# UML: Use-Case Diagrams

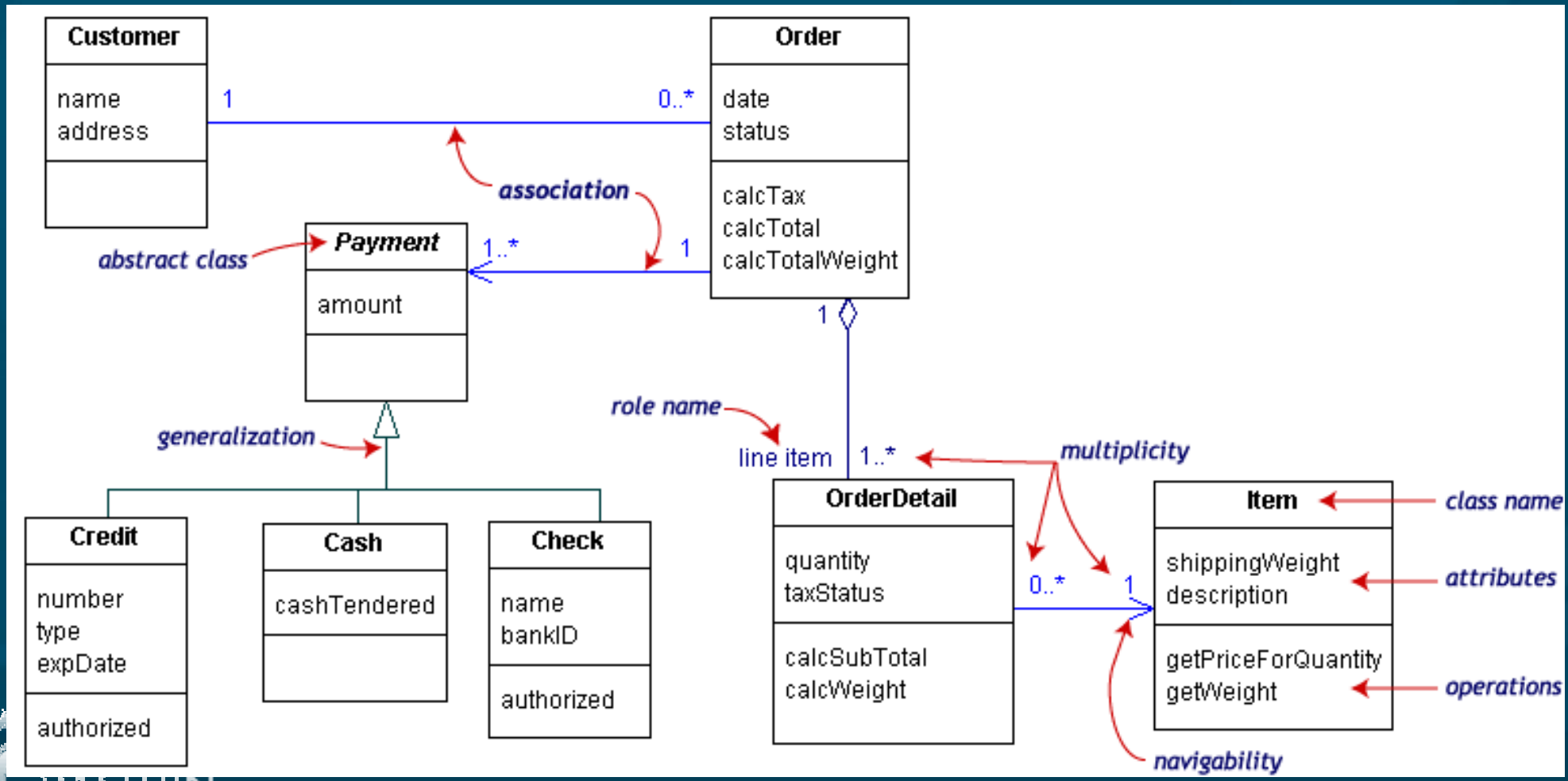
Reference:
Borland's UML tutorial

8 Mar 2010
CMPT166
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

