

# Overture Plug-in Tutorial

**Kenneth Lausdahl**

Department of Engineering, Aarhus University, Denmark

28 August 2013 / 11th International Workshop  
Overture/VDM

# Outline

- 1 Introduction
- 2 Creating Plug-ins For Overture

# Outline

1 Introduction

2 Creating Plug-ins For Overture

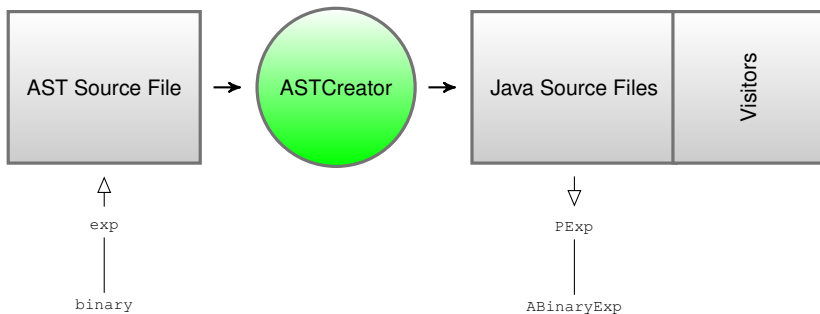
# Previously Presented

This presentation is based on:

## **The Overture AST and Plug-in Development**

*28 August 2012 / 10th International Workshop Overture/VDM*

# AST Generation



# AST Generation

## AST Source File

### Abstract Syntax Tree

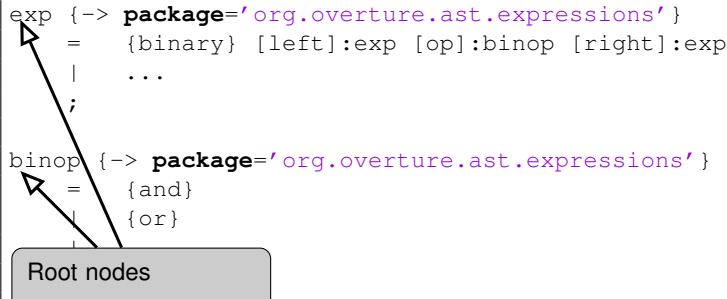
```
exp {-> package='org.overture.ast.expressions'}  
    =    {binary} [left]:exp [op]:binop [right]:exp  
    |  
    ...  
    ;  
  
binop {-> package='org.overture.ast.expressions'}  
      =    {and}  
      |    {or}  
      |  
      ...  
      ;
```

# AST Generation

## AST Source File

### Abstract Syntax Tree

```
exp {-> package='org.overture.ast.expressions'}  
  =  {binary} [left]:exp [op]:binop [right]:exp  
    |  
    ...  
;  
  
binop {-> package='org.overture.ast.expressions'}  
  =  {and}  
    |  
    {or}
```



Root nodes

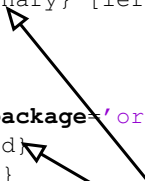
Prefix: P

# AST Generation

## AST Source File

### Abstract Syntax Tree

```
exp {-> package='org.overture.ast.expressions'}  
  = {binary} [left]:exp [op]:binop [right]:exp  
  |  ...  
  ;  
  
binop {-> package='org.overture.ast.expressions'}  
  = {and}  
  | {or}
```



