i219 Software Design Methodology
3. Static modeling

Kazuhiro Ogata (JAIST)

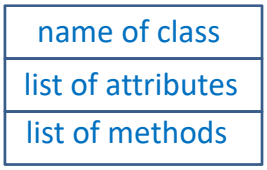
Outline of lecture

Some UML notations for static modeling:

- Class
- Stereotype
- Note
- Relationship
- Class diagram
- Static attribute & method
- Qualification
- Template (parameterized) class
- Object
- Object diagram

Class (1)

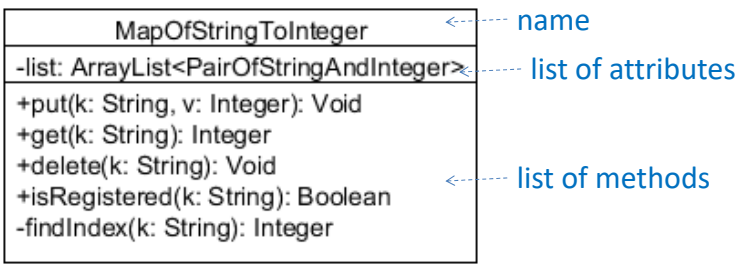
- A class is described as a rectangle that has three partitions.



Note that what are written in the 3rd partition are called operations, and the implementation of an operation is called a method in UML.
In this course, however, terminology "method" is used instead of "operation".

Class (2)

- Let us consider a class of maps that associate integers with strings with lists of pairs of strings and integers.



Class (3)

- An attribute is written as follows:

visibility nameOfAttribute : *class* (or *type*) = *defaultValue*

✓ *visibility* is one of -, +, # and ~ meaning private, public, protected, and package; optional (can be omitted).
✓ *nameOfAttribute* is used to refer to the attribute.
✓ *class* (or *type*) is a class (or type) that is the range of the attribute; optional.
✓ *deafultValue* is the initial value of the attribute; optional.

Note that an attribute may have *multiplicity* and {*property-string*} meaning how many attribute objects or values an object of that class may have and additional properties; e.g., {readOnly} specifies that the attribute cannot be modified.

Class (4)

- One attribute in MapOfStringToInteger:

-list: ArrayList<PairOfStringAndInteger>

In this course, visibility is always - (private), preventing from directly accessing attributes from other classes (even subclasses).

If an attribute in a class *C* needs to be accessed by other classes, *C* provides a getter (a get method) and a setter (a set method) with which the attribute can be observed and modified.

Class (5)

Instead of p_1:$class_1$, ..., p_n:$class_n$, we may write $class_1$ p_1, ..., $class_n$ p_n in this course.

- A method is written as follows:

$visibility$ $nameOfMethod(p_1{:}class_1, \ldots, p_n{:}class_n) : class$

✓ *visibility* is one of -, +, # and ~ meaning private, public, protected, and package; optional.
✓ *nameOfMethod* is used to refer to the method.
✓ $p_i{:}class_i$ is a parameter where p_i is the name and $class_i$ is its class (or type); the parameter list is optional.
✓ *class* is the class (or type) of an object (or value) returned by the method; optional.

Note that a method may have {*property-string*} meaning additional properties; a parameter may be preceded by *direction* that is one of in (if omitted), out, and inout; it may have a default value.

Class (6)

- Five methods in MapOfStringToInteger:

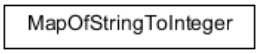
+put(k:String,v:Integer): Void
+get(k:String): Integer
+delete(k:String): Void
+isRegistered(k:String): Boolean
-findIndex(k:String): Integer

In this course, *visibility* is either - (private) or + (public); if a method does not return any objects (values), Void is used.

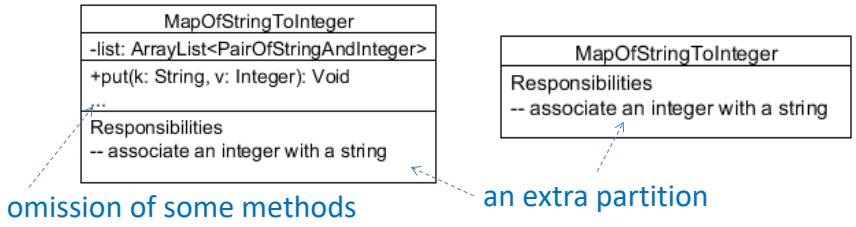
The getter and setter of the attribute list are not provided and then the attribute cannot be accessed by other classes.

Class (6)

- The 2nd & 3rd partitions (attributes & methods) in a class may be omitted.

