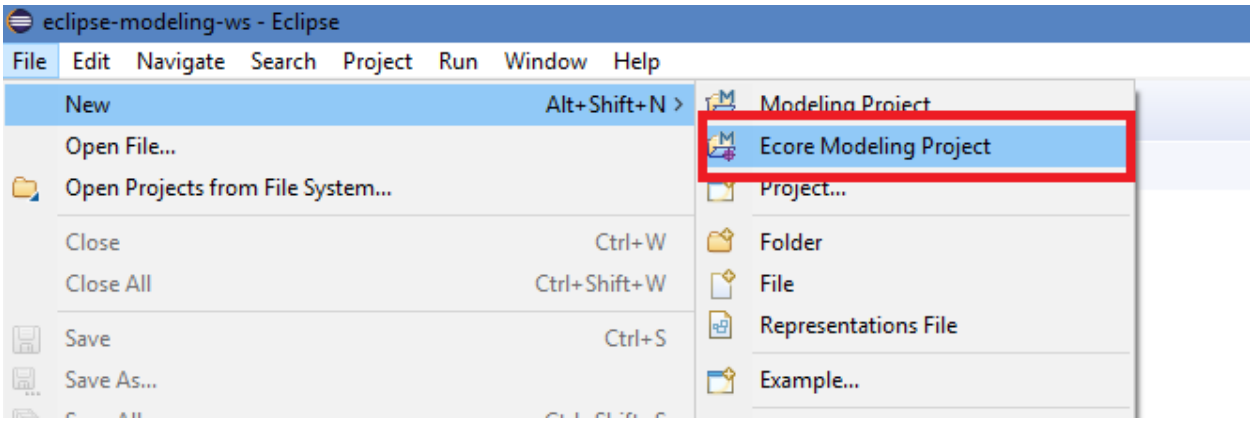
Domain-Specific Modeling Language (DSML), tooling and instantiating using EMF and ECore

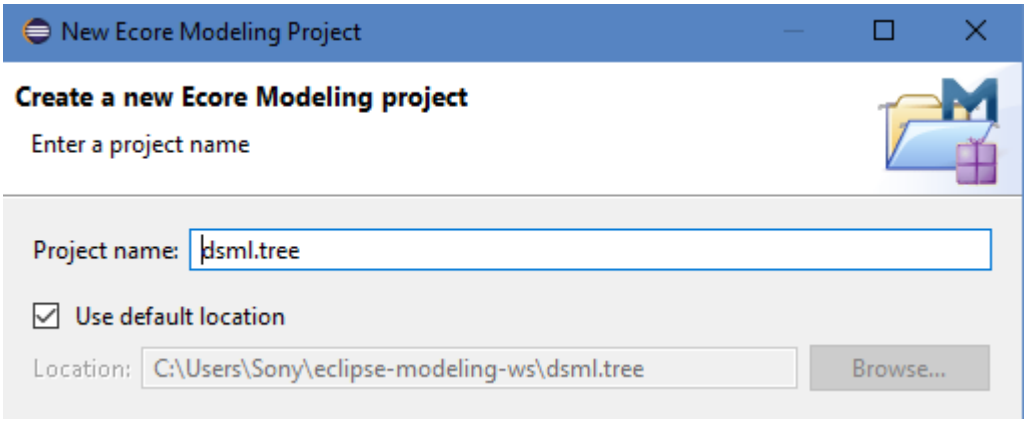
Prerequisites: *This tutorial assumes that the readers have knowledge of domain modeling and the known N-Array Tree. The tutorial is focused on creating DSML of N-Array Tree, tooling the DSML, Instantiating the meta-model both programmatically and using the modeling tool, and reading the instance of DSML in Java code. The tutorial is based on the latest Oxygen Eclipse for modeling (Sep 2017 Release).*

DSML Modeling:

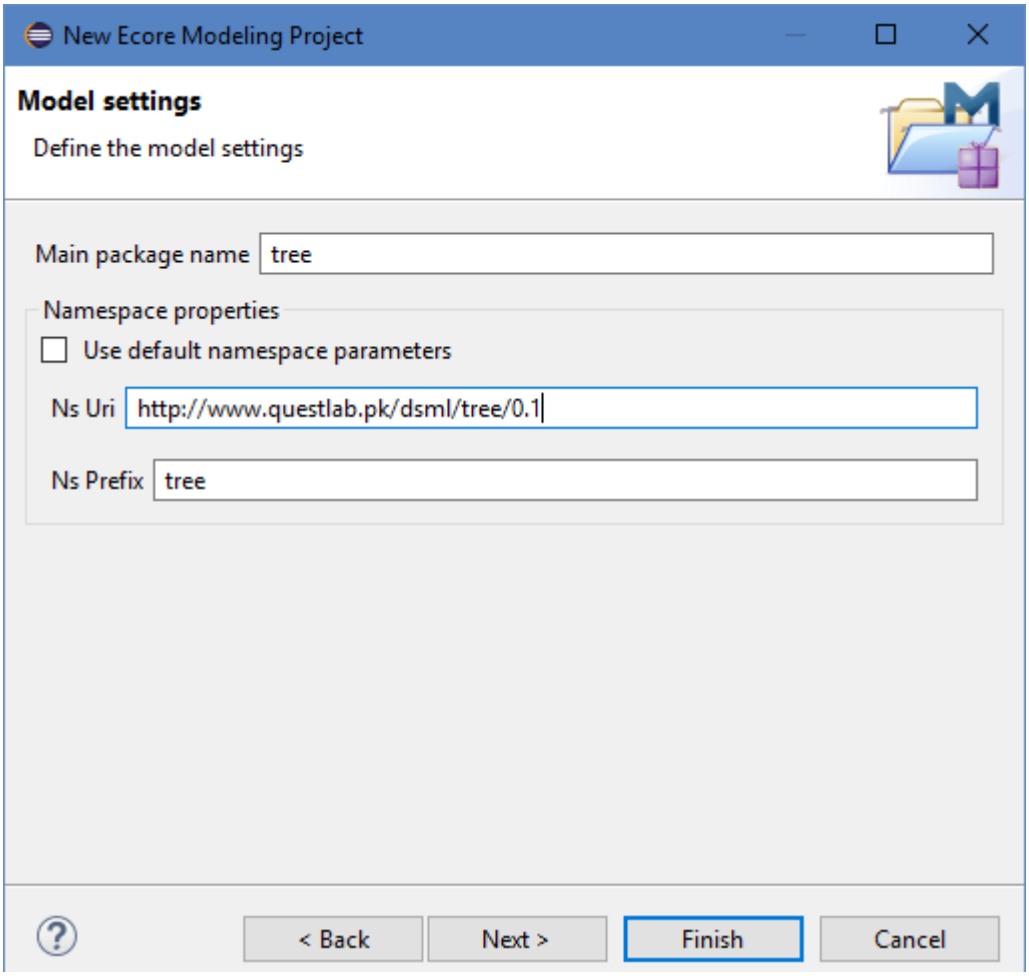
The first step of creating a DSML is to create an appropriate project for the DSML. As we are creating our own DSML (Not an extension to UML), the appropriate project in this case is an ecore modeling project.



Name the project and click next.

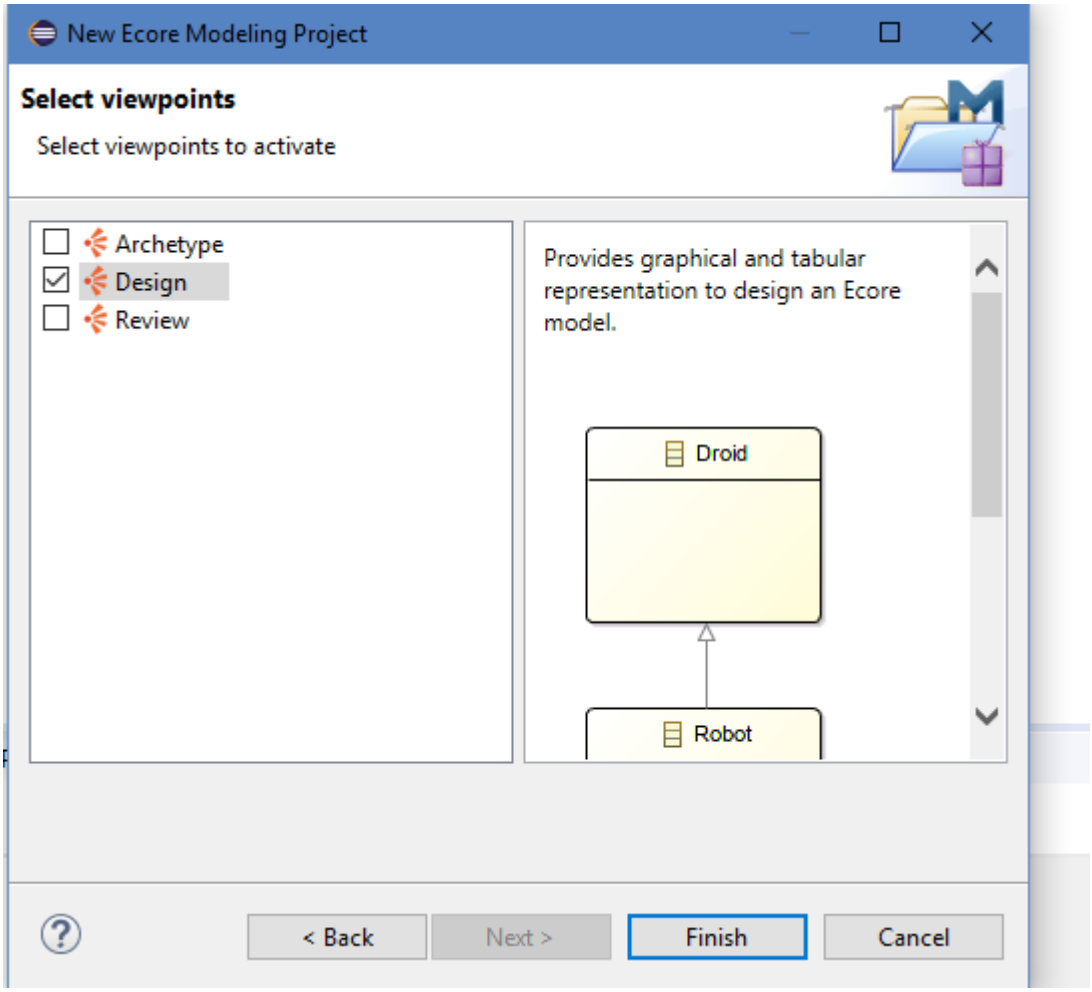


Assign appropriate namespace URI, prefix and hit next.

