```java
import java.awt.Color;
import java.awt.Font;
import java.util.ArrayList;

class Canvas {
   int x = 960, y = 480;
   double[] xScale = { 0, 1.0 };  // MIN, MAX
   double[] yScale = { 0, 1.0 };  // MIN, MAX
   Color bgColor = Color.WHITE;
   Color color = Color.BLACK;
}

class Formats {
   double[] margins = { 0.05, 0.05, 0.05, 0.05 };  // NORTH, SOUTH, WEST, EAST
   boolean isBarFilled = true;
   Color barFillColor = new Color(0x32d3eb);
   boolean hasBarFrame = false;
   Color barFrameColor = new Color(0x60acfc);
   boolean hasBorder = false;
   Color borderColor = new Color(180,180,180);
   Color rulerColor = new Color(100, 100, 100);
   Color rulerMarkColor = new Color(0xf7f7f7);
   boolean hasRightRuler = false;
   Color keyColor = new Color(0x333333);
   boolean hasHeader = false;
   Color headerColor = new Color(0x333333);
   boolean hasFooter = false;
   Color footerColor = new Color(0x333333);
   Font rulerFont =  new Font( "consolas", Font.PLAIN, 12 );
   Font keyFont = new Font( "consolas", Font.PLAIN, 12 );
   Font headerFont = new Font( "calibri", Font.PLAIN, 20 );
   Font footerFont = new Font( "calibri", Font.PLAIN, 20 );
   String rulerNumberFormat = null;
}

class Item {
   double value;
   double index;
}
```

```java

class HistogramData {
   String header = "";
   String footer = "";
   double minValue = 0.0;
   String[] keys = { };
   ArrayList<double[]> state;
   double[] values;
   Item[][] items;
   int itemIndex = 0;
}

public class Histogram {
   Canvas c;
   Formats f;
   HistogramData d;
   double[] xValue;  // MIN, MAX
   double[] yValue;  // MIN, MAX
   double[] xScale;  // MIN, MAX
   double[] yScale;  // MIN, MAX
   int rulerGrade;
   double rulerStep;

   public Histogram(Canvas c, Formats f, HistogramData d) {
      this.c = c;
      this.f = f;
      this.d = d;
      xValue = new double[2];
      yValue = new double[2];
      xScale = new double[2];
      yScale = new double[2];
   }

   private void preCompute() {
      int n_state = d.state.size();
      int n_nums = d.state.get(0).length;
      this.n_nums = n_nums;
      int n_items = (n_state - 1) * INTERP_COUNT + 1;
      d.items = new Item[n_items][n_nums];
      for (int i = 0; i < n_state; i++) {
            for (int j = 0; j < n_nums; j++) {
                d.items[i*INTERP_COUNT][j] = new Item();
                Item item = d.items[i*INTERP_COUNT][j];
                item.value = d.state.get(i)[j];
                item.index = j;
            }
            if (i > 0) interpolate(i - 1);
        }
    }

    private void interpolate(int i) {
        int nLabel = n_nums;
        for (int j = 0; j < nLabel; j++) {
            Item item = d.items[(i) * INTERP_COUNT][j];
            Item[] nextItems = d.items[(i+1)* INTERP_COUNT];
            // find the matching one TODO this algorithm should be improved (or improve data structure?) otherwise the complexity would explode
            Item nextItem = null;
            for (int k = 0; k < nLabel; k++) {
                if (item.value == nextItems[k].value) {
                    nextItem = nextItems[k];
                    break;
                }
            }
            double vStep = (nextItem.value - item.value) / INTERP_COUNT;
            double gSpan = nextItem.index - item.index;

            for (int m = 1; m < INTERP_COUNT; m++) {
                d.items[(i) * INTERP_COUNT + m][j] = new Item();
                Item targetItem = d.items[(i) * INTERP_COUNT + m][j];
//                 targetItem.labelIndex = item.labelIndex;
                targetItem.value = item.value + vStep * m;
                // non-linear interpolate ref:http://inloop.github.io/interpolator?Library=AccelerateDecelerate
                targetItem.index = item.index + ((Math.cos(((double)m / INTERP_COUNT + 1) * Math.PI) / 2.0) + 0.5) * gSpan;

            }
        }
    }

    private double[] getValues(int i) {
        double[] values = new double[n_nums];
        for (int j = 0; j < n_nums; j ++) {
            values[j] = d.items[i][j].value;
        }
        return values;
    }

    private void setHistogramParameters () {
        Item[] a = d.items[0];
        xValue[MIN] = -1;
        xValue[MAX] = a.length;

        yValue[MIN] = d.minValue;

        double max = a[0].value;
        for (int i = 1; i < a.length; i++)
            if (max < a[i].value) max = a[i].value;

        double span = max - yValue[MIN];
        double factor = 1.0;
        if (span >= 1)
            while (span >= 10) { span /= 10; factor *= 10; }
        else
            while (span < 1)   { span *= 10; factor /= 10; }
