UML: Class Diagrams, Use-Case Diagrams

17 February 2011
CMPT166
Sean Ho
Trinity Western University

TRINITY
WESTERN
UNIVERSITY

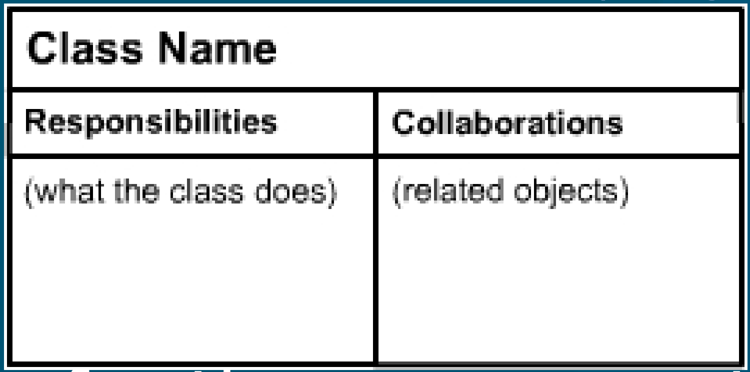
What's on for today

- What is UML?
- CRC diagrams and class diagrams
  - Subclass, association, aggregation, composition
- Use-case diagrams
  - Actors and use cases
  - Direction, multiplicity, includes
  - Basic flow, alternate flows
- Component design

Unified Modeling Language

- Diagrams to help design your programs
- Main diagram types:
  - Static: Class diagram, object, package
  - Dynamic: Use case diagram, sequence diagram, state chart
- Draw by hand, or use software tools: Eclipse EMF, MS Visio, Oracle NetBeans
- By Booch, Rumbaugh, and Jacobson, of OMG (Object Management Group)
  - Current version is 2.0: www.uml.org

CRC diagrams

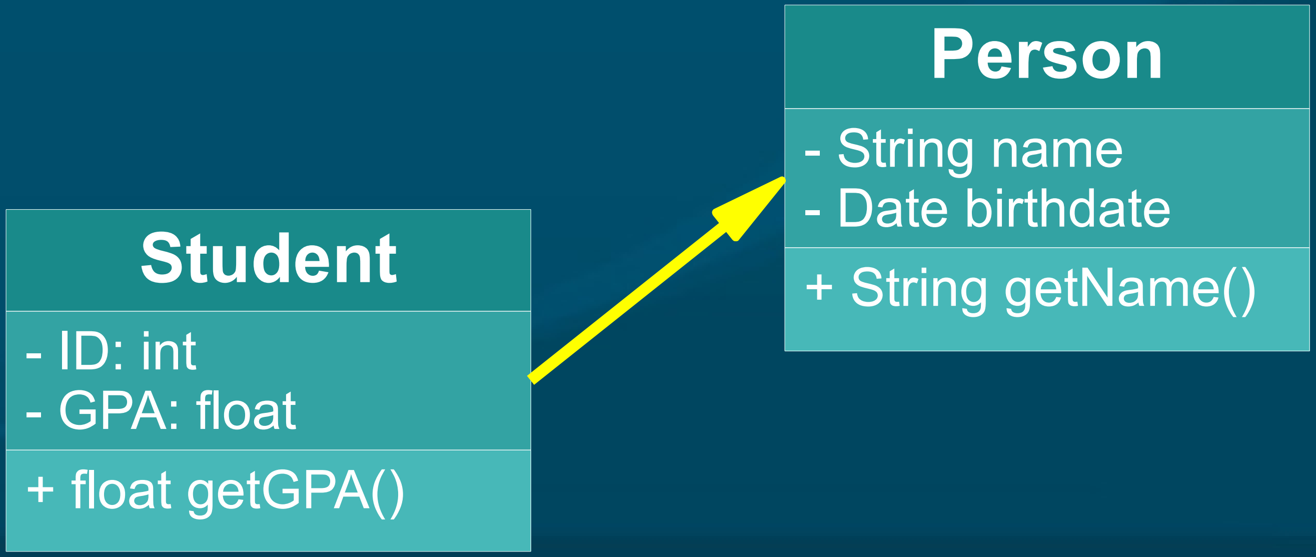


Class Name	
Responsibilities	**Collaborations**
(what the class does)	(related objects)

