DA-IICT

UNIFIED MODELING LANGUAGE

IT313: Software Engineering

Design (Why?)

Saurabh Tiwari

1

"Software Systems" – Designing (Why?)

Software Systems have many components and interactions

- External conditions can trigger unexpected changes
- The problem is beyond the control of any single individual
- Changes forces participants to look for new solutions
- Several participant need to cooperate and develop desired solution
- We are interested in Software-Intensive Systems

Managing Complexity
&
Changeability

W5H2

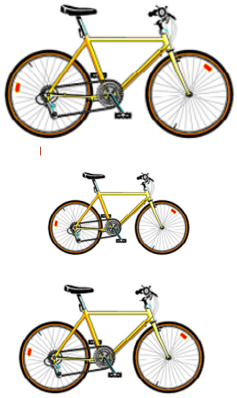
Why OO?
What is OOAD?
Who…
When…
Where…
How to do OOAD?
How much…

Object-Oriented Modeling and Design

- *Is a way of thinking about the problems using models organized around real-world concepts*
- *Concept: Object*

Real-World Objects

- *OO means that we organize software as a collection of discrete objects that incorporate both data structure and behavior.*



Bicycle Objects

Abstract into

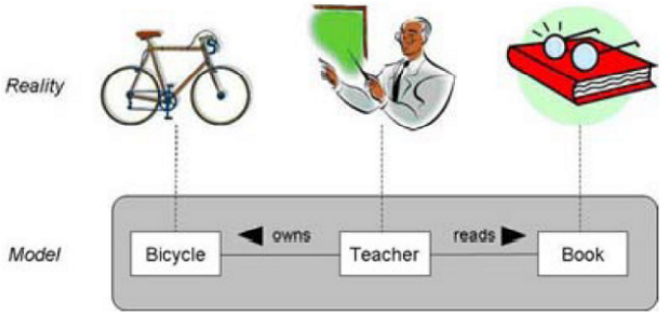
Bicycle Class

Attributes
Frame size
Wheel size
Number of gears
Material

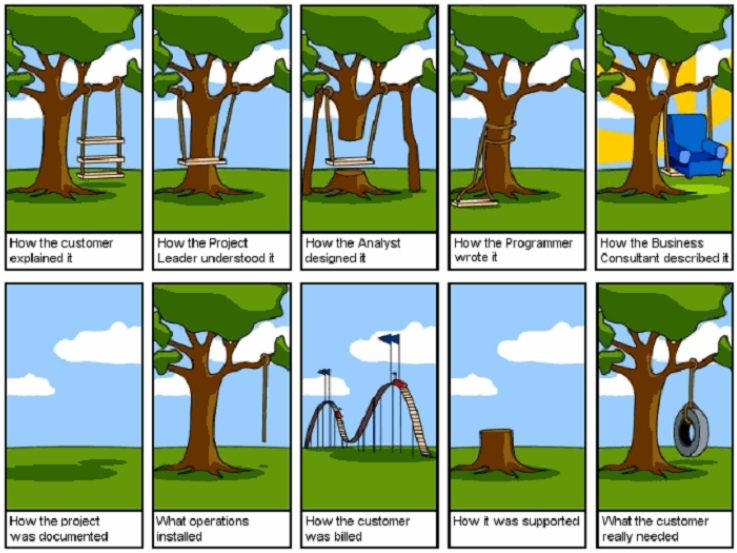
Operations
Shift
Move
Repair

Real-World Objects

- *Have identity and are distinguishable*
  - Identity means that data is quantized into discrete, distinguishable entities called **Objects**
  - Share two characteristics – *State and Behavior*



Process …



Why Object-Oriented?

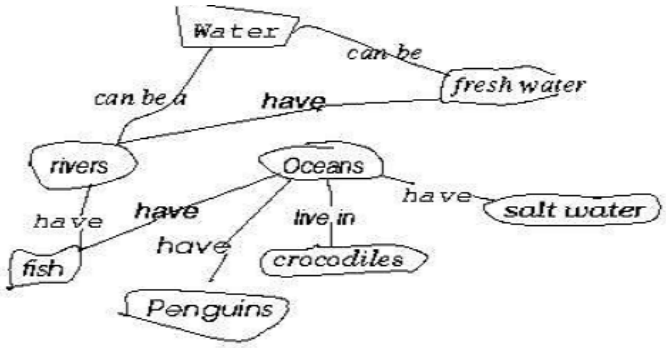
- *"The "software crises" came about when people realized the major problems in software development were … caused by **communication** difficulties and the management of **complexity**" [Budd]*

What kind of language can alleviate difficulties with communication & complexity?

Why Object-Oriented? Concepts and Objects

- **Study of a first grade class** [Martin & Odell][Novak, 1984, Cambridge University Press]

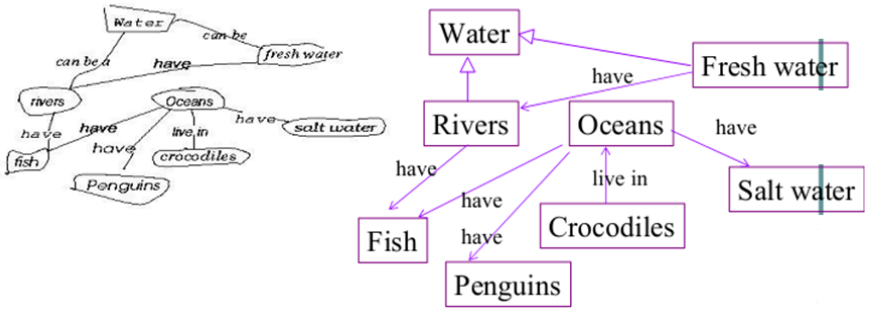
- *When given a list of concepts (water, salt water, Oceans, Penguins,...),*
- *Harry constructed a concept diagram through which he understands his world and communicates meaning*



Do you understand what Harry Understands?

Why Object-Oriented? for Conceptual ... Modeling Reasons...

- What kind of language can be used to create this concept diagram, or Harry's mental image?



Things, Relationships and Diagram

Why Object-Oriented? -> for Modeling

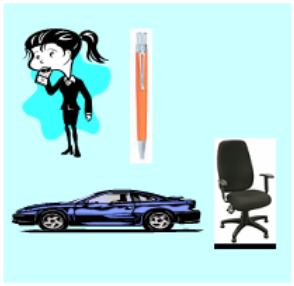
- A model is a simplification of reality.
  - Well…sort of….but not quite
- A model is our simplification of our perception of reality

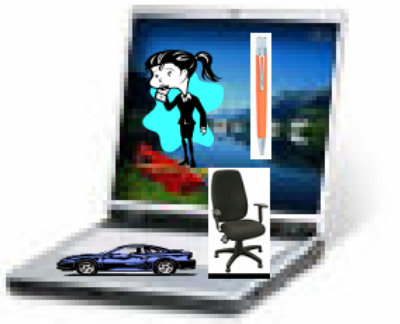
Abstraction
Focus on the essential
Omits tremendous amount of details
…Focus on what an object "is and does"

- To understand *why* a software system is needed, *what* it should do, and *how* it should do it.
- To communicate our understanding of why, what and how.
- To detect commonalities and differences in your perception, my perception, his perception and her perception of reality.
- To detect misunderstandings and miscommunications.

