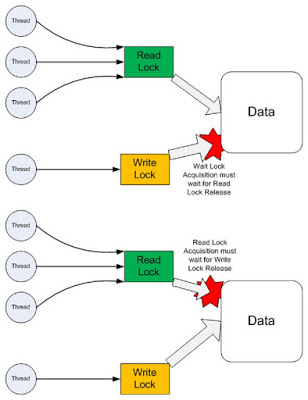
Multithreading Questions:

1. **What is Busy Spinning ? Why should you use it in java ?**

Busy spinning is a waiting strategy, in which a thread just wait for a condition to become true, but instead of calling wait or sleep method and releasing the CPU, it just spins.

By not releasing the CPU or [suspending the thread](http://java67.blogspot.com/2015/06/how-to-pause-thread-in-java-using-sleep.html), your thread retains all the cached data and instruction, which may be lost if the thread was suspended and resumed back in a different core of CPU.

1. **What is Read-Write Lock?**   
   Read - write locks provide simultaneous read access to many threads while the write access stays with one thread at a time.



1. **How to create immutable class in java ?**

Declare the class as final so it can’t be extended.

Make all fields private so that direct access is not allowed.

Don’t provide setter methods for variables

Initialize all the fields via a constructor performing deep copy.

Perform cloning of objects in the getter methods to return a copy rather than returning the actual object reference.

1. **Which Design Pattern Will You Use to Shield Your Code From a Third Party library .Which Will Likely to be Replaced by Another in Couple of Months?**  
   [code against an interface rather than implementation](http://javarevisited.blogspot.com/2014/11/why-use-interface-in-java-or-object-oriented-programming.html) and then use dependency injection to provide a particular implementation.
2. **How do you share an object between threads? or How to pass an object from one thread to another?**   
   BlockingQueue. Used in Consumer Producer problem.
3. **How do find if your program has a deadlock?**  
   By taking thread dump using kill -3, using JConsole or VisualVM)
4. **Write a program to create deadlock.**

Class Deadlock {

Public void method1() {

Synchronized(Integer.class) {

SOP(“lock on Integer1”);

Synchronized(String.class) {

SOP(“lock on String1”);

}

}

}

Public void method2() {

Synchronized(String.class) {

SOP(“lock on String2”);

Synchronized(Integer.class) {

SOP(“lock on Integer2”);

}

}

}

If two or more threads try to execute method1 and method2 at same time.. there are chances of deadlock as thread1 that enters method1 takes lock on Integer class first and at same time Thread2 enters method2 that takes lock on String class. And then Thread1 holds the lock of Integer and try to gain lock of String.. whereas Thread2 holds the lock of String and tries to get lock on Integer. This is deadlock situation.

To prevent this deadlock:

Public void method1() {

Synchronized(Integer.class) {

SOP(“lock on Integer1”);

Synchronized(String.class) {

SOP(“lock on String1”);

}

}

}

Public void method2() {

Synchronized(Integer.class) {

SOP(“lock on Integer2”);

Synchronized(String.class) {

SOP(“lock on String2”);

}

}

both methods are accessing lock on Integer and String class literal in same order. So, if thread A acquires lock on Integer object , thread B will not proceed until thread A releases Integer lock.

1. **Write a Program to solve Producer Consumer problem in Java?**To solve, Use BlockingQueue. It has two methods put() and take() which blocks the user if tries to put() in full queue

And blocks the user if tries to take() from empty queue.

BlockingQueue queue = new LinkedBlockingQueue(); // implements FIFO

1. **Difference between CountdownLatch and CyclicBarrier.**

**CountdownLatch** : When we need a task to start when all other threads are done with their execution.

Example :

Public class Task {

Main(String[] args) {

CountdownLatch latch = new CountdownLatch(3); // Pass no. of Threads should be done with work.

Worker w1 = new Worker(1,latch);

Worker w2 = new Worker(2,latch);

Worker w3 = new Worker(3,latch);

W1.start();

W2.start();

W3.start();

**Latch.await(); // this thread need to wait for worker threads to complete their task.**

Syso(“Task associated work can now be done”);

}

}

Class Worker extends Thread {

Private int workerNum;

Private CountdownLatch latch;

Worker( int num, CountdownLatch latch) {

workerNum = num;

this.latch = latch;

}

Public void run() {

// write code here

**latch.countdown();**

}

}

CyclicBarrier : When multiple threads perform subtasks and output of these subtasks need to be combined to form final output.

