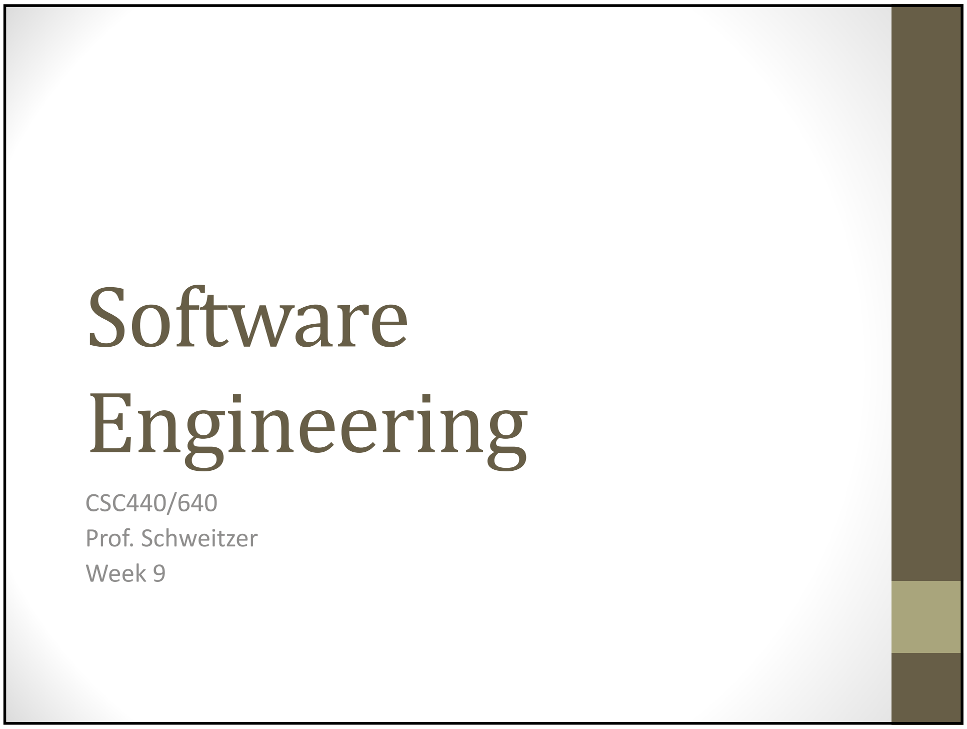
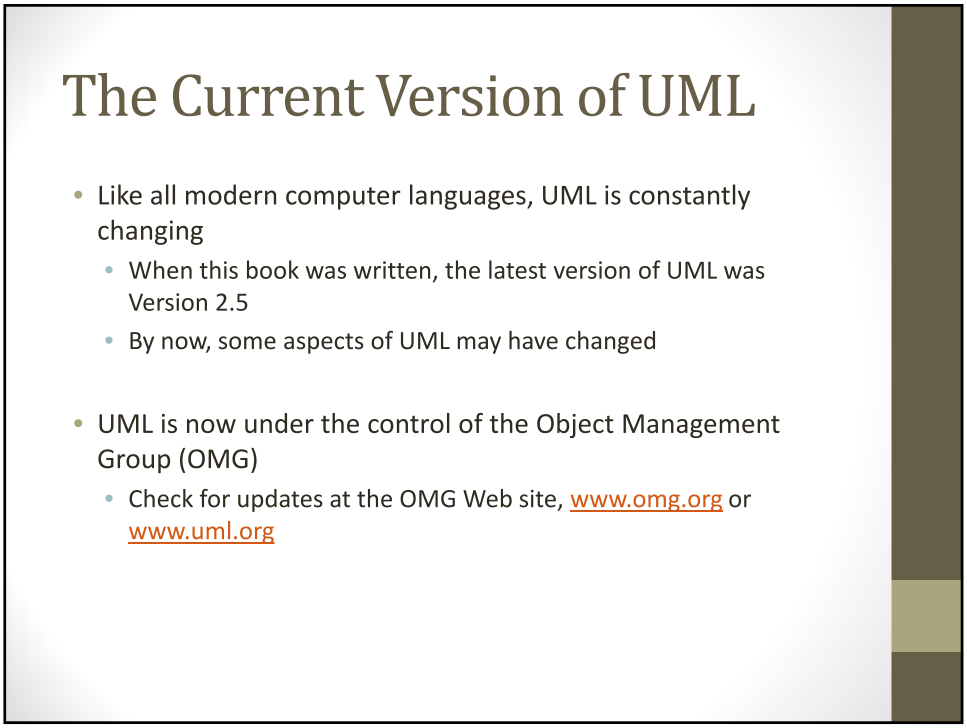


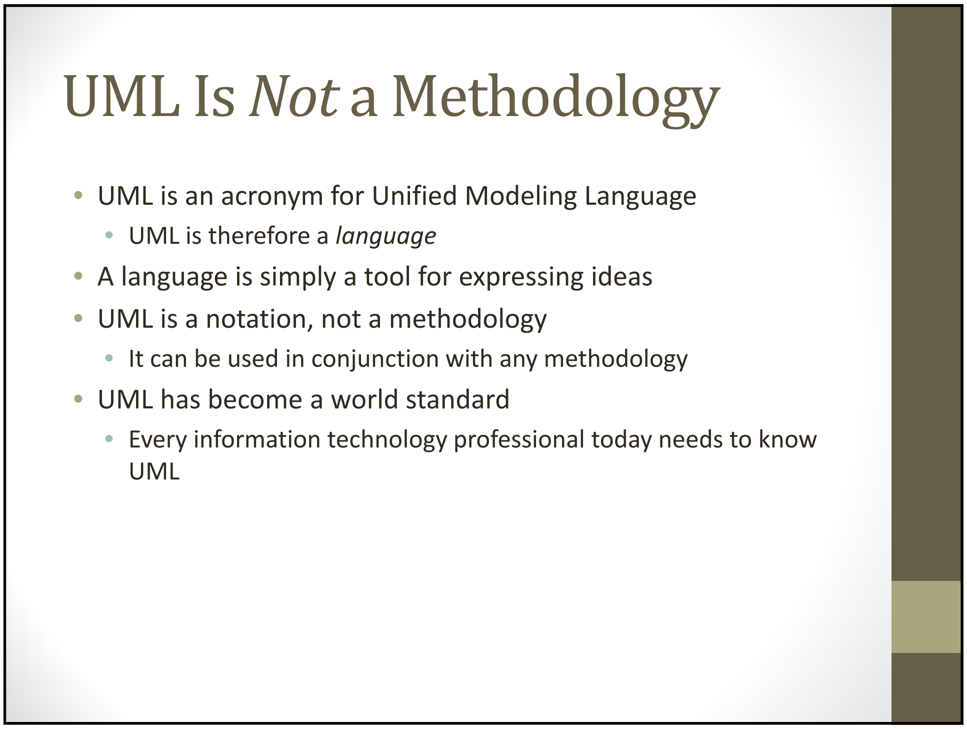
Software Engineering

CSC440/640
Prof. Schweitzer
Week 9



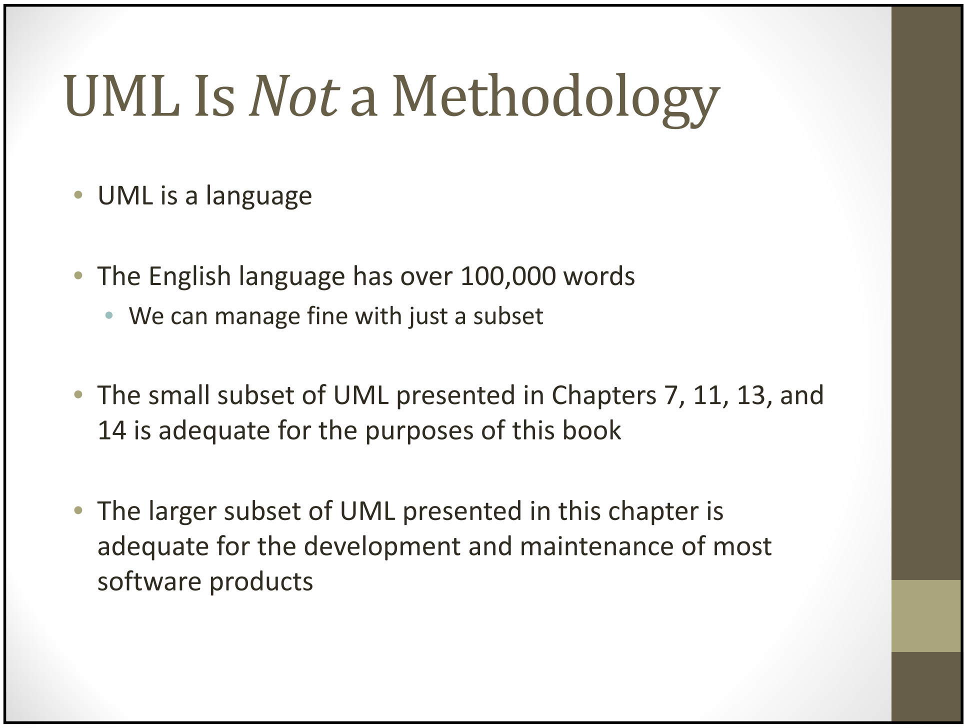
The Current Version of UML

- Like all modern computer languages, UML is constantly changing
  - When this book was written, the latest version of UML was Version 2.5
  - By now, some aspects of UML may have changed
- UML is now under the control of the Object Management Group (OMG)
  - Check for updates at the OMG Web site, www.omg.org or www.uml.org



