Tom Wallis
2025138
13 hours

Dogs.java:

```
import javax.swing.*;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.sql.*;

public class Dogs extends JFrame {
	//swing components
	private JButton motherButton = new JButton(),
			fatherButton = new JButton(),
			originalButton = new JButton("Original dog"),
			restartButton = new JButton("Choose another dog"),
			quitButton = new JButton("Quit");
	private JLabel dogInfo = new JLabel(),
			pedInfo = new JLabel();
	private Stack<String> dogStack = new Stack<String>();
	private String topMother, topFather; //the mother and father names of the
dog
	//at the top of the dogStack
	//database stuff
	private static final String username = "lev3_14_2025138w";
	private static final String password = "2025138w";
	private static final String connStr =
"jdbc:postgresql://yacata.dcs.gla.ac.uk:5432/lev3_14_2025138w";
	private static Connection conn;
	private static PreparedStatement dogInfoStmt, siblingStmt, childStmt,
	grandchildStmt, parentStmt, breedStmt,
	parentBreedStmt;

	//output a supplied error message, the details of the exception, and then
die
	private static void doError(Exception e, String msg)
	{
		System.out.println(msg);
		e.printStackTrace();
		System.exit(1);
	}

	//output any warnings associated with <conn> or <stmt>
	private static void printWarnings(Connection conn, Statement stmt)
	{
		try {
			SQLWarning currConnWarn = conn.getWarnings();
			while (currConnWarn != null)
				System.out.println("Warning: " +
currConnWarn.getMessage());
			SQLWarning currStmtWarn = stmt.getWarnings();
			while (currStmtWarn != null)
				System.out.println("Warning: " +
```

```
currStmtWarn.getMessage());
            } catch(Exception e) {
                  doError(e, "Problem getting warnings");
            }
      }

      static void init() {
            //database stuff - load driver, and then obtain connection
            try {
                  // ************  ENTER CODE HERE TO LOAD THE DRIVER *****
                  Class.forName("org.postgresql.Driver");
            } catch(Exception e) {
                  doError(e, "Failed to load oracle driver");
            }
            try {
                  // *********** ENTER CODE HERE TO CONNECT TO THE DATABASE
*****
                  conn = DriverManager.getConnection(connStr, username,
password);
            } catch(Exception e) {
                  doError(e, "Failure to obtain connection: " + connStr);
            }
      }

      private Dogs() {
            //create prepared statements for later use by the interface code
            //this will form a nested query to tell us how many dogs an owner
has
            //Columns should be renamed, if necessary
            //Result: ownerid, noOfDogs
            // ANSWERING QUESTION 4.
            String ownerCount = "select owner.ownerid, COUNT(dog.ownerid) as
noOfDogs " +
                        "from owner, dog " +
                        "where owner.ownerid = dog.ownerid " +
                        "group by owner.ownerid ";


            //finds all the data that are single values per dog
            String dogInfoQuery =
                        "SELECT Dog.breedname, Dog.mothername, " +
                              "       Dog.fathername, Kennel.kennelname, "
+
                              "       Kennel.address, Owner.name, " +
                              "       OwnerCount.noOfDogs " +
                              "FROM Dog, Kennel, " +
                              "     Owner, (" + ownerCount + ") OwnerCount
" +
                              "WHERE Dog.kennelname=Kennel.kennelname AND
" +
                              "      Dog.ownerid=Owner.ownerid AND " +
                              "      Owner.ownerid=OwnerCount.ownerid AND
" +
                              "      Dog.name=?"; //parameterised by dog
name
            try {
                  dogInfoStmt = conn.prepareStatement(dogInfoQuery);
            } catch(SQLException e) {
                  doError(e, "Dog info statement failed to compile: " +
dogInfoQuery);
            }
            //finds the dog's siblings (incl. half siblings), but not the dog
```

```
itself
            //parameterised by (1) the dog's mothername, (2) the dog's
fathername and (3) the dog's
            //own name
            //Result: name
            // ANSWERING QUESTION 4.