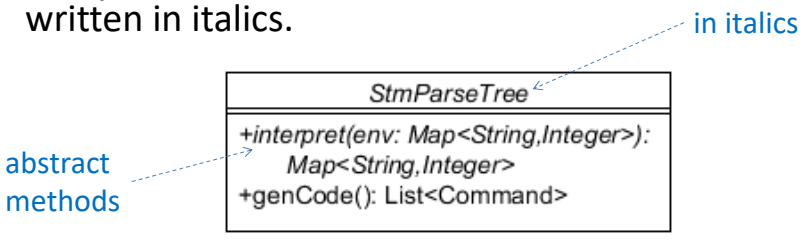


- A class may have an extra partition in which its responsibilities are written.
- Some (or all) of attributes and/or methods may be omitted (which may be indicated with an ellipsis "…").



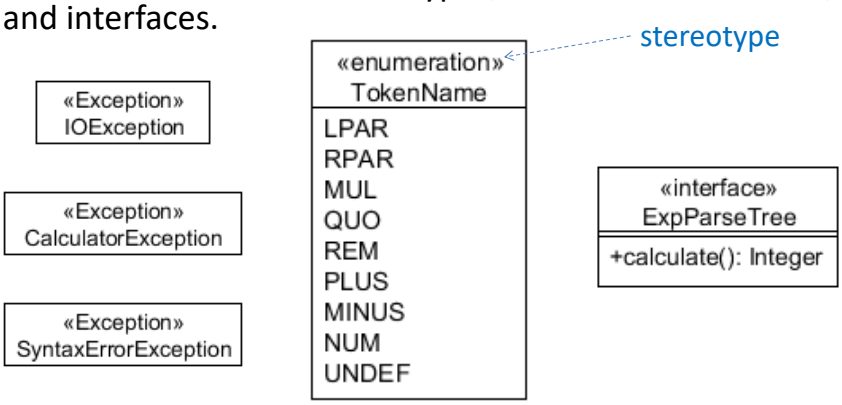
Class (7)

- Abstract classes where some methods (called abstract methods) are not implemented are described in the same way as (concrete) classes except that names & abstract methods are written in italics.



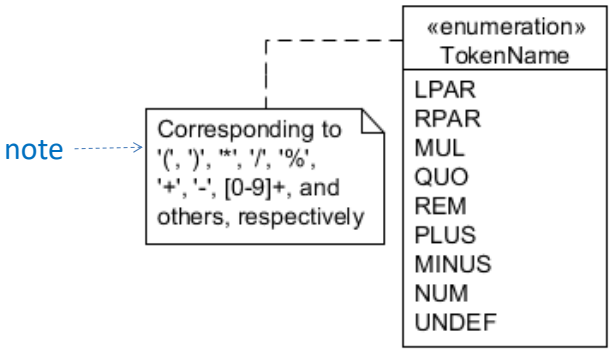
Stereotype

- Some specific classes, such as those for exceptions are indicated with stereotypes whose names are enclosed by double angle brackets (guillemets); stereotypes are also used to describe some data types, such as enumerations, and interfaces.



Note

- A note can be used to give comments or constraints to an element or a collection of elements.

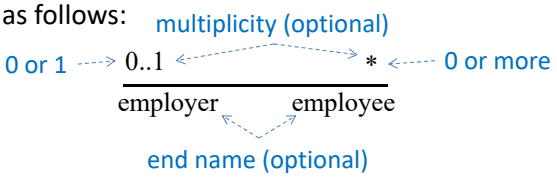


Relationship (1)

- Four kinds of relationships among classes (& others):
  1. Dependency – a change to a class (the dependee) may affect the other class (the depender); written as follows:

     (depender) ----------> (dependee)

  2. Association – a structural relationship among classes that describes a set of links, a link being connections of objects of those classes; written as follows:



Multiplicity : m .. n (greater than or equal m & less than or equal n), * (0 or more; the same as 0 .. *), n (exactly n; the same as n .. n)

Relationship (2)

3. Generalization – a specialization/generalization relationship such as inheritance; written as follows:

   (specialization such as a subclass) ——▷ (generalization such as superclass)

