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

Institution
+ officialTitle: String
+ homepage: URL

University
0..*

+ employer 0..*
+ employee 0..*

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

+ colleague 0..*

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

+ principalPartner 0..*
+ principalProject 0..*
+ partner 0..*

Institution
0..1

Project
+ name: String
+ email: String [0..*]
+ description: String
+ homepage: URL
+ status: int
+ close() : void

Student
0..*

Researcher
+ topic: String [1..*]
+ degree: DegreeType = assistant_prof

DegreeType
«enumeration»
researcher
assistant_prof
associated_prof
full_prof

0..*
1..*
0..1
0..*
0..1
0..*
0..*
0..*
0..1
0..*
0..*
0..*
0..1
0..1
0..*
0..*
0..1
0..1
0..*
0..*
0..*
Class Diagram – Conceptual Example

```
class P...

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

Organization
- legalName
- officialNumber
+contractingAuthority 1
+issuedContract 0..*
+tenderingSupplier 0..*
+awardedSupplier 0..1

Address
- streetName
- streetNumber
- city
- country
+mainAddress 1
+tenderAddress 0..1

ItemType
- code
- title

Tender
- estimatedEndDate
- offeredPrice

+issuedContract 0..*
+tenderedContract 0..*
+contractingAuthority 1
+tenderingSupplier 0..*
+awardedSupplier 0..1

+mainAddress 1
+tenderAddress 0..1

+parentContract 1
+lot 0..*
```
Class Diagram – Implementation Example (GUI)

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

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

GUIContainer

EntityList

Advertisement

class GUI Mo...»
```

```
List
0..* 1..*
```

```
GUIContainer
```

```
Portlet
```

```
PortalConfiguration
```

```
class GUI Mo...
```
Formal Foundations of Software Engineering

class PSM_RELATIONAL
Contract
«column»
* referenceNumber:  NUMBER(8)
* title:  VARCHAR2(50)
description:  CLOB
* ... = code)
«FK»
+PK_ItemType 1
+FK_Contract_Contract
0..*
(parentContractId = contractId)
«FK»
+PK_Contract
1

Organization
«column»
* legalName: VARCHAR2(50)
* officialNumber: NUMBER(9)
*PK organizationId: NUMBER(8)
*FK addressId: NUMBER(8)
«FK»
+FK_Organization_Address(NUMBER)
+FK_OrganizationisFunction(NUMBER)
+UQ_Organization_officialNumber(NUMBER)

Tender
«column»
* estimatedEndDate:  DATE
* offeredPrice:  NUMBER(9)
*PK tenderId: NUMBER(8)
*PK tenderingSupplierId: NUMBER(8)
*PK tenderedContractId: NUMBER(8)
«FK»
+FK_Tender_Contract(NUMBER)
+FK_Tender_Organization(NUMBER)

Contract
«column»
* startDate:  DATE
* endDate:  DATE
* estimatedPrice:  NUMBER(9)
agreedPrice:  NUMBER(9)
actualPrice:  NUMBER(9)
numberOfTenders:  NUMBER(2)
*PK contractId: NUMBER(8)
*FK contractingAuthorityId: NUMBER(8)
FK awardedSupplierId: NUMBER(8)
*FK mainAddressId: NUMBER(8)
FK tenderAddressId: NUMBER(8)
FK parentContractId: NUMBER(8)
«FK»
+FK_Contract_Address(NUMBER)
+FK_Contract_Organization(NUMBER)
+FK_Contract_Organization(NUMBER)
+FK_Contract_Organization(NUMBER)
+PK_Organization 1

ItemType
«column»
*PK code:  NUMBER(8)
* title:  VARCHAR2(50)
«PK»
+PK_ItemType 1
+FK_Item_ItemType(NUMBER)
0..*
(contractId = code)
«FK»
+PK_ItemType 1

Item
«column»
* code:  NUMBER(8)
FK contractId:  NUMBER(8)
«FK»
+FK_Item_Contract(NUMBER)
+FK_Item_ItemType(NUMBER)
0..*
(contractId = contractId)
«FK»
+PK_ItemType 1

Address
«column»
streetName:  VARCHAR2(50)
city:  VARCHAR2(50)
country:  VARCHAR2(50)
«PK»
+PK_Address 1
+PK_Address 1
+FK_Organization_Address 0..*
(addressId = addressId)
«FK»
+PK_Address 1

Tender_Address
«column»
streetName:  VARCHAR2(50)
city:  VARCHAR2(50)
country:  VARCHAR2(50)
«PK»
+PK_Address 1
+PK_Address 1
+FK_Organization_Address 0..*
(addressId = addressId)
«FK»
+PK_Address 1

TenderedContract
«column»
*startDate:  DATE
*endDate:  DATE
*estimatedPrice:  NUMBER(9)
agreedPrice:  NUMBER(9)
actualPrice:  NUMBER(9)
numberOfTenders:  NUMBER(2)
*PK contractId: NUMBER(8)
*FK contractingAuthorityId: NUMBER(8)
FK awardedSupplierId: NUMBER(8)
*FK mainAddressId: NUMBER(8)
FK tenderAddressId: NUMBER(8)
FK parentContractId: NUMBER(8)
«FK»
+FK_Contract_Address(NUMBER)
+FK_Contract_Organization(NUMBER)
+FK_Contract_Organization(NUMBER)
+FK_Contract_Organization(NUMBER)
+PK_Organization 1

TenderingSupplier
«column»
* offeredPrice:  NUMBER(9)
* tenderId: NUMBER(8)
*FK tenderingSupplierId: NUMBER(8)
*FK tenderedContractId: NUMBER(8)
«FK»
+FK_Tender_Contract(NUMBER)
+FK_Tender_Organization(NUMBER)

AwardedSupplier
«column»
* awardedSupplierId: NUMBER(8)
*FK awardedSupplierId: NUMBER(8)
«FK»
+FK_Contract_Organization(NUMBER)
+FK_Contract_Organization(NUMBER)
+FK_Contract_Organization(NUMBER)
+PK_Organization 1

MainAddress
«column»
* mainAddressId: NUMBER(8)
*FK mainAddressId: NUMBER(8)
«FK»
+FK_Contract_Organization(NUMBER)
+FK_Contract_Organization(NUMBER)
+FK_Contract_Organization(NUMBER)
+PK_Organization 1

TenderingAddress
«column»
* tenderAddressId: NUMBER(8)
*FK tenderAddressId: NUMBER(8)
«FK»
+FK_Contract_Organization(NUMBER)
+FK_Contract_Organization(NUMBER)
+FK_Contract_Organization(NUMBER)
+PK_Organization 1

ParentContract
«column»
* parentContractId: NUMBER(8)
*FK parentContractId: NUMBER(8)
«FK»
+FK_Contract_Organization(NUMBER)
+FK_Contract_Organization(NUMBER)
+FK_Contract_Organization(NUMBER)
+PK_Organization 1

Supplier
«column»
*FK supplierId: NUMBER(8)
«FK»
+FK_Tender_Organization(NUMBER)
+FK_Tender_Organization(NUMBER)
+FK_Tender_Organization(NUMBER)
+PK_Organization 1

Organization
«column»
*FK organizationId: NUMBER(8)
*FK addressId: NUMBER(8)
«FK»
+FK_Organization_Address(NUMBER)
+FK_OrganizationisFunction(NUMBER)
+UQ_Organization_officialNumber(NUMBER)

Address
«column»
streetName:  VARCHAR2(50)
city:  VARCHAR2(50)
country:  VARCHAR2(50)
«PK»
+PK_Address 1
+PK_Address 1
+FK_Organization_Address 0..*
(addressId = addressId)
«FK»
+PK_Address 1

Organization
«column»
*FK organizationId: NUMBER(8)
*FK addressId: NUMBER(8)
«FK»
+FK_Organization_Address(NUMBER)
+FK_OrganizationisFunction(NUMBER)
+UQ_Organization_officialNumber(NUMBER)
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
Exercise

- name: ?
- features
  - properties
    - attributes: ?
    - association ends: ?
  - operations
    - ?

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

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

