Threading Review

Creating a Thread

Thread

```
public class Counter extends Thread{
  private int value;

  Counter () {
    value = 0;
  }

  public int getValue () {
    return value;
  }

  public void count () {
    value++;
  }

  public void run () {
    count ();
  }

  public static void main (String[] args){
    Counter counter = new Counter ();
    new Thread ().start ();
    System.out.println (counter.getValue ());
  }
}
```

Runnable

```
public class Counter implements Runnable{
  private int value;

  Counter () {
    value = 0;
  }

  public int getValue () {
    return value;
  }

  public void count () {
    value++;
  }

  public void run () {
    count ();
  }

  public static void main (String[] args){
    Counter counter = new Counter ();
    new Thread (counter).start ();
    System.out.println (counter.getValue ());
  }
}
```

Lambda

```
public class Counter{
  private int value;

  Counter () {
    value = 0;
  }

  public int getValue () {
    return value;
  }

  public void count () {
    value++;
  }

  public static void main (String[] args){
    new Thread(()->count()).start ();
    System.out.println (counter.getValue ());
  }
}
```

Activity: Counter (100)

- Using the previous slide as an example:
  - Create a Counter.java file that starts 100 counters and prints out the resulting Counter value.
  - The run method for each counter should only add one count.

DISCUSS RESULTS

Activity: Counter (10000)

- Update counter to:
  - Start 10,000 counters and print out the resulting Counter value.
  - The run method for each counter should only add one count.

DISCUSS RESULTS

Sleep

```
public class Counter{
  private int value;

  Counter () {
    value = 0;
  }

  public int getValue () {
    return value;
  }

  public void count () {
    value++;
  }

  public static void main (String[] args)
    throws InterruptedException {
    new Thread(()->count()).start ();
    Thread.sleep (500) // Sleep for 500 ms
    System.out.println (counter.getValue ());
  }
}
```

- `Thread.sleep(ms)` is used to force the current thread to give up the CPU for `ms` number of milliseconds
- After the `sleep` window, the thread can the be scheduled to run again
  - It will most likely not start immediately

Activity: Sleepy Counter

- Update your code to add a 1 second sleep after starting all the threads but before printing the count
- Do you get the correct count now?
- If not, try a larger sleep window
  - If you get above 10 seconds you can stop

DISCUSS RESULTS

Activity: Counter (1000000)

- Update your counter to create 1 Million threads
- How does this impact the results?

DISCUSS RESULTS

Join

```
public class Counter{
  private int value;

  Counter () {
    value = 0;
  }

  public int getValue () {
    return value;
  }

  public void count () {
    value++;
  }

  public static void main (String[] args)
    throws InterruptedException {
    Thread thread = new Thread(()->count());
    thread.start ();
    thread.join ();
    System.out.println (counter.getValue ());
  }
}
```

- `join()` causes the current thread to give up CPU time until the thread it is joining completes

Activity: Counter With Join

- Update your example to use `join` instead of `sleep`
  - You cannot `join` in the same loop as you are creating the threads otherwise you will single-thread your application

DISCUSS RESULTS

Activity: Counter (100x100)

- Instead of creating one million threads:
  - Update your code to create 100 threads
  - Each thread should call count 100 times

DISCUSS RESULTS

Activity: Counter (1000x100)

- Update your code to have each thread count 1000 times
  - Still create 100 threads
- Did you still get the correct count at the end?

DISCUSS RESULTS

Activity: Counter (10000x100)

- Lastly, update the threads run so that it counts to 10,000
  - Still start 100 threads
- Is your count still correct at the end?
  - If it's not, why is that the case

DISCUSS RESULTS

Discussion

- Why is creating 1000000 single-count threads so much slower than creating 100 10000-count threads?
- Why did 100 10K-count threads loose so many values in comparison to 1000000 single-count threads?
- How to fix this?