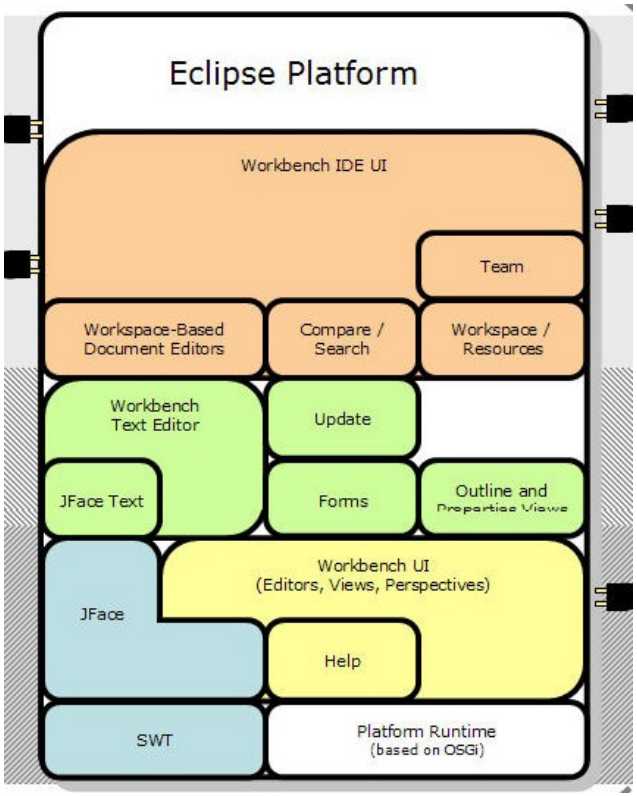
OpenModelica Eclipse Plugin and MetaModelica Exercises

Adrian Pop
adrian.pop@liu.se
PELAB/IDA/LIU, 2007-03-29

Updated 2010-11-03
by Peter Fritzson and Martin Sjölund

OpenModelica MDT – Eclipse Plugin

- Browsing of packages, classes, functions
- Automatic building of executables; separate compilation
- Syntax highlighting
- Code completion,
  Code query support for developers
- Automatic Indentation
- Debugger
  (Prel. version for algorithmic subset)



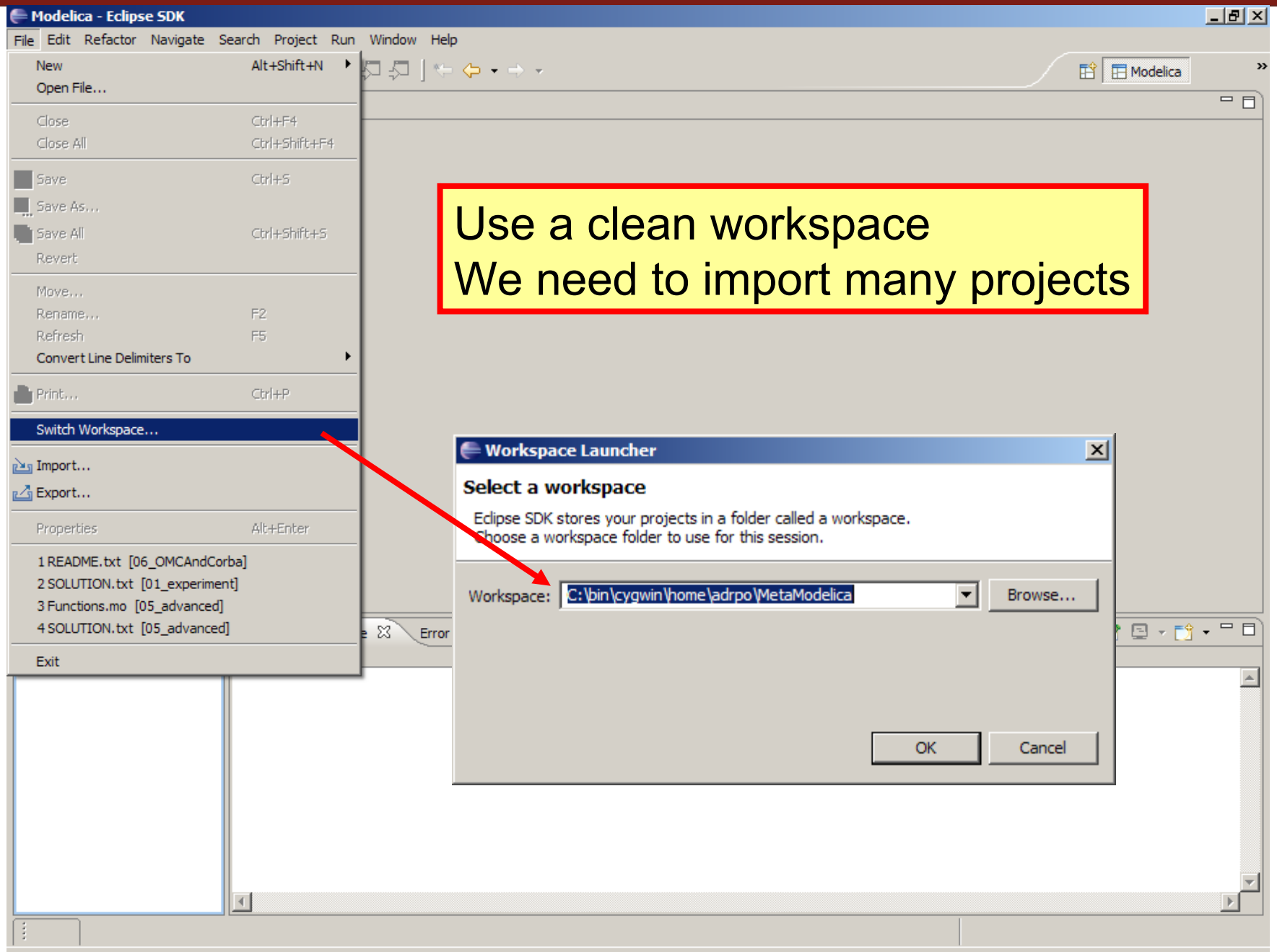
Outline

- Eclipse and Exercise Setup
- All Exercises
  - 00_simplesim
  - 01_experiment
  - 02a_exp1, 02b_exp2
  - 03_symbolicderivative
  - 04_assignment
  - 05a_assigntwotype
  - 05b_modassigntwotype
  - 06_advanced
  - 07_OMCAndCorba
  - 08-11 – as samples of Prog. Lang. modeling

Selected Exercises in this Course

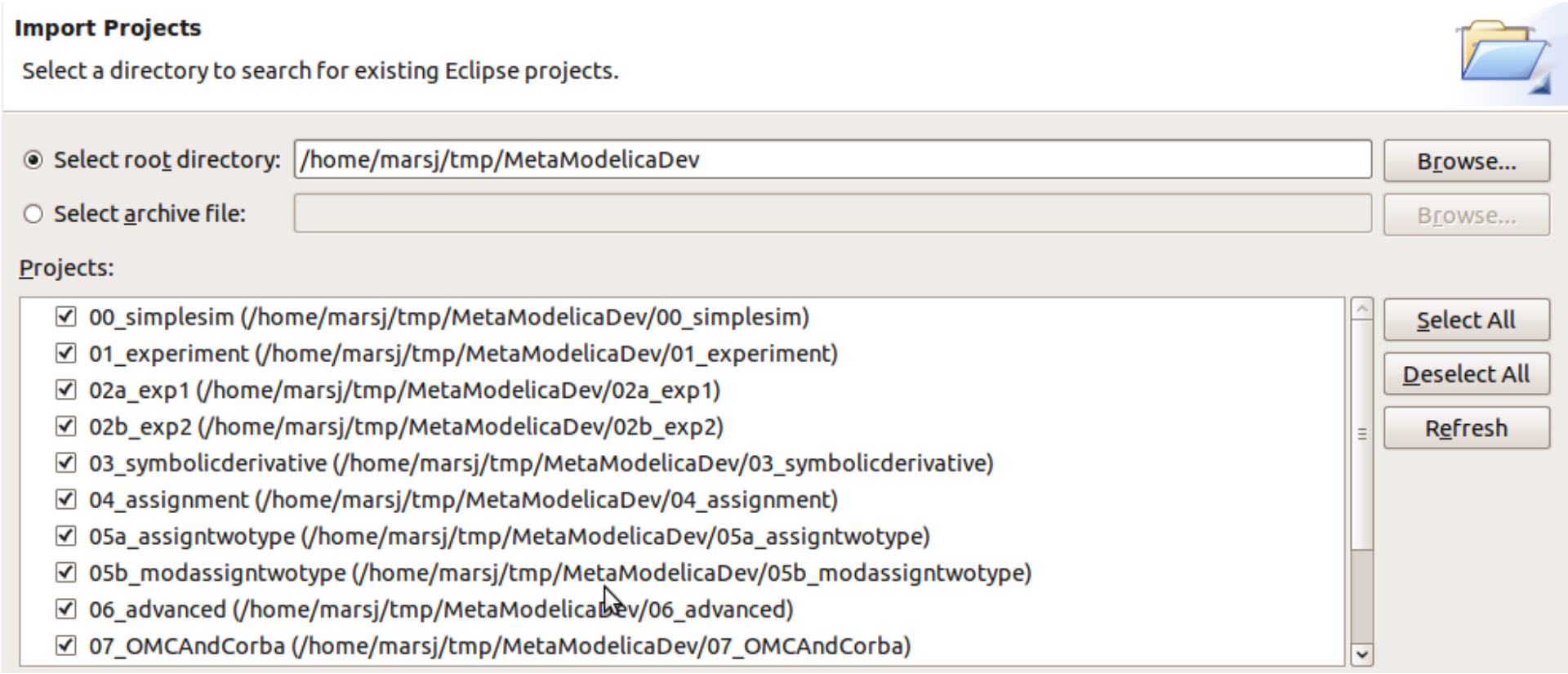
- 00_simplesim
- 01_experiment
- 02a_exp1, (02b_exp2 optional)
- 03_symbolicderivative
- 04_assignment (optional)

