Object-Oriented Modeling

One paradigm of development

The OO Solution

- The OO model closely resembles the problem domain
  - Base your model on the objects in the problem domain

- Iteratively refine the high-level model until you have an implementation
  - Attempt to avoid big conceptual jumps during the development process
Objects

Attributes and Operations

Person objects

Card objects

Person class

Attributes
- name
- age
- height
- weight

Operations
- move
- change-job

Card class

Attributes
- height
- width
- id-number

Operations
- issue
- change
Characteristics of Objects

- **Identity**
  - Discrete and distinguishable entities

- **Classification**
  - Abstract entities with the same structure (attributes) and behavior (operations) into classes

- **Polymorphism**
  - The same operation may behave differently on different classes

- **Inheritance**
  - Sharing of attributes and operations based on a hierarchical relationship

The Class Diagrams
Objects

- Something that makes sense in the application context (application domain)
  - J.Q. Public
  - Joe’s Homework Assignment 1
  - J. Q. Public’s drivers license
- All objects have identity and are distinguishable
- NOT objects
  - Person
  - Drivers license

Classes

- Describes a group of objects with similar properties (attributes), common behavior (operations), common relationships to other classes, and common semantics
- Person
  - J. Q. Public
  - Joe Smith
  - D. Q. Public
- Card
  - Credit card
  - Drivers license
  - Teller card
Class Diagrams

Class with attributes

Objects with values

Class diagram

Instance diagram

Class: Person

Person
age: integer

D. Q. Public: Person

age= 32

J. Q. Public: Person

age= 35

Objects have an identity

Do not explicitly list object identifiers

SSN OK!

Examples

Person

name: String
age: integer

Card

height: integer
width: integer
thickness: integer
id-number: integer

SSN OK!
**Operations and Methods**

- Transformation that can be applied to or performed by an object
- May have arguments

<table>
<thead>
<tr>
<th>Card</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>height: integer</td>
<td>height: integer</td>
</tr>
<tr>
<td>width: integer</td>
<td>width: integer</td>
</tr>
<tr>
<td>thickness: integer</td>
<td></td>
</tr>
<tr>
<td>id-number: integer</td>
<td>rotate(angle: integer)</td>
</tr>
<tr>
<td></td>
<td>move(x: integer, y: integer)</td>
</tr>
<tr>
<td>issue()</td>
<td></td>
</tr>
<tr>
<td>revoke()</td>
<td></td>
</tr>
</tbody>
</table>

**Object Notation - Summary**

<table>
<thead>
<tr>
<th>Class name</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute-1 : data-type-1 = default-value-1</td>
</tr>
<tr>
<td>attribute-2 : data-type-2 = default-value-2</td>
</tr>
<tr>
<td>attribute-3 : data-type-3 = default-value-3</td>
</tr>
<tr>
<td>operation-1(argument-list-1) : result-type-1</td>
</tr>
<tr>
<td>operation-2(argument-list-2) : result-type-2</td>
</tr>
<tr>
<td>operation-3(argument-list-3) : result-type-3</td>
</tr>
</tbody>
</table>
Associations

- Conceptual connection between classes
  - A credit card is issued-by a bank
  - A person works-for a company

Class diagrams

Instance diagram

Associations are Bi-directional

- There is no direction implied in an association (Rumbaugh - OMT)
Associations Have Direction

- Unified adds a direction indicator
  - Inconsistently used

```
Country
  name

Has-capital

City
  name
```

```
Person
  name

Is-issued

Drivers-license
  lic.-number: integer
```

Multiplicity

- One person holds one credit card
- One object can be related to many objects through the same association
- One person can hold zero or more credit cards
### Multiplicity (Cont.)

- One person can hold zero or more credit cards (0..*).
- Each card has zero or one holder (0..1).

**Diagram:**
- Person
  - name: String
  - age: integer
  - 0..1 Holds

- Credit-card
  - card-number: integer
  - 0..*

- Person: JQPublic
  - name = J. Q. Public
  - age = 35

- Person: DQPublic
  - name = D. Q. Public
  - age = 32

- Credit-card: Card789
  - card-number = 123 456 789

- Credit-card: Card123
  - card-number = 111 222 333

- Credit-card: Card456
  - card-number = 444 555 666

### Higher order associations

- Ternary association
  - Project, language, person
- Seldom needed (and should be avoided)

**Diagram:**
- Language
  - 1..*

- Project
  - 1..*

- Person
  - 1..*

- Note: hexagons should be rectangles to represent instances.
**Link Attributes**

- Associations can have properties the same way objects have properties.

**How to represent salary and job title?**

**Use a link attribute!**

**Folding Link Attributes**

**Why not this?**

Salary and job title are properties of the job **not** the person.

**In this case, a link attribute is the only solution.**
Role Names

- Attach names to the ends of an association to clarify its meaning

Aggregation

- A special association, the is-part-of association
  - A sentence is part of a paragraph (a paragraph consists of sentences)
  - A paragraph is part of a document (a document consists of paragraphs)
Aggregation (Cont.)

- Often used in parts explosion

**Composition**

Cheng/CSE 435: Software Engineering
Generalization and Inheritance

- The is-a association
  - Cards have many properties in common
  - Generalize the common properties to a separate class, the base-card
  - Let all cards inherit from this class, all cards is-a base-card (plus possibly something more)

![Diagram of Generalization and Inheritance]

Example

![Diagram of Example]
Aggregation Versus Association

- Can you use the phrase is-part-of or is-made-of
- Are operations automatically applied to the parts (for example, move) - aggregation
- Not clear what it should be......

Aggregation Versus Inheritance

- Do not confuse the is-a relation (inheritance) with the is-part-of relation (aggregation)
- Use inheritance for special cases of a general concept
- Use aggregation for parts explosion
Recursive Aggregates

- A recursive aggregate contains (directly or indirectly) an instance of the same kind of aggregate

Class diagram Metamodel I
Class diagram Metamodel II

Use Case Metamodel I

uu: use case association relationship
_i: includes
_e: extends
_g: generalization
_aa: actor relationship

Cheng: CSE 435: Software Engineering

Use Case Metamodel II

Data flow diagram

Compiler: BNF grammar

Lexical analyzer

Semantic analyzer

Code generator

Code optimizer

Programmer

source program

token stream

abstract syntax tree

object code

Cheng:CSE 435: Software Engineering
Possible Metamodel for DFD

Revised Metamodel for DFD
Object Modeling Summary

- Classes
  - Name
  - Attributes
  - Operations
- Associations
  - Roles
  - Link attributes

- Aggregation/Composition
- Inheritance