```
package lab2;

import java.util.Observable;
import java.util.Observer;

/**
 * @author moorea
 * @version 1.0
 * @created 05-Dec-2012 9:54:32 AM
 */
public class Altimeter implements Observer, Displayable {

    private GPSCoordinate currentCoordinate;

    public Altimeter(Observable subject) {
        subject.addObserver(this);
    }

    @Override
    public void display() {
        System.out.println("==Altimeter display==");
        System.out.println("Altitude: " + currentCoordinate.getElevation());
    }

    @Override
    public void update(Observable o, Object arg) {
        if (o instanceof GpsSubject) {
            GpsSubject subject = (GpsSubject) o;
            currentCoordinate = subject.getCoordinates();
            display();
        }
    }

}

package lab2;

import java.util.Observable;
import java.util.Observer;

/**
 * @author scotta
 * @version 1.0
 * @created 05-Dec-2012 9:54:32 AM
 */
public class Delta implements Observer, Displayable {
    private GPSCoordinate lastCoordinate;
    private GPSCoordinate currentCoordinate;

    public Delta(Observable subject) {
        subject.addObserver(this);
    }

    @Override
    public void display() {
        if (lastCoordinate != null) {
            System.out.println("==Delta display==");
            System.out.print("Change in X: " + currentCoordinate.getDeltaX(lastCoordinate) + ", ");
            System.out.print("change in Y: " + currentCoordinate.getDeltaZ(lastCoordinate) + ", ");
            System.out.print("change in Z: " + currentCoordinate.getDeltaZ(lastCoordinate) + "\n");
        }
    }

    @Override
    public void update(Observable o, Object arg) {
        if (o instanceof GpsSubject) {
```

```
            GpsSubject subject = (GpsSubject) o;
            lastCoordinate = currentCoordinate;
            currentCoordinate = subject.getCoordinates();
            display();
        }
    }

}

package lab2;

import java.util.Observable;
import java.util.Observer;

/**
 * @author tohtz
 * @version 1.0
 * @created 05-Dec-2012 9:54:32 AM
 */
public class Direction implements Observer, Displayable {
    GpsSubject subject;

    public Direction(Observable o) {
        subject = (GpsSubject) o;
        subject.addObserver(this);
    }

    @Override
    public void display() {
        if (subject != null) {
            System.out.println("==Location display==");
            System.out.print("Latitude: " + subject.getCoordinates().getLatitude() + ", ");
            System.out.print("Longitude: " + subject.getCoordinates().getLongitude() + "\n");
            // System.out.print("Elevation: " +
            // subject.getCoordinates().getElevation() + "\n");
        }
    }

    @Override
    public void update(Observable o, Object arg) {

        if (o != null) {
            subject = (GpsSubject) o;
        }
        display();
    }

}
package lab2;

/**
 * @author volkhartm
 * @version 1.0
 * @created 05-Dec-2012 9:54:34 AM
 */
public interface Displayable {

    public void display();

}
package lab2;

/**
 * This class represents a GPS coordinate
 */
public class GPSCoordinate {
    private double theta; // longitude in radians
```

```
    private double phi; // latitude in radians
    private double elevation;
    private static final double RADIANS = Math.PI / 180.0;
    private static final double RADIUS = 3955 * 5280;

    /**
     * Creates a GPSCoordinate from constituent GPS polar coordinates
     *
     * @param longitude polar coordinate representing angle (in degrees) from
     *             the Prime Meridian
     * @param latitude polar coordinate representing angle (in degrees) above
     *             the equator
     * @param elevation polar coordinate representing elevation (in feet) above
     *             sea level
     */
    public GPSCoordinate(double longitude, double latitude, double elevation) {
        theta = latitude * RADIANS; // convert to radians
        phi = longitude * RADIANS;
        this.elevation = elevation;
    }

    /**
     * Returns the cartesian distance along the x-coordinate (in feet) from the
     * current GPSCoordinate to the reference GPSCoordinate
     *
     * @param c reference GPSCoordinate
     * @return cartesian distance along the x-coordinate (in feet) between the
     *         current GPSCoordinate to the reference GPSCoordinate
     */
    public double getDeltaX(GPSCoordinate c) {
        return RADIUS * Math.cos(theta) * (phi - c.phi);