Eclipse Setup – Switching Workspace

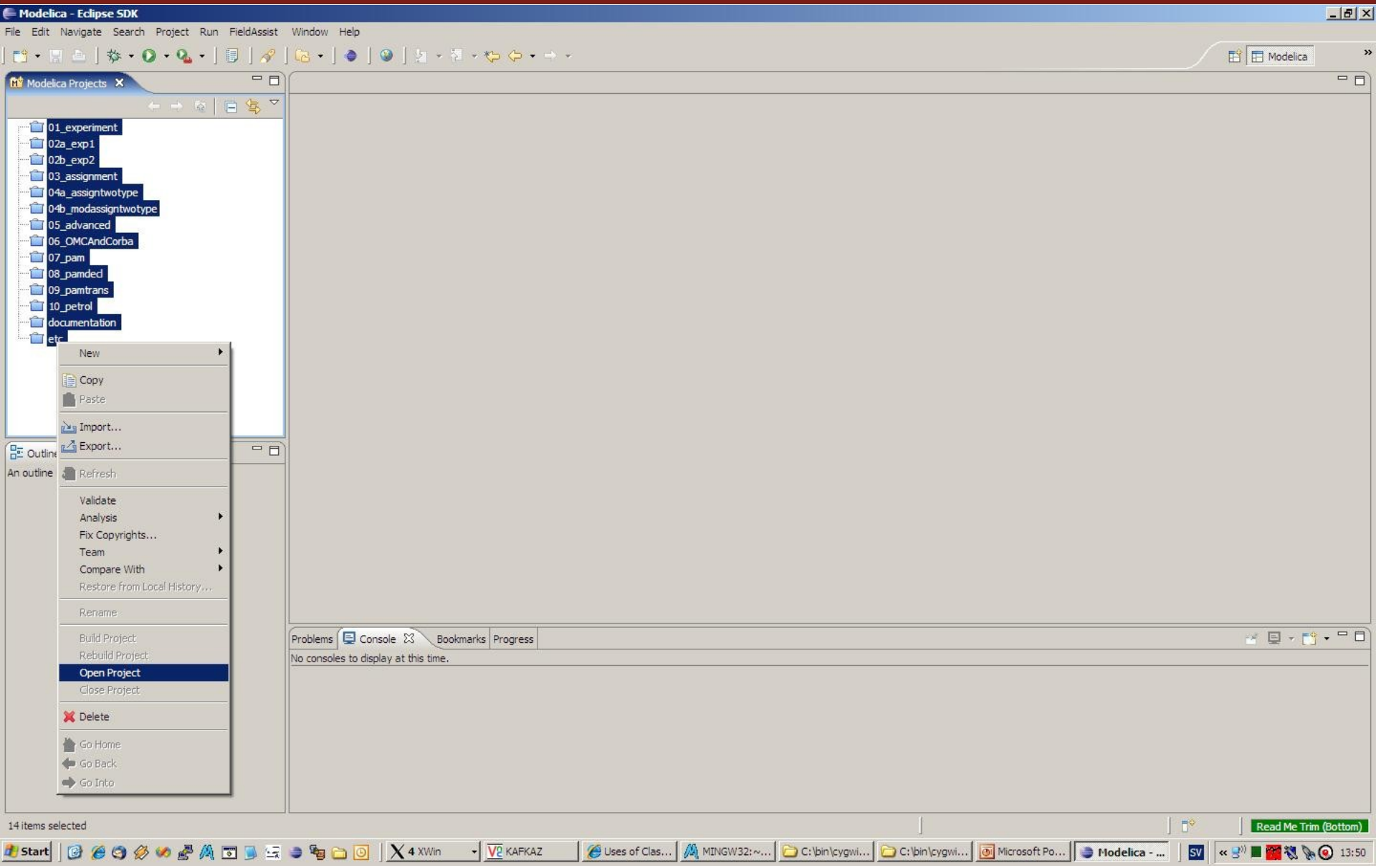


Import Projects

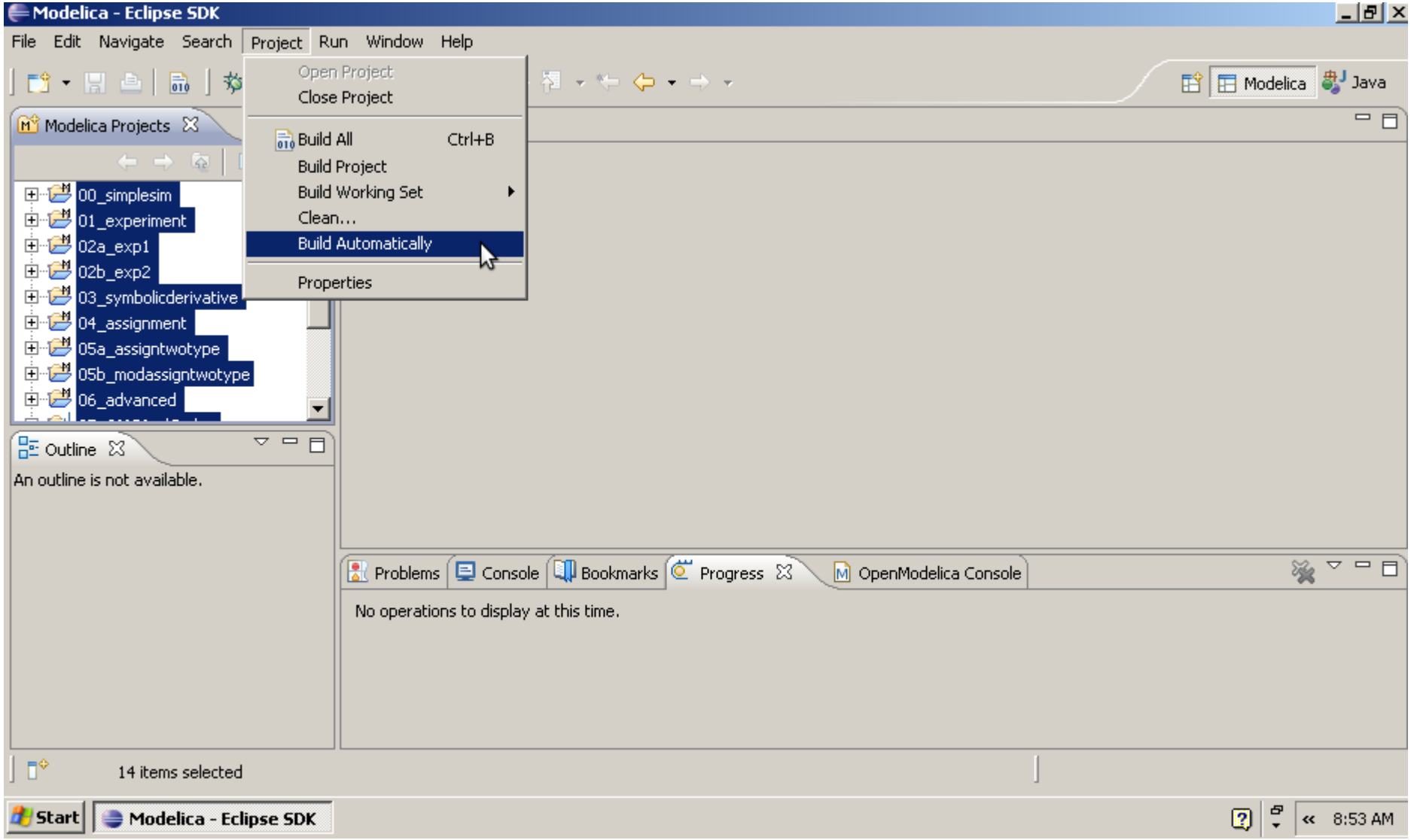
- File → Import
  - General → Existing projects into workspace



