# Lecture 14 — Unified Modelling Language (UML)

Patrick Lam & Jeff Zarnett
p.lam@ece.uwaterloo.ca & jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

November 14, 2015

# What is UML?

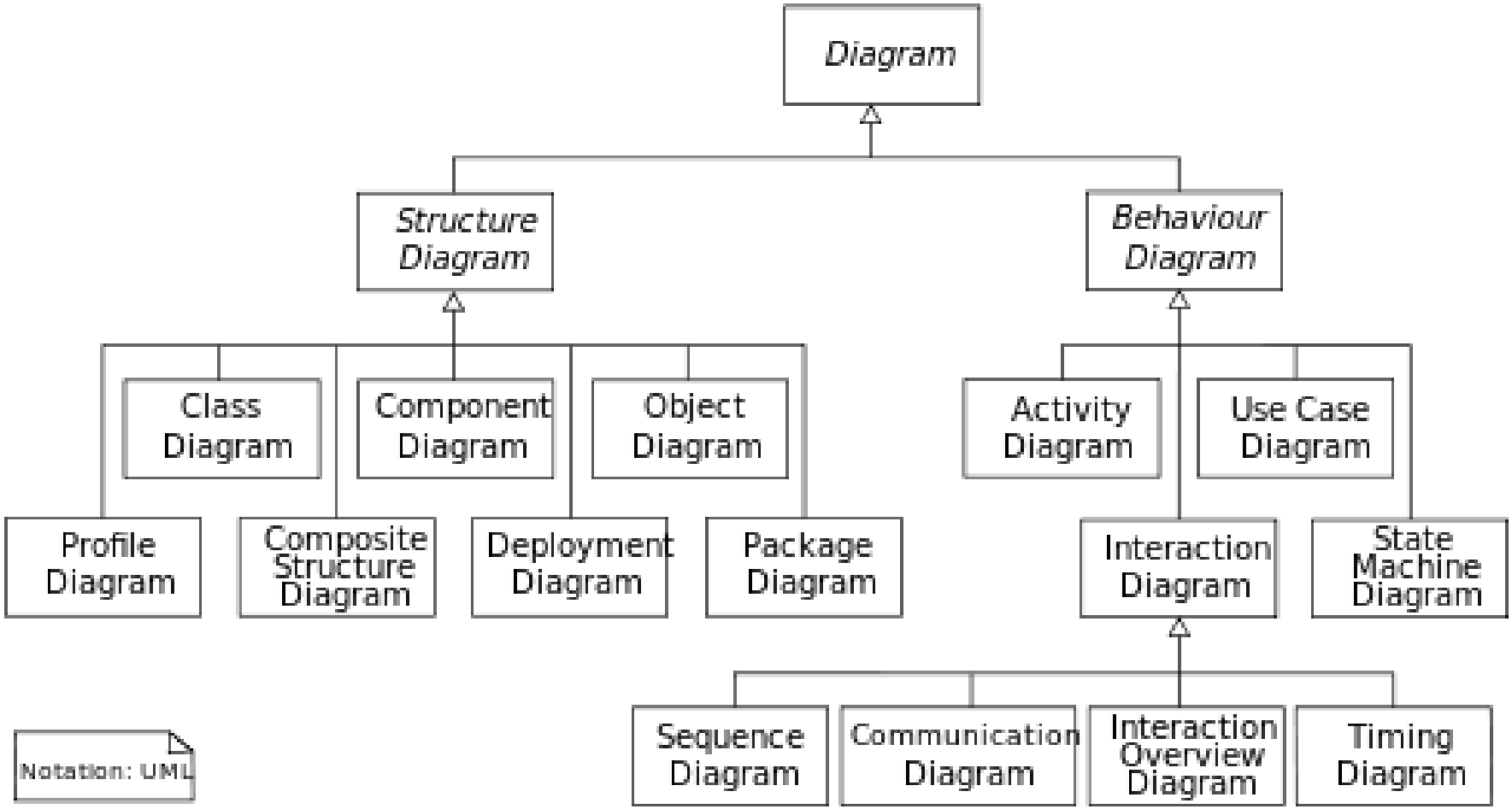
Unified Modelling Language (UML):

> *specify and document architecture of large object-oriented software systems, using diagrams.*

UML version 2.2 defines 13 diagrams to summarize:

> intended structure (e.g. classes); behaviour; and interactions of components.

# UML Diagrams



# Why UML

How much is UML used in industry? Depends where you work.
Some places require it; others forbid it.

Standard way to communicate with other programmers.

Mostly, but not only, for Object-Oriented Programming.

# Resources

Lots of information on the Web, e.g.
`http://edn.embarcadero.com/article/31863`.

Some books:

Grady Booch, James Rumbaugh, and Ivar Jacobson. *Unified Modeling Language User Guide,* 2nd Edition, Addison-Wesley, 2005.

Dan Pilone and Neil Pitman. *UML 2.0 in a Nutshell*. O'Reilly Media, 2005.

