

```

/**
 * Class Name           : UnitedStatesAddress.java
 * Version              : Build 01.10
 * Date of Change      : 2000 MAR 28
 * Author               : Don Baldwin
 * Copyright            : 2000 ASR Strategic Resources
 * Description          : The address objects
 *                      : specific for US addresses
 *
 * Version History :
 *
 * Version      Date      Author
 * Build 01.10  2000 MAR 28  Don Baldwin
 *                      Added additional comments.
 * Build 01.00  2000 MAR 10  Anders Nyström
 *                      Started the class
 *
 * Known Problems: None
 */

import java.util.Vector;
import java.util.Hashtable;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class UnitedStatesAddress extends Address
{
    private Vector  vecStreet;

    private String  strCity;
    private String  strState;
    private String  strPostalRegionCode;
    private String  strPostalBlockCode;

    /**
     * Simple constructor, just assigning a new
     * empty vector to the street 'list'.
     */
    public UnitedStatesAddress()
    {
        vecStreet  = new Vector();
    }

    /**
     * Method that is responsible for appending a street "line" to the street vector.
     * This mutator method is used to append an additional String to the streets address
     * Vector.
     */
    public void appendStreet(String aStrStreet)
    {
        vecStreet.addElement(aStrStreet);
    }

    /**
     * Method that is responsible for setting the street.
     * Mutator method used to set the streets reference variable to a Vector of Strings
     * which contains the United States Street Addresses.
     */
    public void setStreet(Vector aVecStreets) throws IllegalArgumentException
    {
        if(aVecStreets == null || aVecStreets.isEmpty())

```

```

        {
            throw new IllegalArgumentException("The street can not be empty");
        }
    else
    {
        vecStreet = aVecStreets;
    }
}

/**
 * Method that returns the street for the current object.
 * Query method that returns an Address's Street Address. If the Address
 * is a multi-line address, the address is returned as String in which each
 * individual street line is concatenated and separated by a line feed
 * character.
 * For example: "AITP" and "NCC Contest" would be "AITP/nAITP Contest"
 */
public String getStreet()
{
    int intVecIndex;
    String strStreet = "";

    if(vecStreet == null || vecStreet.isEmpty())
    {
        return "No address available";
    }
    else
    {
        intVecIndex = vecStreet.size();

        for(int i=0;i<intVecIndex;i++)
        {
            strStreet = strStreet.concat((String)vecStreet.get(i));
            strStreet = strStreet.concat("\n");
        }
    }
    return strStreet;
}

/**
 * Method that clears all the streets (elements) in the street object.
 * Mutator method used to remove the current street address. The street
 * vector is emptied of all String elements.
 */
public void removeAllStreets()
{
    vecStreet.removeAllElements();
}

/**
 * Method that sets the city name.
 * Mutator method that is used to set the reference variable that points
 * to the String that contains the city name.
 */
public void setCity(String aStrCity) throws IllegalArgumentException
{
    if( (aStrCity.length()==0) || (aStrCity == null) )
    {
        throw new IllegalArgumentException("The city can not be empty");
    }
    else
    {
        strCity = aStrCity;
    }
}

