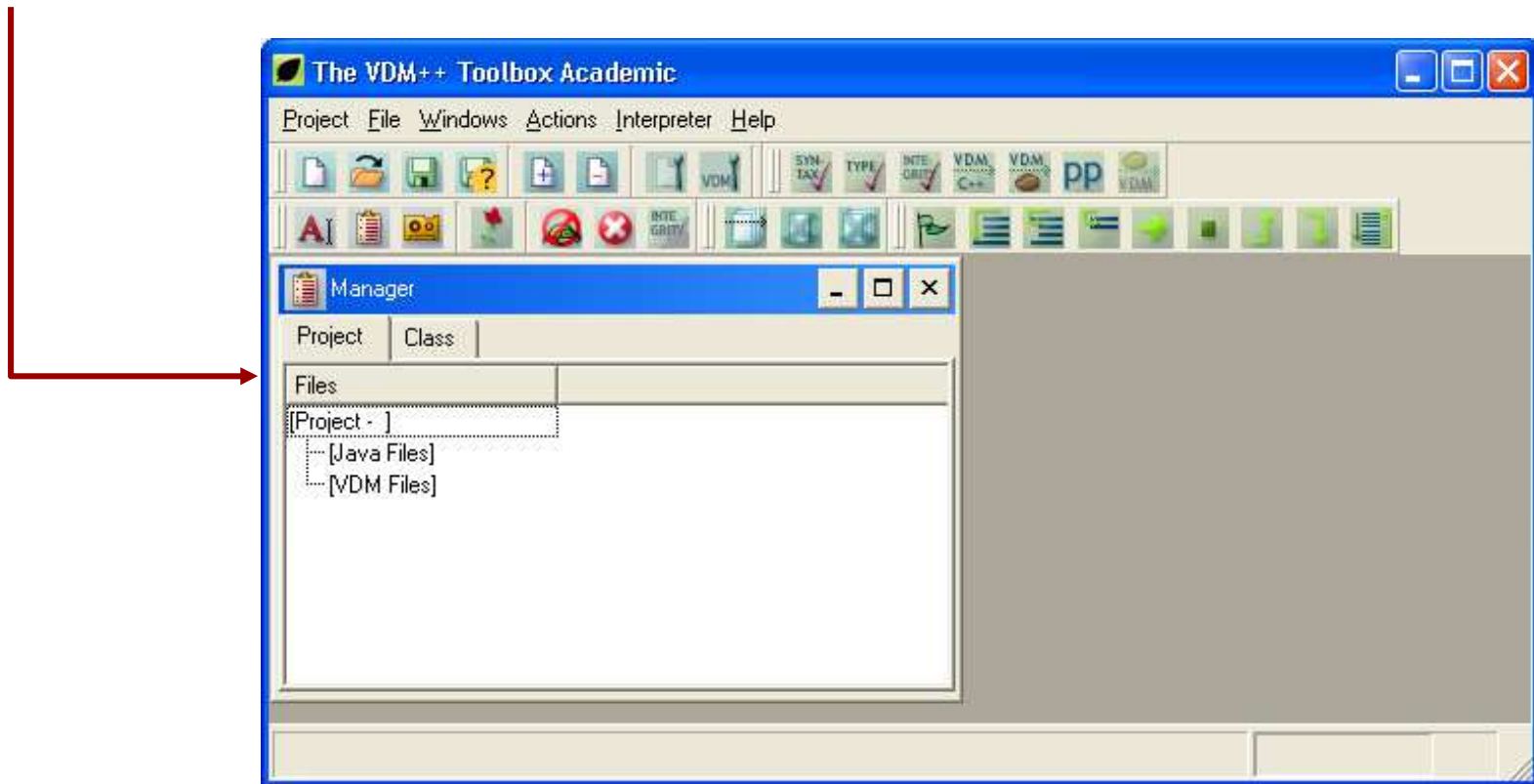
- ◆ VDM++ Toolbox Academic v8.1
  
- ◆ *Download at*  
<http://www.vdmtools.jp/en/index.php>
  
- ◆ Install with administrator privileges
  - Download and run *setup*

# Start

Start -> VDMTools -> VDM++ Toolbox Academic v8.1

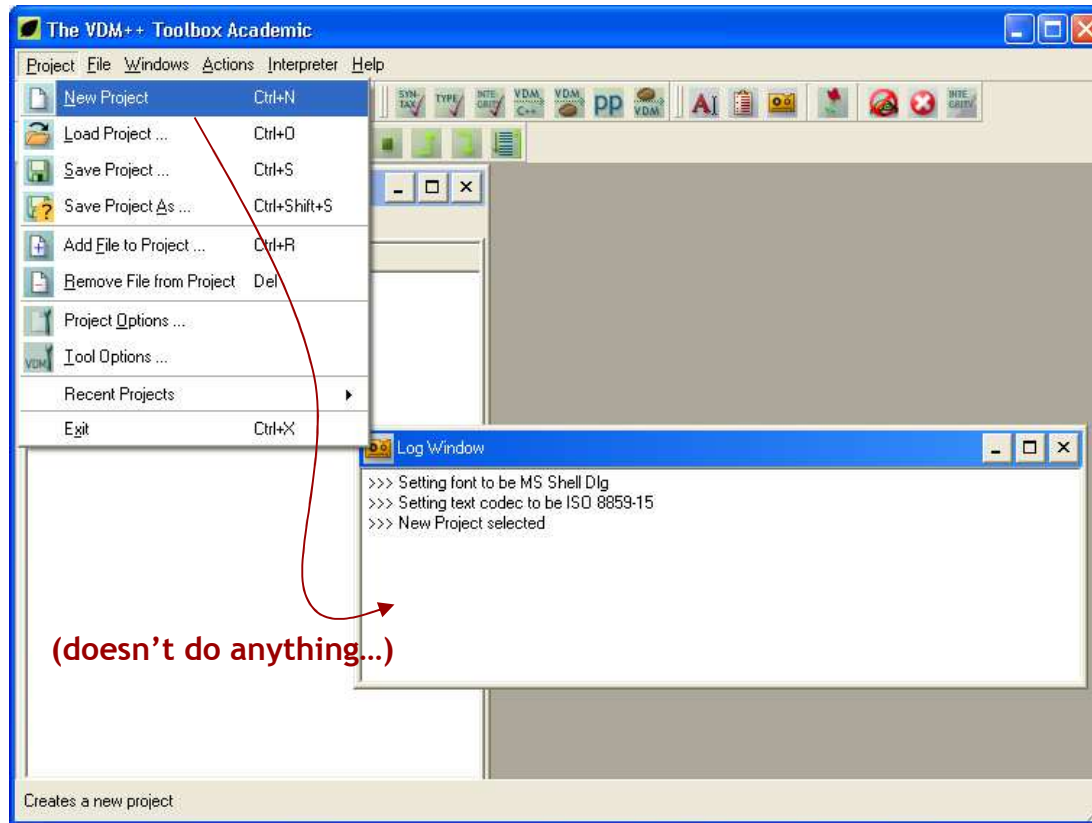
OU

...\The VDM++ Toolbox v9.0.2\bin\vpapgde.exe



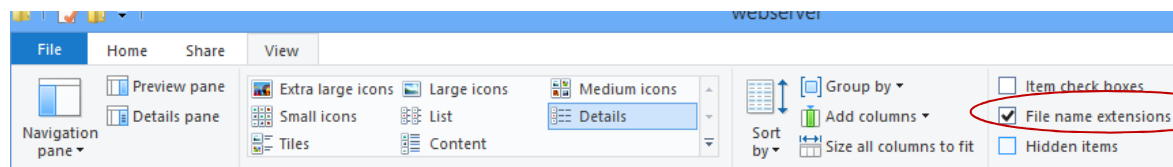
# Create a projet (1/1)

