30/30 points for Program 9

NOTE: The program represents a capstone assignment for the course. It includes object-oriented programming, GUIs, strings, and arrays. It was therefore graded perhaps a bit tougher than previous assignments. Please be aware that this program is actually middle-of-the-road on the complexity scale for work expected in CST 283.

COMMENTS:
- Great solution, overall. Very thorough. Good OO design. Right in line with specifications.
- Error checking very good. All tests accurate.
- As always, code looks great.

```
/*********************************************************************************************
 *  This program simulates a shipping tool to calculate the shipping charge for shipping an item
 *  from a shipping center to another location in the state
 *
 *  This class creates the graphical user interface that includes a keypad to enter a zip code,
 *  a drop down list to select a shipping center and then will calculate and display the cost to the
 *  user in a text area.
 *
 *  One of the other classes will load zip code information for the state so the information the user
 *  requests can be calculated.
 *
 *  The user can clear all the data using the clear button at the bottom of the screen.
 *  The user can clear the zip code using the "C" button.
 *  The user can delete one character in the zip code using the backspace button.
 *
 *  The program will check the input for the following items:
 *      That a shipping center was selected
 *      That a 5-digit zip code was entered
 *      That the 5-digit zip code exists in the state of Michigan
 *
 *  When the user presses the calculate button, an error message is displayed if there are errors,
 *  if there are no errors, information including the shipping distance and cost are displayed in the
 *  text field.  The user can enter more data if desired, or can press the quit button to quit
 *
 *  CST 183 Programming Assignment 9
 *  @author Michael Clinesmith
 *********************************************************************************************/

import javafx.application.Application;
import javafx.scene.control.Button;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.text.Font;
import javafx.scene.text.FontPosture;
import javafx.scene.text.FontWeight;
import javafx.scene.paint.*;
import javafx.geometry.*;
import javafx.event.EventHandler;
import javafx.event.ActionEvent;

public class ShippingInterface extends Application
{
    // main node
    private BorderPane mainLayout;

    // shipping location arrays
    private final String shippingCenter[] = {"University Center", "Mackinaw City", "Grand Rapids", "Marquette", "Traverse City"};
    private final String shippingCenterZip[] = {"48710", "49701", "49501", "49855", "49684"};

    // keypad objects
    private GridPane keyPad;
    private Button keyButton[];
    private TextField zip;
    private VBox keyPadVBox;
    private Label zipLabel;
```

GRADING FOCUS
- Keypad GUI: clarity, ease of use, code implementation
- Object oriented design including required use of aggregation
- Overall code structure, documentation, general design
- Management of shipping center cities in GUI
- Calculation for distance and shipping cost
- Integration of location method for distance calculation
- Efficiency of array/file work
- Error checking; input validation
- Testing:
  * 48001 (Algonac) from Mackinaw City (241 miles; $12.03 shipping)
  * 49971 (White Pine) from Grand Rapids (372 miles; $16.34 shipping)
  * 48654 (Rose City) from University Center (55 miles; $2.73 shipping)
  * Error test: 49999, 123, 555555, etc.

Great use of the array of Button objects.

```
    // center choice objects
    private Label centerLabel;
    private ComboBox<String> centerBox;
    private VBox centerVBox;

    // informational objects
    TextArea messageArea;
    private HBox infoHBox;
    private final String FIRST_MESSAGE = "Select a shipping center, enter a zip code on the key pad then " +
                        "press the calculate button to determine the shipping costs for delivering a " +
                        "package from the shipping center to that zip code.";

    // bottom button objects
    private Button calculateButton, clearButton, quitButton;
    private HBox buttonHBox;

    // holds shipping record
    private ShippingRecord record = new ShippingRecord();

    /**
     * main method of program, used to launch graphical interface
     *
     * @param args String array - arguments are not used besides being passed to launch method
     */
    public static void main(String[] args)
    {
        // Launch the application.