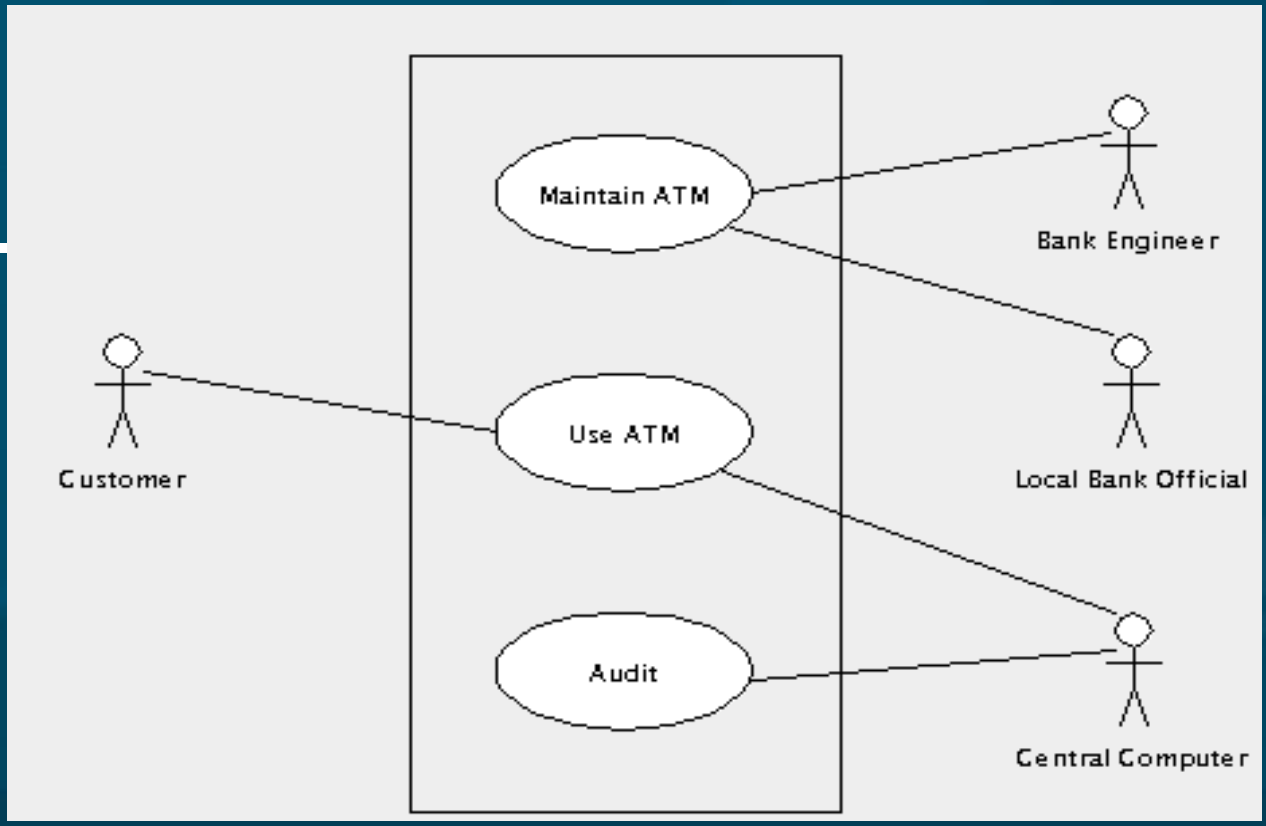
# UML class diagrams

- Ordering system:
  each Order has multiple OrderDetail line items



# System behaviour: use-case

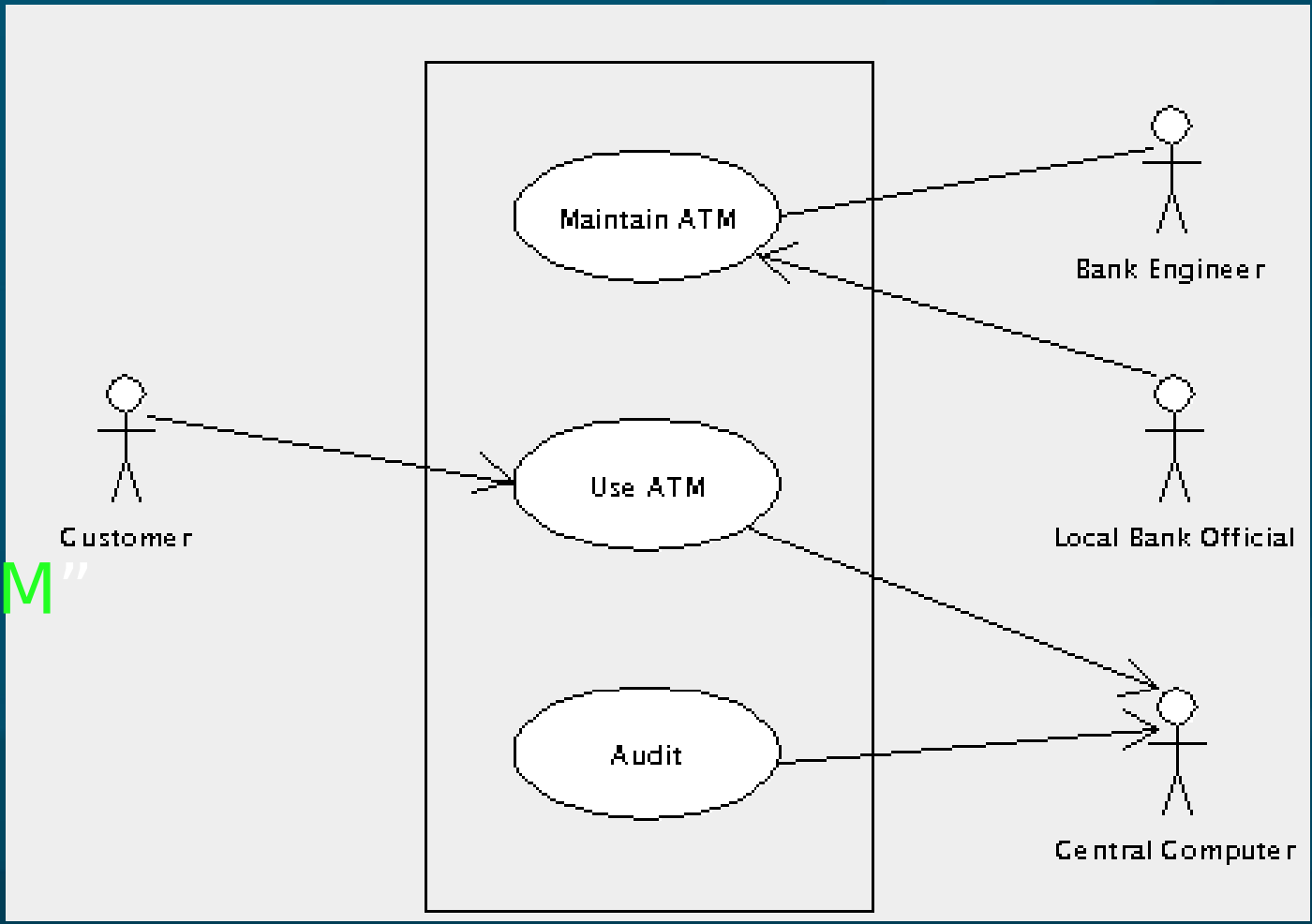
- UML use-case diagrams show:
  - The actors involved (may be nonhuman!)
  - Ways in which the actors interact: relationships, actions, use cases, etc.
- Example: ATM (thanks to ArgoUML)