1°

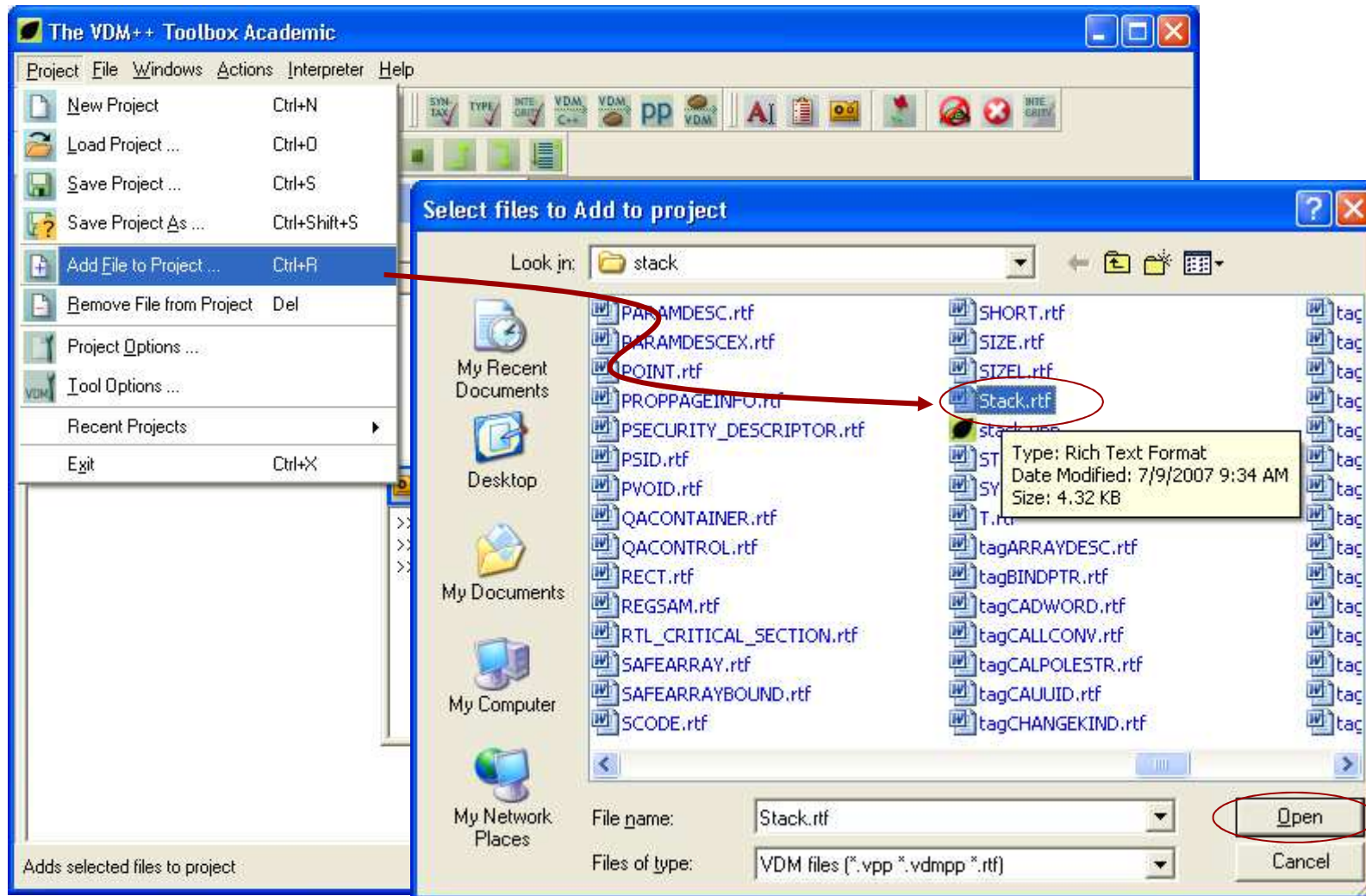


2°

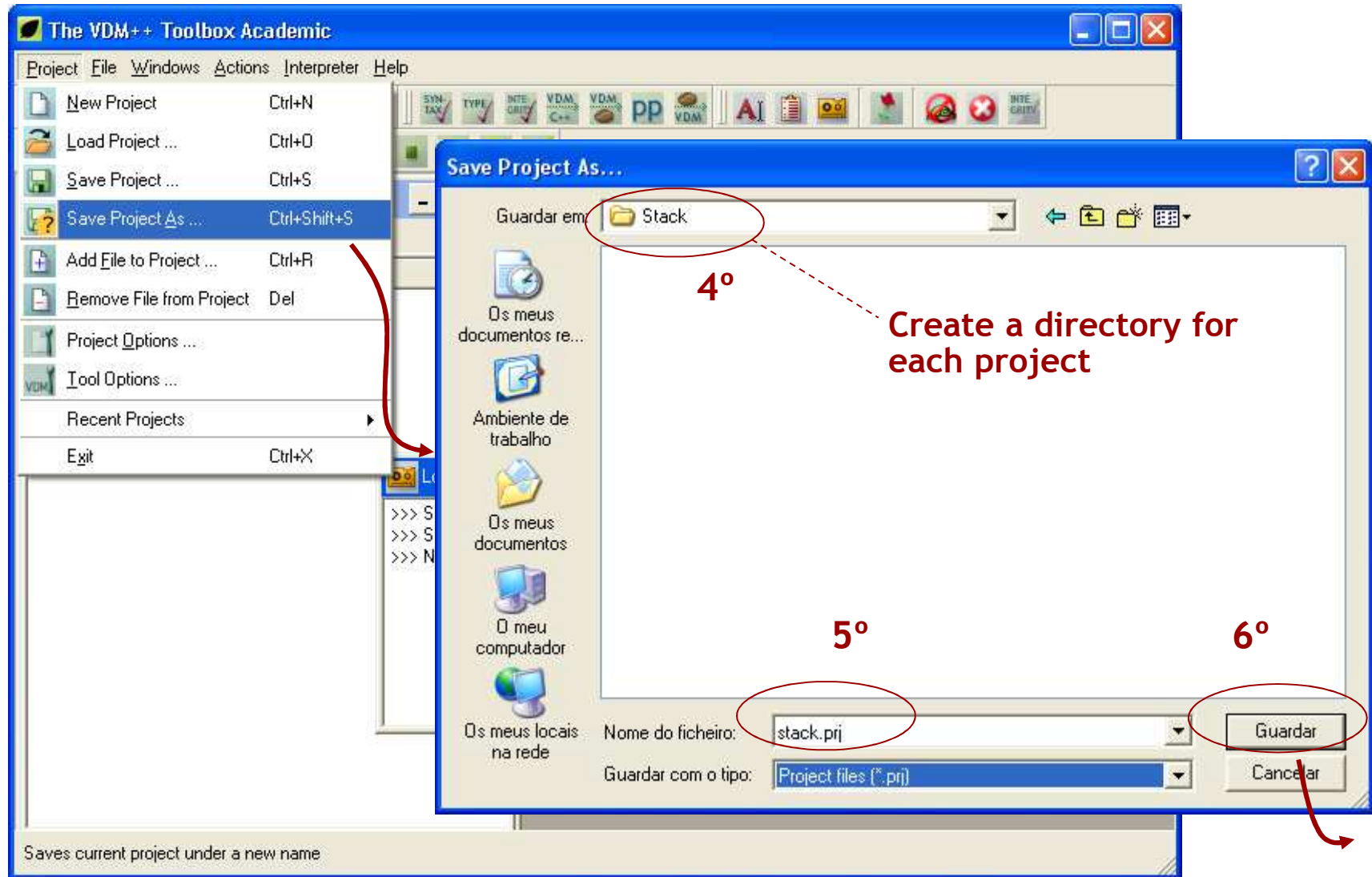
Show “File name extension”



# Add a file to a project (1/2)



# Save a project (1/3)





# Write a specification (2/1)

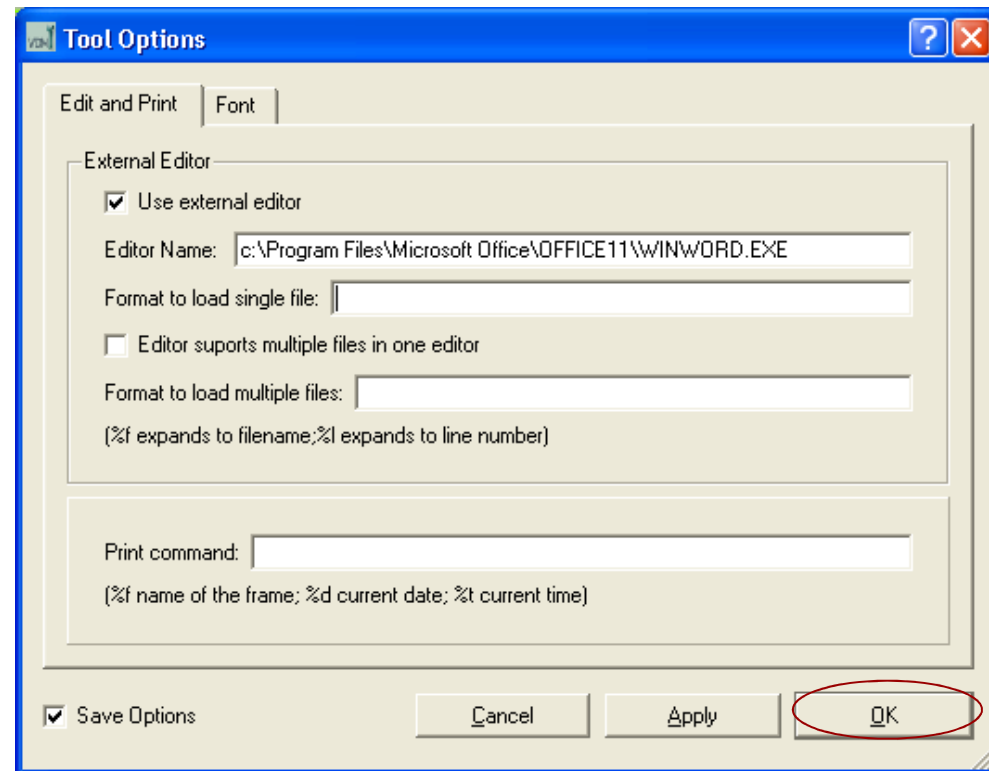
- ◆ Literal programming (code and documentation together)
- ◆ VDM++ code is written in one RTF (Rich Text Format) document, using styles predefined on the template “The VDM++ Toolbox Academic v9.0.2\word\VDM.dot”
- ◆ Open file “VDM.dot” and save it in RTF format, with the name of the class to create, for example “FStack.rtf”, in the project directory
- ◆ Usually, it is good practice to create a directory for each class
- ◆ The file can have VDM++ code and documentation
- ◆ Format the VDM++ code with “VDM” style