        launch(args);
    }

    /**
     * Method that calls the initializeScene method and creates the scene
     * @param primaryStage Stage object used to create the stage
     */
    @Override
    public void start(Stage primaryStage)
    {
        initializeScene();

        // Set up overall scene
        Scene scene = new Scene(mainLayout, 1100, 900);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Shipping Application");
        primaryStage.show();
    }

    /**
     * Method that calls other methods to create the scene, then puts the containers together in thee main node
     */
    public void initializeScene()
    {
        createKeypad2();
        createComboBox();
        createBottomButtons();
        createInformationObjects();

        mainLayout = new BorderPane();
        mainLayout.setCenter(infoHBox);
        mainLayout.setRight(keyPadVBox);
        mainLayout.setLeft(centerVBox);
        mainLayout.setBottom(buttonHBox);
        mainLayout.setStyle("-fx-background-color: lightgreen;");

    }

    /**
     * Method creates the keypad elements including the display
     */
    public void createKeypad2()
    {
        // the display textField
        zip = new TextField("");
        zip.setFont(new Font(20));
        zip.setAlignment(Pos.CENTER);
        zip.setMaxWidth(100);
        zip.setStyle("-fx-text-inner-color: purple; -fx-background-color: lightgray;");
        zip.setEditable(false);

        // the informational label
        zipLabel = new Label("Zip Code");
        zipLabel.setStyle("-fx-text-fill: blue; -fx-font-size: 24px;");
        zipLabel.setAlignment(Pos.CENTER);

        // the keypad design
        keyButton = new Button[12];
        keyPad = new GridPane();

        // design the keypad top three rows
        for (int i=1; i<10; i++)
        {
            keyButton[i]= new Button(Integer.toString(i));
            keyButton[i].setFont(new Font(40));
            keyButton[i].setPrefSize(100, 100);
            keyButton[i].setMinSize(100,100);
            keyButton[i].setMaxSize(100,100);
            keyPad.add(keyButton[i], (i-1)%3, (i-1)/3);
            keyPad.setPrefSize(300,300);
            keyButton[i].setOnAction(new KeypadButtonHandler());
        }

        // bottom keypad row

        // 0 button
        keyButton[0] = new Button("0");
        keyButton[0].setFont(new Font(40));
        keyPad.add(keyButton[0], 1, 3);
        keyButton[0].setPrefSize(100, 100);
        keyButton[0].setMinSize(100,100);
        keyButton[0].setMaxSize(100,100);
        keyButton[0].setOnAction(new KeypadButtonHandler());

        // backspace button
        keyButton[10] = new Button("\u2190");                // back arrow unicode character
        keyButton[10].setFont(new Font(40));
        keyPad.add(keyButton[10], 0, 3);
        keyButton[10].setPrefSize(100, 100);
        keyButton[10].setMinSize(100,100);
        keyButton[10].setMaxSize(100,100);
        keyButton[10].setStyle("-fx-background-color: red;");
        keyButton[10].setOnAction(new KeypadButtonHandler());

        // clear zipcode button
        keyButton[11] = new Button("C");
        keyButton[11].setFont(new Font(40));
        keyPad.add(keyButton[11], 2, 3);
        keyButton[11].setPrefSize(100, 100);
        keyButton[11].setMinSize(100,100);
        keyButton[11].setMaxSize(100,100);
        keyButton[11].setStyle("-fx-background-color: red;");
        keyButton[11].setOnAction(new KeypadButtonHandler());

        keyPadVBox =  new VBox (20, zipLabel, zip, keyPad);
        keyPadVBox.setAlignment(Pos.CENTER);
        keyPadVBox.setPadding(new Insets(20));

    }

    /**
     * Method creates the Combo Box that holds the shipping center choices
     */
    public void createComboBox()
    {
           centerBox= new ComboBox<String>();
           centerBox.getItems().add("");
           centerBox.setStyle("-fx-font-size: 22px;");
           centerBox.setPadding(new Insets(20));

           // add contents to centerBox
           centerBox.setValue("");
           for (int i=0; i<shippingCenter.length; i++)
           {
               centerBox.getItems().add((shippingCenter[i]+ " (" + shippingCenterZip[i] + ")"));
           }

           centerBox.getSelectionModel().selectFirst();                  // select first option in ComboBox

           //   Create label identifying ComboBox
           centerLabel = new Label("Shipping Centers");
           centerLabel.setStyle("-fx-text-fill: blue; -fx-font-size: 24px;");
           centerLabel.setAlignment(Pos.CENTER);

           // put ComboBox and label together
           centerVBox = new VBox(20, centerLabel, centerBox);
           centerVBox.setAlignment(Pos.CENTER);
           centerVBox.setPadding(new Insets(20));
    }

    /**
     * Method creates the buttons to calculate the shipping cost, clear and quit the application
     */
    public void createBottomButtons()
    {
        calculateButton = new Button("\u23CE Calculate Shipping Cost");         // include unicode enter character