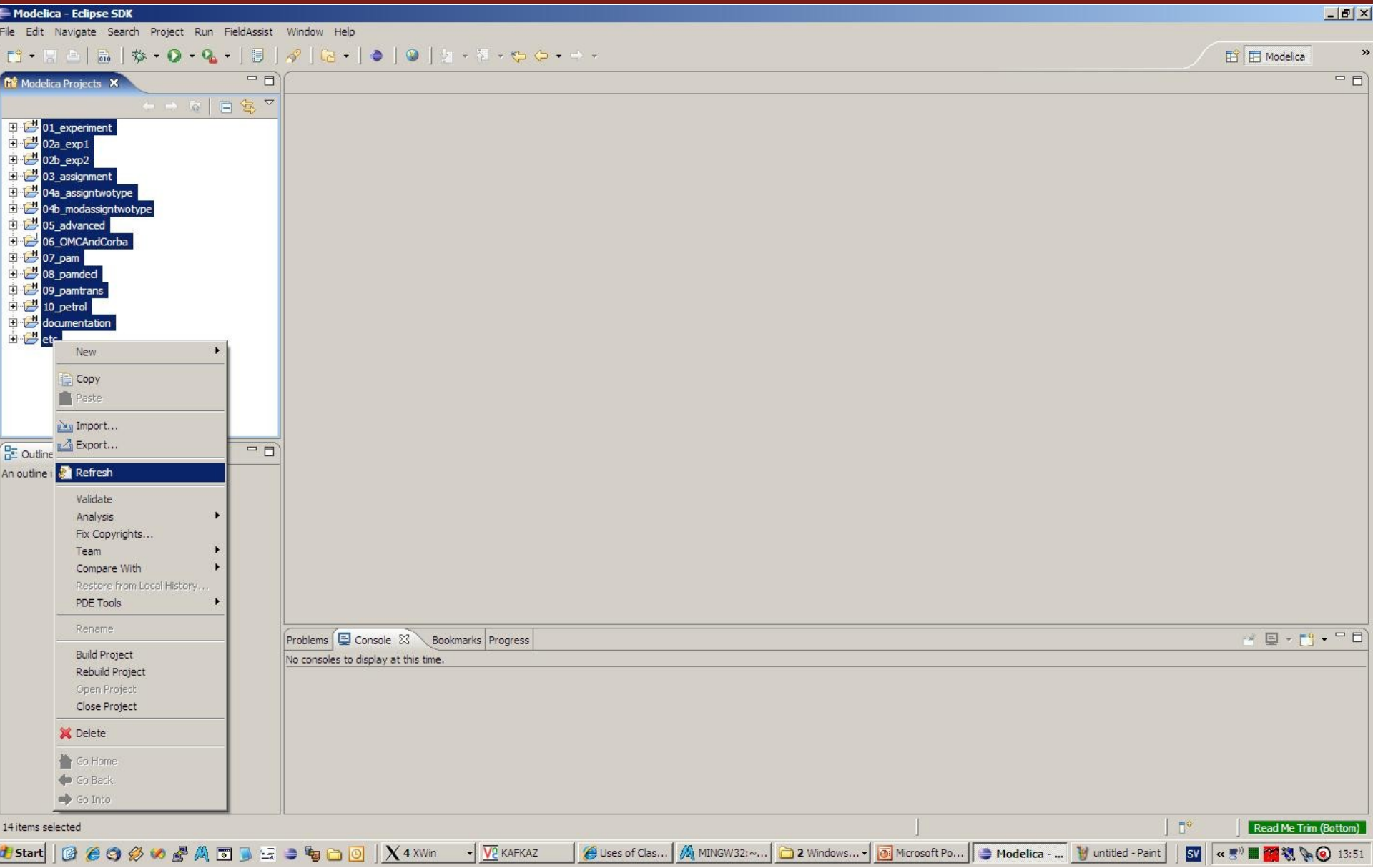
Select and Open all projects



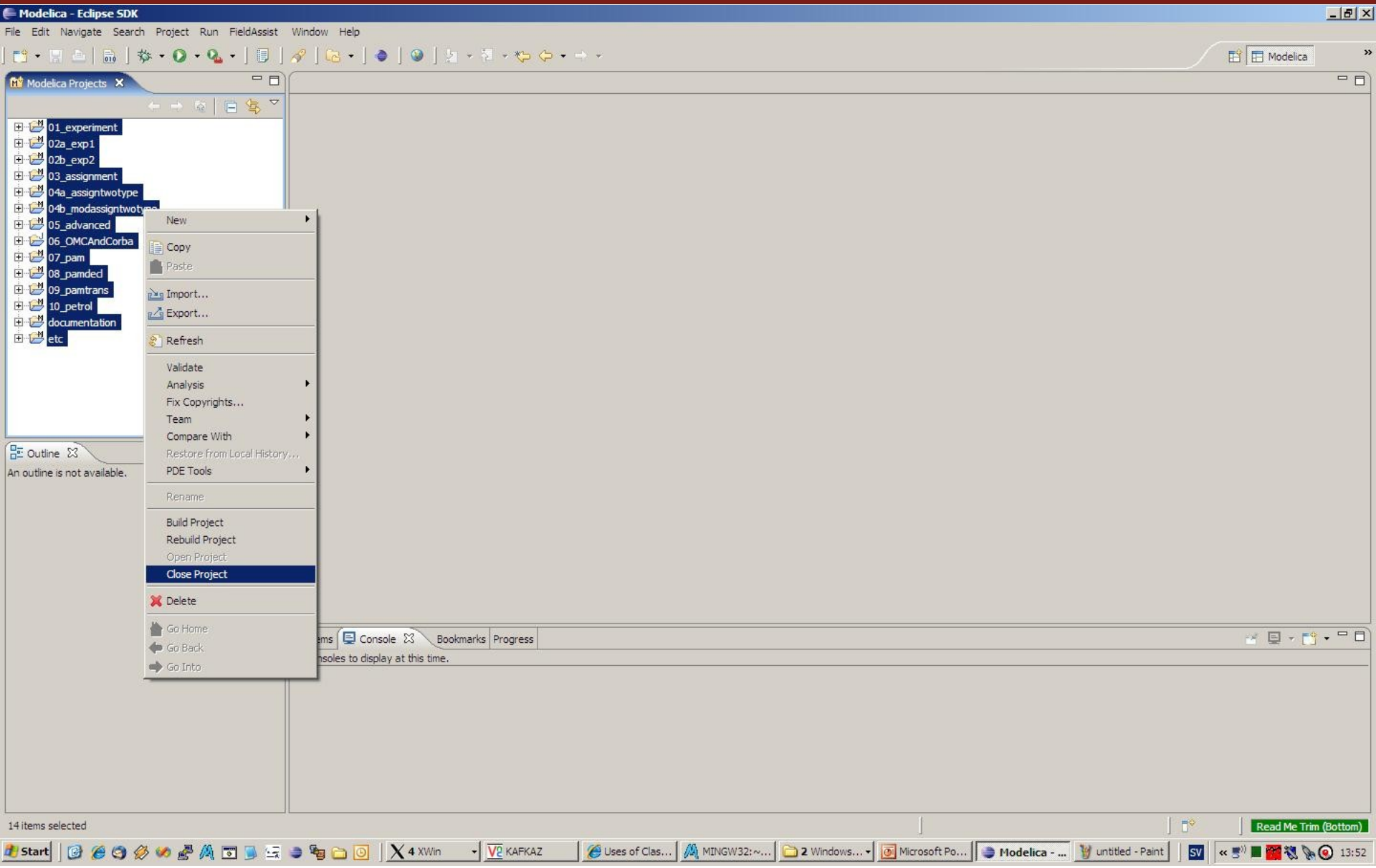
Don't build projects automatically



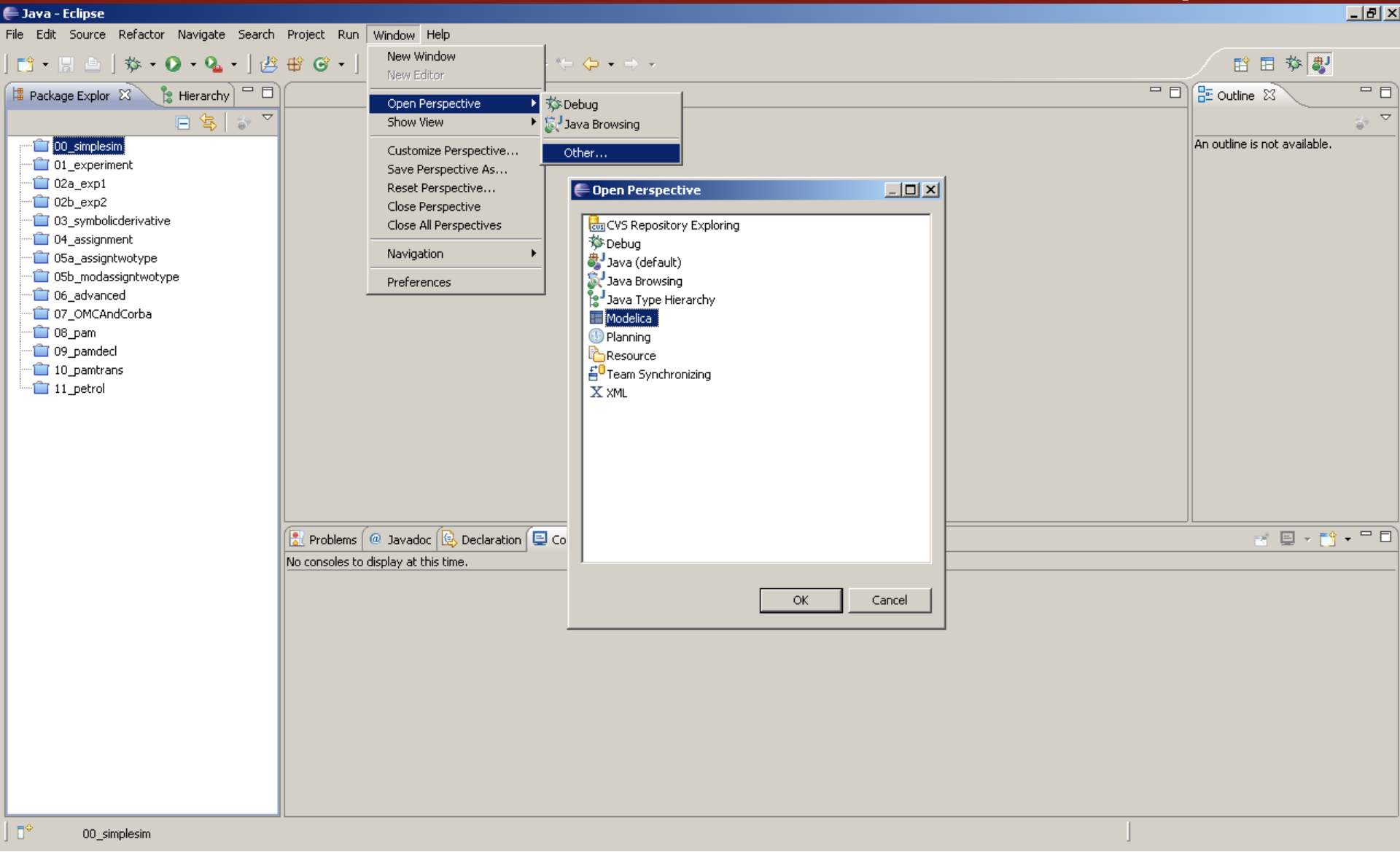
Refresh all projects



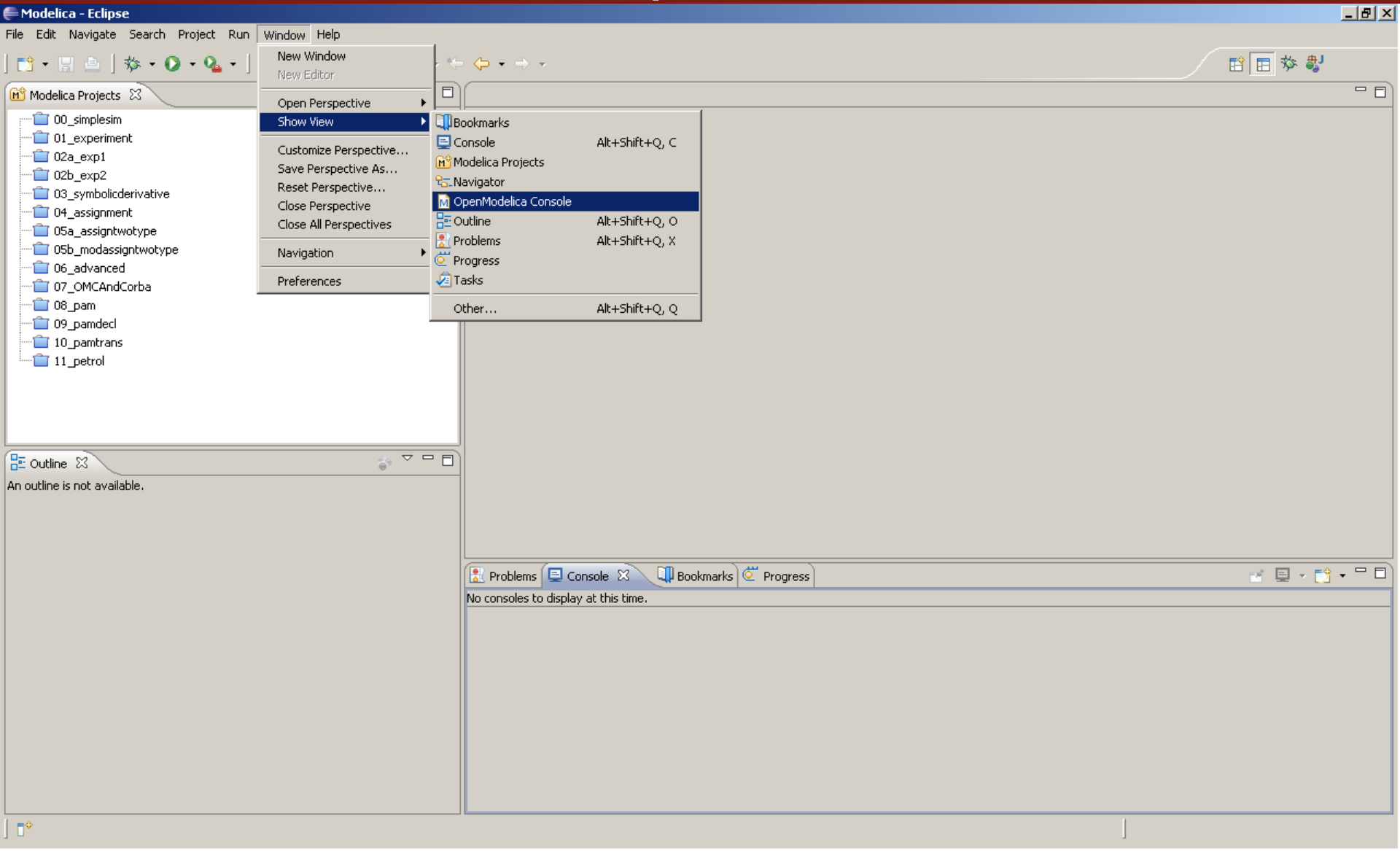
Close all projects



Switch to Modelica Perspective



Show the OpenModelica Console View



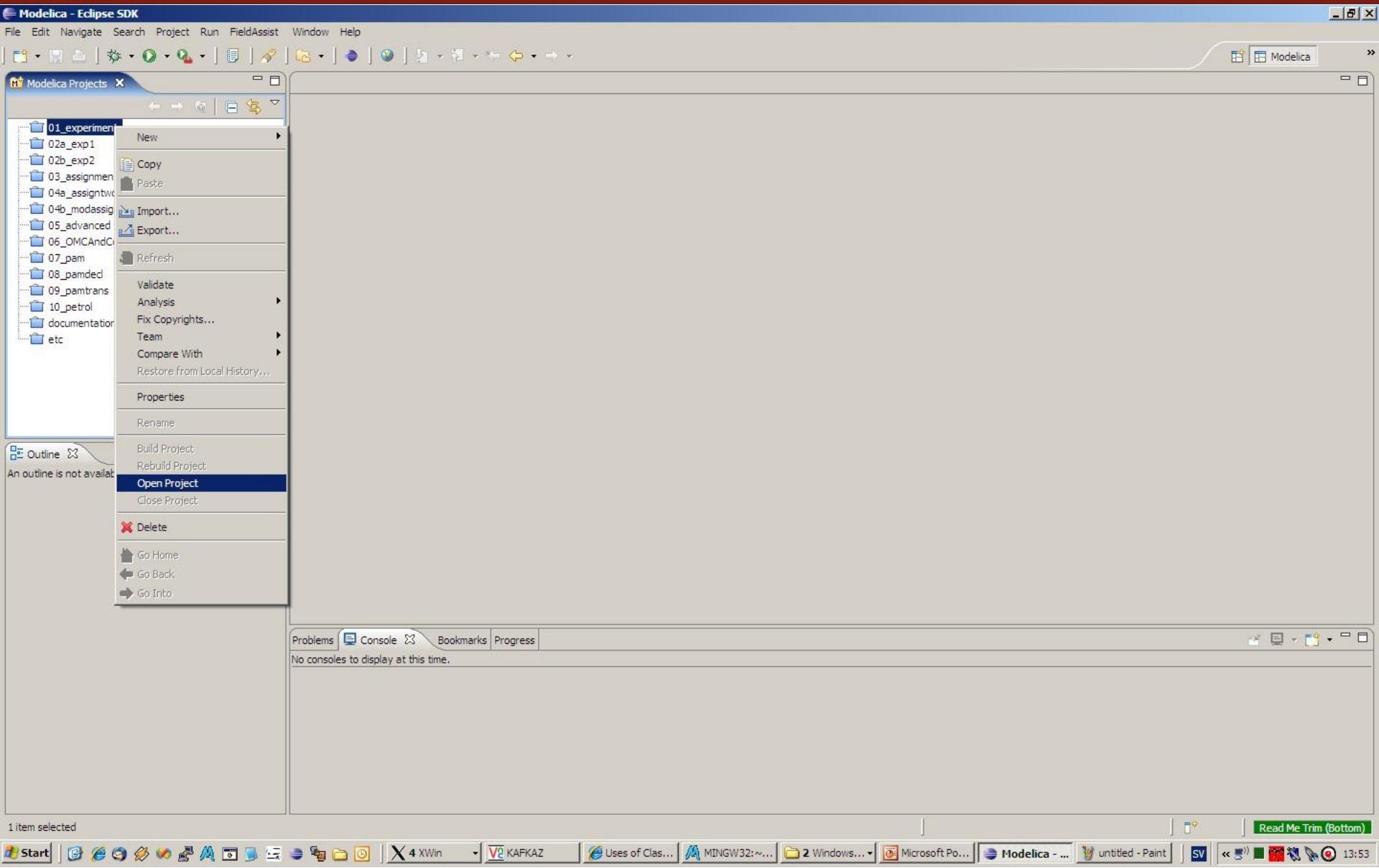
Exercises - 00_simplesim

- See **`README.txt`** in the Eclipse project
- In this exercise you perform a simple simulation in the MDT Eclipse environment
- Assignment
  - Type or copy a simple model into an Eclipse project
  - Open the Eclipse view “OpenModelica console”
  - simulate with the simulate command
  - plot with the plot command.
- Note: In the following exercises you will no longer use the “OpenModelica console”

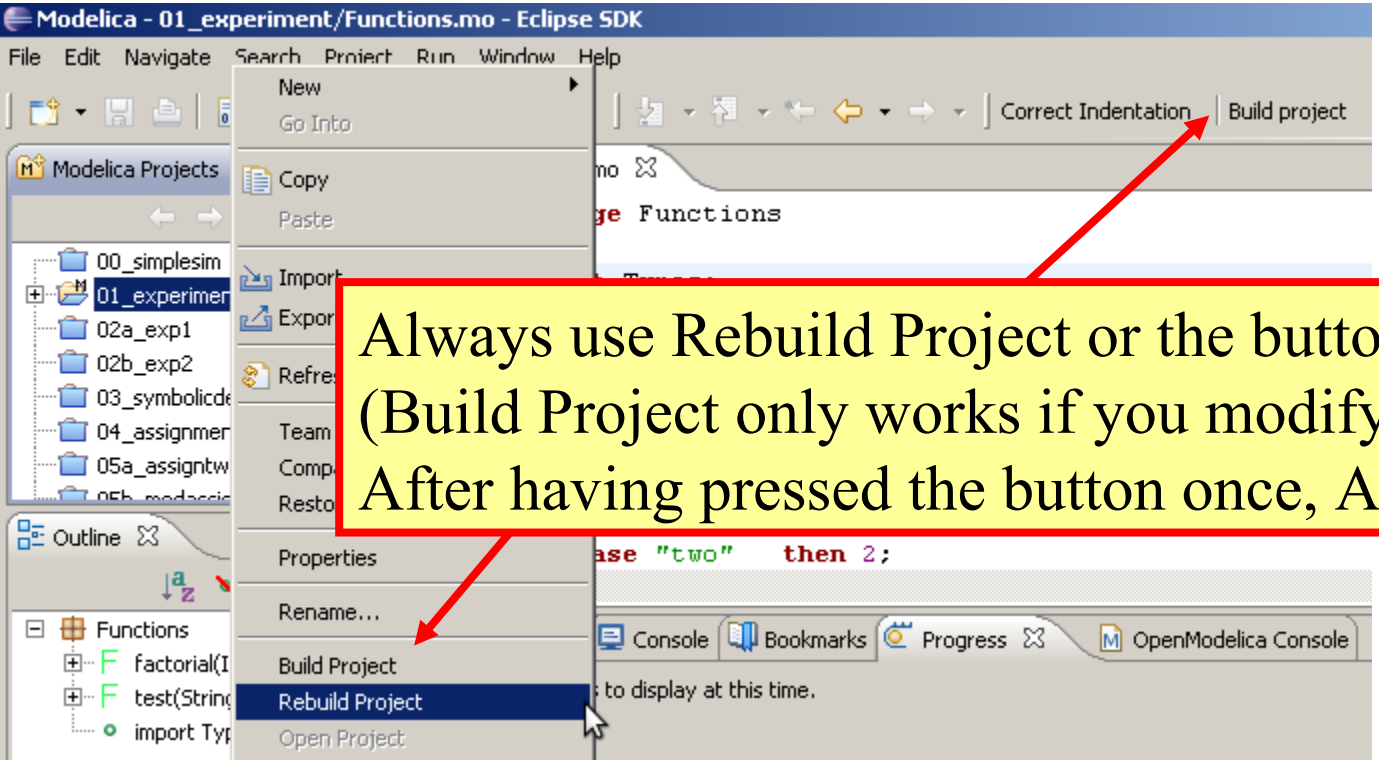
Exercises 01 - 04

- 01_experiment
- 02a_exp1, (02b_exp2 optional)
- 03_symbolicderivative
- 04_assignment (optional)

