# Announcements, 3/21/2023

Today: Unified Modeling Language (UML) Class Diagrams

Break around 11:15am

## Acknowledgements

Some of these slides are based on the lecture notes from Prof. Alex Kuhn at SUNY Korea, Profs. Robert Kelly and Scott Stoller at Stony Brook University, and Martin Fowler’s UML Distilled book.

## Outline

- Uses of software design specifications
- What is Unified Modeling Language (UML)?
- UML Class Diagrams
- How and when to use class diagrams
- Practice

## Software design specifications

- Software design specifications are a way to specify how a system will work without writing a complete implementation
  - The requirements specify the “What” is to be built, the design specification is the “How”

- Use design specifications to:
  - Get feedback from co-workers and users
  - Get approval
  - Avoid problems in implementation

## What is Unified Modeling Language (UML)?

## Using UML

- UML is a standard for modeling object-oriented software
- Many different diagrams and notations
  - In practice, companies use a subset of UML and modify the form
  - Can simplify diagrams if they do not add clarity and are not needed by your team
- We will focus on the most commonly used and most useful diagrams

## UML uses

- UML can be used as:
  - A **sketch** to communicate some part of the system
  - A **blueprint** to follow when implementing the system
  - As a **programming language**, automatically generating code

## Two common UML diagrams

- **Class Diagrams** - describe the static structure of the system
- **Sequence Diagrams** - describe the dynamic behavior between actors in the system

## UML tools

- Many tools for making UML diagrams – You can choose whichever you prefer.
  - LucidCharts is my recommendation. Sign up with your Stony Brook email and you can request a free Pro upgrade – www.lucidcharts.com

- Some alternatives:
  - Violet
  - Dia
  - UMLet
  - ArgoUML
  - ESSModel
  - Visual Paradigm

