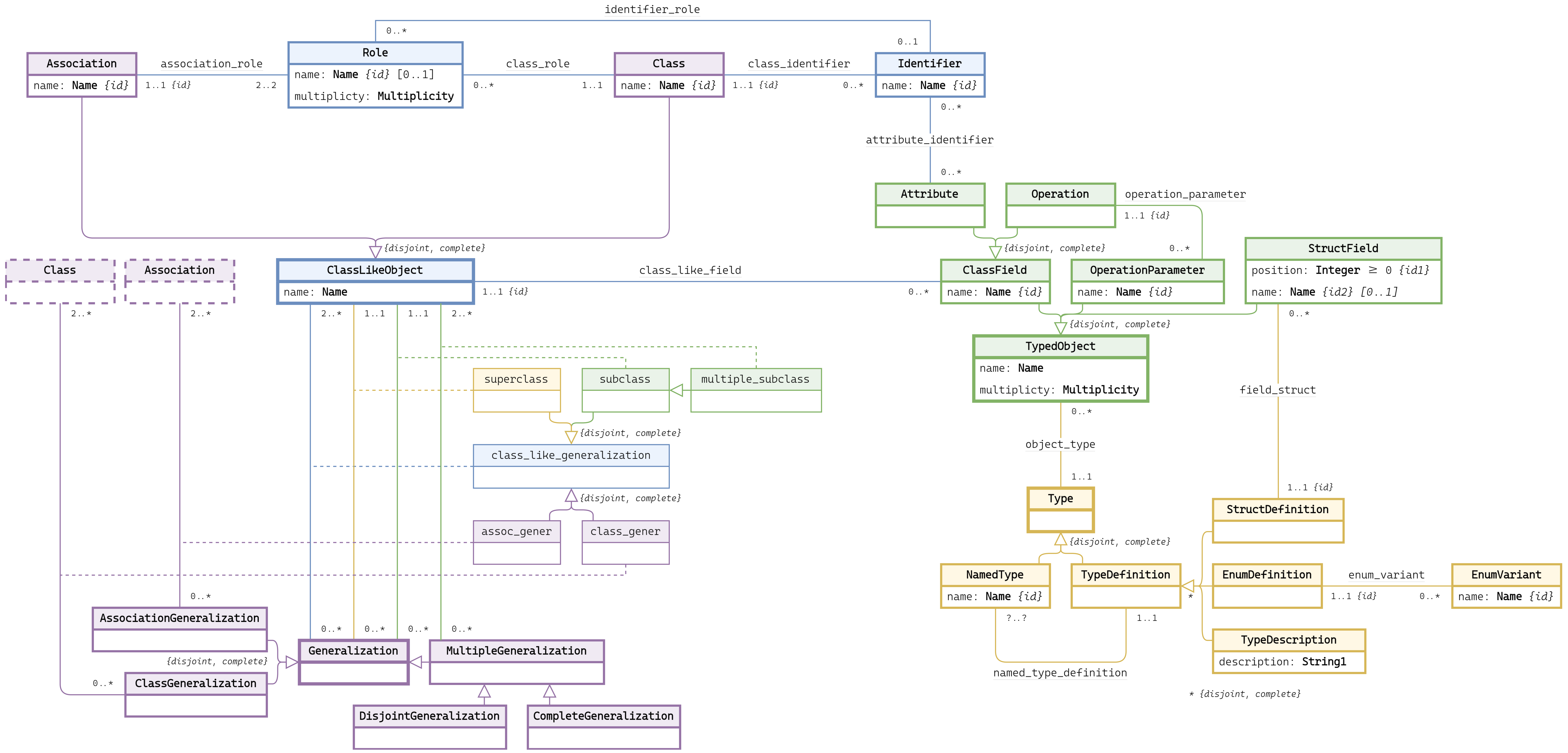
Logic Flows

First Order Logic based domain design

Contents

1 Introduction

2 UML class diagram



3 Data types specification

Name = String matching `/^[A-Za-z][A-Za-z\d]*$/`
Multiplicity = (min: Integer ≥ 0, max: Integer ≥ 0 [0..1])

3.1 Multiplicity

C.Multiplicity.min_less_than_max

```
∀ multiplicity, mult_min, mult_max
    (
        Multiplicity(multiplicity) ∧
        min(multiplicity, mult_min) ∧
        max(multiplicity, mult_max)
    ) →
        mult_min ≤ mult_max
```

4 Classes specification

4.1 Association

If both roles of an association are connected to the same class, then these roles must have names, and their names must be different.

C.Association.same_class_association_mandatory_and_different_role_names

```
∀ association, class, role1, role2
    (
        association_role(association, role1) ∧
        association_role(association, role2) ∧
        class_role(class, role1) ∧
        class_role(class, role2)
    ) →
        ∃ name1, name2
            name(role1, name1) ∧
            name(role2, name2) ∧
            name1 ≠ name2
```

4.2 Attribute

Normally the cycle in the diagram could be removed, but identifiers are required to be unique in each class.

C.Attribute.identifier_in_class

```
∀ class, attribute, identifier
    (
        attribute_identifier(attribute, identifier)
        class_like_field(class, attribute)
    ) →
        class_identifier(class, identifier)
```

4.3 Role

C.Role.identifier_in_class

```
∀ class, role, identifier
    (
        identifier_role(identifier, identifier)
        class_role(class, role)
    ) →
        class_identifier(class, identifier)
```

5 Design

Temporary description fo the wireframe

5.1 Types

On the right is a menu for defining types. It contains both basic types (String, Integer, etc.) and custom types derived from them. This section serves two purposes:

- when creating an attribute its type should be chosen from a drop-down menu
- when exporting the diagram as `JSON` types need to be well-defined in order to process the information better *(i.e. generate first order logic predicates, LaTeX macros or Typst functions)*

5.2 Components

At the top there is a bar with the components you can add, respectively: classes, associations, generalizations and notes. Just select the component and add it to the diagram.