Open the first project

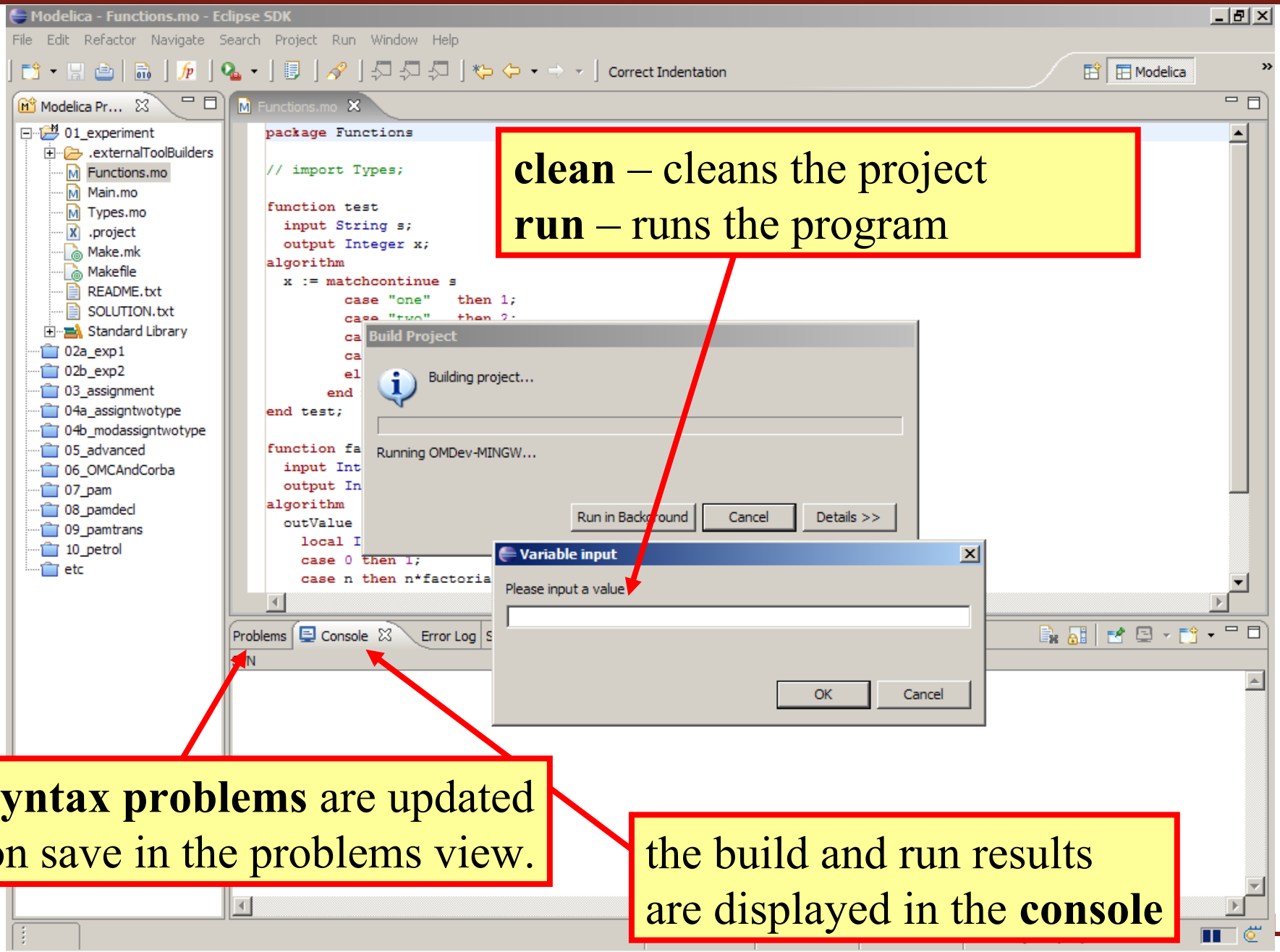


Build the first project



Always use Rebuild Project or the button
(Build Project only works if you modify a source file)
After having pressed the button once, Alt+B can be used

Eclipse - Building a project



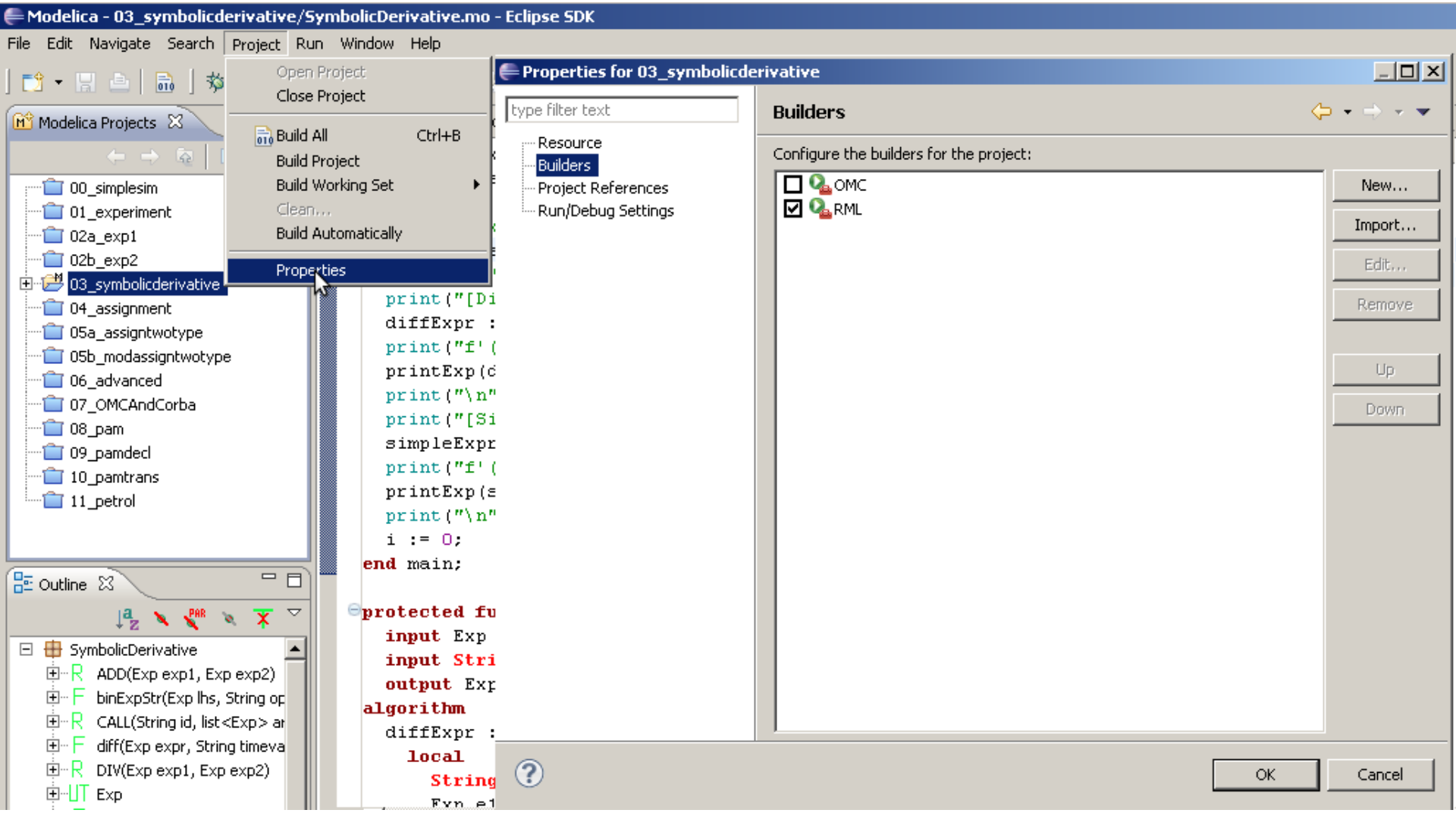
00_simplesim

- 01_experiment
- 02a_exp1, (02b_exp2 optional)
- 03_symbolicderivative
- 04_assignment (optional)

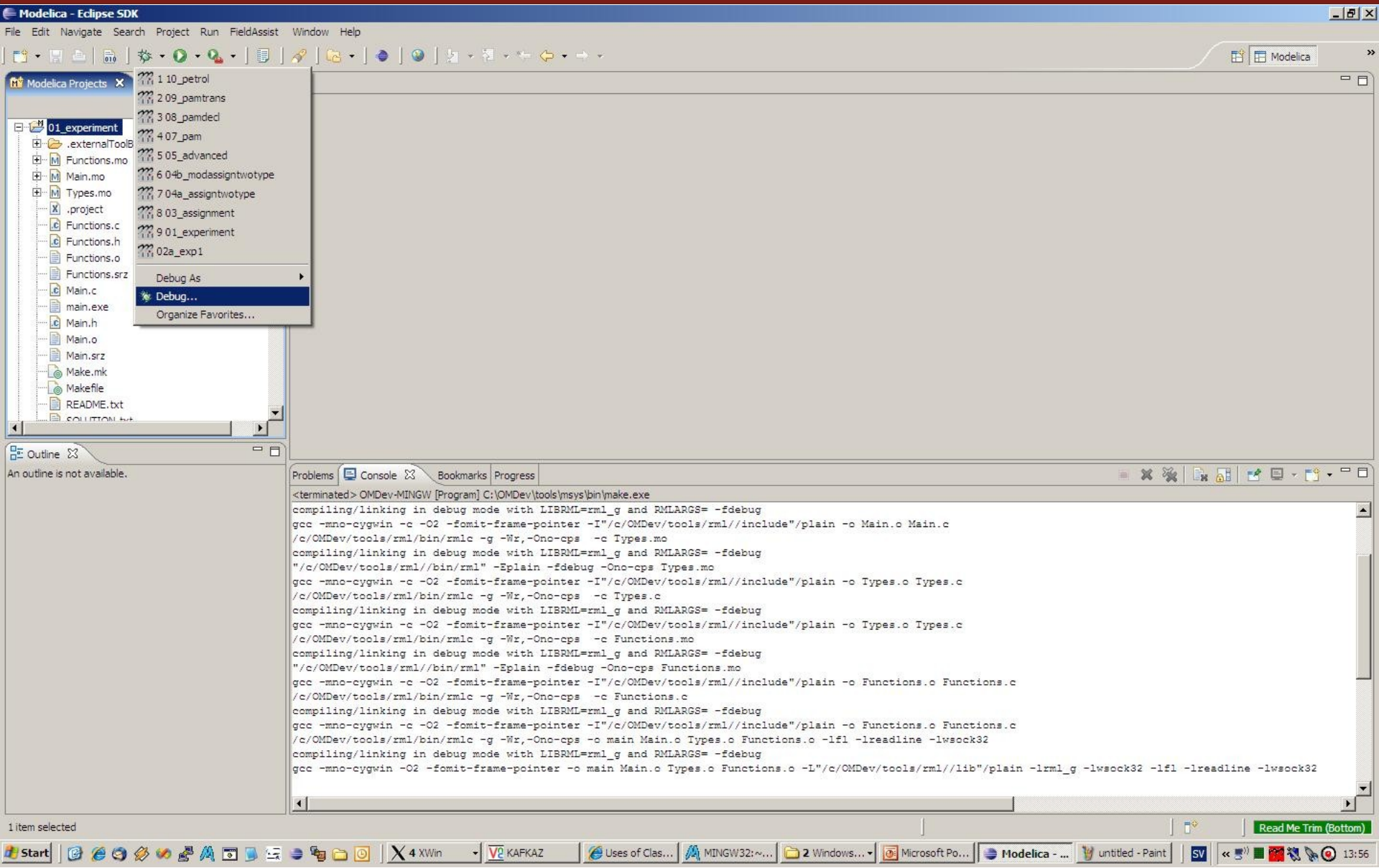
Debugging using OMC

- Not yet available
- Debug data structures using printAny(data)
- You can also switch to RML for an algorithmic debugger

