Métodos Formais em Engenharia de Software

VDMTool Tutorial

Ana Paiva

apaiva@fe.up.pt 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

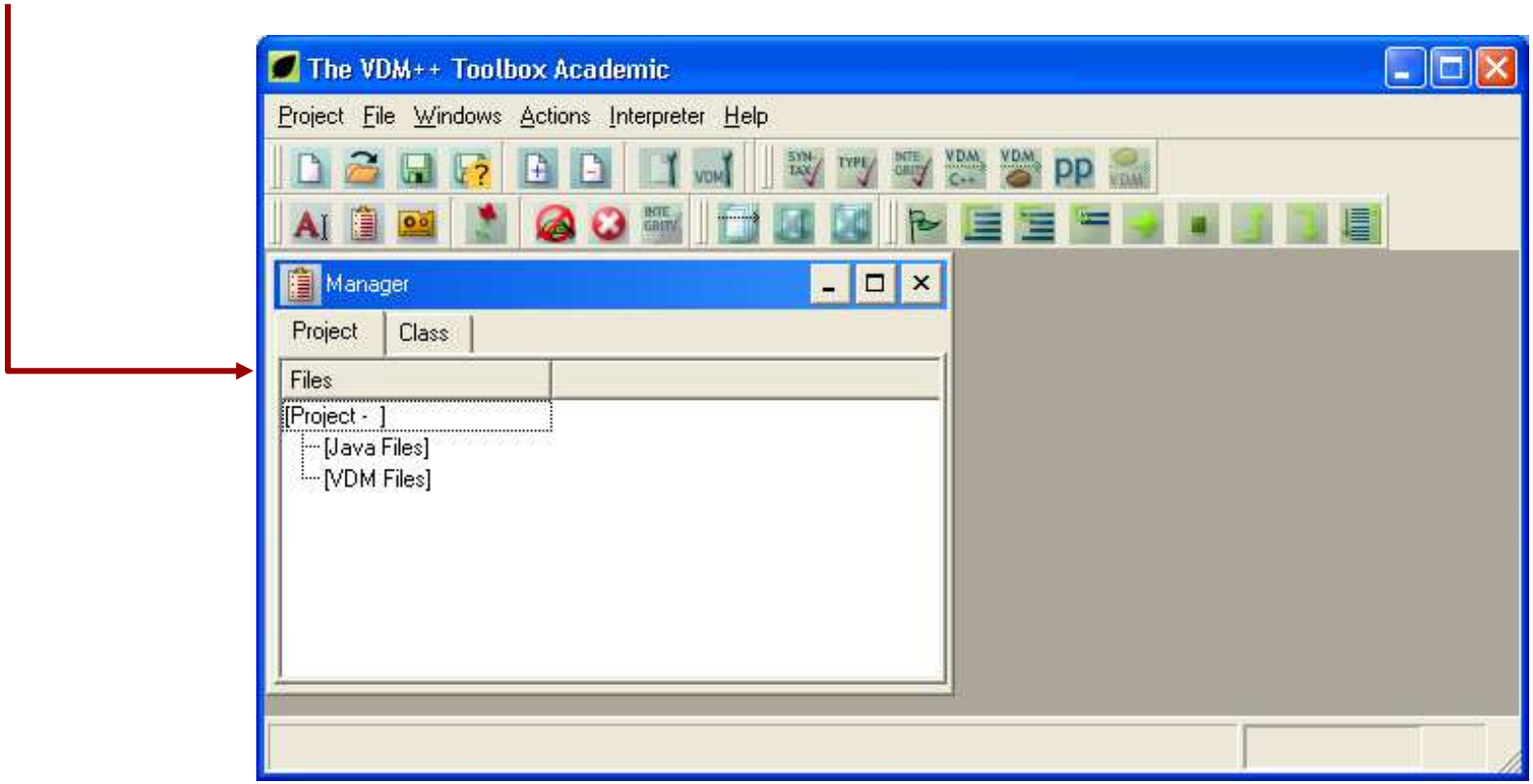
- VDM++ Toolbox Academic v8.1
- *Download* at http://www.vdmtools.jp/en/index.php
- Install with administrator privileges
  - Download and run *setup*

Start

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

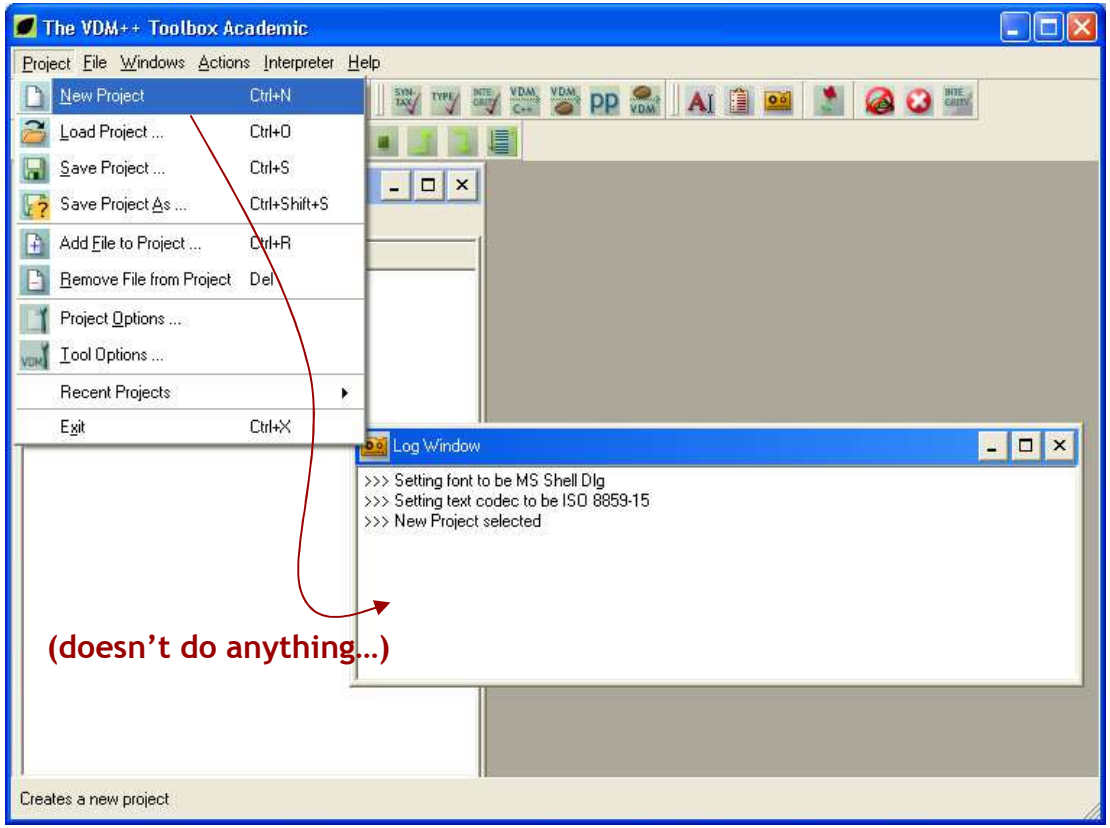
ou

…\The VDM++ Toolbox v9.0.2\bin\vppgde.exe



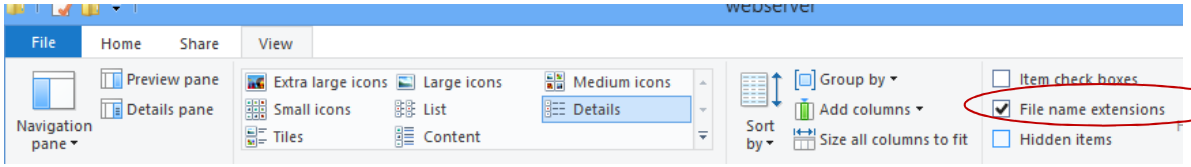
Create a projet (1/1)

1º



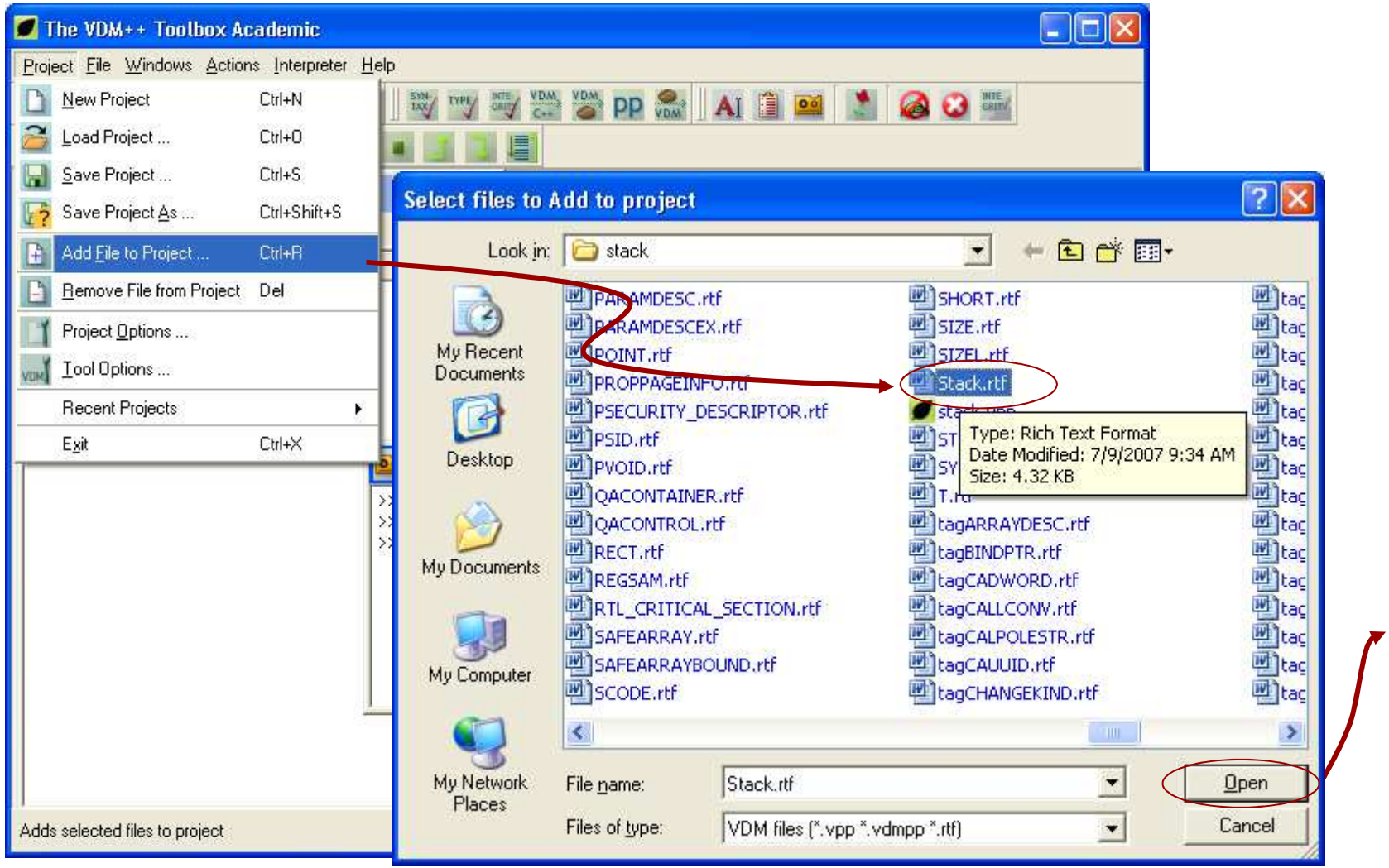
2º

Show "File name extension"

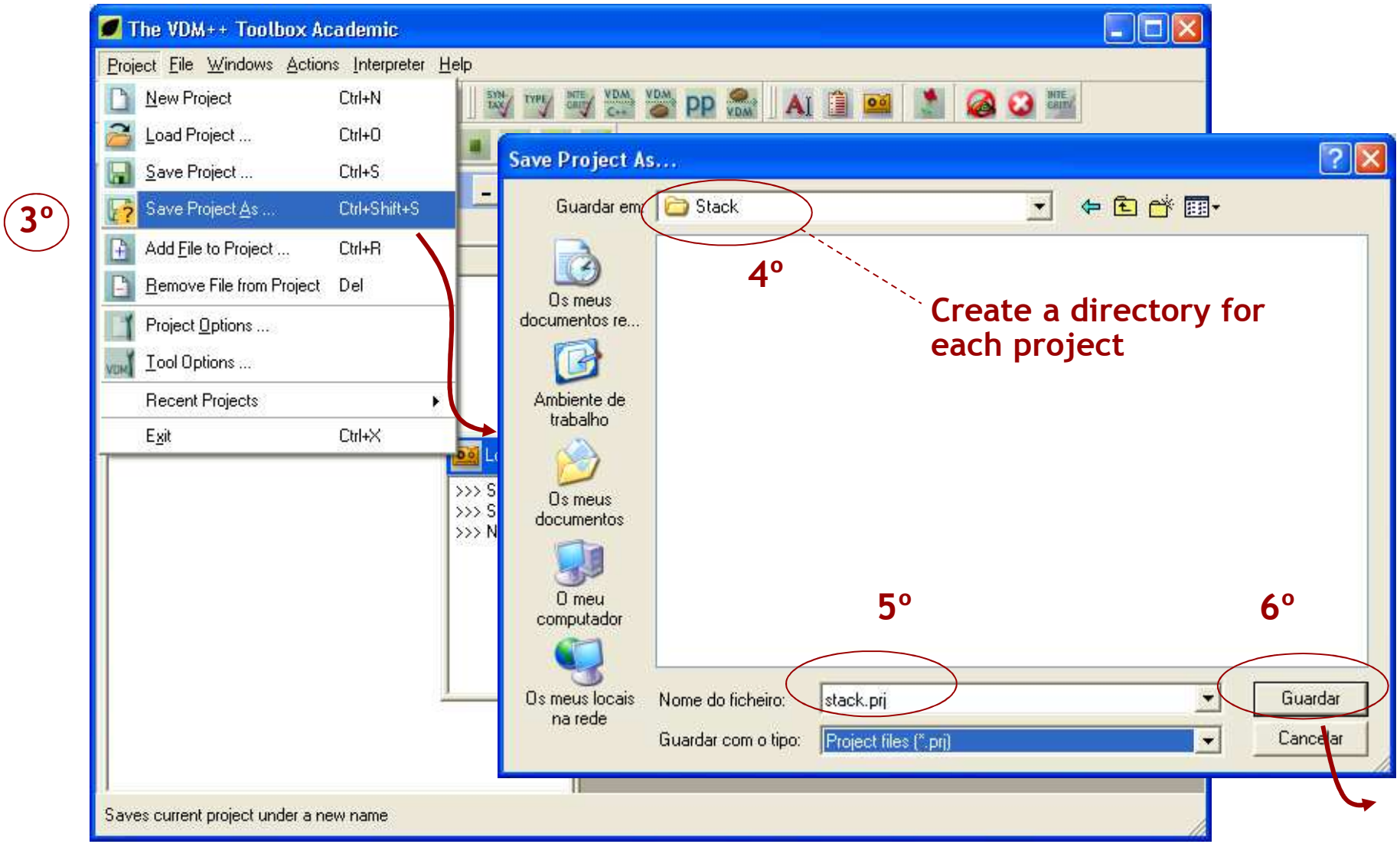


Note: There are *.rtf* and *.rtf.rtf* files

Add a file to a project (1/2)



