

# Unified Modeling Language (UML)

---



**Nelson Padua-Perez**  
**Chau-Wen Tseng**

**Department of Computer Science**  
**University of Maryland, College Park**

## Overview

- **Unified Modeling Language (UML)**
  - **Models & views**
  - **Class diagrams**
  - **Sequence diagrams**

# UML

- **UML (Unified Modeling Language)**
  - Graphic modeling language for describing object-oriented software
  - Started in 1994
  - Combined notations from 3 leading OO methods
    - OMT (James Rumbaugh)
    - OOSE (Ivar Jacobson)
    - Booch (Grady Booch)

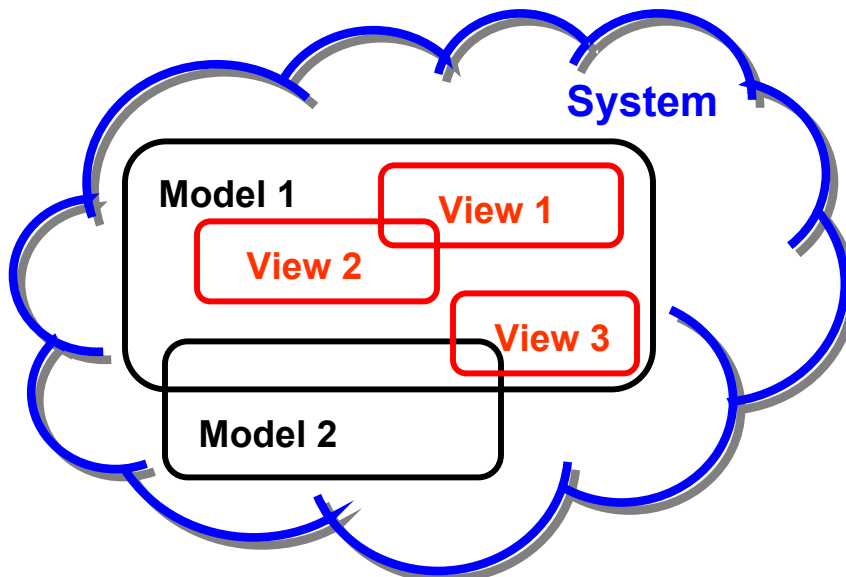
## UML Motivation

- **Software growing larger & complex**
  - Difficult to analyze
- **Need to describe software design**
  - Clearly
  - Concisely
  - Correctly
- **UML equivalent to software “blueprint”**
  - Provides simple yet clear abstraction for software
- **Computer-aided software engineering (CASE)**
  - Tools for generating & analyzing UML

## Definitions

- **Model**
  - Abstraction describing (all or part of) system
- **View**
  - Depicts selected aspects of a model using set of diagrams
- **Diagram**
  - Actual depiction of aspect of model
- **Notation**
  - Set of graphical / textual rules for representing view

## Models & Views



# UML

- **Industry standard**
- **Many features**
  - Large collection of notations
  - Multiple views
  - Multiple diagrams
- **We focus mainly on**
  - Logical view of relationship between classes

## UML Views

- **Use-Case**
  - Show external user (actor) view of system
- **Logical**
  - Show how functionality is implemented
- **Component**
  - Show organization of code components
- **Deployment**
  - Show assignment of code to physical computers

