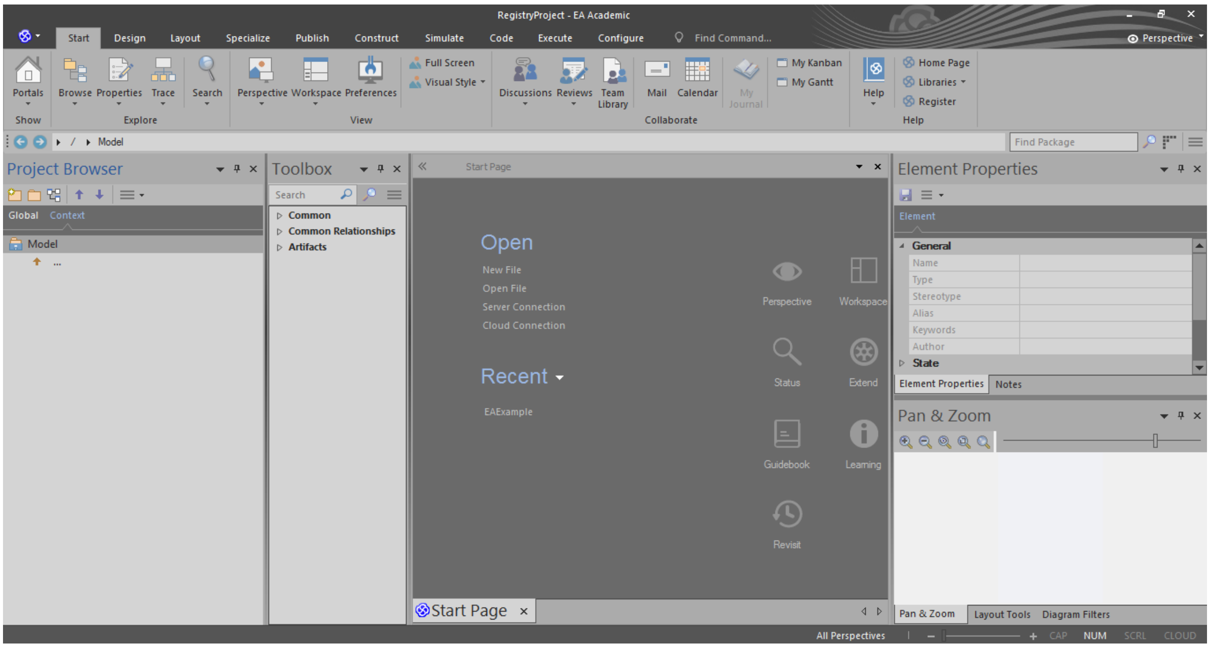
# UECS2344 Software Design: Practical 4

**You are to create an Enterprise Architect Project for the Registry App in Practical 3.**

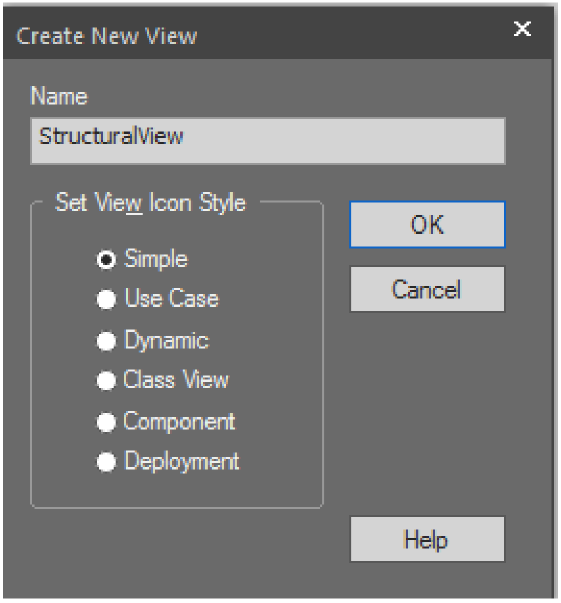
## STEP 1: CREATE NEW PROJECT

From the menu, choose New Project. In the dialog window, select Desktop and type file name 'RegistryProject'.



## STEP 2: CREATE A STRUCTURAL VIEW

In the Project Browser, select Model and click Add View… (Name: ‘StructuralView’; Style: ‘Simple’).


Create New View

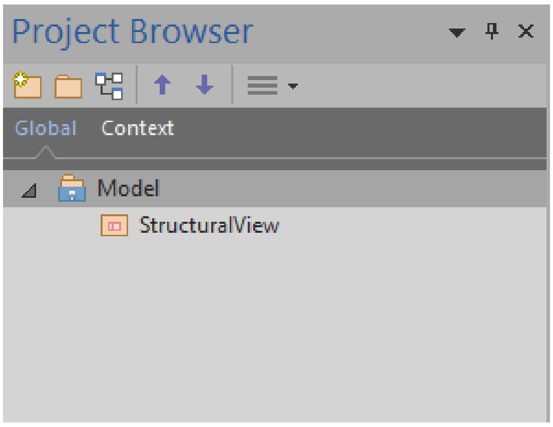
Name

StructuralView

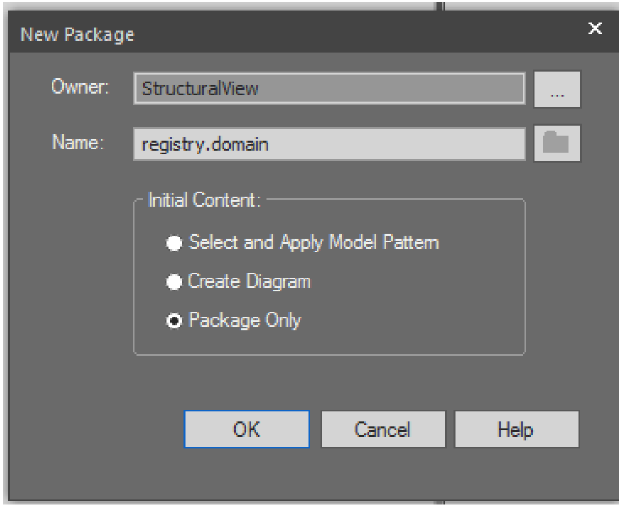
Set View Icon Style

● Simple
○ Use Case
○ Dynamic
○ Class View
○ Component
○ Deployment

A new icon is added below Model.



Right-click on StructuralView and select Add a Package (Name: ‘registry.domain’; Initial Content: ‘Package Only’).


New Package

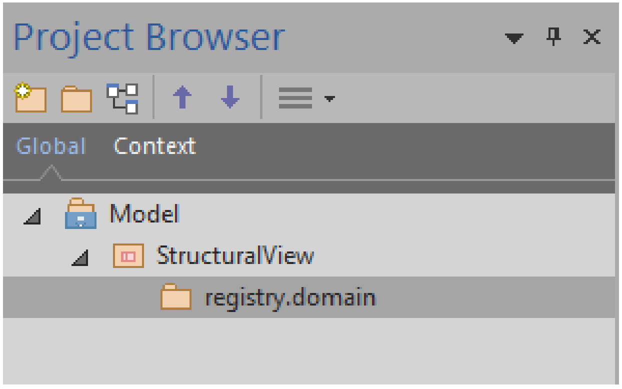
Owner: StructuralView

Name: registry.domain

Initial Content:

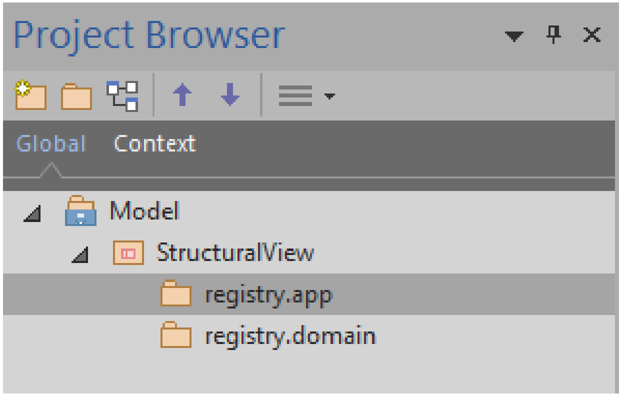
○ Select and Apply Model Pattern
○ Create Diagram
● Package Only

A new package is added under ‘StructuralView’.



