

Building an Ecosystem

- **Objective:** Expand our Creature class into a simulation.
- **Key Concepts:**

- Sub-typing
- Code Reuse
- abstract
- Event Loops
- Multi-Object
- Overloading
- Design Patterns
- Maintaining State
- overriding

- Interfaces
- final
- static
- super
- protected
- composition
- strategy
- flagging

- To Simulate or Not To Simulate
- Scaling Programs
- Package Organization
- Tracing and Debugging Code

Creating Your Project

- Enter your local copy of your git repository on your lab or workstation and create a lab-05 directory.

Project Specifications

- Design a Plant class that inherits from Creature
- The plant class should be able to model the following properties/facts about plants:
 - Some plants are seasonal while others are perennial
 - Some plants have hard, wooden stems while others are soft and green
 - Some plant reproduce via cloning, others use spores, while still others use seeds. Some have more than one way of reproducing.
 - Some plants have pine needles that last year around while others have leaves that fall off in the winter.
 - Some plants run along the ground, other climb walls, some grow really tall, while others stay short and stout.
 - Some plants have flowers and bear fruit. They have phases where they don't have flowers, phases where they do have flowers, and phases where the flower is a fruit.
 - Some plants are very tiny like moss.
 - There are many different types and subtypes of these kinds of plants. Many of the features even mix and match across sub-categories.
- You should generate at least 3 different plant sub classes, as many interfaces as needed, and any necessary component classes that you can use compositionally to model parts of plants. Some properties of plants should be modeled using flags, others string attributes, while others still via composition or an interface. Inheritance should not be your first choice.

- Each particular species of plant should be determined by a String species attribute. The species name should uniquely identify a plant with particular properties even if that species is not a distinct descendant class of Plant. The Plant class itself should have its own variable that keeps track of all of the species that have been invented along with a string that uniquely defines how to build that species.
- The Plant class should have a static method that adds and defines the properties of a species. If a species with a particular name already exists, the Plant class's static method should throw an Exception and not create that kind of creature.
- As much as possible, favor composing objects inside the Plant class to represent unique functionality such as seeding behavior
- The Plant class should override at least one Creature method.
- The Creature class should be abstract.
- At least one function should be overloaded
- There should be at least two different overloaded constructors. The constructor should call the super class's constructor.
- Design a Bird, Mammal, and Fish class that inherits from Creature. Similarly allow for making different species of this class without sub-classing
- Generate a Tile class that holds an ArrayList of Creatures. The Tile class should have water, temperature, and nutrient attributes each of type int. The Tile class should have a takeTurn() method that goes through the list of Creatures and invokes a takeTurn() method for each Creature. The Creatures should be programmed to behave based on what creatures are in the Tile class and based on the values in the attributes of the Tile.
- Generate a World class that contains a 2-dimensional collection of tiles. The World class should have a takeTurn() class that iterates through each Tile in the world object and calls their respective takeTurn() method.
- Your program should have an Interface called TurnTaker that every class in the simulation implements. The TurnTaker interface requires the implementation of a method named takeTurn()
- Build a main class that sets initial conditions for a world simulation from values in a JSON file. Your main program should step through the world's takeTurn() method 100 times.