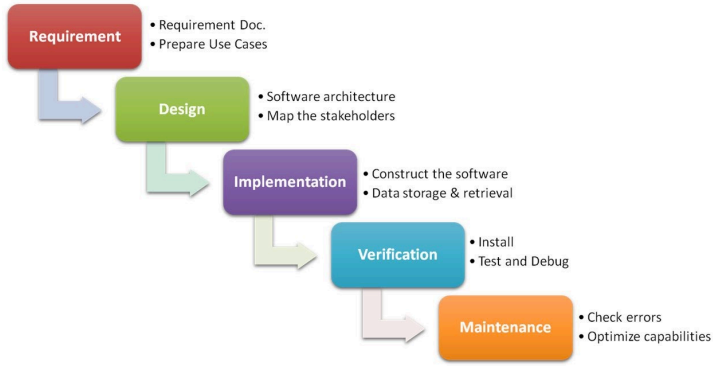
CS 326
Design and UML Class Diagrams

Stephen Freund

Waterfall Software Process

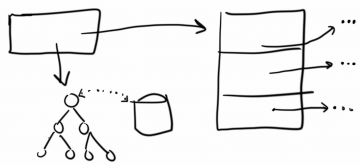


Design Phase

- **Design:** specifying the structure of how a software system will be written and function, without actually writing the implementation
- A transition from "what" the system must do, to "how" the system will do it
  - What classes will we need to implement a system that meets our requirements?
  - What properties and methods will each class have?
  - How will the classes interact with each other?

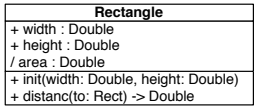
Identify and Design Classes

- Class identification from project spec / requirements / lab handout
  - nouns are potential classes, objects, properties
  - verbs are potential methods or responsibilities of a class
- Need a way to organize and visualize those classes and their relationships



(UML: Unified Modeling Language)

UML Diagrams



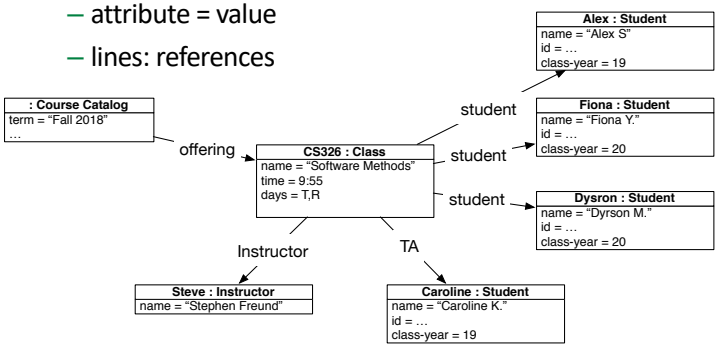
- Diagrams of an OO system
  - PLs are not abstract enough for OO design
  - UML is an open standard; lots of companies use it
- Ways to utilize UML?
  - as a descriptive language: rigid formal syntax (like programming)
  - as a prescriptive language: shaped by usage and convention
    - okay to omit things from UML diagrams if they aren't needed by team/supervisor/instructor

How to Use Diagrams

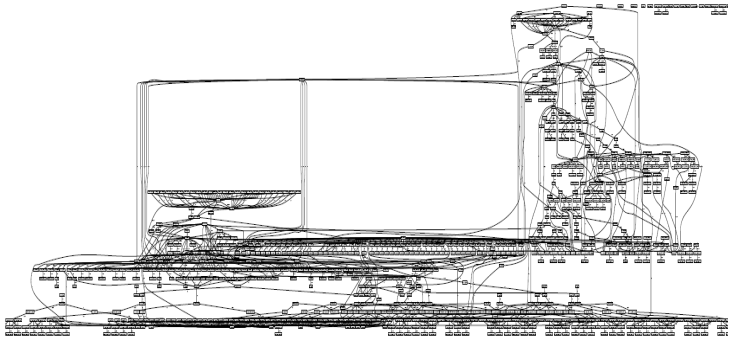
- As a sketch to communicate aspects of system
  - forward design: doing UML before coding
  - backward design: doing UML after coding as documentation
  - often done on whiteboard or paper
  - used to get rough selective ideas
- As a blueprint: a complete design to be implemented
- As a programming language: with right tools, code can be auto-generated from UML diagram

