INF 111 / CSE 121: Software Tools and Methods

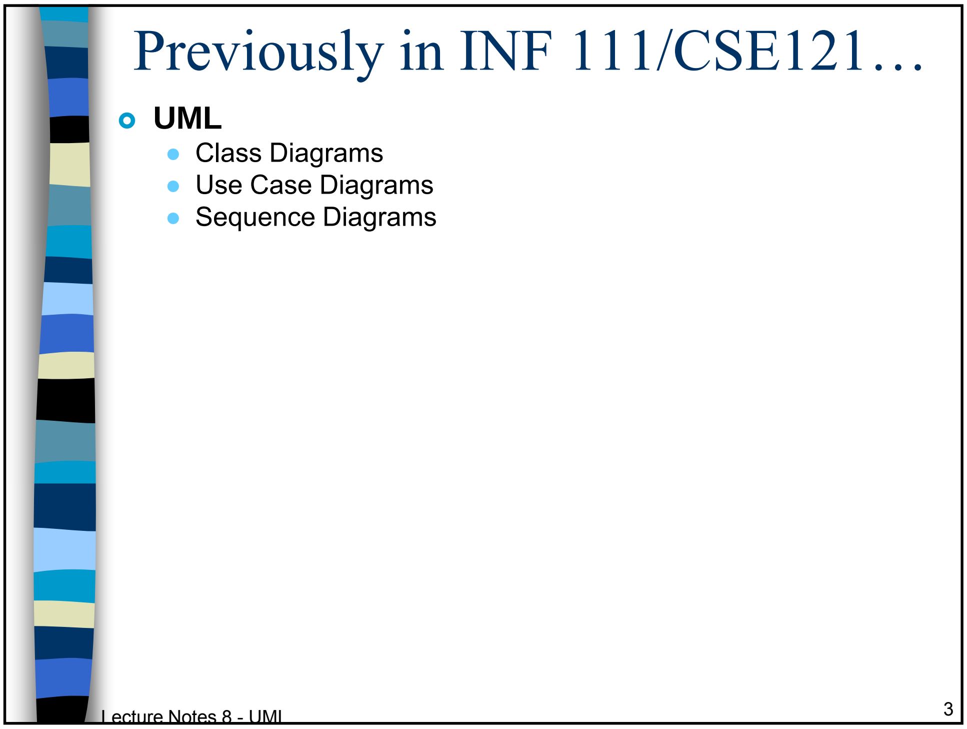
Lecture Notes for Summer Quarter, 2008

Michele Rousseau

Set 8 – UML – Part 2

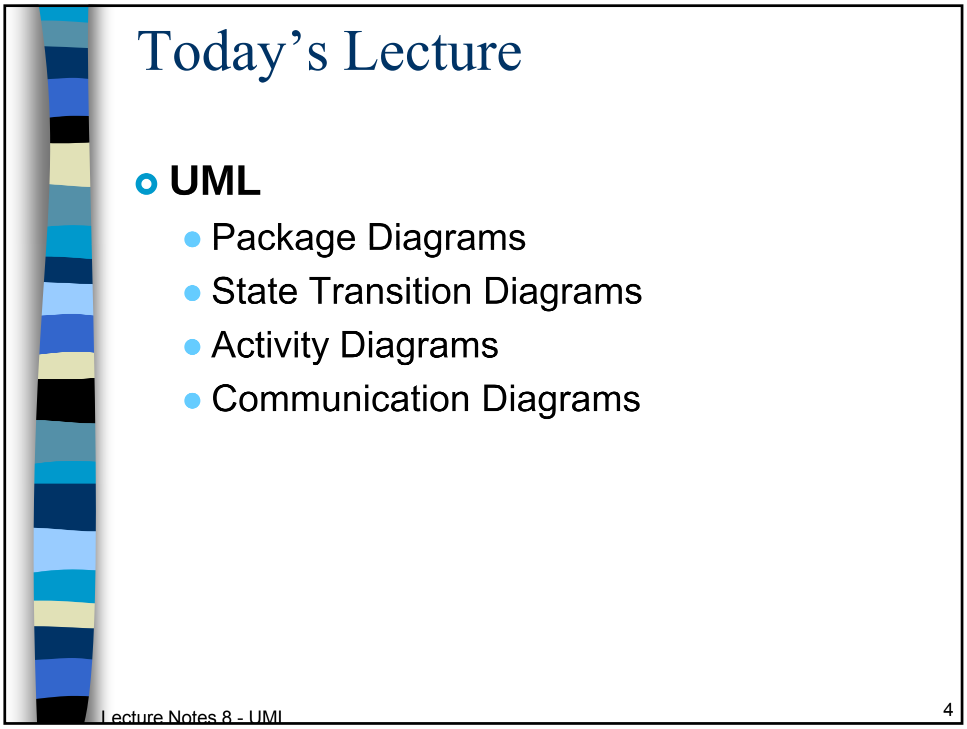
Announcements

- **UML Links:**
- http://dn.codegear.com/article/31863#use-case-diagram



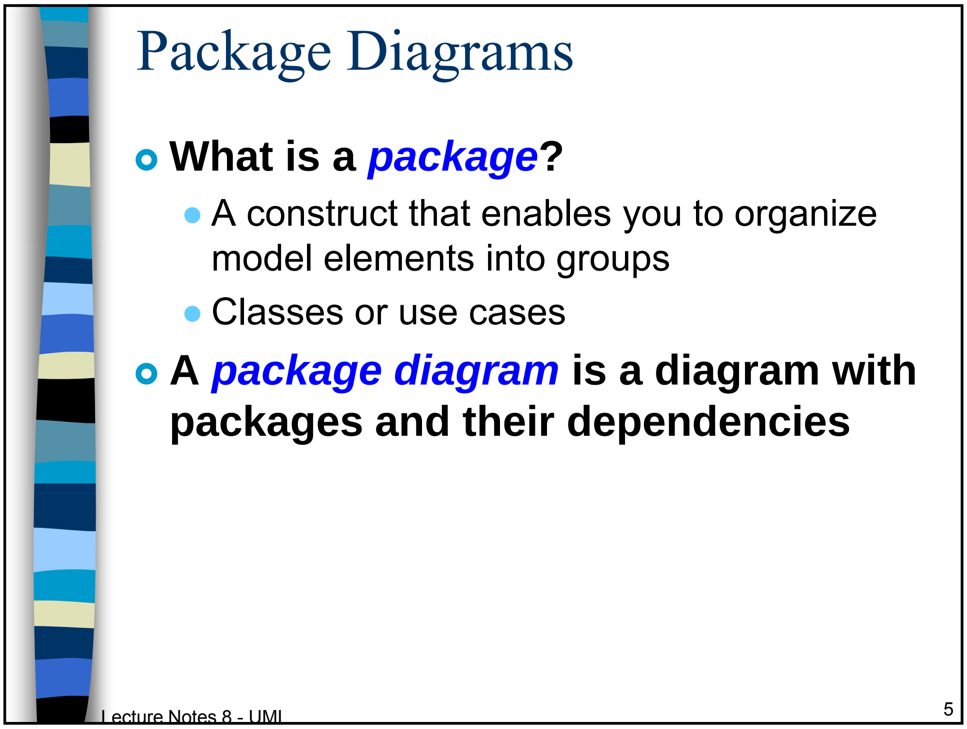
Previously in INF 111/CSE121…

- UML
  - Class Diagrams
  - Use Case Diagrams
  - Sequence Diagrams



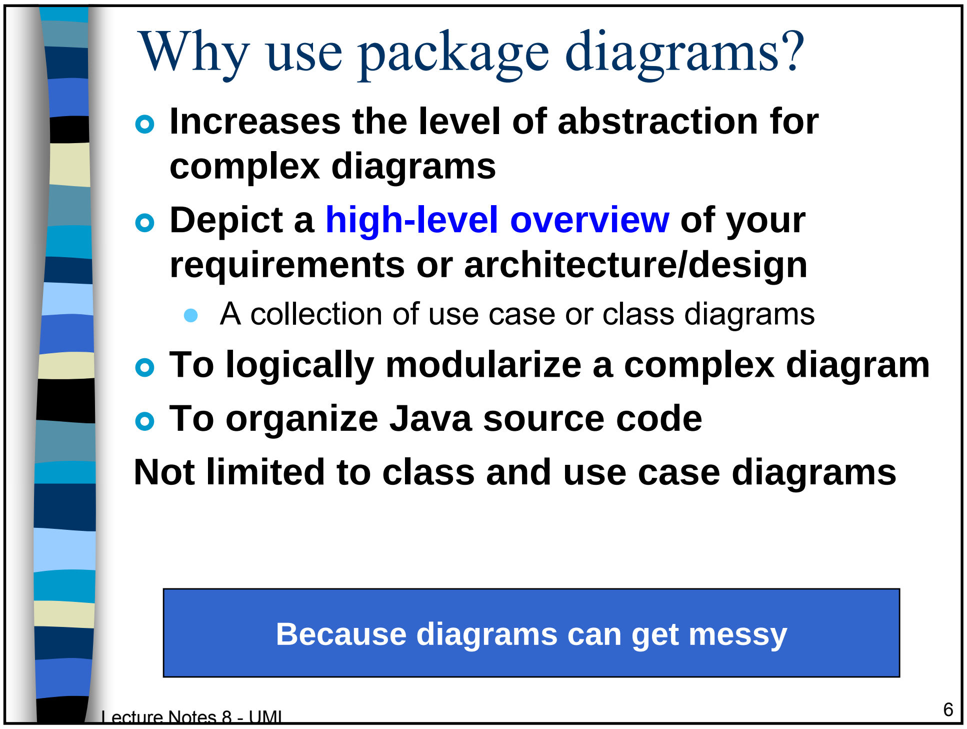
Today’s Lecture

- UML
  - Package Diagrams
  - State Transition Diagrams
  - Activity Diagrams
  - Communication Diagrams



Package Diagrams

- **What is a *package*?**
  - A construct that enables you to organize model elements into groups
  - Classes or use cases
- **A *package diagram* is a diagram with packages and their dependencies**

