Unified modeling language & Object Oriented Design

Amy Liu & Jeremy Wang

A F21 GPI extratation

Why this extratation?

Motivations

Motivation

- Communicate technological ideas regardless of programming language
- Interview questions 👀
- Useful for writing design documents & documentation in general
- Develop good programming habits

Agenda for today

1. Quick intro to the Object-Oriented concept
2. Introduction to UML
3. Practice modeling a system together
4. Go over some pitfalls with examples
5. Future resources

What is OOD?

State & Behavior

5 PRINCIPLES OF OOD

1. Object/Class: A tight coupling or association of data structures with the methods or functions that act on the data.
2. Information hiding: The ability to protect some components of the object from external entities.
3. Inheritance: The ability for a class to extend or override functionality of another class.
4. Interface: The ability to defer the implementation of a method.
5. Polymorphism: The ability to replace an object with its subobjects.

But also, one thing we also thought was important...

Modularity

- Trusting other people's code
- Make sure others can't mess with ours

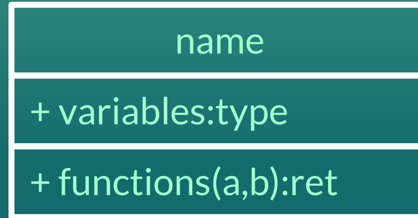
What is UML?

Universal Modeling Language

Notation basics

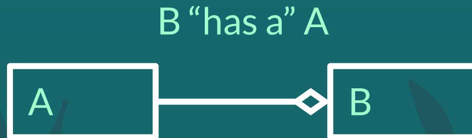
+ : public var/function
# : protected
- : private

