UML Class Diagrams 1

Martin Nečaský
Dept. of Software Engineering
Faculty of Mathematics and Physics
Charles University in Prague
Class Diagram

- describes objects from the structural point of view
  - What kinds of objects and relationships among them are important?
  - What kinds of features the kinds of objects and relationships have?

- at conceptual (domain) level
  - specifies a vocabulary to talk about a problem domain
  - objects = real-world entities

- at implementation (logical) level
  - specifies (a) how a domain is mapped to software elements (i.e. software data structures) or (b) completely new software elements without any counterparts in the real world (e.g. GUI, controller, etc.)
  - elements correspond directly to elements in the developed software system
  - objects = machine-interpretable logical structures
    - (Java/C#/...) class instances, relational tables, XML elements, etc.
Class Diagram – Conceptual Example

- **Organization**
  - legalName
  - officialNumber
  + contractingAuthority
  + issuedContract
  + tenderingSupplier
  + awardedSupplier

- **Address**
  - streetName
  - streetNumber
  - city
  - country
  + mainAddress
  + tenderAddress

- **Item Type**
  - code
  - title

- **Contract**
  - referenceNumber
  - title
  - description
  - mainObject
  - additionalObject [0..*]
  - startDate
  - endDate
  - estimatedPrice
  - agreedPrice
  - actualPrice
  - numberOfTenders
  + lot 0..*
  + parentContract 1

- **Tender**
  - estimatedEndDate
  - offeredPrice

- **Organization**
  - Contract
    - referenceNumber
    - title
    - description
    - mainObject
    - additionalObject [0..*]
    - startDate
    - endDate
    - estimatedPrice
    - agreedPrice
    - actualPrice
    - numberOfTenders

- **Tendering Supplier**
  - +tenderingSupplier

- **Contracting Authority**
  - +contractingAuthority

- **Awarded Supplier**
  - +awardedSupplier

- **Tendered Contract**
  - +tenderedContract

- **Supplied Contract**
  - +suppliedContract

- **Lot**
  - 0..*
Class Diagram – Implementation Example (GUI)

```
PortalConfiguration
- name: String
- isDefault: boolean
- cloneable: boolean
# initialize() : void
+ clone() : void

Portlet
- name: String
- description: String
+ show() : void
+ hide() : void
+ refresh() : void
+ relocate(int, int) : void

GUIContainer

EntityList

Advertisement

List
0..*

List
1..*

List
0..*
```

«interface» List
Class Diagram – Implementation Example (DB)
Class Diagram – Basic Constructs

- Class
- Class Property
  - Class attribute
  - Association end
- Association
Class

- describes a set of objects that share the same specification of features, constraints and semantics

Discussion

- What does the term ‘object’ mean?
- What do the terms ‘feature’, ‘constraint’ and ‘semantics’ mean?
Class

- term ‘object’
  - from the conceptual (domain) point of view
    - real-world entities (objects)
  - from the implementation (logical) point of view
    - machine-interpretable representation of real-world entities (objects)
    - purely programmatic data structures (no counterparts in the real world)

- term ‘feature’
  - feature declares behavioral or structural characteristics of objects

- term ‘constraint’
  - constraint is a condition or restriction which must be held by objects described by class

- term ‘semantics’
  - meaning of the objects for stakeholders
Class

- syntactically, class has a **name** and a set of **features**

- **feature**
  - **property** = structural characteristics
    - two kinds of properties which are the same thing but have different notations
    - **attribute**
      - more suitable for properties with simple data values (strings, integers, ...) or with complex but less important data values (e.g. address in some cases)
    - **association end**
      - more suitable for properties with complex data values (other classes)
  - **operation** = behavioral characteristics
    - actions performed on objects
    - note: actions to manipulate properties usually considered implicitly
Class

- name: Person

features
  - properties
    - attributes: name, email, phone, registrationDate, homepage, dblp
    - association ends: colleague, person (default name), team (default name)
  - operations
    - N/A
Class

- **name:** Team

**features**

- **properties**
  - attributes: name, homepage
  - association ends: member

- **operations**
  - notify

```
Person
+ name: String
+ email: String [0..*]
+ phone: String [0..*]
+ registrationDate: Date
+ homepage: URL
+ dblp: URL

Team
+ name: String
+ homepage: URL
+ notify(Project) : void
```