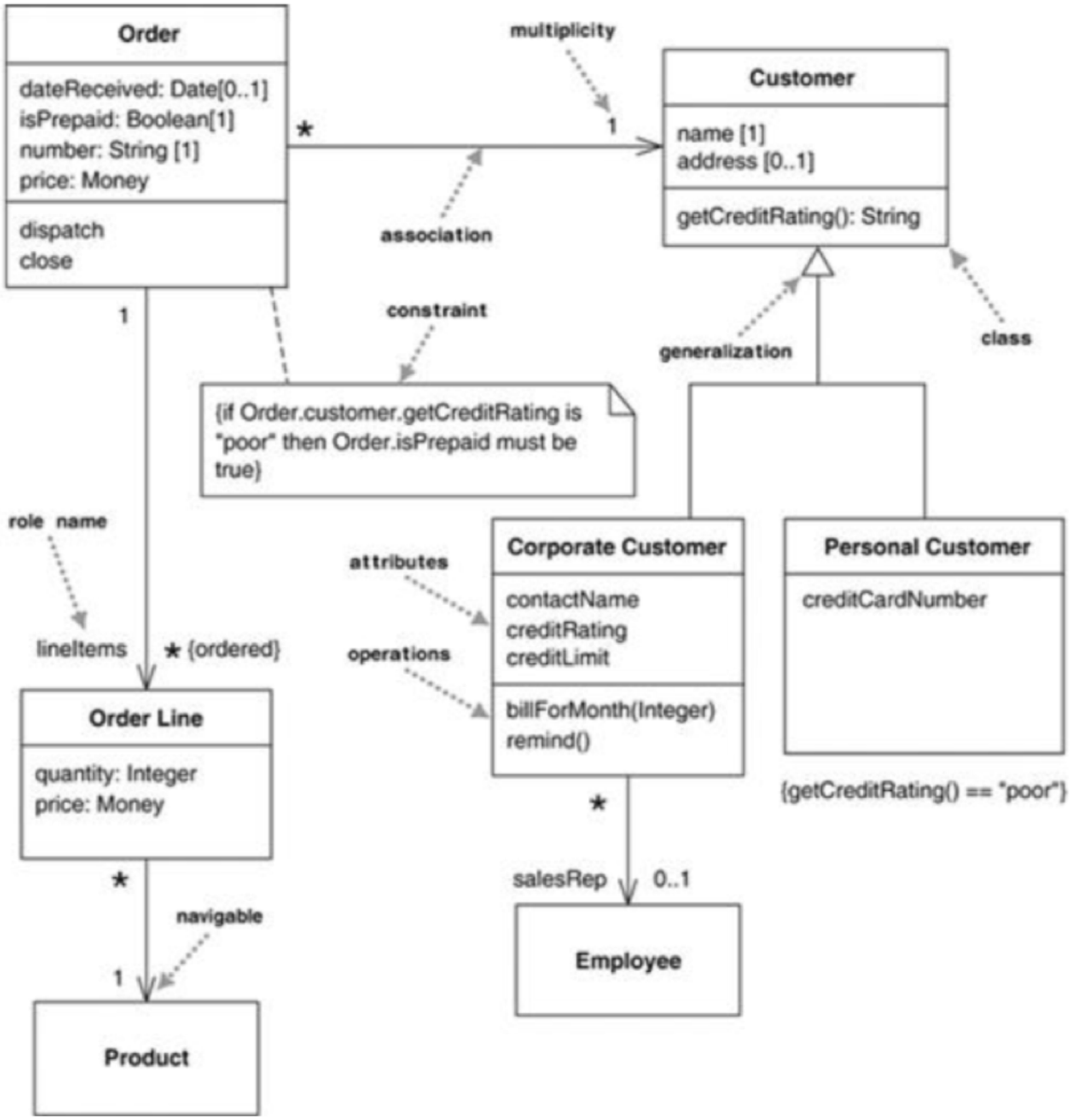
## UML class diagrams

## UML references

- Allen Holub's UML Quick Reference
- Practical UML by Randy Miller
- UML Class Diagram Video Tutorial (LucidCharts)

## Class diagram

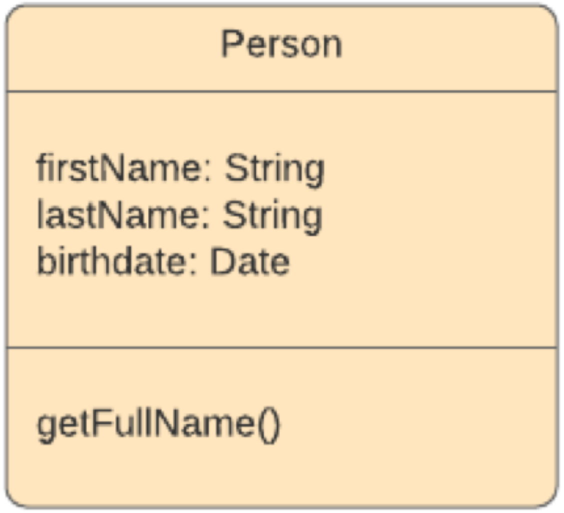
- Goal is to convey information about the static structure of your system.
- Focuses on **classes**, their **attributes** and **methods**, and **relationships** between classes.
- Used by application domain experts and developers to model the domain and design the software
  - Customers and end users are not (usually) interested in Class Diagrams
- Best if you build class diagrams iteratively



From Martin Fowler's UML Distilled

# Class notation

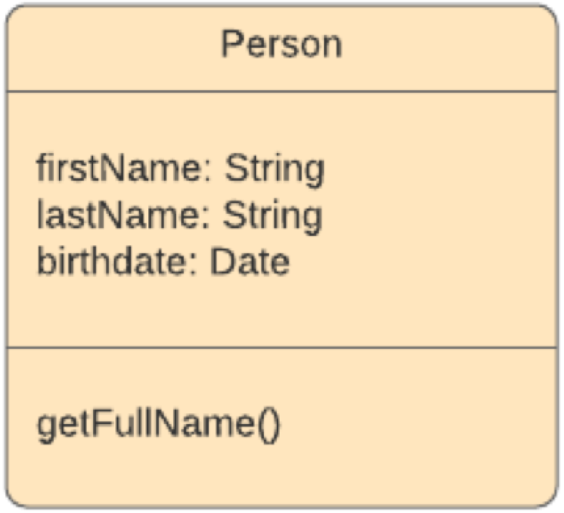
- Consists of:
  - Class Name
  - Attributes
  - Methods (called “Operations” in UML)