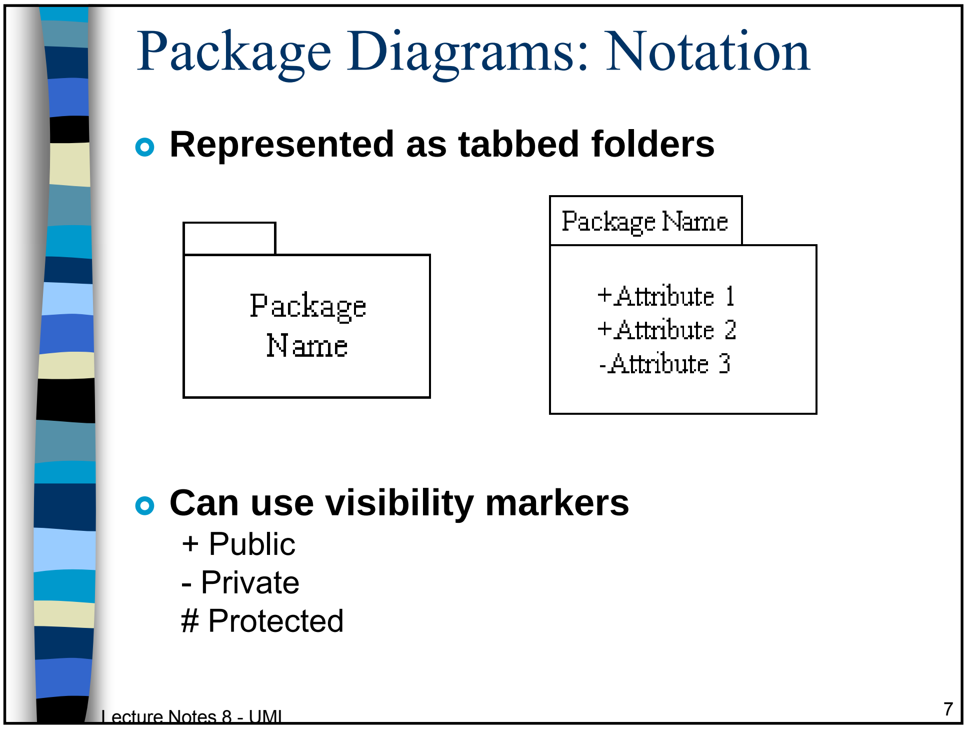


Why use package diagrams?

- **Increases the level of abstraction for complex diagrams**
- **Depict a high-level overview of your requirements or architecture/design**
  - A collection of use case or class diagrams
- **To logically modularize a complex diagram**
- **To organize Java source code**

Not limited to class and use case diagrams

Because diagrams can get messy



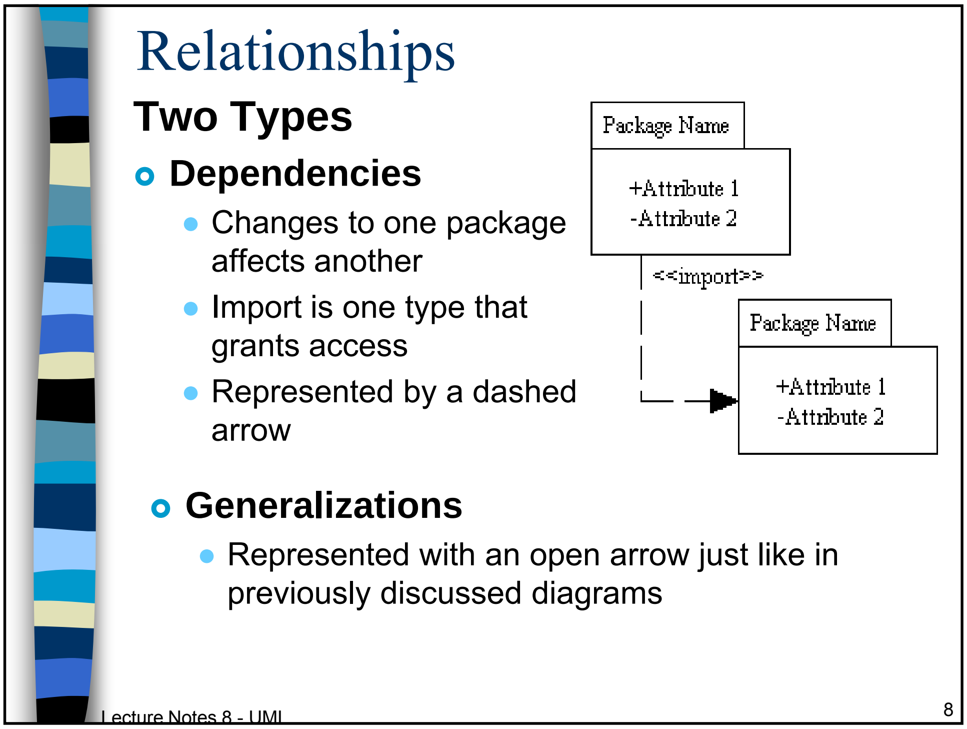
Package Diagrams: Notation

- **Represented as tabbed folders**

Package
Name
Package Name
+Attribute 1
+Attribute 2
-Attribute 3

- **Can use visibility markers**

  + Public
  - Private
  # Protected



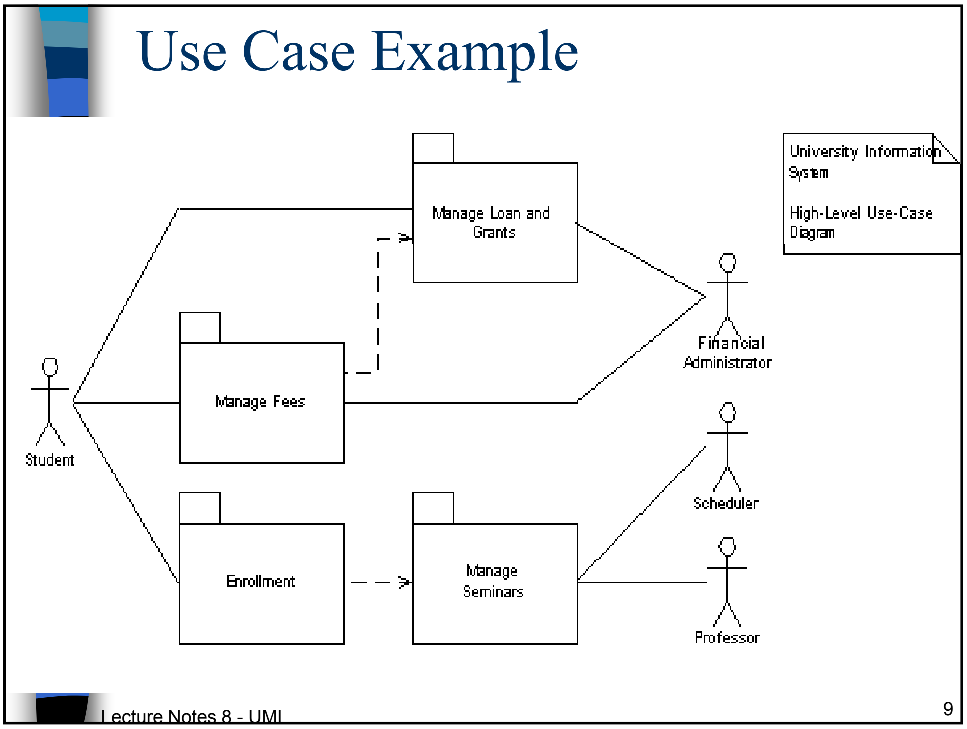
Relationships

Two Types

- **Dependencies**
  - Changes to one package affects another
  - Import is one type that grants access
  - Represented by a dashed arrow

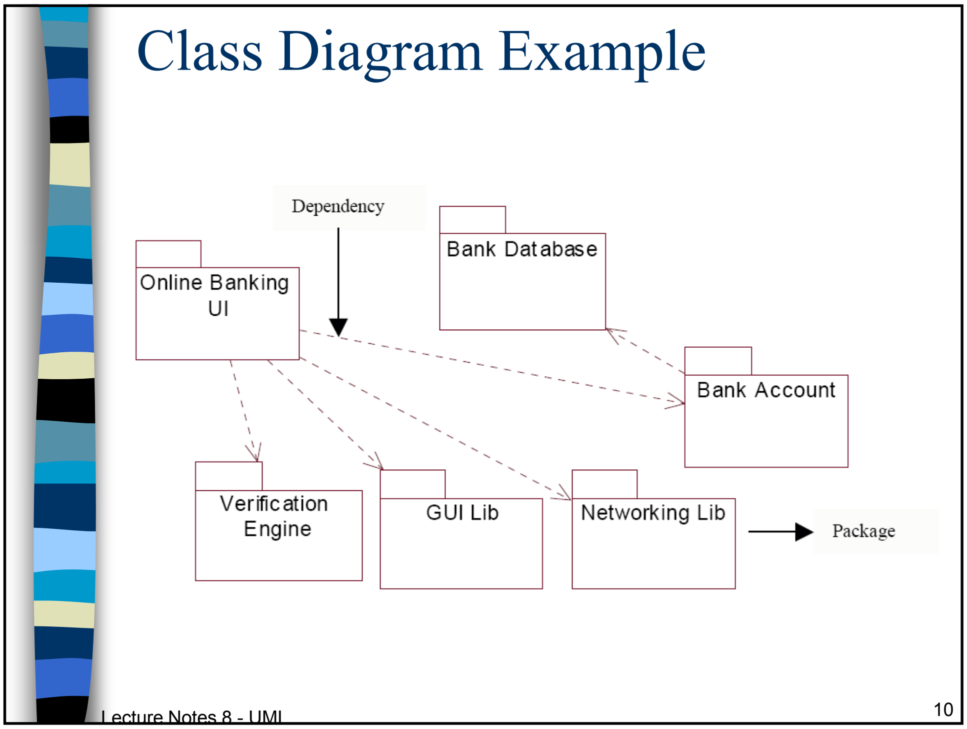
Package Name
+Attribute 1
-Attribute 2
<<import>>
Package Name
+Attribute 1
-Attribute 2

- **Generalizations**
  - Represented with an open arrow just like in previously discussed diagrams