Visibility



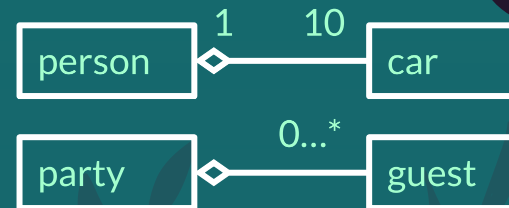
Class

Inheritance / Realization



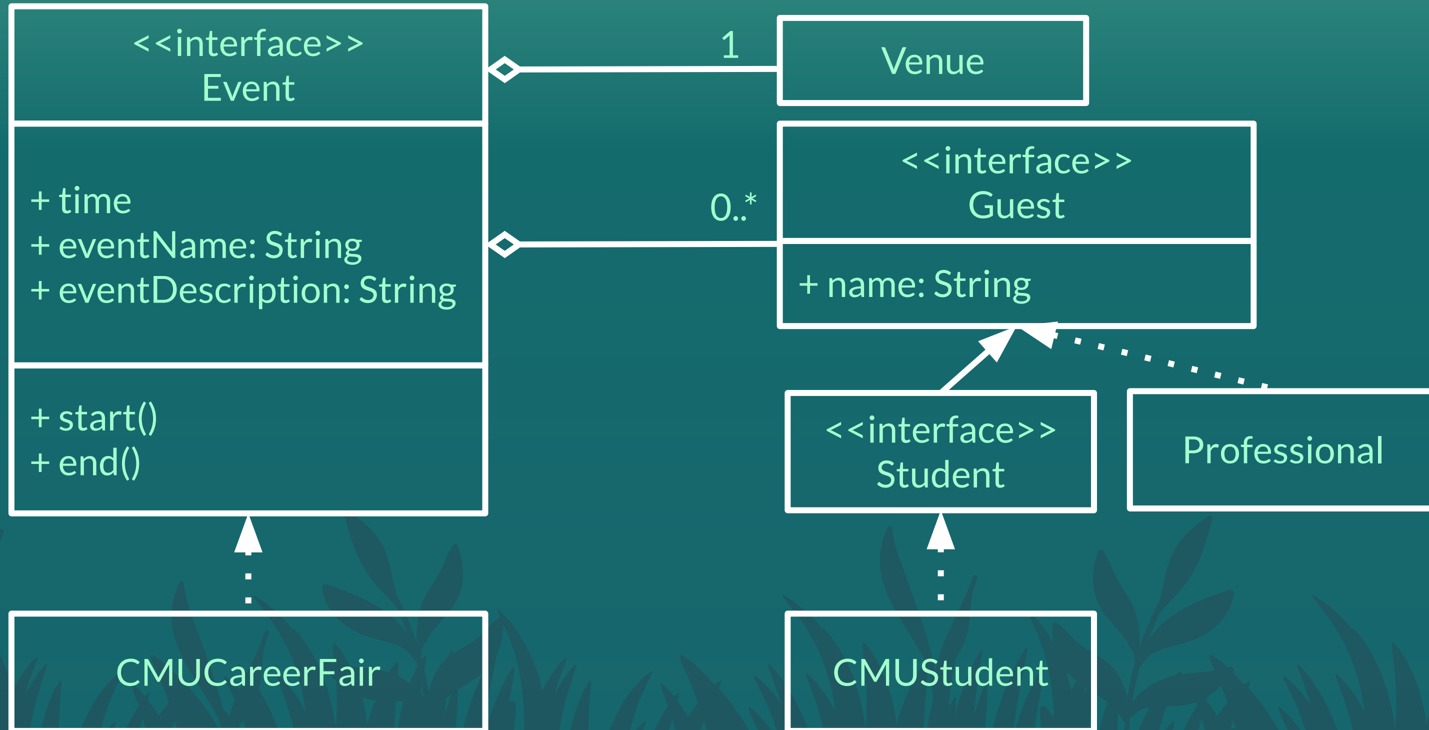
Has-A relationship

Other relationships



Amount

Notation basics - Example



Usage basics

- Capture:
    - Important objects
    - What the states are called
    - Relationships
    - Interactions (between objects in the diagram & outside)
- Omit unimportant details (use … to omit)
- Avoid redundancy (same thing over and over? Just omit it)
- Pseudocode notation like [] is a OK

Modeling

Let’s model a party invite system!

When modeling...

- Usually takes multiple iterations
- Steps:
  a. Identify important nouns... could be classes!
  b. Fill out the classes
  c. Draw in relations

When modeling... (for system design Q's)

- Focus on interactions / how will other classes interact with this class?
    a. get/set functions
    b. who makes decision / performs calculations?
- How will the states(variables) stored?
- How easy is it to update?

Designing a party invitation system for our awesome halloween party ... pt 1

- There are one-time parties, repeated weekly parties, and repeated annual parties; other types might be added in the future.
- Each party has a name and roster of associated users and their roles.
- A user can simultaneously be a host for multiple parties, and be a guest for multiple parties. More party roles may be added in the future.
- Host and guests must have accounts with username and password in the system.