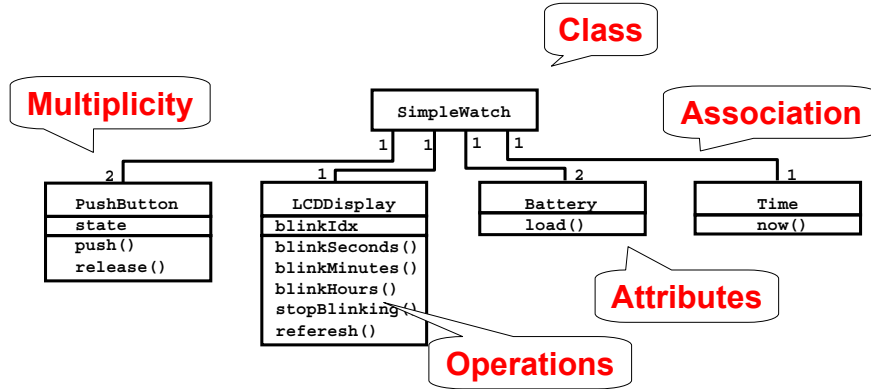
## (Some) UML Diagrams

- **Class**
  - Describe static structure of the classes in system
- **Sequence**
  - Describe dynamic behavior between users and objects
- **Use case**
  - Describe functional behavior seen by (external) user
- **State**
  - Describe dynamic behavior of objects as finite state machine
- **Activity**
  - Model dynamic behavior of a system as a flowchart

## UML Conventions

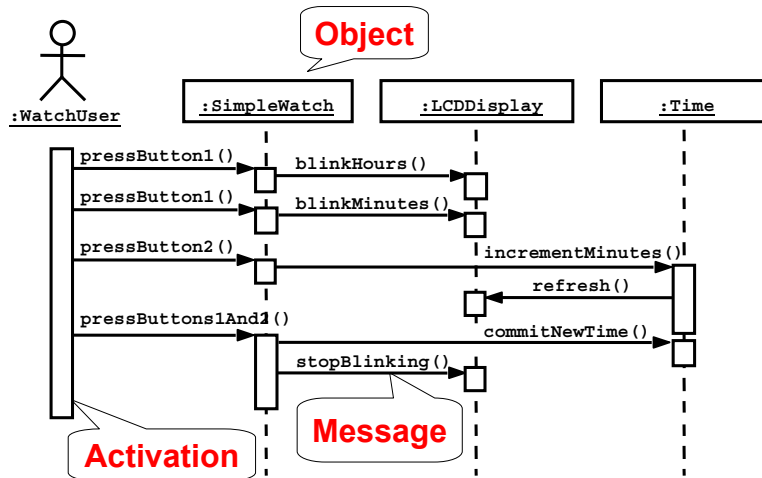
- **Rectangles** ⇒ classes or instances
- **Ovals** ⇒ functions or use cases
- **Types** ⇒ preceded by colon, not underlined
  - `:SimpleWatch`
- **Instances** ⇒ underlined names
  - `myWatch:SimpleWatch`
- **Diagrams** ⇒ graphs
  - **Nodes** ⇒ entities
  - **Arcs** ⇒ relationships between entities

# Class Diagrams



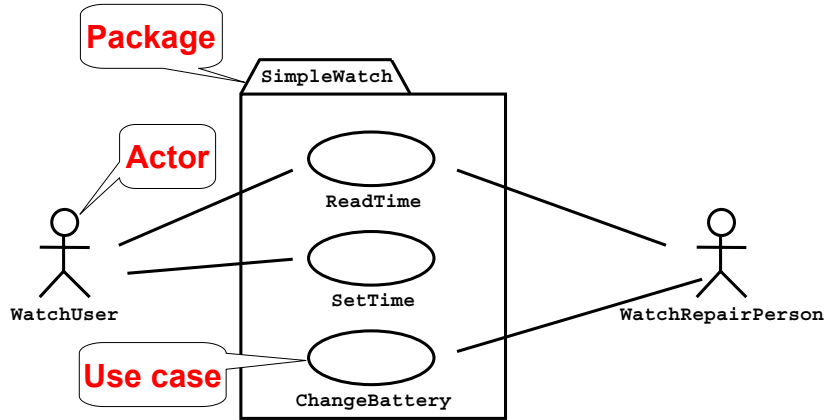
Class diagrams represent structure of system

