```java
package com.delta.bartalk.util;

import android.app.Activity;
import android.os.Build;
import android.view.View;

/**
 * A utility class that helps with showing and hiding system UI such as the
 * status bar and navigation/system bar. This class uses backward-compatibility
 * techniques described in <a href=
 * "http://developer.android.com/training/backward-compatible-ui/index.html">
 * Creating Backward-Compatible UIs</a> to ensure that devices running any
 * version of ndroid OS are supported. More specifically, there are separate
 * implementations of this abstract class: for newer devices,
 * {@link #getInstance} will return a {@link SystemUiHiderHoneycomb} instance,
 * while on older devices {@link #getInstance} will return a
 * {@link SystemUiHiderBase} instance.
 * <p>
 * For more on system bars, see <a href=
 * "http://developer.android.com/design/get-started/ui-overview.html#system-bars"
 * > System Bars</a>.
 *
 * @see android.view.View#setSystemUiVisibility(int)
 * @see android.view.WindowManager.LayoutParams#FLAG_FULLSCREEN
 */
public abstract class SystemUiHider {
    /**
     * When this flag is set, the
     * {@link android.view.WindowManager.LayoutParams#FLAG_LAYOUT_IN_SCREEN}
     * flag will be set on older devices, making the status bar "float" on top
     * of the activity layout. This is most useful when there are no controls at
     * the top of the activity layout.
     * <p>
     * This flag isn’t used on newer devices because the <a
     * href="http://developer.android.com/design/patterns/actionbar.html">action
     * bar</a>, the most important structural element of an Android app, should
     * be visible and not obscured by the system UI.
     */
    public static final int FLAG_LAYOUT_IN_SCREEN_OLDER_DEVICES = 0x1;

    /**
     * When this flag is set, {@link #show()} and {@link #hide()} will toggle
     * the visibility of the status bar. If there is a navigation bar, show and
     * hide will toggle low profile mode.
     */
    public static final int FLAG_FULLSCREEN = 0x2;

    /**
```

```
     * When this flag is set, {@link #show()} and {@link #hide()} will toggle
     * the visibility of the navigation bar, if it’s present on the device and
     * the device allows hiding it. In cases where the navigation bar is present
     * but cannot be hidden, show and hide will toggle low profile mode.
     */
    public static final int FLAG_HIDE_NAVIGATION = FLAG_FULLSCREEN | 0x4;

    /**
     * The activity associated with this UI hider object.
     */
    protected Activity mActivity;

    /**
     * The view on which {@link View#setSystemUiVisibility(int)} will be called.
     */
    protected View mAnchorView;

    /**
     * The current UI hider flags.
     *
     * @see #FLAG_FULLSCREEN
     * @see #FLAG_HIDE_NAVIGATION
     * @see #FLAG_LAYOUT_IN_SCREEN_OLDER_DEVICES
     */
    protected int mFlags;

    /**
     * The current visibility callback.
     */
    protected OnVisibilityChangeListener mOnVisibilityChangeListener = sDummyListener;

    /**
     * Creates and returns an instance of {@link SystemUiHider} that is
     * appropriate for this device. The object will be either a
     * {@link SystemUiHiderBase} or {@link SystemUiHiderHoneycomb} depending on
     * the device.
     *
     * @param activity The activity whose window’s system UI should be
     *            controlled by this class.
     * @param anchorView The view on which
     *            {@link View#setSystemUiVisibility(int)} will be called.
     * @param flags Either 0 or any combination of {@link #FLAG_FULLSCREEN},
     *            {@link #FLAG_HIDE_NAVIGATION}, and
     *            {@link #FLAG_LAYOUT_IN_SCREEN_OLDER_DEVICES}.
     */
    public static SystemUiHider getInstance(Activity activity, View anchorView, int flags) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
            return new SystemUiHiderHoneycomb(activity, anchorView, flags);
```

```java
        } else {
            return new SystemUiHiderBase(activity, anchorView, flags);
        }
    }

    protected SystemUiHider(Activity activity, View anchorView, int flags) {
        mActivity = activity;
        mAnchorView = anchorView;
        mFlags = flags;
    }

    /**
     * Sets up the system UI hider. Should be called from
     * {@link Activity#onCreate}.
     */
    public abstract void setup();

    /**
     * Returns whether or not the system UI is visible.
     */
    public abstract boolean isVisible();

    /**
     * Hide the system UI.
     */
    public abstract void hide();

    /**
     * Show the system UI.
