GMF Exercise 2 Question 1

Use GMF tooling to define and generate a constraint so a connection cannot have a Start node as target, and it cannot have an end node as a source. First construct the OCL for these constraint.s Second, model them in the gmfmap model.

You can use your own projects from last exercise or base this exercise on the provided solution from last time.

Solution hints

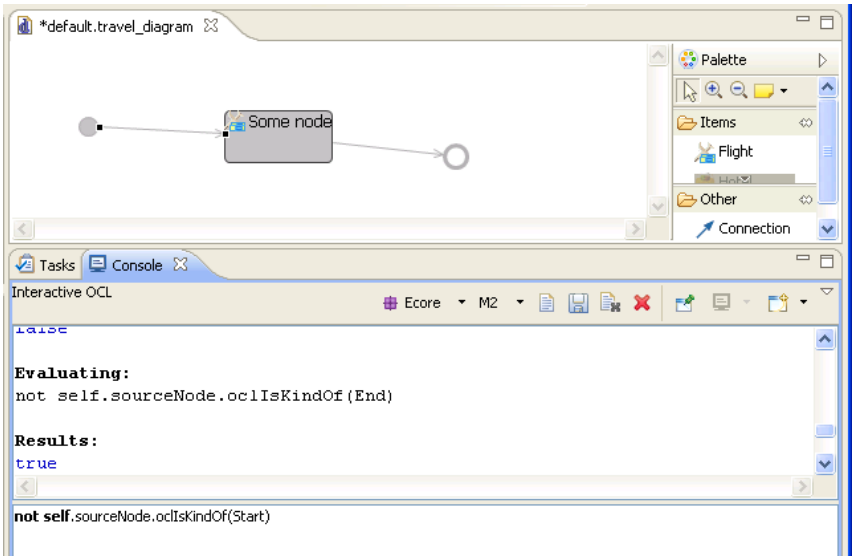
Optionally Import projects into workspace

- Download the complete solution from the first GMF lecture from the blog.
- File-> Import->General->Existing projects into workspace->Select archive file->select the solution zip file>->press finish

The solution projects will now be in the workspace

Optionally find the correct OCL statement using the interactive OCL console

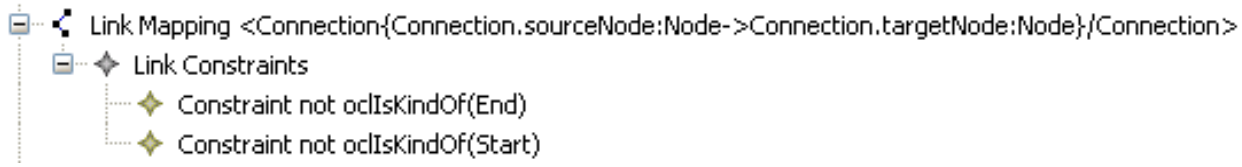
- Start runtime workbench
- Show console from Windows->Show view->Other->General->Console.
- In the console view menu, select ->interactive OCL
- Select a connection in a travel model and start writing OCL at the bottom part of the console. Press enter to evaluate



- The OCL constraint statements for the Connection are
    - **not self**.sourceNode.oclIsKindOf(End)
    - **not self**.targetNode.oclIsKindOf(Start)

Define constraints in GMF model

- Open the travel.gmfmap model
- Select Link mapping Constraint, right-click->add child->Link Constraint
- Add a Target end constraint to the Link constraint and enter the OCL statement in the constraint body. The target node is the context. Therefore the OCL statement is
    - not oclIsKindOf(Start)
- Add a Source end constraint to the Link constraint and enter the OCL statement in the constraint body. The source node is the context. Therefore the OCL statement is
    - not oclIsKindOf(End)



Generate and test

- Regenerate .gmfgen model from .gmfmap model
- Generate diagram code
- Run in runtime workbench

When GMF generates the generator model from the mapping model, it will evaluate the OCL statements. If they are invalid, the generation will stop and you will get an error message.

- Create a new Travel model and model a start, an end, and a flight node
- Try modeling a connection from an end node and see that you cannot
- Try modeling a connection from the flight node to the start node and ensure that you cannot.

GMF Exercise 2 Question 2

Use GMF tooling to create a feature initialser that adds two Parameter instances with name “travel date” and “Number of people” to a Data group, when it is created.