# Sequence Diagram



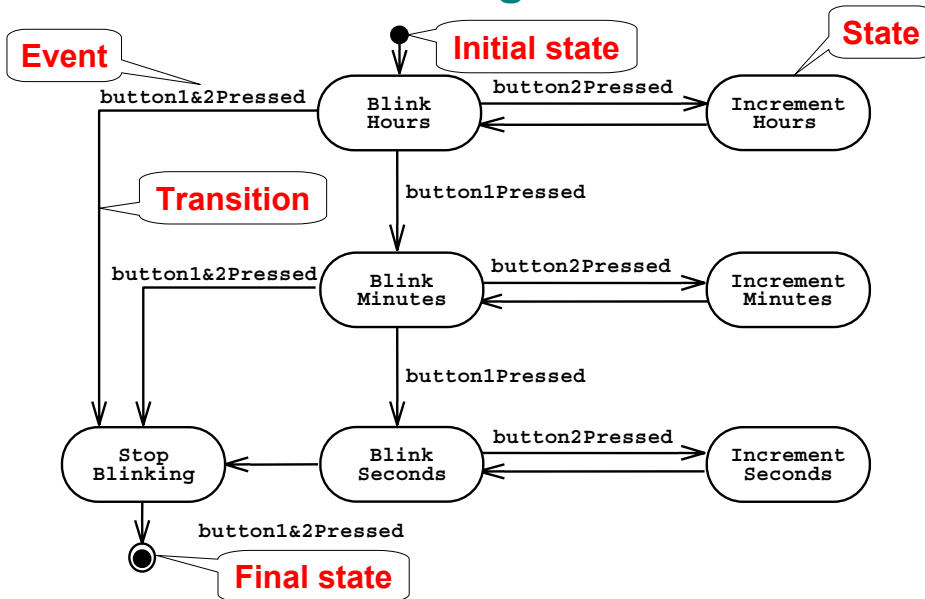
Sequence diagrams represent behavior as interactions

# Use Case Diagrams



Use case diagrams represent functionality of system from external user's point of view

# State Diagrams



## UML Sequence Diagrams

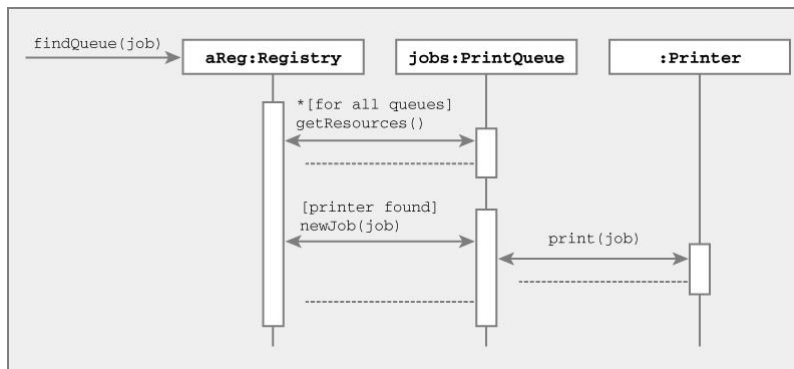
- Represent behavior in terms of interactions
- During requirements analysis
  - Used to refine use case descriptions
- During system design
  - Used to refine subsystem interfaces

## Sequence Diagram Notation

- Columns ⇒ **Classes**
- Arrows ⇒ **Messages**
- Narrow rectangles ⇒ **Activations**

## Sequence Diagram Example

### ■ Sequence of events for printer queue



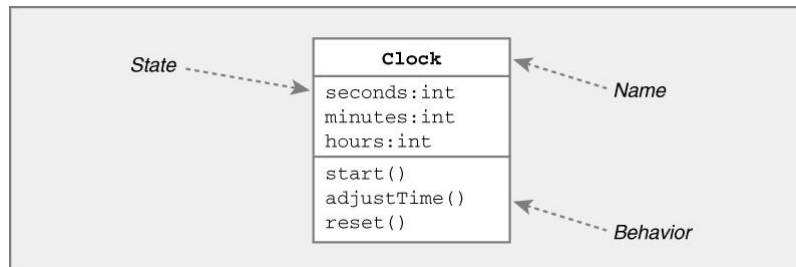
## UML Class Diagrams

- Represent the (static) structure of the system
- During analysis
  - Used to model problem domain concepts
- During detailed design
  - Used to model classes

