**Thread**

**Q-1: What is Multithreading and explain its application areas?**

A: Executing several thread simultaneously where each thread is a separate independent part of the same program is called multithreading. The main important application area of multithreading are video games implementation, animation development, multimedia graphics etc.

**Q-2: What is advantage of Multithreading?**

A: The main advantage of multithreading is reduces response time and improves performance of the system.

**Q-3: Explain about Thread Scheduler?**

A:If multiple threads are waiting for getting the chance for executing then which thread will get chance first decided by Thread Scheduler. It is the part of JVM and its behavior is vendor dependent and we can’t expect exact output.Whenever the situation comes to multithreading the guarantee behavior is very- very low.

**Q-4: How many Life Cycle Stage of Thread**

**A:** 1- New

2-Runnable

3- Running

4- suspended/blocked

5- Dead

**Q-5: Syntax of Thread class**

**A:**A Thread class implement Runnable interface.

Thread class and Runnable interface comes from java.lang package.

public class **Thread extends** Object **implements** Runnable

**{**

**}**

**Q-6: Tell some methods of Thread class.**

**A:** 1-currentThread()

2-destroy()

3-getId()/setId()

4-getName()/setName()

5-getPriority()/setPriority()

6-join()/join(long millis)/join(long millis, int nanos)

7-run()

8-sleep(long millis)/sleep(long millis, int nanos)

9-start()

**Q-7: What is the use of priority of Thread.**

**A:** Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

**Q-8: If we are trying to set priority of a Thread as 100 what will happen?**

**A:** If we are trying to set priority of a Thread as 100 then we will not get any compile time error but at the runtime we will get Runtime exception IllegalArgumentException. Because the valid range of the Thread priority is (1-10) only.

**Q-9: If two threads having same priority then which thread will get chance first for execution?**

**A:** If two threads having same priority then which thread will get the chance first for execution decided by Thread Scheduler. It is the part of JVM and its behavior is vendor dependent and we can’t expect exact output.

**Q-10: If two threads having different priority then which thread will get chance first for execution?**

**A:** If two threads having different priority then the Thread which is having highest priority will get chance first for execution.

**Q-11: How many ways to create Thread. Write the code also. Which options are**

**better for creating Thread**

**A:**There are two ways of creating Thread

1. extends the Thread class
2. implements the Runnable interface

**Note:** by implements the Runnable interface is better options for creating thread because

1. you can implement Runnable and extend another class as well and implement another interface also

but by extend the Thread class there is no choice of extending another class.

public class **Demo** extends **c1** implements **Runnable**

{

}

1. If you implement Runnable interface then only run() method will be come in the sub class ( Runnable interface has only one method run()) but when you extend Thread class then all method(approx. 35) will be come in the sub class.

**ex-> by inherit the Thread class**

class Multi extends Thread

{

public void run()

{

for(int i=1;i<=5;i++)

{

try

{

System.out.println("thread is running...");

Thread.sleep(2000);

}

catch(Exception e)

{

e.printStackTrace();

}

}

}

public static void main(String args[])

{

Multi t1=new Multi();

t1.start();

}

}

**ex->by implementing the Runnable interface**

class Multi1 implements Runnable

{

public void run()

{

for(int i=1;i<=5;i++)

{

try

{

System.out.println("thread is running...");

Thread.sleep(1000);

}

catch(Exception e)

{

e.printStackTrace();

}

}

}

public static void main(String args[])

{

**Multi1 m1=new Multi1();**

**Thread t1 =new Thread(m1);**

t1.start();

}

}

**Q-12: Write the simple Thread class program.**

**A:** class MyThread implements Runnable

{

public void run()

{

Thread th = Thread.currentThread();

for(int i=1;i<=5;i++)

{

System.out.println(th.getName()+" : "+i);

try

{

Thread.sleep(500);

}

catch(Exception e)

{

e.printStackTrace();

}

}

}

}

public class Lab2 {

public static void main(String[] args) {

MyThread th = new MyThread();

Thread t1 = new Thread(th);

t1.setName("one");

t1.start();

Thread t = Thread.currentThread();

for(int i=100;i<=105;i++)

{

System.out.println(t.getName()+" : "+i);

try

{

Thread.sleep(500);

}

catch(Exception e)

{

e.printStackTrace();

}

}

}

}

**Q-13: Why wait(),notify() and notifyAll() method comes from Object class.**

**A:** These methods works on the locks and locks are associated with Object and not Threads. Hence, it is in Object class. As every **object** in java has only one lock(monitor) and **wait**(),**notify**(), **notifyAll**()are used for monitor sharing thats why they are part of **Object class** rather than Thread **class**. the**wait**() method will release the lock on the specified **object** and waits when it can retrieve the lock

**wait( ) ->**tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify( ).

**notify( )** **->** wakes up the first thread that called wait( ) on the same object.

**notifyAll( )** **->** wakes up all the threads that called wait( ) on the same object. The

highest priority thread will run first.

