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

Ans: Multithreading means multiple threads and is considered one of the most important features of Java. As the name suggests, it is the ability of a CPU to execute multiple threads independently at the same time but share the process resources simultaneously. It's main purpose is to provide simultaneous execution of multiple threads to utilise the CPU time as much as possible. It is a Java feature where one can subdivide the specific program into two or more threads to make the execution of the program fast and easy.

2) What are the benefits of using Multithreading?

Ans: There are various benefits of multithreading as given below :

- Allow the program to run continuously even a part of it is blocked.
- Improve performance as compared to traditional parallel programs that use multiple processes.
- Allows to write effective programs that utilise maximum CPU time.
- Improves the responsiveness of complex applications or programs.
- Increase use of CPU resources and reduce costs of maintenance.
- Saves time and parallelism tasks.
- If an exception occurs in a single thread, it will not effect other threads as threads are independent.
- Less resource-intensive than executing multiple processes at same time.

3) What is Thread in Java?

Ans: Threads are basically the lightweight and smallest unit of processing that can be managed independently by a scheduler. Threads are referred to as parts of a process that simply let a program execute efficiently with other parts or threads of the process at the same time. Using threads, one can perform complicated tasks in the easiest way. It is considered the simplest way to take advantage of multiple CPUs available in a machine. They share the common address space and are independent of each other.

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

Ans: There are basically two ways of implementing thread in Java as given below:

Extending the thread class

Example:

```
class MultithreadingDemo extends Thread
{
    public void run()
    {
        System.out.println("My thread is in running state. ");
    }
    public static void main(String args[])
    {
         MultithreadingDemo obj = new MultithreadingDemo();
         obj.start();
    }
}
```

Output: My thread is in running state.

Implementing Runnable interface in Java

Example:

```
class MultithreadingDemo implements Runnable
{
    public void run()
    {
        System.out.println("My thread is in running state.");
    }
    public static void main(String args [])
    {
        MultithreadingDemo obj = new MultithreadingDemo();
        Thread tobj = new Thread(obj);
        tobj.start();
    }
}
```

Output : My thread is in running state.

5) What's the difference between thread and process?

Ans: Thread : It is simply refers to the smallest units of the particular process. It has ability to execute different parts (referred to as thread) of the program at the same time.
Process : It simply refers to a program that is in execution I.e, an active program. A process can be handled using PCB(Process Control Block).

6) How can we create daemon threads?

Ans: We can create daemon threads in Java using the thread class setDeamon(true). It is used to mark the current thread as daemon thread or user thread. isDeamon() method is generally used to check whether the current thread is daemon or not. If the thread is a daemon, it will return True otherwise it will returns false.

Example:

Program to illustrate the use of setDaemon() and isDaemon() method.

```
public class DeamonThread extends Thread
{
    public DaemonThread(String name)
    {
        super(name);
    }
    public void run()
    {
        // Checking whether the thread is Daemon or not
        if(Thread.currentThread().isDeamon())
        {
            System.out.println(getName() + " is Daemon thread");
        }
        else
        {
```

```
                System.out.println(getName() + " is  User thread ");
            }
        }
        public static void main(String args [])
        {
            DaemonThread t1 = new DaemonThread("t1");
            DeamonThread t2 = new DeamonThread("t2");
            DeamonThread t3 = new DeamonThread("t3");
           // Setting user thread t1 to Daemon
           t1.setDaemon(true);
           // starting first 2 threads
            t1.start();
            t2.start();
           // Setting user thread t3 to Daemon
            t3.setDaemon(true);
            t3.start();
        }
}
```

Output:

t1 is Daemon Thread
t3 is Daemon Thread
t2 is User Thread

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

Ans : wait() : As the name suggests, it is a non-static method that causes the current thread to wait and go to sleep until some other threads call the notify() or notifyAll() method for the object's monitor (lock). It simply releases the lock and is mostly used for inter-thread communication. It is defined in the object class, and should only be called from a synchronized context.

Example:

```
synchronized(monitor)
{
   monitor.wait(); // Here Lock is Released by current thread
}
```

Sleep() : As the name suggests, it is a static method that pauses or stops the execution of the current thread for some specified period. It doesn't release the lock while waiting and is mostly used yo introduce pause on execution. It is defined in thread class, and no need to call from a synchronised context.

Example :

```
synchronized(monitor)
{
    Thread.sleep(1000); // Here Lock is Held by the current thread
// after 1000 milliseconds, the current thread will wake up, or after we call that is
interrupt() method
}
```