```
+ member 1..*
```

```
+ colleague 0..*
```

```
+ member 1..*
```

```
+ colleague 0..*
```
Exercise

- name: Person
  - features
    - properties
      - attributes: name, email, phone, registrationDate, homepage, dblp
      - association ends: colleague, person (default name), team (default name)
    - operations
      - no operations
Exercise

- name: ?

- features
  - properties
    - attributes: ?
    - association ends: ?
  - operations
    - ?
Exercise

- name: Team

  features

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

  - operations
    - notify

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

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

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, ...
Exercise

- Explain each attribute below

<table>
<thead>
<tr>
<th>Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>- name: String [0..1] = &quot;no-name&quot; {readOnly}</td>
</tr>
<tr>
<td>+ degree: DegreeType = assistant_prof</td>
</tr>
<tr>
<td>+ phone: String [0..*] {non-unique}</td>
</tr>
<tr>
<td>- contactPhone: String</td>
</tr>
<tr>
<td>+ address: Address [1..3] {ordered}</td>
</tr>
<tr>
<td>- personalPhoneNumber {id}</td>
</tr>
</tbody>
</table>
Binary Association

- 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
Exercise

- Explain the binary association below

Diagram:

- Person
- Address
- + address
  - 1..3
- {ordered}

Address:
- street: String
- city: String
- country: String
Exercise

- Express the previous binary association as an attribute

```
+ address
1..3
{ordered}
```

```
Person
```

```
Address
- street: String
- city: String
- country: String
```
Binary Association vs. Attribute

Person

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

Address

- street: String
- city: String
- country: String
Bidirectional Associations

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

```
class BidirectionalAssociation
Person
+employee
1..*

Institution
+employer
0..*
```
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
Exercise

- Read multiplicities in the following diagram.

- Explain in words multiplicities in the following table.

<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..1</td>
<td>... ? ...</td>
</tr>
<tr>
<td>1 or 1..1</td>
<td>... ? ...</td>
</tr>
<tr>
<td>* or 0..*</td>
<td>... ? ...</td>
</tr>
<tr>
<td>1..*</td>
<td>... ? ...</td>
</tr>
<tr>
<td>1..6</td>
<td>... ? ...</td>
</tr>
</tbody>
</table>
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
Exercise

- Model a membership of a person in a team using an association class
Exercise

- Model a membership of a person in a team using an association class

```java
class AssociationClass

Person

Team

Membership

- fromDate: Date
- toDate: Date
- position: String

+ member

1..* 0..*
```
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?
Exercise

- Read the following diagram
- What would cardinalities 1..1 or 1..* mean?
Exercise

- 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?*
Exercise

- reminder: links between class instances modeled by associations
- How many links modeled by a single association may exist between connected instances?
Exercise

- Only 1 (by default)
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)
Exercise

- What does it mean on the following example?
Exercise

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

Person 1

Person 2

Team A