```java
/**
 * Android: TouchImageView.java
 * Created by: Mike Ortiz
 * Updated by: Vince Pascuzzi
 * Date: 3/14/2013
 *
 * Allows pinching, zooming, translating, and drawing on an ImageView.
 */

package edu.seaaddicts.brockbutler.maps;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Matrix;
import android.graphics.Paint;
import android.graphics.PointF;
import android.graphics.drawable.Drawable;
import android.util.AttributeSet;
import android.util.Log;
import android.view.MotionEvent;
import android.view.ScaleGestureDetector;
import android.view.View;
import android.widget.ImageView;

public class MapsTouchImageView extends ImageView {
  //private static final String TAG = "MapsTouchImageView";

  @SuppressWarnings("unused")
  private static final int MAP_WIDTH = 2000;
  private static final int MAP_HEIGHT = 1100;
  private static final int CLICK = 3;

  private Matrix mMatrixMap;

  // States of touch.
  private static final int NONE = 0;
  private static final int DRAG = 1;
  private static final int ZOOM = 2;
  private int mode = NONE;
  private int stroke = 8;

  // Zooming variables.
  private PointF last = new PointF();
  private PointF start = new PointF();
  private float minScale = 1f;
  private float maxScale = 8f;
  private float[] m;

  // Ratio of screen resolution to map image resolution
  private double mMapRatio;

  private int viewWidth, viewHeight;
  @SuppressWarnings("unused")
  private int oldMeasuredWidth, oldMeasuredHeight;

  private float scaleFactor = 1f;
  private float origWidth, origHeight;

  private final Paint mPathPaint = new Paint();

  private ScaleGestureDetector mScaleDetector;

  //private Context mContext;
  int actionBarHeight;

  public Position[] mPosition = null;

  public MapsTouchImageView(Context context) {
    super(context);
    sharedConstructing(context);
  }

  public MapsTouchImageView(Context context, AttributeSet attrs) {
    super(context, attrs);
    sharedConstructing(context);
  }

  @Override
  protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    mPathPaint.setColor(Color.CYAN);
    mPathPaint.setStrokeWidth(stroke);
    canvas.setMatrix(mMatrixMap);

    if (mPosition != null) {
      for (int i = 0; i < mPosition.length - 1; i++) {
        Position p = mPosition[i];
        Position q = mPosition[i + 1];
        int x1 = p.xPosition;
        int y1 = p.yPosition;
        float[] f1 = convertDimensions(x1, y1);
        int x2 = q.xPosition;
        int y2 = q.yPosition;
        float[] f2 = convertDimensions(x2, y2);
        canvas.drawLine(f1[0], f1[1], f2[0], f2[1], mPathPaint);
      }
    }
  }

  private void sharedConstructing(Context context) {
    super.setClickable(true);
    //this.mContext = context;
    mScaleDetector = new ScaleGestureDetector(context, new ScaleListener());
    mMatrixMap = new Matrix();
    m = new float[9];
    setImageMatrix(mMatrixMap);
    setScaleType(ScaleType.MATRIX);

    setOnTouchListener(new OnTouchListener() {

      public boolean onTouch(View v, MotionEvent event) {
        mScaleDetector.onTouchEvent(event);
        PointF curr = new PointF(event.getX(), event.getY());

        switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
          last.set(curr);
          start.set(last);
          mode = DRAG;
          break;

        case MotionEvent.ACTION_MOVE:
          float fixTransX;
          float fixTransY;
          if (mode == DRAG) {
            float deltaX = curr.x - last.x;
            float deltaY = curr.y - last.y;
            fixTransX = getFixDragTrans(deltaX, viewWidth,
                origWidth * scaleFactor);
            fixTransY = getFixDragTrans(deltaY, viewHeight,
                origHeight * scaleFactor);
            mMatrixMap.postTranslate(fixTransX, fixTransY);
            fixTrans();
            last.set(curr.x, curr.y);
          }
          break;

        case MotionEvent.ACTION_UP:
          mode = NONE;
          int xDiff = (int) Math.abs(curr.x - start.x);
          int yDiff = (int) Math.abs(curr.y - start.y);

          if (xDiff < CLICK && yDiff < CLICK)
            performClick();
          break;

        case MotionEvent.ACTION_POINTER_UP:
          mode = NONE;
          break;
        }

        setImageMatrix(mMatrixMap);
        invalidate();
        return true; // indicate event was handled
      }

    });
  }

  public void setMaxZoom(float x) {
    maxScale = x;
  }

