Interview Questions

# Q1. How do you handle error condition while writing stored procedure or accessing stored procedure from java?

A1. Stored procedure should return an error code if some operation fails but if stored procedure itself fails than catching SQLException is the only choice.

# Q2. What is difference between Executor.submit() and Executer.execute() method ?

A2. Former returns an object of Future which can be used to find result from worker thread

There is a difference when looking at exception handling. If your tasks throw an exception and if it was submitted with executing this exception will go to the uncaught exception handler (when you don't have provided one explicitly, the default one will just print the stack trace to System.err). If you submitted the task with submit any thrown exception, checked exception or not, is then part of the task's return status. For a task that was submitted with submitting and that terminates with an exception, the Future.get() will re-throw this exception, wrapped in an ExecutionException.

# Q3. New Features of JAVA 5

A3. Generics, Enum, autoboxing and varargs

# Q4. Why character array is better than String for Storing password in Java

A4. Since Strings are immutable in Java if you store password as plain text it will be available in memory until Garbage collector clears it and since String are used in String pool for reusability there is pretty high chance that it will be remain in memory for long duration, which pose a security threat. Since anyone who has access to memory dump can find the password in clear text and that's another reason you should always use an encrypted password than plain text. Since Strings are immutable there is no way contents of Strings can be changed because any change will produce new String, while if you char[] you can still set all his element as blank or zero. So, Storing password in character array clearly mitigates security risk of stealing password.

With String there is always a risk of printing plain text in log file or console but if use Array you won't print contents of array instead its memory location get printed. Though not a real reason but still make sense.

# Q5. Why wait, notify and notifyAll is defined in Object Class and not on Thread class in Java

A5. In Java all object has a monitor. Threads waits on monitors so, to perform a wait, we need 2 parameters:

- a Thread

- a monitor (any object)

In the Java design, the thread cannot be specified; it is always the current thread running the code. However, we can specify the monitor (which is the object we call wait on). This is a good design, because if we could make any other thread to wait on a desired monitor, this would lead to an "intrusion", posing difficulties on designing/programming concurrent programs. Remember that in Java all operations that are intrusive in another thread's execution are deprecated (e.g. stop())."

Locks are made available on per Object basis, which is another reason wait and notify is declared in Object class rather than Thread class.

# Q6. What is Encapsulation in Java?

A6. Encapsulation is nothing but protecting anything which is prone to change. Rational behind encapsulation is that if any functionality which is well encapsulated in code i.e maintained in just one place and not scattered around code is easy to change

## Advantage of Encapsulation in Java and OOPS

1. Encapsulated Code is more flexible and easy to change with new requirements.

2. Encapsulation in Java makes unit testing easy.

3. Encapsulation in Java allows you to control who can access what.

4. Encapsulation also helps to write immutable class in Java which is a good choice in multi-threading Environment.

5. Encapsulation reduce coupling of modules and increase cohesion inside a module because all piece of one thing are encapsulated in one place.

6. Encapsulation allows you to change one part of code without affecting other part of code.

## What should you encapsulate in code?

Anything which can be change and more likely to change in near future is candidate of Encapsulation. This also helps to write more specific and cohesive code. Example of this is object creation code, code which can be improved in future like sorting and searching logic.

## Important point’s about encapsulation in Java

1. "Whatever changes encapsulate it" is a famous design principle.

2. Encapsulation helps in loose coupling and high cohesion of code.

3. Encapsulation in Java is achieved using access modifier private, protected and public.

4. Factory pattern , Singleton pattern in Java makes good use of Encapsulation.

Q7. What happens if a thread throws an Exception inside synchronized block?

A7. This is one more tricky question for average Java programmer, if he can bring the fact about whether lock is released or not is a key indicator of his understanding. To answer this question, no matter how you exist synchronized block, either normally by finishing execution or abruptly by throwing exception, thread releases the lock it acquired while entering that synchronized block. This is actually one of the reasons I like synchronized block over lock interface, which requires explicit attention to release lock, generally this is achieved by releasing the lock in a finally block.

Q8. What is the difference between the volatile and atomic variable in Java?

A8. This is an interesting question for Java programmer, at first, volatile and atomic variable look very similar, but they are different. Volatile variable provides you happens-before guarantee that a write will happen before any subsequent write, it doesn't guarantee atomicity. For example count++ operation will not become atomic just by declaring count variable as volatile. On the other hand AtomicInteger class provides atomic method to perform such compound operation atomically e.g. getAndIncrement() is atomic replacement of increment operator. It can be used to atomically increment current value by one. Similarly you have atomic version for other data type and reference variable as well.

Q9. Can we make array volatile in Java?

A9. Yes, you can make an array volatile in Java but only the reference which is pointing to an array, not the whole array. What I mean, if one thread changes the reference variable to points to another array, that will provide a volatile guarantee, but if multiple threads are changing individual array elements they won't be having happens before guarantee provided by the volatile modifier.