        calculateButton.setOnAction(new CalculateButtonHandler());
        calculateButton.setStyle("-fx-font-size: 20px;");

        clearButton = new Button("Clear Input");
        clearButton.setOnAction(new CalculateButtonHandler());
        clearButton.setStyle("-fx-font-size: 20px;");
        quitButton = new Button("Quit");
        quitButton.setOnAction((new CalculateButtonHandler()));
        quitButton.setStyle("-fx-font-size: 20px;");

        // put together button elements
        buttonHBox = new HBox (20, calculateButton, clearButton, quitButton);
        buttonHBox.setAlignment(Pos.CENTER);
        buttonHBox.setPadding(new Insets(20));
    }

    /**
     * Method creates the informational objects in the middle of the interface
     */
    public void createInformationObjects()
    {
        messageArea = new TextArea(FIRST_MESSAGE);
        messageArea.setStyle("-fx-font-size: 16px;");
        messageArea.setEditable(false);
        messageArea.setWrapText(true);
        messageArea.setPadding(new Insets(20));
        messageArea.setPrefColumnCount(20);
        messageArea.setPrefRowCount(60);
        messageArea.setMaxHeight(500);

        infoHBox = new HBox(20, messageArea);
        infoHBox.setAlignment(Pos.CENTER);

    }

    /**
     * This method inputs in a string that has a zip code between parentheses (XXXXX)
     * and extracts the zip code from the string and returns it
     *
     * it does error checking on the string and zip to ensure it is a 5-digit number code
     * if it does not find parentheses, or they are in the wrong order, or the number sequence is
     * not the right length, an empty string is returned.
     *
     * @param str String that should contain a zip code between parentheses
     * @return str a five digit zip code, or an empty string if not found
     */
    public String extractZip(String str)
    {
        String zipString ="";
        int paraLoc1,paraLoc2;

        if(str.length()!=0)
        {
            paraLoc1 = str.indexOf('(');
            paraLoc2 = str.indexOf(')');

            if (paraLoc1>=0 && paraLoc1<paraLoc2)       // make certain characters found and in right order
            {
                zipString = str.substring(paraLoc1+1, paraLoc2);    // set zip code, then make some final checks
            }

            if(zipString.length()==5)                    // make certain zip is right length and all numbers
            {
                boolean allDigits=true;
                for(int i=0; i<5; i++)
                {
                    if (!Character.isDigit(zipString.charAt(i)))    // check each character if it is a digit
                    {
                        allDigits=false;
                    }
                }

                if(!allDigits)                              // return blank string if code not all digits
                {
                    zipString="";
                }

            }
            else                                            // return blank string if code not right length
            {
                zipString="";
            }
        }
        return zipString;
    }
    /**
     * Class ButtonClickHandler handles the button click events
     */
    class KeypadButtonHandler implements EventHandler<ActionEvent>
    {
        /**
         * This method handles button click events for the keypad
         *
         * @param event ActionEvent object that contains data about a button click event
         */
        @Override
        public void handle(ActionEvent event)
        {

            boolean buttonFound = false;

            for (int i = 0; i < 10 && !buttonFound; i++)            // check if number button pressed
            {
                if (event.getSource() == keyButton[i])
                {
                    String keyInput;
                    keyInput = zip.getText();
                    if (keyInput.length() < 5)                      // add to zip code if less than 5 digits
                    {
                        keyInput = keyInput + i;
                        zip.setText(keyInput);
                    }
                    buttonFound = true;
                }
            }

            if (event.getSource() == keyButton[10])                 // backspace button
            {
                String keyInput;
                keyInput = zip.getText();
                if (keyInput.length() != 0)                         // remove a digit if at least one exists
                {
                    keyInput = keyInput.substring(0,keyInput.length()-1);
                    zip.setText(keyInput);
                }
            }
            if (event.getSource() == keyButton[11])                 // clear keypad button
            {
                zip.setText("");
            }
        }
    }
    class CalculateButtonHandler implements EventHandler<ActionEvent>
    {
        /**
         * This method handles button click events for the buttons on the bottom of the screen
         *
         *
         * @param event ActionEvent object that contains data about a button click event
