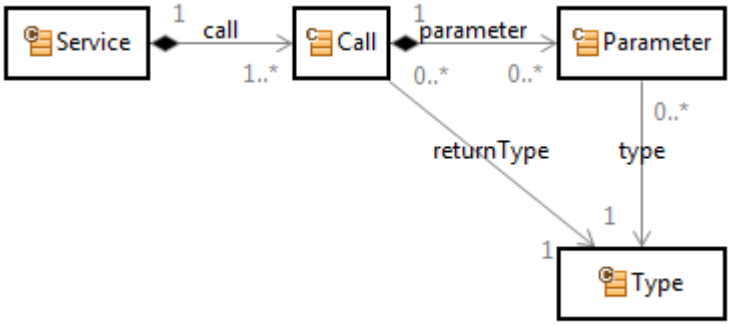
Tutorial

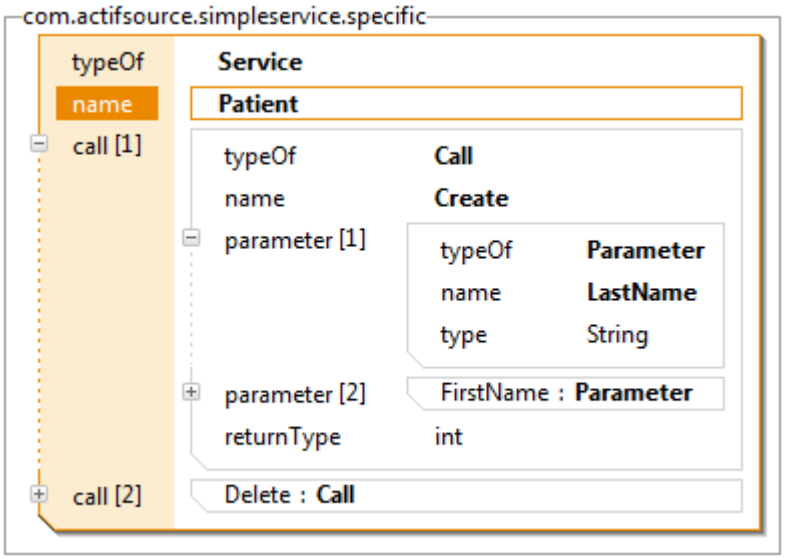
Simple Service

Tutorial	Actifsource Tutorial – Simple Service
Required Time	• 70 Minutes
Prerequisites	• Actifsource Tutorial – Installing Actifsource
Goal	• Developing a generic domain model for a simple service infrastructure • Instantiating specific domain objects according to the generic domain • Writing code templates according to the generic domain • Generate code for every specific domain object
Topics covered	• Creating a new actifsource project • Working with Diagram Editor • Working with Resource Editor • Working with Template Editor • Generating Code
Notation	↳ To do ⓘ Information • **Bold**: Terms from actifsource or other technologies and tools • **Bold underlined**: actifsource Resources • Underlined: User Resources • *UnderlinedItalics*: Resource Functions • `Monospaced`: User input • *Italics*: Important terms in current situation
Disclaimer	The authors do not accept any liability arising out of the application or use of any information or equipment described herein. The information contained within this document is by its very nature incomplete. Therefore the authors accept no responsibility for the precise accuracy of the documentation contained herein. It should be used rather as a guide and starting point.
Contact	**actifsource GmbH** Täfernstrasse 37 5405 Baden-Dättwil Switzerland www.actifsource.com
Trademark	**actifsource** is a registered trademark of **actifsource GmbH** in Switzerland, the EU, USA, and China. Other names appearing on the site may be trademarks of their respective owners.
Compatibility	Created with **actifsource** Version 5.8.5

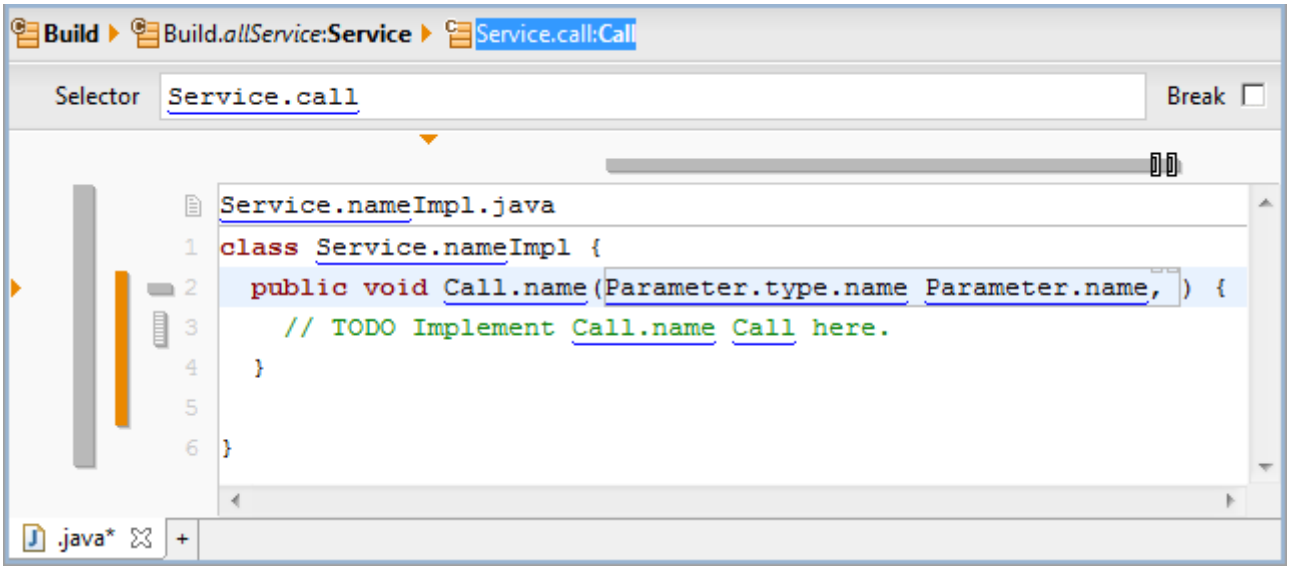
- Create a new actifsource Project
- Structure your models using *Packages*
- Create Domain Classes in a *Generic Domain Model* for a Simple Service Infrastructure consisting of Services, Service-Calls, Arguments and Types:



- Create a *Specific Domain Model* using your own domain-specific type system
- Use **Content Assist** to create new domain-specific types or insert existing domain-objects



- Use *Code Templates* to generate *Specific Code*
- *Template Code* is always written in the context of the *Generic Domain Model*
- Learn how to add model-related *Contexts* in your *Code Templates*
- Learn how to insert model-related *Variables* using *Content Assist*



- Fill in Protected Regions in your generated code

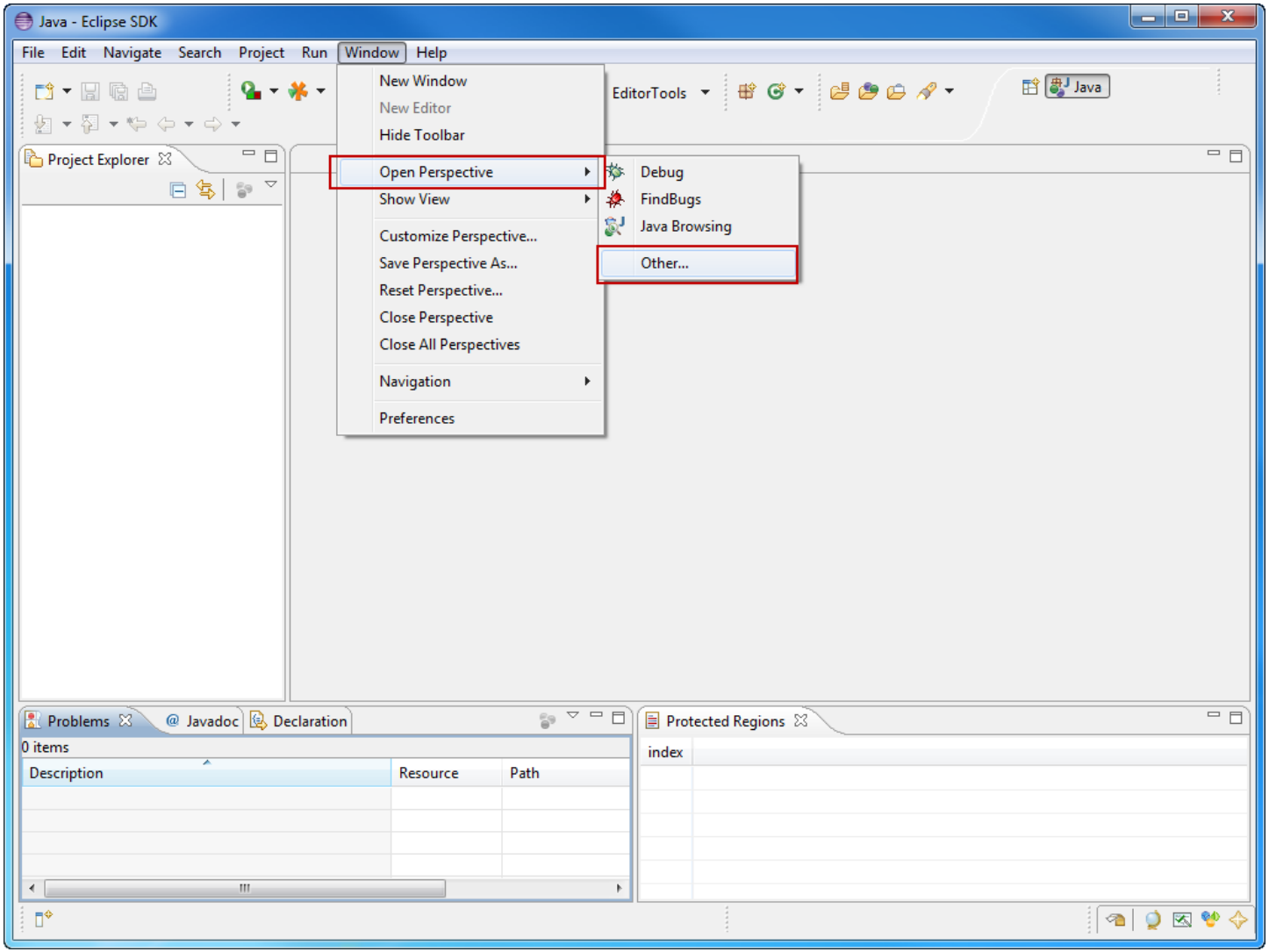
```
class PatientImpl {
  public int Create(String LastName, String FirstName) {
    // Begin Protected Region [[859fc3ce-30e8-11df-afa7-8977f15a88e3]]
    // TODO Implement Create Call here.
    // End Protected Region   [[859fc3ce-30e8-11df-afa7-8977f15a88e3]]
  }

  public boolean Delete(int Id) {
    // Begin Protected Region [[4cd50ca0-30ec-11df-8fa1-0fcf44129cd6]]
    // TODO Implement Delete Call here.
    // End Protected Region   [[4cd50ca0-30ec-11df-8fa1-0fcf44129cd6]]
  }
```

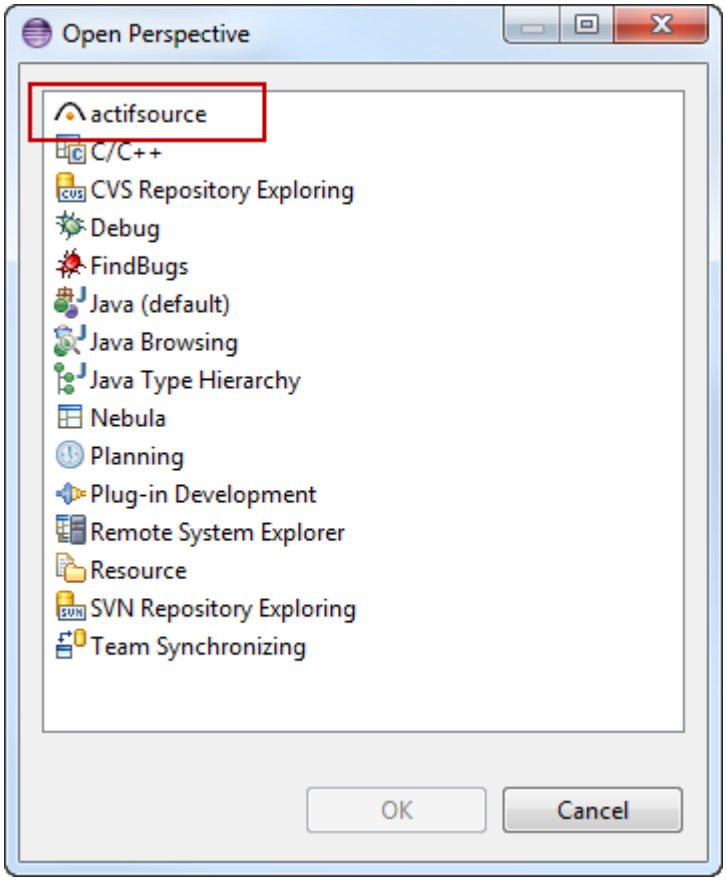
Part I: 5
Create a new actifsource Project

- Enable the actifsource perspective
- Create and setup a new actifsource project “SimpleService”
- Familiarize yourself with the basic structure of an actifsource project

Create a new actifsource Project

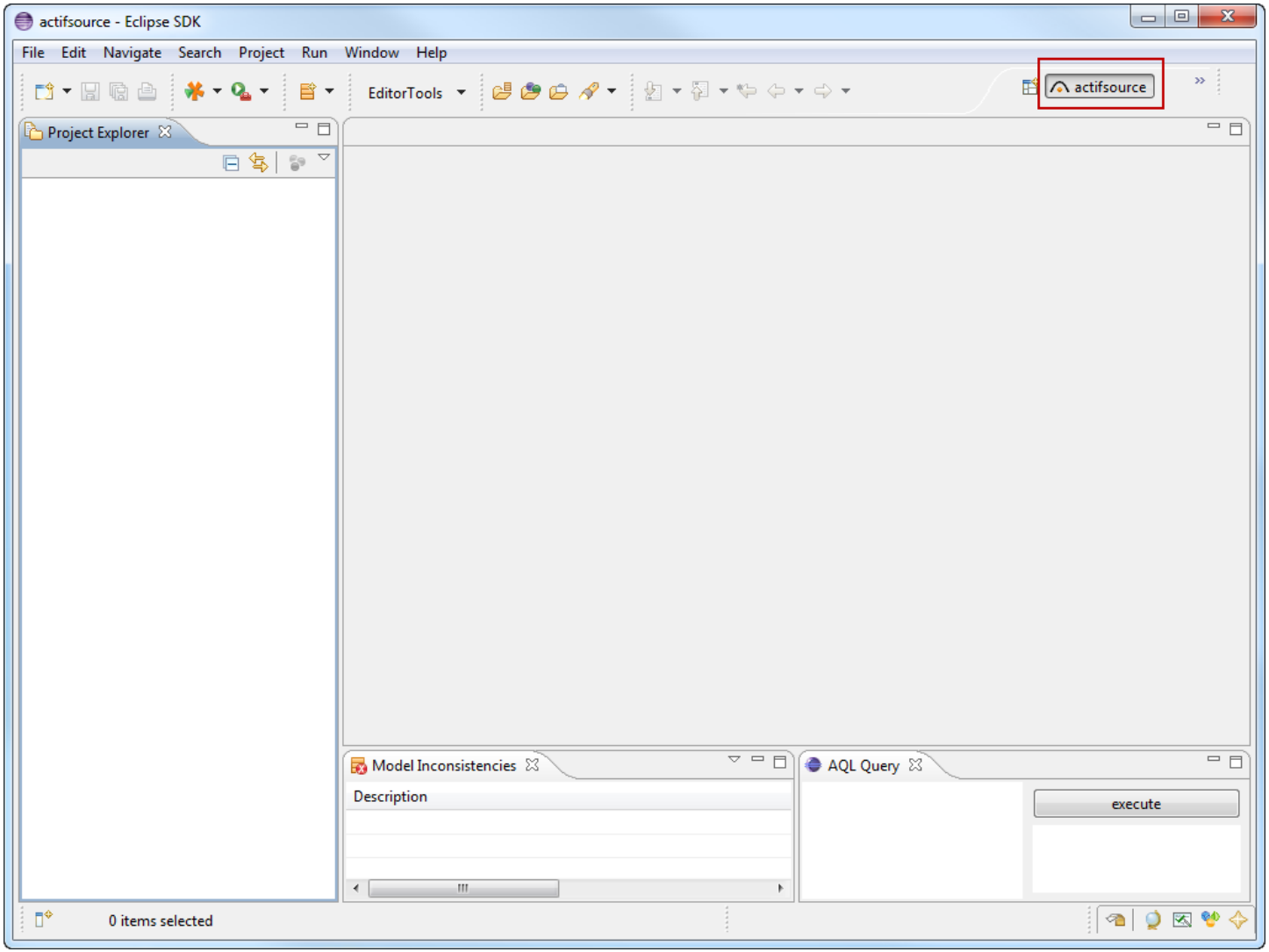


- Install **actifsource** as a plugin from www.**actifsource**.com
- Open the **actifsource Perspective**



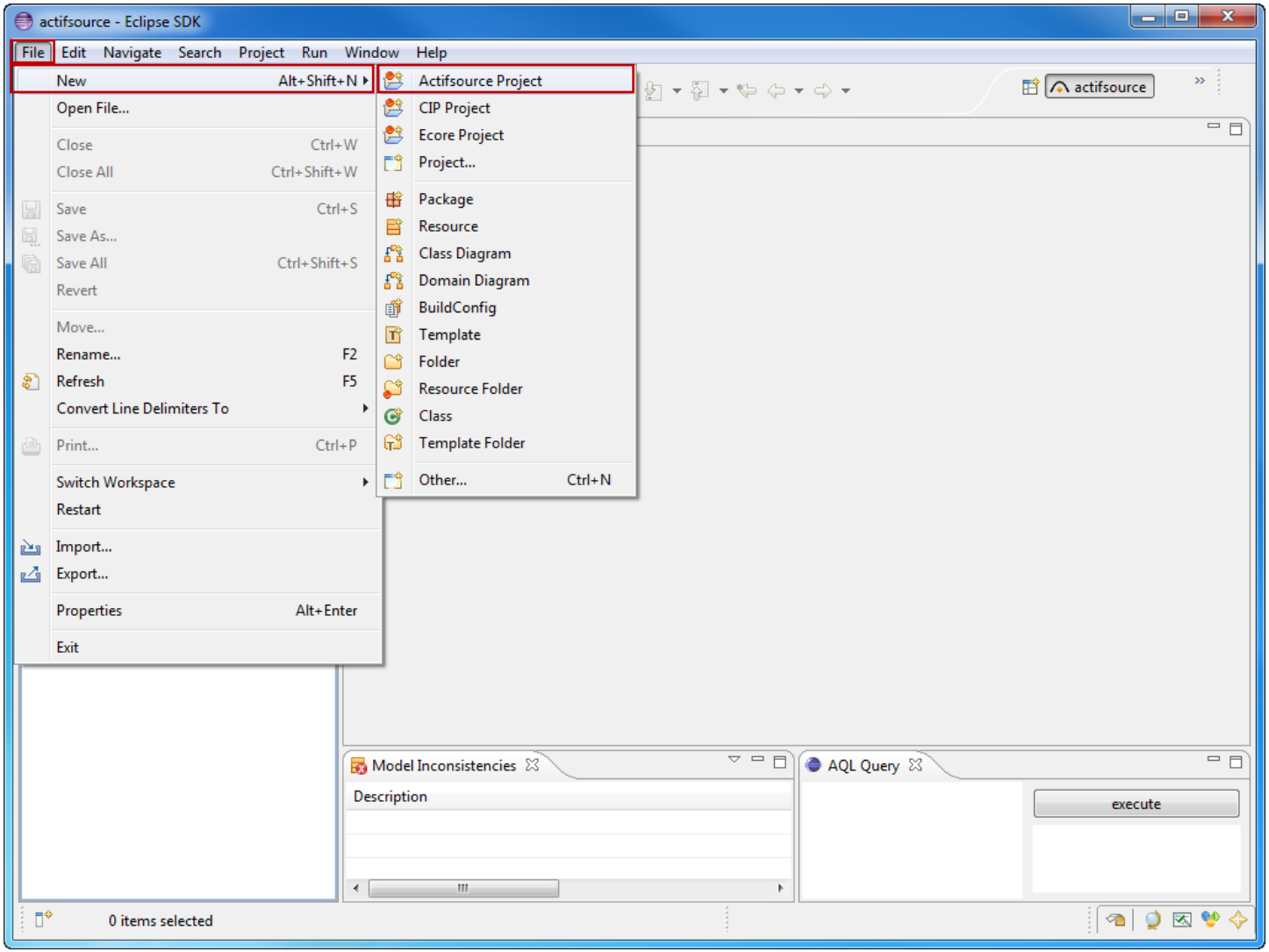
- In the Open Perspective dialog click on ***actifsource***
- Click *OK* to confirm.

Create a new actifsource Project

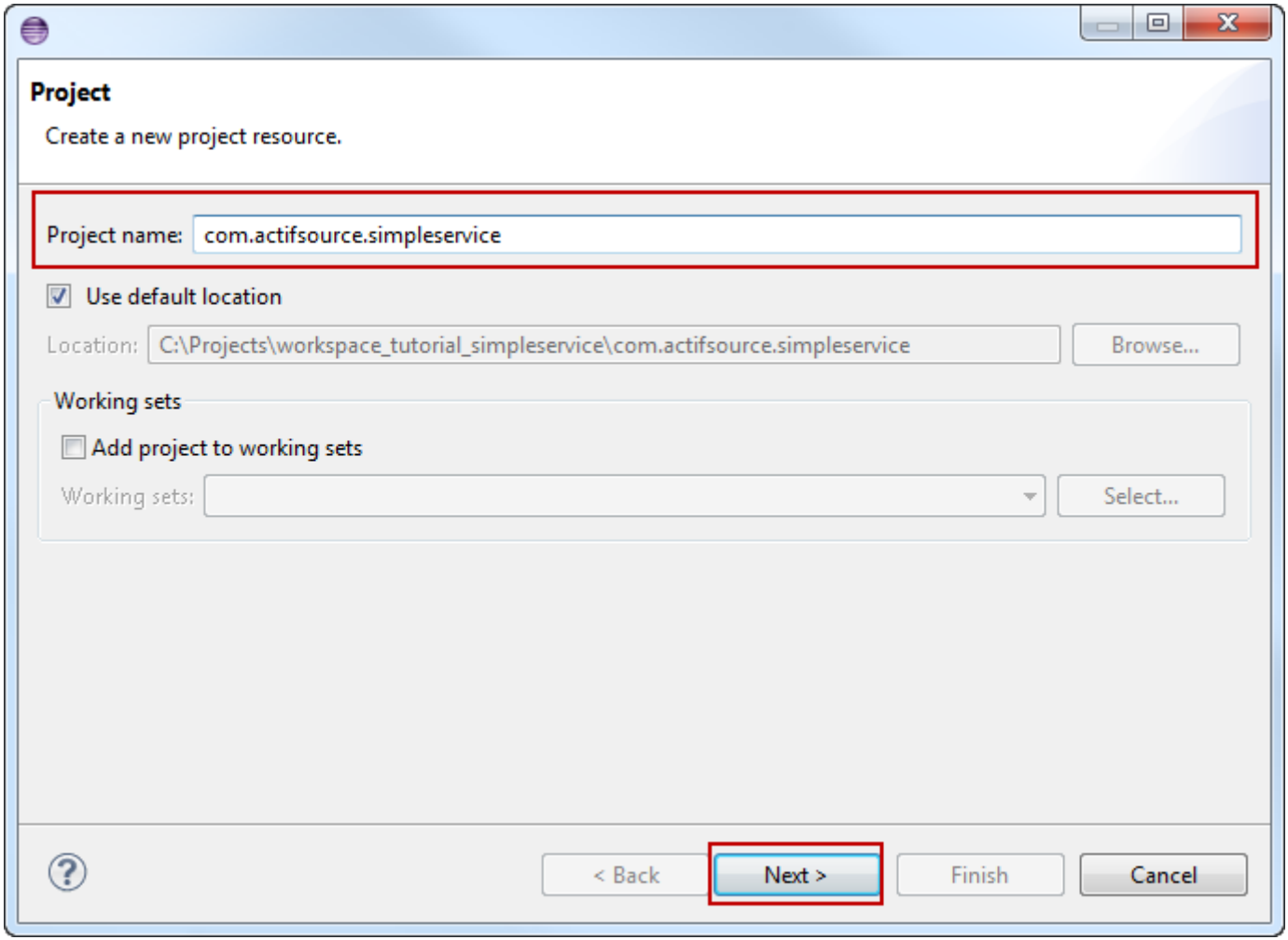


ⓘ Make sure the **actifsource Perspective** is activated

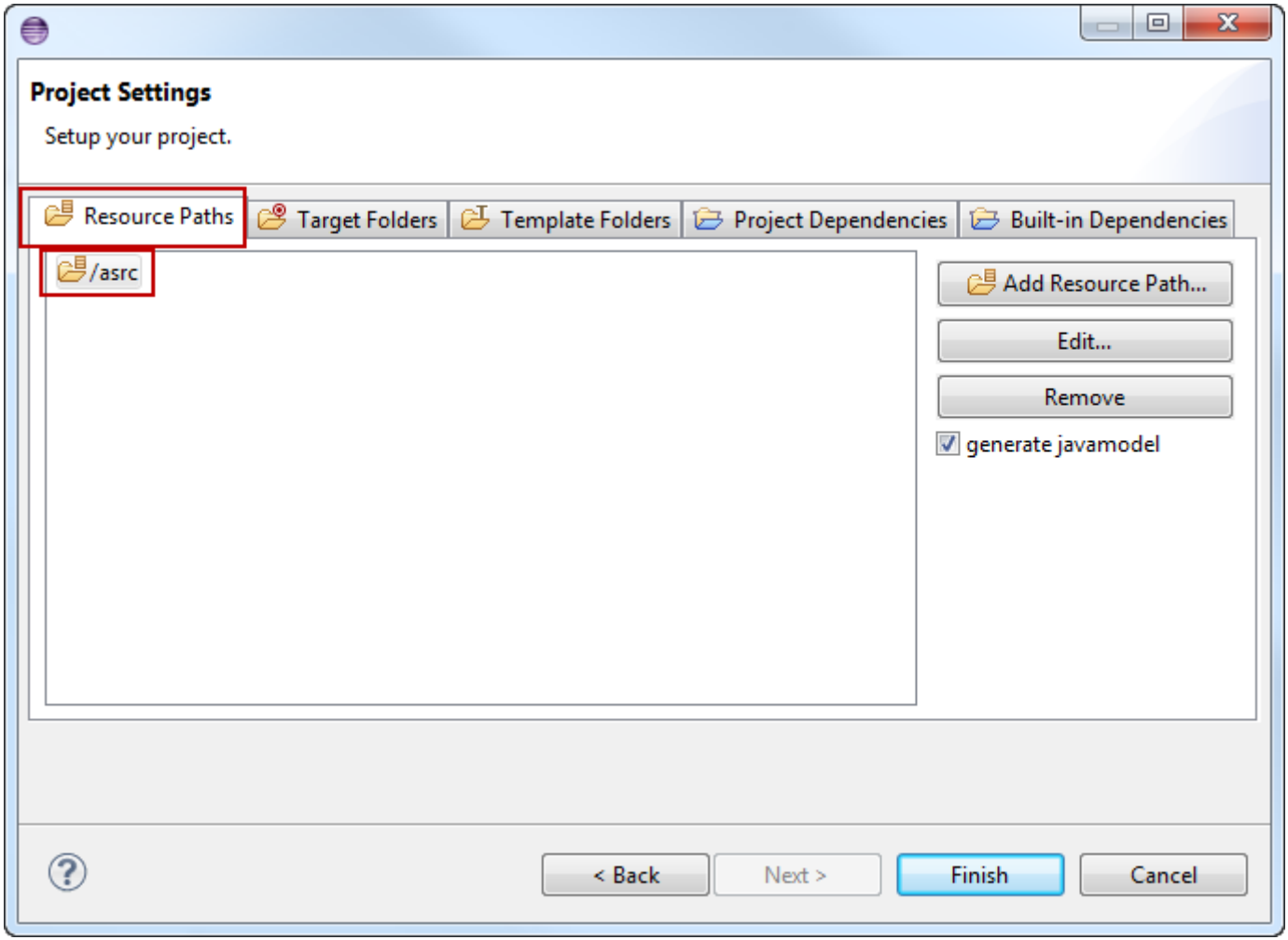
Create and Setup a new actifsource Project “SimpleService” 9



