```
package repressilator;

/*
 * Reconstruction of the Repressilator model found in
 * A Synthetic Oscilatory Network of Transcriptonal
 * Regulators (Elowitz M, et al.) -- Nature vol. 403,
 * p. 335-338, 2000.
 */
public class deterministic
{
    // Setup
    static double[]     vals;
    static double[]     k1, k2, k3, k4;
    // Parameter values
    static final double hill        = 2;
    static final double prothalf    = 10;
    static final double mrnahalf    = 2;
    static final double h           = .0005;
    static final double alpha       = 10000;
    static final double alpha0  = 0;
    static final double beta        = prothalf / mrnahalf;
    static final double ttot        = 100;
    static final double STEPS       = ttot / h;

    public static void main(String[] args)
    {
       // Initialize stuff
       vals = new double[6];
       k1 = new double[6];
       k2 = new double[6];
       k3 = new double[6];
       k4 = new double[6];
       for (int i = 0; i < vals.length; i++)
          vals[i] = Math.random() * 600;

       System.out.print(0 + "\t");
       for (int i = 0; i < vals.length; i++)
          System.out.print(vals[i] + "\t");
       System.out.println();

       // Simulate development process.
       for (int i = 0; i < STEPS; i++)
       {
          double t = i * h;

          // Calculate RK coefficients.
          double[] dvals = derivs(t, vals);
          for (int j = 0; j < k1.length; j++)
             k1[j] = dvals[j];
          double[] tempvals = new double[6];
          for (int j = 0; j < tempvals.length; j++)
             tempvals[j] = vals[j] + h * k1[j] / 2;
          dvals = derivs(t + h / 2, tempvals);
          for (int j = 0; j < tempvals.length; j++)
             k2[j] = dvals[j];
          for (int j = 0; j < tempvals.length; j++)
             tempvals[j] = vals[j] + h * k2[j] / 2;
          dvals = derivs(t + h / 2, tempvals);
          for (int j = 0; j < tempvals.length; j++)
             k3[j] = dvals[j];
          for (int j = 0; j < tempvals.length; j++)
             tempvals[j] = vals[j] + h * k3[j];
          dvals = derivs(t + h, tempvals);
          for (int j = 0; j < tempvals.length; j++)
             k4[j] = dvals[j];

          // Move all values one step in time according to 4th order RK.
          for (int j = 0; j < vals.length; j++)
             vals[j] += h * (k1[j] / 6 + k2[j] / 3 + k3[j] / 3 + k4[j] / 6);

          // Print output. Disallow negative values.
          System.out.print(t + "\t");
          for (int j = 0; j < vals.length; j++)
          {
             if (vals[j] < 0)
                vals[j] = 0;
             System.out.print(vals[j] + "\t");
          }
          System.out.println();
       }
    }

    /*
     * Computes derivatives according to the repressilator ODE's. Note that time
     * is completely aesthetical in this case.
     */
    public static double[] derivs(double t, double[] vals)
    {
       double[] dvals = new double[6];
       // compute p derivatives
       for (int i = 0; i < dvals.length / 2; i++)
          dvals[i] = -beta * (vals[i] - vals[i + 3]);
       // compute m derivatives
       for (int i = dvals.length / 2; i < dvals.length; i++)
          dvals[i] = -vals[i] + alpha / (1 + Math.pow(vals[(i + 2) % 3], hill))
                + alpha0;
       return dvals;
    }

}
```