Introduction to

UML

Part I

What is UML?

- Unified Modeling Language, a standard language for designing and documenting a system in an object-oriented manner.
- It's a language by which technical architects can communicate with developers.
- It's a language by which one can express design of a software architecture.
- It's a blue print of your source code.
- It has nine diagrams which can be used in design document to express design of software architecture.

- Introduction to UML
- What is UML?
- Motivations for UML
- Types of UML diagrams
- UML syntax
- Descriptions of the various diagram types
- Rational Rose (IBM.. More in practical lecture)

What is UML?

- A standardized, graphical "modeling language" for communicating software design.
- Allows implementation-independent specification of:
  - user/system interactions (required behaviors)
  - partitioning of responsibility (OO)
  - integration with larger or existing systems
  - data flow and dependency
  - operation orderings (algorithms)
  - concurrent operations
- Pretty pictures.
- UML is not "process". (That is, it doesn't tell you how to do things, only what you should do.)

Motivations for UML

- UML is a fusion of ideas from several precursor modeling languages.
- We need a modeling language to:
  - help develop efficient, effective and correct
  - designs, particularly Object Oriented designs.
  - communicate clearly with project stakeholders (concerned parties: developers, customer, etc).
  - give us the "big picture" view of the project.

Types of UML diagrams

- There are different types of UML diagram, each with slightly different syntax rules:
  - use cases.
  - class diagrams.
  - sequence diagrams.
  - package diagrams.
  - state diagrams
  - activity diagrams
  - deployment diagrams.

UML syntax, 1

- Actors: a UML actor indicates an interface (point of interaction) with the system.
  - We use actors to group and name sets of system interactions.
  - Actors may be people, or other systems.
  - An actor is NOT part of the system you are modeling.
  - An actor is something external that your system has to deal with.
- Boxes: boxes are used variously throughout UML to indicate discrete elements, groupings and containment.

UML syntax, 2

- Arrows: arrows indicate all manner of things, depending on which particular type of UML diagram they're in. Usually, arrows indicate flow, dependency, association or generalization.
- Cardinality: applied to arrows, cardinalities show relative numerical relationships between elements in a model: 1 to 1, 1 to many, etc.

UML syntax, 3

- Constraints: allow notation of arbitrary constraints on model elements. Used, for example, to constrain the value of a class attribute (a piece of data).
- Stereotypes: allow us to extend the semantics of UML with English. A stereotype is usually a word or short phrase that describes what a diagram element does. That is, we mark an element with a word that will remind us of a common (stereotypical) role for that sort of thing.

UML diagrams: Use Cases

- A use case encodes a typical user interaction with the system. In particular, it:
  - captures some user-visible function.
  - achieves some concrete goal for the user.
- A complete set of use cases largely defines the requirements for your system: everything the user can see, and would like to do.
- The granularity of your use cases determines the number of them (for your system). A clear design depends on showing the right level of detail.
- A use case maps actors to functions. The actors need not be people.

Use-Cases

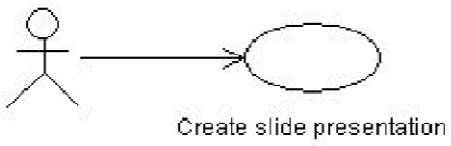
- A collection of *user scenarios* that describe the thread of usage of a system
- Each scenario is described from the point-of-view of an "*actor*"—a person or device that interacts with the software in some way

- Each scenario answers the following questions:
  - Who is the primary actor, the secondary actor (s)?
  - What are the actor's goals?
  - What preconditions should exist before the story begins?
  - What main tasks or *functions* are performed by the actor?
  - What extensions might be considered as the story is described?
  - What variations in the actor's interaction are possible?
  - What system information will the actor acquire, produce, or change?
  - Will the actor have to inform the system about changes in the external environment?
  - What information does the actor desire from the system?
  - Does the actor wish to be informed about unexpected changes?

Use Case Example I

Use case examples, 1

(High-level use case for powerpoint.)



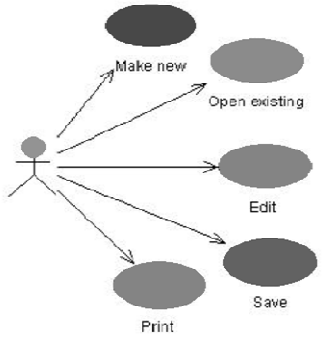
About the last example…

- Although this is a valid use case for PowerPoint, and it completely captures user interaction with PowerPoint, it's too vague to be useful.

Use Case Example II

Use case examples, 2

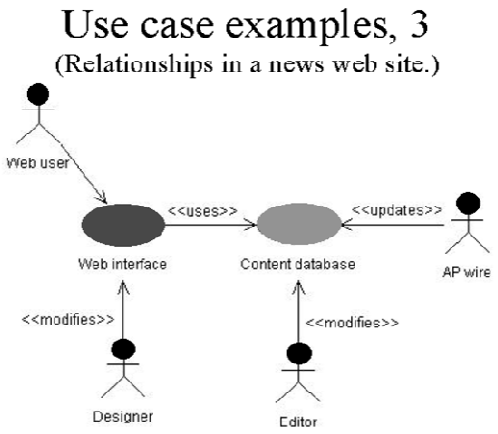
(Finer-grained use cases for powerpoint.)



