Middleware RMI-IIOP

Interoperability met CORBA

Middleware specialisatiethema
Rob Juurlink IID7
2003 / 2004

Voorwoord

In deze middleware opdracht wordt gebruik gemaakt van de RMI-IIOP technologie. De opdracht is onderdeel van een groter middleware project waarbinnen meerdere middleware technologieën aan bod komen.

Voor het ontwerpen van de classediagrammen is er gebruik gemaakt van het pakket Poseiden van Gentleware. Het is een UML ontwikkeltool. Geciteerd van de Gentleware website:

> *Poseidon for UML is a professional UML CASE tool. With roots in the open source project ArgoUML it has evolved into a world-class modeling tool. Its superior* ***usability*** *makes it the easiest tool to learn and work with.*
> *With* ***UML standards compliancy****, it facilitates interoperability with other sets of tools.*

Voor de ontwikkeling van de software is gebruik gemaakt van versie Java versie 2 SDK 1.4.2. Deze versie van Sun Microsystems is op het moment de meest recente.

Om efficiënt Java-applicaties te ontwikkelen, is een IDE (Integrated Development Environment) gebruikt. De gebruikte ontwikkelomgeving is NetBeans versie 3.5.1.

Het besturingssysteem waarop het geheel draait is Debian GNU Linux met een KDE versie 3.2 desktop. Daarnaast is de ontwikkelde applicatie ook getest op een Windows XP ongeving met de JRE1.4.1 geïnstalleerd.

De ontwikkelingen van het Middleware project zijn ook te volgen via een website. Het adres van de website is: http://rob.juurlink.org.

Inhoudsopgave

INLEIDING

De opdracht is het ontwerpen van een electronisch warenhuis. Daarbij hoort een server-deel en een client-deel. Er wordt gebruik gemaakt van de middleware technologie RMI over IIOP. Er is geen database aanwezig. Deze wordt gesimuleerd dmv een properties bestand.

Het document begint met een uitleg over de gebruikte technologie, RMI en RMI-IIOP.

Als de lezer een beeld heeft van de RMI technologie, volgt de gedetaileerde opdrachtomschrijving. De opdracht wordt daarna uitgewerkt. Dit levert een aantal classes op. Een deel van die classes zijn voor de server applicatie, een deel voor de client applicatie en een deel van de classes wordt gebruikt door beide applicaties.

Na de het ontwerp volgt een beschrijving van de implementatie, het compileren en de werking.

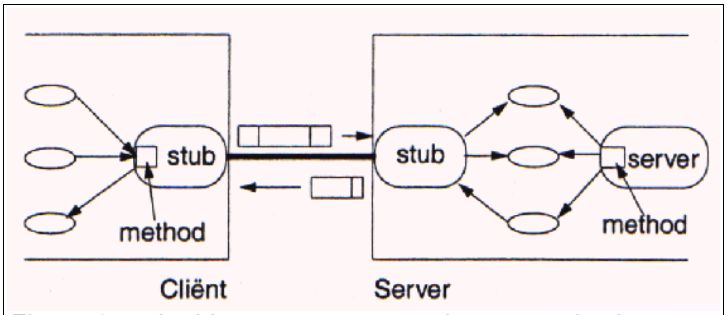
De Java broncode is als bijlage toegevoegd.

WEBWINKEL MBV MIDDLEWARE

RMI

Java RMI (Remote Method Invocation) is een methode waarbij de cliënt een remote object van de server kan benaderen. De methoden van dit object kunnen dus door een andere JVM (Java Virtuele Machine) worden aangeroepen. Dit kan bijvoorbeeld een opdracht zijn om gegevens van de server te krijgen of iets op de server op te slaan.

Om de communicatie tussen de server en de client werkend te krijgen wordt gebruik gemaakt van stub objecten. Zie figuur 1 hieronder.



Figuur 1, stub objecten zorgen voor de communicatie.

De Communicatie maakt gebruik van RMP (Remote Method Protocol). Als een remote object geen stub op de client heeft, wordt het gedownload door de classloader. De RMI Security Manager controleert de stub. De Security manager is alleen aan de client kant nodig.

