

```

/**
 * Class Name    : Address.java
 * Version       : Build 01.10
 * Date of Change : 2000 MAR 28
 * Author        : Don Baldwin
 * Copyright     : 2000 ASR Strategic Resources
 * Description   : This is the abstract class from which country specific
 *                 classes inherit common attribute and method signatures.
 *
 * Version History:
 *
 * Version       Date           Author
 * Build 01.10   2000 MAR 28   Don Baldwin
 *                                     Added comments.
 * Build 01.00   2000 MAR 11   Ulf Hedlund
 *                                     Started the class.
 *
 * Known Problems: None
 */

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import java.util.Vector;
import java.util.Enumeration;

public abstract class Address implements IAddress
{
    private    long    longId;           // The relation identifier of an Address object.
                                                // The relational identifier is the primary key
                                                // of the relational table used to store Address
                                                // data objects.

    private    String  strFirstName;    // This string is the first name of the person who
                                                // is entered into the address book.

    private    String  strLastName;     // This is the person's last name.
    private    String  strCompanyName;  // This is the person's company.

    /**
     * Method that retrieves all names from the
     * database and returns them as a vector of
     * strings.
     */
    public static Enumeration getAllTableNames()
    throws SQLException
    {
        int          intColumns;
        Vector       vecColumns;

        ResultSetMetaData sqlTableData;
        ResultSet        sqlResult;
        Connection       sqlConnection;
        Statement        sqlStatement;

        DBInfo          objDatabaseInfo;

        // Set up the connection to the database, perform the sql query.
        objDatabaseInfo    = DBInfo.getDBInfo();
        sqlConnection      = objDatabaseInfo.initConnection();
    }
}

```

```

sqlStatement      = sqlConnection.createStatement();
sqlResult         = sqlStatement.executeQuery("SELECT * FROM ADDRESS");

// Get information about the database
sqlTableData     = sqlResult.getMetaData();

intColumns       = sqlTableData.getColumnCount();
vecColumns       = new Vector();

// Loop through the columns, add them to a vector
for(int i = 1; i <= intColumns; i++)
{
    String         strFirstChar;
    String         strRest;
    String         strTableName;

    // Capitalize all table names before adding them to the vector.
    strTableName  = sqlTableData.getColumnName(i);
    strTableName  = strTableName.toLowerCase();
    strFirstChar  = "" + strTableName.charAt(0);
    strFirstChar  = strFirstChar.toUpperCase();
    strRest       = strTableName.substring(1);

    strTableName  = strFirstChar + strRest;

    vecColumns.addElement(strTableName);
}

// Close all database related connections
sqlResult.close();
sqlStatement.close();
sqlConnection.close();

return vecColumns.elements();
}

/**
 * Deletes the address with the corresponding id
 */
public static void deleteAddress(long aLongId)
throws SQLException
{
    Connection     sqlConnection;
    PreparedStatement sqlStatement;

    DBInfo         objDatabaseInfo;

    objDatabaseInfo = DBInfo.getDBInfo();
    sqlConnection  = objDatabaseInfo.initConnection();

    sqlStatement    = sqlConnection.prepareStatement("DELETE FROM ADDRESS WHERE ID = "+aLongId);

    sqlStatement.executeUpdate();

    sqlStatement.close();
    sqlConnection.close();
}

/**
 * Updates the address related to the object passed in the
 * database. This version is sloppy, since it will delete
 * the old and create a new, instead of just updating all
 * information in the already existing database row.
 */
public static void updateAddress(Address anObjAddress)

