How to add animation to Maps API for Java ME

This article explains how to add a a `MapComponent` which adds animation to your map.

Introduction

The HERE Maps API for Java ME does not offer a standard mechanism for animating `MapObjects`, or indeed any mechanism to show an animated wait signal (such as a spinner) whilst the device is working in the background. This article defines an animation framework for doing so and explains how to create a `MapComponent` which will bounce a series of `MapMarkers` on command. The concept can be easily extended to cover other animations such as flashing `MapMarker` or drawing a series of `Images` over the map.

Definition of the Issue

Map applications are rarely animated and therefore the standard `MapCanvas` of the HERE Maps API for Java ME is an extension of the standard `Canvas`. Basing an application on `Canvas` is a good choice for rendering images on screen, but unlike the associated `GameCanvas` extension, `Canvas` is not optimized for animation. To add animation to a map, A certain amount of custom code is also required. Specifically the development involves the following steps:

1. Creation and attachment of a `MapComponent` which is able to a paint a series of `Images`) over the map.
2. Creation and initialization of a `Timer` to poll and make regular `repaints()` requests back to the `MapCanvas`
3. Addition of a "well-known" interface to start and stop the animation of the `MapComponent`

Creating an Animated Map Component

Attaching the map

Because the animation of the `MarkerBouncer` involves the appearance of movement of `MapObjects` - i.e. objects that are anchored to a specific `GeoCoordinate`. It is necessary to maintain a reference to the underlying `MapDisplay`. This is because it will be necessary add any animated images at a pixel `Point` location based upon the aforementioned `GeoCoordinate` . It is possible that the map may have been panned or zoomed whilst the animation takes place, and therefore the pixel `Point` location of the animated object will also need to change.

Maintaining a reference to the currently attached map is such a common feature of a `MapComponent`, that the code is usually placed in an abstract base class (in this case called `AbstractMapComponent`). The protected variable `map` can then be used throughout the subclass `MapComponent`. The `AbstractMapComponent` also handles the `getVersion()` and `getId()` methods.

```
 protected MapDisplay map;

...
    public void attach(MapDisplay map) {
        this.map = map;
    }

...
    public void detach(MapDisplay map) {
        this.map = null;
    }
...
    public EventListener getEventListener() {
        return null;
    }
```


(C) Copyright Nokia 2013. All rights reserved.

Note that because the `MarkerBouncer` itself does not rely upon external user interaction it does not need to offer an `EventListener`

Initialization of the `MarkerBouncer`

The initialization of the `MarkerBouncer` calls the `AbstractMapComponent` super-class and sets up a series of member variables

- The `Timer` which will be used to poll the `MapCanvas` to repaint.
- An `Image` which will be drawn to create the "bouncing" marker on screen
- The callback `MapListener` interface which is used to make the repaint requests - specifically `onMapContentUpdated()`

```
public class MarkerBouncer extends AbstractMapComponent {

...
    private TimerTask task;
 private final Timer timer;

...
    private final MapListener listener;

 private Image markerIcon;

 public MarkerBouncer(MapListener listener) {
  super(ID, VERSION);

  timer = new Timer();
  currentMarkerPositions = new Hashtable();
  this.listener = listener;

  // load image resource from MIDlet's jar file
  try {
   markerIcon = Image.createImage("/nma_res/mask.png");
  } catch (IOException e) {
   e.printStackTrace();
  }
 }
```

Rendering a Bouncing Marker

The `paint()` method merely displays an `Image` at the pixel `Point` specified on screen. Details of the underlying `Goecoordinate` of the `MapStandardMarker` are held in a `Hashtable` and the `Point` of the anchor of the `Image` recalculated on each repaint. As mentioned in the previous section, this re-calculation is necessary since the map may have been panned between `paint()` requests.

```
public class MarkerBouncer extends AbstractMapComponent  {

...
    public void paint(Graphics g) {
  // //
  Enumeration keys = currentMarkerPositions.keys();
  while (keys.hasMoreElements()) {
   MapStandardMarker key = ((MapStandardMarker) keys.nextElement());
   Point point = map.geoToPixel((GeoCoordinate) currentMarkerPositions
     .get(key));
   g.drawImage(markerIcon, point.getX() - markerIcon.getWidth() / 2,
     point.getY() - markerIcon.getHeight(), Graphics.TOP
       | Graphics.LEFT);

  }
 }
```

