Module-6 assignment

|  |
| --- |
| AirportMap.java |
|  |  |
|  | package module6; |
|  |  |
|  | import java.util.ArrayList; |
|  | import java.util.HashMap; |
|  | import java.util.List; |
|  |  |
|  | import de.fhpotsdam.unfolding.UnfoldingMap; |
|  | import de.fhpotsdam.unfolding.data.PointFeature; |
|  | import de.fhpotsdam.unfolding.data.ShapeFeature; |
|  | import de.fhpotsdam.unfolding.marker.Marker; |
|  | import de.fhpotsdam.unfolding.marker.SimpleLinesMarker; |
|  | import de.fhpotsdam.unfolding.marker.SimplePointMarker; |
|  | import de.fhpotsdam.unfolding.utils.MapUtils; |
|  | import de.fhpotsdam.unfolding.geo.Location; |
|  | import parsing.ParseFeed; |
|  | import processing.core.PApplet; |
|  |  |
|  | /\*\* An applet that shows airports (and routes) |
|  | \* on a world map. |
|  | \* @author Adam Setters and the UC San Diego Intermediate Software Development |
|  | \* MOOC team |
|  | \* |
|  | \*/ |
|  | public class AirportMap extends PApplet { |
|  |  |
|  | UnfoldingMap map; |
|  | private List<Marker> airportList; |
|  | List<Marker> routeList; |
|  |  |
|  | public void setup() { |
|  | // setting up PAppler |
|  | size(800,600, OPENGL); |
|  |  |
|  | // setting up map and default events |
|  | map = new UnfoldingMap(this, 50, 50, 750, 550); |
|  | MapUtils.createDefaultEventDispatcher(this, map); |
|  |  |
|  | // get features from airport data |
|  | List<PointFeature> features = ParseFeed.parseAirports(this, "airports.dat"); |
|  |  |
|  | // list for markers, hashmap for quicker access when matching with routes |
|  | airportList = new ArrayList<Marker>(); |
|  | HashMap<Integer, Location> airports = new HashMap<Integer, Location>(); |
|  |  |
|  | // create markers from features |
|  | for(PointFeature feature : features) { |
|  | AirportMarker m = new AirportMarker(feature); |
|  |  |
|  | m.setRadius(5); |
|  | airportList.add(m); |
|  |  |
|  | // put airport in hashmap with OpenFlights unique id for key |
|  | airports.put(Integer.parseInt(feature.getId()), feature.getLocation()); |
|  |  |
|  | } |
|  |  |
|  |  |
|  | // parse route data |
|  | List<ShapeFeature> routes = ParseFeed.parseRoutes(this, "routes.dat"); |
|  | routeList = new ArrayList<Marker>(); |
|  | for(ShapeFeature route : routes) { |
|  |  |
|  | // get source and destination airportIds |
|  | int source = Integer.parseInt((String)route.getProperty("source")); |
|  | int dest = Integer.parseInt((String)route.getProperty("destination")); |
|  |  |
|  | // get locations for airports on route |
|  | if(airports.containsKey(source) && airports.containsKey(dest)) { |
|  | route.addLocation(airports.get(source)); |
|  | route.addLocation(airports.get(dest)); |
|  | } |
|  |  |
|  | SimpleLinesMarker sl = new SimpleLinesMarker(route.getLocations(), route.getProperties()); |
|  |  |
|  | System.out.println(sl.getProperties()); |
|  |  |
|  | //UNCOMMENT IF YOU WANT TO SEE ALL ROUTES |
|  | //routeList.add(sl); |
|  | } |
|  |  |
|  |  |
|  |  |
|  | //UNCOMMENT IF YOU WANT TO SEE ALL ROUTES |
|  | //map.addMarkers(routeList); |
|  |  |
|  | map.addMarkers(airportList); |
|  |  |
|  | } |
|  |  |
|  | public void draw() { |
|  | background(0); |
|  | map.draw(); |
|  |  |
|  | } |
|  |  |
|  |  |
|  | } |
|  |  |
|  |  |
|  | AirportMarker.java |
|  |  |
|  |  |
|  | package module6; |
|  |  |
|  | import java.util.List; |
|  |  |
|  | import de.fhpotsdam.unfolding.data.Feature; |
|  | import de.fhpotsdam.unfolding.data.PointFeature; |
|  | import de.fhpotsdam.unfolding.marker.SimpleLinesMarker; |
|  | import processing.core.PGraphics; |
|  |  |
|  | /\*\* |
|  | \* A class to represent AirportMarkers on a world map. |
|  | \* |
|  | \* @author Adam Setters and the UC San Diego Intermediate Software Development |
|  | \* MOOC team |
|  | \* |
|  | \*/ |
|  | public class AirportMarker extends CommonMarker { |
|  | public static List<SimpleLinesMarker> routes; |
|  |  |
|  | public AirportMarker(Feature city) { |
|  | super(((PointFeature)city).getLocation(), city.getProperties()); |
|  |  |
|  | } |
|  |  |
|  | @Override |
|  | public void drawMarker(PGraphics pg, float x, float y) { |
|  | pg.fill(11); |
|  | pg.ellipse(x, y, 5, 5); |
|  |  |
|  |  |
|  | } |
|  |  |
|  | @Override |
|  | public void showTitle(PGraphics pg, float x, float y) { |