# Class notation

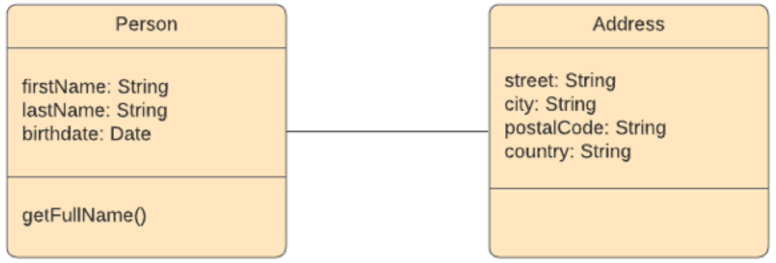
- Details in a class diagram vary based on team, project context, and tool used

- Some things it may include:
  - Parameters
  - Attribute type
  - Getter/setter methods
  - Method return types
  - Visibility



## Association

- Used when:
  - At least one class makes reference to another
  - Relationship is not transient
- Options to include:
  - Name of association, multiplicity, navigability (shown with arrowhead)

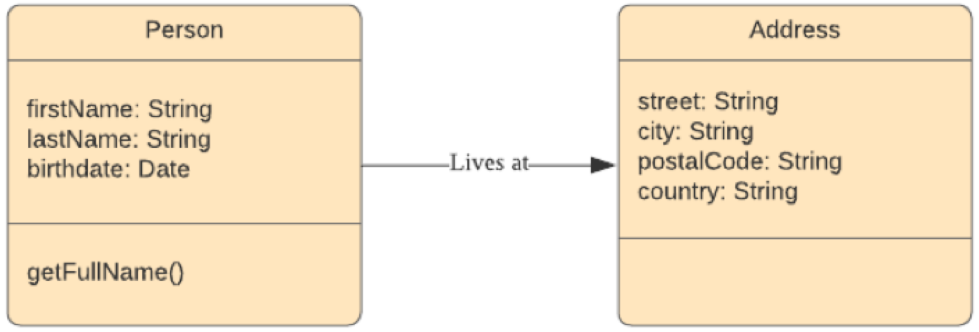


a **Person** lives at an **Address**

# Association arrowhead

- This is optional and usually means the class at the tail has an attribute of the type of the class pointed to by the arrow.

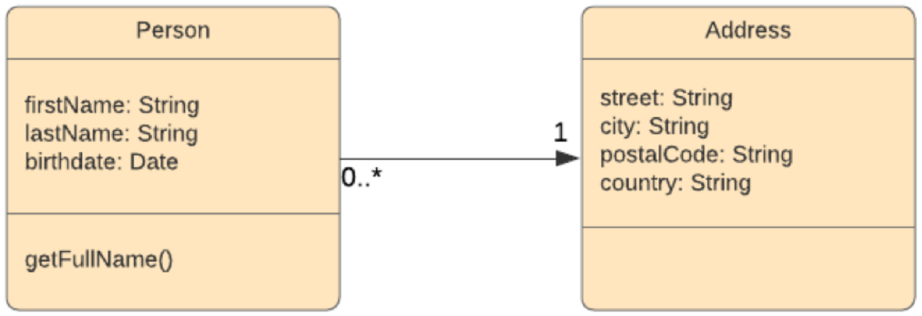
- For example:
  - Person has an attribute for their Address



# Multiplicity

- Multiplicity is used in associations to indicate the number of instances.

- Examples:
  - **0..1** (no instances or 1 instance)
  - ***** (zero or more)
  - **1** (exactly one)
  - **2..*** (2 or more)
  - **3..7** (between 3 and 7, inclusive)