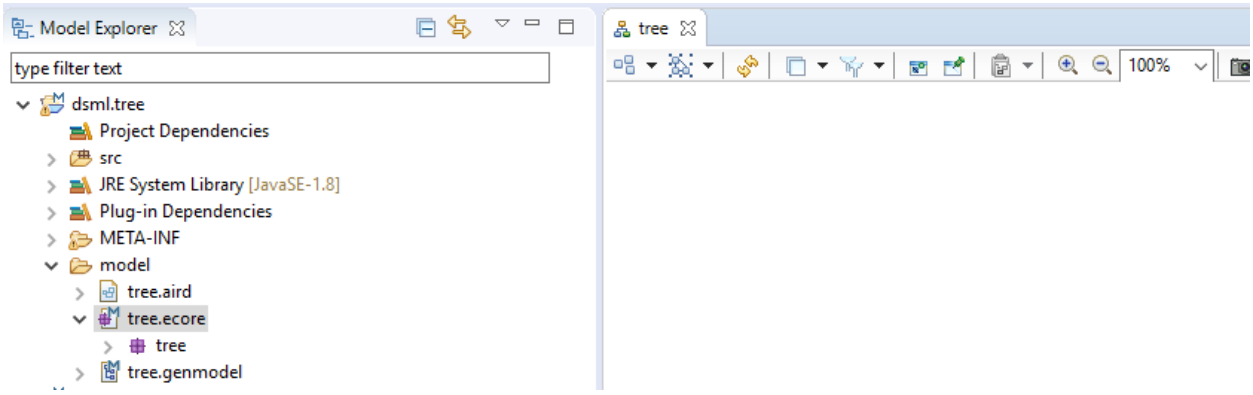


A URI typically consists of a domain name you own, the name of the DSML and version of the DSML.

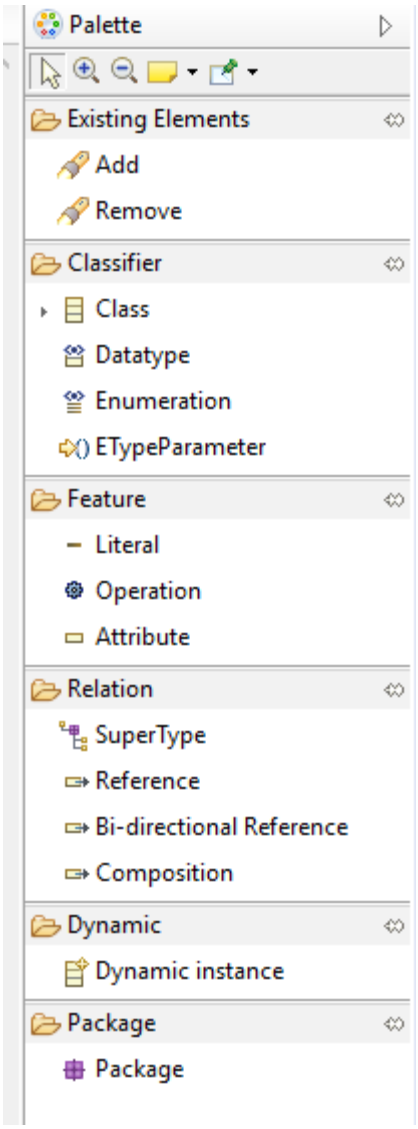
As the new eclipse for modeling now uses the concept of Viewpoints (Not focus of this tutorial, interested readers can see Sirius) make sure your selected viewpoint is only Design and then click finish.



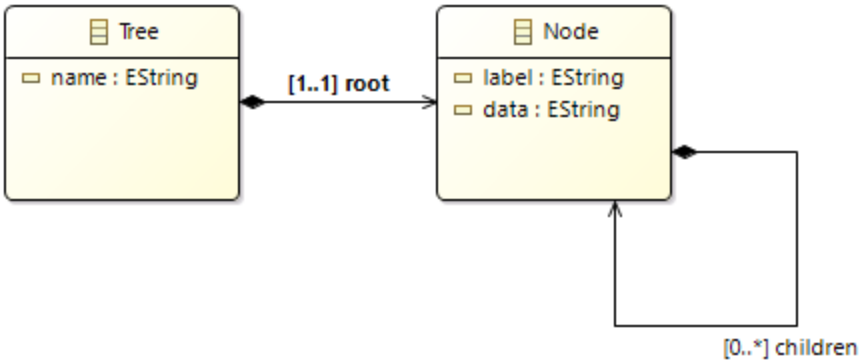
Make sure you project tree looks like the following and a labeled modeling window of your DSML prefix is opened. You will note that the project created have an ecore, aird and a genmodel file. The aird provides means for visualizing data models (ecore in this case), the ecore models will have our meta-model and the genmodel will have the necessary information for generating meta-model code.



