Review

Composition – object has the only reference to any mutable objects – only the object can change the objects it contains

Aggregation – object only keeps references to external objects – it is okay (and sometimes desirable) if they can be changed outside the object

Mutable vs. immutable in aggregation

* Primitives:
	+ Immutable if
		- There are no mutators
		- There are no public nonfinal fields
	+ Objects:
		- There are no mutators
		- There are no public nonfinal fields of primitives or of immutable objects
		- All references to mutable objects are defensively copied
		- There are no public (final or nonfinal) fields of mutable objects

Aggregation and Composition in Collections

* Generally, objects held in a class are not by themselves, they are in a “collection” e.g. ArrayList
* How do we deal with collections in an aggregation?

Can you use “clone”?

* If a collection implements “cloneable”, then it is legal to use the method “clone()”
* clone() makes a field-by-field copy of the object – this is called a “shallow copy” because it doesn’t respect the kind of relationship the fields have with their container (i.e. aggregate or collection)
* In some cases this causes problems, e.g. in the BankAccount case – we had to copy each BankAccount before storing them in the copied collection
* What we did in the BankAccount example is called a “deep copy” – the copy of each object respects the relationship between the object and container
* As a result, we will not use clone() in this course