```java
/*
 * CSC IE-259, May 2006
 */
package edu.harvard.cscie259.ws;

import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import javax.xml.rpc.ServiceException;

import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPException;

import org.apache.axis.client.Stub;

import org.apache.axis.message.SOAPHeaderElement;

import com.ejse.WeatherService.ServiceLocator;
import com.ejse.WeatherService.WeatherInfo;

import com.strikeiron.www.SMSTextMessagingLocator;
import com.strikeiron.www.SMSTextMessagingSoap;


/**
 * Demonstrate the magic of WebServices
 *
 * This program was written in about 1.5 hour, during an E-259 lecture.
 * It combines two web services (weather and SMS messaging) into an
 * interesting application (that send the weather for some zip code, to
 * a phone).  It then, further, makes this service available as a new
 * web service.
 *
 * The services used here were found by browsing:
 *   www.xmethods.com
 *
 * To use this program as a client:
 * 1) be sure that all the Axis libraries are on your classpath
 * 2) generate the remotes for the web services by running WSDL2Java
 *    on the wsdl for each:
 *    a) http://www.ejse.com/WeatherService/Service.asmx?WSDL
 *    b) http://ws.strikeiron.com/globalsmspro2_5?WSDL
 * 3) compile and run
 *
 * To install as a web service:
 * 1) install axis under tomcat
 * 2) copy this application’s classfile tree to
 *    $CATALINA_HOME/webapps/axis/WEB-INF/classes
 * 3) Add the following to $CATALINA_HOME/webapps/axis/WEB-INF/server-config.xml
 *    to allow axis to discover the new service
 *  <service name="wsdemo" provider="java:RPC">
 *      <parameter name="className" value="edu.harvard.cscie259.ws.WSDemo"/>
 *      <parameter name="allowedMethods" value="sendWeatherSMS"/>
 *      <parameter name="scope" value="application"/>
 *  </service>
 * 4) start tomcat
 * 5) obtain the wsdl for this service by running WSDL2Java on:
 *    http://<your tomcat’s port>/axis/services/wsdemo\?wsdl
 * 6) Build, compile and run a new client based on these remotes.
 *
 * @author blake
 */
public class WSDemo implements Runnable {

    // how often the service runs
    public static final long WAIT = 1000 * 60 * 60;

    // the source of the weather SMS messages
    public static final String MSG_SRC = "weather@cscie259.harvard.edu";
    public static final String MSG_SVC = "Weather Service";

    // you need to get an account to use the weather server
    public static final String WEATHER_USER = "you need a working e-mail address";
    public static final String WEATHER_PWD = "you need a password here";

    // you’ll also need to get an account from StrikeIron.
    // you might try talking to david.motsinger@strikeiron.com
    // who has been quite helpful.
    public static final String SMS_USER = "you need a working e-mail address";
    public static final String SMS_PWD = "you need a password here";

    // this is the queue of notifications:
    // Notifications on this list are run every WAIT ms.
    private static final List notifications = new ArrayList();

    // create the instance of this class that processes requests:
    // start it in a thread and begin processing notifications
    static { new Thread(new WSDemo()).start(); }

    /**
     * Notification
     *
     * Instances of this class represent weather for
     * one zipcode being sent to on phone number
     */
    private static class Notification {
        private final String phoneNum;
        private final String zipCode;

        /**
         * Ctor: remember the zipcode whose weather
         * to send and which zipcode to send it to.
         *
         * @param zip the zip code
         * @param phone
         */
        public Notification(String zip, String phone) {
            phoneNum = phone;
            zipCode = zip;
        }

        /**
         * Send the weather for the zip to the phone
         */
        public void run() {
            try { sendSMS(getWeather(zipCode), phoneNum); }
            catch (Exception e) {
                System.out.println("Notification failed");
                e.printStackTrace();
            }
        }

        /**
         * Send an SMS message.
         *
         * @param text the text to send
         * @param phone the phone number to which to send it.