UML Object Diagram

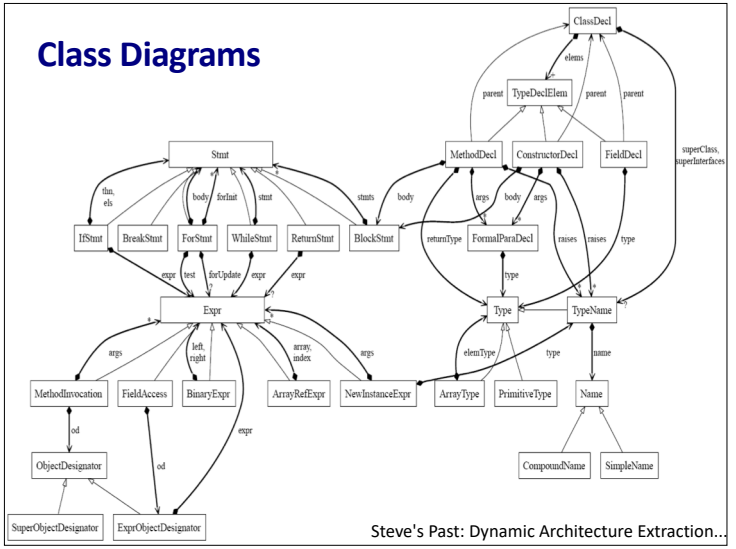
- Show individual objects. (Heap Layout)
  - name : type
  - attribute = value
  - lines: references



- Useful for thinking about organization of single data structure...
- But doesn't scale.



Class Diagrams



Steve's Past: Dynamic Architecture Extraction...

UML Class Diagram

Rectangle
+ width : Double + height : Double / area : Double
+ init(width: Double, height: Double) + distanc(to: Rect) -> Double

- A picture of:
  - the classes in an OO system
  - their properties and methods
  - connections between the classes that interact or inherit from each other
  - "abstraction" of Object Diagram
- Not represented in a UML class diagram:
  - details of how the classes interact with each other
  - algorithmic details; how a particular behavior is implemented

Anatomy of Single Class Box

Rectangle
- width : Double - height : Double / area : Double
+ init(width: Double, height: Double) + distanc(to: Rect) -> Double

- Class name in top of box
  - include protocol, struct
- Attributes
  - should include all properties of the object

struct CGPoint
+ x : Double + y : Double
…

- Operations / methods
  - omit trivial (get/set) methods
  - but don't omit any methods from a protocol!
  - don't include inherited methods

Class Attributes (Properties)

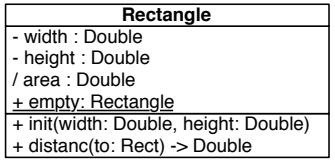
Rectangle
- width : Double - height : Double / area : Double + empty: Rectangle
+ init(width: Double, height: Double) + distanc(to: Rect) -> Double

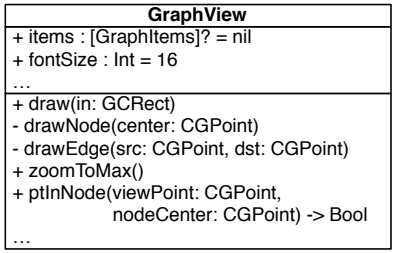
- *visibility name: type = default_value*
- visibility:
  - \+ public
  - # protected
  - - private
  - ~ internal (default)
  - / derived
- static properties:
  - underline

GraphView
+ items : [GraphItems]? = nil + fontSize : Int = 16 …
+ draw(in: GCRect) - drawNode(center: CGPoint) - drawEdge(src: CGPoint, dst: CGPoint) + zoomToMax() + ptInNode(viewPoint: CGPoint, nodeCenter: CGPoint) -> Bool …