UML Is *Not* a Methodology

- UML is an acronym for Unified Modeling Language
  - UML is therefore a *language*
- A language is simply a tool for expressing ideas
- UML is a notation, not a methodology
  - It can be used in conjunction with any methodology
- UML has become a world standard
  - Every information technology professional today needs to know UML

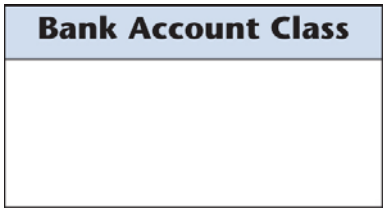


UML Is *Not* a Methodology

- UML is a language
- The English language has over 100,000 words
  - We can manage fine with just a subset
- The small subset of UML presented in Chapters 7, 11, 13, and 14 is adequate for the purposes of this book
- The larger subset of UML presented in this chapter is adequate for the development and maintenance of most software products

Class Diagrams

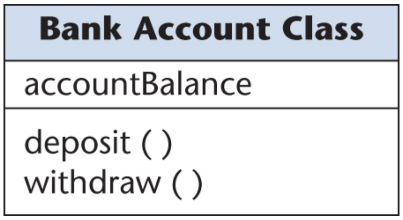
- A class diagram depicts classes and their interrelationships
- Here is the simplest possible class diagram



Bank Account Class

Class Diagrams

- Class diagram showing more details of **Bank Account Class**

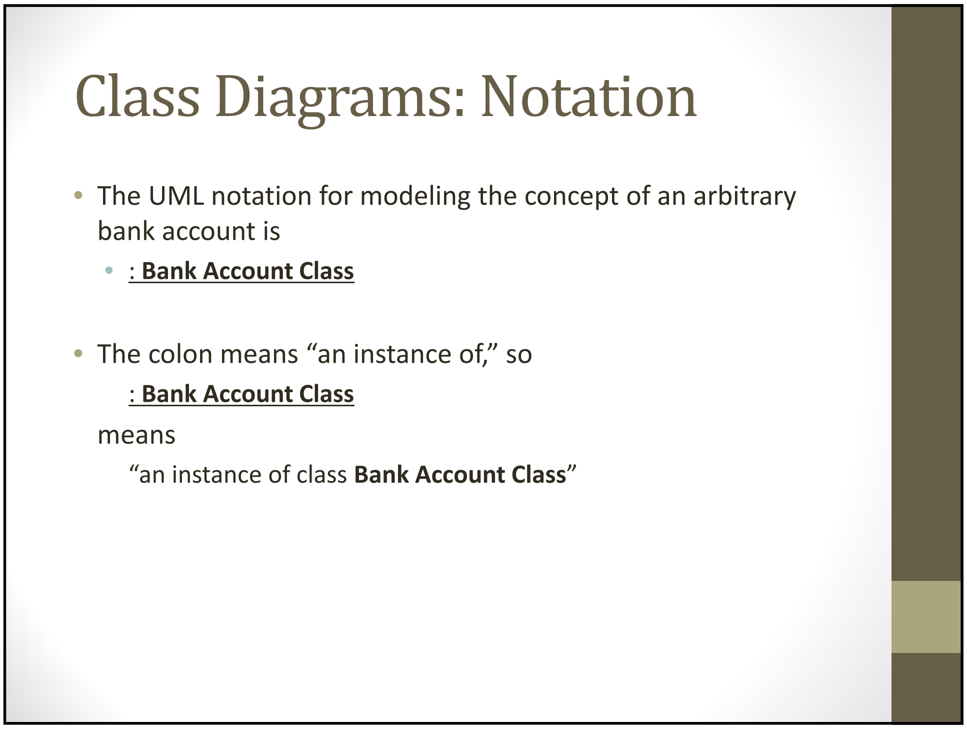


Bank Account Class

accountBalance

deposit ( )
withdraw ( )

- Add as many (or as few) details as appropriate for the current iteration and incrementation



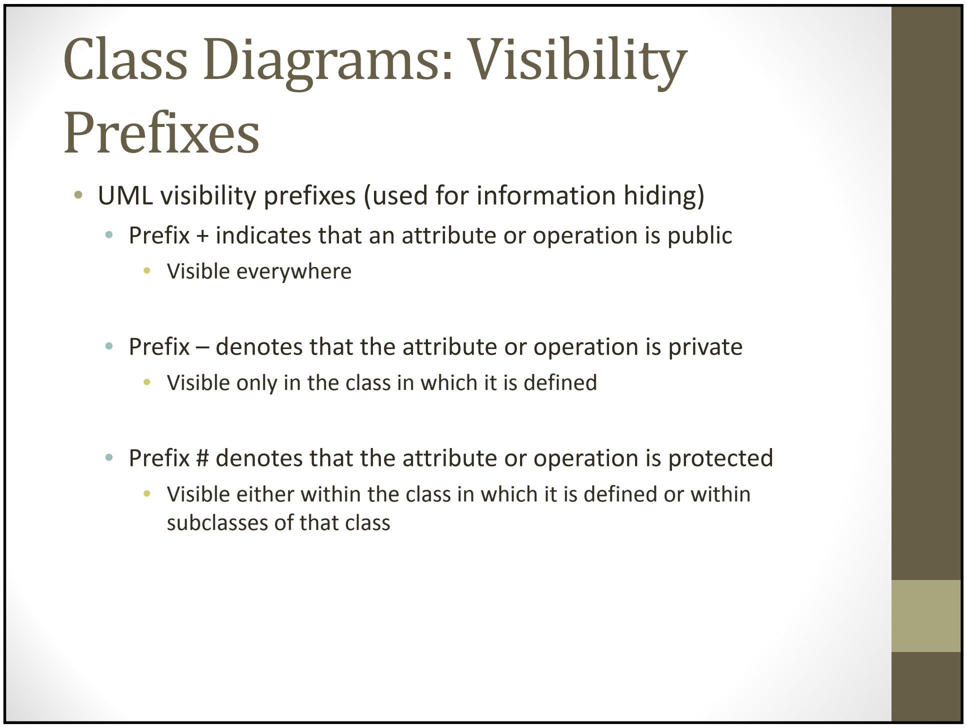
Class Diagrams: Notation

- The UML notation for modeling the concept of an arbitrary bank account is
  - **<u>: Bank Account Class</u>**
- The colon means “an instance of,” so

  <u>: Bank Account Class</u>

  means

  “an instance of class **Bank Account Class**”

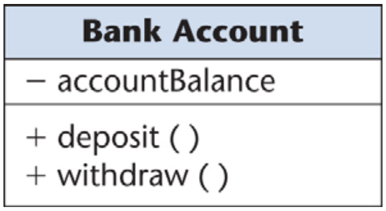


Class Diagrams: Visibility Prefixes

- UML visibility prefixes (used for information hiding)
  - Prefix + indicates that an attribute or operation is public
    - Visible everywhere
  - Prefix – denotes that the attribute or operation is private
    - Visible only in the class in which it is defined
  - Prefix # denotes that the attribute or operation is protected
    - Visible either within the class in which it is defined or within subclasses of that class

Class Diagrams: Visibility Prefixes

- Example:
  - Class diagram with visibility prefixes added



Bank Account

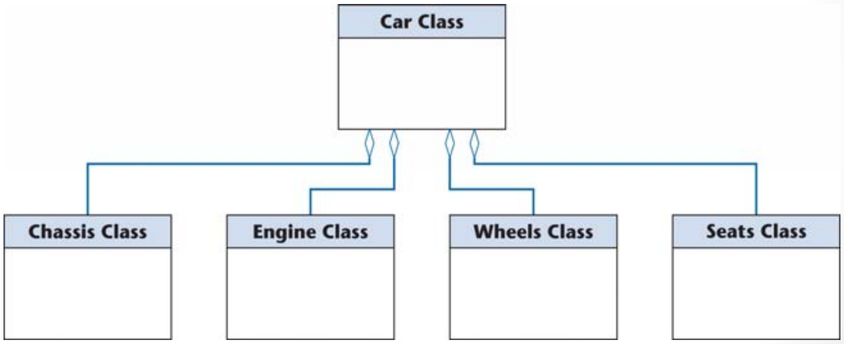
− accountBalance

+ deposit ( )
+ withdraw ( )

  - Attribute accountBalance is visible only within the **Bank Account Class**
  - Operations deposit and withdraw are accessible from anywhere within the software product

Aggregation

- Example: “A car consists of a chassis, an engine, wheels, and seats”



