Aggregation and Composition

* Aggregation is an object that contains other objects
  + Portfolio – contain BankAccount and Stock
* A composition is a special kind of aggregation where the container class “owns” the object
  + Container class has the only reference in the universe to all mutable objects it owns
  + If this doesn’t happen, the outside world can change the object
  + Strategy: Make copies of objects in a composition when they are passed into or out of the class

Mutable vs. immutable

* Simple classes (not aggregates):
  + No public non-final fields
  + No public mutators (i.e. no methods that make changes)
* If the above is satisfied the class is immutable, otherwise it is mutable
* Aggregations:
  + No public non-final fields of primitive or immutable object type
  + No public fields at all of mutable type
  + No public mutators
  + All mutable object fields are defensively copied (separate issue from composition)
* Note: using the keyword “final” with a mutable class does NOT make the contents of that class immutable
  + The value held by e.g. sm in our code example is the REFERENCE to an object – if you make that final, all you mean is that you can’t change the REFERENCE – it is perfectly fine to change the contents at that reference
* Immutable objects and primitives can be handled similarly from a object copying perspective
  + Primitives are stored by value
  + Immutables are stored by reference, but the value at the reference is unchangeable – therefore the reference can substitute for the value

Aggregations using collections

* Collections framework features classes and tools for handling large numbers of objects
  + E.g. ArrayList, HashMap, TreeMap, …
* Collections (e.g. ArrayList) are themselves objects
* But also, they contain objects (they are themselves aggregates)