On the right-most side you will have all the available ecore meta-modeling elements as follow:

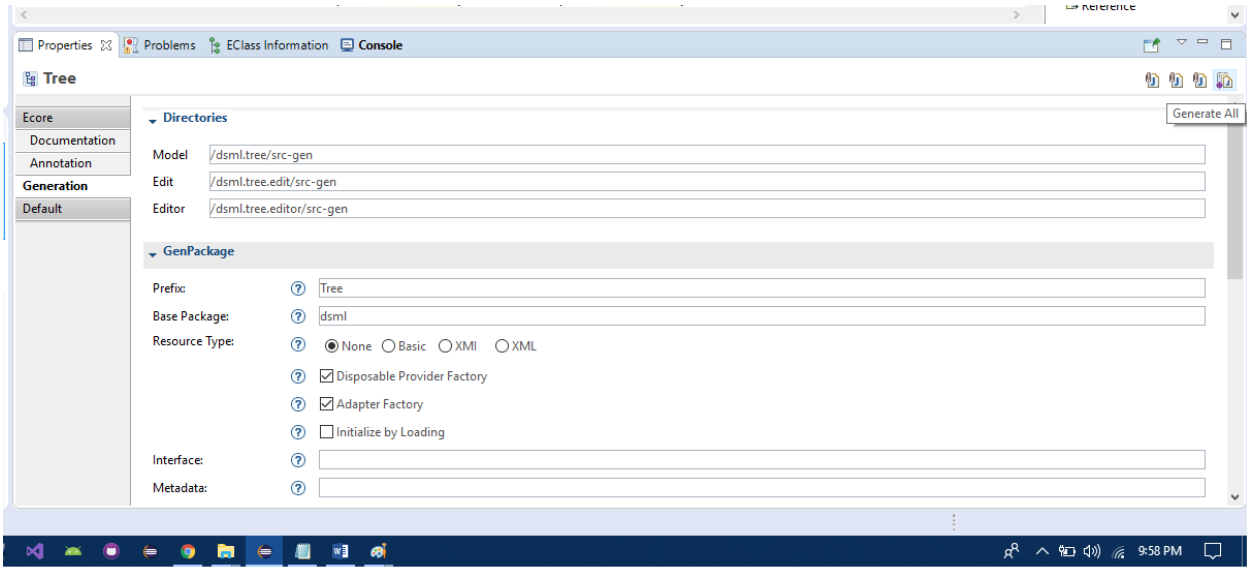


As we know that a tree usually has a root, the root is basically of type Node and the Node can have N-Number of children and these children can have further N-Number of children and so on, so bellow is the meta model for an N-Array Tree:

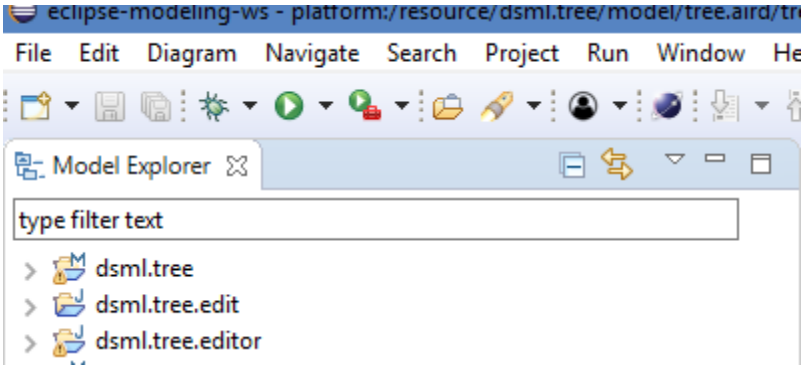


DSML Tooling & Instantiating through the new tool:

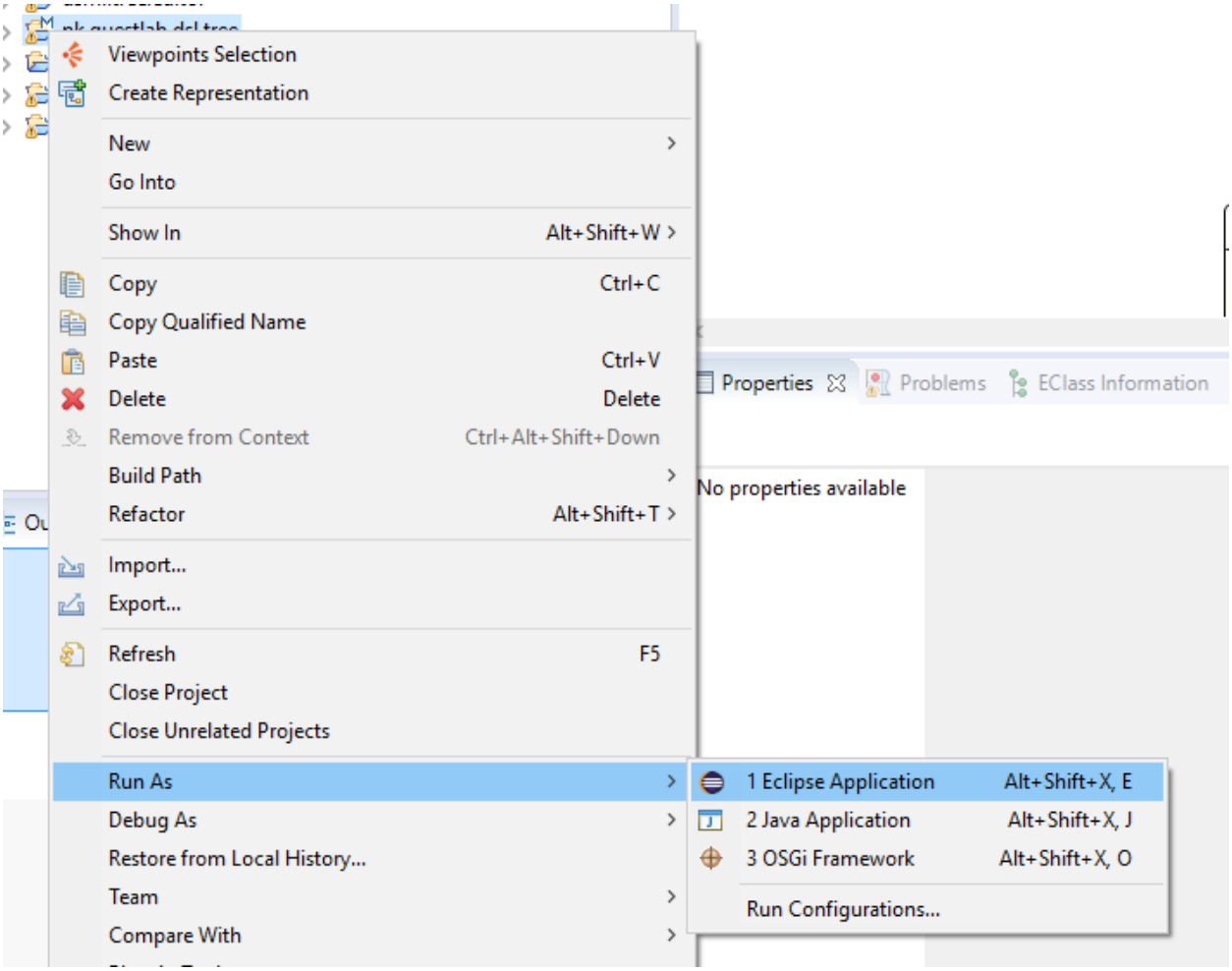
After modeling the concepts of the modeling language now we need to generate code for the tooling and edit (to be use in Java). The tooling code helps in providing end user eclipse based tool for modeling instances of the meta-model. Please note that the tool will use tree editor to visualize the instances of our newly created DSML (As Graphical Modeling Framework is out of the scope of this tutorial). To generate code for tool and edit navigate to the properties window of the root element in your genmodel and click on the generate all button in the right top corner of the properties window.



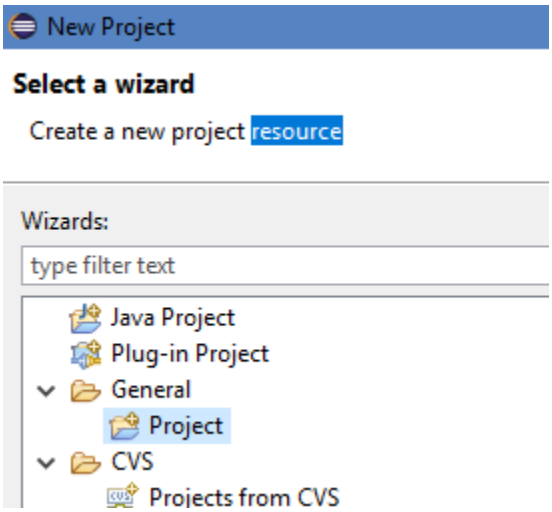
After the process is complete you will be able to see the following folders in your workspace:



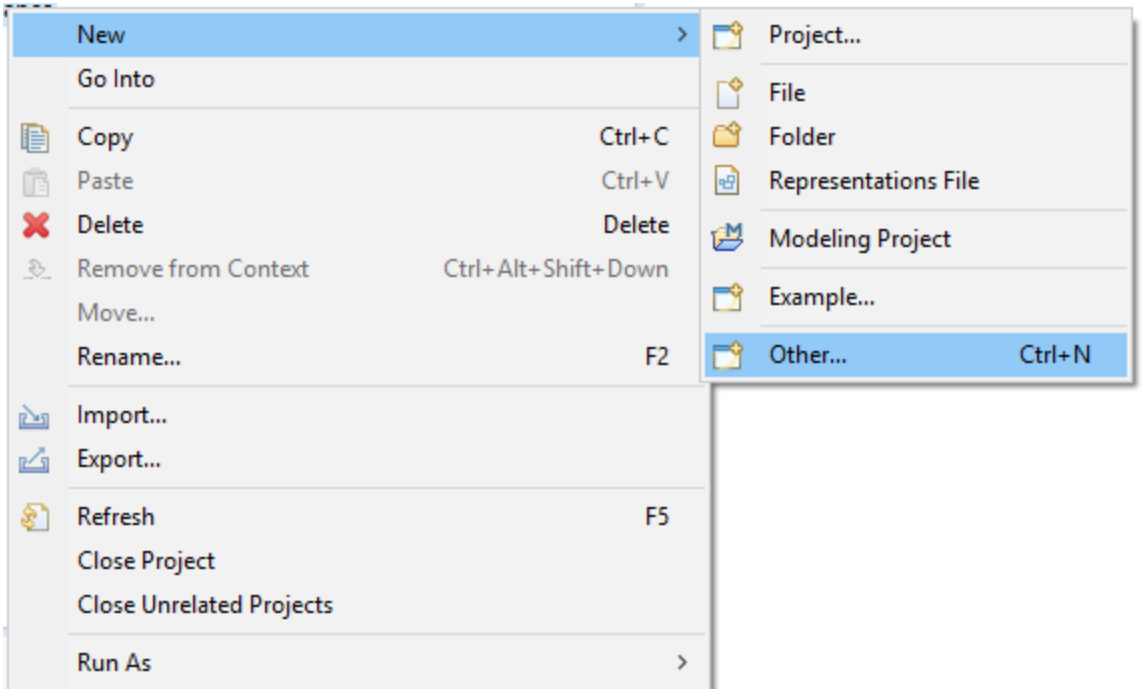
Now you can run the .editor folder as an eclipse application as follow:



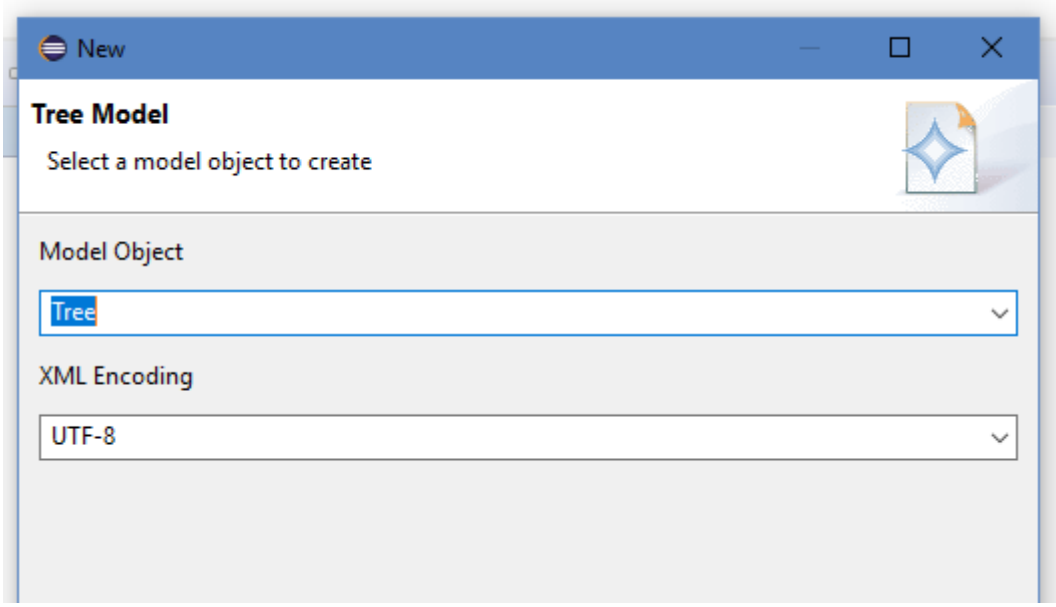
After running the code as an eclipse application create a new general project in the runtime instance of the eclipse application you just launched (File->new->project) and name it as “MyInstance”.



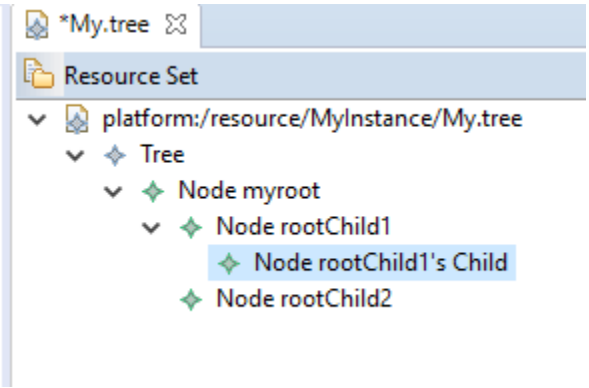
Now right click on your project's folder in your workspace and navigate to add new and click other:



Search for the prefix of your DSML and you will find a model wizard for your DSML. Name the object model as My.tree and select the MyInstance folder as destination. The tool may ask you to select the root model element, in this case you should select the Tree as root model element.