# UML Benefits

UML is a well-known language for software designs. In particular, it:

- is a visual language; UML models are diagrams, so easy to glance at.
- models are largely self-documenting.
- is agnostic in terms of software processes or development lifecycles.
- is an open standard (not owned by any one company).
- is good for object-oriented languages, which are quite common. in industry.

# Complaints about UML

Here are some potential complaints about UML.

- all syntax, no meaning.
- contains redundant and infrequently-used constructs.
- complex and difficult to learn.
- only works for object-oriented languages.
- tools don't play nicely together.

# Categories of UML Diagrams

Three main types:

- Structure Diagrams—static application structure.
- Behaviour Diagrams—usage of components, activity of components, and finite-state machines summarizing components' states.
- Interaction Diagrams—how do components interact?: Communication and synchronization, potentially timed.

# Tool Support

Software tools can create UML diagrams.

e.g. Microsoft Visio, `dia`.

Tools can generate code from UML and UML from code.
Round-tripping: going both ways.

Tools can also (sort of) generate test cases and test suites from UML diagrams.

Part I

# UML Class Diagrams

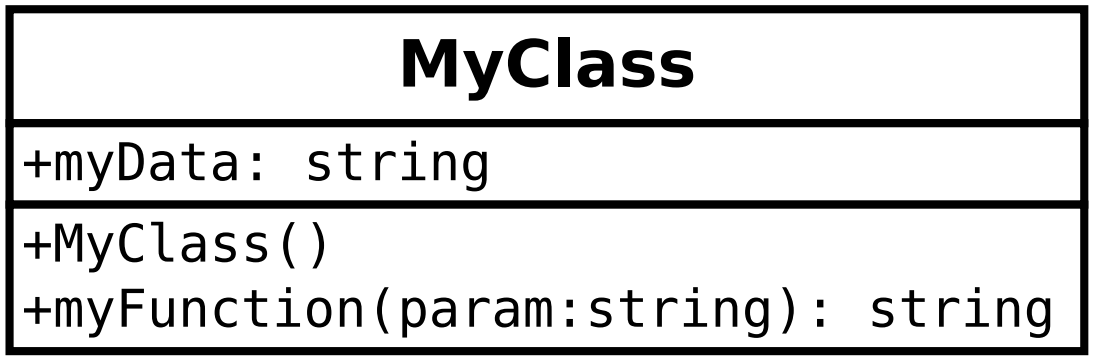
# About Class Diagrams

The basic UML diagram.

Describes the members of a class:

- attributes/fields;
- operations (constructors, destructors, methods, indexers, and properties).

# Basic Class Diagram Example; Visibility Symbols



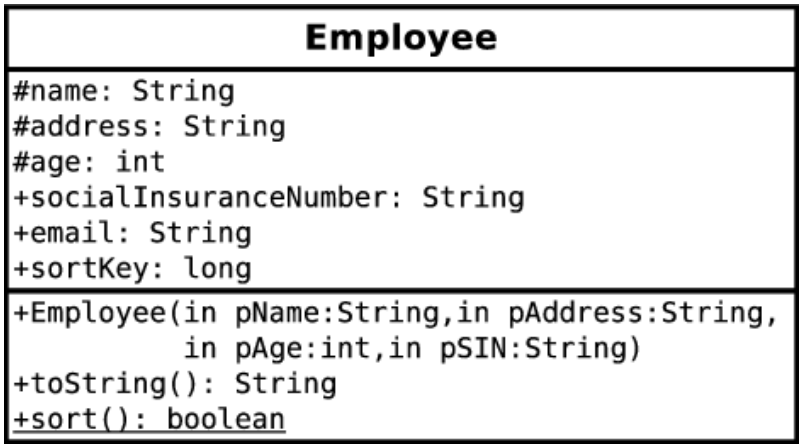

- A + (plus sign) = public visibility (seen above).
- A - (minus sign) = private visibility.
- A # (hash sign) = protected visibility.

# Other UML Symbols

UML also uses the following symbols:

- A $\sim$ (tilde) indicates a destructor (in front of an operation) or package (as a visibility symbol).
- An `..` (ellipsis) indicates a range of values.
- A `:` (colon) separates a name from a type.
- A `,` (comma) separates items in a set.

# Another UML Class Diagram




```
class Employee {
  protected String name;
  protected String address;
  protected int age;
  public String socialInsuranceNumber;
  public String email;
  public long sortKey;

  public Employee(String pName, String pAddress,
                  int pAge, String pSIN) { /* ... */ }
  @Override public String toString() { /* ... */ }
  public static boolean sort() { /* ... */ }
}
```

# Class Hierarchies in UML

UML represents relationships between classes and instances.

