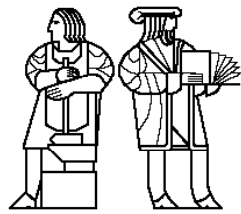


6.170 Lecture 15

Object Models



Michael Ernst
MIT EECS



Formal methods review (incomplete)

Representation invariants

can be assumed on entry to a method
must be maintained by the method

Representation exposure

causes violation of representation invariants

Abstraction function

maps from concrete to abstract value



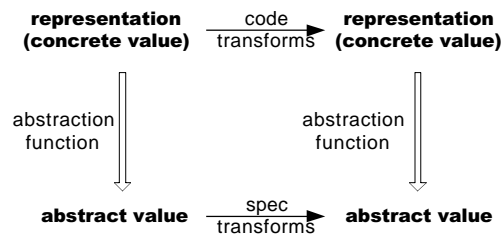
Proving program properties

Use induction

e.g., to prove the representation invariant is maintained

To prove that code satisfies a specification

show that it transforms abstract values as specified



Reasoning about code

Weakest preconditions

Answer to: "If you know the effects, what are the requires?"

"true" means anything goes

"false" means you can't achieve those effects

Loop invariants

Finesse getting to the weakest precondition

Just guess it, then prove it

Formal proofs are sometimes appropriate

Informal reasoning using these techniques is desirable the rest of the time



Object modeling

A notation for expressing properties of a system

code object models
problem object models

Alternatives to 6.170 notation: UML, Booch, etc.

Expressing properties helps you understand them, reason about them, and spot problems earlier



Two models of software development

Waterfall model:

design, specify, code, test, deploy
complete each task before moving on to the next

Spiral or feedback model:

as you perform tasks, reevaluate previous decisions
start with a rough design and refine it

High-level design

Detailed design

Specifications

Pseudocode

Code

Test in parallel



Different models are appropriate in different situations

The earlier you discover errors, the easier it is to fix them



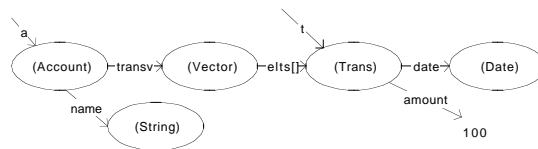
Bank account example revisited

```
class Trans {  
    int amount;  
    Date date;  
    Trans(int amount) { ... }  
    ...  
}  
class Account {  
    String name;  
    Vector transv = new Vector();  
    int balance = 0;  
    Account(String name) { ... }  
    ...  
}
```



Snapshot of one execution

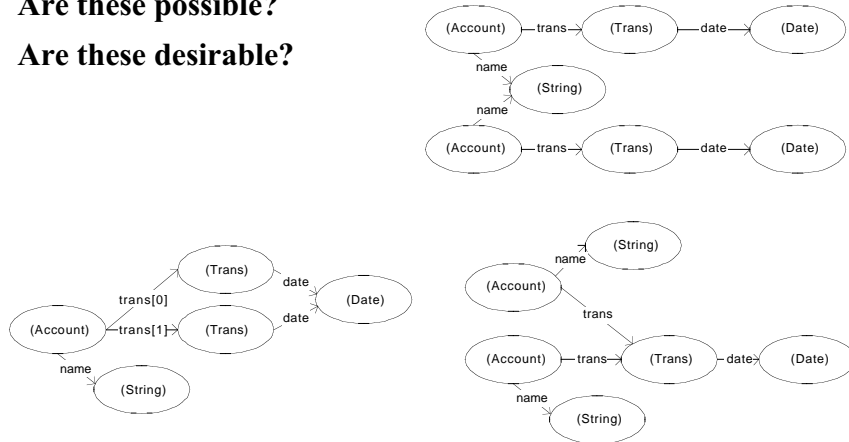
```
Account a = new Account("Rockefeller");  
Trans t = new Trans(100);
```