     */
    public abstract void show();

    /**
     * Toggle the visibility of the system UI.
     */
    public void toggle() {
        if (isVisible()) {
            hide();
        } else {
            show();
        }
    }

    /**
     * Registers a callback, to be triggered when the system UI visibility
     * changes.
     */
    public void setOnVisibilityChangeListener(OnVisibilityChangeListener listener) {
            if (listener == null) {
                listener = sDummyListener;
            }

            mOnVisibilityChangeListener = listener;
        }

        /**
         * A dummy no-op callback for use when there is no other listener set.
         */
        private static OnVisibilityChangeListener sDummyListener = new OnVisibilityChangeListener() {
            @Override
            public void onVisibilityChange(boolean visible) {
            }
        };

        /**
         * A callback interface used to listen for system UI visibility changes.
         */
        public interface OnVisibilityChangeListener {
            /**
             * Called when the system UI visibility has changed.
             *
             * @param visible True if the system UI is visible.
             */
            public void onVisibilityChange(boolean visible);
        }
    }
```

```java
package com.delta.bartalk.util;

import android.app.Activity;
import android.view.View;
import android.view.WindowManager;

/**
 * A base implementation of {@link SystemUiHider}. Uses APIs available in all
 * API levels to show and hide the status bar.
 */
public class SystemUiHiderBase extends SystemUiHider {
    /**
     * Whether or not the system UI is currently visible. This is a cached value
     * from calls to {@link #hide()} and {@link #show()}.
     */
    private boolean mVisible = true;

    /**
     * Constructor not intended to be called by clients. Use
     * {@link SystemUiHider#getInstance} to obtain an instance.
     */
    protected SystemUiHiderBase(Activity activity, View anchorView, int flags) {
        super(activity, anchorView, flags);
    }

    @Override
    public void setup() {
        if ((mFlags & FLAG_LAYOUT_IN_SCREEN_OLDER_DEVICES) == 0) {
            mActivity.getWindow().setFlags(
                    WindowManager.LayoutParams.FLAG_LAYOUT_IN_SCREEN
                            | WindowManager.LayoutParams.FLAG_LAYOUT_NO_LIMITS,
                    WindowManager.LayoutParams.FLAG_LAYOUT_IN_SCREEN
                            | WindowManager.LayoutParams.FLAG_LAYOUT_NO_LIMITS);
        }
    }

    @Override
    public boolean isVisible() {
        return mVisible;
    }

    @Override
    public void hide() {
        if ((mFlags & FLAG_FULLSCREEN) != 0) {
            mActivity.getWindow().setFlags(
                    WindowManager.LayoutParams.FLAG_FULLSCREEN,
                    WindowManager.LayoutParams.FLAG_FULLSCREEN);
        }
```

```java
            mOnVisibilityChangeListener.onVisibilityChange(false);
            mVisible = false;
        }

        @Override
        public void show() {
            if ((mFlags & FLAG_FULLSCREEN) != 0) {
                mActivity.getWindow().setFlags(
                        0,
                        WindowManager.LayoutParams.FLAG_FULLSCREEN);
            }
            mOnVisibilityChangeListener.onVisibilityChange(true);
            mVisible = true;
        }
    }
```

```java
package com.delta.bartalk.util;

import android.annotation.TargetApi;
import android.app.Activity;
import android.os.Build;
import android.view.View;
import android.view.WindowManager;

/**
 * An API 11+ implementation of {@link SystemUiHider}. Uses APIs available in
 * Honeycomb and later (specifically {@link View#setSystemUiVisibility(int)}) to
 * show and hide the system UI.
 */
@TargetApi(Build.VERSION_CODES.HONEYCOMB)
public class SystemUiHiderHoneycomb extends SystemUiHiderBase {
    /**
     * Flags for {@link View#setSystemUiVisibility(int)} to use when showing the
     * system UI.
     */
    private int mShowFlags;

    /**
     * Flags for {@link View#setSystemUiVisibility(int)} to use when hiding the
     * system UI.
     */
    private int mHideFlags;

    /**
     * Flags to test against the first parameter in
     * {@link android.view.View.OnSystemUiVisibilityChangeListener#onSystemUiVisibilityChange(int)}
     * to determine the system UI visibility state.
     */
    private int mTestFlags;

    /**
     * Whether or not the system UI is currently visible. This is cached from
     * {@link android.view.View.OnSystemUiVisibilityChangeListener}.
