Class & Object Diagrams

Software Design Methodology
UML Class Diagrams

- The heart of the UML
- Describes the classes in the system and the relations among them
- Supports most OO concepts
  - associations
  - aggregation
  - inheritance (including multiple)
Classes

- Classes are denoted by rectangles.
  - Attributes
  - Operations
  - Responsibilities

<table>
<thead>
<tr>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute: type</td>
</tr>
<tr>
<td>operation</td>
</tr>
<tr>
<td>operation(parameters)</td>
</tr>
<tr>
<td>responsibility</td>
</tr>
</tbody>
</table>
Classes

- **Attribute**: A named property of a class that describes a range of values that instances of the property may hold.

- **Operation**: Implementation of a service that can be requested.

- **Responsibility**: A contract or an obligation of a class. As the model is refined the responsibilities are transformed into attributes and operations.
Object

- **Object** - a concept, an abstraction, a thing meaningful to the domain
  - Joe Smith  Person
  - Lassie  Dog
  - flight #713  Flight
  - the top window  Window
- **Object Class** - a collection of all objects having the same set of features.
- Each object has a unique *identity* and is an *instance* of its object class.
Class Diagrams vs. Object Diagrams

Person

<table>
<thead>
<tr>
<th>name</th>
<th>employeeID</th>
<th>title</th>
</tr>
</thead>
</table>

p:Person

name=“Yahoo”
employeeID=1212
title=“VP of production”

Joe Smith:Person
Relationships

- A relationship is a connection among things.
- Common relationships: Dependencies, Associations, Generalization and Aggregations.
Dependency

A dependency states that a change in specification of one thing may effect another thing that uses it.

- **CourseSchedule**
  - add(c:Course)
  - remove(c:course)

- **Window**
  - open()
  - close()

- **Student**
  - grade
  - CalculateAverage(grade)

A class takes an object of another class as a parameter.

A class accesses an object of another class.

A class calls a class operation in another class.
An association is a structural relationship that specifies that objects of one thing are connected to objects of another.

**Binary association**

<table>
<thead>
<tr>
<th>Country</th>
<th>Has-capital</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: string</td>
<td></td>
<td>Name: string</td>
</tr>
<tr>
<td>Population: integer</td>
<td></td>
<td>Population: integer</td>
</tr>
<tr>
<td>Area: Km²</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Association Name

- Printed at the center of the line
- Describes the nature of the relationship (usually a verb)
- A direction triangle can be added
Association Multiplicity

- Used when it is important to state how many objects may be connected across an instance of an association.

- When a number is stated at an end of an association it is specified that for each object at the class at the opposite end there must be that many objects at the near end.

- An * denotes many - which can be any number between zero and infinity.
At least one employee in a department
An employee belongs to exactly one department
An employee has zero or one office
An office is assigned to a number of one up to 10 employees
Multiplicility symbols
(participation constraints)

<table>
<thead>
<tr>
<th>Line</th>
<th>0,2..*</th>
<th>Intersects</th>
<th>*</th>
<th>Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: string</td>
<td></td>
<td></td>
<td></td>
<td>Name: string</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* ▶ Is Endpoint Of 2</td>
<td></td>
<td>X-Coordinate: integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y-Coordinate: integer</td>
</tr>
</tbody>
</table>
Association Classes

- Denoted as a class attached to the association
- Specify properties of the association
- Does not belong to any of the connected class

<table>
<thead>
<tr>
<th>Car</th>
<th>*</th>
<th>Owner of</th>
<th>1..*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reg. Num.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Year</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Person

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
</table>

Year Purchased
Association Attributes

Person
- Name: string
- ID: string
- Address: string

1..* Works-for *

Company
- Name: string
- Address: string
- salary: float
- job: string

Student
- Name
- ID
- Faculty

140 submit 4

Assignment
- Number
- Subject
- Submission date

Grade
Association Role Names

- Names may be added at each end of the association
- Provide better understanding of the association meaning
- Especially helpful in self-associated classes
Role Names, Constraints and Qualifiers

**Role**

User --- 1 --- owner
* --- authorized user --- *

Directory --- 0..1 --- container
* --- contents

**Ordering**

Window --- * {ordered} --- 1 --- Screen
visible-on

**Qualifier:**

an attribute that reduces the association’s multiplicity.