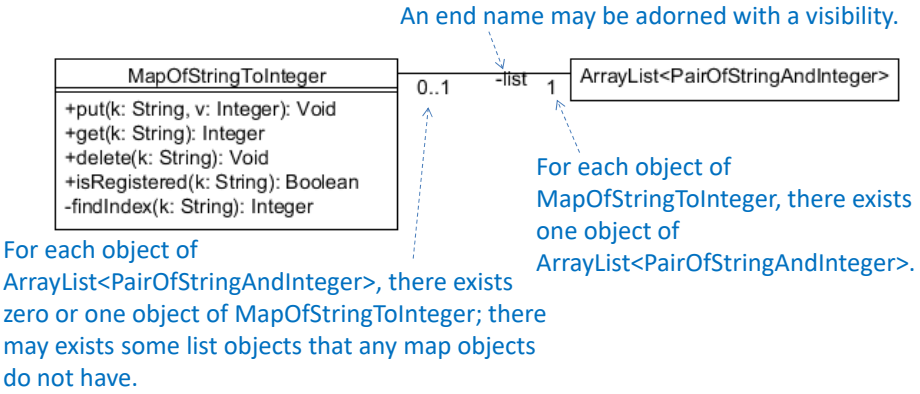
4. Realization – an implementation/specification relationship between two things; one specifies a contraction that the other guarantees to carry out; written as follows:

   (implementation such as a class) - - - - - - - - - - - - - ▷ (specification such as an interface)

Class diagram (1)

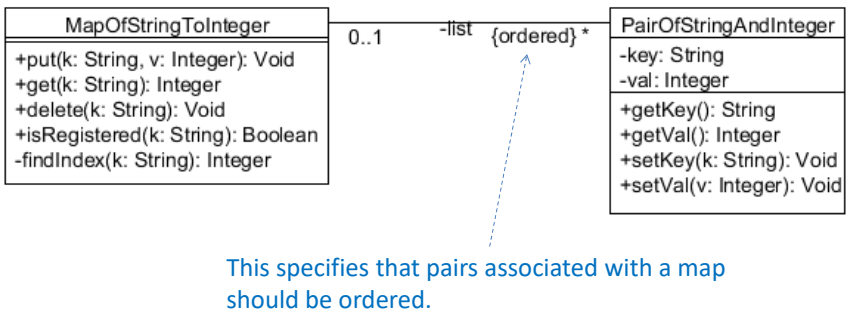
- A class diagram consists of classes (and others such as interfaces) and relationships among them.

Attributes may be described with associations.



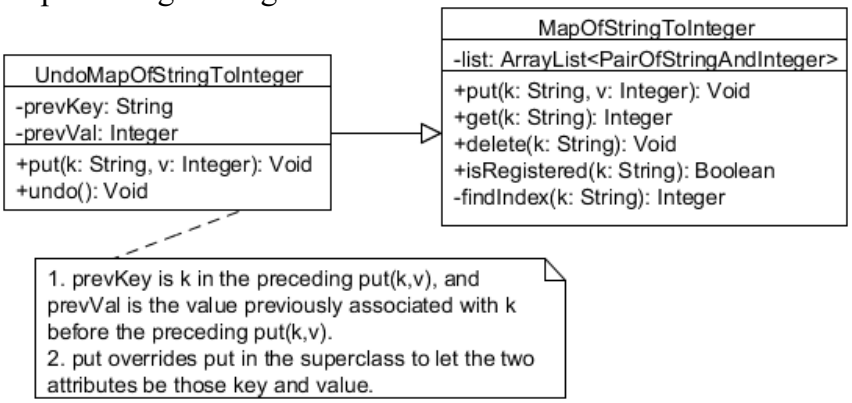
Class diagram (2)

- What the diagram describes on the previous page is also described as follows:



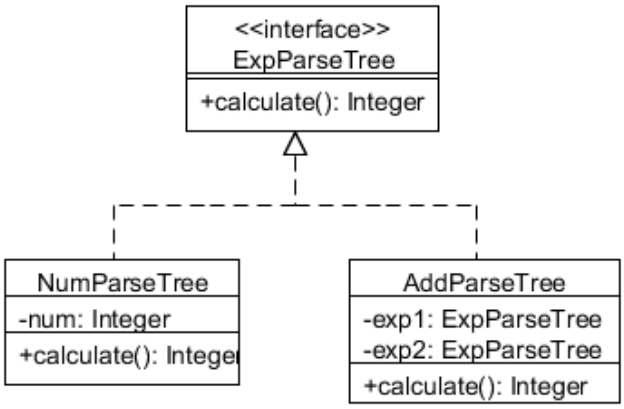
Class diagram (2)

Let us consider a map such that the effect of put(…) can be undone once; the corresponding class is made to extend MapOfStringToInteger.