         */
        @Override
        public void handle(ActionEvent event)
        {

            if (event.getSource() == calculateButton)           // calculate based on data
            {
                boolean isValid=false;
                String boxValue;                                // raw ComboBox String selection
                String zipBoxValue;                             // for shipping center zip code from ComboBox
                String zipDest;                                 // for destination zip code
                String message="";                              // for informational message to user

                boxValue = centerBox.getValue();
                zipBoxValue = extractZip(boxValue);             // gets zip code out of option choices

                zipDest = zip.getText();                        // gets destination zip code

                //  error check data to see if valid and display appropriate messages
                if (zipBoxValue.length()!=5)                                // center does not have valid zip
                {
                    message = "Please select an appropriate shipping center.";
                    messageArea.setText(message);
                }
                else if (zipDest.length()!=5)                               // destination zip not 5 digits
                {
                    message = "Please enter a 5 digit zip code.";
                    messageArea.setText(message);
                }
                else if (zipBoxValue.equals(zipDest))                       // destination equals center zip
                {
                    message = "Destination zip code matches shipping center zip code.\n\n" +
                            "Please enter a zip code that does not match the shipping center " +
                            "zip code to determine shipping costs.";
                    messageArea.setText(message);
                }
                else
                {
                    record = new ShippingRecord(zipBoxValue, zip.getText());

                    if (!record.isValid())                                  // destination zip does not exist
                    {
                        message = "Destination zip code does not exist in the state of Michigan.\n\n" +
                                "Please enter a zip code that exists in Michigan.";
                        messageArea.setText(message);
                    }
                    else                                                        // valid data, make calculations
                    {
                        message += record.toString();

                        message += "\n\nCalculating cost to ship from " + record.getCenterName() + " to " +
                                record.getDestinationName() + ".";

                        message += "\n\nDistance to destination from the shipping source: " +
                                String.format("%.2f", record.calculateShippingDistance()) + " miles.\n\n";

                        message += "Cost for shipping: $" + String.format("%.2f", record.calculateShippingCost()) + ".";

                        messageArea.setText(message);
                    }
                }

            }
            else if (event.getSource() == clearButton)              // clear button
            {
                centerBox.setValue("");
                centerBox.getSelectionModel().selectFirst();        // sets ComboBox to first blank option
                zip.setText("");
                messageArea.setText(FIRST_MESSAGE);                 // resets informational message
            }
            if (event.getSource() == quitButton)                    // quit button
            {
                System.exit(0);
            }
        }
    }

}
```

```
/*********************************************************************************************
 *  This class saves a shipping record and is used to determine the cost to ship from
 *  one location to another
 *
 *  The class calls the class that generates the  static list that holds zip codes date for the
 *  entire state of Michigan which the class can then access to determine the shipping costs
 *
 *  CST 183 Programming Assignment 9
 *  @author Michael Clinesmith
 ********************************************************************************************/

public class ShippingRecord
{
    private String centerZip;
    private String destinationZip;
    private static final double SHIPPING_COST_PER_MILE=.05;                 // given cost of shipping
    private static MiZipCodeList miZipList = new MiZipCodeList();           // create zip code list;

    /**
     * No parameter constructor
     * sets the center and destination Zips to be at University Center
     */
    public ShippingRecord()
    {
        centerZip = "48710";
        destinationZip = "48710";
    }

    /**
     * Constructor with zip code parameters
     * @param center    String: 5 digit zip code of the shipping center