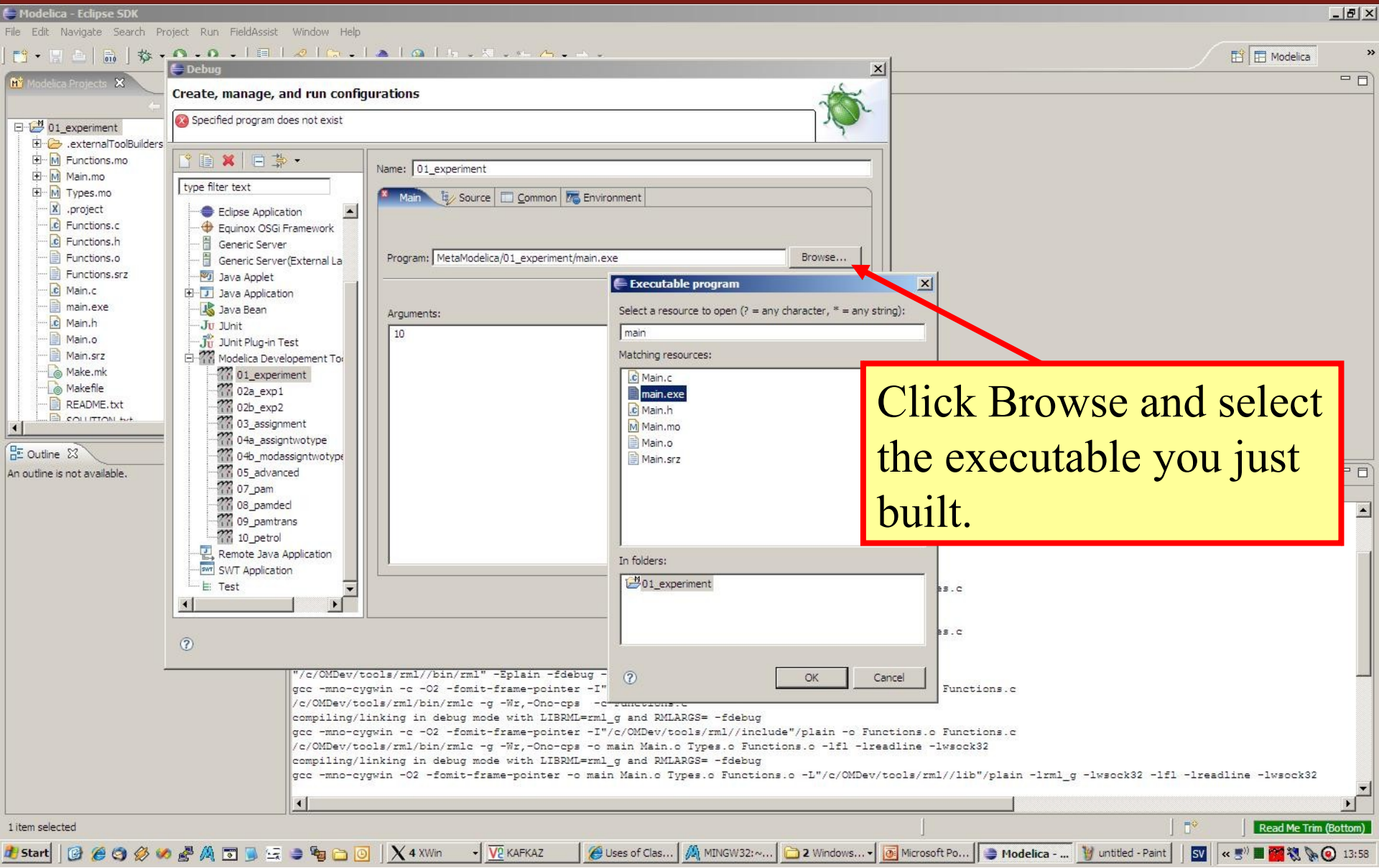
Changing compiler to RML



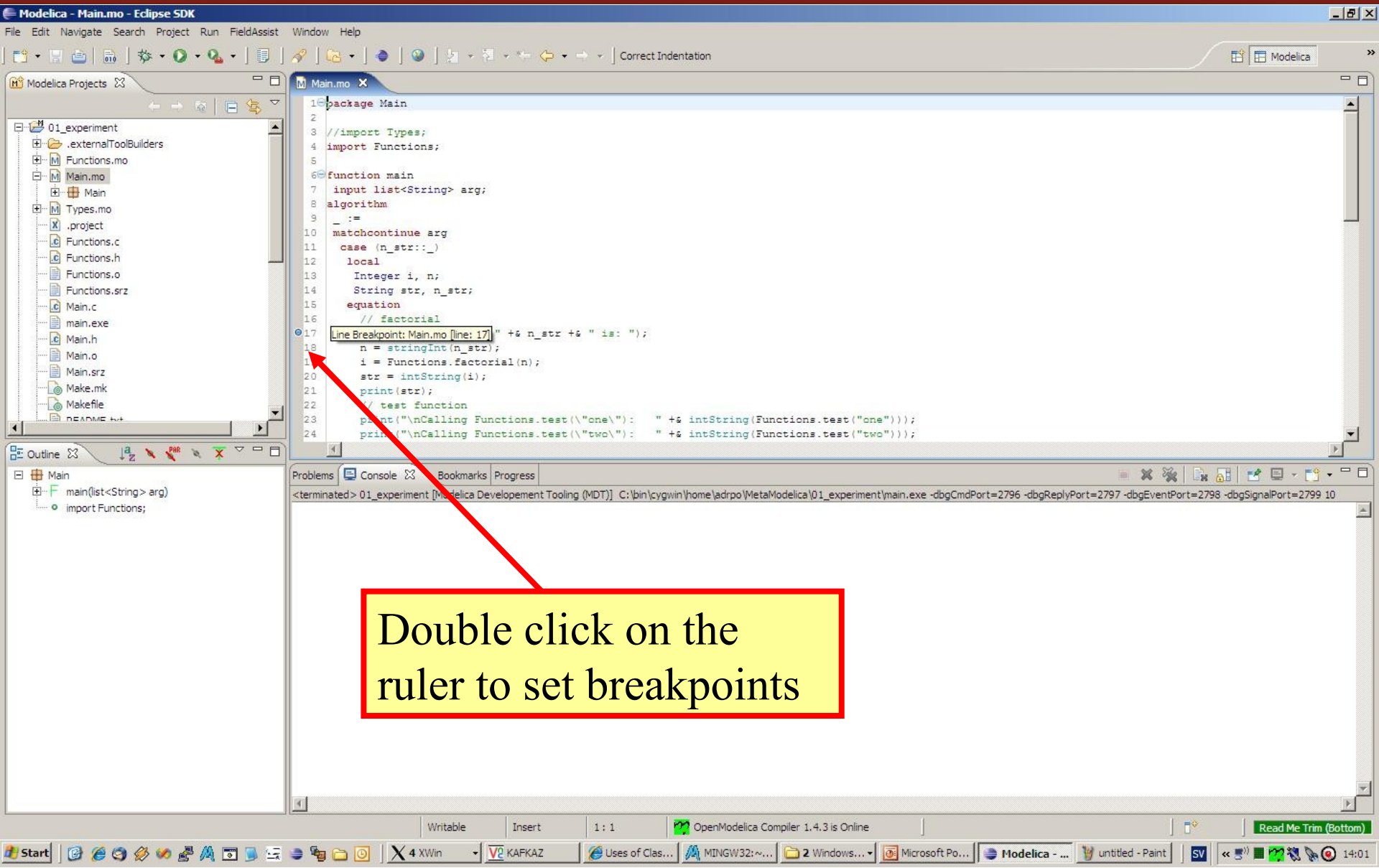
Setting the debug configuration (RML only)