  private class ScaleListener extends
      ScaleGestureDetector.SimpleOnScaleGestureListener {
    @Override
    public boolean onScaleBegin(ScaleGestureDetector detector) {
      mode = ZOOM;
      return true;
    }

    @Override
    public boolean onScale(ScaleGestureDetector detector) {
      float mScaleFactor = detector.getScaleFactor();
      float origScale = scaleFactor;
      scaleFactor *= mScaleFactor;
      if (scaleFactor > maxScale) {
        scaleFactor = maxScale;
        mScaleFactor = maxScale / origScale;
      } else if (scaleFactor < minScale) {
        scaleFactor = minScale;
        mScaleFactor = minScale / origScale;
      }

      if (origWidth * scaleFactor <= viewWidth
          || origHeight * scaleFactor <= viewHeight)
        mMatrixMap.postScale(mScaleFactor, mScaleFactor, viewWidth / 2,
            viewHeight / 2);
      else
        mMatrixMap.postScale(mScaleFactor, mScaleFactor,
            detector.getFocusX(), detector.getFocusY());
      fixTrans();
      return true;
    }
  }

  void fixTrans() {
    mMatrixMap.getValues(m);
    float fixTransX;
    float fixTransY;
    float transX = m[Matrix.MTRANS_X];
    float transY = m[Matrix.MTRANS_Y];

    fixTransX = getFixTrans(transX, viewWidth, origWidth * scaleFactor);
    fixTransY = getFixTrans(transY, viewHeight, origHeight * scaleFactor);

    if (fixTransX != 0 || fixTransY != 0)
      mMatrixMap.postTranslate(fixTransX, fixTransY);
  }

  /*
   * Fixes (when required) the translation matrix.
```

```java
     */
    float getFixTrans(float trans, float viewSize, float contentSize) {
      float minTrans, maxTrans;

      if (contentSize <= viewSize) {
        minTrans = 0;
        maxTrans = viewSize - contentSize;
      } else {
        minTrans = viewSize - contentSize;
        maxTrans = 0;
      }

      if (trans < minTrans)
        return -trans + minTrans;
      if (trans > maxTrans)
        return -trans + maxTrans;
      return 0;
    }

    /*
     * Adjusts the translation when dragging so that this stays in the correct
     * location on screen.
     */
    float getFixDragTrans(float delta, float viewSize, float contentSize) {
      if (contentSize <= viewSize) {
        return 0;
      }
      return delta;
    }

    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
      super.onMeasure(widthMeasureSpec, heightMeasureSpec);
      viewWidth = MeasureSpec.getSize(widthMeasureSpec);
      viewHeight = MeasureSpec.getSize(heightMeasureSpec);

      // Does image rescaling on rotation. Not necessary since our orientation
      // is fixed in landscape.
      if (oldMeasuredHeight == viewWidth && oldMeasuredHeight == viewHeight
          || viewWidth == 0 || viewHeight == 0)
        return;
      oldMeasuredHeight = viewHeight;
      oldMeasuredWidth = viewWidth;

      if (scaleFactor == 1) {
        // Fit to screen.
        float scale;

        Drawable drawable = getDrawable();
        if (drawable == null || drawable.getIntrinsicWidth() == 0
            || drawable.getIntrinsicHeight() == 0)
          return;
        int bmWidth = drawable.getIntrinsicWidth();
        int bmHeight = drawable.getIntrinsicHeight();

        mMapRatio = (double) (bmHeight) / (double) MAP_HEIGHT;

        Log.d("bmSize", "bmWidth: " + bmWidth + " bmHeight : " + bmHeight
            + "ratio" + mMapRatio);

        float scaleX = (float) viewWidth / (float) bmWidth;
        float scaleY = (float) viewHeight / (float) bmHeight;
        scale = Math.min(scaleX, scaleY);
        mMatrixMap.setScale(scale, scale);

        // Center the image
        float redundantYSpace = (float) viewHeight
            - (scale * (float) bmHeight);
        float redundantXSpace = (float) viewWidth
            - (scale * (float) bmWidth);
        redundantYSpace /= (float) 2;
```

```
            redundantXSpace /= (float) 2;

            mMatrixMap.postTranslate(redundantXSpace, redundantYSpace);

            origWidth = viewWidth - 2 * redundantXSpace;
            origHeight = viewHeight - 2 * redundantYSpace;
            setImageMatrix(mMatrixMap);
        }
        fixTrans();
    }

    float[] convertDimensions(float x, float y) {
        float f[] = new float[2];
        f[0] = (float) mMapRatio * x;
        f[1] = (float) mMapRatio * y;
        return f;
    }

    public void drawPosition(Position[] p, int n) {
        stroke = n;
        mPosition = p;
        invalidate();
    }
}
```