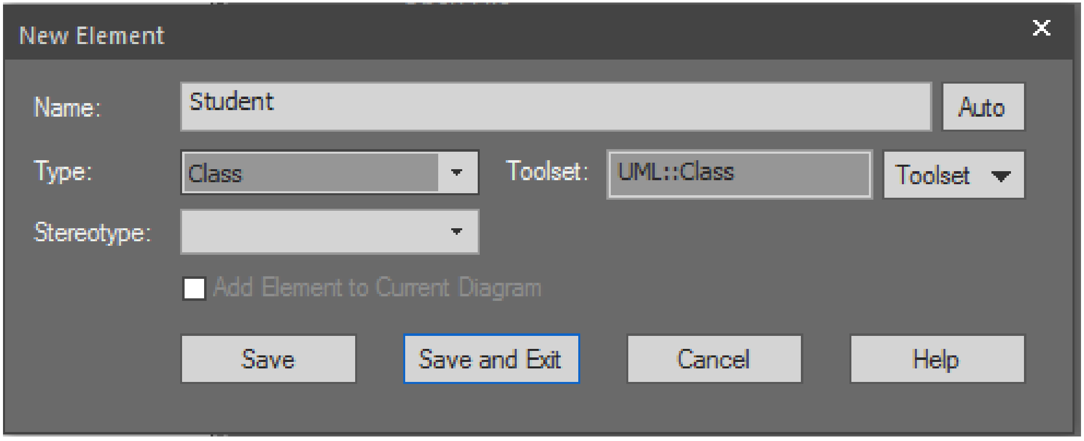
- Model
  - StructuralView
    - registry.domain

Repeat to add a second package ‘registry.app’.



- Model
  - StructuralView
    - registry.app
    - registry.domain

Right-click on registry.domain package and select Add Element (Name: ‘Student’; Type: ‘Class’).



New Element

Name: Student

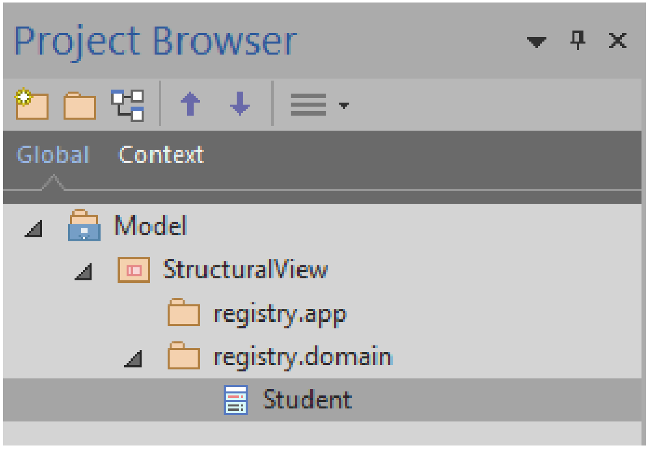
Type: Class

Toolset: UML::Class

Stereotype:

□ Add Element to Current Diagram

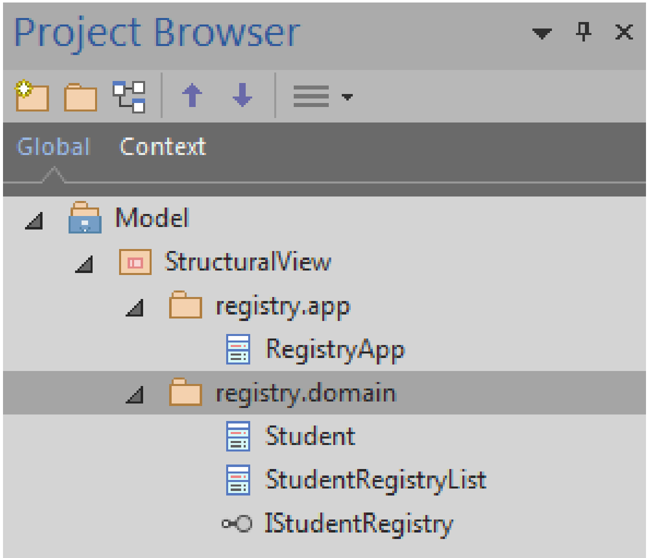
A new class Student is added under package registry.domain.



Project Browser

- Model
  - StructuralView
    - registry.app
    - registry.domain
      - Student

Repeat to add the remaining interface (IStudentRegistry) and classes (StudentRegistryList and RegistryApp) under the respective packages.

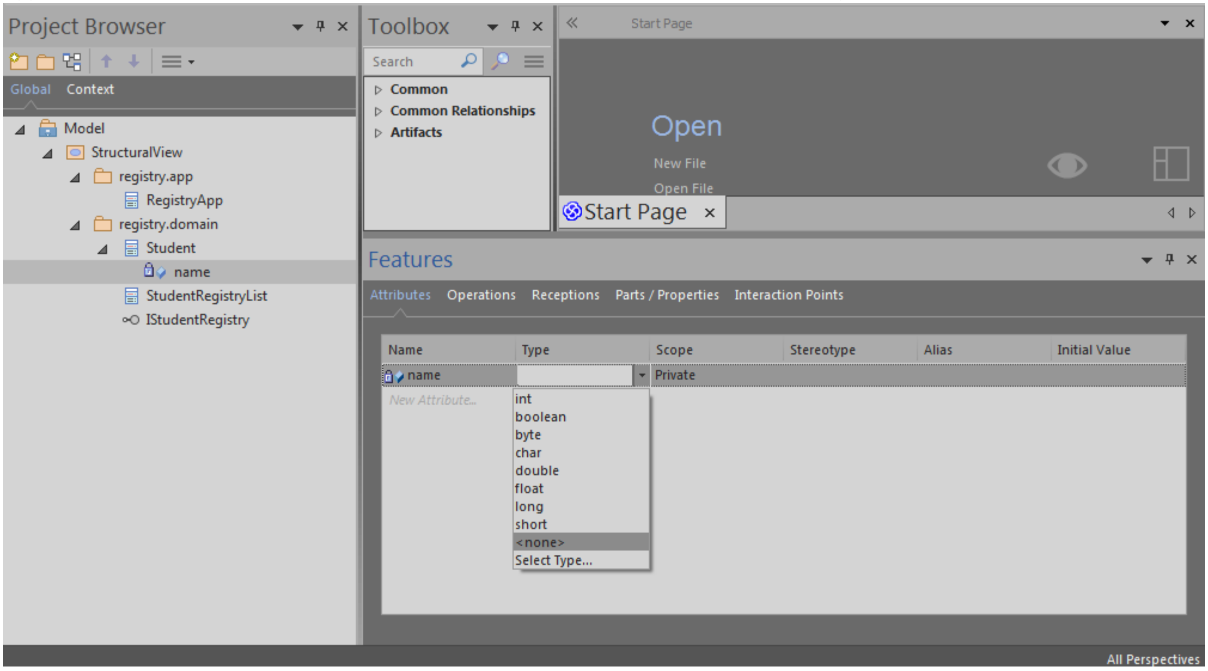


Project Browser

- Model
  - StructuralView
    - registry.app
      - RegistryApp
    - registry.domain
      - Student
      - StudentRegistryList
      - IStudentRegistry

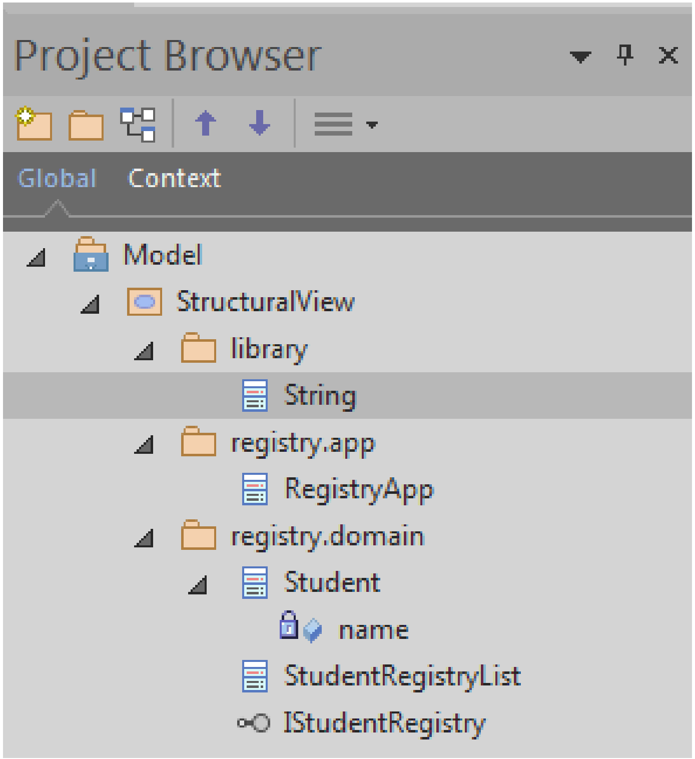
Right-click class Student and select Features & Properties -> Attributes....