```

```

    }

/**
 * Method that returns the city variable.
 * Query method that returns the String object that stores the city address.
 */
public String getCity()
{
    return strCity;
}

/**
 * Method that sets the state for the object.
 * Mutator method that is used to set the city address for an Address object.
 * If the single argument is null or is an empty string, then an
 * IllegalArgumentException is thrown.
 */
public void setState(String aStrState) throws IllegalArgumentException
{
    if( (aStrState.length()==0) || (aStrState == null) )
    {
        throw new IllegalArgumentException("The state can not be empty");
    }
    else
    {
        strState = aStrState;
    }
}

/**
 * Method that returns the current state of the object.
 * Query method that returns a reference variable to the string that contains
 * the city address.
 */
public String getState()
{
    return strState;
}

/**
 * Method that sets the postal code for the object.
 * Mutator method that is used to set the postal code for an address object.
 * If the single arguemnt is null or is an empty string, then an
 * IllegalArgumentException is thrown.
 */
public void setPostalCode(String aStrPostalCode)
    throws IllegalArgumentException
{
    String    strTempRegion;
    String    strTempBlock;
    int       intIndex;
    int       intRegion;
    int       intBlock;
    int       intLength;

    if(aStrPostalCode == null || aStrPostalCode.length() == 0)
    {
        throw new IllegalArgumentException("The postal code can not be empty");
    }

    intLength = aStrPostalCode.length();

    if(intLength == 5)
    {
        setPostalRegionCode(aStrPostalCode);
    }
}

```

```

    }
else if(intLength == 10)
{
    intIndex    = aStrPostalCode.indexOf('-');

    if(intIndex < 0)
    {
        throw new IllegalArgumentException("The postal code format is invalid");
    }

    strTempRegion  = aStrPostalCode.substring(0,intIndex);
    strTempBlock  = aStrPostalCode.substring(intIndex+1);

    setPostalRegionCode(strTempRegion);
    setPostalBlockCode(strTempBlock);
}
else
{
    throw new IllegalArgumentException("The postal code format is invalid");
}
}

/**
 * Method that returns the postal block code.
 */
public String getPostalBlockCode()
{
    return strPostalBlockCode;
}

/**
 * Method that simply returns entire the postal code.
 */
public String getPostalCode()
{
    String          strPostalRegionCode;
    String          strPostalBlockCode;
    String          strResult;

    strPostalRegionCode    = getPostalRegionCode();
    strPostalBlockCode    = getPostalBlockCode();

    if(strPostalBlockCode != null && !strPostalBlockCode.equals(""))
    {
        strResult = strPostalRegionCode + "-" + strPostalBlockCode;
    }
else
    {
        strResult = strPostalRegionCode;
    }

    return strResult;
}

/**
 * Method that sets the postal region code according
 * to the US standards.
 */
public void setPostalRegionCode(String aStrPostalRegionCode)
throws IllegalArgumentException
{
    try
    {
        Integer.parseInt(aStrPostalRegionCode);
    }
}

```

```

    }
    catch(IllegalArgumentException exclA)
    {
        throw new IllegalArgumentException("The postal code format is invalid");
    }

    if(aStrPostalRegionCode == null || aStrPostalRegionCode.length() != 5)
    {
        throw new IllegalArgumentException("The postal code format is invalid");
    }
    else
    {
        strPostalRegionCode = aStrPostalRegionCode;
    }

}

/**
 * Method that returns the postal region code.
 */
public String getPostalRegionCode()
{
    return strPostalRegionCode;
}

/**
 * Method that sets the postal block code according
 * to the US standards (if a block code is passed).
 */
public void setPostalBlockCode(String aStrPostalBlockCode)
    throws IllegalArgumentException
{
    if(aStrPostalBlockCode == null || aStrPostalBlockCode.length() != 4)
    {
        return;
    }
    else
    {
        // Make sure it's an integer.
        try
        {
            Integer.parseInt(aStrPostalBlockCode);
        }
        catch(IllegalArgumentException exclA)
        {
            throw new IllegalArgumentException("The postal code format is invalid");
        }

        strPostalBlockCode = aStrPostalBlockCode;
    }

}

/**
 * Method that simply returns the string "USA" country code
 */
public String getCountryCode()
{
    return "USA";
}

/**
 * Method that simply returns the string "USA" country name

