Inheritance

Inheritance acts like a kind of composition

e.g. CheckingAccount extends BankAccount – it is as though CheckingAccount contains a BankAccount … within CheckingAccount there exists state belonging to BankAccount, e.g., balance.

However, inheritance has some interesting features that go beyond composition

e.g. the child class inherits the parent’s API, unless the child overrides public methods

Polymorphism: A child can be cast to the parent, but still use child methods

CheckingAccount c …

BankAccount b = (BankAccount)c;

// call deposit in b … it calls the CheckingAccount version of deposit

When casting inherited objects:

* You can cast up to the parent … however, the object is still fundamentally of child type … if you call overridden methods, it will call the child methods. In some sense, a child cast to its parent only has the alias of the parent.
* Can cast back down to the child if it started out as child type … otherwise, it’s a runtime error

getClass() vs instanceof in equals

* getClass() gives the fundamental type of the object – in other words, what it started out as
	+ Whether or not I cast an object to a different type (e.g. CheckingAccount to BankAccount), the object retains the fundamental character and state of whatever it started out as
* instanceof only checks whether an object can be cast, e.g., a instanceof b returns true of a can be cast to b
* If we want to check whether the state of two objects are equal, they must be the same fundamental type, regardless of casting