Save a project (1/3)

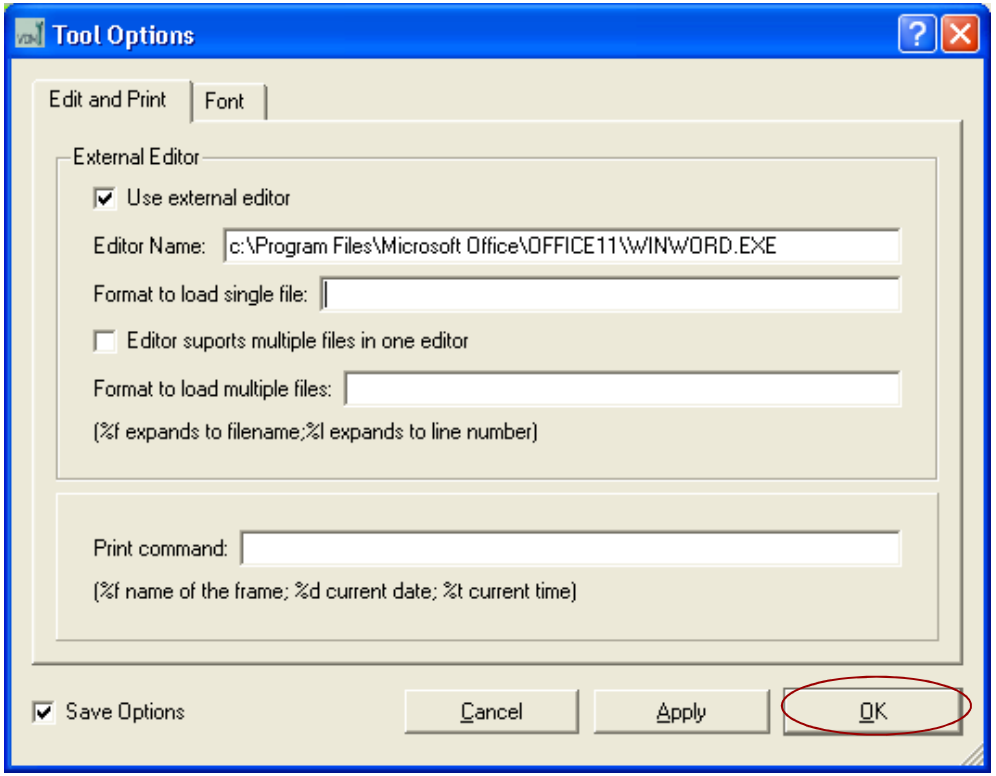


Write a specification (2/1)

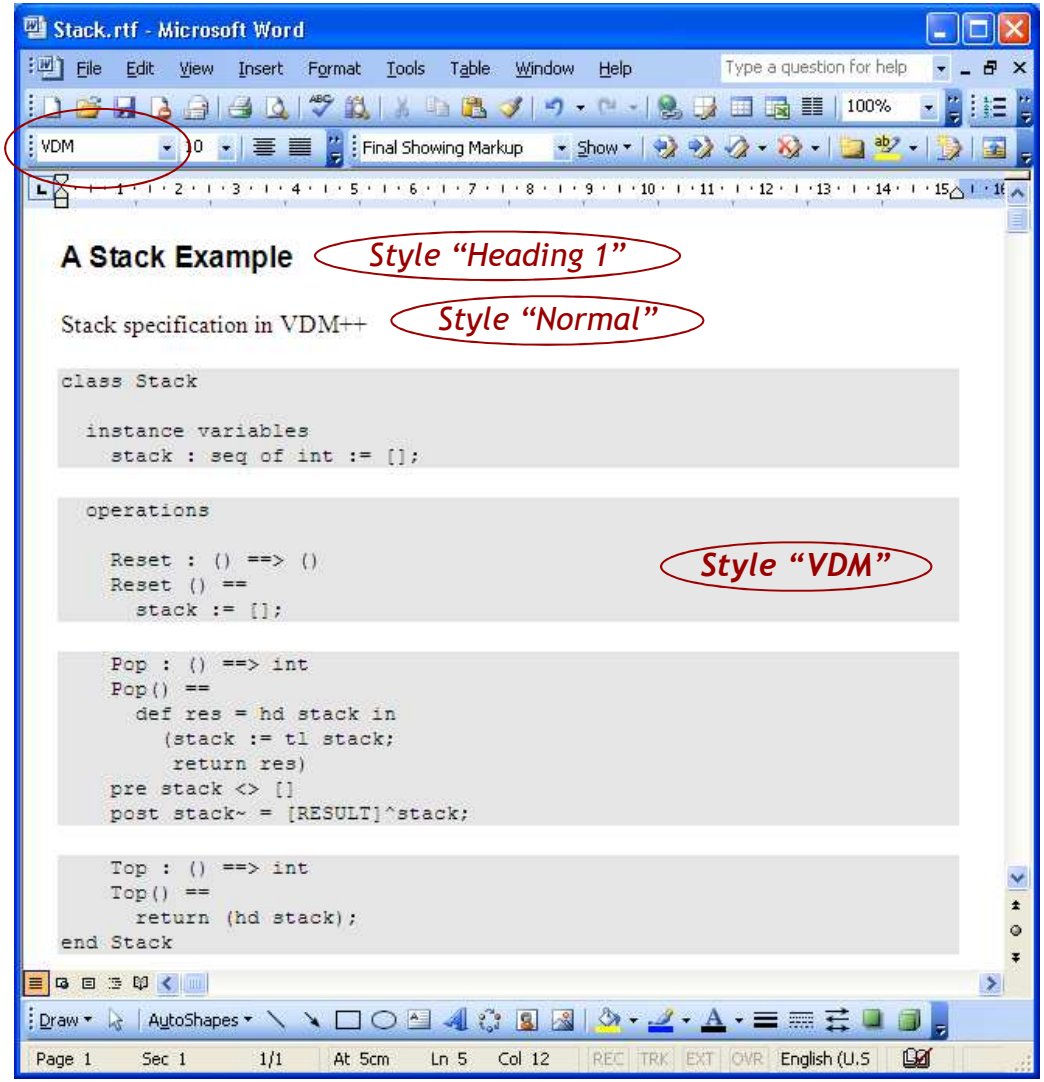
- Literal programming (code and documentation together)
- VDM++ code is written in one RTF (Rich Text Format) document, using syles predefined on the template "The VDM++ Toolbox Academic v9.0.2\word\VDM.dot"
- Open file "VDM.dot" and save it in RTF format, with the name of the class to create, for exemple "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)

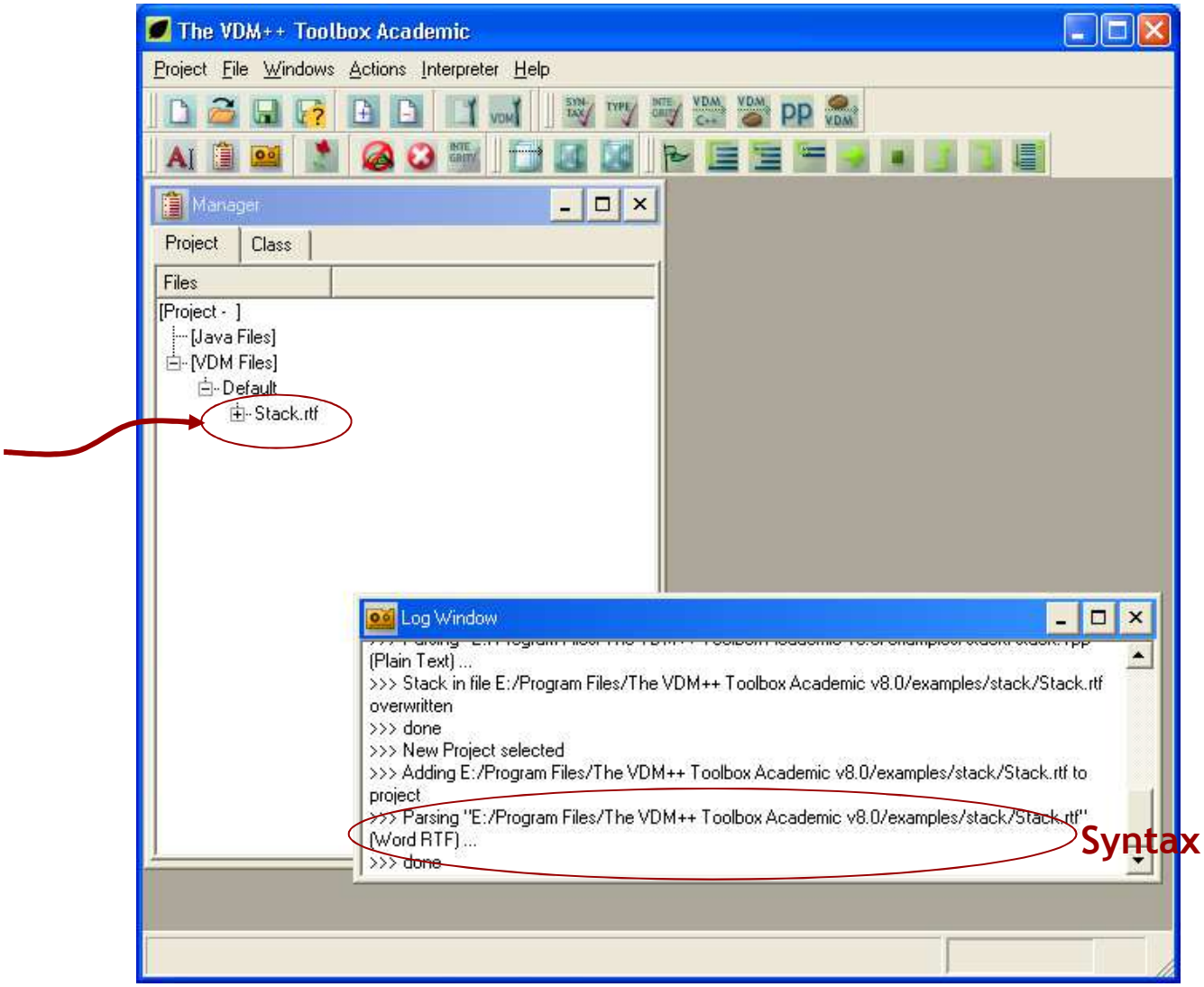
- To open Microsoft Word automatically choose option File->External Editor:
  - Project -> Tool Options