# Use case diagram: navigation

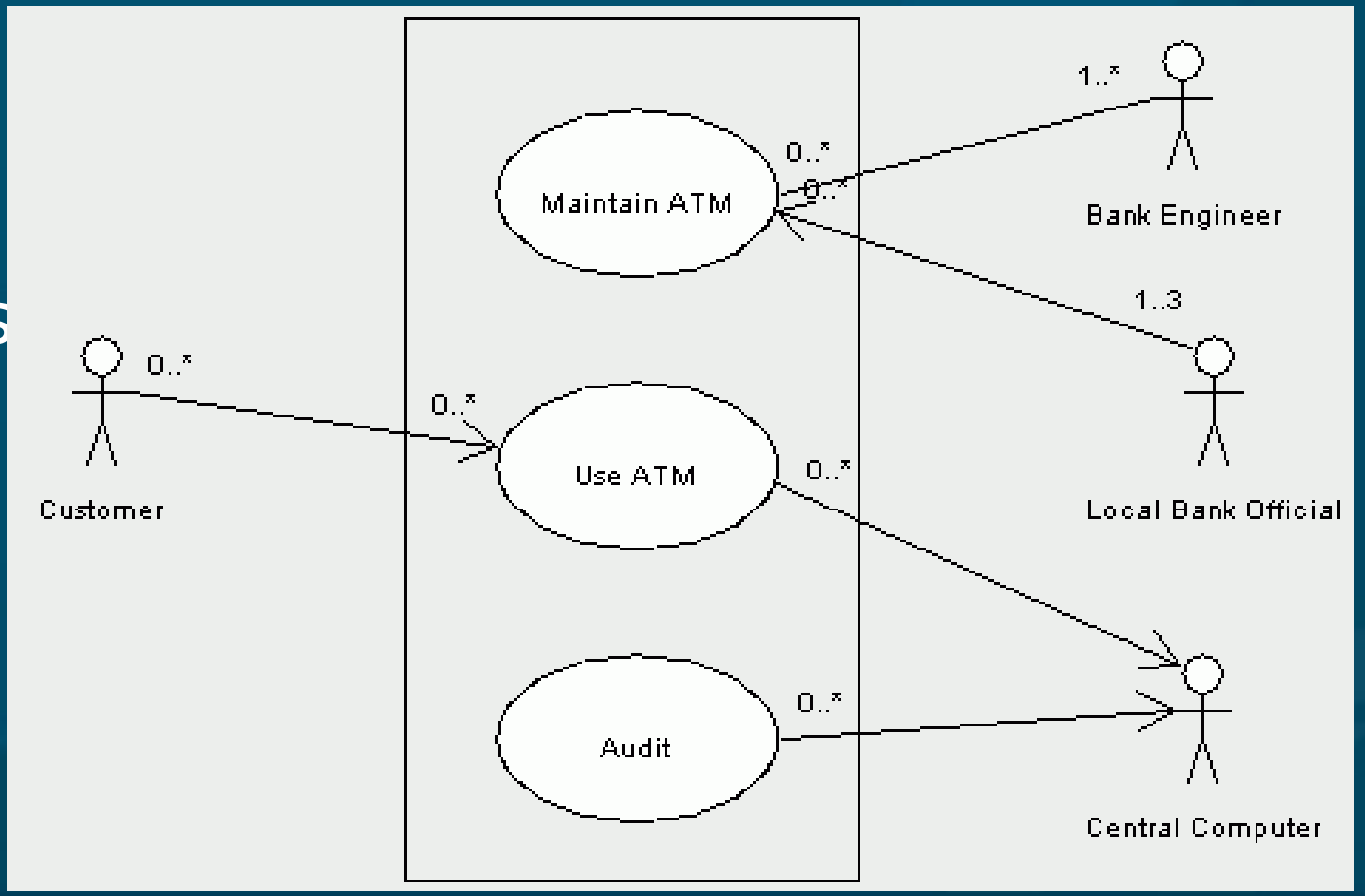
- Direction of arrows indicates which actor is passive and which is active:

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



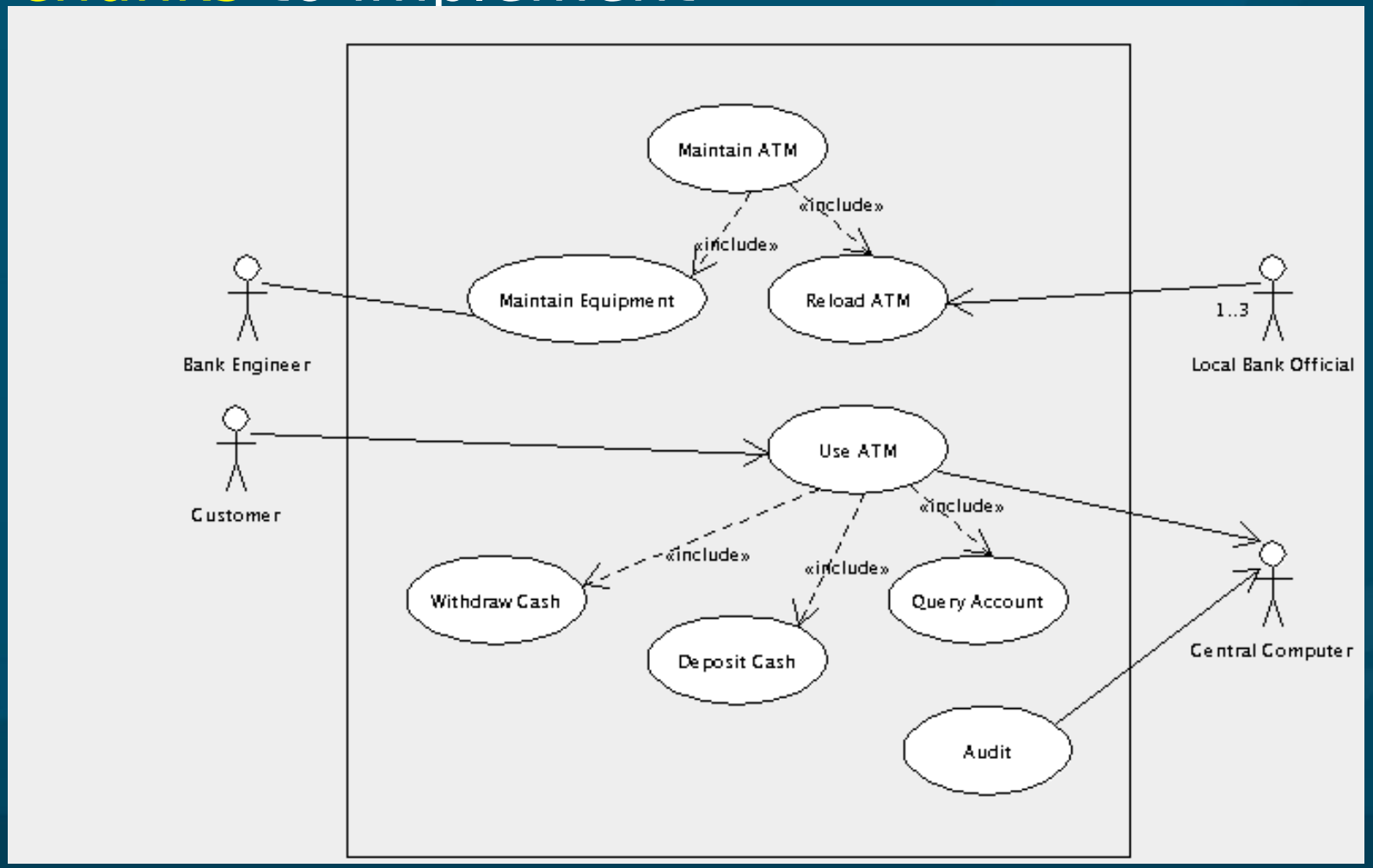
# Use case diagram: multiplicity

- 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 Officials



# 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 account has enough money and keeps 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?
  - *wrong PIN, bank card can't be read*
  - *insuff. funds*
  - *different currency*
  - *mismatch btw check and entered value*
  - *power outage, network probs*
  - *timeout / cust walks away*
- Each results in an alternate flow:
  how to handle that alternate situation