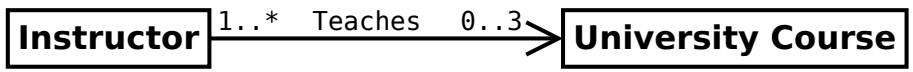
Between instances of classes:

- association: a relationship between instances of 2 classes.
- aggregation: a type of association, typically between a collection or container instance and contents.
- composition: another type of association, typically between a container and its contents.
  For a composition, the contents don't make any sense if the container isn't around.

But: a generalization relates base classes (not instances) and their derived classes.

# Associations: “has-a” links between classes

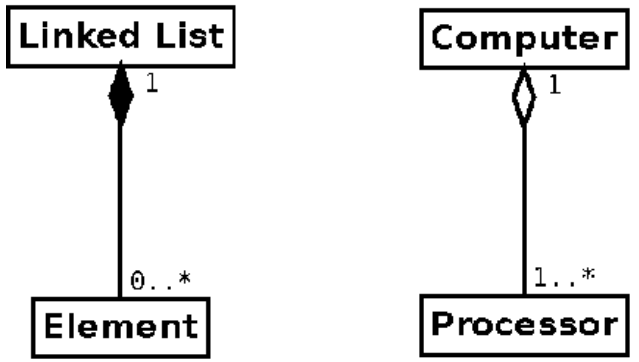
Typically one of the classes can call methods on the other.




- Instructors and courses are associated, but there is no subtype relationship, and the types aren’t collection types.
- Instructors teach between 0 and 3 courses per term.
- Each course has at least 1 instructor.
- A method that the instructor might call on the course might be `giveLecture()`.

# Aggregations

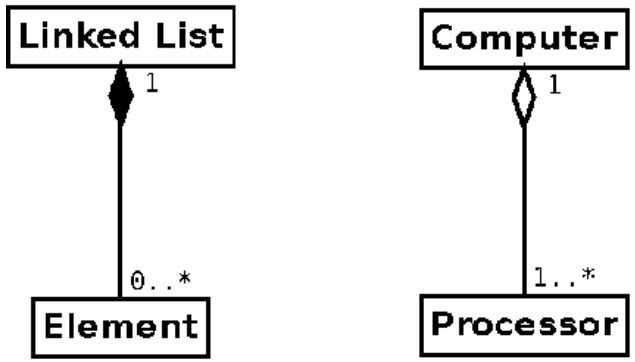
Specialized types of associations;
also represent “has-a” relationships, but more “part-whole” relationships.




Linked List contains 0 or more Elements.
Elements belong to only one Linked List.

Composition means that we are saying that Elements may not exist independently of a Linked List.

# Compositions

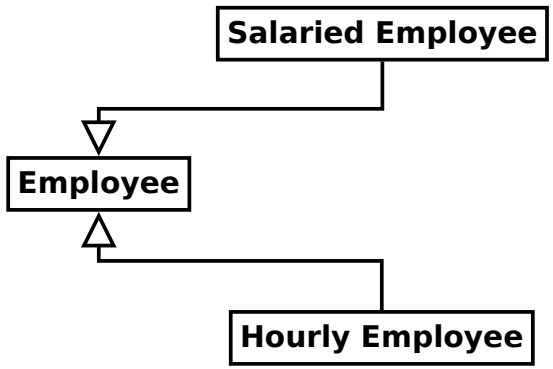



Computers contain 1 or more Processors, but Processors only belong to 1 Computer.

In an aggregation, we are stating that Processors do exist independently of a Computer.