            String siblingQuery = "Select mothername, fathername, name "+
                        "FROM Dog " +
                        "where (mothername=? OR fathername=?) and name<>?";

            try {
                  siblingStmt = conn.prepareStatement(siblingQuery);
            } catch(SQLException e) {
                  doError(e, "Sibling statement failed to compile: " +
siblingStmt);
            }
            //finds the dog's children. We don't know if it's a mother or a
father, so
            //we should match either (although presumably only one for any given
dog!)
            //the query should be parametrised by (1) the dog's mothername, and
(2) the dog's fathername
            //Result: name
            //Sorted by: name
            // ANSWERING QUESTION 4.
            String childQuery = "select name " +
                        "from Dog " +
                        "where mothername=? or fathername=? " +
                        "order by dog.name DESC";
            try {
                  childStmt = conn.prepareStatement(childQuery);
            } catch(SQLException e) {
                  doError(e, "Child query failed to compile: " + childQuery);
            }
            //finds the dog's grandchildren, again matching father or mother as
above
            //you will need two copies of the relation Dog: D1 and D2
            //in this query the current dog "?" should be D1's mother or father,
and D2 should be the
            //grandchild
            //Result: D2.name
            // ANSWERING QUESTION 4.
            String grandchildQuery = "SELECT DISTINCT D2.name " +
                        "FROM Dog as D1, Dog as D2 " +
                        "WHERE (D2.mothername=D1.name OR D2.fathername=D1.name)
" +
                        "AND (D1.mothername=? OR D1.fathername=?)";
            try {
                  grandchildStmt = conn.prepareStatement(grandchildQuery);
            } catch(SQLException e) {
                  doError(e, "Granchild query failed to compile: " +
grandchildQuery);
            }
            //finds parents of two dogs at once, the parameters are the two dogs

            String parentQuery = "SELECT mothername, fathername " +
                        "FROM Dog " +
                        "WHERE name=? OR name=?";
            try {
                  parentStmt = conn.prepareStatement(parentQuery);
            } catch(SQLException e) {
                  doError(e, "Failed to compile grandparent statement: " +
```

```
parentQuery);
            }
            //a query to return all known breeds for the supplied dog, used in
            //getParentsBreeds()

            String parentBreedQuery = "SELECT Parent.breedname " +
                        "FROM Dog Curr, Dog Parent " +
                        "WHERE (Curr.mothername=Parent.name OR " +
                        "       Curr.fathername=Parent.name) " +
                        "  AND Curr.name=?";
            try {
                  parentBreedStmt = conn.prepareStatement(parentBreedQuery);
            } catch(SQLException e) {
                  doError(e, "Failed to compile parent breed stmt: " +
parentBreedStmt);
            }


            //info text boxes and labels
            JPanel dogInfoPanel = new JPanel();
            dogInfoPanel.add(dogInfo);
            dogInfoPanel.setBorder(
                        BorderFactory.createCompoundBorder(
                                    BorderFactory.createTitledBorder("Dog
information"),
                                    BorderFactory.createEmptyBorder(10,10,10,10)
));
            JPanel pedInfoPanel = new JPanel();
            pedInfoPanel.add(pedInfo);
            pedInfoPanel.setBorder(
                        BorderFactory.createCompoundBorder(
                                    BorderFactory.createTitledBorder("Pedigree")
,
                                    BorderFactory.createEmptyBorder(10,10,10,10)
));
            //positioning of components
            Container cp = getContentPane();
            cp.setLayout(new BoxLayout(cp,BoxLayout.X_AXIS));
            Box buttonBox = Box.createVerticalBox();
            buttonBox.add(motherButton);
            buttonBox.add(fatherButton);
            buttonBox.add(originalButton);
            buttonBox.add(Box.createVerticalGlue());
            buttonBox.add(restartButton);
            buttonBox.add(quitButton);
            Box infoBox = Box.createVerticalBox();
            infoBox.add(dogInfoPanel);
            infoBox.add(pedInfoPanel);
            cp.add(infoBox);
            cp.add(buttonBox);
            //button handling
