Multi Threading

1. **What is a thread?**
   Thread refers to an alternative sequence of execution. It refers to a separate stack which is created in the stack segment.

2. **What is a process?**
   A process is a program which has been loaded on to the RAM and is under execution.

3. **What is the difference between process and thread?**

Process	Thread
Process refers to a program which has been loaded on to the RAM and is in execution.	Thread refers to an alternative sequence of execution
Scheduling of a process is under the control of O.S	Scheduling of threads is under the control of thread scheduler
Switching between processes is performed by OS	Switching between threads is performed by thread scheduler
A process is a heavy weight entity. Internally, it would contain either a single thread or multiple threads	Thread is a light weight entity

4. **What are the advantages of threads?**
   It enables efficient utilization of CPU time.

5. **What is meant by multi threading?**
   Multi threading refers to the process of creating multiple stacks on the stack segment and loading multiple independent sub tasks into the stack segments and hence creating multiple sequence of execution. This would result in efficient utilization of CPU time.

6. **Who manages multi tasking?**
   Operating System.

7. **How can a programmer create his own thread?**
   i) By creating an object of the thread class
   ii) By creating an object of the class which implements the **Runnable interface** and
   giving this object as a parameter to the Thread class object.

8. **What is the name of the thread that executes the main method called as?** Main thread.

9. **What is a default priority of main thread?**
   Default priority of main thread is 5.

10. **Can we change priority of main thread?**
    Yes. Using setPriority() method.

11. **Which is the default method that the main thread executes?**
    Main thread always executes main() method.

12. **Can we make the main thread to begin execution from any method of our own choice other than the default method it executes?**
    No.

13. **Can we change the name of main thread?**
    Yes. Using setName() method.

14. **Who creates the main thread?**
    JVM creates main thread and hands it over to the thread scheduler.

15. **What is a lock?**

Lock is a mechanism available in java to ensure that only a single thread makes use of the shared resource. Once any thread applies a lock on a resource, then other threads would not be able to access that resource until and unless the thread which has applied the lock releases it. Lock can be applied on a shared resource in java using synchronized keyword.

16. **Does the priority of the thread matter during scheduling?**

If the application is executing on a preemptive OS, then the priority matters. In such a case, the higher priority thread tends to get more CPU cycles than a lower priority threads.

If the application is executing on a non-preemptive OS, then irrespective of the priority all threads tend to get almost the same CPU time.

17. **What is the problem associated with single threaded programs?**

In a single threaded program, there is a possibility that the CPU cycle would not be efficiently utilized.

18. **What is the advantage of single threaded program?**

Multiple stacks on the stack segment need not be created. The thread scheduler need not get into the activity of scheduling multiple threads. In general, the burden on the thread scheduler would be relatively less.

19. What are the two ways of creating Thread?

i) By extending the Thread class and creating an object of Thread class.
ii) By implementing the Runnable interface and giving this object as a parameter to the Thread class object.

20. **Which is a better way to achieve multithreading?**

Implementing the Runnable interface and giving this object as a parameter to the Thread class object is the better way of creating a

thread since the programmer would have an option of extending another class if required.

21. **Is Runnable a class or interface?**
Runnable is an Interface.

22. **Is Thread a class or interface?**
Thread is a class.

23. **What is the relationship between Thread and Runnable?**
Thread class implements Runnable interface.

24. **Why should we override the run () method of the Thread class?**
Because the run() method which is provided by the Thread class is an empty run() method. It is the duty of the programmer to override an empty run() method and provide a suitable body as per the project requirements.

25. **Why should we start a Thread?**
When the thread is created, it would be in the **new state**. By applying start() method on thread, the thread can be shifted from new state to **runnable state** so that the thread scheduler can schedule it to the **running state**.

26. **Which state would the Thread be when it is created?**
New state.

27. **What are the components of a program on which lock can be applied?**
Lock can be applied on,
i) method
ii) variable
iii) blocks

28. **Can we start a Thread twice?**

No because **IllegalThreadStateException** occurs.

