Project Process

CS 420/520

Don’t plan ahead!

- Well, don’t plan *too far* ahead
  - You don’t yet know enough about your application to plan effectively

    ➡ Plan no more than 2 iterations ahead
- Deliver a working system on each iteration
  - Don’t leave integration until the end

Typical Proposal

1. Design the Core Classes
2. Implement the Core Classes
3. Design “Nice to have” feature
4. Implement “Nice to have” feature
5. Design GUI
6. Implement GUI

Typical Proposal

1. Design the Core Classes
2. Implement the Core Classes
3. Design “Nice to have” feature
4. Implement “Nice to have” feature
5. Design GUI

---

6. Implement GUI

Instead: Step 1

- Design and Implement 1st feature
- Design and implement GUI for 1st feature
- “Deliver” and test 1st feature

Instead: Step 2

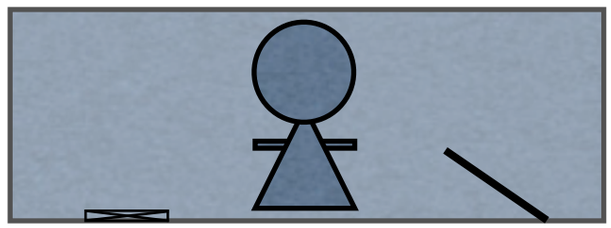
- Design and Implement 2nd feature
- Design and implement GUI for 2nd feature
- “Deliver” and test 1st and 2nd features

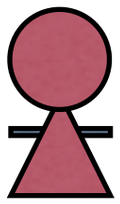
Instead: Step *n*

- Design and Implement n^{th} feature
- Design and implement GUI for n^{th} feature
- “Deliver” and test 1st thru n^{th} features

What's a "Feature"?

- Smaller than you thought!
- Example: home security system
  - One room house
  - add "owner"
  - add a door
  - add a window
  - add "intruder"
  - add a second room
  - add a second window
  - add "guest"





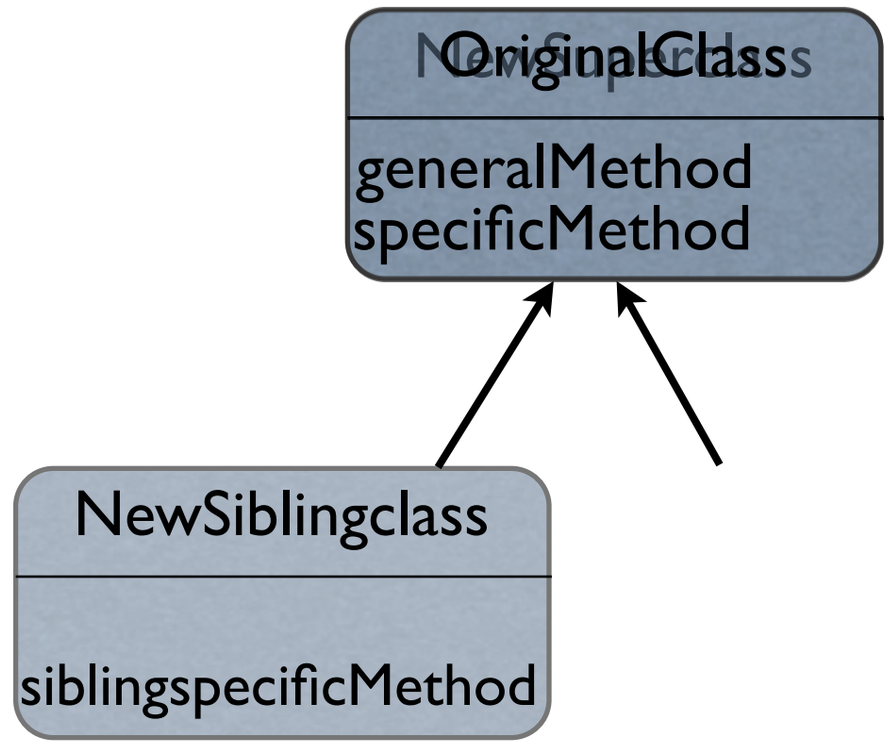
Inheritance is Complex

- Don’t try to design an inheritance hierarchy up front
- Add classes as you need them
- *When* two objects that you thought were the same start to exhibit different behavior, *then*

```
Person ›› openDoor
    self isIntruder ifTrue: [ … ].
    self isResident ifTrue: [ … ].
    …
```

Evolving a class Hierarchy

- add an abstract superclass and make your class a subclass of it
  - refactor class ▷ create superclass
- *add a sibling class*
- *move methods up when both subclases can share it*
- *duplicate and edit a method when subclasses differ*



Evolving a class Hierarchy

- add an abstract superclass and make your class a subclass of it
  - refactor class ▷ create superclass
- *add a sibling class*
- *move methods up when both subclases can share it*
- *duplicate and edit a method when subclasses differ*