CS 247: Software Engineering Principles

UML Modelling

Agenda:  UML class diagrams
         UML object diagrams
         UML sequence diagrams

         (Electronic text available from UW Library Web site)
         www.uml.org
A software model is an abstraction of a software system's code (already developed or to-be-developed).

Analogous to construction plans

Software Modelling
Unified Modeling Language (UML)

UML - A collection of notations for representing different views of a software design.

Structural Diagrams
- Class diagram
- Component diagram
- Composite structure diagram
- Deployment diagram
- Object diagram
- Package diagram

Behaviour Diagrams
- Activity diagram
- State diagram
- Use case diagram

Interaction Diagrams
- Communication diagram
- Interaction overview diagram
- Sequence diagram
- Timing diagram
A box represents a class and defines:

- class name
- set of attributes (data fields, types), initial values
- set of operations (routines, signatures)

### Customer

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
</tr>
<tr>
<td>address</td>
<td>string</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer()</td>
<td>name() : string</td>
</tr>
<tr>
<td></td>
<td>address() : string</td>
</tr>
<tr>
<td></td>
<td>addressIs( newaddr : string ) : void</td>
</tr>
</tbody>
</table>
Abstraction in Classes

Classes can be expressed at different levels of abstraction.

<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: string [1] {readOnly}</td>
</tr>
<tr>
<td>address: string</td>
</tr>
<tr>
<td>+ Customer( name:string );</td>
</tr>
<tr>
<td>+ name() : string {query}</td>
</tr>
<tr>
<td>+ address() : string {query}</td>
</tr>
<tr>
<td>+ addressIs( newaddr : string ) : void</td>
</tr>
</tbody>
</table>

Customer

<table>
<thead>
<tr>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
</tr>
<tr>
<td>birthdate</td>
</tr>
</tbody>
</table>

KEY:

+ public
− private
# protected
static
pure virtual
An **association** between two classes indicates that there exists a physical or conceptual **link** between objects of those classes.

- **Person**
  - firstname : string
  - lastname : string

- **Rental Agreement**
  - driver
  - customer

- **Vehicle**
  - manuf
  - model
  - color
  - VIN

- **Date**
  - day
  - month
  - year

- **Association name**

- **Role names**

---

**Diagram:**

- Rental Agreement —> Person
  - driver
  - customer
- Rental Agreement —> Vehicle
  - rents
  - rented car
- Date
  - start date
  - end date
- Association name
- Role names
Multiplicities

Multiplicity annotations constrain the number of allowable links in an association.

- For each object $x$ of class $X$, there must be at least $p$ links of association $R$ linking $x$ to object of class $Y$; and
- For each object $y$ of class $Y$, there must be between $m$ and $n$ links of association $R$ linking $y$ to object of class $X$.

No annotation means that the multiplicity is unspecified.
Implementing Associations

A → B
0..1

A → B
1

A → B
5

A → B
*  

A ← B
1

*
A class association represents link attributes
- properties of the link, because they cannot be attributed to either of the end objects

```
Person       Race
contestant   *    *

Performance
time : Time
place : int
```

This is an association class, models data and relationships that are associated with a pair of objects
Aggregation is a “part-of” relation between an aggregate (collection) and its members.

- part can be a member of more than one aggregate e.g., students can be members of more than one class roster
- part has an identity outside of the aggregate
Composition is a stronger “part of” relation between a composite object and its components:

- a part does not exist without its composite
- a part belongs to at most one composite
- the composite is responsible for creating, destroying members
On an iPod, songs (and other media) are **physically** stored in a Library, and Playlists are **virtual** collections of songs.

Another Example

Diagram:

```
Library
  1
  * 1
  * 1
Song
  * 1
{ordered}
  * 1
  Playlist
```
Generalization

The UML uses the term **generalization** for the **subtype relationship** between a base class and its derived classes.

- Every member of a derived class is a member of its base class.
- Attributes and associations of the base class are attributes and associations of the derived class.

```
Transcript
- GPA : float
+ calcGPA()

Math Transcript
+ calcGPA()

Engineering Transcript
+ calcGPA()

Student
- name : string
- ID : int
```
The first round, or group stage, is a round-robin competition between thirty-two teams divided into eight groups. The two best teams of each group will progress to the knock-out stage.

The ranking of teams in each group will be based on:

1. Points in all group matches
2. Goal difference in all group matches
3. Goals scored in all group matches
4. Points in matches between tied teams
5. Goal difference in matches between tied teams
6. Goals scored in matches between tied teams
7. Drawing of lots

In the knockout stage there will be four rounds, each eliminating the losers. The four rounds are: the round of 16, quarter-finals, semi-finals, and the final (plus a play-off for third place). A draw in the knockout stages will be followed by two 15-minute periods of extra time to determine a winner. If the teams are still tied, a penalty shootout will be held.
An object model is a run-time instance of a class model.

- Every object is an instantiation of a specific class.
- Every link is an instantiation of a specific association.
A **UML Sequence Diagram** is a graphical model of communication events between objects, as exhibited in one execution trace.
Some UML Drawing Tools

Can use any UML modelling or drawing tool that you would like.
  - Must be able to output PDF files.

• Visio

• OmniGraffle (Mac only)

• UMLet (open source, Windows / OS X / Linux)
  http://www.umlet.com/
Take Aways

**Recognition**
- Recognize class diagrams: class, attribute, association, association name, rolename, association class.
- Recognize rules for designating data as objects or attributes.

**Comprehension**
- Distinguish between aggregation and composition.
- Understand program behaviour described as a sequence diagram.

**Application**
- Model a program's set of classes as a class diagram.
- Use multiplicities to constrain allowable instances of a class diagram.