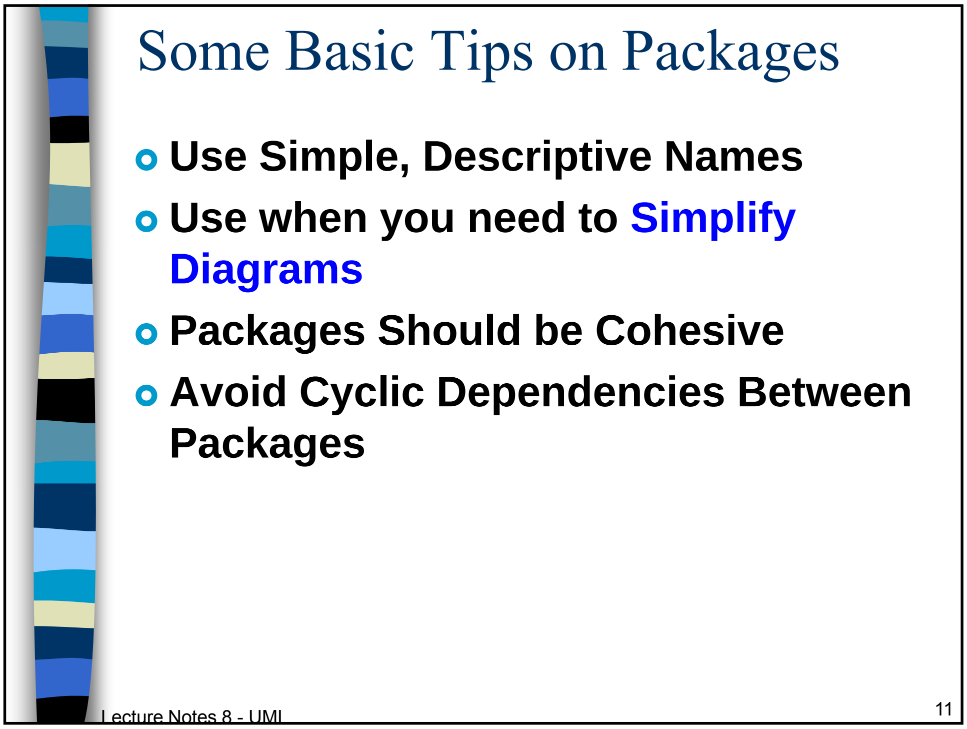
Use Case Example

Lecture Notes 8 - UML 9



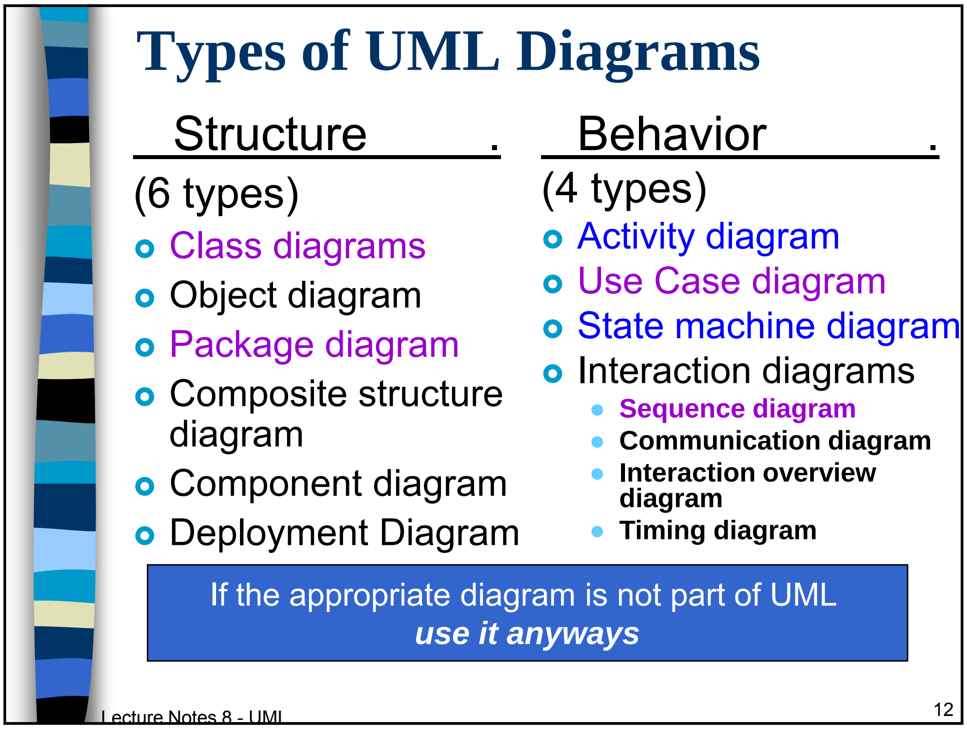
Class Diagram Example

Lecture Notes 8 - UML 10



Some Basic Tips on Packages

- **Use Simple, Descriptive Names**
- **Use when you need to Simplify Diagrams**
- **Packages Should be Cohesive**
- **Avoid Cyclic Dependencies Between Packages**



Types of UML Diagrams

Structure .

(6 types)

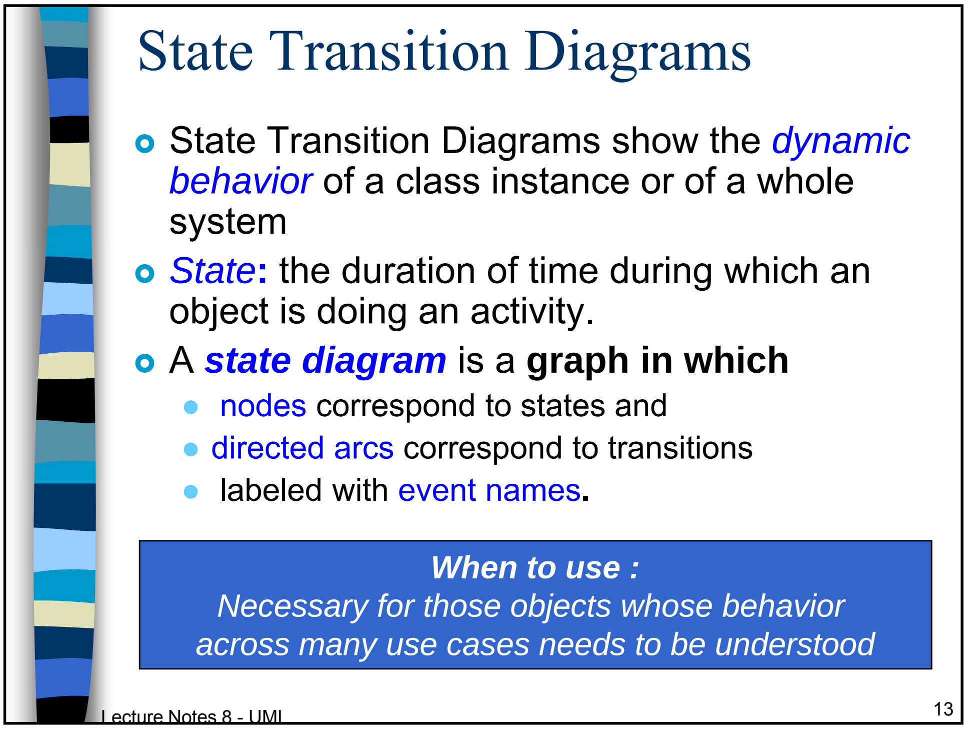
- Class diagrams
- Object diagram
- Package diagram
- Composite structure diagram
- Component diagram
- Deployment Diagram

Behavior .

(4 types)

- Activity diagram
- Use Case diagram
- State machine diagram
- Interaction diagrams
  - **Sequence diagram**
  - **Communication diagram**
  - **Interaction overview diagram**
  - **Timing diagram**

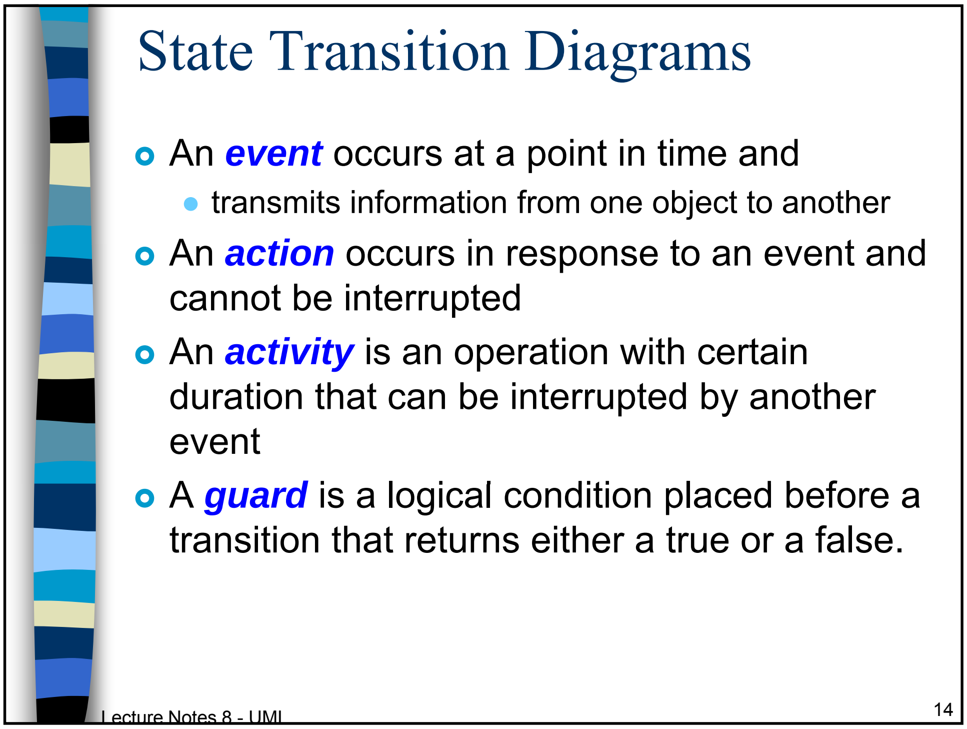
If the appropriate diagram is not part of UML
use it anyways



