**VOCABULARY**

Every read/write operation needs **to acquire lock**

We can use Iterator **to traverse** Set and List and also Map type of Objects

a ListIterator allows you **to traverse in either directions (Both forward and backward)**

the iterator, it will **iterate through the Hashset**

**to Iterate through LinkedList Instance**

**Locking the entire collection** is **a performance overhead**.

# a ConcurrentModificationException happens when one thread tries to modify a collection while another is iterating over it.

SynchronizedHashMap **returns Iterator, which fails-fast** on concurrent modification.

We will **use [ExecutorService](http://crunchify.com/how-to-run-multiple-threads-concurrently-in-java-executorservice-approach/" \t "_blank) to run 5 threads in parallel**

**Create object for each HashTable, SynchronizedMap** and CrunchifyConcurrentHashMap

Add and **retrieve 500k entries from Map**

# How to convert HashMap to ArrayList in Java?

How **to detect a deadlock in** the program?

# While your app is hanging

# use ThreadMXBean to monitor Threads

# how to handle read/write [operation](http://crunchify.com/in-java-how-to-perform-file-search-operation-using-java-nio-file-interface-tutorial-on-file-and-directory-operations/)

# How *to deal with Race Condition*?

# good scalability

# if you need to ensure data consistency

# we would *leverage some form of auto-scaling*

# approach to solving

**INTERVIEW QUESTIONS**

**GOOD URLs**

[**http://crunchify.com/java-collection-how-to-convert-arrays-to-set/**](http://crunchify.com/java-collection-how-to-convert-arrays-to-set/)

***How is INDEX organized in DB?***

***???????***

***Can INDEX be organized on two columns?***

***???????***

***What kind of columns can be used for forming INDEX?***

***???????***

***Can be date columns used for organizing INDEX?***

***???????***

***What tools are used for monitoring\analyzing performance of SQL executions in DB (ORACLE, MySQL)?***

***???????***

***How is SQL query executed in DB? What is the execution plan in DB?***

***???????***

***How to enhance SQL executions in DB?***

***???????***

***How do JDBC Statement and ParameterStatement perform in DB?***

***???????***

***Can the conditions from WHERE be used in HAVING and vice verse?***

***???????***

***Can you get an item from HashSet by index?***

HashSet should be used when you don't want to allow duplicate value in your collection.

In HashSet you **won't be able to find value by index**.

If you still want to find the element you have no other option but to use the iterator, it will iterate through the Hashset and give you one by one element from the Hashset.

***What is the difference between fail-fast and fail-safe ?***

***???????***

# *What is the* [*Difference between Iterator and Listiterator?*](http://stackoverflow.com/questions/10977992/difference-between-iterator-and-listiterator)

There are two differences:

1. We can use Iterator **to traverse** Set and List and also Map type of Objects. While a **ListIterator** can be used to traverse **for List-type Objects**, **but not for Set-type of Objects**.

That is, we **can get a Iterator object by using Set and List**, see here:

By using Iterator we can retrieve the elements from Collection Object in forward direction only.

Methods in Iterator:

* 1. hasNext()
  2. next()
  3. remove()

Iterator iterator = Set.iterator();

Iterator iterator = List.iterator();

1. But we get **ListIterator object only from the List interface**, see here:

Iterator is the super class of ListIterator.

where as a **ListIterator** allows you **to traverse in either directions (Both forward and backward)**. So it has two more methods like hasPrevious() and previous() other than those of Iterator. Also, we can get indexes of the next or previous elements (using nextIndex() and previousIndex() respectively )

Methods in ListIterator:

* 1. hasNext()
  2. next()
  3. previous()
  4. hasPrevious()
  5. remove()
  6. nextIndex()
  7. previousIndex()

ListIterator listiterator = List.listIterator();

i.e., we can't get ListIterator object from Set interface.

 Iterator and ListIterator are **fails-fast on concurrent modification**

**ListIterator** as well as Iterator **throws ConcurentModificationException**

***When should you use synchronized HashMap or ConccurentHashMap?***

# ConcurrentHashMap is more performable than SynchronizedHashMap for getting and putting operations

You should use ConcurrentHashMap when you need very high concurrency in your project.

ConcurrentHashMap is recommended for large Maps or large number of read-write operations because of its good scalability.