            ButtonHandler bh = new ButtonHandler();
            motherButton.addActionListener(bh);
            fatherButton.addActionListener(bh);
            originalButton.addActionListener(bh);
            restartButton.addActionListener(bh);
            quitButton.addActionListener(bh);
            //set up the dog to be displayed
            dogStack.push(getDogChoice());
            redisplay(); //display dog info
            originalButton.setEnabled(false); //cannot go to previous dog
            //window settings
```

```
            Dimension screen = getToolkit().getScreenSize();
            setBounds(0, 0, 700, 700); //position (0,0) and size 700x700
            setDefaultCloseOperation(EXIT_ON_CLOSE);
            setTitle("Dog information");
            setVisible(true);
      }



      //Returns the most accurate possible breed information for the supplied
dog's
      //parents. For a parent that is not in the database we will assume that
the
      //breed is the same as the supplied dog's, unless the supplied dog is not
in
      //the database in which case we assume <assumeBreed>.
      private Vector<String> getParentBreeds(String dogname, String assumeBreed)
      {
            Vector<String> retval = new Vector<String>(); //breeds we've found
            //get as many parent breeds as possible
            try {
                  parentBreedStmt.setString(1, dogname);
                  ResultSet results = parentBreedStmt.executeQuery();
                  printWarnings(conn, parentBreedStmt);
                  //Get all the breed information available from the database.
                  //Due to key constraints there are at most 2 rows in the
result.
                  while (results.next())
                        retval.add(results.getString(1));
            } catch(SQLException e) {
                  doError(e, "Parent breed query failed to execute");
            }
            //if we have all the possible parents, then we can stop now,
otherwise use
            //best guesses as per the comment above
            if (retval.size() == 2)
                  return retval;
            else {
                  try {
                        dogInfoStmt.setString(1, dogname);
                        ResultSet results = dogInfoStmt.executeQuery();
                        printWarnings(conn, dogInfoStmt);
                        if (results.next()) //try to use the supplied dog's real
breed
                              while (retval.size() < 2)
                                    retval.add(results.getString(1));
                        else //if real breed is not available use the assumed
breed
                              while (retval.size() < 2)
                                    retval.add(assumeBreed);
                  } catch(SQLException e) {
                        doError(e, "Failed to execute breed query");
                  }
            }
            return retval;
      }

      //displays the dog data present on the top of the stack of names
      private void redisplay() {
            if (dogStack.empty()) {
                  System.out.println("Unexpectedly exhausted dogs!");
                  System.exit(1);
```

```
            }
            String dogName = dogStack.peek();
            ResultSet results;
            //per-dog information
            String name = null, breed = null, mother = null, father = null,
                        kennel = null, kennAddr = null, owner = null;
            int ownerDogs = 0;
            //information with arbitrary numbers of values per dog
            Vector<String> siblings = new Vector<String>(),
                        children = new Vector<String>(),
                        grandchildren = new Vector<String>(),
                        grandparents = new Vector<String>();
            try {
                  dogInfoStmt.setString(1, dogName);
                  results = dogInfoStmt.executeQuery();
                  printWarnings(conn, dogInfoStmt);
                  //make sure that the dog we are trying to display is in the DB
                  if (!(results.next())) {
                        //remove it and display a warning message
                        JOptionPane.showMessageDialog(this,
                                    "Information for " + dogStack.pop() + "
missing in database.");
                        if (dogStack.size() == 1) //in case there is no stack to
ascend
                              originalButton.setEnabled(false);
                        return; //stop now
                  }
                  //since the data is in the database we continue as normal