State Transition Diagrams

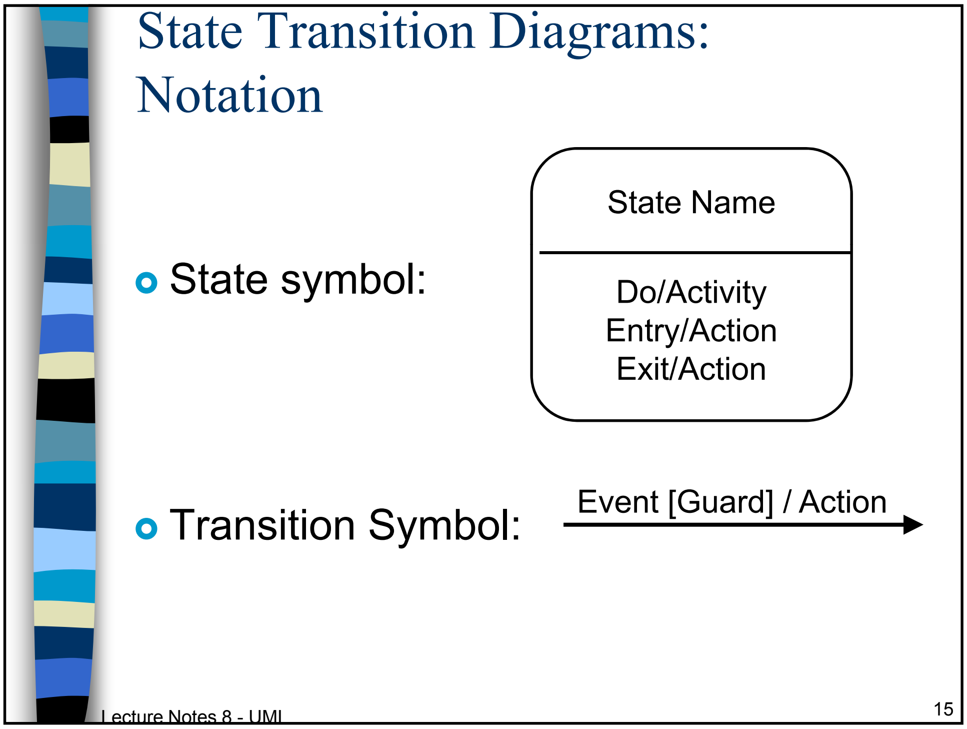
- State Transition Diagrams show the *dynamic behavior* of a class instance or of a whole system
- *State*: the duration of time during which an object is doing an activity.
- A ***state diagram*** is a **graph in which**
  - nodes correspond to states and
  - directed arcs correspond to transitions
  - labeled with event names**.**

When to use :
Necessary for those objects whose behavior across many use cases needs to be understood



State Transition Diagrams

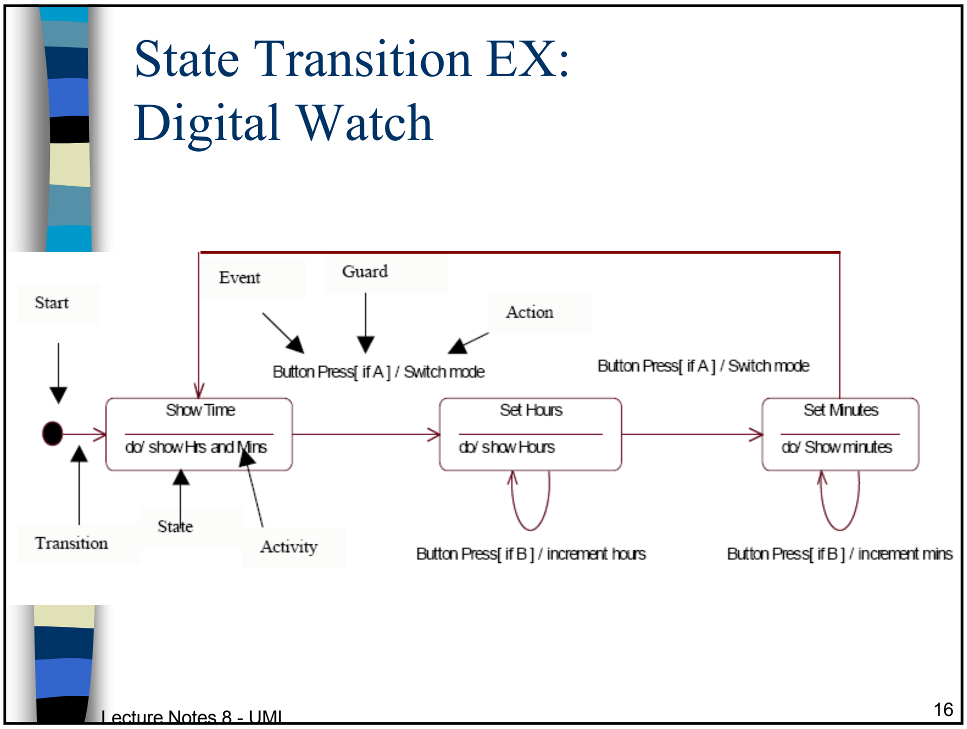
- An ***event*** occurs at a point in time and
  - transmits information from one object to another
- An ***action*** occurs in response to an event and cannot be interrupted
- An ***activity*** is an operation with certain duration that can be interrupted by another event
- A ***guard*** is a logical condition placed before a transition that returns either a true or a false.



State Transition Diagrams: Notation

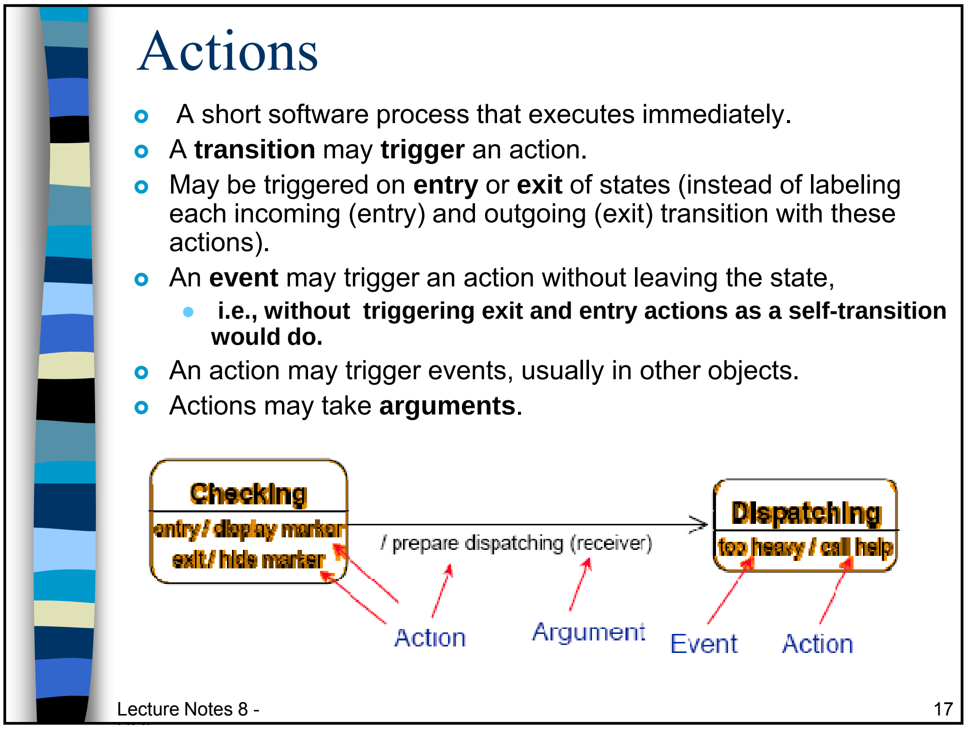
- State symbol:

- Transition Symbol:



State Transition EX: Digital Watch

Start
Event
Guard
Action
Button Press[if A] / Switch mode
Button Press[if A] / Switch mode
Show Time
do/ show Hrs and Mins
Set Hours
do/ show Hours
Set Minutes
do/ Show minutes
Transition
State
Activity
Button Press[if B] / increment hours
Button Press[if B] / increment mins



Actions

- A short software process that executes immediately.
- A **transition** may **trigger** an action.
- May be triggered on **entry** or **exit** of states (instead of labeling each incoming (entry) and outgoing (exit) transition with these actions).
- An **event** may trigger an action without leaving the state,
  - **i.e., without triggering exit and entry actions as a self-transition would do.**
- An action may trigger events, usually in other objects.
- Actions may take **arguments**.

Checking

entry / display marker

exit / hide marker

/ prepare dispatching (receiver)

Dispatching

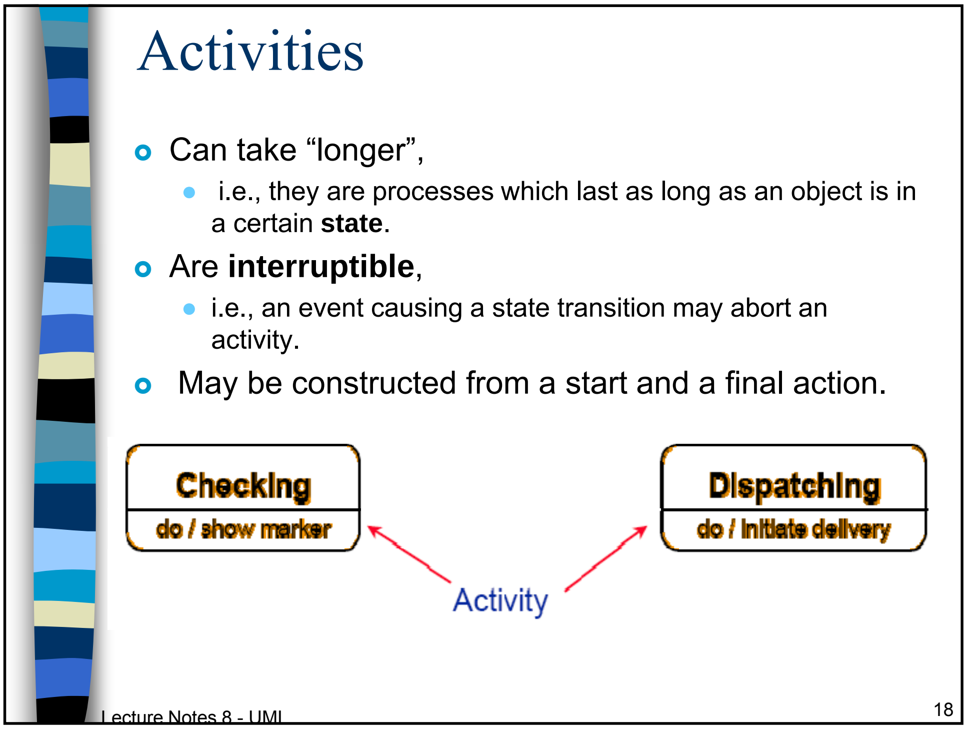
too heavy / call help

Action

Argument

Event

Action



Activities

- Can take “longer”,
  - i.e., they are processes which last as long as an object is in a certain **state**.
- Are **interruptible**,
  - i.e., an event causing a state transition may abort an activity.
- May be constructed from a start and a final action.