     * @param dest      String: 5 digit zip code of the destination
     */
    public ShippingRecord(String center, String dest)
    {
        centerZip = center;
        destinationZip = dest;
    }

    /**
     * Mutator method to set the center zip code
     * @param centerZip String: 5 digit zip code for the shipping center
     */
    public void setCenterZip(String centerZip)
    {
        this.centerZip = centerZip;
    }

    /**
     * Mutator method to set the destination zip code
     * @param destinationZip    String: 5 digit zip code for the destination
     */
    public void setDestinationZip(String destinationZip)
    {
        this.destinationZip = destinationZip;
    }

    /**
     * Accessor method to get the current shipping cost
     * @return double: the shipping cost per mile
     */
    public static double getShippingCostPerMile()
    {
        return SHIPPING_COST_PER_MILE;
    }

    /**
     * Accessor method to get the shipping center's zip code
     * @return  double: the shipping center's 5-digit zip code
     */
    public String getCenterZip()
    {
        return centerZip;
    }

    /**
     * Accessor method to get the destination's zip code
     * @return  double: the destination's 5-digit zip code
     */
    public String getDestinationZip()
    {
        return destinationZip;
    }

    /**
     * Accessor method to get the shipping center's location's name
     * @return  String: the name of the shipping center's zip code location
     */
    public String getCenterName()
    {
        return miZipList.getName(centerZip);
    }

    /**
     * Accessor method to get the destination's name
     * @return  String: the name of the destination's zip code
     */
    public String getDestinationName()
    {
        return miZipList.getName(destinationZip);
    }

    /**
     * Accessor method to get the shipping center's latitude
     * @return  double: the latitude value
     */
    public double getCenterLatitude()
    {
        return miZipList.getLatitude(centerZip);
    }

    /**
     * Accessor method to get the destination's latitude
     * @return  double: the latitude value
     */
    public double getDestinationLatitude()
    {
        return miZipList.getLatitude(destinationZip);
    }
    /**
     * Accessor method to get the shipping center's longitude
     * @return  double: the longitude value
     */
    public double getCenterLongitude()
    {
        return miZipList.getLongitude(centerZip);
    }

    /**
     * Accessor method to get the destination's longitude
     * @return  double: the longitude value
     */
    public double getDestinationLongitude()
    {
        return miZipList.getLongitude(destinationZip);
    }

    /**
     * Method to return a string value of the information stored in the object
     * @return String: the center and destination zip codes
     */
    @Override
    public String toString()
    {
        String message;

        message = "Center ZIP: " + centerZip +
/*
                    "\nCenter Latitude: " + getCenterLatitude() +
                    "\nCenter Longitude: " + getCenterLongitude() +
                    "\nDestination Latitude: " + getDestinationLatitude() +
                    "\nDestination Longitude: " + getDestinationLongitude() +
*/
                  "\nDestination ZIP: " + destinationZip;

        return message;
    }

    /**
     * Method to calculate the shipping cost to go from the shipping center to the destination
     * @return  double: the cost to ship a product
     */
    public double calculateShippingCost()
    {
        return calculateShippingDistance() * SHIPPING_COST_PER_MILE;
    }

    /**
     * Method to calculate the distance from the shipping center to the destination
     * @return  double: the distance in miles from the shipping center to the destination
     */
    public double calculateShippingDistance()
    {
        return miZipList.calculateDistance(centerZip, destinationZip);
    }

    /**
     * Method to check that the zip codes saved are valid
     * @return  boolean: returns true if the zip codes are valid, false if at least one is not
     */
    public boolean isValid()
    {
        boolean isValid= true;
        if(!miZipList.zipCodeExists(centerZip))
        {
            isValid = false;
        }
        if(!miZipList.zipCodeExists(destinationZip))
        {
            isValid = false;
        }
        return isValid;
    }
}
```

```java
/************************************************************************************************
 *  This class saves data for a single zip code record
 *
 *  CST 183 Programming Assignment 9
 *  @author Michael Clinesmith
 ************************************************************************************************/
public class ZipCodeData   √
{
    private String zipCode, stateCode, zipName;
    private double latitude, longitude;

    /**
     * No parameter constructor
     */
    public ZipCodeData()
    {
        zipCode = "00000";
        stateCode = "NA";