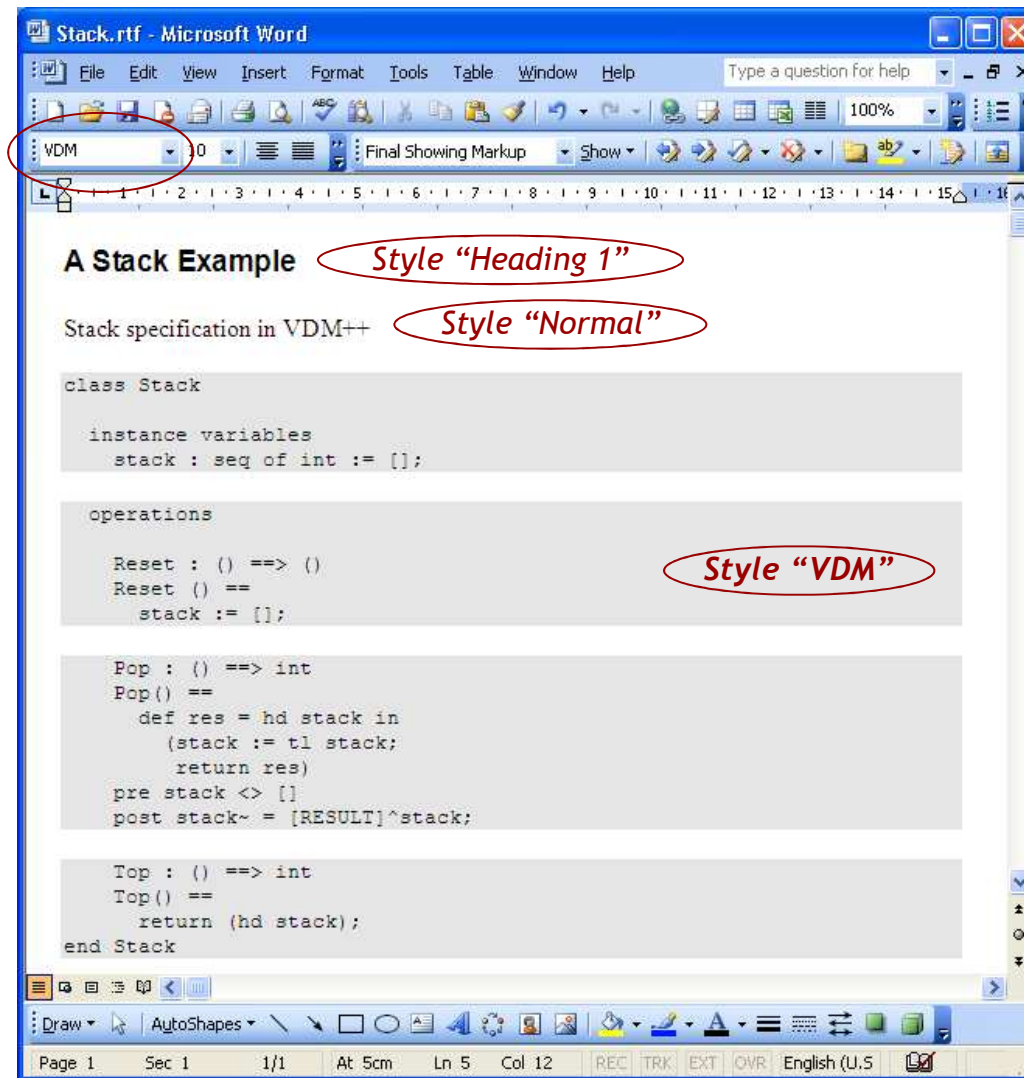


# Write a specification (2/2)

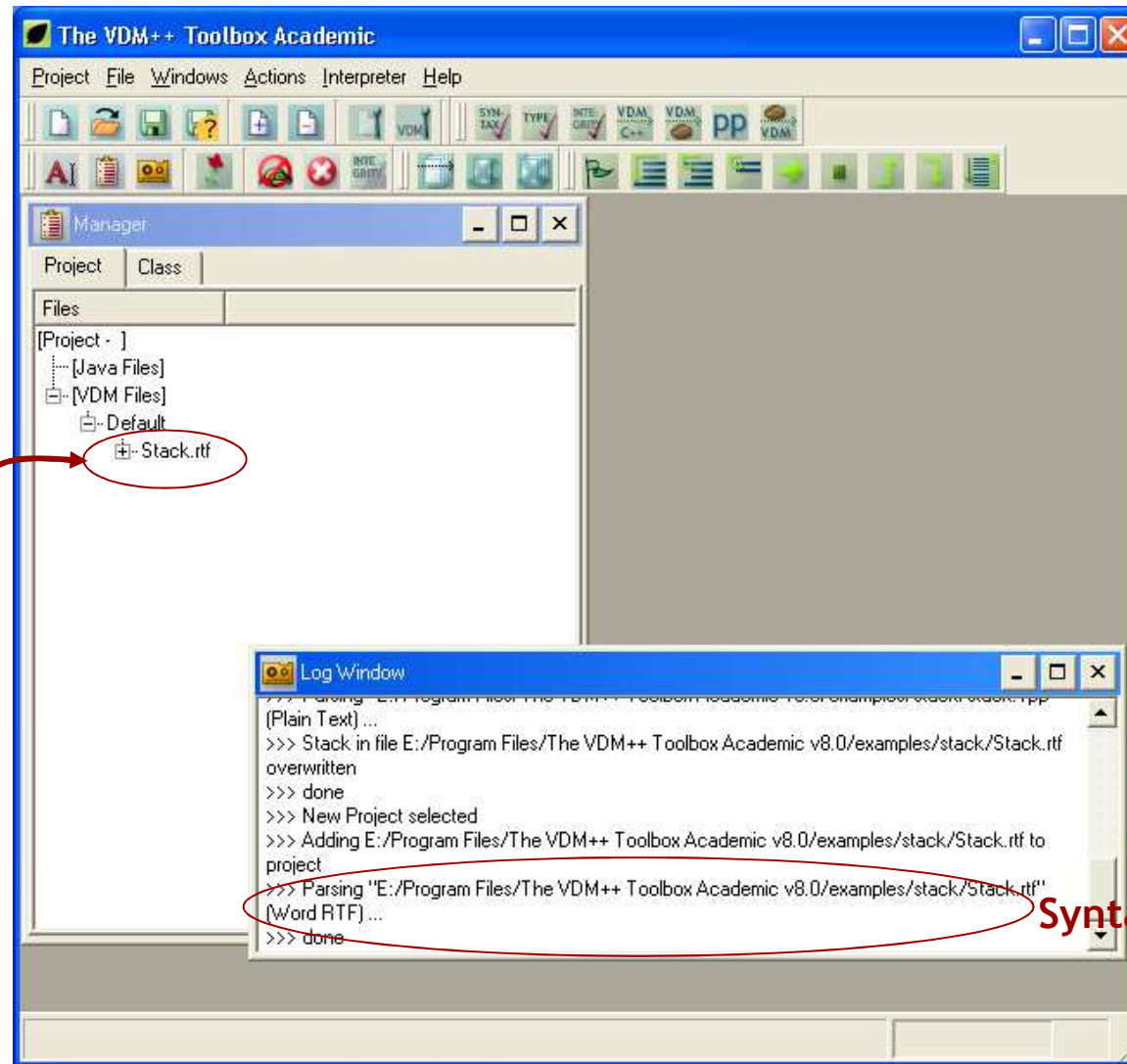
- ◆ To open Microsoft Word automatically choose option File->External Editor:
  - Project -> Tool Options



# Write a specification (2/3)



# Add a file to a project (2/4)



Syntax Ok

# Check syntax and types

The screenshot shows the 'The VDM++ Toolbox Academic' application window. The menu bar includes Project, File, Windows, Actions, Interpreter, and Help. The toolbar contains icons for various actions, with 'SYNTAX' and 'TYPE' icons circled in red. The 'Manager' panel shows a 'Project' view with 'Class' selected, and a 'Stack' project. Below this, 'Syntax' and 'Type' buttons are circled in green, with 'Sintaxe Ok' and 'Tipos Ok' displayed below them. A 'Log Window' at the bottom shows the following output:

```
>>> Stack in the Editor program file: E:\Program Files\The VDM++ Toolbox Academic v8.0\examples\stack\Stack.rtf overwritten  
>>> done  
>>> New Project selected  
>>> Adding E:\Program Files\The VDM++ Toolbox Academic v8.0\examples\stack\Stack.rtf to project  
>>> Parsing "E:\Program Files\The VDM++ Toolbox Academic v8.0\examples\stack\Stack.rtf" [Word RTF] ...  
>>> done  
>>> Type checking Stack ...  
>>> done
```

Annotations in red text indicate the steps: '2° select' points to the 'Stack' project; '1° - see the classes' points to the 'Class' button; '3° - check syntax' points to the 'SYNTAX' icon; and '4° - check types' points to the 'TYPE' icon. Green text annotations include 'Sintaxe Ok' and 'Tipos Ok' near the buttons, and 'Syntax Ok' and 'Types Ok' near the log window output.

# Errors

The screenshot shows the VDM++ IDE interface. The main window displays the source code for 'FStack.rtf'. The error list window is open, showing two errors. The first error is highlighted, and its description is shown in a separate window. The source code window shows the corresponding code with a red circle around the tilde symbol in the assignment statement.

```
1:
2: --LINE 6 1
3: class FStack
4:   instance variables
5:     private ele
6:     private cap
7:     inv len ele
8:   operations
9:     public FSt
10:    FS
11:
12: --LINE 8 1
13:    pc
14:
15: --LINE 10 1
16:   public Push: int ==> ()
17:     Push(x) == elements := [x] ^ elements
18:     pre len elements < capacity
19:     post elements = [x] ^ elements ;
20:   public Pop : () ==> ()
21:     Pop () == elements := tl elements
22:     pre elements <> []
23:     post elements = tl ~elements ;
24:   public Top : () ==> int
25:     Top () == return hd elements
26:     pre elements <> []
27:
```