|  | // show rectangle with title |
|  |  |
|  | // show routes |
|  |  |
|  |  |
|  | } |
|  |  |
|  | } |
|  |  |
|  |  |
|  | CityMarker.java |
|  |  |
|  |  |
|  |  |
|  | package module6; |
|  |  |
|  | import de.fhpotsdam.unfolding.data.Feature; |
|  | import de.fhpotsdam.unfolding.data.PointFeature; |
|  | import de.fhpotsdam.unfolding.geo.Location; |
|  | import processing.core.PConstants; |
|  | import processing.core.PGraphics; |
|  |  |
|  | /\*\* Implements a visual marker for cities on an earthquake map |
|  | \* |
|  | \* @author UC San Diego Intermediate Software Development MOOC team |
|  | \* |
|  | \*/ |
|  | public class CityMarker extends CommonMarker { |
|  |  |
|  | public static int TRI\_SIZE = 5; // The size of the triangle marker |
|  |  |
|  | public CityMarker(Location location) { |
|  | super(location); |
|  | } |
|  |  |
|  |  |
|  | public CityMarker(Feature city) { |
|  | super(((PointFeature)city).getLocation(), city.getProperties()); |
|  | // Cities have properties: "name" (city name), "country" (country name) |
|  | // and "population" (population, in millions) |
|  | } |
|  |  |
|  |  |
|  | // pg is the graphics object on which you call the graphics |
|  | // methods. e.g. pg.fill(255, 0, 0) will set the color to red |
|  | // x and y are the center of the object to draw. |
|  | // They will be used to calculate the coordinates to pass |
|  | // into any shape drawing methods. |
|  | // e.g. pg.rect(x, y, 10, 10) will draw a 10x10 square |
|  | // whose upper left corner is at position x, y |
|  | /\*\* |
|  | \* Implementation of method to draw marker on the map. |
|  | \*/ |
|  | public void drawMarker(PGraphics pg, float x, float y) { |
|  | //System.out.println("Drawing a city"); |
|  | // Save previous drawing style |
|  | pg.pushStyle(); |
|  |  |
|  | // IMPLEMENT: drawing triangle for each city |
|  | pg.fill(150, 30, 30); |
|  | pg.triangle(x, y-TRI\_SIZE, x-TRI\_SIZE, y+TRI\_SIZE, x+TRI\_SIZE, y+TRI\_SIZE); |
|  |  |
|  | // Restore previous drawing style |
|  | pg.popStyle(); |
|  | } |
|  |  |
|  | /\*\* Show the title of the city if this marker is selected \*/ |
|  | public void showTitle(PGraphics pg, float x, float y) |
|  | { |
|  | String name = getCity() + " " + getCountry() + " "; |
|  | String pop = "Pop: " + getPopulation() + " Million"; |
|  |  |
|  | pg.pushStyle(); |
|  |  |
|  | pg.fill(255, 255, 255); |
|  | pg.textSize(12); |
|  | pg.rectMode(PConstants.CORNER); |
|  | pg.rect(x, y-TRI\_SIZE-39, Math.max(pg.textWidth(name), pg.textWidth(pop)) + 6, 39); |
|  | pg.fill(0, 0, 0); |
|  | pg.textAlign(PConstants.LEFT, PConstants.TOP); |
|  | pg.text(name, x+3, y-TRI\_SIZE-33); |
|  | pg.text(pop, x+3, y - TRI\_SIZE -18); |
|  |  |
|  | pg.popStyle(); |
|  | } |
|  |  |
|  | private String getCity() |
|  | { |
|  | return getStringProperty("name"); |
|  | } |
|  |  |
|  | private String getCountry() |
|  | { |
|  | return getStringProperty("country"); |
|  | } |
|  |  |
|  | private float getPopulation() |
|  | { |
|  | return Float.parseFloat(getStringProperty("population")); |
|  | } |
|  | } |
|  |  |
|  |  |
|  | CommonMarker.java |
|  |  |
|  | package module6; |
|  |  |
|  |  |
|  | import de.fhpotsdam.unfolding.geo.Location; |
|  | import de.fhpotsdam.unfolding.marker.SimplePointMarker; |
|  | import processing.core.PGraphics; |
|  |  |
|  | /\*\* Implements a common marker for cities and earthquakes on an earthquake map |
|  | \* |
|  | \* @author UC San Diego Intermediate Software Development MOOC team |
|  | \* |
|  | \*/ |
|  | public abstract class CommonMarker extends SimplePointMarker { |
|  |  |
|  | // Records whether this marker has been clicked (most recently) |
|  | protected boolean clicked = false; |
|  |  |
|  | public CommonMarker(Location location) { |
|  | super(location); |
|  | } |
|  |  |
|  | public CommonMarker(Location location, java.util.HashMap<java.lang.String,java.lang.Object> properties) { |
|  | super(location, properties); |
|  | } |
|  |  |
|  | // Getter method for clicked field |
|  | public boolean getClicked() { |
|  | return clicked; |
|  | } |
|  |  |
|  | // Setter method for clicked field |
|  | public void setClicked(boolean state) { |
|  | clicked = state; |
|  | } |
|  |  |
|  | // Common piece of drawing method for markers; |
|  | // YOU WILL IMPLEMENT. |
