INTERVIEW PREPARATION QUESTIONS

2017 - 21 batch JAVA 8

THREADS IN JAVA

1. What do you mean Thread?

⇨ Thread is a single sequential flow of control within a program or any application and threads are not programs itself as it can not run on its own it can be embedded within any other program or application. Threads are also known as independent paths of execution by which we can take advantage of our CPU by increasing the CPU cycles and reducing the CPU idle time. Suppose let me give you an example as if I have a task that takes 100 milliseconds to finish in a single thread architecture but if you use 10 threads simultaneously then the work will be done in just 10 milliseconds. At a time only one thread can be executed.

2. What is the difference between Multitasking & Multithreading

Multitasking has two different forms one is thread-based multitasking and another one is process-based multitasking but the main difference is that ➔

⇨ In process-based multitasking there are multiple numbers of programs running concurrently by the CPU means here every program share different address spaces and context switching happens in between every program at the run time that's why it creates more overhead as compared to thread-based multitasking.

⇨ In thread-based multitasking there is one single program but has different paths of execution depending on multiple threads each thread is responsible for each path and as this all happens within a single application so it doesn't share different address spaces between themselves so it creates less overhead as compared to the process-based multitasking.

3. Difference between Threads & processes

S.NO	PROCESS	THREAD
1.	Process means any program is in execution.	Thread means a segment of a process.

2.	The process takes more time to terminate.	The thread takes less time to terminate.
3.	It takes more time for creation.	It takes less time for creation.
4.	It also takes more time for context switching.	It takes less time for context switching.
5.	The process is less efficient in terms of communication.	Thread is more efficient in terms of communication.
6.	Process consume more resources.	Thread consume fewer resources.
7.	The process is isolated.	Threads share memory.
8.	The process is called a heavyweight task.	Thread is called lightweight task
9.	Process switching uses interface in the operating system.	Thread switching does not require to call the operating system as the threads share the same address spaces.

10.	If one server process is blocked no other server process can execute until the first process unblocked.	The second thread in the same task could run, while one server thread is blocked.
11.	The process has its own Process Control Block, Stack, and Address Space.	Thread has Parents' PCB, its won Thread Control Block and Stack, and common Address space.

4. How we can create our Threads?

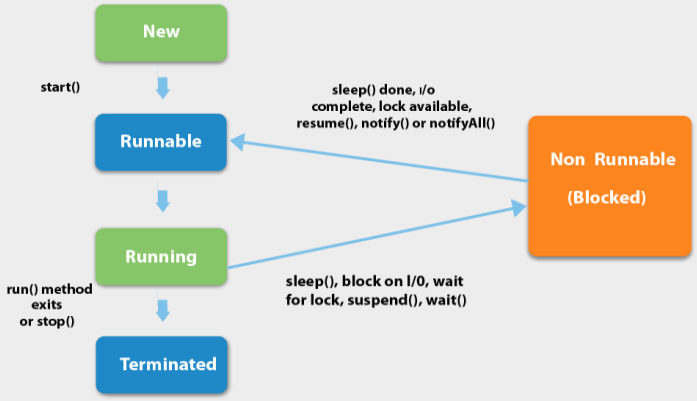
⇨ Java provides a feature to create our user-defined threads according to our needs. We can create user-defined threads in java by two ways one is

- ➔ Extending the Thread class.
- ➔ Implementing the Runnable interface

5. What is the life cycle of a thread

⇨ The life cycle of a thread is divided into five states. All the states are mentioned below

1. New state 2. Runnable 3. Running 4. Not Runnable 5. Terminated



- **New State:**

  When a new thread is created by creating the instance of a thread class but before the invocation of start() method. It first comes into this new state
  Syntax of creating a new thread

  ➔ Thread thread name = new Thread(); //Creating an instance of thread class it's in new born state

- **Runnable state:**

  When we call start() method on any thread instance it will go to runnable state but not running which means that it scheduled in the queue and can run when CPU is allocated to it by the java runtime scheduler and by invoking the void run() method.
  The syntax for this is --> Thread thread name = new Thread();
  thread name.start() // by calling this start method the thread object will go to runnable state from new state

- **Running state:**

  Any thread object comes from runnable to running by invoking the void run() method of the Thread class depending on the priority and scheduling done by java run time scheduler. The run() method will be invoked by different thread objects depending on their priorities. To make any thread run we must override the run() method of thread class without that we can not able to run any user-defined thread. All the operations that a thread performs should be written sequentially inside the overridden run() method.

