* Concurrency

1. Implement Singleton with double checked locking.
2. Write a program to create a deadlock between two threads.
3. Write a program with one thread (the producer) that produces items (in this case simple int's). Another thread (the consumer) consumes items. For communication purposes both threads have access to a bounded buffer which is basically an array.

* Junit

For this exercise you will add Junit tests for the Line Class below:

|  |
| --- |
| // Line.java  public class Line {  // construct a line object  public Line(double x0, double y0, double x1, double y1) {  this.x0 = x0;  this.y0 = y0;  this.x1 = x1;  this.y1 = y1;  }  // calculate the slope of the line  public double getSlope( ) {  // avoid dividing by zero  if(x1 == x0) {  throw new ArithmeticException( );  }  return (y1 - y0) / (x1 - x0);  }  // calculate the distance of the line  public double getDistance( ) {  return Math.sqrt((x1 - x0) \* (x1 - x0) + (y1 - y0) \* (y1 - y0));  }  // return whether a line is parallel to another  public boolean parallelTo(Line l) {  // if the difference between the slopes is very small, consider them parallel  if(Math.abs(getSlope( ) - l.getSlope( )) < .0001) {  return true;  } else {  return false;  }  }  // private member data  private double x0, y0, x1, y1;  } |

1. Create a file called LineTest.java.
2. Create tests for the getSlope, getDistance and parallelTo methods.
3. Because of rounding errors, it is bad practice to test double values for exact equality. To get around this, you can pass a small value (such as .0001) to assertEquals to be used as a delta.