        zipName = "";
        latitude = 0.0;
        longitude = 0.0;
    }

    /**
     * Constructor containing parameters for a zip code location
     * @param zip   String: the 5 digit zip code
     * @param lat   double: the latitude location of the zip code
     * @param lon   double: the longitude location of the zip code
     * @param state String: the 2 character state code
     * @param name  String: the name of the zip code location
     */
    public ZipCodeData(String zip, double lat, double lon, String state, String name)
    {
        zipCode = zip;
        stateCode = state;
        zipName = name;
        latitude = lat;
        longitude = lon;
    }

    /**
     * Copy constructor makes another zipObject
     * @param zipObject ZipCodeData: object to make a copy of
     */
    public ZipCodeData(ZipCodeData zipObject)
    {
        zipCode = zipObject.getZipCode();
        stateCode = zipObject.getStateCode();
        zipName = zipObject.getZipName();
        latitude = zipObject.getLatitude();
        longitude = zipObject.getLongitude();

    }

    /**
     * Mutator method to set the zip code
     * @param zipCode String: a 5 digit zip code
     */
    public void setZipCode(String zipCode)
    {
        this.zipCode = zipCode;
    }

    /**
     * Mutator method to set the state code
     * @param stateCode String: a 2 character state code
     */
    public void setStateCode(String stateCode)
    {
        this.stateCode = stateCode;
    }

    /**
     * Mutator method to set the latitude of the zip code
     * @param latitude double: a latitude value
     */
    public void setLatitude(double latitude)
    {
        this.latitude = latitude;
    }

    /**
     * Mutator method to set the longitude of the zip code
     * @param longitude double: a longitude value
     */
    public void setLongitude(double longitude)
    {
        this.longitude = longitude;
    }

    /**
     * Mutator method to set the name of the zip code area
     * @param zipName String: the name of the zip code area
     */
    public void setZipName(String zipName)
    {
        this.zipName = zipName;
    }

    /**
     * Accessor method to get the zip code
     * @return String: the 5-digit zip code
     */
    public String getZipCode()
    {
        return zipCode;
    }

    /**
     * Accessor method to get the state code
     * @return String: the 2-character state code
     */
    public String getStateCode()
    {
        return stateCode;
    }

    /**
     * Accessor method to get the latitude of a zip code
     * @return double: the latitude value
     */
    public double getLatitude()
    {
        return latitude;
    }

    /**
     * Accessor method to get the longitude of a zip code
     * @return double: the longitude value
     */
    public double getLongitude()
    {
        return longitude;
    }

    /**
     * Accessor method to get the name of a zip code
     * @return String: the name of the zip code area
     */
    public String getZipName()
    {
        return zipName;
    }

    /**
     * Method to return a string value of the information stored in the object
     * @return String: the values stored in the zip code object
     */
    @Override
    public String toString()
    {
        String message = "Zip Code: " + zipCode +
                    "\nState Code: " + stateCode +
                    "\nLocale Name: " + zipName +
                    "\nLatitude: " + latitude +
                    "\nLongitude: " + longitude;

        return message;
    }
}
```

```
/**************************************************************************************************
 *  This class manages the zip code data that it loads from a file
 *
 *  The class generates the static list that holds zip codes date for the entire state of Michigan
 *  which the class can then access to determine the shipping costs
 *
 *  CST 183 Programming Assignment 9
 *  @author Michael Clinesmith
 **************************************************************************************************/
import javax.swing.JOptionPane;
import java.util.Scanner;
import java.io.*;
import java.util.StringTokenizer;

public class MiZipCodeList
{

        private final int ZIP_ARRAY_MAX_SIZE = 2000;                                  // set array maximum size
        private final String ZIP_DATA_FILE = "zipMIcity.txt";                         // file to load for zip code data
        private int zipArrayElements = 0;
        private ZipCodeData[] MiZipList = new ZipCodeData[ZIP_ARRAY_MAX_SIZE];  √   // create array to store zip data
        private final static double RADIUS_EARTH = 3963.189;                          // miles

    /**
     * No-parameter constructor, but it does a lot of work, generating the zip code list
     * It load the file stored in ZIP_DATA_FILE, loads the data into an array of ZipCodeData objects
     *
     * There is error detection done to see determine if the file exists.  If it does not, the method will display
     * an error message and exit the program.