|  | // Note that you should implement this by making calls |
|  | // drawMarker and showTitle, which are abstract methods |
|  | // implemented in subclasses |
|  | public void draw(PGraphics pg, float x, float y) { |
|  | // For starter code just drawMaker(...) |
|  | if (!hidden) { |
|  | drawMarker(pg, x, y); |
|  | if (selected) { |
|  | showTitle(pg, x, y); |
|  | } |
|  | } |
|  | } |
|  | public abstract void drawMarker(PGraphics pg, float x, float y); |
|  | public abstract void showTitle(PGraphics pg, float x, float y); |
|  | } |
|  |  |
|  |  |
|  | EarthquakeCityMap.java |
|  |  |
|  |  |
|  | package module6; |
|  |  |
|  | import java.util.ArrayList; |
|  | import java.util.Arrays; |
|  | import java.util.List; |
|  |  |
|  | import de.fhpotsdam.unfolding.UnfoldingMap; |
|  | import de.fhpotsdam.unfolding.data.Feature; |
|  | import de.fhpotsdam.unfolding.data.GeoJSONReader; |
|  | import de.fhpotsdam.unfolding.data.PointFeature; |
|  | import de.fhpotsdam.unfolding.geo.Location; |
|  | import de.fhpotsdam.unfolding.marker.AbstractShapeMarker; |
|  | import de.fhpotsdam.unfolding.marker.Marker; |
|  | import de.fhpotsdam.unfolding.marker.MultiMarker; |
|  | import de.fhpotsdam.unfolding.providers.Google; |
|  | import de.fhpotsdam.unfolding.providers.MBTilesMapProvider; |
|  | import de.fhpotsdam.unfolding.utils.MapUtils; |
|  | import parsing.ParseFeed; |
|  | import processing.core.PApplet; |
|  |  |
|  | /\*\* EarthquakeCityMap |
|  | \* An application with an interactive map displaying earthquake data. |
|  | \* Author: UC San Diego Intermediate Software Development MOOC team |
|  | \* @author Your name here |
|  | \* Date: July 17, 2015 |
|  | \* \*/ |
|  | public class EarthquakeCityMap extends PApplet { |
|  |  |
|  | // We will use member variables, instead of local variables, to store the data |
|  | // that the setUp and draw methods will need to access (as well as other methods) |
|  | // You will use many of these variables, but the only one you should need to add |
|  | // code to modify is countryQuakes, where you will store the number of earthquakes |
|  | // per country. |
|  |  |
|  | // You can ignore this. It's to get rid of eclipse warnings |
|  | private static final long serialVersionUID = 1L; |
|  |  |
|  | // IF YOU ARE WORKING OFFILINE, change the value of this variable to true |
|  | private static final boolean offline = false; |
|  |  |
|  | /\*\* This is where to find the local tiles, for working without an Internet connection \*/ |
|  | public static String mbTilesString = "blankLight-1-3.mbtiles"; |
|  |  |
|  |  |
|  |  |
|  | //feed with magnitude 2.5+ Earthquakes |
|  | private String earthquakesURL = "http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/2.5\_week.atom"; |
|  |  |
|  | // The files containing city names and info and country names and info |
|  | private String cityFile = "city-data.json"; |
|  | private String countryFile = "countries.geo.json"; |
|  |  |
|  | // The map |
|  | private UnfoldingMap map; |
|  |  |
|  | // Markers for each city |
|  | private List<Marker> cityMarkers; |
|  | // Markers for each earthquake |
|  | private List<Marker> quakeMarkers; |
|  |  |
|  | // A List of country markers |
|  | private List<Marker> countryMarkers; |
|  |  |
|  | // NEW IN MODULE 5 |
|  | private CommonMarker lastSelected; |
|  | private CommonMarker lastClicked; |
|  |  |
|  | public void setup() { |
|  | // (1) Initializing canvas and map tiles |
|  | size(900, 700, OPENGL); |
|  | if (offline) { |
|  | map = new UnfoldingMap(this, 200, 50, 650, 600, new MBTilesMapProvider(mbTilesString)); |
|  | earthquakesURL = "2.5\_week.atom"; // The same feed, but saved August 7, 2015 |
|  | } |
|  | else { |
|  | map = new UnfoldingMap(this, 200, 50, 650, 600, new Google.GoogleMapProvider()); |
|  | // IF YOU WANT TO TEST WITH A LOCAL FILE, uncomment the next line |
|  | //earthquakesURL = "2.5\_week.atom"; |
|  | } |
|  | MapUtils.createDefaultEventDispatcher(this, map); |
|  |  |
|  | // FOR TESTING: Set earthquakesURL to be one of the testing files by uncommenting |
|  | // one of the lines below. This will work whether you are online or offline |
|  | //earthquakesURL = "test1.atom"; |
|  | //earthquakesURL = "test2.atom"; |
|  |  |
|  | // Uncomment this line to take the quiz |
|  | //earthquakesURL = "quiz2.atom"; |
