Quiz 1

Question # 1

What are some of the differences between a process and a thread?

- A process can have many threads, whereas a thread can belong to only one process.
- A thread is lightweight than a process and uses less resources than a process.
- A thread has some state private to itself but threads of a process can share the resources allocated to the process including memory address space.

Question # 2

Given the below code, can you identify what the coder missed?

```
    void defectiveCode(final int n) throws ExecutionException, Interr
uptedException {

        ExecutorService threadPool = Executors.newFixedThreadPool(5);

        Callable<Void> sumTask = new Callable<Void>() {

            public Void call() throws Exception {
                System.out.println("Running");
                return null;
```

```
            }
        };


        threadPool.submit(sumTask);
        f.get();
    }
```

The above code forgets to ***shutdown*** the executor thread pool. The thread pool when instantiated would also create 5 worker threads. If we don't shutdown the executor when exiting the main method, then JVM would also not exit. It will keep waiting for the pool's worker threads to finish, since they aren't marked as daemon. As an example execute the below code snippet.

```
import java.util.concurrent.Ca
import java.util.concurrent.Ex
import java.util.concurrent.Ex
import java.util.concurrent.Ex
import java.util.concurrent.Fu

class Demonstration {
    public static void main( S
        ExecutorService threadPo

        Callable<Void> someTask =

                public Void call()
                    System.out.pri
                    return null;
                }
            };

        threadPool.submit(someTa

        System.out.println( "Prog
    }
}
```

The above program execution will show execution timed out, even though both the string messages are printed. You can fix the above code by adding `threadPool.shutdown()` as the last line of the method.

Question # 3

Which `compute()` method do you think would get invoked when `getWorking` is called?

```
class ThreadsWithLambda {

    public void getWorking() throws Exception {
        compute(() -> "done");
    }

    void compute(Runnable r) {
        System.out.println("Runnable invoked");
        r.run();
    }

    <T> T compute(Callable<T> c) throws Exception {
        System.out.println("Callable invoked");
        return c.call();
    }
}
```

The lamda expression is returning the string done, therefore the compiler will match the call to the second compute method and the expression will be considered a type of interface `Callable`. You can run the below snippet and verify the output to convince yourself.

```
import java.util.concurrent.Callable;
```

```
class Demonstration {
    public static void main( String args[] ) throws Exception{
        (new LambdaTargetType()).getWorking();

    }
}

class LambdaTargetType {

    public void getWorking() throws Exception {
        compute(() -> "done");
    }

    void compute(Runnable r) {
        System.out.println("Runnable invoked");
        r.run();
    }

    <T> T compute(Callable<T> c) throws Exception {
        System.out.println("Callable invoked");
        return c.call();
    }
}
```

Question # 4

What are the ways of representing tasks that can be executed by threads in Java?

Q

A) Runnable interface and subclassing Thread class

B) Passing anonymous class instance to thread constructor

C) IComparable interface and subclassing Thread class

CHECK ANSWERS

Question # 5

Given the code snippet below, how many times will the innerThread print its messages?

```
public void spawnThread() {

    Thread innerThread = new Thread(new Runnable() {

        public void run() {

            for (int i = 0; i < 100; i++) {
                System.out.println("I am a new thread !");
            }
        }
    });

    innerThread.start();
    System.out.println("Main thread exiting");
}
```

Q

A) innerThread prints a few messages and dies when the main thread exits

B) innerThread prints exactly 100 messages even if the main thread exits before innerThread is done

C) innerThread dies as soon as the main thread exits without printing any messages

CHECK ANSWERS

Question # 6

Given the below code snippet how many messages will the innerThread print?

```
public void spawnDaemonThread() {

    Thread innerThread = new Thread(new Runnable() {

        public void run() {

            for (int i = 0; i < 100; i++) {
                System.out.println("I am a daemon thread !");
            }
        }
    });

    innerThread.setDaemon(true);
    innerThread.start();
    System.out.println("Main thread exiting");
}
```

Q

- A) exactly 100
- B) none
- C) a few

CHECK ANSWERS

Question # 7

Say your program takes exactly 10 minutes to run. After reading this course, you become excited about introducing concurrency in your program. However, you only use two threads in your program. Holding all other variables constant, what is minimum time your improved program can theoretically run in?

Q

A) 2 minutes

B) 5 minutes

C) can't be determined.

D) 3.5 minutes

CHECK ANSWERS

Question # 8

A sequential program is refactored to take advantage of threads. However, the programmer only uses two threads. The workload is divided such that one thread takes 9 times as long as the other thread to finish its work. What is the theoretical maximum speedup of the program as a percentage of the sequential running time?

Q

A) 10%

B) 9%

C) 11%

D) 30%

CHECK ANSWERS