Double click on the error

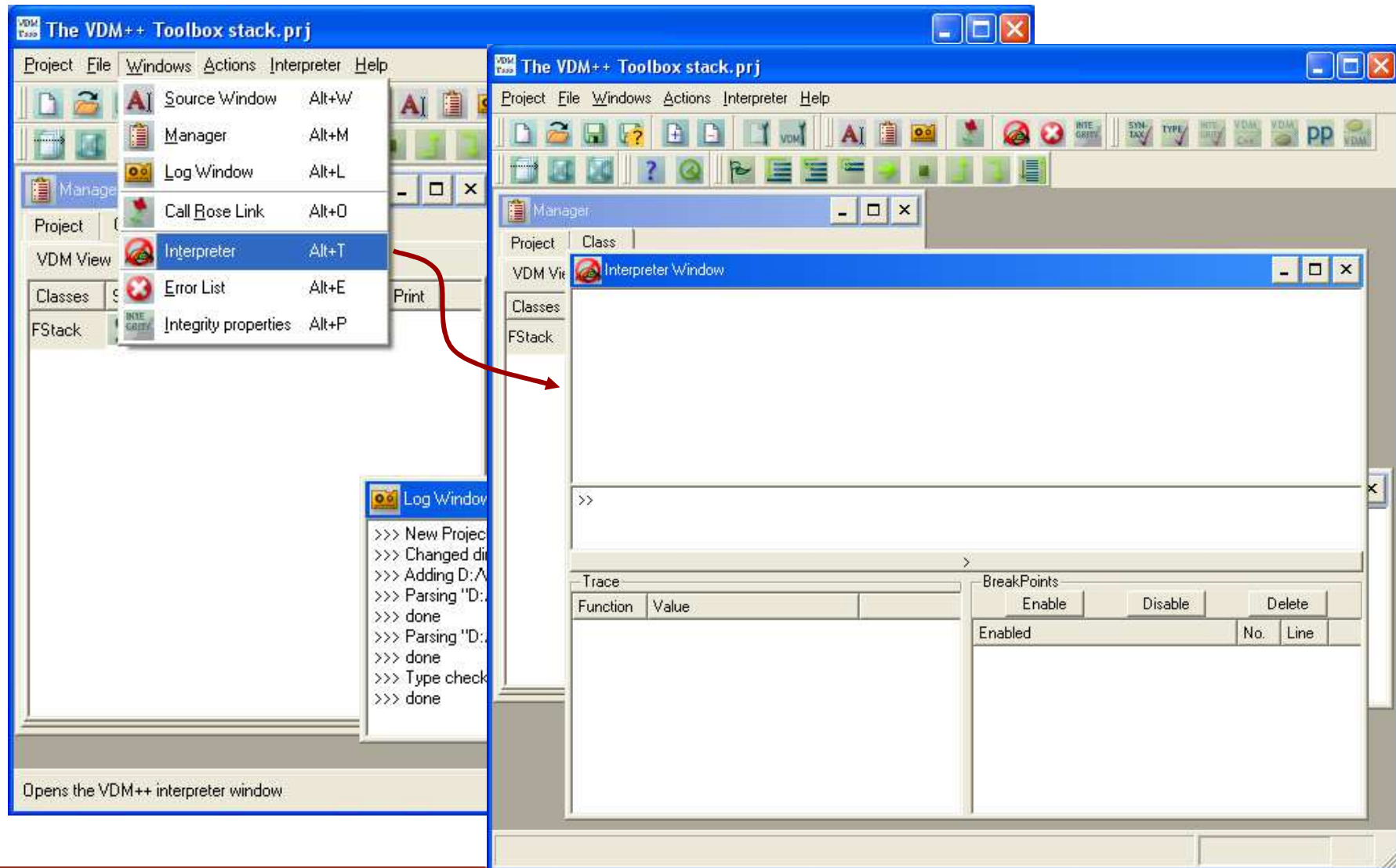
Description of the error

Locate the error

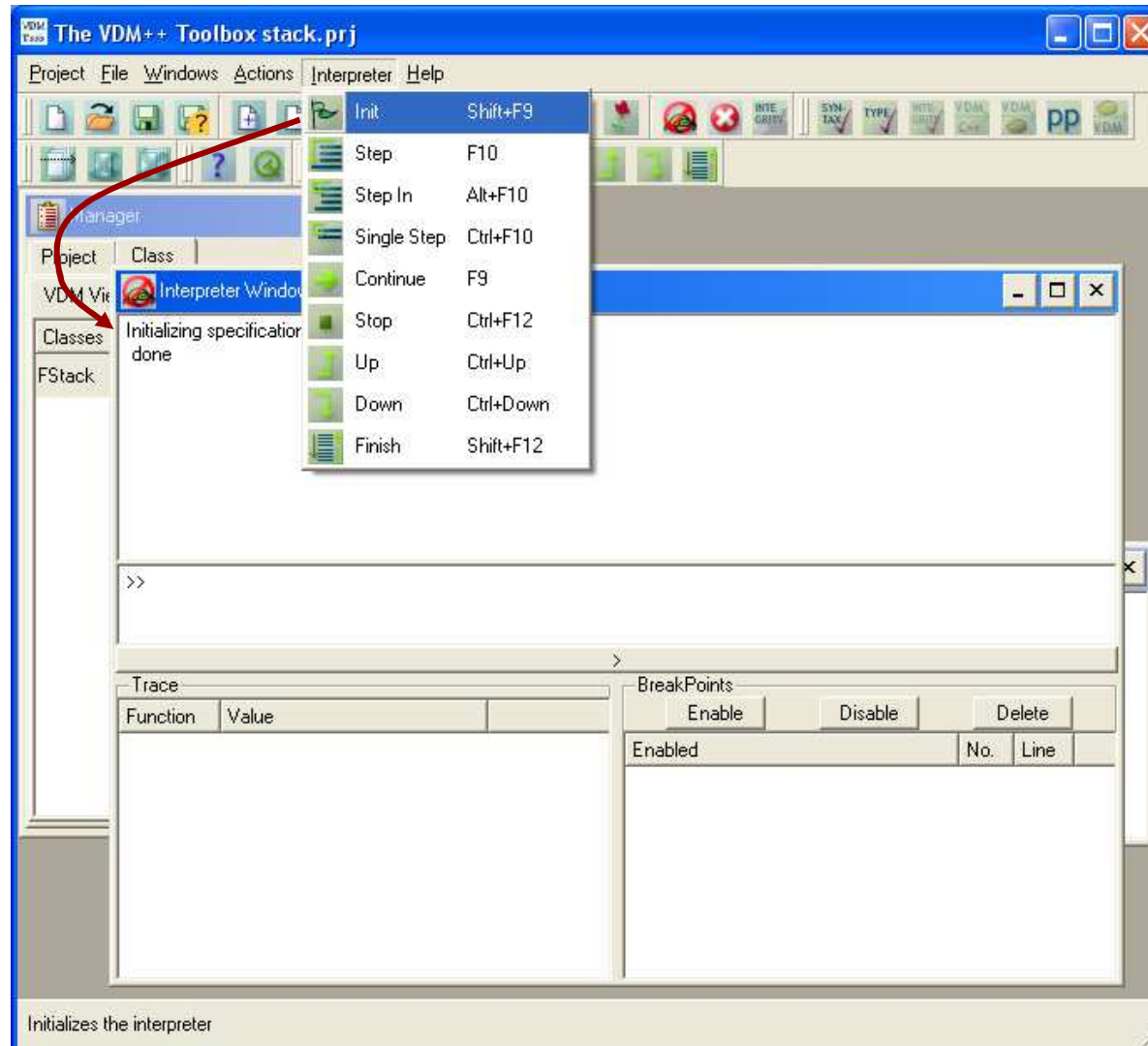
Analysis: the problem is that “~” should be located after the name of the variable and not before



# Run the interpreter - open

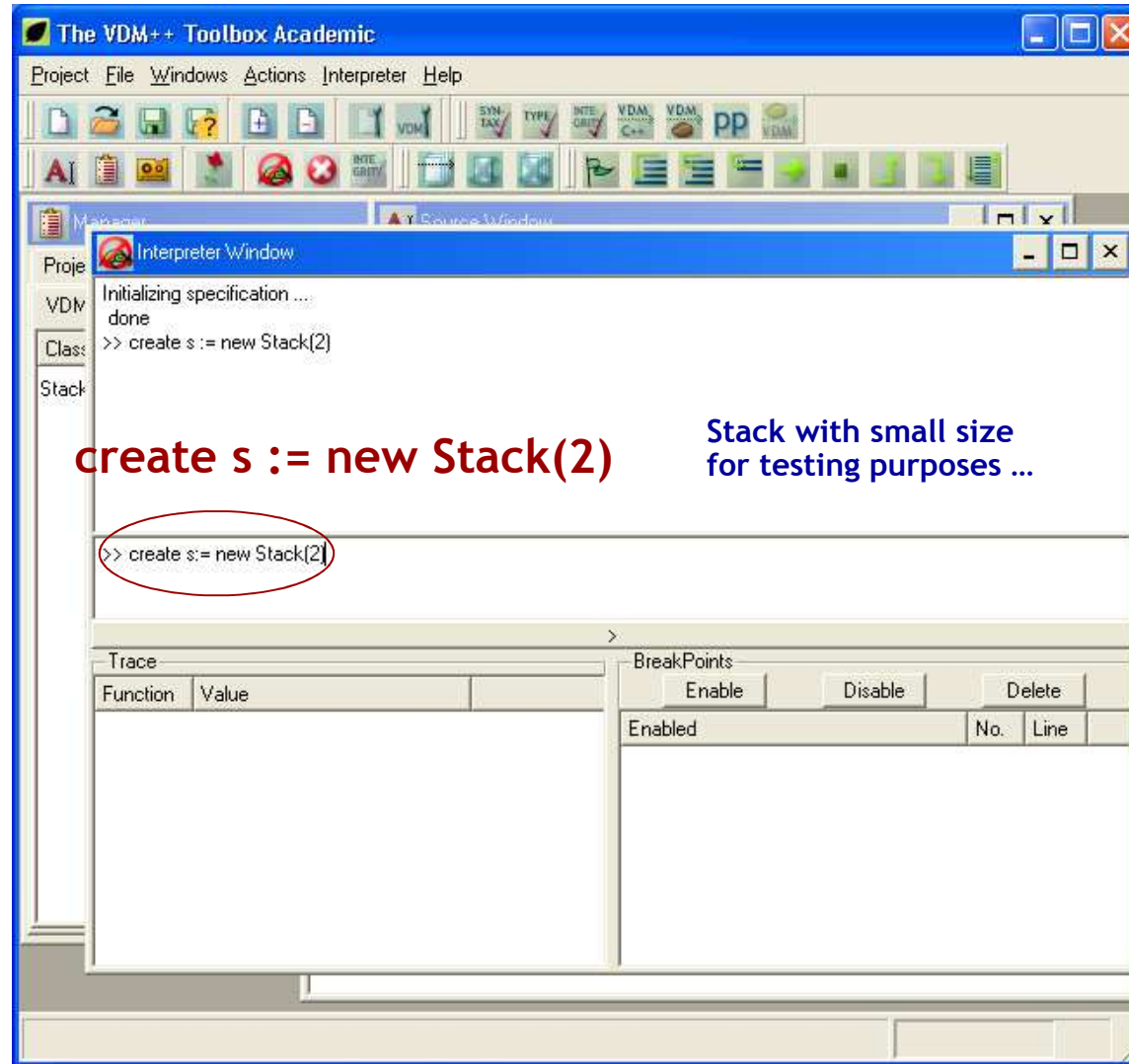


# Run the interpreter - start

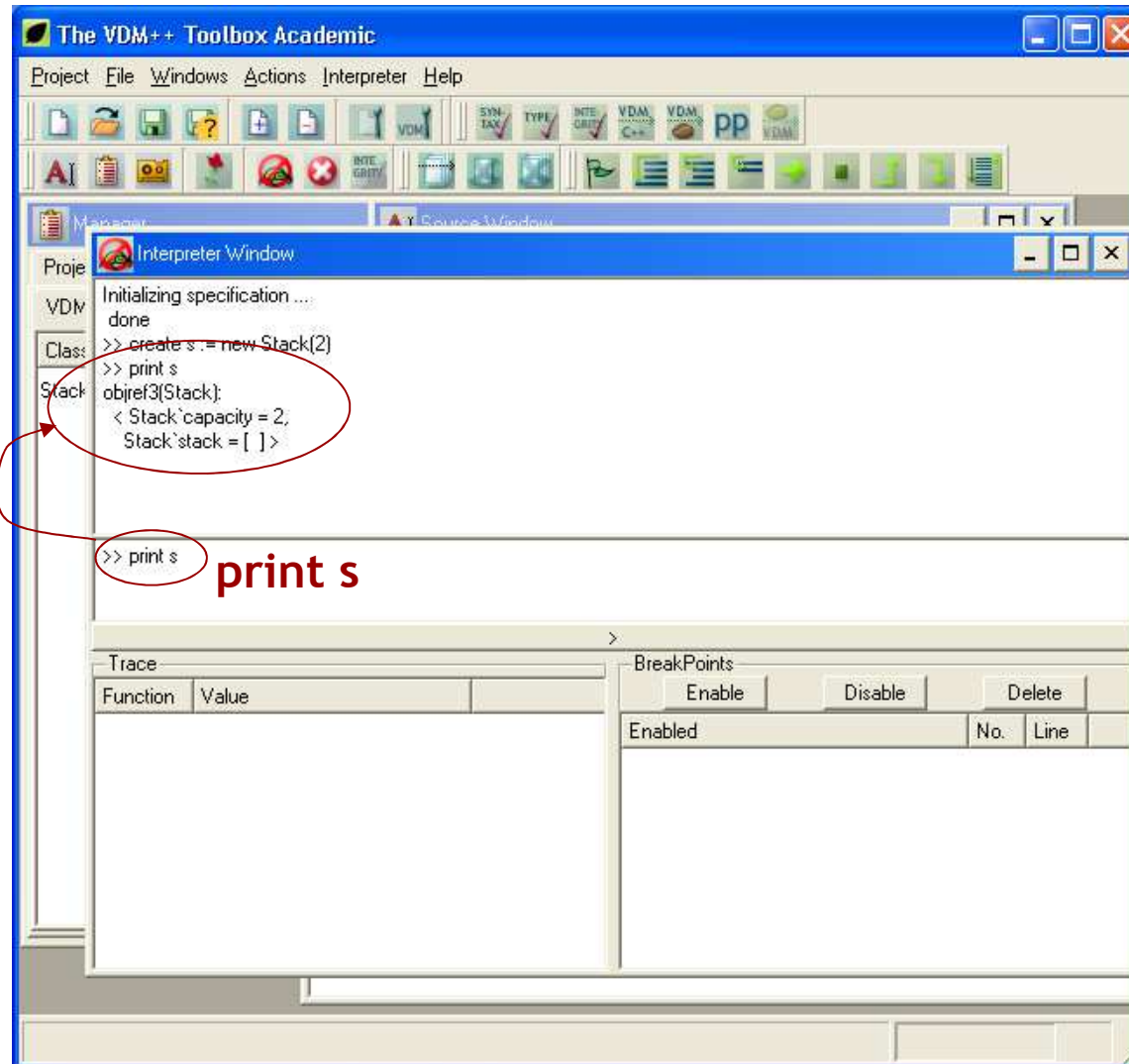




# Run the interpreter – create na object



# Run the interpreter – state of an object



# Run the interpreter – state of an object

The screenshot shows the VDM++ Toolbox Academic interface. The main window displays the state of an object 's' in the interpreter. The state is shown as:

```
>> print s
objref3(Stack):
< Stack`capacity = 2,
  Stack`stack = [ ] >
```

