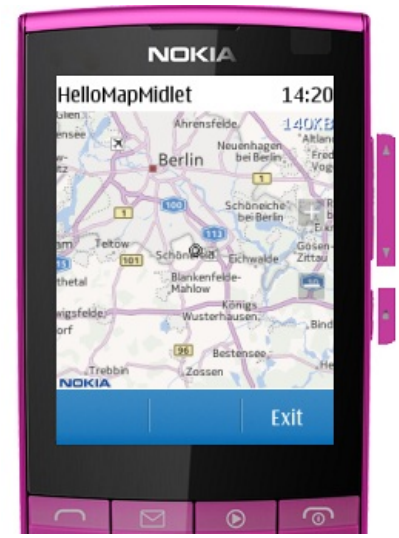
Implementing Kinetic panning with Maps API for Java ME

This article explains how you could add **kinetic panning** feature to the map element implemented with HERE Maps API for Java ME.

Introduction



The HERE Maps API for Java ME allows to easily embed HERE Maps into Java ME applications, however the API is missing implementation for **kinetic panning**. And in this article one possible way on implementing the **kinetic panning** with this Maps API is introduced. This method is designed to work only with touch screen, however the same idea could be modified to work with non-touch devices as well.

General touch event handling

Generally there is two ways on handling touch events, one being on handling them with custom controls as shown in Handling map events with Maps API for Java ME article. Another possibility would be to overwrite the touch event handlers implemented for the Map canvas and handling. This second option is used with this example, thus the stripped down implementation of the `MapCanvas` class would look like the following:

```
public class MyMapCanvas extends MapCanvas implements ...{
 ...
protected void pointerPressed(int x, int y) {
        ...}
protected void pointerDragged(int x, int y) {
 ...}
protected void pointerReleased(int x, int y) {
 }
}
```

The method of calculating the needed kinetic panning depends on the preferences of the designers, with this one, calculations are simplified and the kinetic panning values are calculated on the average of the movements between press and release events. Thus everything starts with press event, in which we need to reset calculations and start new. Also if the event is also needed by the original maps implementation, you would need to use super and call the base implementation:

```
protected void pointerPressed(int x, int y) {
 history_Speed.x = 0;
 history_Speed.y = 0;
 history_Speed.time = 0;
```


(C) Copyright Nokia 2014. All rights reserved.

```
 HistoryCount = 0;
 lastPoint.x = x;
 lastPoint.y = y;
 lastPoint.time = System.currentTimeMillis();
 isPanning = true;

 if(task != null){
  task.SetActive(false);
 }
        super.pointerPressed(x,y);
 }
```

Then each drag event would call its handler. In this if we are using our own panning we would not call the base implementation with super Instead we would use our own calculations for panning the map and would use `panFloatWrapper()` method for the actual panning. Each drag event is also saved for future processing purposes by calling `AddHistoryData()`, which is actually simply adding up the movement as well as counting the moving times, so we can later on calculate the average moving distance.

```
protected void pointerDragged(int x, int y) {
 if(isPanning){
  FPoint deltaXY = new FPoint();
  deltaXY.x = (lastPoint.x - x);
  deltaXY.y = (lastPoint.y - y);
  deltaXY.time = (System.currentTimeMillis() - lastPoint.time);
             if((deltaXY.x != 0) && (deltaXY.y != 0)){
   if(deltaXY.time > 0){
    AddHistoryData(deltaXY);
   }
   lastPoint.x = x;
   lastPoint.y = y;
   lastPoint.time = System.currentTimeMillis();
   panFloatWrapper(deltaXY);
         }
 }else{
         super.pointerDragged(x,y);
 }
}
```

And to finalize the touch event handling, the `pointerReleased()` method needs to be implemented. With this example implementation, the history data of the movements gathered in drag event handler is now used to calculate the movements used with kinetic panning. This after the kinetic panning timer is started to continue the movement with map.

```
protected void pointerReleased(int x, int y) {
  if(isPanning){
  panSpeed.x = panSpeed.y = 0.0;
  if(HistoryCount > 0){
   history_Speed.x = (history_Speed.x / HistoryCount);
   history_Speed.y = (history_Speed.y / HistoryCount);
   history_Speed.time = (history_Speed.time / HistoryCount);
  }
  panSpeed.x = history_Speed.x;
  panSpeed.y = history_Speed.y;
  panSpeed.time = history_Speed.time;

  lastPoint.x = 0.0;
  lastPoint.y = 0.0;
```


(C) Copyright Nokia 2014. All rights reserved.

```
  lastPoint.time = System.currentTimeMillis();
  isPanning = false;

            if(task != null){
   task.SetActive(true);
  }
 }
 super.pointerReleased(x,y);
}
```

Timer implementation

With kinetic panning the movement of the map should continue after user releases the pointing device, with this example this is handled by calculating the movements to be used with additional moving and then using timer to do the actual movements, and each movement the amount is decreased until the movement comes to stop, and then the timer is simply stopped.

For this example the calculations for decreasing the movement is also really simplified, and if better look & feel is needed, you could change the data handling and calculation types to get different behaviors:

The `m_kineticTimer` implementation is really simple, once it is active it will call the `timerCallBack()` method periodically until it has been stopped. And as shown with code snipped with this article, the `Timer` is started in the `pointerReleased()` method, and stopped in teh `timerCallBack()` method as well as in the `pointerPressed()` method.

```
public class MyMapCanvas ...{
 private TestTimerTask task = null;
 private Timer m_kineticTimer= null;

 public MyMapCanvas(Display display, MIDlet midlet) {
  ...

        if(task == null){
             task = new TestTimerTask(this);
             m_kineticTimer = new Timer();
             m_kineticTimer.scheduleAtFixedRate(task,kineticPanningResolution,
kineticPanningResolution);
        }
 }
 ...

 public void timerCallBack() {
        panSpeed.x = (panSpeed.x * 0.5);
        panSpeed.y = (panSpeed.y * 0.5);

        if ((0 == (int)panSpeed.x) && (0 == (int)panSpeed.y)) {
         // Kinetic panning is almost halted -> stop it.
             if(task != null){
                 task.SetActive(false);
             }
             return;
        }

        panFloatWrapper(new Point((int)(panSpeed.x *  5),(int)(panSpeed.y *  5)));
     }
 ...
}
```


(C) Copyright Nokia 2014. All rights reserved.

Resources

Full source codes for the example can be found from File:WikiKineticPan.zip

Summary

The HERE Maps API for Java ME offers rich functionality that allows to integrate all the main HERE Maps features in a Java ME application, with just a few lines of code.

(C) Copyright Nokia 2014. All rights reserved.