        int nSpan = (int)Math.ceil(span);
        yValue[MAX] = yValue[MIN] + factor * nSpan;
        switch (nSpan) {
            case 1 :  rulerGrade = 5; rulerStep = factor/5; break;
            case 2 :
            case 3 :  rulerGrade = nSpan*2; rulerStep = factor/2; break
;
            default : rulerGrade = nSpan; rulerStep = factor; break;
        }
    }

    public void draw () {
//      setCanvas();
```

```java
        plotBars();
        plotRuler();
//        plotKeys();
    }

    public void animate() {
        preCompute();
        d.values = getValues(0);
        setHistogramParameters();
        StdDraw.enableDoubleBuffering();
        setCanvas();
        int n = d.items.length;
        for (int i = 0; i < n; i++) {
            StdDraw.clear();
            d.itemIndex = i;
            draw();
            StdDraw.show();
            StdDraw.pause(5);
        }
    }

    private void setCanvas () {
        StdDraw.setCanvasSize( c.x,  c.y );
        setOriginalScale();
        StdDraw.clear( c.bgColor);
        StdDraw.setPenColor( c.color);
    }

    private void setHistogramScale (int nBars) {
        double span = yValue[MAX] - yValue[MIN] + 1;
        double ySpacing = span / (1 - f.margins[NORTH] - f.margins[
SOUTH]);
        yScale[MIN] = yValue[MIN] - f.margins[SOUTH] * ySpacing - 1;
        yScale[MAX] = yValue[MAX] + f.margins[NORTH] * ySpacing;
        StdDraw.setYscale( yScale[MIN], yScale[MAX]);

        double xSpacing = (nBars+1) / (1 - f.margins[WEST] - f.margins[
EAST]);
        xScale[MIN] = xValue[MIN]- f.margins[WEST] * xSpacing - 1;
        xScale[MAX] = nBars + f.margins[EAST] * xSpacing;
```

```java
      StdDraw.setXscale( xScale[MIN], xScale[MAX]);
   };

   private void setOriginalScale() {
      StdDraw.setXscale( c.xScale[MIN], c.xScale[MAX]);
      StdDraw.setYscale( c.yScale[MIN], c.yScale[MAX]);
   }

   private void plotBars () {
      Item[] a = d.items[d.itemIndex];
      int n = a.length;
      setHistogramScale( n );
      if (f.isBarFilled) {
         StdDraw.setPenColor( f.barFillColor);
         for (int i = 0; i < n; i++) {
            StdDraw.filledRectangle(a[i].index, (a[i].value + d.minValue)/2, 0.25, (a[i].value - d.minValue)/2);
                           // (x, y, halfWidth, halfHeight)
            // the minValue bug have been fixed
         }
      }
      if (f.hasBarFrame) {
         StdDraw.setPenColor( f.barFrameColor);
         for (int i = 0; i < n; i++) {
            StdDraw.rectangle(a[i].index, (a[i].value + d.minValue)/2, 0.25, (a[i].value - d.minValue)/2);
                        // (x, y, halfWidth, halfHeight)
         }
      }
   }

   private void plotRuler() {
//      Font font = new Font( "consolas", Font.PLAIN, 12 ); // TO BE Customized
      StdDraw.setFont( f.rulerFont );
      StdDraw.setPenColor( f.rulerColor );
      final double x0 = xValue[MIN] - 0.05, x1 = xValue[MIN] + 0.05;
      String[] mark = new String[rulerGrade+1];
      for (int i = 0; i <= rulerGrade; i++) {
         double y = yValue[MIN] + i * rulerStep;
         mark[i] = numberForRuler( y );
         StdDraw.line( x0, y, x1, y );
      }
      int len = maxMarkLength( mark );
      final double xs = xScale[MIN] + 0.7 * (xValue[MIN] - xScale[MIN]);
      for (int i = 0; i <= rulerGrade; i++) {
         double y = yValue[MIN] + i * rulerStep;
         StdDraw.text( xs, y, String.format( "%" + len + "s", mark[i] ));
      }
   }

   private String numberForRuler (double x) {   // TO BE Customized
      if (f.rulerNumberFormat != null) return String.format(f.rulerNumberFormat, x); // only accept formats for double type!
      if (yValue[MAX] >= 5 && rulerStep > 1) return "" + (int)x;
      if (rulerStep > 0.1) return String.format( "%.1f", x );
      if (rulerStep > 0.01) return String.format( "%.2f", x );
      if (rulerStep > 0.001) return String.format( "%.3f", x );
      if (rulerStep > 0.0001) return String.format( "%.4f", x );
      if (rulerStep > 0.00001) return String.format( "%.5f", x );
      return String.format( "%g", x );
   }

   private int maxMarkLength (String[] sa) {
      int n = sa[0].length();
      for (String s : sa)
         if (n < s.length()) n = s.length();
      return n;
   }


   private final static int NORTH = 0;
   private final static int SOUTH = 1;
   private final static int WEST  = 2;
   private final static int EAST  = 3;
   private final static int MIN   = 0;
   private final static int MAX   = 1;

```

```
    private final static int INTERP_COUNT = 36;
    private int n_nums;
}

```