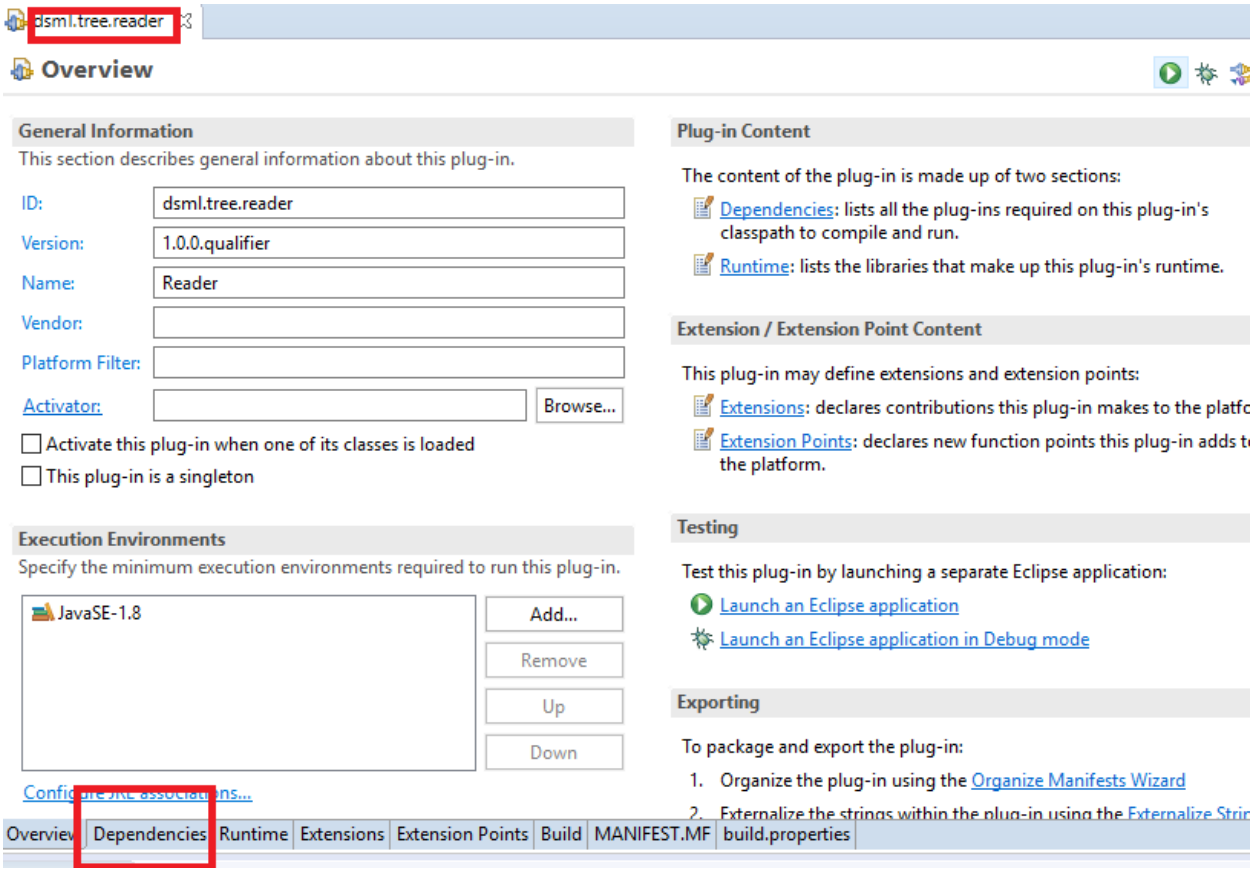
Edit your My.tree in the tree editor



In the above model I created a root with child and one child have its own child. As per now we have just instantiated our meta-model. For setting values in the model navigate to the properties window of any model element. For reading this instance programmatically, copy the My.tree file form the run-time workspace and follow the rest of the tutorial.

Loading & Reading the Instance Model using EMF in Java

You now need to change the eclipse (source eclipse not the runtime instance) perspective from modeling to plug-in development. Create a new plug-in project after changing the perspective. Add the .edit and ecore as plugin dependencies to your new project as follow:



As you can see I have created a plugin project with the name "dsml.tree.reader". Navigate to the dependencies tab to add plugin dependencies. I have also created a folder named "models" in the new plugin project and pasted My.tree file there.

