MFES – Formal Methods in Software Engineering
Overture Quick Start Exercise

1. Download the most recent version (2.4.2) of the Overture tool from http://overturetool.org/, unzip the installation file and launch the Overture executable. The Overture tool is based on Eclipse, so you should get the familiar Eclipse environment.

2. Create a new project named Stack, in File -> New -> Project …-> VDM-PP Project -> Project name: Stack -> Finish. VDM-PP stands for VDM++.

3. Inside that project, create a class named Stack. To do that, press the right mouse button over the project folder in the VDM Explorer view, and select New -> VDM-PP Class -> File name: Stack -> Finish.

4. Open the file "Stack.vdmpp" in the editor and insert the following code.

```
class Stack
        instance variables
                private contents : seq of int := [];
                private capacity:  nat;
                inv len contents <= capacity;

        operations
                public Stack: nat ==> Stack
                Stack(c) == (capacity := c; return self)
                post contents = [] and capacity = c;
                public clear: () ==> ()
                clear() == contents := [];
                public push: int ==> ()
                push(x) == contents := [x] ^ contents
                pre len contents < capacity
                post contents = [x] ^ contents~;
                public pop: () ==> ()
                pop() == contents := tl contents
                pre contents <> []
                post contents = tl contents~;
                public top: () ==> int
                top() == return hd contents
                pre contents <> []
                post RESULT = hd contents;
end Stack
```

5. Experiment introducing some errors in the code. By default, the option "Project/Build Automatically" is enabled, so you should get error messages. Restore the contents of the file to the correct contents.

6. [Using the interpreter] With the mouse over the project folder, select Run As -> Run Configurations … -> Launch Mode: Console -> Run. In the Console view you should get a prompt similar to:

```
**
** Welcome to the Overture Interactive Console
**
>
```

By entering “help” you’ll get a list of the possible commands.
Try the following sequence of commands (notice that in some cases we have to use *print* just to invoke an operation without any return value):

```
create s := new Stack(2)
print s
print s.push(1)
print s
print s.top()
print s.push(2)
print s
print s.pop()
print s
print s.pop()
print s
quit
```

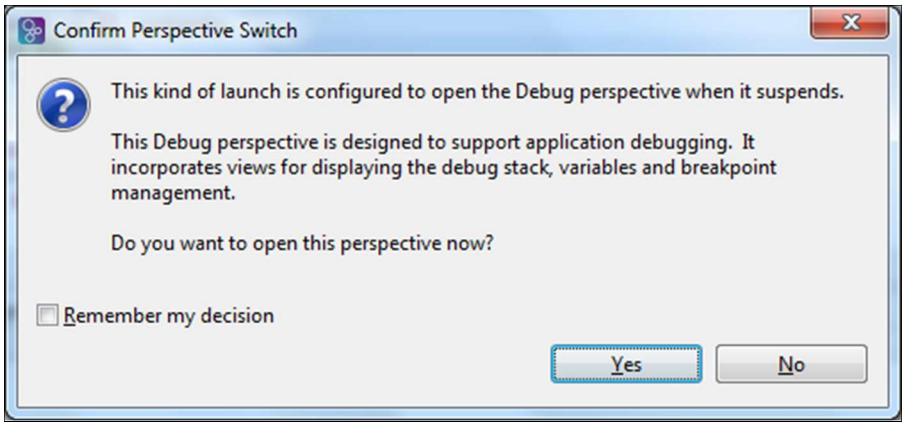
7. [Using the debugger] Run again in the same mode as in the previous step, and enter an invalid call as in:

```
create s := new Stack(2)
print s.pop()
```