.

**Q-14: Explain the following method**

1. **yield()**
2. **join()**
3. **sleep()**

**A:1- yield()** method causes the current executing thread to pause execution and give the chance for waiting thread are same priority. If there is no waiting thread or all the remaining waiting thread have low priority then the same thread will get chance once again for execution. The Thread which is yielded when it will get chance once again for execution depends upon mercy of Thread scheduler.Public static native void yield();

**2-join()**If a Thread wants to wait until some other Thread completion then we should go for join() method.

Example: if a Thread t1 execute t2.join() ; then t1 will entered into waiting state until t2 Thread completion.

**3- sleep()**If a Thread don’t want to perform any operation for a particular amount of time then we should go for sleep() method.Whenever we are using sleep() method compulsory we should handle InterruptedException either by using try-catch or by using throws keyword otherwise we will get compile time error.

**Q-15: Is join() method is overloaded?**

**A:**  Yes join() method is overloaded method.

join()/join(long millis)/join(long millis, int nanos)

**Q-16: What is the difference b/w wait() and sleep().and where you have to use**

**wait() or where sleep().**

#### A: sleep

* It causes current executing thread to sleep for specific amount of time.
* Its accuracy depends on system timers and schedulers.
* It keeps the monitors it has acquired, so if it is called from synchronized context, no other thread can enter that block or method.
* If we call interrupt() method , it will wake up the sleeping thread.

synchronized(lockedObject) {

Thread.sleep(1000); // It does not release the lock on lockedObject.

// So either after 1000 miliseconds, current thread will wake up, or after we call

//t. interrupt() method.

}

#### wait

* It causes current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object
* It must be called from synchronized context i.e. from block or method.It means before wait() method is called,current thread must have lock on that object.
* It releases lock on the object on which it is called and added to wait list, so another thread can acquire lock on the object.

synchronized(lockedObject)

{

lockedObject.wait(); // It releases the lock on lockedObject.

// So until we call notify() or notifyAll() from other thread,It will

// not wake up

}

#### sleep vs wait:

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Wait** | **Sleep** |
| Class | It comes from Object class | It comes from Thread class |
| Synchonized | wait should be called from synchronized context i.e. from block or method, If you do not call it using synchronized context, it will throw IllegalMonitorStateException | It need not be called from synchronized block or methods |
| Calls on | wait method operates on Object and defined in Object class | Sleep method operates on current thread and is in java.lang.Thread |
| Release of lock | wait release lock of object on which it is called and also other locks if it holds any | Sleep method does not release lock at all |
| Wake up condition | until call notify() or notifyAll() from Object class | Until time expires or calls interrupt() |
| Static | wait is non static method | sleep is static method |

**EX:**

**class** Hello

{

**synchronized** **void** show()

{

Thread th = Thread.*currentThread*();

**for**(**int** i=0;i<5;i++)

{

System.*out*.println(th.getName()+" : "+i);

**try**

{

wait(1000);

//Thread.sleep(500);

}

**catch**(Exception e)

{

e.printStackTrace();

}

}

}

}

**class** MyThread **extends** Thread

{

Hello h = **null**;

MyThread(Hello h)

{

**this**.h=h;

}

**public** **void** run()

{

h.show();

}

}

**public** **class** Lab2

{

**public** **static** **void** main(String[] args) {

Hello h1 = **new** Hello();

MyThread th1 = **new** MyThread(h1);

MyThread th2 = **new** MyThread(h1);

th1.start();

th2.start();

}

}

**o/p->**

**1- if we will use Thread.sleep(1000) -:**

then it will only th1 will enter the synchronized method lock on th1 so until th1 do not complete task th2 will not execute , when th1 will execute completely after that th2 will be execute.