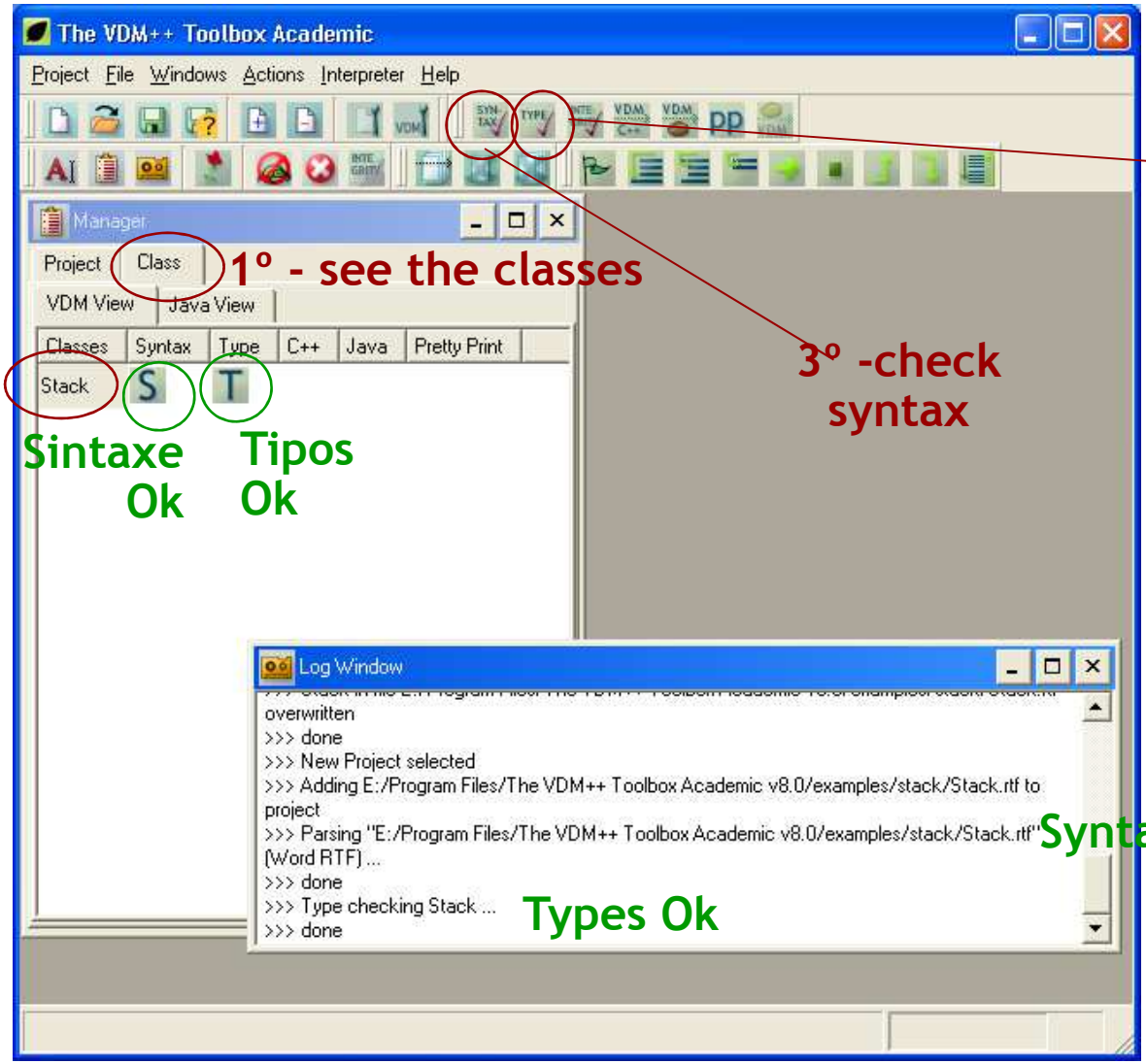
Write a specification (2/3)



Add a file to a project (2/4)



Check syntax and types



1º - see the classes

2º select

3º -check syntax

4º -check types

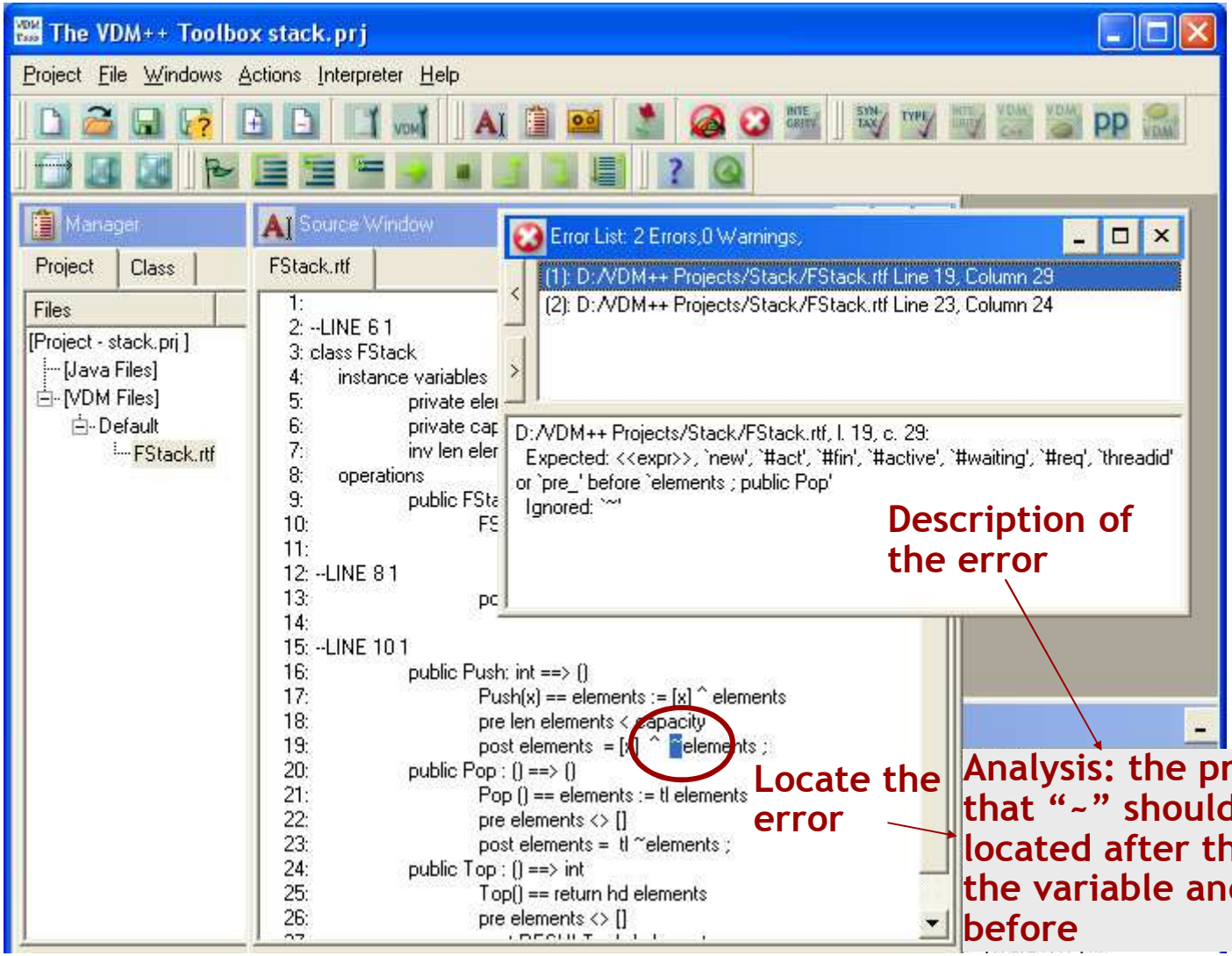
Sintaxe Ok

Tipos Ok

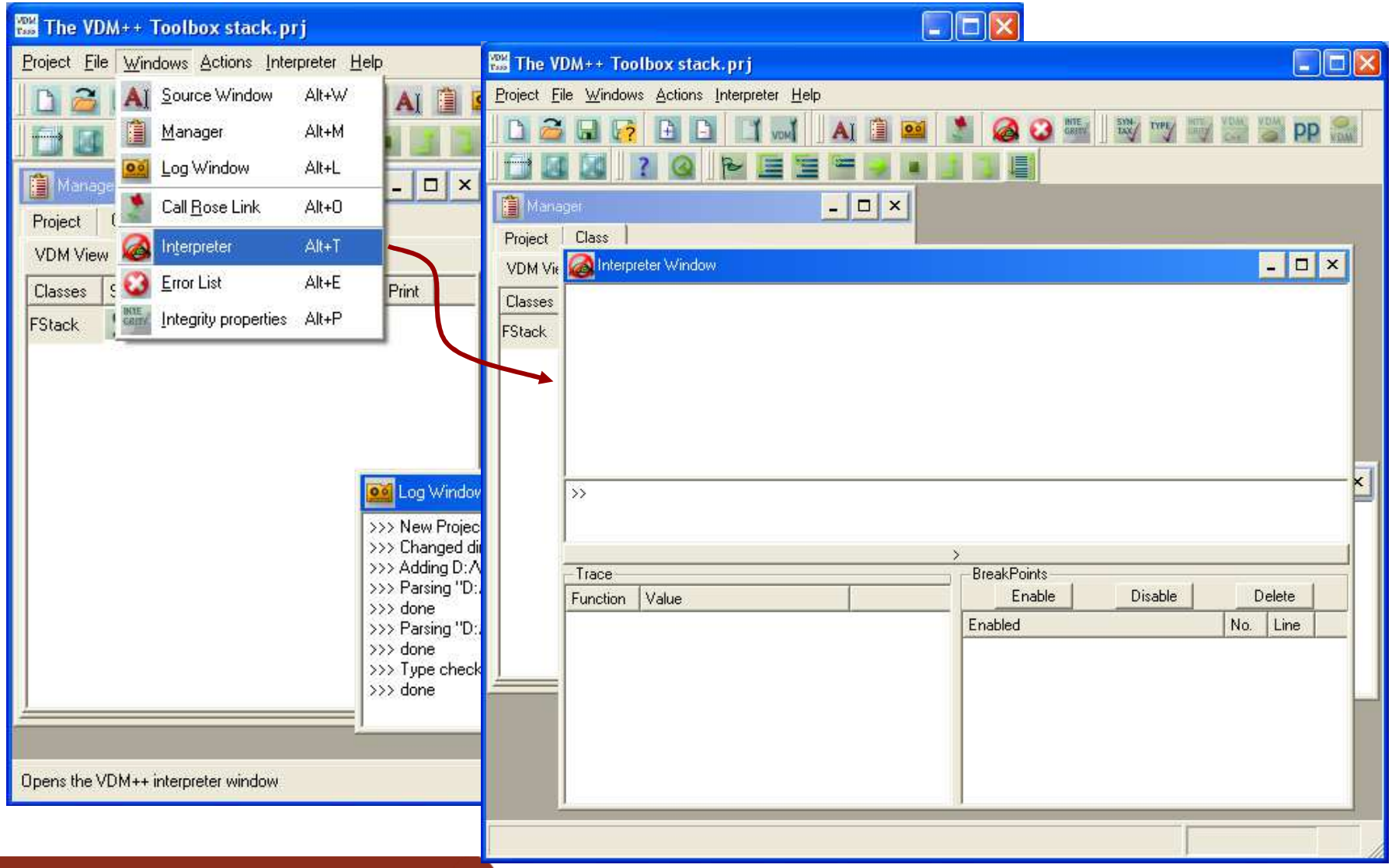
Syntax Ok

Types Ok

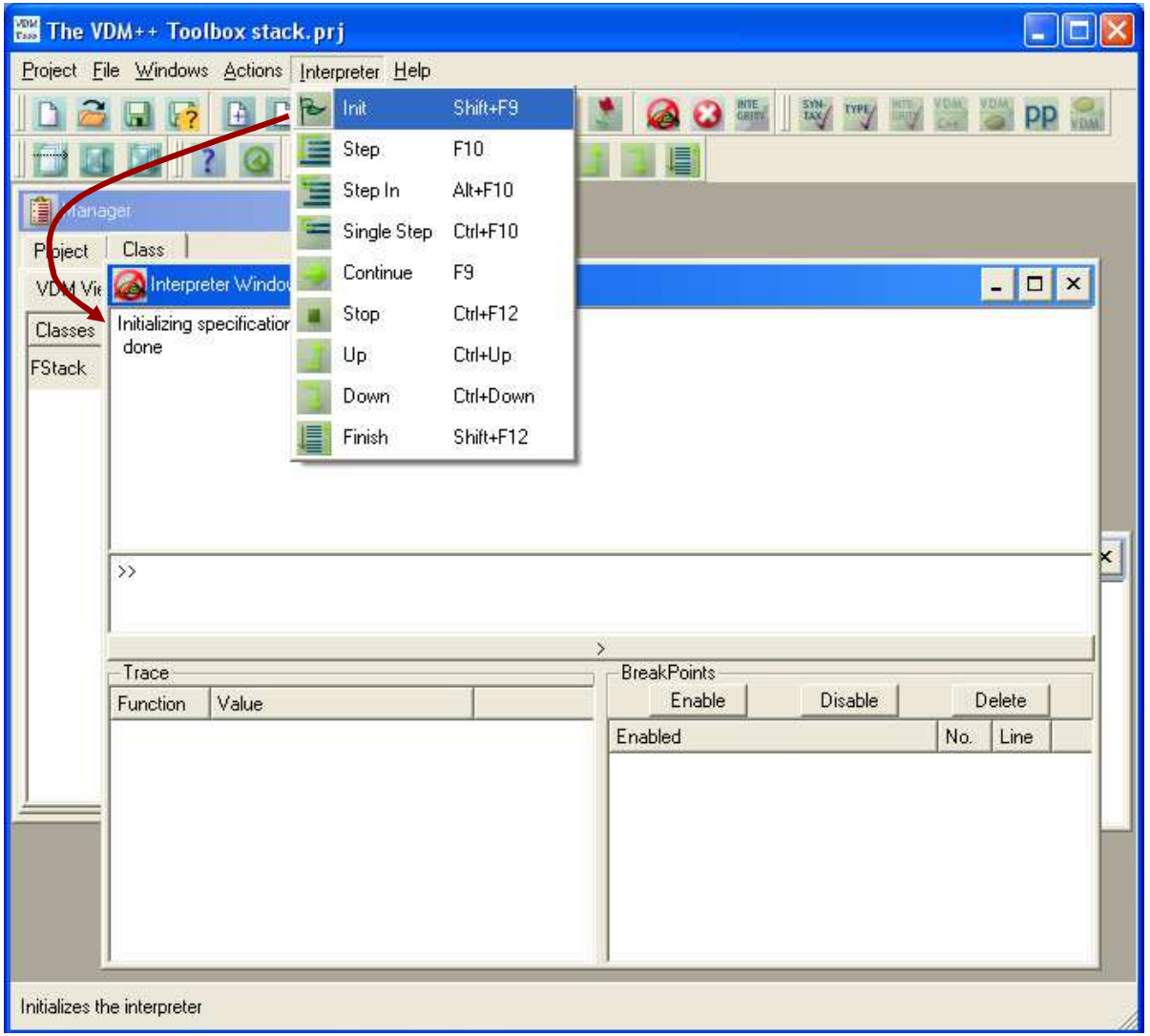
Errors


The VDM++ Toolbox stack.prj
Project File Windows Actions Interpreter Help
Manager
Project Class
Files
[Project - stack.prj]
[Java Files]
[VDM Files]
Default
FStack.rtf
Source Window
FStack.rtf
Error List: 2 Errors,0 Warnings,
(1): D:/VDM++ Projects/Stack/FStack.rtf Line 19, Column 29
(2): D:/VDM++ Projects/Stack/FStack.rtf Line 23, Column 24
D:/VDM++ Projects/Stack/FStack.rtf, l. 19, c. 29:
Expected: <<expr>>, `new', `#act', `#fin', `#active', `#waiting', `#req', `threadid'
or `pre_' before `elements ; public Pop'
Ignored: `~'
1:
2: --LINE 6 1
3: class FStack
4: instance variables
5: private ele
6: private cap
7: inv len eler
8: operations
9: public FSta
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 <> []
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

