Senior Java Lead Interview Questions(Part1)

I would like to share some of the interview questions that I have been asked in last couple of months while I was interviewed for Java Lead position.

Question: What is Meta Space in Java.

Answer: Metaspace is a native (as in: off-heap) memory manager in the hotspot. It is used to manage memory for class metadata. Class metadata are allocated when classes are loaded. Earlier the Class loading mechanism used the permanent generation known as PermGen space however that has been made redundant with Java 8. Metaspace lies in the native memory and non on the heap however PermGen resides on Heap. Please refer to this for more information.

What if you don’t synchronise a method in Java?

Answer :This can give rise to race condition in a multithreaded environment. Below example is how you can have a race condition. With out synchronising a critical section , different threads may access the share data unpredictable and that might poduce different results. This might result in to race condition.

Any operation happens in multiple steps i.e.

Step1: Reading a Variable from Main Memory.

Step2: Updating or processing that data.

Step3: Saving Data in to main memory.

*Thread 1 Executing Step 1 and then 2 and **Then Context Switch Happens.***

Thread 2 May not read the data that was updated by thread 1 as Step three was not completed by Thread 1.

*Thread 2 updates the value. **Context Switch happens.***

Thread 1 overwrites the value updated by thread 2.

```
public class Counter
  {
    private int counter=0;
    public void increment ()
          {
            counter++;
          }
      public int getCounter()
```

```
        {
      return counter;
         }
public static void main(String args[]) throws InterruptedException {
final Counter counter = new Counter();
Runnable r1 = () -> counter.increment ();
Runnable r2 = () -> counter.increment ();
Thread t1 = new Thread (r1);
Thread t2 = new Thread (r2);
t1.start ();
Thread.sleep(1000);
t2.start ();
Thread.sleep(1000);
System.out.println(counter.getCounter ());
}
}
```

How was the internal implementation of Java Hash Map changed with JDK 1.8?

Answer: Since Java 8 the internals of HashMap changed in such a way that if , in case of collisions , the size of the linked list is more than 8 , Java iternally make use of Red black trees to store the elements instead of linked list. This make sure that the get operation are O(log(n)) instead of O(n).

What if you return constant hash code of 1 from hash code method.

And you don't override equals method. i.e you have an employee class and you are storing the object of employee class as a key and the value. What would happen if you hard code the hash code method to return "1". What would happen if you add element to the hash map and what would be the behavior when you try to get by Key

Answer: HashMap will work fine and will be able to store your key value pairs correctly. The hash function may have collisions but that is due to your poor hash function. The index to save the objects is generated as below. You will be able to save and retrieve the key values from the HashMap. In case of collisions the HashMap will return the object asked by you using the equals method defined in the object class.

```
index = hashCode(key) & (n-1).
public class EmployeeExample {

    public static void main(String args[])
    {
        Map <Employee, Employee> empMap=new HashMap<Employee, Employee>();

        Employee emp1=new Employee (101,"vikas","dev", 6789);
        Employee emp2=new Employee (102,"Ravi","QA", 8999);
        Employee emp3=new Employee (103,"Akshaya","DevOps", 7789);
        empMap.put (emp1,emp1 );
        empMap.put (emp2,emp2 );
        empMap.put (emp3, emp3 );
        empMap.put (emp3, emp3 );

        System.out.println(empMap.size ());
        System.out.println(empMap.keySet ().size ());
        empMap.keySet ().stream ().forEach (System.out::println);
        empMap.entrySet ().stream ().forEach (System.out::println);
        System.out.println(empMap.get (emp1));
```

```
    }
```

Synchronized member method versus static synchronized method.

Let say you have two synchronised method in the same class , can two different thread simultaneously access different synchronised methods?

Answer: Two threads can access the public synchronized methods if they are invoked on two differnt object. One thread can't enter in to synchronised block until the other thread has exited the critical section on the same object.

Thread A- Invoking public synchronised method A() on Object A- Allowed.

Thread B — Invoking public synchronised method B() on Object A-Not Allowed until Thread A return.

Thread B-Invoking public synchronised method A() on Object B-Allowed.

The idea here is that all object in java have intrinsic locks which are called ***monitors***. Once a thread acquires a monitor another thread would have to wait for that thread to release the monitor in oder to acquire the lock. One thread can acquire the lock on the same object as many times as needed if this thread has clready acqired the lock on that object.

> *Every time the thread acquires a **monitor** the lock acquire count on that monitor is incremented and when the lock acquired counter is zero means that the thread has released the lock.*

Follow up Question

Let say you have one instance and one static synchronized method , can two different thread execute these methods simultaneously?

Answer: As long as the object on which the lock has been acquired is same , no other thread can access that method in concurrent context. Any thread accessing the Static synchronized method will acquire the lock on the class object so while that thread is executing no other thread can execute any other static synchronized method in the same class but they can execute the other instance methods of the class.

Explain String constant pool.

Answer: String constant pool is a way of optimising the string creation by the JVM. We are highly encouraged to define strings using String s="Java" as opposed to String s= new String("Java"). The reason is that strings are immutable and any point of time you are modifying a string a new object is created. Based on the reference equailty all the string literals are stored on the String constant pool and if you try to create the same literal , you are returned the literal from the pool instead of creating new obeject. There is plethora of docmentation for this and I am not surprised why this is till a popular question.

```
public class Main {
    public static void main(String[] args) {
        String s1 = "abc";
```

```
        String s3 = s1;// Both S1 and S3 will point to the same literal on String Pool.
        String s2 = new String("abc");
        //Reference Equailty
        System.out.println(s1 == s2);//false
        System.out.println(s1 == s3);//true
        System.out.println(s2 == s3);//false
        //Value Equality
        System.out.println(s1.equals(s2));//true
        System.out.println(s1.equals(s3));//true
        System.out.println(s2.equals(s3));//true

    }

}
```

How streams are different from collections?

Answer: This could be a very big answer however there are couple of facts that you need to know about streams.

Streams operation do not modify the underlying collection however they apply the filter and ma operation and provide you with a data structure altogether different from original data structure.

for Exmaple. This will provide you all the even number in the range of 6, 7,8,9,10

IntStream.range(6, 10).filter(n-> n/2==0).collect(toList());

Streams are lazily loaded , just to understand to easily think of the fact that streams resources are only used once you stat collecting those.

for example: The results remain in memory until you call the terminal operation of findFirst on the stream. Stream elements are only processed once you run a terminal oepration.

empList.stream.map(emp-> emp.getSalary()).findfirst();

Internal iteration is a concept when the iteration is managed by the library itself. With streams we don’t need to iterate over a collection however Streams API provides iteration by itself.

Thanks a lot for reading the article. I am very grateful to all of my readers and there valuable suggestion. I have provided some links that I missed here and also formatted this better to remove typos. Thanks again for reading. I have captured some of the design related questions in my part 2 of this series here.