### Sub nodes

Prefix: A

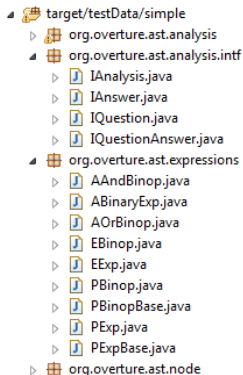
Name: Prefix+name+root name



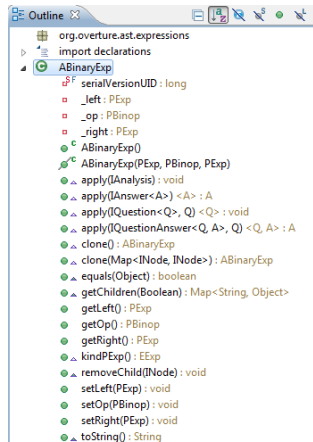
# AST Generation

## Java Output

### Package View



### Outline of ABinaryExp



# AST Generation

## Tree Traversal: Locating Roots

### Node

- `INode parent ()`
- `INode getAncestor (Class<INode> classType)`

# AST Generation

## Tree Traversal

- **Analysis**

- `void apply(IAnalysis analysis)`

- **Question**

- `<Q> void apply(IQuestion<Q> caller, Q question)`

- **Answer**

- `<A> A apply(IAnswer<A> caller)`

- **Question Answer**

- `<Q, A> A apply(IQuestionAnswer<Q, A> caller,  
Q question)`

Generated Adapters

Depth First

# AST Generation

## Tree Traversal

- Analysis

- `void apply(IAnalysis analysis)`

- Question

- `<Q> void apply(IQuestion<Q> caller, Q question)`

- Answer

- `<A> A apply(IAnswer<A> caller)`

- Question Answer

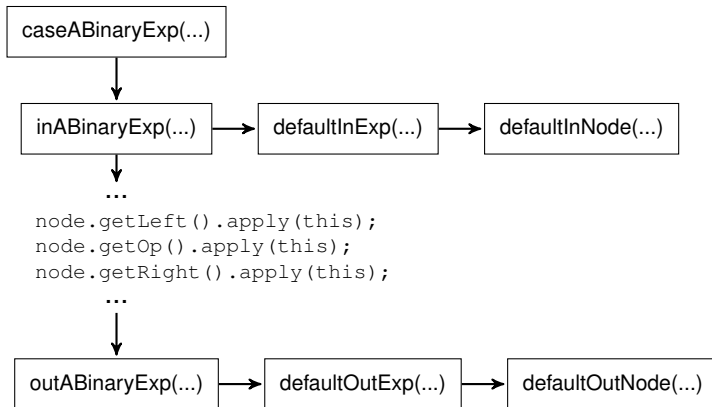
- `<Q, A> A apply(IQuestionAnswer<Q, A> caller,  
Q question)`

## Generated Adapters

### Depth First

# AST Generation

## Analysis Depth First Search

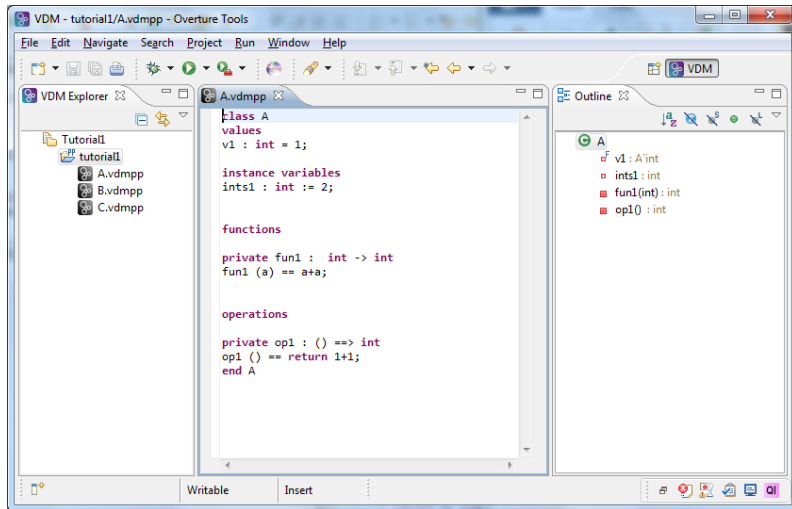


# Outline

1 Introduction

2 Creating Plug-ins For Overture

# Creating a plug-in



# Creating a plug-in

## Development Environment

### Requirements:

- Java 7 SDK
- Eclipse Standard
  - Extended with the Overture core feature



# Creating a plug-in

## Development Environment - Setup

Download Eclipse Standard - [www.eclipse.org](http://www.eclipse.org)



**Eclipse Standard 4.3**, 196 MB

Downloaded 1,139,719 Times [Other Downloads](#)



[Mac OS X 32 Bit](#)

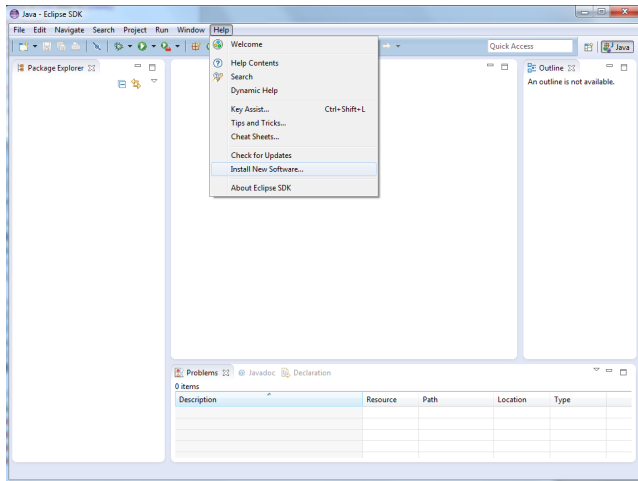
[Mac OS X 64 Bit](#)

The Eclipse Platform, and all the tools needed to develop and debug it: Java and Plug-In Development Tooling, Git and CVS...