Thread-0 : 0

Thread-1 : 0

Thread-1 : 1

Thread-0 : 1

Thread-1 : 2

Thread-0 : 2

Thread-1 : 3

Thread-0 : 3

Thread-0 : 4

Thread-1 : 4

**1- if we will use wait(1000) -:**

then it will only th1 will enter the synchronized method and after execute it will release lock for 1 sec. so in that case th2 will get executed after th2 it will again release the lock for 1 sec. so th1 will get executed.

Thread-0 : 0

Thread-1 : 0

Thread-1 : 1

Thread-0 : 1

Thread-1 : 2

Thread-0 : 2

Thread-1 : 3

Thread-0 : 3

Thread-1 : 4

Thread-0 : 4

**Q-17: Why wait and notify method are called from synchronized block or method?**

**A:** The wait(), notify(), and notifyAll() methods should be called for an object only when the current thread has already locked the object's **lock**. That,s why they are keep in **synchronized block or method?**

**Q-18:In which method threads can release the lock?**

**A:** Once a Thread calls wait() method it immediately releases the lock of that object and then entered into waiting state similarly after calling notify() method Thread releases the lock but may not immediately. Except these three methods( wait(), notify(), notifyAll() ) method Thread never releases the lock anywhere else.

**Q-19: What is the difference between notify() and notifyAll()?**

**A:**  To give notification to the single waiting Thread. We use notify() method and to give notification to all waiting thread we use notifyAll() method.

**Q-20: What is Synchronization. How many ways to achieve Synchronization. And**

**which Options is better. Write the syntax also**

**A:** **(**1) when multiple threads are accessing the same object at a time.

you may get some inconsistent results may leads deadlock

(2) you can use Synchronization to avoid inconsistency

(3)Synchronization is the process of enabling lock of an object

(4) every object will have a lock and that lock is disabled by default

(5) when the lock of an object is enabled then only one thread is allow

to access that object at a time

**you can acheive Synchronization in two ways**

**(1)Method Level Synchronization**

**(2)Block Level Synchronization**

**Note:** When you synchronize a method, you are effectively synchronizing to the object itself. In the case of a static method, you're synchronizing to the class of the object. If you want to synchronization some part of a method to be synchronized to the object, then specify a synchronized block. If you use the synchronized keyword on the method declaration, it will synchronize the whole method to the object or class.

**Syntax of Method Level Synchronization ->**

class Table

{

**synchronized void printTable(int n)** //synchronized method

{

for(int i=1;i<=10;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(20);

}

}

public class **TestSynchronizationMethod**

{

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();

}

}

**Syntax of Block Level Synchronization:**

class Table

{

void printTable(int n)

{

**synchronized(this) //synchronized block**

{

for(int i=1;i<=5;i++){

System.out.println(n\*i);

try{

Thread.sleep(400);

}catch(Exception e){System.out.println(e);}

}

} // end of block

}//end of the method

}

**Q-21: What is the advantage of synchronized block over synchronized method?**

**A:**  If very few lines of the code required synchronization then declaring entire method as the synchronized is not recommended. We have to declare those few lines of the code inside synchronized block. This approach reduces waiting time of the Thread and improves performance of the system.

**Q-22: What is static Synchronization. Write the code also.**

**A:** If you make any static method as synchronized, the lock will be on the class not on object.

**class** Table

{

**synchronized** **static** **void** printTable(**int** n)

{

}

**}**

**Q-23: When we will use to create the thread. By “extending thread” and**

**“Implementing Runnable”.**

**A:** when multiple threads need to execute same task, then use Runnable interface. If multiple threads need to execute different tasks, then go for Thread class.

**Q-24: Difference between start() and run() method of Thread class?**

**A:** In the case of t.start() a new thread will be created and that thread is responsible to execute run() method and context switching is also there.

In the case of t.run() no new thread will be created and run() method will be executed just like normal thread and no context switching is there.

**In the Question- 4 program**

* If you are doing t1.start() then o/p

main : 100

one : 1

one : 2

main : 101

main : 102

one : 3

one : 4

main : 103

main : 104

one : 5

main : 105

* If you are doing t1.run() then o/p

main : 1

main : 2

main : 3

main : 4

main : 5

main : 100

main : 101

main : 102

main : 103

main : 104

main : 105

**Q-25: can be call run() method directly . if we are not invoking start() method and directly invoking run() method and then what will happen.**

**A:** yes you can call.  If you called run directly, then it would be like an ordinary method call and it would run in the context of the current thread instead of the new one.

