CIS 36B – 19th Class / Lab Assignment – **10 Points-**

**Student Name** **Student ID** **Point Total**

# Section 1: Define

## Pg. Chapter 12, pg. 435 - Java Programming *A comprehensive Introduction*

## Thread-

A *thread* is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.

Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority. Each thread may or may not also be marked as a daemon. When code running in some thread creates a new `Thread` object, the new thread has its priority initially set equal to the priority of the creating thread, and is a daemon thread if and only if the creating thread is a daemon.

When a Java Virtual Machine starts up, there is usually a single non-daemon thread (which typically calls the method named `main` of some designated class). The Java Virtual Machine continues to execute threads until either of the following occurs:

- The `exit` method of class `Runtime` has been called and the security manager has permitted the exit operation to take place.
- All threads that are not daemon threads have died, either by returning from the call to the `run` method or by throwing an exception that propagates beyond the `run` method.

There are two ways to create a new thread of execution. One is to declare a class to be a subclass of `Thread`. This subclass should override the `run` method of class `Thread`. An instance of the subclass can then be allocated and started. For example, a thread that computes primes larger than a stated value could be written as follows:

```
class PrimeThread extends Thread {
    long minPrime;
    PrimeThread(long minPrime) {
        this.minPrime = minPrime;
    }

    public void run() {
        // compute primes larger than minPrime
         . . .
    }
}
```

The following code would then create a thread and start it running:

```
PrimeThread p = new PrimeThread(143);
p.start();
```

CIS 36B – 19th Class / Lab Assignment – **10 Points-**

**Student Name** _____ **Student ID** _____ **Point Total**

The other way to create a thread is to declare a class that implements the `Runnable` interface. That class then implements the `run` method. An instance of the class can then be allocated, passed as an argument when creating `Thread`, and started. The same example in this other style looks like the following:

```
class PrimeRun implements Runnable {
    long minPrime;
    PrimeRun(long minPrime) {
        this.minPrime = minPrime;
    }

    public void run() {
        // compute primes larger than minPrime
         . . .
    }
}
```

The following code would then create a thread and start it running:

```
PrimeRun p = new PrimeRun(143);
new Thread(p).start();
```

Every thread has a name for identification purposes. More than one thread may have the same name. If a name is not specified when a thread is created, a new name is generated for it.

Unless otherwise noted, passing a `null` argument to a constructor or method in this class will cause a `NullPointerException` to be thrown.

## start()-

Causes this thread to begin execution; the Java Virtual Machine calls the `run` method of this thread.

The result is that two threads are running concurrently: the current thread (which returns from the call to the `start` method) and the other thread (which executes its `run` method).

It is never legal to start a thread more than once. In particular, a thread may not be restarted once it has completed execution.

CIS 36B – 19th Class / Lab Assignment – **10 Points-**

**Student Name** _____ **Student ID** _____ **Point Total**

## run()-

If this thread was constructed using a separate `Runnable` run object, then that `Runnable` object's `run` method is called; otherwise, this method does nothing and returns.

Subclasses of `Thread` should override this method.

## sleep()-

Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds plus the specified number of nanoseconds, subject to the precision and accuracy of system timers and schedulers. The thread does not lose ownership of any monitors.

**Parameters:**

`millis` - the length of time to sleep in milliseconds

`nanos` - `0–999999` additional nanoseconds to sleep

## Task 1:

## USE OBJECT ORIENTATED PROGRAM DESIGN TO SOLVE PROBLEM

Create basic thread example that executes multiple threads.

The thread example should extend **Thread()** class.

then the program should have the same output

**implement** Runnable

.

CIS 36B – 19th Class / Lab Assignment – **10 Points-**

**Student Name** _____ **Student ID** _____ **Point Total**

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6
7  package javaapplication2;
8
9  class MyThread extends Thread {
10
11     public static int mycount = 0;
12
13     @Override
14     public void run(){
15
16        while(MyThread.mycount <= 10) {
17           try {
18              System.out.println("####: " + (++MyThread.mycount));
19              Thread.sleep(100);
20           } catch(InterruptedException e) {
21
22              System.out.println("Thread interrupted");
23
24           }
25        }
26
27     }
28
29
```

CIS 36B – 19th Class / Lab Assignment – **10 Points-**

**Student Name** _____ **Student ID** _____ **Point Total**

```
30  }
31  public class Thread1 {
32
33      /**
34       * @param args the command line arguments
35       */
36      public static void main(String[] args) {
37
38          System.out.println("Main Tread is starting");
39
40          MyThread a = new MyThread();
41          MyThread b = new MyThread();
42          MyThread c = new MyThread();
43
44          a.start();
45          b.start();
46          c.start();
47
48          while(MyThread.mycount <= 10) {
49              try {
50                 System.out.println("Main Thread: " + (++MyThread.mycount));
                   Thread.sleep(100);
52              } catch(InterruptedException e) {
53
54                 System.out.println("Main Thread interrupted");
55
56              }
57          }
58
59          System.out.println("Main Tread is terminating");
60      }
61
62  }
63
```

CIS 36B – 19th Class / Lab Assignment – **10 Points-**

**Student Name** _____ **Student ID** _____ **Point Total**