Car Class

Chassis Class

Engine Class

Wheels Class

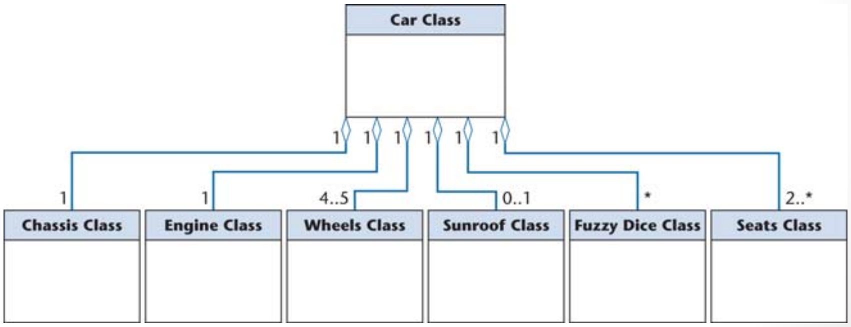
Seats Class

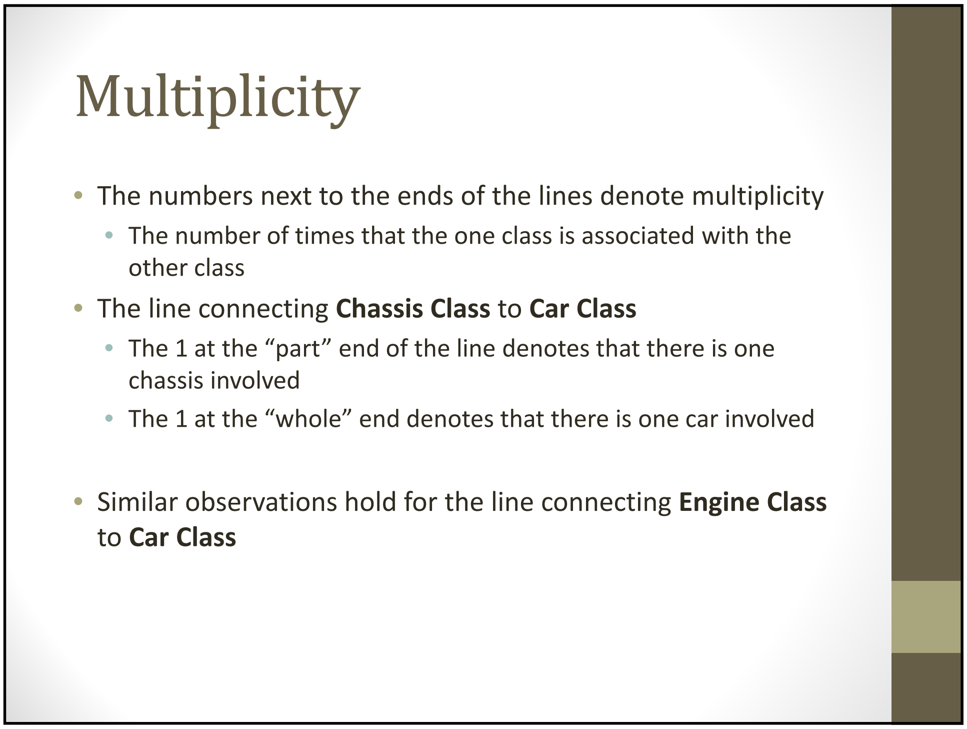
Aggregation

- The open diamonds denote aggregation
  - Aggregation is the UML term for the part–whole relationship
- The diamond is placed at the "whole" (car) end, not the "part" (chassis, engine, wheels, or seats) end of the line connecting a part to the whole

Multiplicity

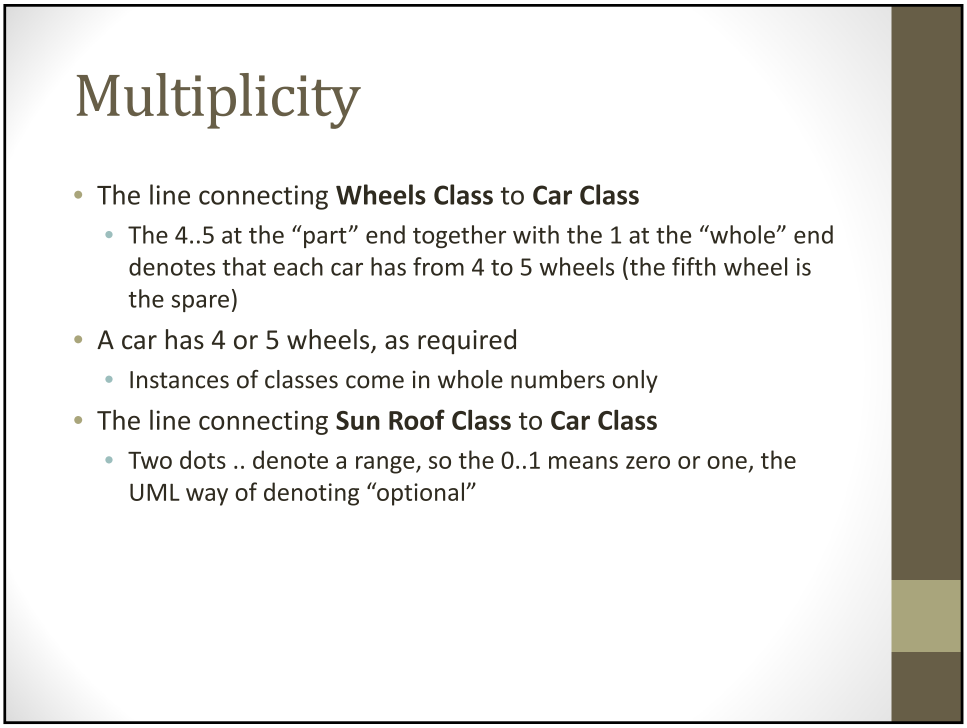
- Example: "A car consists of one chassis, one engine, 4 or 5 wheels, an optional sun roof, zero or more fuzzy dice hanging from the rear-view mirror, and 2 or more seats"





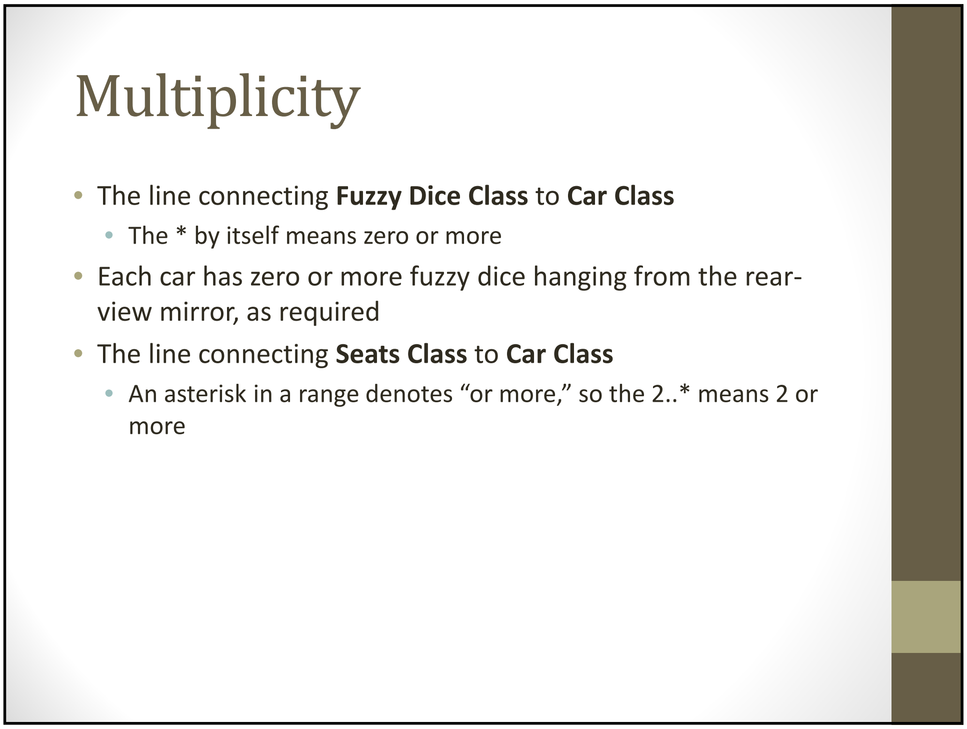
Multiplicity

- The numbers next to the ends of the lines denote multiplicity
  - The number of times that the one class is associated with the other class
- The line connecting **Chassis Class** to **Car Class**
  - The 1 at the "part" end of the line denotes that there is one chassis involved
  - The 1 at the "whole" end denotes that there is one car involved
- Similar observations hold for the line connecting **Engine Class** to **Car Class**



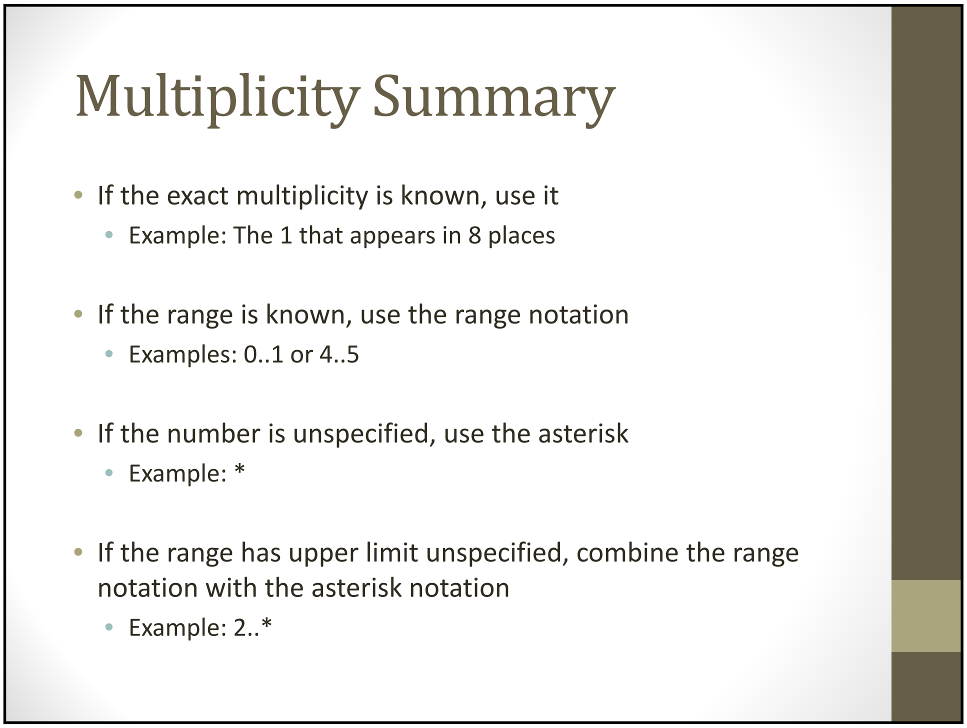
Multiplicity

- The line connecting **Wheels Class** to **Car Class**
  - The 4..5 at the "part" end together with the 1 at the "whole" end denotes that each car has from 4 to 5 wheels (the fifth wheel is the spare)
- A car has 4 or 5 wheels, as required
  - Instances of classes come in whole numbers only
- The line connecting **Sun Roof Class** to **Car Class**
  - Two dots .. denote a range, so the 0..1 means zero or one, the UML way of denoting "optional"



