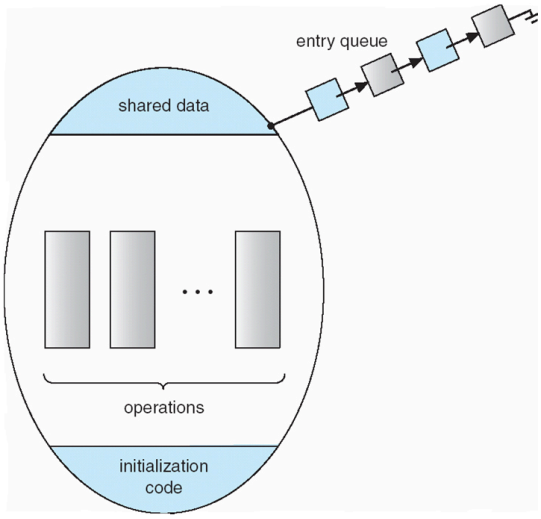
Monitors



Producer/Consumer with Java Monitors

```
class Queue {

   private ...; // queue data

   public synchronized void add(Object item) {
     put item on queue;
     this.notify(); // wake up waiting thread, aka Signal
   }

   public synchronized Object remove() {
     while queue is empty {
       this.wait(); // give up lock and sleep
     }
     remove and return item;
   }

}
```

(Pseudo) C++ Monitors

```
class Queue {

  public:
    add();
    remove();

  private:
    Lock lock;
    ConditionalVariable cv;
    Queue data;

}
```

```
Queue::add(item) {
  lock->acquire();

  put item on queue;
  cv->signal();

  lock->release();
}
```

```
Queue::remove()
  lock->acquire();

  while queue is empty
    cv->wait(lock); // release lock & sleep
  remove item from queue;

  lock->release();
  return item;
}
```

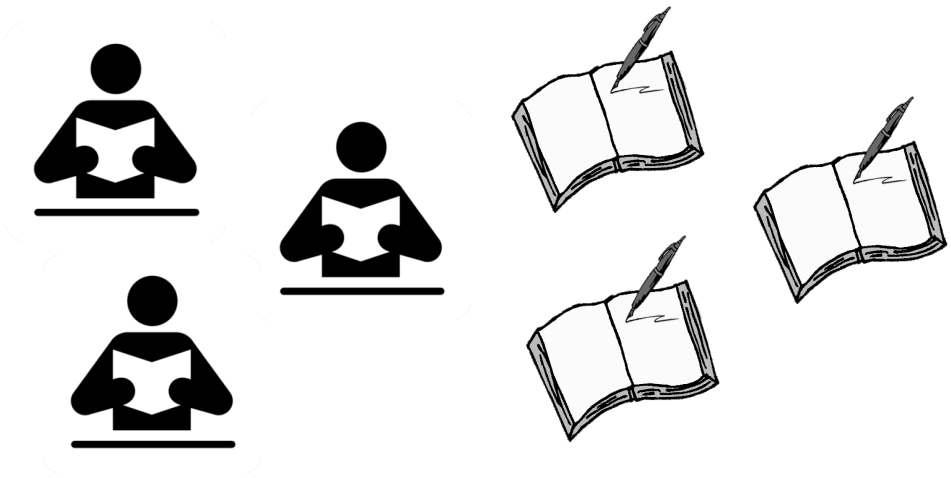

Semaphore using a Montiors

```
Semaphore(initialVal) {

    int counter = initialVal;
    Lock lock;
    ConditionVariable notZero;

}
```

```
Semaphore::Wait() {
    lock.acquire();

    while (counter == 0)
        notZero.wait();
    counter--;

    lock.release();
}
```

```
Semaphore::Signal() {
    lock.acquire();

    counter++;
    notZero.signal();

    lock.release();
}
```

Readers/Writers Problem



Readers/Writers with Semaphores

```
class ReadWrite {
   public:
      void Read();
      void Write();
   private:
      int readers = 0;
      Semaphore mutex = 1;
      Semaphore wrt = 1;
}
```

Readers/Writers with Semaphores

```
class ReadWrite {
   public:
      void Read();
      void Write();
   private:
      int readers = 0;
      Semaphore mutex = 1;
      Semaphore wrt = 1;
}

ReadWrite::Write() {
   wrt.Wait();
   <perform write>
   wrt.Signal();
}
```


Readers/Writers with Monitors

```
class ReadWrite {
   public:
      void Read();
      void Write();
   private:
      int readers = 0;
      Semaphore mutex = 1;
      Semaphore wrt = 1;
}

ReadWrite::Write() {
   wrt.Wait();
   <perform write>
   wrt.Signal();
}
```

```
ReadWrite::Read() {
   mutex.Wait();
   readers++;
   if (readers == 1)
        wrt.Wait();
   mutex.Signal();
   <perform read>
   mutex.Wait();
   readers--;
   if (readers == 0)
        wrt.Signal();
   mutex.Signal();
}
```

Readers/Writers with Monitors

```
private int numReaders = 0;
private int numWriters = 0;

private synchronized void prepareRead() {
  while (numWriters > 0) wait();
  numReaders++;
}

private synchronized void doneRead() {
  numReaders--;
  if (numReaders == 0) notify();
}

public ... doRead() {
  // reads NOT synchronized
  prepareRead();
  <do the reading>
  doneRead();
}
```


Readers/Writers with Monitors

```
private int numReaders = 0;
private int numWriters = 0;

private synchronized void prepareRead() {
  while (numWriters > 0) wait();
  numReaders++;
}

private synchronized void doneRead() {
  numReaders--;
  if (numReaders == 0) notify();
}

public ... doRead() {
  // reads NOT synchronized
  prepareRead();
  <do the reading>
  doneRead();
}
```

```
private void prepareWrite() {
    numWriters++;
    while (numReaders > 0) wait();
}

private void doneWrite() {
    numWriters--;
    notify();
}

public synchronized void doWrite (...) {
    // writes synchronized
    prepareWrite();
    <do the writing>
    doneWrite();
}
```