Red circles highlight the commands in the interpreter window:

- `>> print s.Push(1)` (no return value)
- `>> print s.Top()`
- `>> print s`

The state of the object 's' after these commands is shown in the bottom window:

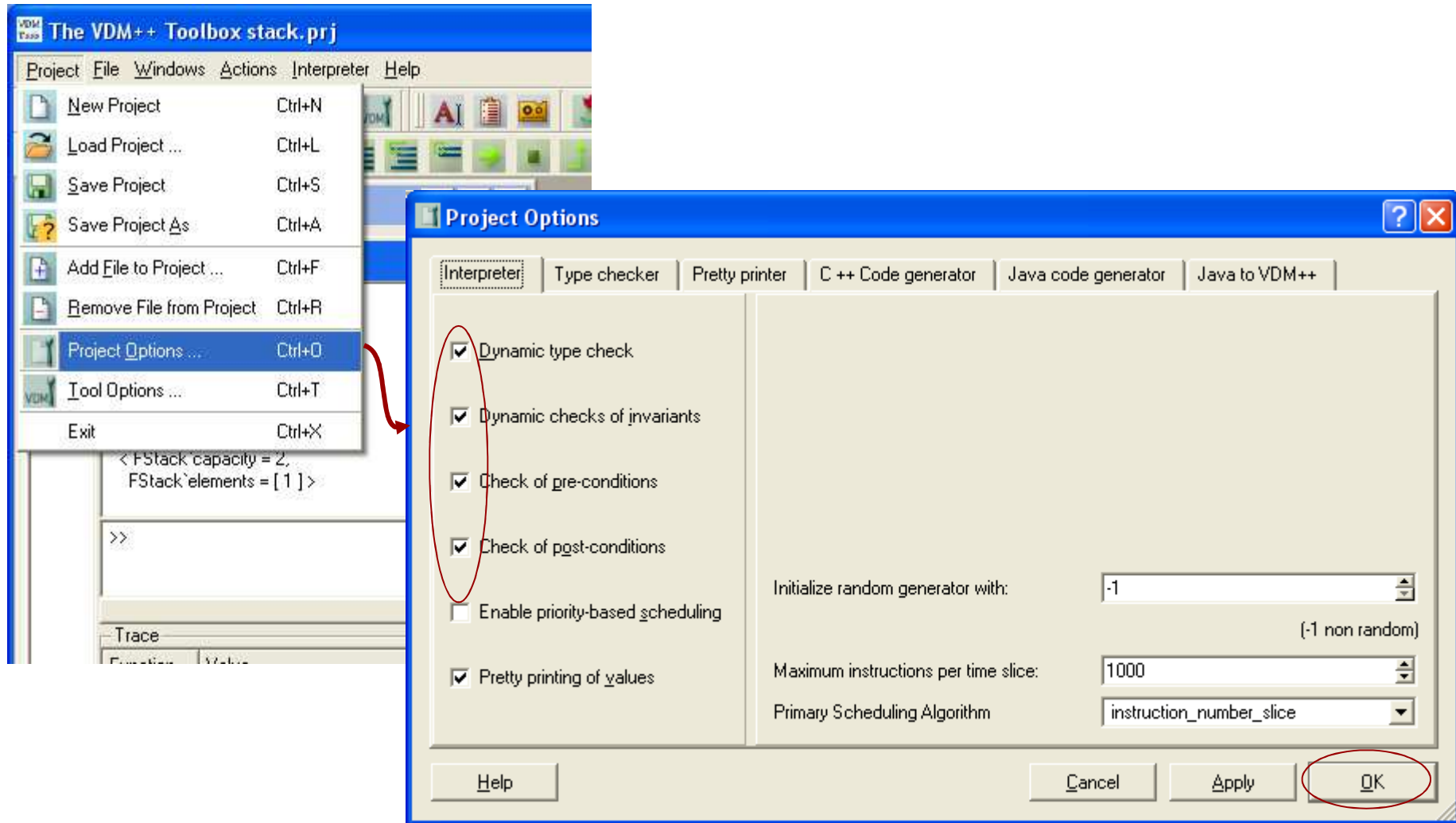
```
>> print s
objref3(Stack):
< Stack`capacity = 2,
  Stack`stack = [ 1 ] >
```

The Trace window shows the function call:

Function	Value
print s.Push(1)[...]	

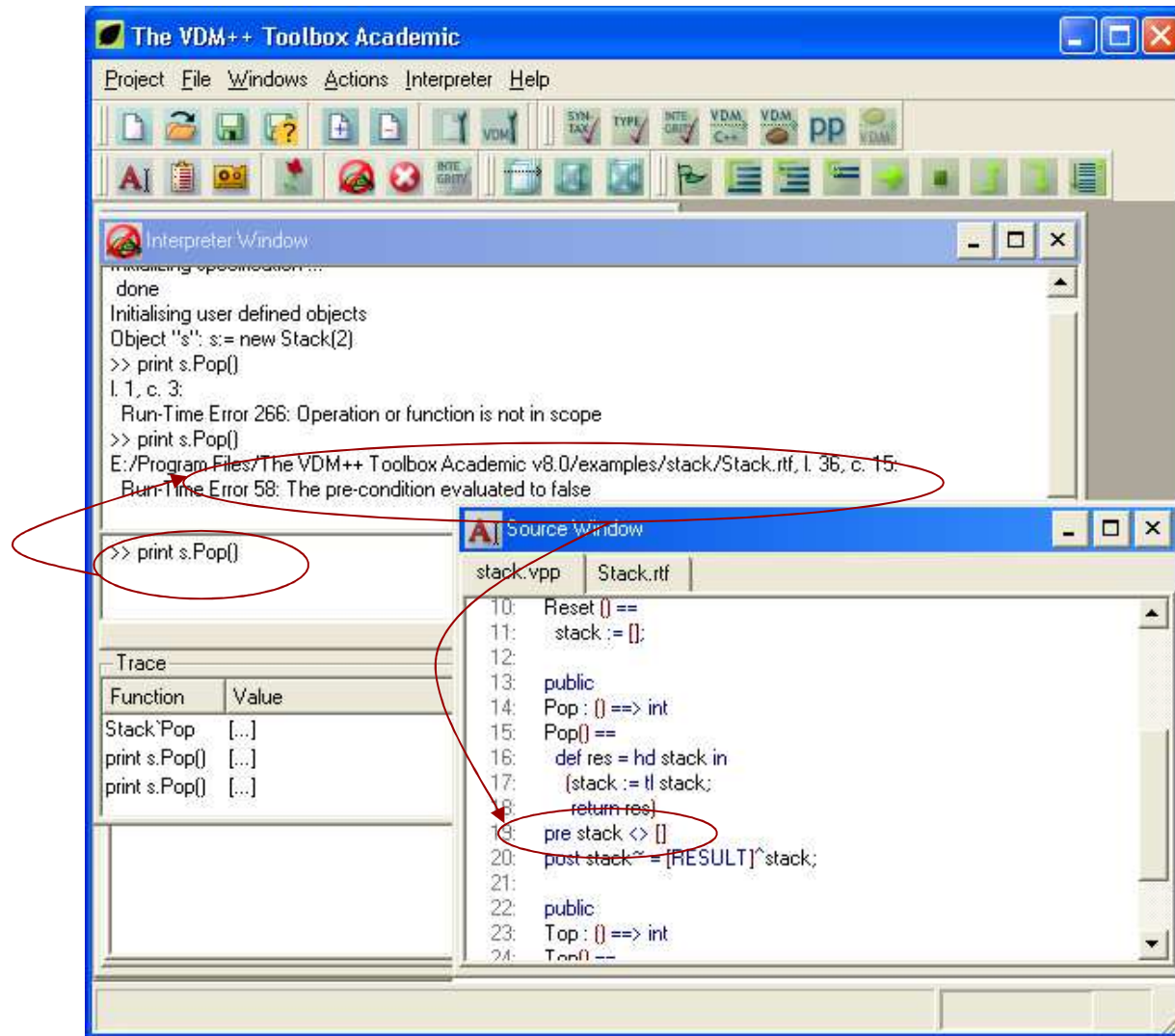
# Check invariants, pre and post conditions

## Activate checks



# Check invariants, pre and post conditions

## Pre condition violation



# Check invariants, pre and post conditions

## Pre condition violation

The screenshot shows the VDM++ Toolbox Academic interface. The Interpreter Window displays a Run-Time Error 58: "The pre-condition evaluated to false". The error message is circled in red. Below the error, the interpreter shows the state of the stack: "objref9(Stack): < Stack`capacity = 2, Stack`stack = [ 2,1 ] >". The source code in the Source Window shows the definition of the Stack type and the Push function. The pre-condition for the Push function is circled in red: "pre len stack < capacity".

```
Run-Time Error 58: The pre-condition evaluated to false
>> p s
objref9(Stack):
< Stack`capacity = 2,
  Stack`stack = [ 2,1 ] >
>> p s.Push(3)
E:/Program Files/The VDM++ Toolbox Academic v8.0/examples/stack/Stack.rtf, l. 24, c. 19:
Run-Time Error 58: The pre-condition evaluated to false
```

Só tem capacidade para 2 ...

```
stack.vpp Stack.rtf
15: ( capacity := x;
16:   return self; );
17:
18: Reset : () ==> ()
19: Reset () ==
20:   stack := [];
21:
22: Push : int ==> ()
23: Push(x) == stack := [x] ^ stack
24: pre len stack < capacity
25: post stack = [x] ^ stack;
26:
27:
28: --I INF 11 1
```





# Consistency of the specification

Gera propriedades de integridade para as classes seleccionadas

Checked	Class	Member	Location	Index	Type
No	Stack	Pop	operation	1	non-empty sequence
No	Stack	Pop	operation	2	state invariants
No	Stack	Pop	operation	3	non-empty sequence
No	Stack	Pop	operation	4	post condition
No	Stack	Push	operation	1	state invariants
No	Stack	Push	operation	2	post condition
No	Stack	Stack	operation	1	state invariants

```
1:
2: --LINE 6 1
3: class Stack
4:   instance variables
5:     private stack : seq of int := [];
6:     private capacity: nat := 0;
7:     inv len stack <= capacity;
8:   operations
9:     public Stack : nat ==> Stack
10:       Stack(c) == [stack := []; capac
11:
12: --LINE 8 1
13:     post stack = [] and capacity =
14:
15: --LINE 10 1
16:     public Push: int ==> ()
17:       Push(x) == stack := [x] ^ stack
18:       pre len stack < capacity
19:       post stack = [x] ^ stack~;
20:     public Pop : () ==> ()
21:       Pop () == stack := tl stack
22:       pre stack <> []
23:       post stack = tl stack~;
24:     public Top : () ==> int
25:       Top() == return hd stack
26:       pre stack <> []
```





