Quiz 8

Questions on working with ThreadLocal variables

Question # 1

Consider the class below:

```
public class Counter {

    ThreadLocal<Integer> counter = ThreadLocal.withInitial(() -> 0);

    public Counter() {
        counter.set(10);
    }

    void increment() {
        counter.set(counter.get() + 1);
    }
}
```

What would be the output of the method below when invoked?

```
    public void usingThreads() throws Exception {

        Counter counter = new Counter();
        Thread[] tasks = new Thread[100];

        for (int i = 0; i < 100; i++) {
            Thread t = new Thread(() -> {
                for (int j = 0; j < 100; j++)
                    counter.increment();
            });
            tasks[i] = t;
            t.start();
        }

        for (int i = 0; i < 100; i++) {
            tasks[i].join();
```

```
        }

        // What is the output of the the below line?
        System.out.println(counter.counter.get());
    }
```

Q

A) 0

B) 10

C) 100

D) 110

E) 10000

COMPLETED 0%

1 of 1

Show Explanation

Question # 2

Given the same `Counter` class as in the previous question, what is the output of println statement below:

```
    public void usingSingleThreadPool() throws Exception {
```

```
        Counter counter = new Counter();

        ExecutorService es = Executors.newFixedThreadPool(1);
        Future<Integer>[] tasks = new Future[100];

        for (int i = 0; i < 100; i++) {
            tasks[i] = es.submit(() -> {
                for (int j = 0; j < 100; j++)
                    counter.increment();

                return counter.counter.get();
            });
        }

        // What is the output of the below line?
        System.out.println(tasks[99].get());

        es.shutdown();
    }
```

Q

A) 0

B) 10

C) 100

D) 10000

COMPLETED 0%

1 of 1

Show Explanation

Question # 3

What would have been the output of the print statement from the previous question if we created a pool with 20 threads?

Q

A) 0

B) 10

C) 100

D) 1000

E) between 100 and 10000 inclusive

COMPLETED 0%

1 of 1

Show Explanation

The code for all the three scenarios discussed above appears below.

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
```

```
class Demonstration {
    public static void main( String args[] ) throws Exception {

      usingThreads();
      usingSingleThreadPool();
      usingMultiThreadsPool();
    }

    static void usingThreads() throws Exception {

        Counter counter = new Counter();
        Thread[] tasks = new Thread[100];

        for (int i = 0; i < 100; i++) {
            Thread t = new Thread(() -> {
                for (int j = 0; j < 100; j++)
                    counter.increment();
            });
            tasks[i] = t;
            t.start();
        }

        for (int i = 0; i < 100; i++) {
            tasks[i].join();
        }

        System.out.println(counter.counter.get());
    }

    @SuppressWarnings("unchecked")
    static void usingSingleThreadPool() throws Exception {

        Counter counter = new Counter();
        ExecutorService es = Executors.newFixedThreadPool(1);
        Future<Integer>[] tasks = new Future[100];

        for (int i = 0; i < 100; i++) {
            tasks[i] = es.submit(() -> {
                for (int j = 0; j < 100; j++)
                    counter.increment();

                return counter.counter.get();
            });
        }

        System.out.println(tasks[99].get());

        es.shutdown();
    }

    @SuppressWarnings("unchecked")
    static void usingMultiThreadsPool() throws Exception {

        Counter counter = new Counter();
        ExecutorService es = Executors.newFixedThreadPool(20);
        Future<Integer>[] tasks = new Future[100];

        for (int i = 0; i < 100; i++) {
            tasks[i] = es.submit(() -> {
                for (int j = 0; j < 100; j++)
                    counter.increment();
```

```
                return counter.counter.get();
            });
        }

        System.out.println(tasks[99].get());

        es.shutdown();
    }

}

class Counter {

    ThreadLocal<Integer> counter = ThreadLocal.withInitial(() -> 0);

    public Counter() {
        counter.set(0);
    }

    void increment() {
        counter.set(counter.get() + 1);
    }
}
```




Question # 4

Consider the below method:

```
    int countTo100() {

        ThreadLocal<Integer> count = ThreadLocal.withInitial(() -> 0
);
        for (int j = 0; j < 100; j++)
            count.set(count.get() + 1);

        return count.get();

    }
```

The above code is invoked like so:

```
        ExecutorService es = Executors.newFixedThreadPool(1);
        Future<Integer>[] tasks = new Future[100];

        for (int i = 0; i < 100; i++) {
            tasks[i] = es.submit(() -> countTo100());
```

```
        }


        for (int i = 0; i < 100; i++)
            System.out.println(tasks[i].get());

        es.shutdown();
```

What would the output of the print statement for the 100 tasks?

Q

A) 100

B) 10000

C) between 100 and 10,000 inclusive

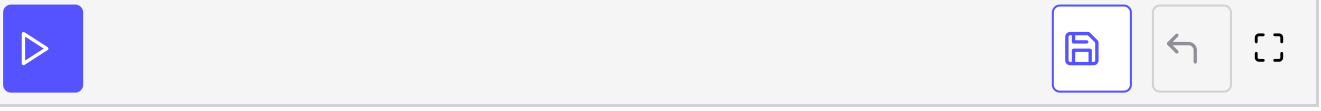
COMPLETED 0%

1 of 1

Show Explanation

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;


class Demonstration {

    @SuppressWarnings("unchecked")
    public static void main( String args[] ) throws Exception {

        ExecutorService es = Executors.newFixedThreadPool(1);
        Future<Integer>[] tasks = new Future[100];

        for (int i = 0; i < 100; i++) {
            tasks[i] = es.submit(() -> countTo100());
        }
```

```
        for (int i = 0; i < 100; i++)
            System.out.println(tasks[i].get());


        es.shutdown();
    }

    static int countTo100() {

        ThreadLocal<Integer> count = ThreadLocal.withInitial(() -> 0);
        for (int j = 0; j < 100; j++)
            count.set(count.get() + 1);

        return count.get();

    }
}
```



Question # 5

Is there any benefit to declaring `count` as a threadlocal variable in the method `countTo100()`?

```
    int countTo100() {

        ThreadLocal<Integer> count = ThreadLocal.withInitial(() -> 0
);
        for (int j = 0; j < 100; j++)
            count.set(count.get() + 1);

        return count.get();

    }
```

The variables defined inside an instance method are already created on a per-thread basis and live on the thread stack without any sharing with other threads. The per-thread level isolation for a variable that we can achieve using threadlocal is already being provided because of the scope of the variables declared within an instance method. Therefore, there's no benefit to declaring variables within instance methods as threadlocal.