What is Object-Orientation? What is object?

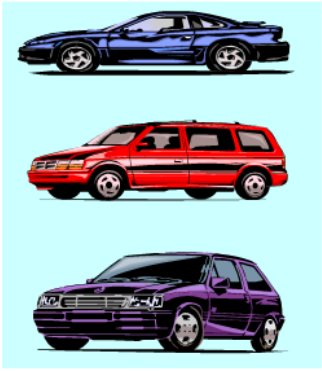
- An "object" is anything to which a concept applies, ***in our awareness***
- Things drawn from the problem domain or solution space.
  - E.g., a living person in the problem domain, a software component in the solution space.





What is Object-Orientation?

Classification → classes → Instantiation



<<instanceOf>>

<<instanceOf>>

<<instanceOf>>

Class Car

Attributes
- ❑ Model
- ❑ Location
- ❑ #Wheels = 4

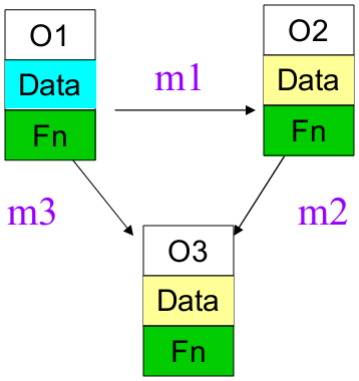
Operations
- ❑ Start
- ❑ Accelerate

Encapsulation

- ***CLASS***
  - a collection of objects that share common properties, attributes, behavior and semantics, in general.
  - A collection of objects with the same data structure (attributes, state variables) and behavior (function/code/operations) in the solution space.
- **Classification**
  - Grouping of common objects into a class
- Instantiation.
  - The act of creating an instance.

What is Object-Orientated Application?

- Collection of discrete objects, interacting with each other
- Objects have property and behavior
- Interactions through message passing



O1, O2 and O3 are discrete Objects

What is OOAD?

Analysis – understanding, finding and describing concepts in the problem domain.

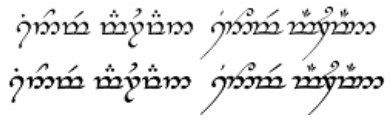
Design – understanding and defining software solution/objects that *represent* the analysis concepts and will eventually be implemented in code.

OOAD – Analysis is object-oriented and design is object-oriented. A software development approach that emphasizes a logical solution based on objects.

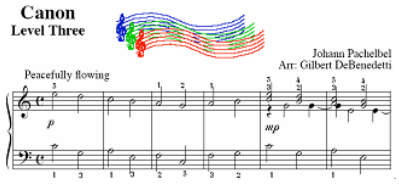
Traceability!!!

How to DO OOAD?

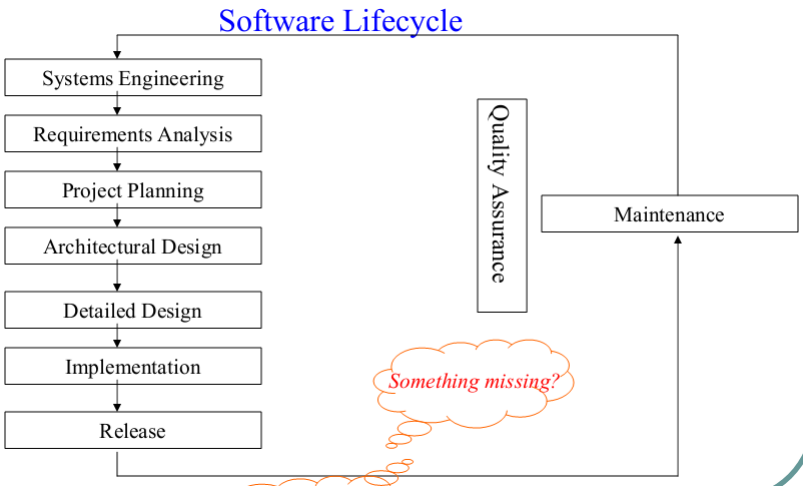
Notation Vs Process…



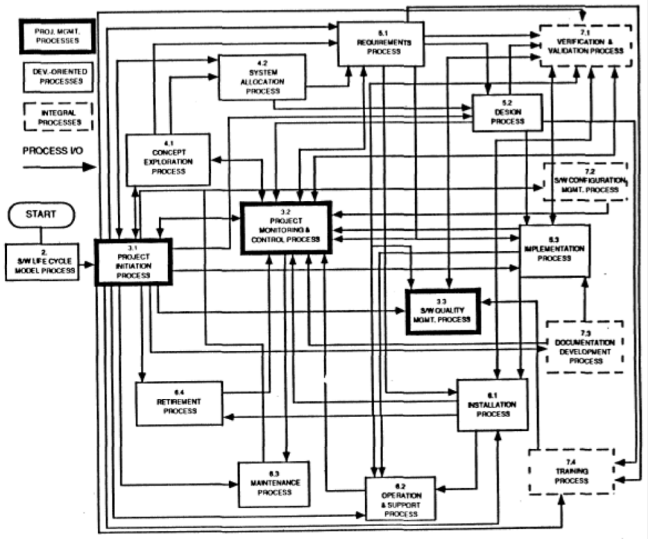
Unifying Modeling Language (UML) is a Notation
So are English, Elivish, Ku …..



Where to use OOAD?

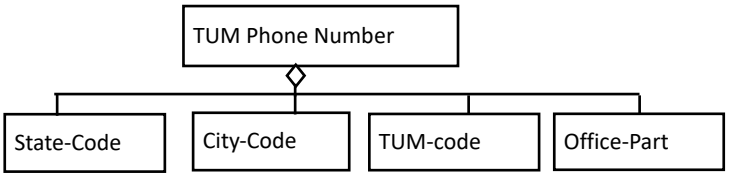


What is the problem with this Drawing?



