UML Class Diagrams

Kirk Knoernschild
QWANtify, Inc.
www.qwantify.com
kirk@qwantify.com
Outline

➔ UML Introduction
  • Class Diagrams
  • Java Mapping
What is UML?

- Unified Modeling Language (UML)
- Specification, Visualization, Construction, & Documentation of software systems.
- Precise and Unambiguous.
- It is *not* a software process.
  - Does have focus on software process.
- Extensible Language
  - Stereotypes, Constraints, Tagged Values
UML’s Roots

- OOSE
- OMT
- Booch Method
- Schlaer-Mellor
- Coad-Yourdon
- Wirfs-Brock
- Many Others
  (Fusion, SOMA, MOSES, and on and on…)

Patterning Solutions for your Success
Software Process

- UML and Process are loosely coupled
- UML customized to your process
  - Architecture-Centric
  - Use Case Driven
  - Iterative and Incremental
- Diagrams work together to form views.
### Process and UML

<table>
<thead>
<tr>
<th>Phase</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>Workers perform Activities and produce Artifacts.</td>
</tr>
<tr>
<td>Analysis &amp; Design</td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
</tr>
<tr>
<td>Deployment</td>
<td></td>
</tr>
<tr>
<td>Process and UML</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td></td>
</tr>
</tbody>
</table>

**Requirements**

**Analysis & Design**

**Implementation**

**Test**

**Deployment**

---

**Iteration #n**

Workers perform Activities and produce Artifacts.

---

**Release**
# Process and UML

<table>
<thead>
<tr>
<th></th>
<th>Iteration #n</th>
<th>Iteration #n + 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis &amp; Design</td>
<td></td>
<td>Workers perform Activities and produce Artifacts.</td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deployment</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Workers perform Activities and produce Artifacts.

Incremental Release
Process and UML

- Requirements
- Analysis & Design: Workers perform Activities and produce Artifacts.
- Implementation
- Test
- Deployment

Iteration #n

Release

Phase

Incremental Release

Class Diagrams

Patterning Solutions for your Success
UML Diagrams

- Object Diagram
- Use Case Diagram
- Timing Diagram
- Interaction Overview Diagram
- Class Diagram
- Deployment Diagram
- State Diagram
- Use Case Diagram
- Activity Diagram
- Component Diagram
- Sequence Diagram
- Package Diagram
- Communication Diagram

Patterning Solutions for your Success
Core Elements
Diagrams

Individual Diagrams

Core Elements
Views

Class Diagrams
Sequence Diagrams

Individual Diagrams

Core Elements
View of Software

Model

Views

Class Diagrams
Sequence Diagrams

Class Diagrams

Individual Diagrams

Core Elements
Diagram Categories

• Behavioral
  – Focus on the dynamic aspect of a system.
  – Allow us to map out a system’s behavior.

• Structural
  – Focus on the static aspects of a system.
  – Allow us to focus on the system’s structure.
Diagram Composition

- Diagrams are composed of fundamental modeling elements.
- Fundamental elements are reused across many diagrams.
- Diagrams are used together to form complete models.
  - Different views of our system.
Outline

- UML Introduction
- Class Diagrams
- Java Mapping
Notes

- Graphically represented as a dog-eared rectangle.
- Can contain a textual description which documents a portion of your model.
- Similar to a comment in code.
- Can be attached to any, or no, modeling element.

This is a note...

I can attach it to any element using a dashed line.
Class

- Represented with a rectangle.
- Must have a name.
- Can have optional compartments representing attributes and operations.

Student

Professor
Class Compartments

• Attributes represent a property of an object.
• Operations represent the behavior of an object.

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>-studentID : Integer</td>
<td>-maxStudentsAllowed : Integer</td>
</tr>
<tr>
<td>-studentName : String</td>
<td>+getNumStudents() : Integer</td>
</tr>
<tr>
<td>+registerForClasses()</td>
<td>+getStudentsAllowed()()</td>
</tr>
</tbody>
</table>

+ → public
# → protected
- → private
Relationships

- Association
  - Aggregation
  - Composition
- Dependency
- Generalization
- Realization
Association

• Structural relationship between two classes.
• “Uses a” relationship between two classes.
• Exists at the instance level.
• Classes at the same conceptual level.

Department  

Professor
Navigability

- Specification of object references in an associative type of relationship.
- Bi-Directional and Uni-Directional

Department can call methods on Professor, but Professor cannot call methods on Department.
• Specifies number of instances one class has of another.

A Department may consist of many Professors.

A Professor can belong to many Departments.

A Schedule belongs to one and only one Student.

A Student can have many schedules.
Multiplicity

- Exactly One
  - `1`
- Zero or More
  - `0..*`
- Zero or One
  - `0..1`
- One or More
  - `1..*`
- Range
  - `2..4`
Names and Roles

• Associations are typically assigned names or roles to help identify the semantic relationship.

• Roles map to the variable names in code.
Aggregation

“Has a” relationship between two classes. Exists at the instance level. Whole/Part relationship modeling different conceptual levels.
Composition

- “Has a” relationship between two classes.
- Exists at the instance level.
- Whole/Part relationship modeling different conceptual levels.
- Whole make “lifetime decision” of part.
- Part is not shared across instances.

```
Student 1 Schedule 0..*
```
Dependency

- “Uses a” relationship between two classes.
- Local, Global, or Parameter scope to an object.
- Always a 1:1 relationship.
Generalization

- Relationship between a general and specific type of entity.
- “Is a special kind of” relationship.

```
Student

FullTimeStudent

PartTimeStudent
```
Realization

- Specification of a contract in one entity that is carried out by another entity.
- Used to model interface inheritance.
Common Class Diagrams

- Diagrams to depict the classes responsible for fulfilling responsibilities of a Use Case.
- Diagrams to illustrate the relationships between packages in our system.
- Diagrams to document the structural relationships that exist amongst classes contained in the packages.
- Any other class diagram to satisfy need.
Register for Courses VOPC

Diagram:
- MainForm
- ScheduleForm
- RegCont
- CourseCat
- Schedule
- Offering
Design View

• Class diagrams
  – Classes and Packages
• Interaction diagrams
• Software Architecture Document
• Use Case Realizations
  – Translation of Use Case Flow of Events to collaborations amongst objects.
  – Consists of Interaction diagrams and View of Participating Classes (VOPC).
The Lifecycle

Focus on architectural stability

Use Case

Requirements documentation

Focus on Allocation of Behavior, identification of responsibilities, and structure amongst classes which work together to fulfill responsibilities of a Use Case.

Use Case Realization

«Trace»

Layer1

Layer2

Layer3

Development View

Components

«Contain»

Process View

Threads and Processes

«Trace»

Physical View

Physical Nodes

Allocation to appropriate layers

Behavioral

A

B

C

do() -> do2()

Structural

View of Participating Classes

© Kirk Knoernschild 2000
Outline

• UML Introduction
• Class Diagrams
→ Java Mapping
Language Mappings

• UML maps very well to Java.
• Modeling tools engineer code.
  – Forward engineering.
  – Reverse engineering.
• UML Profiles define concrete mappings.

UML and Java Reference Card