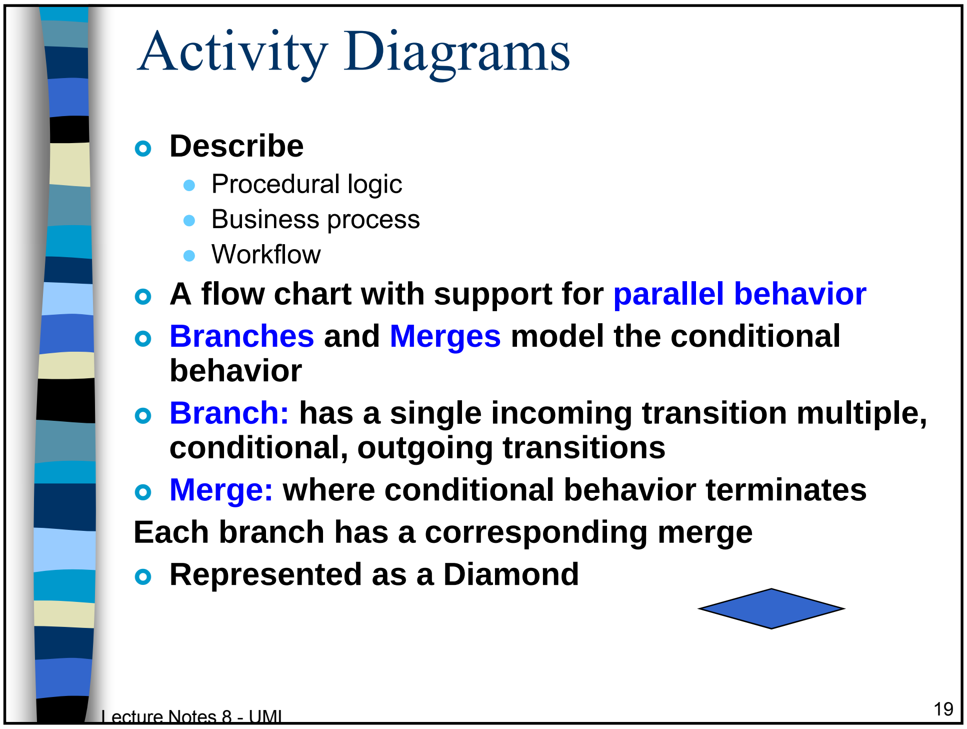
Checking

do / show marker

Dispatching

do / initiate delivery

Activity

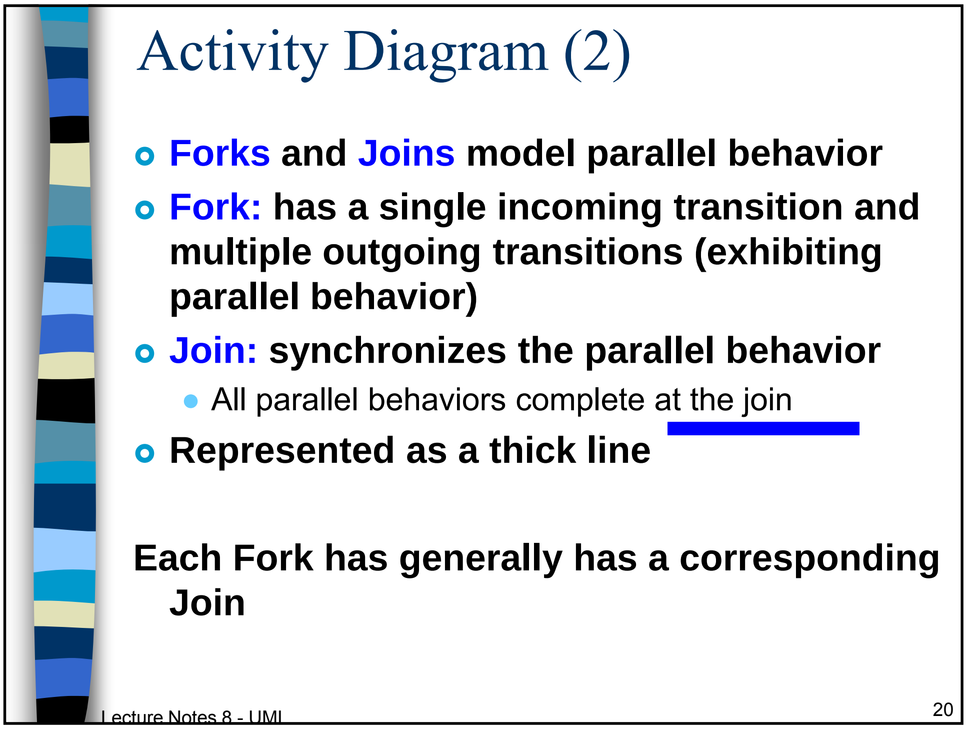


Activity Diagrams

- Describe
  - Procedural logic
  - Business process
  - Workflow
- A flow chart with support for parallel behavior
- Branches and Merges model the conditional behavior
- Branch: has a single incoming transition multiple, conditional, outgoing transitions
- Merge: where conditional behavior terminates

Each branch has a corresponding merge

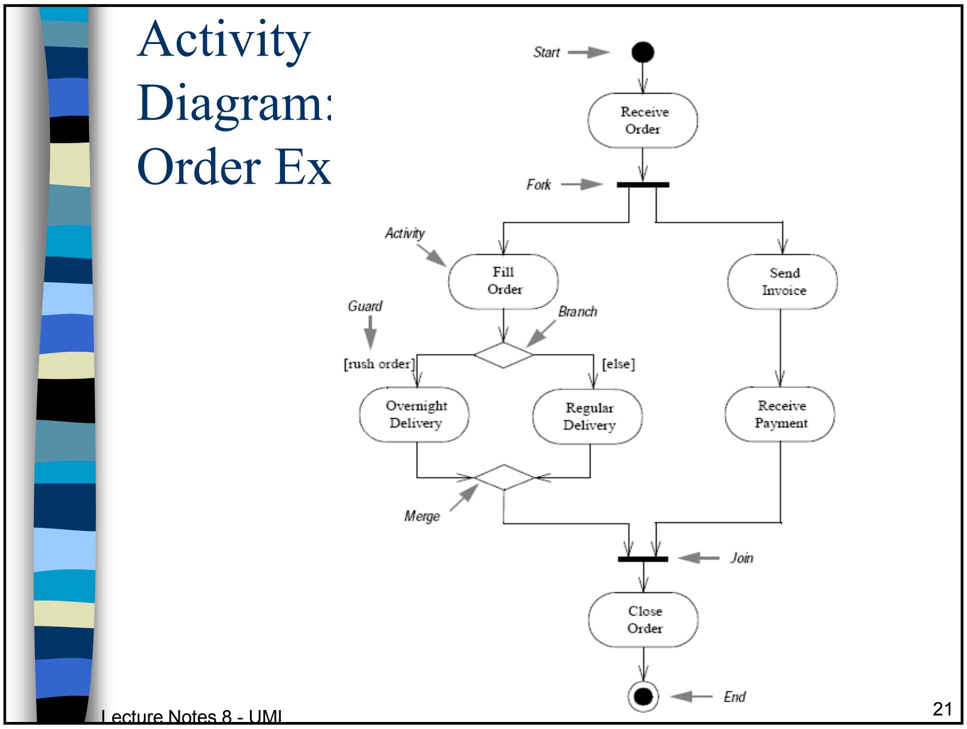
- Represented as a Diamond



Activity Diagram (2)

- Forks and Joins model parallel behavior
- Fork: has a single incoming transition and multiple outgoing transitions (exhibiting parallel behavior)
- Join: synchronizes the parallel behavior
  - All parallel behaviors complete at the join
- Represented as a thick line

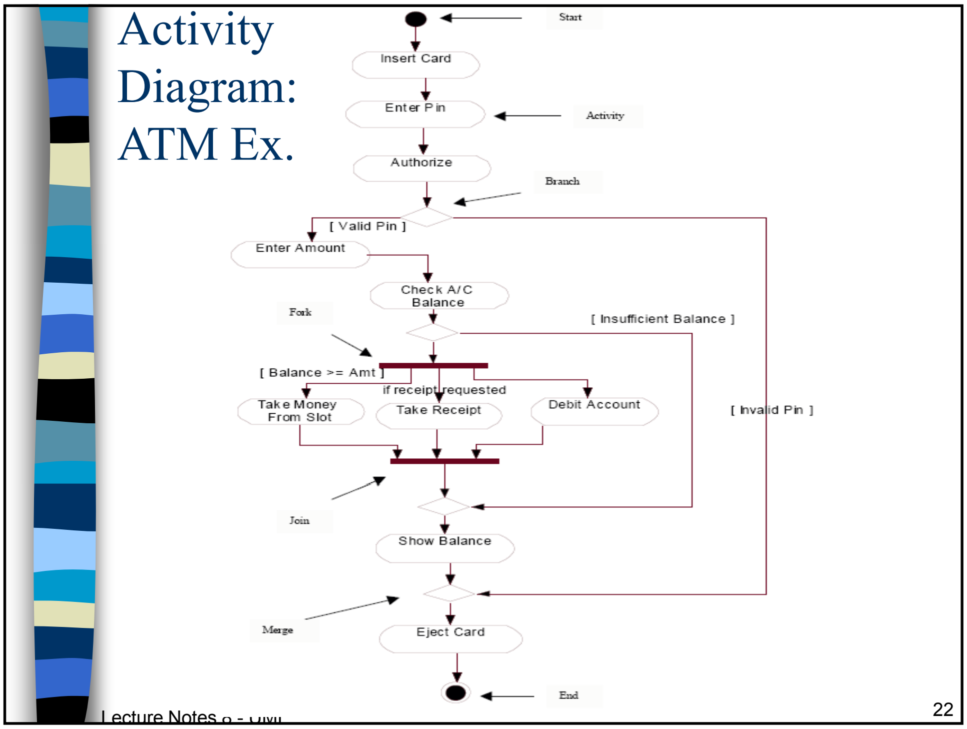
Each Fork has generally has a corresponding Join



Activity Diagram: Order Ex

Lecture Notes 8 - UML

21



Activity Diagram: ATM Ex.