**Q-26: Expalin the following case:**

**Case 1-if we are not overriding the run() method**

If we are not overriding run() method. Then Thread class run() will be executed with has empty implementation and hence we would not get any o/p.

class Lab2 extends Thread

{

public static void main(String args[])

{

Lab2 t1=new Lab2();

t1.start();

}

}

**o/p->** no o/p come

**Note:** it is highly recommended to override **run()** to define our job.

**Case 2-if we are overloading the run() method->**

Overloading of the run() is possible but Thread class start() will always call no argument run() only. but the other run() we have to call explicitly just like a normal method call.

**Case 3-if we override the start() method->**

If we override start() then start() will be executed just like a normal call and no new Thread will be created.

Ex-1:

**class** Lab2 **extends** Thread

{

**public** **void** start()

{

System.*out*.println("start");

}

**public** **void** run()

{

System.*out*.println("run");

}

**public** **static** **void** main(String args[])

{

Lab2 t1=**new** Lab2();

t1.start();

}

}

o/p-> start

Ex-2:

**class** Lab2 **extends** Thread

{

**public** **void** start()

{

**super**.start();

System.*out*.println("start");

}

**public** **void** run()

{

System.*out*.println("run");

}

**public** **static** **void** main(String args[])

{

Lab2 t1=**new** Lab2();

t1.start();

}

}

o/p-> start

run

**Q-15: What is thread pool. Why should we use thread pool in java.**

**A:** **Java Thread pool** represents a group of worker threads that are waiting for the job and reuse many times.

In case of thread pool, a group of fixed size threads are created. A thread from the thread pool is pulled out and assigned a job by the service provider. After completion of the job, thread is contained in the thread pool again.

**java.util.concurrent.Executors** provide implementation of **java.util.concurrent.Executor** interface to create the thread pool in java.

#### Advantage of Java Thread Pool

#### Better performance It saves time because there is no need to create new thread.

#### Real time usage

It is used in Servlet and JSP where container creates a thread pool to process the request.

**Q-16: There are three threads T1, T2, T3? How to ensure sequence T1, T2, T3 in java.**

**A:** by using join because join wait a thread for die.

class TestJoinMethod1 extends Thread

{

public void run()

{

Thread t = Thread.currentThread();

for(int i=1;i<=5;i++){

try

{

Thread.sleep(500);

System.out.println(t.getName()+" "+i);

}

catch(Exception e)

{

System.out.println(e);

}

}

}

public static void main(String args[]) throws InterruptedException

{

TestJoinMethod1 t1=new TestJoinMethod1();

TestJoinMethod1 t2=new TestJoinMethod1();

TestJoinMethod1 t3=new TestJoinMethod1();

t1.setName("One");

t2.setName("Two");

t3.setName("Three");

t1.start();

t1.join(); / / t1.join(1500); then t1 will execute three times.

t2.start();

t2.join();

t3.start();

t3.join();

}

}

**Q-17: What is the difference b/w class level lock and Object level lock.**

**A:Object Level Lock:**

If a Thread wants to execute a synchronized method first it has to get the lock of the object. Once a Thread got the lock then it is allow to execute any synchronized method on that object. After completing synchronized method execution Thread releases the lock automatically.While a Thread executing synchronized method on the given object the remaining Threads are not allow to execute any synchronized method on that object simultaneously. But remaining Threads are allow to execute any non-synchronized method simultaneously. (Lock concept is implemented based on object but not based on method.)

 This can be done as below

1-public class DemoClass

{

**public synchronized void demoMethod(){}**

}

2-public class DemoClass

{

public void demoMethod(){

**synchronized (this)**

{

//will lock on current object

}

}

}

3-public class DemoClass

{

private final Object l1 = new Object();

public void demoMethod(){

**synchronized (l1)**

{

//will lock on specified obj1 object

}

}

}

**Class Level Locking->**

if a Thread wants to execute static synchronized method that Thread has to get class level lock once a Thread got class level lock then only it is allow to execute static synchronized method.