Type 'name' in the Name column of the Attributes tab in the Features panel.

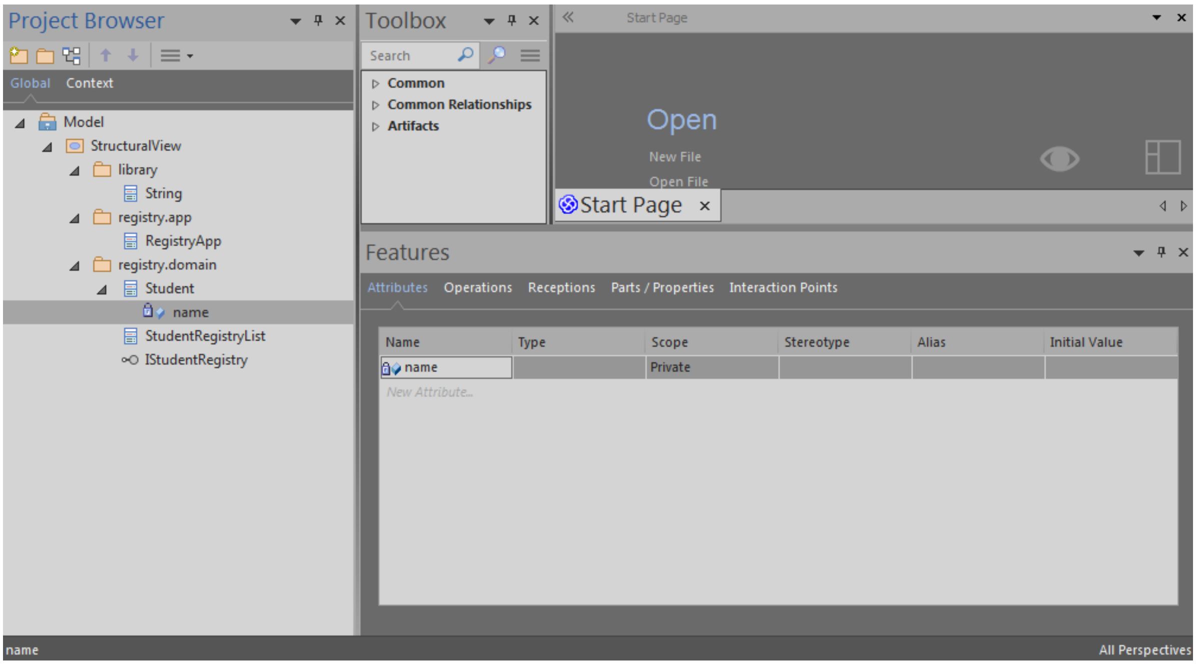


Note that for the type, there is no type String.

Add a new package (Name: 'library') under StructuralView. Add a class (Name: 'String') under package library.

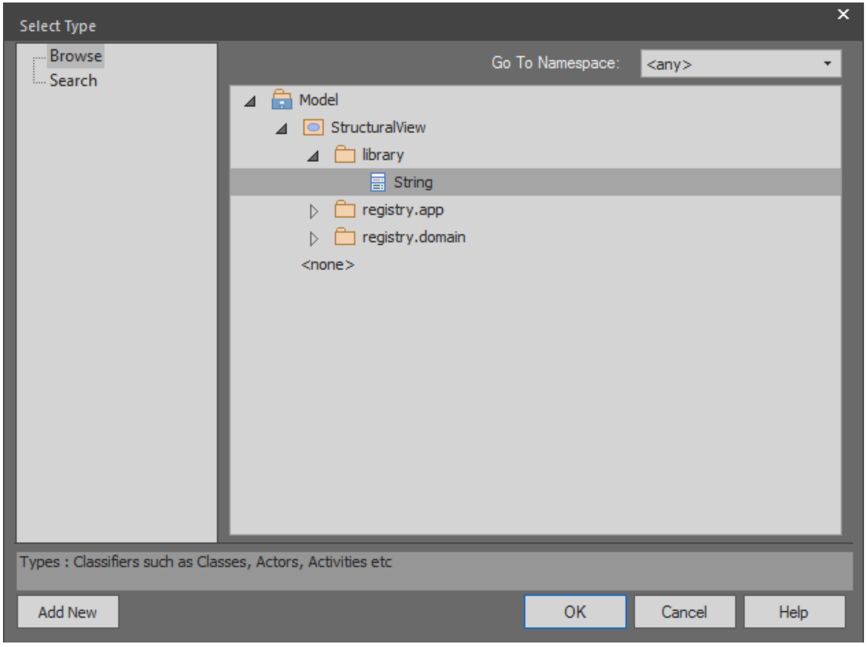


Click on attribute name under class Student.

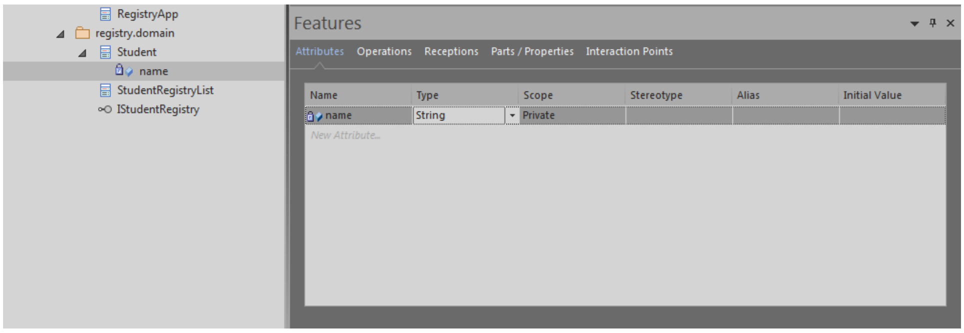


| Name | Type | Scope | Stereotype | Alias | Initial Value |
|---|---|---|---|---|---|
| name | | Private | | | |

For the type, choose ‘Select Type …’ and in the dialog window, look for and choose String under library.

