Object-oriented design unit:

- OO design goals (last week Wednesday)
- Class extension (last week Friday and this week Monday)
- Big example; refactoring (**Today**)
- (New unit:) Computer memory and pointers (Friday)

Today:

- Recap predator-prey example premise
- Extract method
- Pull-up method; template method
- Factory methods
- Mediator class

A ***refactoring*** *(noun) is a change made to the internal structure of code to make it easier to understand and cheaper to modify without changing its observable behavior.*

To ***refactor*** *(verb) is to restructure code by modifying its design without changing its observable behavior.*

Adapted from Martin Fowler, Refactoring, 1999, pg 53–53.

Refactoring allows us to fix “bad smells in code” like duplicated code, long methods, and lazy classes by extracting methods and interfaces, moving code up in the type hierarchy, etc.

- Version 0
  - Problem: Various `Agent` classes have common code. It is difficult to add new species.
  - Solution: Make the `Agent` interface into an abstract class; pull commonality into the new parent class. *Refactoring: Pull-up Method and Pull-up Instance Variable*

- Version 0
  - Problem: Various `Agent` classes have common code. It is difficult to add new species.
  - Solution: Make the `Agent` interface into an abstract class; pull commonality into the new parent class. *Refactorings: Pull-up Method and Pull-up Instance Variable*
- Version 1
  - Change: `Agent` is now an abstract class.
  - Problem: The `act()` method is redundant, but not fully redundant; it is also long. *Bad smells: Redundant Code and Long Method*
  - Solution: Extract parts of `act()` into helper methods and pull up what's left of `act()`. *Refactorings: Extract Method and Pull-up Method. Design Pattern: Template Method*

- Version 0
  - Problem: Various `Agent` classes have common code. It is difficult to add new species.
  - Solution: Make the `Agent` interface into an abstract class; pull commonality into the new parent class. *Refactorings: Pull-up Method and Pull-up Instance Variable*
- Version 1
  - Change: `Agent` is now an abstract class.
  - Problem: The `act()` method is redundant, but not fully redundant; it is also long. *Bad smells: Redundant Code and Long Method*
  - Solution: Extract parts of `act()` into helper methods and pull up what's left of `act()`. *Refactorings: Extract Method and Pull-up Method. Design Pattern: Template Method*
- Version 2
  - Change: There is a new class, `Animal`. The `act()` method is now a Template.
  - Problem: The `tryToReproduceMethod()` and `scan()` methods are partially redundant; constructors and `instanceof` can't be used polymorphically.
  - Solution: Encapsulate the constructors into methods and add `isPred()` and `isPrey()` methods. *Design Pattern: Factory Method*

- Version 3
  - Change: The child classes of `Animal` now have factory methods that can be called polymorphically.
- Version 4
  - Change: A new species, class `Bear` has been added.
  - Problem: Classes are hard-wired to work with each other. *Design flaw: Tight Coupling*
  - Solution: Move the distinguishing between predator and prey to a “third party.” *Design Pattern: Mediator class*

- Version 3
  - Change: The child classes of `Animal` now have factory methods that can be called polymorphically.
- Version 4
  - Change: A new species, class `Bear` has been added.
  - Problem: Classes are hard-wired to work with each other. *Design flaw: Tight Coupling*
  - Solution: Move the distinguishing between predator and prey to a “third party.” *Design Pattern: Mediator class*
- Version 5
  - Change: A mediator class, `PreyArbitor` has been added.
- Version 6
  - Change: A new species, class `Cougar` has been added.

Coming up:

- **Due Thurs, Mar 5.** *Take quiz on Canvas.*
- **Due Fri, Mar 6.** *Do Project 4, "Text-based adventure game."*