Overview

- Unified Modeling Language (UML)
  - Background
  - UML diagrams
  - Class diagrams
    - Examples
UML (Unified Modeling Language)

- UML is a modeling language for
  - Specifying
  - Visualizing
  - Constructing
  - Documenting

object-oriented software

Motivation

- Software growing larger & complex
  - Difficult to describe and analyze

- Use UML to help
  - Visualize design of software
  - Provide abstract model of software
Goals

- Provide a software “blueprint”
  - Simple yet clear abstraction for software

- Describe software design
  - Clearly
  - Concisely
  - Correctly

History of UML

- Started in 1994
- Combines 3 leading OO methods
  - OMT (James Rumbaugh)
  - OOSE (Ivar Jacobson)
  - Booch (Grady Booch)
**UML Diagrams**

- UML provides a number of diagrams that
  - Describe a model of all or part of system
  - From a particular point of view
  - With varying level of abstraction
  - Using certain set of notations

**Example UML Diagrams**

- Use case
  - Functional behavior seen by (external) user
- Sequence
  - Dynamic behavior between users and objects
- State
  - Dynamic behavior of objects as finite state machine
- Class
  - Static structure of the classes in system
- Activity
  - Dynamic behavior of a system as a flowchart
Use Case Diagram

- Displays interactions between user and system

- Each use case
  - Provides one or more scenarios
  - Conveys how system should interact with user
  - Shows how to achieve a specific goal

- From perspective of external user (actor)

Use case diagrams represent functionality of system from external user’s point of view.
Sequence Diagram

- Describes interactions among objects
  - For a use case

- A sequence diagram displays
  - Objects participating in use case
  - Order messages passed (methods invoked)

- Notation
  - Columns ⇒ Objects
  - Arrows ⇒ Messages
  - Narrow rectangles ⇒ Activations

Sequence Diagrams show the message sequence
State Diagram

- Displays behavior of system as finite automata

- A finite automata contains
  - Nodes \(\rightarrow\) states of system
  - Edges \(\rightarrow\) transitions between states

State diagram represents system as finite automaton
Class Diagram

- Represents (static) structure of system

A class diagram displays
- Information for class
- Relationships between classes
Class Diagrams

- Information for class contains
  - Name
  - State
  - Behavior

Class Diagram

- Class name is required
- Other information optional
  - State, behavior
  - Types, visibility…
UML Class Diagrams ↔ Java Code

- Different representation of same information
  - Name, state, behavior of class
  - Relationships between classes
- Should be able to derive one from the other

Motivation
- UML ⇒ Java
  - Implement code based on design written in UML
- Java ⇒ UML
  - Create UML to document design of existing code

Java → UML: Clock Example

Java

class Clock { // name
  // state
  int seconds;
  int minutes;
  int hours;
  // behavior
  void start();
  void adjustTime();
  void reset();
}

Java Code

<table>
<thead>
<tr>
<th>Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>seconds:int</td>
</tr>
<tr>
<td>minutes:int</td>
</tr>
<tr>
<td>hours:int</td>
</tr>
<tr>
<td>start()</td>
</tr>
<tr>
<td>adjustTime()</td>
</tr>
<tr>
<td>reset()</td>
</tr>
</tbody>
</table>
Overview

Unified Modeling Language (UML)
- Background
- UML diagrams
- Class diagrams
  - Examples

Class Diagram Notation

UML notation
- Type  \( \Rightarrow \) type name preceded by colon :
- Static  \( \Rightarrow \) field / method is underlined
- Abstract  \( \Rightarrow \) class / method is italicized
- Interface  \( \Rightarrow \) labeled with angled brackets ‹‹ I ››
- Visibility  \( \Rightarrow \) prefix symbol
  - + public
  - – private
  - # protected
  - ~ package
Java → UML : Clock Example

Java
class Clock {
    // name
    // state
    private int seconds;
    private int minutes;
    private int hours;
    // behavior
    public void setTime( );
    public void adjustTime(int value);
    public void reset( );
}