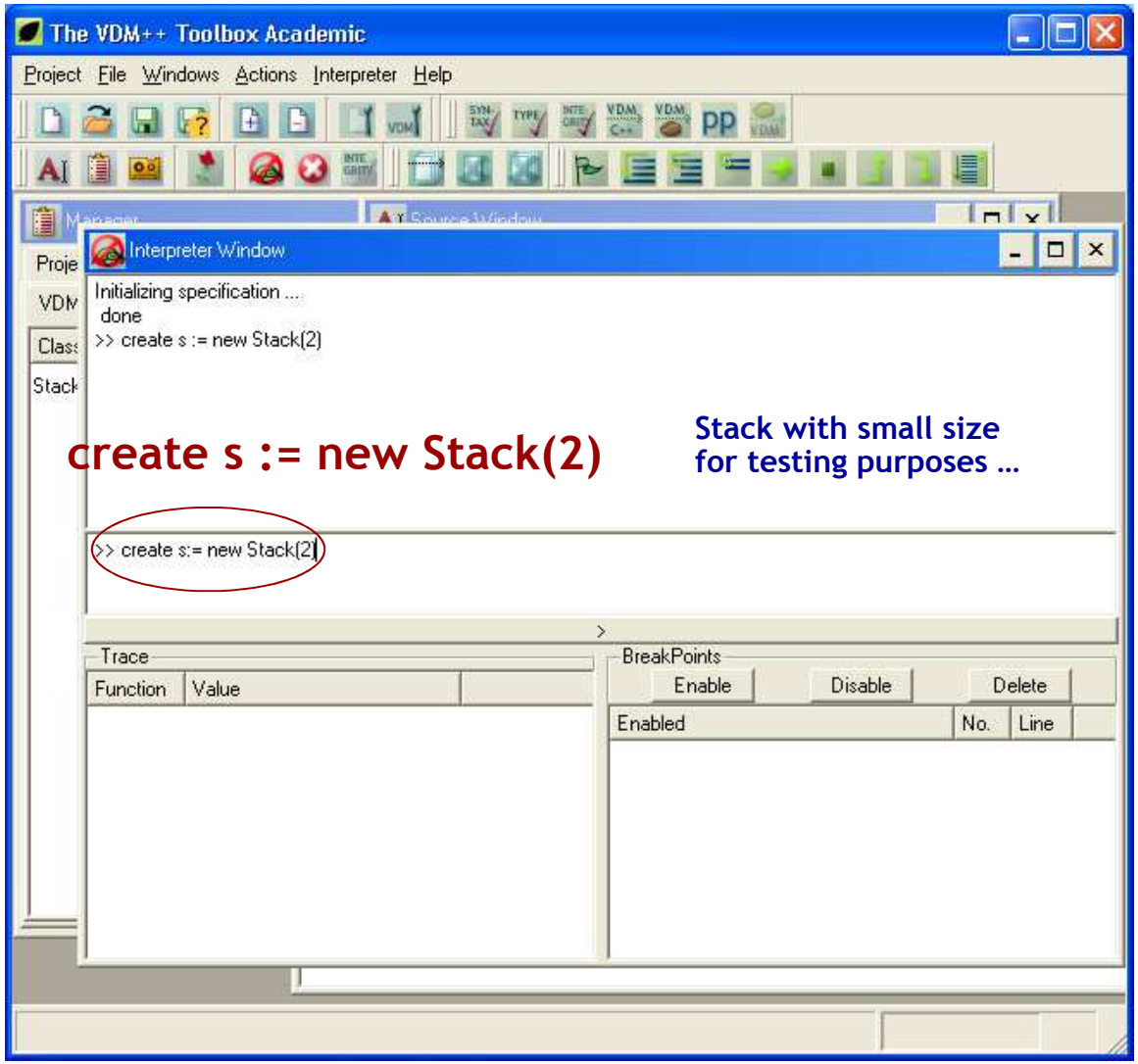
Run the interpreter - open



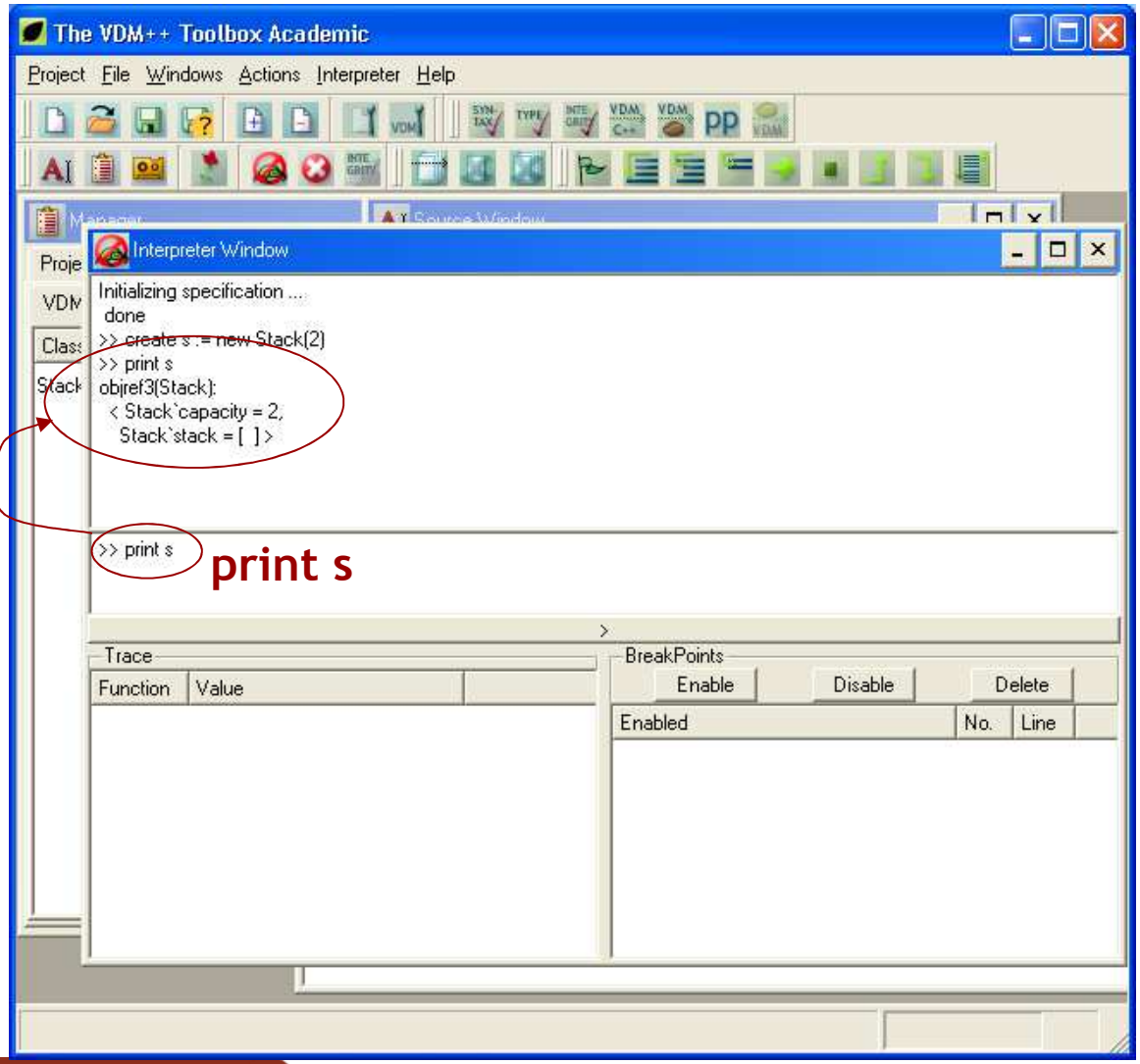
Run the interpreter - start



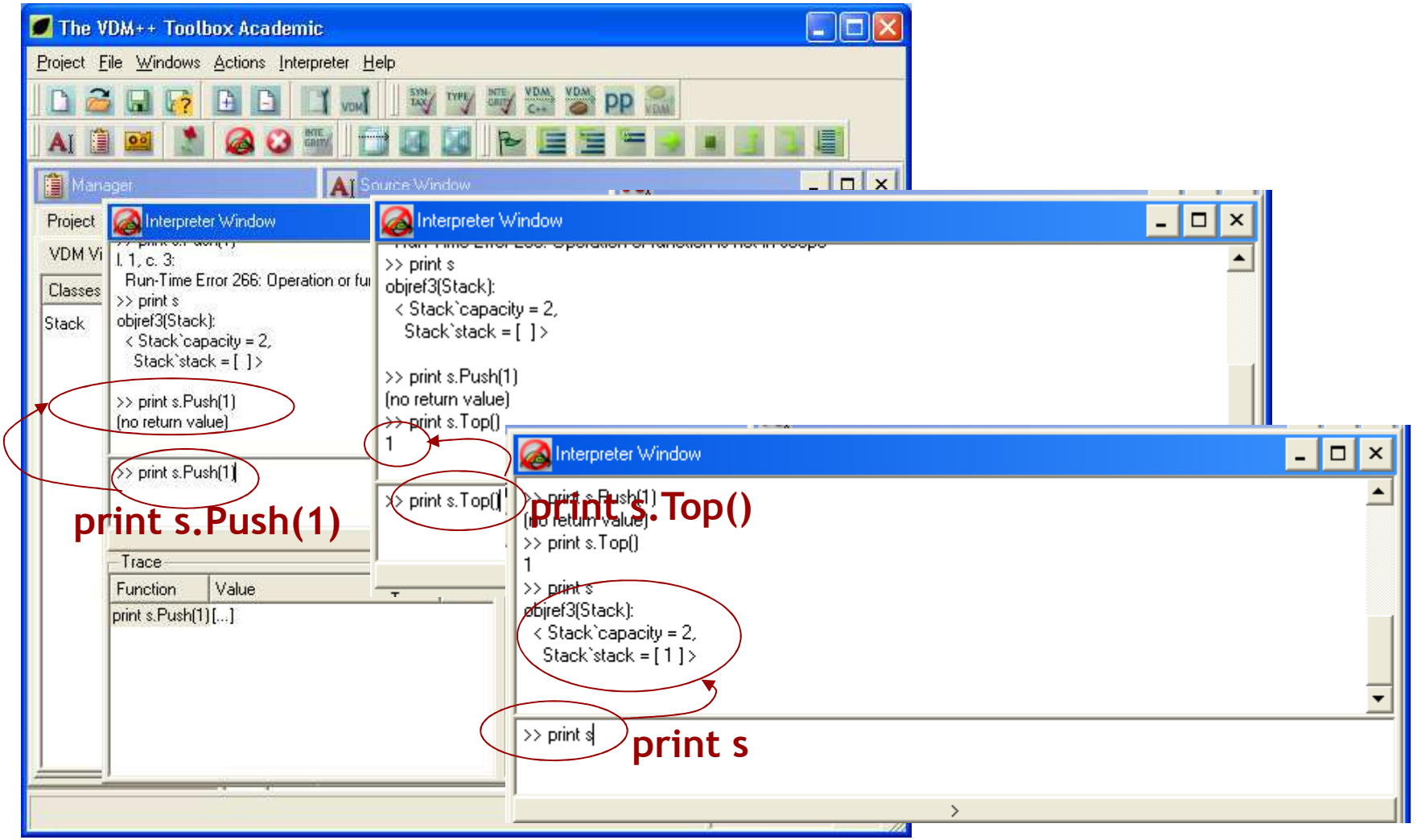
Run the interpreter – create na object



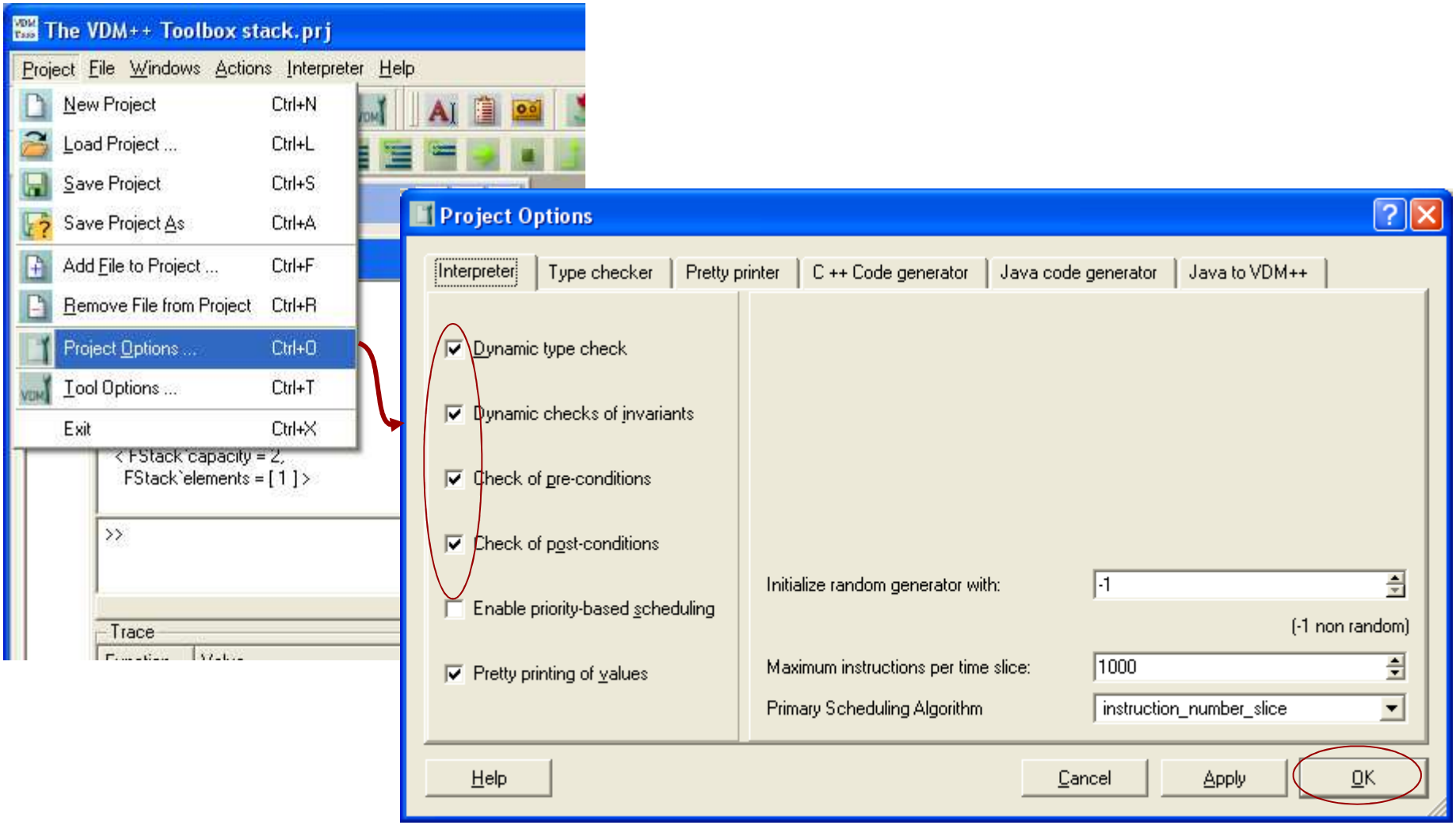
Run the interpreter – state of an object



