Patterns as Help in OO Design

- Different algorithms can accomplish the same task, might need different algorithm depending on context
  - Could have different scenarios in one program, or could use as a design decision to facilitate alternative development
    - Stable sort, integer sort, bogosort
    - Swap buttons, move buttons, animate buttons

- Use *Strategy*, aka *Policy* when
  - Related classes differ in behavior, configure other classes with the behavior
  - Use algorithm/method that clients shouldn’t be aware of or care about
  - Move conditional statements into strategy class [refactor]
Collaborations in building strategies

- Context uses a strategy, may pass information to strategy, may pass self to strategy

- Clients might choose strategy, pass to context
  - Or context (Model/View) could choose strategy

- Don’t need AnimatedBejeweled, SimpleBejeweled, CoolSlidingBejeweled
  - Alternative: create a single Bejeweled that uses a moving strategy, configure view/model with the strategy
  - Use in defining architecture, implement simple approach first
Reflection

- In Java a class is introspective, knows about it itself and can make this information available to clients
  - Create a class without knowing what kind of class it is (factory)
  - Get methods/fields of a class and use these in constructing other objects, in making calls, etc.
  - See java.lang.reflect for details

- Reflection opens many opportunities for programs that modify other programs, not available in languages like C++ that don’t support reflection