Multiplicity

- The line connecting **Fuzzy Dice Class** to **Car Class**
  - The * by itself means zero or more
- Each car has zero or more fuzzy dice hanging from the rear-view mirror, as required
- The line connecting **Seats Class** to **Car Class**
  - An asterisk in a range denotes “or more,” so the 2..* means 2 or more

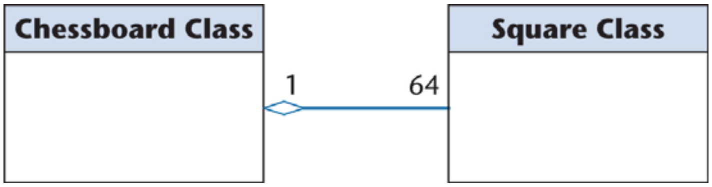


Multiplicity Summary

- If the exact multiplicity is known, use it
  - Example: The 1 that appears in 8 places
- If the range is known, use the range notation
  - Examples: 0..1 or 4..5
- If the number is unspecified, use the asterisk
  - Example: *
- If the range has upper limit unspecified, combine the range notation with the asterisk notation
  - Example: 2..*

Composition

- Aggregation example: Every chess board consists of 64 squares



- This relationship goes further
  - It is an instance of *composition*, a stronger form of aggregation

Composition

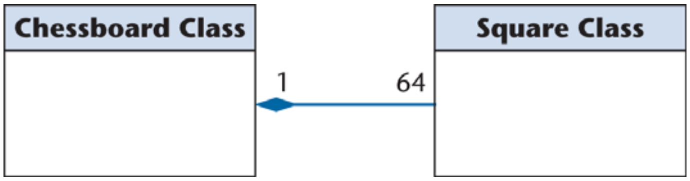
- Aggregation
  - Models the part–whole relationship

- Composition
  - Also models the part–whole relationship but, in addition,
  - Every part may belong to only one whole, and
  - If the whole is deleted, so are the parts

- Example: A number of different chess boards
  - Each square belongs to only one board
  - If a chess board is thrown away, all 64 squares on that board go as well

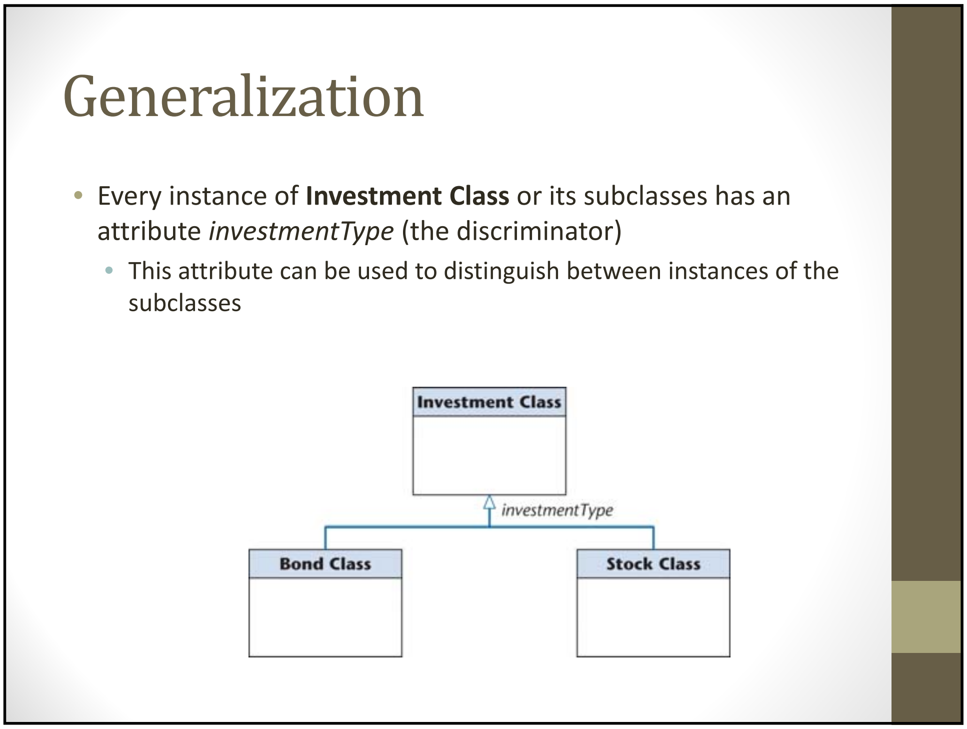
Composition

- Composition is depicted by a solid diamond



Generalization

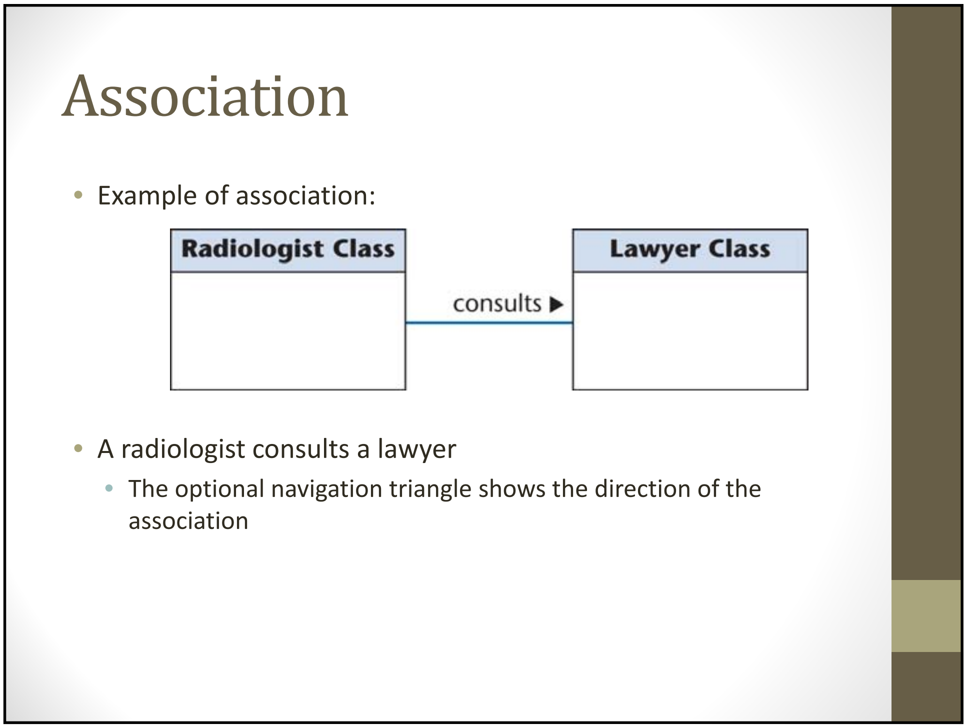
- Inheritance is a required feature of object orientation
- Inheritance is a special case of generalization
  - The UML notation for generalization is an open triangle
  - Sometimes the open triangle is labeled with a discriminator



Generalization

- Every instance of **Investment Class** or its subclasses has an attribute *investmentType* (the discriminator)
  - This attribute can be used to distinguish between instances of the subclasses

Investment Class
investmentType
Bond Class
Stock Class

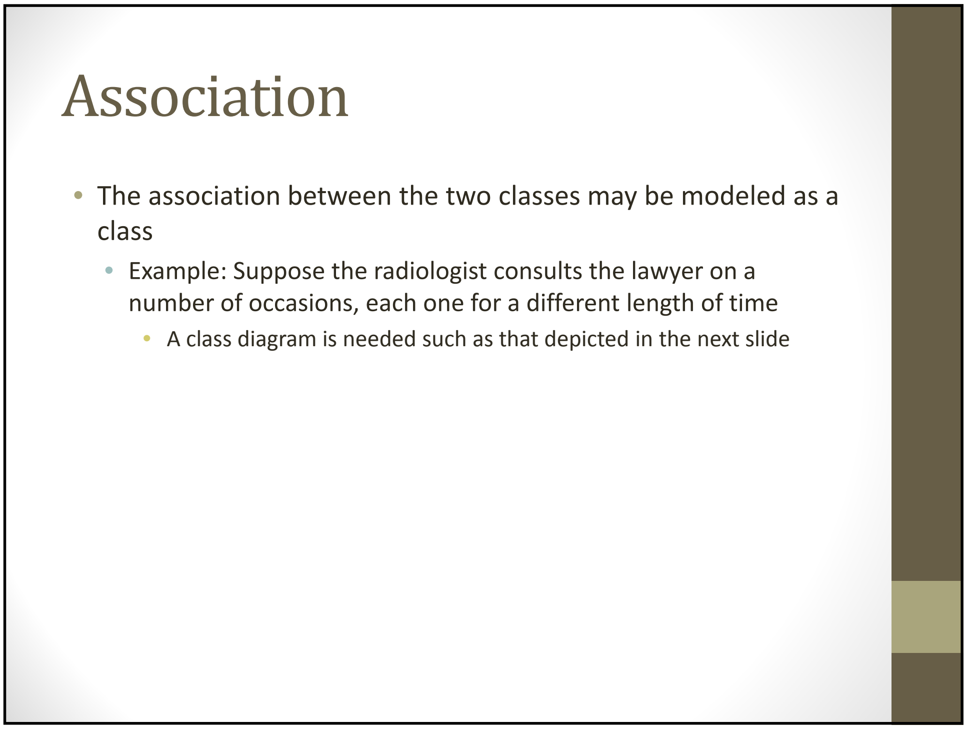


Association

- Example of association:

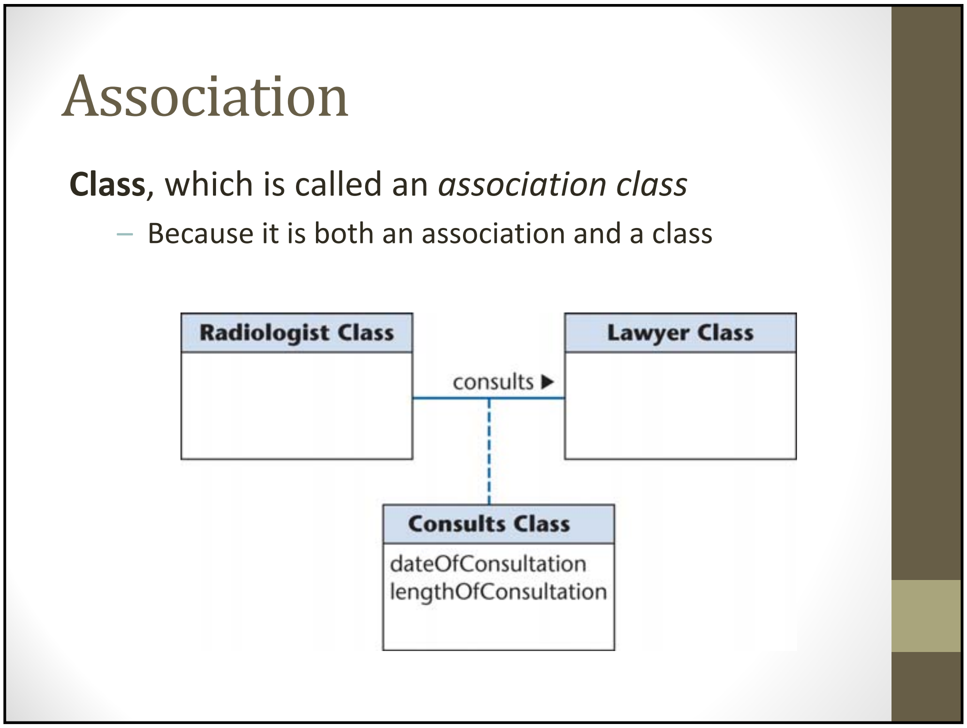
Radiologist Class
consults ▶
Lawyer Class

- A radiologist consults a lawyer
  - The optional navigation triangle shows the direction of the association



Association

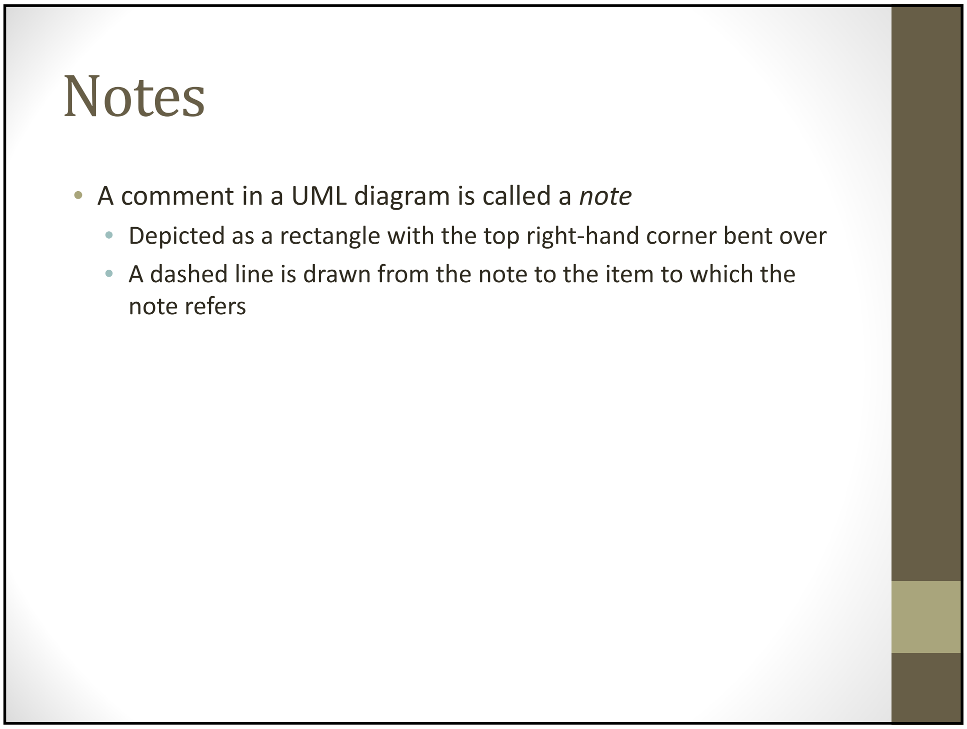
- The association between the two classes may be modeled as a class
  - Example: Suppose the radiologist consults the lawyer on a number of occasions, each one for a different length of time
    - A class diagram is needed such as that depicted in the next slide



Association

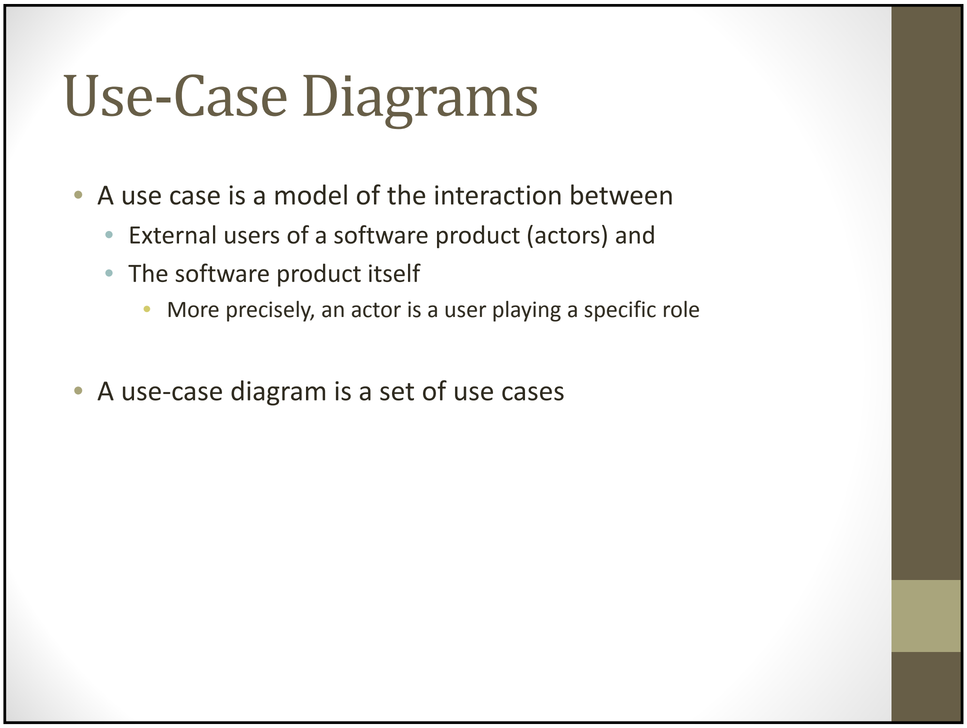
Class, which is called an *association class*

– Because it is both an association and a class



Notes

- A comment in a UML diagram is called a *note*
  - Depicted as a rectangle with the top right-hand corner bent over
  - A dashed line is drawn from the note to the item to which the note refers



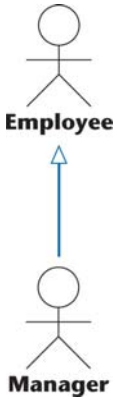
Use-Case Diagrams

- A use case is a model of the interaction between
  - External users of a software product (actors) and
  - The software product itself
    - More precisely, an actor is a user playing a specific role

- A use-case diagram is a set of use cases

Use-Case Diagrams

- Generalization of actors is supported
  - The open triangle points toward the more general case

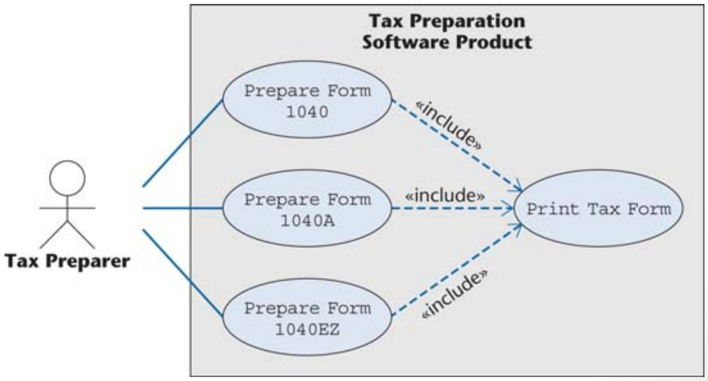


Stereotypes

- A stereotype in UML is a way of extending UML
- The names of stereotypes appear between guillemets
  - Example: «This is my own construct»
- Common Language Specific Examples
  - Static
  - Enum
  - Struct
- Can also be more generic
  - Entity
  - Process

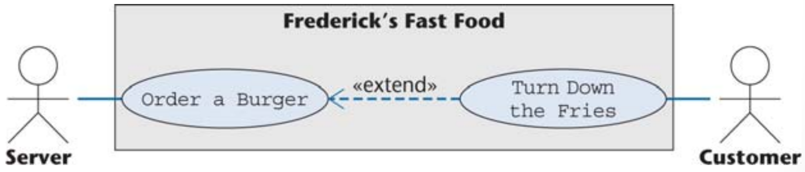
Stereotypes

- Example:
  - All three primary U.S. tax forms need to be printed
  - The other three use cases incorporate `Print Tax Form`



Stereotypes

- In the «extend» relationship, one use case is a variation of the standard use case
  - Example: A separate use case to model the situation of a diner ordering a burger but turning down the fries.