```

```

*/
public String getCountry()
{
    return "USA";
}

/**
 * Method that returns a string for the mailing label
 */
public String getMailingLabel()
{

    /**
     * Using a stringBuffer to prevent multiple objects to be cloned.
     */
    StringBuffer    bfrMailingLabel;
    String          strResult;

    bfrMailingLabel = new StringBuffer();
    strResult      = null;

    bfrMailingLabel.append(getFirstLastName());
    bfrMailingLabel.append("\n");
    bfrMailingLabel.append(getCompanyName());
    bfrMailingLabel.append("\n");
    bfrMailingLabel.append(getStreet());
    bfrMailingLabel.append(getCity());
    bfrMailingLabel.append(", ");
    bfrMailingLabel.append(getState());
    bfrMailingLabel.append(", ");
    bfrMailingLabel.append(getPostalCode());
    bfrMailingLabel.append("\n");
    bfrMailingLabel.append(getCountry());

    strResult = bfrMailingLabel.toString();

    return strResult;
}

/**
 * Creates a new united states address in the database
 * with the data retrieved from the address object passed
 * to the method.
 */
public void writeAddress()
    throws SQLException
    {
        String          strTempFirstName;
        String          strTempLastName;
        String          strTempCompanyName;
        String          strTempStreet;
        String          strTempCity;
        String          strTempState;
        String          strTempPostalCode;
        String          strTempPostalBlock;
        String          strTempCountryCode;
        String          strTempCountry;

        Connection      sqlConnection;
        PreparedStatement sqlStatement;
        DBInfo          objDatabaseInfo;

        // Retrieve info from the address object
        strTempFirstName = getFirstName();
        strTempLastName  = getLastName();
    }

```

```

strTempCompanyName = getCompanyName();
strTempStreet      = getStreet();
strTempCity        = getCity();
strTempState       = getState();
strTempPostalCode  = getPostalRegionCode();
strTempPostalBlock = getPostalBlockCode();
strTempCountryCode = getCountryCode();
strTempCountry     = getCountry();

// Necessary to trap a possible database error
if(strTempPostalBlock == null)
{
    strTempPostalBlock = "";
}

// Setup the database connection
objDatabaseInfo = DBInfo.getDBInfo();
sqlConnection   = DBInfo.initConnection();

// Prepare the statement, setup all field information in it.
sqlStatement =
sqlConnection.prepareStatement("INSERT INTO ADDRESS (FIRSTNAME, LASTNAME, "+
    "COMPANYNAME, STREET, CITY, STATE, POSTALCODE, POSTALBLOCK, COUNTRYCODE, COUNTRY)" +
    " VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)");

sqlStatement.setString(1,strTempFirstName);
sqlStatement.setString(2,strTempLastName);
sqlStatement.setString(3,strTempCompanyName);
sqlStatement.setString(4,strTempStreet);
sqlStatement.setString(5,strTempCity);
sqlStatement.setString(6,strTempState);
sqlStatement.setString(7,strTempPostalCode);
sqlStatement.setString(8,strTempPostalBlock);
sqlStatement.setString(9,strTempCountryCode);
sqlStatement.setString(10,strTempCountry);

// Execute the sql statement.
sqlStatement.executeUpdate();

// Close all database object connections, prevents crashes.
sqlStatement.close();
sqlConnection.close();
}

/**
 * Method that returns a String representation of the object's attributes.
 */
public String toString()
{
    Hashtable hshToString;
    String strHshInformation;

    hshToString = null;
    strHshInformation = null;

    hshToString.put("Id",new Long(getId()));
    hshToString.put("First Name",getFirstName());
    hshToString.put("Last Name",getLastName());
    hshToString.put("Company",getCompanyName());
    hshToString.put("Street",getStreet());
    hshToString.put("City",getCity());
    hshToString.put("State",getState());
    hshToString.put("PostalCode",getPostalCode());
    hshToString.put("CountryCode",getCountryCode());
    hshToString.put("Country",getCountry());
}

```

```
hshToString.put("MailingAddress",getMailingLabel());  
  
strHshInformation = hshToString.toString();  
return strHshInformation;  
}  
  
} // END UnitedStatesAddress.java
```