# Generalizations

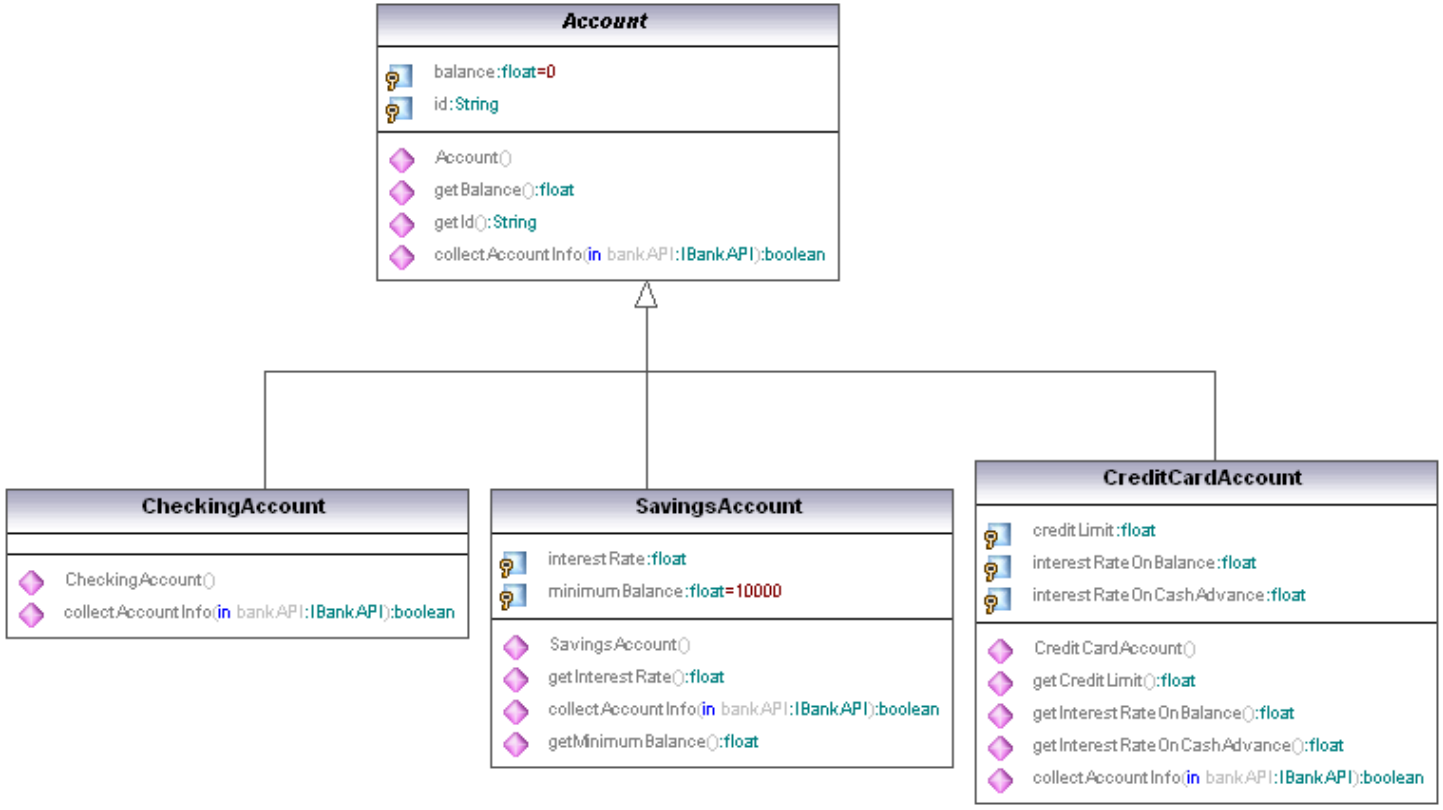
Relationships between classes.




Every Salaried Employee and Hourly Employee is an Employee,
so Employees are generalizations of the Salaried and Hourly Employee classes.

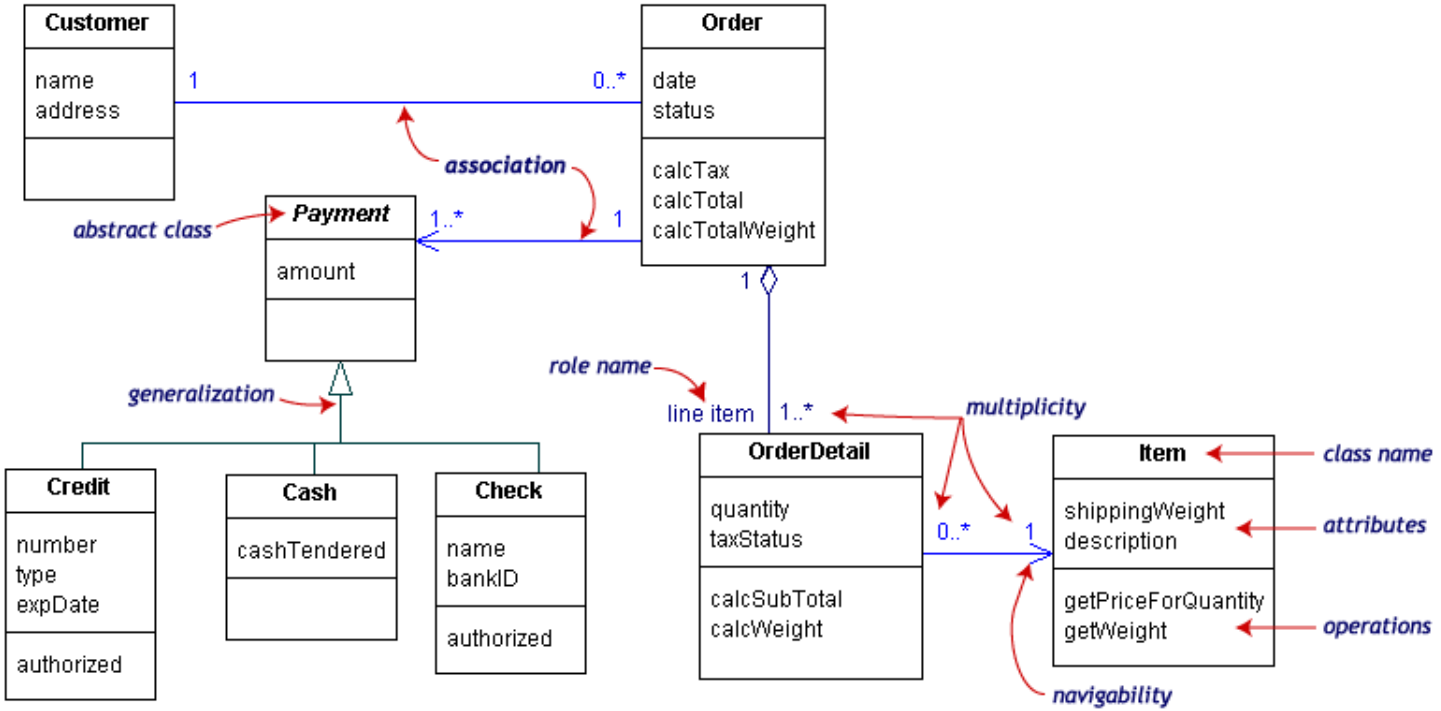
In code: Salaried Employee and Hourly Employee would be subclasses of ("extend") Employee.