0..* +member 1..*

+colleague 0..*

0..*
Class Attribute

- notation for a class property displayed inline in the class
- each attribute displayed as a single line
- use it to model properties with simple, non-structured, values
- you can also use it to model properties with more complex, structured, values but they should be the less important ones in your schema
Class Attribute

<property> ::=  
  [ <visibility> ] <name> ['::'] <prop-type> ]  
  [ '[' <multiplicity> ']' ]  
  [ '=' <default> ]  
  [ '{' <prop-modifier> [ ',', <prop-modifier> ]* '}' ]

- <visibility>: public (+) / private (-) / protected (#) / package (~)
- <name>: label we use to refer the attribute in the context of the class
- <prop-type>: restricts possible values of the attribute only to values of a given data type or instances of a given class
- <multiplicity>: restricts the number of possible values the attribute may have (default is 1)
- <default>: initial value of the attribute
- <prop-modifier>: additional meta-properties of the attribute
  - readOnly, non-unique, id, ordered, ...
## Class Attribute

| Person |
|-----------------
| - name: String [0..1] = "no-name" {readOnly} |
| + degree: DegreeType = assistant_prof |
| + phone: String [0..*] {non-unique} |
| - contactPhone: String |
| + address: Address [1..3] {ordered} |
| - personalNumber {id} |
Association End

- notation for a class property displayed as a solid line between the class (source class) and class which is the type of the property (target class)
- declares that there can be semantic links between instances of associated classes
- use it to model properties with structured values
Association End

Person

Address

+ address
1..3

{ordered}

- street: String
- city: String
- country: String
Association End vs. Attribute

Person

+ address: Address [1..3] {unique, ordered}

Address

- street: String
- city: String
- country: String
Binary Associations

- Binary association can be used to model pair of properties that are linked together as inverses.
Navigability

- navigable association end means that its instances can be accessed efficiently from instances of the other end (symbol > at the association end)
- non-navigable end means that navigation is possible but is not optimized (symbol X at the association end)
- no symbol at association end means that navigability is not specified
More on Multiplicities

A Team has none or one manager.

A researcher manages zero or more teams.

<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..1</td>
<td>zero or one (optional)</td>
</tr>
<tr>
<td>1 or 1..1</td>
<td>exactly one</td>
</tr>
<tr>
<td>* or 0..*</td>
<td>zero or more (arbitrary number)</td>
</tr>
<tr>
<td>1..*</td>
<td>1 or more (at least one)</td>
</tr>
<tr>
<td>1..6</td>
<td>1 up to 6</td>
</tr>
</tbody>
</table>
Association Classes

- combination of class and association
- use it to model association with properties
N-ary Associations

- association with three or more association ends
  - declares a link between three or more class instances

- how to read it: take N-1 association ends, instantiate them (you get instances I{1}, ..., I{N-1}) and ask “What instances of N the tuple (I{1}, ..., I{N-1}) is linked to?”
  - **How would you read diagram below?**

```
[Diagram]
Person + worker 0..* 0..1
Project
Team
0..1
```

What would cardinalities 1..1 or 1..* mean?
N-ary Association vs. Attribute

- compare properties name (attribute) and worker (association end)
- attribute assigns its type instance to a class instance
- association end assigns its class instance to a combination of instances of each class at the other ends
- *Can you explain it on the example above?*
More on Links

- reminder: links between class instances modeled by associations

- **But, how many links modeled by a single association may exist between connected instances?**

```
class Nary
Person  Team  Project
+worker  0..*    0..1
         0..1
```

```
class Multiplic...
Team  Researcher
+managedTeam  0..*
+manager      0..1
```

```
Person  Team
+worker    0..*
         0..1
```

```
Project
0..1
```
More on Links

- we can use **non-unique** property modifier to specify that there can be more links connecting the same instances
  - one or more association ends with **non-unique**
  - some tools use **bag** property modifier (from OCL)
More on Links

```plaintext
class AssociationClass
Person Team
Membership
- fromDate:  Date
- toDate:  Date
- position:  String
+member
1..*
{ordered}
0..*
```

Diagram:
- Person 1
- Person 2
- Team A

Relationships:
- Person 1 to Team A
- Person 2 to Team A