# Creating a plug-in

## Development Environment - Setup

### Install New Software



# Creating a plug-in

## Development Environment - Setup

Add the Overture repository

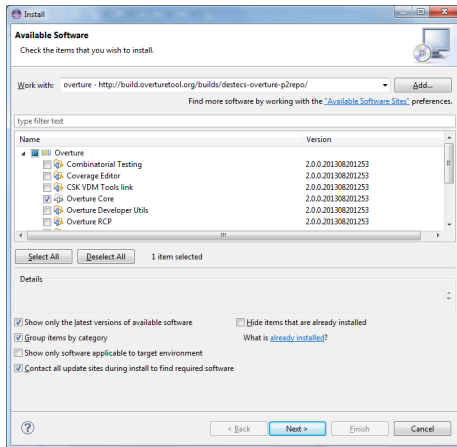
**Name:** Overture

**Location:** `http://overture.sourceforge.net/updates/release/`

# Creating a plug-in

## Development Environment - Setup

### Select **Overture Core** and install



# Tutorial

## Simple Analysis Plug-in

- For each class count:
  - Values
  - Instance Variables
  - Functions
  - Operations
- For each Operation and Function count the total number of expressions and statements

# Tutorial

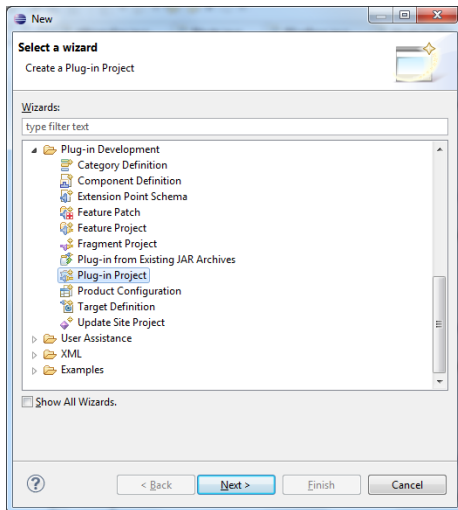
## Overview

What do we need to do?

- 1 Create a plug-in project
- 2 Add dependencies to Eclipse and Overture
- 3 Add a command to invoke the functionality
- 4 Add a menu to show the command in the UI
- 5 Add a Handler to invoke the analysis
- 6 Add a view to show the result
- 7 Add the Overture-specific VDM analysis code

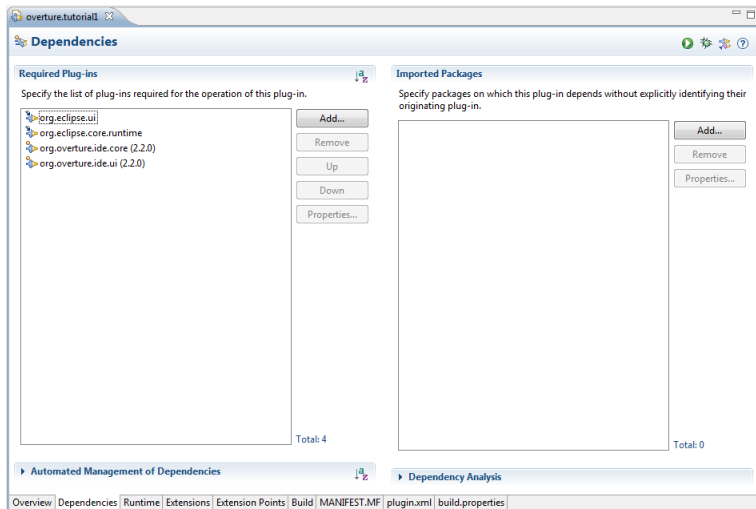
# Tutorial

## Create plug-in project



# Tutorial

## Add dependencies





# Tutorial





## Add dependencies

### Dependencies

#### Required Plug-ins



Specify the list of plug-ins required for the operation of this plug-in.