                  breed = results.getString(1);
                  topMother = mother = results.getString(2);
                  topFather = father = results.getString(3);
                  kennel = results.getString(4);
                  kennAddr = results.getString(5);
                  owner = results.getString(6);
                  ownerDogs = results.getInt(7);
            } catch(SQLException e) {
                  doError(e, "Failed to execute dog info query for " + dogName);
            }
            try {
                  //looking for dogs with the same parents
                  siblingStmt.setString(1, mother);
                  siblingStmt.setString(2, father);
                  siblingStmt.setString(3, dogName); //don't match the dog
itself
                  results = siblingStmt.executeQuery();
                  printWarnings(conn, siblingStmt);
                  while (results.next()) //store each sibling name for later
display
                        siblings.add(results.getString(1));
            } catch(SQLException e) {
                  doError(e, "Failed to execute sibling query for " + dogName);
            }
            try {
                  //looking for child dogs
                  childStmt.setString(1, dogName); childStmt.setString(2,
dogName);
                  results = childStmt.executeQuery();
                  printWarnings(conn, childStmt);
                  while (results.next()) //store each name for display later on
                        children.add(results.getString(1));
            } catch(SQLException e) {
                  doError(e, "Failed to execute child query for " + dogName);
```

```
            }
            try {
                  //looking for grandchildren
                  grandchildStmt.setString(1, dogName);
                  grandchildStmt.setString(2, dogName);
                  results = grandchildStmt.executeQuery();
                  printWarnings(conn, grandchildStmt);
                  while (results.next())
                        grandchildren.add(results.getString(1));
            } catch(SQLException e) {
                  doError(e, "Failed to execute grandchild query for " +
dogName);
            }
            try {
                  parentStmt.setString(1, mother); //maternal grandparents
                  parentStmt.setString(2, father); //and paternal grandparents
                  results = parentStmt.executeQuery();
                  printWarnings(conn, parentStmt);
                  while (results.next()) { //zero, one or two rows as a result
                        if (results.getString(1) != null)
                              grandparents.add(results.getString(1));
//grandmother
                        if (results.getString(2) != null)
                              grandparents.add(results.getString(2));
//grandfather
                  }
            } catch(SQLException e) {
                  doError(e, "Failed to execute grandparent query for " +
dogName);
            }


            Vector <String> ancestors = getAncestors(dogName,null);
            Vector <String> des = getDescendents(dogName,null);

            // THIS IS ANSWERING QUESTION 7.
            int length = 0;

            String formattedAncestorData = "";
            for (String ancestor : ancestors) {
                  formattedAncestorData = formattedAncestorData + ancestor + ",
";
                  length += ancestor.length() + 2;
                  if (length > 25) {
                        formattedAncestorData =
formattedAncestorData.substring(0, formattedAncestorData.length() - 2) + "<br>";
                        length = 0;
                  }
            }

            length = 0;
            String formattedDescendentData = "";
            for (String descendent : des) {
                  length += descendent.length() + 2;
                  formattedDescendentData = formattedDescendentData + descendent
+ ", ";
                  if (length > 25) {
                        formattedDescendentData =
formattedDescendentData.substring(0, formattedDescendentData.length() - 2) +
"<br>";
                  }
            }
```

```
            if (kennAddr == null) kennAddr = "Unknown";

            //output dog info data
            dogInfo.setText("<html>Name: " + dogName + "<br>" +
                        "Breed: " + breed + "<br>" +
                        "Kennel: " + kennel + "<br>" +
                        "Address: " + kennAddr + "<br>" +
                        dogName + "'s owner: " +
                        owner + " (owns " + ownerDogs + " dogs)" + "</html>");