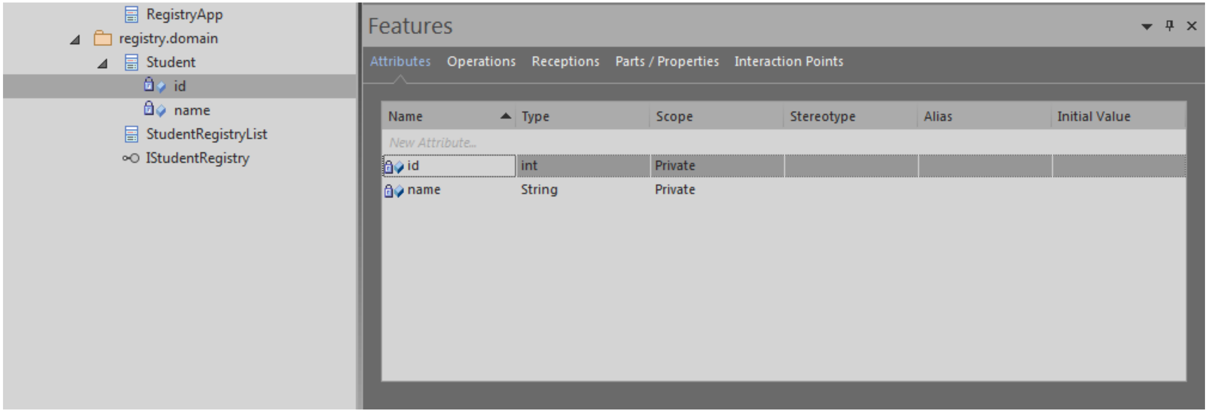


Types : Classifiers such as Classes, Actors, Activities etc



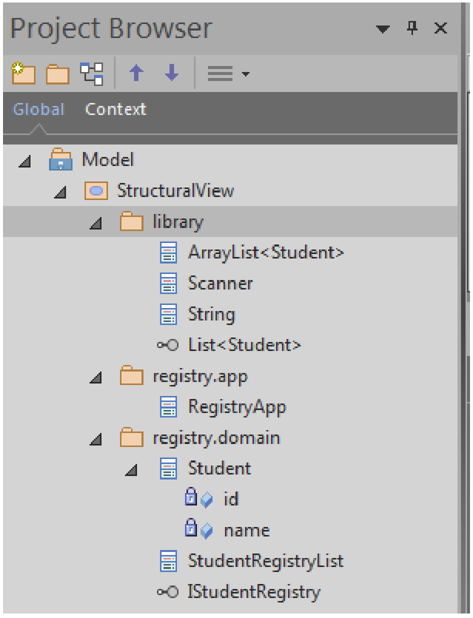
| Name | Type | Scope | Stereotype | Alias | Initial Value |
|---|---|---|---|---|---|
| name | String | Private | | | |

Add attribute id for class Student with type int.



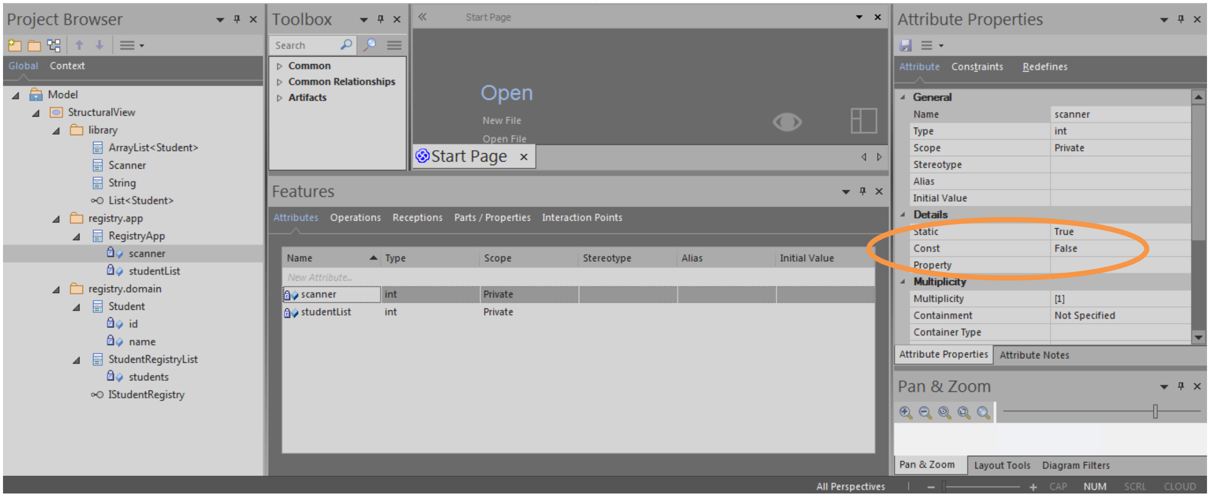
| Name | Type | Scope | Stereotype | Alias | Initial Value |
|---|---|---|---|---|---|
| New Attribute... | | | | | |
| id | int | Private | | | |
| name | String | Private | | | |

Now for class StudentRegistryList, we need an interface List<Student>.
Also we need a class Scanner for the class RegistryApp and class ArrayList<Student> for the Class Diagram.
Add these in the library package.



Add all the attributes for the classes.

For static attribute (scanner and studentList in RegistryApp), set the Attribute Property Static to True.

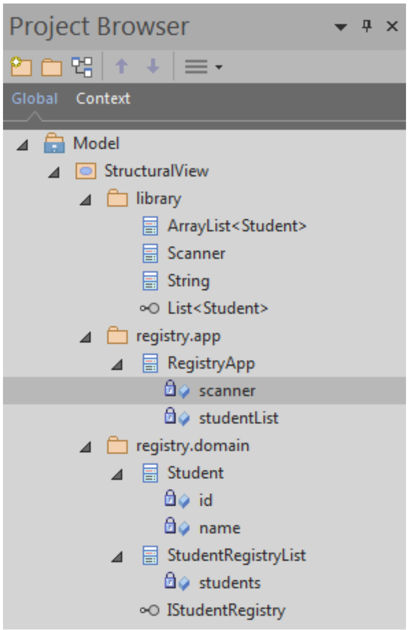


```
Model
  StructuralView
    library
      ArrayList<Student>
      Scanner
      String
      List<Student>
    registry.app
      RegistryApp
        scanner
        studentList
    registry.domain
      Student
        id
        name
      StudentRegistryList
        students
      IStudentRegistry
```

| Name | Type | Scope | Stereotype | Alias | Initial Value |
|---|---|---|---|---|---|
| *New Attribute...* | | | | | |
| scanner | int | Private | | | |
| studentList | int | Private | | | |

| | |
|---|---|
| **General** | |
| Name | scanner |
| Type | int |
| Scope | Private |
| Stereotype | |
| Alias | |
| Initial Value | |
| **Details** | |
| Static | True |
| Const | False |
| Property | |
| **Multiplicity** | |
| Multiplicity | [1] |
| Containment | Not Specified |
| Container Type | |

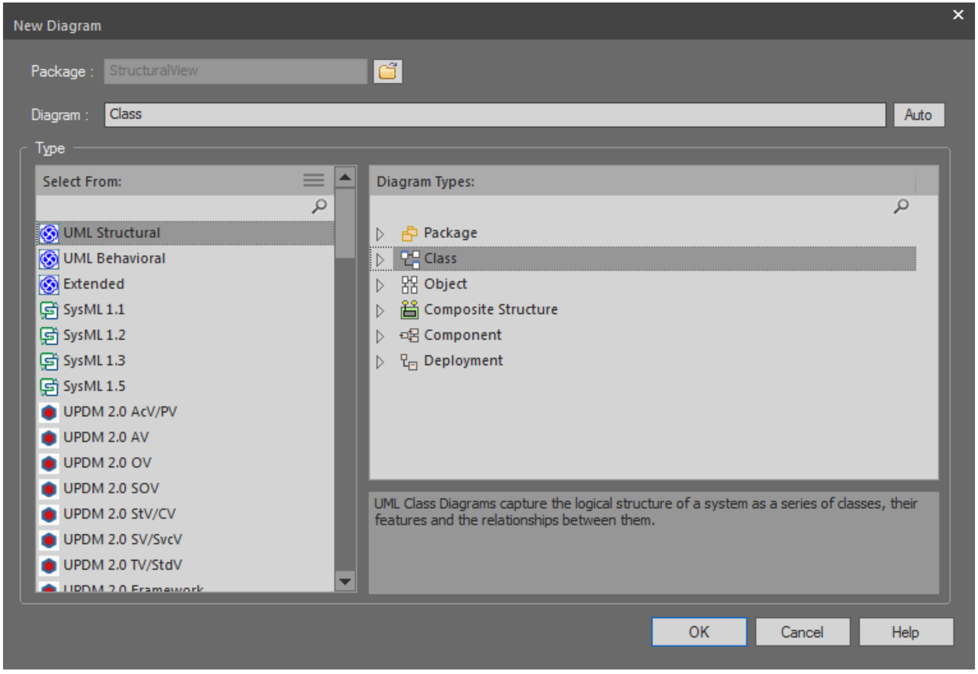
This is the final list of classes/interfaces and their attributes.



```
Model
  StructuralView
    library
      ArrayList<Student>
      Scanner
      String
      List<Student>
    registry.app
      RegistryApp
        scanner
        studentList
    registry.domain
      Student
        id
        name
      StudentRegistryList
        students
      IStudentRegistry
```

## STEP 3: CREATE A CLASS DIAGRAM

Right-click package StructuralView and select Add Diagram (Diagram: ‘Class’; Diagram Type: ‘Class’).