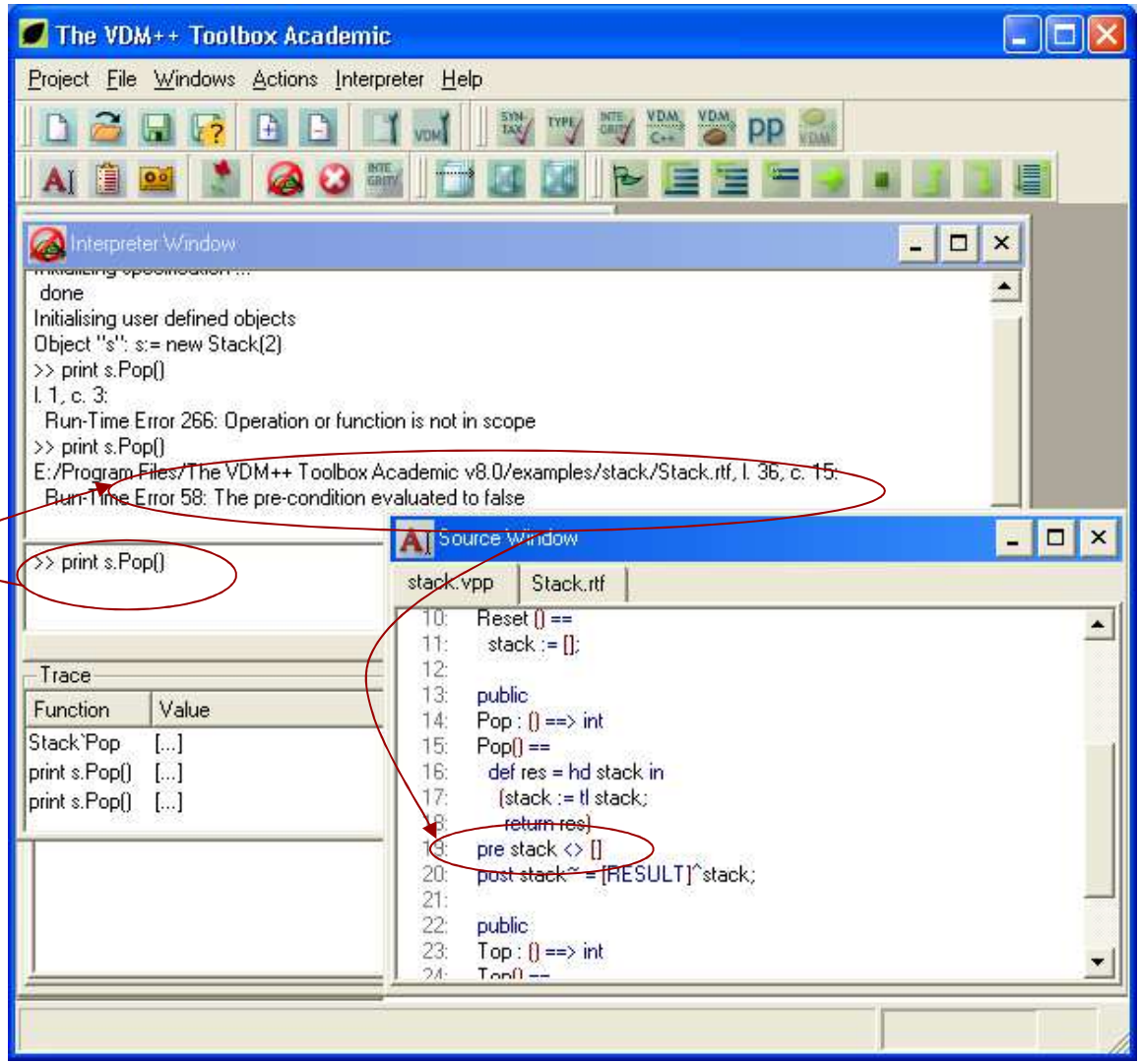
Run the interpreter – state of an object



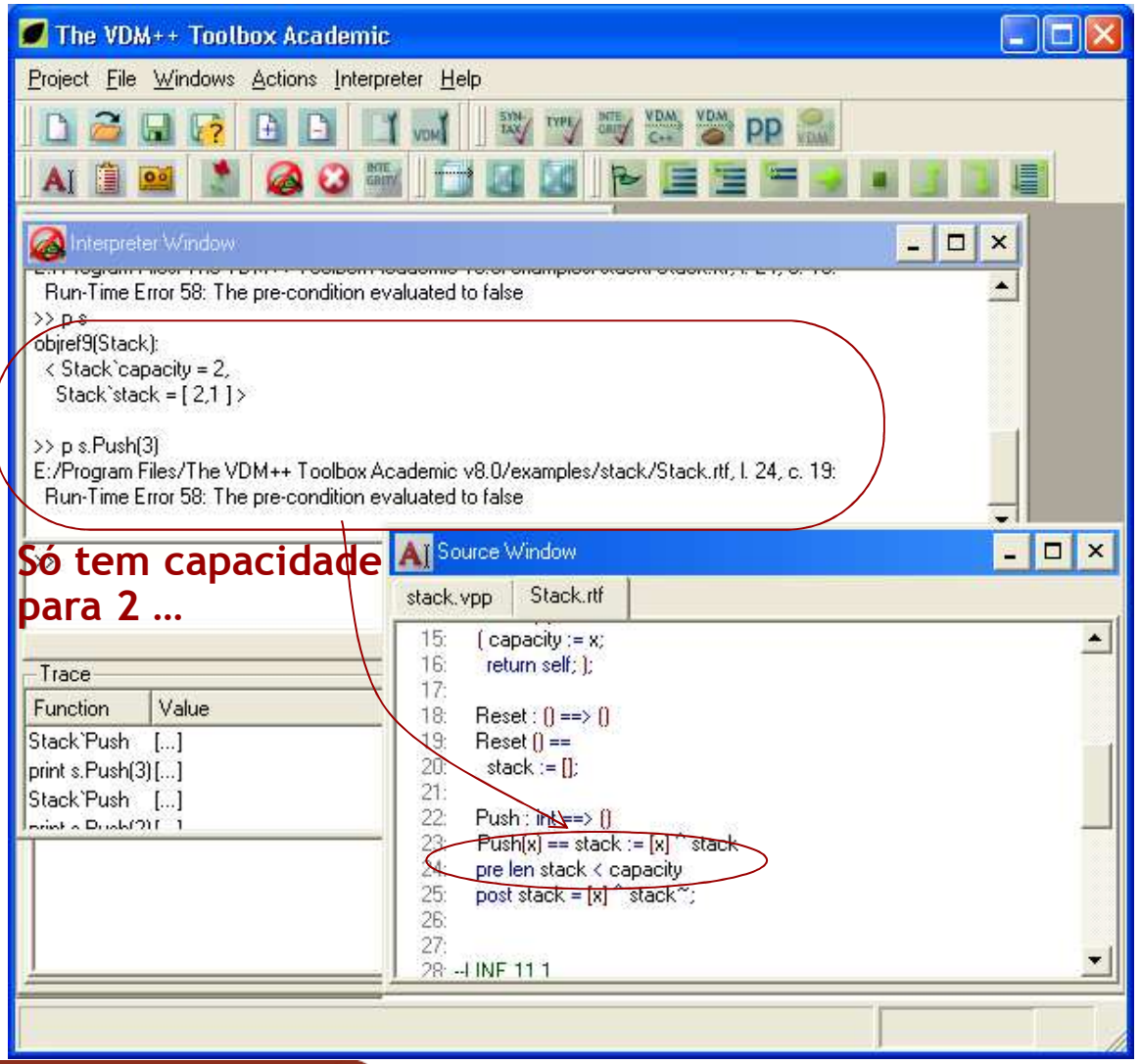
Check invariants, pre and post conditions
Activate checks



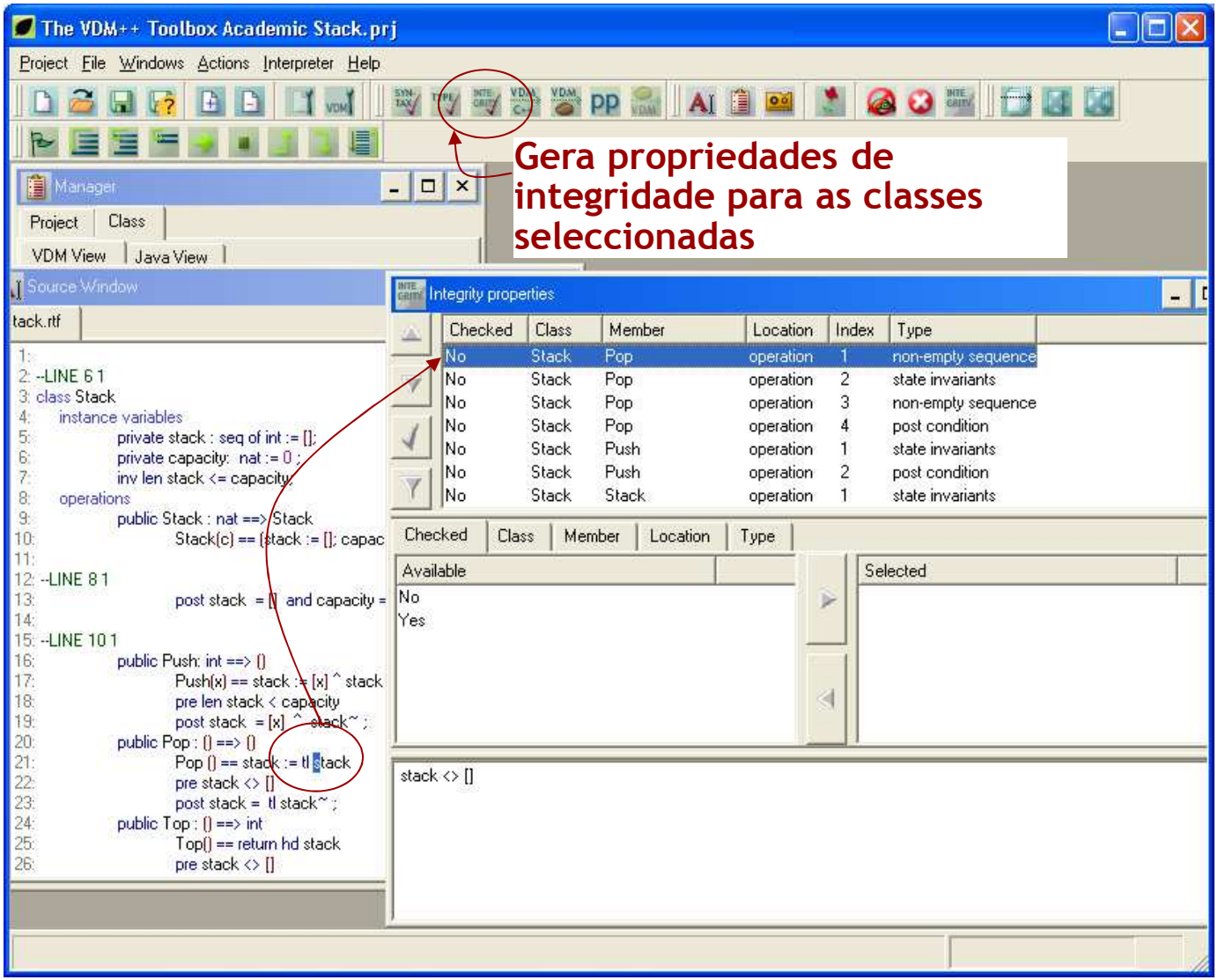
Check invariants, pre and post conditions
Pre condition violation