- Class:
  - Short descriptive name for the component
- Responsibility:
  - Data stored in the class
  - Restrictions on access to the data
  - Actions the class is responsible for
- Collaborator:
  - e.g., types of our attributes/data
  - Other classes whose methods we call
  - Other classes who call our methods

UML class diagram

- Each box represents a class (type)
  - Name, attributes, methods
  - Static (class) members are underlined
  - Flag: public (+), private (-), protected (#)
- Lines show relationships between classes



Student

- ID: int
- GPA: float

+ float getGPA()

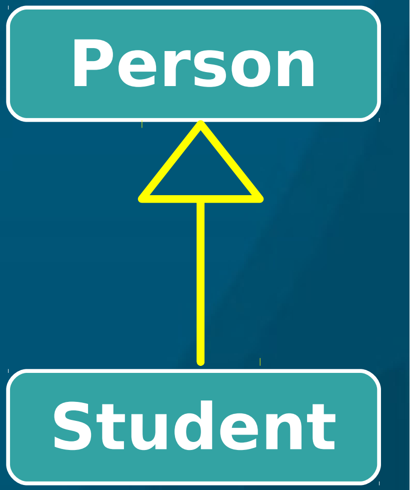
Person

- String name
- Date birthdate

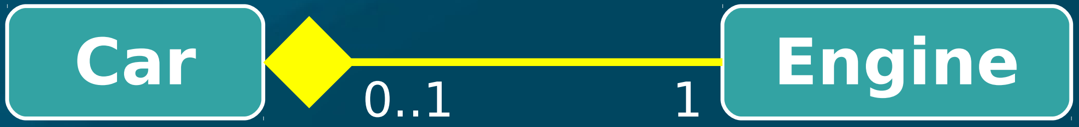
+ String getName()

Class diagram: relationships

- Class relationships: e.g., superclass:
  - Hollow arrow-head pointing to super



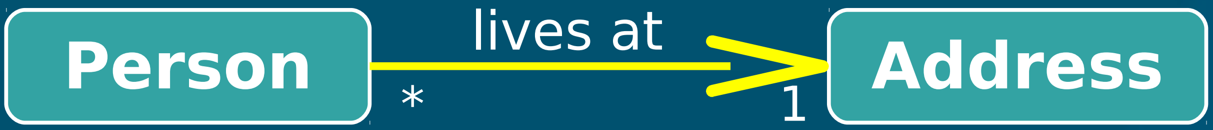
- Instance relationships:
  - e.g., every Car has an instance of Engine
  - relationship between instances of the classes, not entire classes
- Multiplicity: e.g., "*": any number of instances



  - "1..*": one or more instances
  - "0..1": optional one instance

Instance relationships

- Association: label with the relationship; arrow indicates direction of dependency



Person

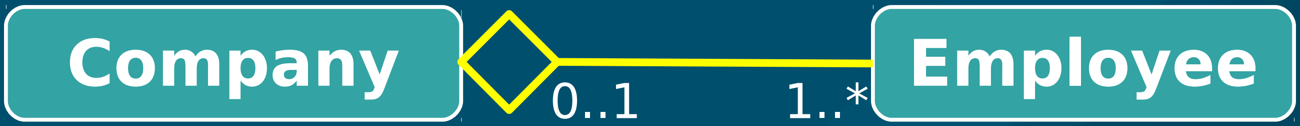
lives at

*

1

Address

- Aggregation: container
  “A is part of B” (but can exist apart from B)