     */
    private boolean mVisible = true;

    /**
     * Constructor not intended to be called by clients. Use
     * {@link SystemUiHider#getInstance} to obtain an instance.
     */
    protected SystemUiHiderHoneycomb(Activity activity, View anchorView, int flags) {
        super(activity, anchorView, flags);

        mShowFlags = View.SYSTEM_UI_FLAG_VISIBLE;
```

```java
            mHideFlags = View.SYSTEM_UI_FLAG_LOW_PROFILE;
            mTestFlags = View.SYSTEM_UI_FLAG_LOW_PROFILE;

            if ((mFlags & FLAG_FULLSCREEN) != 0) {
                // If the client requested fullscreen, add flags relevant to hiding
                // the status bar. Note that some of these constants are new as of
                // API 16 (Jelly Bean). It is safe to use them, as they are inlined
                // at compile-time and do nothing on pre-Jelly Bean devices.
                mShowFlags |= View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN;
                mHideFlags |= View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
                        | View.SYSTEM_UI_FLAG_FULLSCREEN;
            }

            if ((mFlags & FLAG_HIDE_NAVIGATION) != 0) {
                // If the client requested hiding navigation, add relevant flags.
                mShowFlags |= View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION;
                mHideFlags |= View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION
                        | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION;
                mTestFlags |= View.SYSTEM_UI_FLAG_HIDE_NAVIGATION;
            }
        }

        /** {@inheritDoc} */
        @Override
        public void setup() {
            mAnchorView.setOnSystemUiVisibilityChangeListener(mSystemUiVisibilityChangeListener);
        }

        /** {@inheritDoc} */
        @Override
        public void hide() {
            mAnchorView.setSystemUiVisibility(mHideFlags);
        }

        /** {@inheritDoc} */
        @Override
        public void show() {
            mAnchorView.setSystemUiVisibility(mShowFlags);
        }

        /** {@inheritDoc} */
        @Override
        public boolean isVisible() {
            return mVisible;
        }

        private View.OnSystemUiVisibilityChangeListener mSystemUiVisibilityChangeListener
                = new View.OnSystemUiVisibilityChangeListener() {
```

```
        @Override
        public void onSystemUiVisibilityChange(int vis) {
            // Test against mTestFlags to see if the system UI is visible.
            if ((vis & mTestFlags) != 0) {
                if (Build.VERSION.SDK_INT < Build.VERSION_CODES.JELLY_BEAN) {
                    // Pre-Jelly Bean, we must manually hide the action bar
                    // and use the old window flags API.
                    mActivity.getActionBar().hide();
                    mActivity.getWindow().setFlags(
                            WindowManager.LayoutParams.FLAG_FULLSCREEN,
                            WindowManager.LayoutParams.FLAG_FULLSCREEN);
                }

                // Trigger the registered listener and cache the visibility
                // state.
                mOnVisibilityChangeListener.onVisibilityChange(false);
                mVisible = false;

            } else {
                mAnchorView.setSystemUiVisibility(mShowFlags);
                if (Build.VERSION.SDK_INT < Build.VERSION_CODES.JELLY_BEAN) {
                    // Pre-Jelly Bean, we must manually show the action bar
                    // and use the old window flags API.
                    mActivity.getActionBar().show();
                    mActivity.getWindow().setFlags(
                            0,
                            WindowManager.LayoutParams.FLAG_FULLSCREEN);
                }

                // Trigger the registered listener and cache the visibility
                // state.
                mOnVisibilityChangeListener.onVisibilityChange(true);
                mVisible = true;
            }
        }
    };
}
```

```java
package com.delta.bartalk;

import android.annotation.TargetApi;
import android.content.Context;
import android.content.res.Resources;
import android.graphics.RectF;
import android.os.Build;
import android.text.Layout.Alignment;
import android.text.StaticLayout;
import android.text.TextPaint;
import android.util.AttributeSet;
import android.util.SparseIntArray;
import android.util.TypedValue;
import android.widget.TextView;

/*
  Taken from http://stackoverflow.com/users/1112882/m-wajeeh 's code on Stack Overflow

  here:

  http://stackoverflow.com/questions/5033012/auto-scale-textview-text-to-fit-within-bounds/17782522#17782522

  m-wajeeh provided a completely amazingly robust widget to handle auto-resizing!