|  |  |
|  |  |
|  | // (2) Reading in earthquake data and geometric properties |
|  | // STEP 1: load country features and markers |
|  | List<Feature> countries = GeoJSONReader.loadData(this, countryFile); |
|  | countryMarkers = MapUtils.createSimpleMarkers(countries); |
|  |  |
|  | // STEP 2: read in city data |
|  | List<Feature> cities = GeoJSONReader.loadData(this, cityFile); |
|  | cityMarkers = new ArrayList<Marker>(); |
|  | for(Feature city : cities) { |
|  | cityMarkers.add(new CityMarker(city)); |
|  | } |
|  |  |
|  | // STEP 3: read in earthquake RSS feed |
|  | List<PointFeature> earthquakes = ParseFeed.parseEarthquake(this, earthquakesURL); |
|  | quakeMarkers = new ArrayList<Marker>(); |
|  |  |
|  | for(PointFeature feature : earthquakes) { |
|  | //check if LandQuake |
|  | if(isLand(feature)) { |
|  | quakeMarkers.add(new LandQuakeMarker(feature)); |
|  | } |
|  | // OceanQuakes |
|  | else { |
|  | quakeMarkers.add(new OceanQuakeMarker(feature)); |
|  | } |
|  | } |
|  |  |
|  | // could be used for debugging |
|  | printQuakes(); |
|  |  |
|  | // (3) Add markers to map |
|  | // NOTE: Country markers are not added to the map. They are used |
|  | // for their geometric properties |
|  | map.addMarkers(quakeMarkers); |
|  | map.addMarkers(cityMarkers); |
|  |  |
|  |  |
|  | } // End setup |
|  |  |
|  |  |
|  | public void draw() { |
|  | background(0); |
|  | map.draw(); |
|  | addKey(); |
|  |  |
|  | } |
|  |  |
|  |  |
|  | // TODO: Add the method: |
|  | // private void sortAndPrint(int numToPrint) |
|  | // and then call that method from setUp |
|  |  |
|  | /\*\* Event handler that gets called automatically when the |
|  | \* mouse moves. |
|  | \*/ |
|  | @Override |
|  | public void mouseMoved() |
|  | { |
|  | // clear the last selection |
|  | if (lastSelected != null) { |
|  | lastSelected.setSelected(false); |
|  | lastSelected = null; |
|  |  |
|  | } |
|  | selectMarkerIfHover(quakeMarkers); |
|  | selectMarkerIfHover(cityMarkers); |
|  | //loop(); |
|  | } |
|  |  |
|  | // If there is a marker selected |
|  | private void selectMarkerIfHover(List<Marker> markers) |
|  | { |
|  | // Abort if there's already a marker selected |
|  | if (lastSelected != null) { |
|  | return; |
|  | } |
|  |  |
|  | for (Marker m : markers) |
|  | { |
|  | CommonMarker marker = (CommonMarker)m; |
|  | if (marker.isInside(map, mouseX, mouseY)) { |
|  | lastSelected = marker; |
|  | marker.setSelected(true); |
|  | return; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | /\*\* The event handler for mouse clicks |
|  | \* It will display an earthquake and its threat circle of cities |
|  | \* Or if a city is clicked, it will display all the earthquakes |
|  | \* where the city is in the threat circle |
|  | \*/ |
|  | @Override |
|  | public void mouseClicked() |
|  | { |
|  | if (lastClicked != null) { |
|  | unhideMarkers(); |
|  | lastClicked = null; |
|  | } |
|  | else if (lastClicked == null) |
|  | { |
|  | checkEarthquakesForClick(); |
|  | if (lastClicked == null) { |
|  | checkCitiesForClick(); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | // Helper method that will check if a city marker was clicked on |
|  | // and respond appropriately |
|  | private void checkCitiesForClick() |
|  | { |
|  | if (lastClicked != null) return; |
|  | // Loop over the earthquake markers to see if one of them is selected |
|  | for (Marker marker : cityMarkers) { |
|  | if (!marker.isHidden() && marker.isInside(map, mouseX, mouseY)) { |
|  | lastClicked = (CommonMarker)marker; |
|  | // Hide all the other earthquakes and hide |
|  | for (Marker mhide : cityMarkers) { |
|  | if (mhide != lastClicked) { |
|  | mhide.setHidden(true); |
|  | } |
|  | } |
|  | for (Marker mhide : quakeMarkers) { |
|  | EarthquakeMarker quakeMarker = (EarthquakeMarker)mhide; |
|  | if (quakeMarker.getDistanceTo(marker.getLocation()) |
|  | > quakeMarker.threatCircle()) { |
|  | quakeMarker.setHidden(true); |
|  | } |
|  | } |
|  | return; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | // Helper method that will check if an earthquake marker was clicked on |
|  | // and respond appropriately |
|  | private void checkEarthquakesForClick() |
|  | { |
|  | if (lastClicked != null) return; |
|  | // Loop over the earthquake markers to see if one of them is selected |
