Multithreading

Q1) What is the difference between Thread and Process?

Q2) What is Multithreading?

Executing more than one thread simultaneously is called as Multithreading.

Q3) What is the difference between Multithreading and Multiprocessing?

Multiprocessing	Multithreading
Process is also called as a task which is an executing instance of a program.	A thread is a lightweight subprocess and executed within the process.
Executing more than one process simultaneously is called as Multiprocessing	Executing more than one thread simultaneously is called as Multithreading
Multiple processes running in the system will utilize different memory areas.	Multiple threads running in the process will utilize same memory area.
Data sharing between processes is usually more expensive.	Data sharing between threads is usually less expensive.
Context switching between processes is usually more expensive.	Context switching between threads is usually less expensive.

Q4) What is main Thread?

The main thread is the primary thread of execution in a Java application. It is automatically created by the JVM when the application starts and is responsible for executing the main method.

Q5) What is Finilizer Thread?

A "finalizer thread" is a thread used by the Java Virtual Machine (JVM) to run the finalize() method of objects before they are garbage collected

Q6) How can I define custom Thread?

You can either extend the Thread class or implement the Runnable interface.

Q7) Which is the best or recommendable option to define custom thread and why?

Implementing Runnable interface, because it provides more flexibility and reusability of code.

Q8) Is it possible to call start() method on running thread?

No, it is not possible to call the start() method on a running thread. Once a thread has been started, it can't be started again.

Q9) Can I overload the run() method in thread class?

No, you cannot overload the run() method in the Thread class.

Q10) Which method will be called by th JVM when I overload run() method?

When you overload the `run()` method, the JVM will only call the `run()` method that matches the signature of the `Runnable` interface or `Thread` class. Overloaded `run()` methods with different parameters are not called by the JVM and will not affect the thread's execution.

Q11) Can I define the run() method as abstract?

No, you cannot define the run() method as abstract.

Q12) Can I define the run() method as static?

No, you cannot define the run() method as static

Q13) Can I define the run() method as synchronized?

No, you cannot define the run() method as synchronized

Q14) Can I define a thread class without overriding the run () method?

Yes, you can define a thread class without overriding the run() method.

Q15) How do I pause thread execution for specified amount of time?

You can use the Thread.sleep(milliseconds) method.

Q16) Can I specify the priority of the thread less than 0 or more than 10?

No, you cannot specify the priority of a thread less than 0 or more than 10.

Q17) What is object locking?

Object locking is a mechanism used to control access to an object's methods or critical sections of code, ensuring that only one thread can execute a synchronized block or method on that object at a time.

Q18) Which object will be locked when I am using method level synchronization with instance methods?

When using method-level synchronization with instance methods, the instance of the class (i.e., `this`) will be locked.

Q19) Which object will be locked when I am using method level synchronization with static methods?

When using method-level synchronization with static methods, the class object (i.e., ClassName.class) will be locked.

Q20) Which object will be locked when I am using block level synchronization?

The object that is being synchronized on, which is specified in the synchronized block, will be locked. For example:

```
synchronized (myObject) {
// code here
}
```

In this case, myObject will be locked.

Q21) Can I call the wait() method from synchronized context with the unlocked object?

No, you cannot call the wait() method from a synchronized context with an unlocked object.

Q22) Why should I call wait() method with locked object?

you should call the **wait()** method with a locked object because **wait()** can only be called from within a synchronized block or method, where the thread holds the object's lock.

Q23) What is the reason for providing wait() or notify() or notifyAll() methods in object class?

The wait(), notify(), and notifyAll() methods are provided in the Object class to allow threads to communicate and synchronize with each other, enabling them to wait for or notify other threads about changes to shared resources.

Q24) What is deadlock?

So here both processes are requesting for unavailable resources, both will placed in

BLOCKED state and both will not come out of BLOCKED state. This situation is called as

Deadlock.

Q25) What is the use of join() method?

This method will be used to join one thread at the end of another thread.

- In Main thread -> t1.join()

  Main thread will be joined at the end of t1.

- In t1 thread -> t2.join()

  t1 thread will be joined at the end of t2.

Q26) What is daemon thread?

Daemon threads are generally service thread which will run to provide service to user

threads.

Q27) How can I check whether a thread is daemon or not?

public final boolean isDaemon() : to verify whether the thread is DAEMON or not.

Q28) How can I mark user thread as a daemon thread?

public final void setDaemon(boolean daemon): to mark the thread as DAEMON thread.

Q29) Can I mark thread as daemon thread, if it is started?

No, you cannot mark a thread as a daemon thread after it has been started.

Q30) What is the difference between sleep() and wait() method?

sleep():

- Belongs to the Thread class.
- Causes the current thread to pause execution for a specified amount of time.
- Does not release any locks held by the thread.
- It is a static method and can be called from any thread.
- Example: Thread.sleep(1000); pauses the thread for 1000 milliseconds.

 wait():

- Belongs to the Object class.
- Causes the current thread to release the lock on the object and wait until another thread invokes notify() or notifyAll() on the same object.
- It must be called from within a synchronized block or method.
- Used in inter-thread communication to coordinate actions between threads.
- Example: synchronized(obj) { obj.wait(); } makes the thread wait until it is notified

Q31) What is the use of volatile keyword in java?

the **volatile** keyword is used to mark a variable so that its value is always read directly from and written to the main memory, rather than from a thread's local cache.

Q32) What is race condition in Java?

A **race condition** in Java occurs when two or more threads try to access and modify shared data at the same time, leading to unpredictable and incorrect results.

Q33) What is thread pool?

A **thread pool** is a collection of pre-initialized threads that are ready to perform tasks, rather than creating new threads each time a task is requested.

Q34) How to create thread pool in java?

Creating a thread pool in Java can be done using the ExecutorService framework, which is part of the java.util.concurrent package.

ExecutorService threadPool = Executors.newFixedThreadPool(numberOfThreads);

Q35) What is Producer Consumer problem in Java?

The Producer-Consumer problem is a classic concurrency issue where two types of threads share a common buffer:

- **Producers** create data and put it into the buffer.
- **Consumers** take data from the buffer and use it.