1. What do you mean by Multithreading? Why is it important?

Ans: [Multithreading in Java is a process of executing multiple threads simultaneously to achieve concurrency and maximum utilization of CPU1](https://www.guru99.com/multithreading-java.html)[2](https://www.javatpoint.com/multithreading-in-java). [A thread is a lightweight sub-process, the smallest unit of processing, that can run in parallel with other threads within the same process2](https://www.javatpoint.com/multithreading-in-java)[3](https://www.geeksforgeeks.org/multithreading-in-java/). Multithreading is important because it can improve the performance, responsiveness, and scalability of a program by allowing it to perform multiple tasks at the same time, such as:

* Handling multiple user requests: A server application can use multithreading to handle multiple client requests concurrently, without blocking or waiting for each request to finish. [This can increase the throughput and efficiency of the server](about:blank)[1](https://www.guru99.com/multithreading-java.html).
* Performing background tasks: A GUI application can use multithreading to perform long-running or intensive tasks in the background, without affecting the user interface or user interaction. [This can enhance the user experience and avoid freezing or hanging of the application](about:blank)[1](https://www.guru99.com/multithreading-java.html).
* [Taking advantage of multicore processors: A CPU-intensive application can use multithreading to distribute its workload among multiple cores or processors, which can speed up the execution and reduce the overall processing time1](https://www.guru99.com/multithreading-java.html).

1. What are the benefits of using Multithreading?

Ans: I just mentioned some of the benefits of using multithreading in the previous answer. To summarize, multithreading can help a program to:

* Improve performance: By executing multiple tasks concurrently, a program can make better use of the available CPU resources and reduce the idle time or waiting time of the threads.
* Improve responsiveness: By performing long-running or intensive tasks in the background, a program can keep the user interface responsive and interactive, and avoid freezing or hanging of the application.
* Improve scalability: By handling multiple user requests concurrently, a program can increase its throughput and efficiency, and serve more clients without compromising the quality of service.

1. What is Thread in Java?

Ans: A thread in Java is a lightweight sub-process, the smallest unit of processing, that can run in parallel with other threads within the same process . A thread has its own stack, program counter, and local variables, but it shares the heap, static variables, and resources with other threads of the same process. A thread can be created by extending the Thread class or implementing the Runnable interface, and it can be started by calling the start() method . A thread can have one of the following states: new, runnable, running, waiting, timed waiting, blocked, or terminated.

1. What are the two ways of implementing thread in Java?

Ans: The two ways of implementing thread in Java are:

* Extending the Thread class: This is a simple way of creating a thread by subclassing the Thread class and overriding its run() method. The run() method contains the code that the thread will execute. The thread can be started by creating an instance of the subclass and calling its start() method. However, this way has some limitations, such as:
  + A class can only extend one superclass, so it cannot extend any other class if it extends the Thread class.
  + A thread can only inherit the properties and behaviors of the Thread class, which may not be suitable for some scenarios.
* Implementing the Runnable interface: This is a more flexible way of creating a thread by implementing the Runnable interface and providing an implementation for its run() method. The run() method contains the code that the thread will execute. The thread can be started by creating an instance of the Runnable implementation and passing it to the constructor of the Thread class, and then calling its start() method. This way has some advantages, such as:
  + A class can implement multiple interfaces, so it can implement other interfaces besides the Runnable interface.
  + A thread can have any properties and behaviors that are defined by the Runnable implementation, which may be more suitable for some scenarios.

1. What’s the difference between thread and process?

Ans: The difference between a thread and a process in Java is similar to the general difference between a thread and a process, but there are some Java-specific aspects, such as:

* Creation and termination: A process in Java is created by invoking the Runtime.exec() method, which executes a system command and returns a Process object. [A process in Java is terminated by calling the destroy() method on the Process object, or by waiting for the process to finish normally1](https://perfectelearning.com/blog/what-is-the-difference-between-a-thread-and-a-process-in-java)[2](https://www.javatpoint.com/process-vs-thread-in-java). [A thread in Java is created by extending the Thread class or implementing the Runnable interface, and it is terminated by returning from the run() method or by calling the stop() method (which is deprecated) on the Thread object3](https://techdifferences.com/difference-between-process-and-thread-in-java.html)[4](https://www.cs-fundamentals.com/tech-interview/java/differences-between-thread-and-process-in-java).
* Communication and synchronization: A process in Java communicates with other processes via input and output streams, which are obtained from the Process object. [A process in Java can also communicate with the parent process via system properties or environment variables1](https://perfectelearning.com/blog/what-is-the-difference-between-a-thread-and-a-process-in-java)[2](https://www.javatpoint.com/process-vs-thread-in-java). [A thread in Java communicates with other threads of the same process via shared variables, which are synchronized using locks, monitors, or atomic classes](about:blank)[3](https://techdifferences.com/difference-between-process-and-thread-in-java.html)[5](https://www.javatpoint.com/process-vs-thread).
* Performance and overhead: A process in Java is more costly and slower to create and terminate than a thread in Java, as it involves invoking a system command and creating a separate memory space and resources. [A process in Java also has more context-switching overhead than a thread in Java, as it involves switching the input and output streams as well](about:blank)[1](https://perfectelearning.com/blog/what-is-the-difference-between-a-thread-and-a-process-in-java)[2](https://www.javatpoint.com/process-vs-thread-in-java).

1. How can we create daemon threads?

Ans: A daemon thread in Java is a thread that provides services to user threads, such as garbage collection or finalization. A daemon thread has a lower priority than a user thread, and its life depends on the user threads. [When all the user threads finish their execution, the JVM automatically terminates the daemon threads](about:blank)[1](https://www.geeksforgeeks.org/daemon-thread-java/)[2](https://www.javatpoint.com/daemon-thread).

We can create daemon threads in Java by calling the setDaemon(true) method on a Thread object before starting it. [This method sets the thread as a daemon thread, which means it will not prevent the JVM from exiting when the program finishes1](https://www.geeksforgeeks.org/daemon-thread-java/)[3](https://www.educba.com/daemon-thread-in-java/). [Alternatively, we can create daemon threads by using the Executors.newCachedThreadPool() method, which returns a thread pool that creates daemon threads as needed](about:blank)[4](https://www.baeldung.com/java-daemon-thread).

1. What are the wait() and sleep() methods?

Ans: The wait() and sleep() methods in Java are both used to pause the execution of a thread for a specified amount of time, but they have some differences, such as:

* Synchronization: The wait() method can only be called from a synchronized context, which means the thread must have acquired the lock on an object before calling wait() on that object. [The sleep() method can be called from any context, synchronized or not](about:blank)[1](https://blog.stackademic.com/6-difference-between-wait-and-sleep-methods-in-java-4c8f06c26e56)[2](https://www.tutorialspoint.com/differences-between-wait-and-sleep-method-in-java).
* Lock release: The wait() method releases the lock on the object while waiting, which allows other threads to enter the synchronized block and access the object. [The sleep() method does not release any lock while waiting, which prevents other threads from accessing the shared resource until the sleeping thread wakes up1](https://blog.stackademic.com/6-difference-between-wait-and-sleep-methods-in-java-4c8f06c26e56)[2](https://www.tutorialspoint.com/differences-between-wait-and-sleep-method-in-java).
* Wakeup condition: The wait() method can be woken up by another thread that calls notify() or notifyAll() on the same object, or by a timeout if specified. [The sleep() method can be woken up by an interrupt() call on the sleeping thread, or by a timeout if specified](about:blank)[3](https://www.geeksforgeeks.org/wait-method-in-java-with-examples/)[4](https://www.javatpoint.com/wait-vs-sleep-in-java).