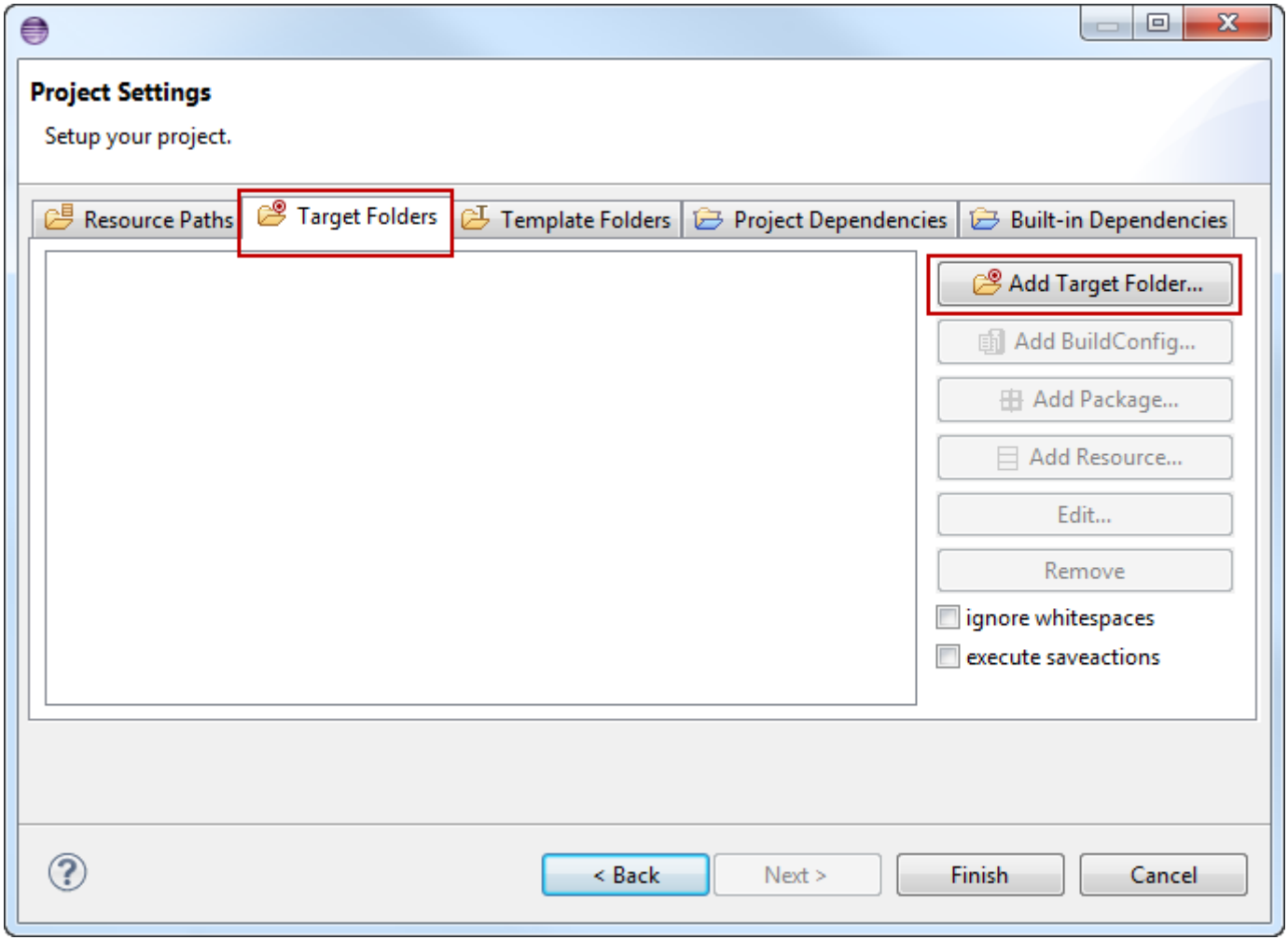
↳ Create a new **actifsource Project**



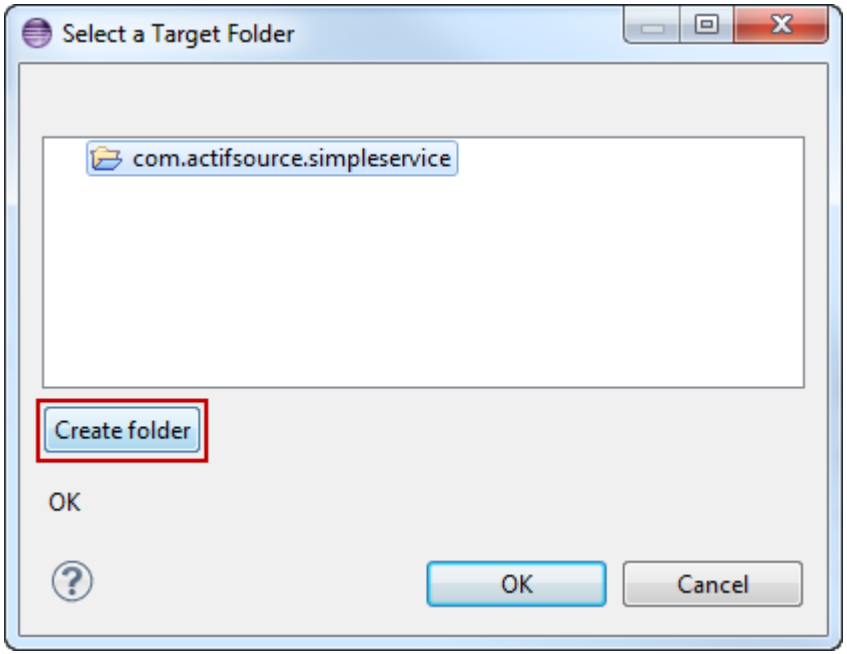
- Choose the project’s name com.actifsource.s`impleservice`
- Click *Next*

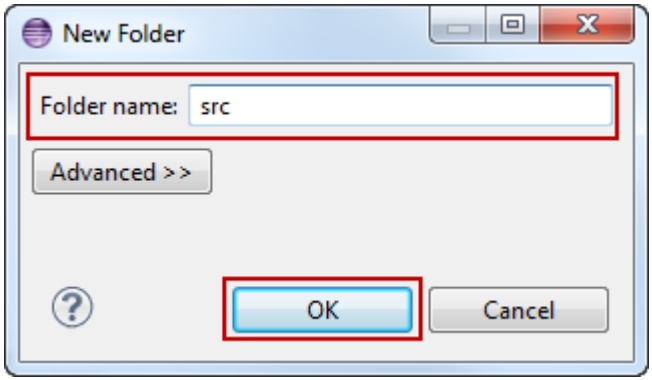


- ⓘ In **actifsource**, artifacts are called **Resources**
- ⓘ **Resources** are placed in folders and files
- ⓘ The default resource folder is called *asrc*

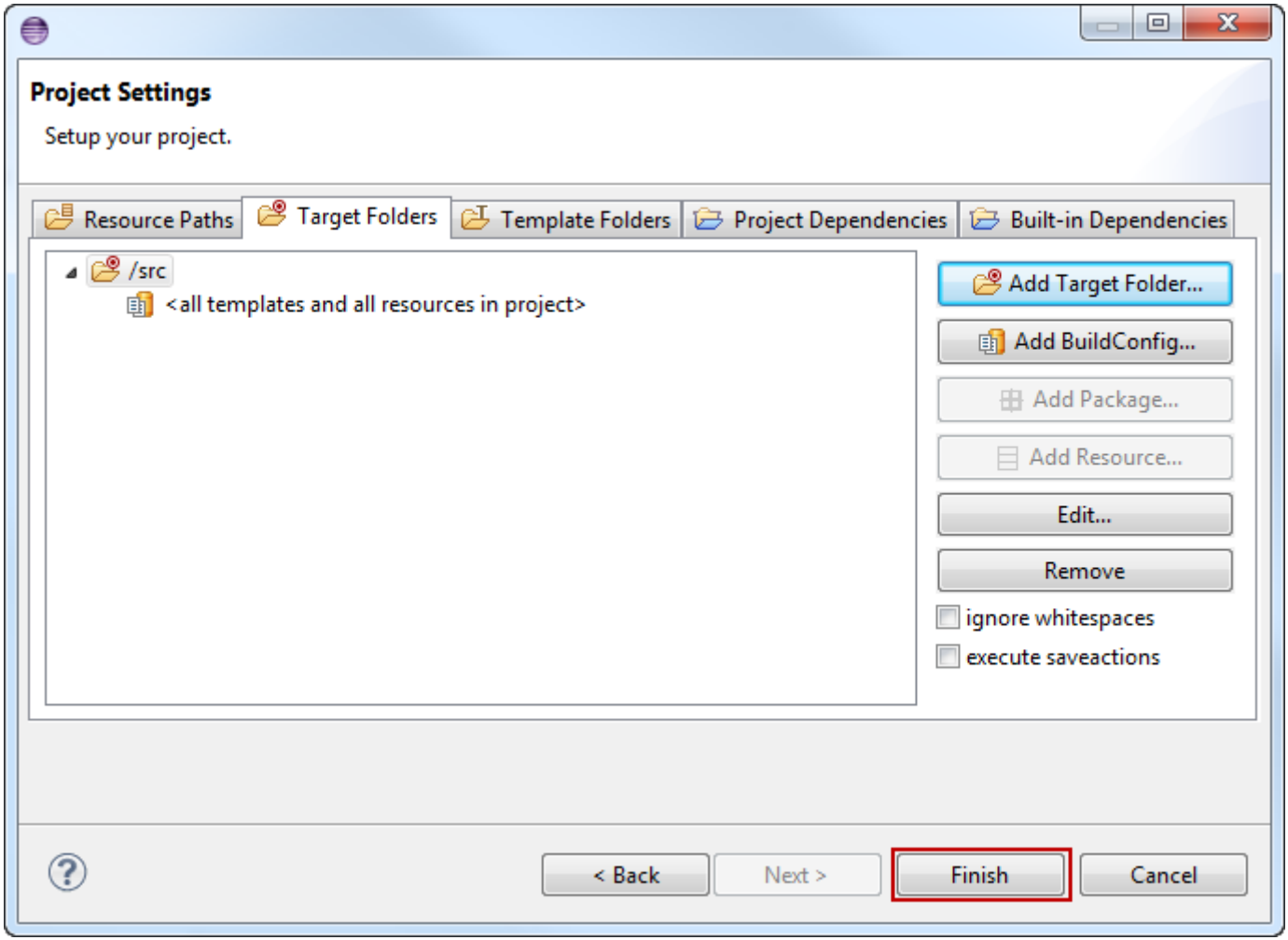


- Switch to the tab *Target Folders*
- Click *Add Target Folder…*
- ⓘ Note: Generated Code is placed in target folders
- ⓘ Change project settings anytime using the project properties: Project/Properties/actifsource

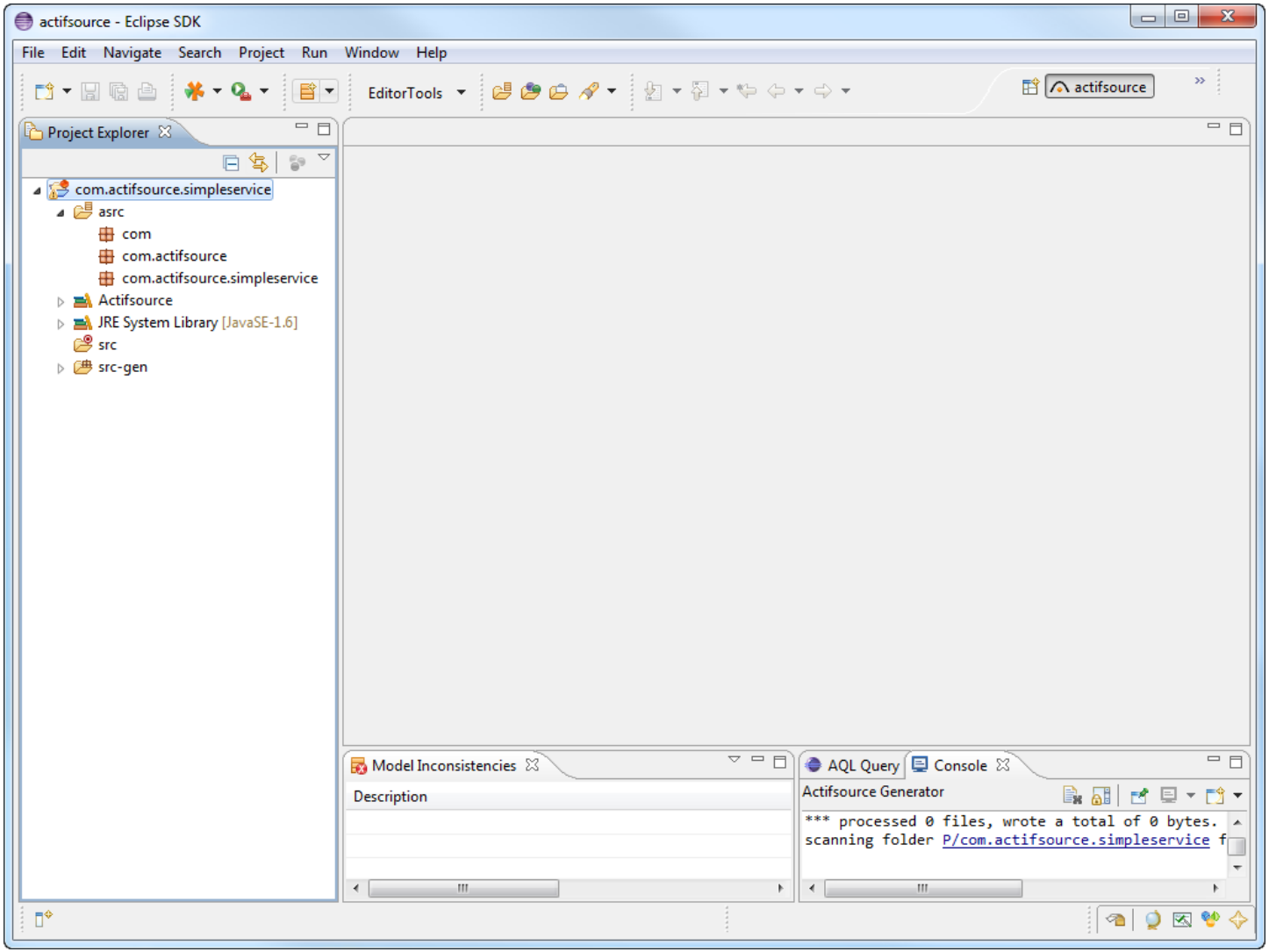




↳ Add a target folder named *src*



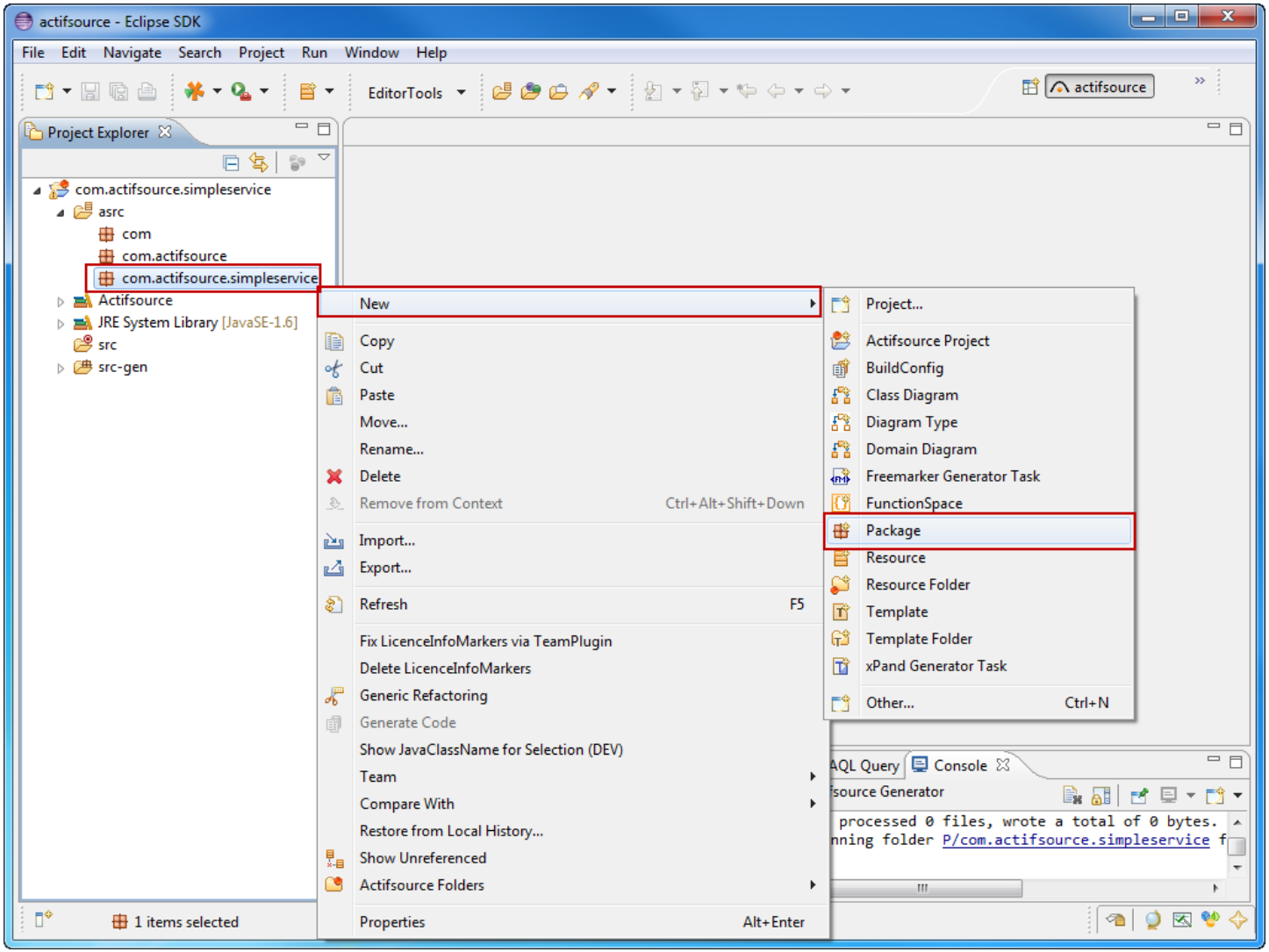
↳ Click *Finish*



- ⓘ We have created a new **actifsource Project** named com.actifsource.simpleservice
- ⓘ Resource Folder *asrc* is where we place all **actifsource Resources**
- ⓘ **Target Folder** *src* is where the generated code is placed

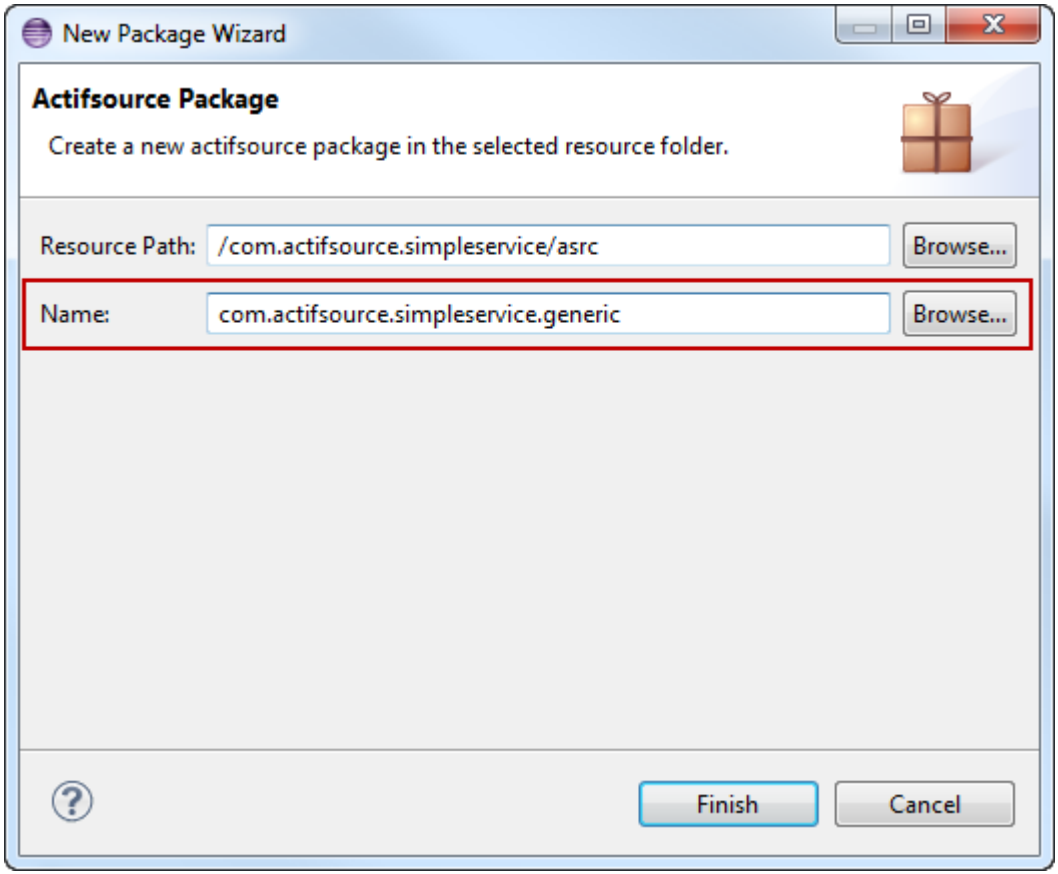
Part II: 16
Create a Generic Domain Model

- Setup an appropriate package structure
- Create a new diagram to model the generic domain model using generic classes and relations

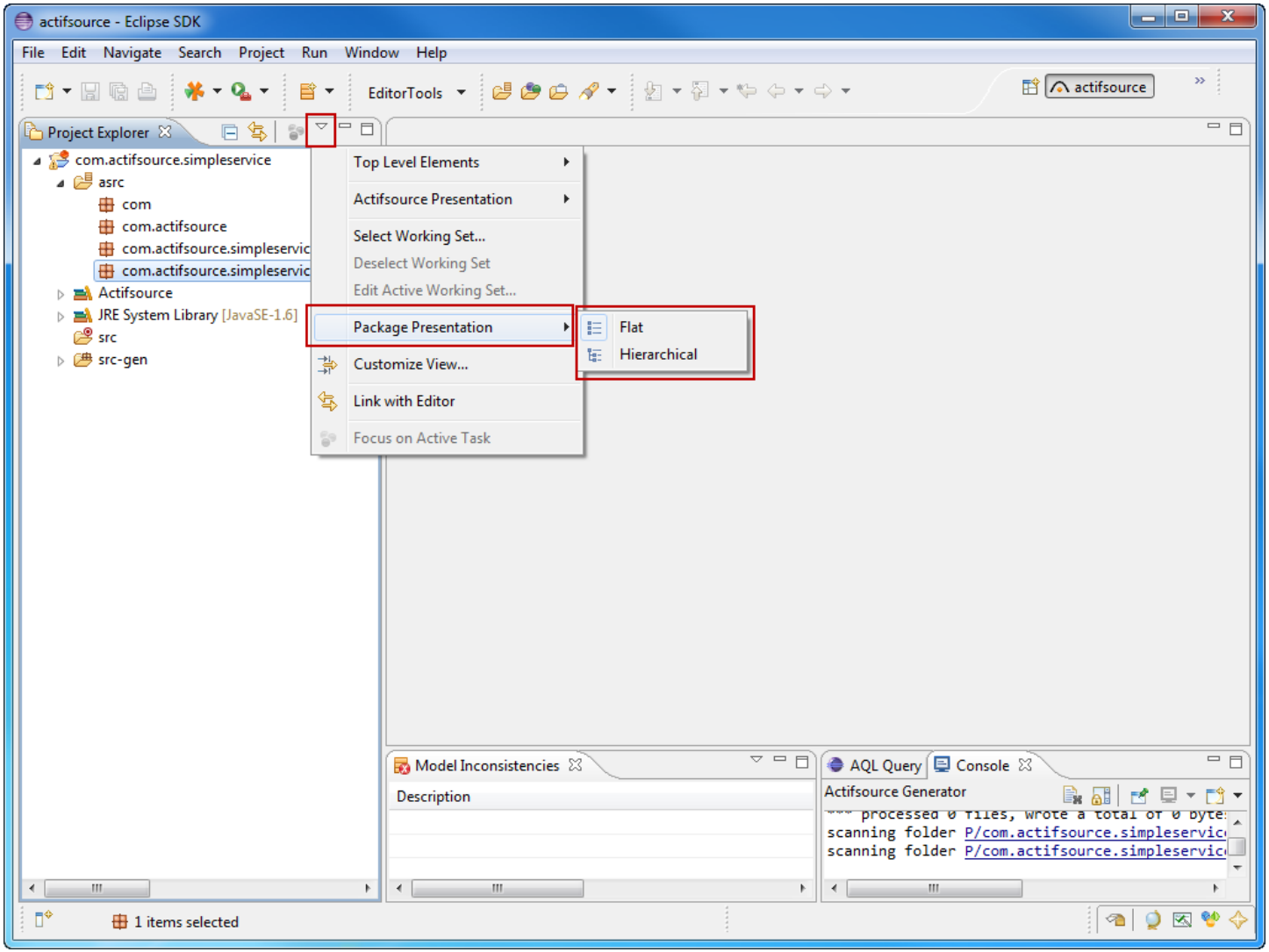


ⓘ In **actifsource**, **Resources** are organized within **Packages**

↳ Create a new **Package** in the resource folder *asrc*

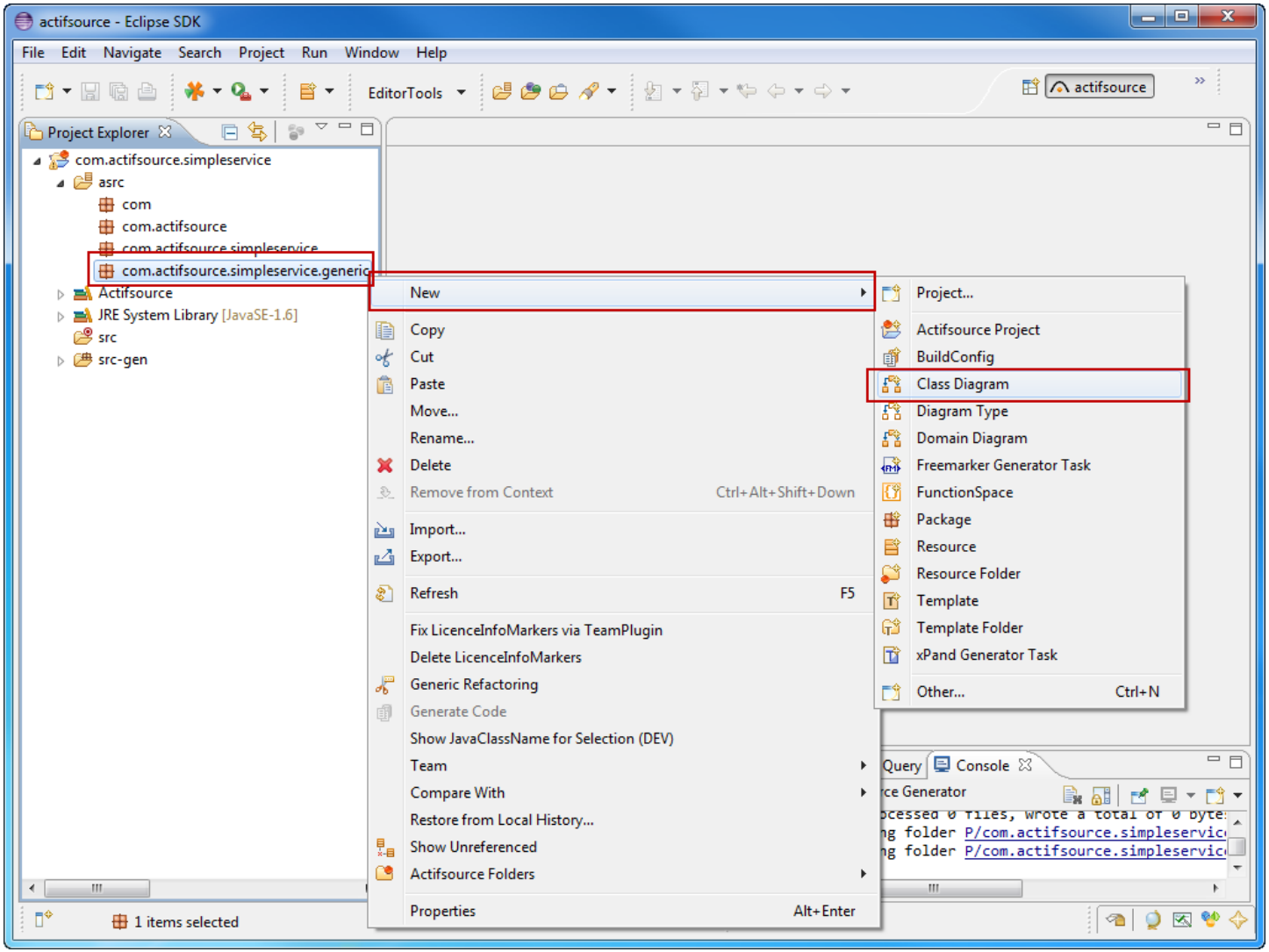


- Name your package `com.actifsource.simpleservice.generic` as shown above
- Click *Finish*