Solution hints

Define feature initialisers in GMF model

- In the gmfmap model, expand the Data top level node and select the Node Mapping node
- Right-click ->add child-> Feature seq initialiser
- Right-click the new feture seq initialiser -> New child -> Reference new element spec.
- In the new element, select the Feature to be Data.parameters.

This means that when when a Data instance is created, there will be generated a Parameter instance and added to the Data.parameters reference.

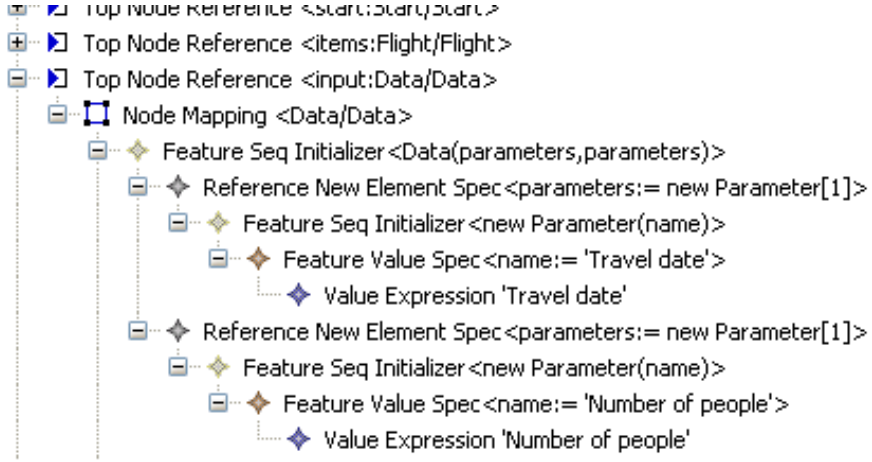
We will now set the name of the parameter by adding a feature sequence initialiser to it:

- Select the Reference new element spec node
- Right-click->add child->Feature sequence initialiser. This node is used to set the name of the parameter
- Select the new Feature sequence initialiser, right-click -> add child-> Feature value spec. Select the Parameter.name as the Feature value in the properties view.
- Select the new Feature Value spec, right-click->new child->Value Expression.
- Set the Body to 'Travel date'. Remember to set the '' as it is an OCL expression!

Now, you have defined a feature initialiser that creates a Parameter instance under a Data instance when this is created. Furthermore, you used a nested feature initialiser to set the name of the parameter.

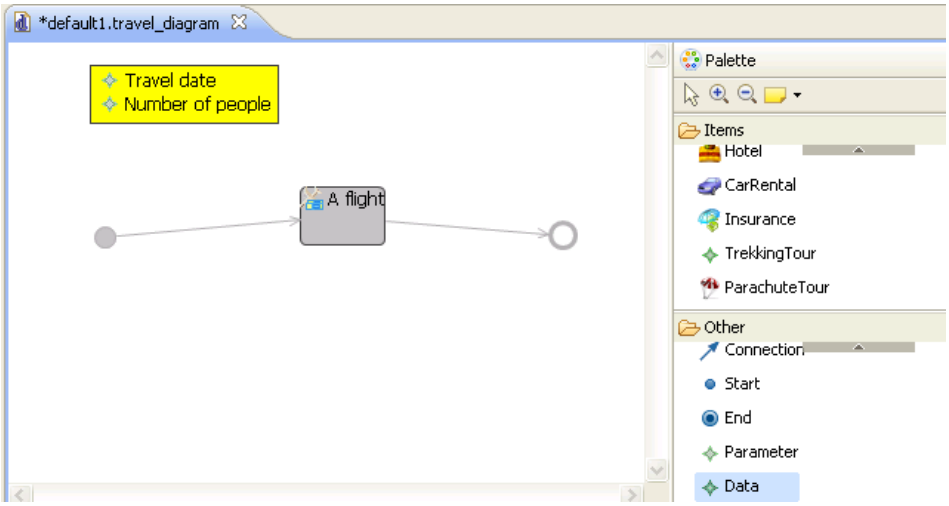
- Optionally, add one more Parameter with name "Number of people" under the first feature initialiser.
- NOTE: Both gmfmap and gmf gen will complain over two feature initialisers. Click the "ignore validation errors "and continue

Your model should look like this



Generate and test

Regenerate the gmfgen model and diagram code and start runtime workbench. Try creating a Data object and evaluate that one or two parameters with name set are created automatically as in the picture below.



