|  |
| --- |
| **Threads Interview Questions-1**  **Note:** Why wait and notify method in object because threads wait for lock and lock is implemented on Object level. it’s as simple as this. |
| **Q1) What is a Thread?**  Ans) In Java, "thread" means two different things:   * An instance of class java.lang.Thread. * A thread of execution.   An instance of Thread is just…an object. Like any other object in Java, it has variables and methods, and lives and dies on the heap. But a thread of execution is an individual process (a "lightweight" process) that has its own call stack. In Java, there is one thread per call stack—or, to think of it in reverse, one call stack per thread. Even if you don't create any new threads in your program, threads are back there running.  The main() method, that starts the whole ball rolling, runs in one thread, called (surprisingly) the main thread. If you looked at the main call stack (and you can, any time you get a stack trace from something that happens after main begins, but not within another thread), you'd see that main() is the first method on the stack— the method at the bottom. But as soon as you create a new thread, a new stack materializes and methods called from that thread run in a call stack that's separate from the main() call stack. |
| **Q2) What is difference between thread and process?**  Ans) **Differences between threads and processes are:-** 1. Threads share the address space of the process that  created it; processes have their own address.  2. Threads have direct access to the data segment of its process; processes have their own copy of the data segment of the parent process.  3. Threads can directly communicate with other threads of its process; processes must use interprocess communication to communicate with sibling processes.  4. Threads have almost no overhead; processes have considerable overhead.  5. New threads are easily created; new processes require duplication of the parent process.  6. Threads can exercise considerable control over threads of the same process; processes can only exercise control over child processes.  7. Changes to the main thread (cancellation, priority change, etc.) may affect the behavior of the other threads of the process; changes to the parent process do not affect child processes. |
| **Q3) What are the advantages or usage of threads?**  Ans)  **Threads support concurrent operations. For example,**  • Multiple requests by a client on a server can be handled as an individual client thread.  • Long computations or high-latency disk and network operations can be handled in the background without disturbing foreground computations or screen updates.   **Threads often result in simpler programs.** • In sequential programming, updating multiple displays normally requires a big while-loop that performs small parts of each display update. Unfortunately, this loop basically simulates an operating system scheduler. In Java, each view can be assigned a thread to provide continuous updates. • Programs that need to respond to user-initiated events can set up service routines to handle the events without having to insert code in the main routine to look for these events.  **Threads provide a high degree of control.** • Imagine launching a complex computation that occasionally takes longer than is satisfactory. A "watchdog" thread can be activated that will "kill" the computation if it becomes costly, perhaps in favor of an alternate, approximate solution. Note that sequential programs must muddy the computation with termination code, whereas, a Java program can use thread control to non-intrusively supervise any operation.   **Threaded applications exploit parallelism.** • A computer with multiple CPUs can literally execute multiple threads on different functional units without having to simulating multi-tasking ("time sharing"). • On some computers, one CPU handles the display while another handles computations or database accesses, thus, providing extremely fast user interface response times. |
| **Q4)What are the two ways of creating thread?**  Ans) There are two ways to create a new thread.  1)**Extend the Thread class** and override the run() method in your class. Create an instance of the subclass and invoke the start() method on it, which will create a new thread of execution. e.g.  public class NewThread extends Thread{  public void run(){  // the code that has to be executed in a separate new thread goes here }  public static void main(String [] args){  NewThread c = new NewThread();  c.start();  }  }  2)**Implements the Runnable interface.**The class will have to implement the run() method in the Runnable interface. Create an instance of this class. Pass the reference of this instance to the Thread constructor a new thread of execution will be created. e.g. class  public class NewThread implements Runnable{  public void run(){  // the code that has to be executed in a separate new thread goes here }  public static void main(String [] args){  NewThread c = new NewThread();  Thread t = new Thread(c); t.start(); }  } |
| **Q5) What are the different states of a thread's lifecycle?**  Ans) The different states of threads are as follows:  1) **New** – When a thread is instantiated it is in New state until the start() method is called on the thread instance. In this state the thread is not considered to be alive.  2) **Runnable** – The thread enters into this state after the start method is called in the thread instance. The thread may enter into the Runnable state from Running state. In this state the thread is considered to be alive.  3) **Running** – When the thread scheduler picks up the thread from the Runnable thread’s pool, the thread starts running and the thread is said to be in Running state.  4)**Waiting/Blocked/Sleeping** – In these states the thread is said to be alive but not runnable. The thread switches to this state because of reasons like wait method called or sleep method has been called on the running thread or thread might be waiting for some i/o resource so blocked. 5)      Dead – When the thread finishes its execution i.e. the run() method execution completes, it is said to be in dead state. A dead state can not be started again. If a start() method is invoked on a dead thread a runtime exception will occur. |
