```scheme
;;;;      Layered Programming: An Experimental Implementation!
;;;;                          (not in the book)
```

```
;;;     Layer

(define (layer name dependent-layers compute-default-value)
  (list 'layer name dependent-layers compute-default-value))

(define (layer-name layer)
  (cadr layer))

(define (layer-dependent-layers layer)
  (caddr layer))

(define (layer-compute-default-value layer)
  (cadddr layer))

(define base-layer
  (layer 'base
         '()
         (lambda (base-value)
           base-value)))
```

```
;;;     Layered Data

(define (layered-datum base-value layer-bindings)
  (flatten-layers
   (make-entity
    dispatch-layers
    (cons* 'layered-datum base-value layer-bindings))))

(define (layered-datum-base-value datum)
  (cadr (entity-extra datum)))

(define (layered-datum-layer-bindings datum)
  (cddr (entity-extra datum)))

(define (layered-datum? obj)
  (and (entity? obj)
       (pair? (entity-extra obj))
       (eq? 'layered-datum (car (entity-extra obj)))))
```

```
;; register predicate so that we can print layered data nicely
;; (use MIT/GNU Scheme's register-predicate!)
((access register-predicate! system-global-environment)
 layered-datum? 'layered-datum)

;; make layered data print nicely
(define-print-method layered-datum?
  (standard-print-method
   'layers
   (lambda (datum)
     (cons (layered-datum-base-value datum)
           (map (lambda (layer-binding)
                  (list (layer-name (car layer-binding))
                        (cdr layer-binding)))
                (layered-datum-layer-bindings datum))))))

;; have a slightly nicer syntax
(define-syntax layers
  (syntax-rules ()
    ((_ base-value (layer value) ...)
     (layered-datum base-value (list (cons layer value) ...)))))

;; example
(layers 1 ((layer 1 2 3) 9) ((layer 3 4 5) 10))
;; #[layers 1 (1 9) (3 10)]
```

```
;;;     Flatten Layers and Dispatch Layers

;; flatten-layers ensures that the base value is not a layered datum.
;; If the base value is a layered datum, then merge the layer
;; bindings
(define (flatten-layers datum)
  (let ((base-value (layered-datum-base-value datum)))
    (if (layered-datum? base-value)
        (layered-datum
         (layered-datum-base-value base-value)
         (merge-layer-bindings
          (layered-datum-layer-bindings datum)
          (layered-datum-layer-bindings base-value)))
        datum)))

;; combine layer bindings, keeping those of bindings1 if both bindings
;; have them
;; idea: maybe each layer can have its own way of merging
(define (merge-layer-bindings bindings1 bindings2)
  (define (maybe-add-binding binding bindings)
    (if (bindings-have-layer? bindings (car binding))
        bindings
        (cons binding bindings)))
  (fold maybe-add-binding bindings1 bindings2))
```

```
;; example
(define layer1
  (layer 1 2 3))
(define layer4
  (layer 4 5 6))
(define layer7
  (layer 7 8 9))

(layers (layers 1 (layer1 3) (layer4 5))
        (layer1 6)
        (layer7 9))
;Value: #[layers 1 (4 5) (1 6) (7 9)]
```

```
(define (dispatch-layers proc . args)
  (define (get-arguments-per-layer layer)
    (map (lambda (arg)
           (get-layer-value arg layer))
         args))
  (define (get-layer-value-per-layer layer)
    (apply (get-layer-value proc layer)
           (map get-arguments-per-layer
                (cons layer
                      (layer-dependent-layers layer)))))
  (let ((all-args-layers (apply lset-union
                                eq?
                                (map get-object-layers args)))
        (base-result (apply (layered-datum-base-value proc)
                            args)))
    (let ((layers-to-be-computed
           (lset-intersection eq?
                              all-args-layers
                              (get-object-layers proc))))
      (if (null? layers-to-be-computed)
          base-result
          (layered-datum
           base-result
           (map (lambda (layer)
                  (cons layer
                        (get-layer-value-per-layer layer)))
                layers-to-be-computed))))))
```

```scheme
;;;     Details

(define (default-value layer base-value)
  ((layer-compute-default-value layer) base-value))

(define (get-layer-value datum layer)
  (cond ((layered-datum? datum)
         (let ((binding (assq layer
                              (layered-datum-layer-bindings datum))))
           (if binding
               (cdr binding)
               (default-value layer
                 (layered-datum-base-value datum)))))
        (else
         (default-value layer datum))))
```

```
(define (get-object-layers obj)
  (if (layered-datum? obj)
      (map car (layered-datum-layer-bindings obj))
      '()))

(define (bindings-have-layer? bindings layer)
  (assq layer bindings))

(define (get-base-value obj)
  (if (layered-datum? obj)
      (layered-datum-base-value obj)
      obj))

;; used to help write compute-default-value for layers
(define ((simple-compute-default-value default-value) base-value)
  (if (procedure? base-value)
      (lambda args default-value)
      default-value))
```

```
;; the following is really hacky. todo: make it less hacky.