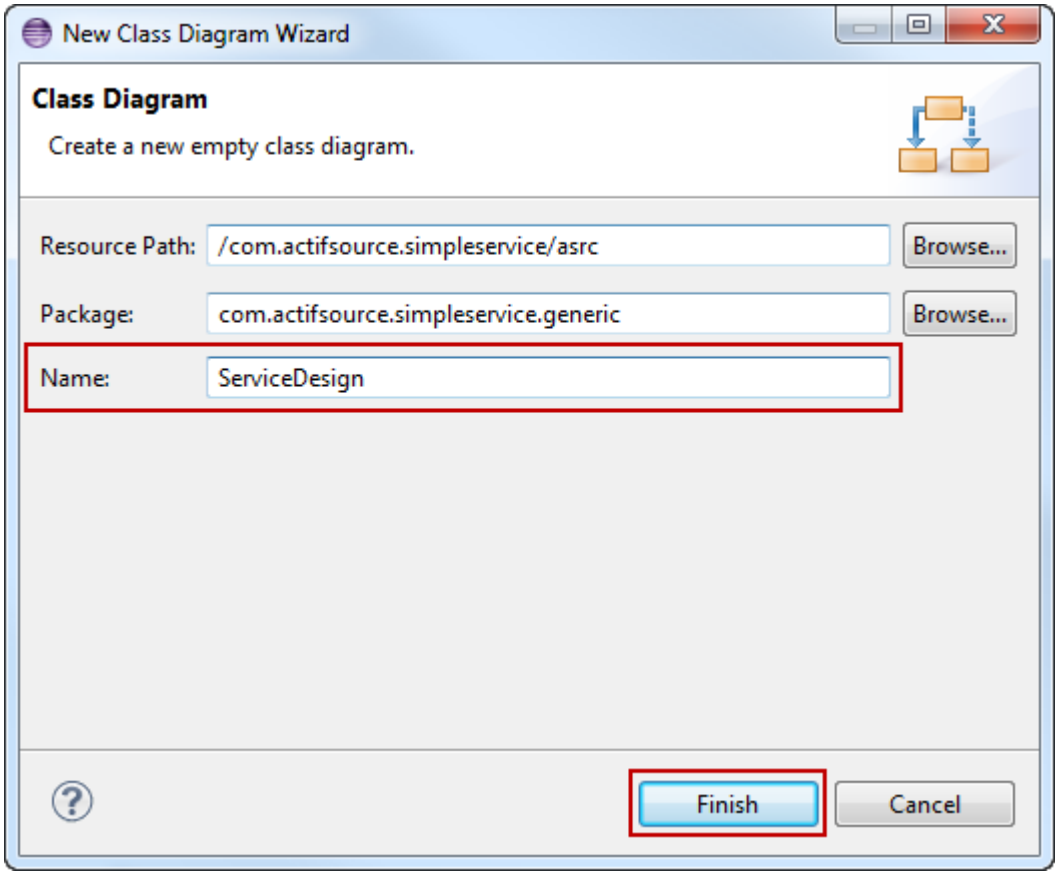


↳ Change the style for **Package Presentation** as you like

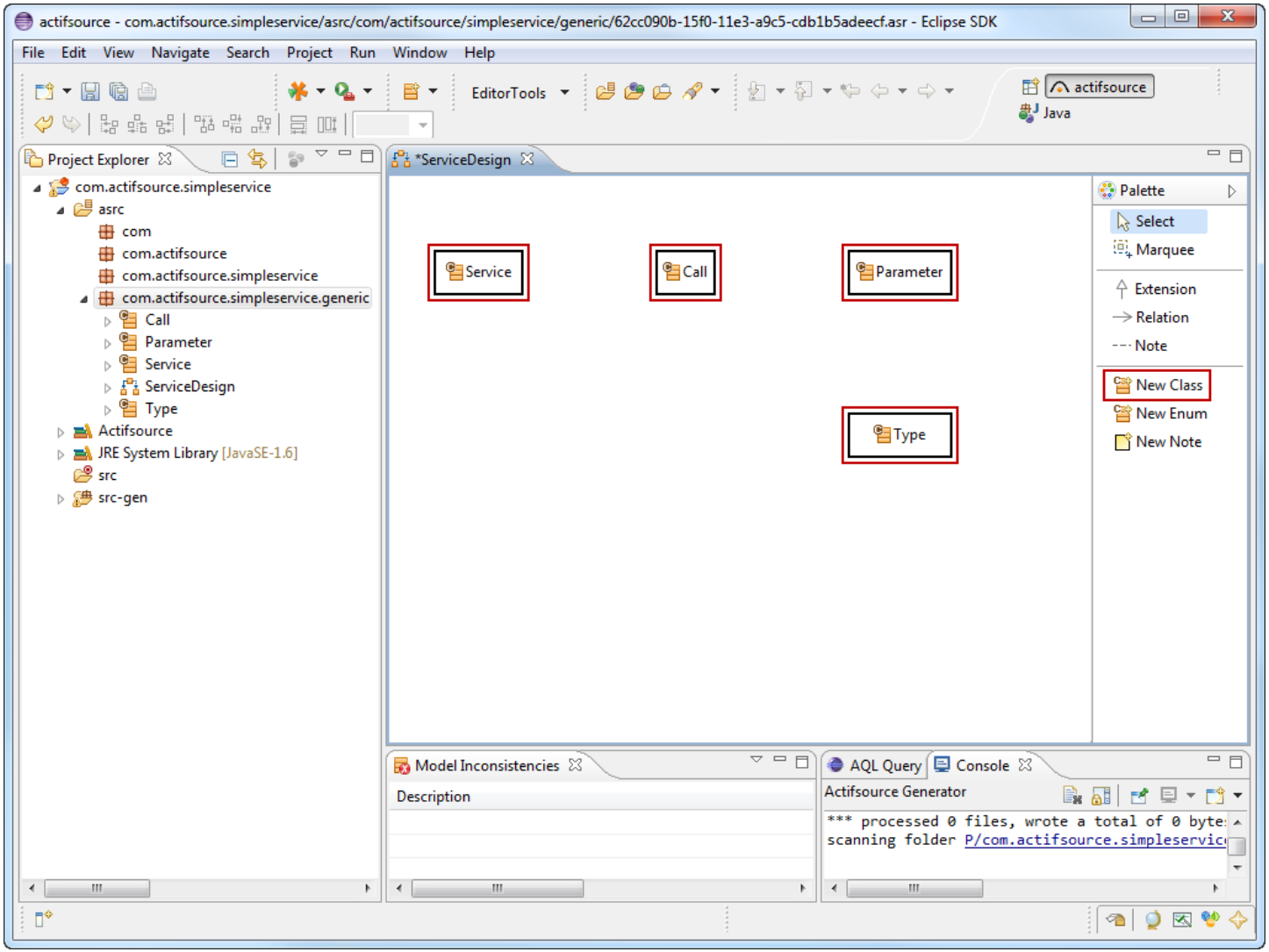
Create a new Diagram to Model the Generic Domain Model 20



- Start with a **Class Diagram** of your generic domain model
- Create a new **Class Diagram** in the **Package** *generic*



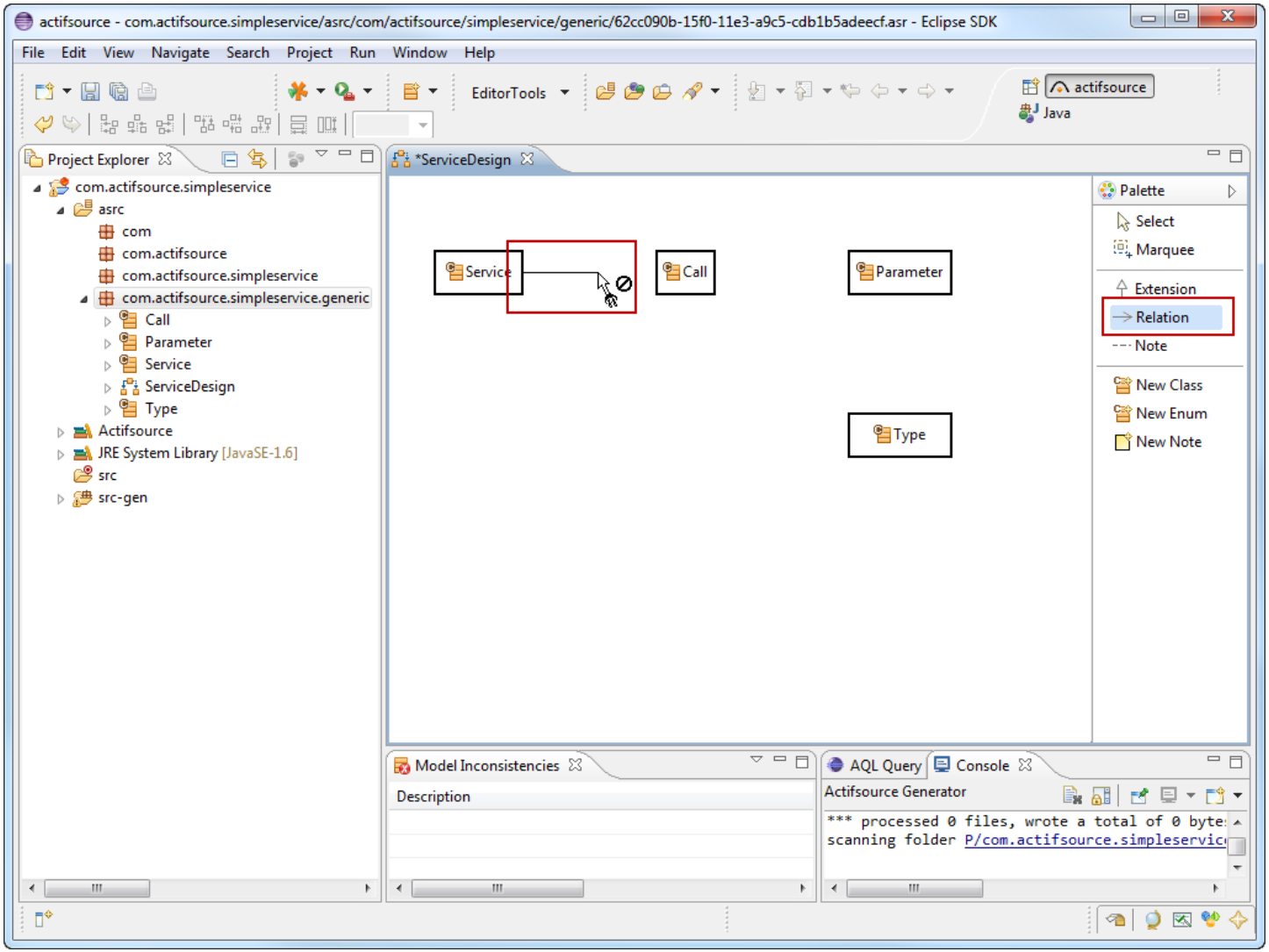
- Name your **Diagram** `ServiceDesign`
- Click *Finish*



ⓘ The **Diagram** *ServiceDesign* is opened in the **Diagram Editor**

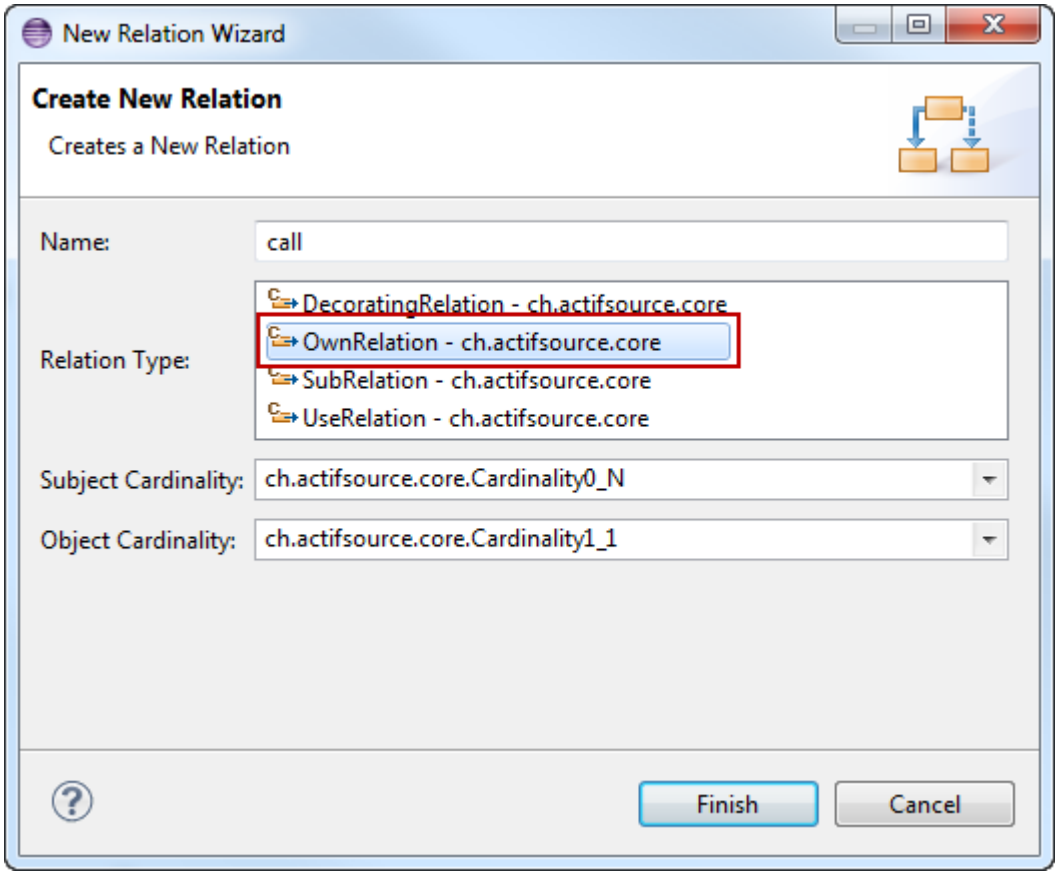
↳ Insert the following **Classes** using the *New Class* Tool from the *Palette:* `Service`, `Call`, `Parameter` and `Type`

Add Generic Relations

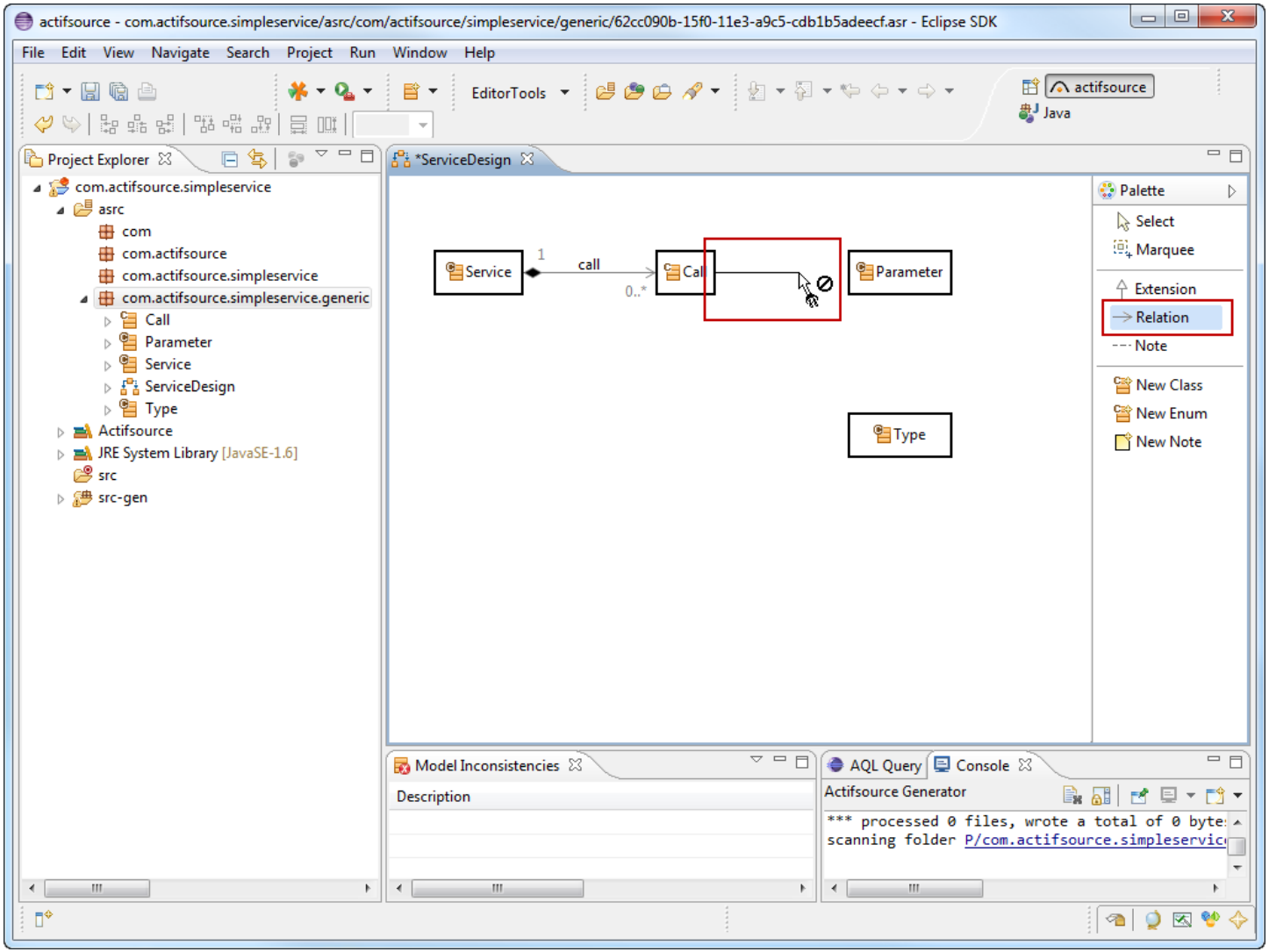


- Specify the **Relations** between your **Classes** using the *Relation* Tool from the *Palette on the right*
- First, link Service and Call by clicking on Service and then on Call in the diagram

Own Relations

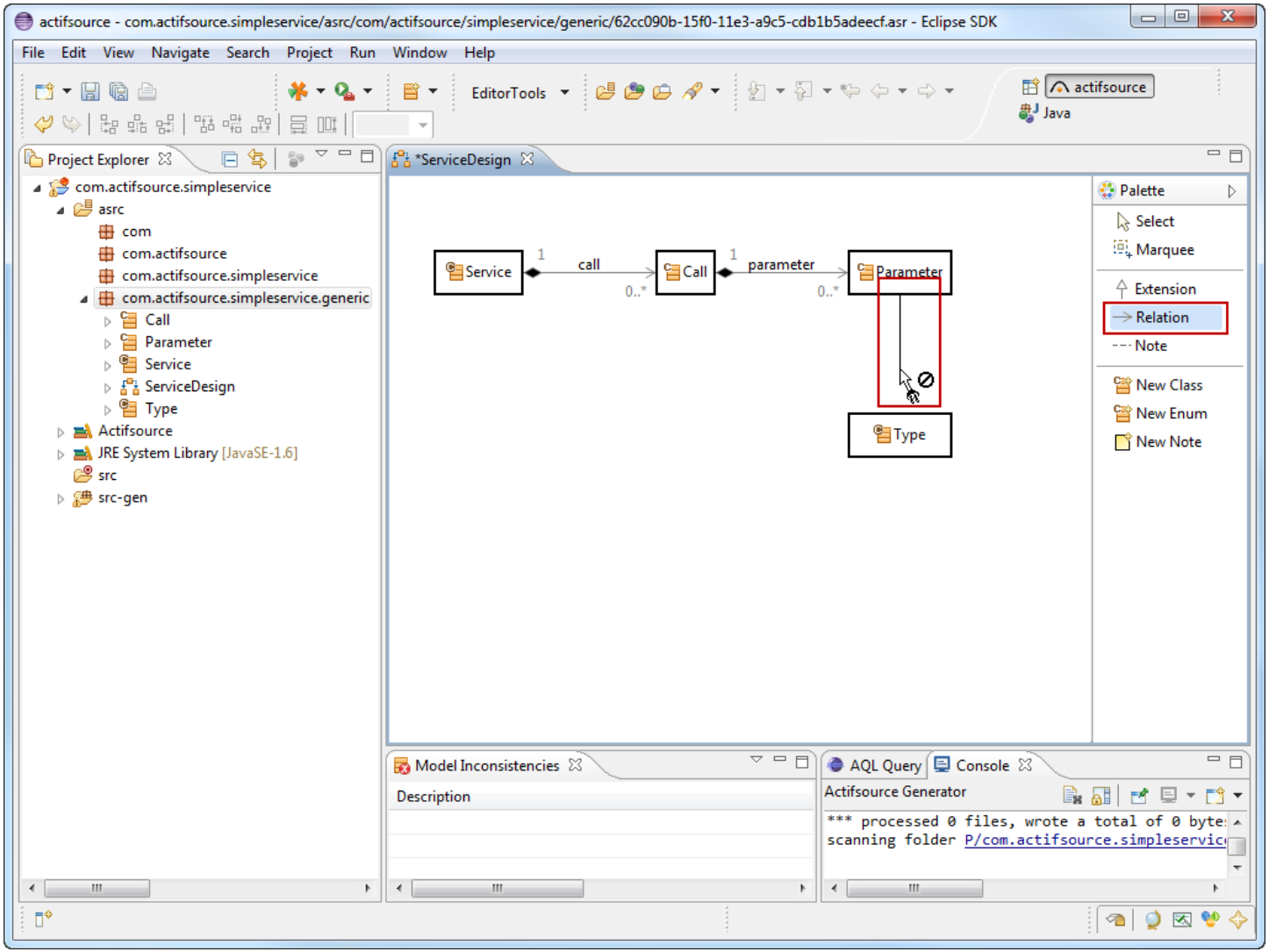


- ⓘ **actifsource** distinguishes basically **OwnRelation** and **UseRelation**, all other relation types based upon them.
- ⓘ Owned resources live within the context of their owner
- ⓘ Used resources have their own lifetime and are referenced only
- ⓘ A Call is owned by Service because no Call must exist without its Service
- ⓘ Default name *call* is fine here
- ↳ Select the **OwnRelation**



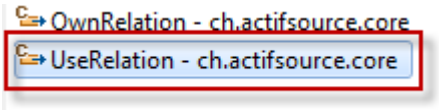
↳ Insert another **OwnRelation** between Call and Parameter

Use Relations

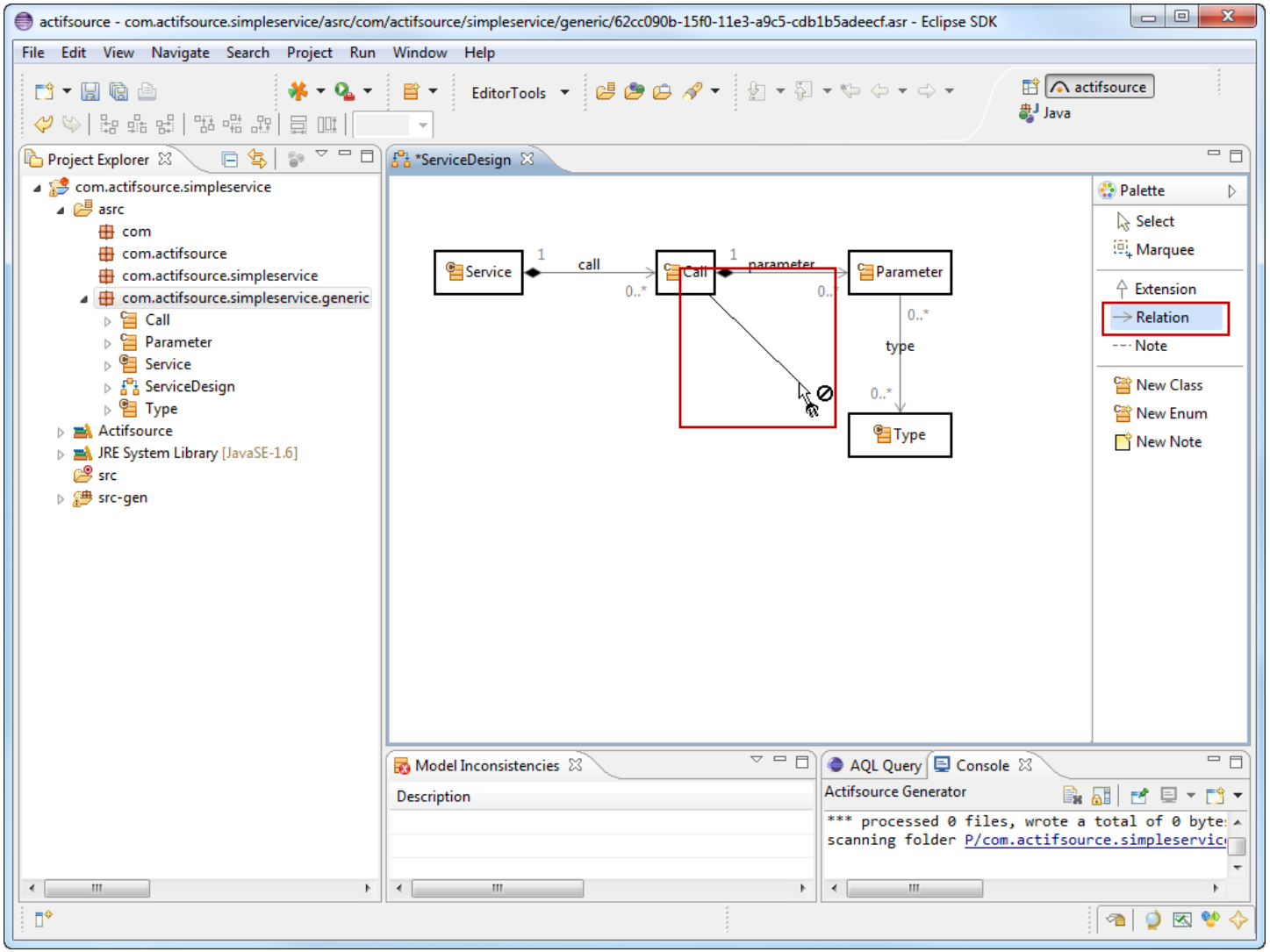


ⓘ Type is used by Parameter because types have their own lifetime independent from referencing parameters

↳ Insert a **UseRelation** between Parameter and Type

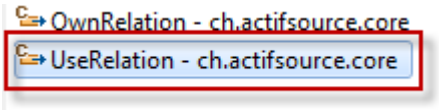


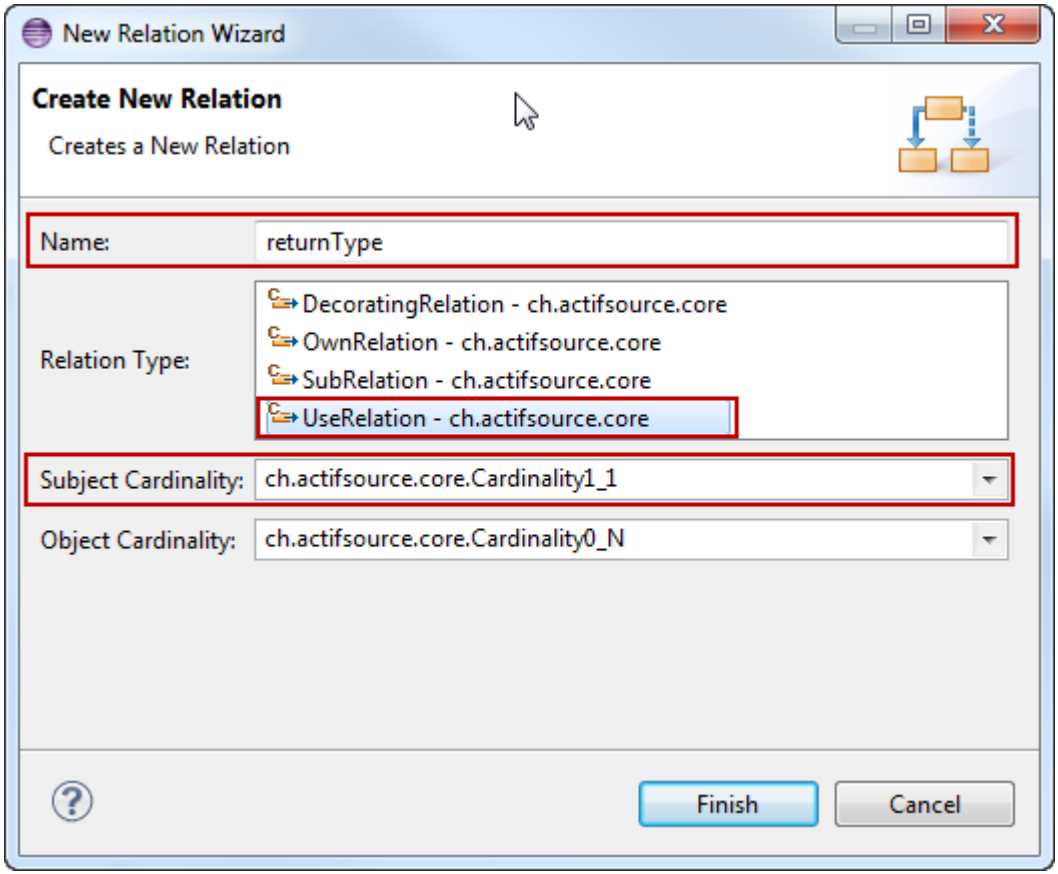
Use Relations



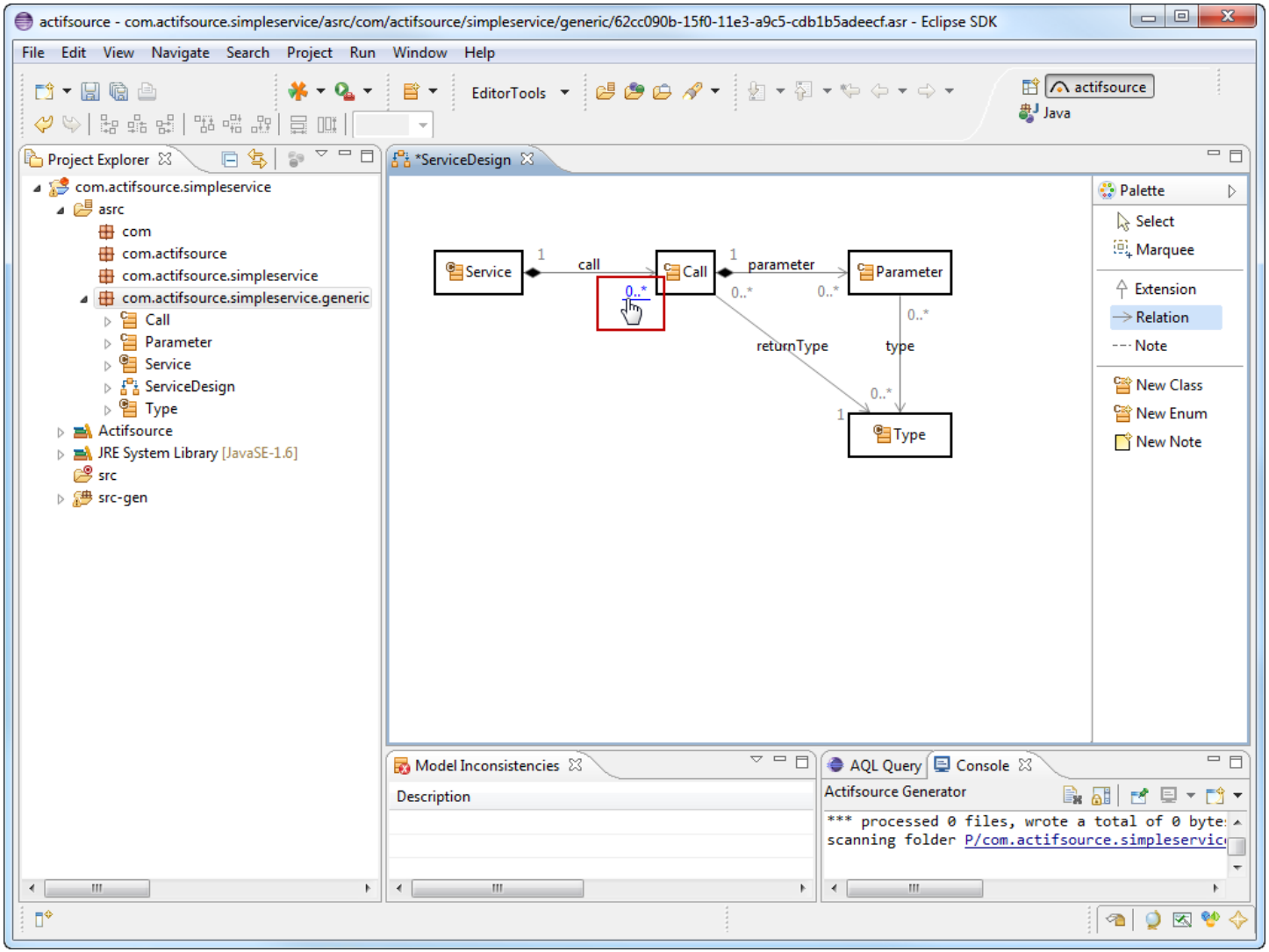
ⓘ Call should support a return type

↳ Insert a **UseRelation** between Call and Type





- Name the relation `returnType`
- Choose **SubjectCardinality** 1..1 since we always want a return type in this example
- Click *Ok*