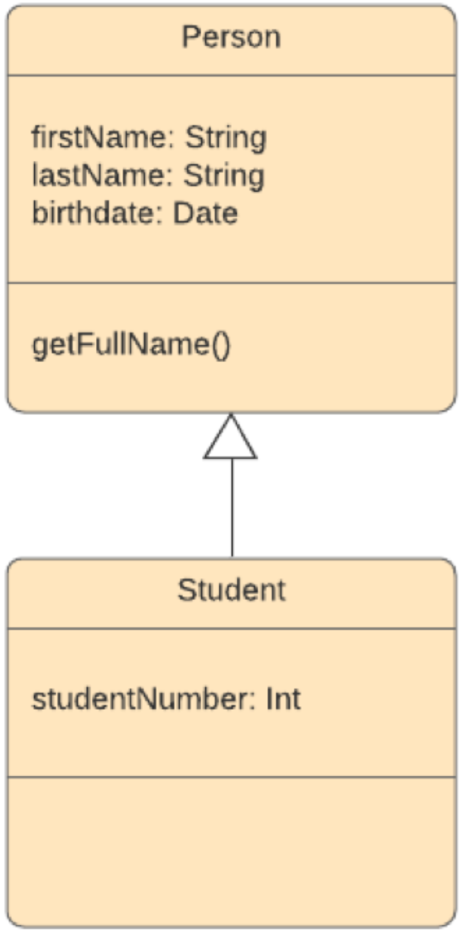


A Person lives at 1 address.
Each Address has 0 or more people.

## Generalization

- Represents an inheritance relationship (is-a)
  - Represented by an unfilled triangle.
  - e.g., Student is a Person (inherits from Person)

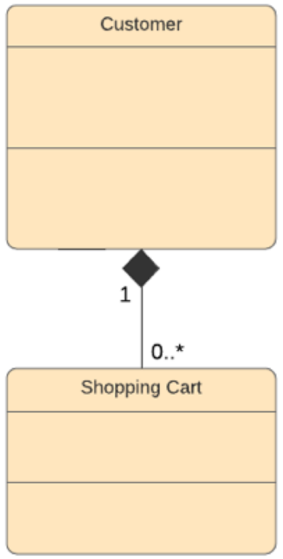
- Do not display parent's attributes and methods in subclass



# Association types

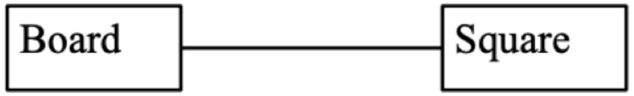
- Composition
  - When an object is composed of the other objects.
  - The parts will be removed with the whole
  - Shown as a filled in diamond

- For example: the Shopping Cart is part of a Customer and will be deleted when that Customer is deleted

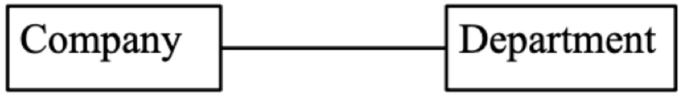


# Which associated types are compositions?

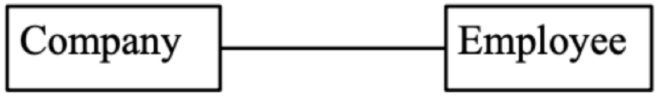
Yes



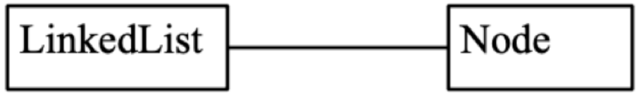
Yes



No



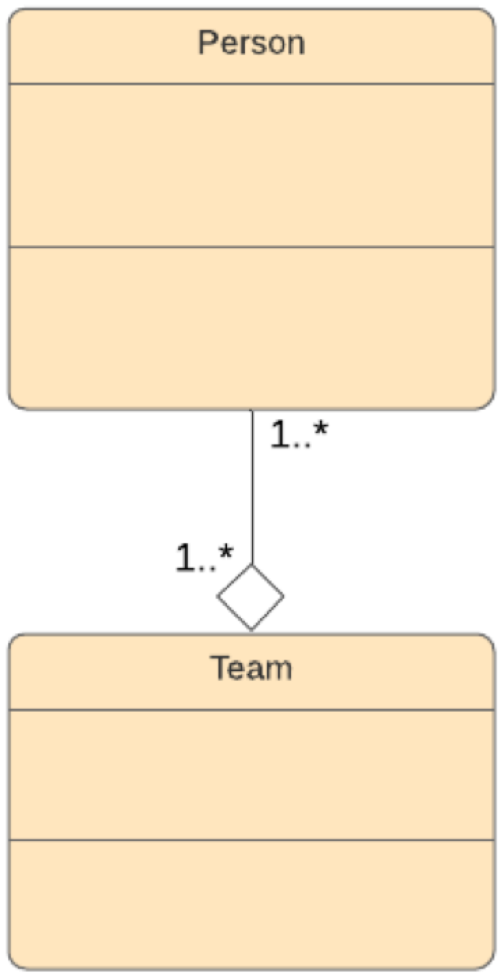
Maybe*



- *It depends on whether nodes may be shared between lists and on the desired semantics of copying.