-  org.eclipse.ui
-  org.eclipse.core.runtime
-  org.overture.ide.core (2.2.0)
-  org.overture.ide.ui (2.2.0)

Add...

Remove

Up

Down

Properties...

#### Imported Packages

Specify packages on which this plug-in originates.

# Tutorial

## Add a command

```
<extension
    point="org.eclipse.ui.commands">
    <command
        id="overture.tutorial1.commandAnalysis"
        name="Analysis">
    </command>
</extension>
```

# Tutorial

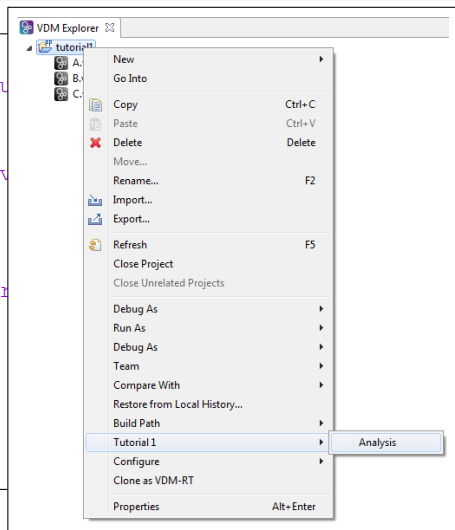
## Add a menu

```
<extension
    point="org.eclipse.ui.menus">
<menuContribution
    allPopups="false"
    locationURI="popup:org.overture.ide.ui.VdmExplorer">
<menu
    label="Tutorial 1">
    <command
        commandId="overture.tutorial1.commandAnalysis"
        label="Analysis"
        style="push">
    </command>
</menu>
</menuContribution>
</extension>
```

# Tutorial

## Add a menu

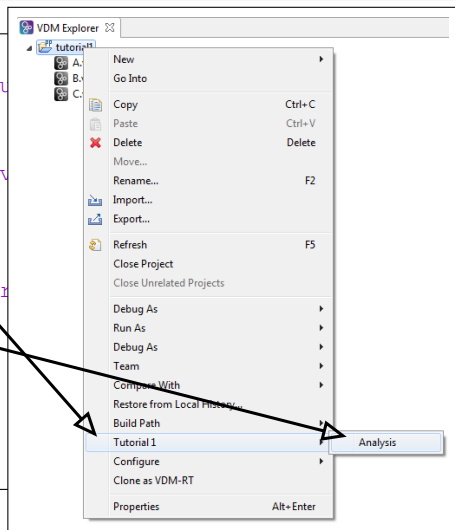
```
<extension
    point="org.eclipse.ui"
    <menuContribution
        allPopups="false"
        locationURI="popup:org.ov
    <menu
        label="Tutorial 1">
        <command
            commandId="overture
            label="Analysis"
            style="push">
        </command>
    </menu>
</menuContribution>
</extension>
```



# Tutorial

## Add a menu

```
<extension
    point="org.eclipse.ui"
    <menuContribution
        allPopups="false"
        locationURI="popup:org.ov
    <menu
        label="Tutorial 1">
        <command
            commandId="overture
            label="Analysis"
            style="push">
        </command>
    </menu>
</menuContribution>
</extension>
```



# Tutorial

## Add a Handler

```
<extension
  point="org.eclipse.ui.handlers">
  <handler
    class="org.overture.ide.analysis.AnalysisHandler"
    commandId="overture.tutorial1.commandAnalysis">
  </handler>
</extension>
```

# Tutorial

## Handler Implementation

```
public class AnalysisHandler extends AbstractHandler
{
    public Object execute(ExecutionEvent event) {
        ISelection selection = HandlerUtil.getCurrentSelection(event);
        IStructuredSelection structuredSelection = (IStructuredSelection)
            selection;

        Object firstElement = structuredSelection.getFirstElement();

        IProject p = ((IResource)firstElement).getProject();

        IVdmProject vdmProject =
            (IVdmProject) p.getAdapter(IVdmProject.class);

        a = new VdmAnalysis(vdmProject, HandlerUtil.getActivePart(event),
                           HandlerUtil.getActiveShell(event));
        a.runAnalysis();

        ....
    }
}
```

