11/12/2024

Date :
Page No.:

JAVA ASSIGNMENT - 2

1) What do you mean by a thread? Explain the different ways of creating threads.

Ans: A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread.

In the most general sense, you create a thread by instantiating an object of type Thread. There are two ways -

→ You can implement the Runnable interface.

→ You can extend the Thread class, itself.

Implementing Runnable

→ The easiest way to create a thread is to create a class that implements the Runnable interface.

→ Runnable abstracts a unit of executable code.

→ You can construct a thread on any object that implements Runnable.

→ To implement Runnable, a class need only implement a single method run(), which is declared like this: public void run()

→ Inside run(), you will define the code that constitutes the new thread.

→ It is important to understand that run() can call other methods, use other classes and declare variables, just like the main thread can.

→ The only difference is that run() establishes the entry point for another, concurrent thread of execution within your program.

→ This thread will end when run() returns.

→ After you create a class that implements Runnable, you

will instantiate an object of type Thread from within that class.

→ Thread defines several constructors.

→ The one that we will use is shown here:
Thread(Runnable threadOb, String threadName)

→ In this constructor, threadOb is an instance of a class that implements the Runnable interface. This defines where execution of the thread will begin.

→ The name of the new thread is specified by threadName. After the new thread is created, it will not start running until you call its start() method, which is declared within Thread.

→ In essence, start() executes a call to run(). The start() method is shown here: void start()

Extending Thread

The second way to create a thread is to create a new class that extends Thread, and then to create an instance of that class. The extending class must override the run() method, which is the entry point for the new thread. It must also call start() to begin execution of the new thread.

2) What is the need of synchronization? Explain with an example how synchronization is implemented in JAVA.

Ans:- In a multi-threaded environment, multiple threads run concurrently and may try to access the same resource (e.g., a shared variable, object or file). Without synchronization, inconsistent or unexpected results can occur due to simultaneous access. This is called a race condition, where the final output depends on the timing of thread execution. Synchronization ensures that -

1) Data Consistency: Only one thread can access the critical section at a time.
2) Thread Safety: Shared resources are used safely in a multi-threaded environment.
3) Avoiding Deadlocks: Threads coordinate effectively to prevent indefinite blocking.

With Synchronization

```
class Counter {
  int count = 0;
  public void increment() {
    synchronized(this) {
      count++;
    }
  }
}
public class WithoutSync {
  public static void main(String[] args) throws InterruptedException
  {
    Counter counter = new Counter();
    Thread t1 = new Thread(() -> {
```

```
        for (int i=0; i<1000; i++) {
            counter.increment();
        }
    });
    Thread t2 = new Thread(() -> {
        for(int i=0; i<1000; i++) {
            counter.increment();
        }
    });
    t1.start();
    t2.start()
    t1.join();
    t2.join();
    System.out.println("Final count: "+ counter.count);
  }
}
```

O/P:

Final count: 2000

Without the Synchronization, due to the race condition the output maybe less than 2000; like 1987 or 1994, depending on thread scheduling.

3.) What is multithreading? Write a program to create multiple threads in JAVA.

Ans: Multithreading in JAVA is a process of executing multiple threads simultaneously. A thread is the smallest unit of a program that can be scheduled for execution independ-ently. Multithreading enables concurrent execution of two or more threads to perform tasks efficiently.

```
//Create multiple threads.
class NewThread implements Runnable {
  String name;
  Thread t;
  NewThread(String threadname) {
    name = threadname;
    t = new Thread(this, name);
    System.out.println("New Thread: "+t);
    t.start();
  }
  //This is the entry point for thread
  public void run() {
    try {
      for(int i=5; i>0; i--) {
        System.out.println(name + "[illegible] :" +i);
        Thread.sleep(1000);
      }
    } catch (InterruptedException e) {
      System.out.println(name + "Interrupted");
    }
```

```
        System.out.println(name + "existing.");
    }
}
class MultiThreadDemo {
    public static void main(String[] args) {
        new NewThread("One");
        new NewThread("Two")
        new NewThread("Three");
        try {
            //wait for other threads to end
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            System.out.println("Main thread Interrupted");
        }
        System.out.println("Main thread exiting.");
    }
}
```

O/P:

```
New thread: Thread [One, 5, main]
New thread: Thread [Two, 5, main]
New thread: Thread [Three, 5, main]
One: 5
Two: 5
Three: 5
One: 4
Two: 4
Three: 4
```

One : 3
Two : 3
Three : 3
One : 2
Two : 2
Three : 2
One : 1
Three : 1
Two: 1
One exiting.
Two exiting.
Three exiting.
Main thread exiting.

4) Explain with an example how inter thread communication is implemented in JAVA.

Ans:

```
class Q {
  int n;
  synchronized int get() {
    System.out.println("Got: "+n);
    return n;
  }
  synchronized void put(int n) {
    this.n=n;
    System.out.println("Put: "+n);
  }
}
```

```
class Producer implements Runnable {
    Q q;
    Producer(Q q) {
        this.q = q;
        new Thread(this, "Producer").start();
    }
    public void run() {
        int i = 0;
        while(true) {
            q.put(i++);
        }
    }
}
class Consumer implements Runnable {
    Q q;
    Consumer(Q q) {
        this.q = q;
        new Thread(this, "consumer").start();
    }
    public void run() {
        while(true) {
            q.get();
        }
    }
}
class PC {
    public static void main(String[] args) {
        Q q = new Q();
        new Producer(q);
        new Consumer(q);
```

```
        System.out.println("Press Control-C to stop.");
    }
}
```

O/P:

```
Put: 1
Got: 1
Got: 1
Got: 1
Got: 1
Got: 1
Put: 2
Put: 3
Put: 4
Put: 5
Put: 6
Put: 7
Got: 7
```

5) Explain the lifecycle of a Thread with a neat diagram.