About the last example…

- The last example gives a more useful view of PowerPoint (or any similar application).
- The cases are vague, but they focus your attention and the key features, would help in developing a more detailed requirements specification.
- It still doesn't give enough information to characterize PowerPoint, which could be specified with tens or hundreds of use cases (though doing so might not be very useful either).

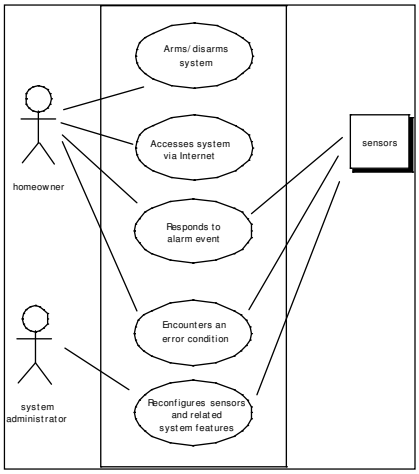
Use Case Example III



About Example III

- It's a more complicated and a realistic use case diagram.
- It captures several key use cases for the system.
- Note the multiple actors. In particular, Associated Press 'AP wire' is an actor, with an important interaction with the system, but is not a person (or even a computer system, necessarily).
- The notes between << >> marks are *stereotypes:*
  - identifiers added to make the diagram more informative.
  - Here they differentiate between different roles (i.e., different meanings of an arrow in this diagram).

Use-Case Diagram for Alarm System



Developing a Use-Case

- What are the main tasks or functions that are performed by the actor?
- What system information will the the actor acquire, produce or change?
- Will the actor have to inform the system about changes in the external environment?
- What information does the actor desire from the system?
- Does the actor wish to be informed about unexpected changes?

Object Oriented (OO)

More UML later, now on to OO

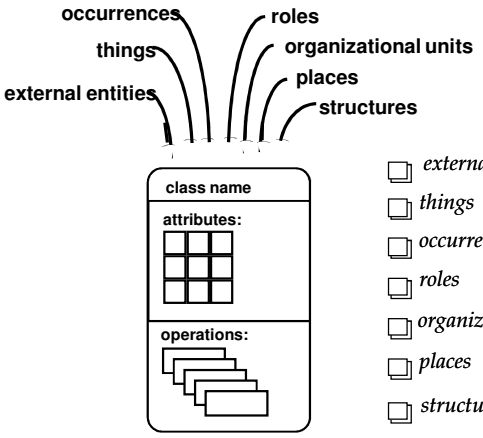
Object-Oriented Concepts

- Key concepts:
  - Classes and objects
  - Attributes and operations
  - Encapsulation and instantiation
  - Inheritance

Classes

- object-oriented thinking begins with the definition of a class, often defined as:
  - template, generalized description, "blueprint" ... describing a collection of similar items
- a metaclass (also called a superclass) establishes a hierarchy of classes
- once a class of items is defined, a specific instance of the class can be identified

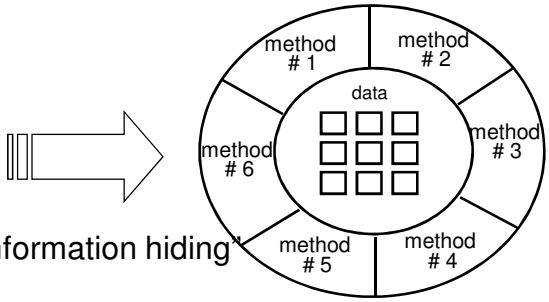
What is a Class?



- *external entities* (printer, user, sensor)
- *things* (e.g, reports, displays, signals)
- *occurrences or events* (e.g., interrupt, alarm)
- *roles* (e.g., manager, engineer, salesperson)
- *organizational units* (e.g., division, team)
- *places* (e.g., manufacturing floor)
- *structures* (e.g., employee record)

Encapsulation/Hiding

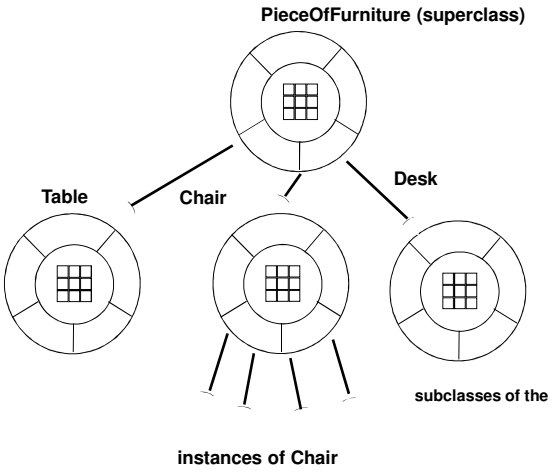
The object encapsulates both data and the logical procedures required to manipulate the data



Achieves "information hiding"

A method is invoked via message passing.
An executable procedure that is encapsulated in a class and is designed to operate on one or more data attributes that are defined as part of the class.

Class Hierarchy



Class Diagram

From the *SafeHome* system ...