GMF Exercise 2 Question 3

Define the validations we developed in the OCL lecture in the GMF map model and evaluate that we are able to see violations of the validation constraints when creating models.

Solution hints

The constraints were:

Items must have ingoing and outgoing connects
Context: item

```
Inv: self.ingoingConnections->size()>0 and self.outgoingConnections->size()>0
```

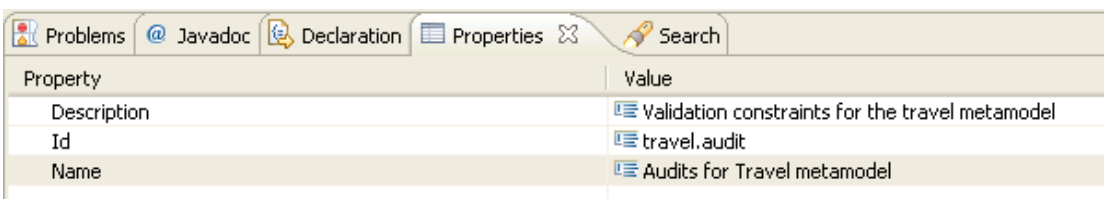
Start node must not have ingoing connections
Context start

```
Inv: self.ingoingConnections->size()=0
```

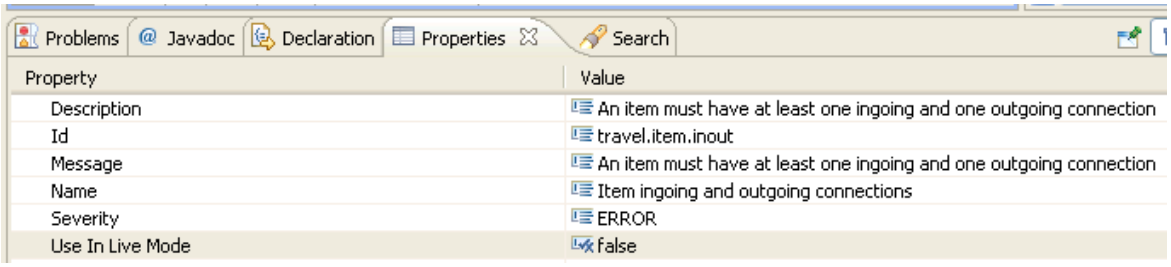
Well – this validation constraint will never be violated because of our previously defined link constraint.

CarRental item must be directly followed by an Insurance item with type== CarInsurance
Context CarRental
Inv: self.outgoingConnections->exists(targetNode.oclAsType(Insurance).type = InsuranceType::CarInsurance)

- In the gmfmap model, select the Mapping node, right-click->Add Audit container
- Define an id, name and description for the Audit container