Voordat de server objecten door de client gebruikt kunnen worden, moeten ze bekend zijn gemaakt aan de naming service. De objecten worden ge-identificeerd aan de hand van een unieke naam. De serverclasses erven de code voor de communicatie van RemoteServer.

RMI-IIOP

RMI-IIOP staat voor Java Remote Method Invocation over Internet Inter-ORB Protocol technology. RMI-IIOP maakt CORBA objecten benaderbaar mbv de RMI API. Geciteerd uit de J2SDK1.4 documentatie:

> *RMI-IIOP is for developers who program in the Java programming language and want to program to the RMI interfaces, but use IIOP as the underlying transport. RMI-IIOP provides interoperability with other CORBA objects implemented in various languages - but only if all the remote interfaces are originally defined as Java RMI interfaces.*

IIOP (Internet Inter-Orb Protocol) is een protocol voor de communicatie tussen CORBA ORBs. Net als CORBA is RMI-IIOP gebaseerd op open standaarden die zijn samengesteld door leden van de Object Management Group. Met behulp van IIOP kunnen applicaties geschreven in andere programmeertalen toch communiceren met componenten geschreven voor het Java platform.

De opdrachtomschrijving

Ontwerp een remote interface volgens welke een cliënt applicatie kan communiceren met de [S]Aktiehuis server. De server biedt de prijslijst aan in de vorm van een Properties object. Een Properties object is een soort Hashtable, waarin zowel key als value String objecten zijn. Het warenhuis heeft nog geen administratie voor de inventaris en stuurt daarom bij een bestelling alleen de totale prijs van de bestelling terug en onderneemt verder geen actie.

De cliënt zal bij opstarten de prijslijst ontvangen en middels een simpele grafische user interface bestellingen kunnen doen.

Bij de implementatie dient gebruik gemaakt te worden van RMI-IIOP (RMI Internet Inter Orb Protocol).

De implementatie dient getest te worden, zowel met cliënt en server op één machine, als met cliënt en server op verschillende machines.

Requirements

De meeste requirements zijn al verwerkt in de opdrachtomschrijving, deze worden overgenomen en aangevuld.

Functionele requirements

- De server biedt de prijslijst aan in de vorm van een Properties object. Een Properties object is een soort Hashtable, waarin zowel key als value String objecten zijn.
- Het warenhuis heeft nog geen administratie voor de inventaris en stuurt daarom bij een bestelling alleen de totale prijs van de bestelling terug en onderneemt verder geen actie.
- De cliënt zal bij opstarten de prijslijst ontvangen en middels een simpele grafische user interface bestellingen kunnen doen.

Technische requirements

- Bij de implementatie dient gebruik gemaakt te worden van RMI-IIOP (RMI Internet Inter Orb Protocol).
- De implementatie dient getest te worden, zowel met cliënt en server op één machine, als met cliënt en server op verschillende machines.

Het ontwerp

De gebruikte technologie stelt een aantal eisen aan het ontwerp. Een remote interface declareert de methoden die de client straks remote mag aanroepen. Een remote interface moet voldoen aan de volgende woorwaarden:

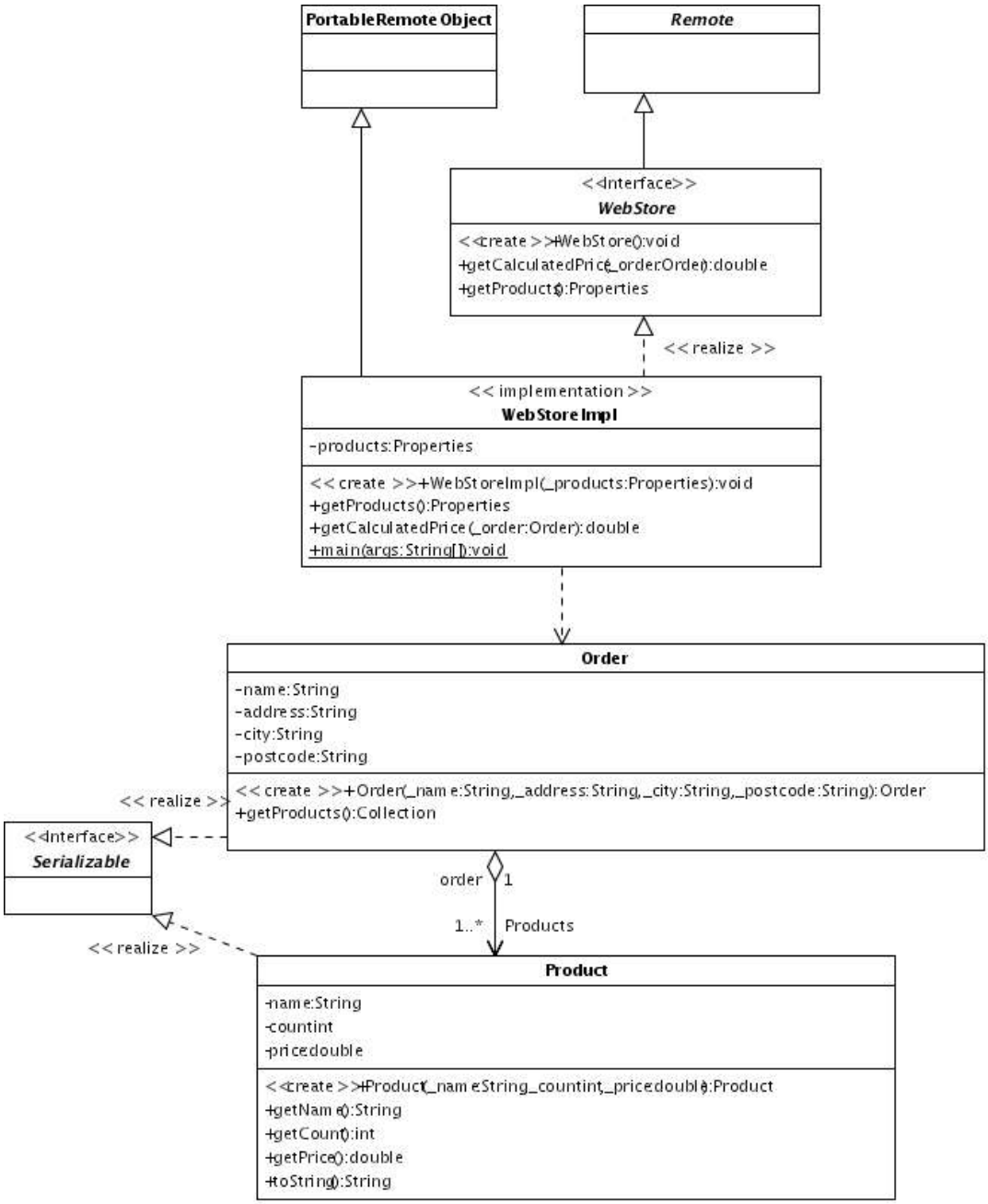
- Een remote interface is public
- Een remote interface extends de interface java.rmi.Remote
- De contructor en elke methode moeten een RemoteException kunnen kunnen gooien (throw java.rmi.RemoteException)

De server moet na het starten minimaal één remote object registreren bij een naming service. Via dit object kunnen straks wel andere remote objecten worden benaderd; bootstrappen (daar is echt geen nederlands woord voor).

Het ontwerp in UML levert een aantal classediagrammen op. De onderlinge relaties tussen de verschillende classes worden zo goed duidelijk.