- **Not Runnable state:**

  When there is any unavailability of of any i/o operation or any type of interrupt occurs or wait() , block() , suspend() these type of method is called then the thread will go to the not runnable state from the running state. That means the thread is alive but currently not eligible to run.

- **Terminated state:**

  A thread will be terminated or dead when it naturally exits the run method or it can be terminated forcefully by calling an interrupt().

6. What is the work of sleep(), wait(), join() and what is the difference between them

⇨ The main difference is that the purpose of these three methods is fully different but their working is somewhat like the same to each other.

I. Sleep():
Sleep() is a method of thread class in java that is used for timed waiting in threads when sleep is called one thread waits for another thread for a specified amount of time. Automatically when time limit exceeded the threads become alive and start execution from where it left. By calling this method thread is going to the waiting state that is known non runnable state for a specified time.

II. Join():
join is a method of thread class when it is called the java runtime system waits for that particular thread to die. In simple words, we can say that java runtime environment stops currently executing threads until the thread it joins completely all of its operations.

III. wait():
wait() is a method of the object class that is used in a scenario when we don't know the exact timing of a thread to complete a task. When wait() is called on a thread it will wait for a thread for an indefinite amount of time until another thread performs some execution. By calling this methods the thread is going to waiting state.

7. What is the differences between run() & start() methods

⇨ Both are very similar to each other but the difference is that by

=> calling the threadname.start() method we initiate the execution of thread after instantiation but that doesn't mean the thread is currently running, it means the thread is eligible to run and also be scheduled in the queue to be selected by the java runtime system. currently it is in the ready queue until java runtime system selects it as a currently executing thread until then thread obj will wait for the system and the necessary resources allocated to it. One more thing is that if we call start method() on any thread it will internally call the run() method on the thread object

=> the invocation of the run() method is not in our hands it fully depends on the java runtime environment how it selects the runnable threads from the waited queue and then after selecting the run() method will invoke automatically and the thread object will set as a currently executing thread means in running state.

8. What will happen if we start a thread two times?

⇨ If we start a thread two times by calling start() method on the same thread object then it will be given us an IllegalthreadStateException. Because once a thread object is started it can never be started again

Here as we observe that the main reason is that one thread object can not be started twice as it's already present in the ready queue .

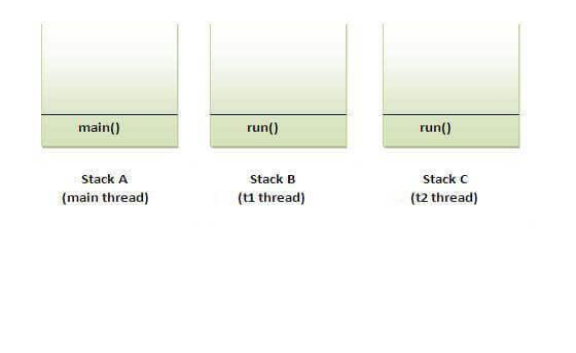
```
public class TestThreadTwice1 extends Thred
{
 public void run(){
  System.out.println("running...");
 }
 public static void main(String args[]){
  TestThreadTwice1 t1=new TestThreadTwice1(
);
  t1.start();  // starting a thread first time (ok)
  t1.start();  // same object we again call start()
 }            // creates a problem here
}
```

```
O/P =>
 running
 Exception in thread "main" java.lang.IllegalThreadStateException
```

9. What is the internal Mechanism of thread

⇨ The internal mechanism of each thread is that they use stack memory for run rime o/p individually not shared with others as they work independently as well. That doesn't mean their memories are not shared.

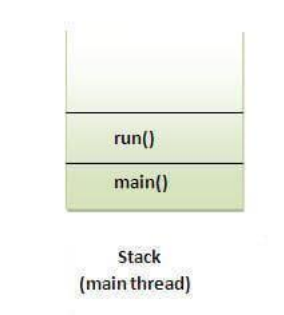
⇨ Each thread has its specific runtime stack area for performing arithmetical and all functional operations



⇨ That is the main reason why if one faces any problem during the operation then it doesn't affect the others.