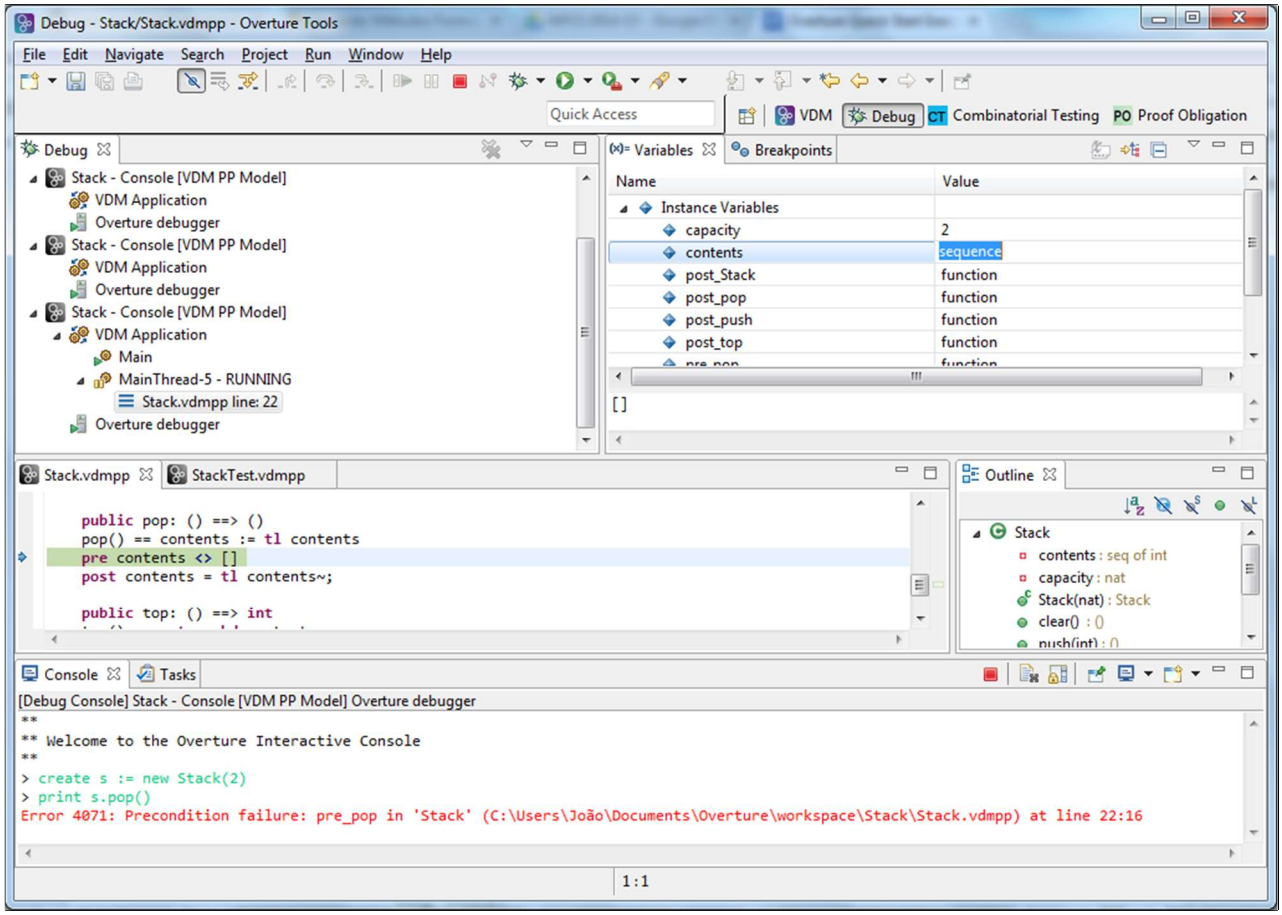
You should get an error message similar to:

```
Error 4071: Precondition failure: pre_pop in 'Stack' (...)
```

and a dialog box suggesting to switch to the Debug perspective:



Select Yes and try to understand the different windows in the Debug perspective. You should be able to inspect the position in the source code where the violation occurs (pre contents <> []), the stack trace, and the contents of the instance variables (by selecting “Stack.vdmpp” in the stack trace), as illustrated below.



8. [Testing1] In the same project, create a test class named StackTest with the following contents:

```
class StackTest
      instance variables
            s : Stack := new Stack(4);
       operations
            private assertTrue: bool ==> ()
            assertTrue(cond) == return
            pre cond;

            private testPushPop: () ==> ()
            testPushPop() ==
            (
                  s.clear();
                  s.push(1);
                  s.push(2);
                  assertTrue(s.top() = 2);
                  s.pop();
                  assertTrue(s.top() = 1);
                  s.pop()
            );
            public static main: () ==> ()
```

```
                main() ==
                (
                        new StackTest().testPushPop();

                );
end StackTest
```

To execute this test code, create a new Launch configuration with Run As -> Launch Mode: Entry Point, Class: StackTest, Function/Operation: main() -> Run. If everything runs Ok, you should get in the Console

```
**
** Overture Console
**
StackTest`main() = ()
```

