# MONITORS:
* User defined class
* They have implicit mutex on all methods
* Condition Variables.

\[ \text{wait}(c) \] releases method mutex.
\[ \text{signal}(c) \] releases just next thread from wait.
\[ \text{broadcast}(c) \] releases every thread present in waiting queue.

* Gives synchronization mechanism.

# SEMAPHORE AS A MONITOR :=

Implementation

\[ \text{monitor } S: \]
\[ \text{int } \text{count, condition } C \]
\[ \text{sem_wait} : \]
\[ \text{test} \]
\[ \text{if } \text{count} < 1 \]
\[ \text{wait}(c) \]
\[ \text{count --} \]
\[ \text{sem_signal} : \]
\[ \text{count} + 1 \]
\[ \text{signal}(c) \]

\[ \text{EXAMPLE} \]

\[ \text{sem_wait} \rightarrow \]
\[ \text{sem_wait} \leftarrow 0 \]
\[ \text{sem_signal} \rightarrow 1 \]
\[ \text{sem_signal} \leftarrow 0 \]
EXAMPLE: BOUNDED BUFFER PROBLEM (SIMPLE).

\[ \text{put}(\text{val}, \text{len}) \rightarrow \text{get}(\text{len}) \]

"some_text" \[ \text{get}(5) \rightarrow "some" \]

MONITOR BOUNDED BUFFER PROBLEM

int count
queue buffer
count condition counters.

put: if count = max
    wait(f_writers)
    add item to buffer.
    count ++
    signal(0readers)

get: if count = 0
    wait(0readers)
    remove
    count --
    signal(0writers)

// changing condition to avoid wait
while (max - count < len)
    wait(f_writers)
    add to buffer
    count = count + len
    signal(0readers)
In practice, Java’s `java.util.concurrent.ReentrantLock` is used instead of Monitor.

Java threads:
- `wait()`
- `notify()`, `notifyAll()`

Synchronization:
- `wait()`
- `notify()`, `notifyAll()`

Semaphore:
- Fixed type
- Counting logic
- Release thread
- Signal broadcast

Difference:
- Semaphore
- User-defined logic
- Monitor
- No history

Thread:
- `wait()`, `notify()`, `notifyAll()`

Class:
- Method `p`:
  - `p.wait()` to `p.wait()`
  - `p.notify()` to `p.notify()`

Class `m`:
- Method `condition` to `m.signal()`