29. **What happens if we call start () method on a Thread twice?**

IllegalThreadStateException occurs.

30. **Can we explicitly call a run () method on a Thread?**

The programmer should not call run() method since it would prevent multithreading functionality of the thread scheduler. Threads get executed in the same sequence in which their run() methods are called. It goes against the purpose/ethics of multithreading.

31. **What happens if we explicitly call run () method on a Thread?**

The programmer should not call run() method since it would prevent multithreading functionality of the thread scheduler. Threads get executed in the same sequence in which their run() methods are called. It goes against the purpose/ethics of multithreading.

32. **Ideally who must call the run () method of a Thread?**

Thread Scheduler.

33. **How do we verify if a Thread is alive?**

Using **isAlive()** method.

34. **When the Thread is executing the run () method in which state it is present?**

It would be present in **running state**.

35. **In which state should the Thread be present in order to execute the run () method?**

It should be readily available in **runnable state**.

36. **Which are the possible states that the Thread may go to while it is in the running state?**

From the running state, the thread may go to

i) Runnable state by executing yield() method

ii) Sleep state by executing sleep() method,
iii) Blocked state when the required resource is not available ,
iv) Dead state when the execution of the thread is completed or if interrupt() method is executed,
v) Wait state by executing wait()

37. **When does the Thread enter sleep state?**
When the sleep() method is executed.

38. **When does the Thread enter wait state?**
When the wait() method is executed.

39. **When does the Thread enter blocked state?**
When the resource is not currently available because some other thread has applied a lock on that resource, then the thread would enter into blocked state.

40. **Which keyword is used to apply a lock on a resource?**
"synchronized" keyword would be used.

41. **Is the Thread alive when it is created?**
No.

42. **How can we bring the Thread to life?**
By executing a start() method.

43. **When does the Thread enter dead state?**
Thread would enter into dead state when the thread completes its execution or if the interrupt() is executed.

44. **How many Threads can be in the running state at any given point of time?**
Only one thread can be present in the running state.

45. **Does the Thread class provide an implementation to the abstract run () method of the Runnable interface?**

Yes. It provides empty body. It is for the programmer to provide body for run() method based on the project requirements.

46. **What are the possible states from which a Thread can enter Runnable state?**

i) sleep state to runnable state after sleep duration is completed.
ii) new state to runnable state using start() method.
iii) blocked state to runnable state when the previously unavailable resource becomes available.
iv) wait state to runnable state when notify() or notifyAll() method is executed by another thread.
v) running state to runnable state using yield() method.

47. **Can we have multiple Threads in the new state?**

Yes.

48. **Can we have multiple Threads in the runnable state?**

Yes.

49. **Can we have multiple Threads in the running state?**

No.

50. **Can we have multiple Threads in the sleep state?**

Yes.

51. **Can we have multiple Threads in the blocked state?**

Yes.

52. Can we have multiple Threads in the wait state?

Yes.

53. **Can we have multiple Threads in the dead state?**

Yes.

54. **How can we manually kill a Thread?**

Using interrupt() method.

55. **Which operator is used to create a Thread of execution?**
"new" operator would be used.

56. **When a Thread is killed are all its lock relinquished (released)?**
Yes.

57. **Does a Thread relinquish locks when it enters the sleep state?**
No.

58. **Does a Thread relinquish locks when it enters the wait state?**
No.

59. **Does a Thread relinquish locks when it enters the blocked state?**
No.

60. **What is the difference between yield() and sleep()?**
yield() method takes a thread from the running state to the runnable state whereas sleep() method takes the thread from the running state to the sleep state.

61. **What is the difference between wait() and sleep()?**
wait() method takes the thread from running state to the wait state whereas sleep() method takes a thread from running state to the sleep state.

62. **Can we achieve synchronization effect by using join () method?**
Yes. But it is not recommended.