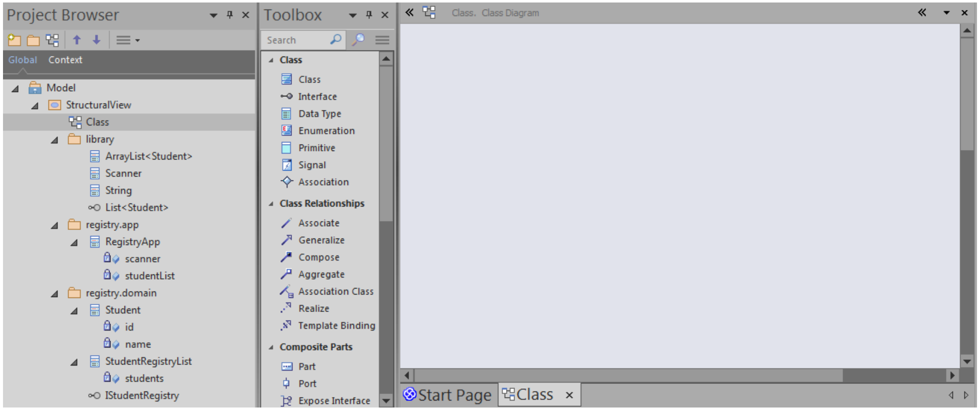


Package : StructuralView

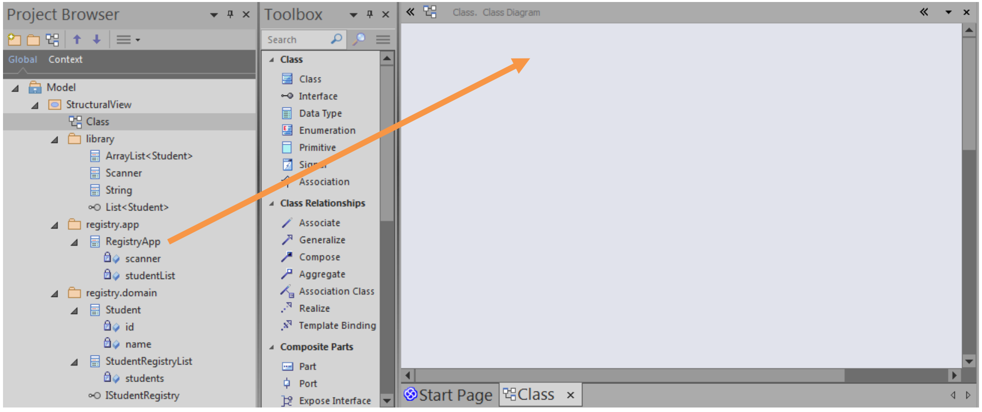
Diagram : Class

UML Class Diagrams capture the logical structure of a system as a series of classes, their features and the relationships between them.

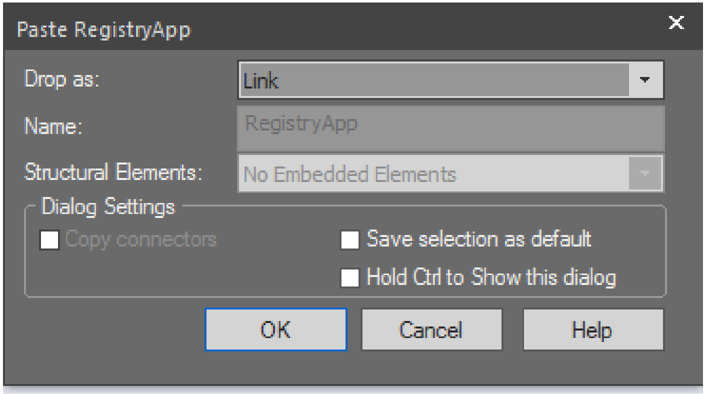
A class diagram panel appears.



Drag the class RegistryApp from the Model and drop it onto the class diagram panel.



In the dialog window that appears, choose Drop as ‘Link’.



Paste RegistryApp

Drop as: Link

Name: RegistryApp

Structural Elements: No Embedded Elements

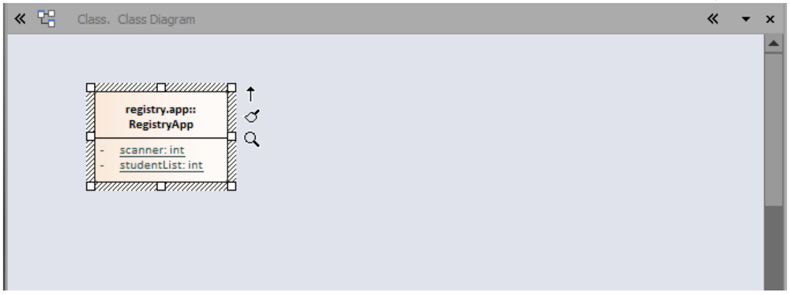
Dialog Settings

□ Copy connectors

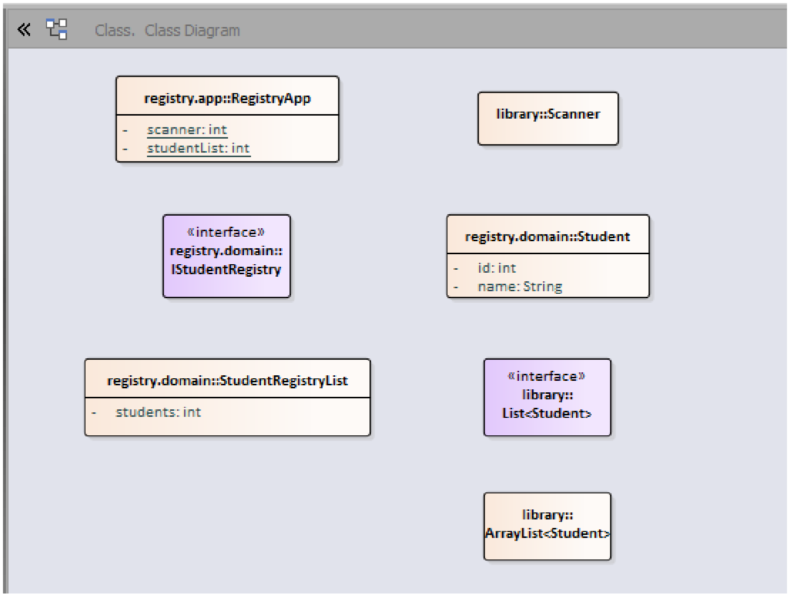
□ Save selection as default

□ Hold Ctrl to Show this dialog

The class is added to the panel.
Note that the attributes of the class are added automatically and the static attributes are underlined.



Drag and drop the remaining classes and interface and arrange them.



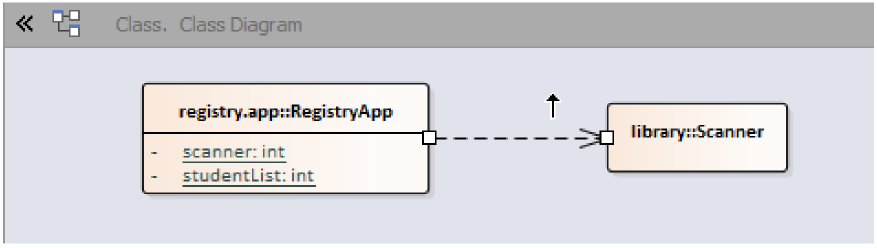
We need to add the relationships between the classes/interface.