Now the rest of the code is the same as per the EMF demo. We just need few changes like adding .tree extension as:

```
Map extensionFactoryMap =
org.eclipse.emf.ecore.resource.Resource.Factory.Registry.INSTANCE.getExtensionToFacto
ryMap();
            extensionFactoryMap.put("tree", new TreeResourceFactoryImpl());
```

We also need to register our ecore and our DSML packages as follow:

```
Map packageRegistry = resourceSet.getPackageRegistry();
            packageRegistry.put(EcorePackage.eNS_URI, EcorePackage.eINSTANCE);
            packageRegistry.put(TreePackage.eNS_URI, TreePackage.eINSTANCE);
```

Below is the output of the code:

```
Node: myroot Data: 10
Node: rootChild1 Data: 11
Node: rootChild1's Child Data: 11
Node: rootChild2 Data: 12
```

As our language deals with the recursive manner of N-Array Tree I have written a recursive method to print the tree. Below is my complete class for printing instance:

```
package dsml.tree.reader;

import java.io.File;
import java.io.IOException;
import java.util.Map;

import org.eclipse.emf.common.util.URI;
import org.eclipse.emf.ecore.EcorePackage;
import org.eclipse.emf.ecore.resource.ResourceSet;
import org.eclipse.emf.ecore.resource.impl.ResourceSetImpl;
import org.eclipse.emf.ecore.util.EcoreUtil;
import pk.questlab.dsl.tree.util.TreeResourceFactoryImpl;
import dsml.tree.Node;
import dsml.tree.Tree;
import dsml.tree.TreePackage;
import dsml.tree.impl.TreeFactoryImpl;
import dsml.tree.*;
import dsml.tree.util.*;

public class DSMLInstanceReader {

	private final static ResourceSet RESOURCE_SET=new ResourceSetImpl();;
	public static void main(String[] args)
	{
		Object rootModelElement= loadModel("models/My.tree");
		if(rootModelElement instanceof Tree)
		{
			Tree tree= (Tree) rootModelElement;
			Node root=tree.getRoot();
			printTreeNodesRec(root);
		}
	}
	public static void printTreeNodesRec(Node r)
	{
		System.out.println("Node: "+r.getLabel()+" Data: "+r.getData());
		for(Node n:r.getChildren())
		{
			if(n.getChildren().isEmpty())
				System.out.println("Node: "+n.getLabel()+" Data:
"+n.getData());
			else
			{
				System.out.println("Node: "+n.getLabel()+" Data:
"+n.getData());
				for(Node sub_n:n.getChildren())
					printTreeNodesRec(sub_n);
			}
		}
	}

	public static Object loadModel(String uri){
		URI modelUri = URI.createURI(uri);
```

```
		registerPackages(RESOURCE_SET);
		registerResourceFactories();
		String relPath = null;
		try {
			relPath = new File(".").getCanonicalPath();
		} catch (IOException e1) {
			e1.printStackTrace();
		}
		org.eclipse.emf.ecore.resource.Resource resource = null;
		try {
			resource = RESOURCE_SET.getResource(modelUri, true);
		}
		catch (Exception e) {
			e.printStackTrace();
		}

		Object result;

		Tree _model = (Tree) EcoreUtil.getObjectByType(resource.getContents(),
TreePackage.Literals.TREE);
		result = _model;
		if (_model == null) {
			result = resource.getContents().get(0);
		}
		return result;
	}

	private static void registerResourceFactories()
	{
		Map extensionFactoryMap =
org.eclipse.emf.ecore.resource.Resource.Factory.Registry.INSTANCE.getExtensionToFacto
ryMap();
		extensionFactoryMap.put("tree", new TreeResourceFactoryImpl());
	}

	private static void registerPackages(ResourceSet resourceSet)
	{
		Map packageRegistry = resourceSet.getPackageRegistry();
		packageRegistry.put(EcorePackage.eNS_URI, EcorePackage.eINSTANCE);
		packageRegistry.put(TreePackage.eNS_URI, TreePackage.eINSTANCE);
	}
}
```

Instantiating the DSML programmatically:

In this part of the tutorial we will try to instantiate our DSML using the generated factory in java. This can be helpful in many cases (such as model transformation). As the code is very simple to understand you can just see the following class which instantiates and print the tree DSML.

```
package dsml.tree.reader;

import dsml.tree.Node;
import dsml.tree.Tree;
import dsml.tree.TreeFactory;


public class DSMLDynamicIns {
	public static void main(String[] args)
	{
		TreeFactory treeFactory= TreeFactory.eINSTANCE;
		Tree myTree=treeFactory.createTree();
		myTree.setName("My Tree");
		Node myroot= treeFactory.createNode();
		myroot.setData("10");
		myroot.setLabel("myroot");
		Node childNode= treeFactory.createNode();
		childNode.setLabel("rootChild1");
		childNode.setData("11");
		myroot.getChildren().add(childNode);
		Node tempNode1= treeFactory.createNode();
		tempNode1.setLabel("rootChild2");
		tempNode1.setData("13");
		myroot.getChildren().add(tempNode1);
		myTree.setRoot(myroot);
		printTreeNodesRec(myTree.getRoot());
	}
	public static void printTreeNodesRec(Node r)
	{
		System.out.println("Node: "+r.getLabel()+" Data: "+r.getData());
		for(Node n:r.getChildren())
		{
			if(n.getChildren().isEmpty())
				System.out.println("Node: "+n.getLabel()+" Data:
"+n.getData());
			else
			{
				System.out.println("Node: "+n.getLabel()+" Data:
"+n.getData());
				for(Node sub_n:n.getChildren())
					printTreeNodesRec(sub_n);
			}
		}
	}
}
```