63. **How does a Thread come out of a sleep state?**
When the sleep duration is completed, thread comes out of sleep state and enters into runnable state.

64. **In which state can a Thread be interrupted?**
Except new state and dead state, if the thread is in any other state, it can be interrupted.

65. **How does a Thread come out of a block state?**

Whenever the required resource which was previously unavailable becomes available, then a thread comes out of the block state.

66. **How does a Thread come out of a wait state?**

When some other thread calls notify() or notifyAll() method, then the thread comes out of the wait state.

67. **How does a Thread come out of a dead state?**

Thread cannot come out of the dead state.

68. **How does a Thread come out of a runnable state?**

When the thread scheduler decides to schedule the thread.

69. **What are the possible reasons for the Thread to move out of running state?**

i) If the thread has been executed for a longer duration, then the thread scheduler would execute the yield() method and send it back to the runnable state.

ii) If the thread completes its execution, it would move to the dead state.

iii) If the thread is interrupted using interrupt() method, it would move to the dead state.

iv) If sleep() method is encountered, it would move to the sleep state.

v) If wait() method is encountered, it would move to the wait state.

vi) If the required resource is not available, then the thread would enter into blocked state.

70. **When should we synchronize a resource?**

If the resource is a shared resource where only a single thread must be able to access the resource at a time, then the resource should be synchronized.

71. **What is the advantage of synchronization?**

Shared resource can be efficiently used without race condition.

72. **What is the disadvantage of synchronization?**

It goes against the purpose of multithreading.

73. **How do we make one Thread wait for the other Thread?**

Using join() method.

74. **Is a Thread alive when it is in the new state?**

No.

75. **Is a Thread alive when it is in the running state?**

Yes.

76. **Is a Thread alive when it is in the runnable state?**

Yes.

77. **Is a Thread alive when it is in the sleep state?**

Yes.

78. **Is a Thread alive when it is in the block state?**

Yes.

79. **Is a Thread alive when it is in the wait state?**

Yes.

80. **Is a Thread alive when it is in the dead state?**

No.

81. **Is interrupting a Thread equal to killing a Thread?**

Yes.

82. **What is a deadlock?**

Deadlock is a situation in a multithreaded programming where multiple threads would be stuck in the blocked state, mutually waiting for one another to release the resource in order to proceed with their execution. In the process, none of the threads would release the resource and hence would not be able to come out of the blocked state. Because of this,

execution of program would reach a standstill. This is called as deadlock.

83. **When does a deadlock occur?**

Deadlock occurs whenever there is a cyclic dependency for shared resources between multiple threads.

84. **How can a deadlock be prevented?**

i) By ensuring that cyclic dependency does not occur between threads

ii) By assigning different priorities to the threads. However this approach works only on pre-emptive OS and it is not a widely recommended choice.

85. **What is daemon Thread?**

A daemon thread is a thread that keeps on executing in the background at fixed intervals of time. They do not complete their execution until and unless the important thread of the process completes its execution. Daemon threads would always execute subsidiary activities.

86. **What role does the daemon Thread play?**

It performs subsidiary activities such as releasing the acquired resources, cleaning up memory i.e. garbage collection, spell check , auto save etc.

87. **How do we make a Thread as daemon Thread?**

i) setDaemon() must be made as true.

ii) The activity that has to be executed by daemon thread must be placed within an infinite loop.

iii) It must be given a low priority.

88. **Should the priority of daemon Thread be low or high?**

It should be low priority.

89. **Can you name a few daemon Threads?**

Examples of daemon threads include Garbage collection thread, spell check thread, auto save thread etc.

90. **What is the difference between deadlock and livelock?**
In deadlock, the activity would not get completed because the threads would stuck in the blocked state.
In livelock, even though the threads are not in blocked state yet the program would not proceed with the execution and comes to a stand still due to faulty coding.

91. **Which method can be used to enable interaction between Threads executing the different subtasks?**
"synchronized" key word, wait(), notify() can be used.

