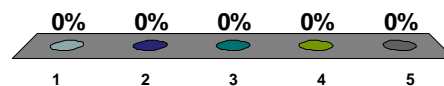


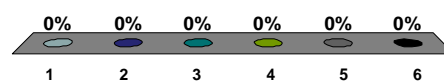
What layer does TFTP implement?

1. Network interface
2. Internet
3. Application
4. Transport
5. Network hardware



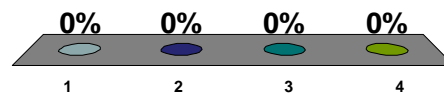
What type of threads are created directly by the programmer in Java?

1. Demon
2. Non-demon
3. Daemon
4. Non-Daemon
5. 1 and 2
6. 3 and 4



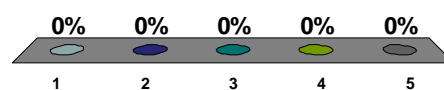
If we have created several Java threads in “main”, when will our program end?

1. When all the non-daemon threads are finished
2. When all the non-daemon threads and any daemon threads that are eligible to run have terminated
3. Never -- it will run forever
4. When the main function returns



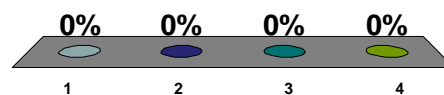
What are the two types of thread synchronization studied in this course?

1. Daemon Thread Synchronization and non-Daemon Thread Synchronization
2. Mutual Exclusion and Condition Synchronization
3. Mutual Synchronization and Condition Exclusion
4. Critical Section and Program State
5. Critical Section and Condition Synchronization



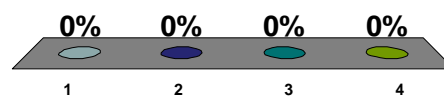
What is the purpose of mutual exclusion?

1. To ensure that threads execute only when the program state is appropriate
2. To ensure that only one thread runs at a time
3. To protect critical sections
4. To force all threads to run simultaneously



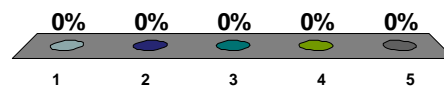
What is the purpose of condition synchronization?

1. To ensure that threads execute only when the program state is appropriate
2. To ensure that only one thread runs at a time
3. To protect critical sections
4. To force all threads to run simultaneously



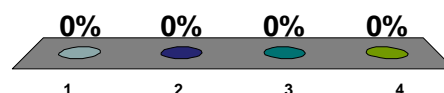
How do we implement mutual exclusion in Java?

1. The wait(), notify() and notifyAll() methods
2. Condition variables
3. The DatagramSocket and DatagramPacket classes
4. The synchronized keyword in a method header
5. By adjusting thread priorities



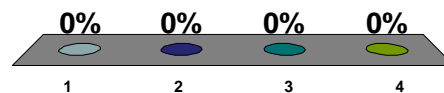
How do we implement condition synchronization in Java?

1. The wait(), notify() and notifyAll() methods
2. The DatagramSocket and DatagramPacket classes
3. The synchronized keyword in a method header
4. By adjusting thread priorities



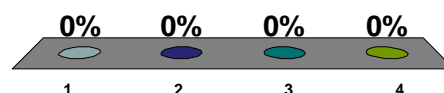
What does it mean if we have a class with 3 methods, all of which are synchronized?

1. At most 3 threads can access the class at the same time
2. At most 3 threads can access each instance of the class at the same time
3. At most one thread can access the class at once
4. At most one thread can access each instance of the class at once



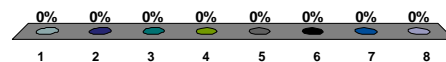
Where can we use the wait() method?

1. In any method
2. In any unsynchronized method
3. In any synchronized method
4. Only in the main program



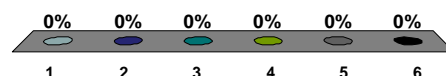
In what context should we always put
“wait()”?

1. Inside a try/catch block
for InterruptedException
2. Inside an if statement
3. Inside a while statement
4. Inside a try/catch block
for SocketException
5. 1 and 2
6. 1 and 3
7. 2 and 4
8. 3 and 4

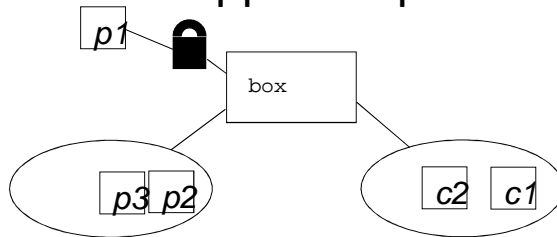


What is the purpose of the wait loop idiom?

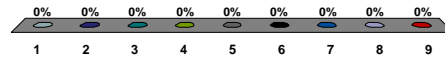
1. Liveness
2. Safety
3. Mutual Exclusion
4. Liveness and
Safety
5. Busy-Waiting
6. Condition
Variables



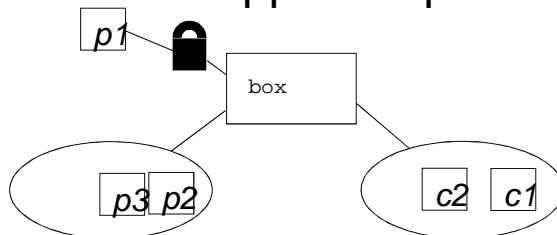
What happens if p1 invokes notifyAll()?