     *
     * There is error detection done to check if a line of data is in the proper format.  If it is not, the method
     * will inform the user with a message and will skip that line and go to the next one.
     *
     * It will display a message indicating to the user that data was uploaded into memory.
     *
     */
    public MiZipCodeList()
        {
            String message;
            File zipData;                         // file that holds the population data
            Scanner inputFile;                    // used to get data from file
            int i = 0;
            String inputLine;                     // String used to get a line of file input

            String zipCode, zipLatitudeString, zipLongitudeString, zipStateCode, zipName;   // data fields
            double zipLatitude, zipLongitude;
            StringTokenizer lineTokens;     //  used to get tokens from data input

            try
            {
                // Attempt to open file
                zipData = new File(ZIP_DATA_FILE);

                if (!zipData.exists())  // file not found
                {
                    message = "The file " + ZIP_DATA_FILE + " does not exist for processing data.\n" +
                            "The program will now end.";

                    JOptionPane.showMessageDialog(null, message, "File Not Found", JOptionPane.ERROR_MESSAGE);
                    System.exit(0);
            }

            inputFile = new Scanner(zipData);
            // build list of zip code data

            // Read input file while more data exist
            // Read one line at a time (assuming each line contains one username)
            i = 0;          // used to work through array elements

            while (inputFile.hasNext())
            {
                try                     // used to catch possible error in formating in data file
                {
                    inputLine = inputFile.nextLine();
                    lineTokens = new StringTokenizer(inputLine);
```

Great job with the javadoc.

```
                    // Read all data on one line
                    zipCode = lineTokens.nextToken();
                    zipLatitudeString = lineTokens.nextToken();
                    zipLongitudeString = lineTokens.nextToken();
                    zipStateCode = lineTokens.nextToken();
                    zipName = lineTokens.nextToken();

                    // format data
                    zipLatitude = Double.parseDouble(zipLatitudeString);
                    zipLongitude = Double.parseDouble(zipLongitudeString);

                    // add to zip array
                    MiZipList[i] = new ZipCodeData(zipCode, zipLatitude, zipLongitude, zipStateCode, zipName);

                    i++;    // count number of valid lines of data
                }
                catch (NumberFormatException e)
                {
                    message = "There was an error processing a line of data in " + ZIP_DATA_FILE + ".\n" +
                            "The line will be skipped and the program will continue processing.";
                    JOptionPane.showMessageDialog(null, message, "Data Corrupted", JOptionPane.ERROR_MESSAGE);

                }

            }
            zipArrayElements = i;    // Capture number of elements
            inputFile.close();
        }
        catch (IOException e)  // if error loading data, give error message and end program
        {
            message = "There was an error opening the file " + ZIP_DATA_FILE + ".\n" +
                    "The program will now end.";

            JOptionPane.showMessageDialog(null, message);
            System.exit(0);
        }

        message = "The data from the file " + ZIP_DATA_FILE +
                "\nis now uploaded into memory.";
        JOptionPane.showMessageDialog(null, message);

    }

/**
```

```
     * Accessor method returning the number of elements in the array
     * @return  int: The number of elements in the array
     */
       public int getNumberOfElements()
       {
           return zipArrayElements;
       }

    /**
     * Method to calculate the distance between to zip codes
     * @param zip1  String: the first 5-digit zip code
     * @param zip2  String: the second 5-digit zip code
     * @return  double: The distance in miles between the two zip codes
     */
       public double calculateDistance(String zip1, String zip2)
       {
           int index1, index2;
           double la1, lo1, la2, lo2;
           double distance = -1.0;                                   // flag value if distance not found
           index1 = zipCodeIndex( zip1 );                            // gets index value of first zip code
