```
/**
 * --------------------------------------------------------------------------
 * File name: Example1.java
 * Project name: Example1
 * --------------------------------------------------------------------------
 * Creator's name and email: Donald Sanderson and Kellie Price
 * Course-Section:  CSCI2020
 * Creation Date: Apr 06, 2008
 * Date of Last Modification: Nov 27, 2011
 * --------------------------------------------------------------------------
 */

import java.sql.DriverManager;   // DriverManager for establishing linkage to DB
import java.sql.SQLException;    // SQLException handles exception for DB classes
import java.sql.Connection;      // Connection establishes a logon to the DB
import java.sql.Statement;       // Holds and executes the sql statements
import java.sql.ResultSet;       // Holds the results of a query

public class Example1
{
   /* This is an example class that demonstrates the use of jdbc to connect to
    * and retrieve information from a database.
    */
   public static void main(String[] args)
   {

    /* 1 - Declare necessary variables */
       String  myConnStr;     // holds the connection string
       String  myUser;        // holds the user name
       String  myPasswd;      // holds the user password
       String  myQuery;       // holds sql query to be run
       String  myUpdate;      // holds sql update statement

       String  lastName;      // holds the student to query
       int studentId;         // holds the student to update
       String  newMajor;      // the new student major
       int numRows;           // the number of rows that were updated

       int resultId;          //holds the id from the results of the query
       String resultFamily;   //holds the family name from the results of the query
       String resultMajor;    //holds the major form the results of the query

       /*Setting up values for the student to change and the new value
        * for their major.  Since these are variables they could have come
        * in via parameters in a method call, or from user input.
        */
        lastName = "Crusher";
        newMajor = "CSIT";
        studentId = 14193418;

    /* 2 - Start the try block */
      /* Since all of these calls to the jdbc classes may return run-time errors
       * they need to be encapsulated in a try block with an associated error
```

```
       * handler.  Luckily all of the errors generated are subtypes of the
       * generic type SQLExcpetion so one catch block can take care of all
       * the errors.
       */
      try
      {
    /* 3 - Link the class to the jdbc driver */

        DriverManager.registerDriver(new oracle.jdbc.OracleDriver());

    /* 4 - Establish a connection with the db server */
        /* Now that the class can communicate with the db server
        * we need to start a connection, think of it like logging in
        * to the server.  We need to specify which server, and who we are.
        * The 3 strings hold the information and then we build a new connection
        * object, and ask the DriverManager to establish the connection using
        * this information.
        */

        myConnStr = "jdbc:oracle:thin:@pythia.etsu.edu:1521:csdb";
        myUser = "myuserid";
        myPasswd = "mypassword";
        Connection myCon = null; //Build the connection object
        myCon = DriverManager.getConnection(myConnStr, myUser, myPasswd); // establish a
connection via the jdbc driver


    /* 5 - Declare and Create the statement objects */
         /*A statement object holds sql to be executed it needs to be allocated
         * and then liked to the connection as a statement to be executed
         * in this connection
         */

        Statement myQueryStmt = null;           // build the statement object
        myQueryStmt = myCon.createStatement(); // and associate it with myCon

        Statement myUpdateStmt = null;          // build the statement object
        myUpdateStmt = myCon.createStatement();// and associate it with myCon



    /* 6 - Declare the result set object */

        ResultSet myResults = null;    // build the result set


    /* 7 - Build and execute the query */
        /* To avoid lines being too long, long SQL statements can be built
         * up by concatenating pieces together.  Also note the first use of
         * a variable to hold a value for a where clause.  Since it is a
         * number it needs no delimeters around it.  Had it been a string we
         * would have had to code in the open and close quotes around it.
         * Another good debugging tool is to have the sql statement displayed
```

```
         * to you, so if there is an error you can quickly see it.
         */

        myQuery = "SELECT STUDENT.ID, STUDENT.FAMILY_NAME, STUDENT.MAJOR" ;
        myQuery += " FROM STUDENT" ;
        myQuery += " WHERE STUDENT.FAMILY_NAME = " + "'" + lastName + "'";
        System.out.println("Query to run: " + myQuery); // the sql to run
        // tell the jdbc connection to run the query
        // and put the results in my_results
        myResults = myQueryStmt.executeQuery(myQuery);


/* 8 - Process the result set */
        /* Now that the query has been run we need the results!  So set up a
         * loop that will keep pulling in the results of the query one row at
         *  a time so that we can process them
         */

        System.out.println("The results are:");
        while (myResults.next())  //ResultSet.next gets the next record,
                                  //it returns false if no more records exist.
        {
           /* We can pull the fields out of the result set by placing them in
            * a compatible variable.
            */
           resultId = myResults.getInt("ID");
           resultFamily = myResults.getString("FAMILY_NAME");
           resultMajor = myResults.getString("MAJOR");
           //Now use all these pieces to build up a nice output.
           System.out.println
             ("Student:  "+ resultId + " " + resultFamily + " " + resultMajor);
        }  //end while



      /* Build and execute an update to change a student's major */

        myCon.setAutoCommit(false);   //Tell the system not to autoupdate

        /* Build up the update statement, note the need to put quotes around
         * what will be a string literal in the update.
         */
        myUpdate = "update STUDENT set STUDENT.MAJOR = " + "'" + newMajor + "'";
        myUpdate += " where STUDENT.ID = " + studentId;
        System.out.println("Update to run: " + myUpdate);

        /*
         * Run the update, and get back the number of rows the update affected
         */
        numRows = myUpdateStmt.executeUpdate(myUpdate);

        /* You have the same commit and rollback control as you did command
         * line by default when you close your connection all changes are
```

```
         * committed. You could use the calls below to commit or rollback your
         * changes
         */

        //  myCon.rollback();
        //  myCon.commit();

        /*Rerun the same query a second time to check for the update */
        ResultSet myResults2 = null;
        myResults2 = myQueryStmt.executeQuery(myQuery);

        /* Now that the query has been re-run we need the results!
        */
        System.out.println("The results after updating " + numRows +  " rows are:");
        while (myResults2.next())  //ResultSet.next gets the next record,
                                   //it returns false if no more records exist.
        {
           resultId =  myResults2.getInt("ID");
           resultFamily = myResults2.getString("FAMILY_NAME");
           resultMajor = myResults2.getString("MAJOR");
           System.out.println
           ("Student:  "+ resultId + " " + resultFamily + " " + resultMajor);
        }  //end while


      /* rollback to undo changes since this is just for demonstration purposes */
        myCon.rollback();


   /* 9 - Close the result set(s), statement(s), connection */

        /* Now close the things we opened to avoid any side effects. While
         * many systems will do this for you at the end of execution,  some do
         * not.  Always err on the side of caution and explicitly close your
         * result sets, statements and connections.  It takes just a few lines
         *  of code and can save you from some rather bizarre errors.
         */
        myResults2.close();      // close the first result set
        myResults.close();       // close the second result set
        myQueryStmt.close();     // close the query statement
        myUpdateStmt.close();    // close the update statement
        myCon.close();           // close the connection
     }  // end try
     /* 10 - End the try block */

     /* 11 - Begin the catch block */

     catch (SQLException mySqlError)
     {
        /* Simplistic error handling just catch the error,
        *  and dump the error stack
        */
        System.out.println("An Error has occured, dumping error stack");
```

```
         mySqlError.printStackTrace();
      }  //end catch
      /* 12 - End the catch block */

   } // end main
} // end class Example1
```