When a thread completes its task, it calls await() method to wait for other threads to reach the barrier. When all threads reach barrier, it gives way for threads to proceed.

Difference : Countdownlatch can beused only once until its count reaches 0.

CyclicBarrier can be used again once all threads in a barrier is released.

1. **Difference between wait and sleep**

Wait release lock and must be called from synchronized block. If you call wait method without synchronization, it will throw IllegalMonitorStateException in Java.  
Sleep only pause the thread for sometime, keep the lock and release the CPU and giving another thread an opportunity to execute.

1. **How ConcurrentHashMap is implemented in java ?**

In ConcurrentHashMAp, concurrency level is maintained which allows Map partition. Default concurrency level is 16 and accordingly Map is divided into 16 parts and each is governed with a different lock. This means that 16 threads can operate on CHM at same time but on different parts of map.

Hence, ConcurrentHashMap allows multiple readers to read concurrently without any [blocking](http://javarevisited.blogspot.com/2012/02/what-is-blocking-methods-in-java-and.html)  
ConcurrentHashMap also uses ReentrantLock to internally lock its segments.  
During the update operation, ConcurrentHashMap only locks a portion of Map instead of whole Map.  
All operations of ConcurrentHashMap are [thread-safe](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html).

Since ConcurrentHashMap implementation doesn't lock whole Map, there is chance of read overlapping with update operations like put() and remove(). In that case result returned by get() method will reflect most recently completed operation from there start.  
Iterator are fail-safe and never throw ConcurrentModificationException in java.

ConcurrentHashMap doesn't allow null as key or value.  
During putAll() and clear() operations, the concurrent read may only reflect insertion or deletion of some entries.

1. **When to use CHM ?**

When you have multiple readers and few writers. If writers outnumber reader, or writer is equal to reader, than performance of ConcurrentHashMap effectively reduces to [synchronized map](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) or [Hashtable](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html)

1. **What is Executor in Multithreading? What is difference between execute() and submit() method?**

Executor is concurrency API to manage and launch threads. Executor uses pool of active threads and queue. Thread is used to execute task and if all threads are running , the new task waits in queue for a thread to become free.

Main Interface is Executor.

Subinterface of Executor is ExecutorService.

Executor provides method execute() and ExecutorService provides method submit(). Both are used to submit a task to thread pool.

Execute() method accepts only Runnable Implemented class. Submit() method accepts Runnable as well as Callable implemented class.

Example : ExecutorService e = Executors.newFixedThreadPool(5); // 5 threads will be maintained.

Suppose we have class Worker1 implements Runnable and implementing run method.

e.execute(new Worker1()); // this method does not return

Suppose we have class Worker2 implements Callable and implementing call method. (call method returns Object)

e.submit(new Worker2()); // this method returns a Future Object which can be used to retrieve value returned by Callable.

1. **How do you share data between two threads in Java?**  
   Can share data using BlockingQueue.

To share variable , can make variable global so that both threads can have access on it OR pass the variable in constructor of Threads.

1. **What is race condition in Java?**

a race condition occurs due to race between multiple threads, if a thread which is supposed to execute first lost the race and executed second, behaviour of code changes

1. **What happens when an Exception occurs in a thread?**  
   A thread terminates abruptly due to an uncaught exception and JVM will call Thread.UncaughtExceptionHandler, passing the thread and exception as arguments.
2. What is the difference between synchronized and concurrent collection in Java?  
   synchronized allow only one thread to execute at a time, whereas concurrent collection lets multiplie threads to execute at same time.
3. **What is livelock in java?**

Livelock is a special case of resource starvation. A real-world example of livelock occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but they end up swaying from side to side without making any progress because they both repeatedly move the same way at the same time.

1. **How do you check if a Thread holds a lock or not?**a method called holdsLock() on java.lang.Thread, it returns true if and only if the current thread holds the monitor lock on the specified object.
2. **Which JVM parameter is used to control stack size of a thread ?**

 -Xms        set initial Java heap size

  -Xmx        set maximum Java heap size

  -Xss>         set java thread stack size

1. **There are three threads T1, T2, and T3? How do you ensure sequence T1, T2, T3 in Java?**  
   Use join method. It will put the current thread on wait until the thread on which it is called is dead.