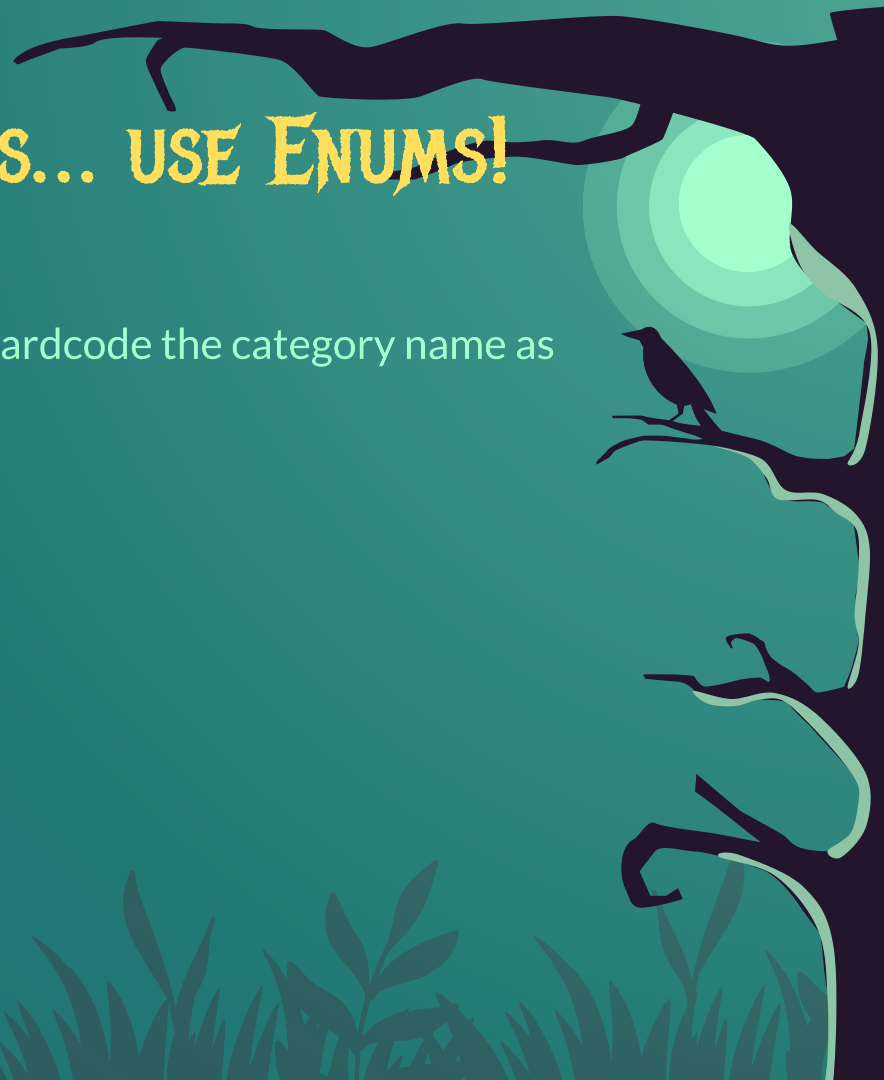
Designing a party invitation system for our awesome halloween party... pt 2

- A host can use the system to notify all party guests of a message, and a guest can use the system to notify the host of her intentions to attend (or not attend).
- The system supports SMS and email, and more notification mechanisms might be added in the future.
- Users can set their preferred notification mechanism(s) in their accounts.

PITFALLS

Don’t do these things...

#1 - Don't abuse Strings... use Enums!



- If there are set categories/types, don’t hardcode the category name as Strings…
  - Typos (needs checks everywhere)
  - Hard to extend
  - Bad to case on (equals)
- Instead, use an Enum class!
- enum Suites{

```
enum Suites{
  HEART,
  DIAMOND,
  SPADE,
  CLUB
}
```

But like... don't abuse Enums either!

- enum Cards{
  HEART1,
  HEART2,
  HEART3,
  ...
  CLUBJ,
  CLUBQ,
  CLUBK
  }

#2 - Don't overstuff a class

- class chessPiece{
    howToMove
    whiteOrBlack

    pieceLocation   //← is this necessary?
    locationOfOtherPieces // ???
    scoreOfTheGame // ??????
    listOfDefeatedPieces // ??????????
  }

Other good to knows...

- Methods should be verbs
    - Eg. If we want to feed a dog, call the method feedDog() instead of dogFood()
    - Less ambiguous, more expected conventions since methods do actions
- Return empty collections instead of null
    - Dereferencing `null` causes NullPointerException
    - Requires the caller to remember do null checks <-- error-prone
- Refer to objects by their interfaces (use List instead of ArrayList)
    - Interfaces show the desired properties of the types we use
    - More flexible, client can use multiple implementations
    - (Unless desired behavior is implementation-specific)

THANKS!

Resources:

- 17-214
- UML and Patterns textbook
- Effective Java textbook

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**