10. What will happen if we run the thread directly from the main without calling start()?

⇨ If we call the run() method directly without calling the start() then it will not give any exception or error the program will be compiled successfully but the problem is that as we are not calling start() so no new thread will be created at all the run() will be called on the same runtime stack that is main thread runtime stack so the object will behave just like a normal object rather than a thread object



⇨ Here we can see that the run() method placed on the same stack that is the main stack but it doesn't create any other new stack for a new thread for operation so it means that it will behave like a normal object as we normally perform the operation

11. When we should use the RUNNABLE interface to implement thread in our application?

⇨ We have two different options to make user-defined threads but which one we should use is exactly depends on the need there is no fixed rule for that.
We can use runnable interface when there is a need to extend another class so then if we use the first approach to create a thread then we can not extend that class we need because java doesn't support multiple inheritances so in that kind of scenario we

should use runnable interface that we can extend that class we want as the interface supports multiple inheritances.

12. How CurrentThread() works ?

⇨ CureenThread() is such a method by which we can get the reference of currently executing thread. And we can get the information on the main thread by this reference only.

13. How to set and get names of threads

⇨ We can set the names of the threads as we want by this method
Tread.currentThread.setName(“ Any string name ”)

⇨ We can get the current thread name by this method

Thread.CurrentThread.getName();

14. How to set priority to threads?

⇨ There are different priorities for different threads and with the help of this priorities java runtime system schedules the threads and if we need we can set our priorities as well with the help of this method =>
ThreadObjectname.setpriority(Thread.MIN_PRIORITY);

ThreadObjectname.SetPriority(Thread.MAX_PRIORITY);

ThreadObjectname.SetPriority(Thread.NORM_PRIORITY);

In java, we have thread priorities all in between 1 to 10 all the threads are having priorities 1 to 10 and by default, the priority is set to 5 that is normal priority

15. How JVM schedules the threads

⇨ JVM schedules the threads on pre-empting scheduling process in general and in pre-emptive scheduling all the higher priority processes will be executed at first until it goes to the waiting state or any higher priority thread comes into the picture.

⇨ Sometime JRE also schedule the threads by time-slicing mechanism

16. What are Demon threads

⇨ Demon threads are the service provider threads or invisible threads that are created when JVM creates the main thread and calls the main method in it. These threads provide services to the user threads.

- ⇨ All the demon threads are fully dependent on the user threads and its life depends on the user threads because if there is no user thread present in the application then JVM automatically terminates these demon threads.
- ⇨ It performs some background task when the program running
- ⇨ These types of threads are low priority threads.

17. Why the user-defined threads are non-daemon threads?

- ⇨ A newly created user thread is a non-daemon thread because all the child threads inherit the daemon status of their parents so if we create the user threads inside the main method then the user threads are also known as daemon threads by default.
- ⇨ We can change this status of daemon by void setDaemon(true / false)

```
Output ➔
daemon thread work
user thread work
user thread work
```

Note ➔ We should start the thread after setting Daemon or if we start it before setting it Daemon then it will throw IllegalThreadStateException

```
public class TestDaemonThread1 extends Thread{
 public void run(){
  if(Thread.currentThread().isDaemon()){//checking for daemon thread
   System.out.println("daemon thread work");
  }
  else{
  System.out.println("user thread work");
 }
 }
 public static void main(String[] args){
  TestDaemonThread1 t1=new TestDaemonThread1();//creating thread
  TestDaemonThread1 t2=new TestDaemonThread1();
  TestDaemonThread1 t3=new TestDaemonThread1();

  t1.setDaemon(true);//now t1 is daemon thread

  t1.start();//starting threads
  t2.start();
  t3.start();
 }
}
```

18. What are the methods for demon threads

⇨ To check whether a thread is a daemon or not ➔

Public Boolean isDaemon();

⇨ To set any thread as Daemon thread ➔

public void setDaemon("True / false");

19. What happens when an exception occurs on any thread?

⇨

20. What is a thread pool?

⇨ Java thread pool is a collection of some fixed size threads and from the thread pool threads are pulled out and assigned a job by the service provider and after completion of the job the thread comes back to the thread pool again.

The real-life application of thread pool is that in Jsp and servlet applications containers always create a thread pool to maintain or handle different server and user requests concurrently.

21. What is the thread group?

⇨

22. What is the runtime class in java?

⇨ Runtime class is a special type of class in java that is used to communicate with the java runtime system and java runtime class provides many methods for invoking garbage collectors getting information about memory usage etc.

