Class modelling (part 1)

Fabrizio Maria Maggi
Institute of Computer Science
(These slides are derived from the book “Object-oriented modeling and design with UML”)
UML and OO approaches

**Unified Modeling Language (UML)** is a standardized, general-purpose modeling language. UML includes a set of graphic notation techniques to create visual models of object-oriented (OO) software-intensive systems.

- An OO approach includes 4 aspects:
  - **Identity**
    - Data organized into discrete distinguishable entities (**objects**)
  - **Classification/Abstraction**
    - Objects with the same attributes and operations are grouped into a **class**
    - Each object is said to be an **instance** of its class
  - **Inheritance**
    - Sharing of attributes and operations (**features**) among classes based on a hierarchical relationship
    - A superclass has general information that subclasses refine and elaborate
  - **Polymorphism**
    - The same operation may behave differently for different classes
UML and OO approaches
The hard part of software development is the manipulation of its essence, owing the inherent complexity of the problem, rather than the accidents of its mapping into a particular language.

- A clean design in a precise notation
  - facilitates integration, maintenance, enhancement and the entire software lifecycle
  - provides useful documentation
- Design flaws that surface during implementation are more costly to fix than those that are found earlier
- A premature focus on implementation restricts design choices and leads to an inferior product
OO development
UML models

- **Class model**
  - Static structure of objects and their relationships
  - *Class diagrams*
    - Nodes are classes and arcs are relationships among classes

- **Interaction model**
  - How the objects in a system cooperate to achieve broader results
  - *Use cases*
    - Describe the functionalities of a system
    - Are elaborated with *sequence diagrams* (object interactions and time sequence of their interaction) and *activity diagrams* (processing steps)

- **State model**
  - Aspects of an object that change over time
  - *State diagrams*
    - Nodes are states and arcs are transitions between states caused by events
UML models

Configuration

installables

Package

+name
+copyright
+description
+license
+license_file
+homepage
+source
+source_type
+source_directory
+version
+patches

0..*
1

Platform

+build_directory
+dependences
+manifest
+name

0..*

Archive

hash
hash_algorithm
url

Autobuild class model

BuildConfiguration

+default
+name

configure
1
build
1

Executable

+command
+arguments[0..*]
+options[0..*]
+execute()

0..*

If set, the executable overrides its parent. Any options are appended to the parent's options.
Class modelling

- **Classes**
  - A class describes a group of objects with the same properties (attributes), behavior (operations), kinds of relationships and semantics
  - Classes often appear as nouns in problem descriptions with users

- **Objects**
  - An object is a concept, abstraction or thing with identity that has a meaning for an application
  - An object is an instance of a class
How Many Classes? And Instances?
Class diagrams

- **Class**
  - **UML notation:** box with a class name

- **Object**
  - **UML notation:** box with an object name followed by a colon and a class name. The object name and the class name are both underlined.
Attributes and values

- **Attribute**
  - An attribute is a named property of a class that describes a value held by each object of the class.
  - UML notation: attributes are listed in the second compartment of the class box. Optional details, such as type and default value, may follow each attribute.

- **Value**
  - A value is a piece of data.
  - UML notation: values are listed in the second compartment of the object box.
Attributes and values
Attributes and values
Operations and methods

- **Operation**
  - An operation is a function or procedure that may be applied by or to objects in a class
  - **UML notation**: operations are listed in the third compartment of the class box

- **Method**
  - A method is the implementation of an operation for a class
Operations and methods
Operations and methods
Visibility for attributes and operations

- + public
- # protected
- - private
- ~ package
Visibility for attributes and operations

<table>
<thead>
<tr>
<th>Keyword</th>
<th>C#</th>
<th>C++</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>private</td>
<td>class</td>
<td>class</td>
<td>class</td>
</tr>
<tr>
<td>protected</td>
<td>derived classes</td>
<td>derived classes</td>
<td>derived classes and/or within same package</td>
</tr>
<tr>
<td>package</td>
<td>-</td>
<td>-</td>
<td>within its package</td>
</tr>
<tr>
<td>public</td>
<td>everybody</td>
<td>everybody</td>
<td>everybody</td>
</tr>
</tbody>
</table>
(Binary) links and associations

- **Link**
  - A link is a physical or conceptual connection among objects
  - **UML notation**: line between objects. A link can have a name (underlined)

- **Association**
  - An association is a description of a group of links with common structure and common semantics
  - **UML notation**: line between classes. An association can have a name (not underlined)
How many associations? Links?
Multiplicity

- Specifies the number of instances of one class that may relate to a single instance of an associated class.