|  | for (Marker m : quakeMarkers) { |
|  | EarthquakeMarker marker = (EarthquakeMarker)m; |
|  | if (!marker.isHidden() && marker.isInside(map, mouseX, mouseY)) { |
|  | lastClicked = marker; |
|  | // Hide all the other earthquakes and hide |
|  | for (Marker mhide : quakeMarkers) { |
|  | if (mhide != lastClicked) { |
|  | mhide.setHidden(true); |
|  | } |
|  | } |
|  | for (Marker mhide : cityMarkers) { |
|  | if (mhide.getDistanceTo(marker.getLocation()) |
|  | > marker.threatCircle()) { |
|  | mhide.setHidden(true); |
|  | } |
|  | } |
|  | return; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | // loop over and unhide all markers |
|  | private void unhideMarkers() { |
|  | for(Marker marker : quakeMarkers) { |
|  | marker.setHidden(false); |
|  | } |
|  |  |
|  | for(Marker marker : cityMarkers) { |
|  | marker.setHidden(false); |
|  | } |
|  | } |
|  |  |
|  | // helper method to draw key in GUI |
|  | private void addKey() { |
|  | // Remember you can use Processing's graphics methods here |
|  | fill(255, 250, 240); |
|  |  |
|  | int xbase = 25; |
|  | int ybase = 50; |
|  |  |
|  | rect(xbase, ybase, 150, 250); |
|  |  |
|  | fill(0); |
|  | textAlign(LEFT, CENTER); |
|  | textSize(12); |
|  | text("Earthquake Key", xbase+25, ybase+25); |
|  |  |
|  | fill(150, 30, 30); |
|  | int tri\_xbase = xbase + 35; |
|  | int tri\_ybase = ybase + 50; |
|  | triangle(tri\_xbase, tri\_ybase-CityMarker.TRI\_SIZE, tri\_xbase-CityMarker.TRI\_SIZE, |
|  | tri\_ybase+CityMarker.TRI\_SIZE, tri\_xbase+CityMarker.TRI\_SIZE, |
|  | tri\_ybase+CityMarker.TRI\_SIZE); |
|  |  |
|  | fill(0, 0, 0); |
|  | textAlign(LEFT, CENTER); |
|  | text("City Marker", tri\_xbase + 15, tri\_ybase); |
|  |  |
|  | text("Land Quake", xbase+50, ybase+70); |
|  | text("Ocean Quake", xbase+50, ybase+90); |
|  | text("Size ~ Magnitude", xbase+25, ybase+110); |
|  |  |
|  | fill(255, 255, 255); |
|  | ellipse(xbase+35, |
|  | ybase+70, |
|  | 10, |
|  | 10); |
|  | rect(xbase+35-5, ybase+90-5, 10, 10); |
|  |  |
|  | fill(color(255, 255, 0)); |
|  | ellipse(xbase+35, ybase+140, 12, 12); |
|  | fill(color(0, 0, 255)); |
|  | ellipse(xbase+35, ybase+160, 12, 12); |
|  | fill(color(255, 0, 0)); |
|  | ellipse(xbase+35, ybase+180, 12, 12); |
|  |  |
|  | textAlign(LEFT, CENTER); |
|  | fill(0, 0, 0); |
|  | text("Shallow", xbase+50, ybase+140); |
|  | text("Intermediate", xbase+50, ybase+160); |
|  | text("Deep", xbase+50, ybase+180); |
|  |  |
|  | text("Past hour", xbase+50, ybase+200); |
|  |  |
|  | fill(255, 255, 255); |
|  | int centerx = xbase+35; |
|  | int centery = ybase+200; |
|  | ellipse(centerx, centery, 12, 12); |
|  |  |
|  | strokeWeight(2); |
|  | line(centerx-8, centery-8, centerx+8, centery+8); |
|  | line(centerx-8, centery+8, centerx+8, centery-8); |
|  |  |
|  |  |
|  | } |
|  |  |
|  |  |
|  |  |
|  | // Checks whether this quake occurred on land. If it did, it sets the |
|  | // "country" property of its PointFeature to the country where it occurred |
|  | // and returns true. Notice that the helper method isInCountry will |
|  | // set this "country" property already. Otherwise it returns false. |
|  | private boolean isLand(PointFeature earthquake) { |
|  |  |
|  | // IMPLEMENT THIS: loop over all countries to check if location is in any of them |
|  | // If it is, add 1 to the entry in countryQuakes corresponding to this country. |
|  | for (Marker country : countryMarkers) { |
|  | if (isInCountry(earthquake, country)) { |
|  | return true; |
|  | } |
|  | } |
|  |  |
|  | // not inside any country |
|  | return false; |
|  | } |
|  |  |
|  | // prints countries with number of earthquakes |
|  | // You will want to loop through the country markers or country features |
|  | // (either will work) and then for each country, loop through |
|  | // the quakes to count how many occurred in that country. |
|  | // Recall that the country markers have a "name" property, |
|  | // And LandQuakeMarkers have a "country" property set. |
|  | private void printQuakes() { |
|  | int totalWaterQuakes = quakeMarkers.size(); |
|  | for (Marker country : countryMarkers) { |
|  | String countryName = country.getStringProperty("name"); |
|  | int numQuakes = 0; |
|  | for (Marker marker : quakeMarkers) |
|  | { |
|  | EarthquakeMarker eqMarker = (EarthquakeMarker)marker; |