From
`http://www.altova.com/umodel/class-diagrams.html`
(accessed March 10, 2011).:




Note the member field initializations and the generalization link.

From `http://edn.embarcadero.com/article/31863` (accessed March 10, 2011).



# State Diagrams: Learning Outcome

You should be able to:

- read a description of a finite state machine (either in code or in text) and produce a syntactically correct UML state diagram summarizing that design.

(Examples are from *UML Distilled, Second Edition*, by Martin Fowler with Kendall Scott.)

# UML State Diagram: Order Processing System

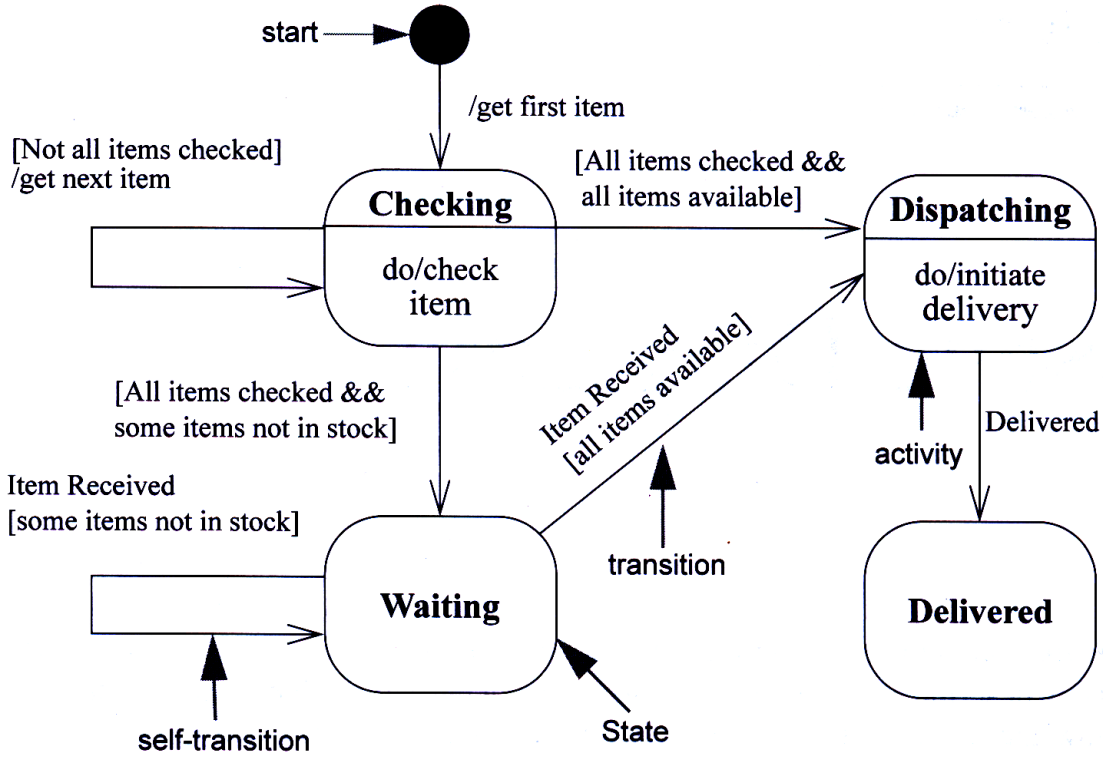



**Figure 8-1:** *State Diagram*

# UML State Diagram: States

Example: states of an order in an order processing system.
Order object has 4 states, represented by boxes.

Upper half of the box = name,
e.g. “checking”, “dispatching”, “waiting”, “delivered”.
Lower half = what the activity in that state is.

“Dispatching” state initiates the delivery of the order.

In ECE155, you can always put “do” before the slash and the activity after the slash.

State have names and may have activities.
Objects carry out activities, and these activities take some amount of time.

# UML State Diagram: Transitions

Edges = transitions between states, optionally labelled.
Three parts to a transition label: *Event [Guard] / Action*.

An object takes transition if event occurs and guard is true.
Example: self-transition from "checking" contains guard "[not all items checked]" and performs action "get next item".

