CECS 277 – Lecture 17 – Threads

A multi-threaded program is one that is written so that different parts of the program can be run concurrently. This is done to better utilize the CPU to minimize idle time, allow the computer to multitask, and now with multi-core processors, a task can be divided up to be executed on multiple processors.

The Thread class can be used to create threads for your class. Thread implements the class Runnable which has a function called run() to define what your thread should do.

Function	Description
join()	Waits for the thread to finish.
start()	Starts the thread by calling your run() method and then immediately returns (does not wait for method to finish)
sleep(ms)	Pauses the thread for the given number of milliseconds (1000 ms = 1 s)

There are two basic ways of creating a new thread:

1. Extend the Thread class to make a new Thread to run.
2. Implement the Runnable interface and then create a Thread to use your class.

Both of the above will have the same effect.

Example: Extending the Thread class.

```
public class ThreadX extends Thread {
    private String str;
    public ThreadX( String s ) {
        str = s;
    }
    public void run(){ //overridden from Runnable
        try{
            for( int i = 0; i < 5; i++ ) {
                System.out.print( str + ": " + i );
                Thread.sleep( 1000 );
            }
        } catch( InterruptedException e ) {
            System.out.println( str + " Interrupted" );
        }
    }
}
public class ThreadMain{
    public static void main( String [] args ) {
        ThreadX a = new ThreadX( "A" );
        ThreadX b = new ThreadX( "B" );
        a.start();
        b.start();
    }
}
```

Example: Implementing the Runnable interface.

```
public class RunnerX implements Runnable {
    private String str;
    public RunnerX( String s ) {
        str = s;
    }
    @Override
    public void run(){
        try{
            for( int i = 0; i < 5; i++ ) {
                System.out.print( str + ": " + i );
                Thread.sleep( 1000 );
            }
        } catch( InterruptedException e ) {
            System.out.println( str + " Interrupted" );
        }
    }
}
public class ThreadMain{
    public static void main( String [] args ) {
        Thread a = new Thread( new RunnerX( "A" ) );
        Thread b = new Thread( new RunnerX( "B" ) );
        a.start();
        b.start();
    }
}
```

Anonymous Inner Class – An anonymous inner class is a class that can be declared without creating a whole new class. It is often useful when making an object that requires these overridden methods, but it would be unnecessary to create an entire class for it, since it will only ever be used once.

Example: Both as anonymous inner classes. Not a good example of usage since they both do the same thing, but it shows how to make both thread and runnable inner classes

```
public class ThreadMain {
    public static void main( String [] args ) {
        Thread a = new Thread( ) {
            public void run() {
                try {
                    for( int i = 0; i < 5; i++ ) {
                        System.out.print( "A: " + i );
                        Thread.sleep( 1000 );
                    }
                } catch( InterruptedException e ) {
                    System.out.println( "A Interrupted" );
                }
            }
        };
```

```
        Thread b = new Thread( new Runnable() {
            public void run(){
                try{
                    for( int i = 0; i < 5; i++ ) {
                        System.out.print( "B: " + i );
                        Thread.sleep( 1000 );
                    }
                } catch( InterruptedException e ){
                    System.out.println( "B Interrupted" );
                }
            }
        });
        a.start();
        b.start();
    }
}
/*
A: 0
B: 0
A: 1
B: 1
A: 2
B: 2
A: 3
B: 3
A: 4
B: 4
*/
```