Ans: In Java, a thread goes through several states during its execution. The thread lifecycle refers to the various states a thread can exist in, from its creation to its terminations. These states are defined in the Thread.State enumeration in Java.

Thread States in Java

1) New (Created)

→ A thread is in the "new" state when it is created but

not yet started.
→ It enters this state when the Thread object is instantiated.
→ Ex: Thread t1 = new Thread();

2) Runnable
→ A thread is in the "runnable" state when the start() method is called.
→ It is ready to run but waiting for the CPU scheduler to allocate execution time.
→ Ex: t1.start();

3) Running:
→ The thread is executing its run() method.
→ Only one thread can be in the "running" state at a time for a single processor.

4) Blocked:
→ A thread is in the "blocked" state when it is waiting for a monitor lock to enter a synchronized block or method.
→ It will remain blocked until it gets access to the lock.

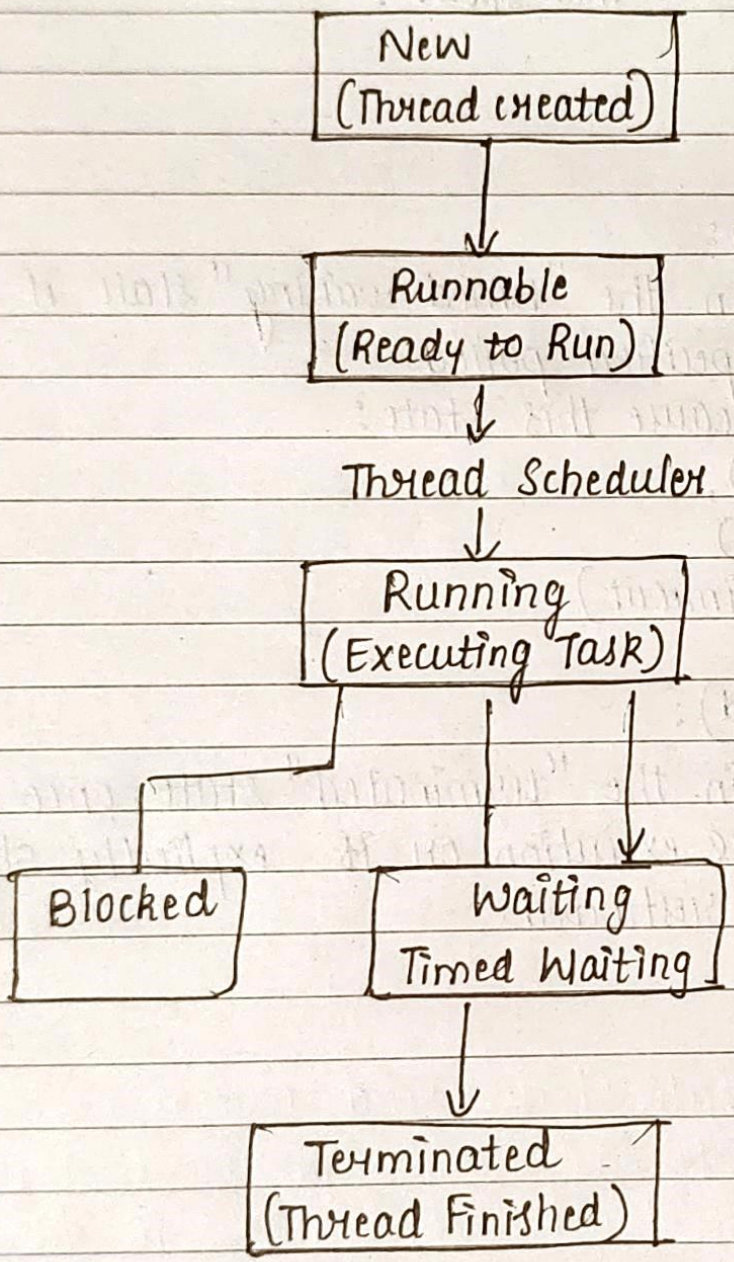
5) Waiting:
→ A thread is in the "waiting" state if it is waiting indefinitely for another thread to perform a specific action.

→ Methods that cause this ~~state~~ :
→ Object.wait()
Thread.join()

6.) → Timed Waiting:
→ A thread is in the "timed waiting" state if it is waiting for a specified period.
→ Methods that cause this state:
Thread.sleep();
Object.wait()
Thread.join(timeout)

7.) Terminated (Dead):
→ A thread is in the "terminated" state once its run() method completes execution or is explicitly stopped.
→ It cannot be restarted.

Lifecycle


New
(Thread created)
Runnable
(Ready to Run)
Thread Scheduler
Running
(Executing Task)
Blocked
Waiting
Timed Waiting
Terminated
(Thread Finished)