1. What do you mean by Multithreading? Why is it important?
Ans: In simple terms, Multithreading means multiple threads. It is one of the most essential features of Java. If an application/program is executing, it is called a Process. Within a Process, there can be multiple tasks. If multiple tasks in a single Process can perform simultaneously and independently of each other then it is called Thread-based Multitasking or Multithreading and since it happens inside a Process it is Process-level Multitasking. It is the ability of the CPU to execute multiple Threads independently at the same time but share the process resources simultaneously. Due to this simultaneous execution of multiple threads, the CPU time cycle is utilized efficiently.
Due to this feature in Java, a single program can be divided into multiple tasks(Threads) to make the execution of the program quick and easy.

2. What are the benefits of using Multithreading?
Ans: There are various benefits of multithreading as given below:
i. Allow the program to run continuously even if a part of it is blocked.
ii. Improve performance as compared to traditional parallel programs that use multiple processes.
iii. Allows writing effective programs that utilize maximum CPU time.
iv. Improves the responsiveness of complex applications or programs.
v. Increases the use of CPU resources and reduces costs of maintenance.
vi. Savestimeandparallelismtasks.
vii. If an exception occurs in a single thread, it will not affect other threads as threads are independent.
viii. Less resource-intensive than executing multiple processes at the same time.

3. What is Thread in Java?
Ans: A very light-weighted and smallest unit of a process that allows a program to operate more efficiently by running multiple tasks simultaneously. Threads can be independently managed by a Thread Scheduler. Using threads, we can perform complex tasks in an easy way and also utilize CPU time cycle. 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: Threads can be implemented in the following 2 ways in Java:
A. By extending the Thread class and the task to be performed should be written inside the run() method of the class.
For using the Thread, an object of the class must be created.

```
class MyThread extends Thread{
    public void run(){
        System.out.println("Thread1 is being executed!");
    }
}
```

```
public class Test{
    public static void main(String[] args) {
        Thread t = new MyThread();
        t.start();
    }
}

Output:
Thread1 is being executed!
```

B. By implementing the Runnable Interface and the task to be performed should be written inside the run() method of the class.
For using the Thread, an object of the class must be created. Then an object of Thread class must be created and the reference of the implemented class object must be given to it.

```
class Library implements Runnable{
    public void run(){
        System.out.println("Thread1 is being executed!");
    }
}

public class Test{
    public static void main(String[] args) {
        Library lib = new Library();
        Thread t = new Thread(lib);
        t.start();
    }
}

Output:
Thread1 is being executed!
```

5. What's the difference between thread and process?
Ans: Thread refers to the smallest and very light-weighted unit of a particular process. It has the ability to execute multiple parts(referred to as Threads) of a single process/program at the same time.
Process refers to the program/application that is in execution as a whole, i.e., an active program. A process can be handled using a Process Control Block(PCB).

6. How can we create daemon threads?
Ans: We can create daemon threads in Java using the thread `class setDaemon(true)`. It is

used to mark the current thread as a daemon thread or user thread. `isDaemon()` 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 returns false.

Example:

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

```
public class DaemonThread extends Thread {
    public DaemonThread(String name) {
        super(name);
    }
    public void run() {
        // Checking whether the thread is Daemon or not
        if(Thread.currentThread().isDaemon()) {
            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");
        DaemonThread t2 = new DaemonThread("t2");
        DaemonThread t3 = new DaemonThread("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(): 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(resource1){
    resource1.wait(); // Here, lock is released by current Thread
}
```

sleep(): It is a static method that pauses or stops the execution of the current thread for some specified period. The programmer can specify the amount/period of time the thread should be running the sleep() method. It doesn't release the lock while waiting and is mostly used to introduce a pause on execution. It is defined in the Thread class, and there is no need to call from a synchronized context.

Example:

```
synchronized(resource1){
    try{
        Thread.sleep(3000);
        //Here the Thread will sleep for 3 seconds and the resource is
blocked by this Thread
        //After 3 seconds, the Thread will wake up
    }
    catch(Exception e){
        System.out.println("Error" + e.getMessage());
    }
```