Java Code

Class Diagram

Relationships Among Classes

UML class diagrams
- Can depict different relationships among classes

Types of relationships
- Generalization
  - Inheritance
  - Implementation
- Association
  - Aggregate
  - Composition
- Dependency
Generalization

- Denotes inheritance between classes
  - Can view as “is-a” relationship

- Example
  - Lecturer is a person (Lecturer extends Person class)

- Types of generalization
  - Subclass extends superclass
    - Solid line ending in (open) triangle
  - Class implements interface
    - Dotted line ending in (open) triangle

Generalization Example

- Inheritance

Laptop, Desktop, PDA inherit state & behavior from Computer
Generalization Example

Implementation

Laptop implements DVDplayer interface

Association

Denotes interaction between two classes

Example

Lecturer teaches course

Indicates relationship between Lecturer & Course
Association Notation

- Associations may be labeled
  - Name ⇒ name of association
  - Role ⇒ name of data field
  - Multiplicity ⇒ number of objects referenced

Example

```
A
```
```
B
```
```
Name
```
```
MultiplicityA
```
```
RoleA
```
```
MultiplicityB
```
```
RoleB
```

Multiplicity of Associations

- Some relationships may be quantified
- Multiplicity denotes how many objects are related to the source object

Notation

- * ⇒ 0, 1, or more
- 5 ⇒ 5 exactly
- 5..8 ⇒ between 5 and 8, inclusive
- 5..* ⇒ 5 or more
**Multiplicity of Associations**

- **Many-to-one**
  - Bank has many ATMs, ATM knows only 1 bank

  ![Diagram showing ATM has many to one relationship with Bank]

- **One-to-many**
  - Inventory has many items, items know 1 inventory

  ![Diagram showing Inventory has many to one relationship with Item]

**Association w/ Navigation**

- **Navigation information**
  - Relationship between classes may be directional
    - Only class A can send messages to class B
    - Arrowhead indicates direction of relationship

- **Example**

  ```java
  Class Course {
      Lecturer TheBoss;
  }
  ```

  ```java
  Class Lecturer {
      ...
  }
  ```
Association w/ Navigation

- Undirected edge
  - Relationship between classes may be bi-directional
  - Direction of relationship may be unknown

Examples

```java
Class Course {
    Lecturer TheBoss;
}

Class Lecturer {
    Course [ ] class;
}
```

Association

- Causes of association
  - Permanent
    - Data field
  - Transitory
    - Return value
    - Parameter
    - Local variable
  - Example
    ```java
    Class A {
        B myB;
        B Foo(B x) {
            B y = new( );
            return y;
        }
    }
    ```
  - Cause not necessarily known
    ```java
    Class B { ... }
    ```
Permanent Association

- Permanent / structural association
  - Class A contains reference to class B in data field
  - Can view as “has-a” relationship

Example
- Course has a Lecturer

```
Class Course {
    Lecturer TheBoss;
}
```

Permanent Association

- Types of permanent association
  - Aggregation
    - Class contains a collection of other classes
    - Solid line ending in (open) diamond
  - Composition
    - Class formed as a collection of other classes
    - Solid line ending in (filled) diamond
Aggregation

- **Owned objects have independent existence**
  - Object is not exclusively owned
  - May be owned by other classes
  - Life cycle of owned object is not connected to owner
  - Owned object may exist longer than owner

- **Example**
  - Course has a Textbook (but so does Library)

```
Course
  | Textbook b
Library
  | Textbook b
```

Composition

- **Owned objects have no independent existence**
  - Object is exclusively owned
  - May not be owned by other classes
  - Life cycle of owned object is connected to owner
  - Owned object only exists while owner exists

- **Example**
  - Course has a CourseEvaluation

```
Course
  | CourseEval E
CourseEval
  | CourseEval
```
**Dependency**

- A transitory relationship between classes
  - Always directed (class A depends on B)
  - Indicates change in class B may affect class A
  - Can view as "uses a" relationship
  - Represented by dotted line with arrowhead

