```
import java.util.Arrays;
import java.util.Random;


public class GradientDescent {
   static double[] weight = new double[2];
   static final double learningRate = 0.001;
   //I created this array because it made the math I'm performing in the below functions a lot easier.
   static final int[] x0 = {1,1,1,1,1,1,1,1};
//Here I calculate the weighted sum for every data point in the training set. It gets called every time I
   //calculate the log-likelihood
   public static double weightedSum(int x) {
      double weightedSum;
      weightedSum = weight[0] + (weight[1] * x);
      return weightedSum;
   }
   // This is where I calculate the log-likelihood
   public static double logLinearModel(int n, int[] x, int[] y) {
      double deltaWeight = 0;
      double[] numerator = new double[n];
      double[] denominator = new double[n];

      for(int j = 0; j < n; j++) {
         numerator[j] = (-1 * y[j] * x[j]);
         denominator[j] = (1 + Math.exp(y[j] *  weightedSum(x[j])));
         deltaWeight += (numerator[j] / denominator[j]);
      }
      deltaWeight *= (1.0 / n);

      return deltaWeight;
   }
   //This function calls the log-likelihood function and uses it to update the weights.
   public static void weightUpdateFcn(int n, int[] x, int[] y) {
      weight[0] = weight[0] - (learningRate * logLinearModel(n, x0, y));
      weight[1] = weight[1] - (learningRate * logLinearModel(n,x,y));
   }

   public static double getWeight0() {
      return weight[0];
   }
   public static double[] getWeight() {return weight;}

}

public class StochasticGradientDescent {
   //Everything in this class operates the same as the GradientDescent class, with the exception of
   //calling the log-likelihood.
   static double[] weight = new double[2];
   static final double learningRate = 0.001;
   static final int[] x0 = {1,1,1,1,1,1,1,1};

   public static double weightedSum(int x) {
      double weightedSum;
      weightedSum = weight[0] + (weight[1] * x);
      return weightedSum;
```

```
    }
    //The only difference here is that I didn't need to iterate through the training data array, so there's
    //no loop.
    public static double logLinearModel(int n, int x, int y) {
        double deltaWeight = 0;
        double numerator = (-1 * y * x);
        double denominator = (1 + Math.exp(y * weightedSum(x)));
        deltaWeight += (numerator / denominator);
//      deltaWeight *= (1.0 / n);

        return deltaWeight;
    }
    //Instead of passing the whole training set when updating weights, I used the Random package to choose one
    //of the data points and only passed that one point.
    public static void weightUpdateFcn(int n, int[] x, int[] y) {
        int rnd = new Random().nextInt(x.length);

        weight[0] = weight[0] - (learningRate * logLinearModel(n, 1, y[rnd]));
        weight[1] = weight[1] - (learningRate * logLinearModel(n,x[rnd],y[rnd]));
    }

    public static double getWeight0() {
        return weight[0];
    }
    public static double[] getWeight() {return weight;}

}


public class Main {
    public static void main(String[] args) {
        int size = 8;
        int[] x = {1,2,3,4,5,6,7,8};
        int[] y = {0,1,0,1,0,1,1,1};
        //I chose this tolerance level because I wanted something that would go out at least 5 decimal places.
        double error = 1;
        double tolerance = Math.exp(-10);
        int count = 0;
// Below here is where I perform Gradient Descent. More comments in the class I created for it.
        while(error > tolerance) {
            double temp = GradientDescent.getWeight0();
            GradientDescent.weightUpdateFcn(size,x,y);
            error = Math.abs(temp - GradientDescent.getWeight0());
            count++;
        }
        System.out.println(Arrays.toString(GradientDescent.getWeight()));
        System.out.println(count);
// Below here is where I perform Stochastic Gradient Descent. More comments in the class I created for it.
        count = 0;
        // I was having trouble finding a good way to look for the error when I was only adjusting after checking
        // one data point. However, once I knew the code was working, I used the fact that SGD and GD should
        //converge to the same answer, and compared the two of them using newTolerance below.
        double newTolerance = 0.01;

        while(Math.abs(StochasticGradientDescent.getWeight0()-GradientDescent.getWeight0()) > newTolerance) {
```

```
            StochasticGradientDescent.weightUpdateFcn(size,x,y);
            count++;
        }
        System.out.println(Arrays.toString(StochasticGradientDescent.getWeight()));
        System.out.println(count);
    }
}
```