Notification bubbles are 2D UI element that looks similar to word bubbles in cartoons. It contains a set of user defined 2D UI elements and has a pointer pointing to a specified scene whether in AR view or Map view. It is useful to show overlay of 2D information of a selected scene. This tutorial shows how to use the OANotificationBubbleView class in both the AR View and Map View.

1) Creating the Notification Bubble class

Create a class TutorialSceneNotificationBubble that extends OANotificationBubbleView. Define two final static int fields on the class, widthMargin and height, these fields are used to define the size of the notification bubble on the screen. The widthMargin field determines the total size of the padding area between the edges of the screen and the sides of the notification bubble, in pixels, while the height field sets the height of the notification bubble, also in pixels. Be mindful of the size of the screen that the application will be displayed on when setting these values.

Another field to declare is the protected variable currentScene of type OAScene, which will store the currently selected scene object. This field should be initialised to null.

We must also declare some private fields that will contain the views that comprise the content of the notification bubble. In this example we will use three fields, a TextView called popup_name, another TextView called popup_text, and an ImageView called popup_iconClose.

Next define a constructor for the class, it should look like this:

```java
public TutorialSceneNotificationBubble(Context context, int screenWidth, int screenHeight) {
    super(context, widthMargin / 2, screenHeight - height, screenWidth - widthMargin, height);
    int width = screenWidth - widthMargin;
    int left = widthMargin / 2;
    int top = screenHeight - height;
```
setAnchorPoints(left + width / 10, top, left + 2 * width / 10, top);

The superclass constructor requires the distance of the left side of the bubble from the edge of the screen (\(widthMargin / 2\)), the distance of the top of the bubble from the top of the screen (\(screenHeight - height\)), the total width of the bubble (\(screenWidth - widthMargin\)), and the total height of the bubble (\(height\)). These values are then used to set the anchor points of the callout on the bubble via a call to the superclass method setAnchorPoints().

Now we must override the abstract superclass method setupNotificationView() in order to define the views that will be shown inside the notification bubble. The method will look something like this:

```java
@Override
public View setupNotificationView() {
    View notificationView = View.inflate(getContext(), R.layout.notificationwindow, null);
    notificationView.setBackgroundColor(Color.BLACK); // Set default color
    // Text fields
    popup_name = (TextView) notificationView.findViewById(R.id.name);
    popup_text1 = (TextView) notificationView.findViewById(R.id.text1);
    popup_iconClose = (ImageView) notificationView.findViewById(R.id.closing);
    // Set the onClick listener for the Close icon to hide the notification // bubble instance.
    popup_iconClose.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            hide();
        }
    });
    return notificationView;
}
```

Since we have not yet created the layout that is referenced by this method it doesn’t make much sense right now. What it does is retrieve the views contained in the notification window layout which we will define soon, and stores them in the fields we declared earlier. It also defines an onClick method for the close icon that will hide the notification bubble if the icon is clicked.

Lastly we define a getter and setter for the currentScene field. The getter simply returns currentScene, while the setter takes a new OAScene instance and updates the text fields of the notification window layout with the relevant data from the new scene object, as
well as setting the value of `currentScene` to the new `OAScene`.

2) The Notification Window Layout

Create a new Android XML Layout document and call it `notificationwindow.xml`, the root element of this layout is a `RelativeLayout`. Inside the `RelativeLayout` put all of the elements that you want the notification window to display. In this case we will put the two `TextViews` we mentioned earlier, `popup_name` and `popup_text`, as well as an `ImageView`, `popup_iconClose`, with a reference to a close window icon drawable.

The layout should look like this:

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@android:color/black"
    android:padding="10px">
    <TextView
        android:id="@+id/name"
        android:text="Name"
        android:textColor="#00ffff"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:layout_width="wrap_content"
        android:layout_marginLeft="18dp"/>

    <TextView
        android:id="@+id/text1"
        android:text="Text1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#ffffff"
        android:textSize="15sp"
        android:layout_below="@id/name"
        android:layout_alignLeft="@id/name"/>

    <ImageView
        android:id="@+id/closing"
        android:layout_width="30sp"
        android:layout_height="30sp"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="3dp"
        android:src="@drawable/z_close"/>
</RelativeLayout>
```
Note that you must have an appropriate drawable to use as a close icon (in this case z_close.png), and it must be located in one of the res/drawable folders.

3) The AR View Activity

The AR View Activity is similar to the usual setup but with a few additions. Firstly a few fields must be declared on the class. Two private ints, `screenWidth` and `screenHeight`, will store the dimensions of the device screen in order to pass them to the notification bubble constructor. Another field we need is a private RelativeLayout field which we will call `uiLayout`. This field will store the root view of the AR View layout which we will need to add the notification bubble view to. Lastly we need a field that will contain the instance of TutorialSceneNotificationBubble, which we will call `notificationBubble`.

In order to add the notification bubble view to the AR View layout we need to add to the `setupUILayout()` method. Firstly we need to retrieve the screen dimensions of the current device, then we must retrieve the root view of the AR View layout, and finally we construct the notification bubble object and it to the layout. The method will end up looking like this:

```java
@Override
protected void setupUILayout(View arView) {
    super.setupUILayout(arView);
    DisplayMetrics displaymetrics = new DisplayMetrics();
    getWindowManager().getDefaultDisplay().getMetrics(displaymetrics);
    screenHeight = displaymetrics.heightPixels;
    screenWidth = displaymetrics.widthPixels;
    // add main ui layout
    LayoutInflater controlInflater = LayoutInflater.from(getBaseContext());
    uiLayout = (RelativeLayout)controlInflater.inflate(R.layout.arview, null);
    setContentView(uiLayout, new LayoutParams(LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT));
    // setup notification bubble
    notificationBubble = new TutorialSceneNotificationBubble(this, screenWidth, screenHeight) {
        @Override
        public void updatePoint(Point point) {
            getScreenPoint(getScene(), point);
        }
    };
    uiLayout.addView(notificationBubble);
}
```
The `updatePoint()` method defined for the notification bubble is used to keep the notification bubble callout pointing at the location on the screen of the currently selected scene.

The last method we must alter is the `onSceneSelected()` method, which will simply set the `currentScene` field of `notificationBubble` to the OAScene passed as a parameter (using the setter we defined earlier), then it will call the `show()` method of `notificationBubble` (defined on the superclass `OANotificationBubbleView`).

We now have a working notification bubble on our AR View Activity.

4) The Map View Activity

We can add the notification bubble to the Map View Activity in just about the same way as we did for the AR View Activity. We define a field to contain the root view of the layout. We alter the `setupUILayout()` method to get the screen dimensions, construct the notification bubble instance, and add it to the layout. Lastly we override the `onSceneSelected()` method to set the scene of the notification bubble instance and make it visible.

Now we have a notification bubble that will work in both the AR View and the Map View.

If you have further problems or questions, visit our website ([http://www.hitlabnz.org/mobileAR](http://www.hitlabnz.org/mobileAR)) and post your problem on the Mobile AR Framework Support Forum.