Lecture Notes 8 - UML

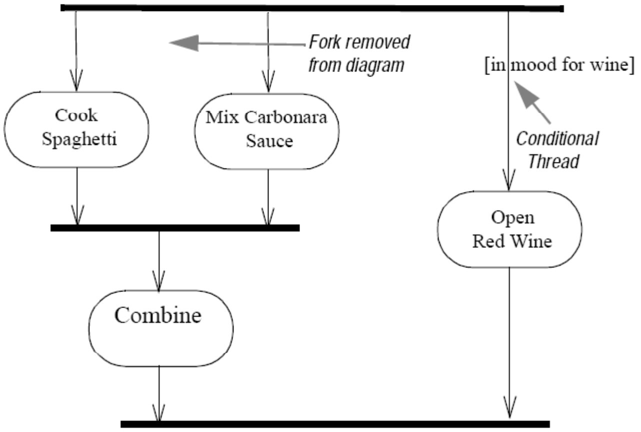
22

Conditional Thread

There are some exceptions to the each fork having a corresponding join:

- **Conditional Thread: A condition on the thread originating from the fork to create an exception for the join rule.**
  - If the condition is false then that condition is considered to be complete

Conditional Thread: Example

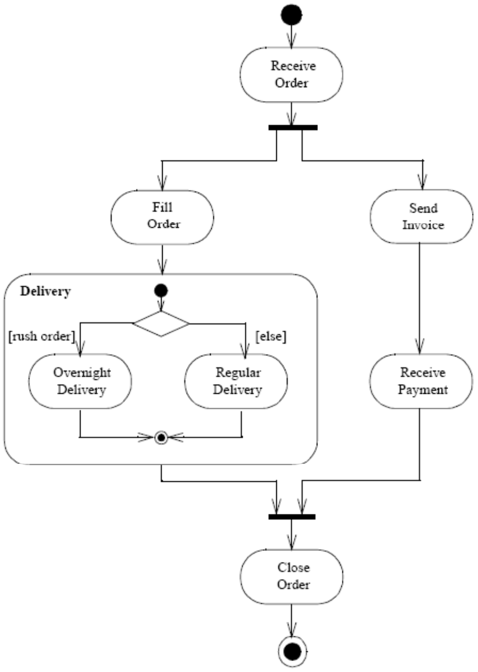


Superstates

- **What if you need to decompose your activity diagram?**
- **Superstates**
  - You can show the superstate with the internal behavior inside or
  - You can show these in a parent diagram
  - You can also use explicit initial and final states

    Adv: you can decouple the parent from the subsidiary and use it in other contexts

Activity Diagram: Superstate



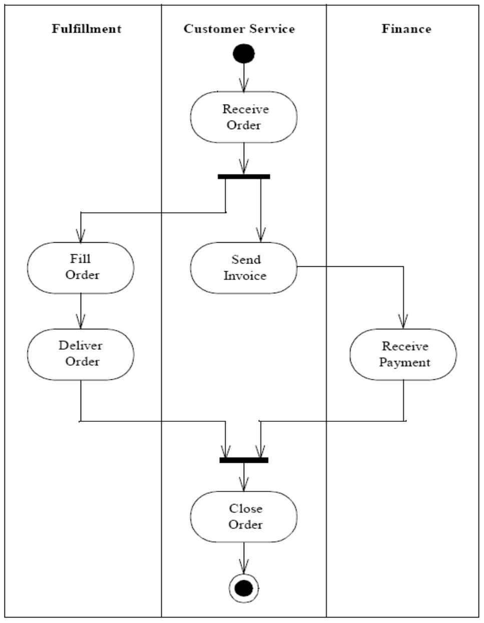
Partitioning an Activity Diagram

Activity diagrams tell you what is happening, but how do you know who does what?

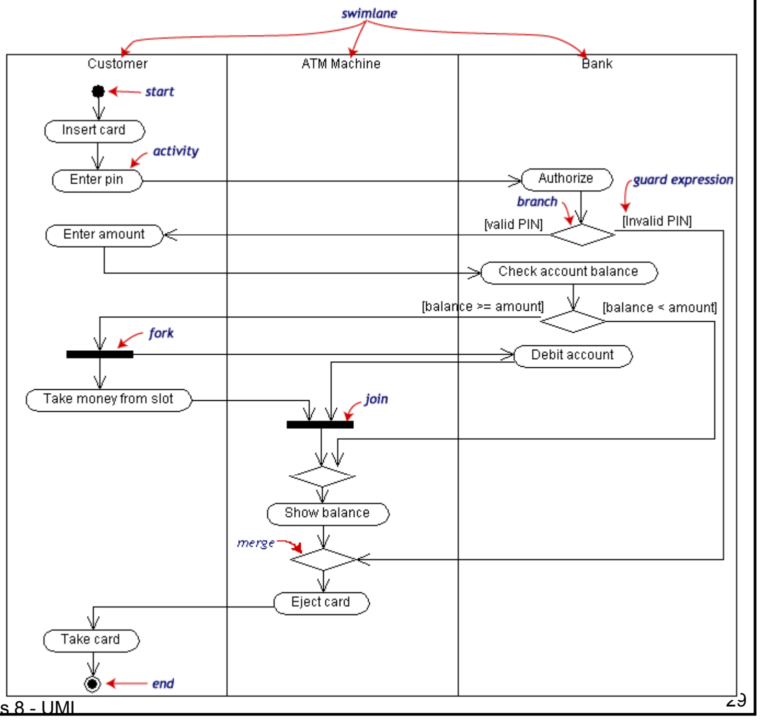
(in programming – which class is responsible for each activity)

- **Swimlanes: group related activities into one column (usually organizationally)**
  - You must arrange your diagram into vertical zones separated by lines.
  - Can be difficult with complex diagrams
    - In this case use non-linear zones – better than nothing

Swimlanes

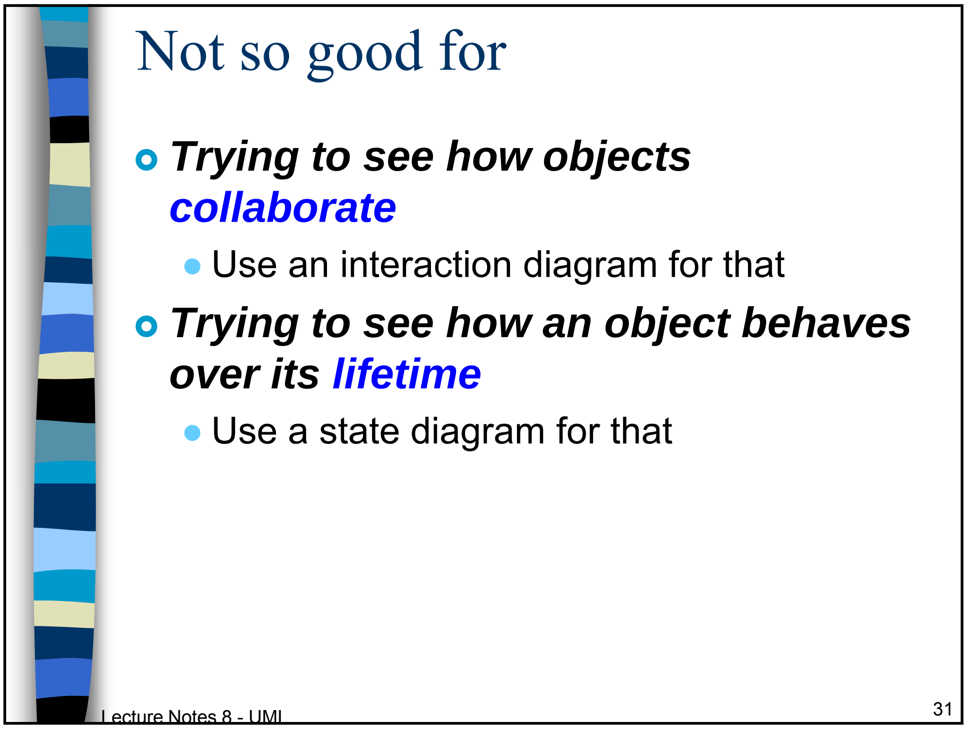


Atm Ex.



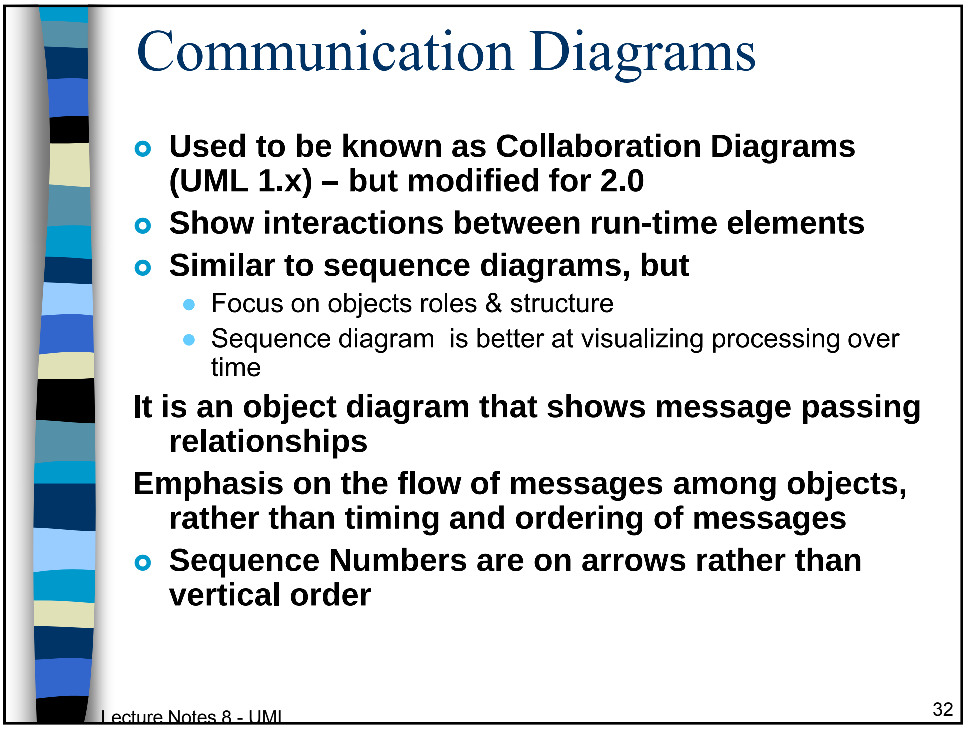
When do you use Activity Diagrams?

