3.0 Object-Oriented Modeling Using UML

Subject/Topic/Focus:
- Introduction to UML

Summary:
- History of OOAD leading to UML
- UML Diagrams: Overview
- UML Models in the Objectory Software Development Process

Literature:
- [Fowler99]

Evolution of OO Development Methods

70/80ies
- structured programming
  - structured design
    - structured analysis

80/90ies
- object-oriented programming
  - object-oriented design
    - object-oriented analysis
History of OOAD leading to UML

1970  First object-oriented languages (Simula-67, Smalltalk).

1980  More than 50 different OOAD languages cause the users trouble to find complete and appropriate tools.

      Booch '93, OOSE (Jacobson), OMT-2 (Rumbaugh)

1995  Unification, UML 0.9 by Booch, Rumbaugh

1997  Standardization, UML 1.1 by Booch, Rumbaugh, Jacobson

1999  Object Management Group (OMG) adapts UML as OOAD standard

1999  Evolving standard in version 1.3

History of UML

UML 1.3
UML 1.1
UML 0.9 & 0.91
Unified method 0.8
Booch '93
OMT - 2
Other methods
Booch '91
OMT - 1
OOSE
Industrialization
Standardization
Unification
Fragmentation
The Unified Modeling Language

- Booch and Rumbaugh started working towards a Unified Modeling Language (UML) in 1994 under the auspices of Rational Inc.

- UML only offers a model notation, not a methodology for how to do good modeling.

- UML is used by the development method Objectory (Jacobson at Rational).

- UML was proposed by Rational Inc. and by Hewlett-Packard as a standard for object-oriented analysis and design and was adopted by the OMG.

- Vendors modify their CASE tools to make them consistent with UML.

---

UML Diagrams (1)

<table>
<thead>
<tr>
<th>Diagram Type</th>
<th>Nodes</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Diagrams</td>
<td>Actor, Use (case)</td>
<td>Involvement, Extension, Inclusion, Generalization</td>
</tr>
<tr>
<td>Class Diagrams</td>
<td>Class</td>
<td>Association, Generalization</td>
</tr>
<tr>
<td>Interaction Diagrams</td>
<td>Object</td>
<td>Message, Lifeline</td>
</tr>
<tr>
<td>State Diagrams</td>
<td>State, Sub-State</td>
<td>Transition</td>
</tr>
<tr>
<td>Activity Diagrams</td>
<td>Activity</td>
<td>Guard, Synchronization</td>
</tr>
<tr>
<td>Package Diagrams</td>
<td>Package</td>
<td>Dependency</td>
</tr>
<tr>
<td>Deployment Diagrams</td>
<td>Processor, Node</td>
<td>Dependency</td>
</tr>
</tbody>
</table>
Object-Oriented Modeling Using UML

UML Diagrams (2)

Use Case Diagrams
- scenarios
- structures
- interaction sequences between objects

Activity Diagrams
- workflows
- inter-class behavior

Interaction Diagrams
- intra-class behavior

Class Diagrams
- structuring

Package Diagrams
- structuring

State Diagrams

“Unified Process” and UML

Transition
Construction
Elaboration
Inception

Methodology

Software development is a process in **phases**. This process has to follow a **methodology**. Each phase is supported by graphs & diagrams. There are different kinds of documents and various usage of them. **UML** is an essential **language** for diagrams, offering computer support as well as the right patterns for the various stages of refinement and viewpoints.
Object-Oriented Modeling Using UML

Diagrams and Process

Use Case Diagrams

Inception

Elaboration
Diagrams and Process

Class & Package Diagrams

Inception

Elaboration

Object-Oriented Modeling Using UML

Diagrams and Process

Interaction Diagrams (Scenarios)

Inception

Elaboration

Object-Oriented Modeling Using UML
Object-Oriented Modeling Using UML

3.0.13

Object-Oriented Modeling Using UML

3.0.14

Activity Diagrams (Workflow, Interclass Behavior)

Diagrams and Process

Activity 1

Activity 2

Activity 3

Activity 4

Activity 5

Inception

Elaboration

State Transition Diagrams (Intra class Behavior)

Diagrams and Process

State 1

State 2

State 3

State 4

Inception

Elaboration
Texts and Process

Source Code

```java
import java.util.*;

class Shape {
    void draw() {}
    void erase() {}
}

class Circle extends Shape {
    void draw() {
        System.out.println("Circle.draw()" OUTER); }
    void erase() {
        System.out.println("Circle.erase()" OUTER); }
}

public static void main(String args[]) {
    Shape s[] = new Shape[9];
    // Fill up the array with shapes:
    for (int i = 0; i < s.length; i++)
        s[i] = randShape();
    // Make polymorphic method calls:
    for (int i = 0; i < s.length; i++)
        s[i].draw();
}
```

Diagrams and Process

Deployment Diagrams