Interfaces: improving Shotgun

- We don't want to use a String to represent a DNA strand/sequence
  - Probably not efficient for merging two strands
  - Strands can't have annotations: names, features,..
  - Doesn't mirror what's done in BioJava library
- We want to use a class to represent a sequence, but we don't want to force client code into using a specific class
  - Doesn't allow for future improvements
  - Doesn't allow for idiosyncratic coding requirements
  - Doesn't adhere to good engineering principles
- What are our options?

What's a class? State and behavior

- What behavior does a Strand have?
  - Construct from …
  - As far as shotgun goes, what are operations on strand?
  - In addition to shotgun, what might we do with strands?
  - List constructors, methods,
  -
- What state does a strand need to accomplish this behavior?
  - Postpone this decision as long as possible
  - Once we decide, how can we allow for future changes? Refactoring?
  - Private state and public behavior

Starting design/implementation

- Start with something simple we can test
  - Develop a plan for adding behavior
  - Add behavior and test incrementally
  - Design and implement tests with outside classes or methods internal to the class we're building
  - Advantages and disadvantages?
- White box/Clear box and black box testing
  - Black box means we can't see inside what we're testing
  - Advantages and disadvantages?
  - Other approaches?

Example: Rectangle

- We want to implement a Rectangle class. Why?
  - Could appear as window on screen, not necessarily
  - We'll want to merge two rectanges, intersect two rectangles
  - How to construct? Alternatives?
- What other behavior do we anticipate?
  - Methods that return primitives
  - Methods that return objects
  - What is a rectangle?
- What state is needed to implement the rectangle?
  - Where do we start with implementation