 */

public class AutoResizeTextView extends TextView {
    private interface SizeTester {
        /**
         *
         * @param suggestedSize
         *            Size of text to be tested
         * @param availableSpace
         *            available space in which text must fit
         * @return an integer < 0 if after applying {@code suggestedSize} to
         *         text, it takes less space than {@code availableSpace}, > 0
         *         otherwise
         */
        public int onTestSize(int suggestedSize, RectF availableSpace);
    }

    private RectF mTextRect = new RectF();

    private RectF mAvailableSpaceRect;

    private SparseIntArray mTextCachedSizes;

    private TextPaint mPaint;

    private float mMaxTextSize;

    private float mSpacingMult = 1.0f;

    private float mSpacingAdd = 0.0f;

    private float mMinTextSize = 20;

    private int mWidthLimit;

    private static final int NO_LINE_LIMIT = -1;
    private int mMaxLines;

    private boolean mEnableSizeCache = true;
    private boolean mInitiallized;

    public AutoResizeTextView(Context context) {
        super(context);
        initialize();
    }

    public AutoResizeTextView(Context context, AttributeSet attrs) {
        super(context, attrs);
        initialize();
    }

    public AutoResizeTextView(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
        initialize();
    }

    private void initialize() {
        mPaint = new TextPaint(getPaint());
        mMaxTextSize = getTextSize();
        mAvailableSpaceRect = new RectF();
        mTextCachedSizes = new SparseIntArray();
        if (mMaxLines == 0) {
            // no value was assigned during construction
            mMaxLines = NO_LINE_LIMIT;
        }
        mInitiallized = true;
    }

    @Override
    public void setText(final CharSequence text, BufferType type) {
        super.setText(text, type);
        adjustTextSize(text.toString());
```

```java
    }

    @Override
    public void setTextSize(float size) {
        mMaxTextSize = size;
        mTextCachedSizes.clear();
        adjustTextSize(getText().toString());
    }

    @Override
    public void setMaxLines(int maxlines) {
        super.setMaxLines(maxlines);
        mMaxLines = maxlines;
        reAdjust();
    }

    public int getMaxLines() {
        return mMaxLines;
    }

    @Override
    public void setSingleLine() {
        super.setSingleLine();
        mMaxLines = 1;
        reAdjust();
    }

    @Override
    public void setSingleLine(boolean singleLine) {
        super.setSingleLine(singleLine);
        if (singleLine) {
            mMaxLines = 1;
        } else {
            mMaxLines = NO_LINE_LIMIT;
        }
        reAdjust();
    }

    @Override
    public void setLines(int lines) {
        super.setLines(lines);
        mMaxLines = lines;
        reAdjust();
    }

    @Override
    public void setTextSize(int unit, float size) {
        Context c = getContext();
        Resources r;

        if (c == null)
            r = Resources.getSystem();
        else
            r = c.getResources();
        mMaxTextSize = TypedValue.applyDimension(unit, size,
                r.getDisplayMetrics());
        mTextCachedSizes.clear();
        adjustTextSize(getText().toString());
    }

    @Override
    public void setLineSpacing(float add, float mult) {
        super.setLineSpacing(add, mult);
        mSpacingMult = mult;
        mSpacingAdd = add;
    }

    /**
     * Set the lower text size limit and invalidate the view
     *
     * @param minTextSize
     */
    public void setMinTextSize(float minTextSize) {
        mMinTextSize = minTextSize;
        reAdjust();
    }

    private void reAdjust() {
        adjustTextSize(getText().toString());
    }

    private void adjustTextSize(String string) {
        if (!mInitiallized) {
            return;
        }
        int startSize = (int) mMinTextSize;
        int heightLimit = getMeasuredHeight() - getCompoundPaddingBottom()
                - getCompoundPaddingTop();
        mWidthLimit = getMeasuredWidth() - getCompoundPaddingLeft()
                - getCompoundPaddingRight();
        mAvailableSpaceRect.right = mWidthLimit;
        mAvailableSpaceRect.bottom = heightLimit;
        super.setTextSize(
                TypedValue.COMPLEX_UNIT_PX,
                efficientTextSizeSearch(startSize, (int) mMaxTextSize,
                        mSizeTester, mAvailableSpaceRect));
    }

    private final SizeTester mSizeTester = new SizeTester() {
        @TargetApi(Build.VERSION_CODES.JELLY_BEAN)
        @Override
        public int onTestSize(int suggestedSize, RectF availableSPace) {
            mPaint.setTextSize(suggestedSize);
            String text = getText().toString();
            boolean singleline = getMaxLines() == 1;
            if (singleline) {
                mTextRect.bottom = mPaint.getFontSpacing();