            //put together a message about how many parents are not named in the
DB
            String parentMissingInfo = ""; //nothing by default
            if (father == null && mother == null)
                  parentMissingInfo = "(2 missing)";
            else if (father == null ^ mother == null) //exclusive OR
                  parentMissingInfo = "(1 missing)";
            //message about how many grandparents are missing, assume dog isn't
inbred
            String grandparentMissingInfo = "";
            if (grandparents.size() != 4)
                  grandparentMissingInfo = "(" + (4 - grandparents.size()) + "
missing)";
            //put together the required breed information



            Vector<String> breeds = getParentBreeds(mother, breed);
            breeds.addAll(getParentBreeds(father, breed));
            breeds.add("ZZZZZZZZZZZZZZZZZZZZZZZ"); //sentinel to help the
output code


            String[] breedStrings = breeds.toArray(new String[breeds.size()]);
            Arrays.sort(breedStrings);
            boolean isPureBred = (breedStrings[0]).equals(breedStrings[1]) &&
                        (breedStrings[1]).equals(breedStrings[2]) &&
                        (breedStrings[2]).equals(breedStrings[3]);
            String breedingString = "";
            for (int i = 0; i < 4; i++)
                  for (int j = i; j < 5; j++)
                        if (!(breedStrings[i].equals(breedStrings[j]))) {
                              breedingString += "(" + (j - i) + "/4 " +
breedStrings[i] + ") ";
                              i = j - 1; //start again at the start of the next
run of breeds
                              break;
                        }
            //output pedigree info
            pedInfo.setText("<html>" + dogName + "'s siblings: " + siblings +
"<br>" +
                        /*dogName + "'s Parents: " +
                        (father != null ? father : "") +
                        (father != null && mother != null ? ", " : "") +
                        (mother != null ? mother : "") +
                        " " + parentMissingInfo +
                        "<br>" + dogName + "'s Grandparents: " + grandparents +
                        " " + grandparentMissingInfo +*/
                        // THIS IS ANSWERING QUESTION 7.
                        dogName + "'s Ancestors: " + formattedAncestorData +
```

```
                        "<br>" + dogName + "'s Descendants: " +
formattedDescendentData + "<br>" +
                        //dogName + "'s Grandchildren: " + grandchildren +
                        "<br><br>" +
                        (isPureBred ? "Purebred<br>" : "Not purebred<br>") +
                        "Breeding: " + breedingString +
                        "</html>");
            if (mother != null)
                  motherButton.setText(mother + "'s details");
            else {
                  motherButton.setText("Not available");
                  motherButton.setEnabled(false);
            }
            if (father != null)
                  fatherButton.setText(father + "'s details");
            else {
                  fatherButton.setText("Not available");
                  fatherButton.setEnabled(false);
            }
      }

      //gets a dog name that the user chooses somehow
      private String getDogChoice() {
            try {
                  Statement dogStmt = conn.createStatement();
                  ResultSet dogRes = dogStmt.executeQuery("SELECT name " +
                              "FROM Dog " +
                              "ORDER BY name");
                  Vector<String> dogNames = new Vector<String>();
                  while (dogRes.next())
                        dogNames.add(dogRes.getString(1));
                  if (dogNames.size() == 0) {
                        System.out.println("No dogs present in database,
exiting.");
                        System.exit(1);
                  }
                  String choice;
                  do {
                        //use a modal JOptionPane dialog to let the user choose
between dogs
                        choice = (String)(JOptionPane.showInputDialog(this,
//parent dialog
                                    "Select a dog to view", //message
                                    "Select a dog", //title of dialog
                                    JOptionPane.PLAIN_MESSAGE, //urgency
                                    null,
                                    dogNames.toArray(), //choices
                                    dogNames.get(0))); //default choice