|  | if (eqMarker.isOnLand()) { |
|  | if (countryName.equals(eqMarker.getStringProperty("country"))) { |
|  | numQuakes++; |
|  | } |
|  | } |
|  | } |
|  | if (numQuakes > 0) { |
|  | totalWaterQuakes -= numQuakes; |
|  | System.out.println(countryName + ": " + numQuakes); |
|  | } |
|  | } |
|  | System.out.println("OCEAN QUAKES: " + totalWaterQuakes); |
|  | } |
|  |  |
|  |  |
|  |  |
|  | // helper method to test whether a given earthquake is in a given country |
|  | // This will also add the country property to the properties of the earthquake feature if |
|  | // it's in one of the countries. |
|  | // You should not have to modify this code |
|  | private boolean isInCountry(PointFeature earthquake, Marker country) { |
|  | // getting location of feature |
|  | Location checkLoc = earthquake.getLocation(); |
|  |  |
|  | // some countries represented it as MultiMarker |
|  | // looping over SimplePolygonMarkers which make them up to use isInsideByLoc |
|  | if(country.getClass() == MultiMarker.class) { |
|  |  |
|  | // looping over markers making up MultiMarker |
|  | for(Marker marker : ((MultiMarker)country).getMarkers()) { |
|  |  |
|  | // checking if inside |
|  | if(((AbstractShapeMarker)marker).isInsideByLocation(checkLoc)) { |
|  | earthquake.addProperty("country", country.getProperty("name")); |
|  |  |
|  | // return if is inside one |
|  | return true; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | // check if inside country represented by SimplePolygonMarker |
|  | else if(((AbstractShapeMarker)country).isInsideByLocation(checkLoc)) { |
|  | earthquake.addProperty("country", country.getProperty("name")); |
|  |  |
|  | return true; |
|  | } |
|  | return false; |
|  | } |
|  |  |
|  | } |
|  |  |
|  |  |
|  | EarthquakeMarker.java |
|  |  |
|  |  |
|  |  |
|  | package module6; |
|  |  |
|  | import de.fhpotsdam.unfolding.data.PointFeature; |
|  | import processing.core.PConstants; |
|  | import processing.core.PGraphics; |
|  |  |
|  | /\*\* Implements a visual marker for earthquakes on an earthquake map |
|  | \* |
|  | \* @author UC San Diego Intermediate Software Development MOOC team |
|  | \* |
|  | \*/ |
|  | // TODO: Implement the comparable interface |
|  | public abstract class EarthquakeMarker extends CommonMarker |
|  | { |
|  |  |
|  | // Did the earthquake occur on land? This will be set by the subclasses. |
|  | protected boolean isOnLand; |
|  |  |
|  | // The radius of the Earthquake marker |
|  | // You will want to set this in the constructor, either |
|  | // using the thresholds below, or a continuous function |
|  | // based on magnitude. |
|  | protected float radius; |
|  |  |
|  |  |
|  | // constants for distance |
|  | protected static final float kmPerMile = 1.6f; |
|  |  |
|  | /\*\* Greater than or equal to this threshold is a moderate earthquake \*/ |
|  | public static final float THRESHOLD\_MODERATE = 5; |
|  | /\*\* Greater than or equal to this threshold is a light earthquake \*/ |
|  | public static final float THRESHOLD\_LIGHT = 4; |
|  |  |
|  | /\*\* Greater than or equal to this threshold is an intermediate depth \*/ |
|  | public static final float THRESHOLD\_INTERMEDIATE = 70; |
|  | /\*\* Greater than or equal to this threshold is a deep depth \*/ |
|  | public static final float THRESHOLD\_DEEP = 300; |
|  |  |
|  | // ADD constants for colors |
|  |  |
|  |  |
|  | // abstract method implemented in derived classes |
|  | public abstract void drawEarthquake(PGraphics pg, float x, float y); |
|  |  |
|  |  |
|  | // constructor |
|  | public EarthquakeMarker (PointFeature feature) |
|  | { |
|  | super(feature.getLocation()); |
|  | // Add a radius property and then set the properties |
|  | java.util.HashMap<String, Object> properties = feature.getProperties(); |
|  | float magnitude = Float.parseFloat(properties.get("magnitude").toString()); |
|  | properties.put("radius", 2\*magnitude ); |
|  | setProperties(properties); |
|  | this.radius = 1.75f\*getMagnitude(); |
|  | } |
|  |  |
|  | // TODO: Add the method: |
|  | // public int compareTo(EarthquakeMarker marker) |
|  |  |
|  |  |
|  | // calls abstract method drawEarthquake and then checks age and draws X if needed |
|  | @Override |
|  | public void drawMarker(PGraphics pg, float x, float y) { |
|  | // save previous styling |