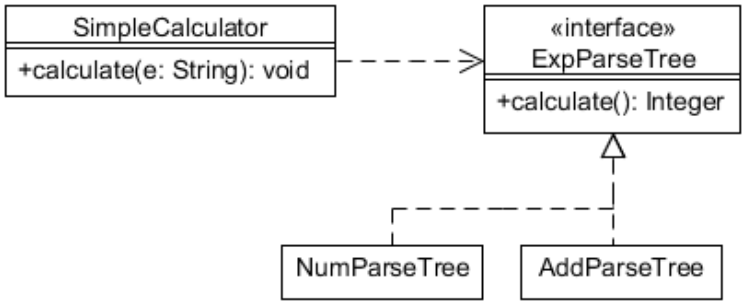
Class diagram (3)

- Let us implement ExpParseTree with two classes NumParseTree & AddParseTree.



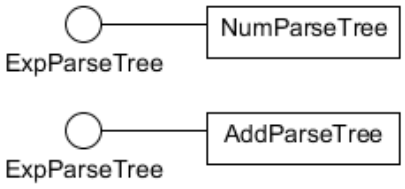
Class diagram (4)

- Let us consider an arithmetic calculator that takes an expression written as a string, converts it into a list of tokens, makes a parse tree from the list, calculates the expression, and displays the result; the calculator depends on parse trees.

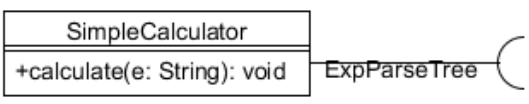


Class diagram (5)

- Provided (implemented) interfaces are described as follows:

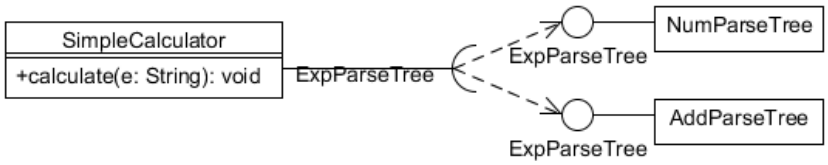


- Required interfaces are described as follows:



Class diagram (5)

- A simplified version of the diagram on the previous, previous page is described as follows:



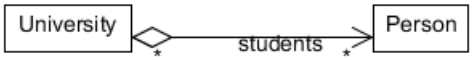
Class diagram (6)

- Composition

The multiplicity on the composite side is 1 or 0..1 (which is often omitted) b/c a component class is designed s.t. a component can have at most or only one composite as its owner.

When a composite is destructed, so do all of its components.

- Aggregation



Class diagram (6)

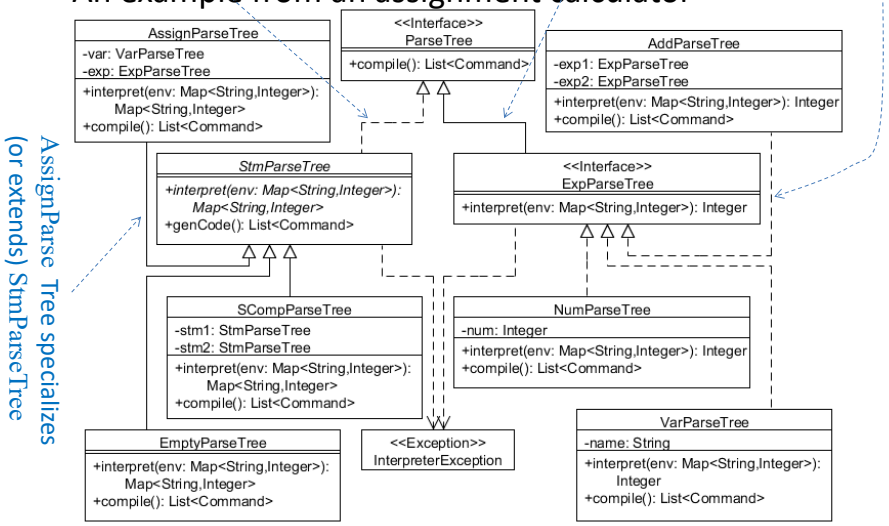
StmParseTree implements ParseTree

ExpParseTree specializes (or extends) ParseTree

AddParseTree implements ExparseTree

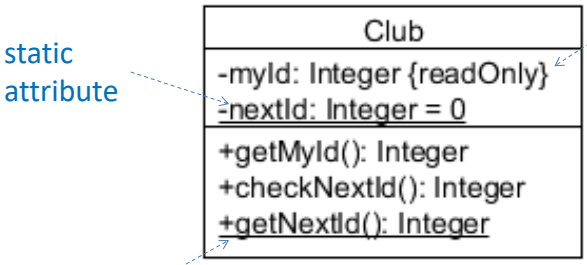
AssignParse Tree specializes (or extends) StmParseTree

- An example from an assignment calculator



Static attribute & method

- Static attributes & methods in a class are shared by all objects of that class.
- Underlines are used to specify static attributes & methods.



