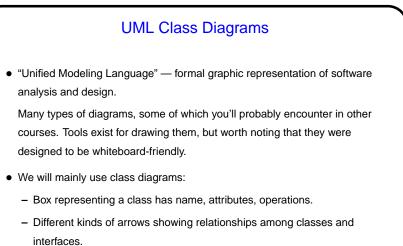
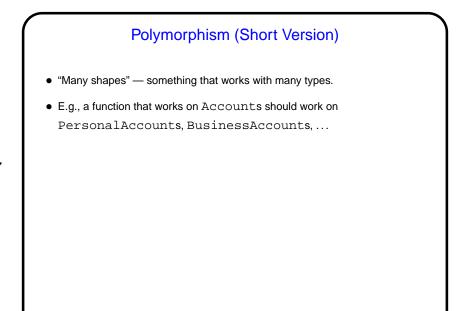


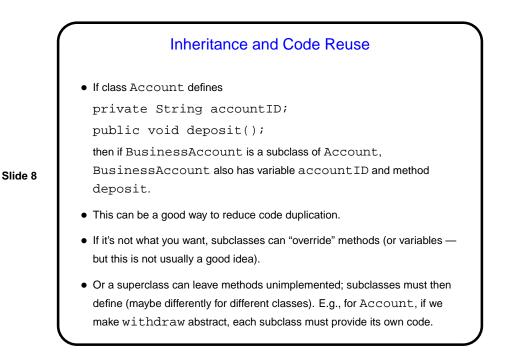
Another Example • Let's start sketching another example — Account class representing bank accounts. • What variables seem useful? what methods?

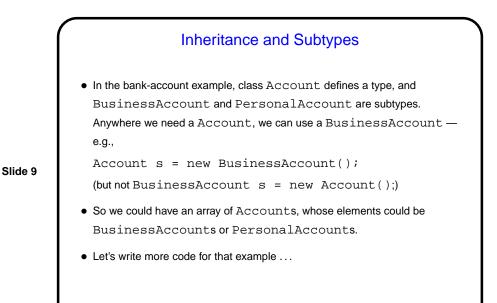


Inheritance (Short Version) Given a class, it can be useful to define specialized versions — "subclasses". A subclass inherits attributes and operations from its superclass (which can in turn have a superclass ...). Subclasses also form "subtypes" — e.g., if PersonalAccount is a subclass of Account, can use a PersonalAccount anywhere we need an Account.

Slide 5







Inheritance Versus Interfaces
What if you don't need/want the superclass to provide any code? you just want it to define a "contract" that all subclasses must meet (i.e., a list of methods they must provide?) then you want a Java *interface*.
In the Account example, we could define a PaysInterest interface with method addMonthlyInterest. This would let us decide for each type of account whether it should pay interest — e.g., BusinessAccount and some subclasses of PersonalAccount.
A class can "implement" as many interfaces as you like. (This helps if you want a class to inherit from multiple classes — Java, unlike some languages (e.g., C++), doesn't allow that because of possible confusion/ambiguity, but you can fake it by implementing multiple interfaces.)

Interfaces and Types Interfaces also define types. So if BusinessAccount implements interface PaysInterest, we can use a Account anywhere a PaysInterest is required. PaysInterest p = new BusinessAccount(); Slide 11 This is "inclusion polymorphism" — and is what will allow your project code to plug neatly into Dr. Lewis's framework. (The framework is written in terms of interfaces such as Block and Screen; your classes will implement those interfaces.)