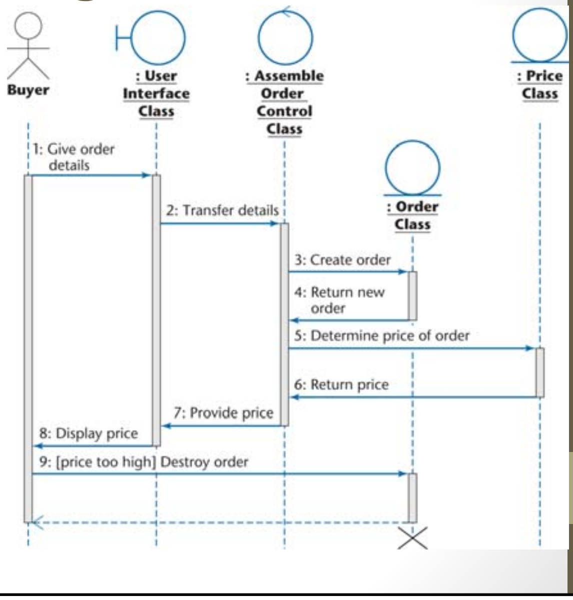
  - The open-headed arrow goes in the other direction

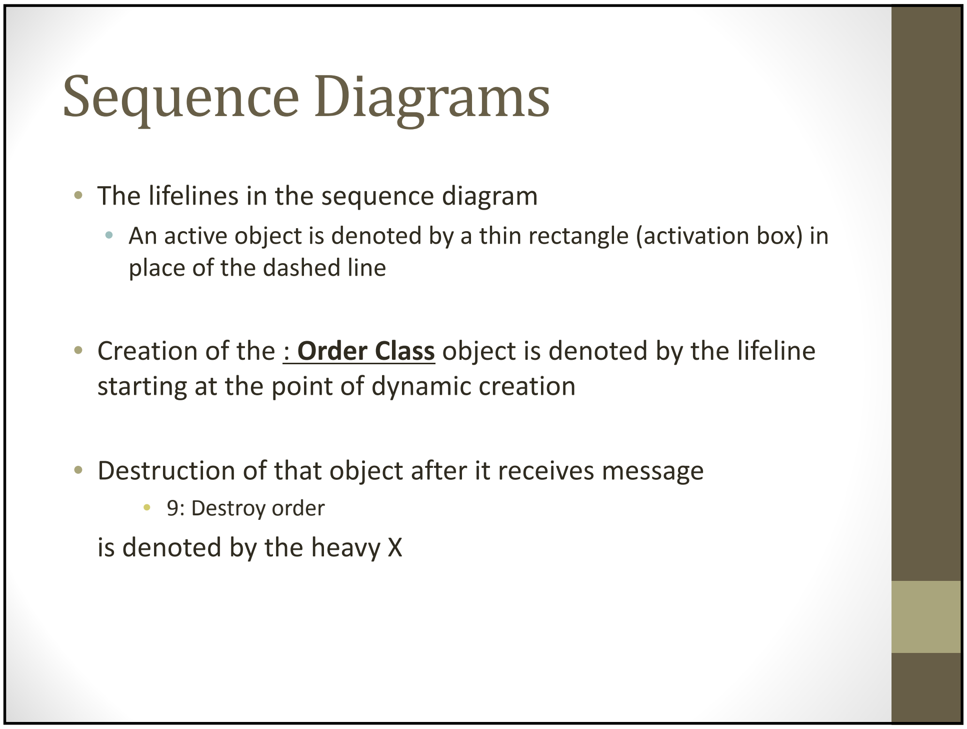
Interaction Diagrams

- Interaction diagrams show how objects interact with one another
- UML supports two types of interaction diagrams
  - Sequence diagrams
  - Collaboration diagrams

Sequence Diagrams

- Example:
  - Dynamic creation followed by destruction of an object

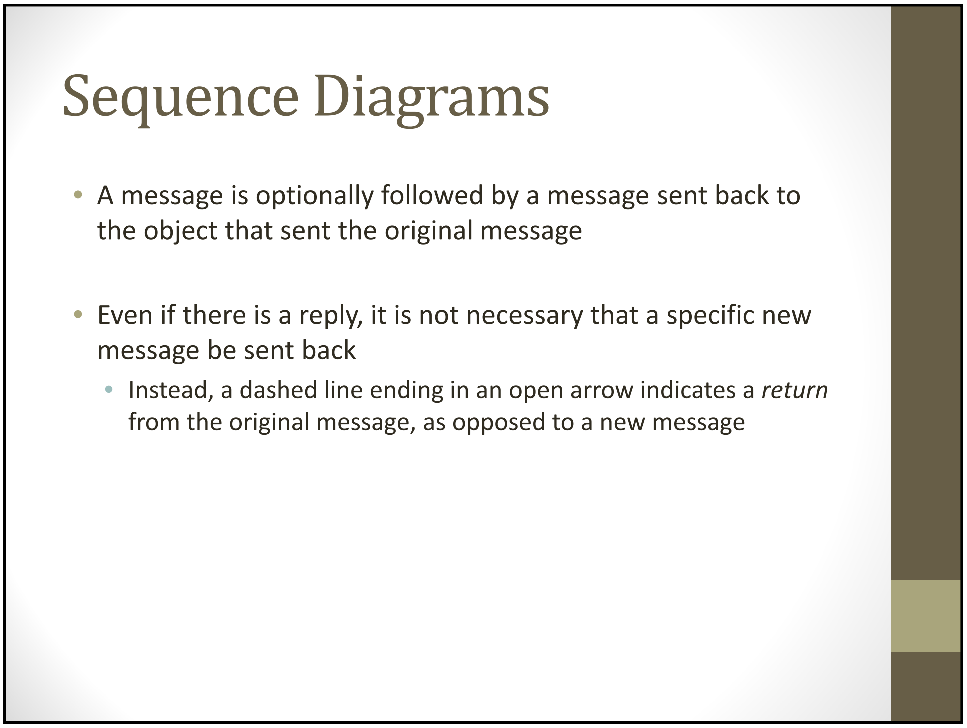




Sequence Diagrams

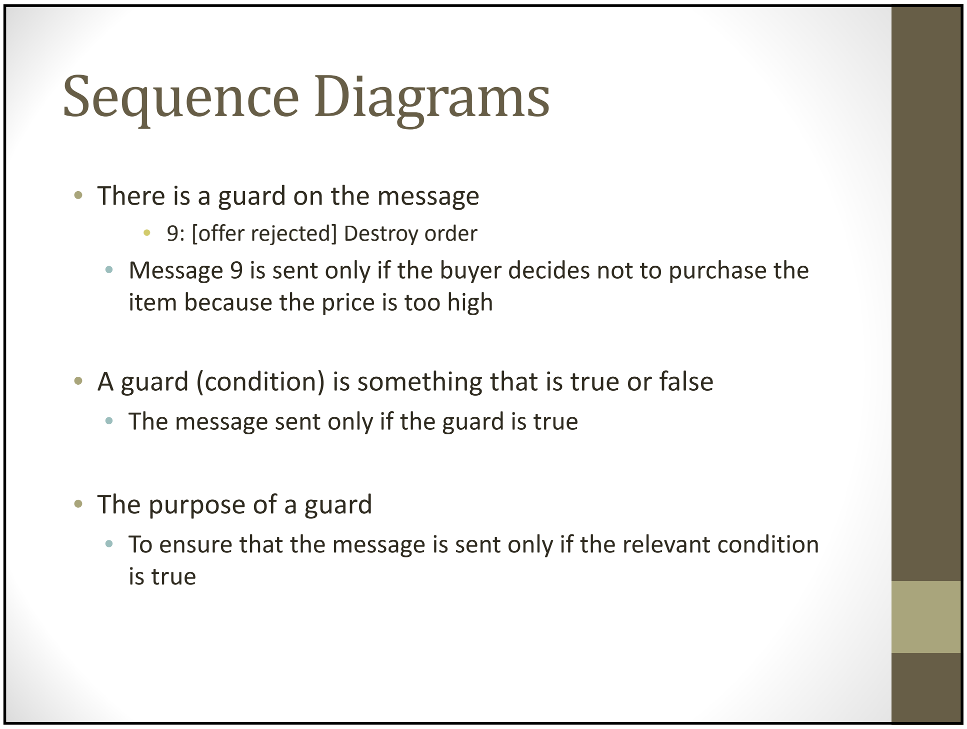
- The lifelines in the sequence diagram
  - An active object is denoted by a thin rectangle (activation box) in place of the dashed line
- Creation of the **: Order Class** object is denoted by the lifeline starting at the point of dynamic creation
- Destruction of that object after it receives message
    - 9: Destroy order

  is denoted by the heavy X



Sequence Diagrams

- A message is optionally followed by a message sent back to the object that sent the original message
- Even if there is a reply, it is not necessary that a specific new message be sent back
  - Instead, a dashed line ending in an open arrow indicates a *return* from the original message, as opposed to a new message

