CMSC 132: Object-Oriented Programming II

Unified Modeling Language (UML)

Department of Computer Science
University of Maryland, College Park

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 (Unified Modeling Language)

- UML is a modeling language for
  - Specifying
  - Visualizing
  - Constructing
  - Documenting
- object-oriented software

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
Class Diagram

- Represents (static) structure of system

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

Class Diagams

- Information for class contains
  - Name
  - State
  - Behavior

UML Class Diagrams ↔ Java Code

- Different representation of same information
  - Name, state, behavior of class
  - Relationships between classes

Java → UML: Clock Example

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

- UML
  ```text
  + Clock
    - seconds: int
    - minutes: int
    - hours: int
    + start(): void
    + adjustTime(): void
    + reset(): void
  ```
**Class Diagram Notation**

- **UML notation**
  - Type ⇒ type name preceded by colon :
  - Visibility ⇒ prefix symbol
    - + public
    - – private

- **Types of relationships**
  - Generalization
  - Implementation
  - Association
  - Dependency

**Java → UML : Clock Example**

- **Java**
  ```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( );
}
```

- **Class Diagram**

**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**

- Laptop, Desktop, PDA inherit state & behavior from Computer

**Association**

- Denotes interaction between two classes

- **Example**
  - Lecturer teaches course
  - Indicates relationship between Lecturer & Course

**Generalization Example**

- Implementation

- Laptop implements DVDplayer interface
**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;
    ...
}
class Lecturer {
}
```

**Association w/o Navigation**

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

**Examples**

```java
class Course {
    Lecturer TheBoss;
    Course [] class;
}
class Lecturer {
}
class Foo
class Bar
```

**Permanent Association**

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

**Example**

```java
class A {
    B x;
    ...
}
class B {
    ...
}
```

A has a B

**Temporary Association (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**

```java
class A {
    B foo(B x) {
        B y = new();
        ...
    }
    ...
}
class B {
    ...
    ...
}
```

A depends on B

**Dependency**

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

**Example**

```java
class A {
    class B {
        ...
    }
}
class B {
    ...
    ...
}
```

**UML Examples**

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

**Examples**

- Computer disk organization
- Banking system
- Home heating system
- Printing system
UML Example – Computer System

- Try to read & understand UML diagram

  - CPU is associated with Controllers
  - DiskDrive is associated with SCSIController
  - SCSIController is a (type of) Controller

UML Example – Banking System

- Bank associated with Accounts
- Checking, Savings, MoneyMarket are type of Accounts

UML Example – Home Heating System

- Thermostat associated with (has a) Room
- Thermostat associated with (has a) Heater
- ElectricHeater is a specialized Heater
- AubeTH101D is a specialized Thermostat

UML Example – Library System

- Books are associated with (has some) Pages
- Patron & Shelf depend on (temporarily use) Books

UML → Java: Computer System

- Java
  
  ```java
  class Controller {
  }
  class SCSIController extends Controller {
  }
  ```

UML → Java: Computer System

- Java
  
  ```java
  class Controller {
  }
  class SCSIController extends Controller {
  }
  ```

- Design code using all available information in UML...
UML → Java: Computer System

Java

class CPU {
    Controller myCtlrs[];
}
class Controller {
    CPU myCPU;
}
class SCSIController extends Controller {
    DiskDrive myDrive[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) {
        ...
    }
}

UML Tools

Can automatically generate
- UML diagrams from code
- Code from UML diagrams

Examples
- AmaterasUML
- Violet

Amateras UML Editor

Drag-n-drop classes
into UML diagram
- Auto creates class w/ attributes & methods
- Add links manually
- No directed associations
- Use undirected association + directed dependency together
### Amateras UML Editor – Eclipse Plugin

- Drag-n-drop classes into UML diagram
- Auto creates class with attributes & methods
- Add links manually
  - No undirected associations
  - Use directed association in both directions instead

### Violet UML Editor

- **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