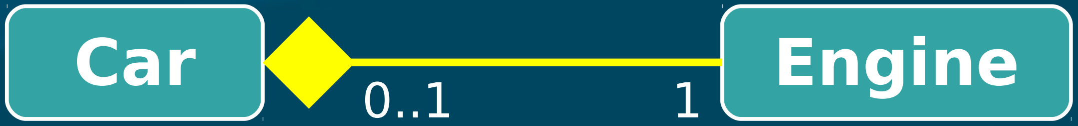
Company

0..1

1..*

Employee

- Composition: “B owns an A”
  Life-cycle dependency:
  when B dies, so should its instance of A



Car

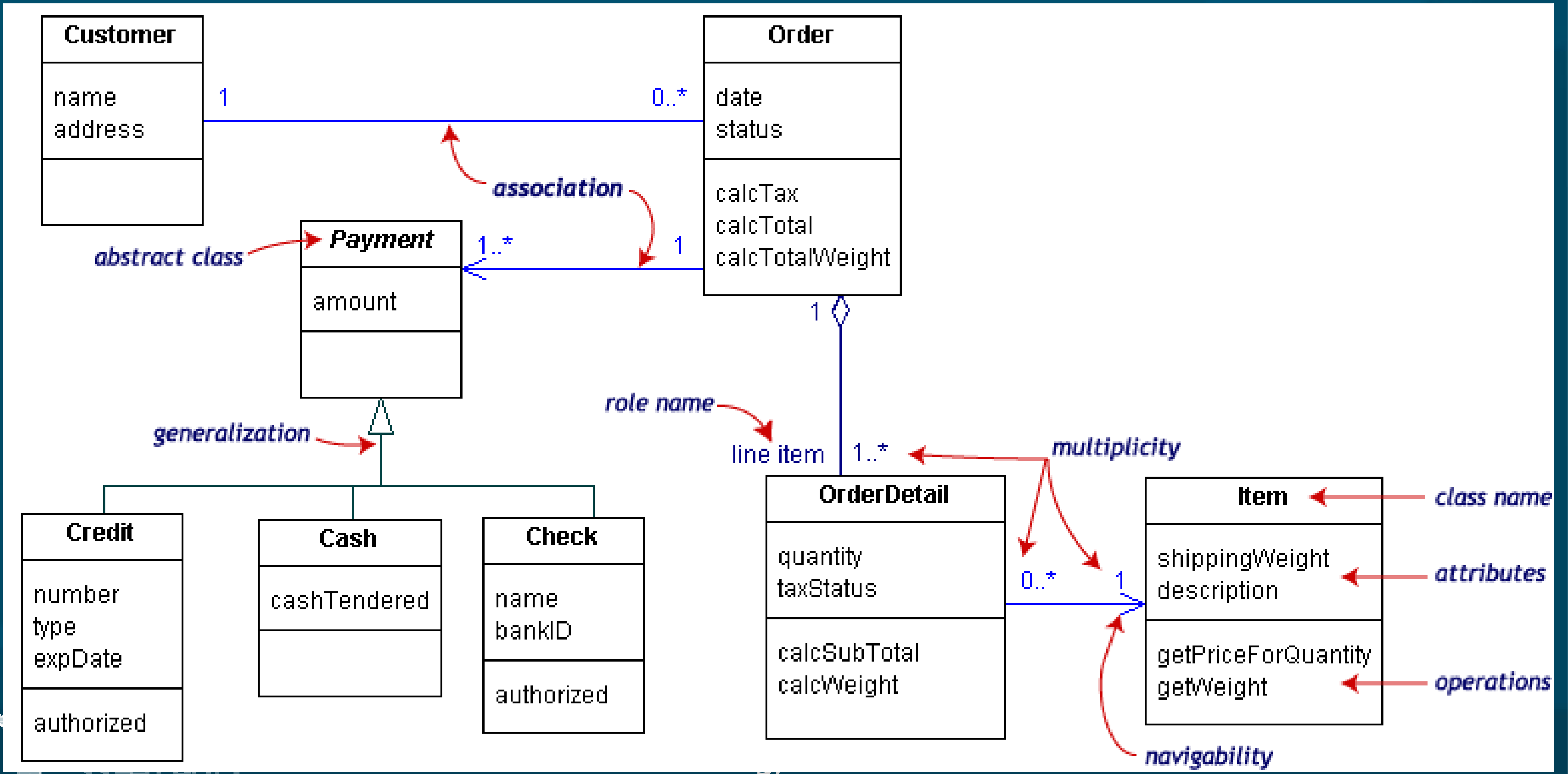
0..1

1

Engine

An example class diagram

- Ordering system: each Order has multiple OrderDetail line items