92. **Which facility can be used to enable interaction between Threads executing the same subtask?**
"synchronized" keyword can be used.

93. **What is the difference between notify () and notifyAll () methods?**
notify() method would take one thread from the wait state to the runnable state, whereas notifyAll() would take all the threads present in the wait state to the runnable state.

94. **Which method can be used to set a name to a Thread?**
setName() method can be used.

95. **Which method can be used to obtain the name of Thread?**
getName() method can be used.

96. **Can the overridden run() method throw exception object? Why?**
No because it changes the signature which is not permitted.

97. **Does C support multithreading?**
No.

98. **What happens if an exception occurs in one of the Thread of a multithreaded program?**

If an exception occurs in the multithreaded environment then exception handling would be performed only on that stack in which exception occurred. The other threads (other stacks) would proceed with their execution as normal.

99. **How do we verify if the Thread is a daemon Thread?**

Using isDaemon() method.

100. **Does C++ support multithreading?**

No.

101. **How many stacks would be present in the stack segment by default?**

One stack would be present by default called as main stack.

102. **How can we create extra stacks in the stack segment?**

By creating the objects of the thread class.

103. **Who manages the multiple stacks on the stack segment?**

Thread scheduler.

104. **From which version of java was Thread pool feature made available?**

From jdk 1.7 onwards this feature made available.

105. **What happens if a start () method is invoked on a Thread?**

The thread moves from the new state to the runnable state. The thread scheduler

becomes aware that there is a thread available in the runnable state and can be

scheduled.

106. **What is a role of volatile keyword?**

(Refer class notes)

107. **What is the datatype of parameter provided to the sleep() method?**

Long type.

108. **If a Thread creates a new Thread then what will be the priority of newly created Thread?**

The child threads priority would be the same as that of its parent thread.

109. **Does the JVM have to wait for all the daemon Threads to complete to wind up its operation?**

Yes.

110. **What are the four variants of constructors present in the Thread class?**

Thread t=new Thread(this);
Thread t=new Thread ();
Thread t=new Thread(job);
Thread t=new Thread("BANKING");

111. **How many locks can we apply on an object?**

Only one lock can be applied on an object. After the lock is relinquished another lock can be applied by another thread.

112. Can a class contain both synchronized and non-synchronized methods?

Yes.

113. Can a class contain both synchronized and non-synchronized variables?

Yes.

114. **Can a class contain both synchronized and non-synchronized blocks?**

Yes.

115. **Can we apply synchronized keyword on a class?**

No.

116. **In which package are the interfaces and classes related to Threading available?**

java.lang package.

117. **In which version of java was the Callable interface introduced?**

From JDK 1.5 onwards.

118. What is the difference between Runnable's run () and Callable's call () methods?

Runnable's run ()	Callable's call ()
Needs to implement run() method	Needs to implement call() method
Runnable interface is present from JDK 1.0 onwards	Callable interface has been introduced from JDK 1.5 onwards
Run() method does not return any value	call() method returns a value
It cannot throw checked exception	It can throw checked exception
Runnable use execute() method to put in the task queue	Callable use submit() method to put in the task queue

119. **Can the run() method return the result of execution?**

Since in the signature the return type of run is void, the run() method cannot return the result of execution.

120. **What is meant by Thread safety?**

Thread safety refers to a phenomenon in multithreaded programming in which a shared resource would be utilized by multiple threads such that improper accessing of the shared resource does not occur. Also ensures race condition does not occur. Thread safety can be achieved using "synchronized", join() , immutable classes and by using thread safe classes.

121. **What is meant by race condition in java?**

(Refer class notes)

122. **How can we share data between two Threads?**

Using shared object concept.

123. **Are wait (), notify () and notifyAll () methods belong to the Thread class?**

No, they belong to Object class.

124. **To which class does wait (), notify () and notifyAll () methods belong to?**

They belong to Object class.

125. **Why are wait (), notify () and notifyAll () methods members of Object class rather than Thread class?**

That is because in java, locks are not applied on threads. Rather they would be applied on Objects.

