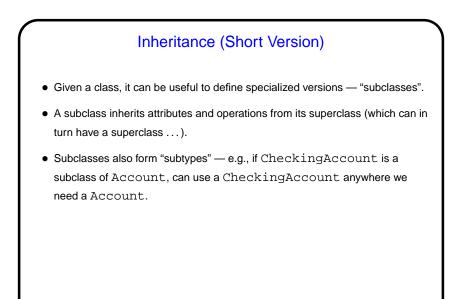


## UML Class Diagrams "Unified Modeling Language" — formal graphic representation of software analysis and design. Many types of diagrams, some of which you'll probably encounter in other courses. Tools exist for drawing them, but worth noting that they were designed to be whiteboard-friendly. We will mainly use class diagrams: Box representing a class has name, attributes, operations. Subclass points to its superclass (represents the path to follow to figure out inheritance).

Slide 2

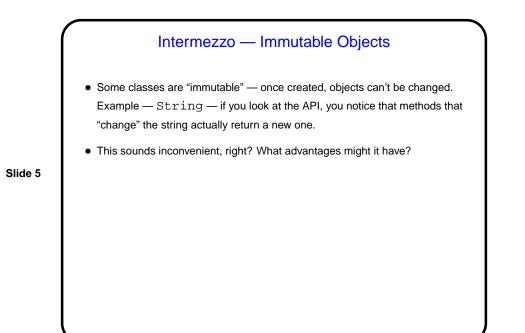


Slide 3

## Polymorphism (Short Version)

- "Many shapes" something that works with many types.
- E.g., a function that works on Accounts should work on CheckingAccounts, SavingsAccounts, ...

Slide 4



Inheritance and Code Reuse
If class Account defines
private double balance;
public double getBalance();
then if SavingsAccount is a subclass of Account,
SavingsAccount also has variable balance and method
getBalance.
This can be a good way to reduce code duplication.
If it's not what you want, subclasses can "override" methods (or variables —
but this is not usually a good idea).
Or a superclass can leave methods unimplemented; subclasses must then
define — for Account, addInterest could be abstract.



 In the "shapes" example, class Account defines a type, and SavingsAccount and CheckingAccount are subtypes. Anywhere we need a Account, we can use a SavingsAccount — e.g., Account s = new SavingsAccount();
 (but not SavingsAccount s = new Account();)

Slide 7

• So we could have an array of Accounts, whose elements could be SavingsAccounts or CheckingAccounts. (More about arrays soon.)



- What if you want a class to inherit from multiple classes? C++ allows this ("multiple inheritance"). To avoid possible confusion/ambiguity, Java doesn't.
- Instead, define "interfaces" classes in which all methods are abstract.

Slide 8

- In Account example, we could define a HasPersonName interface with method getPersonName. Not obviously useful — unless there's another kind of object that could have a person's name but shouldn't be a subclass of Account. (A prospective customer?)
- A class can "implement" as many interfaces as you like.

## Interfaces and Types Interfaces also define types. So if Account implements interface HasPersonName, we can use a Account anywhere a HasPersonName is required. HasPersonName o = new Account(); Slide 9 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.)

A Few Words About Generics
Java library has many useful "container classes" (for vectors, sets, linked lists, etc.) that can hold any kind of object — which is useful, but also sometimes inconvenient (e.g., no way to say "I want a vector of Accounts only").
A solution — Java "generics", new with 1.5/5.0.
Slide 10
Syntax uses angle brackets, e.g., a Vector that can hold only Accounts: Vector<Account> v = new Vector<Account>();
Most of Dr. Lewis's game framework uses this feature — e.g., in Homework 1 you need not a MainFrame object but a MainFrame<BasicBlock, BasicEntity> object.

