Object-Oriented Programming
Ch 2.2

CS201
Spring 2005
Week 2
Objectives of this Chapter

Students should be able to:
- summarize conceptual underpinnings of OO programming
- Define state, behavior and identity (objects)
- Define terminology of OO programming:
  - message, method, constructor, accessor
  - mutator, destructor, instantiation, reference
  - name or content equivalence
Objectives of this Chapter

Students should be able to:

- Explain what a class and object is
- Explain and use the different kinds of inheritance: specialization, extension, limitation, combination
Gentle reminder

- Schedule of readings in BOC Memo (on the web)
- You should read (quick reading, at least) before class
Bored?

- you have experience
- you have studied classes, inheritance, interfaces, Java (or are a quick study), collections
- want a higher octane experience?
Bored?

- Team project – all semester (3-4 people, good coder, good team player, good communicator, quick study)
- Meet with me (your customer) at least one hour per week and some weeks for 2 hours
- Must take exams (open book)
- Don't have to attend lab or lecture
- Talk to me after class
Objects

- An object is an entity that has:

state, behavior and identity
Object: state

- Properties
  - Integer
  - String
  - MyClass
- Values
  - \{1,8, 9\}
  - \{"hello", "finally!", "whew" \}

Properties don't change, they are static
Values are what change, but they are dictated by the properties

not appropriate to assign a string to an integer, etc.
Objects: behavior

- How it reacts
  - state changes
  - interactions with other objects

- Terminology
  - messages, methods

from book, thermostat will change based on temp falling below certain temp
from our tic-tac-toe, board will react to method call to put a player
Methods: 4 types

- What are they?

accessors, mutators, constructors, destructors
Methods: 4 types

- What are they?
  - accessors
  - mutators
  - constructors
  - destructors

accessors, mutators, constructors, destructors
Objects: Identity

"What's in a name? That which we call a rose By any other word would smell as sweet."
--Romeo and Juliet (II, ii, 1-2)

- How we name something?
- Equality
  - 2 types, what are they?
  - how are they implemented?

The point about names, is that we have objects, but we must be able to name them to get them to do something, etc. We must be able to get at the thing to change its state, interact, etc.

Equality – name equivalence – we saw that last time, content equivalence. bitwise equivalence
Classes

- Group of objects that share common state and behavior
  - examples?
- Difference between classes and objects?
- What does instantiate mean?

integers, Strings, etc.

A class is an abstraction or description
An object is a concrete entity that exists in space and time.

classes define the state, behavior – blueprint, if you will
Objects is an instance of a class -
Abstraction

if (xpos < 0 || xpos > XPOSMAX) ||
    (ypos < 0 || ypos > YPOSMAX) {
    System.err.println("oops, bad move");
} else {
    board[xpos][ypos] = player;
}
Abstraction

```java
if (xpos < 0 || xpos > XPOSMAX) ||
    (ypos < 0 || ypos > YPOSMAX) {
    System.err.println("oops, bad move");
} else {
    board[xpos][ypos]= player;
}
```
private boolean isLegal(int pos, int min, int max) {
    if (pos < min || pos > max) {
        System.err.println("oops, bad move");
        return false;
    } else {
        return true;
    }
}

if (xpos < 0 || xpos > XPOS_MAX) ||
(ypos < 0 || ypos > YPOS_MAX) {
    System.err.println("oops, bad move");
} else {
    board[xpos][ypos] = player;
}

...
Abstraction - refactoring

if (islegal(xpos, 0, XPOSMAX) ||
    islegal(ypos, 0, YPOSMAX) ) {
    System.err.println("oops, bad move");
} else {
    board[xpos][ypos] = player;
}
...

CS201 Introduction to Software
Development Methods
Inheritance

- Often we have things that are similar
  - e.g.
    - ProgrammableThermostat
    - AnalogThermostat
    - Thermostat
- How are they similar?
- How are they the same?

- similar to refactoring
  - similar in behavior and state
  - hold current temp
  - hold desired temp
  - turn heater a/c on and off when desired
Abstraction is the Thermostat

Thermostat is the superclass
Digital, Analog, programmable are all subclasses
Programmable is a subclass of digital (and by transitivity, thermo)

IS-A
Digital IS-A thermostat
Analog IS-A Thermostat

Types – of inheritance

specification – super defines the behavior in the subclass, but not the superclass. Useful for very generalized forms of behavior. Might
specialization – subclass satisfies superclass spec in all relevant aspects. Inherits both behavior spec and implementation – e.g. digital and analog inherit behaviors from thermo, but implement display of temp differently. How they set temp is defined by thermo

Extension – expands functions of a superclass – add new capabilities – programmable, for instance, gets spec and implementation from digital, but adds ability to program
Limitation form of inheritance – display temp but not allow anyone to set it from front panel (might be wireless)

combination allows class to inherit from 2 or more superclasses. This is great in theory, but causes problems. We’ll see what they do in
Story, tasks, prioritize, code, acceptance

- Did it take longer or shorter?
- Was it clear what to do?
- Was it clear what I would accept?
- How did you know you were done?
- Please sign and pledge that you attended class on Wednesday and that the work is your own.
Next week

- prelab
- reading finish chapter 2
  - finding classes
  - UML
- lab
  - code review
  - working with others
  - eclipse and debugging
Our Software Development Big Picture

- People work with people to develop software
- Who are the people
  - customers
  - programmers

customers – the folks who want the software
programmers – the folks doing that
Customers

- Coaches, TAs, Profs
  - stories
  - prioritize the story
  - acceptance testing
  - provide information
Programmers

- Estimate how long something will take
- Write code (duh)
- Test the code with unit testing
Meeting and planning

- Routine
  - Make rough plan (quickly)
  - Refine as you find out
- Artifacts
  - index cards with stories
  - rough plan for next iteration
Let's try it with Tic-Tac-Toe

- **Set up**
  - Due on Friday
  - Work alone
  - Must work (do something)
  - Want fair evaluation
How to start

- Get stories
- Write tasks
- Estimate how long
- Prioritize with customer
  - based on budget (whose budget?)
Stories

- Board game, 3 by 3 board, with players, usually a moderator
- Board
  - 3 by 3
  - States
    - array of 9 positions
    - behaviors
      - place a piece
      - report on certain positions
  - create a class
    - constructors
    - instance variables (fields)
    - array of 9 pos
    - constants (2)
    - methods - accessor, mutator

- Moderator
  - determine who goes first
  - can we play over network
  - gets info about the players
  - registers move with board
  - decides if move is legal
  - decides who wins
  - tally wins and losses

- Players
  - what do they do?
  - place their symbol on the board in attempt to get 3 in a row
  - WHO determines the next move (human or computer)
Stories

- Player
  - symbol x or o
  - decides on next move
- Board
  - keeps track of position of players
  - displays
- Moderator
  - keeps track of whose turn it is
  - gets input from players
  - decides if legal
  - gets board to move player
  - gets board to display
Tasks

- Player
  - create – write a constructor
  - hold symbol – constructor
  - make decision or get input from user
Tasks

- Board
  - positions
  - holds players
  - move player
  - error checking?
Moderator