## Class Diagrams

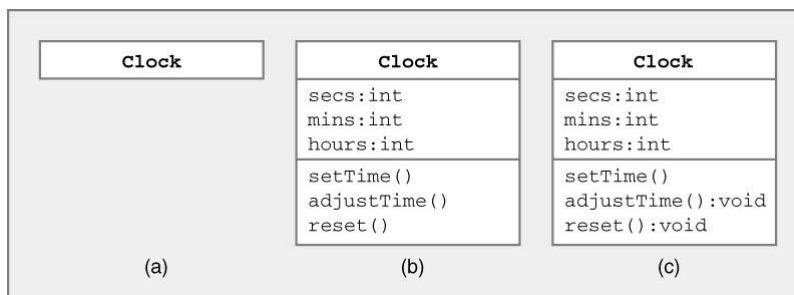
### ■ Class contains

- Name
- State
- Behavior



## Class Diagrams

- Types may be included
- Only class name is required



## Sequence Diagram Observations

- Represents behavior in terms of interactions
- Complement the class diagrams which represent structure
- Useful for finding participating objects

## UML Class Diagrams ↔ Java Code

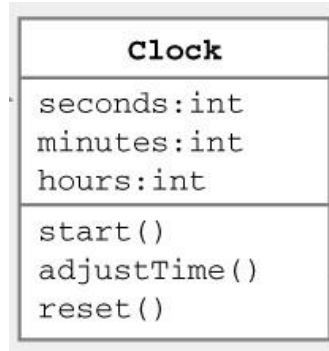
- Different representation of **same** information
  - Name, state, behavior of class
  - Relationship(s) between classes
  - Should be able to derive one from the other
- Motivation
  - UML ⇒ Java
    - Implement code based on design written in UML
  - Java ⇒ UML
    - Create UML to document design 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



Class Diagram

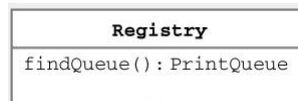
## Java → UML : Printing System

### ■ Java

```
class Job {
}
```



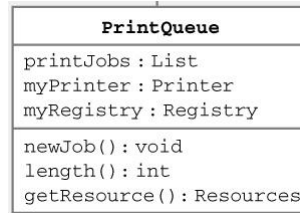
```
class Registry {
  PrintQueue findQueue();
}
```



## Java → UML : Printing System

### ■ Java

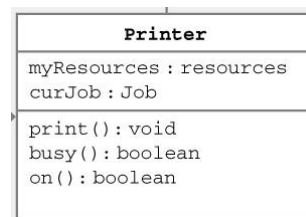
```
class PrintQueue {  
    List printJobs;  
    Printer myPrinter;  
    Registry myRegistry;  
    void newJob();  
    int length();  
    Resources getResource();  
}
```



## Java → UML : Printing System

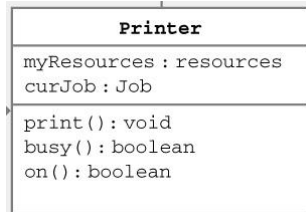
### ■ Java

```
class Printer {  
    Resources myResources;  
    Job curJob;  
    void print();  
    boolean busy();  
    boolean on();  
}
```



## UML → Java : Printing System

### ■ UML



### ■ Java

```
class Printer {
    Resources myResources;
    Job curJob;
    void print();
    boolean busy();
    boolean on();
}
```

A red arrow points from the UML class diagram to the Java code block.

## Summary

- Overview of UML
- Focus on **class diagrams**
  - How they represent structure of software
  - Conversion to/from Java
- Next lecture will examine
  - How UML represents relationships between classes
  - More UML ↔ Java conversions