|  | pg.pushStyle(); |
|  |  |
|  | // determine color of marker from depth |
|  | colorDetermine(pg); |
|  |  |
|  | // call abstract method implemented in child class to draw marker shape |
|  | drawEarthquake(pg, x, y); |
|  |  |
|  | // IMPLEMENT: add X over marker if within past day |
|  | String age = getStringProperty("age"); |
|  | if ("Past Hour".equals(age) || "Past Day".equals(age)) { |
|  |  |
|  | pg.strokeWeight(2); |
|  | int buffer = 2; |
|  | pg.line(x-(radius+buffer), |
|  | y-(radius+buffer), |
|  | x+radius+buffer, |
|  | y+radius+buffer); |
|  | pg.line(x-(radius+buffer), |
|  | y+(radius+buffer), |
|  | x+radius+buffer, |
|  | y-(radius+buffer)); |
|  |  |
|  | } |

|  |
| --- |
| // reset to previous styling |
|  | pg.popStyle(); |
|  |  |
|  | } |
|  |  |
|  | /\*\* Show the title of the earthquake if this marker is selected \*/ |
|  | public void showTitle(PGraphics pg, float x, float y) |
|  | { |
|  | String title = getTitle(); |
|  | pg.pushStyle(); |
|  |  |
|  | pg.rectMode(PConstants.CORNER); |
|  |  |
|  | pg.stroke(110); |
|  | pg.fill(255,255,255); |
|  | pg.rect(x, y + 15, pg.textWidth(title) +6, 18, 5); |
|  |  |
|  | pg.textAlign(PConstants.LEFT, PConstants.TOP); |
|  | pg.fill(0); |
|  | pg.text(title, x + 3 , y +18); |
|  |  |
|  |  |
|  | pg.popStyle(); |
|  |  |
|  | } |
|  |  |
|  |  |
|  | /\*\* |
|  | \* Return the "threat circle" radius, or distance up to |
|  | \* which this earthquake can affect things, for this earthquake. |
|  | \* DISCLAIMER: this formula is for illustration purposes |
|  | \* only and is not intended to be used for safety-critical |
|  | \* or predictive applications. |
|  | \*/ |
|  | public double threatCircle() { |
|  | double miles = 20.0f \* Math.pow(1.8, 2\*getMagnitude()-5); |
|  | double km = (miles \* kmPerMile); |
|  | return km; |
|  | } |
|  |  |
|  | // determine color of marker from depth |
|  | // We use: Deep = red, intermediate = blue, shallow = yellow |
|  | private void colorDetermine(PGraphics pg) { |
|  | float depth = getDepth(); |
|  |  |
|  | if (depth < THRESHOLD\_INTERMEDIATE) { |
|  | pg.fill(255, 255, 0); |
|  | } |
|  | else if (depth < THRESHOLD\_DEEP) { |
|  | pg.fill(0, 0, 255); |
|  | } |
|  | else { |
|  | pg.fill(255, 0, 0); |
|  | } |
|  | } |
|  |  |
|  |  |
|  | /\*\* toString |
|  | \* Returns an earthquake marker's string representation |
|  | \* @return the string representation of an earthquake marker. |
|  | \*/ |
|  | public String toString() |
|  | { |
|  | return getTitle(); |
|  | } |
|  | /\* |
|  | \* getters for earthquake properties |
|  | \*/ |
|  |  |
|  | public float getMagnitude() { |
|  | return Float.parseFloat(getProperty("magnitude").toString()); |
|  | } |
|  |  |
|  | public float getDepth() { |
|  | return Float.parseFloat(getProperty("depth").toString()); |
|  | } |
|  |  |
|  | public String getTitle() { |
|  | return (String) getProperty("title"); |
|  |  |
|  | } |
|  |  |
|  | public float getRadius() { |
|  | return Float.parseFloat(getProperty("radius").toString()); |
|  | } |
|  |  |
|  | public boolean isOnLand() |
|  | { |
|  | return isOnLand; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | } |
|  |  |
|  | LandQuakeMarker.java |
|  |  |
|  |  |
|  | package module6; |
|  |  |
|  | import de.fhpotsdam.unfolding.data.PointFeature; |
|  | import processing.core.PGraphics; |
|  |  |
|  | /\*\* Implements a visual marker for land earthquakes on an earthquake map |
|  | \* |
|  | \* @author UC San Diego Intermediate Software Development MOOC team |
|  | \* |
|  | \*/ |
|  | public class LandQuakeMarker extends EarthquakeMarker { |
|  |  |
|  |  |
|  | public LandQuakeMarker(PointFeature quake) { |
|  |  |
|  | // calling EarthquakeMarker constructor |
|  | super(quake); |
|  |  |
|  | // setting field in earthquake marker |
|  | isOnLand = true; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public void drawEarthquake(PGraphics pg, float x, float y) { |
|  | // IMPLEMENT: drawing circle for LandQuake |
|  | // DO NOT set the fill color. That will be set in the EarthquakeMarker |
|  | // class to indicate the depth of the earthquake. |
|  | // Simply draw a centered square. |
|  | // HINT: Notice the radius variable in the EarthquakeMarker class |
|  | // and how it is set in the EarthquakeMarker constructor |