5.3 Component-specific interactions

On the left, when a component is selected, a menu specific to that component appears. In this menu, you can do two things:

- change the component's style
- interact with the component

The interaction could consist in

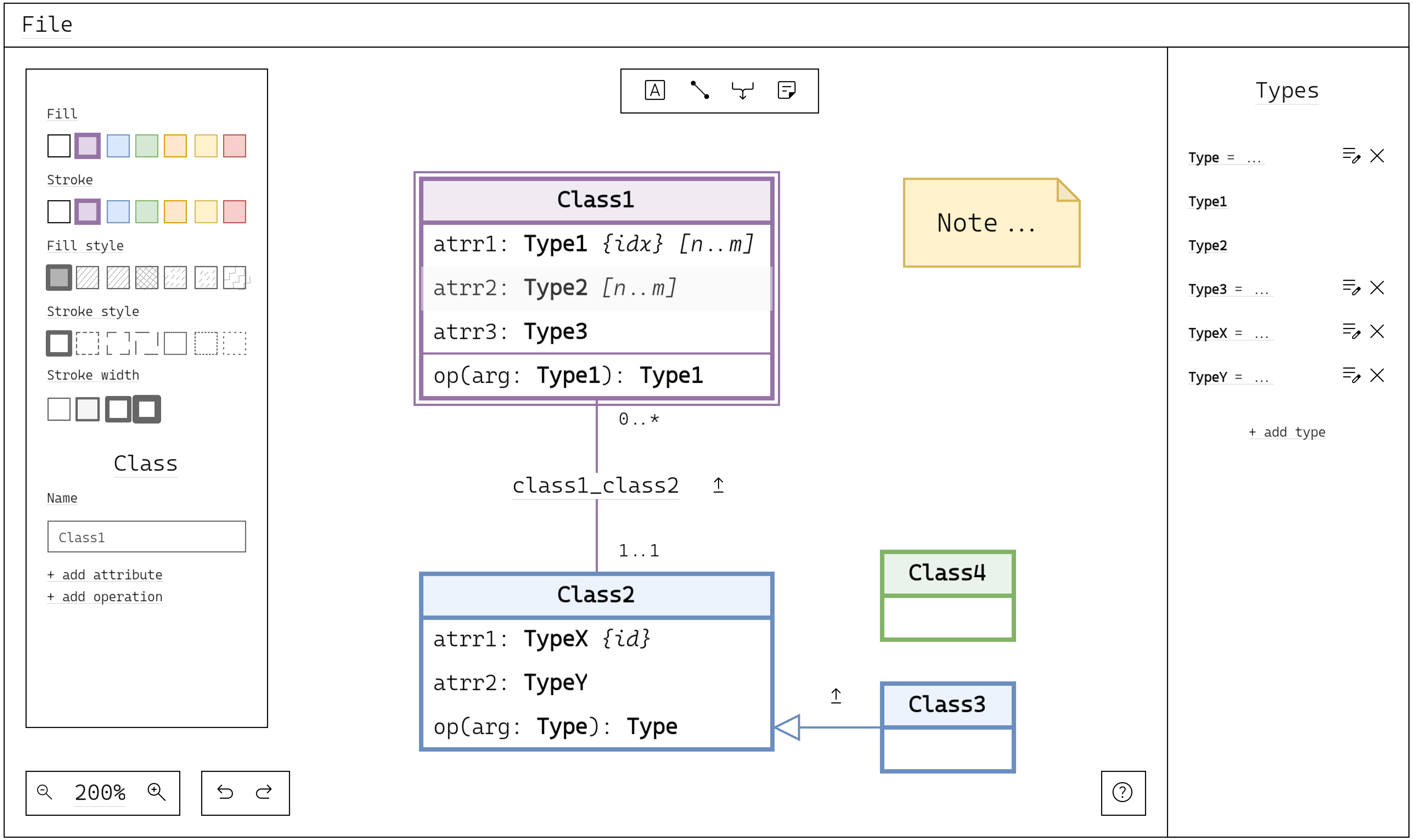
- changing some information *(class/association name, role multiplicity, attribute information)*
- a general action *(add an attribute to the class/association class or make the generalization {disjoint, complete}, etc.)*

For example, if someone selects an attribute, the editable information for the attribute appears:

- a **text field** to edit its name
- a **drop-down menu** to select its type
- a **checkbox** to decide whether it has an id
- a **numeric field** with the minimum multiplicity (an integer from 0 and up)
- a **numeric field** with the maximum multiplicity (an integer greater than "min," or just **n**)

When you hover over a component, icons appear for interacting with it (for example, an association can be transformed into an association class, an attribute can be deleted, etc.).

6 Wireframe


File
Fill
Stroke
Fill style
Stroke style
Stroke width
Class
Name
Class1
+ add attribute
+ add operation
200%
Class1
atrr1: Type1 {idx} [n..m]
atrr2: Type2 [n..m]
atrr3: Type3
op(arg: Type1): Type1
0..*
class1_class2
1..1
Class2
atrr1: TypeX {id}
atrr2: TypeY
op(arg: Type): Type
Note ...
Class4
Class3
Types
Type = ...
Type1
Type2
Type3 = ...
TypeX = ...
TypeY = ...
+ add type