Other states of the system

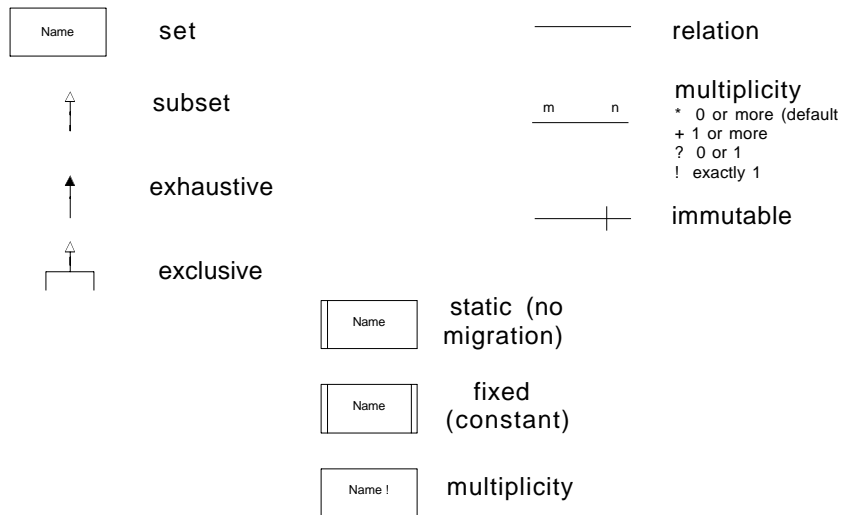
Are these possible?
Are these desirable?

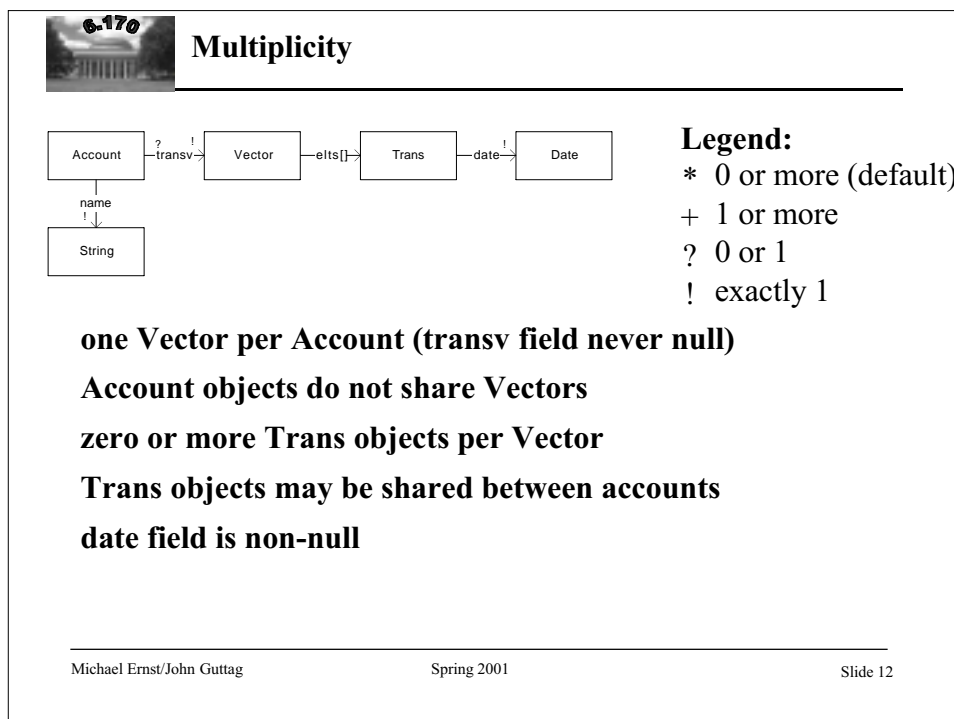
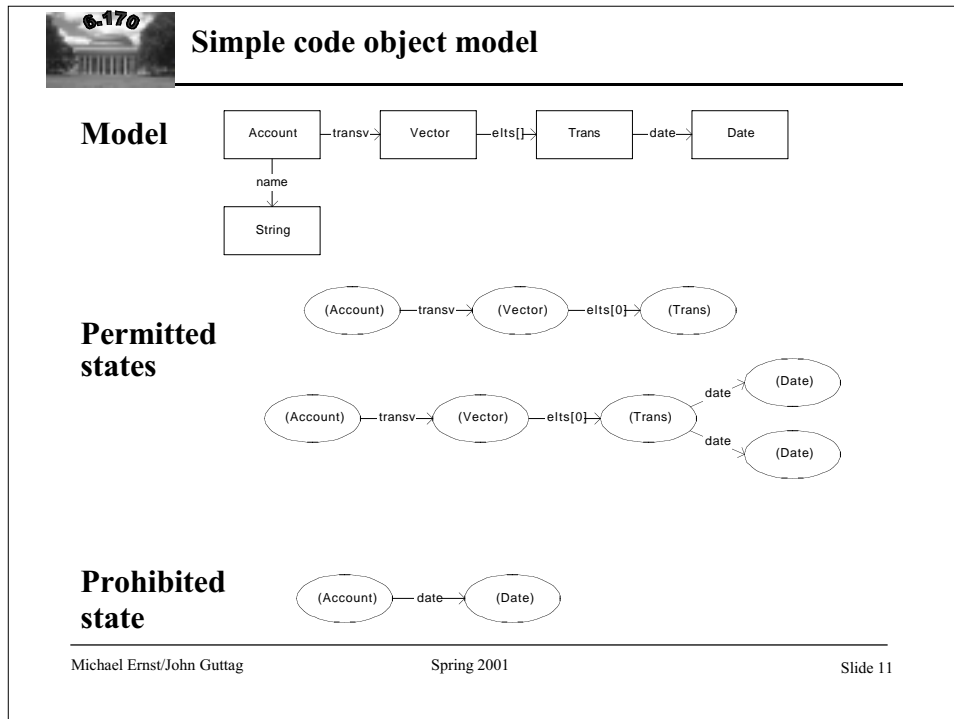