# Steps to OO design: wADes

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

# An OO design exercise

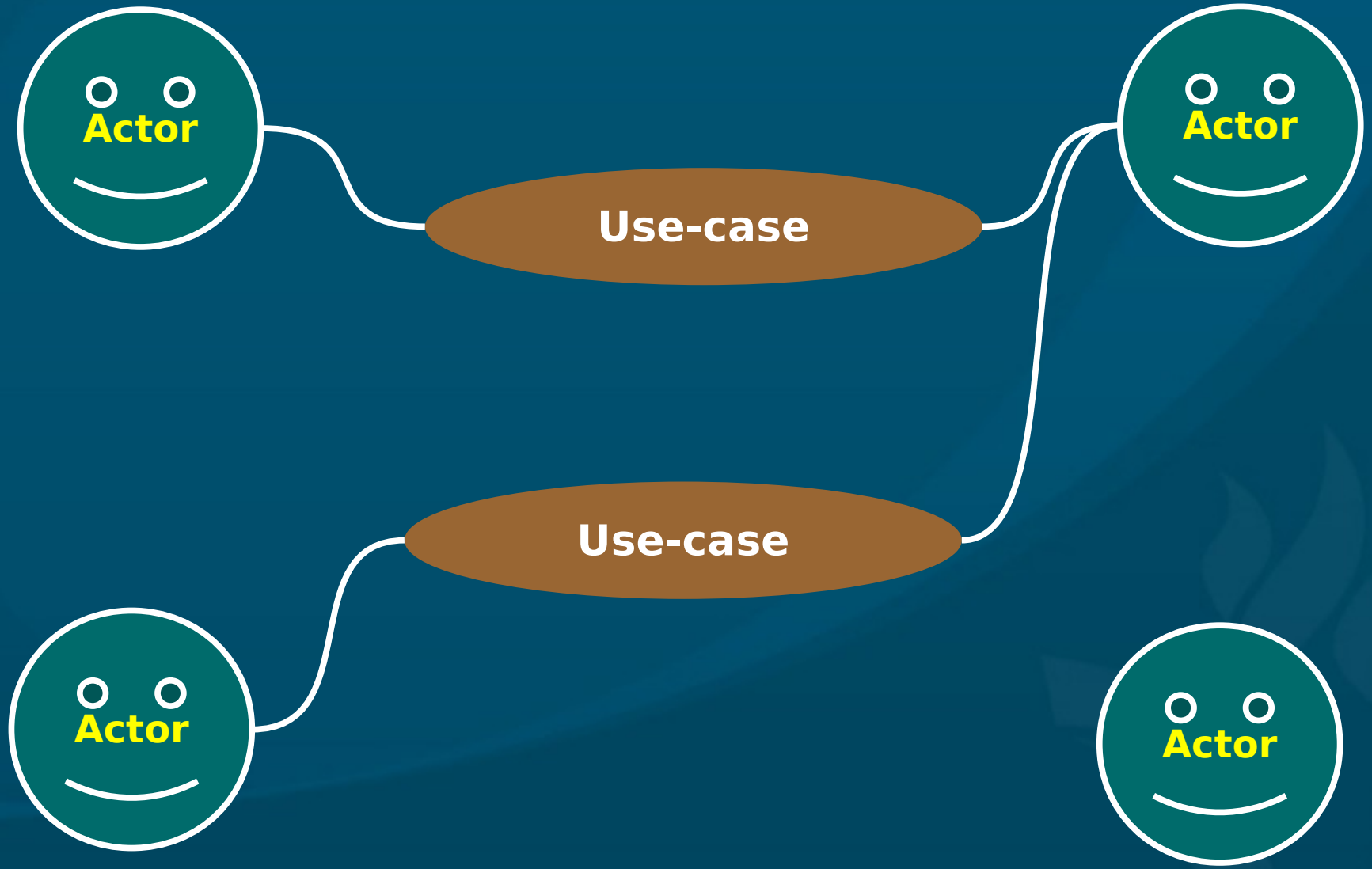
- Problem statement:

  Design a student enrolment database like we have at TWU

# (1) Actors and actions

- Use-case scenarios: actors and actions
- Who are the actors? Who will interface with us?
  - *students, advisors, teachers, admins*
  - *(courses: w/in system or outside?), grade system?*
- What are the actions? Scenarios of use?
  - *apply for admission, check course capacity*
  - *add course, approve courses*
  - *remove unpaying student, change capacity of course*
  - *change teacher/room/number, create new course*
  - *get/change contact info?*
  - *get list of courses for a student / students for a course*