Abstraction

- Complex systems are hard to understand
  - The 7 +- 2 phenomena
    - Our short term memory cannot store more than 7+-2 pieces at the same time -> limitation of the brain
    - TUM Phone Number: 498928918204
- Chunking:
  - Group collection of objects to reduce complexity
  - State-code, city-code, TUM-code, Office-Part

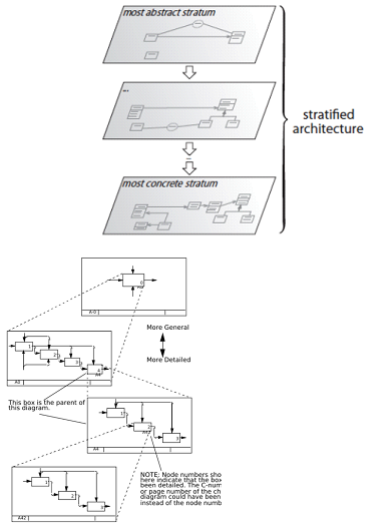


Abstraction

- Abstraction allows us to ignore unessential details
- Two definitions for abstraction:
  - Abstraction is a *thought process* where ideas are distanced from objects
    - Abstraction as activity
  - Abstraction is the *resulting idea* of a thought process where an idea has been distanced from an object
    - Abstraction as entity
- Ideas can be expressed by models

Abstraction

- Define levels of abstractions
- Define models at these levels
- Define transformation of models from higher to lower levels
- Implement



We use Models to describe Software Systems

- Object model: What is the structure of the system? What are the objects and how are they related?
- Functional model: What are the functions of the system? How is data flowing through the system?
- Dynamic model: How does the system react to external events? How is the event flow in the system ?
- System Model: Object model + functional model + dynamic model

Other models used to describe Software Development

- Task Model:
  - PERT Chart: What are the dependencies between tasks?
  - Schedule: How can this be done within the time limit?
  - Organization Chart: What are the roles in the project?
- Issues Model:
  - What are the open and closed issues?
    - What blocks me from continuing?
  - What constraints were imposed by the client?
  - What resolutions were made?
    - These lead to action items

Technique to deal with Complexity: Decomposition

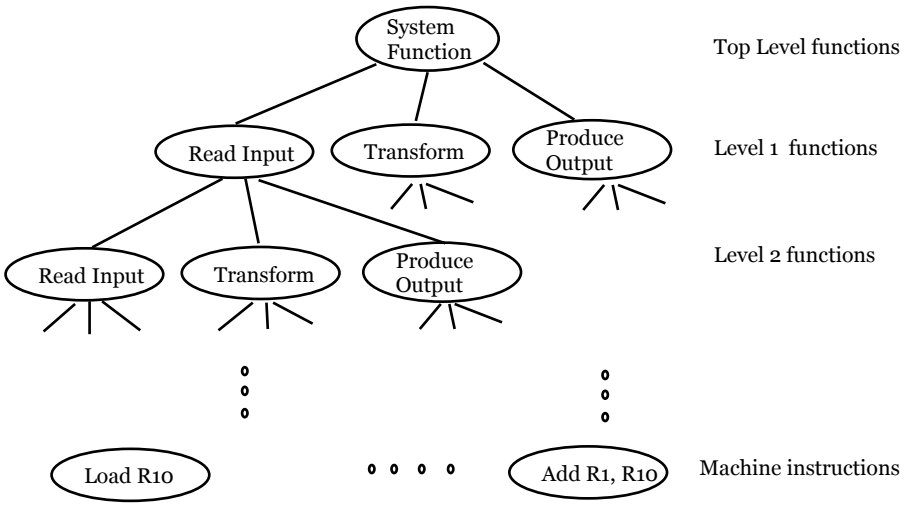
- A technique used to master complexity ("divide and conquer")
- Two major types of decomposition
  - Functional decomposition
  - Object-oriented decomposition
- Functional decomposition
  - The system is decomposed into modules
  - Each module is a major function in the application domain
  - Modules can be decomposed into smaller modules.

Decomposition (cont'd)

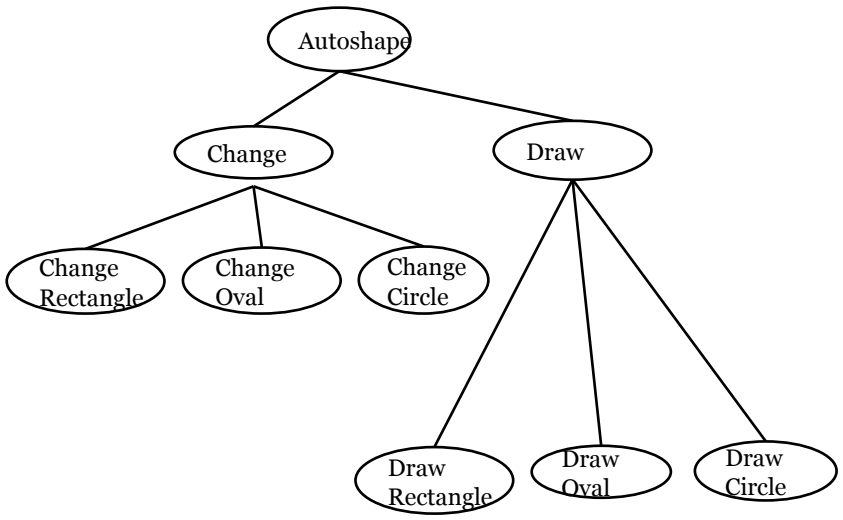
- Object-oriented decomposition
  - The system is decomposed into classes ("objects")
  - Each class is a major entity in the application domain
  - Classes can be decomposed into smaller classes
- Object-oriented vs. functional decomposition

Which decomposition is the right one?

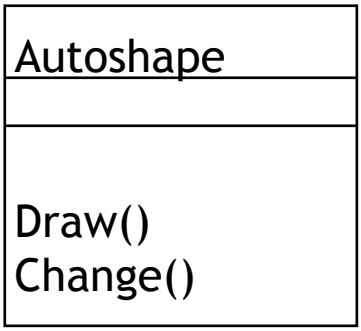
Functional Decomposition



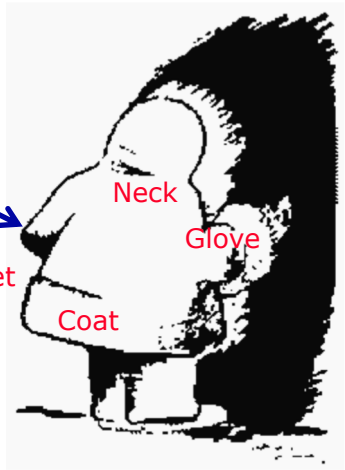
Functional Decomposition: Autoshape



Object-Oriented View

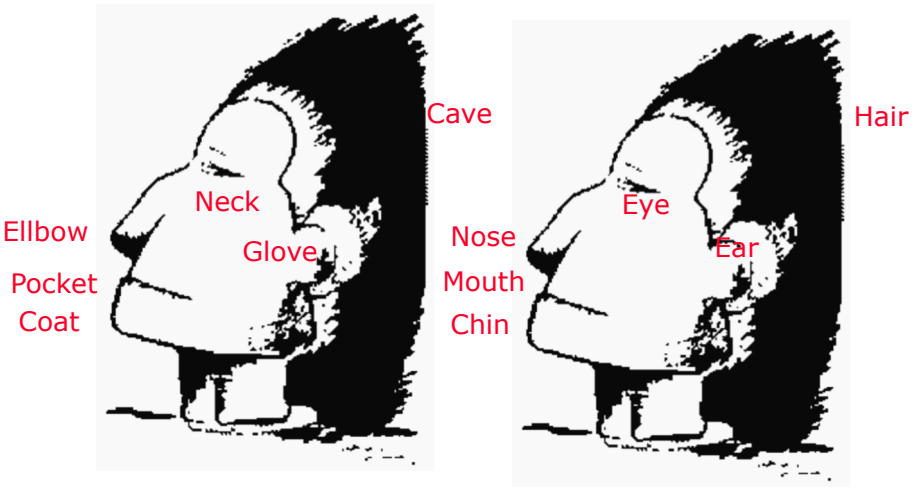


What is This?