Het server-gedeelte

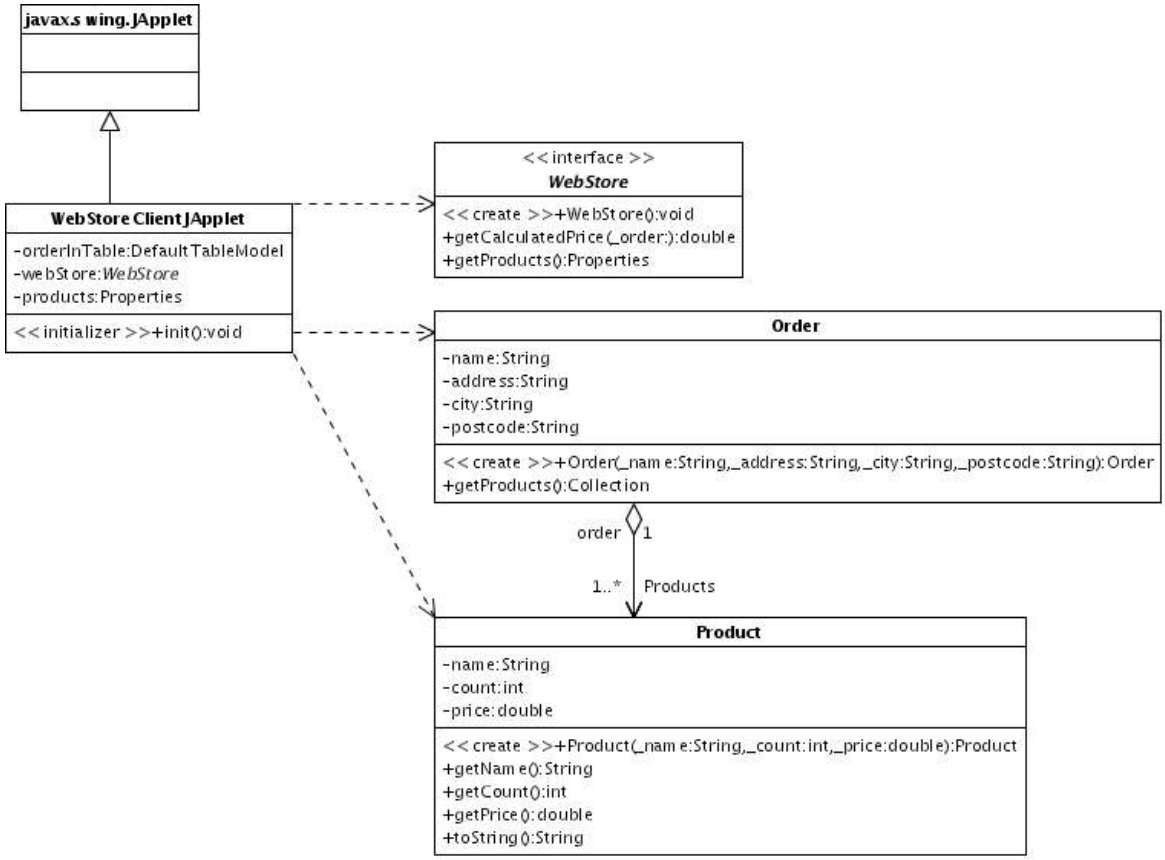
In figuur 2 is de WebStore server afgebeeld.



Figuur 2. Het classediagram van het serverdeel.

Het client-gedeelte

In figuur 3, de clientsoftware is geimplementeerd als een Java Applet. (preciezer, een Swing Japplet).



Figuur 3. Het classediagram van de client die aan de kant van de klant wordt opgestart.

De implementatie

Het server-gedeelte

Het WebStore server object heeft twee remote methoden. Eén om alle producten op te vragen, deze methode levert een properties object op. En een methode om de order te laten uitrekenen. Deze methode heeft al argument een object van het type Order.

Een order bevat de NAW gegevens en een collectie met objecten van het type product. Deze objecten zijn voor te stellen als regels op een order, ze bevatten een productnaam, aantal en een prijs per stuk.

Als de server wordt gestart, leest deze het properties bestand dat de producten en hun prijs bevat. Het WebStore object wordt geregistreerd bij de orb daemon. De server wacht op een verbinding.

Het client-gedeelte

De client is geimplementeerd als een JApplet, maar kan door een eenvoudige aanpassing ook als losstaande client worden gestart.

Als de client wordt gestart, vraagt een systeem methode vanaf welke host de client is gestart. De client verwacht op die machine een orb daemon te vinden die een WebStore object heeft geregistreerd.

Als de cleint het remote object heeft gevonden, vraagt het alle producten op. Deze worden aangeleverd in een Properties object.

