

```

/**
 * Class Name      : SwedishAddress.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 Swedish
 *                  addresses.
 *
 *                  AITP NCC Address SOLUTION
 *
 * 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.util.Hashtable;

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

public class SwedishAddress extends Address
{
    private String strStreet;           // European street name and number
    private String strCity;
    private String strSECountryCode;   // This is the ISO country code for Sweden (SE)
    private String strSECountry;      // This is the country name (Sweden)

    private String strPostalRegionCode; // This is the Swdish region code
    private String strPostalBlockCode; // This is the Swedish block code

    /**
     * Mutator method used to set the European street name and number. For simplicity we
     * treat this as a single string. If the method argument is either null or an empty
     * string the method throws an IllegalArgumentException.
     */
    public void setStreet(String aStrStreet)
        throws IllegalArgumentException
    {
        if(aStrStreet == null ||
            aStrStreet.equals(""))
        {
            throw new IllegalArgumentException("The street name cannot be null or empty");
        }
        else
        {
            strStreet = aStrStreet;
        }
    }

    /**
     * Query method that returns the Street and Sreet Number of the Swedish address

```

```

* and street number.
*/
public String getStreet()
{
    return strStreet;
}

/**
 * Mutator method used to set the Swedish country code.
 * Method requires a non-empty String argument. If a null or empty String
 * argument is passed an IllegalArgumentException is thrown.
 */
public void setCountryCode(String aStrSECountryCode)
    throws IllegalArgumentException
    {
        if(aStrSECountryCode == null ||
            aStrSECountryCode.equals(""))
            {
                throw new IllegalArgumentException("The country code cannot be null or empty");
            }
        else
            {
                strSECountryCode = aStrSECountryCode;
            }
    }

/**
 * Query method that returns the country code for a Swedish Address object.
 */
public String getCountryCode()
    {
        return strSECountryCode;
    }

/**
 * Method to set the country name
 */
public void setCountry(String aStrSECountry)
    throws IllegalArgumentException
    {
        if(aStrSECountry == null ||
            aStrSECountry.equals(""))
            {
                throw new IllegalArgumentException("The country name cannot be null or empty");
            }
        else
            {
                strSECountry = aStrSECountry;
            }
    }

/**
 * Method to get the country name
 */
public String getCountry()
    {
        return strSECountry;
    }

/**
 * Method to set the city name

```

```

*/
public void setCity(String aStrCity)
    throws IllegalArgumentException
    {
    if(aStrCity == null ||
        aStrCity.equals(""))
        {
        throw new IllegalArgumentException("The city name cannot be null or empty");
        }
    strCity = aStrCity;
    }

/**
 * Method to get the city name
 */
public String getCity()
    {
    return strCity;
    }

/**
 * Method to set the postal region code
 */
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() != 3)
        {
        throw new IllegalArgumentException("The postal code format is invalid");
        }
    else
        {
        strPostalRegionCode = aStrPostalRegionCode;
        }

    }

/**
 * Method to get the postal region code
 */
public String getPostalRegionCode()
    {
    return strPostalRegionCode;
    }

/**
 * Method to set the postal block code
 */
public void setPostalBlockCode(String aStrPostalBlockCode)
    throws IllegalArgumentException
    {

    try
        {
        Integer.parseInt(aStrPostalBlockCode);

```

```

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

    if(aStrPostalBlockCode == null || aStrPostalBlockCode.length() != 2)
    {
        throw new IllegalArgumentException("The postal code format is invalid");
    }
    else
    {
        strPostalBlockCode = aStrPostalBlockCode;
    }

}

/**
 * Mutator method that returns the Address object's postal block code
 * as an integer primitive.
 */
public String getPostalBlockCode()
{
    return strPostalBlockCode;
}

/**
 * Mutator method to set the Swedish postal code.
 */
public void setPostalCode(String aStrPostalCode)
    throws IllegalArgumentException
{
    String    strRegionCode;
    String    strBlockCode;

    int    intRegionCode;
    int    intBlockCode;
    int    intCodeLength;

    char    chaMiddle;

    if(aStrPostalCode == null ||
        aStrPostalCode.equals(""))
    {
        throw new IllegalArgumentException("The postal code cannot be empty");
    }

    intCodeLength = aStrPostalCode.length();

    if(intCodeLength < 5 ||
        intCodeLength > 6)
    {
        throw new IllegalArgumentException("The postal code format is invalid");
    }

    chaMiddle = aStrPostalCode.charAt(3);

    if(intCodeLength == 6 &&
        chaMiddle != '-' &&
        chaMiddle != ')')
    {
        throw new IllegalArgumentException("The postal code format is invalid");
    }

    // First 3 characters

```

```

strRegionCode = aStrPostalCode.substring(0,3);

// Last 2 characters
strBlockCode = aStrPostalCode.substring(intCodeLength-2);

// Simply just call the "sub-methods" with the new information
setPostalRegionCode(strRegionCode);
setPostalBlockCode(strBlockCode);
}

/**
 * Return the postal code as "RegionCode BlockCode" (xxx xx)
 */
public String getPostalCode()
{
    String strRegionCode;
    String strBlockCode;

    strRegionCode = getPostalRegionCode();
    strBlockCode = getPostalBlockCode();

    return strRegionCode + " " + strBlockCode;
}

/**
 * Build the mailing label which will be displayed like this in the address book
 */
public String getMailingLabel()
{
    StringBuffer bfrLabel;
    String strResult;

    bfrLabel = new StringBuffer();

    bfrLabel.append(getCompanyName());
    bfrLabel.append("\n");
    bfrLabel.append(getFirstLastName());
    bfrLabel.append("\n");
    bfrLabel.append(getStreet());
    bfrLabel.append("\n");
    bfrLabel.append(getCountryCode());
    bfrLabel.append("-");
    bfrLabel.append(getPostalCode());
    bfrLabel.append(" ");
    bfrLabel.append(getCity().toUpperCase());
    bfrLabel.append("\n");
    bfrLabel.append(getCountry());

    strResult = bfrLabel.toString();
    return strResult;
}

/**
 * Creates a new european 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      strTempPostalRegion;
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();
strTempPostalRegion = getPostalRegionCode();
strTempPostalBlock = getPostalBlockCode();
strTempCountryCode = getCountryCode();
strTempCountry    = getCountry();

// 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,"");
sqlStatement.setString(7,strTempPostalRegion);
sqlStatement.setString(8,strTempPostalBlock);
sqlStatement.setString(9,strTempCountryCode);
sqlStatement.setString(10,strTempCountry);

sqlStatement.executeUpdate();

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

/**
 * toString() displays the class attributes in an ugly way,
 * for debugging purposes.
 */
public String toString()
{
    Hashtable hshAllData;
    String strResult;

    hshAllData = new Hashtable();

    hshAllData.put("id",new Long(getId()));
    hshAllData.put("first name",getFirstName());
    hshAllData.put("last name",getLastName());
    hshAllData.put("company name",getCompanyName());
    hshAllData.put("street",getStreet());

```

```
hshAllData.put("country code",getCountryCode());
hshAllData.put("country name",getCountry());
hshAllData.put("mailing label",getMailingLabel());

strResult = hshAllData.toString();

return strResult;
}

} // END SwedishAddress.java
```