Multithreading Assignment Question

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

Answer – 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. Its main purpose is to provide simultaneous execution of multiple threads to utilize 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.

Question 2: What are the benefits of using Multithreading?

Answer – There are various benefits of multithreading as given below:

- **Allow the program to run continuously even if a part of it is blocked.**
- **Improve performance as compared to traditional parallel programs that use multiple processes.**
- **Allows to write effective programs that utilize 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 affect other threads as threads are independent.**
- **Less resource-intensive than executing multiple process at the same time.**

Question 3: What is Tread in java?

Answer – 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.

Question 4: What are the two ways of implementing thread in java?

Answer – There are basically two ways of implementing thread in java as given below:

Extending the Thread class

Example:

```
// Java code for thread creation by extending

// the Thread class
```

```
class MultithreadingDemo extends Thread {
    public void run()
    {
        try {
            // Displaying the thread that is running
            System.out.println(
                "Thread " + Thread.currentThread().getId()
                + " is running");
        }
        catch (Exception e) {
            // Throwing an exception
            System.out.println("Exception is caught");
        }
    }
}

// Main Class
public class Multithread {
    public static void main(String[] args)
    {
        int n = 8; // Number of threads
        for (int i = 0; i < n; i++) {
            MultithreadingDemo object
                = new MultithreadingDemo();
            object.start();
        }
    }
}
```

Output

```
Thread 15 is running
Thread 14 is running
Thread 16 is running
Thread 12 is running
Thread 11 is running
Thread 13 is running
Thread 18 is running
Thread 17 is running
```

Implementing Runnable interface in java

```java
// Java code for thread creation by implementing
// the Runnable Interface
class MultithreadingDemo implements Runnable {
    public void run()
    {
        try {
            // Displaying the thread that is running
            System.out.println(
                "Thread " + Thread.currentThread().getId()
                + " is running");
        }
        catch (Exception e) {
            // Throwing an exception
            System.out.println("Exception is caught");
        }
    }
}

// Main Class
class Multithread {
    public static void main(String[] args)
    {
        int n = 8; // Number of threads
        for (int i = 0; i < n; i++) {
            Thread object
                = new Thread(new MultithreadingDemo());
            object.start();
        }
    }
}
```

Output

```
Thread 13 is running
Thread 11 is running
Thread 12 is running
Thread 15 is running
Thread 14 is running
Thread 18 is running
Thread 17 is running
Thread 16 is running
```

Question 5: What's the difference between thread and process?

Answer: Thread: It simply refers to the smallest units of the particular process. It has the 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).

Question 6: How can we create daemon thread?

Answer : We can create daemon threads in java using the thread class setDaemon(true).

It is used to mark the current thread as daemon thread or user thread. isDaemon() method is generally used to crack whether the current thread is daemon or not. If the thread is a daemon, it will return true otherwise it return false.

Daemon thread in Java is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.

Points to remember for Daemon Thread in Java

It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.

Its life depends on user threads.

It is a low priority thread.

Example:

```
public class TestDaemonThread1 extends Thread{
 public void run(){
  if(Thread.currentThread().isDaemon()){//checking for daemon thread
   System.out.println("daemon thread work");
  }
  else{
  System.out.println("user thread work");
 }
 }
 public static void main(String[] args){
  TestDaemonThread1 t1=new TestDaemonThread1();//creating thread
  TestDaemonThread1 t2=new TestDaemonThread1();
  TestDaemonThread1 t3=new TestDaemonThread1();

  t1.setDaemon(true);//now t1 is daemon thread

  t1.start();//starting threads
  t2.start();
  t3.start();
 }
}
```

Output:

```
daemon thread work

user thread work

user thread work
```

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

Answer:

Sleep()

The Sleep () method is related to the Thread class that is used to stop the execution of the current Thread for few seconds. The Sleep () method takes the sleeping time in milliseconds. The monitor's ownership is not lost when we use the Sleep () method and start the execution again from where it stops. In simple words, the Sleep() method is responsible for sending the current Thread into the "Non Runnable" state.

Wait()

The Wait() method is related to the Object class. The Wait() method is responsible for sending the calling thread into the waiting state. The Thread remains in the waiting state until another thread doesn't invoke the notify() or notifyAll() method for that object. The Thread resumes the execution after obtaining the ownership of the monitor.

Before understanding the differences between both of them, let's understand the similarities between them. So, both the Wait() and Sleep() methods are the native methods that make the current Thread go into the Non-Runnable State.