De gebruiker stelt een order samen (De GUI is intuitief genoeg en heeft geen verdere uitleg nodig hoe een order samen te stellen). Na een druk op “Verzenden” wordt de order verzonden door de aanroep van een remote methode. Het totale bedrag van de bestelling komt terug.

Starten server en client

Voor het compileren hebben we gebruikt gemaakt van de Java 2 SDK versie 1.4.2. Hieronder staan de stappen voor het compileren van de java classes en het staren van respectievelijk de server en de client onder een Linux besturingssysteem.

Compileren

Compileer alle java broncode. Daarna wordt de client stub en de server tie gegenereerd.

- javac *.java
- rmic -iiop WebStoreImpl

Starten server

Het is belangrijk dat de Tie class aanwezig is bij de server.

Na het starten van de serverpplicatie, wordt het remote object geregistreerd bij een naming server. Dit is de orb daemon.

- orbd -ORBInitialPort 1050& //Start de orb daemon
- java WebStoreImpl& //Starten in de achtergrond

Starten client

Een applet kan alleen verbinding maken met de host waarvan het is gestart. Dit houdt in dat als de server en client niet dezelfde machines zijn, de applet geladen moet worden vanaf een webserver waarop ook de WebStore server draait.

Het is belangrijk dat de Stub class aanwezig is bij de client. Is deze daar niet aanwezig, dan levert dit een classCastException op.

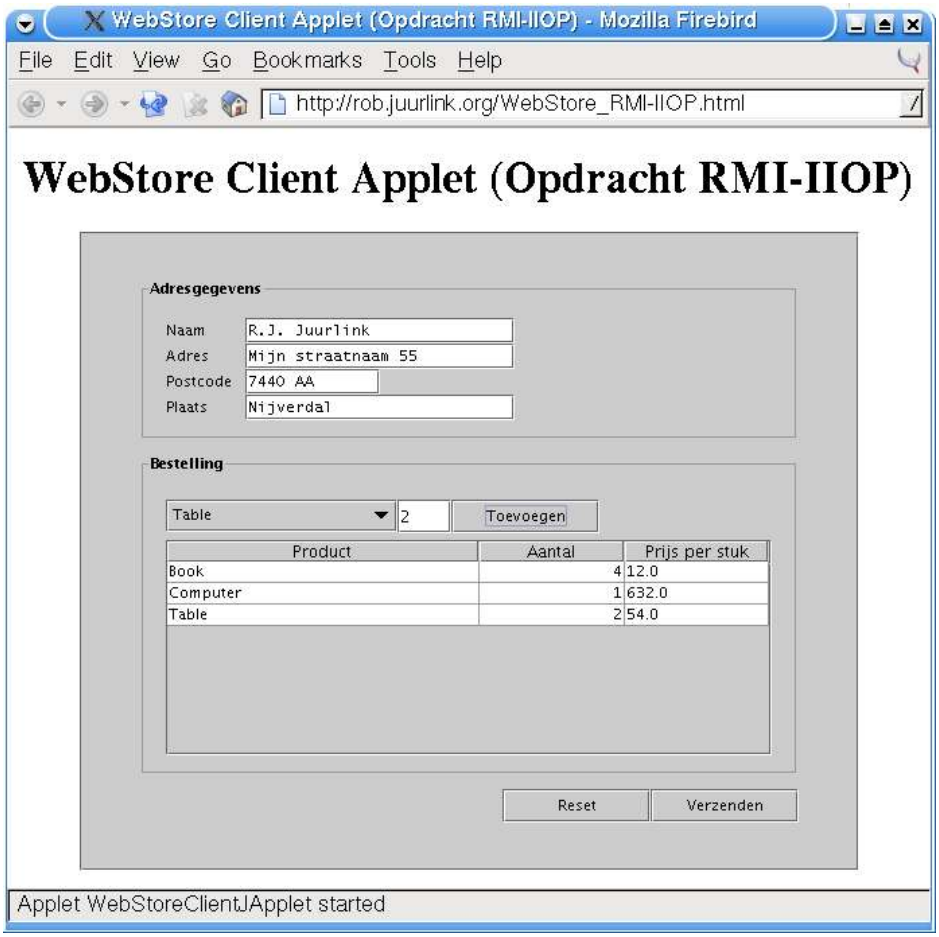
Draaien de server en de client op dezelfde machine, dan kan het eenvoudiger. De class kan niet rechtstreeks worde gestart, dit moet via een HTML bestand waarin de applet staat gedefinieerd in <applet> of <object> tags:

- appletviewer WebStoreClientJApplet.html

SCHERMAFDRUK

Zie figuur 4 voor een schermafruk van de applicatie.