Class MyThread extends Thread {

…..

}

Class MainClass {

Main() {

MyThread t1 = new MyThread();

MyThread t2 = new MyThread();

MyThread t3 = new MyThread();

T1.start()

T1.join(); // here the thread associated with MAinClass will stop further execution until T1 thread gets terminated.

T2.start()

T2.join();

T3.start();

T3.join();

}

}

1. **What does yield() method do ?**

Yield is a static method and only guarantees that current thread will relinquish the CPU but doesn't say anything about which other thread will get CPU. Its possible for the same thread to get CPU back and start its execution again.

1. **What happens if you submit task, when queue of thread pool is already full?**

ThreadPool Executor’s submit() method throws RejectedExecutionException.

1. **What happens if a thread throws an Exception inside synchronized block?**no matter how you exist synchronized block, either normally by finishing execution or abruptly by throwing exception, thread releases the lock it acquired
2. **Is it possible to make array volatile in java?**

Yes, only reference will be volatile but not array elements. If one thread changes the reference variable points to another array, which will provide a volatile guarantee.

1. **How would you call wait() method? Would you use if block or loop and why ?**

wait() method should always be called in loop. It is likely that, until thread gets CPU to start running again, the condition may not hold.

1. CASE 1: There are two threads T1 and T2. T1 is accessing m1 method. Will T2 be able to access m2 method on the same instance at the same time?

Class A {

Synchronized void m1() {

….

}

Void method2() {

…

}

}

Output: Yes, T2 will be able to access m2 as it does not require lock to access m2 method.

CASE 2: There are two threads T1 and T2. T1 is accessing m1 method. Will T2 be able to access m2 method on the same instance at the same time?

Class A {

Synchronized void m1() {

….

}

Synchronized Void method2() {

…

}

}

Output: No

CASE 3: There are two threads T1 and T2. T1 is accessing m1 method. Will T2 be able to access m2 method on the same instance at the same time?

Class A {

Synchronized static void m1() {

….

}

Synchronized Void method2() {

…

}

}

Output: No, T2 will not be able to access m2 as it requires object level lock to access m2 method and T1 thread has taken class level lock.You can read more about Object level lock vs Class level lock.

1. **What is Thread scheduler and time slicing?**

Thread Scheduler is the Operating System service that allocates the CPU time to the available runnable threads.

Time Slicing is the process to divide the available CPU time to the available runnable threads. Allocation of CPU time to threads can be based on thread priority or the thread waiting for longer time will get more priority in getting CPU time.

1. **What is context switching in multithreading?**

process of storing and restoring of CPU state so that Thread execution can be resumed from the same point at a later point of time.

1. **Why Thread sleep() and yield() method are static ?**

Because these methods work on current thread only.

1. **What is Java TimerTask?**

**java.util.Timer** is a utility class that can be used to schedule a thread to be executed at regular intervals.

**java.util.TimerTask** is an [abstract class](https://www.journaldev.com/1582/abstract-class-in-java) that implements Runnable interface and we need to extend this class to create our own **TimerTask** that can be scheduled using *java Timer* class.

Java Timer class is thread safe

Timer class uses **java.util.TaskQueue** to add tasks at given regular interval and at any time there can be only one thread running the TimerTask, for example if you are creating a Timer to run every 10 seconds but single thread execution takes 20 seconds, then Timer object will keep adding tasks to the queue and as soon as one thread is finished, it will notify the queue and another thread will start executing.

Java Timer class uses Object [wait and notify](https://www.journaldev.com/1037/java-thread-wait-notify-and-notifyall-example) methods to schedule the tasks.

Example:-

public class MyTimerTask extends TimerTask {

@Override

public void run() {

// write code here

}

public static void main(String args[]){

TimerTask timerTask = new MyTimerTask();

Timer timer = new Timer(true); //Boolean argument : true means its daemon thread.

timer.scheduleAtFixedRate(timerTask, 0, 10\*1000); // (task to be scheduled, delay, period)

}

}