| **Q6) What is use of synchronized keyword?**  Ans) synchronized keyword can be applied to static/non-static methods or a block of code. Only one thread at a time can access synchronized methods and if there are multiple threads trying to access the same method then other threads have to wait for the execution of method by one thread. Synchronized keyword provides a lock on the object and thus prevents race condition. E.g.  public void synchronized method(){}   public void synchronized staticmethod(){} public void myMethod(){              synchronized (this){             // synchronized keyword on block of  code             }  } |
| **Q7) What is the difference when the synchronized keyword is applied to a static method or to a non static method?**  Ans) When a synch non static method is called a lock is obtained on the object. When a synch static method is called a lock is obtained on the class and not on the object. The lock on the object and the lock on the class don’t interfere with each other. It means, a thread accessing a synch non static method, then the other thread can access the synch static method at the same time but cannot access the synch non static method. |
| **Q8) What is a volatile keyword?**  Ans) In general each thread has its own copy of variable, such that one thread is not concerned with the value of same variable in the other thread. But sometime this may not be the case. Consider a scenario in which the count variable is holding the number of times a method is called for a given class irrespective of any thread calling, in this case irrespective of thread access the count has to be increased so the count variable is declared as volatile. The copy of volatile variable is stored in the main memory, so every time a thread access the variable even for reading purpose the local copy is updated each time from the main memory. The volatile variable also have performance issues. |
| **Q9) What is the difference between yield() and sleep()?**  Ans)  yield() allows the current the thread to release its lock from the object and scheduler gives the lock of the object to the other thread with same priority.          sleep() allows the thread to go to sleep state for x milliseconds. When a thread goes into sleep state it doesn’t release the lock. |
| **Q10) What is the difference between wait() and sleep()?**  Ans)  1) wait() is a method of Object class. sleep() is a method of Object class.  2) sleep() allows the thread to go to sleep state for x milliseconds. When a thread goes into sleep state it doesn’t release the lock. wait() allows thread to release the lock and goes to suspended state. The thread is only active when a notify() or notifAll() method is called for the same object. |
| **Q11) What is difference between notify() and notfiyAll()?**  Ans) notify( ) wakes up the first thread that called wait( ) on the same object.  notifyAll( ) wakes up all the threads that called wait( ) on the same object. The  highest priority thread will run first. |
| **Q12) What happens if a start method is not invoked and the run method is directly invoked?**  Ans) If a thread has been instantiated but not started its is said to be in new state. Unless until a start() method is invoked on the instance of the thread, it will not said to be alive. If you do not call a start() method on the newly created thread instance thread is not considered to be alive. If the start() method is not invoked and the run() method is directly called on the Thread instance, the code inside the run() method will not run in a separate new thread but it will start running in the existing thread. |
| **Q13) What happens when start() is called?**  Ans) A new thread of execution with a new call stack starts. The state of thread changes from new to runnable. When the thread gets chance to execute its target run() method starts to run. |
| **Q14) If code running is a thread creates a new thread what will be the initial priority of the newly created thread?**  Ans) When a code running in a thread creates a new thread object , the priority of the new thread is set equal to the priority of the thread which has created it. |
| **Q15) When jvm starts up, which thread will be started up first?**  Ans) When jvm starts up the thread executing main method is started. |
| **Q16) What are the daemon threads?**  Ans) Daemon thread are service provider threads run in the background,these not used to run the application code generally.When all user threads(non-daemon threads) complete their execution the jvm exit the application whatever may be the state of the daemon threads. Jvm does not wait for the daemon threads to complete their execution if all user threads have completed their execution.  To create Daemon thread set the daemon value of Thread using setDaemon(boolean value) method. By default all the threads created by user are user thread. To check whether a thread is a Daemon thread or a user thread use isDaemon() method.  Example of the Daemon thread is the Garbage Collector run by jvm to reclaim the unused memory by the application. The Garbage collector code runs in a Daemon thread which terminates as all the user threads are done with their execution. |
| **Q17) What all constructors are present in the Thread class?**  Ans) Thread()  Thread(Runnable target) Thread(Runnable target, String name) Thread(String name) |
| **Q18) Can the variables or classes be Synchronized?**  Ans) No. Only methods can be synchronized. |
| **Q19) How many locks does an object have?**  Ans) Each object has only one lock. |
| **Q20) Can a class have both Synchronized and non-synchronized methods?**  Ans) Yes a class can have both synchronized and non-synchronized methods. |