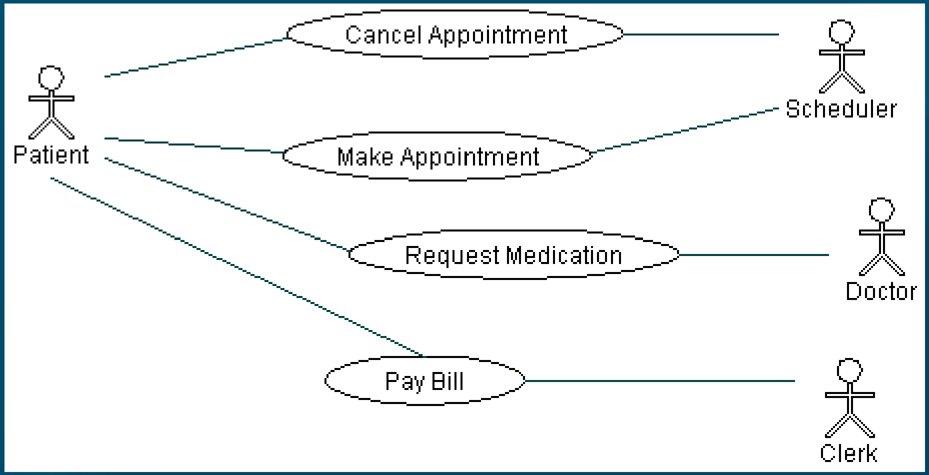


Steps to OO design: wADes

  - (Prereq: understand client requirements)
- System behaviour
  - Use-case scenarios
  - User interface mockups
- Components
  - Self-contained blocks with narrow interactions
- From components to classes
  - Attributes, methods, relationships