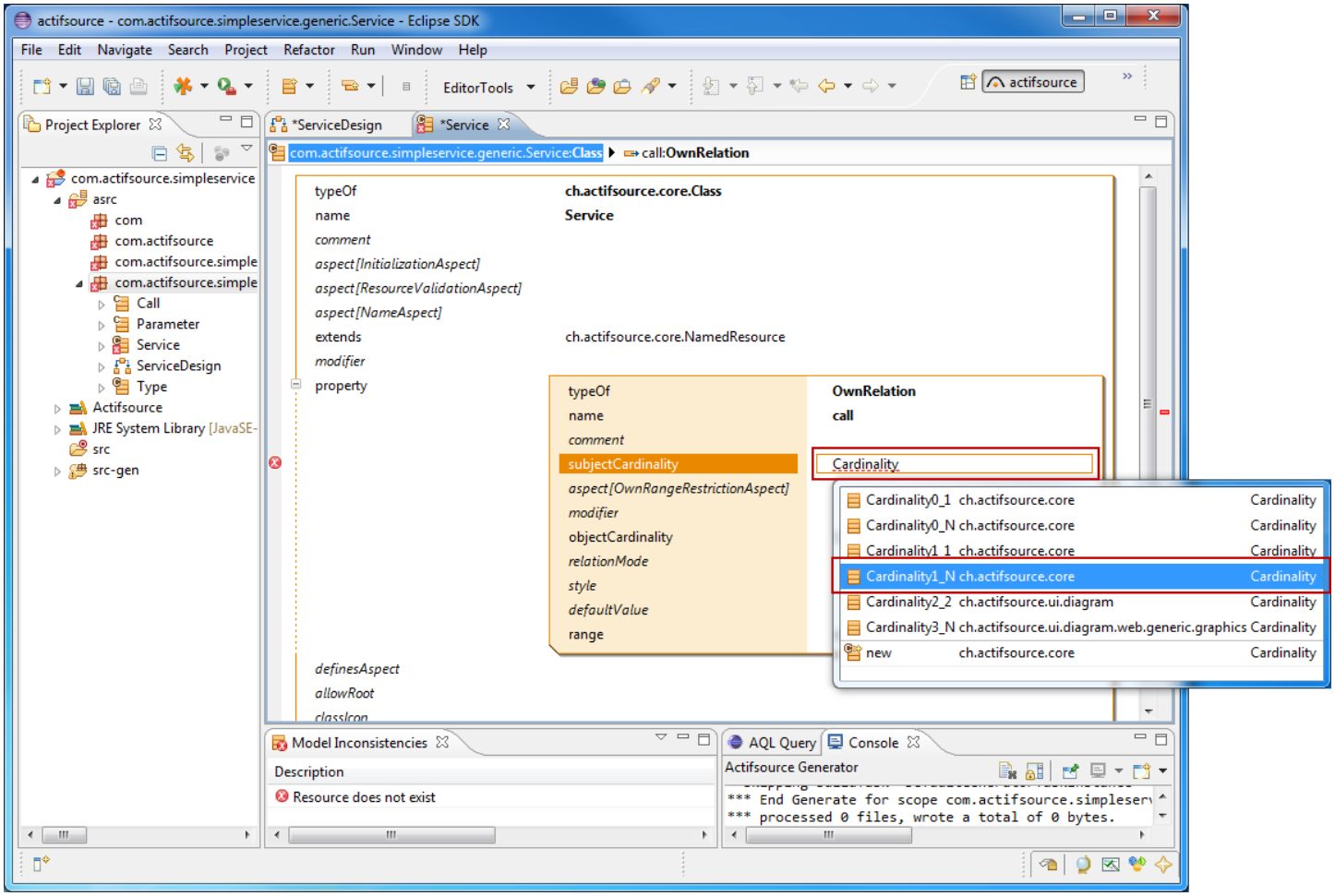


ⓘ You can open the **actifsource Resource Editor** on any **actifsource** resource by left-clicking on the desired resource while holding down the Ctrl-Key (Ctrl+ Left-Click)

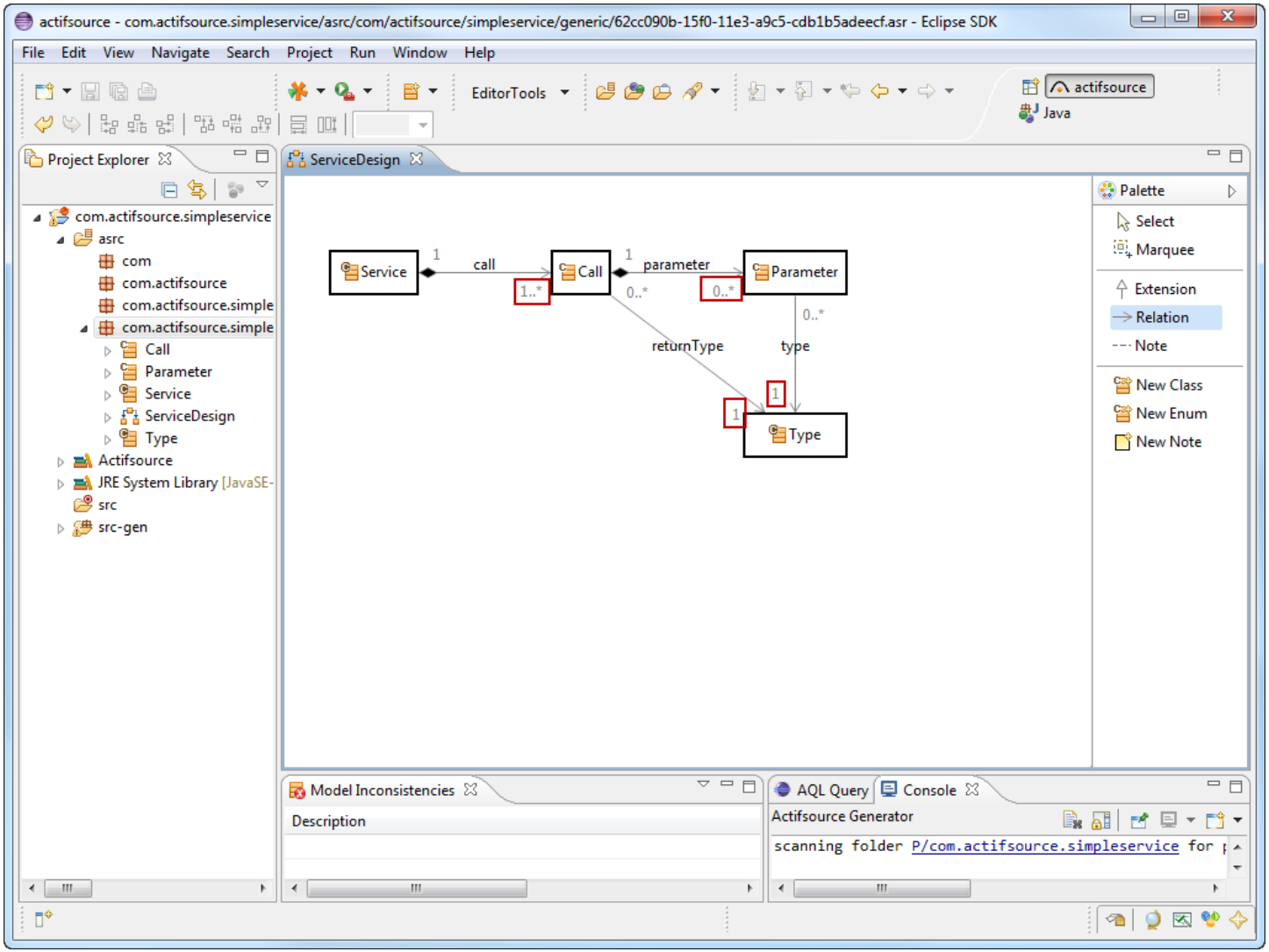
ⓘ Use the Cmd-Key on Apple computers

↳ Adjust the **subjectCardinality** of the call Relation

Adjust Cardinalities



- Change the **subjectCardinality** from **Cardinality0_N** to **Cardinality1_N**
- Use **Content Assist** (Ctrl+Space) and select the desired **Cardinality**

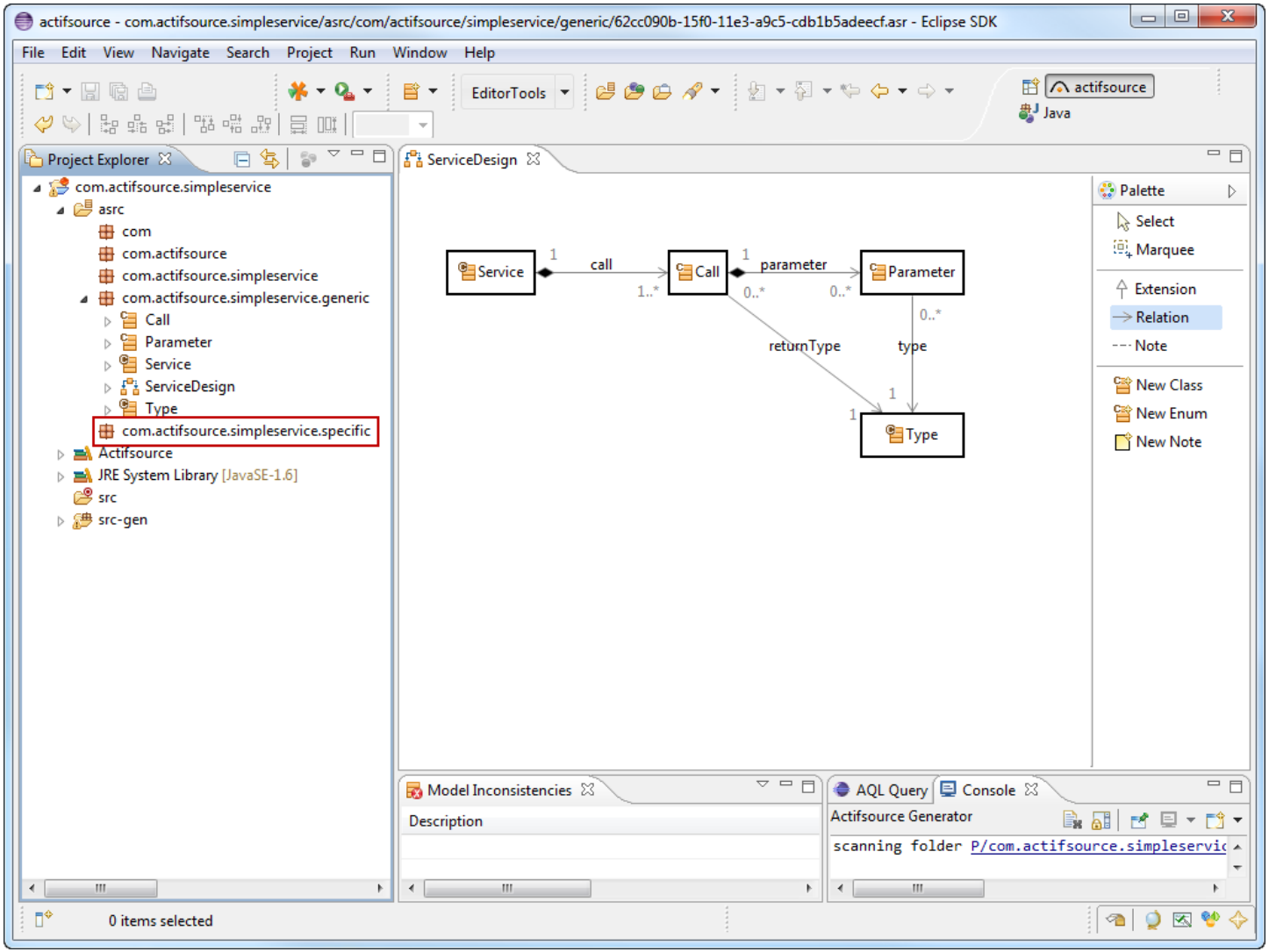


- Make sure to change all marked cardinalities as shown above
- Choose **subjectCardinality** **Cardinality1_1** for the relations type and returnType

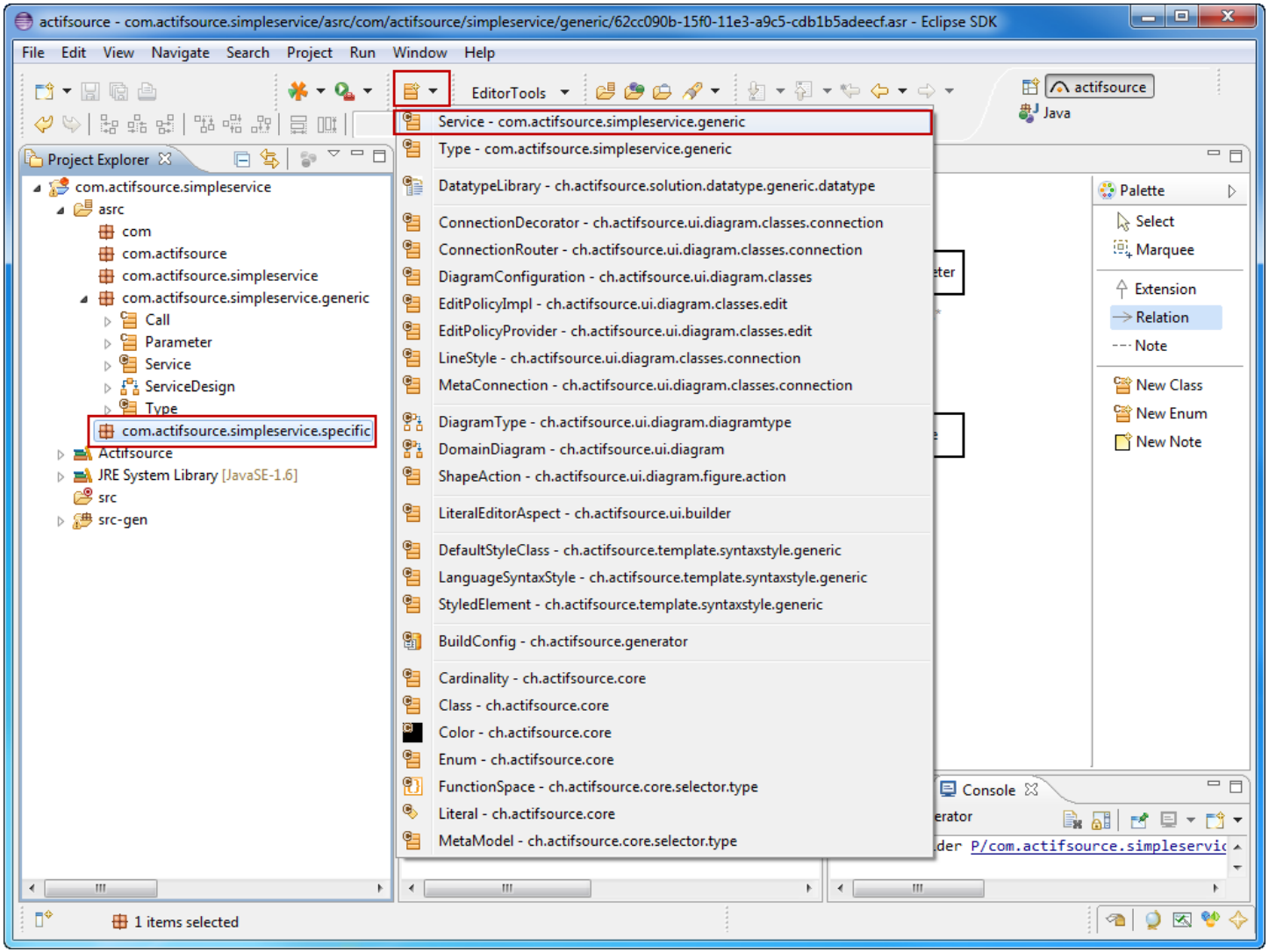
Part III: 32
Create a Specific Domain Model

- Create a package to store specific classes
- Create specific domain objects based on the generic domain model

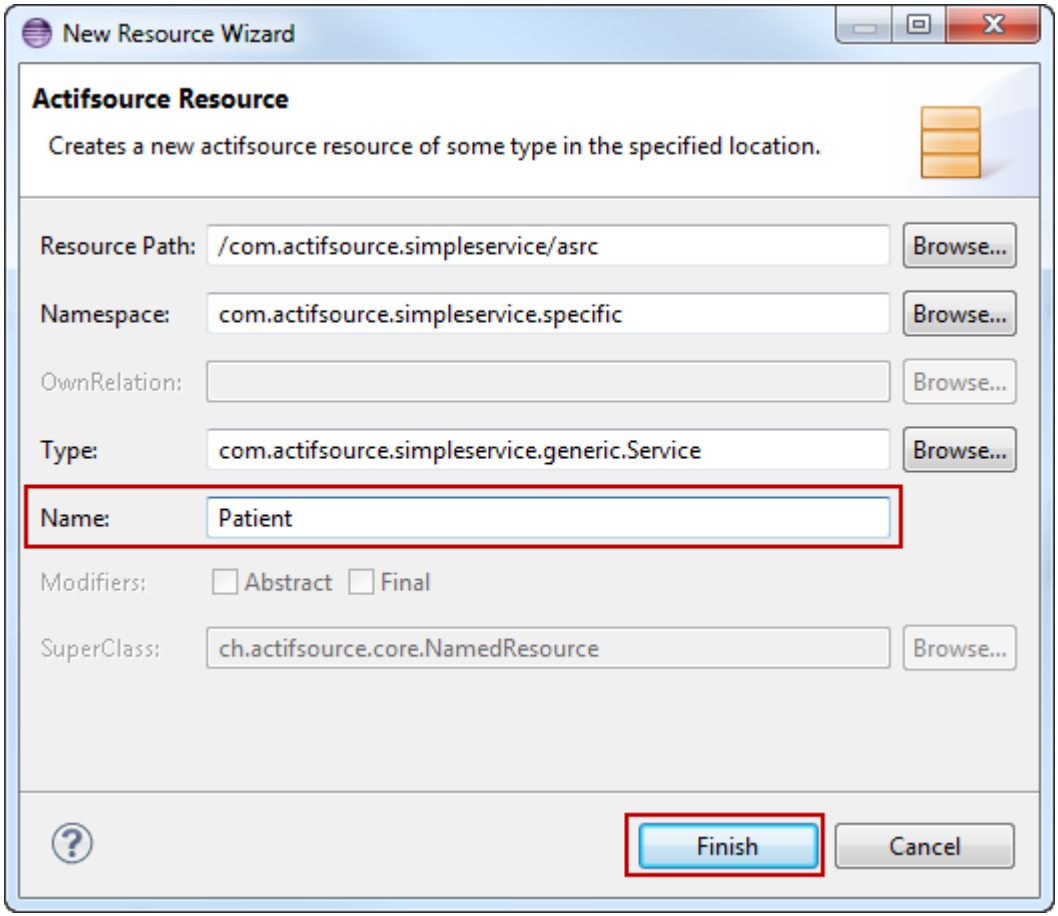
Create a Package to Store Specific Domain Objects 33



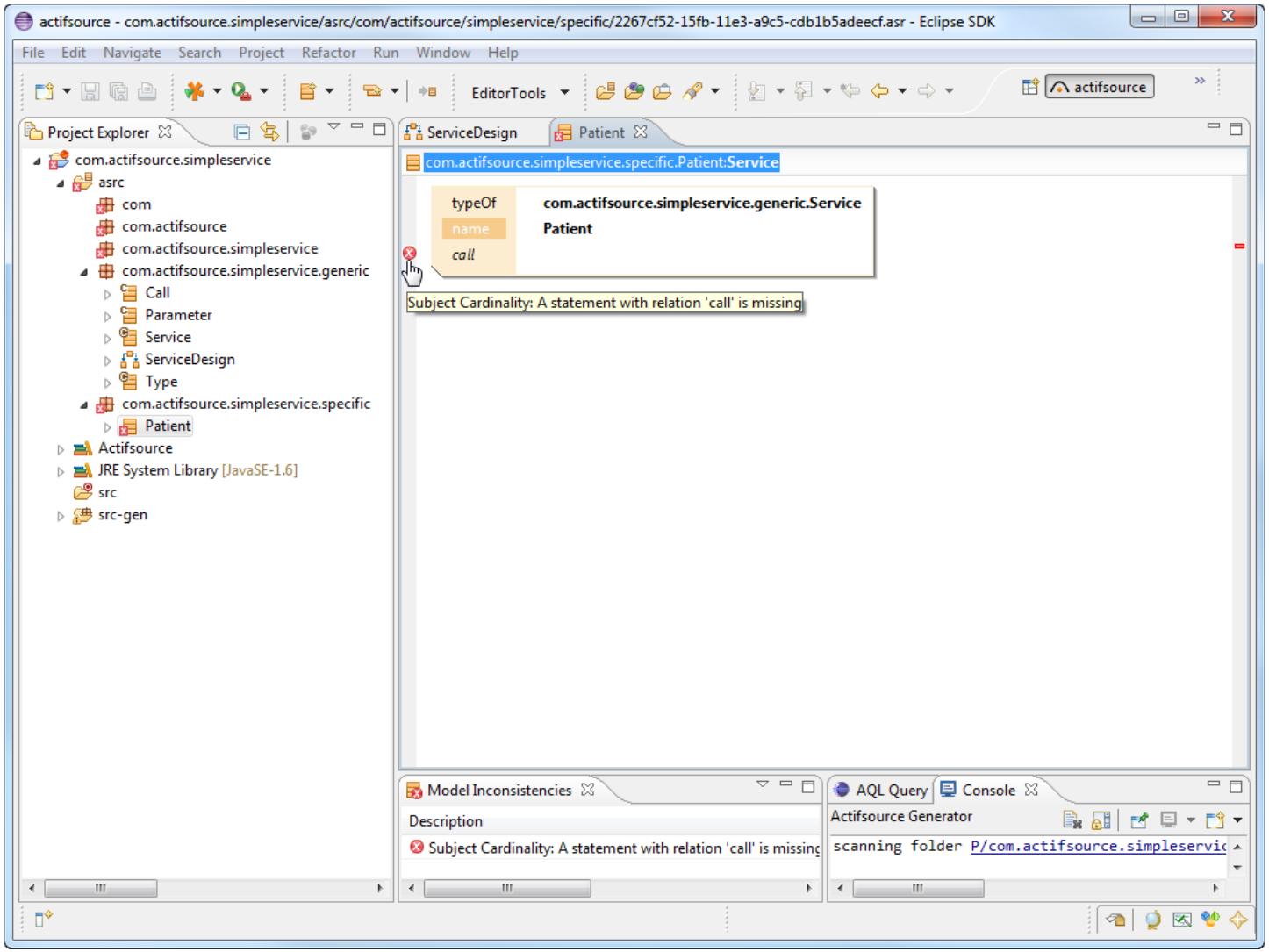
↳ Create a new **Package** `specific` in *simpleservice*



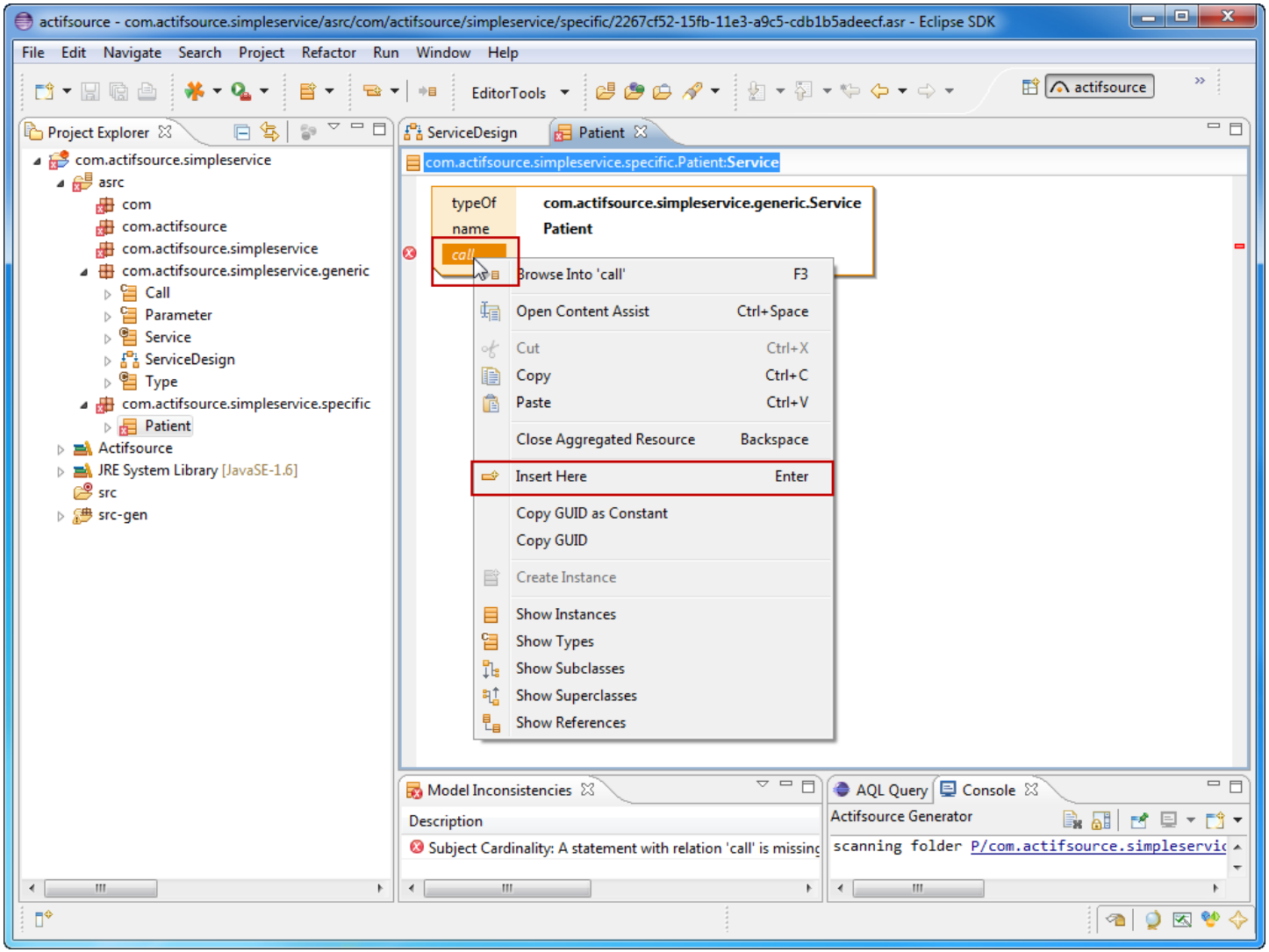
- Select the **Package** *specific*
- Use the **New Resource** Tool to create a **Resource** of type Service