# ConcurrentHashMap doesn’t throw a ConcurrentModificationException if one thread tries to modify it while another is iterating over it

# *HashMap Vs. ConcurrentHashMap Vs. SynchronizedMap ?*

### **ConcurrentHashMap**

* You should use ConcurrentHashMap when you need very high concurrency in your project.
* It is thread safe without synchronizing the whole map.
* Reads can happen very fast while write is done with a lock.
* There is no locking at the object level.
* The locking is at a much finer granularity at a hashmap bucket level.
* ConcurrentHashMap doesn’t throw a ConcurrentModificationException if one thread tries to modify it while another is iterating over it.
* ConcurrentHashMap uses multitude of locks.

### **SynchronizedHashMap**

* Synchronization at Object level.
* Every read/write operation needs to acquire lock.
* Locking the entire collection is a performance overhead.
* This essentially gives access to only one thread to the entire map & blocks all the other threads.
* It may cause contention.
* SynchronizedHashMap returns Iterator, which fails-fast on concurrent modification.

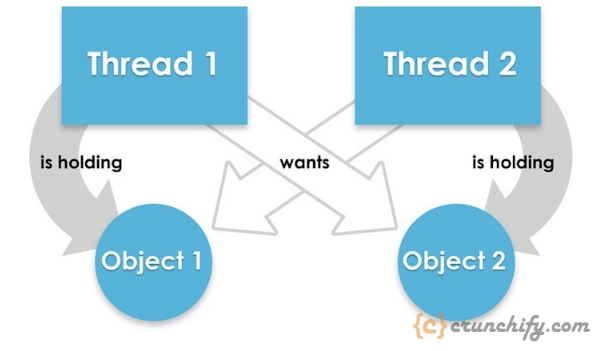
### **ConcurrentHashMap is more performable than SynchronizedHashMap which is more performable than HashMap**

***How to resolve hash collition in HashSet and HashMap in java 8?***

????

# *How to convert HashMap to ArrayList in Java?*

# *How to Generate Java Deadlock Programmatically and How to Analyze Deadlock*



**Deadlock** is a programming situation where two or more threads are blocked forever, this situation arises with at least two threads and two or more resources.

While your app is hanging like in the example above, you can get a thread dump and see the deadlocked threads. For example, on Mac, you can either do Ctrl-\ or simply **use jstack** and process id to get the thread dump which makes it very obvious where the deadlock is.

<http://crunchify.com/how-to-generate-java-deadlock-programmatically-and-how-to-analyze-deadlock/>

***How to detect a deadlock in the program?***

????

# *Have you noticed Race Condition in Java Multi-threading Concurrency Example? How to deal with it?*

<http://crunchify.com/have-you-noticed-race-condition-in-java-multithreading-concurrency-example-how-to-deal-with-it/>

[***Differences between BLOCKED vs WAITING***](http://geekexplains.blogspot.com/2008/07/threadstate-in-java-blocked-vs-waiting.html)***?***

When a thread calls Object.wait method, it releases all the acquired monitors and is put into WAITING (or TIMED\_WAITING if we call the timeout versions of the waitmethod) state. Now when the thread is notified either by notify() or by notifyAll() call on the same object then the waiting state of the thread ends and the thread starts attempting to regain all the monitors which it had acquired at the time of wait call. At one time there may be several threads trying to regain (or maybe gain for the first time) their monitors. If more than one threads attempt to acquire the monitor of a particular object then only one thread (selected by the JVM scheduler) is granted the monitor and all other threads are put into BLOCKED state. Got the difference?

***Difference between WAITING and TIMED\_WAITING states?***  
The difference is quite obvious between the two. A thread in a TIMED\_WAITING state will wait at max for the specified timeout period whereas a thread in the WAITING state keeps waiting for an indefinite period of time. For example, if a thread has called Object.wait method to put itself into WAITING state then it'll keep waiting until the thread is interrupted either by notify() method (OR by notifyAll() method) call on the same object by another thread. Similarly, if a thread has put itself into WAITINGstate by calling Thread.join method then it'll keep waiting until the specified thread terminates.