                  } while (choice == null); //keep going until the user picks
one
                  return choice;
            } catch(SQLException e) {
                  doError(e, "Failed to get dog name choices from DB");
            }
            return null; //unreachable
      }

      //handler to do the work of all the buttons, namely the parent buttons,
the
      //back button and the exit button
      private class ButtonHandler implements ActionListener {
            public void actionPerformed(ActionEvent ae) {
```

```
                  Object src = ae.getSource(); //the button that causes the
event
                  if (src == motherButton) {
                        dogStack.push(topMother); //put mother at the top of the
stack
                        originalButton.setEnabled(true);
                        motherButton.setEnabled(true);
                        fatherButton.setEnabled(true);
                        redisplay(); //refresh the details on screen to show the
mother
                  } else if (src == fatherButton) {
                        dogStack.push(topFather);
                        originalButton.setEnabled(true);
                        motherButton.setEnabled(true);
                        fatherButton.setEnabled(true);
                        redisplay();
                  } else if (src == originalButton) {
                        dogStack.pop(); //discard currently displayed dog
                        if (dogStack.size() == 1)
                              originalButton.setEnabled(false); //can't go any
further back
                        motherButton.setEnabled(true);
                        fatherButton.setEnabled(true);
                        redisplay();
                  } else if (src == restartButton) {
                        dogStack.clear();
                        dogStack.push(getDogChoice());
                        redisplay();
                  } else if (src == quitButton) {
                        try {
                              conn.close();
                        } catch(SQLException e) {
                              System.out.println("Cannot close DB connection");
                              e.printStackTrace();
                        } finally {
                              System.exit(0);
                        }
                  }
            }
      }

      public static void main(String[] args) {
            init();
            if (args.length == 3)
            {
                  //Example usage: java Dogs 10 'Guid blinds show' '10-06-2003'
                  int dogid = Integer.parseInt(args[0]);
                  String showname = args[1];
                  String showdate = args[2];
                  disqualifyDogFromShow(dogid, showname, showdate);
            }
            else
            {
                  Dogs window = new Dogs();
            }
      }

      private static void disqualifyDogFromShow(int dogid, String showname,
String showdate)
      {
            //INSERT CODE HERE TO DISQUALIFY DOG FROM SHOW // TODO
      }
```

```
	private boolean parentFunction = true;
	private Vector<String> getAncestors(String dogname, Vector<String> anc) {
		boolean thisisParent = parentFunction; // If we're the parent
function, this will be True.
		parentFunction = false; // Anything recursing after this will have
// THIS IS ANSWERING QUESTION 6.

thisIsParent=false.
		if (anc == null) anc = new Vector<String>();

		try {
			parentStmt.setString(1, dogname);
			parentStmt.setString(2, dogname);
			ResultSet resultsRS = parentStmt.executeQuery();

			// Data collection
			ResultSetMetaData resultsMeta = resultsRS.getMetaData();
			int noColumns = resultsMeta.getColumnCount();
			ArrayList<String> data = new ArrayList<String>();
			if (resultsRS.next()) {
				for (int i = 1; i <= noColumns; i++) {
					//if (resultsRS.next()) {
					String currentAncestor = resultsRS.getString(i);
					if (currentAncestor != null) {
						data.add(currentAncestor);
					}
				}
			}

			// Main loop
			for (String currentDog : data) {
				getAncestors(currentDog, anc);
			}
			if (!thisisParent) {
				anc.add(dogname);
			} else {
				parentFunction = true; // Set it up again for next time!
			}
		}
		catch(Exception e) {
			doError(e, "Failed to execute ancestor query in getBreeding");
		}
		return anc;

	}



	private Vector<String> getDescendents(String dogname, Vector<String> desc)
{
		boolean thisisParent = parentFunction; // If we're the parent
function, this will be True.
		parentFunction = false; // Anything recursing after this will have
thisIsParent=false.