While a Thread executing any static synchronized method then remaining Threads are not allow to execute any static synchronized method of the same class simultaneously. But the remaining Threads are allow to execute the following method simultaneously

This can be done as below

1-public class DemoClass

{

**public synchronized static void demoMethod(){}**

}

2-public class DemoClass

{

public void demoMethod(){

**synchronized (DemoClass.class)**

{

}

}

}

3-public class DemoClass

{

**private final static Object lock = new Object();**

public void demoMethod(){

**synchronized (lock)**

{

}

}

}

**Q-18: What is volatile variable in java.**

**A:** volatile is used to indicate that a variables,s value will be modified by different thread. By making a variable volatile using volatile keyword in Java, application programmer ensures that  volatile variable will be read from the computer's main memory, and not from the CPU cache, and that every write to a volatile variable will be written to main memory, and not just to the CPU cache.

**Q-19: What happens when an Exception occurs in a thread?**

**A:** when an exception occurs in a thread and if we will not caught then thread will die.so when a thread is about to terminate due to an uncaught exception, jvm

the Java Virtual Machine will query the thread for its **UncaughtExceptionHandler** using **Thread.getUncaughtExceptionHandler()** and will invoke the handler's uncaughtException method, passing the thread and the exception as arguments.

**Ex:**

public class MyApp {

...

public MyApp() {

Thread.setDefaultUncaughtExceptionHandler(

new Thread.UncaughtExceptionHandler() {

@Override public void uncaughtException(Thread t, Throwable e) {

System.out.println(t.getName()+": "+e);

MyWorker worker = new MyWorker();

worker.start();

}

});

MyWorker worker = new MyWorker();

worker.start();

}

public static void main(String[] args) {

MyApp te = new MyApp();

}

}

**Q-20: What happens if a thread throws an Exception inside synchronized block?**

**A:** no matter how you exist synchronized block, either normally by finishing execution or sudden by throwing exception, thread releases the lock it acquired while entering that synchronized block.

**Q-21: What happened if i start a thread twice times.**

**t1.start ();**

**t1.start ();**

**A:** it will shows exception java.lang.IllegalThreadStateException

**Q-22:How a thread can interrupt another thread?**

**A:**  A Thread can interrupt another Thread by using interrupt() method.

**Q-23: What is DeadLock? Is it possible to resolve DeadLock situation?**

**A:**  If two Threads are waiting for each other forever such type of situation is called DeadLock.

For the DeadLock, there are no resolution techniques but prevention techniques are available . By using Synchronization we can prevent deadlock.

**Q-24: How to detect the deadlock threads in java.**

**A:** You can detect the threads which have entered into deadlock condition and also you can retrieve the details about them. This can be done using **ThreadMXBean** interface of  **java.lang.Management**  package.

**Q-25: What is Inter Thread Communication .How achieve it**

**A:** if two or more thread is communicating to each other is known Inter Thread Communication. we can achieve by wait() ,notify(), notifyAll()

**Q-26: How two thread will communicate with each other?**

**A:**  Two Threads will communicate with each other by using wait(), notify(), notifyAll() methods.

**Q-27: What is daemon thread**

**A:** (1)there are two type of threads

(A) user thread (B)daemon thread

(2)Daemon thread is a service provider thread.

(3) it provide service to user thread

(4) its life depends on the user thread i.e when all user thread dies , JVM

terminates the thread automatically

(5)daemon thread is also called "Garbage Collector" thread because it runs

alternate in the background doing garbage collection operation

Ex:

public class CDaemonThread extends Thread

{

public static void main(String[] args)

{

CDaemonThread t = new CDaemonThread();

**t.setDaemon(true);**

// When false, (i.e. when it's a user thread), the Worker thread

// continues to run.

// When true, (i.e. when it's a daemon thread), the Worker thread

// terminates when the main thread terminates.

t.start();

}

}

**Q-28: Is main thread is Daemon or non-daemon?**

**A:**  By default main thread is always non-daemon nature.

**Q-29: Once we created a new thread is it daemon or non-daemon.**

**A:**  Once we created a new Thread, The Daemon nature will be inheriting from parent to child. If the parent is Daemon the child is also Daemon and if the parent is non-daemon then child is also non-daemon.

**Q-30:Explain about Thread group?**

**A:**. Every Java thread is a member of a thread group. Thread groups provide a mechanism for collecting multiple threads into a single object and manipulating those threads all at once, rather than individually. For example, you can start or suspend all the threads within a group with a single method call. Java thread groups are implemented by the ThreadGroup api class in the java.lang package.

**Q-31: Can a variable can be synchronized.**

**A:** A variable can never be synchronized only method and block can be synchronized.