There are 3 types of relationships involved here:

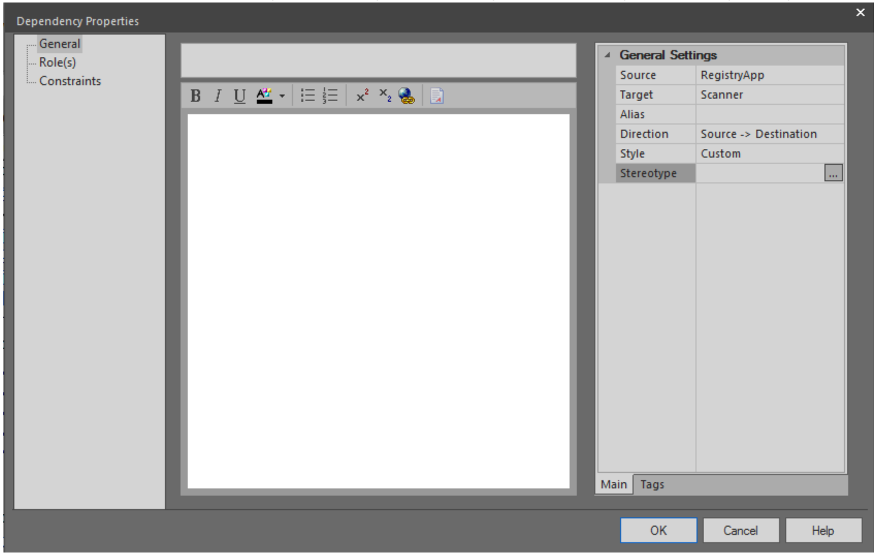
- Association (solid line)
- Dependency (dashed line with arrowhead)
- Interface Realisation (dashed line with triangle – *implements* relationship between class and interface)

To draw the relationship lines, click on one of the classes and drag the arrow symbol to the other class/interface.

For example, click on class RegistryApp and drag the arrow to class Scanner. Select 'Dependency' option.

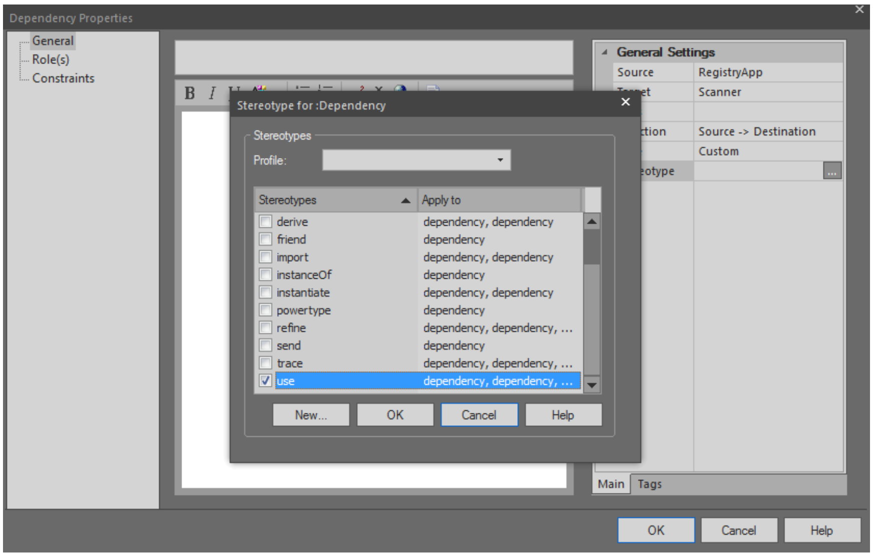


Double-click on the dependency line to open the Dependency Properties dialog window.



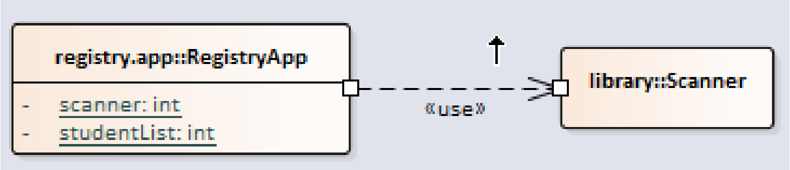
| | |
|---|---|
| Source | RegistryApp |
| Target | Scanner |
| Alias | |
| Direction | Source -> Destination |
| Style | Custom |
| Stereotype | |

Click the '...' for Stereotype to open the Stereotype for: Dependency dialog window. Select Stereotype 'use'.

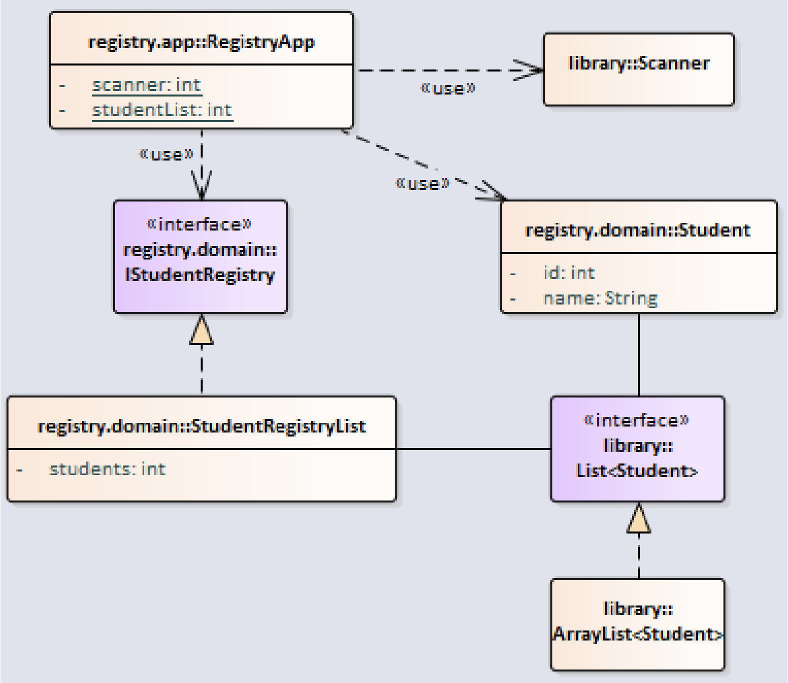


| Stereotypes | Apply to |
|---|---|
| □ derive | dependency, dependency |
| □ friend | dependency |
| □ import | dependency, dependency |
| □ instanceOf | dependency |
| □ instantiate | dependency, dependency |
| □ powertype | dependency |
| □ refine | dependency, dependency, ... |
| □ send | dependency |
| □ trace | dependency, dependency, ... |
| ☑ use | dependency, dependency, ... |

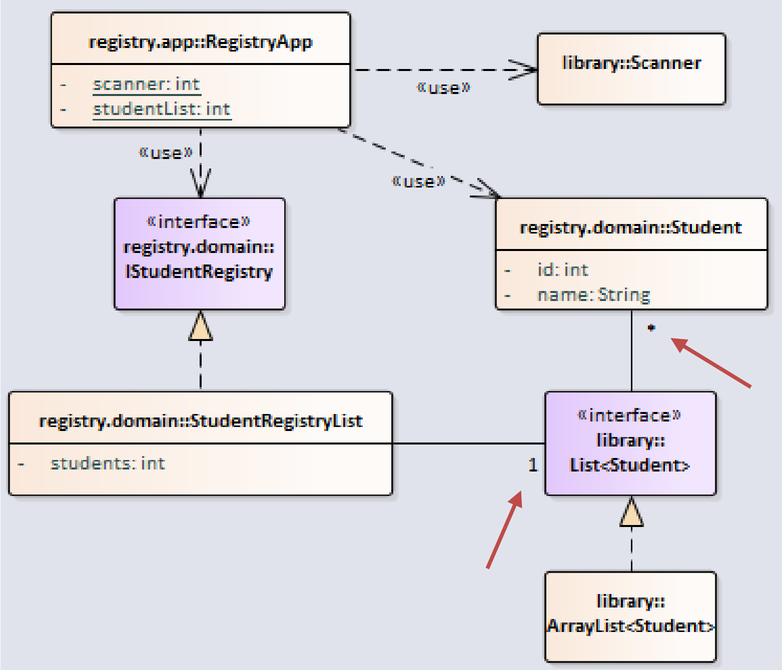
The stereotype is added to the line.