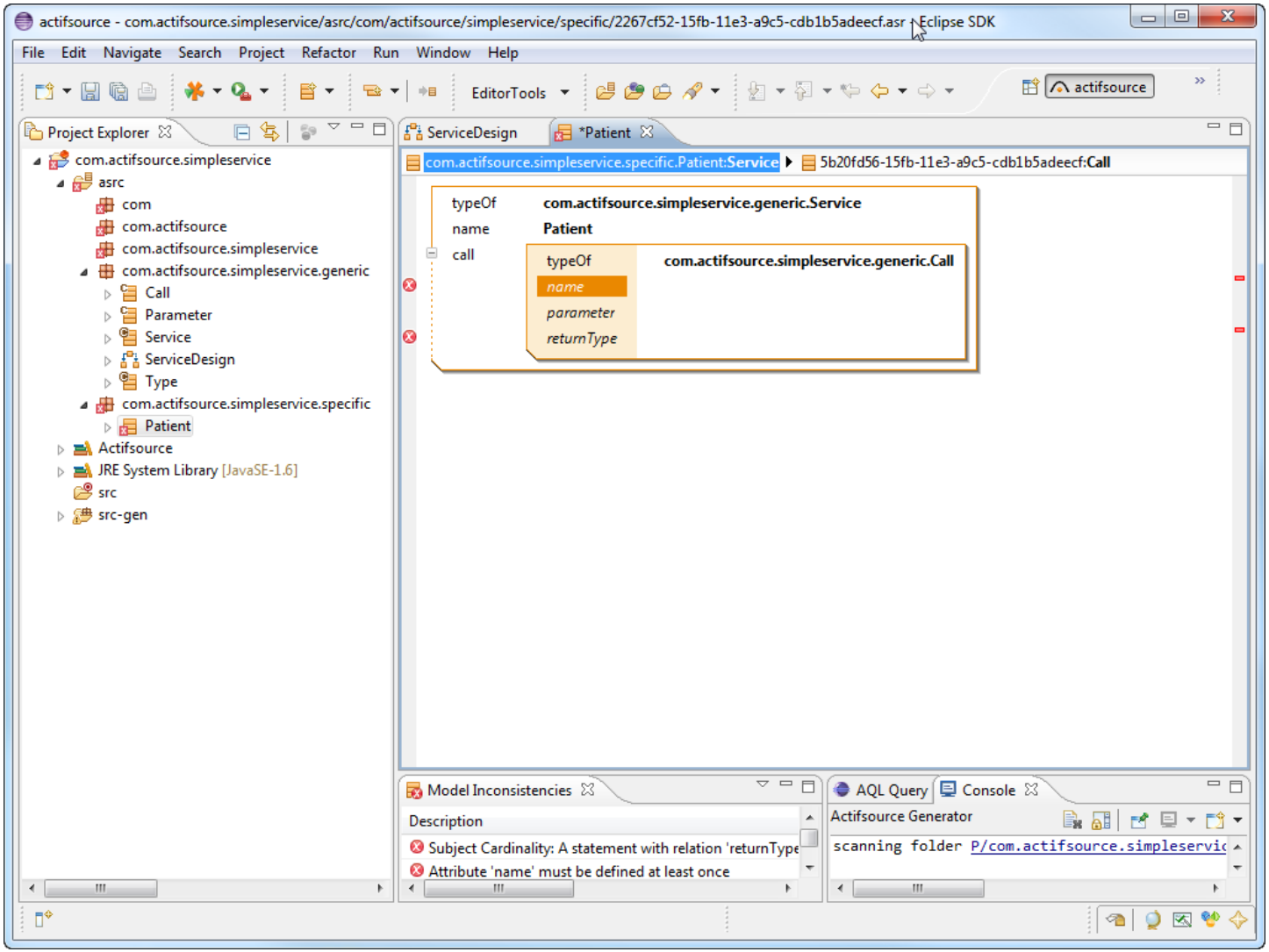
- Name the Service `Patient`
- Click *Finish*
- ⓘ Note: Being **Subclass** of **NamedResource** provides Patient with an **Attribute** **name**



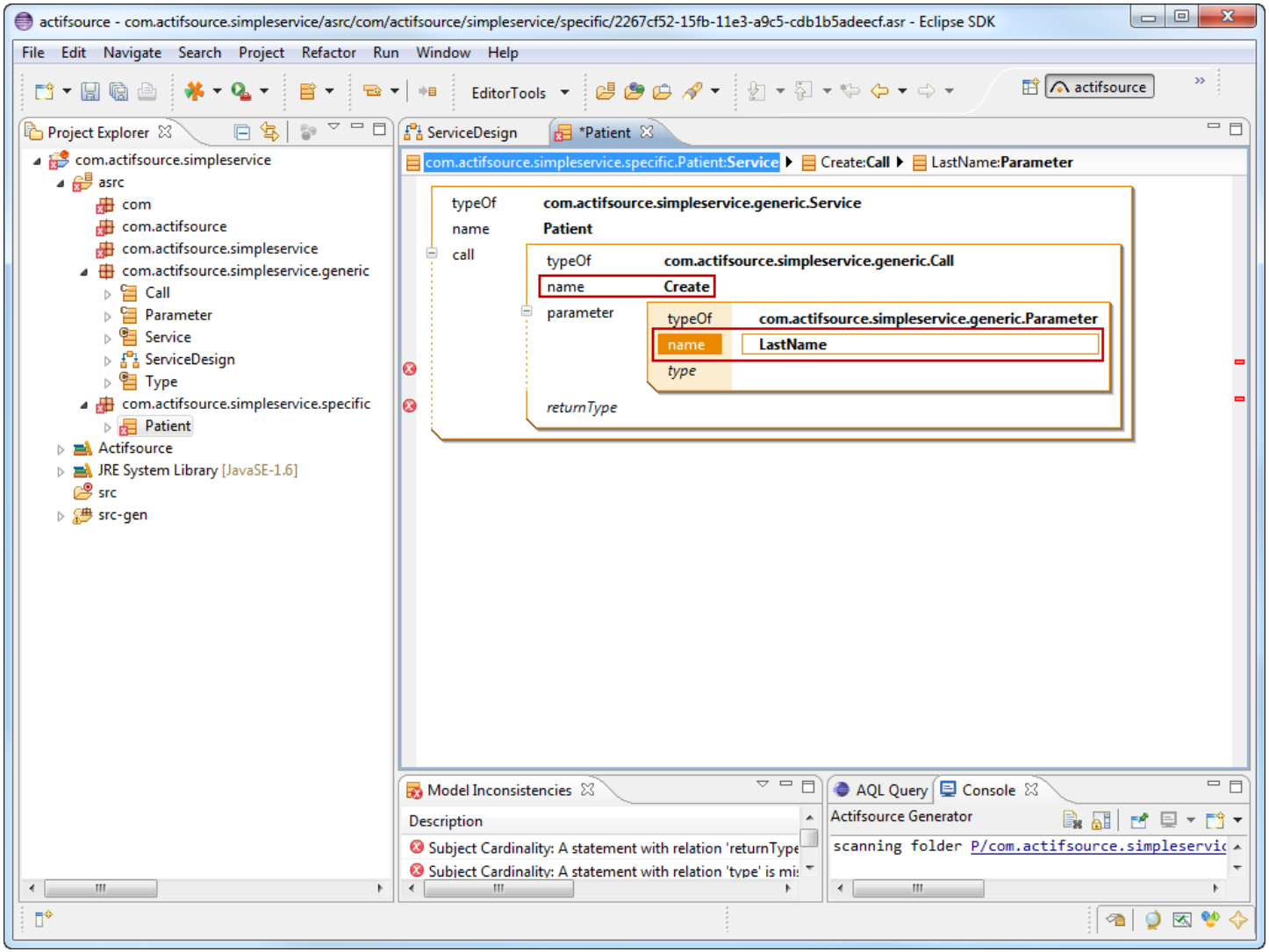
- ⓘ The **Resource** Patient of type Service has been created
- ⓘ The validator constantly checks all resources against the model
- ⓘ Since we decided a Service must have 1..N Call resources by selecting the **subjectCardinality Cardinality1_N,** the validator detects the incorrect subject cardinality and flags an error



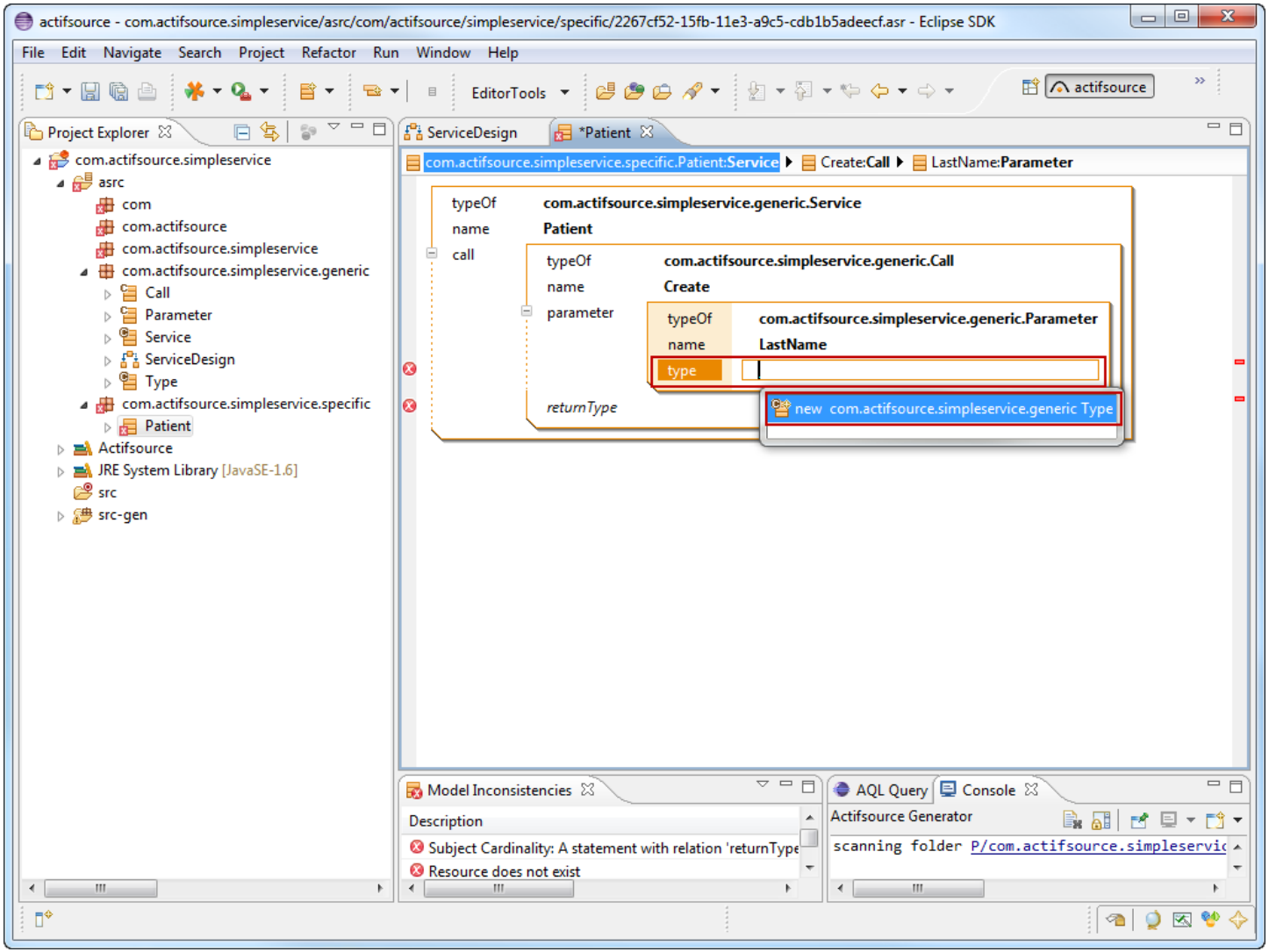
- Use the context menu on the **Relation** call to explore your options
- Press *Enter* to create a **Resource** of type Call for the Patient Service



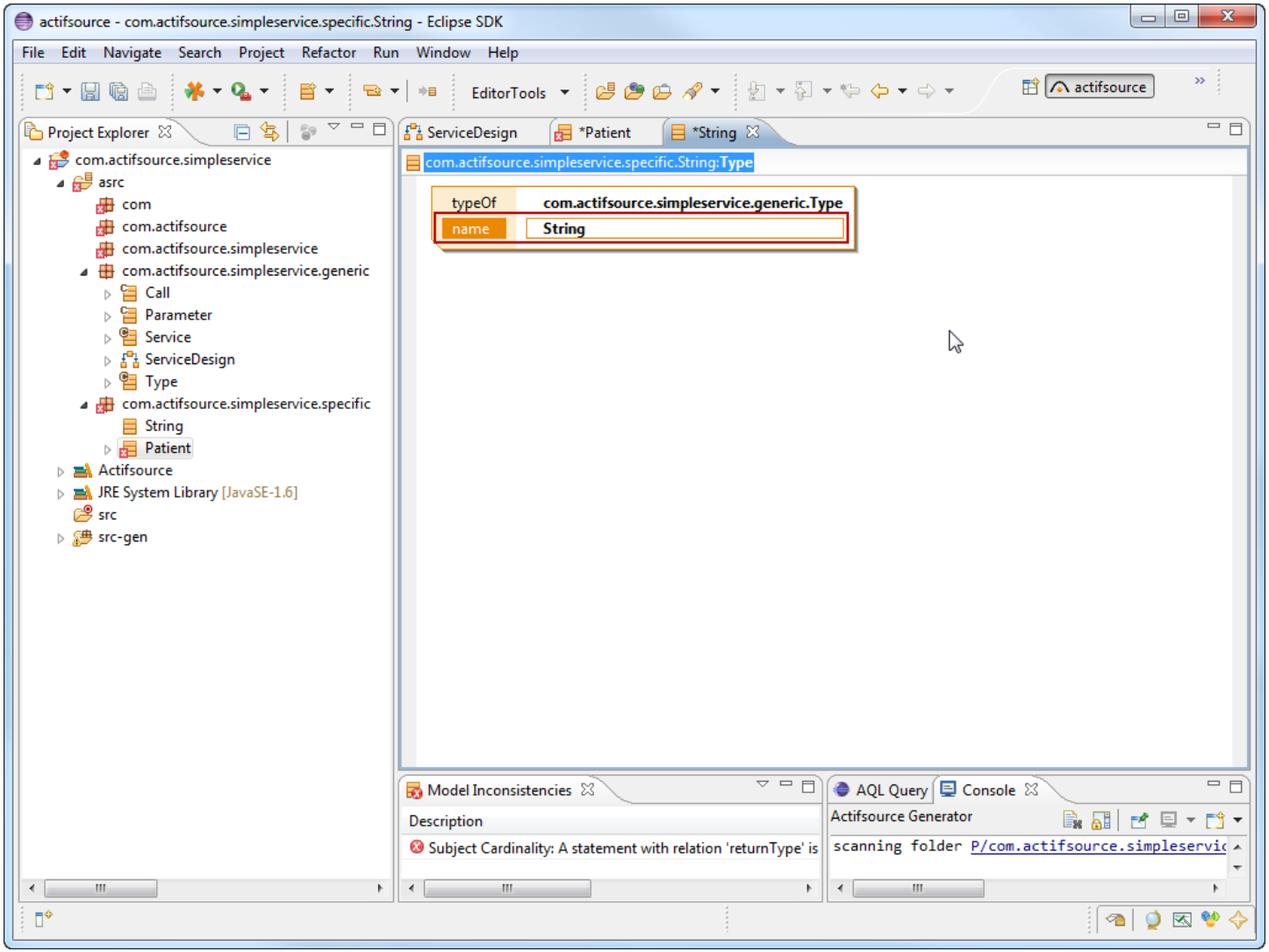
ⓘ The validator tells you that the created Call is missing a **name** and the returnType



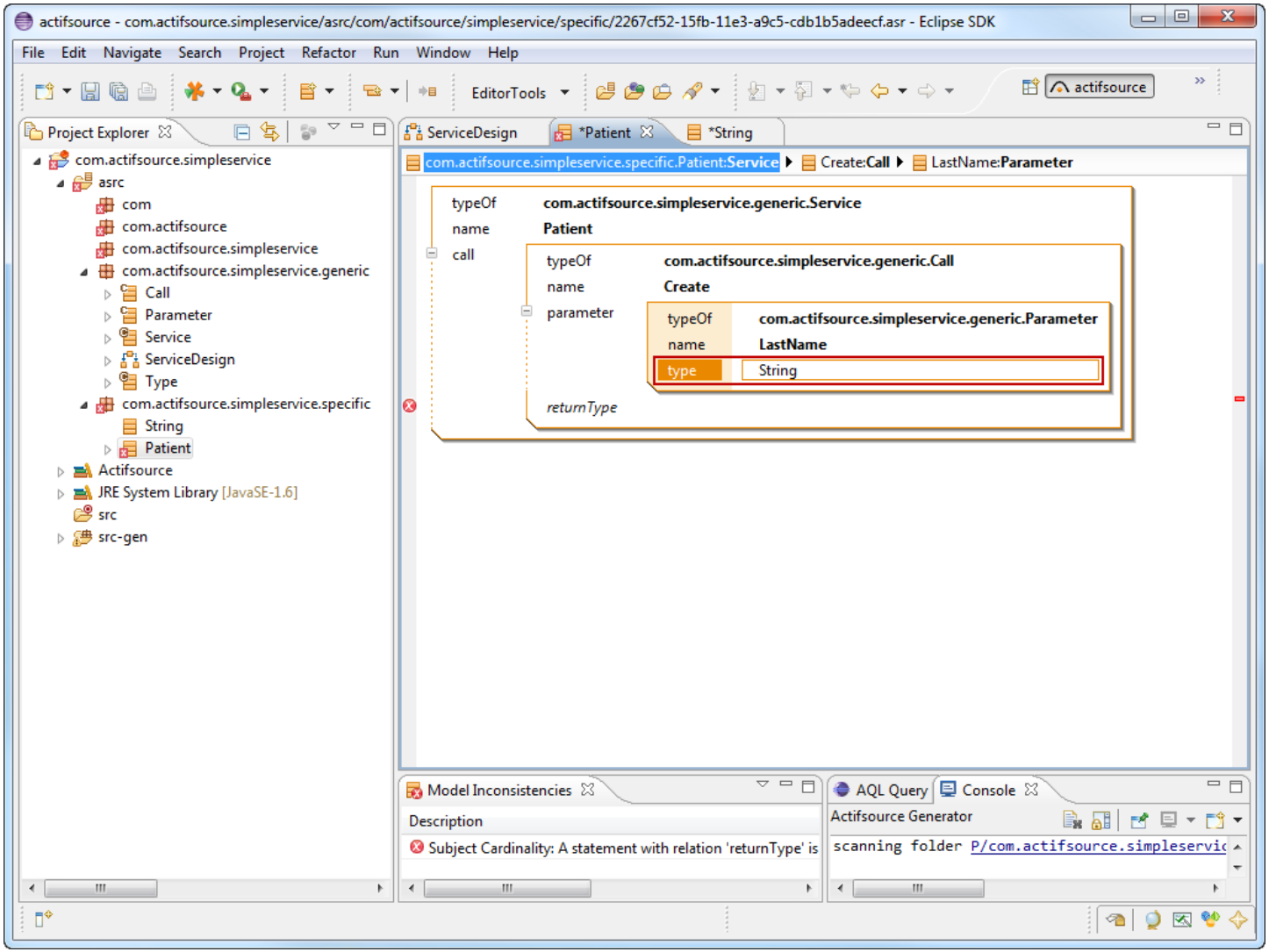
- Enter the name `Create` by pressing *enter* on ***name*** or use the context menu
- Enter a new Parameter by pressing *enter* on parameter or use the context menu
- Enter the name `LastName` of the created Parameter
- ⓘ Type and returnType are still missing



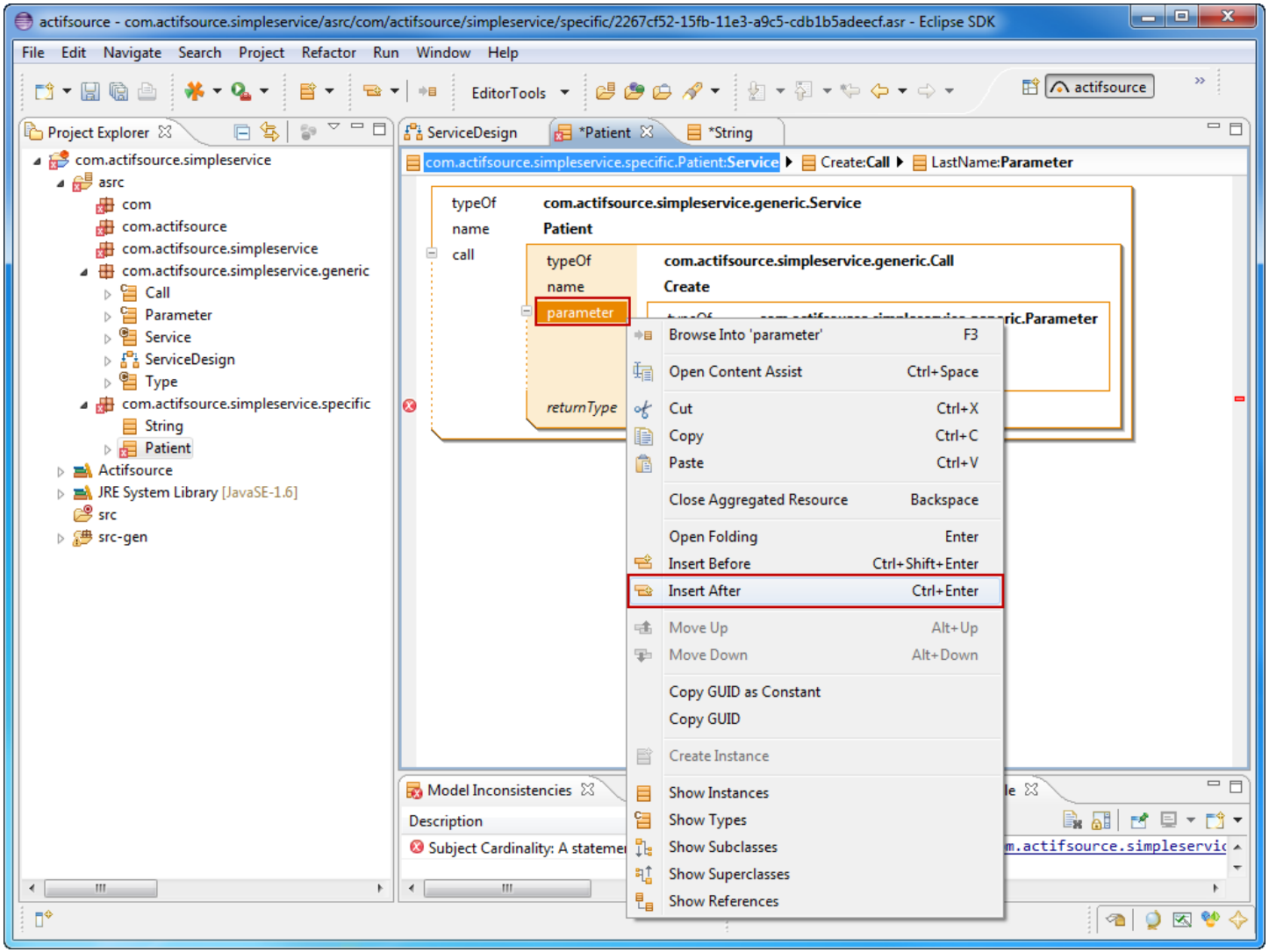
- ⓘ So far there is no **Resource** of type Type
- ⓘ Type of Parameter *LastName* shall be String
- ↳ Activate **Content Assist** (Ctrl+Space) to insert a **Resource** of type Type
- ↳ Create a new Type String using **Content Assist** (Ctrl+Space)



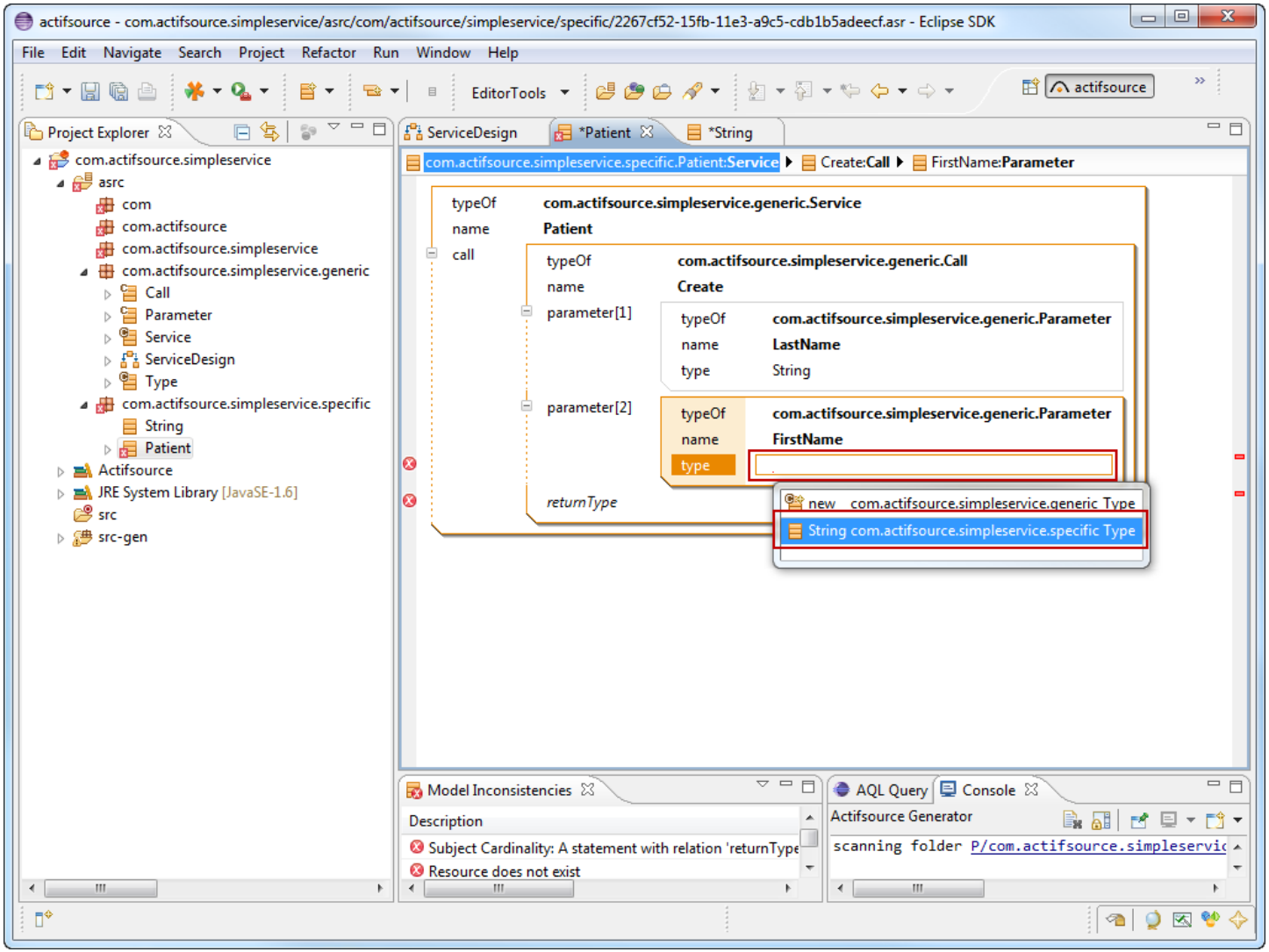
- Name the new Type `String`
- Switch back to the Service Patient by clicking the *Patient* Tab or by pressing Alt+Left to navigate to the last edit location



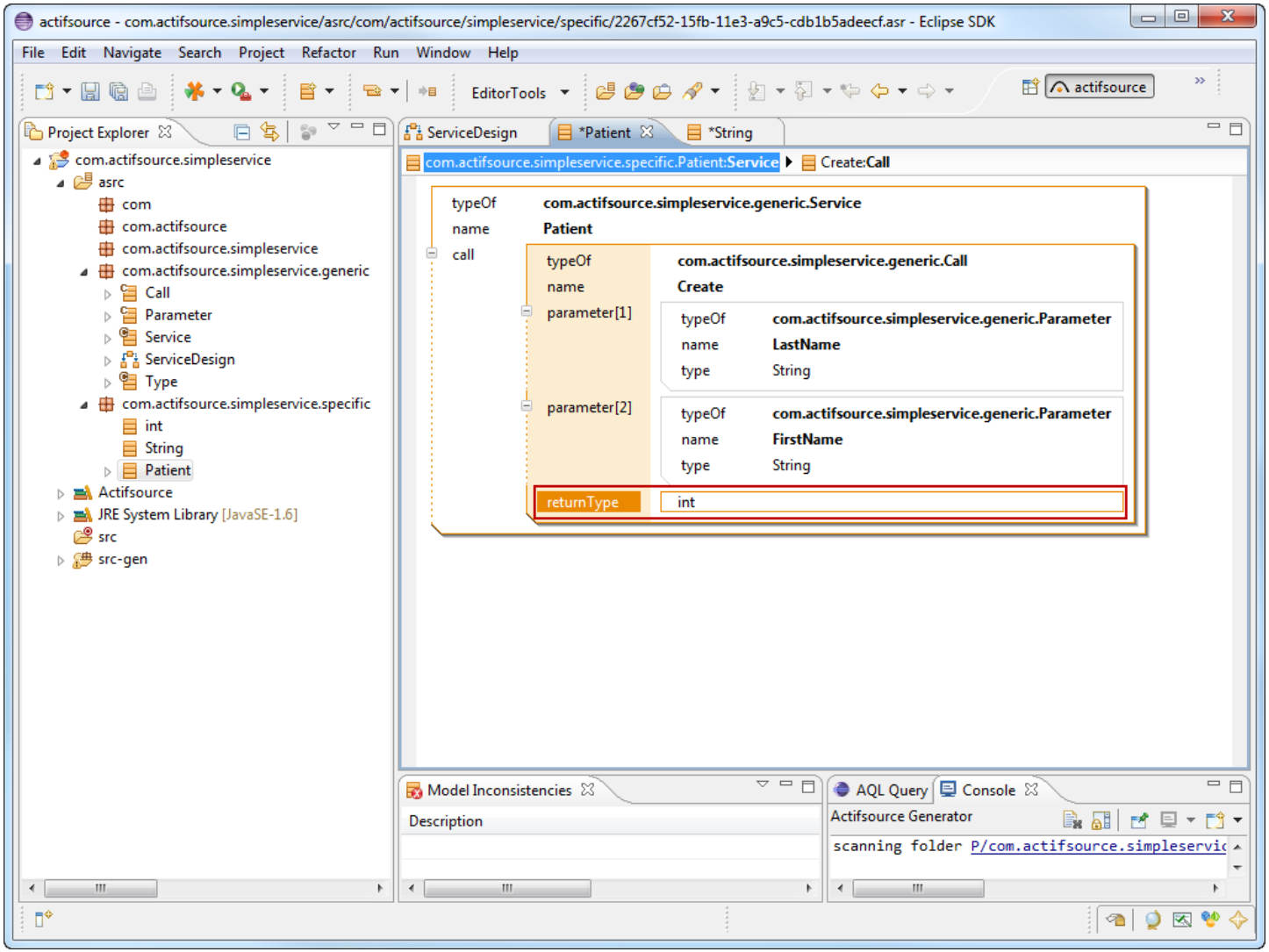
ⓘ The created Type String is inserted automatically



