10 - Threads

< Previous | Next >

- There are two ways to create a thread:

```
    //Here MyClassThatExtendsThread should extend Thread and override the run meth
od.
new MyClassThatExtendsThread().start();

    // MyClassThatImplementsRunnable should implement the interface Runnable.
new Thread( new MyClassThatImplementsRunnable() ).start();
```

- If a class implements Runnable then a thread can be created and started as follows:

```
class MyRunnable implements Runnable { public void run() { //other valid code. } }

Thread t = new Thread ( new MyRunnable() );
t.start()
```

- Threads can use the method wait() to pause their execution until signaled by another thread to continue.
- Callable interface is very similar to Runnable interface and is used with the classes of the Executor framework. It has one method with the following signature: `V call() throws Exception`.
- Fundamental difference between Runnable and Callable: The Thread class, which is responsible for the creation of a new thread and its execution, works only with Runnable (and not Callable).
- So if you want to create a class that can be executed in a separate thread without the use of any other framework classes, then you have to use Runnable.
- The run() method should be overridden to provide the code executed by the thread, when extending the Thread class as the Thread class's run() method does practically nothing. This is similar to implementing the run() method of the Runnable interface.
- suspend(), resume() and stop() are deprecated methods in the Thread class because they don't give the Thread a chance to release shared resources, which may cause deadlocks. We do not expect any questions on these methods in the exam but it is good to know about them for interview purpose.
- new Thread().run() does not start a thread. start() does.
- A call to start() returns immediately but before returning, it internally causes a call to the run method of either the Thread instance or of the Runnable instance.
- Thread class's run() is an interesting method. If the thread object was constructed using a separate Runnable object, then that Runnable object's run method is called (not in separate thread) otherwise, this method does nothing and returns.
- If a Thread is created with a Runnable, Thread class's run() method calls the run() method of the given

Runnable. However, when a class extending Thread is overriding Thread class's run() method, the default behavior of Thread's run() is lost. In that class the overriding run() method is always called, and the Runnable object is ignored.

- The Thread class is a Runnable so you can pass a thread object to the constructor of another thread.
- A thread can enter a synchronized method only after it acquires a lock. Note that acquiring a lock is necessary only for entering synchronized methods, there is no need to get a lock (of the object on which it is trying to execute the method) for entering a normal method. There is only one lock with one object.
- A thread can reacquire a lock(reentrant synchronization). This means once it enters a synchronized method it can call any other synchronized method or the same method (recursively) within it. Hence there is no deadlock.
- Java does not have any mechanism to prevent deadlocks. It is up to the developer to use the locks correctly to avoid deadlocks. One strategy that is commonly used to prevent deadlocks is to always acquire locks of the objects in the same sequence.
- When a static synchronized method is invoked, the thread acquires the intrinsic lock for the Class object associated with the class. Thus access to class's static fields is controlled by a lock that's distinct from the lock for any instance of the class.
- The exam needs you to understand and differentiate among Deadlock, Starvation, and Livelock.
- If you try to run a program without a standard main() method from the command line, the JVM will throw an Error.
- In this case, the program is trying to manipulate the scheduling of thread using priorities which is a bad idea. Simply because operating systems behave differently about priorities. For example, Windows uses Time Slicing i.e. it gives time to all the thread in proportion of their priorities but many unix systems do not let low priority threads run at all if a higher priority thread is running. So the output cannot be determined.
- The run() method of the Thread class can be synchronized!
- The Java exception mechanism is integrated with the Java synchronization model, so that locks are released if synchronized statements and invocations of synchronized methods complete abruptly. Note that this does not apply to java.util.concurrent.locks.Lock. These locks must be released by the programmer explicitly.
- ConcurrentModificationException is thrown by the methods of the Iterator interfaces. It is thrown when, while one thread is iterating through a collection, another thread modifies the collection.
- Calling sleep on a thread only pauses the thread. It does not kill the thread.
- Once the run method of the Thread ends, the thread dies because there is nothing left to do for that thread.
- There is no method kill( ) for a Thread.
- You can start a suspended Thread by calling the method resume() oh the thread.
- When a non-abstract class implements Runnable interface, it must implement public void run() method. (`run` with no parameters).
- `count++` is not an atomic operation, which means that it can be broken up into three separate steps:

1. get the current value of count in cache
2. increment the cache value and
3. update the count with updated cache value.

- It is possible that the JVM may not exit even after the main method returns. The JVM will wait for any non-daemon thread to end, even after the main() method has returned. You can make a thread a daemon thread by calling `setDaemon(true)` before calling start().
- The synchronized keyword can be applied only to **non-abstract methods** that are defined in a class or a block of code appearing in a method or static or instance initialization blocks. It cannot be applied to methods in an interface.

```
public class Test extends Thread {
    static int x, y;
    public synchronized void run(){  for(;;){ x++; y++; System.out.println(x+" "+y
);} }
    public static void main(String[] args) {
        new Test().start();
        new Test().start();
    }
}
```

- You may be tempted by the synchronized keyword on the run method. But note that there are two different thread objects and both the threads are acquiring locks for their own thread object. Therefore, in this case, the synchronized block will not help.

< Previous | Next >