***How to use ThreadMXBean to monitor Threads?***

package com.secutix.gui.seatmap;

import java.lang.management.ManagementFactory;

import java.lang.management.ThreadInfo;

import java.lang.management.ThreadMXBean;

public class ThreadStatus {

private static final ThreadMXBean mbean = ManagementFactory.getThreadMXBean();

public static void main(String[] args) {

for (int i = 0; i < 3; i++) {

buildAndLaunchThread(i);

}

Thread t = new Thread(){

@Override

public void run() {

while(true){

printThreadStatus();

try {

sleep(3000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

};

t.setName("detector");

t.start();

}

protected static void printThreadStatus() {

ThreadInfo[] infos = mbean.dumpAllThreads(true, true);

for (ThreadInfo threadInfo : infos) {

System.out.println(threadInfo.getThreadName() + " state = " + threadInfo.getThreadState());

}

}

private static void buildAndLaunchThread(int i) {

Thread t1 = new Thread(){

@Override

public void run() {

while(true){

try {

sleep(3000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

};

t1.setName("t" + i);

t1.start();

}

}

***What is Thread Dump?***

**The thread dump** is a snapshot of exactly what’s executing at a moment in time in your [Java Program](http://crunchify.com/category/java-web-development-tutorial/).. While the thread dump format and content may vary between the different Java vendors, at the bare minimum it provides you a list of the stack traces for all Java threads in the [Java Virtual Machine](http://crunchify.com/jvm-tuning-heapsize-stacksize-garbage-collection-fundamental/). Using this information, you can either analyze the problem yourself, or work with those who wrote the running code to analyze the problem.

[**Thread**](http://crunchify.com/what-is-daemon-thread-in-java-example-attached/)**dump is just a list of all threads and the full stack trace of code running on each thread.**

**Java Method to Generate Thread Dump:**

public static String crunchifyGenerateThreadDump() {

final StringBuilder dump = new StringBuilder();

final ThreadMXBean threadMXBean = ManagementFactory.getThreadMXBean();

final ThreadInfo[] threadInfos = threadMXBean.getThreadInfo(threadMXBean.getAllThreadIds(), 100);

for (ThreadInfo threadInfo : threadInfos) {

dump.append('"');

dump.append(threadInfo.getThreadName());

dump.append("\" ");

final Thread.State state = threadInfo.getThreadState();

dump.append("\n java.lang.Thread.State: ");

dump.append(state);

final StackTraceElement[] stackTraceElements = threadInfo.getStackTrace();

for (final StackTraceElement stackTraceElement : stackTraceElements) {

dump.append("\n at ");

dump.append(stackTraceElement);

}

dump.append("\n\n");

}

return dump.toString();

}

**Here is a sample**[**Thread**](http://crunchify.com/tag/java-thread-example/)**Dump:**

**"crunchifyThread3"**

**java.lang.Thread.State: RUNNABLE**

at sun.management.ThreadImpl.getThreadInfo1(Native Method)

at sun.management.ThreadImpl.getThreadInfo(ThreadImpl.java:174)

at com.crunchify.tutorials.CrunchifySynchronizeThread.getThreadDump(CrunchifyThreadDeadLock.java:64)

at com.crunchify.tutorials.CrunchifySynchronizeThread.run(CrunchifyThreadDeadLock.java:50)

at java.lang.Thread.run(Thread.java:722)

**"crunchifyThread1"**

**java.lang.Thread.State: BLOCKED**

at com.crunchify.tutorials.CrunchifySynchronizeThread.run(CrunchifyThreadDeadLock.java:53)

at java.lang.Thread.run(Thread.java:722)

**"Finalizer"**

**java.lang.Thread.State: WAITING**

at java.lang.Object.wait(Native Method)

at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:135)

at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:151)

at java.lang.ref.Finalizer$FinalizerThread.run(Finalizer.java:189)

***How to use shutdown() and awaitTermination(Long.MAX\_VALUE, TimeUnit.DAYS) methods?***

ExecutorService crunchifyExServer = Executors.newFixedThreadPool(THREAD\_POOL\_SIZE);

for (int j = 0; j < THREAD\_POOL\_SIZE; j++) {

crunchifyExServer.execute(new Runnable() {

Override

public void run() {....}

});

}

**// Make sure executor stops**

crunchifyExServer.**shutdown();**

// Blocks until all tasks have completed execution after a shutdown request

crunchifyExServer.**awaitTermination(Long.MAX\_VALUE, TimeUnit.DAYS);**

# *Java NIO (Non-blocking I/O) with Server-Client ?*

<http://crunchify.com/java-nio-non-blocking-io-with-server-client-example-java-nio-bytebuffer-and-channels-selector-java-nio-vs-io/>