***MultiThreading solution***

1. Multithreading refers to the concurrent execution of multiple threads within a single process. It allows programs to perform multiple tasks simultaneously, enhancing performance, responsiveness, and resource utilization. Multithreading is important for utilizing modern multicore processors efficiently and for creating responsive user interfaces.

2. Benefits of using multithreading include:

- Improved performance: Multithreading enables concurrent execution of tasks, making better use of available resources and reducing idle time.

- Enhanced responsiveness: Multithreading allows applications to remain responsive while performing time-consuming tasks in the background.

- Simplified code structure: Multithreading can simplify program structure by dividing complex tasks into smaller, manageable threads.

- Resource sharing: Threads within the same process can share resources such as memory, files, and network connections, facilitating efficient communication and coordination.

- Parallelism: Multithreading enables parallel execution of tasks, leading to faster execution and better scalability.

3. In Java, a thread is a lightweight process that represents an independent path of execution within a program. Threads allow concurrent execution of tasks and can be used to perform multiple operations simultaneously.

4. There are two ways of implementing threads in Java:

- Extending the Thread class: In this approach, a class extends the Thread class and overrides its run() method to define the code that will be executed by the thread.

- Implementing the Runnable interface: In this approach, a class implements the Runnable interface and provides an implementation for the run() method. The class is then passed as an argument to a Thread object, which is responsible for executing the run() method.

5. The main differences between a thread and a process are:

- Thread: A thread is a lightweight unit of execution within a process. Multiple threads can exist within a single process and share resources such as memory and files.

- Process: A process is a standalone unit of execution that has its own memory space, resources, and execution environment. Processes are independent of each other and communicate through inter-process communication mechanisms.

6. Daemon threads in Java are threads that run in the background and provide services to other threads or perform background tasks. To create a daemon thread, you can call the setDaemon(true) method on a Thread object before starting it. Daemon threads automatically terminate when all non-daemon threads have finished executing.

7. In Java, the wait() and sleep() methods are used for thread synchronization and timing control, respectively:

- wait(): The wait() method is used to pause the execution of a thread and release the object's lock until another thread invokes the notify() or notifyAll() method on the same object.

- sleep(): The sleep() method is used to temporarily suspend the execution of a thread for a specified period of time. Unlike wait(), sleep() does not release the object's lock.