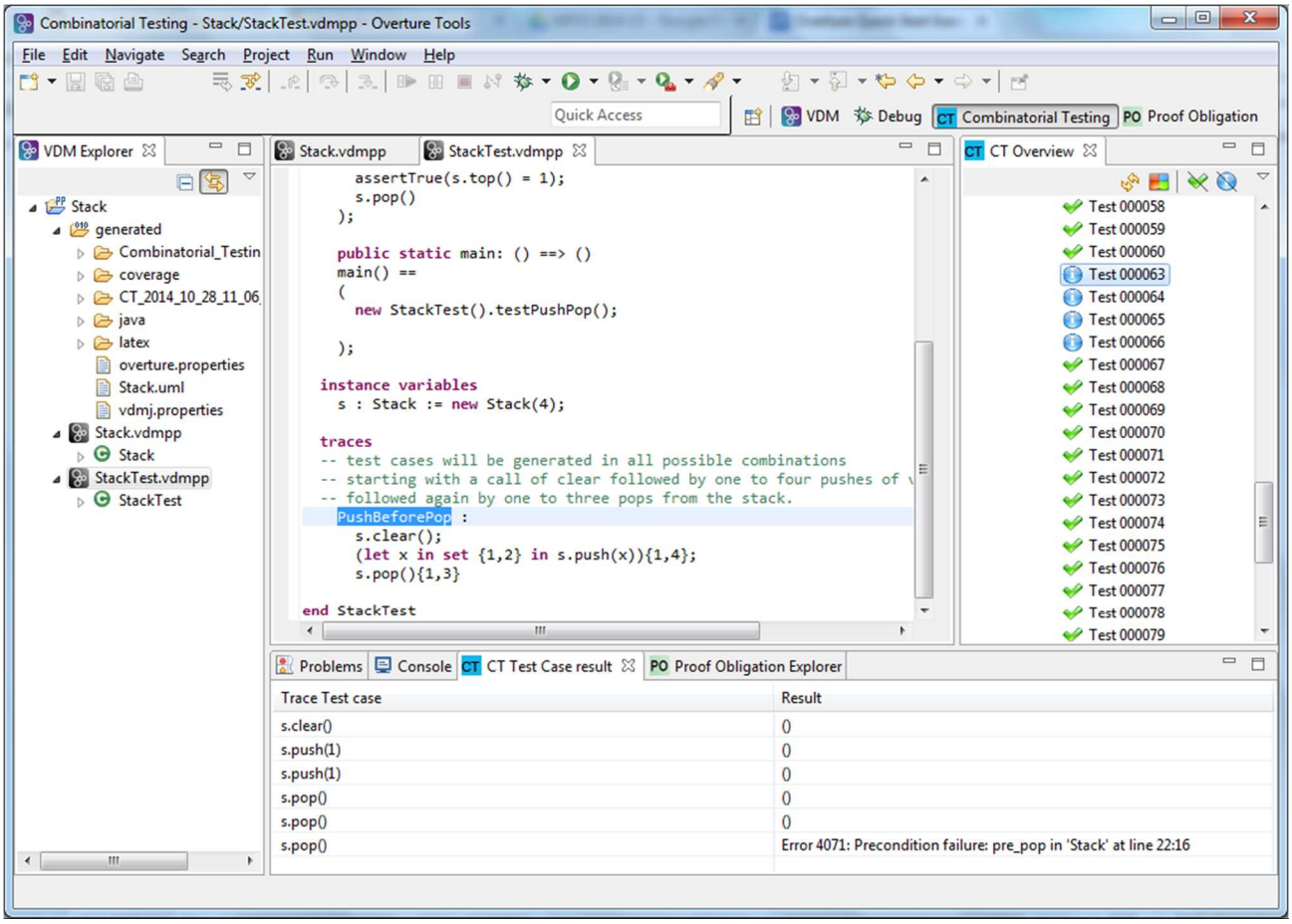
9. [Coverage analysis] After executing the 'main' operation in the previous step, because the option "Generate coverage" is enabled by default in the Launch configuration, coverage information should have been generated in the folder Stack -> generated -> coverage -> (execution time stamp) -> (classname).vdmppcov. Open each of the coverage files to check if everything was covered.

10. [Combinatorial testing] Add the following code to the StackTest class (just before `end StackTest`):

```
traces
-- test cases will be generated in all possible combinations
-- starting with a call of clear followed by one to four pushes of
-- values onto the stack
-- followed again by one to three pops from the stack.

        PushBeforePop :
                s.clear();
                (let x in set {1,2} in s.push(x)){1,4};
                s.pop(){1,3}
```