# Test cases into a test class

```
Class TestStack

Definição em VDM++ da classe TestStack, para testar a classe Stack.
class TestStack
  operations
    -- operação auxiliar, que tira partido do facto do
    -- interpretador parar quando se viola uma pré-condição
    public AssertTrue : bool ==> ()
    AssertTrue(a) == return
    pre a;

    public TestGoodUsage : () ==> ()
    TestGoodUsage() ==
    (
      decl s : Stack := new Stack(2);
      s.Push(1);
      AssertTrue(s.Top() = 1);
      s.Push(2);
      AssertTrue(s.Top() = 2);
      s.Pop();
      AssertTrue(s.Top() = 1)
    );

    public TestPopEmptyStack : () ==> ()
    TestPopEmptyStack() == new Stack(2).Pop();

    public TestTopEmptyStack : () ==> int
    TestTopEmptyStack() == new Stack(2).Top();

    public TestPushStackFull : () ==> ()
    TestPushStackFull() == new Stack(0).Push(1);

end TestStack
```

1. Create the file
2. Add to the project
3. Check syntax
4. Check types
5. Run the interpreter
6. Run the following commands:

```
Interpreter Window

Initializing specification ...
done
>> cr t := new TestStack()
>> p t.TestGoodUsage()
(no return value)
>> p t.TestPopEmptyStack()
E:/AULAS/MFES/0708/Stack/Stack.rtf, l. 22, c. 14:
  Run-Time Error 58: The pre-condition evaluated to false
>> p t.TestTopEmptyStack()
E:/AULAS/MFES/0708/Stack/Stack.rtf, l. 26, c. 14:
  Run-Time Error 58: The pre-condition evaluated to false
>> p t.TestPushStackFull()
E:/AULAS/MFES/0708/Stack/Stack.rtf, l. 18, c. 18:
  Run-Time Error 58: The pre-condition evaluated to false

>>
```

# Coverage analysis

- ◆ 1º - Put placeholders for coverage analysis tables

Name of the class, style VDM\_TC\_TABLE.

It will substituted with table with test case coverage information after doing “pretty print”.

The screenshot shows a Microsoft Word document titled "Stack.rtf". The document content is as follows:

**Classe Stack**

Definição em VDM++ da classe Stack, conforme slides da aula teórica (ficheiro VDM1.ppt).

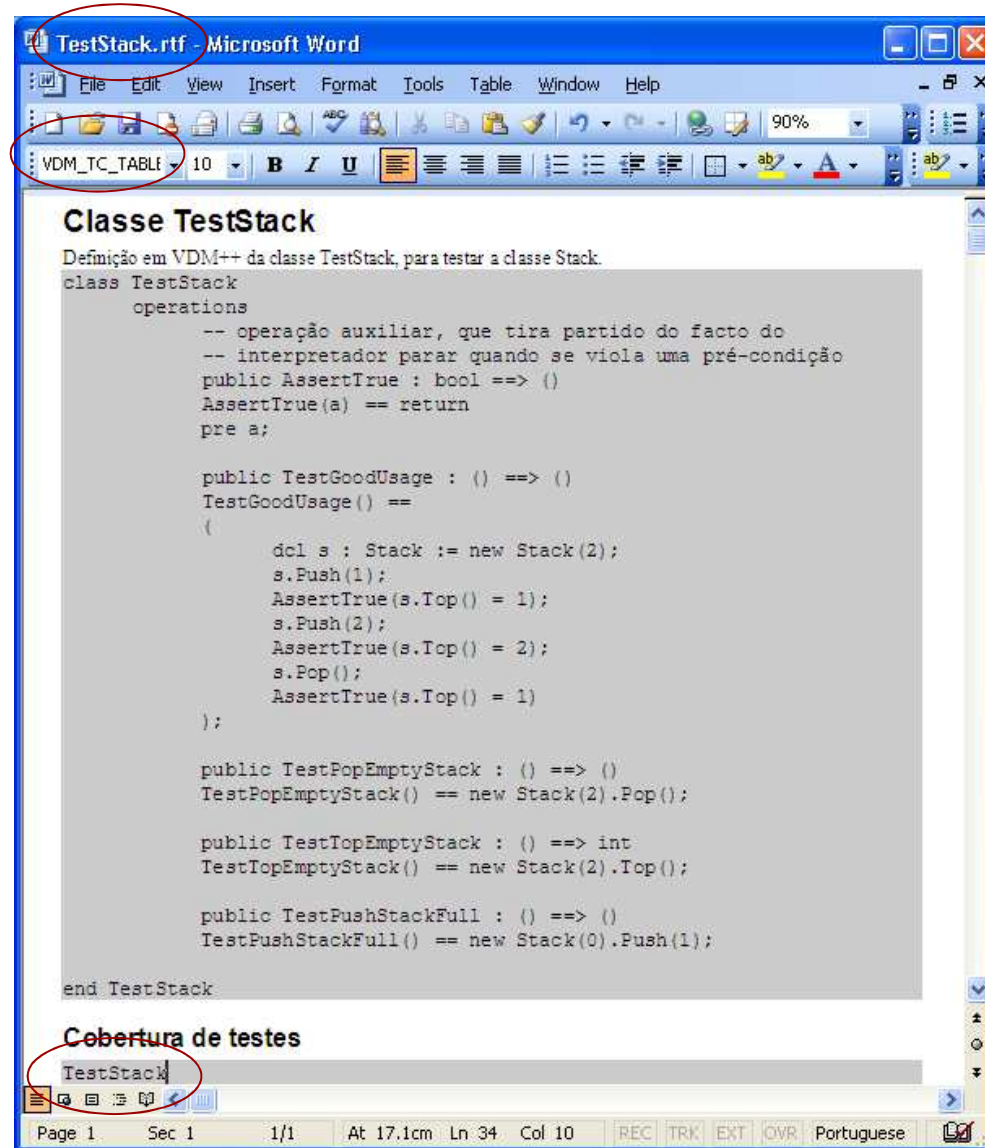
```
class Stack
instance variables
  private stack : seq of int := [];
  private capacity: nat := 0 ;
  inv len stack <= capacity;
operations
  public Stack : nat ==> Stack
    Stack(c) == (stack := []; capacity := c)
    post stack = [] and capacity = c ;
  public Push: int ==> ()
    Push(x) == stack := [x] ^ stack
    pre len stack < capacity
    post stack = [x] ^ stack~ ;
  public Pop : () ==> ()
    Pop () == stack := tl stack
    pre stack <> []
    post stack = tl stack~ ;
  public Top : () ==> int
    Top() == return hd stack
    pre stack <> []
    post RESULT = hd stack;
end Stack
```

**Cobertura de testes**

Stack
-------

The status bar at the bottom indicates: Page 1, Sec 1, 1/1, At 15,5cm Ln 30 Col 6, REC TRK EXT CVR Portuguese.

# Coverage analysis



The screenshot shows a Microsoft Word document titled 'TestStack.rtf'. The document contains VDM++ code for a class named 'TestStack'. The code includes several test operations: 'TestGoodUsage', 'TestPopEmptyStack', 'TestTopEmptyStack', and 'TestPushStackFull'. Below the code, there is a table titled 'Cobertura de testes' (Test Coverage) with one row for 'TestStack'. The table is currently empty. The status bar at the bottom indicates 'Page 1', 'Sec 1', '1/1', 'At 17.1cm Ln 34 Col 10', and 'Portuguese'.

```
class TestStack
operations
  -- operação auxiliar, que tira partido do facto do
  -- interpretador parar quando se viola uma pré-condição
  public AssertTrue : bool ==> ()
  AssertTrue(a) == return
  pre a;

  public TestGoodUsage : () ==> ()
  TestGoodUsage() ==
  (
    dol s ; Stack := new Stack(2);
    s.Push(1);
    AssertTrue(s.Top() = 1);
    s.Push(2);
    AssertTrue(s.Top() = 2);
    s.Pop();
    AssertTrue(s.Top() = 1)
  );

  public TestPopEmptyStack : () ==> ()
  TestPopEmptyStack() == new Stack(2).Pop();

  public TestTopEmptyStack : () ==> int
  TestTopEmptyStack() == new Stack(2).Top();

  public TestPushStackFull : () ==> ()
  TestPushStackFull() == new Stack(0).Push(1);

end TestStack
```

Cobertura de testes	
TestStack	

It is possible to do the same thing with test code

...

# Coverage analysis

- ◆ 2º - For each test case XPTO, prepare two text file (for example in the directory of the project):
  - file XPTO.arg, with command for the interpreter
  - File XPTO.arg.exp, with result expected after running the command (print command)

Input



```
new TestFStack().TestGoodUsage()
```



```
new TestFStack().TestPopEmptyStack()
```

Etc.