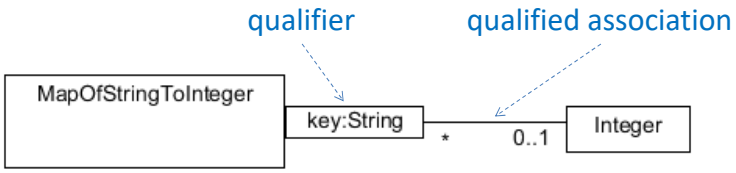
static attribute

property string saying that the attribute cannot be modified once it is initialized.

static method

Qualification

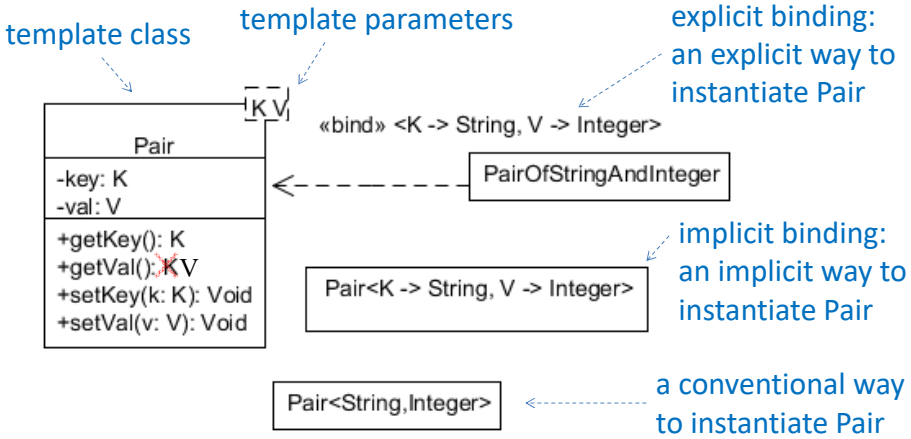
- The design of MapOfStringToInteger was dependent on lists.
- Qualification lets us make the design more abstract.



For each map and each string, there exists at most one integer.

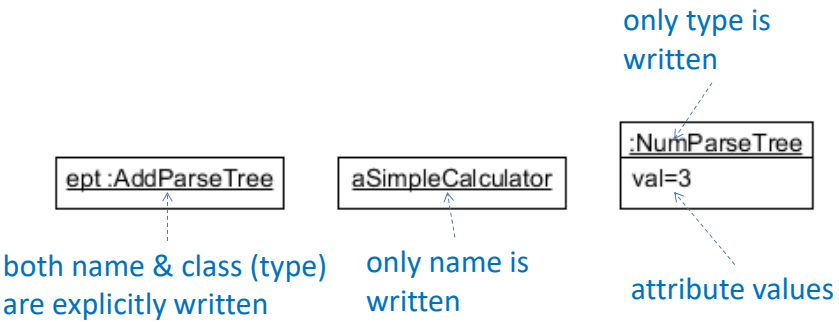
Template (parameterized) class

- Useful to describe generic things, such as generic lists



Object

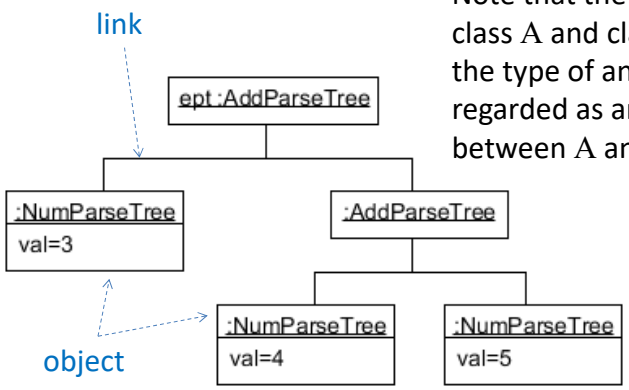
- Objects are described as rectangles; names/types are underlined; they may have attribute values.



Object diagram

- A snapshot of the objects in a system

Links are instances of associations. Note that the relation between class A and class B that is used as the type of an attribute in A is regarded as an association between A and B in the course.



Summary

Some UML notations for static modeling:

- Class
- Stereotype
- Note
- Relationship
- Class diagram
- Static attribute & method
- Qualification
- Template (parameterized) class
- Object
- Object diagram