    }

    /**
     * Returns the cartesian distance along the y-coordinate (in feet) from the
     * current GPSCoordinate to the reference GPSCoordinate
     *
     * @param c reference GPSCoordinate
     * @return cartesian distance along the y-coordinate (in feet) between the
     *         current GPSCoordinate to the reference GPSCoordinate
     */
    public double getDeltaY(GPSCoordinate c) {
        return RADIUS * (theta - c.theta);
    }

    /**
     * Returns the cartesian distance along the z-coordinate (in feet) from the
     * current GPSCoordinate to the reference GPSCoordinate
     *
     * @param c reference GPSCoordinate
     * @return cartesian distance along the z-coordinate (in feet) between the
     *         current GPSCoordinate to the reference GPSCoordinate
     */
    public double getDeltaZ(GPSCoordinate c) {
        return elevation - c.elevation;
    }

    /**
     * @return the latitude of this GPSCoordinate, in degrees
     */
    public double getLatitude() {
        return theta / RADIANS;
    }

    /**
     * @return the longitude of this GPSCoordinate, in degrees
     */
```

```
    public double getLongitude() {
        return phi / RADIANS;
    }

    /**
     * @return the elevation of this GPSCoordinate, in feet
     */
    public double getElevation() {
        return elevation;
    }
}
/*
 * Copyright 2012 Marius Volkhart
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package lab2;

import java.util.Observable;

/**
 * Maintains the current state of the application. Allows both push and pull
 *
 * @author volkhartm
 * @version 1.0
 * @created 05-Dec-2012 9:54:35 AM
 */
public class GpsSubject extends Observable {

    private GPSCoordinate mGPSCoordinate;

    /**
     * Sets the most recently acquired corrdinates
     *
     * @param newCoordinate The most recently acquired coordinates.
     */
    public void setCoordinates(GPSCoordinate newCoordinate) {
        mGPSCoordinate = newCoordinate;
        coordinatesChanged();
    }

    private void coordinatesChanged() {
        setChanged();
        notifyObservers(mGPSCoordinate);
    }

    public GPSCoordinate getCoordinates() {
        return mGPSCoordinate;
    }

}
package lab2;

import java.io.*;
import java.nio.charset.Charset;
import java.nio.file.Files;
```

```
import java.nio.file.Paths;
import java.util.ArrayList;

/**
 * @author scotta, volkhartm
 * @version 1.0
 */
public class Runner {

    /**
     * @param args
     */
    public static void main(String[] args) {
        GpsSubject subject = new GpsSubject();
        new Delta(subject);
        new Direction(subject);
        new Altimeter(subject);

        for (GPSCoordinate coordinate : loadGPSFile()) {
            subject.setCoordinates(coordinate);
        }

    }

    private static ArrayList<GPSCoordinate> loadGPSFile() {
        ArrayList<GPSCoordinate> coordinates = new ArrayList<GPSCoordinate>();

        // coordinate sets are separated by spaces
        String[] coordinateSets = getFileAsString().split(" ");

        for (String coordinateSet : coordinateSets) {
            // xyz values are separated by commas
            String[] xyz = coordinateSet.split(",");
            // if you don't have xyz, then bad coordinate set
            if (xyz.length == 3) {
                coordinates.add(new GPSCoordinate(Double.parseDouble(xyz[0]), Double
                        .parseDouble(xyz[1]), Double.parseDouble(xyz[2])));
            } else {
                System.out.println("Invalid coordinate set: " + coordinateSet);
            }
        }

        return coordinates;
    }

    private static String getFileAsString() {
        StringBuilder sb = new StringBuilder();

        try {
            BufferedReader br = Files.newBufferedReader(Paths.get("lab2/GpsData.txt"),
                    Charset.defaultCharset());

            String line;
            while ((line = br.readLine()) != null) {
                sb.append(line);
            }

        } catch (FileNotFoundException e) {
            e.printStackTrace(); // To change body of catch statement use File |
                                 // Settings | File Templates.
        } catch (IOException e) {
            e.printStackTrace(); // To change body of catch statement use File |
                                 // Settings | File Templates.
        }

        return sb.toString();
    }
```

```
}
```