

## Lecture 10

### Object-Oriented Design

CS 600.120 Intermediate Programming

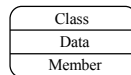
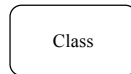
## UML -- Unified Modeling Language

- Experts in OOA/OOD each had their own proposals
  - Booch
  - Rumbaugh
  - Coad-Yurdon
- Got together and chose the best design elements of each

CS 600.120 Intermediate Programming

## Notation

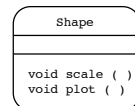
- Representation of classes
  - classes as boxes
  - annotated with data and member functions



CS 600.120 Intermediate Programming

## Classes

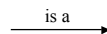
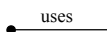
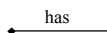
- Shape class from shape example
  - two methods (member functions)
  - no data members



CS 600.120 Intermediate Programming

## Representing Class Relationships

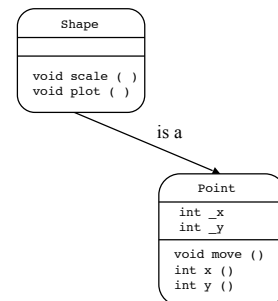
- Aggregation (has)
- Association (uses)
- Inheritances (is a)



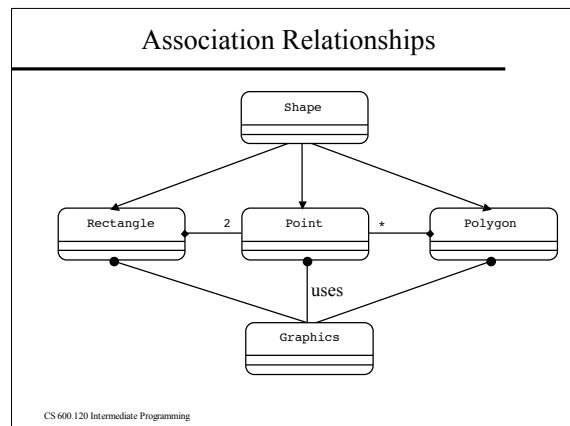
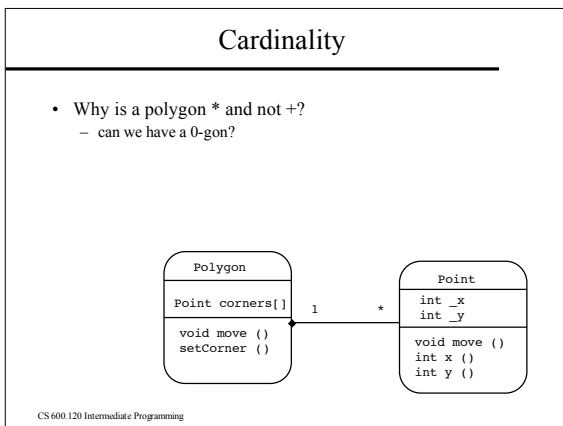
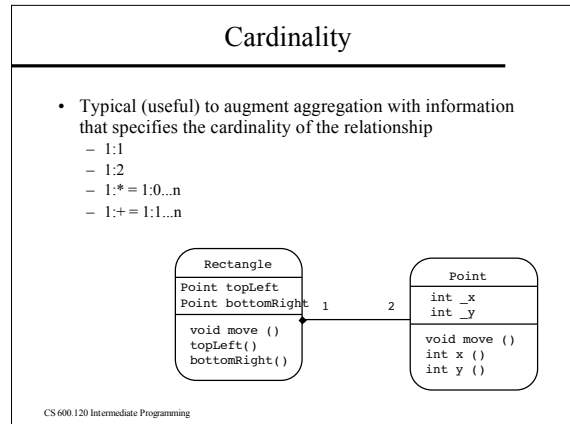
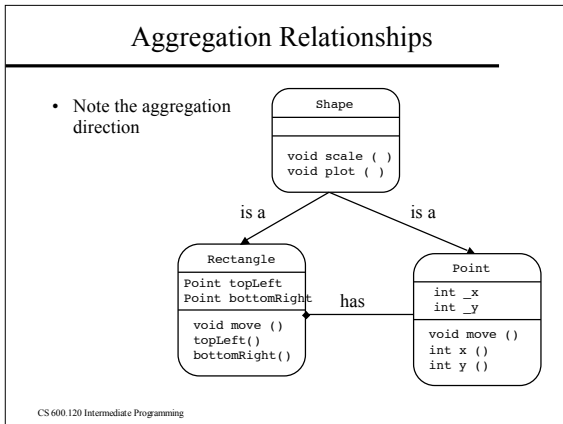
CS 600.120 Intermediate Programming