# (1) Use-case diagrams



# (1) Specify one use case

- Name: *Add Course*
- Actors: *Student*
- Goal(s): *Course is added to student's schedule*
- Pre-conditions: *student is current/admitted, currently within registration window, course is offered and open for registration, ...*

# (1) Use case: basic flow

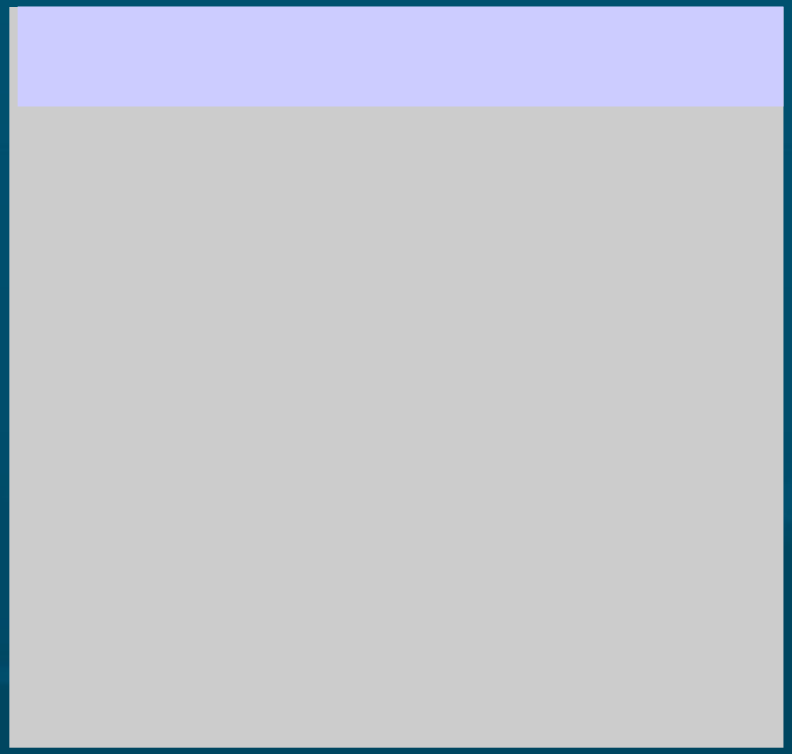
- Basic flow:
  - ...

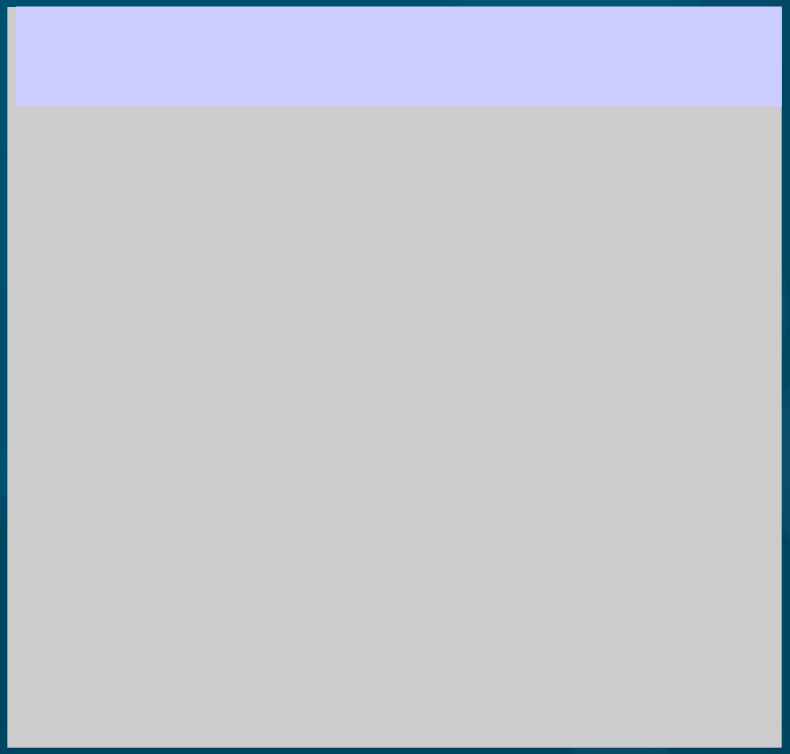
# (1) Use case: alternate flows

- What might not go according to the basic flow?
  - ...