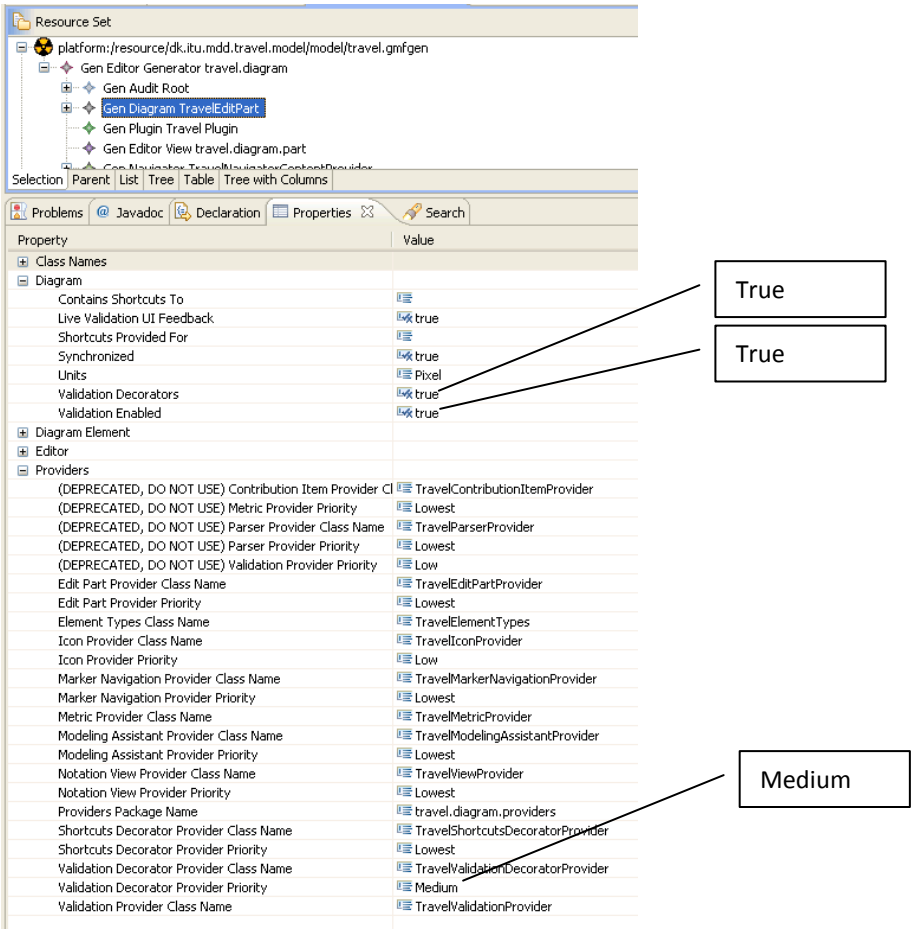
- Right-click the Audit container->New child->Audit rule
- Enter id, name, description, message for the first constraint



- Right-click the Audit rule ->New child ->Domain Element target. This element is used to define the OCL context.
- Select the Element attribute of the Domain Element Target node to be Item
- Right-click the Audit rule ->New child ->Constraint. This element is used to contain the OCL statement
- Enter the OCL statement for the first constraint into the Body parameter of the Contraint node.