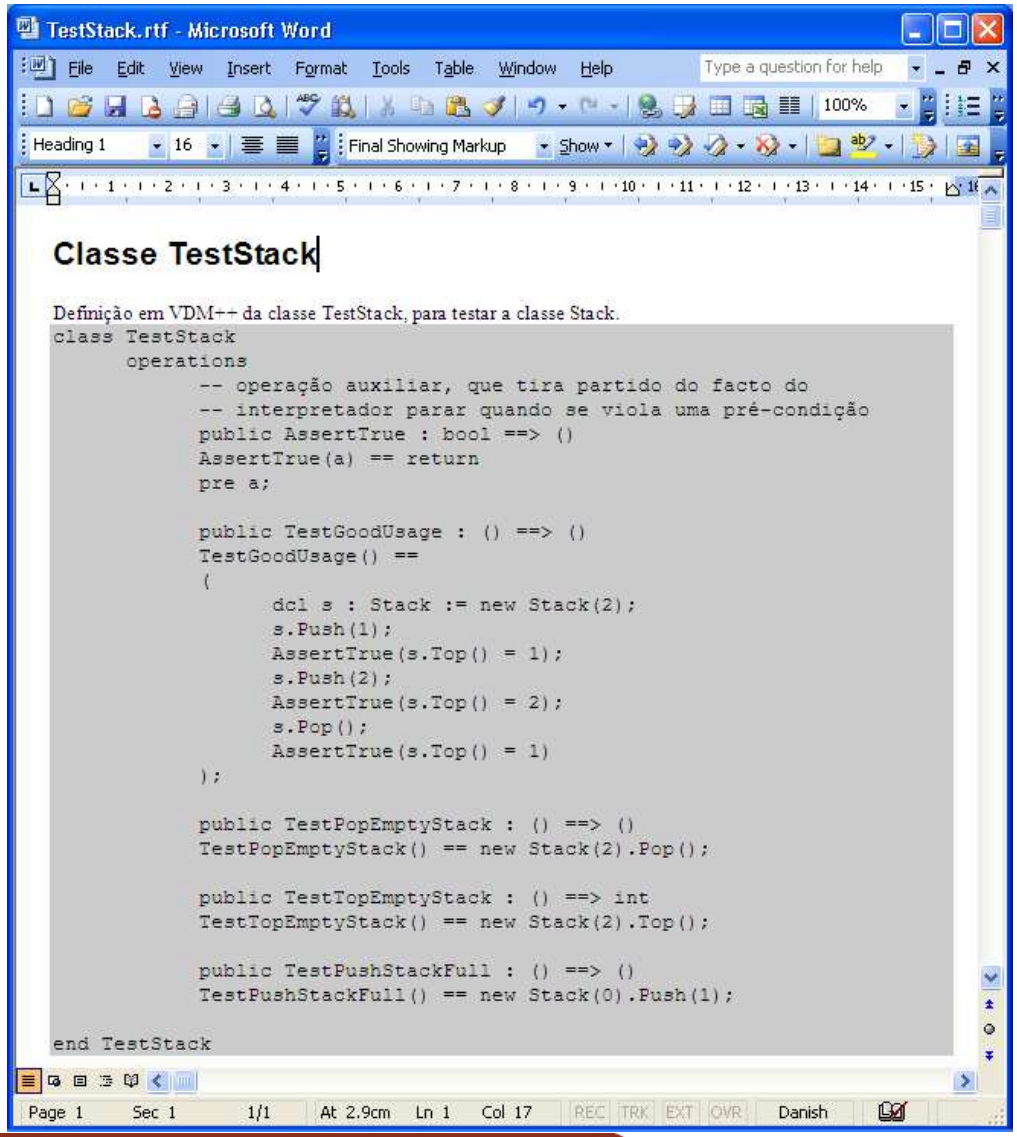
Check invariants, pre and post conditions
Pre condition violation



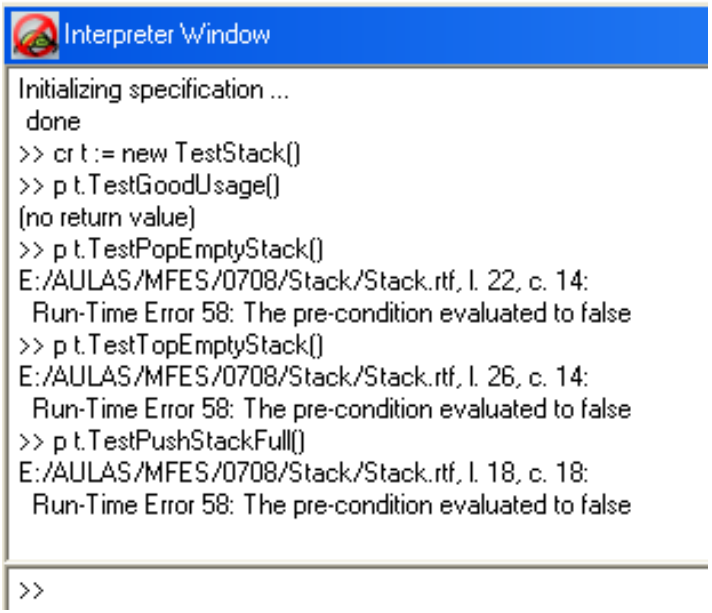
Consistency of the specification



Test cases into a test class

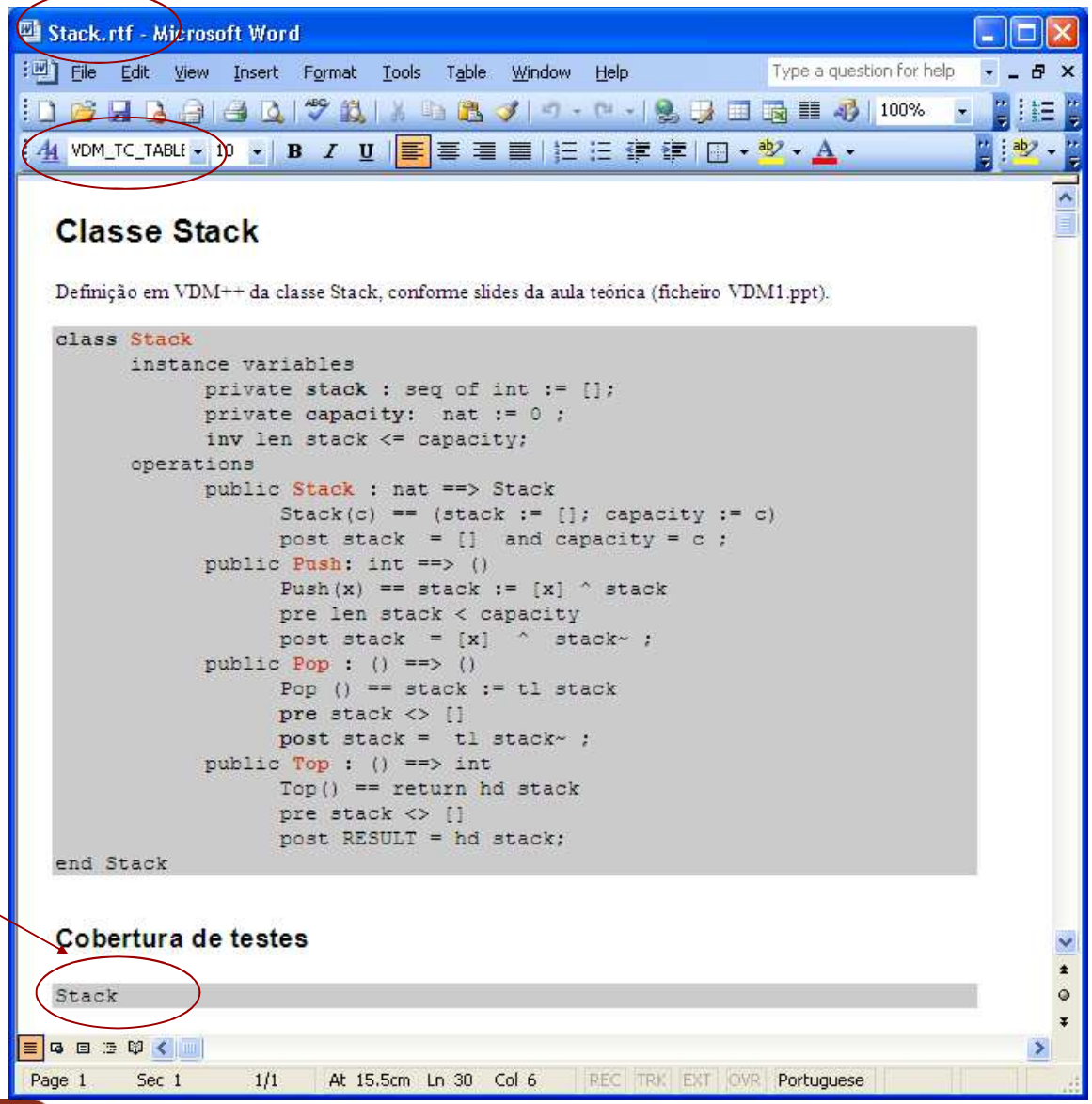


1. Create the file
2. Add to the project
3. Check syntax
4. Check types
5. Run the interpreter
6. Run the following commands:



Coverage analysis

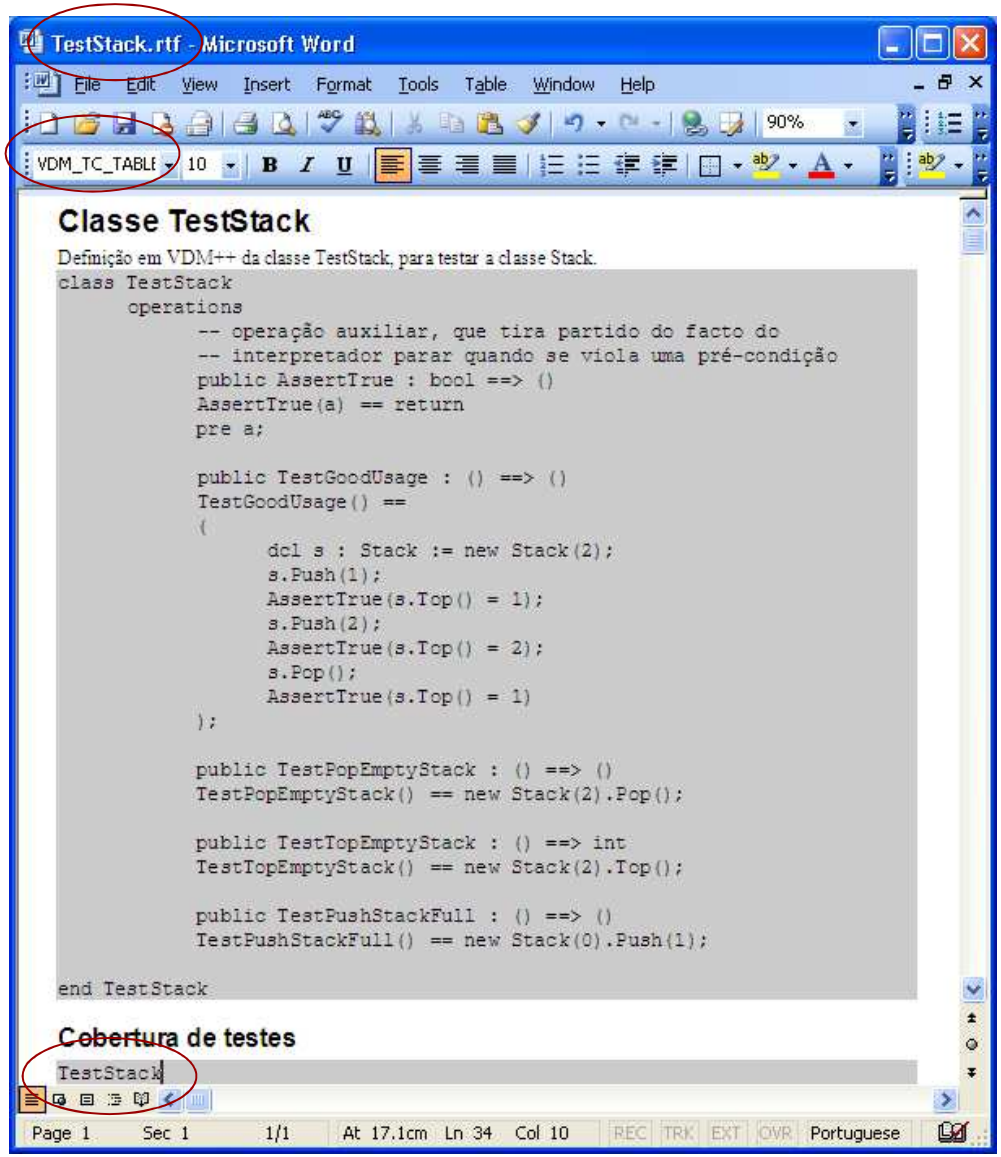
- 1º - Put *placeholders* for coverage analysis tables



Name of the class, style VDM_TC_TABLE.