126. **What is a Thread local variable?**

(Refer class notes)

127. **Why should wait () and notify () methods be called from synchronized block?**

It is the expectation from java API. If the "synchronized" key word not associated with wait() and notify() then IllegalMonitorStateException occurs.

Basically wait() and notify() should be placed under "synchronized" keyword to avoid race condition.

128. **What is a Thread group?**

(Refer class notes)

129. What is the advantage of using Thread group (Thread pool)?

(Refer class notes)

130. **How do we verify if a Thread holds a lock or not?**

Using hasLock() method. If the thread holds the lock then the method would return true otherwise false would be return.

131. **There are three Threads t1, t2 and t3? How do you ensure the sequence t1, t2 and t3 in java?**

It can be ensured using join() method.

132. **What does the yield () method of Thread class do?**

When yield() method is encountered the thread which is running state would be sent back to runnable state.

133. **What is a semaphore in java?**

Semaphore is a collection of statements which have to be executed by a single thread at any given point of time. It can be achieved using "synchronized" keyword.

134. **What is a monitor in java?**

Monitor is also called as semaphore. It is a collection of statements which have to be executed by a single thread at any given point of time. It can be achieved using "synchronized" keyword.

135. **What happens if all the Threads in the Thread pool are busy and still a new task is submitted?**

RejectedExecutionException would occur.

136. **What is meant by busy spin in multithreading?**

Busy spin is a phenomenon in multi-threaded environment where loop of threads would be continuously waiting for another to complete its execution so that they can start with their execution.

Busy spin leads to a wastage of CPU cycles.

137. **Mention any three multithreading practices?**

1.Threads should be given such names which clearly represents the activities that they perform, rather than calling the threads as t1,t2,t3 etc. It is better to name them as typing, spellcheck, autosave.
2.Avoid applying locks (Using synchronized) as much as possible, because locks go against the principle of multithreading.
3.If lock has to be inevitably applied then it should be done using only synchronized keyword.

138. **Is there any means by which a programmer can forcefully take a Thread to the running state?**

System.gc() method.

139. **What is the difference between preemptive scheduling and time slicing?**

preemptive scheduling	**time slicing** (non pre-emptive scheduling)
It is priority-based scheduling. It gives importance to the priority of a thread.	It is non-priority-based scheduling. It does not give importance to the priority of a thread.
Higher priority threads tend to get more CPU time.	All threads almost get equal CPU time irrespective of their priority.
Dead lock condition does not occur.	Dead lock condition occurs.
Unix OS.	Windows OS.

140. **Can we make the user Thread as daemon Thread after starting it?**

NO.

141. **Can we make the user Thread as daemon Thread before starting it?**

Yes.

142. **What is shutdown hook?**

JVM would perform cleanup operations at the end of program execution using "shutdownhook" thread. After execution of this thread the JVM stops its execution and gets popped from the stack.

143. **What is starvation?**

Starvation refers to a situation in multi-threaded environment where a thread in the block state which is waiting for the resource would never gets the resource because another thread which is holding the resource would not relinquish it.

144. Does the priority of a Thread play a vital role in its scheduling?

Priority of a thread matters only in case of pre-emptive OS. However, in non-pre-emptive OS priority of a thread does not play any role.

145. Name the methods available in Runnable interface?

public void run() method.

146. Name the methods available in Thread class?

sleep(), start(), yield(), join(), isAlive() etc.

147. Name the methods available in Object class?

toString(), hashCode(), notify(), notifyAll() etc.

148. **Name the methods used for inter-Thread communication**?

wait(), notify(), notifyAll().

149. What are the values of maximum priority, minimum priority and normal priority in java?

Minimum priority – 1
Maximum priority – 10
Normal priority – 5

150. How many Threads can access a monitor at a time?

Only one thread.

151. What is the priority of garbage collector Thread?

Garbage collector thread would be having low priority because it is daemon thread.