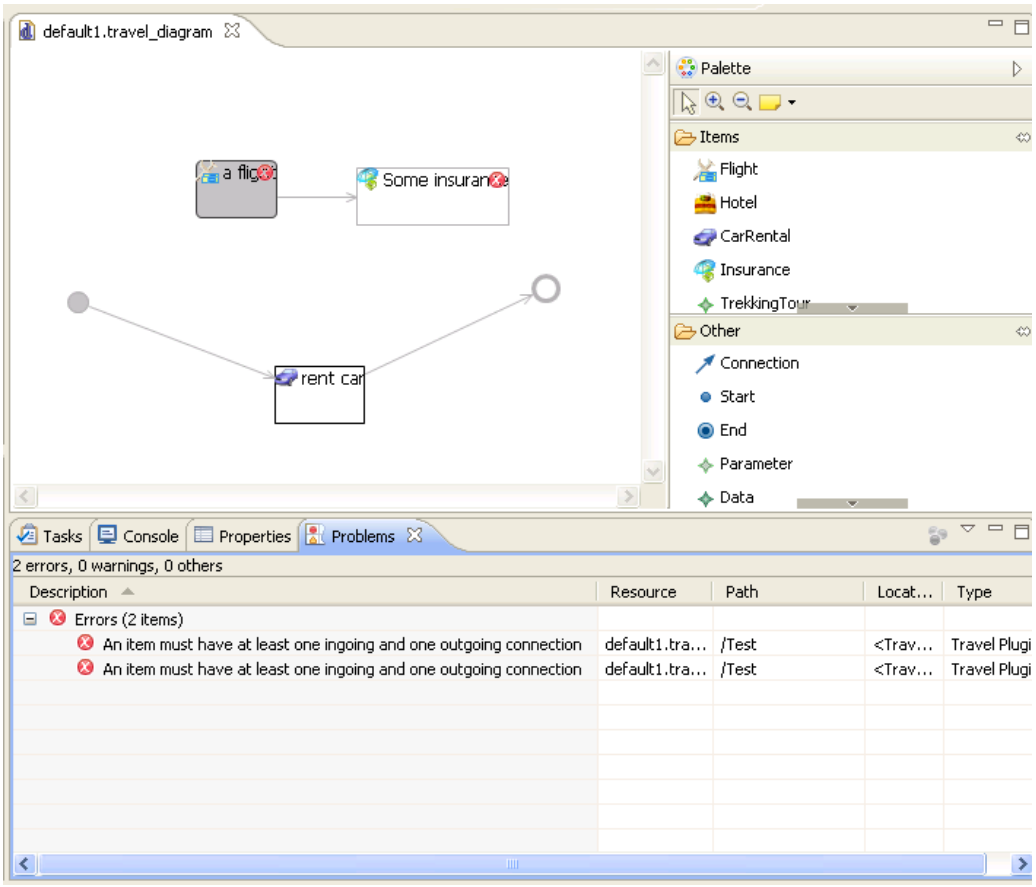
Generate and Test

- Regenerate the genmodel
- Modify genmodel as shown below



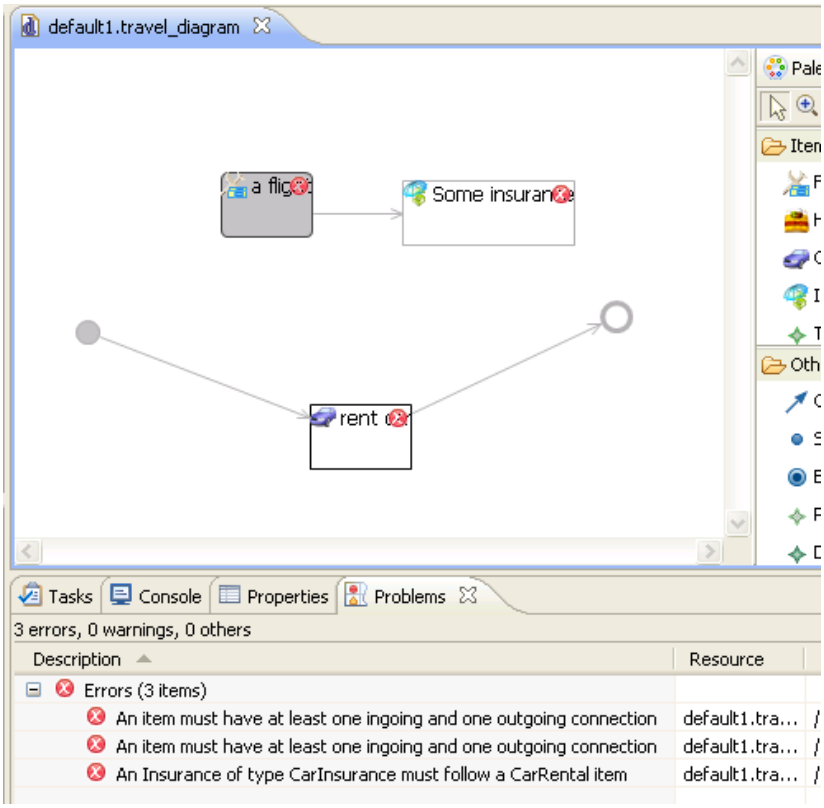
- Generate diagram code, and start runtime workbench
- Create a model and model items without connections.
- Execute validation from Edit->Validate.
- Ensure that you get red crosses in the diagram and errors in the problem view. The Problem view can be shown from Window->show view->other->General->problem view