(define current-environment
  (the-environment))

(define (make-layered! proc-name)
  (let ((old-proc (environment-lookup current-environment proc-name)))
    (if (not (layered-datum? old-proc))
        (environment-define
         current-environment
         proc-name
         (layers (lambda args
                   (apply old-proc
                          (map get-base-value args))))))))

(define (install-handler! layer proc-name handler)
  (make-layered! proc-name)
  (environment-define
   current-environment
   proc-name
   (layers (environment-lookup current-environment proc-name)
           (layer handler))))
```

```
;;; lastly, fix if
;; if needs to access the base value of the object. Note: somewhat
;; dangerous!
(define (if-2 pred then-thunk)
  (if (get-base-value pred)
      (then-thunk)))
(define (if-3 pred then-thunk else-thunk)
  (if (get-base-value pred)
      (then-thunk)
      (else-thunk)))

(define-syntax if
  (syntax-rules ()
    ((_ pred then)
     (if-2 pred (lambda () then)))
    ((_ pred then else)
     (if-3 pred (lambda () then) (lambda () else)))))

;; example
(if (layers #t)
    1
    2)
;; 1
(if (layers #f)
    1
    2)
;; 2
```

```
;;;     Fake Arithmetic

;; fairly hacky. todo: use real arithmetic
(define old* *)
(define (*2 x y) (old* x y))
(define (* . args)
  (fold *2 1 args))

(define old/ /)
(define (1/ x) (old/ x))
(define (/ . args)
  (if (pair? (cdr args))
      (* (car args) (1/ (apply * (cdr args))))
      (1/ (car args))))
```

```
;;;     Support Layer

(define support-layer
  (layer 'support
         (list base-layer)
         (simple-compute-default-value (support-set))))
(install-handler! support-layer '*2 support:*2)
(install-handler! support-layer '1/ support:default-procedure)
(install-handler! support-layer '+ support:default-procedure)
(install-handler! support-layer '- support:default-procedure)
...
```

```
;;;     Unit Layer

(define unit-layer
  (layer 'unit
         '()
         (simple-compute-default-value unit:none)))

;; wrapper to make sdf's unit procedures compatible with ours
(define (unit-layer-wrapper unit-proc)
  (lambda (units)
    (apply unit-proc units)))

(install-handler! unit-layer '*2
                  (unit-layer-wrapper unit:*))
(install-handler! unit-layer '1/
                  (unit-layer-wrapper unit:invert))
(install-handler! unit-layer '+
                  (unit-layer-wrapper unit:simple-binary-operation))
(install-handler! unit-layer '-
                  (unit-layer-wrapper unit:simple-binary-operation))
...
```

```
;;;     Examples

(define (F m1 m2 r)
  (/ (* G m1 m2)
     r r))

(define G
  (layers 6.67408e-11
          (support-layer (support-set 'CODATA-2018))
          (unit-layer (unit 'meter 3 'kilogram -1 'second -2))))

(define M-Earth
  (layers 5.9722e24
          (support-layer (support-set 'Astronomical-Almanac-2016))
          (unit-layer (unit 'kilogram 1))))

(define M-Moon
  (layers 7.342e22
          (support-layer (support-set 'NASA-2006))
          (unit-layer (unit 'kilogram 1))))

(define a-Moon
  (layers 384399e3
          (support-layer (support-set 'Wieczorek-2006))
          (unit-layer (unit 'meter 1))))
```

```
(F M-Earth M-Moon a-Moon)
;; #[layers 1.9805035857209e20
;;          (support (support-set nasa-2006
;;                                codata-2018
;;                                astronomical-almanac-2016
;;                                wieczorek-2006))
;;          (unit (unit kilogram 1 meter 1 second -2))]

;; now make F layered
(define F
  (layers F
          (support-layer
           (lambda (supports bases)
             (apply support-set-union
                    (support-set 'newton)
                    (get-layer-value G support-layer) ; maybe merge?
                    supports)))))

(F M-Earth M-Moon a-Moon)
;; #[layers 1.9805035857209e20
;;          (unit (unit kilogram 1 meter 1 second -2))
;;          (support (support-set wieczorek-2006
;;                                nasa-2006
;;                                astronomical-almanac-2016
;;                                codata-2018
;;                                newton))]
```