UML: Use case diagram

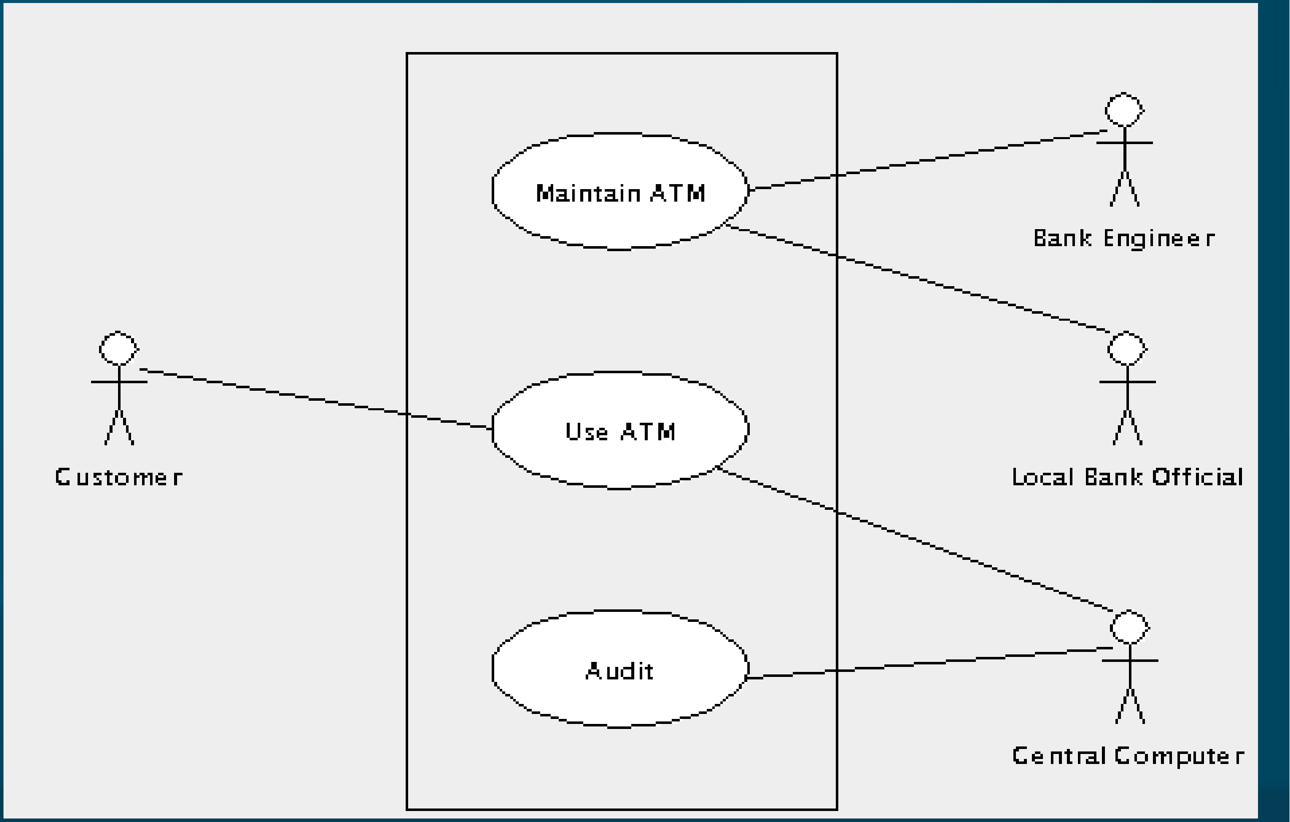
- Describes relationships between actors:
  - Patient calls the clinic to make an appointment
  - Receptionist books timeslot
  - Patient sees doctor and requests medication
  - Patient pays bill to clerk