Note: For an animation based on non-`MapObjects`, the calculation above may not be necessary. However careful consideration should be given to maintaining the correct anchor point of the `Image`. This is especially the case if both landscape and portrait orientations are supported, where the anchor point will need to be based on `map.getWidth()` and `map.getHeight()`

Starting and Stopping the Animation

Starting and Stopping the animation takes place through a"well-known" interface - in this case two methods called `bounce()` and `stop()`. The `Geocoordinate` anchor points of the bouncing markers are cached and a `TimerTask` initiates the animation loop (`MapCanvasRefresher`) at a given rate.

```
public class MarkerBouncer extends AbstractMapComponent  {

...

 public void bounce(MapObject[] mapObjects) {

  currentMarkerPositions.clear();
  height = getDropHeight();

  for (int i = 0; i < mapObjects.length; i++) {

   if (mapObjects[i] instanceof MapStandardMarker) {
    GeoCoordinate coord = new GeoCoordinate(
      ((MapStandardMarker) mapObjects[i]).getCoordinate());
    currentMarkerPositions.put(mapObjects[i], coord);
    mapObjects[i].setVisible(false);
   }
  }

  task = new MapCanvasRefresher();
  timer.scheduleAtFixedRate(task, 0, 70);

 }

 public void stop() {

  if (task != null) {
```


(C) Copyright Nokia 2013. All rights reserved.

```
    task.cancel();
    task = null;
   }

   Enumeration keys = currentMarkerPositions.keys();
   while (keys.hasMoreElements()) {
    MapStandardMarker key = ((MapStandardMarker) keys.nextElement());
    key.setVisible(true);

   }
   currentMarkerPositions.clear();
  }
 }
```

The Animation loop

The animation loop is defined in a private inner class called (`MapCanvasRefresher`). This updates any animation variables and finally calls `listener.onMapContentUpdated()` which request the `MapCanvas` refresh.

```
public class MarkerBouncer extends AbstractMapComponent  {

...

      private class MapCanvasRefresher extends TimerTask {

  public void run() {
   Enumeration keys = currentMarkerPositions.keys();

   // Calculation in Pixel coordinates
   velocity = velocity - getAcceleration();
   height = height + velocity;

   if (height < 0) {
    height = 0;
    velocity = -velocity * getDecay();
    if (velocity < 1) {
     stop();
    }
   }

   while (keys.hasMoreElements()) {
    MapStandardMarker key = ((MapStandardMarker) keys.nextElement());
    GeoCoordinate currentPos = (GeoCoordinate) currentMarkerPositions
      .get(key);

    // Place the Marker in Geographic coordinates
    double latitudeOffset = map.pixelToGeo(new Point(0, 0)).getLatitude() -
      map.pixelToGeo(new Point(0, (int) height)).getLatitude();
    currentPos.setLatitude(key.getCoordinate().getLatitude() +
       latitudeOffset);


   }
   listener.onMapContentUpdated();

  }
 }

}
```

Attaching and Using an Animated Map Component

Adding the `MarkerBouncer` component can be done in the usual manner:

```
    public BouncingMarkerExample(Display display, MIDlet midlet) {
       ...
            bouncer = new MarkerBouncer(this);
   map.addMapComponent(bouncer);
        }
```

Because the interface of the `MarkerBouncer` is known it can be stopped and started as shown:

```
         public void commandRun(Command c) {
   if (c == BOUNCE_ALL) {
    bouncer.stop();
    bouncer.bounce(map.getAllMapObjects());
   }
   if (c == BOUNCE_ONE) {
    MapObject [] all = map.getAllMapObjects();

    bouncer.stop();
    bouncer.bounce(new MapObject[] {all[0]});
   }

  }
```

Summary

As can be seen in the Example MIDlet, simple animation may be added to any `MapCanvas`-based application, and can be used to improve the UI of an app. However, it should be noted that animation is both processor and resource intensive, and should be limited in nature given the limitations of the Java ME environment.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.