Event: something happening to an object, e.g. "Item Received".
Guard: logical condition that states the conditions under which the event may occur (e.g. "[some items not in stock]").

Objects carry out actions, e.g. "get next item".
Actions must occur quickly and be uninterruptible.

UML state machines are deterministic: at most one transition should be activated at once.

# UML State Diagram: Superstates

State diagrams may get too complicated.

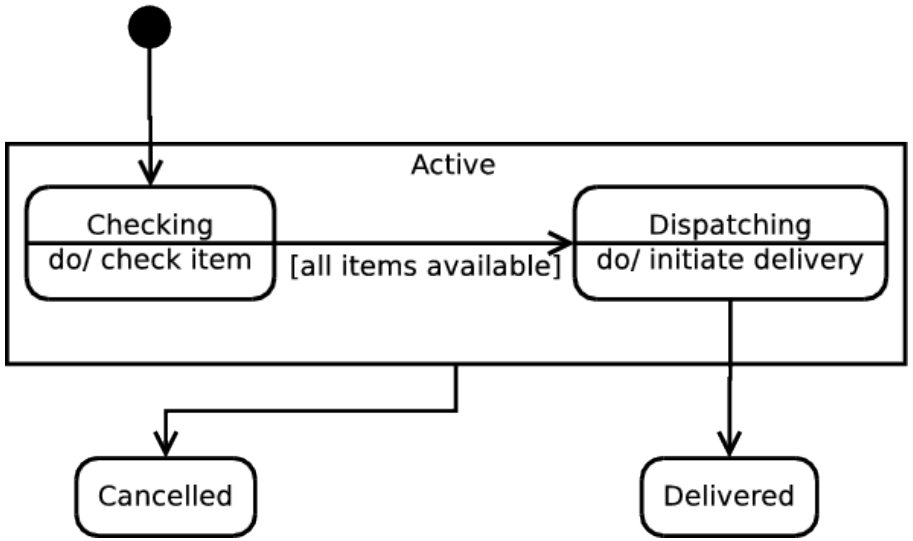
UML's solution: hierarchical nesting:

- can help write clearer diagrams;
- allow understanding system's higher-level structure.

Concept: superstate.

- may contain nested states and transitions;
  make up a state machine for some subset of the entire machine's behaviour.

# UML State Diagram: Superstate Example




(http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/state.htm. Accessed July 3, 2011.)

Superstate "active" contains "checking" and "dispatching".

Note transitions from states inside superstate,
e.g. "dispatching" to "delivered",
plus transition from the superstate to the "cancelled" state.
So, either of the states in the superstate can get to "cancelled".

Superstate transition = transitions from both "dispatching" and "checking" to "cancelled".

# UML State Diagram: Timing

Recall: timing is important for embedded systems.

Your documentation should explain what "quickly"
(actions happen quickly) means.
UML doesn't say; it depends on the system.

Can also write an event that happens after some time,
e.g. "after (20 minutes)".

Part II

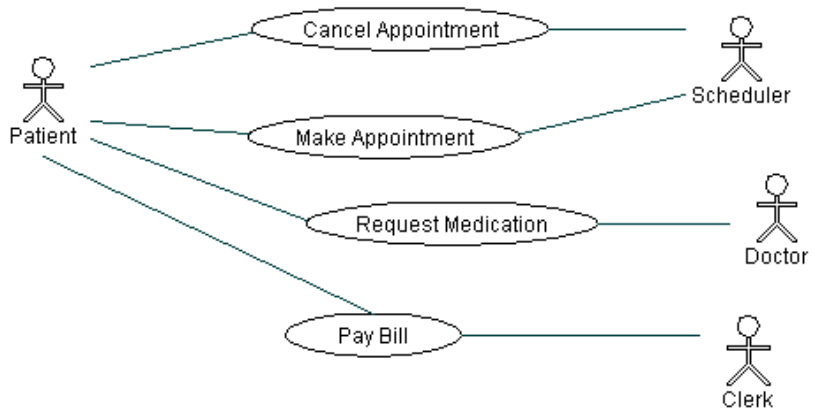
# UML Use Case Diagrams

# Use cases in UML; actors

Recall: a use case describes a user's interaction with a system.

Actors represent users (or other systems).