An Eskimo!

A Face!



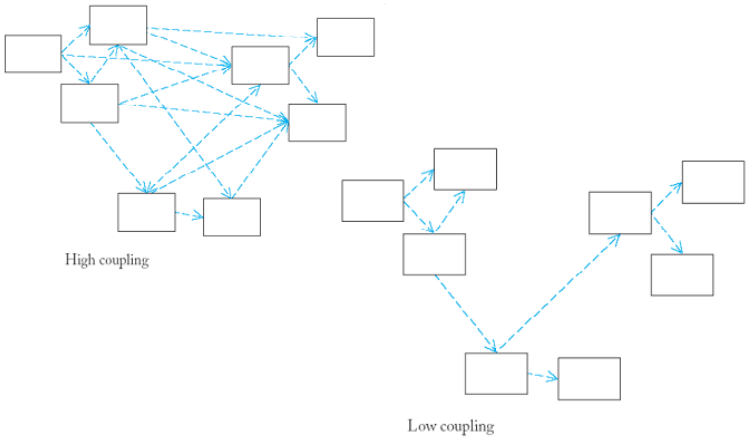
Depending on the purpose of the system different objects might be found: A nose is suddenly an elbow, hair is a cave, an ear turns out to be a glove.

Modularity

Highly cohesive structure
- Loosely coupled
- Interactions through clearly defined

Interfaces
- Separation of concerns

- Improved modifiability/maintainability



Hierarchies

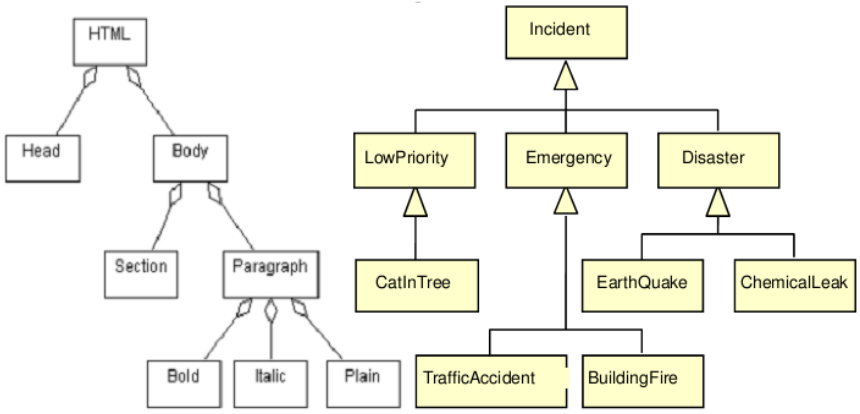
We got abstractions and decomposition
- This leads us to chunks (classes, objects) which we view with object model
- This should be modular

- Another way to deal with complexity is to provide simple relationships between the chunks

- One of the most important relationships is hierarchy

Two important hierarchies
- "Part of" hierarchy
- "Is-kind-of" hierarchy

34



Part-Of Kind-Of

35

Principles of S/W Development

Core Principles [David Hooker 1996]

- Ensure Usability
  - Produce to be consume
- Communication
- Simplicity
- Open to change
- Employ Reuse
- Feedback
- Think

36

Principles of S/W Development

Analysis Modeling Principles

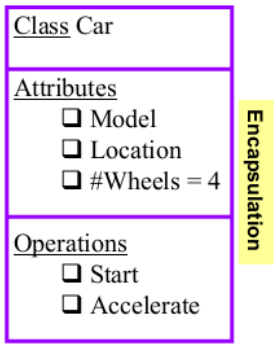
- Information domain of a problem must be understood and represented
- Define functional and non functional requirements clearly
- Define a complete and consistent set of requirements
- Analysis should help development

37

What constitutes a good model?

- A model should
  - use a standard notation
  - be understandable by clients and users
  - lead software engineers to have insights about the system
  - provide abstraction
- Models are used:
  - to help create designs
  - to permit analysis and review of those designs.
  - as the core documentation describing the system.

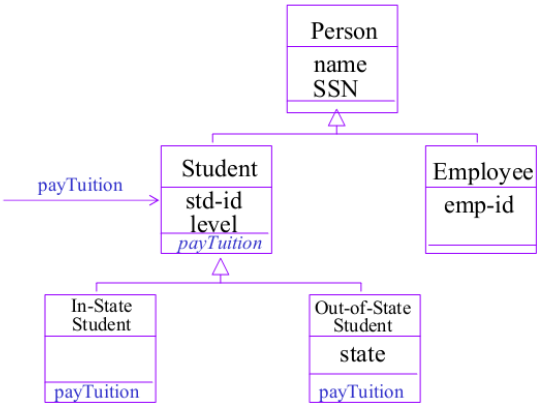
OO Concepts



- Objects
- Classes
- Interfaces
- Inheritance
- Attributes
- Methods
- Encapsulation
- Information Hiding, or Visibility
- Instances
- Delegation
- Polymorphism
- Dynamic Binding
- Aggregation
- Association
- Message Passing, or interactions
- Genericity

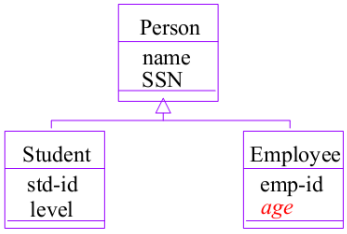
Polymorphism

Objects of different classes respond to the same message differently.



Super class vs. Sub class

What is Generalization??



Specialization: The act of defining one class as a refinement of another.

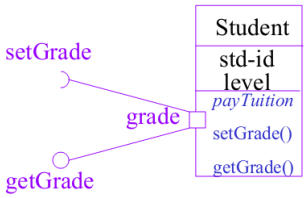
Subclass: A class defined in terms of a specialization of a superclass using inheritance.

Superclass: A class serving as a base for inheritance in a class hierarchy

Inheritance: Automatic duplication of superclass attribute and behavior definitions in subclass.

Interface

- Information hiding - all data should be hidden within a class, at least in principle
- make all data attributes private
- provide public methods to get and set the data values (cf. Java design patterns)
- e.g. Grade information is usually confidential, hence it should be kept private to the student. Access to the grade information should be done through *interfaces*, such as setGrade and getGrade



Other concepts will ....