Directory --- * --- 0..1 --- File
file name
# Classes versus Objects

## Class Diagram

<table>
<thead>
<tr>
<th>Classes</th>
<th>name</th>
<th>attributes</th>
<th>operations</th>
<th>responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Association</td>
<td>name</td>
<td>multiplicity</td>
<td>role</td>
<td></td>
</tr>
<tr>
<td>Association Classes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Object Diagram

<table>
<thead>
<tr>
<th>Objects</th>
<th>values for attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link</td>
<td></td>
</tr>
<tr>
<td>Link Object</td>
<td></td>
</tr>
</tbody>
</table>
Class Diagram Example

Reader
- Name
- ID
- Address

* (ordered)

reserved

Book
- Code
- Name
- Edition
- Author

0..1 holds

Return date

Copy
- Code
- Shelf location
- Status
- Loan duration

*
Object Diagram Example

357:Reader
Name="Joe"
ID=357
Address="Jerusalem"

57168-1:Copy
Code=57168-1
Shelf location=681.3.06
Status="at loan"
Loan duration=2 weeks
Return date=16.11.00

482:Reader
Name="Michel"
ID=482
Address="Haifa"

57168-2:Copy
Code=57168-2
Shelf location=681.3.06
Status="at loan"
Loan duration=1 month
Return date=24.11.00

936:Reader
Name="David"
ID=936
Address="Haifa"

UML:Book
Code=57168
Name="The Unified Modeling Language User Guide"
Edition=1
Author="Booch, Rumbaugh, Jacobson"

reserved

of

of

20
Aggregation

- “whole-part” relationship between classes
- Assemble a class from other classes
  Combined with “many” - assemble a class from a couple of instances of that class
- May be replaced by associations...

```
Document
  text font
  pages

* Paragraph
  *

* Sentence

* Picture
```

21
Aggregation (whole-part; part-of; and)
Constraint

A functional relationship between entities (objects, classes, attributes, links, associations) of an object model.

- Employee
  - salary

  \{salary < boss.salary\}

- Window
  - length
  - width

  \{0.8 \leq \text{length/width} \leq 1.5\}

- Company
  - Office

  \{ordered\}

- Person

- Committee

\{subset\}

\{ordered\}
Constraint (Cont.)

Portfolio

\{secured\}

BankAccount

\{xor\}

Corporation

\{secured\} = Across the association communicated is encrypted.

Person

\{female, male\}

\{self.wife.gender=female and self.husband.gender=male\}

0..1

wife

husband
Inheritance (Generalization)

- Denoted by a triangle - connects one superclass to many subclasses
- Multiple inheritance is allowed
  - a subclass may be connected to more than one superclass
  - not recommended - hard to understand, hard to implement
- Four applicable standard constraints:
  - Complete
  - Incomplete
  - disjoint
  - overlapping
Generalization (gen-spec; kind-of; or)

- Figure
  - color
  - center position
  - pen type
- Move
- Rotate
- Display

**0 Dimensional**

**1 Dimensional**
- Orientation
- Scale

**2 Dimensional**
- Orientation
- fill type
- fill
- Scale

**subclasses**

- Line
  - Endpoints
  - Display
- Arc
  - Radius Angle
  - Display
- Spline
  - Control pts
  - Display

**superclass**
Multiple Inheritance
Abstract Class - a class that has no direct instances
Recursive Aggregation

Program

Block

Compound statement

Simple statement
Recursive Terms (Multiple Binary term use)

Term

First Operand

Expression

Binary Operator

Variable

Name

Constant

Value

Second Operand
\[
\frac{(x+y/2)}{(x/3+y)}
\]
Recursive Terms
(Unary & Binary & Single Term Use)
Abstract Operations

The origin class defines
the protocol:

• semantics;
• attribute types;
• number and type of
  arguments for methods
Generalization as Extension and Restriction

Extension: addition of new features.
(e.g. Person vs. Student)

Restriction: constraining ancestor attributes.
(e.g. nullifying some attributes)
Overriding Operations

- Tension between use of inheritance for abstraction vs. implementation reuse.
- Reasons for overriding:
  - Extension
    (e.g. Person has Report_Marriage; Employee has Report_Marriage that extends the code of Report_Marriage)
  - Restriction
    (e.g. Subclass operation use some of superclass sub-operations)
Reasons for overriding (cont.)

- Optimization
  e.g. IntegerSet has findMax; SortedIntegerSet has another implementation for findMax.

- Convenience
  Look for a similar class and use it as superclass. This is ad hoc use which is semantically wrong. Better: define a third common class, from which both will inherit.

Rule for method overriding:

All methods that implement an operation must have the same protocol.