Object-Oriented Design: Topics & Skills

- Rational Unified Process
- Unified Modeling Languages (UML)
  - Provide a range of notations that can be used to document an object-oriented design.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Design</th>
<th>Structure</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Class Diagrams</td>
<td>Package Diagrams</td>
<td>Use Case Diagrams</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sequence Diagrams State Diagrams Activity Diagrams</td>
</tr>
</tbody>
</table>

- Design Patterns
- Reuse solutions, rather than solve every problem from first principles.
The History of UML

- Developed by the following three people at the Rational Software Corporation in 1996 (UML 0.x).
  - Grady Booch (the Booch method)
  - Ivar Jacobson (the OOSE method)
  - James Rumbaugh (the OMT method)
- The Object Management Group (OMG) started the process of UML standardization in 1997 (UML 1.x).
- The current version is UML 2.4.x.
UML Diagrams

Diagram

Structure Diagram
  - Class Diagram
  - Component Diagram
  - Object Diagram
  - Profile Diagram
  - Composite Structure Diagram
  - Deployment Diagram
  - Package Diagram

Behavior Diagram
  - Activity Diagram
  - Use Case Diagram
  - Interaction Diagram
  - State Machine Diagram
  - Sequence Diagram
  - Communication Diagram
  - Interaction Overview Diagram
  - Timing Diagram

Notation: UML
UML Class Diagram

• Definition: a class diagram describes the types of objects in the system and the various kinds of static relationships that exist among them.

• Essential elements
  • Classes
    • Attributes
    • Operations
  • Associations
    • Generalization
    • Aggregation
    • Composition
An Example of UML Class Diagram
Classes

• Identity

• Attributes
  • \textit{visibility name: type multiplicity = default \{property-string\}}

• Operations
  • \textit{visibility name (parameter-list) : return-type \{property-string\}}
  • parameter-list: \textit{direction name: type = default}
Attribute Syntax

- visibility **name**: type multiplicity = default {property-string}
- optional visibility: + public, - private, # protected
- name: the name of this attribute
- optional type: data type of this attribute
- optional multiplicity: optional [0..1], mandatory [1]
- optional default: initial value of attribute
- optional property string: OCL, e.g. ordered, readonly

Examples
- firstName, -lastName: String
- -age: int {>=0}
- - name: String [1] = "Untitled" {readOnly}
Property String

• Any information that cannot be expressed in the diagram notation can be included as text.
  • Most of them are constraints.

• Can be
  • Informal English
  • Object Constraint Language

• The only rule
  • Put them inside braces ({})
Operation Syntax

• visibility **name** *(parameter-list) : return-type {property-string}*
  • optional visibility: + public, - private, # protected
  • name: the name of this operation
  • optional parameters-list, parameters to this operation (separated by a comma): *direction name: type = default*
  • optional direction: in, out, inout
  • name
  • optional type
  • optional default
  • optional type: return type of operation
  • optional property string
Operation Syntax

- Examples
  - `getFirstName() {query}`
  - `+getMiddleName(): String`
  - `-setLastName(name: String) {sequential}`
  - `+paintPortrait(inout c: Canvas, subject: Person)`
Associations

• Relationships between instances of classes
  • E.g. a person works for a company; a company has a number of offices.

• Roles
  • Name labeled on an association end.
  • If no label, an association end can be named after the target class.

• Multiplicities
  • Indicates how many objects may participate in the given relationship.
  • 0, 1, *, 0..1, 1..*, 5..6, and so on.
  • Would be realized through Arrays, Sets, Lists, and so on.

• Navigability
  • Bidirectional: each class references the other.
  • Unidirectional: A knows B, but B doesn’t know A.
  • No arrow heads: means either “bidirectional” or “not specified”.
Associations

• Associations are another way of representing attributes.

```
Order
+ dateReceived: Date [0..1]
+ isPrepaid: Boolean [1]
+ lineItems: OrderLine [*] {ordered}
```
Question

• When to use attributes versus association lines?

• Data type refers to objects for which unique identity is not important.

• Common data types are primitive-oriented types, such as Boolean, Date, Number, Character, String, Time, Address, Color, Phone Number, Social Security Number, Zip code.

• Use the attribute for data type objects and the association line notation for others. Both are semantically equal, but showing an association line to another class box in the diagram gives visual emphasis.
Variations of Associations

• Generalization

• Inheritance relationship between instances of two classes. Correspond to the Java keyword "extends" (is-a).

• Aggregation

• Special associations that represent ‘part-whole’ relationships (part-of) - parts can join the whole and later leave; one part could be part of more than one whole.

• Composition

• A strong kind of aggregation (has-a) - parts are created with the whole and stay with exactly one whole until both are destroyed together.
Generalization

• Generalization corresponds to "extends"
  • Generalizations are drawn with a solid line and a white triangular arrow touching the superclass.

• It is common practice to arrange the diagram so that:
  • Generalization arrows point upward.
  • If one class has many subclasses, the Generalization arrows overlap.
Aggregation

- Aggregations are special associations that represent ‘part-whole’ relationships.
- The ‘whole’ side is often called the assembly or the aggregate.
- This symbol is a shorthand notation association named isPartOf.
Composition

• A composition is a strong kind of aggregation.
  • If the aggregate is destroyed, then the parts are destroyed as well.
Some Mnemonics

- Generalization = is-a
- Aggregation = part-of
- Composition = has-a

Examples
- Toaster is-a Appliance
- Member part-of club
- Toaster has-a slot
Advanced Concepts of Class Diagrams

- Stereotypes and profiles
  - The core extension mechanism of the UML
  - Stereotypes are usually shown in text between guillemots (<<>>)
  - A profile is defined by a set of stereotypes, a set of related constraints (in OCL), and a set of tagged valued.

- Object Diagram (Instance Diagram)
  - A snapshot of the objects in a system at a point in time
  - The names are underlined. Each name takes the form: instance name : class name.

- Interfaces and Abstract Classes
Example: Class to Object Diagrams

```
Person   owns   Toaster
0..1     n

Cinderella  owns  ProctorSilex
:Person     :Toaster
```
Interfaces and Abstract Classes

• Question: what is different between abstract classes and interfaces in an object-oriented programming language (e.g. Java)?
Interfaces and Abstract Classes

• UML Interface
  • A class that has only public operations with no method bodies or attributes.

• Representation
  • Italicize the name of interfaces or abstract classes.
  • Use the \{abstract\} constraint for abstract classes (optional) and the <<interface>> stereotype for interfaces.

• Realization
  • Relationship between an implementation class and an interface.
  • Drawn with a dashed line and a white triangular arrow touching the interface.
Example
To summarize:

- This class is associated with this class.
- You can navigate from this class to this class.
- This class inherits from this class.
- This class is a realization of this interface.
- These classes compose without belonging to this class.
- These classes compose and are contained by this class.
Package Diagrams

• Package is a grouping construct.
  • Most commonly used for class diagrams, but can be used with any UML diagram or elements.
  • Used to create a hierarchy or higher level of abstraction.
  • Corresponds to package in Java.

• Each package represents a namespace.
  • Like Java, can have classes with same name in different packages.
Representing Packages

- Contents listed in box
- Contents diagramed in box

- Fully qualified package name
- Nested packages
- Fully qualified class name
References

• Software Engineering

• UML Modeling
Reminder

• Lab #2 is on next Tuesday.
• Assignment #2 will be out after the lab.
• Assignment #1 is due next Monday midnight.