```
(define front-product<%>
  (interface ()
    print-json
    read-json
    divide-json
    send-receive-list))

(define front-end%
  (class* object%(front-product<%>)

    (super-new)

    (field [list-of-objects '()]
           [lol-of-10 '()])

    (define (print-json)
      ;Prints 'lol-of-10' json objects to STDOUT
      )

    (define (read-json)
      ;Readis json objects from STDIN and appending
      ;them to 'list-of-objects'
      )

    (define (divide-json)
      ;Divides 'list-of-objects' into 'lol-of-10'
      ;a list of lists of 10 objects
      )

    (define (send-receive-list)
      ;Iterates through 'lol-of-10' and sends each list to back-end
      ;Receives sorted lists one by one and replaces the
      ;existing elements in 'lol-of-10'
      )
    )
  )

(define respectful-front-end/c
  (class/c
   [print-json (-> (is-a?/c front-product<%>))]
   [read-json (->i ([this (is-a?/c front-product<%>)])
                   ((not (empty? (send this list-of-objects)))))]
   [divide-json (-> (is-a?/c front-product<%>))]
   [send-receive-list (->i ([this (is-a?/c front-product<%>)])
                           ((not (empty? (send this lol-of-10)))))]))




(define back-product<%>
  (interface ()
    sort-list-of-json))

(define back-end%
  (class* object% (back-product<%>)

    (super-new)

    (define (sort-list-json)
      ;Sorts list of json received from front end interface
      )
    )
  )

(define respectful-back-end/c
  (class/c
   [print-json (->i ([(and (not (empty?)) (lambda list (= (len list) 10)))])
```

```
                ([(and (not (empty?)) (lambda list (= (len list) 10)))])
                )]
  )
)
```