- More details: Borland's UML tutorial

System behaviour: use-case

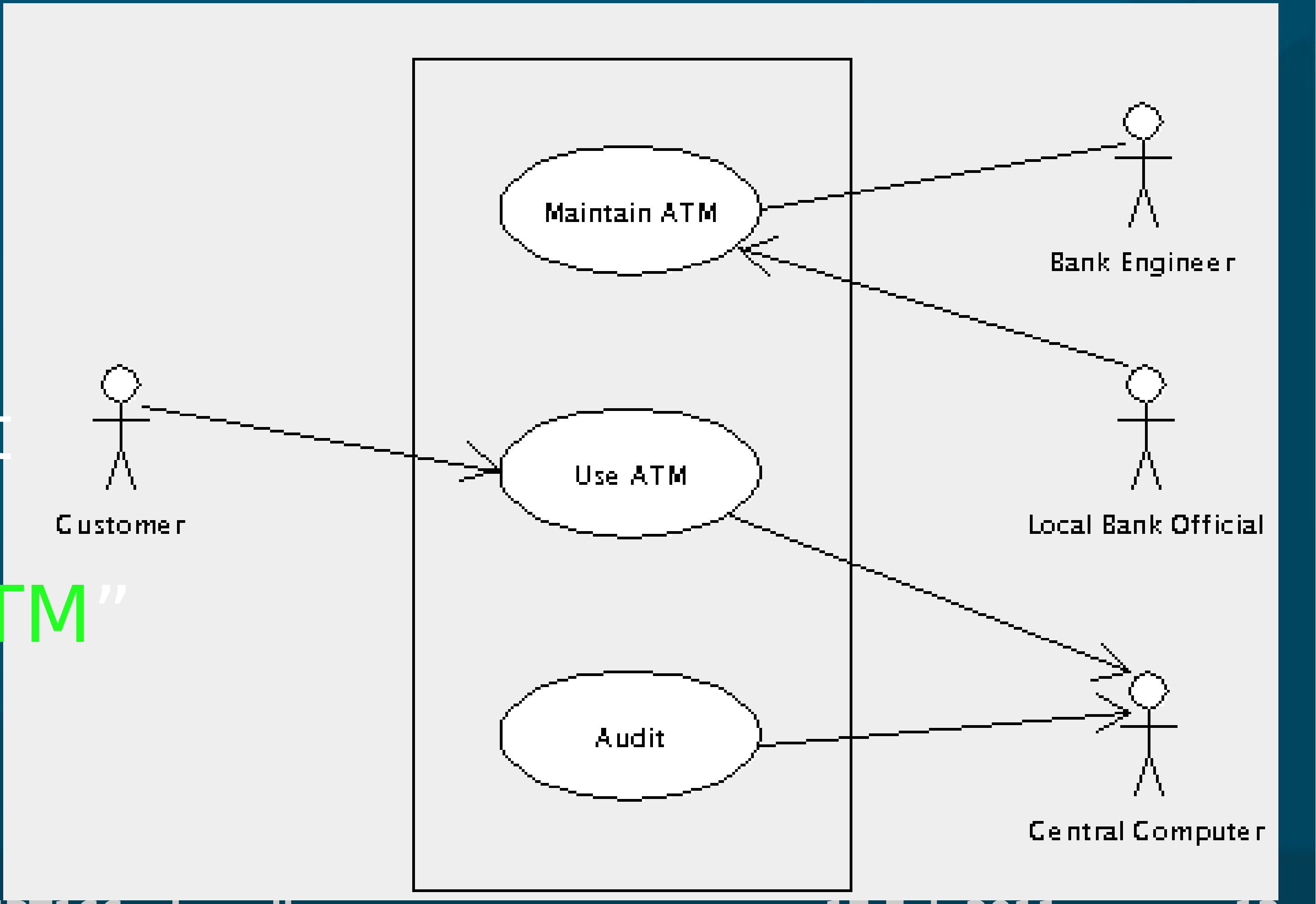
- UML use-case diagrams show:
  - The actors involved (may be nonhuman!)
  - Ways in which the actors interact: relationships, actions, use cases, etc.
- Example: ATM (thanks to ArgoUML)



