

Nov. 3, 2004
CPTS 355

Object-oriented programming and languages

Initial insight: associate data and operations in one data structure:
came to be called "an object".

Another way to say it: associate state and behavior in one D.S.

Two schools of ~~thought~~ terminology.

behaviors ^{are} response to messages (i.e. send message to object)
behaviors are procedures that are called — method invocation,
methods

the two are the same (and I dislike the 'send a message'
terminology because it conflicts with a similar
but different use in operating systems, ~~and~~ distributed systems,
and agent programming.)

2nd insight: abstraction / interface

Clients/users of an object should not have access to its
state except via a defined interface — a set of operations
~~is~~ often expressed in application level terms rather than
implementation-level terms:

Sets: union, intersection, member
maybe implemented as lists ~~is~~ but the interface doesn't
reveal this (maybe it should?)



interface: what are the legal operations
on the object and what do they do.

the signature of an object is the names and types
of all its operations — doesn't fully specify
the interface because it doesn't say what operations
mean. (semantics)

3rd insight: subtyping: once we begin to discuss interfaces we realize:

If the interface of object A is a subtype of that of object B ~~then~~ provides all the operations that object B does then

A can be used wherever B can. write $A \leq B$

it is clear what this means for signatures
→ same operation names taking operands of same types.

it is not so clear what it means for semantics of objects: just because obj A has ~~foo~~ a method foo(int) and B has a method foo(int) it is not ~~at all clear~~ ^{necessary} that their behavior be related in any way.

To get around ~~the~~ ~~language~~ ^{problem} the language doesn't understand meaning O-O languages usually require subtyping to be explicit (and couple it with inheritance - see below) but the two are actually separate notions.

notice $I_1 \leq I_2 \Rightarrow$ Signatures (ops I_1) \supseteq Signatures (ops I_2)

Interfaces are ~~just~~ another kind of type.

4th insight: Dynamic dispatch

Suppose ~~$x: I_1 = v_1$~~ x is variable of type I_1
 $y: I_2 = v_2$ y " " " " I_2
 ~~$I_1 \leq I_2$~~ $I_1 \leq I_2$

by def of subtyping

if () {
 $y := (v_1: I_1)$ ~~is~~ legal ~~as~~ an object of ~~type~~ a subtype can be used wherever an object of the supertype can be.
} else
 $y := (v_2: I_2)$
}

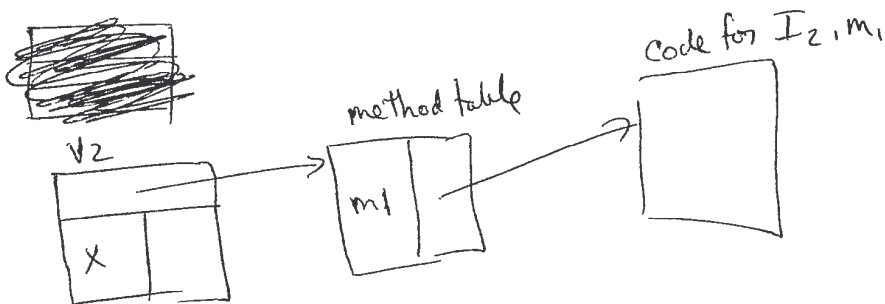
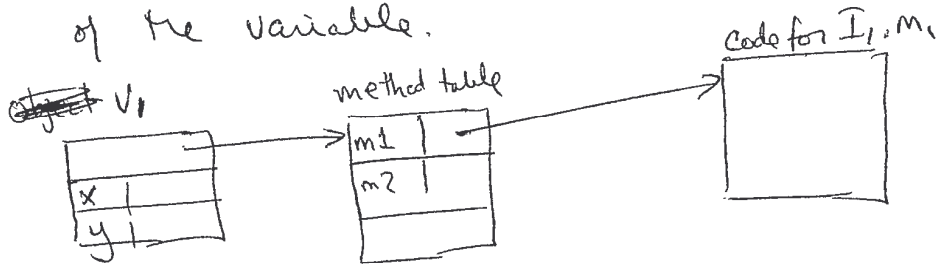
but then if later in the program we

find $y.m_1(...)$;

which ~~is~~ m_1 should be called: ~~which~~ it might be either $v_1.m_1$ or $v_2.m_2$ — so that

is what is done.

Dynamic dispatch: ~~the~~ ^{which to} method called at ~~any~~ point ~~it~~ is determined by the current ~~value~~ ~~of~~ the binding of the variable.



insight 5 : Inheritance

Subtyping requires values of a subtype to implement the operations of its supertype with "similar" meaning.

What is ~~the~~ ^{an easy} ~~best~~ way to accomplish this?

Let the operations of the subtype be the same as the operations of the supertype.

(lots of fine points here: private, protected, public).

Note that ^{well} ~~the~~ inheritance is a property of implementations while subtyping is a property of interfaces). ^{Many} ~~are~~ 0-0

languages intermingle the two in a single mechanism ~~would~~ called ~~inheritance~~ subclassing.

~~A subtype~~

A class defines an interface and ^{possibly an} ~~is~~ implementation of ~~it~~. (parts of) it

A subclass inherits implementations from its superclass, ^{possibly} (widens) extends the interface of its superclass, and possibly over-rides the ~~super-class's~~ ^{super-class's} implementations with ones of its own.

(C++ subclassing also allows ~~narrowing~~ ^{restricting} (narrowing) the interface which means a C++ subclass may not be a subtype of its superclass).

Once a class is defined, objects have that time are called instances of the class.

Warning: p. 284 for "typed languages" read "statically typed languages"
for "untyped languages" read "dynamically typed languages"