Exercise 3.48.

Explain in detail why the deadlock-avoidance method described above, (i.e., the accounts are numbered, and each process attempts to acquire the smaller-numbered account first) avoids deadlock in the exchange problem. Rewrite `serialized-exchange` to incorporate this idea. (You will also need to modify `make-account` so that each account is created with a number, which can be accessed by sending an appropriate message.)

Answer.

By numbering the shared resources and acquiring them in order, all the processes involved in are forced to access them in the same order. This indicates that when multiple processes race on a particular amount of shared resources, the preceded one won't be locked out from the next account it attempts to access. By the completion of its modification on the present account, it will set the account unprotected, permitting the subsequent process to access it. Similarly, all the backward processes will successfully go through these shared resources in order.

For instance, using this strategy the multiple shared accounts $a1$ and $a2$ in exachange problem above are labled with two incresing numbers say, 1 and 2. No matter whose process between Peter's and Paul's enter a serialized procedure protecting $a1$, the other one has to wait for access $a1$ until the preceded one leaves. The same rule applies to $a2$. Hence, this deadlock-avoidance method is competent to avoid deadlock in the exchange problem.

The reimplemented `serialized-exchange` invokes the serialized `exchange` with two accounts as arguments arranging increasingly by their identification numbers.

```
(define (serialized-exchange account1 account2)
  (let ((n1 (account1 'order))
        (n2 (account2 'order))
        (serializer1 (account1 'serializer))
        (serializer2 (account2 'serializer)))
    (if (< n1 n2)
        ((serializer1 (serializer2 exchange))
         account1
         account2)
        ((serializer2 (serializer1 exchange))
         account2
         account1))))
```

`Make-account` has also to be modified, with an extension on a identification number.

```
(define (make-account balance order)
  (define (withdraw amount)
    (if (>= balance amount)
        (begin (set! blance (- balance amount))
               balance)
        "Insufficient funds"))
  (define (deposit amount)
    (set! balance (+ balance amount))
    balance)
  (let ((balance-serializer (make-serializer)))
    (define (dispatch m)
      (cond ((eq? m 'withdraw) withdraw)
            ((eq? m 'deposit) deposit)
            ((eq? m 'balance) balance)
            ((eq? m 'order) order)
            ((eq? m 'serializer) balance-serializer)
            (else
              (error "Unknown request -- MAKE-ACCOUNT" m))))
    dispatch))
```