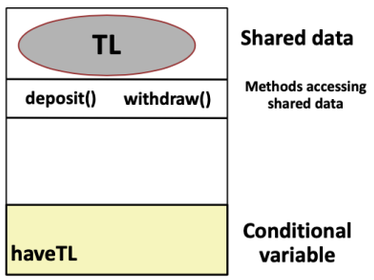
monitors and condition variables

there might be unmatched wait() and signal() in semaphores.

monitor = an object (data + methods), at a time at most one thread may be executing any of its methods (sleeping thread is not in monitor nor queue)

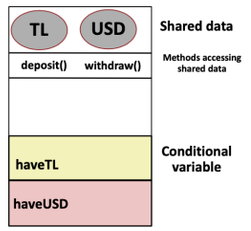
condition variables = queue (not FIFO) of threads associated with a monitor, upon which a thread may wait for some association to become true.

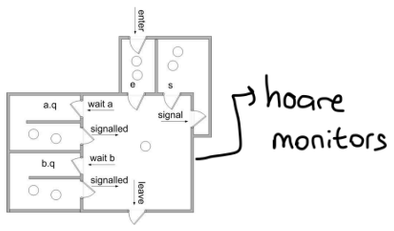



```
monitor Bank{
    int TL = 1000;
    condition haveTL;

    void withdraw(int amount) {
        while (amount > TL)
            wait(haveTL);
        TL -= amount;
    }

    void deposit(int amount) {
        TL += amount;
        notifyAll(haveTL);
    }
}
```

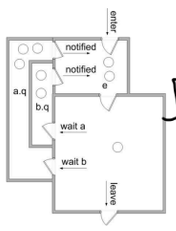
even if one added money, still can be less than amount, check repeatedly.

there can be multiple waiters





→ hoare monitors



→ mesa monitors

monitor notify()

① hoare monitors = run waiting thread immediately, block current notifying thread
- awoken thread will runs nearly most of the time (ex: money can be not enough)

② mesa monitors = put waiting thread to ready queue, current thread is still running
- awoken thread needs to recheck the condition

readers/writers problem =

```
Monitor ReadersNWriters {
  int WaitingWriters, WaitingReaders,NReaders, NWriters; →all initialized to 0
  Condition CanRead, CanWrite; →all initialized to NULL

Void BeginWrite()
 {
       if(NWriters == 1 || NReaders > 0)
       {
            ++WaitingWriters;
           wait(CanWrite);
           --WaitingWriters;
       }
       NWriters = 1;
 }
 Void EndWrite()
 {
        NWriters = 0;
        if(WaitingReaders)
             notify(CanRead);
        else
             notify(CanWrite);
 }
```

don't notify all the readers, one can notify next reader

↳ only one writer at a time

```
Void BeginRead()
 {
      if(NWriters == 1 || WaitingWriters > 0)
      {
           ++WaitingReaders;
           Wait(CanRead);
           --WaitingReaders;
      }
      ++NReaders;
      notify(CanRead);
 }

Void EndRead()
 {
      if(--NReaders == 0)
           notify(CanWrite);
 }
```

→ notify only readers

↳ notify writer only there is no reader

→ reduce number of notify() for efficiency

* in monitor signal() is lost if there is no wating thread
  - ↳ but in semaphores signal() increases the semaphore count, allowing future entry

monitors in java =

```
class Counter
{
     private int count = 0;
     public void synchronized increment()
     {
          int n = count;
          count = n+1;
     }
}
Counter c, d; /* shared objects */
```

Thread A

```
…
c.increment();
…
```

Thread B

```
…
c.increment();
…
```

```
class Buffer {
     private char [] buffer;
     private int count = 0, in = 0, out = 0;
     Buffer(int size) {
          buffer = new char[size];
     }
     public synchronized void insert(char c) {
          while(count == buffer.length) {
               try { wait(); }
               catch (InterruptedException e) { }
               finally { }
          }
          System.out.println("Producing " + c + " ...");
          buffer[in] = c;
          in = (in + 1) % buffer.length; count++;
          notify();
     }
     public synchronized char remove() {
          while (count == 0) {
               try { wait(); }
               catch (InterruptedException e) { }
               finally { }
          }
          char c = buffer[out];
          out = (out + 1) % buffer.length; count--;
          System.out.println("Consuming " + c + " ...");
          notify();
          return c;
     }
}
```