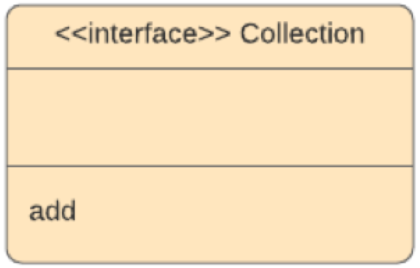
## Association types

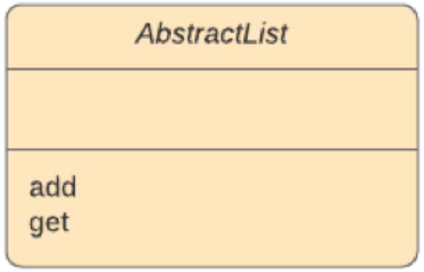
- Aggregation
  - When an object is part of another object
  - Shown as an empty diamond
  - Not very useful to model



# Keywords

- Some additional keywords used inside **<<  >>** or **{ }**
- Examples:
  - **<< interface >>**
    - A class that has no implementation code – all its features are abstract
  - **{ abstract }**
    - A class that cannot be instantiated
    - Alternatively, write the class name in *italics*





# Class attributes

- Can use additional notation to define a class attribute
- The format follows:

  **visibility name: type [multiplicity] = default {property-string}**

- For example:

  **- name: String [1] = "Untitled" {readOnly}**

  **+ dateReceived: Date [0..1]**

- Also, underline <u>static attributes</u>

**visibility options**

+ public
# protected
- private
~ package
/ derived

## Naming conventions

- CamelCase – when you write words together capitalizing the first letter of each word. Two styles:
  - **Lower camel case:** don't capitalize the first letter (e.g., lowerCamelCase)
  - **Upper camel case:** capitalize the first letter (e.g., UpperCamelCase)

## Naming conventions

- Use upper camel case for classes
- Lower camel case for attributes
- Make classes singular
- Make attributes singular, unless they are a collection
- Use nouns for class and attribute names, verbs for method names
- Use application domain terms (not programming terms)

## How do you determine classes?

- Identify from project requirements
- Abbott’s Textual Analysis (also called noun-verb analysis)
  - Nouns are good candidates for classes and attributes
  - Verbs are good candidates for methods
- Apply design knowledge
  - Follow design patterns

## Non-domain classes

- Also will need to determine necessary classes not associated with the domain

- Some examples:
  - Controller objects
  - Authentication objects
  - Web sharing objects (e.g., session)

## Object modeling

- Steps during object modeling
  - Identify classes
  - Determine the attributes
  - Determine the methods
  - Determine the associations between classes
- What happens if you make the wrong abstractions?
  - Iterate and correct your model. Before writing code.
- This will in essence build a stubbed version of your system

## Class diagram uses

- Can be useful for:
  - Understanding entities and their relationships in the system
  - Assisting designing and refactoring
  - Seeing dependencies between objects
- Not useful for understanding:
  - Application control flow
  - Steps to solving use cases
  - Algorithmic operations

## Practice

## Exercise: Class model for flight reservation system

- Draw a class model for a flight reservation system
- Take a top-down approach:
  - Identify all the classes
  - Identify their relationships (mainly associations and generalizations). Show multiplicity on associations
  - Identify their attributes
- To save time, do not show the operations of the classes
- Start with a sketch on paper
- Work individually and email me a photo of your sketch before end of day

## Class model for flight reservation system

- The class model should be able to represent the following information, plus any other information that would be essential in a flight reservation system.
  - Airlines offer flights.
  - Customers make reservations for passengers.
    - Note: The customer pays. The passenger flies.
  - A reservation has a confirmation number and a seat assignment.
  - Flight information:
    - departure date, time and airport
    - arrival date, time and airport
    - stopovers (airport, arrival date & time, departure date & time)
  - An airport serves one or more cities.

## Questions?