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

Coverage analysis



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

Output expected

Etc.

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\vppde" -p -R vdm.tc %S1% %S2%
for /R %%f in (*.arg) do call vdmtest "%%f"
```

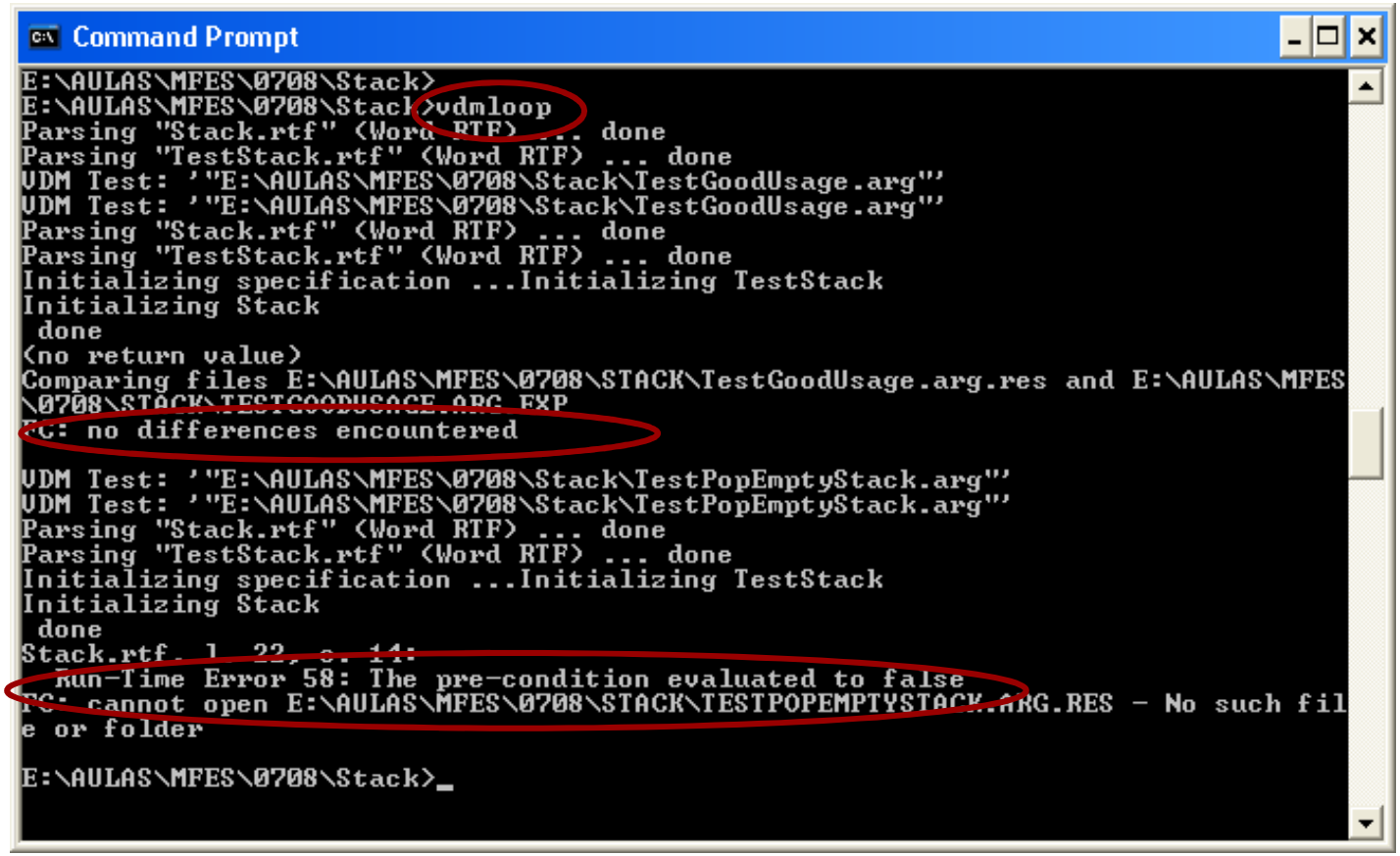
vdmtest.bat - Notepad

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\vppde" -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



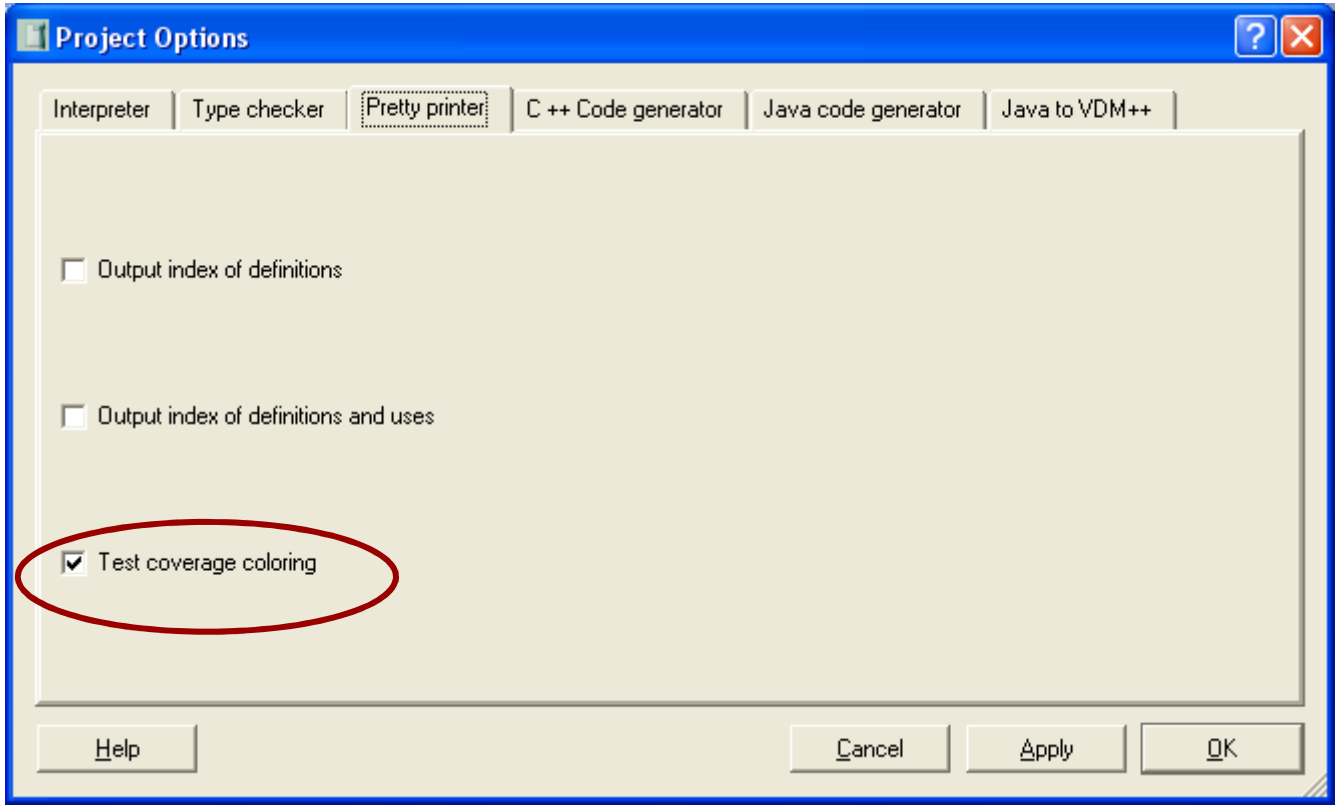
OK

As expected

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

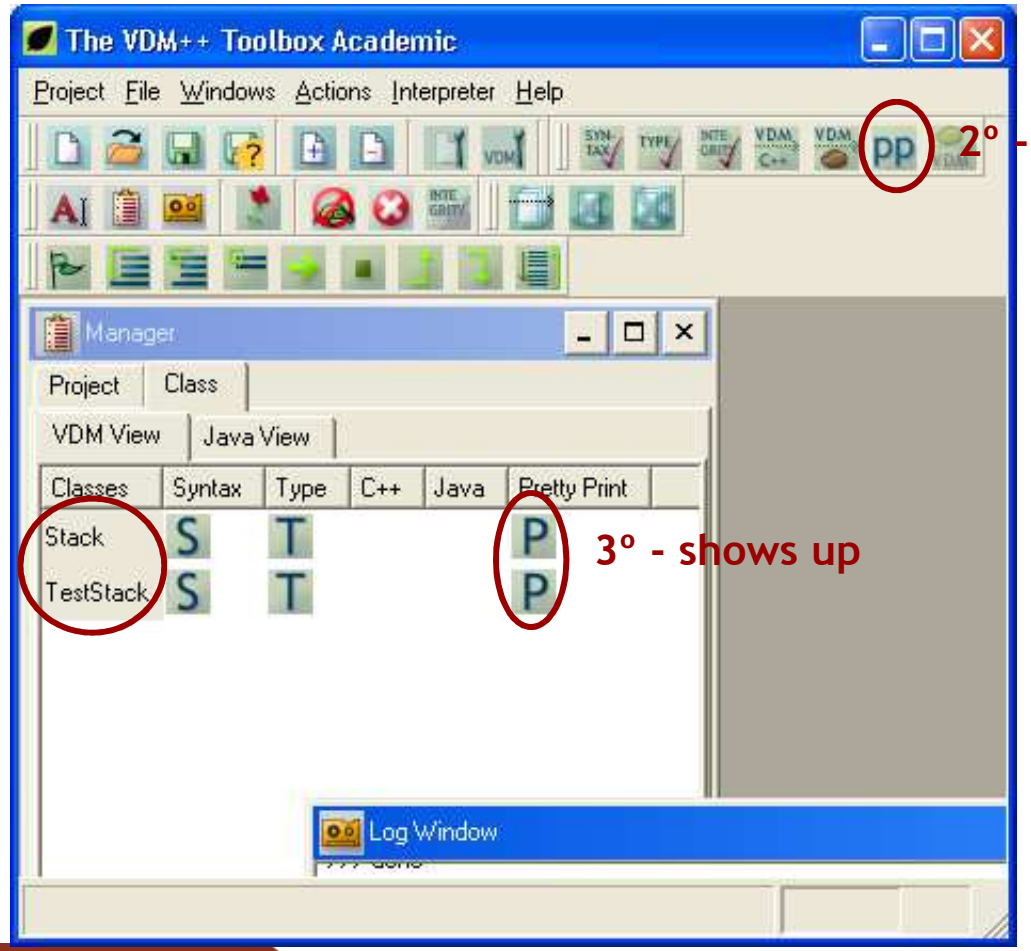
Coverage analysis

- 5º - Prepare “pretty printer” configuration



Coverage analysis

- 6º - Run “pretty print”