Create all the relationships (3 dependencies, 2 realisations, and 2 associations)

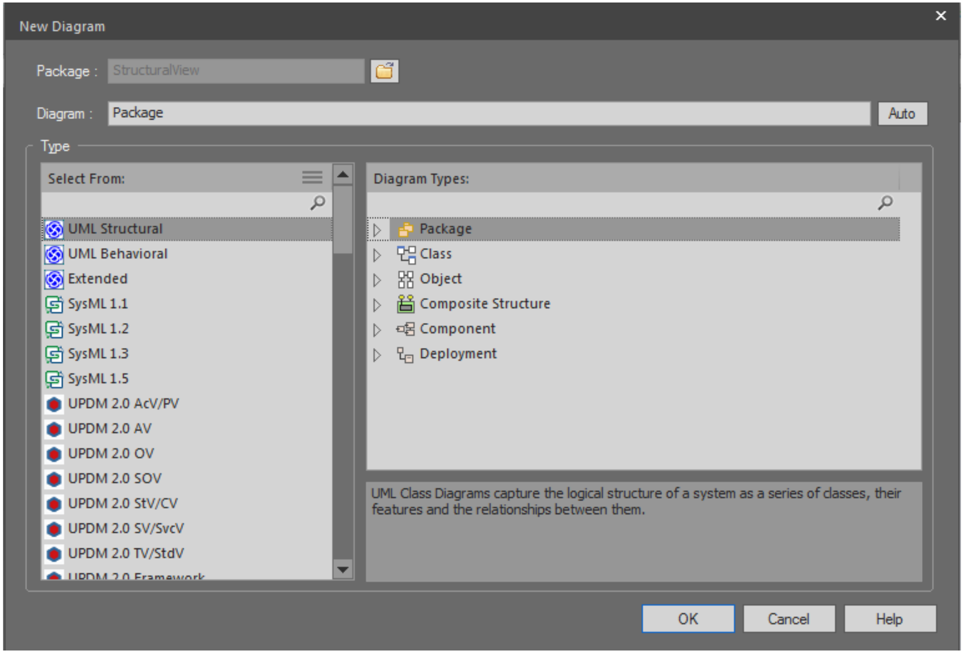


Add multiplicity for the associations by right-clicking on the side of the line to add and select Multiplicity.



## STEP 4: CREATE A PACKAGE DIAGRAM

Right-click package StructuralView and select Add Diagram (Diagram: ‘Package’; Diagram Type: ‘Package’).


New Diagram

Package : StructuralView

Diagram : Package

Type

Select From:

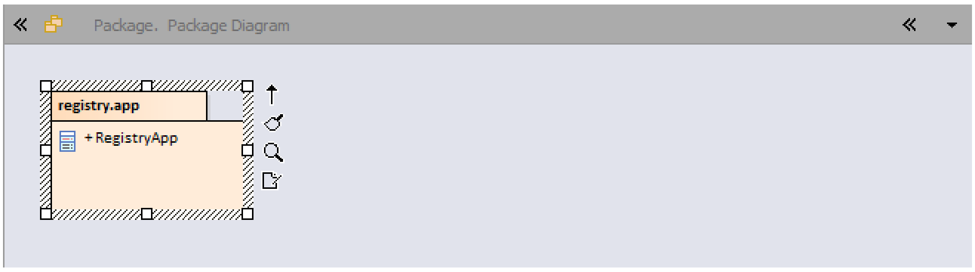
- UML Structural
- UML Behavioral
- Extended
- SysML 1.1
- SysML 1.2
- SysML 1.3
- SysML 1.5
- UPDM 2.0 AcV/PV
- UPDM 2.0 AV
- UPDM 2.0 OV
- UPDM 2.0 SOV
- UPDM 2.0 StV/CV
- UPDM 2.0 SV/SvcV
- UPDM 2.0 TV/StdV
- UPDM 2.0 Framework

Diagram Types:

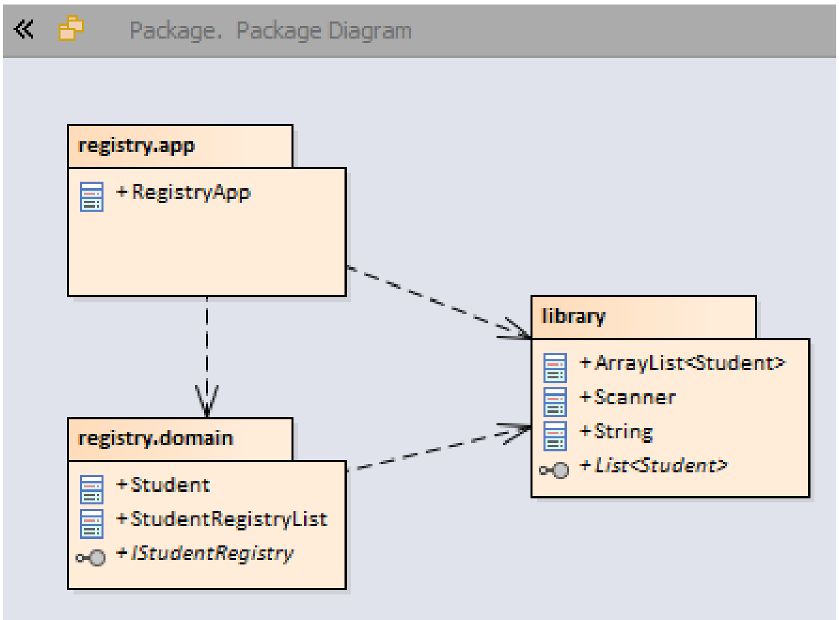
- Package
- Class
- Object
- Composite Structure
- Component
- Deployment

UML Class Diagrams capture the logical structure of a system as a series of classes, their features and the relationships between them.

Drag package registry.app from Model and drop into package diagram panel. Select ‘package element’ option.



Add the other packages and the Dependency Relationships.



## STEP 5: ADD OPERATIONS TO CLASSES

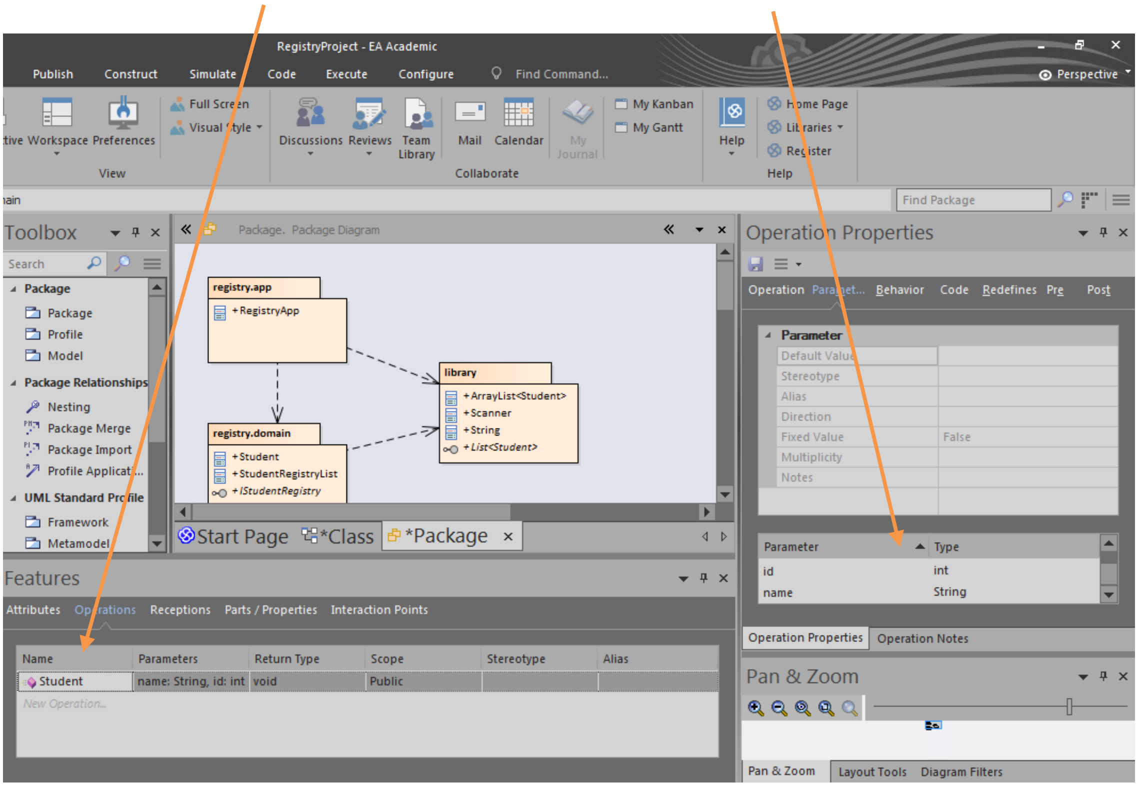
Rightclick Student class in the Model. Select Features & Properties -> Operations ....