You should see something like below

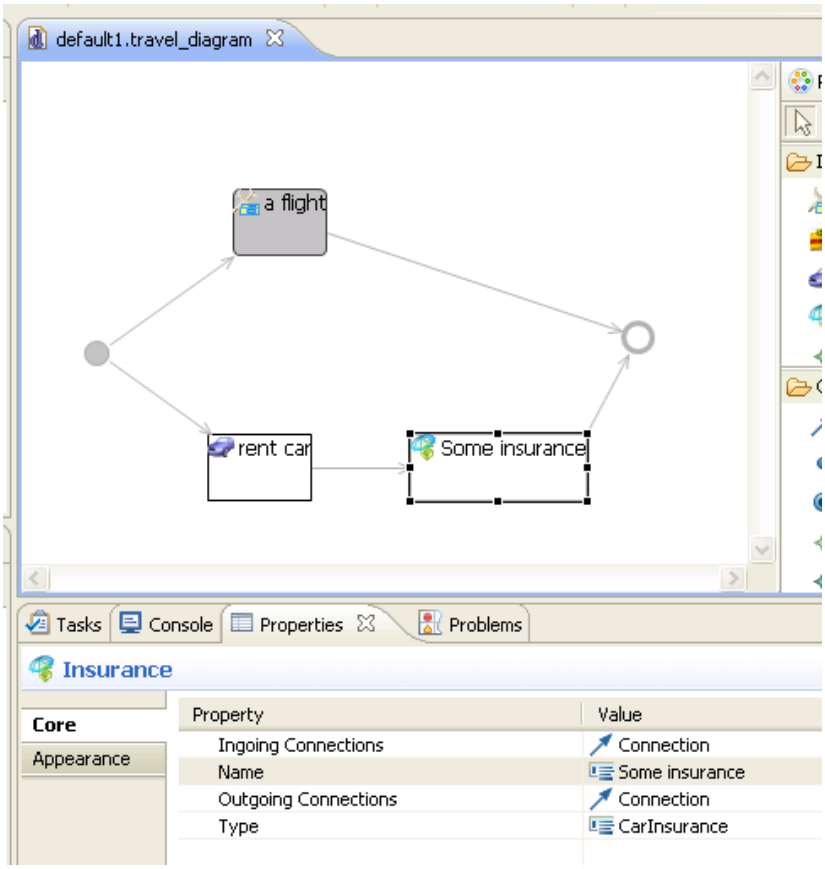


Add the 3. Constraint

- Follow same procedure to add the 3. Constraint and test it.
- First, create an invalid model as below and validate it from Edit->validate



Next, create a valid model as below, and validate it with edit->Validate



GMF Exercise 2 Question 4

Create a new plugin which adds an EditPolicyProvider to a CarRental item. The provider must install an EditPolicy that accepts open requests, shows a dialog to the user and asks for a name, and then creates an insurance node of type CarInsurance and a connection from the CarRental node.

This is probably not a good the way to implement such functionality. However, it illustrates both the use of a GMF extension point and the use of GMF commands.

This exercise requires Java development. You should develop 3 java classes called

- **EditPolicyProvider**. Installs an OpenEditPolicy in the CarRentalEditPart
- **OpenEditPolicy**. Asks the user about a name and instantiates a Command to create model elements and diagram node
- **CreateCarInsuranceCommand**. The command which creates the insurance model element, the connection, and the diagram node

Solution hints

- Create a new plugin project called <your project name>.diagram.custom, e.g. dk.itu.mdd.traval.diagram.custom.
- Add the following dependencies to the Required bundles in the manifest file:
  - `dk.itu.mdd.travel.diagram`
  - `org.eclipse.gmf.runtime.emf.core`
  - `org.eclipse.gmf.runtime.emf.commands.core`
  - `org.eclipse.gmf.runtime.emf.ui.properties`
  - `org.eclipse.gmf.runtime.diagram.ui`
- Add a new org.eclipse.gmf.runtime.diagram.ui.editpolicyProviders extension to the plugin.xml so it looks like below

```
<extension
         point="org.eclipse.gmf.runtime.diagram.ui.editpolicyProviders">
      <editpolicyProvider
            class="dk.itu.mdd.travel.diagram.custom.EditPolicyProvider">
         <Priority
               name="Medium">
         </Priority>
         <context
               editparts="dk.itu.mdd.travel.diagram.edit.parts.CarRentalEditPart">
         </context>
         <object
               class="dk.itu.mdd.travel.diagram.edit.parts.CarRentalEditPart"
               id="dk.itu.mdd.travel.diagram.edit.parts.CarRentalEditPart">
         </object>
      </editpolicyProvider>
   </extension>
```

  - If the manifest file complaints over a singleton directive, add ;singleton:=true to the Bundle-symbolic name