                mTextRect.right = mPaint.measureText(text);
            } else {
                StaticLayout layout = new StaticLayout(text, mPaint,
                        mWidthLimit, Alignment.ALIGN_NORMAL, mSpacingMult,
                        mSpacingAdd, true);
                // return early if we have more lines
                if (getMaxLines() != NO_LINE_LIMIT
                        && layout.getLineCount() > getMaxLines()) {
                    return 1;
                }
                mTextRect.bottom = layout.getHeight();
                int maxWidth = -1;
                for (int i = 0; i < layout.getLineCount(); i++) {
                    if (maxWidth < layout.getLineWidth(i)) {
                        maxWidth = (int) layout.getLineWidth(i);
                    }
                }
                mTextRect.right = maxWidth;
            }

            mTextRect.offsetTo(0, 0);
            if (availableSPace.contains(mTextRect)) {
                // may be too small, don’t worry we will find the best match
                return -1;
            } else {
                // too big
                return 1;
            }
        }
    };

    /**
     * Enables or disables size caching, enabling it will improve performance
     * where you are animating a value inside TextView. This stores the font
     * size against getText().length() Be careful though while enabling it as 0
     * takes more space than 1 on some fonts and so on.
     *
```

```java
     * @param enable
     *            enable font size caching
     */
    public void enableSizeCache(boolean enable) {
        mEnableSizeCache = enable;
        mTextCachedSizes.clear();
        adjustTextSize(getText().toString());
    }

    private int efficientTextSizeSearch(int start, int end,
                                        SizeTester sizeTester, RectF availableSpace) {
        if (!mEnableSizeCache) {
            return binarySearch(start, end, sizeTester, availableSpace);
        }
        String text = getText().toString();
        int key = text == null ? 0 : text.length();
        int size = mTextCachedSizes.get(key);
        if (size != 0) {
            return size;
        }
        size = binarySearch(start, end, sizeTester, availableSpace);
        mTextCachedSizes.put(key, size);
        return size;
    }

    private static int binarySearch(int start, int end, SizeTester sizeTester,
                                    RectF availableSpace) {
        int lastBest = start;
        int lo = start;
        int hi = end - 1;
        int mid = 0;
        while (lo <= hi) {
            mid = (lo + hi) >>> 1;
            int midValCmp = sizeTester.onTestSize(mid, availableSpace);
            if (midValCmp < 0) {
                lastBest = lo;
                lo = mid + 1;
            } else if (midValCmp > 0) {
                hi = mid - 1;
                lastBest = hi;
            } else {
                return mid;
            }
        }
        // make sure to return last best
        // this is what should always be returned
        return lastBest;

    }

    @Override
    protected void onTextChanged(final CharSequence text, final int start,
                                 final int before, final int after) {
        super.onTextChanged(text, start, before, after);
        reAdjust();
    }

    @Override
    protected void onSizeChanged(int width, int height, int oldwidth,
                                 int oldheight) {
        mTextCachedSizes.clear();
        super.onSizeChanged(width, height, oldwidth, oldheight);
        if (width != oldwidth || height != oldheight) {
            reAdjust();
        }
    }
}
```

```java
package com.delta.bartalk;

import com.delta.bartalk.util.SystemUiHider;

import android.annotation.TargetApi;
import android.app.ActionBar;
import android.app.Activity;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.MotionEvent;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

/**
 * An example full-screen activity that shows and hides the system UI (i.e.
 * status bar and navigation/system bar) with user interaction.
 *
 * @see SystemUiHider
 */

public class BartalkActivity extends Activity {
    /**
     * Whether or not the system UI should be auto-hidden after
     * {@link #AUTO_HIDE_DELAY_MILLIS} milliseconds.
     */
    private static final boolean AUTO_HIDE = true;

    /**
     * If {@link #AUTO_HIDE} is set, the number of milliseconds to wait after
     * user interaction before hiding the system UI.
     */
    private static final int AUTO_HIDE_DELAY_MILLIS = 3000;

    /**
     * If set, will toggle the system UI visibility upon interaction. Otherwise,
     * will show the system UI visibility upon interaction.
     */
    private static final boolean TOGGLE_ON_CLICK = true;

    /**
     * The flags to pass to {@link SystemUiHider#getInstance}.