An object model represents a collection of states (intended use)



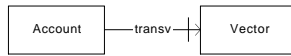
Object model glossary



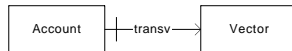




Mutability



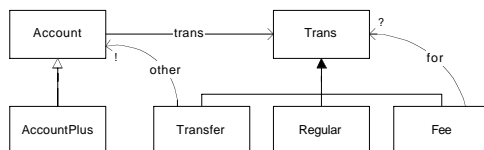
**Vector is set on creation and may not change
(Its contents may change)**



**Which Account is associated with a Vector may not change
(If the Account is destroyed, the Vector may not be reused
for another Account)**



Subclassing



**AccountPlus is a type of account
Transfer, Regular, and Fee are disjoint
Transfer, Regular, and Fee are all the varieties of Trans**



Set multiplicity

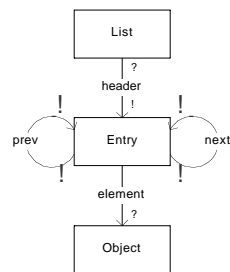
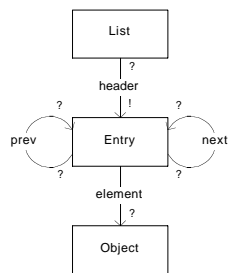


This system is designed for a single bank



A doubly linked list example

What do these object models tell us?





Problem object models

Code object models: objects on the Java heap

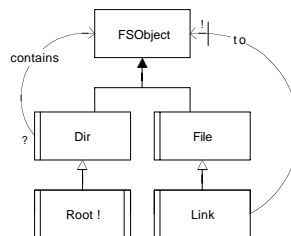
Problem object models: concepts in the real world, the problem domain, or your design (not specific to your implementation)

A problem object model must always have associated text that explains the domains (the sets): define your terms!

There is no single best model; it depends on the task

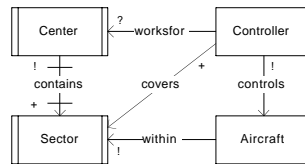


File system

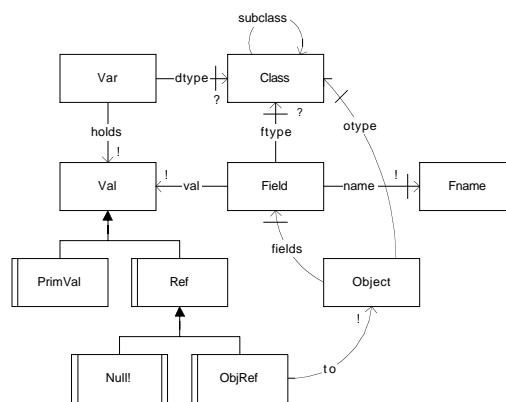




Air traffic control



Java objects



6.170
Spring 2001

