Inheritance

So far we have seen classes that do single tasks

* Bank Accounts – keep track of balances
* Stocks – keep track of stock prices

We have also seen classes that contain other classes

* Portfolio – lists of Bank Accounts and Stocks
* Aggregation – keep the reference, can change in the outside world
* Composition – object has the only reference in the universe, can’t change except via the object’s API

What happes if you have two classes that do almost the same thing?

e.g.

- BankAccount – keeps track of a balance

- SavingsAccount – does everything BankAccount does, but every transaction has a fee associated

The way we have learned so far, we would have to write these two classes separately, even though they overlap a great deal

Through inheritance, we can have a “parent” class in common (BankAccount) to handle services that both classes need

Then, we could also have a child class (SavingsAccount) that does certain things slightly differently

We have already seen an example of inheritance from “Object”

* All objects in Java are children of Object
* All objects inherit all public features of Object: the methods toString(), hashCode() – this is why you can use these methods without declaring them
* On the other hand, you can declare your own versions and override those given by Object – provide exactly the same method signature as provided in the parent class (Object)
* You can cast any object to its parent type – any object can be cast to Object, however, you are restricted to use the parent API
* You can cast parent type to child type as long as the object was originally of child type

BankAccount b = new BankAccount();

// can cast up

Object o = (Object)b;

// can cast o down to BankAccount

// because it originally started as

// BankAccount

BankAccount c = (BankAccount)o;

// not ok: String s = (String)o;

// not ok: o.getBalance();

String s = “Hello World”;

Object x = (Object)s;

// not ok: BankAccount d = (BankAccount)x;

Class declaration for inheritance

public class B

extends A { … }

* A is the parent
* B is the child
* B inherits all public features of A
* You cannot inherit from more than one parent at the same time
* Public constructors are not inherited from the parent – public fields and methods are
* Private fields are not visible through inheritance – in fact NO private features are available (private constructors, private methods)
* The keyword super accesses parent resources … remember this() accesses the constructor of this object, super() accesses the constructor of the parent object