|  | pg.ellipse(x, y, 2\*radius, 2\*radius); |
|  |  |
|  | } |
|  |  |
|  |  |
|  | // Get the country the earthquake is in |
|  | public String getCountry() { |
|  | return (String) getProperty("country"); |
|  | } |
|  |  |
|  |  |
|  | } |
|  |  |
|  |  |
|  | OceanquakeMarker.java |
|  |  |
|  | package module6; |
|  |  |
|  | import de.fhpotsdam.unfolding.data.PointFeature; |
|  | import processing.core.PGraphics; |
|  |  |
|  | /\*\* Implements a visual marker for ocean earthquakes on an earthquake map |
|  | \* |
|  | \* @author UC San Diego Intermediate Software Development MOOC team |
|  | \* |
|  | \*/ |
|  | public class OceanQuakeMarker extends EarthquakeMarker { |
|  |  |
|  | public OceanQuakeMarker(PointFeature quake) { |
|  | super(quake); |
|  |  |
|  | // setting field in earthquake marker |
|  | isOnLand = false; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public void drawEarthquake(PGraphics pg, float x, float y) { |
|  | //IMPLEMENT: drawing centered square for Ocean earthquakes |
|  | // DO NOT set the fill color. That will be set in the EarthquakeMarker |
|  | // class to indicate the depth of the earthquake. |
|  | // Simply draw a centered square. |
|  | // HINT: Notice the radius variable in the EarthquakeMarker class |
|  | // and how it is set in the EarthquakeMarker constructor |
|  | pg.rect(x-radius, y-radius, 2\*radius, 2\*radius); |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | } |
|  |  |
|  |  |
|  | LifeExpactency.java |
|  |  |
|  |  |
|  | package module6; |
|  |  |
|  | import processing.core.PApplet; |
|  | import de.fhpotsdam.unfolding.UnfoldingMap; |
|  | import de.fhpotsdam.unfolding.utils.MapUtils; |
|  | import parsing.ParseFeed; |
|  | import de.fhpotsdam.unfolding.providers.\*; |
|  | import de.fhpotsdam.unfolding.providers.Google.\*; |
|  |  |
|  | import java.util.List; |
|  | import de.fhpotsdam.unfolding.data.Feature; |
|  | import de.fhpotsdam.unfolding.data.GeoJSONReader; |
|  |  |
|  | import java.util.HashMap; |
|  |  |
|  |  |
|  | import de.fhpotsdam.unfolding.marker.Marker; |
|  |  |
|  | /\*\* |
|  | \* Visualizes life expectancy in different countries. |
|  | \* |
|  | \* It loads the country shapes from a GeoJSON file via a data reader, and loads the population density values from |
|  | \* another CSV file (provided by the World Bank). The data value is encoded to transparency via a simplistic linear |
|  | \* mapping. |
|  | \*/ |
|  | public class LifeExpectancy extends PApplet { |
|  |  |
|  | UnfoldingMap map; |
|  | HashMap<String, Float> lifeExpMap; |
|  | List<Feature> countries; |
|  | List<Marker> countryMarkers; |
|  |  |
|  | public void setup() { |
|  | size(800, 600, OPENGL); |
|  | map = new UnfoldingMap(this, 50, 50, 700, 500, new Google.GoogleMapProvider()); |
|  | MapUtils.createDefaultEventDispatcher(this, map); |
|  |  |
|  | // Load lifeExpectancy data |
|  | lifeExpMap = ParseFeed.loadLifeExpectancyFromCSV(this,"LifeExpectancyWorldBank.csv"); |
|  |  |
|  |  |
|  | // Load country polygons and adds them as markers |
|  | countries = GeoJSONReader.loadData(this, "countries.geo.json"); |
|  | countryMarkers = MapUtils.createSimpleMarkers(countries); |
|  | map.addMarkers(countryMarkers); |
|  | System.out.println(countryMarkers.get(0).getId()); |
|  |  |
|  | // Country markers are shaded according to life expectancy (only once) |
|  | shadeCountries(); |
|  | } |
|  |  |
|  | public void draw() { |
|  | // Draw map tiles and country markers |
|  | map.draw(); |
|  | } |
|  |  |
|  | //Helper method to color each country based on life expectancy |
|  | //Red-orange indicates low (near 40) |
|  | //Blue indicates high (near 100) |
|  | private void shadeCountries() { |
|  | for (Marker marker : countryMarkers) { |
|  | // Find data for country of the current marker |
|  | String countryId = marker.getId(); |
|  | System.out.println(lifeExpMap.containsKey(countryId)); |
|  | if (lifeExpMap.containsKey(countryId)) { |
|  | float lifeExp = lifeExpMap.get(countryId); |
|  | // Encode value as brightness (values range: 40-90) |
|  | int colorLevel = (int) map(lifeExp, 40, 90, 10, 255); |
|  | marker.setColor(color(255-colorLevel, 100, colorLevel)); |
|  | } |
|  | else { |
|  | marker.setColor(color(150,150,150)); |
|  | } |
|  | } |
|  | } |
|  |  |
|  |  |
|  | } |