```

```

throws SQLException
{
    long longId;

    longId = anObjAddress.getId();

    deleteAddress(longId);
    anObjAddress.writeAddress();
}

/**
 * Retrieves all database rows in the database, as a vector
 * of address objects.
 */
public static Vector getAllAddresses()
    throws SQLException
    {
        Vector        vecAddresses;
        String        strSqlQuery;

        strSqlQuery    = new String("SELECT * FROM ADDRESS");
        vecAddresses = performSearch(strSqlQuery);

        return vecAddresses;
    }

/**
 * Searches for all addresses that has a value that the passed
 * value is a substring of, in the passed column. Returns all
 * matches as a vector of address objects.
 */
public static Vector searchAddresses(String aStrColumn, String aStrValue)
    throws SQLException
    {
        Vector        vecAddresses;
        String        strSqlQuery;

        aStrColumn    = aStrColumn.toUpperCase();

        strSqlQuery    = new String("SELECT * FROM ADDRESS WHERE "+
            aStrColumn+" LIKE '%" +aStrValue+"%'");

        vecAddresses = performSearch(strSqlQuery);

        return vecAddresses;
    }

/**
 * Used to execute a given sql query string and return all
 * addresses in the resultset as a vector of address objects.
 * Used in searchAddresses and getAllAddresses right now.
 */
private static Vector performSearch(String aStrSqlQuery)
    throws SQLException
    {
        Connection    sqlConnection;
        Statement      sqlStatement;
        ResultSet      sqlResult;
        DBInfo         objDatabaseInfo;
        Vector         vecAddresses;

        objDatabaseInfo = DBInfo.getDBInfo();
        sqlConnection = objDatabaseInfo.initConnection();
        vecAddresses = new Vector();
    }

```

```

sqlStatement = sqlConnection.createStatement();
sqlResult    = sqlStatement.executeQuery(aStrSqlQuery);

// For all matches, create an address object, insert the
// correct fields into it and add the finished object into
// the array.
while(sqlResult.next())
{
    Address objAddress;

    long longTempId;
    String strTempFirstName;
    String strTempLastName;
    String strTempCompanyName;
    String strTempCountryCode;
    String strTempCountry;
    String strTempPostalCode;
    String strTempPostalBlock;
    String strTempStreet;
    String strTempState;
    String strTempCity;

    longTempId      = sqlResult.getLong("ID");
    strTempFirstName = sqlResult.getString("FIRSTNAME");
    strTempLastName  = sqlResult.getString("LASTNAME");
    strTempCompanyName = sqlResult.getString("COMPANYNAME");
    strTempCountryCode = sqlResult.getString("COUNTRYCODE");
    strTempCountry    = sqlResult.getString("COUNTRY");
    strTempPostalCode = sqlResult.getString("POSTALCODE");
    strTempPostalBlock = sqlResult.getString("POSTALBLOCK");
    strTempStreet     = sqlResult.getString("STREET");
    strTempState      = sqlResult.getString("STATE");
    strTempCity       = sqlResult.getString("CITY");

    // Check for an USA address and deal with information
    // specific for these objects.
    if(strTempCountryCode.equals("USA"))
    {
        UnitedStatesAddress objTempAddress;
        Vector              vecStreet;
        int                 intIndex;

        intIndex = strTempStreet.indexOf('\n');

        vecStreet = new Vector();

        do
        {
            String strStreetPartial;

            if(intIndex > -1)
            {
                strStreetPartial = strTempStreet.substring(0,intIndex);
                strTempStreet    = strTempStreet.substring(intIndex+1);
            }
            else
            {
                strStreetPartial = strTempStreet;
            }

            intIndex = strTempStreet.indexOf('\n');

            vecStreet.addElement(strStreetPartial);
        }
        while(intIndex > -1);
    }
}