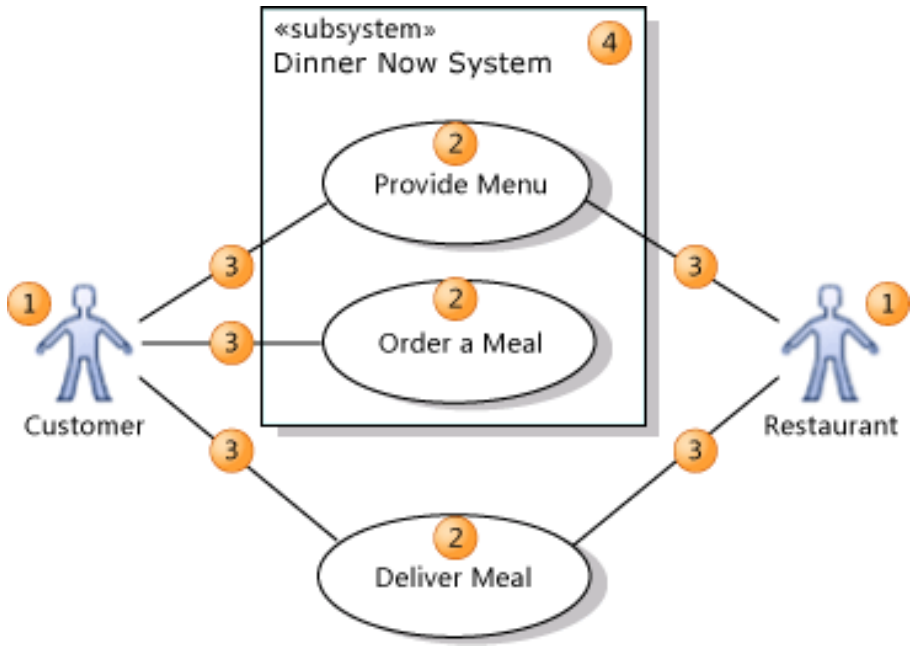
# Example: Multiple actors




It's just a coincidence that each use case has two actors.

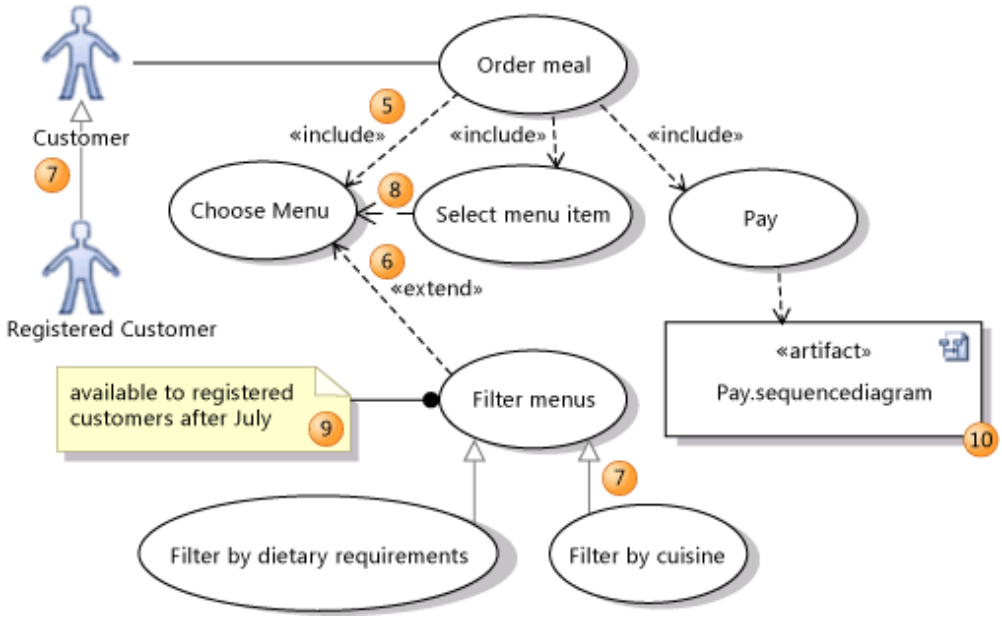
# Another use case example

`http://msdn.microsoft.com/en-us/library/dd409427%28VS.100%29.aspx` (accessed March 10, 2011).




Components: 1) actors, 2) use cases, 3) communication, and, new to this figure, 4) subsystems or components.

# Use cases and actors




Here we have: 5) inclusion stereotypes on dependencies; 6) extension stereotypes on dependencies; 7) inheritance relationships between use cases; 8) plain dependencies; 9) comments; and 10) references to other artifacts.

# Things in use cases

- A *communication* (solid line) represents a relationship between an actor and a use case.
- An *inclusion* (dashed line, <<include>> stereotype) represents an invocation relationship between use cases; the first use case must call the second one.
- An *extension* (dashed line,<<extend>> stereotype) represents an optional invocation of a use case.
- A *generalization* or *specialization* represents a case where an actor or use case inherits from another actor or use case.

# UML use cases: why?

Can be useful for:

1. documenting essential features (requirements) of a software system;
2. communicating system behaviour to clients;
3. generating appropriate test cases for scenarios.