## Inheritance Relationships

- Represent inheritance
  - point takes on all of shape's properties
  - indicate only the extensions (specialization)
- The "is a" label is redundant, but is frequently used



CS 600.120 Intermediate Programming



### Automation and Tools

---

- Much of the value of UML comes from the ability to translate between a “code view” and a “UML” view
  - modern development products do exactly this
- Write code and it creates a UML diagram
- Create a UML diagram and it does automatic code generation
  - generally creates a skeleton that the implementer completes
- Rational software is the leader in this regard
  - purchased by IBM

CS 600.120 Intermediate Programming

### How do I use UML?

---

- Kind of like the book says to use CRC (index cards)
- But, I do it with Post-Its and a whiteboard
- Create classes on a big post-it
  - annotate with “key” members and functions
- Put post-its on a white board
- Draw relationships on board
- Evolve design
  - move post its
  - erase/change relationships
- Take a picture

CS 600.120 Intermediate Programming

## A Couple More Thoughts

---

- Many people find OOD/UML tedious
  - it often is, but a useful and necessary tool
- Model the important stuff, not everything
  - keeps the task manageable
- Model and remodel
  - design should be and is an iterative process
  - when your code is not working out, it is frequently the design as opposed to some fundamental flaw with the programmer

CS 600.120 Intermediate Programming

## Identifying Classes

---

- An important part of OOD is finding the “right” set of abstractions to model
- There are some guidelines as to what kind of things become classes/objects in programs
- Let’s consider an example:
  - the USCIS (formerly the INS) has deployed a fingerprinting system for foreign visitors and visa holders
  - Let’s spend some time designing the system

CS 600.120 Intermediate Programming

## Goals and Requirements

---

- What is your system going to do?
  - what tasks does it need to perform?
  - what materials will it use in these tasks?
- This is actually object-oriented analysis, but we cannot jump the gun

CS 600.120 Intermediate Programming

## Goals and Requirements

---

CS 600.120 Intermediate Programming

## Class Categories

---

- Objects/classes are derived from both form and function
- We will consider
  - tangible things
  - agents
  - events and transactions
  - users and roles
  - systems
  - containers
  - foundation classes
- All of these things are still derived from nouns

CS 600.120 Intermediate Programming

## Tangible Things

---

- Easiest classes to discover, because they are visible in the problem domain
- The book separates *system interfaces and devices* from tangible things, but these seem pretty tangible to me
- The difference in our example may be:
  - tangible thing -- subject/person, fingerprint
  - system interface or device -- fingerprint reader

CS 600.120 Intermediate Programming

## Tangible Things

---

CS 600.120 Intermediate Programming

## Agents, Events, and Transactions

---

- Frequently one will choose to represent actions in the system as objects, rather than as function calls
  - helps encapsulate their state
- Much like the concept of a gerund
- Agent example:
  - query class to fetch candidate fingerprints
- Events and transactions
  - help to model (keep state) on things that happened
  - Fingerprint read event

CS 600.120 Intermediate Programming

## Users and Roles

---

- Software classes/objects that stand in for the users of the system
- These are a subclass of tangible things
- In our example, we can use to separate:
  - operator role (USCIS employee)
  - subject role (person getting fingerprinted)

CS 600.120 Intermediate Programming

## Systems and Subsystems

---

- Create a system object
  - or many subsystem objects for different tasks
- Useful for construction/destruction, instantiation/tear down of the entire system
- In main(), we want to create a FingerprintSystem class
  - this single object will create all of the subclasses and all of the components
  - its constructor will start a process to initialize the whole system
  - similarly, its destructor will bring down the entire system.

```
int main ()
{
    FingerprintSystem F;
    F.run();
}
```

CS 600.120 Intermediate Programming

## Containers/Iterators

---

- Data manipulation objects
- Containers tend to be things like
  - lists, heaps, tables, etc.
- Iterators help review the contents of containers
  - iterators pass over data items in the container
- In our example:
  - container -- list of candidate matching fingerprints
  - iterator -- iterates over list of fingerprints to do finer grained analysis

CS 600.120 Intermediate Programming