		if (desc == null) desc = new Vector<String>();

		try {
```

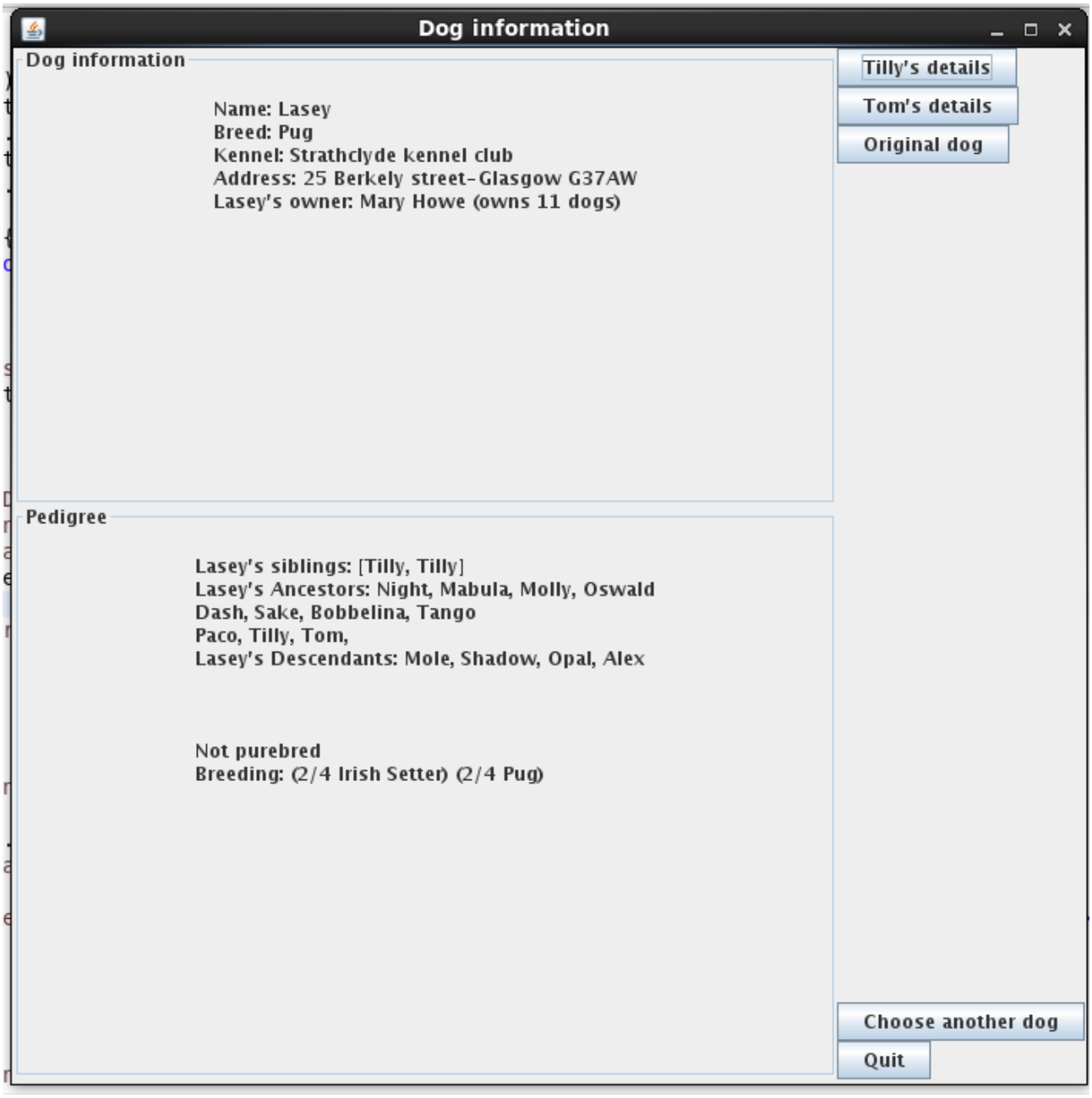
```
                childStmt.setString(1, dogname);
                childStmt.setString(2, dogname);
                ResultSet resultsRS = childStmt.executeQuery();

                // Data collection
                ResultSetMetaData resultsMeta = resultsRS.getMetaData();
                int noColumns = resultsMeta.getColumnCount();
                ArrayList<String> data = new ArrayList<String>();
                if (resultsRS.next()) {
                      boolean looping = true;
                      while (looping) {
                            for (int i = 1; i <= noColumns; i++) {

                                  String currentDescendent =
resultsRS.getString(i);
                                  if (currentDescendent != null) {
                                        data.add(currentDescendent);
                                  }
                            }
                            looping = resultsRS.next();
                      }
                }

                // Main loop
                for (String currentDog : data) {
                      getDescendents(currentDog, desc);
                }
                if (!thisisParent) {
                      desc.add(dogname);
                } else {
                      parentFunction = true; // Set it up again for next time!
