CS 340

UML

Class Diagrams
A **model** is an abstraction of a system, specifying the modeled system from a **certain viewpoint** and at a **certain level of abstraction**
Complexity and Modeling

• Modeling is a well-accepted engineering technique for dealing with complexity

• Models let us focus on one view of the system while ignoring irrelevant details (a simplification of reality)
  – Example: Plumbers only care about plumbing, so we give them plumbing diagrams; electricians only care about electrical work, so we give them electrical diagrams, etc.

• Models can be created at different levels of abstraction
  – 10,000 ft. view vs. 10 ft. view
  – High-level: sketch of the building's exterior for the customer
  – Low-level: detailed electrical plan for the electrician
Software Modeling

- Why do we create models of software systems?
- Models help us visualize and understand the system we're building.
- Models are used to document the design of the system and communicate it to others.
- Models serve as a guide during system construction.
UML

• Unified Modeling Language
• History
• Models
  – Data
  – Behavior
  – Interaction
  – Packaging
Levels of Models

- Conceptual Model – Problem Understanding
- Specification/Requirement Document – Solutions
- High-Level Design Model
  - Systems, Packages, Classes
  - Interactions, Interfaces
- Low-Level Design Model
  - Class Conceptual Model
  - May be better done with Class documentation such as Javadoc
- Implementation
  - Should be done in code not UML
UML Models

• Structural/Data
  – *Class Diagram*
  – Object Diagram

• Behavior
  – Use Case Diagram
  – State Diagram
  – Activity Diagram

• Interaction
  – Communication Diagram
  – Sequence Diagram

• Physical Diagram
  – Component Diagram
  – Deployment Diagrams
Classes At the Conceptual Level

A Class – typically named with a common noun.

Mother

At the conceptual level the class is a set of objects with common properties.
Classes At the Design Level

A Class – typically named with a common noun.
At the design level a Class is a Type.

<table>
<thead>
<tr>
<th>Mother</th>
</tr>
</thead>
<tbody>
<tr>
<td>- name : Name [1..*]</td>
</tr>
<tr>
<td>- age : int</td>
</tr>
<tr>
<td>+ print()</td>
</tr>
<tr>
<td>+ equals(Object)</td>
</tr>
</tbody>
</table>

Visibility constraints (optional):
+ (public), - (private), # (protected), / (derived), _ (static), ~ (package).

We may add cardinality constraints to an attribute. In this case it means that a Mother has 1 or more names. If the cardinality constraint is omitted it is assumed to be the cardinality constraint “1”.

For some situations we may omit the Method Section or the Attribute Section.
Objects

An Object – notice the name is underlined.

While some find Objects useful in the Class diagram others do not.
Associations

An association can be thought of as a declarative sentence.

The name in the association is optional but suggested.

Typical association constraints are 1, 0..1, *, 1..*, n..m.

UML 1.0 supported a notation for an arbitrary set of non-negative integers (e.g. 2:4, 6,12 meaning the set {2,3,4,6,12}

Most associations are binary.
Associations Continued

• Roles

- Although this example has 2 roles, there may be only 1, or most often 0 roles on an association.
- Roles can be viewed as Specializations

• While not all versions of UML support the same annotations, you may find some when reading UML diagrams.

![Diagram showing associations between Male, Husband, Wife, Female, Mother, and Child.]
Directed Associations

• Should only be used as a design construct

- Means that we can access an Address via a Person, but, given an Address we can’t determine the Person associated with it.
- If the association constraint on the opposite side of the navigability arrow is a 1, then this is equivalent to

<table>
<thead>
<tr>
<th>Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>address : Address [1..*]</td>
</tr>
</tbody>
</table>

- Notice the cardinality constraint to the right of the attribute.
Aggregation/Composition

- **Aggregation**
  - Can be read as Person is subpart of a Company
  - Can be transitive but acyclic.
  - Some people prefer to just use associations.
  - Can use “comb” representation of there are different types of subparts.

- **Composition**
  - Can be read as “A Car is composed of 1 or more Car Parts”.
  - A much stronger form of Aggregation. Some suggest it implies a Car Part cannot exist unless it is part of a car.
  - Notice there is no association constraint next to the black diamond because it is always 1.
Generalization/Specialization

- For conceptual modeling every Faculty Member, Staff Member, and Student is a BYU Person. “Is A” Semantics. That is, subset semantics. In this case the set, Student, is a subset of the set BYU Person.
- At the Design level it represents “subtype” or “inheritance” semantics.
- A generalization/specialization often has a single specialization.
- Though not standard UML, I may include the following annotations: {union}, {disjoint}, and {partition}. 
General Constraints/Notes

• General Constraints
  – {A Father must be a male Person}
  – I often omit the braces.
  – Can be expressed formally in OCL, a type of first-order logic.

• Notes
  – I use text such as
    NOTE: A Note
    or
    Note: Information
Dependency

• Used in design models. Especially good for representing dependencies that aren’t obvious, such as assumptions.

• There is no notation for indicating why Person depends on Address. I use notes to add the information.
Association Class

- Useful when thinking of an association as an object.
- Could use an n-ary relation.

```
Husband 1..* married 1..* Wife
```

```
Marriage Information

date : Date
temple : Temple
```
N-ary Associations

- In UML the association constraints are binary functional dependencies.
- They tried to extend that idea to n-ary associations. It didn’t work well.
- Here the 4 association constraints mean
  - For a given Husband, Wife, and Temple in the association there are 1 or more Dates.
  - For a given Husband, Wife, and Date in the association there is 1 and only 1 Temple.
  - For a given Husband, Date and Temple in the association there is 1 and only 1 Wife.
  - For a given Wife, Date, and Temple in the association there is 1 and only 1 Husband.
- Because this is awkward it is not used as much as it could be.
- If we interpret the constraints as participation constraints (see ORM, and OSA) then the semantics are much easier to understand.

Diagram:
- Husband
- Date
- Wife
- Temple

A Husband married a Wife on a given Date at a particular Temple
No arrows; info can flow in both directions

Aggregation – Order class contains OrderDetail classes. Could be composition?
Class Diagram

Example

{A Person is always associated with a Floor or an Elevator, but never both at the same time}

{An Elevator has a button for each Floor}

{A Floor has one Down button and one Up button}
Class Diagram Example

Source: Booch et. al., The UML User Guide