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

Multithreading is a programming technique where multiple threads are executed concurrently within a single process.

* **Importance:**
  + Enhances **performance** by utilizing CPU resources more efficiently.
  + Allows for **parallel execution** of tasks, improving responsiveness (e.g., GUI applications).
  + Simplifies the management of multiple tasks like file reading, computations, and networking simultaneously.

### **2. What are the benefits of using Multithreading?**

* **Improved performance:** Tasks can run concurrently, reducing total execution time.
* **Resource sharing:** Threads share the same memory space, reducing the need for heavy resource allocation.
* **Better system utilization:** Exploits multicore CPUs effectively.
* **Enhanced responsiveness:** Ideal for applications like games or GUIs where the interface remains responsive.
* **Simplified maintenance:** Modularizing complex tasks into threads makes the code easier to maintain.

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

A thread in Java is the smallest unit of a program that can execute independently.

* It's a lightweight subprocess managed by the Java Virtual Machine (JVM).
* Each thread has its own stack but shares resources like memory with other threads of the same process.

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

1. **By extending the Thread class:**
   * Override the run() method.

class MyThread extends Thread {

public void run() {

System.out.println("Thread is running");

}

}

1. **By implementing the Runnable interface:**
   * Pass an instance of the class to a Thread object.

class MyRunnable implements Runnable {

public void run() {

System.out.println("Thread is running");

}

}

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

| **Feature** | **Thread** | **Process** |
| --- | --- | --- |
| **Definition** | Smallest unit of execution. | Independent program under execution. |
| **Resource Use** | Shares memory/resources of the process | Requires separate resources. |
| **Creation** | Lightweight, faster to create. | Heavyweight, slower to create. |
| **Communication** | Uses shared memory. | Requires IPC (Inter-Process Communication). |
| **Failure Impact** | A thread crash may not affect others. | Process crash terminates all threads. |

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

* A daemon thread is a background thread that supports non-daemon threads and terminates automatically when all user threads complete.
* **Steps to create a daemon thread:**
  1. Create a thread instance.
  2. Call the setDaemon(true) method before starting the thread.

java

Thread t = new Thread(() -> {

System.out.println("Daemon thread running");

});

t.setDaemon(true);

t.start();

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

| **Feature** | **wait()** | **sleep()** |
| --- | --- | --- |
| **Purpose** | Releases lock and waits until notified. | Pauses thread execution for a specified time. |
| **Usage** | Used in thread communication (sync). | Used for delaying thread execution. |
| **Lock** | Releases lock on the object. | Does not release lock. |
| **Class** | Defined in Object class. | Defined in Thread class. |
| **Throws** | Throws IllegalMonitorStateException if used without synchronization. | Throws InterruptedException. |

**Example for wait()**:

java

synchronized (obj) {

obj.wait();

}

**Example for sleep()**:

java

Thread.sleep(1000); // Pauses thread for 1 second.