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

Ans - Multithreading involves the utilization of multiple threads and stands as a significant feature within the Java programming language. As the name implies, it enables a central processing unit (CPU) to concurrently execute multiple threads while also allowing these threads to share process resources concurrently. The primary objective is to facilitate the simultaneous execution of numerous threads in order to optimize CPU utilization. This Java feature permits the division of a particular program into two or more threads, thereby facilitating streamlined and more efficient program execution.

**2 - What are the benefits of using Multithreading?**

Ans - The advantages of multithreading are outlined below:

* Facilitates uninterrupted program execution, even if a segment of it becomes blocked.
* Enables the creation of efficient programs that can make the most of available CPU time.
* Enhances the responsiveness of complex applications or programs.
* Maximizes CPU resource utilization, leading to reduced maintenance costs.
* Time-saving and concurrent execution of tasks.
* Requires fewer resources than executing multiple processes concurrently.

**3. What is Thread in Java?**

Ans - Threads are the smallest independent processing units, efficiently managed by a scheduler. They function as segments of a process, enabling effective simultaneous execution with other threads. Utilizing threads simplifies complex tasks and optimizes multiple available CPUs. They share an address space while remaining independent.

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

Ans **- Extending the Thread Class**: Create a class that extends the Thread class, override the run() method to define thread behavior, and then create instances of this class to start threads.

**Implementing the Runnable Interface:** Implement the Runnable interface by providing a run() method implementation. Then create instances of the class that implements Runnable and pass them to Thread objects for execution using the constructor.

Both methods involve defining the behavior of the thread in the run() method and starting the thread using the start() method.

**5. What's the difference between thread and process?**

Ans - Thread: The term "thread" signifies the tiniest components within a process, capable of executing various segments (also termed as threads) of a program concurrently.

Process: "Process" denotes an active program in execution. It can be managed via a Process Control Block (PCB).

**6.How can we create daemon threads?**

Ans - In Java, daemon threads can be established by employing the setDaemon(true) method within the thread class. This function designates the current thread as either a daemon or a user thread. To ascertain whether the present thread is a daemon, the isDaemon() method is typically employed. A true outcome indicates that the thread is a daemon, while false signifies otherwise.

**7. What are the wait() and sleep() methods?**

Ans - wait(): This non-static method, as its name implies, induces the present thread to pause and enter a dormant state until other threads invoke the notify() or notifyAll() methods for the object's monitor (lock). It effectively relinquishes the lock and primarily facilitates inter-thread communication. This method is located within the object class and ought to be invoked exclusively from a synchronized context.

sleep(): This static method, aptly named, halts or suspends the ongoing execution of the current thread for a designated duration. Unlike releasing the lock while waiting, it's primarily employed to introduce pauses during execution. Part of the thread class, this method doesn't require invocation within a synchronized context.