```

```

        objTempAddress = new UnitedStatesAddress();

        objTempAddress.setState(strTempState);
        objTempAddress.setStreet(vecStreet);
        objTempAddress.setCity(strTempCity);

        objAddress = objTempAddress;
    }
    // Else it must be a swedish address, deal with it
    // correspondingly
    else
    {
        SwedishAddress objTempAddress;

        objTempAddress = new SwedishAddress();

        objTempAddress.setCountry(strTempCountry);
        objTempAddress.setCountryCode(strTempCountryCode);
        objTempAddress.setStreet(strTempStreet);
        objTempAddress.setCity(strTempCity);

        objAddress = objTempAddress;
    }

    // Common data for all address objects
    objAddress.setId(longTempId);
    objAddress.setFirstName(strTempFirstName);
    objAddress.setLastName(strTempLastName);
    objAddress.setCompanyName(strTempCompanyName);
    objAddress.setPostalRegionCode(strTempPostalCode);
    objAddress.setPostalBlockCode(strTempPostalBlock);

    vecAddresses.addElement(objAddress);
}

// Close all database object connections.
sqlResult.close();
sqlStatement.close();
sqlConnection.close();

return vecAddresses;
}

/**
 * Used to set the relational identifier of an Address object.
 * The relational identifier is the primary key of the RDBMS table that
 * is used to store Address data objects.
 * This method is a set once method. If the current value of the identifier
 * is a value greater than zero, then the value of the identifier is not changed.
 * If the anID argument is zero or negative, an Illrgal Argument Exception will
 * be thrown.
 */
public void setId(long aLongId)
    throws IllegalArgumentException
{
    if(longId != 0)
    {
        return;
    }

    if(aLongId < 1)
    {
        throw new IllegalArgumentException("The Id value must be 1 or greater");
    }
}

```

```

    }
else
    {
        longId = aLongId;
    }

}

/**
 * Query method that returns an Address object's relational identifier.
 */
public long getId()
{
    return longId;
}

/**
 * Mutator method used to set the reference variable that points to a String
 * object used to store the Address object's first name.
 * If the argument to this function is either null or an empty string,
 * then an IllegalArgumentException is thrown.
 */
public void setFirstName(String aStrFirstName)
    throws IllegalArgumentException
{
    if(aStrFirstName == null ||
        aStrFirstName.equals(""))
    {
        throw new IllegalArgumentException("The first name cannot be empty");
    }
    else
    {
        strFirstName = aStrFirstName;
    }
}

/**
 * Query method that returns an Address object's first name.
 */
public String getFirstName()
{
    return strFirstName;
}

/**
 * Mutator method used to set the reference variable that points to a String
 * object used to store the person's last name.
 * If the argument to this function is either a null or empty string, then an
 * IllegalArgumentException is thrown.
 */
public void setLastName(String aStrLastName)
    throws IllegalArgumentException
{
    if(aStrLastName == null ||
        aStrLastName.equals(""))
    {
        throw new IllegalArgumentException("The last name cannot be empty");
    }
    else
    {
        strLastName = aStrLastName;
    }
}

```

```

    }

/**
 * Query method that returns an Address object's last name.
 */
public String getLastName()
{
    return strLastName;
}

/**
 * Mutator method used to set the reference variable that points to a String
 * object used to store the company name.
 */
public void setCompanyName(String aStrCompanyName)
    throws IllegalArgumentException
    {
        if(aStrCompanyName == null || aStrCompanyName.equals(""))
            {
                throw new IllegalArgumentException("The company name cannot be empty");
            }
        else
            {
                strCompanyName = aStrCompanyName;
            }
    }

/**
 * Query method that returns an Address object's company name.
 */
public String getCompanyName()
    {
        return strCompanyName;
    }

/**
 * Query method that returns a full person name in first name, space,
 * and last name order.
 * For example: Don Baldwin (Where Don is first name and Baldwin is last name).
 */
public String getFirstLastName()
    {
        String strFName;
        String strLName;
        String strResult;

        strFName = getFirstName();
        strLName = getLastName();

        strResult = strFName + " " + strLName;

        return strResult;
    }

/**
 * Query method that returns a full person name in first name, space,
 * and last name order.
 * For example: Baldwin, Don (Where Don is first name and Baldwin is last name).
 */
public String getLastFirstName()
    {
        String strFName;

```

```
String strLName;
String strResult;

strFName = getFirstName();
strLName = getLastName();

strResult = strLName + ", " + strFName;

return strResult;
}

/**
 * Mutator method that is used to set the postal code for an address object.
 * If the single argument is null or an empty String then an
 * IllegalArgumentException is thrown.
 */
public abstract void setPostalCode(String aStrPostalCode) throws IllegalArgumentException;

/**
 * Query method that returns the country code for an Address object.
 */
public abstract String getCountryCode();

} // END Address.java
```