**Threads Interview Questions-2**

**Q1) If a class has a synchronised method and non-synchronised method, can multiple threads execute the non-synchronised methods?**

Ans) Yes. If a class has a synchronised and non-synchronised methods, multiple threads can access the non-synchronised methods.

**Q2) If a thread goes to sleep does it hold the lock?**

Ans) Yes when a thread goes to sleep it does not release the lock.

**Q3)Can a thread hold multiple locks at the same time?**

Ans) Yes. A thread can hold multiple locks at the same time. Once a thread acquires a lock and enters into the synchronized method / block, it may call another synchronized method and acquire a lock on another object.

**Q4) Can a thread call multiple synchronized methods on the object of which it hold the lock?**

Ans) Yes. Once a thread acquires a lock in some object, it may call any other synchronized method of that same object using the lock that it already holds.

**Q5) Can static methods be synchronized?**

Ans) Yes. As static methods are class methods and have only one copy of static data for the class, only one lock for the entire class is required. Every class in java is represented by java.lang.Class instance. The lock on this instance is used to synchronize the static methods.

**Q6) Can two threads call two different static synchronized methods of the same class?**

Ans) No. The static synchronized methods of the same class always block each other as only one lock per class exists. So no two static synchronized methods can execute at the same time.

**Q7)Does a static synchronized method block a non-static synchronized method?**

Ans)No As the thread executing the static synchronized method holds a lock on the class and the thread executing the non-satic synchronized method holds the lock on the object on which the method has been called, these two locks are different and these threads do not block each other.

**Q8) Once a thread has been started can it be started again?**

Ans) No. Only a thread can be started only once in its lifetime. If you try starting a thread which has been already started once an IllegalThreadStateException is thrown, which is a runtime exception. A thread in runnable state or a dead thread can not be restarted.

**Q9) When does deadlock occur and how to avoid it?**

Ans) When a locked object tries to access a locked object which is trying to access the first locked object. When the threads are waiting for each other to release the lock on a particular object, deadlock occurs .

**Q10) What is a better way of creating multithreaded application? Extending Thread class or implementing Runnable?**

Ans) If a class is made to extend the thread class to have a multithreaded application then this subclass of Thread can not extend any other class and the required application will have to be added to this class as it can not be inherited from any other class. If a class is made to implement Runnable interface, then the class can extend other class or implement other interface.

**Q11) Can the start() method of the Thread class be overridden? If yes should it be overridden?**

Ans) Yes the start() method can be overridden. But it should not be overridden as itâ€™s implementation in thread class has the code to create a new executable thread and is specialised.

**Q12) What are the methods of the thread class used to schedule the threads?**

Ans) The methods are as follows:

* public static void sleep(long millis) throws InterruptedException
* public static void yield()
* public final void join() throws InterruptedException
* public final void setPriority(int priority)
* public final void wait() throws InterruptedException
* public final void notify()
* public final void notifyAll()

**Q13) Which thread related methods are available in Object class?**

Ans) The methods are:

* public final void wait() throws Interrupted exception
* public final void notify()
* public final void notifyAll()

**Q14) Which thread related methods are available in Thread class?**

Ans) Methods which are mainly used :

* public static void sleep(long millis) throws Interrupted exception
* public static void yield() public final void join() throws Interrupted exception
* public final void setPriority(int priority)
* public void start()
* public void interrupt()
* public final void join()
* public void run()
* public void resume()

**Q15) List the methods which when called the thread does not release the locks held?**

Ans) Following are the methods.

* notify()
* join()
* sleep()
* yield()

**Q16) List the methods which when called on the object the thread releases the locks held on that object?**

Ans) wait()

**Q17) Does each thread has its own thread stack?**

Ans) Yes each thread has its own call stack. For eg

Thread t1 = new Thread();  
Thread t2 = new Thread();  
Thread t3 = t1;

In the above example t1 and t3 will have the same stack and t2 will have its own independent stack.

**Q18) What is thread starvation?**

Ans) In a multi-threaded environment thread starvation occurs if a low priority thread is not able to run or get a lock on the resoruce because of presence of many high priority threads. This is mainly possible by setting thread priorities inappropriately.

**Q19) What is threadLocal variable?**

Ans) ThreadLocal is a class. If a variable is declared as threadLocal then each thread will have a its own copy of variable and would not interfere with the other's thread copy. Typical scenario to use this would be giving JDBc connection to each thread so that there is no conflict.

**ThreadLocal class by JAVA API**  
public class ThreadLocal {   
  public Object get();  
  public void set(Object newValue);  
  public Object initialValue();  
}

**Implementation of ThreadLocal**  
public class ConnectionDispenser {   
  private static class ThreadLocalConnection extends ThreadLocal {   
    public Object initialValue() {   
      return DriverManager.getConnection(ConfigurationSingleton.getDbUrl());   
    }   
  }   
  
  private static ThreadLocalConnection conn = new ThreadLocalConnection();   
  
    public static Connection getConnection() {   
      return (Connection) conn.get();   
    }   
  }