Class Modeling

Objects

The most fundamental entity in an OO Program

"An Object is a concept, abstraction or thing with identity that has meaning for an application"

- Defined by class data type

Consists of

- Identifier
- Values
- Behavior

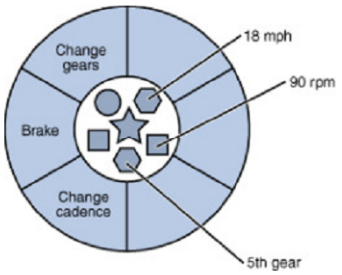
JoeSmith: Person
name = "Joe Smith" birthdate = 21 October 1988

MarySharp: Person
name = "Mary Sharp" birthdate = 16 March 1990

Object with values

Software Object: Cycle

```
1.  class Bicycle {
2.      int cadence = 0;
3.      int speed = 0;
4.      int gear = 1;
5.      void changeCadence(int newValue) {
6.          cadence = newValue;
7.      }
8.      void changeGear(int newValue) {
9.          gear = newValue;
10.     }
11.     void speedUp(int increment) {
12.         speed = speed + increment;
13.     }
14.     void applyBrakes(int decrement) {
15.         speed = speed - decrement;
16.     }
17.     void printStates() {
18.         System.out.println("cadence:"+cadence+" speed:"+speed+" gear:"+gear);
19.     }
20. }
```



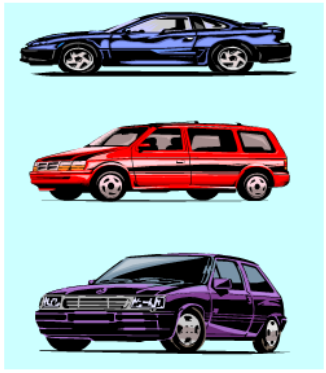
45

Classes

- A class is a pattern, a blue print, or a template for a category of structurally identical entities
- Created entities are called objects or instances
  - Class is like instance factory
  - Static entity
- ▫ A class has three components
  - Name
  - Attributes (also termed as variables, fields, or properties)
  - Methods (also termed as operations, features, behavior, functions)

46

Classes



Objects

<<instanceOf>>

<<instanceOf>>

<<instanceOf>>

Class Car

Attributes
- ❑ Model
- ❑ Location
- ❑ #Wheels = 4

Operations
- ❑ Start
- ❑ Accelerate

Encapsulation

- ***CLASS***
  - a collection of objects that share common properties, attributes, behavior and semantics, in general.
  - A collection of objects with the same data structure (attributes, state variables) and behavior (function/code/operations) in the solution space.
- **Classification**
  - Grouping of common objects into a class
- Instantiation.
  - The act of creating an instance.

47

Class

ClassName
Attributes
Operations

Customer
-name:String -userID:String -password:String
+getName():String +setName(newName:String) +getUserID():String +setUserID(newUserID:String) +getPassword():String +setPassword(newPassword:String)

Classes are represented by rectangles, with the name of the class, and can also show the attributes and operations of the class in two other “compartments” inside the rectangle.

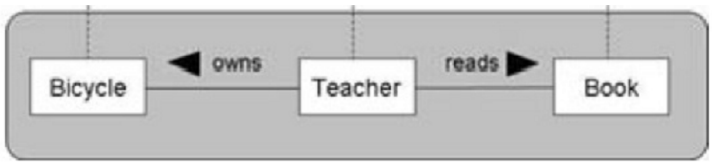
48

Class Diagrams

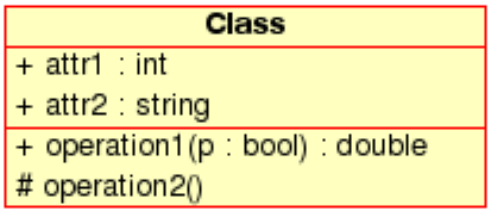
A Class defines the attributes and the methods of a set of objects.

All objects of this class (instances of this class) share the same behaviour, and have the same set of attributes (each object has its own set).

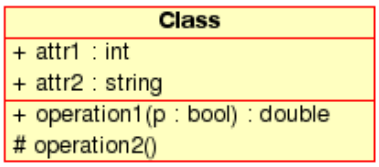
Class diagrams provide a graphic notation for modeling classes and their relationships thereby describing possible objects.



Class Diagrams



Class Diagrams

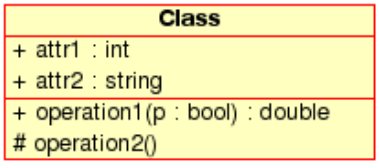


Attributes

In UML, Attributes are shown with at least their name, and can also show their type, initial value and other properties. Attributes can also be displayed with their visibility:

- \+ public attributes
- \# protected attributes
- \- private attributes

Class Diagrams

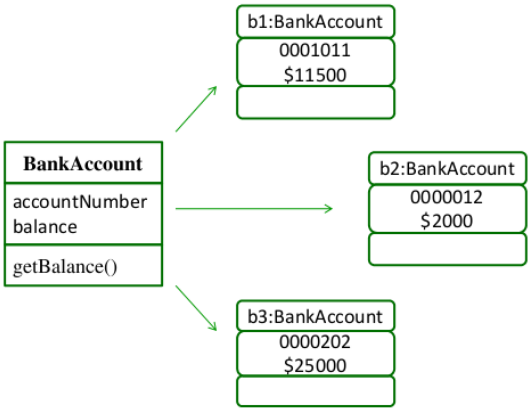


Operations

Operations (methods) are also displayed with at least their name, and can also show their parameters and return types. Operations can, just as Attributes, display their visibility:

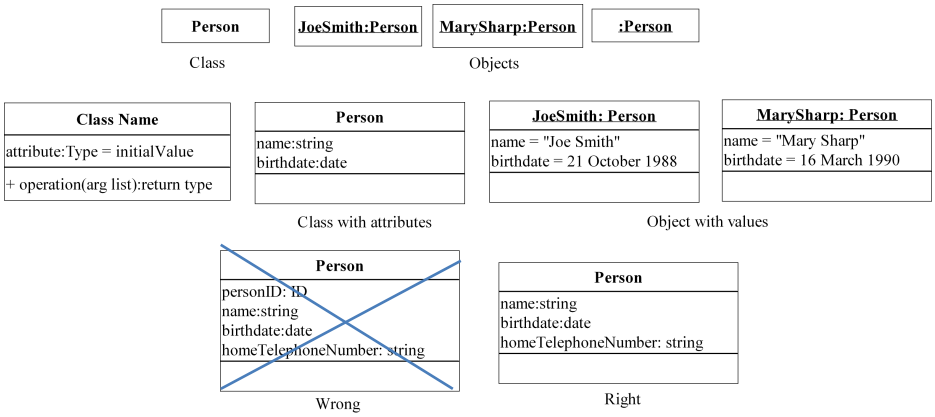
- \+ *public* operations
- \# *protected* operations
- \- *private* operations