# Part III

# UML Sequence Diagrams

# About UML Sequence Diagrams

UML sequence diagrams:

*express system behaviour as a sequence of events and activities.*

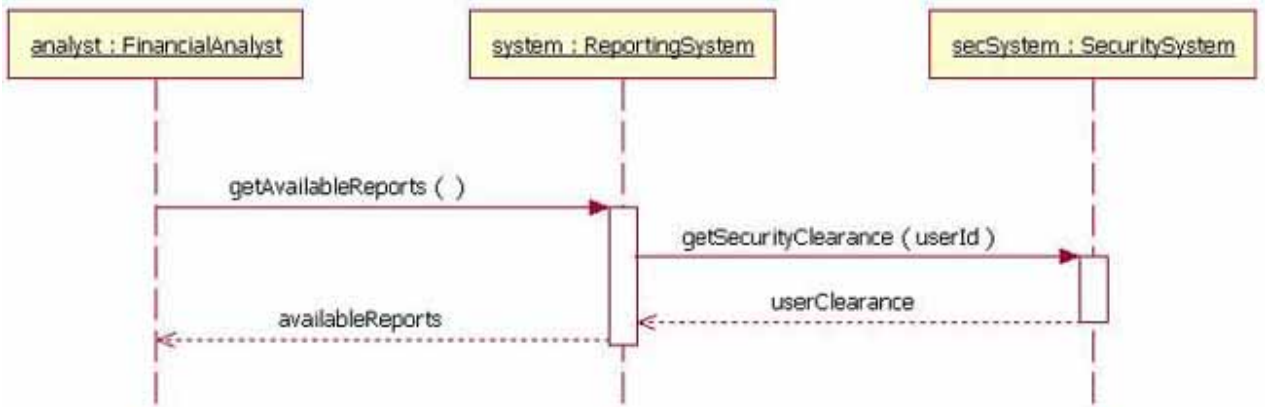
Things you'll find:
objects, lifelines, messages, action boxes, and gates.

# Elements in UML Sequence Diagrams

- Instances of objects: boxes at the top of the diagram.
- Lifelines (vertical dashed lines, extending downwards from instances of the objects): denote the passing of time.
- Messages (horizontal lines): denote communication between objects.
- Action boxes (boxes on top of lifelines): denote the occurrence of activities.
- Gates (filled circles on the boundary of the diagram): denote occurrence of external events.

# Sequence diagram, financial reporting system

(IBM, *UML basics: The sequence diagram*, `http://www.ibm.com/developerworks/rational/library/3101.html`, accessed March 10, 2011).



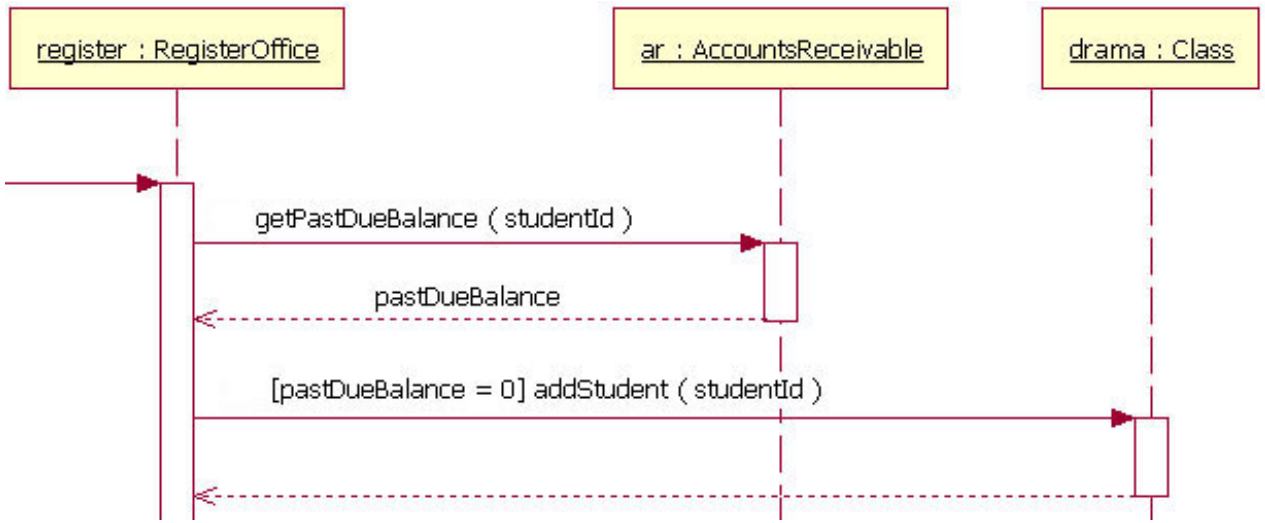

Solid lines are initiating messages.

Dashed lines are responses.

Also: synchronous messages (solid arrowhead—all initiating messages in the above example)
and asynchronous messages (stick arrowhead—responses).

You can also use: actors, destroy elements, scenario elements, timer elements.

# Sequence diagram, student records




Note guards on messages; must be satisfied before the message gets sent.

# Sequence diagram: when?

UML sequence diagrams help document messages going between different instances of objects,
including the ordering in which these events occur (for synchronous events).

Exercise judgment to include the right level of detail.

- too much detail makes the model difficult to understand;
- too little detail doesn't document anything.

You can use more than one UML sequence diagram to document a large system.