                }
          }
          catch(Exception e) {
                doError(e, "Failed to execute ancestor query in getBreeding");
          }
          return desc;

      }

}
```

Create statements

```
-- DROP TABLE Breed;
-- DROP TABLE Show;
-- DROP TABLE Kennel;
-- DROP TABLE Dog;
-- DROP TABLE Owner;
-- DROP TABLE Attendance;


CREATE TABLE Breed (
      breedname VARCHAR(64) PRIMARY KEY
);

CREATE TABLE Show (
```

```
      showname  VARCHAR(64) NOT NULL,
      opendate  VARCHAR(12) NOT NULL,
      closedate VARCHAR(12),
      PRIMARY KEY(showname, opendate)
);

CREATE TABLE Kennel (
      kennelname VARCHAR(64) PRIMARY KEY,
      address    VARCHAR(64) NOT NULL,
      phone      VARCHAR(16)
);

CREATE TABLE Owner (
      ownerid INT            PRIMARY KEY,
      name    VARCHAR(32)    NOT NULL,
      phone   VARCHAR(16)
);

CREATE TABLE Dog (
      dogid      INT           PRIMARY KEY,
      name       VARCHAR(32),
      ownerid    INT         REFERENCES Owner(ownerid),
      kennelname VARCHAR(64),
      breedname  VARCHAR(64),
      mothername VARCHAR(64),
      fathername VARCHAR(64)
);

CREATE TABLE Attendance ( -- RELATIONSHIP BETWEEN Dog AND Show
      dogid    INT           REFERENCES Dog(dogid),
      showname VARCHAR(64),
      opendate VARCHAR(12),
      place    INT,
      FOREIGN KEY(showname, opendate) REFERENCES Show(showname, opendate),
      PRIMARY KEY(showname, opendate, place)
);
```

Screenshots


Dog information
Dog information
Name: Lasey
Breed: Pug
Kennel: Strathclyde kennel club
Address: 25 Berkely street-Glasgow G37AW
Lasey's owner: Mary Howe (owns 11 dogs)
Pedigree
Lasey's siblings: [Tilly, Tilly]
Lasey's Ancestors: Night, Mabula, Molly, Oswald
Dash, Sake, Bobbelina, Tango
Paco, Tilly, Tom,
Lasey's Descendants: Mole, Shadow, Opal, Alex
Not purebred
Breeding: (2/4 Irish Setter) (2/4 Pug)
Tilly's details
Tom's details
Original dog
Choose another dog
Quit


Dog information
Dog information
Name: Tilly
Breed: Irish Setter
Kennel: Novar kennel club
Address: 30 argyle street-Glasgow-G38YJ
Tilly's owner: Joe Keogh (owns 10 dogs)
Pedigree
Tilly's siblings: [Sake, Sake]
Tilly's Ancestors: Night, Mabula, Molly, Oswald
Dash, Sake, Bobbelina, Tango
Paco,
Tilly's Descendants: Link, Mole, Shadow, Opal
Alex
Lasey
Snake
Sage
Fatso
Purebred
Breeding: (4/4 Irish Setter)
Sake's details
Paco's details
Original dog
Choose another dog
Quit