- Create the EditPolicyProvider class in the dk.itu.mdd.travel.diagram.custom package. It should extend the org.eclipse.gmf.runtime.common.core.service.AbstractProvider and implement org.eclipse.gmf.runtime.diagram.ui.services.editpolicy.IEditPolicyProvider
- Create the OpenEditPolicy class in the same package. It must extend org.eclipse.gmf.runtime.diagram.ui.editpolicies.OpenEditPolicy
- Implement the following for OpenEditPolicy
    - Constructor which gets an EditPart
    - Override the getOpenCommand method and do the following:
    - Open a dialog to ask for the insurance name
    - Instantiate the CreateCarInsuranceCommand to create the Insurance element
- It should be similar to below.

```
public class OpenEditPolicy extends
      org.eclipse.gmf.runtime.diagram.ui.editpolicies.OpenEditPolicy {

   private EditPart editpart;
   public OpenEditPolicy(EditPart editpart) {
      this.editpart = editpart;
   }
   @Override
   protected Command getOpenCommand(Request request) {

      return new Command() {
         @Override
         public void execute() {
            InputDialog d = new InputDialog(Display.getCurrent()
                  .getActiveShell(), "Enter name of insurance",
                  "Enter the name of the  car insurance node",
                  "Car insurance", null);
            if (d.open() == Window.OK) {
               // Now, create a command which creates an insurance node
               Node node = (Node) editpart.getModel();
               Diagram diagram = node.getDiagram();
               CarRental element = (CarRental) node.getElement();
               TransactionalEditingDomain domain = TransactionUtil.getEditingDomain(element);
               CreateCarInsuranceCommand cmd = new CreateCarInsuranceCommand(domain, element,
d.getValue(), diagram);
               try {
                  cmd.execute(new NullProgressMonitor(), null);
               } catch (ExecutionException e) {
                  e.printStackTrace();
               }
            }
         }
      };
   }
}
```

- Implement the EditPolicy provider to install the OpenEditPolicy class at the CarRentalEditPart. It should look like below

```
public class EditPolicyProvider extends AbstractProvider implements
      IEditPolicyProvider {

   @Override
   public void createEditPolicies(EditPart editPart) {
      editPart.installEditPolicy(EditPolicyRoles.OPEN_ROLE, new OpenEditPolicy(editPart));
   }

   @Override
   public boolean provides(IOperation operation) {
      if(operation instanceof CreateEditPoliciesOperation){
         EditPart part = ((CreateEditPoliciesOperation) operation).getEditPart();
         if(part instanceof CarRentalEditPart){
            return true;
         }
      }
      return false;
   }

}
```

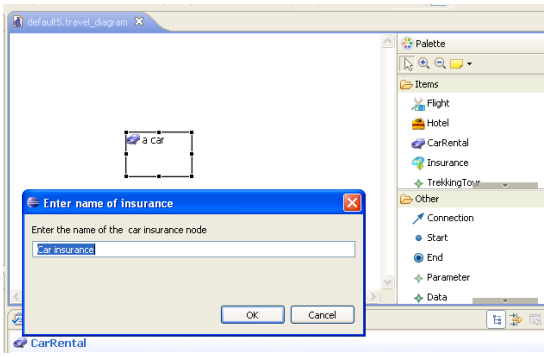
- Implement the custom command CreateCarInsuranceCommand like below

```
public class CreateCarInsuranceCommand extends AbstractTransactionalCommand {
   private String name ;
   private CarRental rental;
   private Diagram diagram;

   public CreateCarInsuranceCommand(TransactionalEditingDomain domain, CarRental rental,
String name, Diagram diagram) {
      super(domain, "No label", null);
      this.name=name;
      this.rental = rental;
      this.diagram = diagram;
   }

   @Override
   protected CommandResult doExecuteWithResult(IProgressMonitor monitor,
         IAdaptable info) throws ExecutionException {
      // Create domain element
      Insurance insurance = TravelFactory.eINSTANCE.createInsurance();
      insurance.setName(name);
      insurance.setType(InsuranceType.CAR_INSURANCE);
      Connection connection = TravelFactory.eINSTANCE.createConnection();
      connection.setSourceNode(rental);
      connection.setTargetNode(insurance);
      Travel travel = (Travel) rental.eContainer();
      travel.getItems().add(insurance);
      travel.getConnections().add(connection);

      // Create Node
      Node insuranceNode = ViewService.createNode(diagram, travel,
            TravelPackage.Literals.INSURANCE.getName(),
            TravelDiagramEditorPlugin.DIAGRAM_PREFERENCES_HINT);

      return CommandResult.newOKCommandResult();
   }

}
```

Test the code

You should not regenerate diagram code as you have not modified the mapping model!

- Start runtime workbench, create a model with a CarRental node
- Doubleclick at the border of the CarRental node. Enter a name in the dialog and press OK



- Evaluate that an Insurance node has been created and a connection from the CarRental node is created