# Tutorial

## Handler Implementation

Get Selection through  
HandlerUtil

```

Handler extends AbstractHandler

public Object execute(ExecutionEvent event) {
    ISelection selection = HandlerUtil.getCurrentSelection(event);
    IStructuredSelection structuredSelection = (IStructuredSelection)
        selection;

    Object firstElement = structuredSelection.getFirstElement();

    IProject p = ((IResource)firstElement).getProject();

    IVdmProject vdmProject =
        (IVdmProject) p.getAdapter(IVdmProject.class);

    a = new VdmAnalysis(vdmProject, HandlerUtil.getActivePart(event),
        HandlerUtil.getActiveShell(event));

    a.runAnalysis();

    ....
}

```



# Tutorial

## Handler Implementation

Get Selection through  
HandlerUtil

Handler **extends** AbstractHandler

```
public Object execute(ExecutionEvent event) {
    ISelection selection = HandlerUtil.getCurrentSelection(event);
    IStructuredSelection structuredSelection = (IStructuredSelection)
```

Get Project

```
Object firstElement = structuredSelection.getFirstElement();
```

```
IProject p = ((IResource)firstElement).getProject();
```

```
IVdmProject vdmProject =
    (IVdmProject) p.getAdapter(IVdmProject.class);
```

```
a = new VdmAnalysis(vdmProject, HandlerUtil.getActivePart(event),
    HandlerUtil.getActiveShell(event));
```

```
a.runAnalysis();
```

```
....
```

# Tutorial

## Handler Implementation

Get Selection through  
HandlerUtil

Handler **extends** AbstractHandler

```
public Object execute(ExecutionEvent event) {
    ISelection selection = HandlerUtil.getCurrentSelection(event);
    IStructuredSelection structuredSelection = (IStructuredSelection)
```

Get Project

```
Object firstElement = structuredSelection.getFirstElement();
```

```
IProject p = ((IResource)firstElement).getProject();
```

```
IVdmProject vdmProject =
    (IVdmProject) p.getAdapter(IVdmProject.class);
```

```
a = new VdmAnalysis(vdmProject, HandlerUtil.getActivePart(event),
    HandlerUtil.getActiveShell(event));
a.runAnalysis();
```

```
....
```

Adapt selection to a VDM Project

# Tutorial

## View

```
<extension
    point="org.eclipse.ui.views">
    <view
        class="org.overture.ide.analysis.AnalysisViewPart1"
        id="overture.tutorial1.view1"
        name="Analysis View"
        restorable="true">
    </view>
</extension>
```

The implementation takes a data object and displays it in a table view. (Not shown here)

# Tutorial

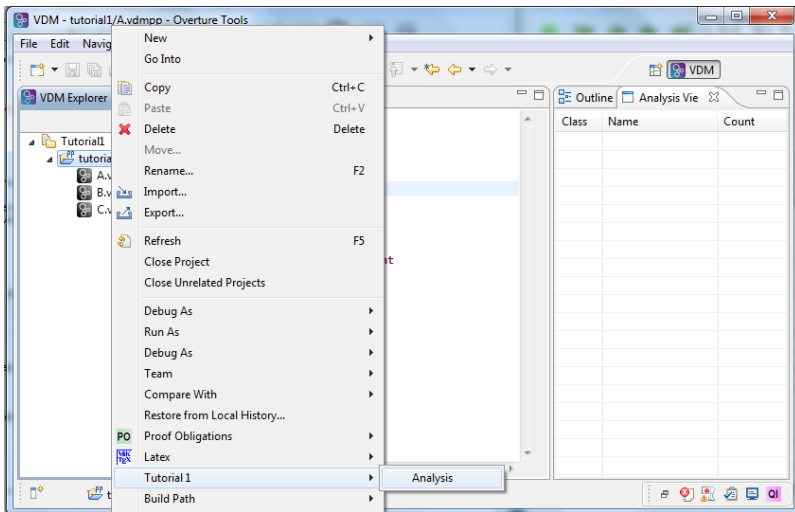
## View

```
<extension
    point="org.eclipse
    <view
        class="org.overture
        id="overture.tutori
        name="Analysis View
        restorable="true">
    </view>
</extension>
```

Analysis View		
Class	Name	Count
A	Values	1
A	Instance Variables	1
A	Operations	1
A	Functions	1
A	fun1	3
A	op1	4
B	Values	1
B	Instance Variables	0
B	Operations	1
B	Functions	1
B	fun1	6
B	op1	7
C	Values	1
C	Instance Variables	0
C	Operations	1
C	Functions	0
C	opTest1	6

