1. What Do You Mean by Multithreading? Why Is It Important:

- Multithreading is a Java feature that allows the execution of multiple threads simultaneously. Each thread runs independently and can perform different tasks concurrently, making efficient use of CPU resources.

- Importance: Multithreading improves the performance of a program by allowing multiple operations to run concurrently. It's essential for developing responsive applications, especially in scenarios where tasks can be performed in parallel, such as handling multiple user requests in a web server.

2. What Are the Benefits of Using Multithreading:

- **Improved Performance:** Tasks can be executed in parallel, leading to faster execution.

- **Resource Sharing:** Threads can share resources like memory and data, which is more efficient than using separate processes.

- **Responsive UI:** In GUI applications, multithreading helps keep the interface responsive by performing background tasks simultaneously.

- **Better Utilization of CPU:** Multithreading allows the CPU to perform tasks during idle times, such as I/O operations, leading to better CPU utilization.

3. What Is a Thread in Java:

- A thread in Java is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution, and Java's \`Thread\` class or \`Runnable\` interface is used to create threads. Threads can run concurrently with other threads, allowing multiple tasks to be performed at the same time.

4. What Are the Two Ways of Implementing Thread in Java:

- **Extending the \`Thread\` Class:**

  - In this method, a class extends the \`Thread\` class and overrides its \`run()\` method.

  - Example:

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread is running...");
    }
}
```

- **Implementing the `Runnable` Interface:**

  - In this method, a class implements the `Runnable` interface and overrides its `run()` method. The `Thread` object is created by passing the `Runnable` object.

  - Example:

```
class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Thread is running...");
    }
}
```

5. What's the Difference Between Thread and Process:

- **Thread:**

  - A thread is a lightweight subprocess and is the smallest unit of execution within a process. Multiple threads can exist within a single process, sharing its resources (e.g., memory).

- **Process:**

  - A process is an independent program in execution, with its own memory space and resources. Multiple processes do not share memory or resources, making inter-process communication more complex.

6. How Can We Create Daemon Threads:

- Daemon threads are background threads that run in the background to perform tasks like garbage collection. They automatically terminate when all user threads finish their execution.

- To create a daemon thread, call the `setDaemon(true)` method on a thread before starting it.

- Example:

```
Thread t = new Thread(new MyRunnable());
t.setDaemon(true);
t.start();
```

7. What Are the `wait()` and `sleep()` Methods:

- **`wait()` Method:**

  - The `wait()` method is used in inter-thread communication. It causes the current thread to release the lock and wait until another thread calls `notify()` or `notifyAll()` on the same object.

- Example:

```
synchronized(obj) {
    obj.wait();
}
```

- **`sleep()` Method:**

  - The `sleep()` method is used to pause the execution of the current thread for a specified duration. It does not release any locks during the sleep period.

  - Example:

```
Thread.sleep(1000); // Sleep for 1 second
```