- ***Modeling parallel behavior***
- ***Analyzing a use case***
  - Trying to understand what actions need to take place
  - Determine behavioral dependencies
- ***Understanding workflow***
  - Documenting the logic of a business process
- ***Describing a complicated sequential algorithm***
- ***Dealing with multi-threaded applications***



Not so good for

- ***Trying to see how objects collaborate***
  - Use an interaction diagram for that
- ***Trying to see how an object behaves over its lifetime***
  - Use a state diagram for that



Communication Diagrams

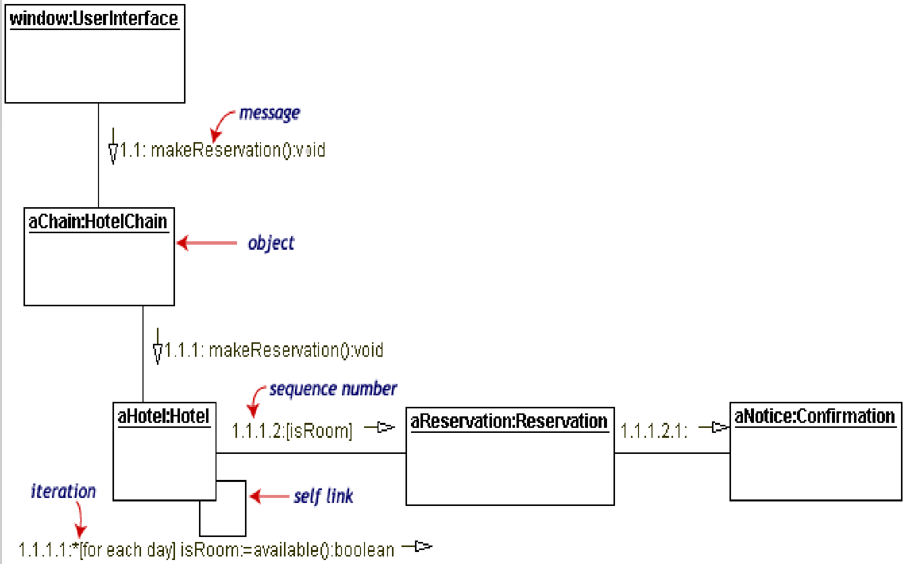
- **Used to be known as Collaboration Diagrams (UML 1.x) – but modified for 2.0**
- **Show interactions between run-time elements**
- **Similar to sequence diagrams, but**
  - Focus on objects roles & structure
  - Sequence diagram is better at visualizing processing over time

It is an object diagram that shows message passing relationships

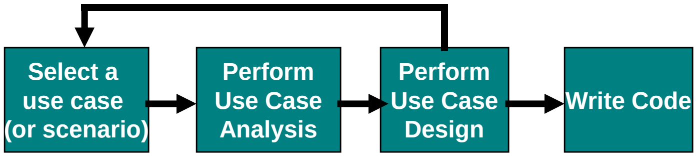
Emphasis on the flow of messages among objects, rather than timing and ordering of messages

- **Sequence Numbers are on arrows rather than vertical order**

Communication Diagrams: Ex



From Use Cases to Code



Select a use case (or scenario)

Perform Use Case Analysis

Perform Use Case Design

Write Code

Use Case Analysis

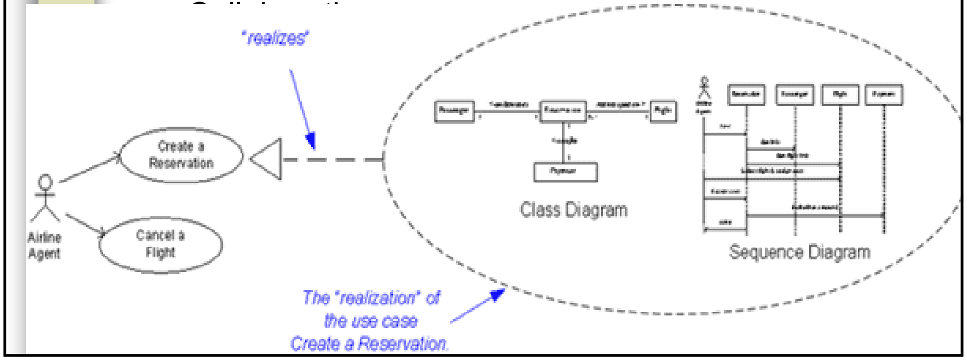
For each use case in an iteration…

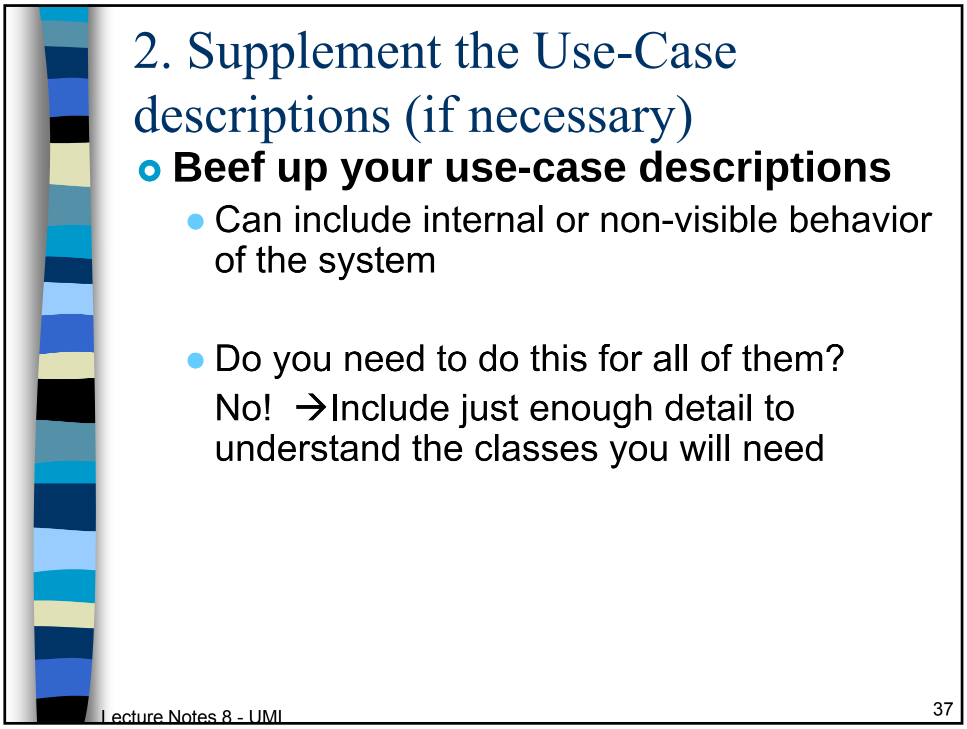
1. **Create a use case realization**
2. **Supplement the Use-Case descriptions**
   - if necessary
3. **Find Analysis Classes from Use-Case Behavior**
4. **Distribute Behavior to Analysis Classes**

1. Use-Case Realization

A *use-case realization* is a collection of UML diagrams which together validate that we have

- the classes → Class Diagrams (static relationships)
- responsibilities
- object interactions → Interaction Diagrams (dynamic relationships) – could be Sequence or

