Goals

- review monitor model
- study Reader-Writer monitor in Java
- identify limitations of monitors
- study serializers and a serializer solution to the Reader-Writer problem

Limitations of Monitors

The lack of concurrency with a monitor causes:

- **weakening of encapsulation.** In the Readers-Writers problem the protected database is outside of the monitor. There is no guarantee that a Reader or Writer will actually use the monitor. A mal-programmed (or malicious) Reader or Writer could simply access the database directly, ignoring the monitor and its synchronization.

- **deadlock in nested monitor calls.** In cases of complex synchronization deadlock may occur among processes each of which hold the mutual exclusion on some monitors and try to use the service (acquire the mutual exclusion) of a monitor held by one of the other processes.
Serializers

- crowds: a named group of processes executing within a hole

  joincrowd (crowd-name) then actions end

- queue: a place where processes can wait until a condition on the state of crowds is satisfied

  enqueue (queue-name) until condition;
Serializer Solution to the Readers-Writer problem

readerwriter : serializer
var
  readq : queue;
  writeq : queue;
  rcrowd : crowd;
  wcrowd : crowd
  db : database;
procedure read (k: key; var data : datatype);
  begin
    enqueue(readq) until empty(wcrowd);
    joincrowd (rcrowd) then
      data := ...read from db...
    end
  return data;
end read;

procedure write (k: key; data : datatype);
  begin
    enqueue(writeq) until empty(rcrowd) AND empty(wcrowd);
    joincrowd (wcrowd) then
      ...write data to db...
    end
  end write;
end readerwriter;