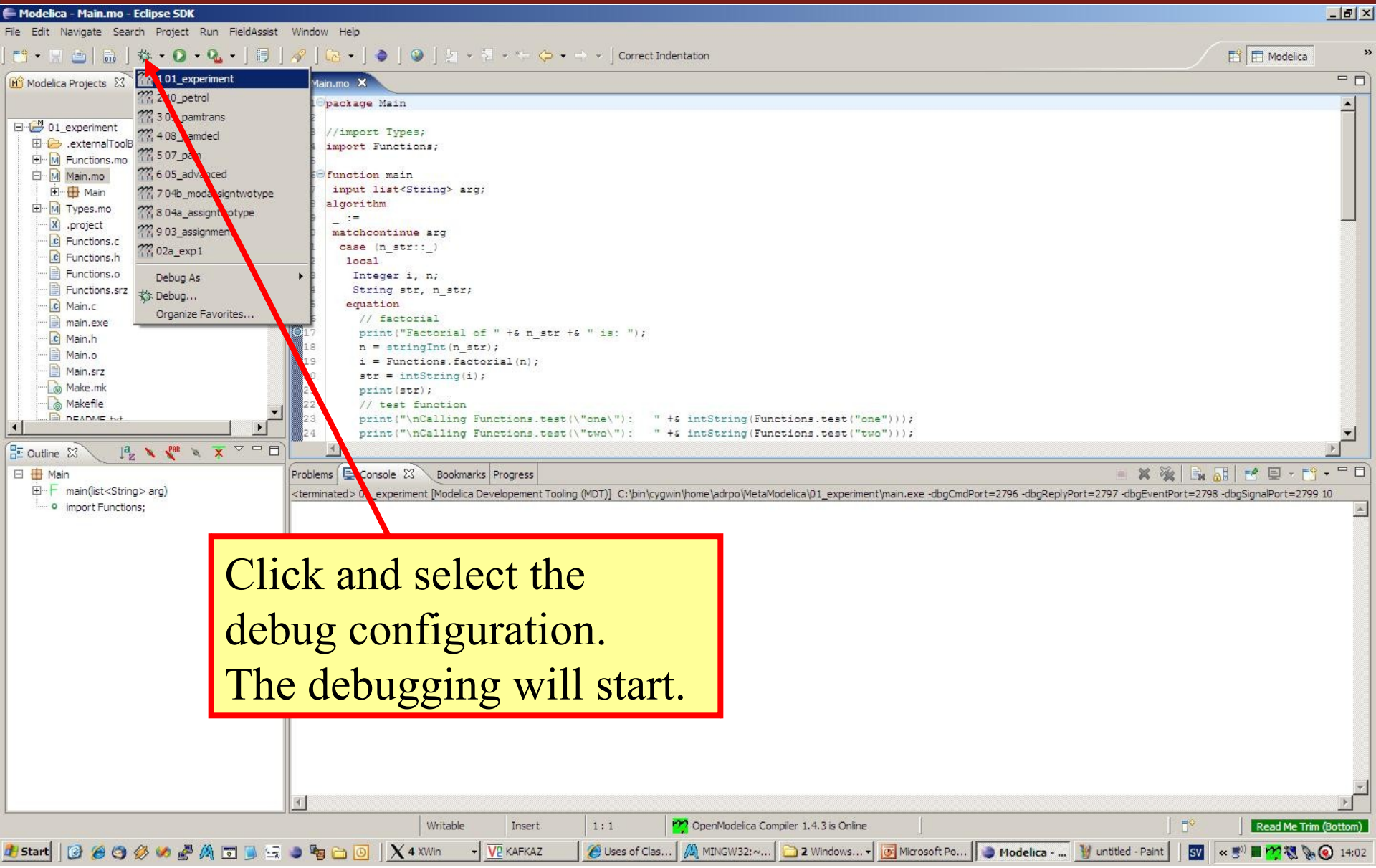
Specify the name of the executable



Set breakpoints in .mo file

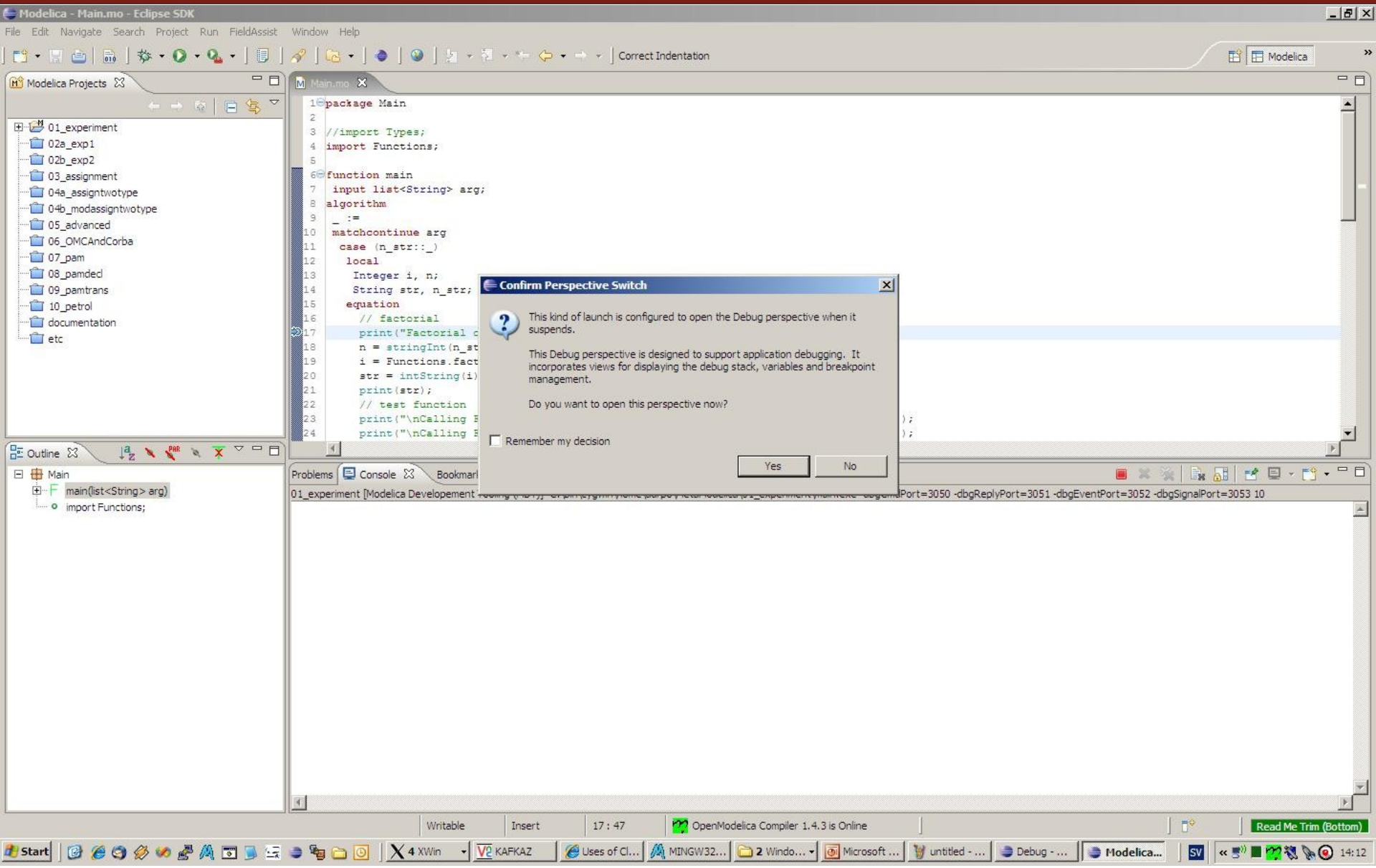


Run the debug configuration to start debugging

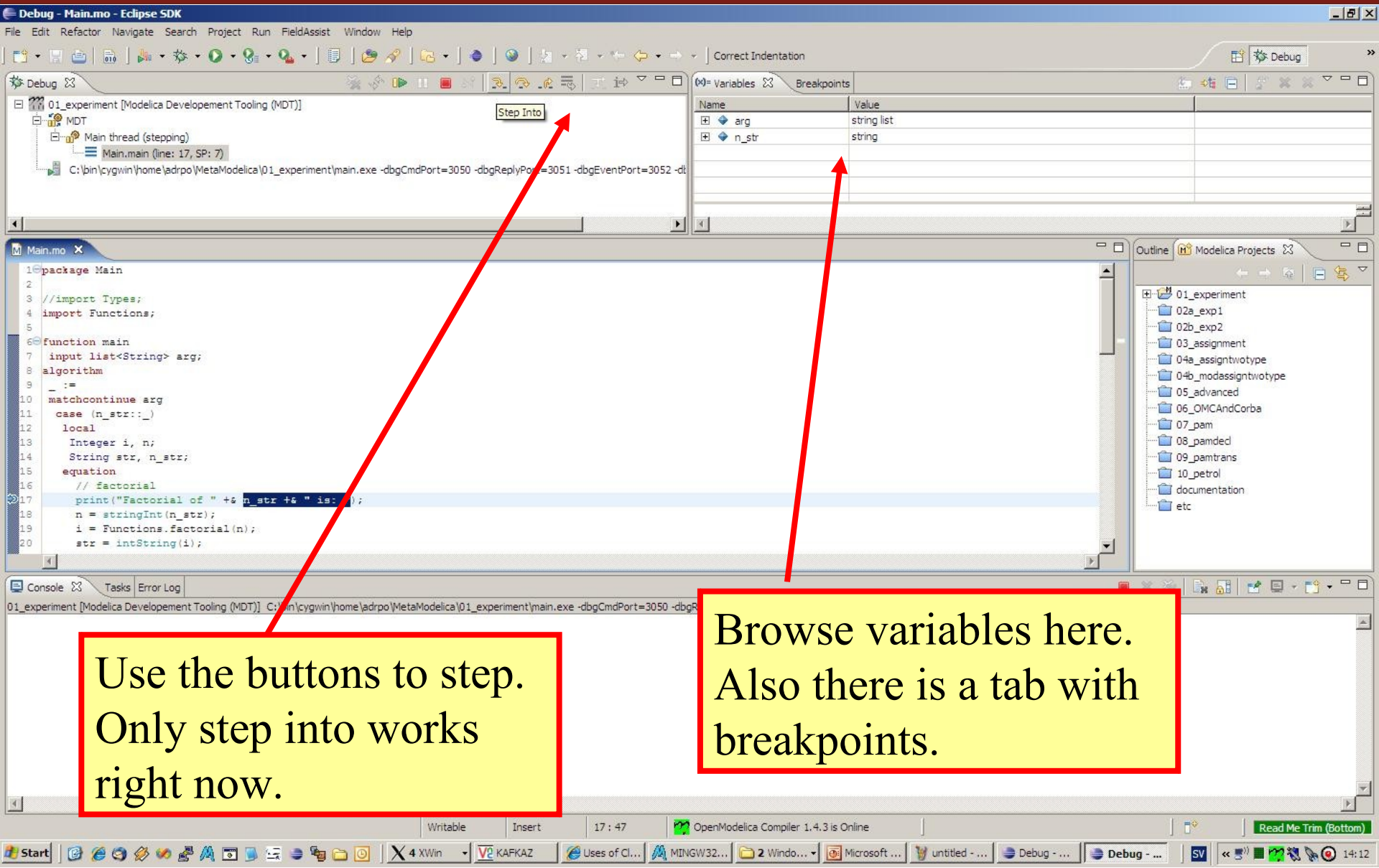


Click and select the debug configuration.
The debugging will start.

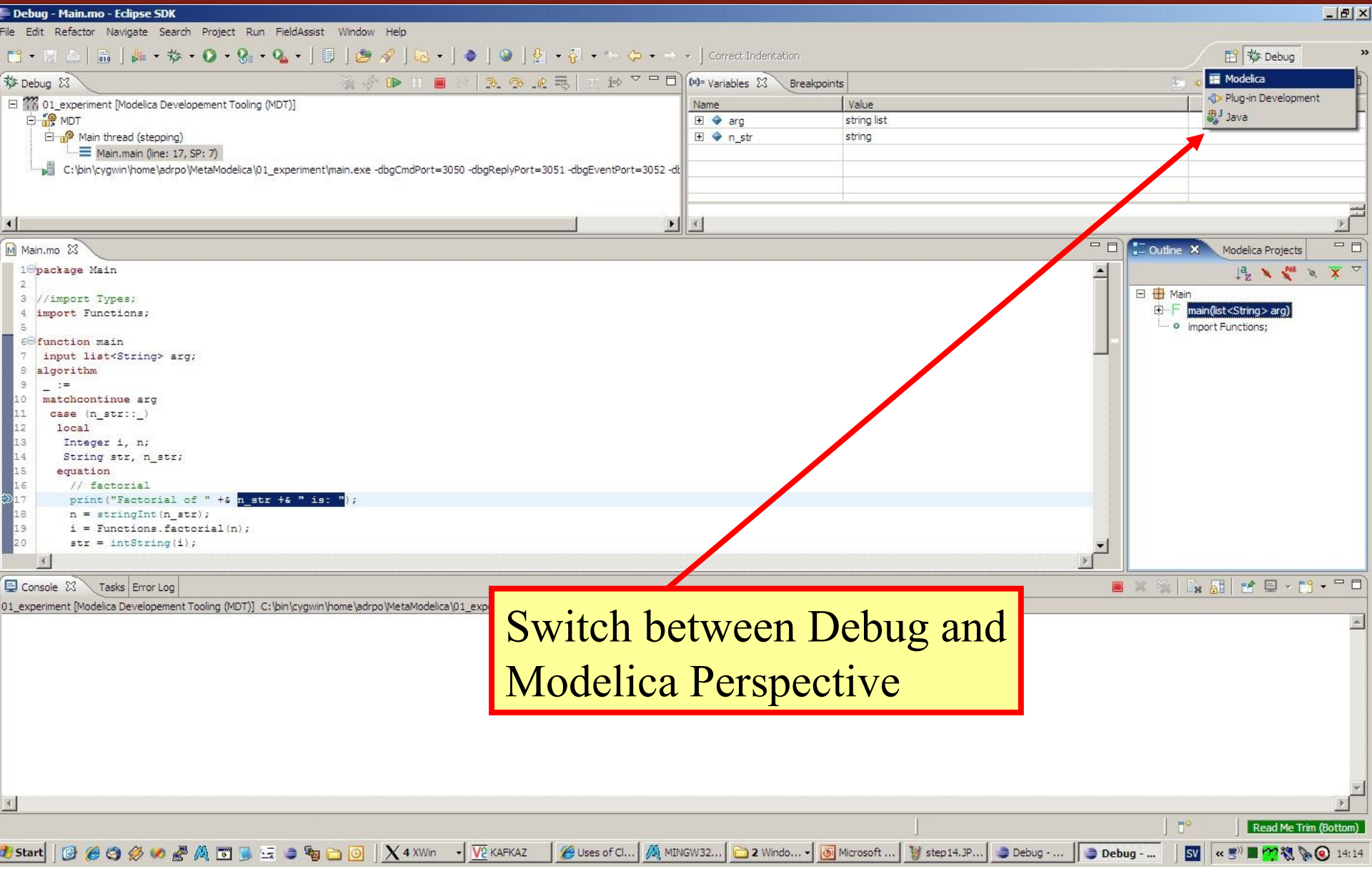
Eclipse will ask to switch to debugging perspective