**Example**

```
A B
```

A depends on B

---

**Dependency**

- Dependence may be caused by
  - Local variable
  - Parameter
  - Return value

**Example**

```
Class A {
    B Foo(B x) {
        B y = new();
        return y;
    }
}
```

```
Class B {
    ...
    ...
    ...
}
```
**UML for Inner Class**

- **Notation**
  - Solid line between class and inner class
  - Anchor (cross inside circle) by enclosing class

- **Example**

```
A ——+—— B
     |   |
```

B is an inner class of A

**UML for Generic Classes**

- **Notation**
  - Generic parameter(s) placed in upper right corner
  - Inside dotted rectangle

- **Example**

```
A
```

E is type variable for generic class A
Overview

- Unified Modeling Language (UML)
  - Background
  - UML diagrams
  - Class diagrams
    - Examples

UML Examples

- Read UML class diagram
  - Try to understand relationships
  - Practice converting to / from Java code

- Examples
  - Pets & owners
  - Computer disk organization
  - Library books
  - Banking system
  - Home heating system
  - Printing system
**UML Example – Veterinary System**

Try to read & understand UML diagram

- 1 or more Pets associated with 1 PetOwner

**UML Example – Computer System**

Try to read & understand UML diagram

- 1 CPU associated with 0 or more Controllers
- 1-4 DiskDrives associated with 1 SCSCController
- SCSCController is a (specialized) Controller
**UML Example – Library System**

- Try to read & understand UML diagram

![Diagram showing the relationship between Shelf, Book, Patron, and Page.

- 1 or more Book associated with 1 or more Pages
- Patron & Shelf temporarily use (depend on) Books

**UML Example – Banking System**

- 1 Bank associated with 0 or more Accounts
- Checking, Savings, MoneyMarket are Accounts
UML Example – Home Heating System

- Each Thermostat has 1 Room
- Each Thermostat associated with 0 or more Heaters
- ElectricHeater is a specialized Heater
- AubeTH101D is a specialized Thermostat

UML → Java : Veterinary System

**UML**

```
Pet 1..* 1 PetOwner
```

**Java**

```java
class Pet {
    PetOwner myOwner; // 1 owner for each pet
}
class PetOwner {
    Pet [ ] myPets; // multiple pets for each owner
}
```
Java → UML : Veterinary System

Java

```java
class Pet {
    PetOwner myOwner; // 1 owner for each pet
}
class PetOwner {
    Pet[] myPets; // multiple pets for each owner
}
```

UML

```
Pet 1..* 1 PetOwner
```

UML → Java : Computer System

```
UML

```
```
Controller

SCSIController
```

Java

```java
class Controller {
}
class SCSIController extends Controller {
}
```
UML → Java : Computer System

UML

```
CPU  1  *  Controller
```

```
DiskDrive  1..4  1  SCSIController
```

Java

Design code using all available information in UML...

```java
class CPU {
    Controller [] myCtIrs;
}
class Controller {
    CPU myCPU;
}
class SCSIController extends Controller {
    DiskDrive [] myDrives = new DiskDrive[4];
}
Class DiskDrive {
    SCSIController mySCSI;
}
```
Java → UML : Printing System

Java

class Registry {
  PrintQueue findQueue();
}
class PrintQueue {
  List printJobs;
  Printer myPrinter;
  Registry myRegistry;
  void newJob();
  int length();
  Resources getResource();
}

Java → UML : Printing System

Java

Class Printer {
  Resources myResources;
  Job curJob;
  void print();
  boolean busy();
  boolean on();
}
class Job {
  Job(Registry r) {
  ...
  }
}
Java → UML: Printing System

Java

All together

UML Summary

UML → modeling language
Visually represents design of software system
We focused on class diagrams
  Contents of a class
  Relationship between classes
You should be able to
  Draw UML class diagram given Java code
  Write Java code given UML class diagram