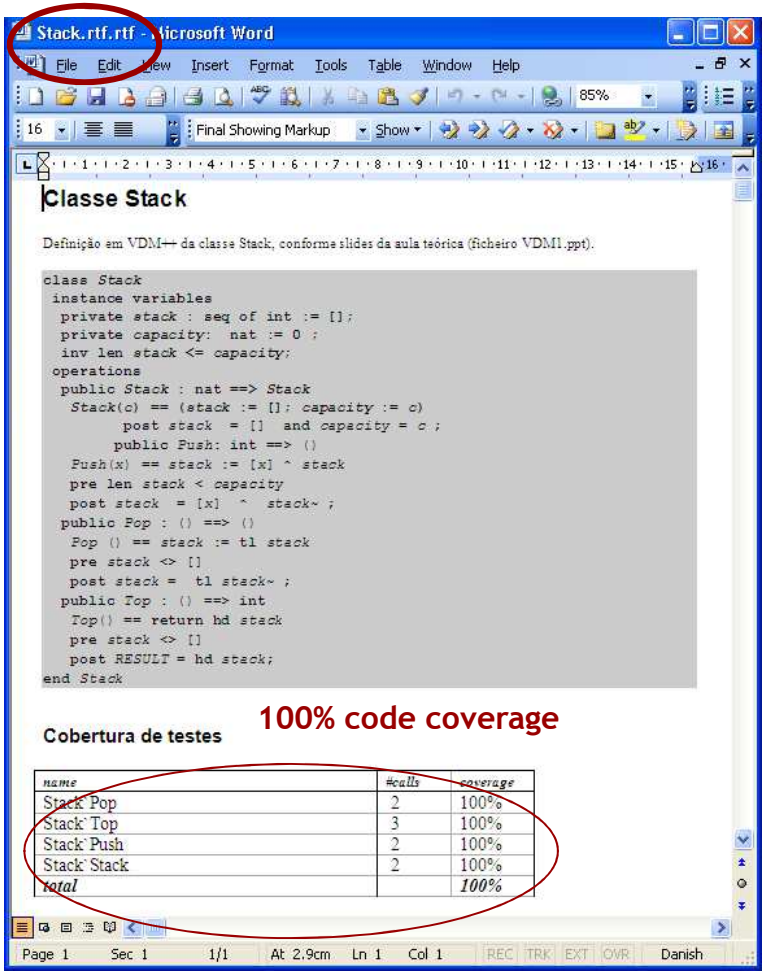
2º - pretty print

1º - select

3º - shows up

Coverage analysis

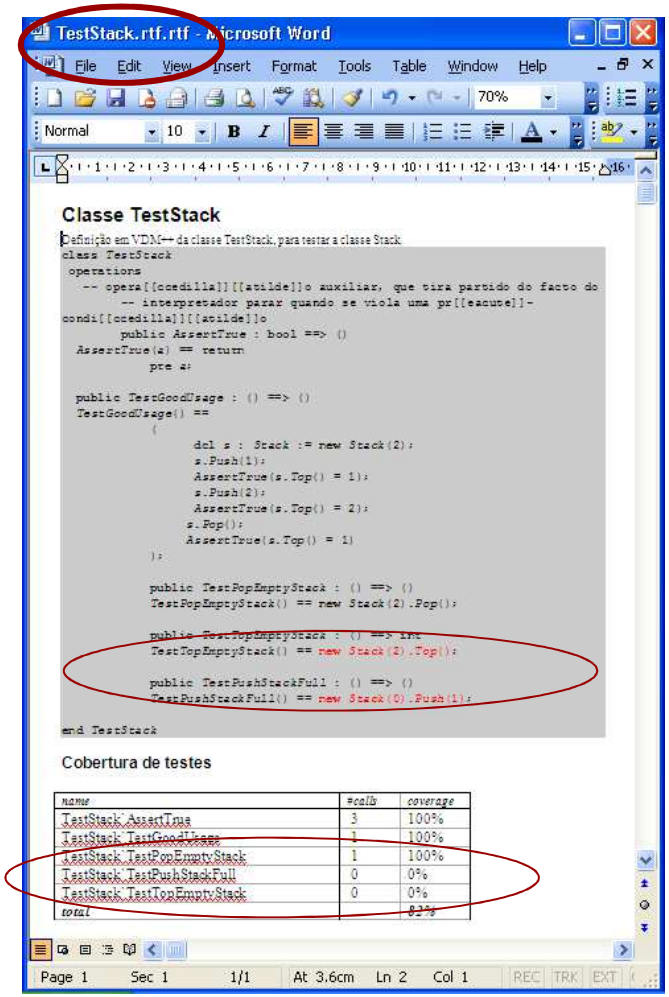
- 7º - Analyse the results (files .rtf.rtf !)



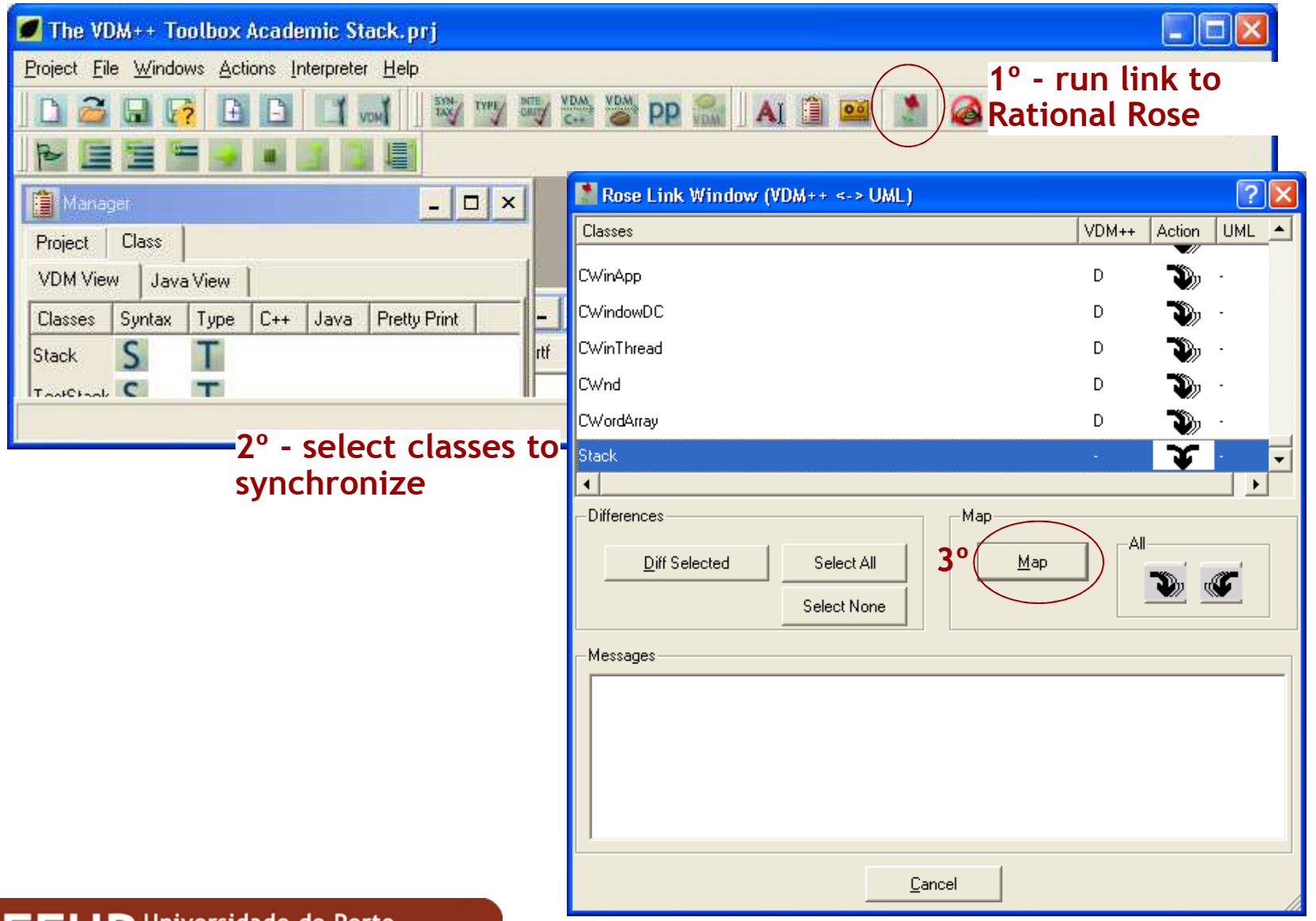
100% code coverage

Not executed code in red

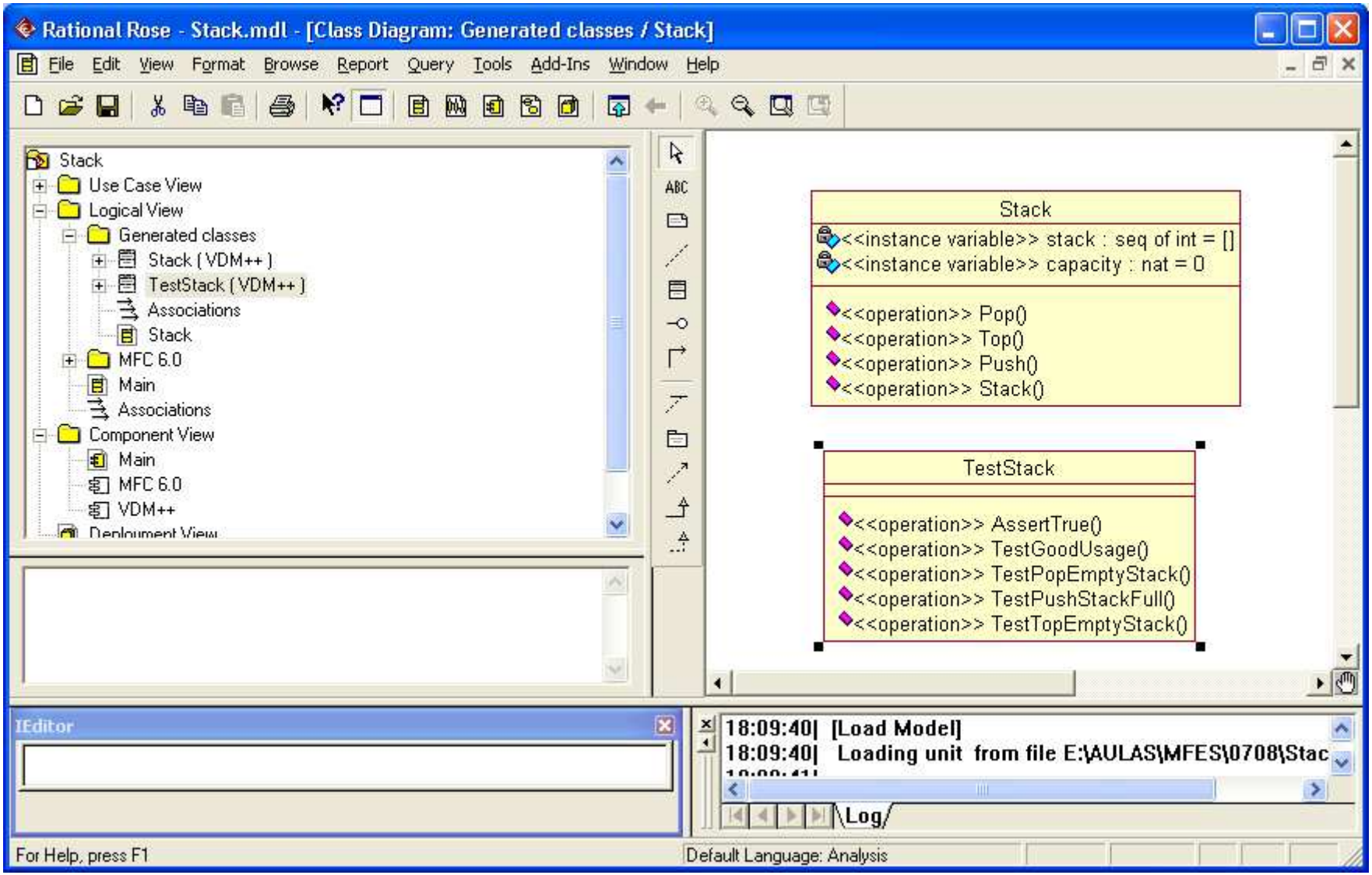
Two oparation that were not executed by test cases



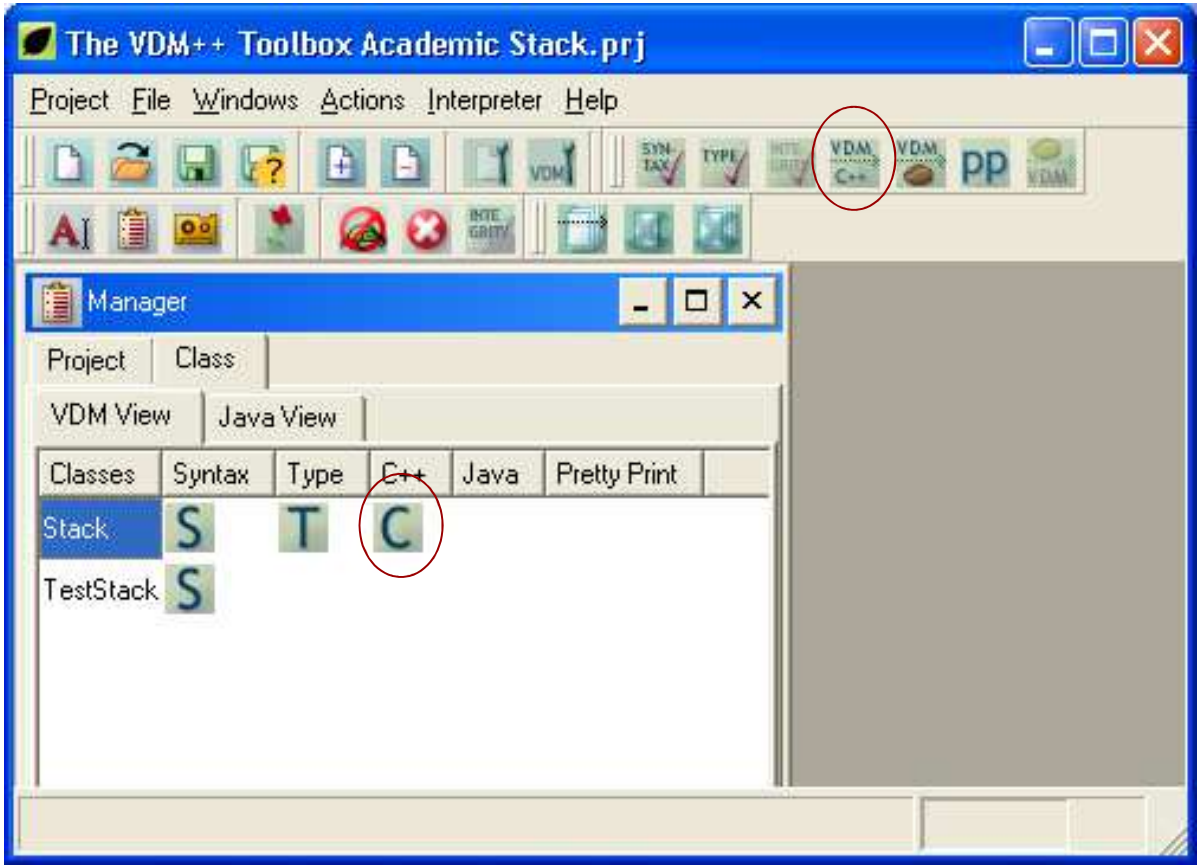
Link to Rational Rose



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

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

- *VDMTools*: User manual

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

- VDMTools: Overture/VDM++

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