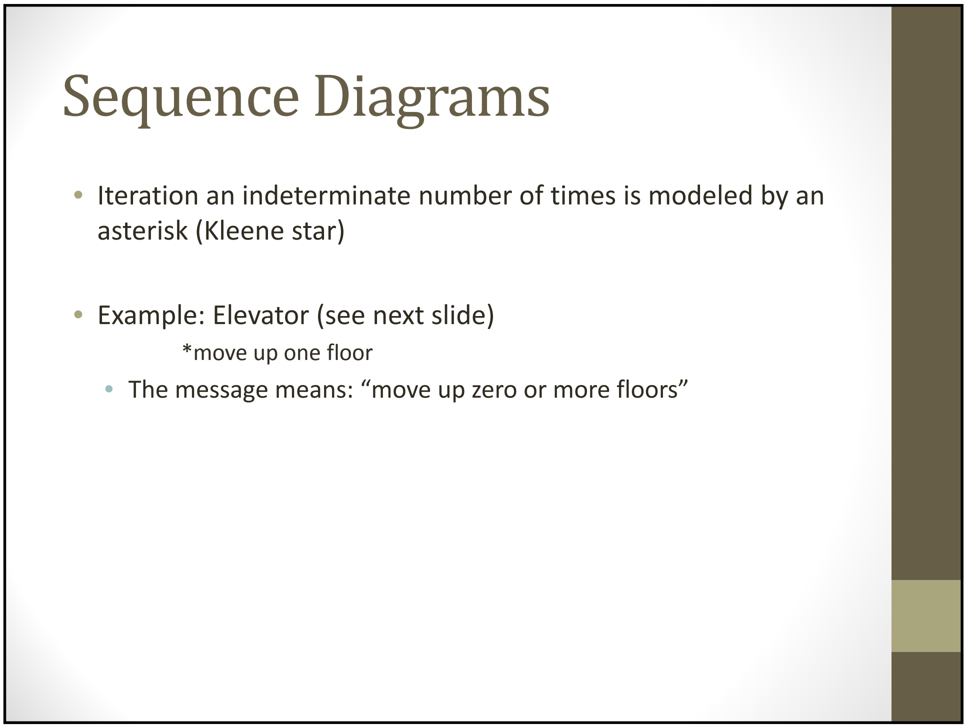


Sequence Diagrams

- There is a guard on the message
    - 9: [offer rejected] Destroy order
  - Message 9 is sent only if the buyer decides not to purchase the item because the price is too high

- A guard (condition) is something that is true or false
  - The message sent only if the guard is true

- The purpose of a guard
  - To ensure that the message is sent only if the relevant condition is true



Sequence Diagrams

- Iteration an indeterminate number of times is modeled by an asterisk (Kleene star)

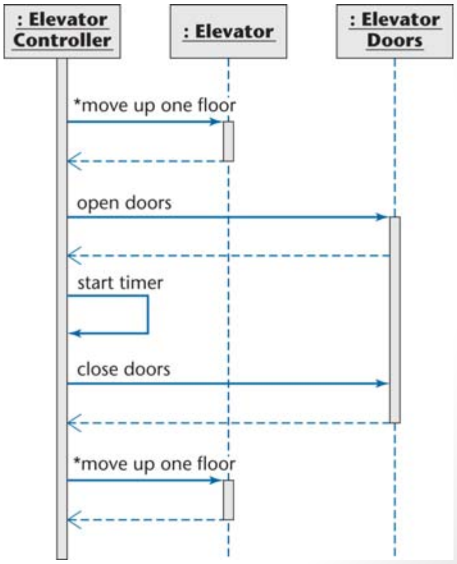
- Example: Elevator (see next slide)

  *move up one floor

  - The message means: “move up zero or more floors”

Sequence Diagrams

- Sequence diagram for elevator



Sequence Diagrams

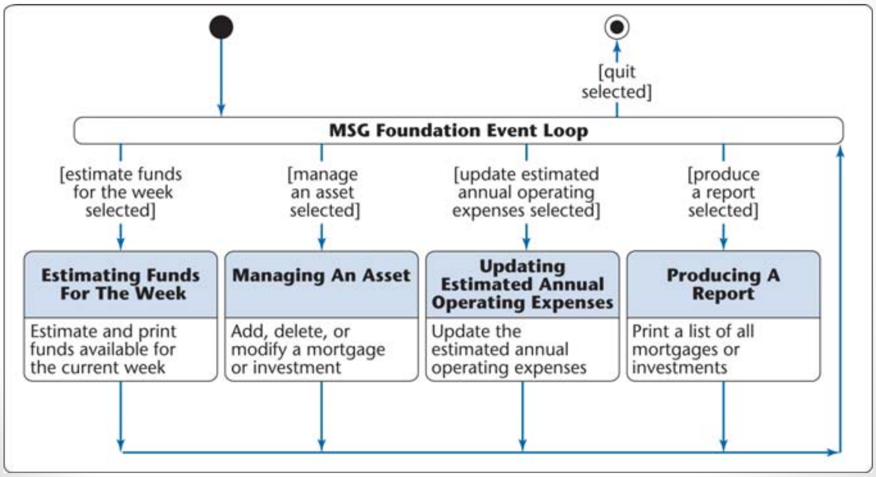
- An object can send a message to itself
  - A *self-call*
- Example:
  - The elevator has arrived at a floor
  - The elevator doors now open and a timer starts
  - At the end of the timer period the doors close again
  - The elevator controller sends a message to itself to start its timer — this self-call is shown in the previous UML diagram

Collaboration Diagrams

- Collaboration diagrams are equivalent to sequence diagrams
  - All the features of sequence diagrams are equally applicable to collaboration diagrams
- Use a sequence diagram when the transfer of information is the focus of attention
- Use a collaboration diagram when concentrating on the classes

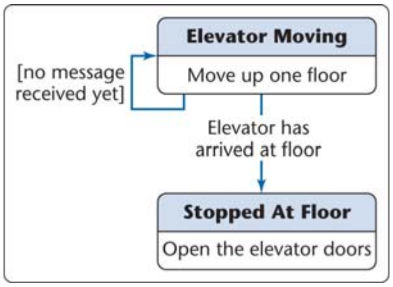
Statecharts

- Statechart with guards


[quit selected]
MSG Foundation Event Loop
[estimate funds for the week selected]
[manage an asset selected]
[update estimated annual operating expenses selected]
[produce a report selected]
Estimating Funds For The Week
Estimate and print funds available for the current week
Managing An Asset
Add, delete, or modify a mortgage or investment
Updating Estimated Annual Operating Expenses
Update the estimated annual operating expenses
Producing A Report
Print a list of all mortgages or investments

Statecharts

- An event also causes transitions between states
- Example: The receipt of a message



Statecharts

- The most general form of a transition label is
  - event [guard] / action
- If
  - event

  has taken place and
  - [guard]

  is true, the transition occurs, and, while it is occurring,
  - action

  is performed

Statecharts

- Equivalent statement with the most general transition

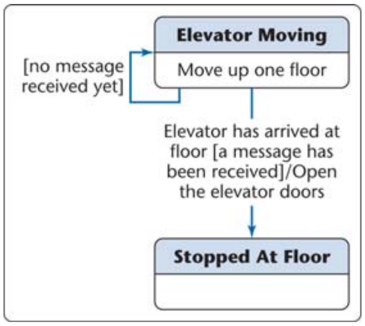

Elevator Moving
Move up one floor
[no message received yet]
Elevator has arrived at floor [a message has been received]/Open the elevator doors
Stopped At Floor

Figure 17.18

Statecharts

- There are two places where an action can be performed in a statechart
  - When a state is entered
    - Activity
  - As part of a transition
    - Action
- Technical difference:
  - An activity can take several seconds
  - An action takes places essentially instantaneously

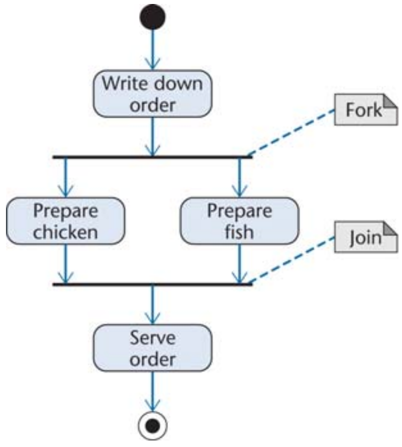
Activity Diagrams

- Activity diagrams show how various events are coordinated
  - Used when activities are carried on in parallel

- Example:
  - One diner orders chicken, the other fish
  - The waiter writes down their order, and hands it to the chef
  - The meal is served only when both dishes have been prepared

Activity Diagrams

- Example:

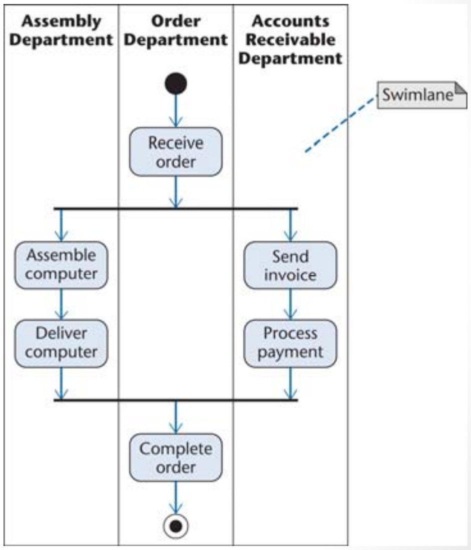


Activity Diagrams

- A *fork* has
  - One incoming transition, and
  - Many outgoing transitions, each of which starts an activity to be executed in parallel with the other activities
- A *join* has
  - Many incoming transitions, each of which lead from an activity executed in parallel with the other activities, and
  - One outgoing transition that is started when all the parallel activities have been completed