Use case: navigation

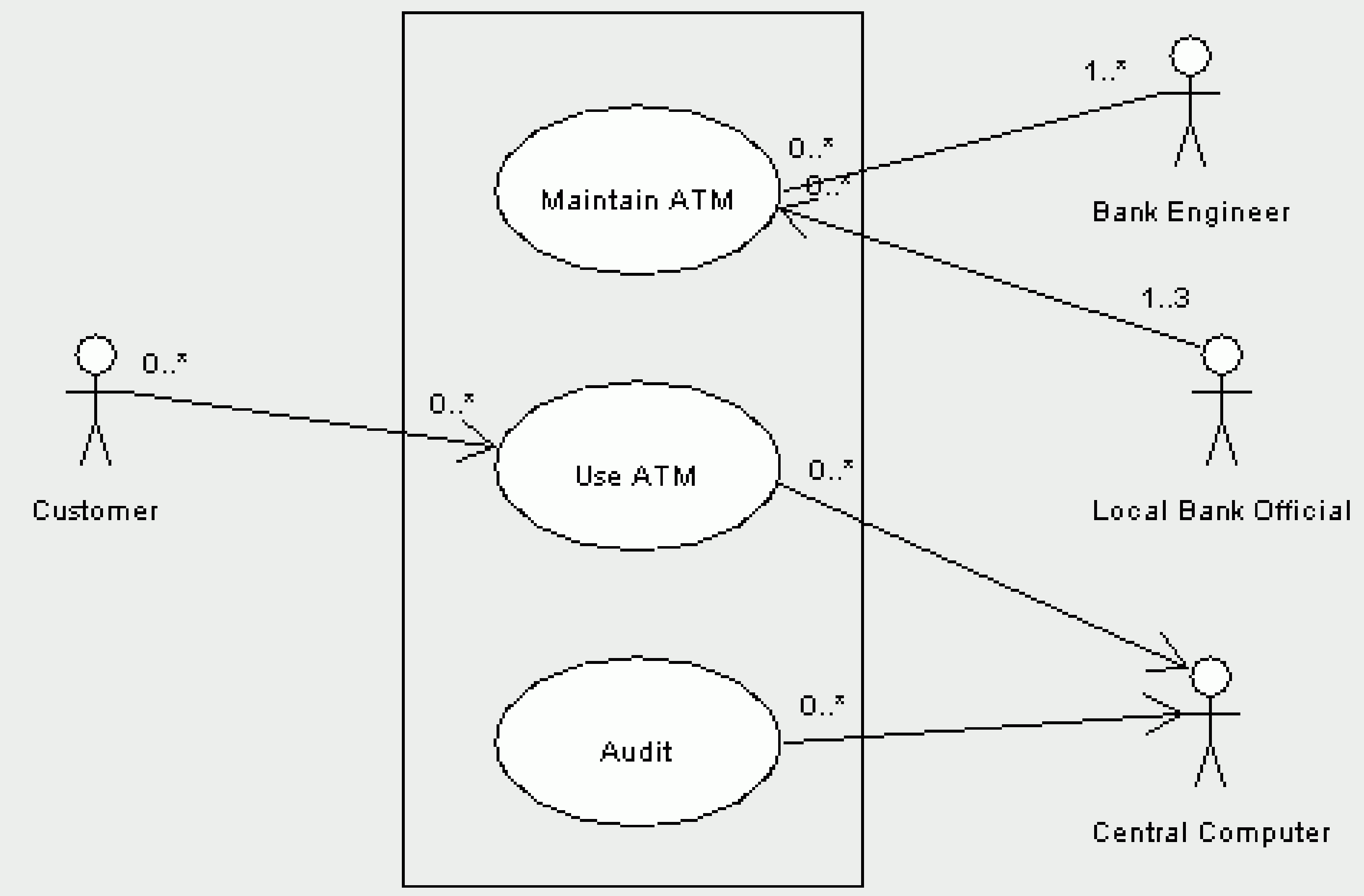
- Direction of arrows indicates which actor is passive and which is active:

- What direction should the arrows point between "Maintain ATM" and "Engineer"?