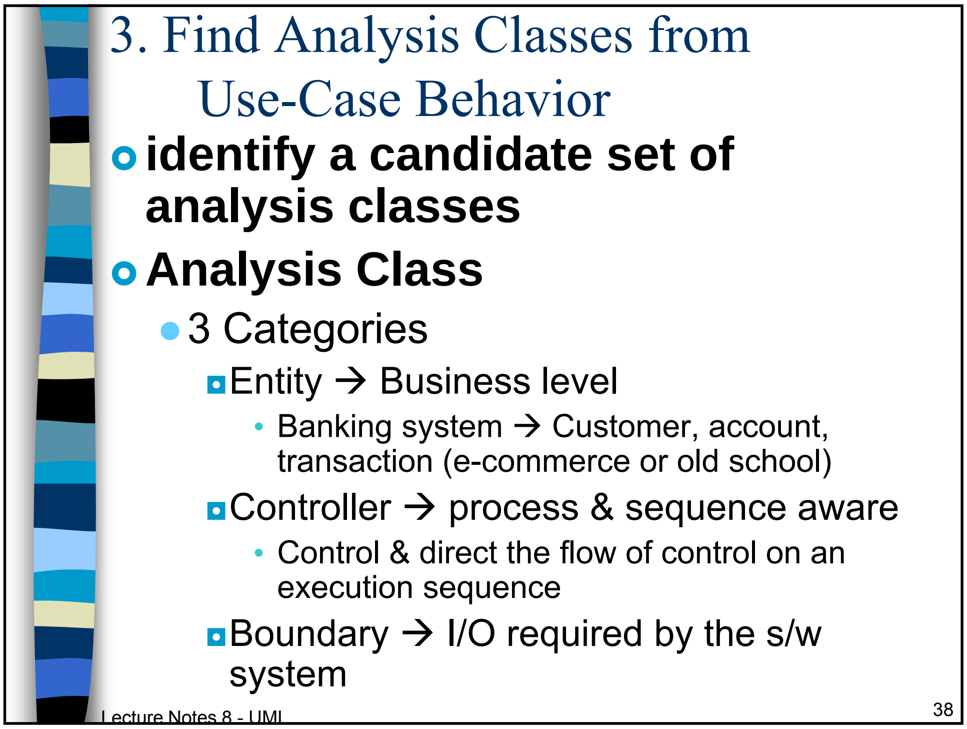




2. Supplement the Use-Case descriptions (if necessary)

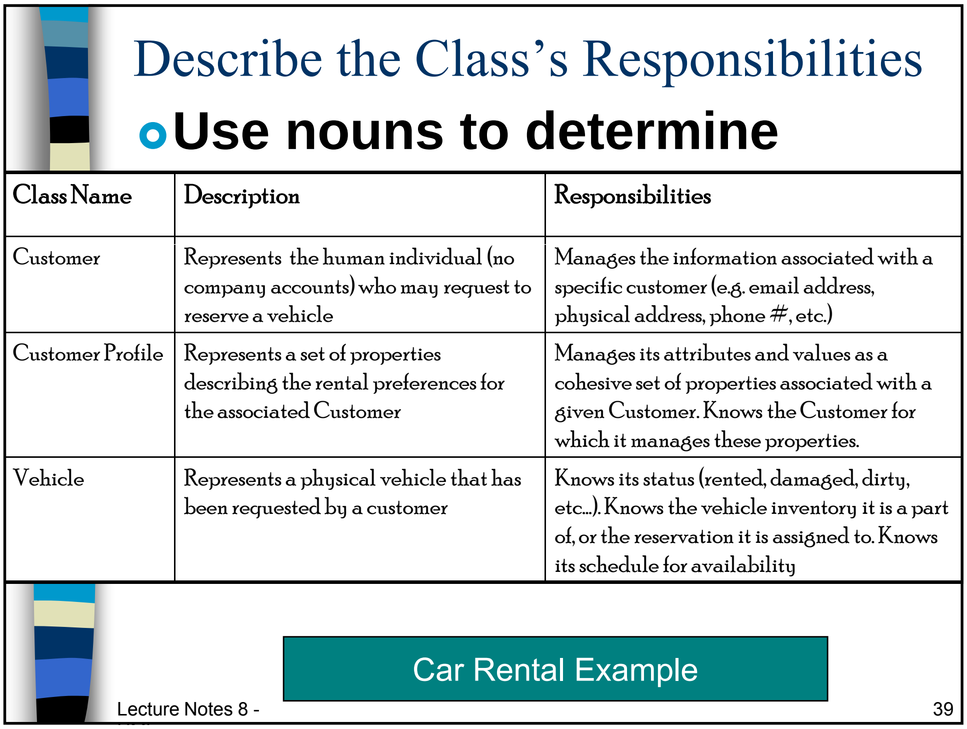
- **Beef up your use-case descriptions**
  - Can include internal or non-visible behavior of the system
  - Do you need to do this for all of them?

    No! →Include just enough detail to understand the classes you will need



3. Find Analysis Classes from Use-Case Behavior

- **identify a candidate set of analysis classes**
- **Analysis Class**
  - 3 Categories
    - Entity → Business level
      - Banking system → Customer, account, transaction (e-commerce or old school)
    - Controller → process & sequence aware
      - Control & direct the flow of control on an execution sequence
    - Boundary → I/O required by the s/w system

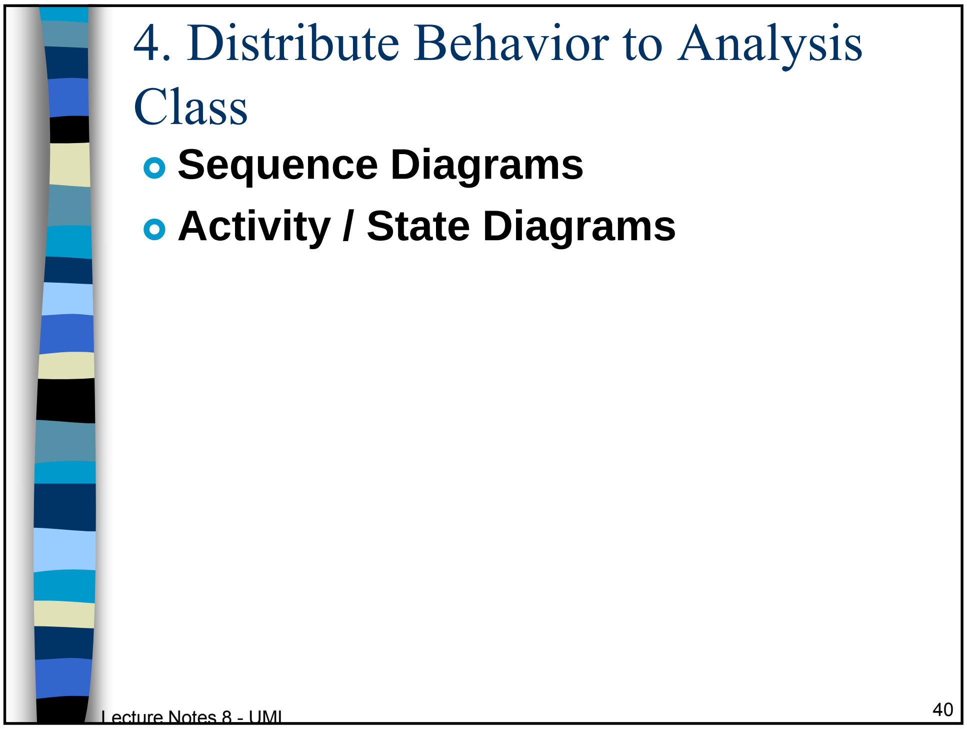


Describe the Class’s Responsibilities

- **Use nouns to determine**

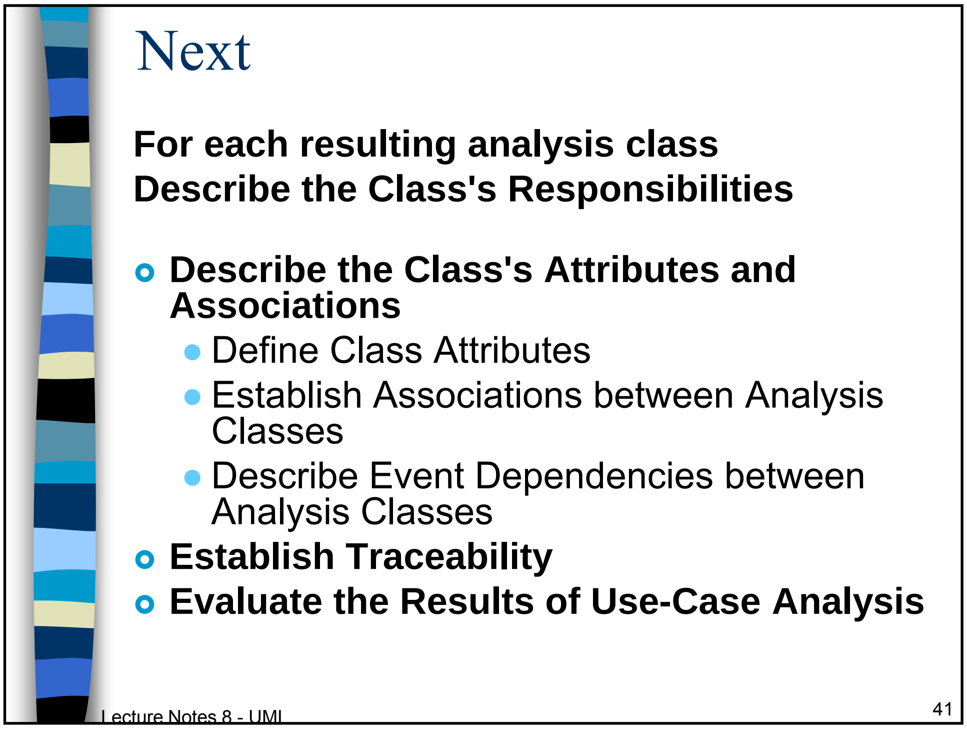
Class Name	Description	Responsibilities
Customer	Represents the human individual (no company accounts) who may request to reserve a vehicle	Manages the information associated with a specific customer (e.g. email address, physical address, phone #, etc.)
Customer Profile	Represents a set of properties describing the rental preferences for the associated Customer	Manages its attributes and values as a cohesive set of properties associated with a given Customer. Knows the Customer for which it manages these properties.
Vehicle	Represents a physical vehicle that has been requested by a customer	Knows its status (rented, damaged, dirty, etc…). Knows the vehicle inventory it is a part of, or the reservation it is assigned to. Knows its schedule for availability

Car Rental Example



4. Distribute Behavior to Analysis Class

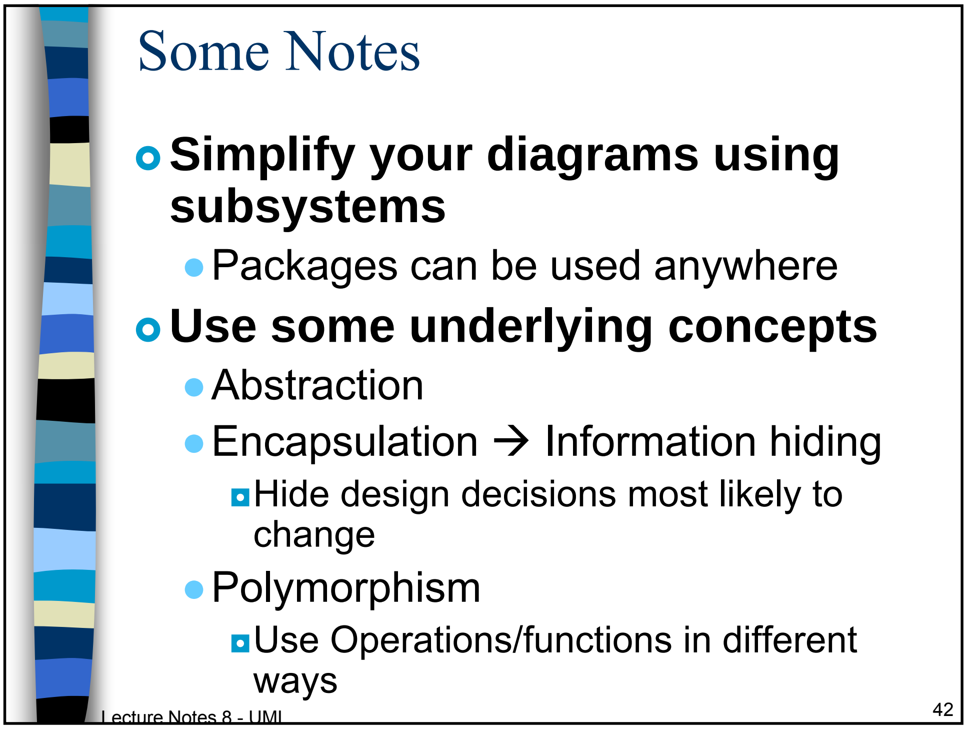
- **Sequence Diagrams**
- **Activity / State Diagrams**



Next

For each resulting analysis class
Describe the Class's Responsibilities

- **Describe the Class's Attributes and Associations**
  - Define Class Attributes
  - Establish Associations between Analysis Classes
  - Describe Event Dependencies between Analysis Classes
- **Establish Traceability**
- **Evaluate the Results of Use-Case Analysis**



Some Notes

- **Simplify your diagrams using subsystems**
  - Packages can be used anywhere
- **Use some underlying concepts**
  - Abstraction
  - Encapsulation → Information hiding
    - Hide design decisions most likely to change
  - Polymorphism
    - Use Operations/functions in different ways