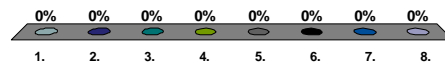


How can a thread move from “Ready to Run” to “Running”?

1. preempted
2. timeslice expires
3. dispatched
4. wait() returns
5. sleep() returns
6. all of the above
7. 1 and 2
8. 1, 2 and 3



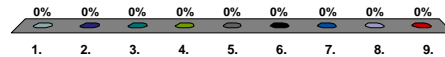
How can a thread move from “Running” to “Ready to Run”?

1. preempted
2. timeslice expires
3. dispatched
4. wait() returns
5. sleep() returns
6. all of the above
7. 1 and 2
8. 1, 2 and 3



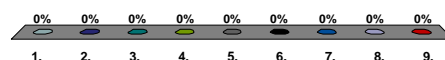
Under what circumstances can a thread leave the “Waiting” state?

1. sleep time expires
2. notify()
3. notifyAll()
4. wait time expires
5. interrupt()
6. I/O complete
7. all of the above
8. 1-5
9. 2-5



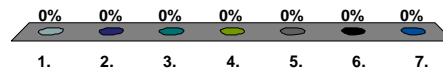
Which of the following keep the locks?

1. wait()
2. sleep()
3. yield()
4. stop()
5. suspend()
6. all of the above
7. 2, 3, and 5
8. 2, 3, 4, and 5
9. 1 and 4



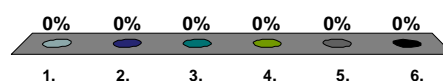
What are the thread priorities in Java?

1. 0 to 9 (integers)
2. 1 to 9 (integers)
3. 0 to 10 (integers)
4. 1 to 10 (integers)
5. [0,10) (reals)
6. 0, 0.5, 1, ..., 9
7. -1, 0, 1, e, pi



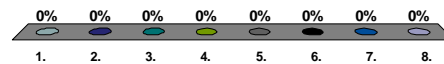
What type of scheduling does the JVM use?

1. dispatch-based priority scheduling
2. priority-based preemptive scheduling
3. preemptive-based priority scheduling
4. priority-based dispatch scheduling
5. any of the above
6. it's up to the JVM implementation to choose



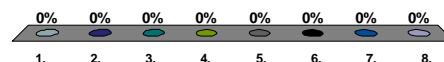
How many of our 10 Java priorities map onto the normal Win-32 priority (THREAD_PRIORITY_NORMAL)?

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. n
8. JVM's choice



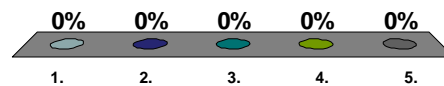
Assuming a uni-processor and that our JVM is using priority-based preemptive scheduling with timeslicing, and we have 5 threads ready to run and no thread currently running, which of the following threads will move to running?
t1 (priority 3); t2 (prio 6); t3 (prio 8); t4 (prio 6); t5 (prio 8)

1. t1
2. t2
3. t3
4. t4
5. t5
6. t2 or t4
7. t3 or t5
8. it's up to the JVM



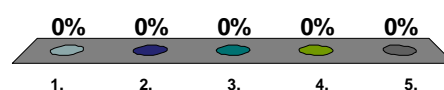
What is priority inversion?

1. when a higher priority thread takes the lock from a lower priority thread
2. when a higher priority thread preempts a lower priority thread
3. when two equal priority threads take turns
4. when a higher priority thread has to wait because a lower priority thread has the lock
5. it's up to the JVM



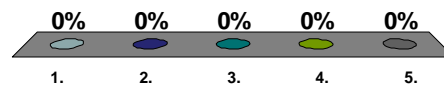
When does the priority inversion become unbounded?

1. when medium priority threads also compete for the lock
2. when an even higher priority thread needs the lock
3. when two equal priority threads take turns
4. when medium priority threads that do not need the lock also compete for the processor
5. it's up to the JVM



What are green threads?

1. when a JVM uses the underlying O/S threading system
2. when a JVM provides its own thread package
3. threads of priority higher than 5
4. threads that are recycled and do not add to your carbon footprint
5. it's up to the JVM



What is/are the advantage(s) of green threads?

1. the JVM has complete control over its threads
2. it's less work than having the O/S deal with the threads
3. we can have as many Java thread priorities as we like
4. your Java threads will behave exactly the same regardless of O/S
5. it's up to the JVM
6. all of the above
7. 2 and 3
8. 1 and 4



What is the opposite of green threads?

1. red threads
2. Canadian threads
3. ungreen threads
4. foreign threads
5. native threads
6. threads that are not recycled and add to your carbon footprint
7. it's up to the JVM

