Q1) What do you mean by Multithreading? Why is it important?
Multithreading is a programming concept in which the application can create a small unit of tasks to execute in parallel.

If you ate working on a computer, it runs multiple applications and allocates processing power to them. A simple program runs in sequence and the code statements execute one by one. This is a single-threaded application. But, if the programming language supports creating multiple threads and passes them to the operating system to run in parallel, it’s called multithreading.

Q2) What are the benefits of using Multithreading?
Benefits of Java Multithreading:
1. It doesn't block the user because threads are independent and you can perform multiple operations at the same time.
2. You can perform many operations together, so it saves time.
3. Threads are independent, so it doesn't affect other threads if an exception occurs in a single thread

Q3) Differentiate between process and thread.

Process	**Thread**
● Process means any program is in execution. ● The process takes more time to terminate ● It takes more time for creation. ● It also takes more time for context switching. ● The process is less efficient in terms of communication. ● Multiprogramming holds the	● Thread means a segment of a process. ● The thread takes time to terminate. ● It takes less time for creation. ● It takes less time for context switching. ● Thread is more efficient in terms of communication.

concepts of multi-process. ● The process is isolated. ● The process is called the heavyweight process. ● Process switching uses an interface in an operating system. ● The process has its own process control block, stack, and address space. ● A system call is involved in it. ● The process does not share data with each other.	● We don't need multi programs in action for multiple threads because a single process consists of multiple threads. ● Threads share memory. ● A thread is lightweight as each thread is a process shares code, data, and resources. ● Thread switching does not require calling an operating system and causes an interrupt to the kernel. ● Thread has parent's PCB, its own thread control block, and stack and common address space. ● No system call is involved, it is created using APIs. ● Threads share data with each other.

Q4) What are the different states of a thread, or what is thread lifecycle?
Following are the stages of the life cycle or the states of thread :

- **New** - a new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
- **Runnable -** After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting** - Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. Thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed waiting** - A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.

- **Terminated -** A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Q5) Differentiate between the Thread class and Runnable interface for creating a Thread?

Thread	Runnable
● Thread is a class. It is used to create a thread ● It has multiple method including start() and run() ● Each thread creates a unique object and gets associated with it ● More memory required ● Multiple inheritance is not allowed in java hence after a class extends thread class, it cannot extend any other class	● Runnable is a functional interface which is used to create a thread ● It has only abstract method run() ● Multiple threads share the same objects. ● Less memory required ● If a class is implementing the runnable interface, then your class can extend another class.

Q6) What if we call Java run() method directly instead start() method?
If we call run method directly, then it will execute not in another thread, but in the current thread. If start isn't called, then the thread created will never run. The main thread will finish and the thread will be garbage collected.