- Select *parameter*
- Insert a new Parameter using *Insert After* (Ctrl+Enter) or *Insert Before* (Alt+Shift+Enter)
- ⓘ You may rearrange the resources as you see fit using *Move Up* (Alt+Up) or *Move Down* (Alt+Down)

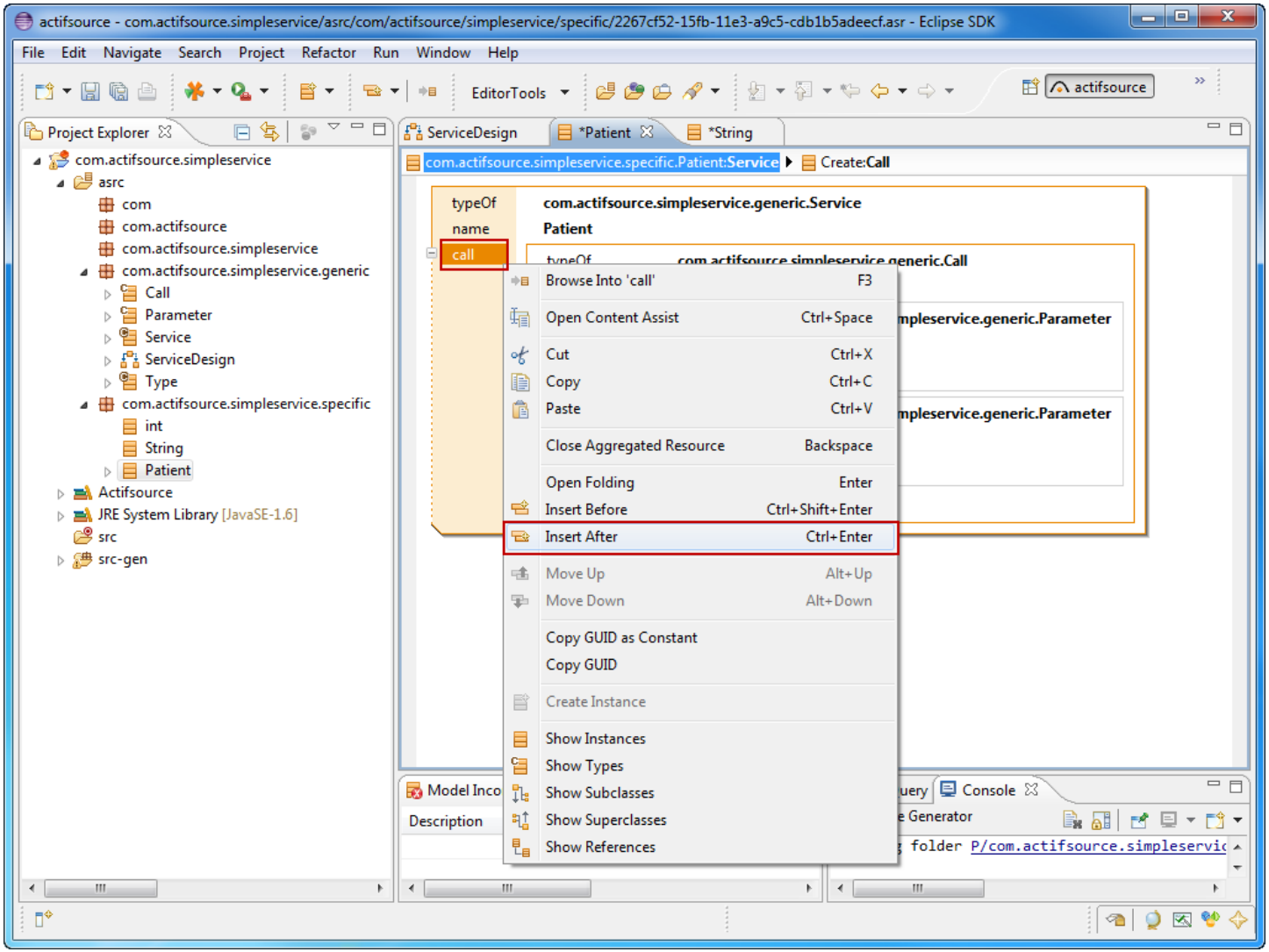


- Name the Parameter `FirstName`
- Insert the already created Type String using **Content Assist** (Ctrl+Space)

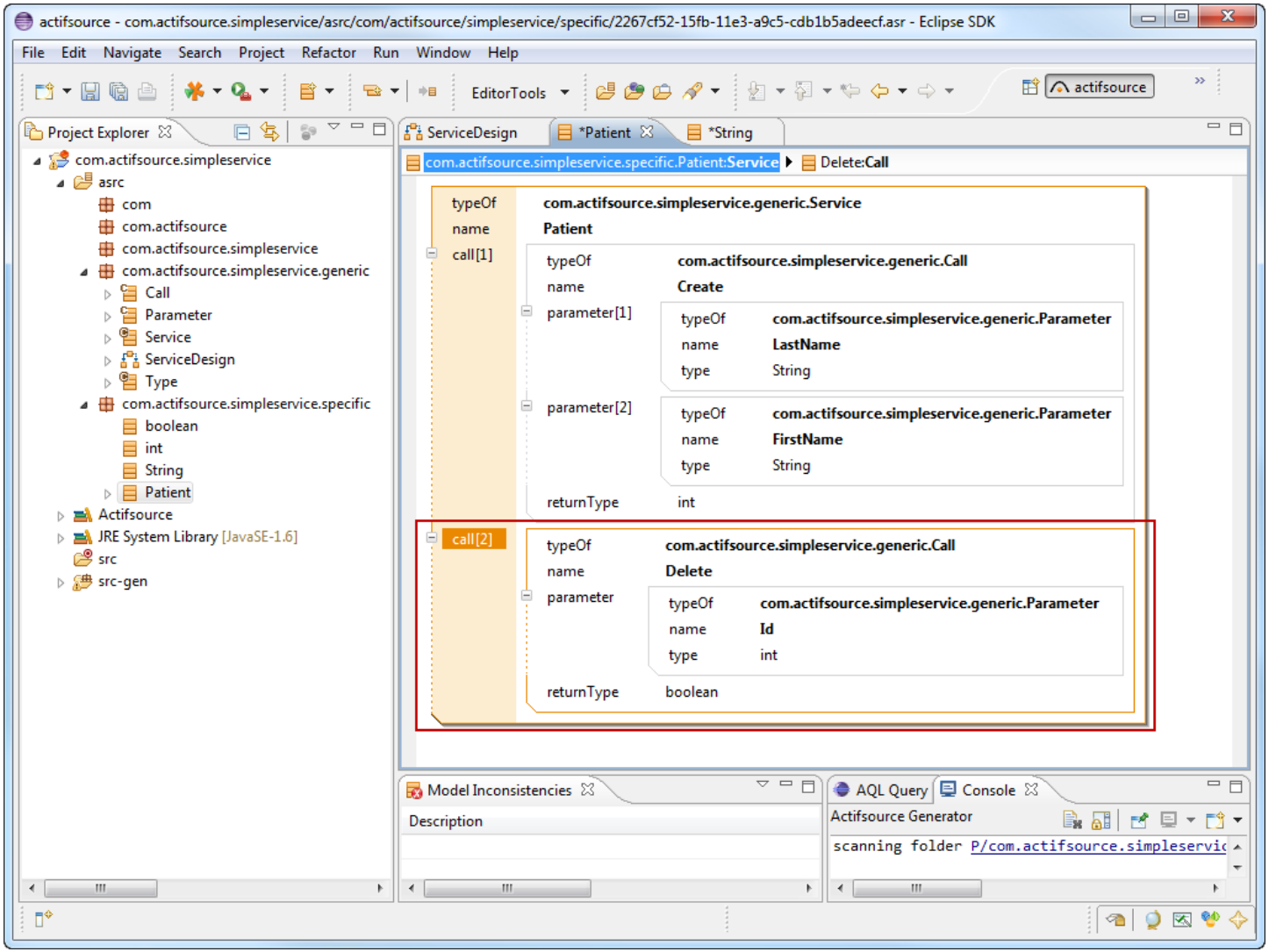


↳ For returnType create a new Type int using **Content Assist** (Ctrl+Space)

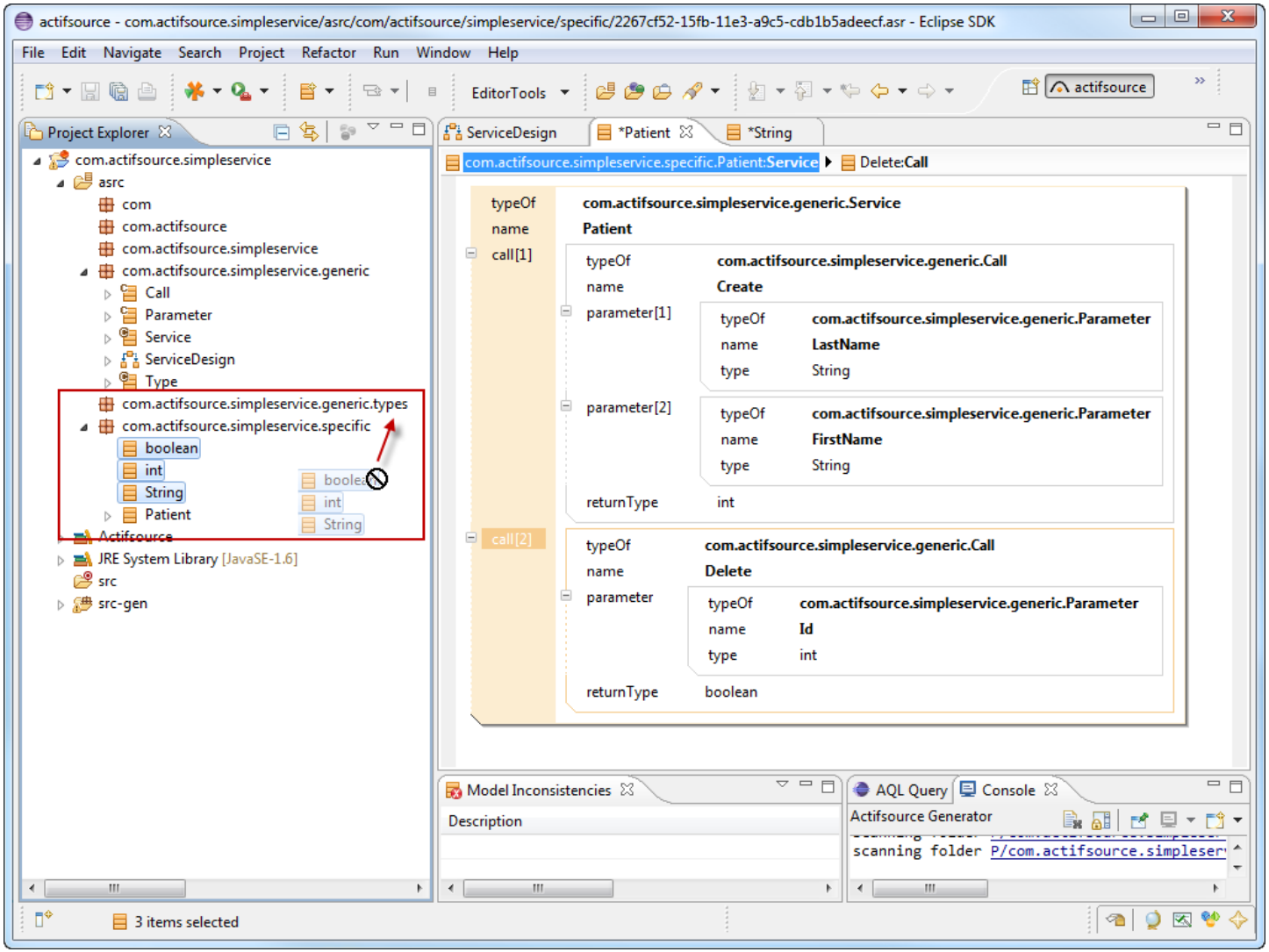
ⓘ The validator reports no more *Model Inconsistencies*



- Select call
- Insert a new Call using *Insert After* (Ctrl+Enter) or *Insert Before* (Alt+Shift+Enter)
- ⓘ You may rearrange the resources using *Move Up* (Alt+Up) or *Move Down* (Alt+Down)



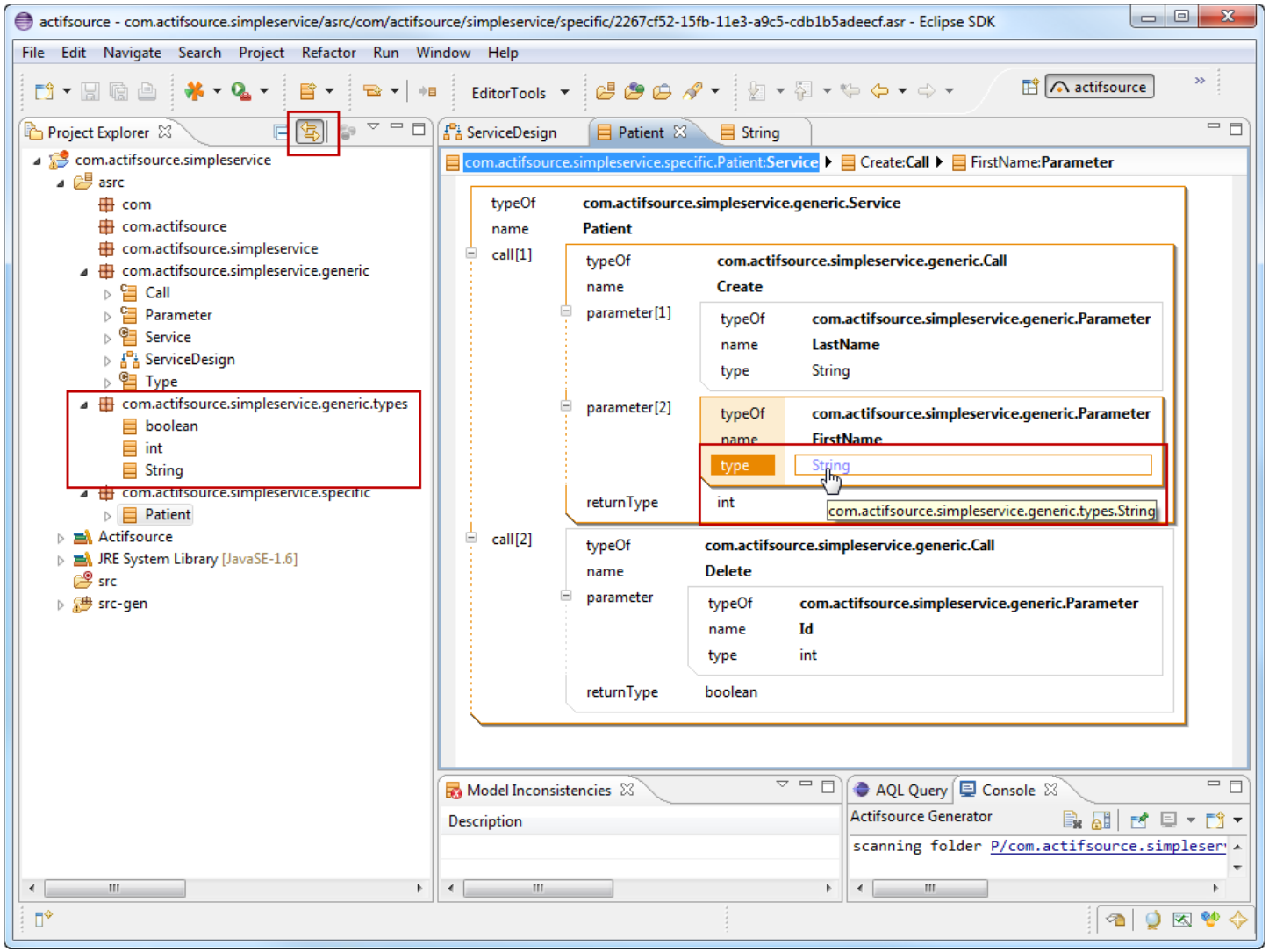
- Edit the Call Delete as shown above
- Insert Parameter Id of Type int
- Insert returnType boolean



ⓘ New resources are created in the same **Package** as they are referenced

↳ Create a new **Package** `types` in *generic*

↳ Move the Types boolean, int and String into the **Package** *types* using drag & drop

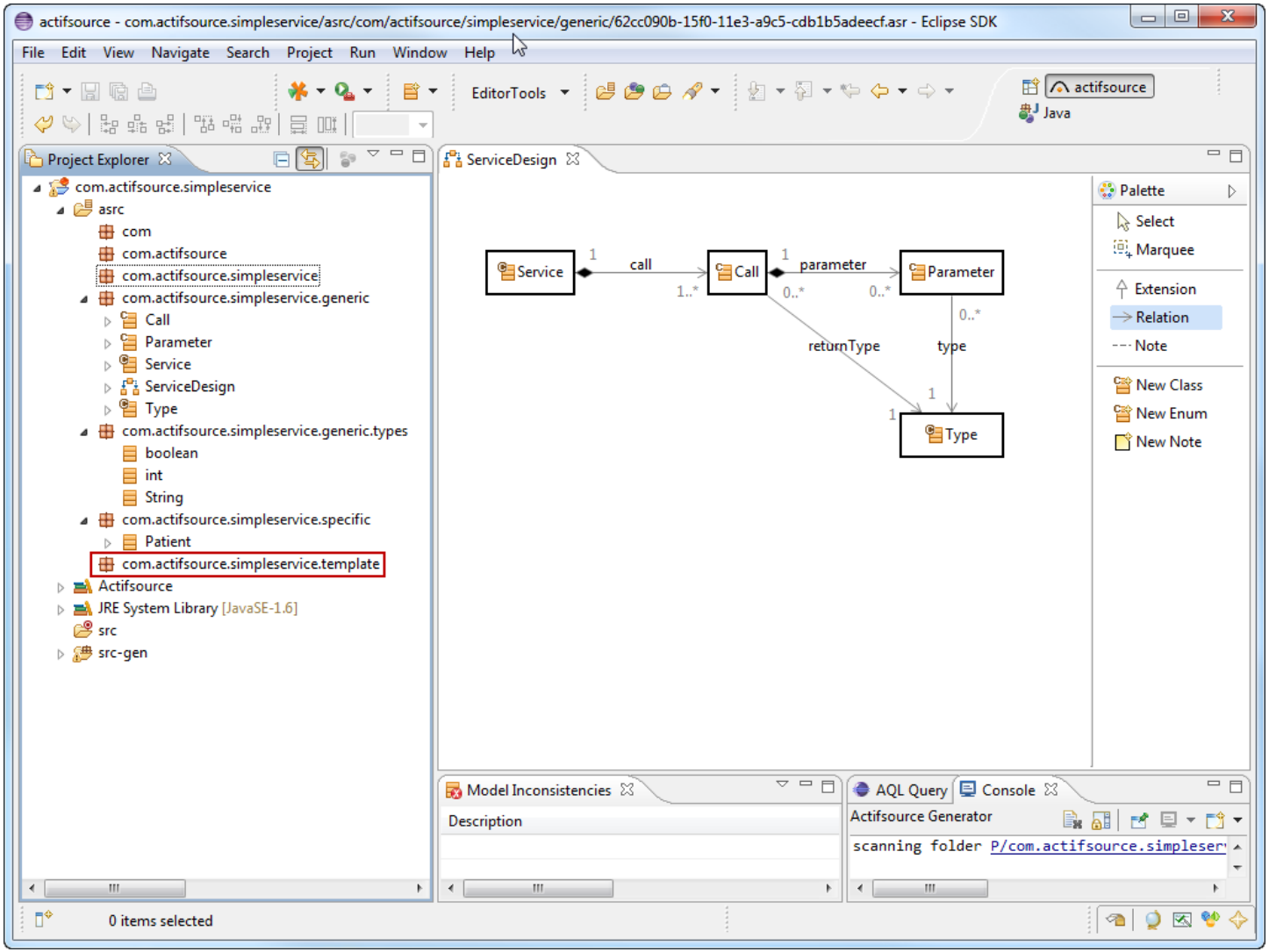


↳ Activate **Link with Editor** to link the **Project Explorer** with the editor

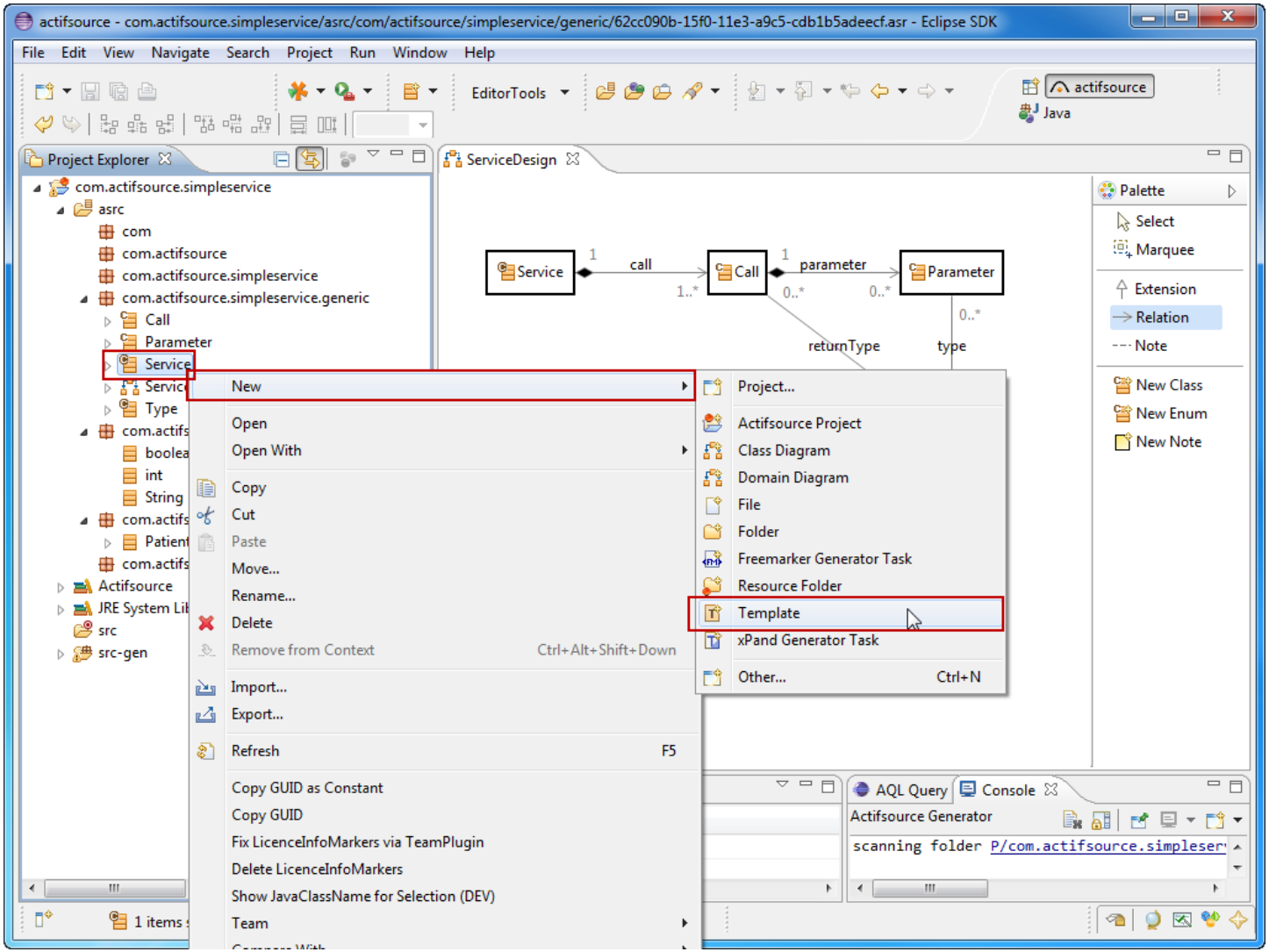
ⓘ Notice that types can still be located by Ctrl+Mouse-Left-Click on any type in the **Resource Editor**

Part IV: 50
Use Code Templates to Generate Specific Code

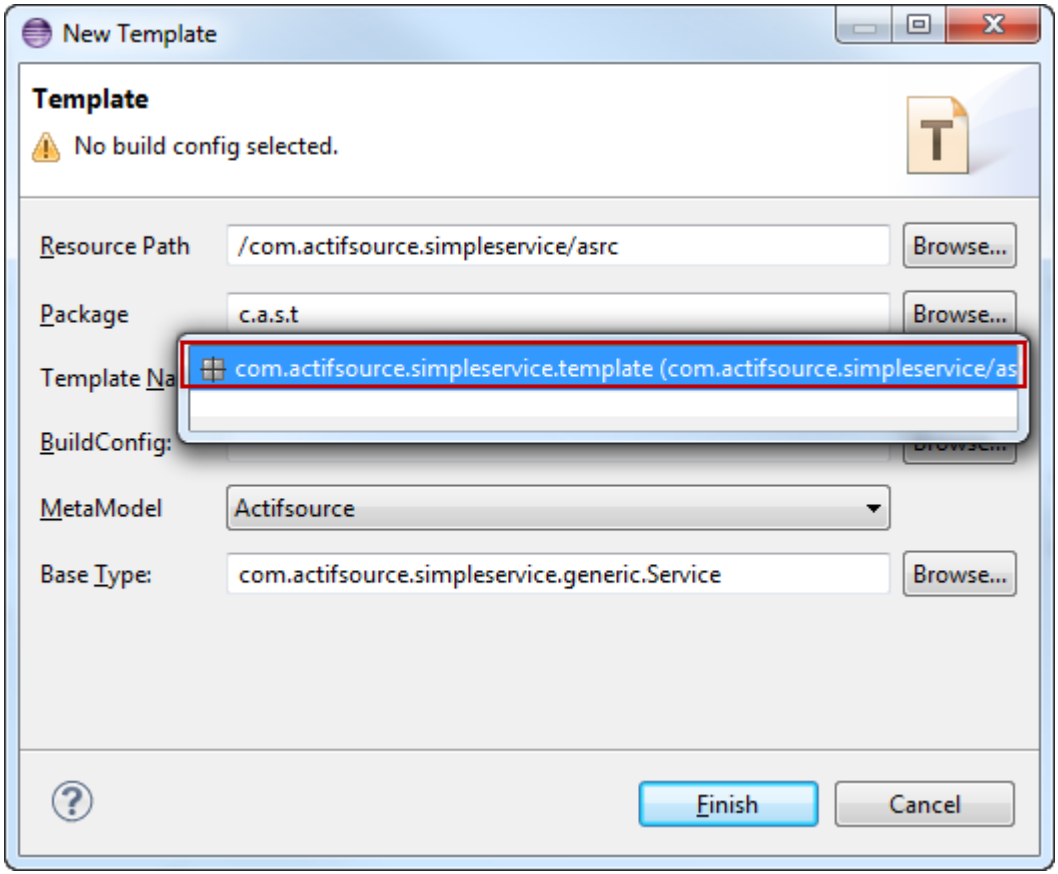
- Create a generator template
- Enable Java functions for templates
- Write generic code based on the generic domain model and generate code on the fly



- ⓘ **Code Templates** are also **Resources** in **actifsource**
- ⓘ Like any **Resource**, a **Template** is located in a **Package**
- ↳ Create a new **Package** `template` in *simpleservice*