# Tutorial

## Status



# Tutorial

## Overture Analysis Implementation

### Check the model and run the Type Checker:

```
final IVdmModel model = project.getModel();  
if (model != null && model.isParseCorrect())  
{  
    if (!model.isTypeCorrect())  
    {  
        VdmTypeCheckerUi.typeCheck(shell, project);  
    }  
}
```

### Run the analysis:

```
if (model.isTypeCorrect())  
{  
    AnalysisData data = analyse(model.getRootElementList());  
}
```

# Tutorial

## Overture Analysis Implementation 2

```
private AnalysisData analyse(List<INode> rootElementList){  
    AnalysisVisitor visitor = new AnalysisVisitor();  
  
    for (INode node : rootElementList)  
    {  
        try  
        {  
            node.apply(visitor);  
        } catch (AnalysisException e){  
        }  
    }  
  
    return visitor.data;  
}
```

# Tutorial

## Overture Analysis Implementation 3 Visitor

```
class AnalysisVisitor extends DepthFirstAnalysisAdaptor
{

    void inAClassClassDefinition(AClassClassDefinition node){
        data.initClass(node.getName().name);
    }

    void inAExplicitFunctionDefinition(...){
        expCount = 0;
    }

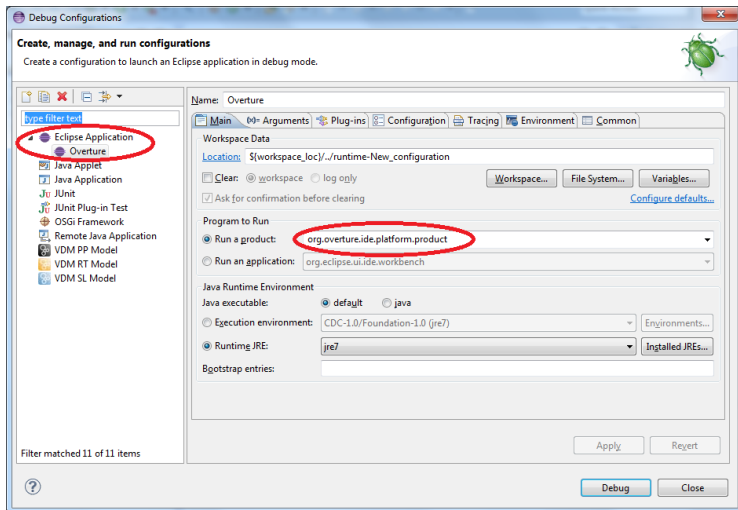
    void outAExplicitFunctionDefinition(... node){
        data.fun.get(data.activeClass).put(node.getName().name, expCount);
    }

    void defaultInPExp(PExp node){
        expCount++;
    }
}
```



# Creating a plug-in

## Launching the Debugger



# Creating a plug-in

## Final plug-in

The screenshot displays the Overture IDE interface with the VDM plug-in. The main window is titled "VDM - tutorial1/A.vdmpp - Overture Tools". The menu bar includes File, Edit, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations, execution, and navigation.

The VDM Explorer on the left shows a project structure with "Tutorial1" containing "tutorial1", which in turn contains "A.vdmpp", "B.vdmpp", and "C.vdmpp".

The central editor displays the source code for "A.vdmpp":

```
class A
values
v1 : int = 1;

instance variables
ints1 : int := 2;

functions

private fun1 : int -> int
fun1 (a) == a+a;

operations

private op1 : () ==> int
op1 () == return 1+1;
end A
```

The Outline/Analysis View on the right shows a table summarizing the code elements:

Class	Name	Count
A	Values	1
A	Instance Variables	1
A	Operations	1
A	Functions	1
A	fun1	3
A	op1	4
B	Values	1
B	Instance Variables	0
B	Operations	1
B	Functions	1
B	fun1	6
B	op1	7
C	Values	1
C	Instance Variables	0
C	Operations	1
C	Functions	0
C	opTest1	6

# Creating a plug-in

## Tutorial 2: Extension

Try to extend the example given in tutorial 1 by:

- Count expressions in initialization of:
  - Values
  - Instance Variables
- Enable the analysis of Modules
- ...

# Thanks

You can download the tutorial at:

`http://tinyurl.com/overture11tutorial`

Wiki Overture Workshop 11