Class and Objects



53

Class and Objects: Notations



Do not list object identifier; they are implicit in models

54

Object Relationships

- Association
  - Aggregation
  - Composition
- Inheritance
- Dependency

55

UML History

- OO languages appear mid 70's to late 80's (cf. Budd: communication and complexity)
- Between '89 and '94, OO methods increased from 10 to 50.
- Unification of ideas began in mid 90's.
  - Rumbaugh joins Booch at Rational '94
  - v0.8 draft Unified Method '95
  - Jacobson joins Rational '95
  - UML v0.9 in June '96

  } pre-UML

  - UML 1.0 offered to OMG in January '97
  - UML 1.1 offered to OMG in July '97
    - Maintenance through OMG RTF
  - UML 1.2 in June '98
  - UML 1.3 in fall '99
  - UML 1.5 http://www.omg.org/technology/documents/formal/uml.htm

  } UML 1.x

  - UML 2.0 underway http://www.uml.org/

  } UML 2.0

- IBM-Rational now has *Three Amigos*
  - Grady Booch - Fusion
  - James Rumbaugh – Object Modeling Technique (OMT)
  - Ivar Jacobson – Object-oriented Software Engineering: A Use Case Approach (Objectory)
  - (And David Harel - StateChart)
- Rational Rose http://www-306.ibm.com/software/rational/

Unified Modeling Language

- An effort by IBM (Rational) – OMG to standardize OOA&D notation
- Combine the best of the best from
  - Data Modeling (Entity Relationship Diagrams); Business Modeling (work flow); Object Modeling
  - Component Modeling (development and reuse - middleware, COTS/GOTS/OSS/…:)
- Offers vocabulary and rules for communication
- *Not* a process but a language

de facto industry standard

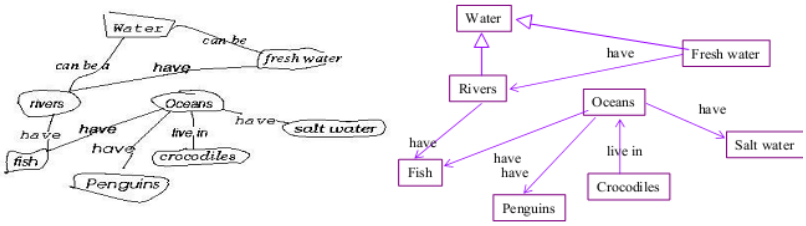
UML is for Visual Modeling

A picture is worth a thousand words!

- standard graphical notations: Semi-formal
- for modeling enterprise info. systems, distributed Web-based applications, real time embedded systems, …

- ***Specifying & Documenting:*** models that are precise, unambiguous, complete
  - UML symbols are based on well-defined syntax and semantics.
  - analysis, architecture/design, implementation, testing decisions.
- ***Construction:*** mapping between a UML model and OOPL.

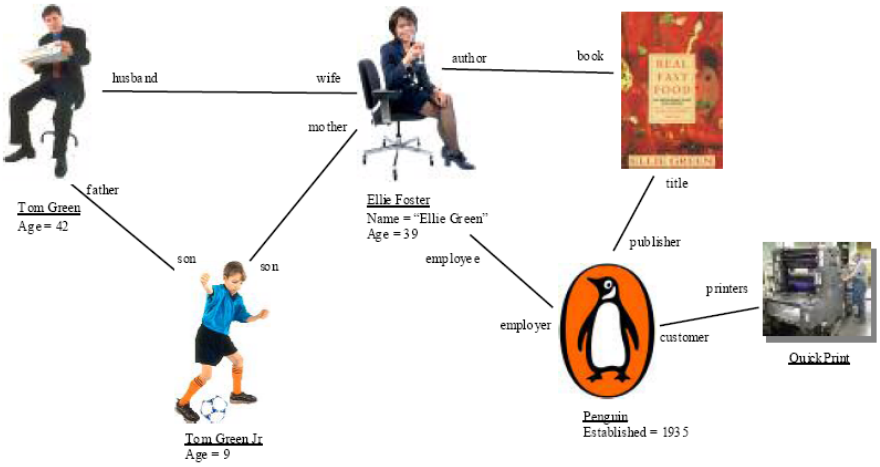
Three (3) basic BUILDING blocks of UML



- Things - important modeling concepts
- Relationships - tying individual things
- Diagrams - grouping interrelated collections of things and relationships

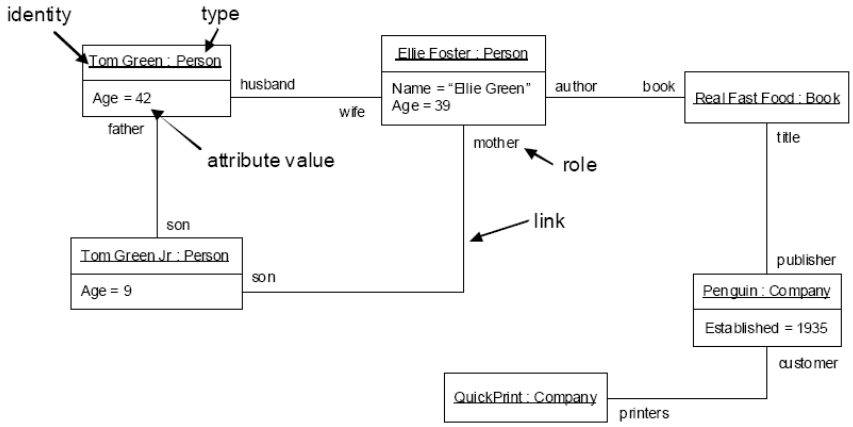
Just glance thru for now

Relationships between objects at some point



Object Diagrams

More formally in UML



A connected graph: Vertices are things; Arcs are relationships/behaviors.

UML 1.x: 9 diagram types.

Structural Diagrams

Represent the static aspects of a system.

- Class; Object
- Component
- Deployment

Behavioral Diagrams

Represent the dynamic aspects.