         * @throws ServiceException can’t find the service
         * @throws SOAPException authentication problem
         * @throws RemoteException who knows: something went wrong
         */
        private void sendSMS(String text, String phone)
            throws ServiceException, RemoteException, SOAPException
        {
            System.out.println("To " + phone + ": " + text); // logging
            SMSTextMessagingSoap smsSvc
                = new SMSTextMessagingLocator()
                    .getSMSTextMessagingSoap();

            setSMSAuth(smsSvc); // authentication
            smsSvc.sendMessage(
                phone,
                MSG_SRC,
                MSG_SVC,
                text);
        }

        /**
         * Get the weather.
         *
         * @param zip the zipcode whose weather we want
         * @return a string describing the weather
         * @throws NumberFormatException
         * @throws ServiceException can’t find the service
         * @throws RemoteException who knows: something went wrong
         */
        private String getWeather(String zip)
            throws NumberFormatException, RemoteException, ServiceException
        {
            WeatherInfo weather
                = new ServiceLocator().getServiceSoap()
                    .getWeatherInfo2(
                        WEATHER_USER,
                        WEATHER_PWD,
                        Integer.parseInt(zip));
            return "in " + zip + ": " + weather.getTemprature()
                + " and " + weather.getForecast();
        }

        /**
         * I’ve been unable to get this to work, and the problem
         * may be here.  It’s a pity that authentication makes
         * this so complex...
         *
         * @param stub the stub that will generate the SOAP message
         * @throws SOAPException on failure to generate the SOAP
         */
        private void setSMSAuth(SMSTextMessagingSoap stub)
        throws SOAPException
        {
            SOAPHeaderElement header = new SOAPHeaderElement(
                "http://ws.strikeiron.com",
                "LicenseInfo");

            SOAPElement elem = header.addChildElement("RegisteredUser");
            elem.addChildElement("UserID").addTextNode(SMS_USER);
            elem.addChildElement("Password").addTextNode(SMS_PWD);

            ((Stub) stub).setHeader(header);
        }
    }

    /**
     * Main:  Run the service from the command line
     *
     * @param args command line arguments
     */
    public static void main(String[] args) {
        if (2 != args.length) {
            System.out.println("Usage: WSDemo zip phone");
            System.exit(-1);
        }

        try { new Notification(args[0], args[1]).run(); }
        catch (Exception e) {
            System.out.println("Failed!");
            e.printStackTrace();
        }
    }


    ////////////////// I N S T A N C E   M E M B E R S //////////////////
    //
    // A new instance is created for each call/request

    /**
     * Enqueue a notification task, for a single zip and phone
     *
     * @param zip the zipcode whose weather we will send
     * @param phone ... and the phone to which we’ll send it.
     */
    public String sendWeatherSMS(String zip, String phone) {
        synchronized (notifications) {
            notifications.add(new Notification(zip, phone));
        }
        return "Queued";
    }

    /**
     * Process events: wake up every so often and walk the list
     * of currently queued notification, running each.
     * Since notification is likely to take a while (it includes
     * several network interactions):
     * 1) be sure not to let the notification time affect the time
     *    until the next notification starts.
     * 2) copy the list of notification, so that we don’t hold
     *    the lock on it for too very long.
     * To make this scale even farther, it might be a good idea
     * to run each notification in its own thread.
     *
     * @see java.lang.Runnable#run()
     */
    public void run() {
        long t1;
        long t = System.currentTimeMillis() + WAIT;
        while (true) {
            while (10 < (t1 = t - System.currentTimeMillis())) {
                try { Thread.sleep(t1); }
                catch (InterruptedException e) { }
            }
            t = System.currentTimeMillis() + WAIT;

            List notes;
            synchronized (notifications) {
                notes = new ArrayList(notifications);
            }

            for (Iterator i = notes.iterator(); i.hasNext(); ) {
                ((Notification) i.next()).run();
            }
        }
    }
}
```

```
#!/bin/sh
# run with a WSDL URL as an argument
# produces the corresponding Java source, in the current directory

CLASSPATH="$JAVA_LIB/jaf-1.0.2/activation.jar:$JAVA_LIB/javamail-1.3.2/lib/mailapi.jar"
for jar in $JAVA_LIB/axis-1_3/lib/*.jar; do
    CLASSPATH=${CLASSPATH}:$jar
done

java -cp $CLASSPATH  org.apache.axis.wsdl.WSDL2Java $@
```