# (1) UI mockup

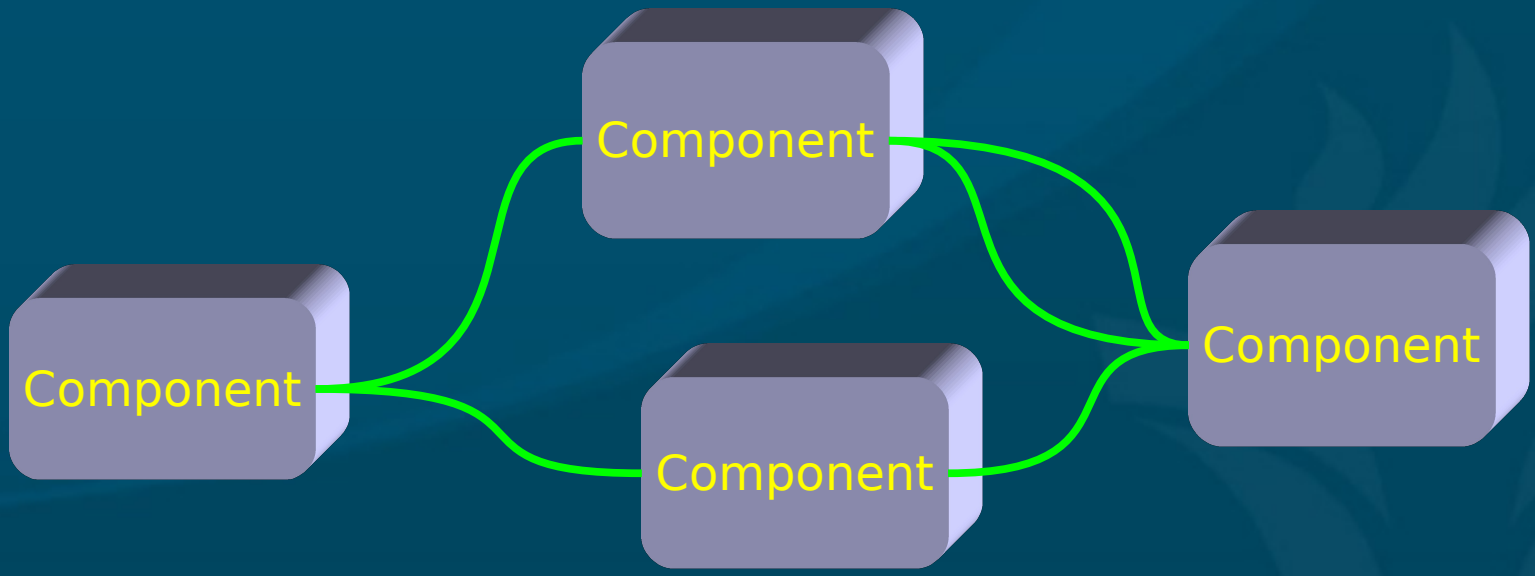
- For each use-case (action), describe/mockup what the user interface will be like:
  - Text Q&A? Windows?  Interactivity?





# (2) Component design

- This is often the hardest part!
- Components need not be classes
- Thinly coupled: describe all interfaces between components



# Component: ....

- Name: ...
- Description: ...
- Interface to (*component*):
  - ...
- Interface to (*component*):
  - ...

# (3) From components to classes

- Each component may need several classes to implement it
- Component: ...
  - Class: ...
    - Attributes: ...
    - Methods: ...
  - Class: ...
    - Attributes: ...
    - Methods: ...