Debugging perspective



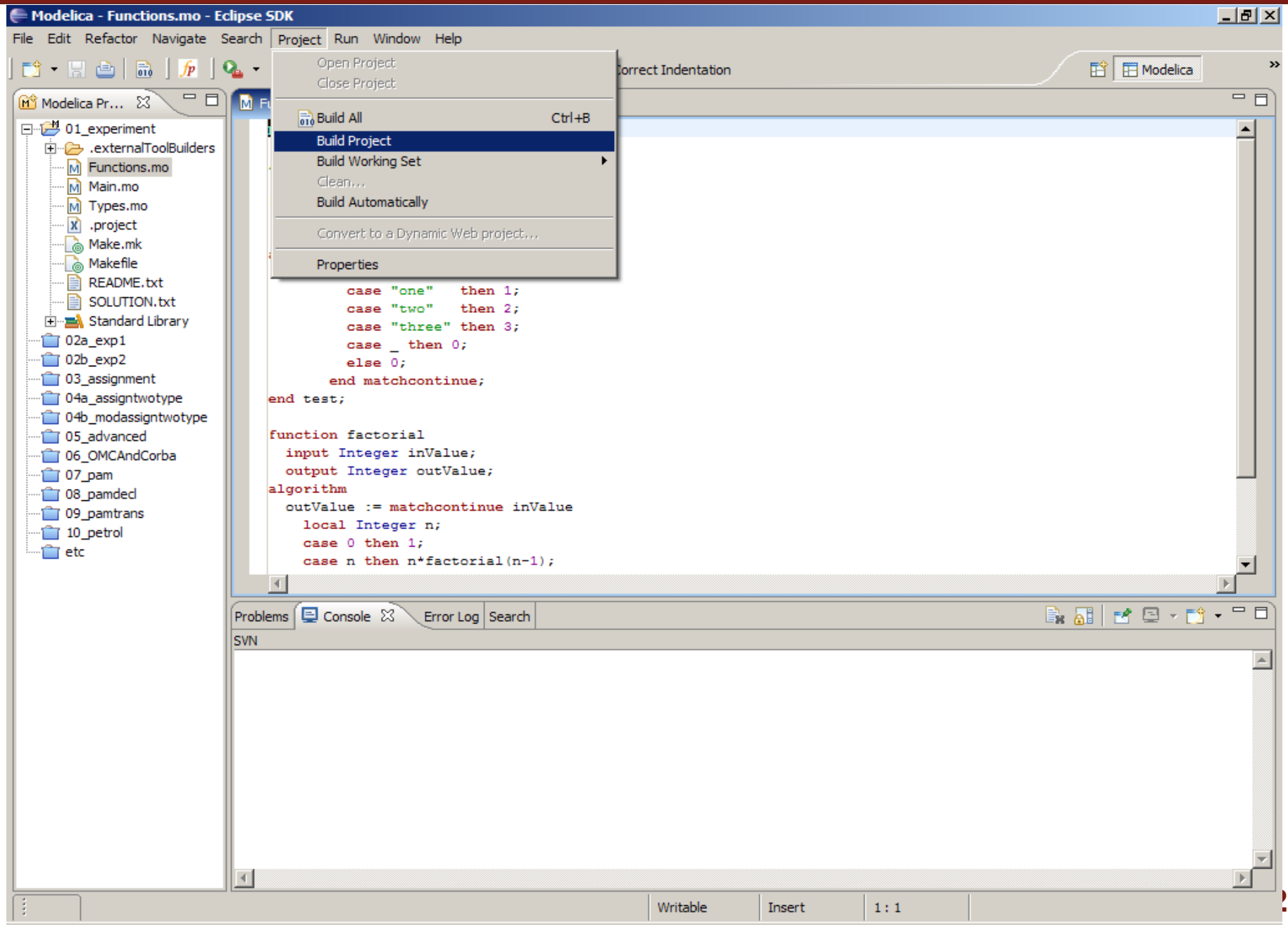
Switching perspectives



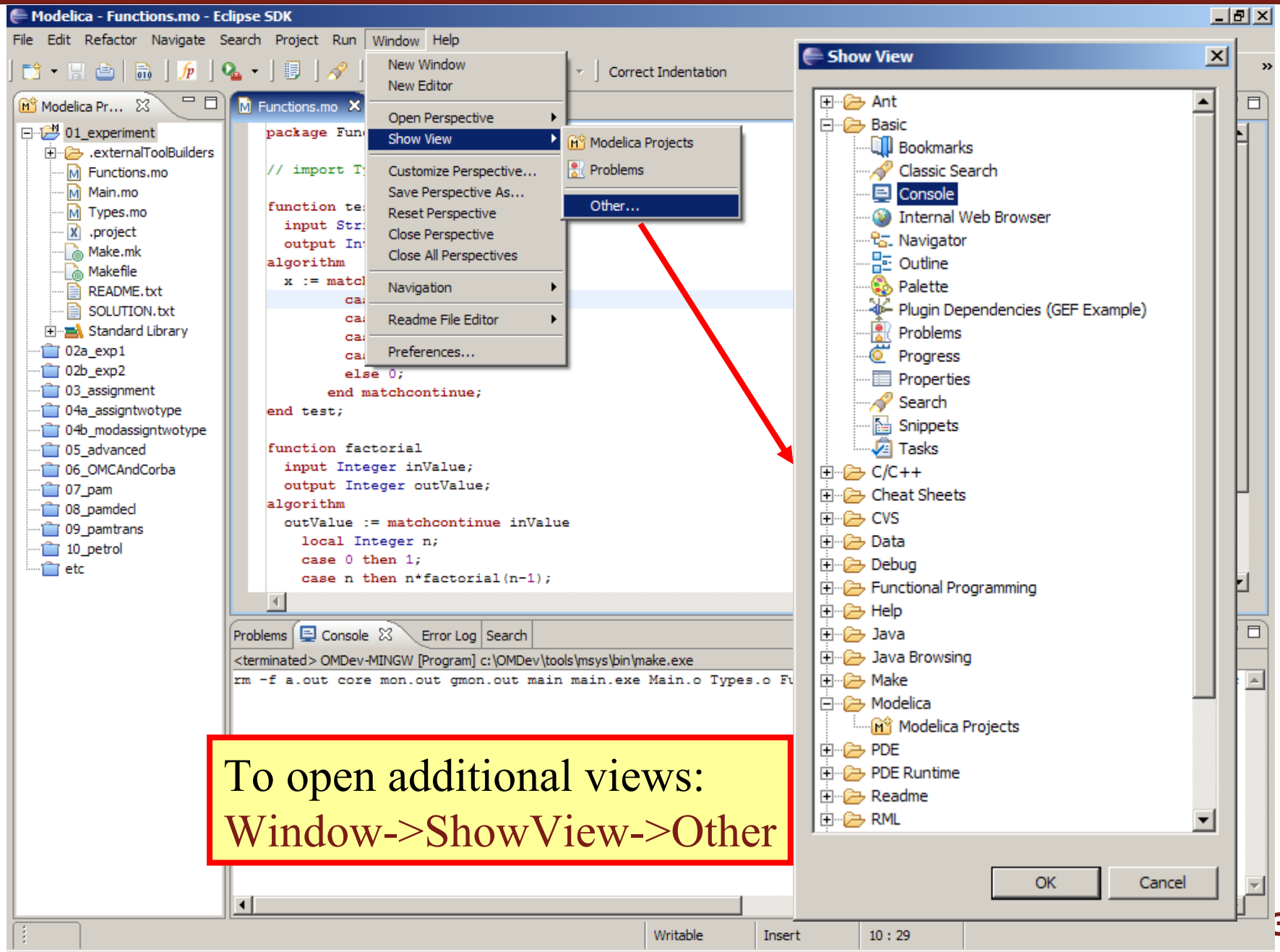
Eclipse Setup – Creating the projects

- repeat the procedure for all exercises
  - 01_experiment
  - 02a_exp1, 02b_exp2
  - 03_symbolicderivative
  - 04_assignment
  - 05a_assigntwotype
  - 05b_modassigntwotype
  - 06_advanced
  - 07_OMCAndCorba
- leave open only the project you are working on! close all the others

Eclipse – Building a project



Eclipse – Opening views



To open additional views:
Window->ShowView->Other

Exercises

- Each exercise is in a different Eclipse project
- All exercises have :
  - **`README.txt`** - information about the exercise
  - **`SOLUTION.txt`** – the solution of the exercise (if the exercise has some implementation assignment)
  - **`program.txt`** – input program to the exercise, edit if needed (for the exercises which have an input)
- Consult the **`MetaModelica Programming Guide`** and the slides if you need additional information during the exercises.
- Of course, feel free to ask us any type of questions, it is faster and better!

Exercises - 01_experiment

- See **`README.txt`** in the Eclipse project
- In this exercise you experiment with
  - Types
  - Constants
  - Functions
- Assignment
  - Write functions in **`Functions.mo`** to display the constants defined in **`Types.mo`**.
  - Search for **`// your code here`** in **`Main.mo`** and **`Functions.mo`**
- Compare your solution with the **`SOLUTION.txt`** you find in the Eclipse project

Exercises - 02a_exp1

- See **`README.txt`** in the Eclipse project
- In this exercise you will add new constructs to the exp1 language and deal with their evaluation.
- Assignment - add new constructs to the language
  - a power operator `(^)`
  - a factorial operator `(!)`
  - search for **`// your code here`** within **`Exp1.mo`**
- Note
  - The parser/lexer are ready, but give parser errors for the new operators until they are added in **`Exp1.mo`**
- Compare your solution with the **`SOLUTION.txt`** you find in the Eclipse project

Exercises – 02b_exp2

- See **`README.txt`** in the Eclipse project

- In this exercise you will explore a different way to model the **`exp1`** language using different **`Exp`** trees.
- Explore the **`Exp2.mo`** file and compare it with **`Exp1.mo`** file.

Exercises – 03_symbolicderivative

- See **`README.txt`** in the Eclipse project
- Assignment:
  - add rules to derive ‘-’, ‘*’, sine, cosine and power expressions
  - add rules to simplify ‘-’, sine, cosine and power expressions
  - search for **`// your code here`** within **`SymbolicDerivative.mo`**
- Compare your solution with the **`SOLUTION.txt`** you find in the Eclipse project

Exercises – 04_assignment

- See **`README.txt`** in the Eclipse project
- Assignment - add functions to print:
  - the assignments present in the current program before the actual evaluation
  - the environment after it was augmented with the assignments
  - search for **`// your code here`** within **`Assignment.mo`**
- Compare your solution with the **`SOLUTION.txt`** you find in the Eclipse project

Exercises – 05a_assigntwotype

- See **`README.txt`** in the Eclipse project
- Assignment - add functions to print:
  - add a new **`String`** type which can hold only integers as strings to the current **`Exp`** node
  - add cases to evaluate expressions/assignments of the form `"2" + 1 + "1" + 1.0` in the **`eval`** function
  - search for **`// your code here`** within **`AssignTwoType.mo`**
- Compare your solution with the **`SOLUTION.txt`** you find in the Eclipse project

Exercises – 05b_modassigntwotype

- See **`README.txt`** in the Eclipse project
- In this exercise you will explore a different way to structure your code within different packages.
- The code from **`05a_assigntwotype`** is now split over 4 packages.
- Compare the 05a/b projects.

Exercises - 06_advanced (I)

- See **`README.txt`** in the Eclipse project
- In this exercise you experiment with
  - polymorphic types
  - constants
  - higher order functions
- Assignment 1
  - Write a polymorphic function that orders a list of any type.
  - The function has as input a list and a compare function between the objects of that list.
  - Write the comparison functions for **`Integers`**, **`Strings`** and **`Reals`**.
  - Test your function on the **`Types.intList`**

Exercises – 06_advanced (II)

- See `README.txt` in the Eclipse project
- Assignment 2
  - Write a polymorphic map function that applies a function over a list and returns a new list with the result.
  - Write three functions that transform from:
    - integer to real
    - integer to string
    - real to string
  - Use your map function and the two transformation functions to transform the `Types.intList` to a list of reals and a list of string, then apply the ordering function from Assignment 1 on the newly created lists

Exercises – 06_advanced (III)

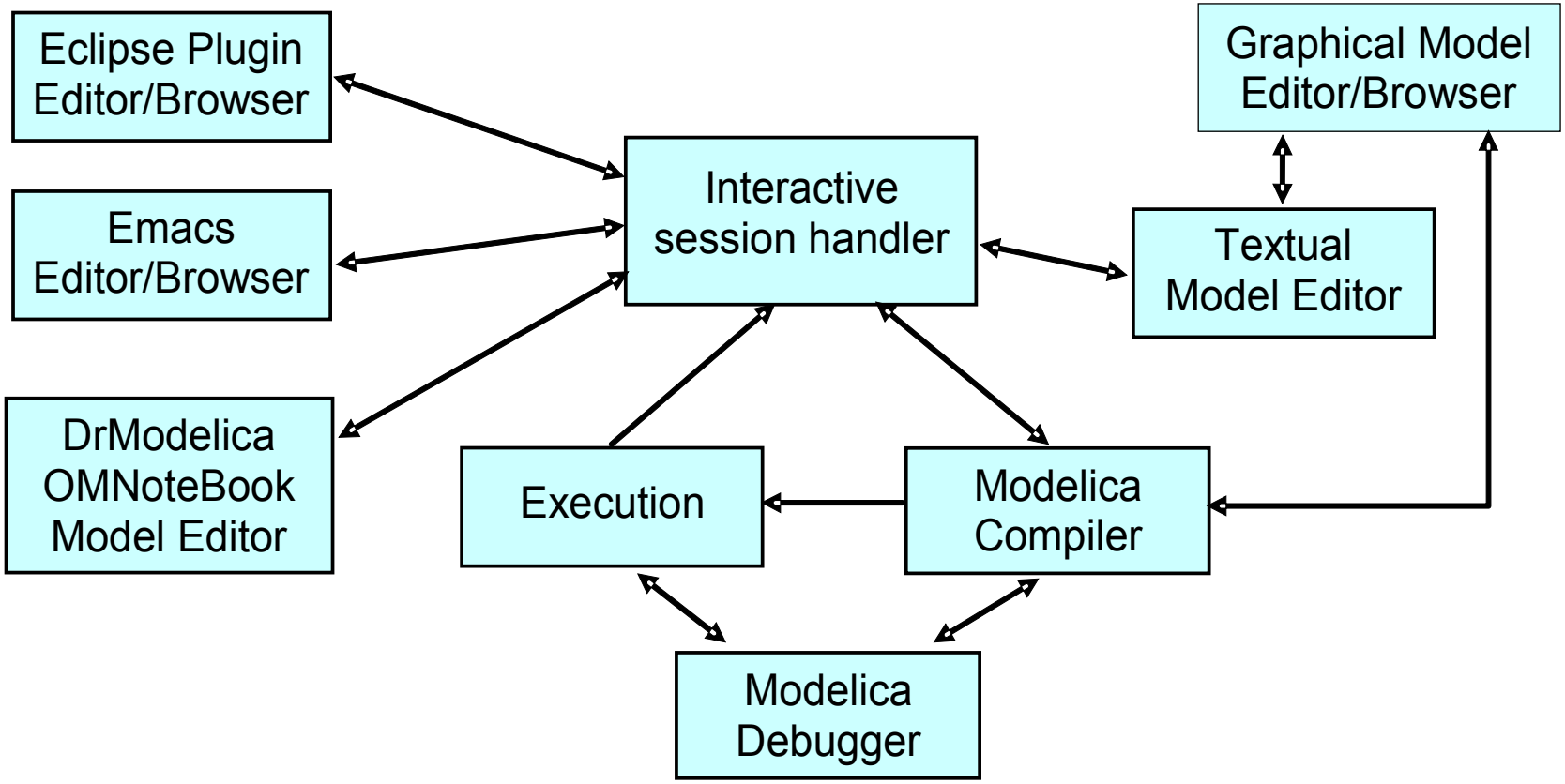
- See **`README.txt`** in the Eclipse project
- Assignment 3
  - Write a polymorphic map function that applies a print function over a list (of Strings) and prints the it.
  - Use the transformer functions from real->string and integer->string from Assignment 2 to transform the real list or the integer list to a string list for printing.
- Compare your solution with the **`SOLUTION.txt`** you find in the Eclipse project

We are Switching to OMC Overview now!

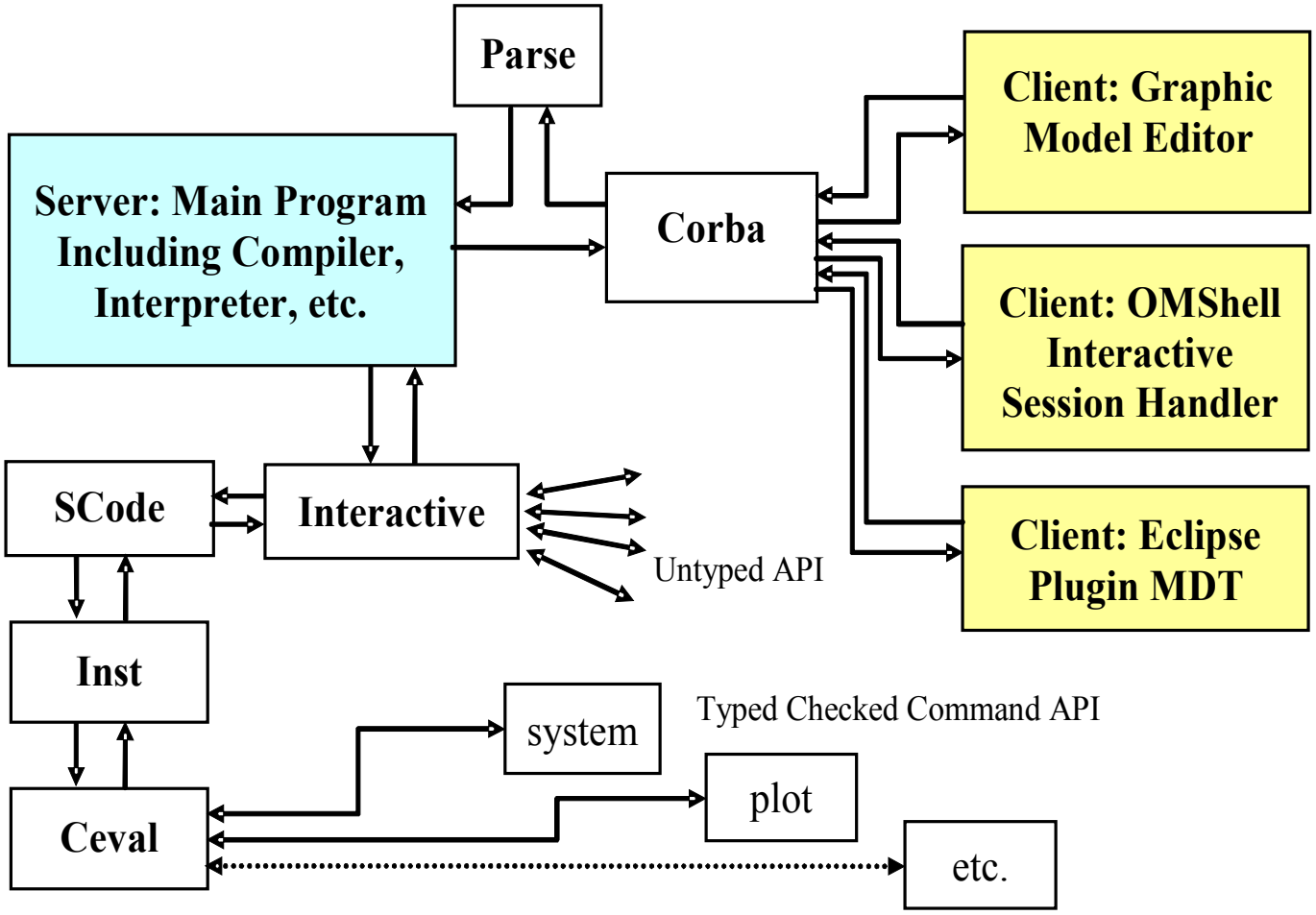
OpenModelica Development Toolkit (OMDev)