- Use case
- Sequence; Collaboration
- Statechart
- Activity

UML 2.0: 12 diagram types

Structural Diagrams

- Class; Object
- Component
- Deployment
- Composite Structure
- *Package*

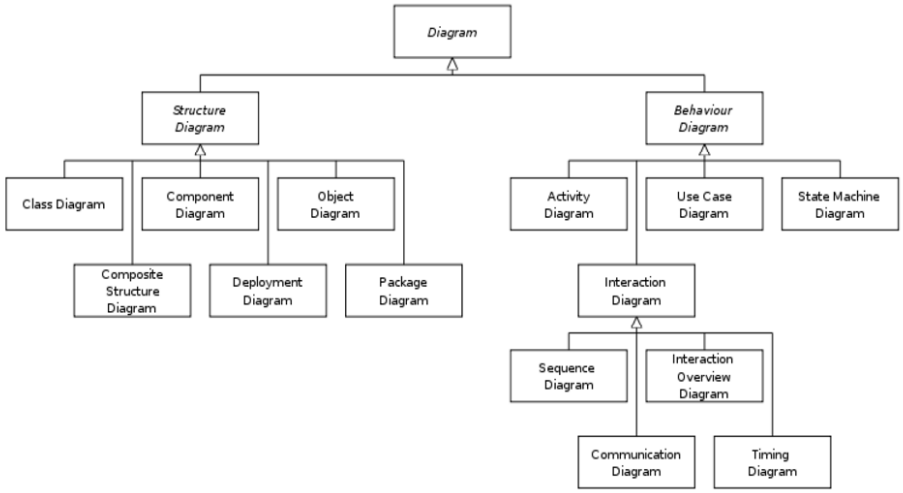
Behavioral Diagrams

- Use case
- Statechart
- Activity

Interaction Diagrams

- Sequence; *Communication*
- Interaction Overvie
- Timing

Unified Modeling Language (UML) - Diagrams



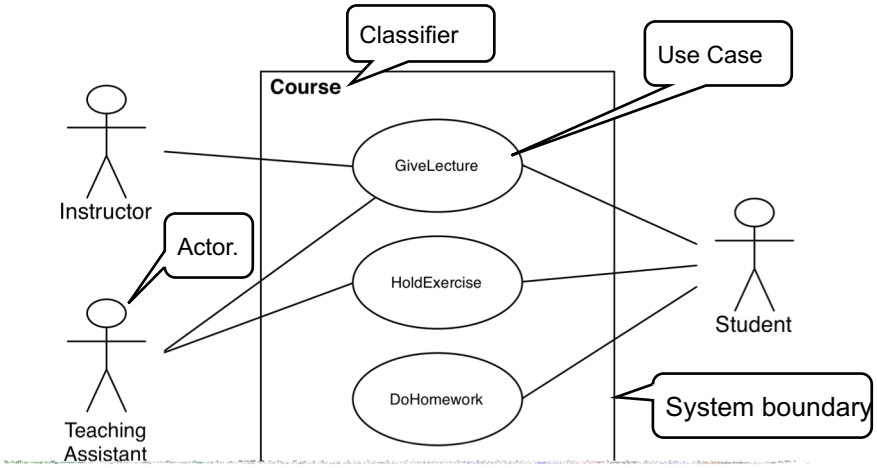
Source: Wikipedia

UML Core Conventions

- All UML Diagrams denote graphs of nodes and edges
  - Nodes are entities and drawn as rectangles or ovals
  - Rectangles denote classes or instances
  - Ovals denote functions
- Names of Classes are not underlined
  - SimpleWatch
  - Firefighter
- Names of Instances are underlined
  - myWatch:SimpleWatch
  - Joe:Firefighter
- An edge between two nodes denotes a relationship between the corresponding entities

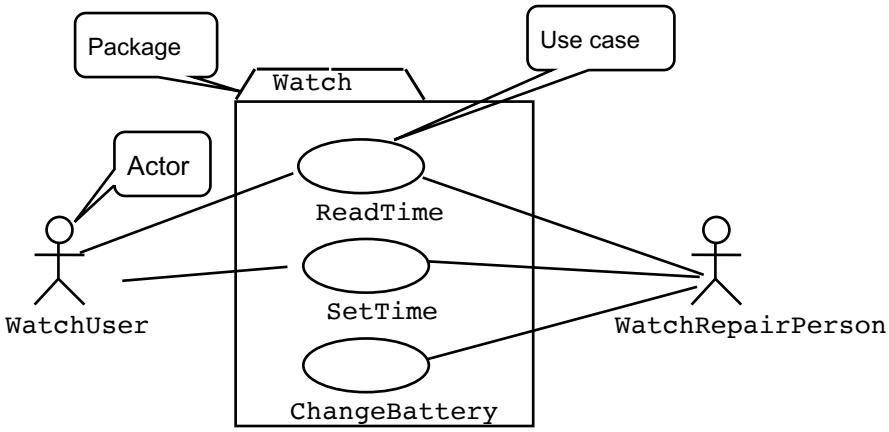
64

UML first pass: Use case diagrams



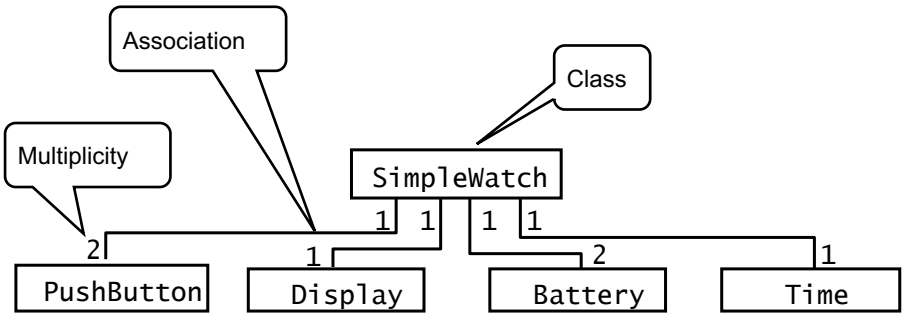
Use case diagrams represent the functionality of the system from user's point of view

Historical Remark: UML 1 used packages



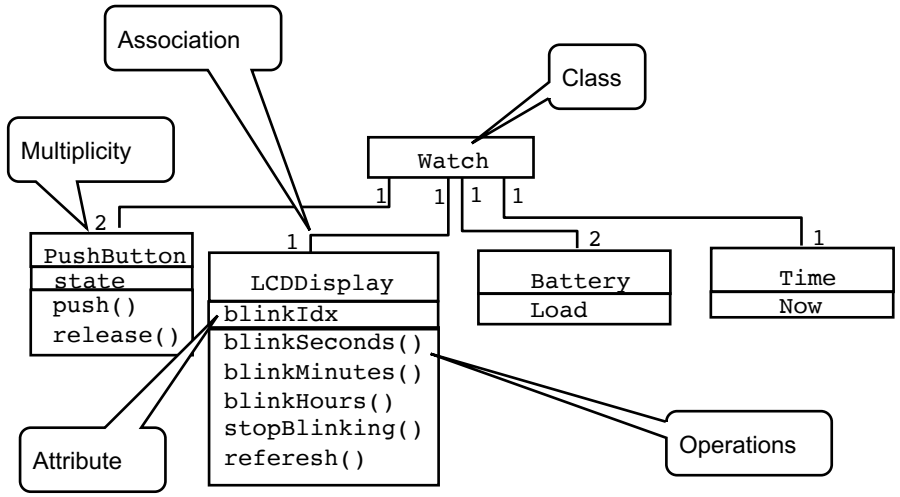
Use case diagrams represent the functionality of the system from user's point of view