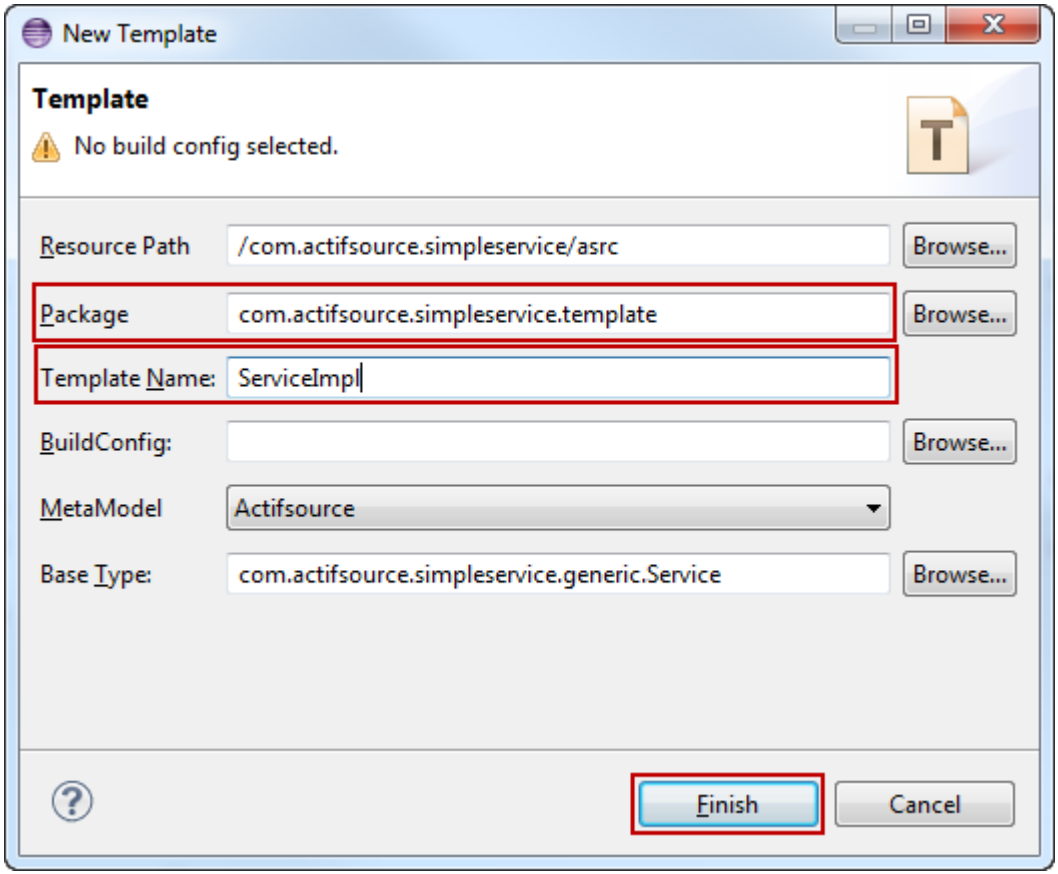


- ⓘ Let's write template code for any Service
- ↳ Select the **Resource** Service
- ↳ Choose *New Template* from the context menu

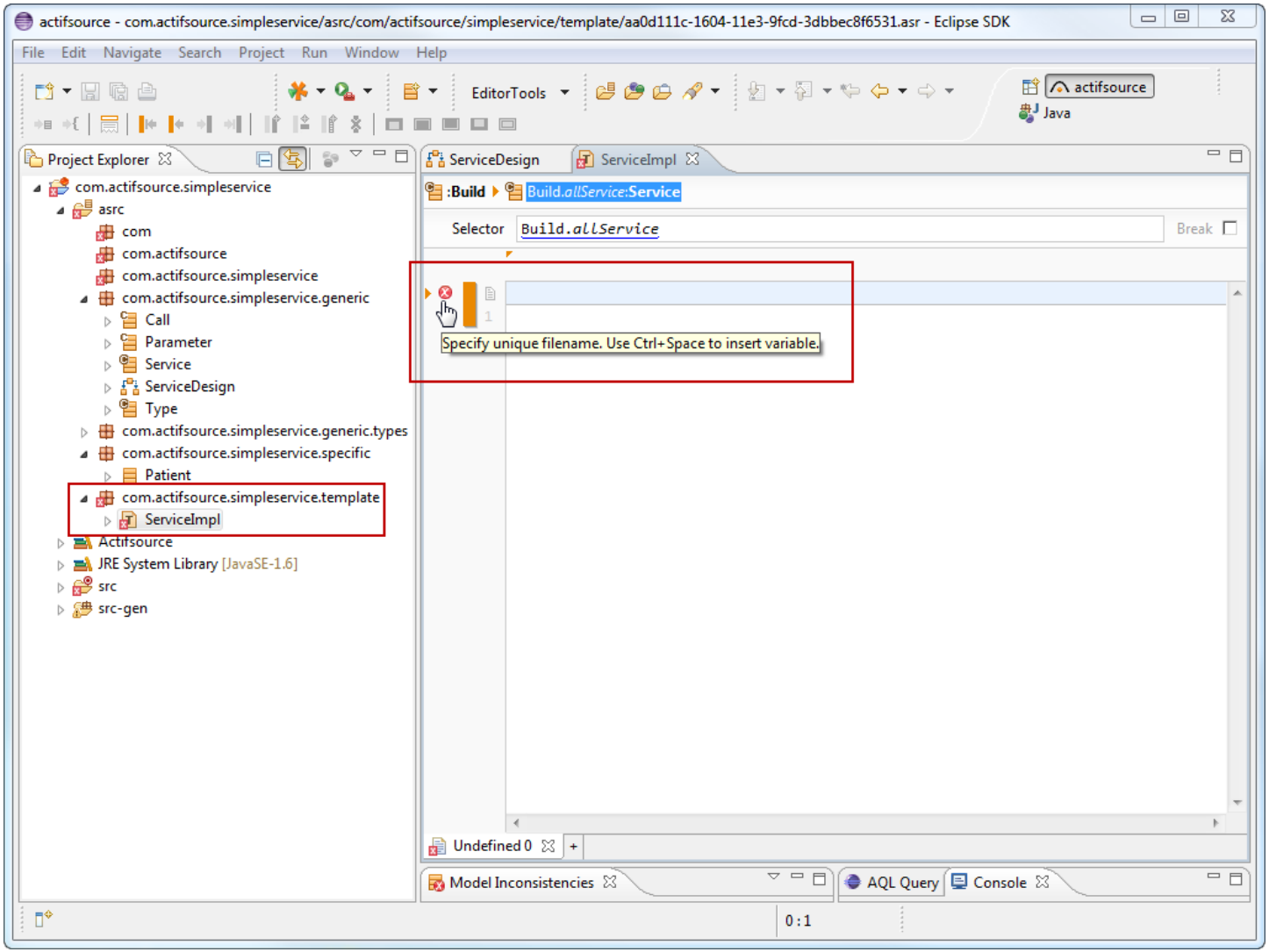


↳ Choose the **Package** *template* using **Content Assist** (Ctrl+Space)

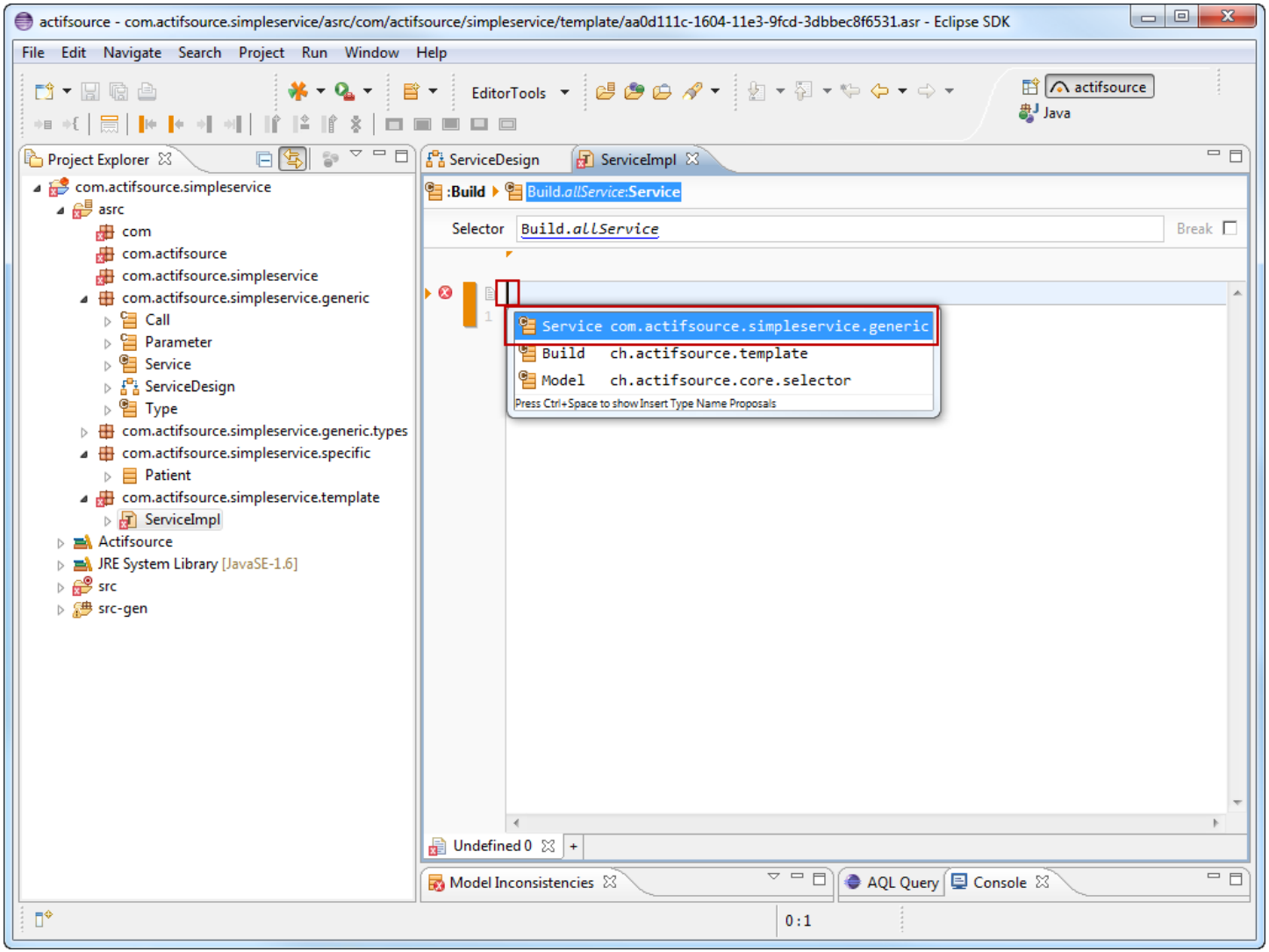
ⓘ You might use just the first letters to select your packages



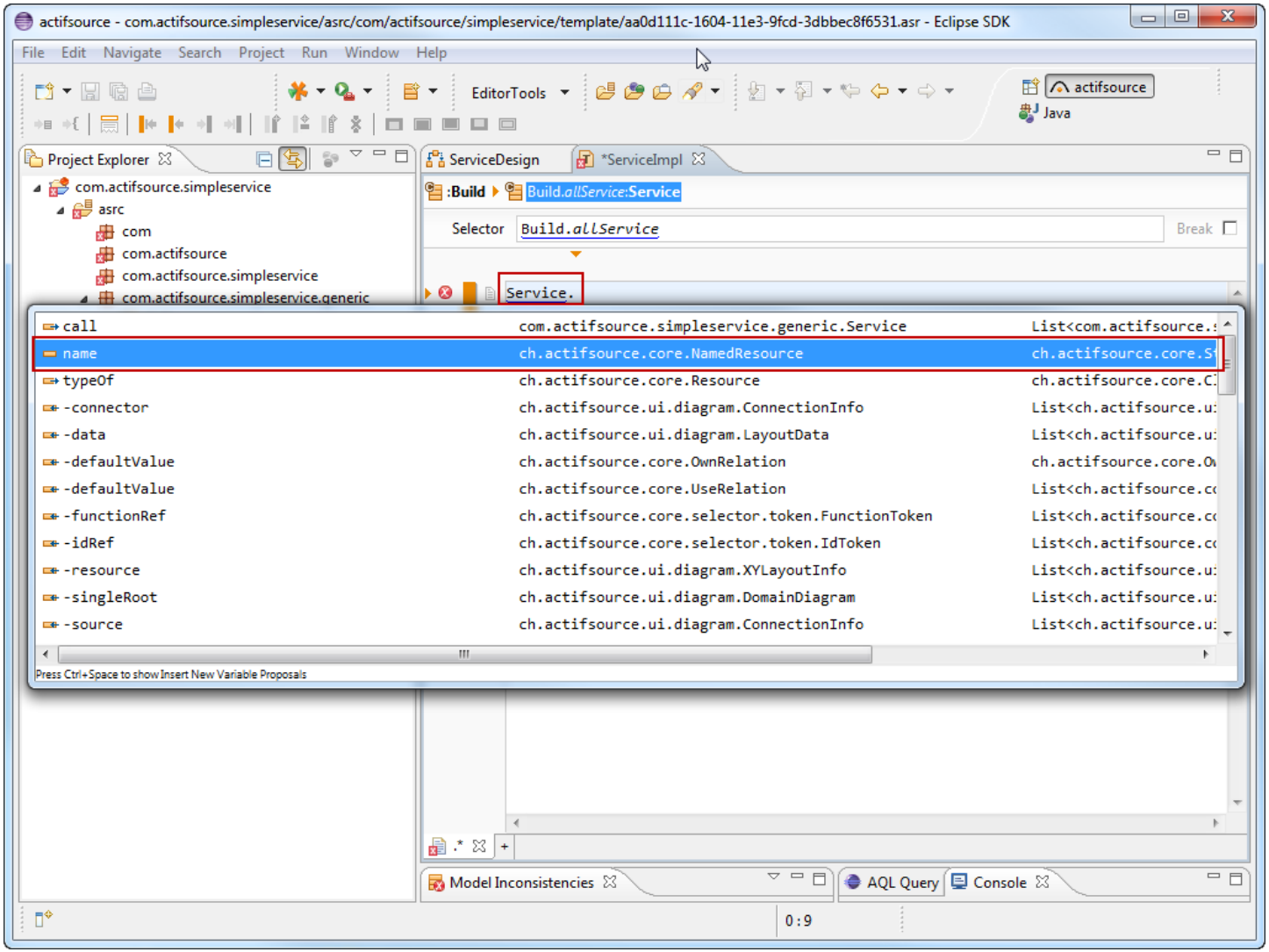
- Name the template `ServiceImpl`
- Click *Finish*



- ⓘ Note: The **Actifsource Template Editor** is a text editor
- ⓘ Writing a template for Service means that any specific information on services comes from the specific domain model
- ⓘ The generic filename must contain a **Variable** to be unique for any specific Service

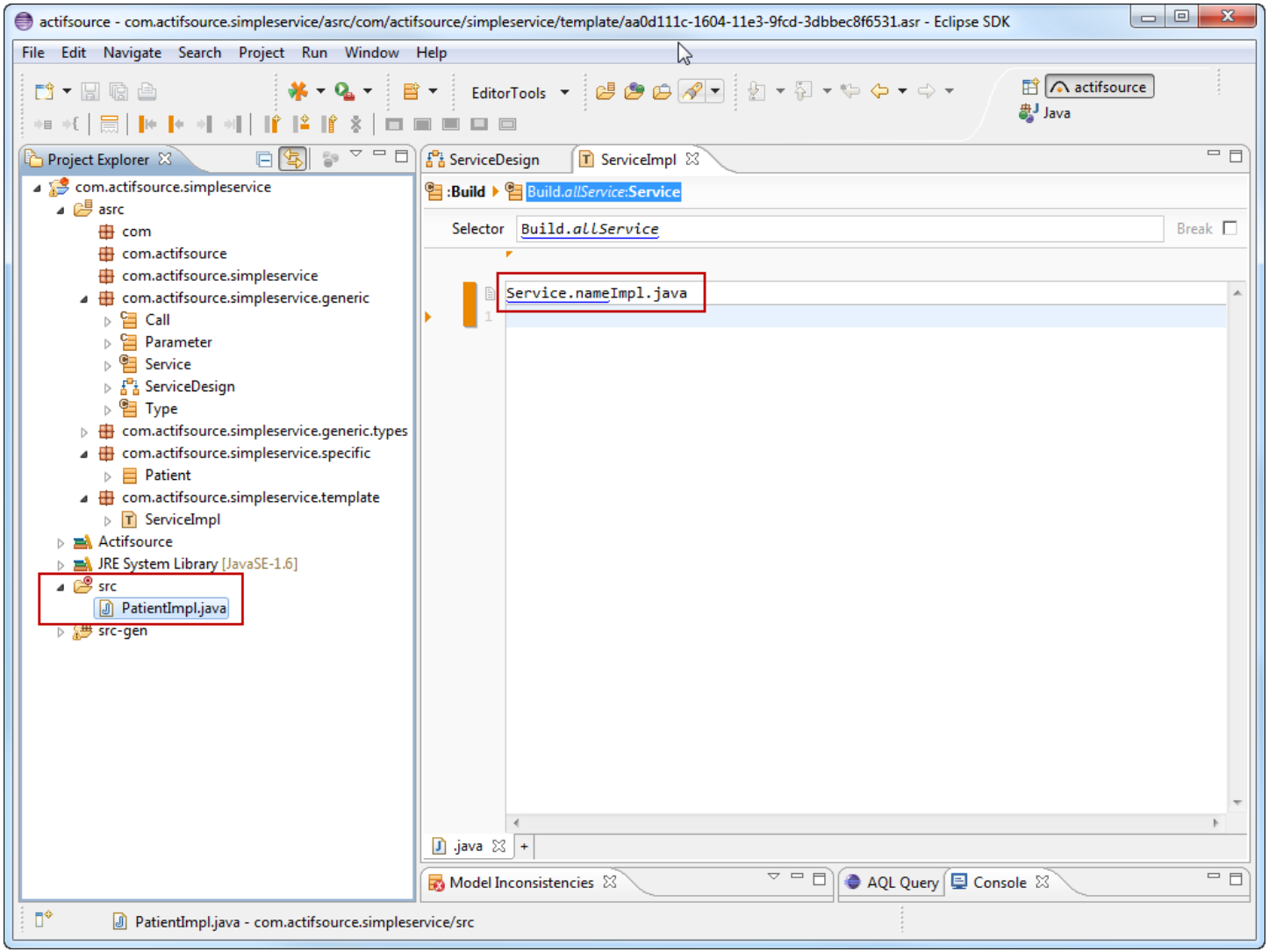


- Place the cursor on the **Filename Line**
- Use **Content Assist** (Ctrl+Space) to insert a **Variable**
- Choose the **Resource** Service

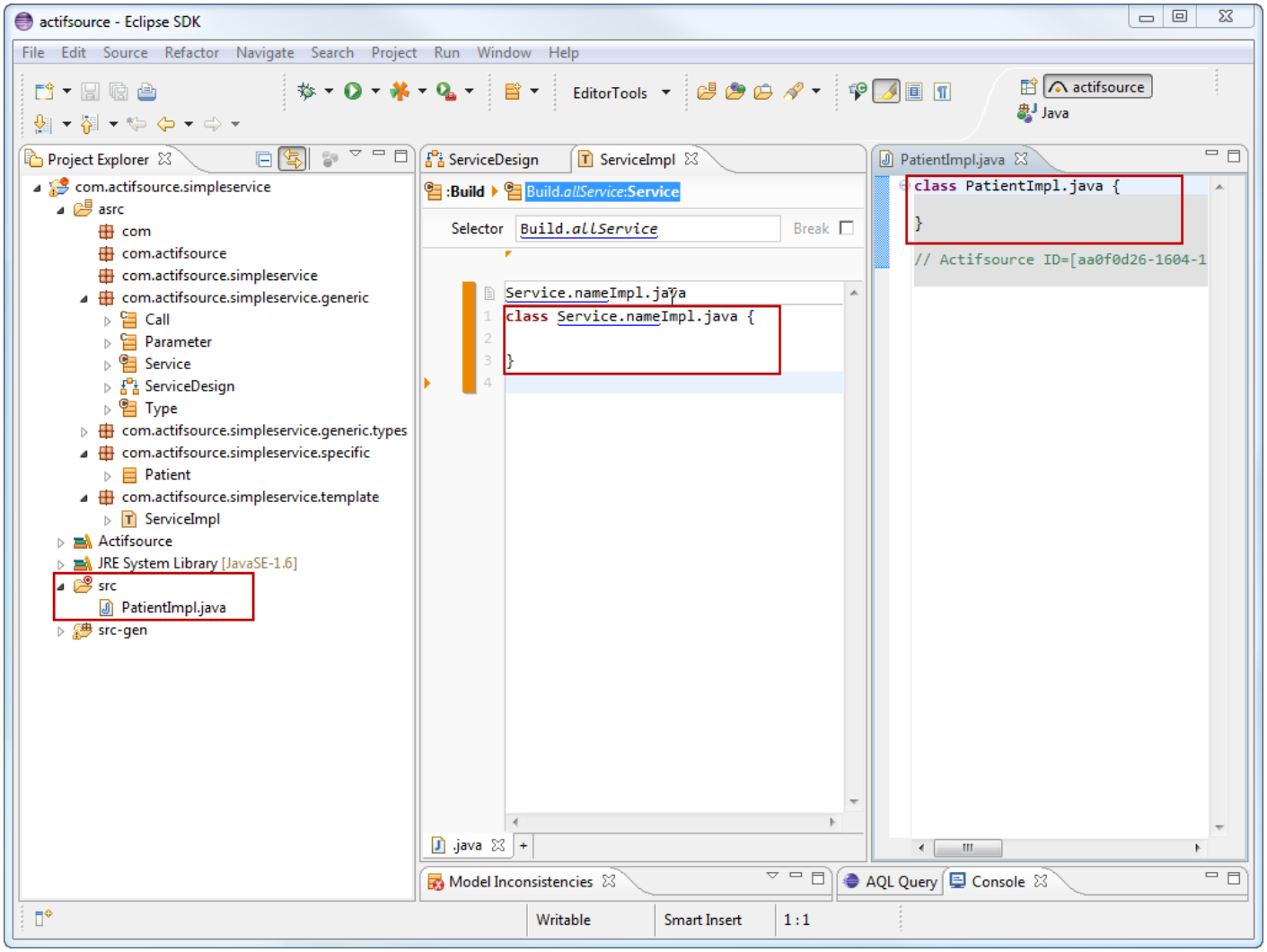


- Press '. ' (dot) after Service
- **Content Assist** opens automatically.
- Press Ctrl+Space to reopen **Content Assist** manually
- Select the **Attribut name** for Service

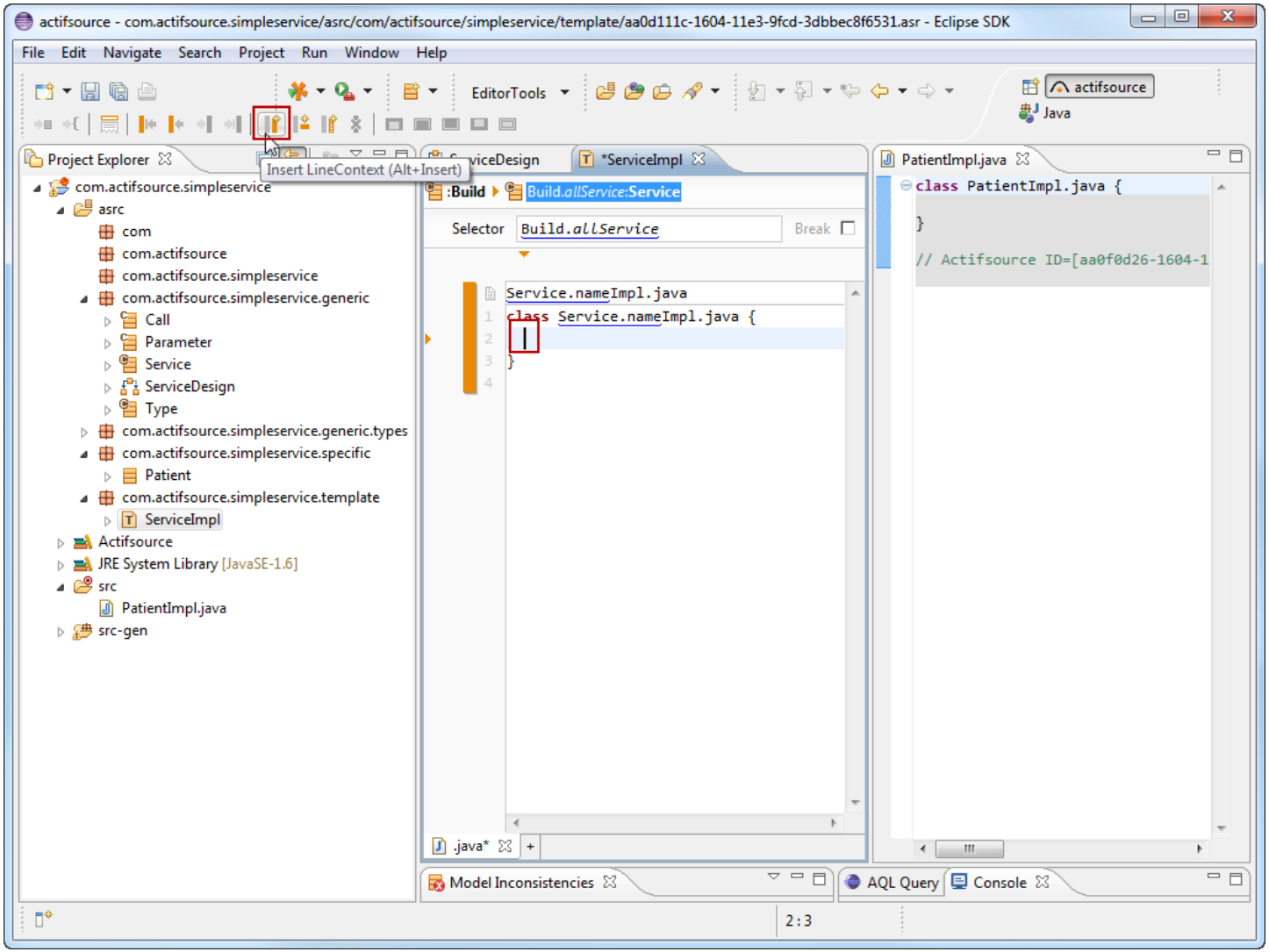
Write Generic Code Based on the Generic Domain Model



- Add the postfix `Impl` to the file name
- Add the language extension `.java` to the file name
- Save the file → a new file in your **Target Folder** src is created
- ⓘ Note that syntax highlighting is automatically activated based on the file extension



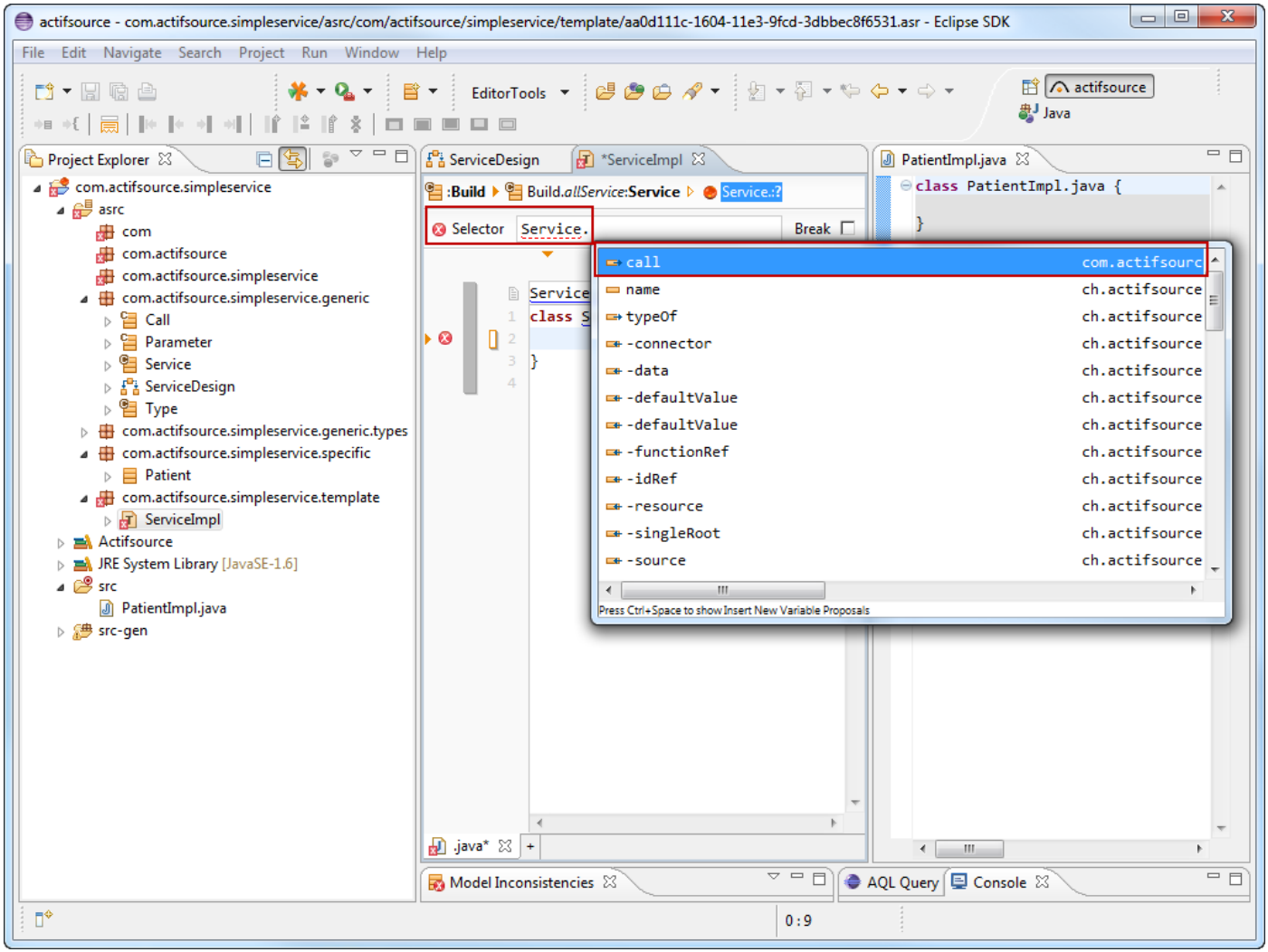
- ⓘ The orange bar on the left hand side is called **main context**
- ⓘ All text in the **main context** is generated for every specific **Resource** of type Service
- ↳ Write a Java Class as shown above. Make sure the class name contains the **Variable** Service.**name**
- ↳ Use **Content Assist** (Ctrl+Space) to insert the **Variable**
- ↳ Or use Copy/Paste (Ctrl+C/Ctrl+V) to copy the class name from file name
- ↳ Save the file (Ctrl+S) → Generated code can be found in the target folder *src*