Output expected



```
(no return value)
```

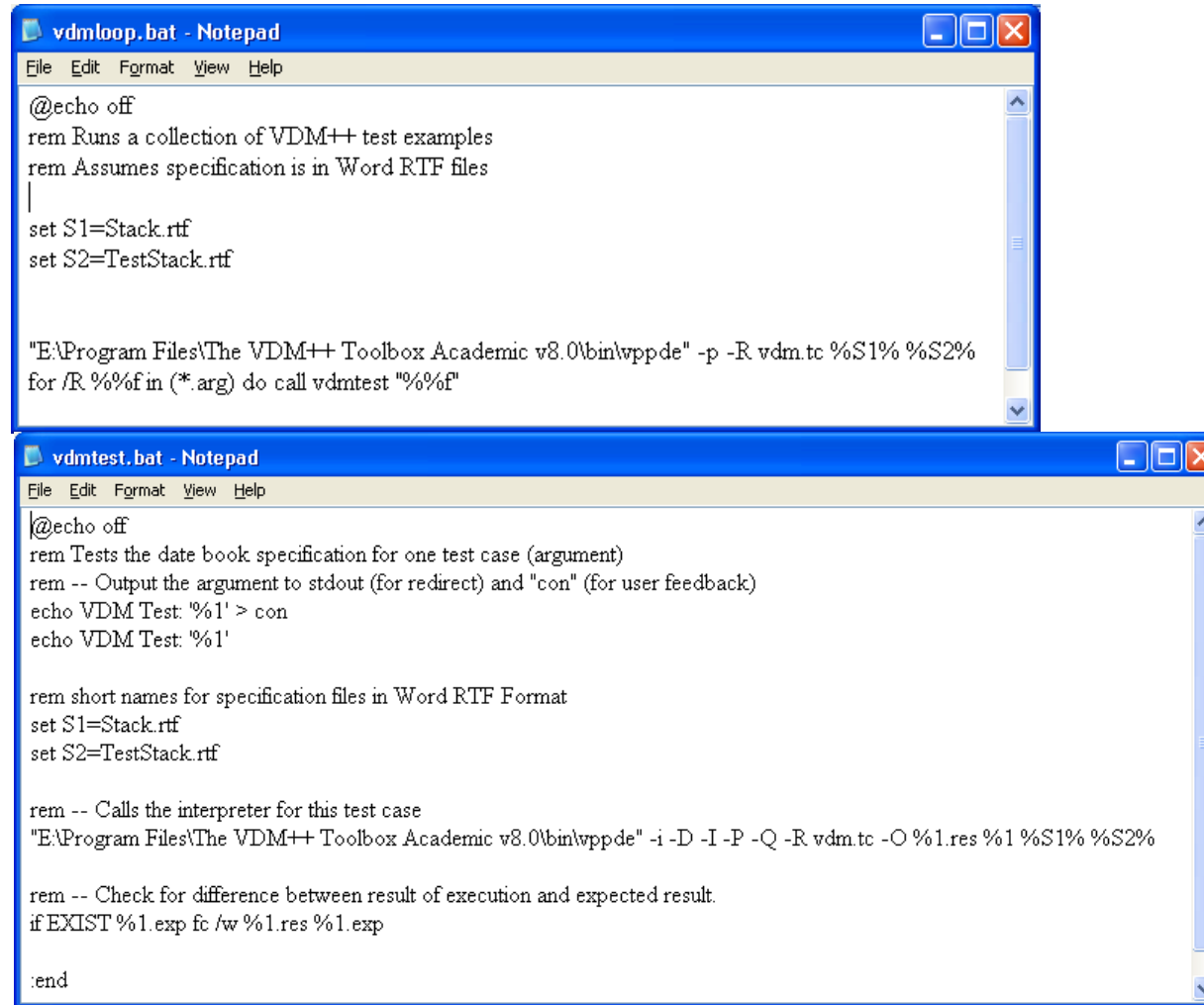


```
(no return value)
```

It should violate pre condition, but there is no way to indicate it (!?)

# Coverage analysis

- ◆ 3º - Prepare test *scripts* (in same directory)



```
vdmloop.bat - Notepad
File Edit Format View Help
@echo off
rem Runs a collection of VDM++ test examples
rem Assumes specification is in Word RTF files
|
set S1=Stack.rtf
set S2=TestStack.rtf

"E:\Program Files\The VDM++ Toolbox Academic v8.0\bin\wppde" -p -R vdm.tc %S1% %S2%
for /R %%f in (*.arg) do call vdmtest "%%f"

vdmtest.bat - Notepad
File Edit Format View Help
@echo off
rem Tests the date book specification for one test case (argument)
rem -- Output the argument to stdout (for redirect) and "con" (for user feedback)
echo VDM Test: %1' > con
echo VDM Test: %1'

rem short names for specification files in Word RTF Format
set S1=Stack.rtf
set S2=TestStack.rtf

rem -- Calls the interpreter for this test case
"E:\Program Files\The VDM++ Toolbox Academic v8.0\bin\wppde" -i -D -I -P -Q -R vdm.tc -O %1.res %1 %S1% %S2%

rem -- Check for difference between result of execution and expected result.
if EXIST %1.exp fc /w %1.res %1.exp

:end
```



# Coverage analysis

## ◆ 4º - Run test script

```
C:\ Command Prompt
E:\AULAS\MFES\0708\Stack>
E:\AULAS\MFES\0708\Stack>vdmloop
Parsing "Stack.rtf" (Word RTF) ... done
Parsing "TestStack.rtf" (Word RTF) ... done
UDM Test: 'E:\AULAS\MFES\0708\Stack\TestGoodUsage.arg'
UDM Test: 'E:\AULAS\MFES\0708\Stack\TestGoodUsage.arg'
Parsing "Stack.rtf" (Word RTF) ... done
Parsing "TestStack.rtf" (Word RTF) ... done
Initializing specification ...Initializing TestStack
Initializing Stack
done
(no return value)
Comparing files E:\AULAS\MFES\0708\STACK\TestGoodUsage.arg.res and E:\AULAS\MFES\0708\STACK\TESTGOODUSAGE.ARG.EXP
FC: no differences encountered
UDM Test: 'E:\AULAS\MFES\0708\Stack\TestPopEmptyStack.arg'
UDM Test: 'E:\AULAS\MFES\0708\Stack\TestPopEmptyStack.arg'
Parsing "Stack.rtf" (Word RTF) ... done
Parsing "TestStack.rtf" (Word RTF) ... done
Initializing specification ...Initializing TestStack
Initializing Stack
done
Stack.rtf. 1 22, s. 11:
Run-Time Error 58: The pre-condition evaluated to false
FC: cannot open E:\AULAS\MFES\0708\STACK\TESTPOPEMPTYSTACK.ARG.RES - No such file or folder
E:\AULAS\MFES\0708\Stack>_
```

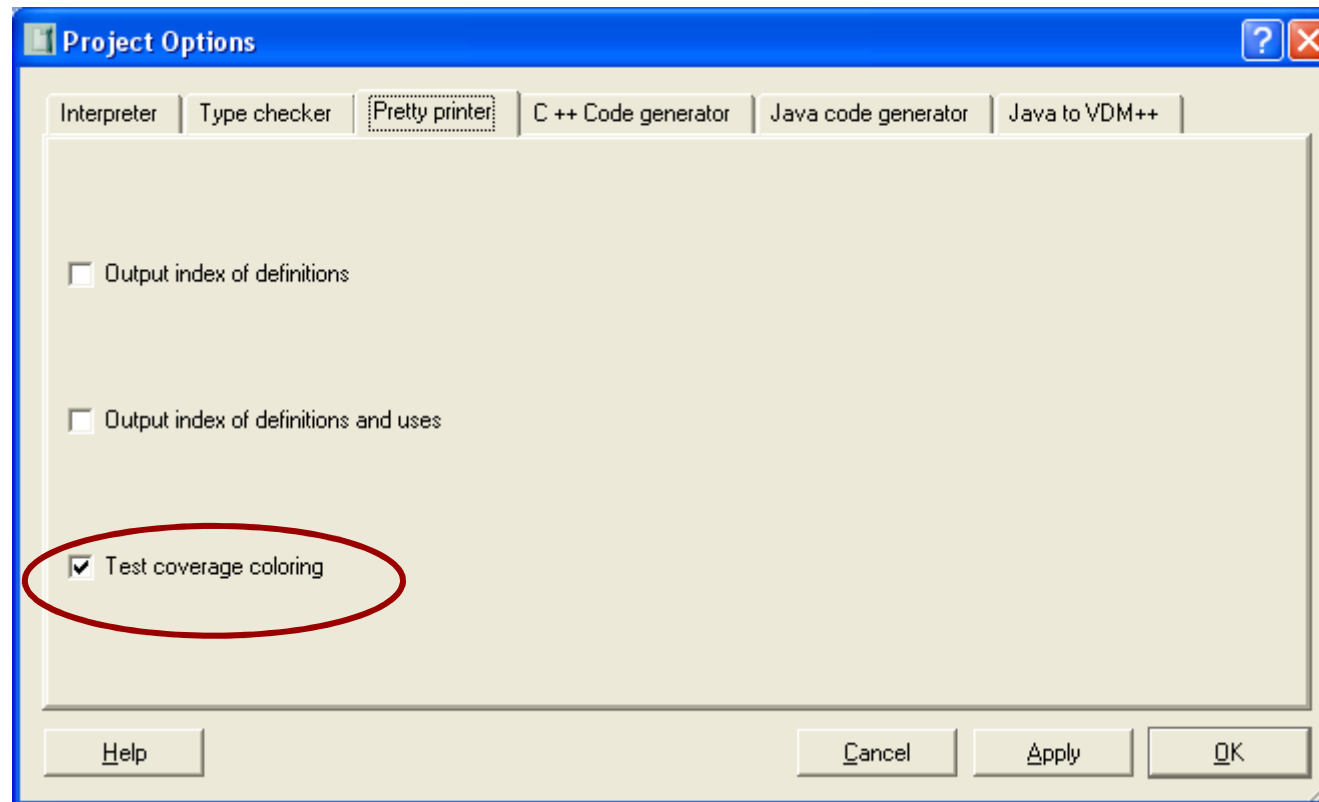
OK

As expected

It also builds a file “vdm.tc” with code coverage information, to use by “pretty printer”!

# Coverage analysis

- ◆ 5° - Prepare “pretty printer” configuration





# Coverage analysis

- ◆ 6° - Run “pretty print”

The screenshot shows the 'The VDM++ Toolbox Academic' application window. The 'Manager' window is open, displaying a table with columns for 'Classes', 'Syntax', 'Type', 'C++', 'Java', and 'Pretty Print'. The 'Stack' and 'TestStack' rows are selected, and the 'Pretty Print' column contains 'P' for both. A toolbar at the top contains a 'pp' icon circled in red. A 'Log Window' is visible at the bottom.

1° - select

2° - pretty print

3° - shows up

Classes	Syntax	Type	C++	Java	Pretty Print
Stack	S	T			P
TestStack	S	T			P

# Coverage analysis

- 7º - Analyse the results (files .rtf.rtf !)