     */
    private static final int HIDER_FLAGS = SystemUiHider.FLAG_HIDE_NAVIGATION;

    /**
     * The instance of the {@link SystemUiHider} for this activity.
     */
    private SystemUiHider mSystemUiHider;

    //
    // Bartalk instance variables
    //
    TextView mOutputText;
    EditText mInputText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_bartalk);

        //
        //  Bartalk code
        //
        mOutputText = (TextView) findViewById(R.id.fullscreen_content);
        mInputText= (EditText) findViewById(R.id.input_text);

        mInputText.addTextChangedListener(new TextWatcher(){
            public void afterTextChanged(Editable s) {}
            public void beforeTextChanged(CharSequence s, int start, int count, int after){}
            public void onTextChanged(CharSequence s, int start, int before, int count){
                mOutputText.setText(mInputText.getText());
            }
        });

        ActionBar actionBar = getActionBar();
        actionBar.hide();

        final View controlsView = findViewById(R.id.fullscreen_content_controls);
        final View contentView = findViewById(R.id.fullscreen_content);

        // Set up an instance of SystemUiHider to control the system UI for
        // this activity.
        mSystemUiHider = SystemUiHider.getInstance(this, contentView, HIDER_FLAGS);
        mSystemUiHider.setup();
        mSystemUiHider
                .setOnVisibilityChangeListener(new SystemUiHider.OnVisibilityChangeListener() {
                    // Cached values.
                    int mControlsHeight;
                    int mShortAnimTime;

                    @Override
```

```java
                    @TargetApi(Build.VERSION_CODES.HONEYCOMB_MR2)
                    public void onVisibilityChange(boolean visible) {
                        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB_MR2) {
                            // If the ViewPropertyAnimator API is available
                            // (Honeycomb MR2 and later), use it to animate the
                            // in-layout UI controls at the bottom of the
                            // screen.
                            if (mControlsHeight == 0) {
                                mControlsHeight = controlsView.getHeight();
                            }
                            if (mShortAnimTime == 0) {
                                mShortAnimTime = getResources().getInteger(
                                        android.R.integer.config_shortAnimTime);
                            }
                            controlsView.animate()
                                    .translationY(visible ? 0 : mControlsHeight)
                                    .setDuration(mShortAnimTime);
                        } else {
                            // If the ViewPropertyAnimator APIs aren’t
                            // available, simply show or hide the in-layout UI
                            // controls.
                            controlsView.setVisibility(visible ? View.VISIBLE : View.GONE);
                        }

                        if (visible && AUTO_HIDE) {
                            // Schedule a hide().
                            delayedHide(AUTO_HIDE_DELAY_MILLIS);
                        }
                    }
                });

        // Set up the user interaction to manually show or hide the system UI.
        contentView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                if (TOGGLE_ON_CLICK) {
                    mSystemUiHider.toggle();
                } else {
                    mSystemUiHider.show();
                }
            }
        });

        // Upon interacting with UI controls, delay any scheduled hide()
        // operations to prevent the jarring behavior of controls going away
        // while interacting with the UI.
        findViewById(R.id.input_text).setOnTouchListener(mDelayHideTouchListener);
```

```java
    }

    @Override
    protected void onPostCreate(Bundle savedInstanceState) {
        super.onPostCreate(savedInstanceState);

        // Trigger the initial hide() shortly after the activity has been
        // created, to briefly hint to the user that UI controls
        // are available.
        delayedHide(100);
    }


    /**
     * Touch listener to use for in-layout UI controls to delay hiding the
     * system UI. This is to prevent the jarring behavior of controls going away
     * while interacting with activity UI.
     */
    View.OnTouchListener mDelayHideTouchListener = new View.OnTouchListener() {
        @Override
        public boolean onTouch(View view, MotionEvent motionEvent) {
            if (AUTO_HIDE) {
                delayedHide(AUTO_HIDE_DELAY_MILLIS);
            }

            return false;
        }
    };

    Handler mHideHandler = new Handler();
    Runnable mHideRunnable = new Runnable() {
        @Override
        public void run() {
            mSystemUiHider.hide();
        }
    };

    /**
     * Schedules a call to hide() in [delay] milliseconds, canceling any
     * previously scheduled calls.
     */
    private void delayedHide(int delayMillis) {
        mHideHandler.removeCallbacks(mHideRunnable);
        mHideHandler.postDelayed(mHideRunnable, delayMillis);
    }
}
```