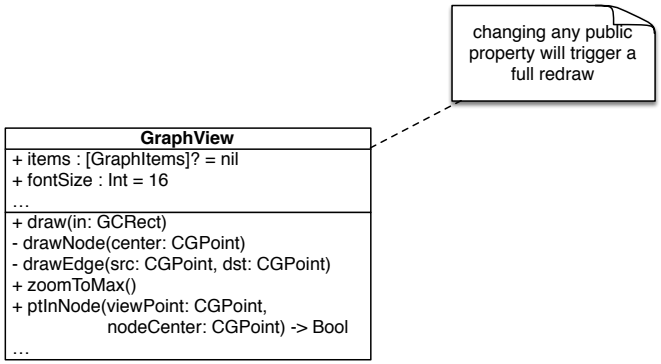
Class Operations (Methods)

- *visibility name(params) -> return_type*
- visibility:
  - \+ public
  - # protected
  - - private
  - ~ internal (default)
- static methods:
  - – underline

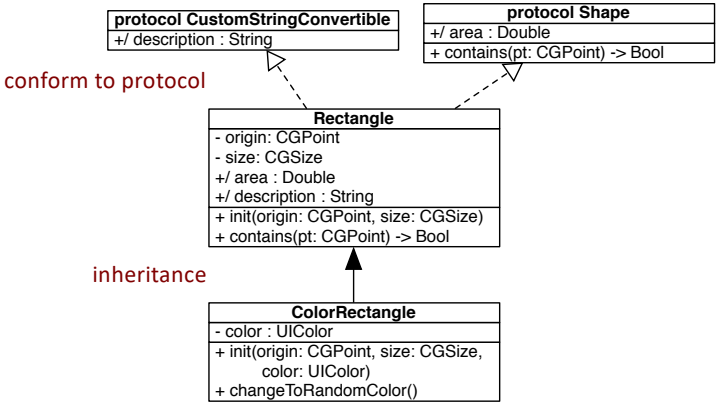




Comments

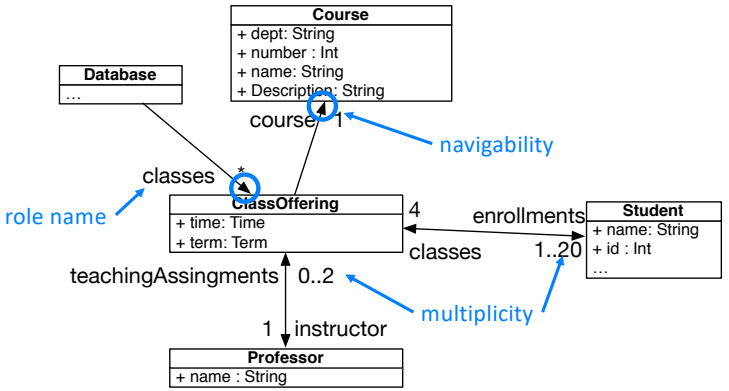


Generalization Relationships



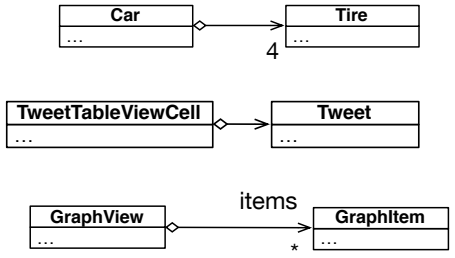
Associations: Usage Relationships

- Multiplicity, Name, Navigability



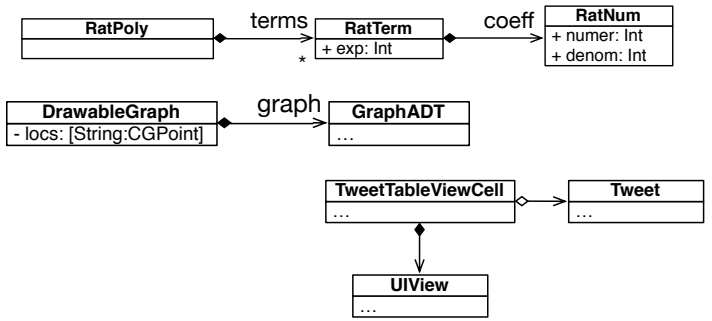
Associations: Usage Relationships

- Aggregation: "is part of"