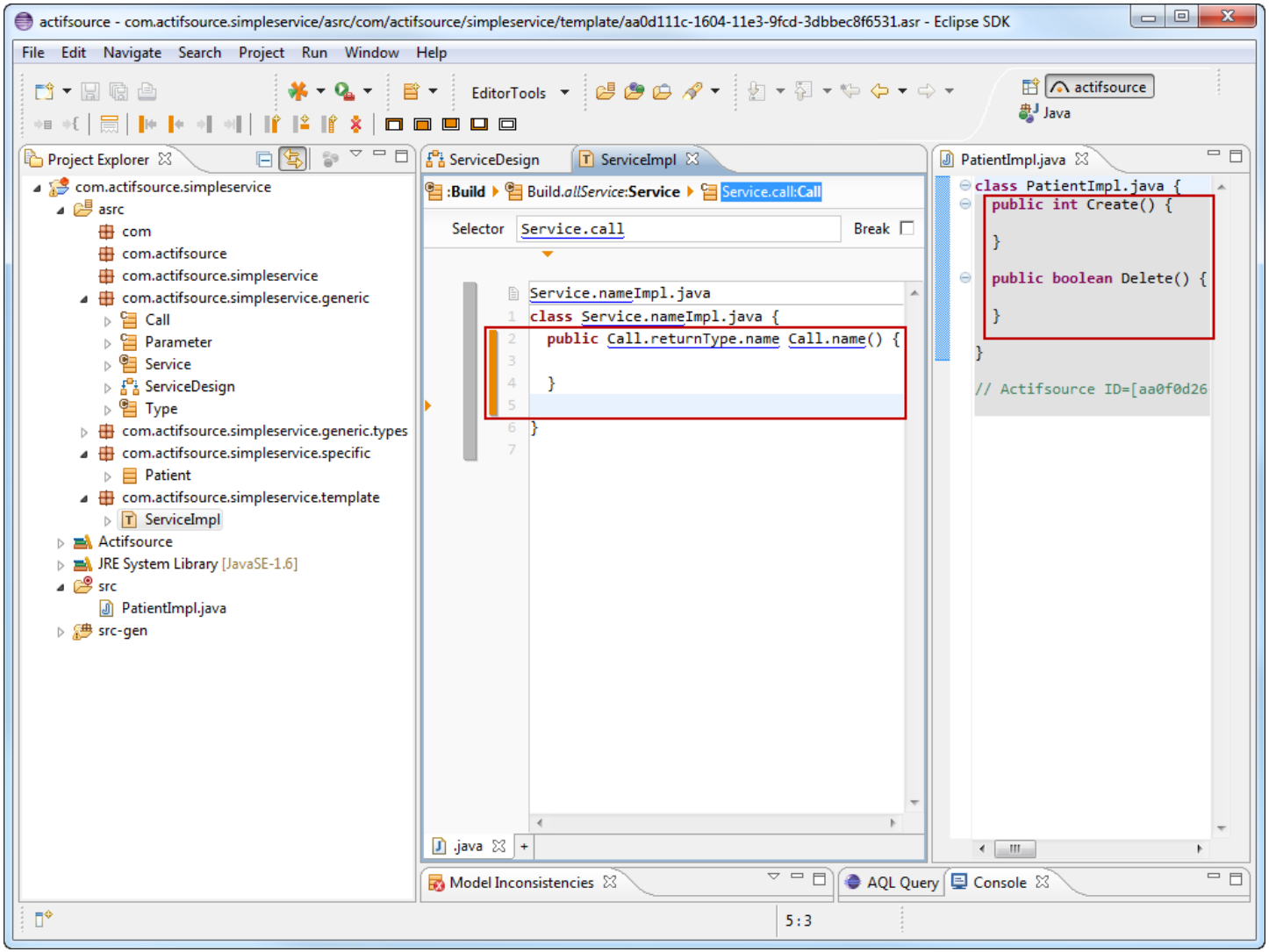
ⓘ Let's create a function for every Call in the Service

↳ Place the cursor in the class body

↳ Use the tool **Insert Line Context** (Alt+Insert) from the tool bar



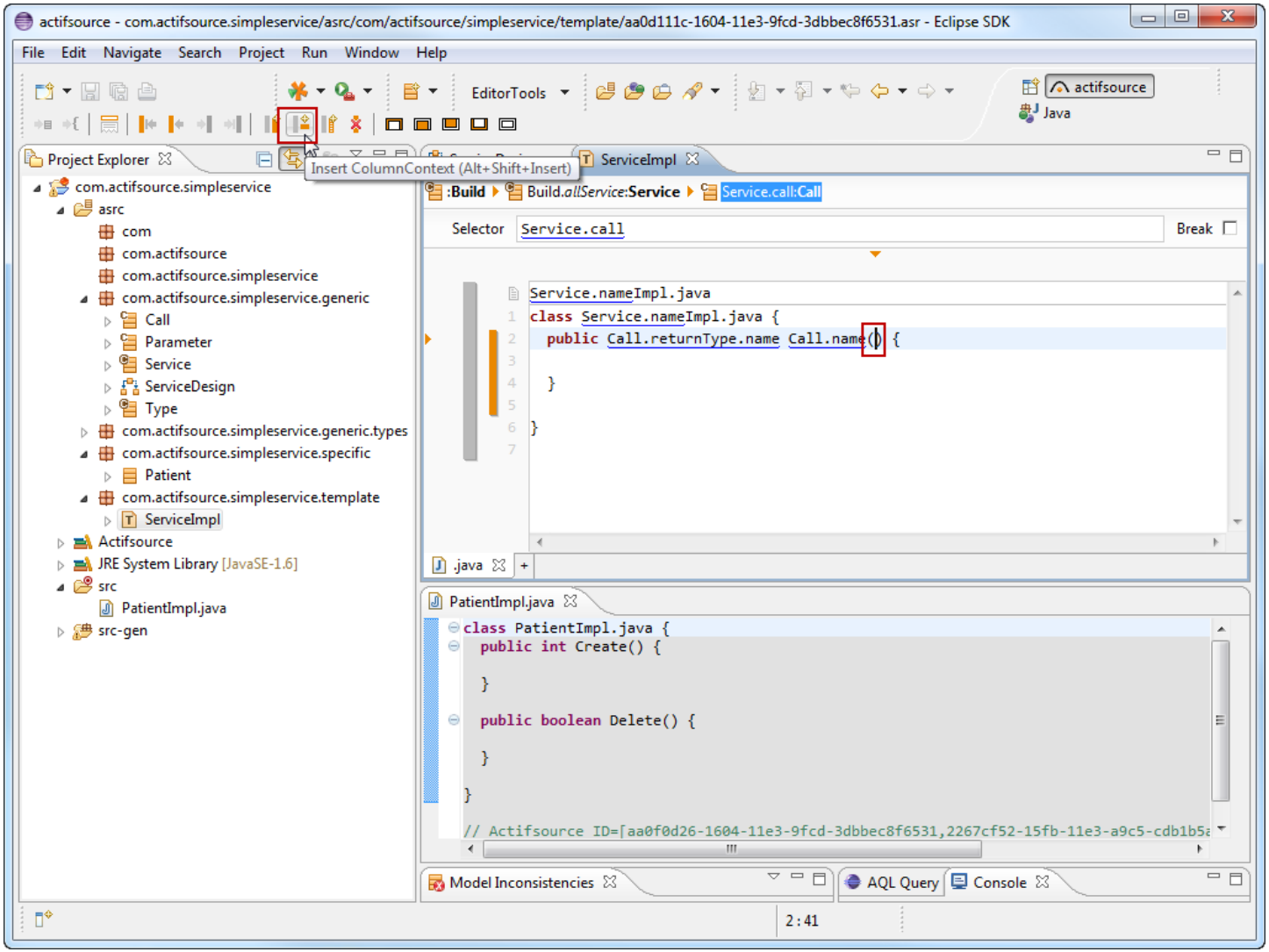
- ⓘ A new **Line Context** has been created
- ⓘ All text in the **Sub Context** shall be generated for every **Resource** of type Call in this Service
- ↳ Attach the **Sub Context** to the **Parent Context** using the **Selector**
- ↳ The given **Selector** must therefore navigate from Service to Call using Service.call
- ↳ Use **Content Assist** (Ctrl+Space) to specify the **Selector**
- ↳ Press *Enter* in the **Selector** to return to code



↳ Write a function in the Call **Context** as shown above

ⓘ The context grows as used

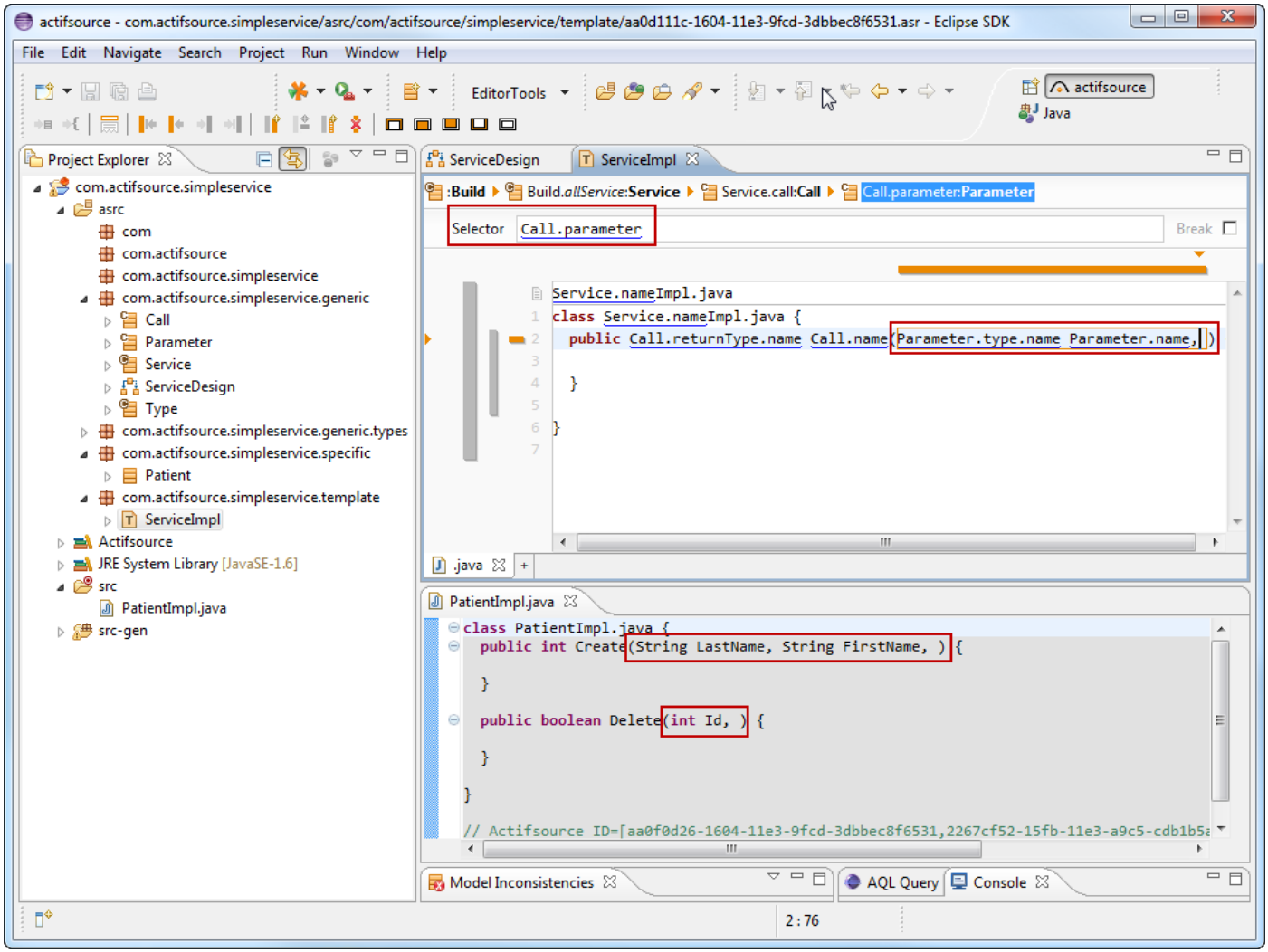
ⓘ Save the file (Ctrl+S). The generated file should contain a class and its methods. Tip: Add another Service/Call.



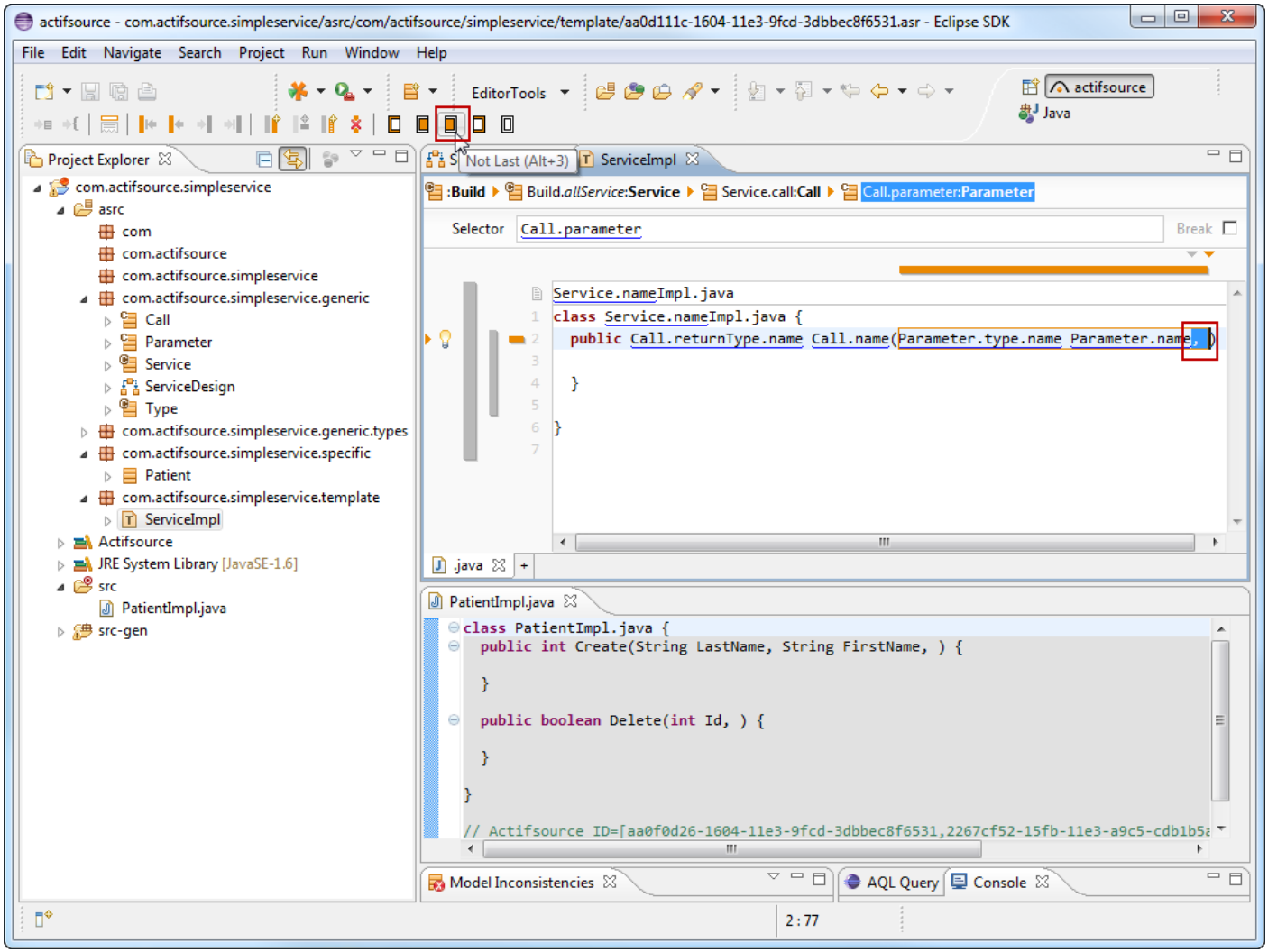
ⓘ Let's create a Java parameter for every Parameter in the Call

↳ Place cursor between the brackets

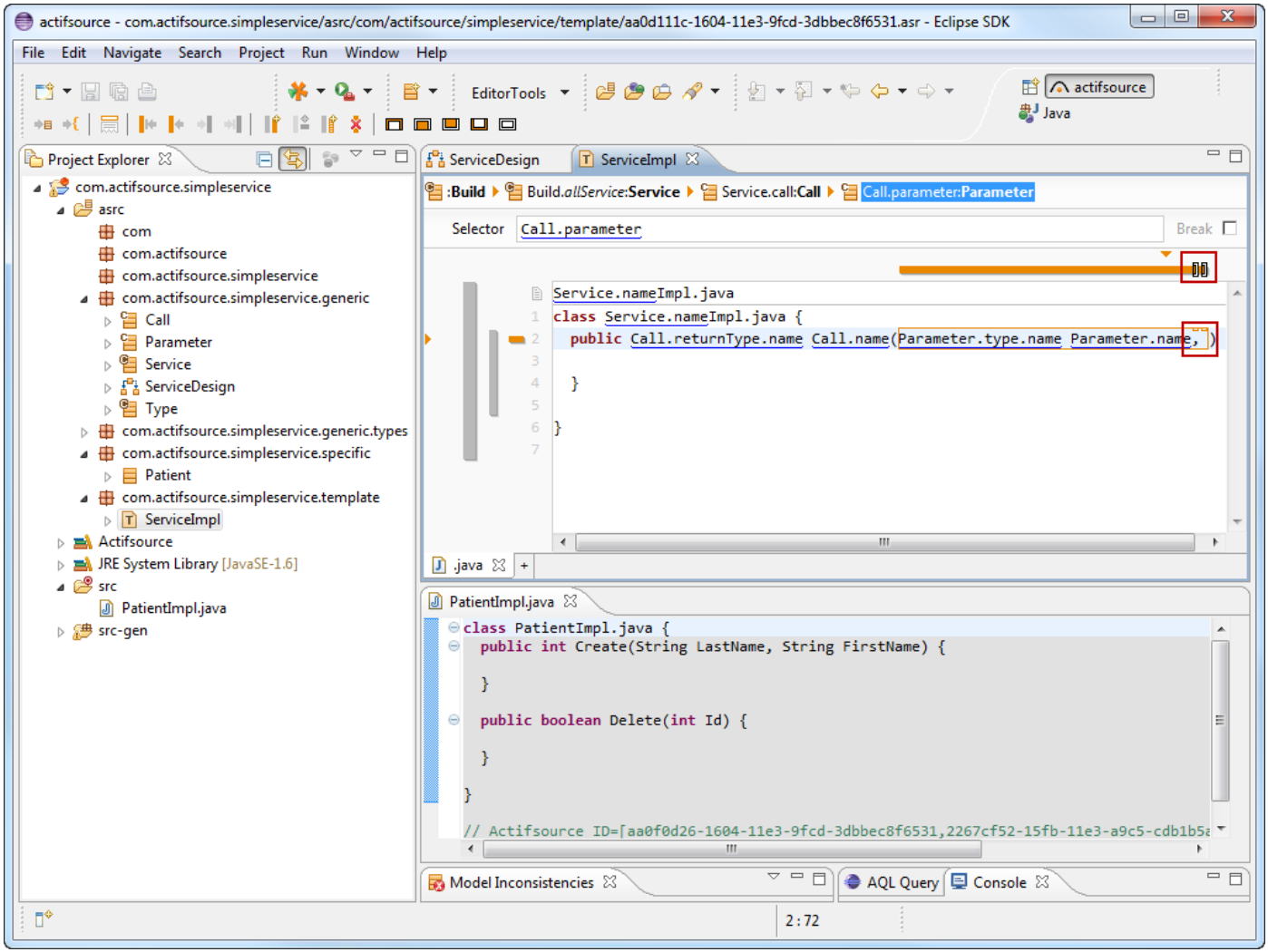
↳ Use the tool **Insert Column Context** (Alt+Shift+Insert) from the tool bar



- Write a parameter list in the Parameter **Context** as shown above
- Don't forget the ', ' (Comma) at the end
- Save the file (Ctrl+S). The generated file should contain a Java class , its methods and parameters
- ⓘ Note that all text in the Parameter **Context** shall be generated
- ⓘ For that reason, there is a superfluous comma at the end of the last Parameter



- ⓘ Single characters, but even lines, can be marked with **Attributes**
- ⓘ **First** (Alt+1) → Applied for the first iteration of the **Context**
- ⓘ **Not First** (Alt+2) → Applied for any iterations of the **Context** but the first
- ⓘ **Not Last** (Alt+3) → Applied for any iterations of the **Context** but the last
- ⓘ **Last** (Alt+4) → Applied for the first iteration of the **Context**
- ⓘ **Empty** (Alt+5) → Applied if **Context** is never iterated



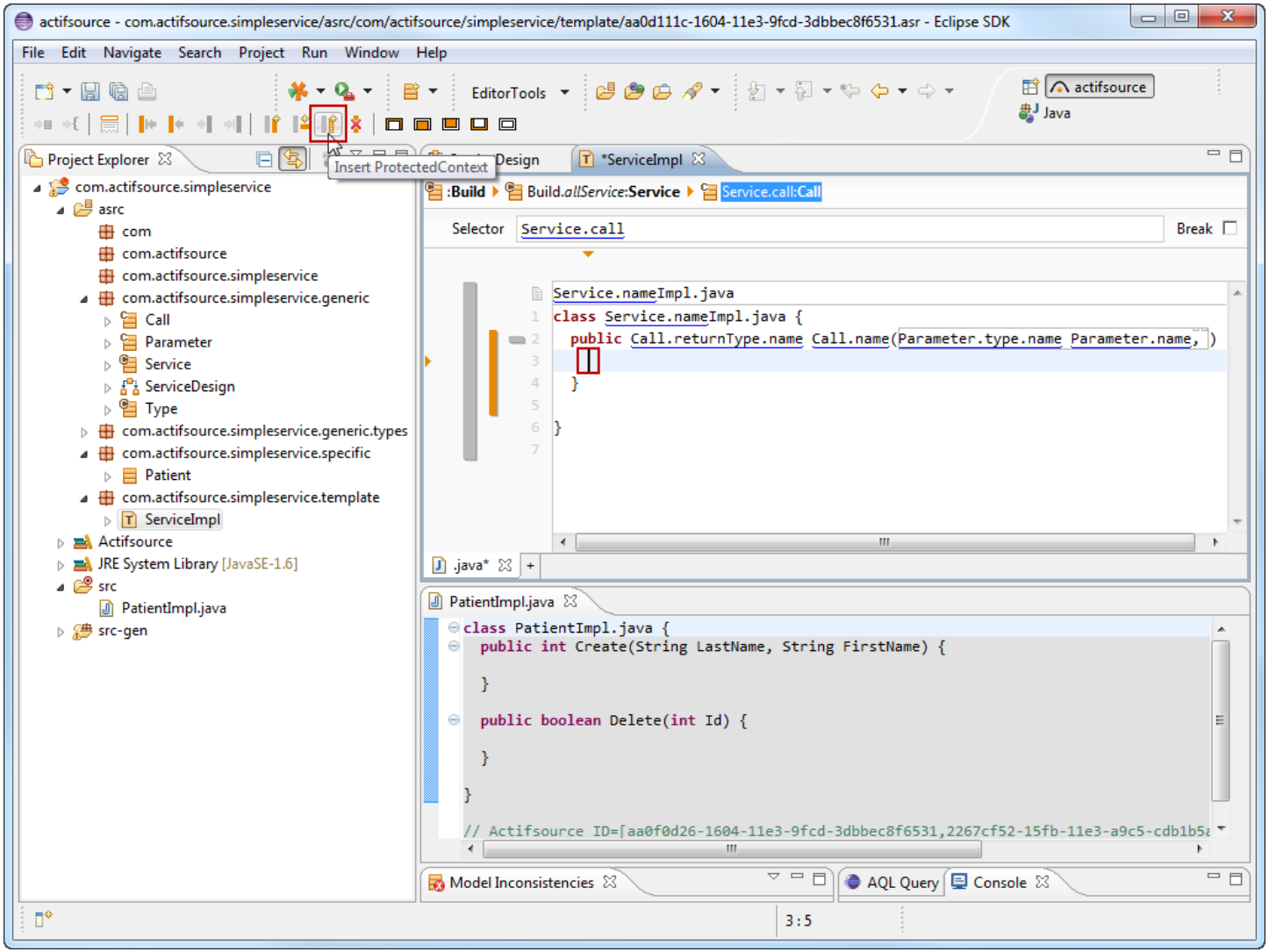
ⓘ The comma has to be written after each Parameter except *after the last one*

↳ Select both the comma and the following space

↳ Choose **Not Last** (Alt+3) from the tool bar

↳ Save the file (Ctrl+S). The generated file should contain a class , its methods and parameters

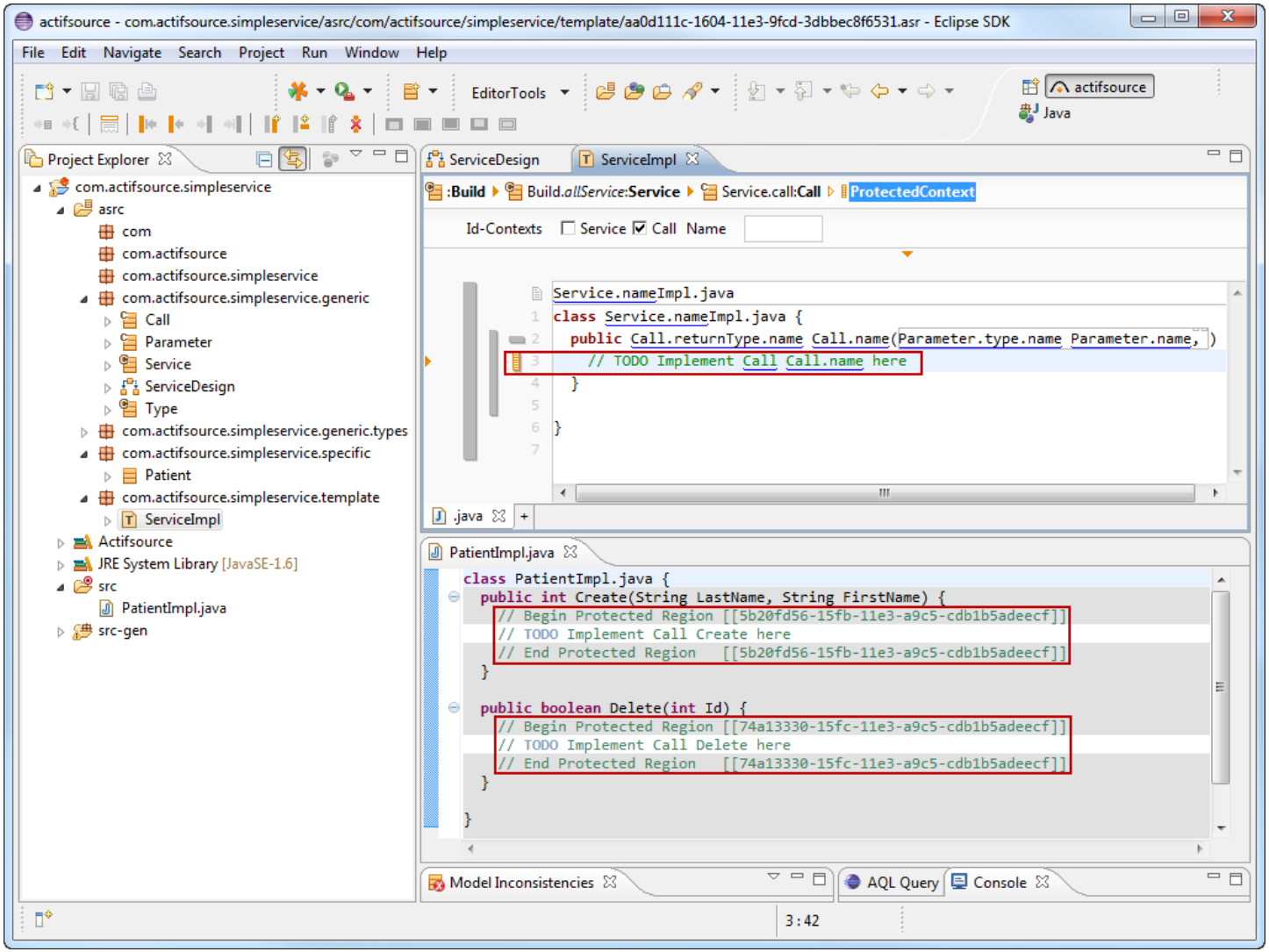
ⓘ No superfluous comma is generated this time



ⓘ Let's write specific code for the generated functions

↳ Place cursor in the function body

↳ Insert a **Protected Context** in the function body with the tool shown above



- Write any text in the **Protected Context**, e.g. a TODO comment
- Note that Call is translated in `Call`. Tip: Rename Call to ServiceCall to find out why this makes sense
- Write specific code within the *Protected Regions* of the generated files
- Warning: You must not delete the Tags **Begin Protected Region** and **End Protected Region**
- Tip: Alter the template code. Save the template. See what happens with the **Protected Regions**