**Stack.rtf.rtf - Microsoft Word**

Definição em VDM da classe Stack, conforme slides da aula teórica (ficheiro VDMI.ppt).

```

class Stack
instance variables
private stack : seq of int := [];
private capacity: nat := 0 ;
inv len stack <= capacity;
operations
public Stack : nat ==> Stack
Stack(c) == (stack := []; capacity := c)
post stack = [] and capacity = c ;
public Push: int ==> ()
Push(x) == stack := [x] ^ stack
pre len stack < capacity
post stack = [x] ^ stack~ ;
public Pop : () ==> ()
Pop () == stack := tl stack
pre stack <> []
post stack = tl stack~ ;
public Top : () ==> int
Top() == return hd stack
pre stack <> []
post RESULT = hd stack;
end Stack
    
```

**Cobertura de testes** **100% code coverage**

name	#calls	coverage
Stack Pop	2	100%
Stack Top	3	100%
Stack Push	2	100%
Stack Stack	2	100%
<b>total</b>		<b>100%</b>

**TestStack.rtf.rtf - Microsoft Word**

Definição em VDM da classe TestStack, para testar a classe Stack

```

class TestStack
operations
-- opera[cedilla] [[avilde]] auxiliar, que tira partido do facto do
-- interpretador parar quando se viola uma pr[[eacute]]-
condi[[cedilla]] [[avilde]]
public AssertTrue : bool ==> ()
AssertTrue(a) == return
pre a;

public TestGoodUsage : () ==> ()
TestGoodUsage() ==
{
del s : Stack := new Stack(2);
s.Push(1);
AssertTrue(s.Top() = 1);
s.Push(2);
AssertTrue(s.Top() = 2);
s.Pop();
AssertTrue(s.Top() = 1)
};

public TestPopEmptyStack : () ==> ()
TestPopEmptyStack() == new Stack(2).Pop();

public TestPopEmptyStack : () ==> int
TestPopEmptyStack() == new Stack(2).Top();

public TestPushStackFull : () ==> ()
TestPushStackFull() == new Stack(0).Push(1);
end TestStack
    
```

**Cobertura de testes**

name	#calls	coverage
TestStack.AssertTrue	3	100%
TestStack.TestGoodUsage	1	100%
TestStack.TestPopEmptyStack	1	100%
TestStack.TestPushStackFull	0	0%
TestStack.TestTopEmptyStack	0	0%
<b>total</b>		<b>83%</b>

Not executed code in red

Two operation that were not executed by test cases

# Link to Rational Rose

The screenshot shows the VDM++ Toolbox Academic Stack interface. The main window has a menu bar (Project, File, Windows, Actions, Interpreter, Help) and a toolbar. A red circle highlights a specific icon in the toolbar. Below the main window, a 'Rose Link Window (VDM++ <-> UML)' dialog is open. The dialog has a table with columns 'Classes', 'VDM++', 'Action', and 'UML'. The 'Stack' class is selected. Below the table, there are buttons for 'Diff Selected', 'Select All', 'Select None', and 'Map'. A red circle highlights the 'Map' button. The 'Messages' section is empty.

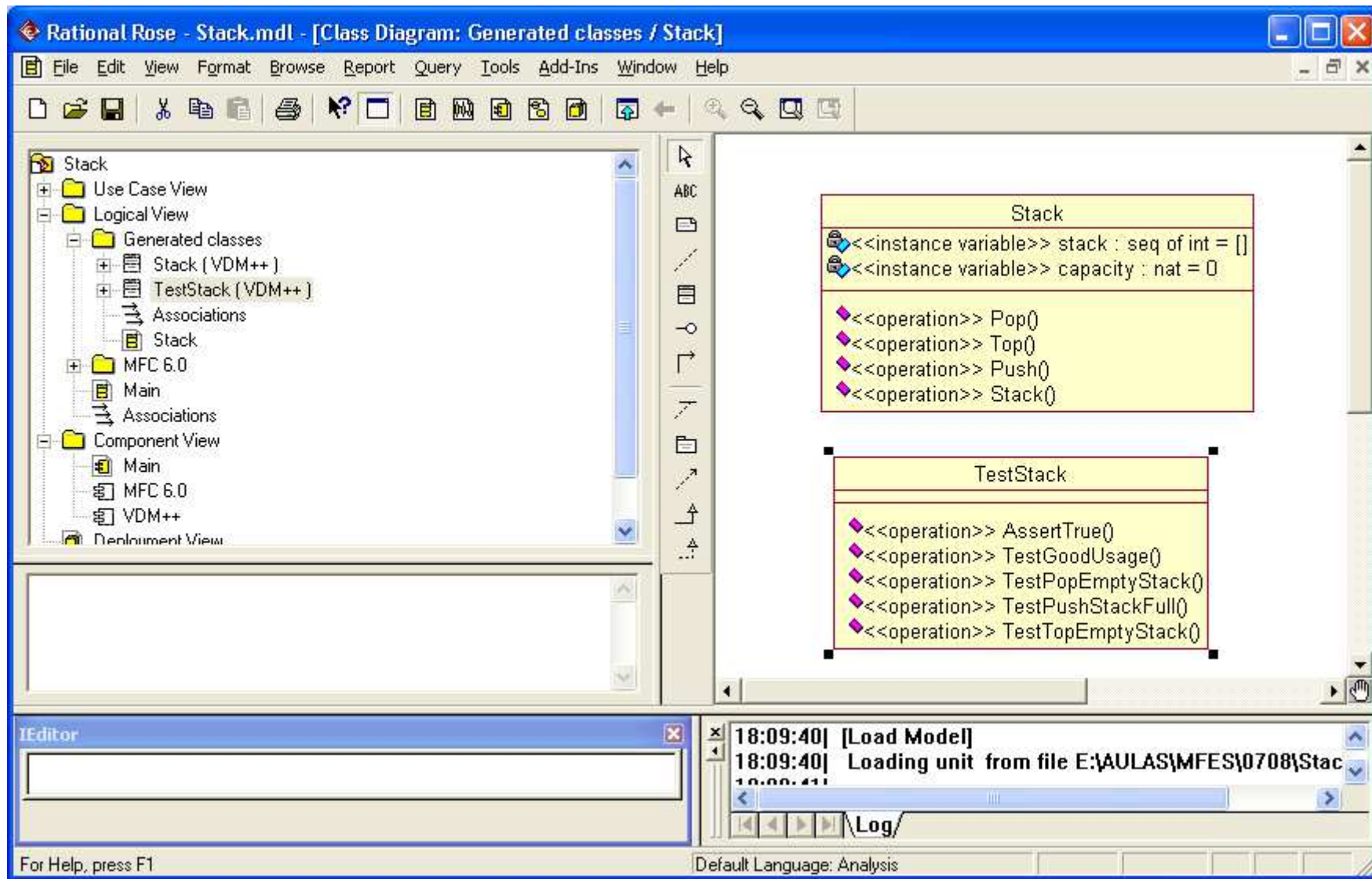
**1° - run link to Rational Rose**

**2° - select classes to synchronize**

**3° - Map**

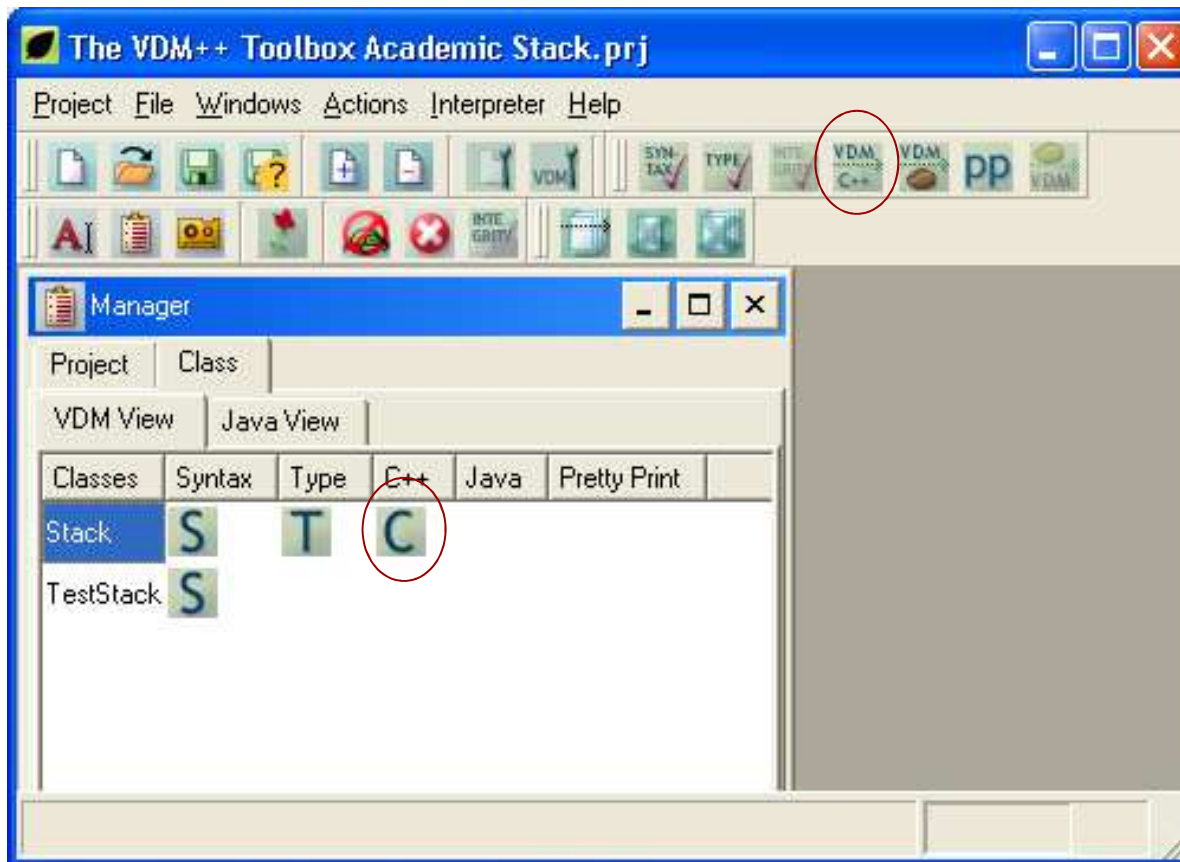
Classes	VDM++	Action	UML
CWinApp	D	[Hand icon]	-
CWindowDC	D	[Hand icon]	-
CWinThread	D	[Hand icon]	-
CWnd	D	[Hand icon]	-
CWordArray	D	[Hand icon]	-
Stack	-	[Hand icon]	-

# Rational Rose





# Code generation



## Files generated:

CGBase.cpp

CGBase.h

Stack.cpp

Stack.h

Stack\_anonym.cpp

Stack\_anonym.h

# Build final report of your projects

- ◆ Build a Word document, for example “master.rtf”, in which you should insert links for the several files “Insert File ...” -> button “Insert as Link”
- ◆ You should insert files produced by pretty print (*ClassName.rtf.rtf*)
- ◆ Use “Update field” to update the links
- ◆ See example in “Stack\master.rtf”
- ◆ See example with every files in [paginas.fe.up.pt/~apaiva/teach/MFES\\_material/VDMStack.rar](http://paginas.fe.up.pt/~apaiva/teach/MFES_material/VDMStack.rar)



# References and additional reading

- ◆ *VDM information web site:*

<http://www.vdmtools.jp/en/index.php>

- ◆ *VDM++: Language manual*

[http://paginas.fe.up.pt/~apaiva/teach/0910/MFES\\_files/langmanpp\\_a4E.pdf](http://paginas.fe.up.pt/~apaiva/teach/0910/MFES_files/langmanpp_a4E.pdf)

- ◆ *VDMTools: User manual*

[http://paginas.fe.up.pt/~apaiva/teach/0910/MFES\\_files/usermanpp\\_a4E.pdf](http://paginas.fe.up.pt/~apaiva/teach/0910/MFES_files/usermanpp_a4E.pdf)

- ◆ *VDMTools: Link to Rational Rose*

[http://paginas.fe.up.pt/~apaiva/teach/0910/MFES\\_files/roselinkpp\\_a4E.pdf](http://paginas.fe.up.pt/~apaiva/teach/0910/MFES_files/roselinkpp_a4E.pdf)

- ◆ *VDMTools: Overture/VDM++*

<http://wiki.overturetool.org/images/d/d1/VDMPPGuideToOvertureV1.pdf>