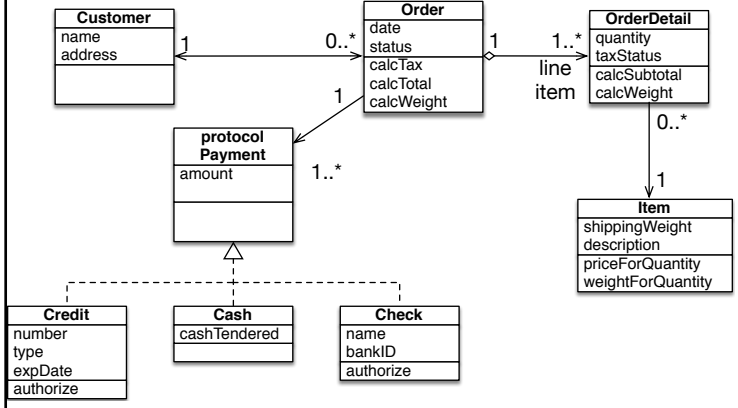


Associations: Usage Relationships

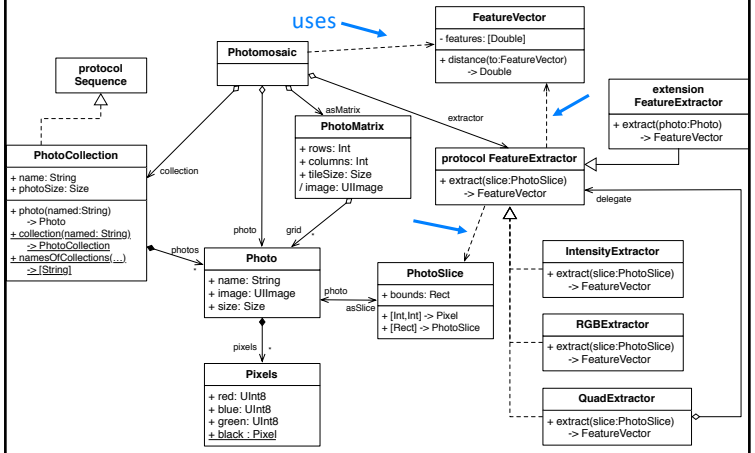
- Aggregation: "is part of"
- Composition: "is made of" (identical life cycles)



Order Tracking System



Dependency Relationships



(note: omitted inits, some other parts...)

Exercise: Texas Hold'em Poker Game

- 2 to 8 human or computer players
- Each player has a name and stack of chips
- Computer players have a difficulty setting: easy, medium, hard
- Summary of each hand:
  - Dealer collects ante from appropriate players, shuffles the deck, and deals each player a hand of 2 cards from the deck.
  - A betting round occurs, followed by dealing 3 shared cards from the deck.
  - As shared cards are dealt, more betting rounds occur, where each player can fold, check, or raise.
  - At the end of a round, if more than one player is remaining, players' hands are compared, and the best hand wins the pot of all chips bet so far.

Exercise: Texas Hold'em Poker Game

- What classes are in this system?
  - What are their responsibilities?
  - Which classes collaborate?
- Draw a class diagram for this system.
- Include relationships between classes (generalization and associational).

Class Diagrams Wrap Up

+ Discover related data and attributes
+ Get a quick picture of the important entities in a system
+ See whether you have too few/many classes
+ See whether the relationships between objects are too complex, too many in number, simple enough, etc.
+ Spot dependencies between one class/object and another

Class Diagrams Wrap Up

- But...
- Can't discover algorithmic (not data-driven) behavior
- Can't see steps for objects to solve a given problem
- Can't understand the app's overall control flow (event-driven? web-based? sequential? etc.)

- Other types of UML Diagrams, but less useful to us right now...