Java Thread Creation

To create a new thread in Java:

1. Create new class that implements `Runnable`, and place logic in a public, `void`, no-argument method named `run`
2. Instantiate an object of the new class
3. Instantiate an object of type `Thread`; pass the new object as the argument to `Thread`'s constructor
4. Call `start()` method of the `Thread` object

Example

New class:

```
public class ThreadExample implements Runnable {
    public void run() {
        System.out.println("I am a new thread!");
    }
}
```

Instantiations:

```
> ThreadExample x = new ThreadExample()
> Thread t = new Thread(x)
```

Call `start`:

```
> t.start()
I am a new thread!
```

This is Java's equivalent of

`pthread_create(&t, NULL, run, NULL)`

Exercise: Start vs. Run

Q: Why call `t.start()` (which then calls `run`)? Why not just call `run()`?

For example ...

```
public class ThreadExample implements Runnable {
    public void run() {
        System.out.println("I am a new thread!");
    }
}

> ThreadExample x = new ThreadExample()
> x.run()
```

Thread Arguments

Q: How to pass arguments to new thread?

A: Pass arguments to constructor; `run` takes no arguments

Java Thread Creation, II

There is another way to create a new thread (discouraged):

1. Create new class that extends `Thread`, and place logic in a public, `void`, no-argument method named `run`
2. Instantiate an object of the new class
3. Call object's `start()` method

Join, I

Java also has the equivalent of `pthread_join`:

```
> ThreadExample x = new ThreadExample()
> Thread t = new Thread(x)
> t.start()
I am a new thread!
> t.join()
> t.getState()
TERMINATED
```

In this case, at moment when `t.join()` call was made the Thread had already ended