Type in the operation name in the Name column of the Operations tab in the Features panel.

In the Operation Properties panel, type the name and type of the parameters.

Type ‘Student’

Type ‘name’: String and ‘id’: int

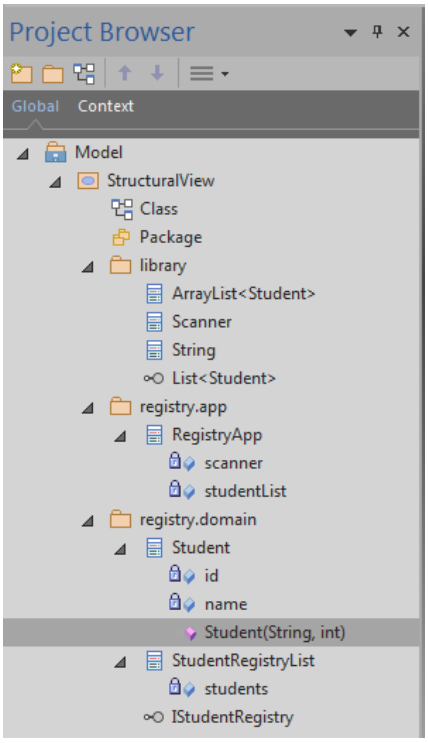


| Name | Parameters | Return Type | Scope | Stereotype | Alias |
|---|---|---|---|---|---|
| Student | name: String, id: int | void | Public | | |

| Parameter | Type |
|---|---|
| id | int |
| name | String |

Fixed Value: False

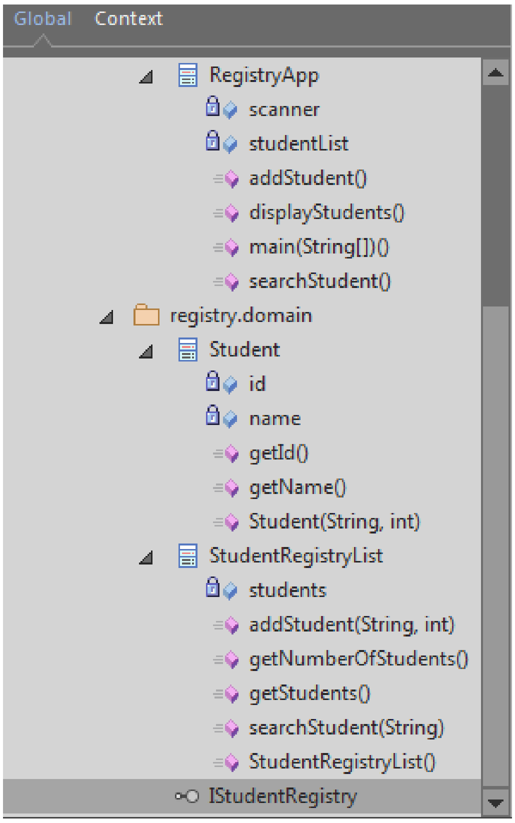
The operation is listed under the Student class in the Model.



- Model
  - StructuralView
    - Class
    - Package
    - library
      - ArrayList<Student>
      - Scanner
      - String
      - List<Student>
    - registry.app
      - RegistryApp
        - scanner
        - studentList
    - registry.domain
      - Student
        - id
        - name
        - Student(String, int)
      - StudentRegistryList
        - students
      - IStudentRegistry

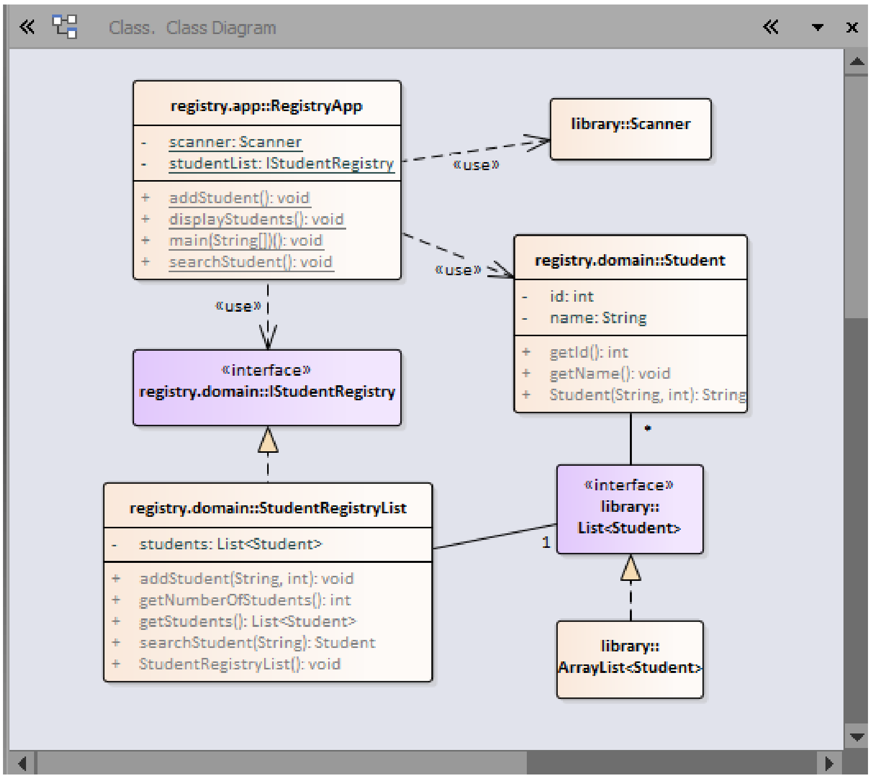
Add the remaining operations for all the classes.

Remember to set the Static property of the methods in RegistryApp to True.



- RegistryApp
  - scanner
  - studentList
  - addStudent()
  - displayStudents()
  - main(String[])()
  - searchStudent()
- registry.domain
  - Student
    - id
    - name
    - getId()
    - getName()
    - Student(String, int)
  - StudentRegistryList
    - students
    - addStudent(String, int)
    - getNumberOfStudents()
    - getStudents()
    - searchStudent(String)
    - StudentRegistryList()
  - IStudentRegistry

Look at the Class Diagram you created earlier. Take note that the operations have been added automatically for the classes.


Class. Class Diagram
registry.app::RegistryApp
- scanner: Scanner
- studentList: IStudentRegistry
+ addStudent(): void
+ displayStudents(): void
+ main(String[])(): void
+ searchStudent(): void
library::Scanner
«use»
«use»
«use»
registry.domain::Student
- id: int
- name: String
+ getId(): int
+ getName(): void
+ Student(String, int): String
«interface»
registry.domain::IStudentRegistry
*
«interface»
library::
List<Student>
1
registry.domain::StudentRegistryList
- students: List<Student>
+ addStudent(String, int): void
+ getNumberOfStudents(): int
+ getStudents(): List<Student>
+ searchStudent(String): Student
+ StudentRegistryList(): void
library::
ArrayList<Student>