⇨ In a java application, only one instance of runtime class can be created.

23. What Runtime().getRuntime() method does?

⇨ This method returns the singleton instance of runtime class.

24. What Runtime().getRuntime().exec() method does?

⇨ This method is used to open any other application at runtime by specifying the path of that particular application and we can perform many other operations as well like shutting down the system

25. What do you mean by Synchronisation?

⇨ In general, synchronization is a concept or capability to control multiple thread access to any shared resource at the same time to make data consistent in nature this feature is known as Synchronisation.

26. How does Synchronisation work in java?

⇨ Java provides us language-level support for synchronization this is the same concept as the operating system uses the concept of semaphore. But in java, we have the **monitors**. Here Monitors are special type objects that are used to create a mutually exclusive lock on each shared resource that can be accessed by multiple threads at the same time. Only one thread can acquire a monitor at a time. Another thread can not access the same monitor until the first one finishes the execution or releases the monitor. Until then all the other threads must wait for the first monitor.

27. How we can achieve Synchronization in java?

⇨ We can achieve synchronization in java by

- **By synchronized methods**
- **By synchronized statements**

28. What are synchronized methods?

⇨ If we declare any method with a synchronized keyword beforehand is known as synchronized method or thread-safe in simple word. Because the system will not allow multiple threads to access the synchronized methods at the same time when one thread is using any synchronized method others must wait**.** The synchronized method is used to create a lock on the object for multiple thread access.

```
class Table{
 synchronized void printTable(int n){//synchronized method
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }

 }
}

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}

public class TestSynchronization2{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
} }
```

```
O/P➔    5
        10
        15
        20
        25
        100
        200
        300
        400
        500
--------------------
```

Note ➔ Here we can see as this is a synchronized method so though it is shared method but when first thread was executing other waits after finishing first the next thread starts the execution until the n it waits.

But if we don't declare it as synchronized then we can able to see illegal access.

29. What do you mean by Synchronized block?

⇨ Synchronized methods and blocks both have the same functionalities. Where we want to make some of the lines of our method synchronized in that case we can use the synchronized block of statements.

```
⇨  void printTable(int n){
⇨    synchronized(this){//synchronized block  //Syntax
⇨      for(int i=1;i<=5;i++){                  // Synchronized(object reference)
⇨       System.out.println(n*i);                // {
⇨       try{                                   // ……
⇨        Thread.sleep(400);                    // }
⇨       }catch(Exception e){System.out.println(e);}
⇨      }
⇨    }
⇨  }//end of the method
⇨ }
```

Without specifying the whole method as synchronized only declare a synchronized block

30. What do you mean deadlock in java?

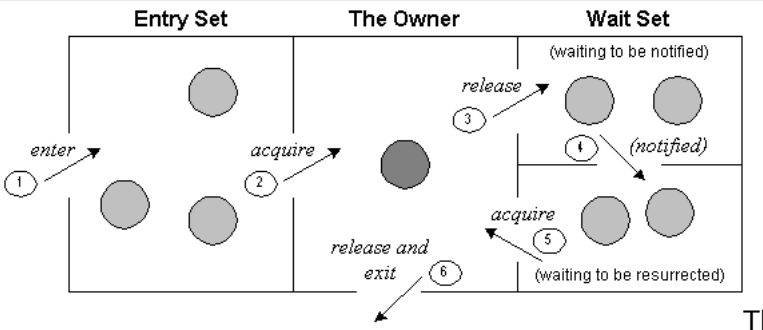
⇨ Deadlock in java happens when one thread is waiting for an object that is acquired by another thread and that thread is waiting for an object that is acquired by the first thread. Since the threads are waiting for each other means they are in a deadlock condition.

31. What do you mean by inter thread communication?

Inter-thread communication or **Co-operation** is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods of **Object class**:

- wait()
- notify()
- notifyAll()



The point to point explanation of the above diagram is as follows:

1. Threads enter to acquire lock.
2. Lock is acquired by on thread.
3. Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
4. If you call notify() or notifyAll() method, thread moves to the notified state (runnable state).
5. Now thread is available to acquire lock.
6. After completion of the task, thread releases the lock and exits the monitor state of the object.

32. Why wait() , notify() , notifyall() method defined in the object class not thread class?

⇨ Because it is related to lock mechanism and object has a lock.