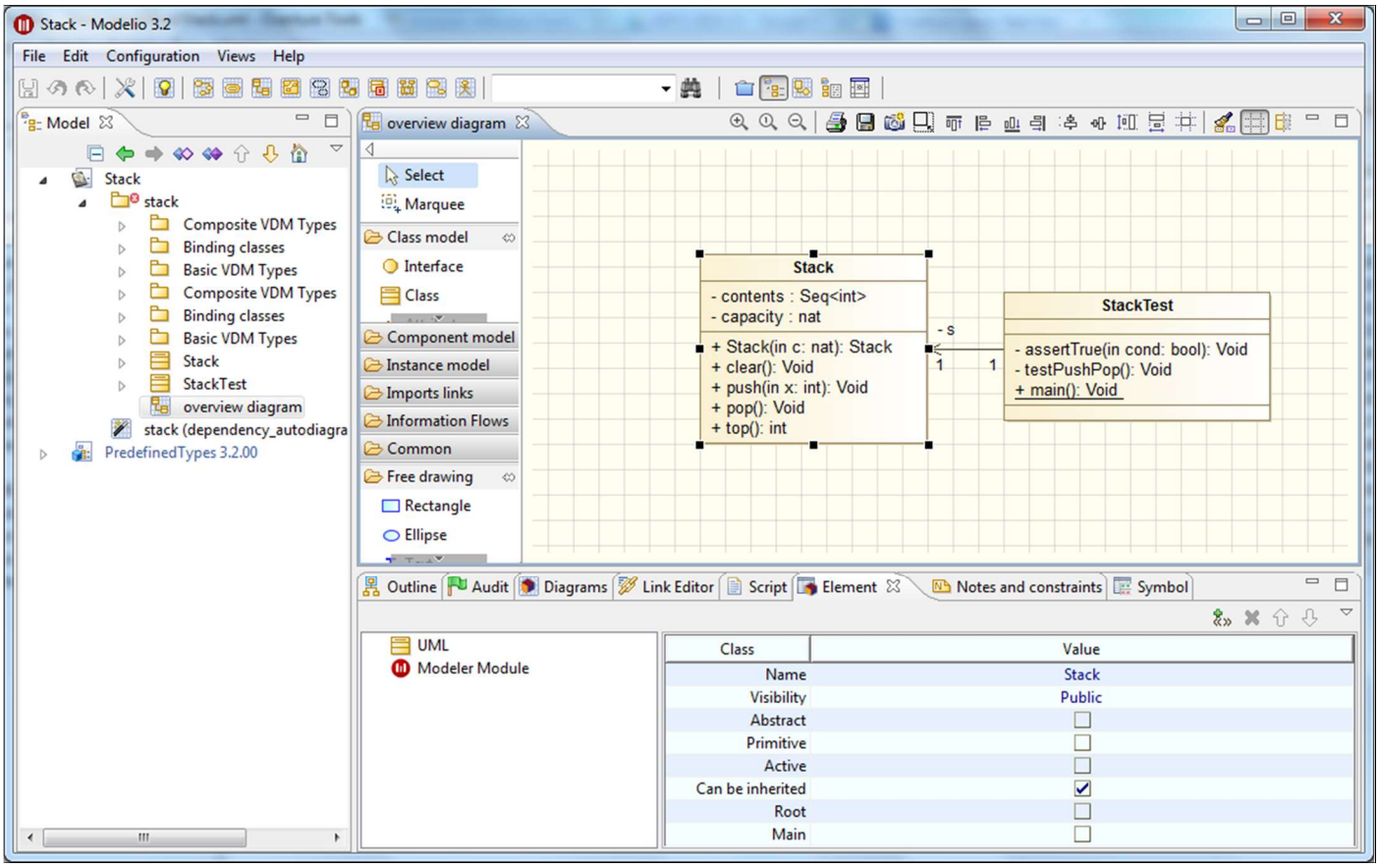
Now open the Combinatorial Testing perspective, select "Refresh" (with the right mouse button over the perspective), expand the tree and select "Full Evaluation" with the mouse over "PushBeforePop". Expand the node to see the test cases generated and the execution results (passed, failed, etc.). Inspect some passed and failed test cases, as illustrated in the following picture.



11. [Documentation generation] In the VDM Explorer window, with the mouse over the project folder, select the option Latex -> PdfLaTex. Latex and PDF documentation files (including coverage information - check Properties - VDM Settings - Latex) should be generated in the folder “Stack/generated/latex”. You may need to have a LaTex editor installed (such as TeXworks) for the PDF file to be properly generated.

12. [Java code generation] In the VDM Explorer window, with the mouse over the project folder, select the option Code Generation-> Generate Java (Configurarion based). A Java project with source files, libraries and settings should be generated in the folder “Stack/generated/java”. Import the project into Eclipse, inspect the code and execute the generated entry point (Main.java).

13. [UML model generation] In the VDM Explorer window, with the mouse over the project folder, select the option UML Transformation > Convert to UML. A file Stack.uml should be generated in the folder “Stack/generated”. The file can be opened with the open source Modelio tool, available in www.modelio.org. Instal the most recent of the tool (3.6.0), create a project in the tool, and import Stack.uml to the modeling project (with XMI->Import XMI). After dragging the appropriate elements (classes and associations) to the overview diagram, and making visible the attributes and operations compartiments, you should be able to get a diagram similar to:



For information on how to perform the reverse transformation (UML to VDM), see the Overture documentation.
Note: Apparently importation works well only with Modelio 3.3.1.

14. Explore the Overture and VDM++ documentation, available in the installation folder, namely:

    OvertureIDEUserGuide.pdf
    VDMPPGuideToOverture.pdf (more information on how to use the tool)