Activity Diagrams

- Example:
  - A company that assembles computers as specified by the customer



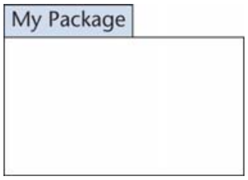
Activity Diagrams

- The three departments involved
  - Assembly Department
  - Order Department
  - Accounts Receivable Department

  are each in their own swimlane

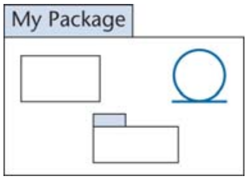
Packages

- A large information system is decomposed into relatively independent packages
  - UML notation for a package
  - Namespaces



Packages

- Example showing the contents of My Package



Component Diagrams

- A component diagram shows dependencies among software components, including
  - Source code (represented by a note)
  - Compiled code
  - Executable load images
- Helpful for showing what components will be in what deployable file (dll, exe, etc)

Component Diagrams

- Example:

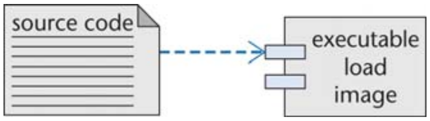


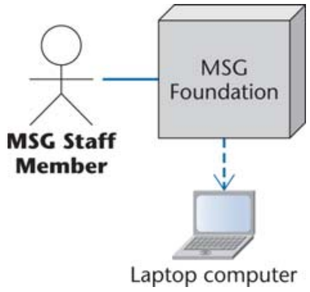
Figure 17.25

Deployment Diagrams

- A deployment diagram shows on which hardware component each software component is installed (or deployed)
- It also shows the communication links between the hardware components

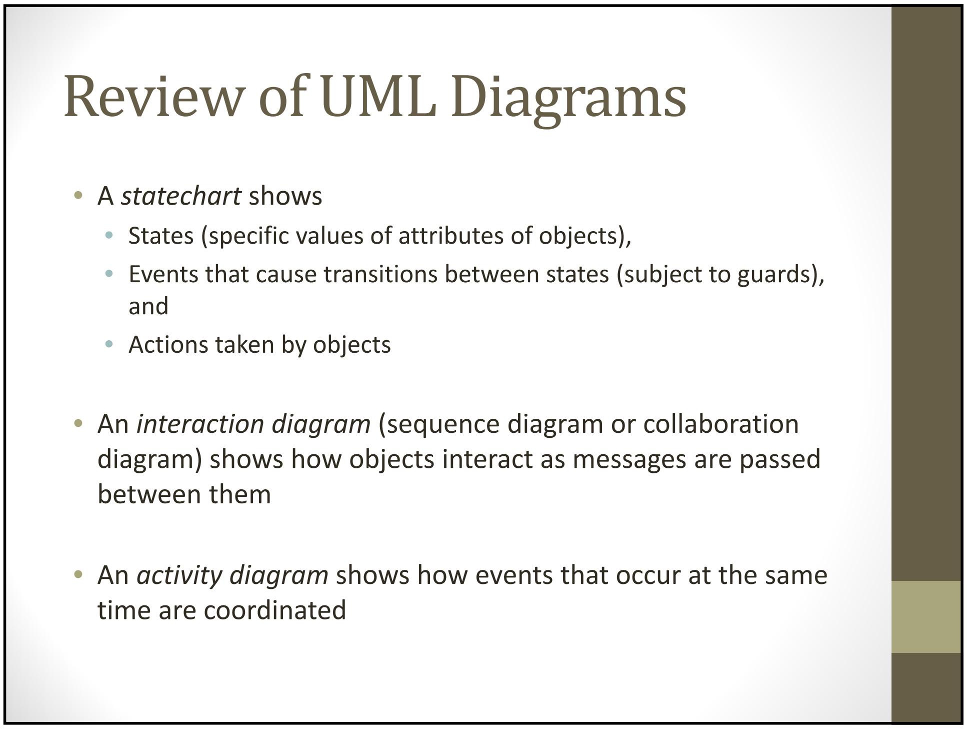
Deployment Diagrams

- Example:



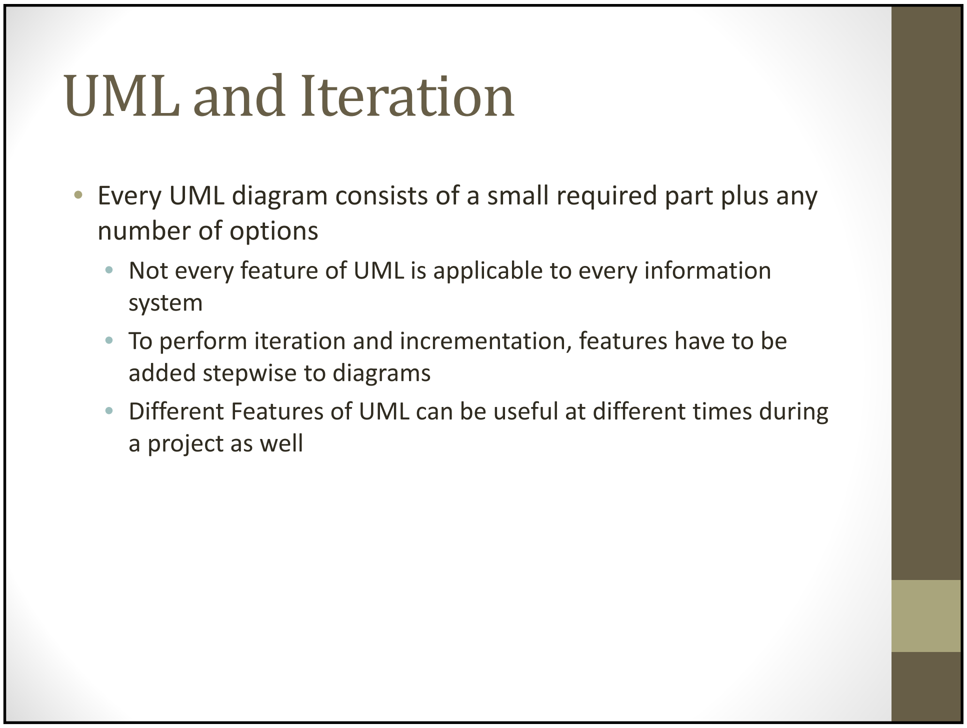
Review of UML Diagrams

- Some diagrams that could be confused include:
  - A *use case* models the interaction between actors and the information system
  - A *use-case diagram* is a single diagram that incorporates a number of use cases
  - A *class diagram* is a model of the classes showing the static relationships between them
    - Including association and generalization



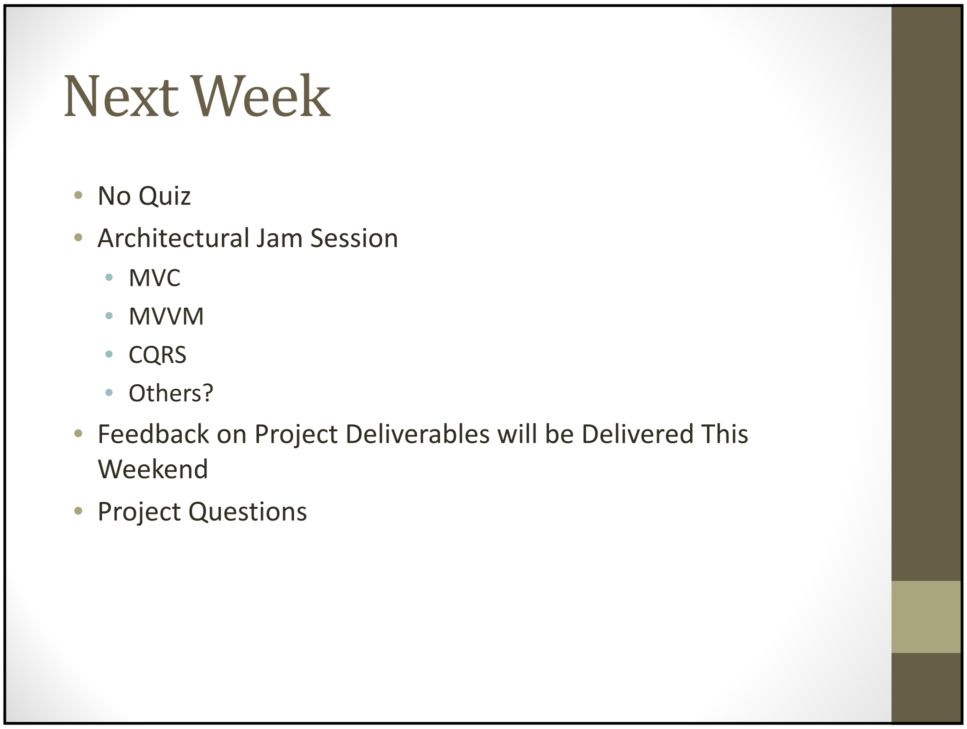
Review of UML Diagrams

- A *statechart* shows
  - States (specific values of attributes of objects),
  - Events that cause transitions between states (subject to guards), and
  - Actions taken by objects
- An *interaction diagram* (sequence diagram or collaboration diagram) shows how objects interact as messages are passed between them
- An *activity diagram* shows how events that occur at the same time are coordinated



UML and Iteration

- Every UML diagram consists of a small required part plus any number of options
  - Not every feature of UML is applicable to every information system
  - To perform iteration and incrementation, features have to be added stepwise to diagrams
  - Different Features of UML can be useful at different times during a project as well



Next Week

- No Quiz
- Architectural Jam Session
  - MVC
  - MVVM
  - CQRS
  - Others?
- Feedback on Project Deliverables will be Delivered This Weekend
- Project Questions