- OMDev is a pre-packaged pre-compiled kit containing all tools needed for OpenModelica development.
  - Just unpack and set some environment variables. (Windows)
  - apt-get build-dep openmodelica (Ubuntu/Debian Linux)
- MetaModelica Compiler (MMC) – for developing OMC
- OpenModelica Compiler (OMC) – for browsing support
- Eclipse plugin MDT – (Modelica Development Tooling), e.g. for compiler (OMC) development
- Pre-compiled Corba (MICO or omniORB) for tool communication
- Packaged Gnu compiler (GCC; Mingw version for Windows)
- Emacs mode
- Online (web) Subversion for version handling
- Online (web) Codebeamer for bug reporting and management
- Automatic regression testing using a test suite
- Unit testing using the bootstrapped OpenModelica Compiler
- Interactive MetaModelica debugger

OpenModelica Environment Architecture



OpenModelica Client-Server Architecture



Released in OpenModelica 1.5.0

- OpenModelica Compiler/Interpreter – OMC
- Interactive session handler – OMShell
- OpenModelica Notebook with DrModelica and DrControl – OMNotebook
- OpenModelica Eclipse plugin MDT
- SimForge graphic editor
- MetaModelica Debugger

OpenModelica Compiler/Interpreter

- OpenModelica 1.5.1
- Currently implemented in 280 000 lines of MetaModelica (1.4.5 was 180 000)
- Includes code generation, BLT-transformation, index reduction, connection to DASSL, etc.
- Most of the Modelica 3.1 language including classes, functions, inheritance, modifications, import, etc.
- Hybrid/Discrete event support

- Simple text-based (string) communication in Modelica Syntax
- API supporting model structure query and update

Example Calls:
Calls fulfill the normal Modelica function call syntax.:

```
saveModel("MyResistorFile.mo",MyResistor)
```

will save the model MyResistor into the file “MyResistorFile.mo”.

For creating new models it is most practical to send a model, e.g.:

```
model Foo   end Foo;
```

or, e.g.,

```
connector Port   end Port;
```

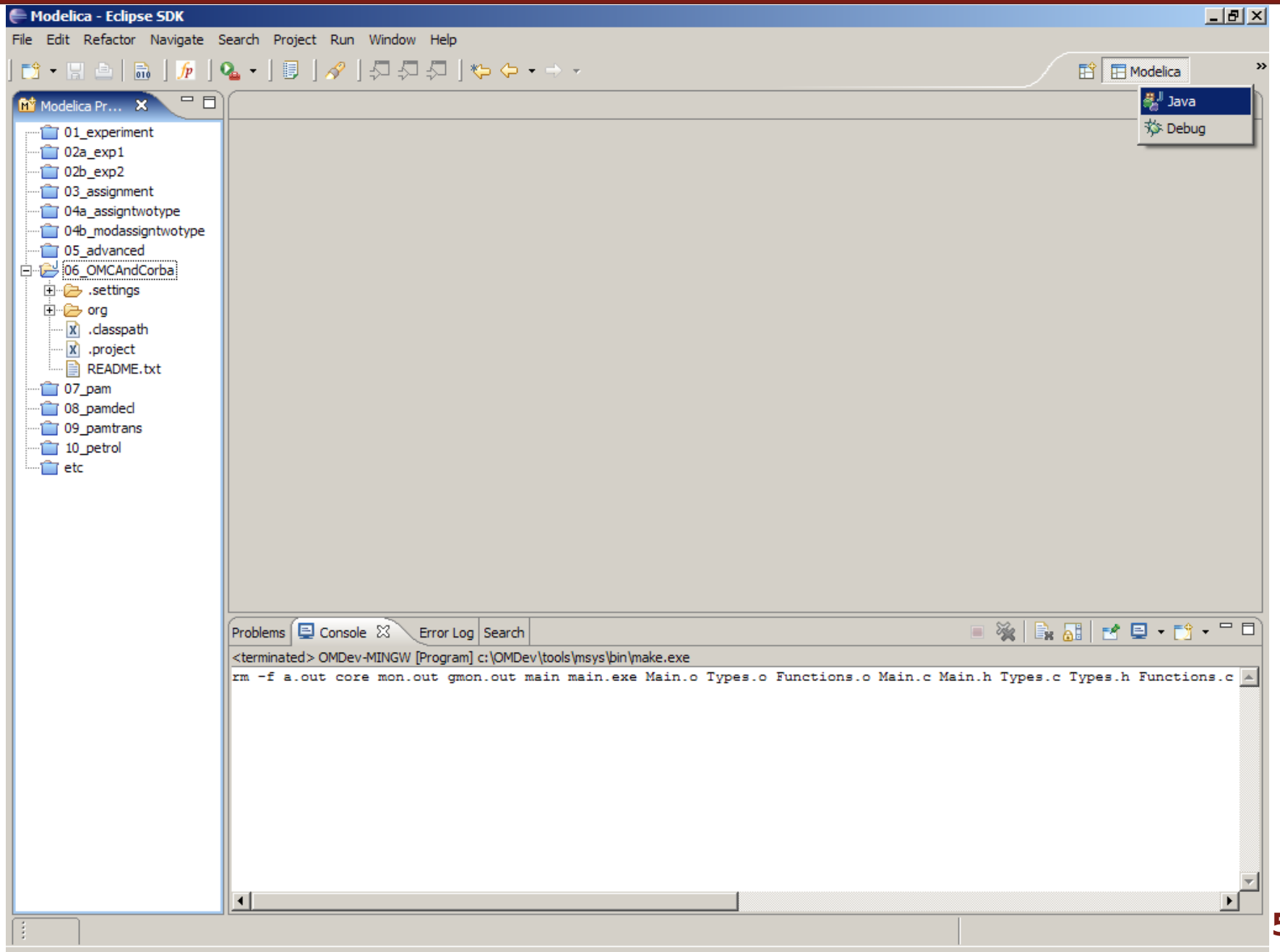
Some of the Corba API functions

saveModel(A1<string>,A2<cref>)	Saves the model (A2) in a file given by a string (A1). This call is also in typed API.
loadFile(A1<string>)	Loads all models in the file. Also in typed API. Returns list of names of top level classes in the loaded files.
loadModel(A1<cref>)	Loads the model (A1) by looking up the correct file to load in `$MODELICAPATH`. Loads all models in that file into the symbol table.
deleteClass(A1<cref>)	Deletes the class from the symbol table.
addComponent(A1<ident>,A2<cref>, A3<cref>,annotate=<expr>)	Adds a component with name (A1), type (A2), and class (A3) as arguments. Optional annotations are given with the named argument `annotate`.
deleteComponent(A1<ident>, A2<cref>)	Deletes a component (A1) within a class (A2).
updateComponent(A1<ident>, A2<cref>, A3<cref>,annotate=<expr>)	Updates an already existing component with name (A1), type (A2), and class (A3) as arguments. Optional annotations are given with the named argument `annotate`.
addClassAnnotation(A1<cref>, annotate=<expr>)	Adds annotation given by A2(in the form `annotate= classmod(...))` to the model definition referenced by A1. Should be used to add Icon Diagram and Documentation annotations.
getComponents(A1<cref>)	Returns a list of the component declarations within class A1: `{{Atype,varidA,"commentA"},{Btype,varidB,"commentB"}, {...}}`
getComponentAnnotations(A1<cref>)	Returns a list `{...}` of all annotations of all components in A1, in the same order as the components, one annotation per component.
getComponentCount(A1<cref>)	Returns the number (as a string) of components in a class, e.g return `"2"` if there are 2 components.
getNthComponent(A1<cref>,A2<int>)	Returns the belonging class, component name and type name of the nth component of a class, e.g. `"A.B.C,R2,Resistor"`, where the first component is numbered 1.

Platforms

- All OpenModelica GUI tools (OMShell, OMNotebook, ...) are developed on the Qt4 GUI library, portable between Windows, Linux, Mac
- Both compilers (OMC, MMC) are portable between the three platforms and compiled nightly
  - Windows – main release platform
  - Linux – main development platform
  - Mac – available

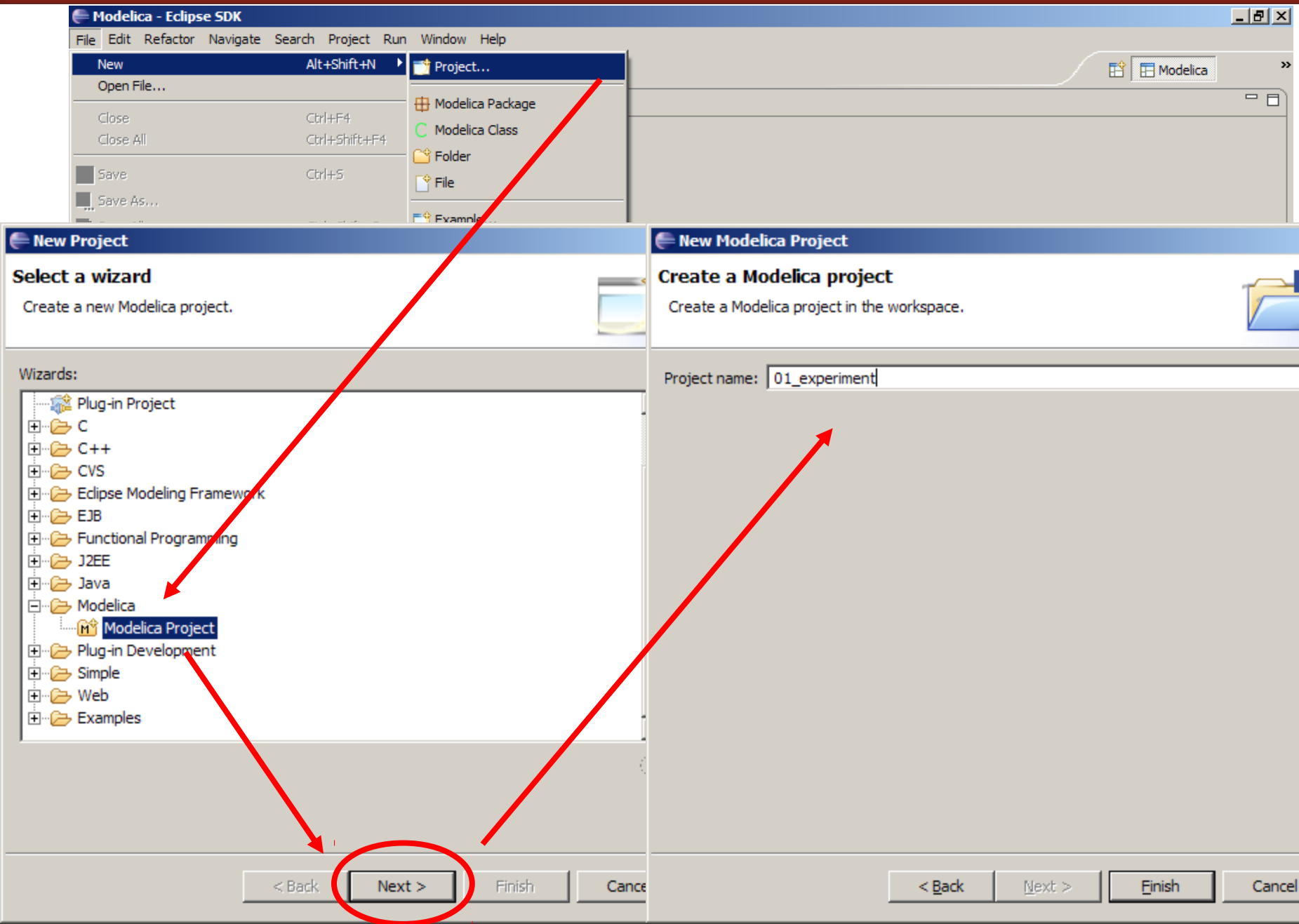
Eclipse – Switching to Java Perspective



Exercises - 07_OMCAndCorba

- See **`README.txt`** in the Eclipse project
- In this exercise you will send commands to the OMC Compiler (**`omc.exe`**) via CORBA
- OMCProxy.java has functionality for
  - starting the omc process if is not already started
    - the starting is a bit different for Windows/Linux
  - sending commands to OMC
  - logging facilities
- If you need clients in C++ or Python check
  - http://www.ida.liu.se/labs/pelab/modelica/OpenModelica.html
  - developer pages

Eclipse Setup - Creating the projects



Thank you!

Administrative Question:

What would you like to implement tomorrow in the OpenModelica Compiler?