- UML notation: specified at the end of the association lines.
  - Examples: “1” (exactly one); “3..5” (three to five, inclusive); “*” (many, zero or more).
Multiplicity many-to-many

Person
- name

Company
- name

John : Person
- = GE
- name = "John"

Mary : Person
- = GE
- name = "Mary"

Sue : Person
- = GE, IBM
- name = "Sue"

Alice : Person
- = IBM
- name = "Alice"

Jeff : Person
- name = "Jeff"

GE : Company
- = John, Mary, Sue

IBM : Company
- = Sue, Alice
Multiplicity one-to-one

Country

name

1

HasCapital

CapitalCity

name

1

Canada: Country

= Ottawa
name = "Canada"

Ottawa: CapitalCity

= Canada
name = "Ottawa"

France: Country

= Paris
name = "France"

Paris: CapitalCity

= France
name = "Paris"

Senegal: Country

= Dakar
name = "Senegal"

Dakar: CapitalCity

= Senegal
name = "Dakar"
Association end names

- Association ends can be provided with a name as well as with a multiplicity

![Diagram showing association between Person and Company with multiplicity 0..1 for Company's association end.](image)
Association end names

- Association end names are necessary for associations between two objects of the same class. They can also distinguish multiple associations between a pair of classes.
- Association end names as pseudo attributes.
Directed associations

1. Class A → Class B

2. Association end name

Class A
- m_ClassB: ClassB

Class B
Association classes

- An association class is an association that is also a class
- Like a class, an association class can have attributes and operations and participate in associations
- UML notation: class box attached to the association by a dashed line
Association or standard class?
Association or standard class?
Association or standard class?
UML models
Generalization and Inheritance

- **Generalization** is the relationship between a class (superclass) and one or more variations of the class (subclasses)
  - The superclass holds common attributes, operations and associations. The subclasses add specific attributes, operations and associations (each subclass is said to inherit the features of its superclass)
  - Simple generalization organizes classes into a hierarchy
  - There can be multiple levels of generalizations
  - A large arrowhead denotes generalization. The arrowhead points to the superclass
- A **generalization set name** is an enumerated attribute that indicates which aspect of an object is being abstracted by a particular generalization
Ancestors and Descendants

- Generalization is transitive across an arbitrary number of levels:
  - An instance of a subclass is also an instance of all its ancestor classes
  - An instance includes a value for every attribute of every ancestor class
  - An instance can invoke any operation of any ancestor class
  - Each subclass not only inherits all the features of its ancestors but also adds its own specific features as well
Generalization and Inheritance

Diagram
- name

Figure
- penThickness
- color
- centerPosition
- penType
- move()
- select()
- rotate()
- display()

ZeroDimensional
- orientation
- scale()

Point
- display()

OneDimensional
- orientation
- scale()

Line
- endPoints
- display()

Arc
- radius
- startAngle
- arcAngle
- display()

Spline
- controlPoints
- display()

TwoDimensional
- orientation
- fillType
- scale()
- fill()

Polygon
- numOfSides
- vertices
- display()

Circle
- diameter
- display()
- rotate()
Use of Generalization

- **Polymorphism**
  - Increases the flexibility of software. You can add a new subclass to inherit the superclass behavior without disrupting existing code
    - Overriding features

- **Objects taxonomy**
  - Organizes objects on the basis of their similarities and differences

- **Reuse of code**
  - You can inherit code within your application as well as from past work