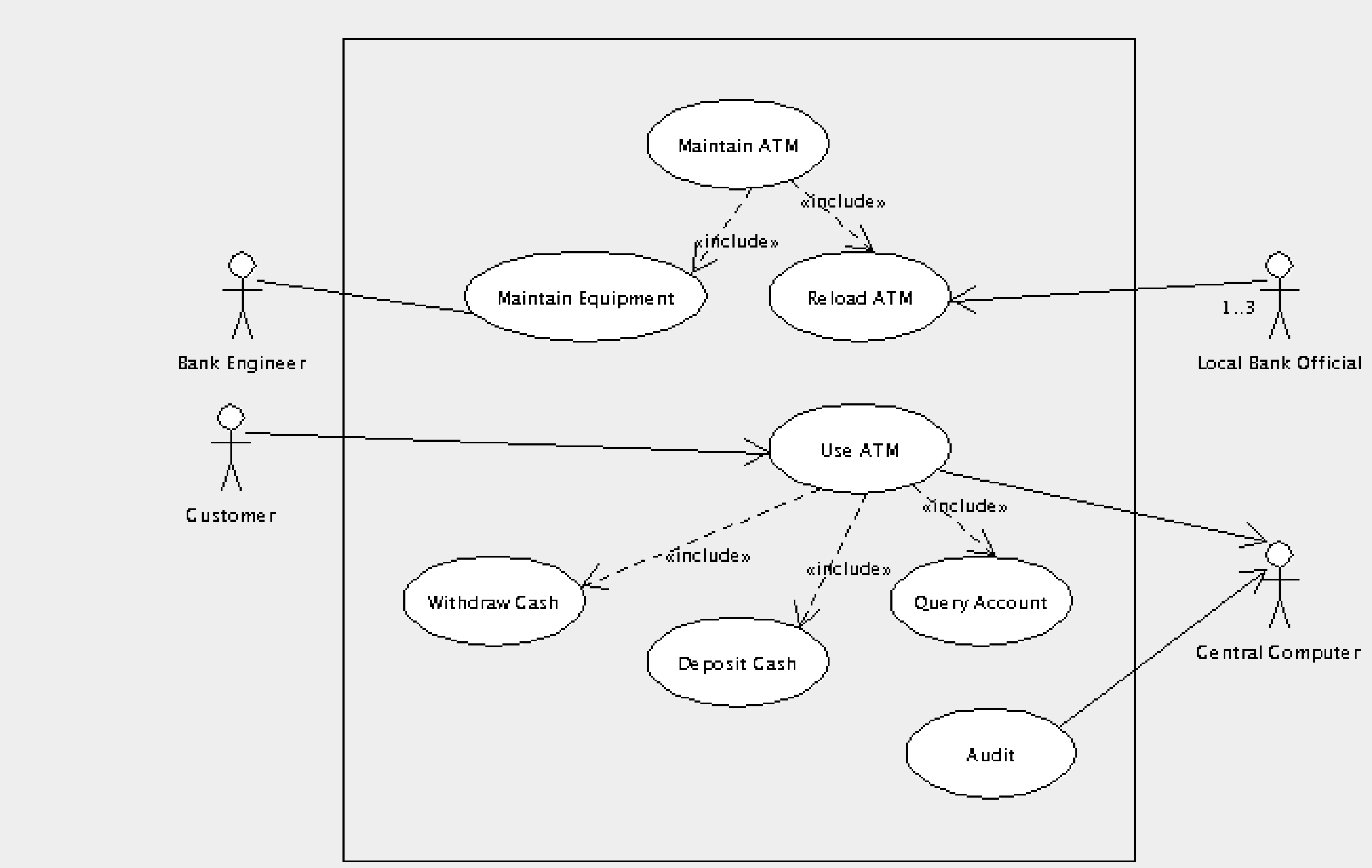
Use case: multiplicity

- Numbers indicate how many instances of an actor can be doing how many instances of the use case
- e.g., only allow up to 3 Bank Official



Use case diagram: includes

- We may need to break down each use case into smaller chunks to implement



Components of a use case

- Each use case should have:
  - Short name
  - Goal: what does it achieve for its actors?
  - Names of actor(s) involved
  - Pre/post-conditions?
  - Basic flow: break down into steps (pseudocode!)
  - Alternate flows: what if user inputs something different from the usual?

Ex. use case: Withdraw Cash

- Name: Withdraw cash
- Goal: Customer gets cash; Computer ensures sufficient funds, logs a record
- Actors: Customer, Central Computer
- Basic flow:
  - Customer selects account to withdraw from
  - Customer inputs dollar amount of cash
  - ATM verifies with Computer enough money
  - ATM dispenses cash to Customer
  - ATM prints receipt

Ex. use case: alternate flows

- How might the basic flow not work? What might go wrong?
  - *ATM out of cash*
  - *NSF in customer account*
  - *Wrong PIN, bad card*
  - *Withdraw negative amount*
  - *Unverified deposit*
  - *Network error, forget to logout*
- Each results in an alternate flow: how to handle that alternate situation