UML first pass: Class diagrams

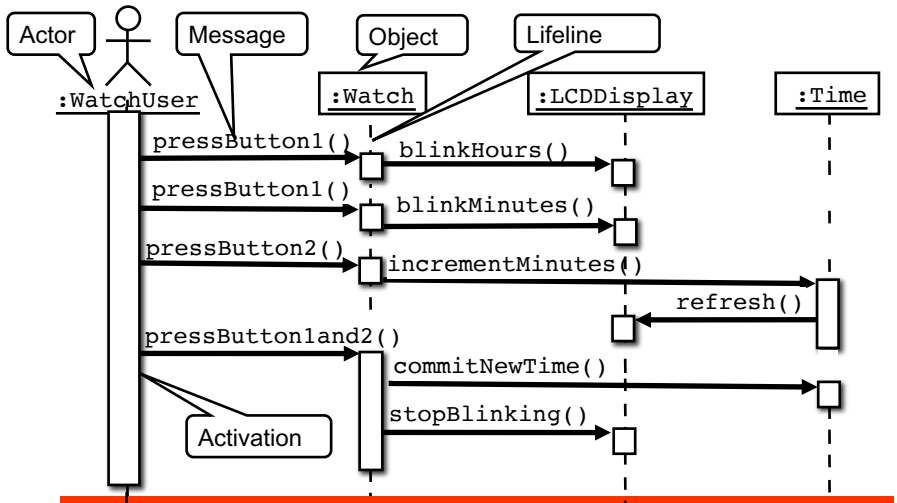


Class diagrams represent the structure of the system

UML first pass: Class diagrams

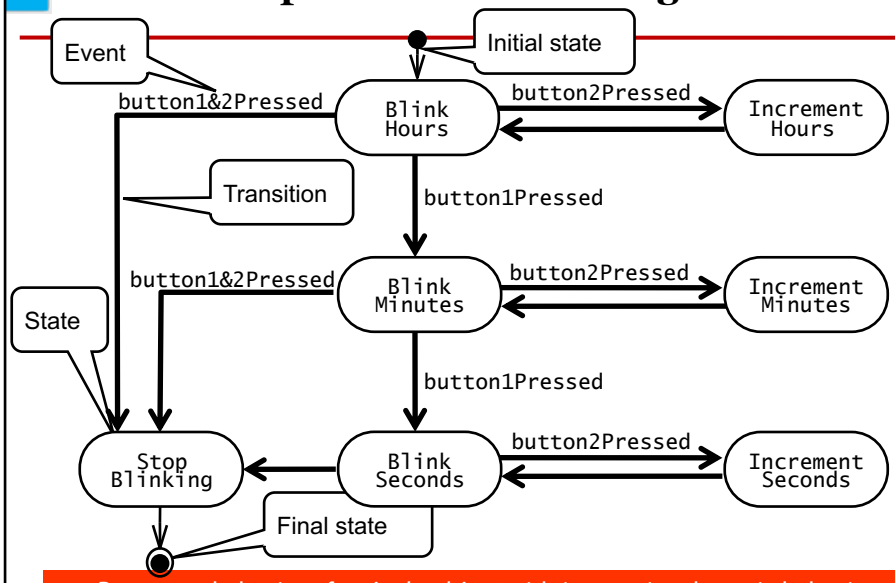


UML first pass: Sequence diagram



Sequence diagrams represent the behavior of a system as messages ("interactions") between *different objects*

UML first pass: Statechart diagrams



Represent behavior of *a single object* with interesting dynamic behavior.

UML Basic Notation Summary

- UML provides a wide variety of notations for modeling many aspects of software systems
- Today we concentrated on a few notations:
  - Functional model: Use case diagram
  - Object model: Class diagram
  - Dynamic model: Sequence diagrams, state chart

Diagrams in UML

The UTD wants to computerize its registration system

- The Registrar sets up the curriculum for a semester
- Students select 3 core courses and 2 electives
- Once a student registers for a semester, the billing system is notified so the student may be billed for the semester
- Students may use the system to add/drop courses for a period of time after registration
- Professors use the system to set their preferred course offerings and receive their course offering rosters after students register
- Users of the registration system are assigned passwords which are used at logon validation

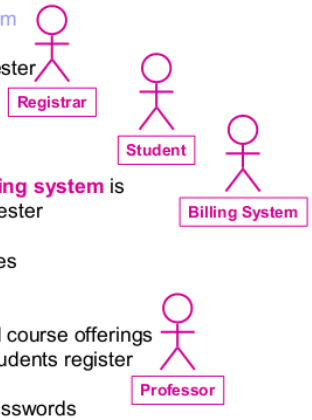
What's most important?

Diagrams in UML – Actors in Use case

- An actor is someone or some thing that must interact with the system under development

The UTD wants to computerize its registration system

- The **Registrar** sets up the curriculum for a semester
- **Students** select 3 core courses and 2 electives
- Once a student registers for a semester, the **billing system** is notified so the student may be billed for the semester
- Students may use the system to add/drop courses for a period of time after registration
- **Professors** use the system to set their preferred course offerings and receive their course offering rosters after students register
- Users of the registration system are assigned passwords which are used at logon validation

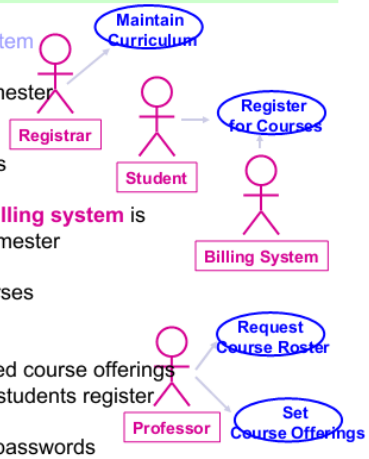


Diagrams in UML – Use cases in Use case diagram

- A use case is a sequence of interactions between an actor and the system

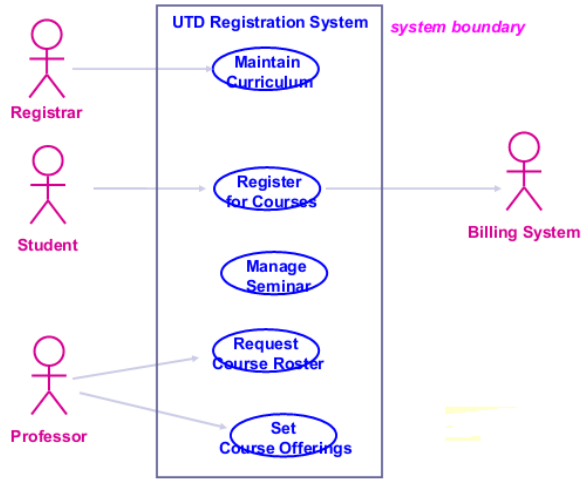
The UTD wants to computerize its registration system

- The **Registrar** sets up the curriculum for a semester
- **Students** select 3 core courses and 2 electives
- Once a student registers for a semester, the **billing system** is notified so the student may be billed for the semester
- Students may use the system to add/drop courses for a period of time after registration
- **Professors** use the system to set their preferred course offerings and receive their course offering rosters after students register
- Users of the registration system are assigned passwords which are used at logon validation



Diagrams in UML – Use case Diagram

- Use case diagrams depict the relationships between actors and use cases



Next Lecture: *We Discuss*

- Use Case Diagram
- Sequence Diagram
- Collaboration Diagram
- Statechart
- Activity Diagram
- Class Diagram
- Package Diagram
- Component/Deployment Diagram