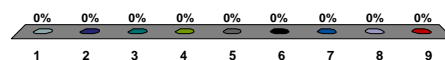
1. p2 and p3 move to the right oval
2. c1 and c2 move to the left oval
3. p2 moves to the right oval
4. c2 moves to the left oval
5. p1 moves to the left oval
6. p1 moves to the right oval
7. One of c2 or c1 moves to the left oval
8. One of p3 or p2 moves to the right oval
9. p1 disappears from the diagram



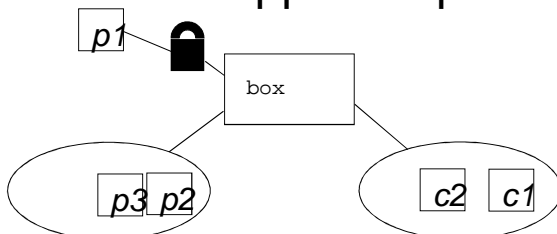
What happens if p1 invokes notify()?



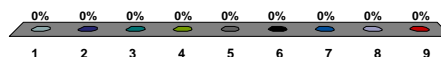
1. p2 and p3 move to the right oval
2. c1 and c2 move to the left oval
3. p2 moves to the right oval
4. c2 moves to the left oval
5. p1 moves to the left oval
6. p1 moves to the right oval
7. One of c2 or c1 moves to the left oval
8. One of p3 or p2 moves to the right oval
9. p1 disappears from the diagram



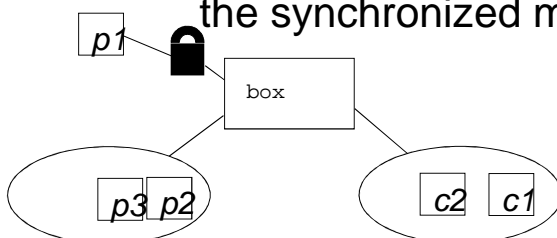
What happens if p1 invokes wait()?



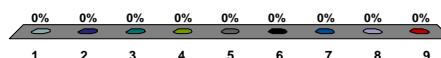
1. p2 and p3 move to the right oval
2. c1 and c2 move to the left oval
3. p2 moves to the right oval
4. c2 moves to the left oval
5. p1 moves to the left oval
6. p1 moves to the right oval
7. One of c2 or c1 moves to the left oval
8. One of p3 or p2 moves to the right oval
9. p1 disappears from the diagram



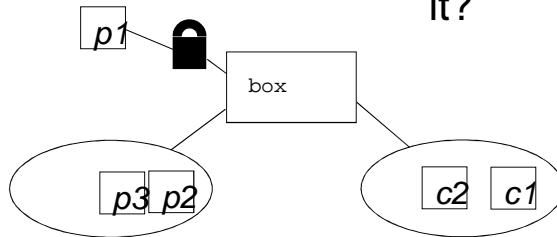
What happens when p1 finishes executing the synchronized method?



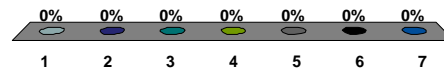
1. p2 and p3 move to the right oval
2. c1 and c2 move to the left oval
3. p2 moves to the right oval
4. c2 moves to the left oval
5. p1 moves to the left oval
6. p1 moves to the right oval
7. One of c2 or c1 moves to the left oval
8. One of p3 or p2 moves to the right oval
9. p1 disappears from the diagram



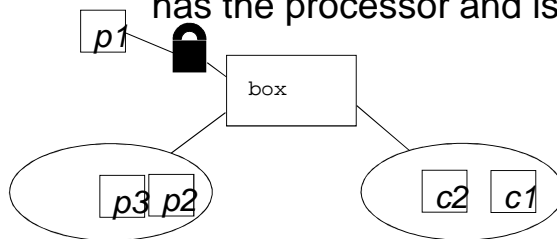
Which thread gets the lock after p1 releases it?



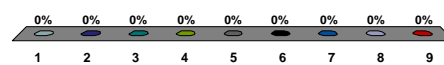
1. P2
2. P3
3. C1
4. C2
5. Another thread not pictured
6. C1 or c2
7. P2 or p3



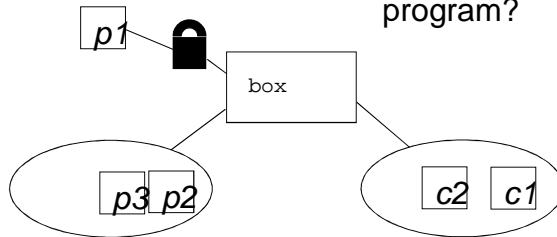
Assuming a uni-processor, what thread currently has the processor and is executing?



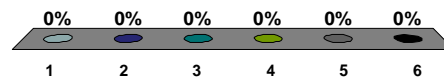
1. P1
2. P2
3. P3
4. C1
5. C2
6. Another thread not pictured
7. C1 or c2
8. P2 or p3
9. P1 or another thread not pictured



Which of the thread(s) shown must currently have or have had the processor at some time in the execution of this program?



1. We can't tell (not enough information)
2. P1
3. P1, p2 and p3
4. P1, c1 and c2
5. P1, c2 and p3
6. All of them



When can we use `notify()` instead of `notifyAll()`?

1. Always
2. Never
3. When all threads are waiting for the same condition
4. When all threads are waiting for the same condition and at most one can benefit
5. As long as we have fewer than 10 threads