           index2 = zipCodeIndex( zip2 );                            // gets index value of second zip code

           if (index1 >=0 && index2 >=0)                             // records found
           {
               // Convert latitude and longitude to radians
               la1 = Math.toRadians(MiZipList[index1].getLatitude());
               lo1 = Math.toRadians(MiZipList[index1].getLongitude());
               la2 = Math.toRadians(MiZipList[index2].getLatitude());
               lo2 = Math.toRadians(MiZipList[index2].getLongitude());

               // Calculate great circle distance and return
               distance = RADIUS_EARTH * Math.acos(Math.sin(la1) * Math.sin(la2)
                       + Math.cos(la1) * Math.cos(la2) * Math.cos(lo2 - lo1));
           }
           return distance;
       }

    /**
     * Method to return a ZipCodeData object the cooresponds to the given 5-digit zip code
     * @param zip String: A 5-digit zip code
     * @return ZipCodeData: the object with the 5-digit zip code, or a blank object if it did not exist
     */
       public ZipCodeData findZipCodeDataObject(String zip)
       {
           ZipCodeData zipObject;
           int index = zipCodeIndex(zip);
           if (index>=0)                                   // record found
           {
               zipObject = new ZipCodeData(MiZipList[index]);    // makes copy of record requested
           }
           else
           {
               zipObject = new ZipCodeData();                    // returns empty record
           }
           return zipObject;
       }

    /**
     * Method to get the name cooresponding to a zip code
     * @param zip String: a 5-digit zip code
```

```
 * @return  String: the name the cooresponds to the zip code, or blank if it does not exist
 */
    public String getName(String zip)
    {
        String name="";
        int index = zipCodeIndex(zip);
        if (index>=0)                                   // record found
        {
            name = MiZipList[index].getZipName();
        }

        return name;
    }

/**
 * Method to get the latitude of a given zip code
 * @param zip String: a 5-digit zip code
 * @return  double: the latitude of the zip code or returns -360.0 if it does not exist
 */
    public double getLatitude(String zip)
    {
        double lat = -360.0;                            // some never used value
        int index = zipCodeIndex(zip);
        if (index>=0)                                   // record found
        {
            lat = MiZipList[index].getLatitude();
        }
        return lat;
    }

/**
 * Method to get the longitude of a given zip code
 * @param zip String: a 5-digit zip code
 * @return  double: the longitude of the zip code or returns -360.0 if it does not exist
 */
    public double getLongitude(String zip)
    {
        double lon = -360.0;                            // some never used value
        int index = zipCodeIndex(zip);
        if (index>=0)                                   // record found
        {
            lon = MiZipList[index].getLongitude();
        }
        return lon;
    }

/**
 * Method to get the state code of a zip code
 * @param zip   String: a 5-digit zip code
 * @return  String: "MI" if the code exists or "NA" if it does not (all zips are in MI)
 */
    public String getStateCode(String zip)
    {
        String stCode= "NA";

        if(zipCodeExists(zip))
        {
            stCode = "MI";
        }
        return stCode;
        }

    /**
     * Method to determine if a given zip code exists
     * @param zip   String: The zip code being searched
     * @return  boolean: true if the zip code is found, false if it was not
     */
        public boolean zipCodeExists(String zip)
        {
            boolean doesExist=true;
            int index = zipCodeIndex(zip);
            if (index<0)
            {
                doesExist = false;
            }
            return doesExist;
        }

    /**
     * Method that gets the index of a zip code in the MiZipList array
     * @param zip   String: a 5-digit zip code
     * @return  int:    The index of the zip code in the array, or -1 if it is not found
     */
        private int zipCodeIndex(String zip)
        {
            int index=-1;
            boolean isFound = false;

            for (int i=0; i<zipArrayElements && !isFound; i++)
            {
                if (MiZipList[i].getZipCode().equals(zip))
                {
                    index = i;
                    isFound = true;
                }
            }
            return index;
        }

}
```