Review for final

- Ch 11, 12 — o.o.p
  - data encapsulation (data / operation) — information hiding, interface/implementation

procedures/functions
↓
modules/packages — data/operation encapsulation
↓
class/objects — auto init/finalization, inheritance

accessibility checking
info. hiding

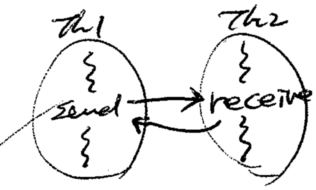
- membership of a class
  - public/private/protected members
  - friend class — can access private members of offering class.

- inheritance
  - derived class, base class — C++ — public/private base class — prog. assign queue/stack
  - super/sub class — Java

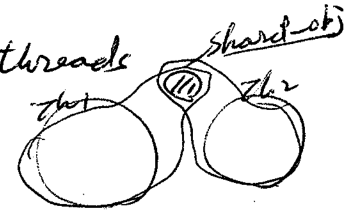
- support for o.o.p.
  - inheritance
  - polymorphism
    - template class
    - overloaded func names

vs
- C++ — copy semantics for arrays/objects
  - $a \leftarrow b$ value is copied
- Java — copy semantics for primitive types
  - reference semantics for arrays/objects
  - $a \leftarrow b$

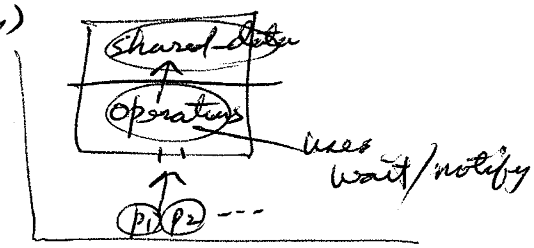
Ch 13. — Concurrent programming



- Communication — msg passing — ex) Ada
  vs.
  shared mem — Java threads

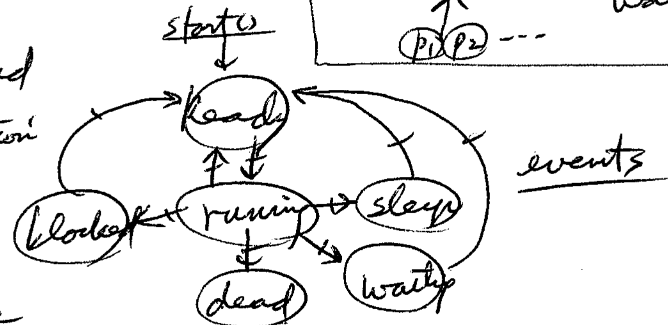


- synchronization
  - ex) Ada — Randezvous
  - Java Thread — monitor/synchronized method (wait/notify)



Java Threads

- life cycle of a Thread
- monitor synchronization



ex) serially reusable resource management problem.
release(), acquire() operations. — synchro. methods in the shared resource class

readers/writers problem., etc.

- How to implement shared data among multiple threads.

— Functional programming

∄ (var, assign st, loop) pure func. paradigm

— lambda calculus

$((\lambda x \cdot x * x)\ 2)$

— β-reduction rule :: $\boxed{((\lambda x \cdot M)\ N) \Rightarrow M[N/x]}$

inner most / outer most reduction

vs.
- LISP — untyped, dynamic scope, prefix, --
- scheme — untyped, static scope, prefix, --
- ML — strongly typed, static scope, infix, --

— Scheme syntax

cons / car / cdr

func. definition / application — ex
(define (square x) (* x x))
(square 5) ⇒ 25

map

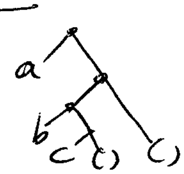
Conditionals / branch

Let

(cond (p1 E1)
 (p2 E2)
 ⋮
 (else En))

— structure of list

ex (a (b c))

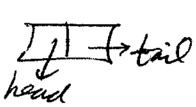


— cons / append — ex (cons '(a b c) '(d)) ⇒ ((abc) d)

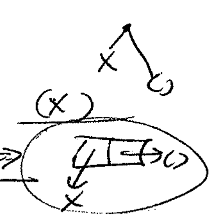
(append '(a b c) '(d)) ⇒ (a b c d)

— storage alloc. for lists

cons —



ex (cons 'x '()) ⇒ (x)



— ML — strongly typed
  static scope
  infix

— rewrite rule for let expression

let (val (V, E_1), E_2) $\Rightarrow$ $E_2[V/E_1]$ — (V in E_2 is replaced by E_1)

— rewrite rule for functions

let (fun (F, para, body), letbody) $\Rightarrow$ Let-body [F(arg) / Func body [para/arg]]

body = Func. body

F(arg) = func call

— nested binding

let val $x_1 = E_1$
  val $x_2 = E_2$ in
  E
end

$\equiv$

let val $x_1 = E_1$ in
  let val $x_2 = E_2$ in
    E
  end
end

— Simultaneous binding

let fun $f_1(x_1) = E_1$
  and $f_2(x_2) = E_2$ in
  E
end

$\Rightarrow$ Scope of X_1 is E_1
  Scope of X_2 is E_2

— Scope of f_1 and f_2 includes E_1, E_2, E.

— rewrite rule for conditional expression

AST: if (B, E_1, E_2) $\Rightarrow$

if (True, E_1, E_2) $\Rightarrow E_1$

if (False, E_1, E_2) $\Rightarrow E_2$

— ML supports
- overloading (+, *, etc.)
- polymorphism
  - (parameterized type
  - polymorphic func.
- no coercion

Ch16 – Logic programming – non-procedural programming

— 2 basic statement forms

headless / headed — Horn clause

ex) fact, goal (query) statement — ex) male(jake), father(bill, jake).

ex) rule statement

consequence :- antecedent_expr.

ex) ancestor(mary, shelley) :- mother(mary, shelley)

— usage of para.

ex) grandparent(X, Z) :- parent(X, Y), parent(Y, Z).

— Goalstatement

— inferencing process (resolution)

matching a goal to a fact in database

vs.
- bottomup resolution
- top-down resolution (prolog) (goal ⇓ facts/rules) — backward chaining
  - ∃ small set of candidate answers

— prolog uses depth-first searching — each subgoal is completely resolved one by one.
(≠ breath-first)

— backtracking — for a compound goal
(∃ multiple subgoals)