Bovenin het venster staan velden voor de NAW-gegevens. Daaronder kan het te bestellen product geselecteerd worden uit een “drop-down” lijst. Met de knop “Toevoegen” wordt het geselecteerde artikel toegevoegd. De knop “Verzenden” laat de order remote uitrekenen, zie figuur 5.



Figuur 4. Een schermafdruk van de WebStore Applet die draait in de Mozilla Firebird browser.

Figuur 5. De verzonden order wordt remote berekend.

Conclusie

De applicatie werkt erg vlot. Een gebruiker merkt niet dat de data van een remote server komt. De applicate is natuurlijk niet representatief voor een compleet transactiesysteem, omdat er nauwelijk data wordt verstuurd, maar het doet wel waar het voor bedoeld is, laten zien dat het oversturen van objecten mbv IIOP (het CORBA protocol) mogelijk is.

Het binnenhalen van de Java applet gaat erg snel. Het is dan ook maar zo'n 13Kb ingepakt in een JAR archief. Het starten van de applet in de browser daarentegen neemt iets meer tijd in beslag, maar dit is erg afhankelijk van de snelheid en de configuratie van de gebruikte machine.

Het is belangrijk dat de juiste classes zich op de server en client bevinden. De client bijvoorbeeld gebruikt alleen de stub class, terwijl de server gebruikt maakt van de tie class. Beide classes zijn gegenereerd uit de WebStoreImpl class.

Referenties

[1] **Sun Microsystems**, *Java RMI over IIOP Technology Documentation Home Page*, september 2003, http://java.sun.com/j2se/1.4.2/docs/guide/rmi-iiop/

[2] **Sun Microsystems**, Getting Started Using Java RMI over IIOP, september 2003, http://java.sun.com/j2se/1.4.2/docs/guide/rmi-iiop/tutorial.html

[3] **J.Meuleman**, Dictaat Gedistribueerd Programmeren (1693), 2001

[4] **Cay s. Horstmann – Gary Cornell**, *Core Java volume II – Advanced Features*, Sun Microsystem Press, Palo Alto California, 2000.

BIJLAGEN

BRONCODE

WEBSTORECLIENTJAPPLET.HTML

```
<HTML>
<HEAD>
   <TITLE>Applet HTML Page</TITLE>
</HEAD>
<BODY>

<H3><HR WIDTH="100%">Applet HTML Page<HR WIDTH="100%"></H3>

<P>
<APPLET code="WebStoreClientJApplet.class" width=350 height=200></APPLET>
</P>

<HR WIDTH="100%"><FONT SIZE=-1><I>Generated by NetBeans IDE</I></FONT>
</BODY>
</HTML>
```

WEBSTORE.JAVA

```
import java.io.*;
import java.util.*;
import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 *
 */
public interface WebStore extends Remote {

  ////////////////////////////////////////
  // operations

/**
 * Uitrekenen wat de order kost ...
 *
 * @return double De totale prijs
 * @param _order De Order welke de producten met hun waarde bevat
 */
    public double getCalculatedPrice(Order _order) throws Exception;


/**
 * Levert een properties object met producten op.
 *
 * @return Properties een collectie met producten
 */
    public Properties getProducts() throws Exception;

} // end WebStore
```

WebStoreImpl.java

```
 1:
 2:import javax.rmi.PortableRemoteObject;
 3:import java.rmi.RemoteException;
 4:import javax.naming.InitialContext;
 5:import javax.naming.Context;
 6:import java.util.*;
 7:import java.io.*;
 8:
 9:
10:/**
11: *
12: */
13:public class WebStoreImpl extends PortableRemoteObject implements WebStore {
14:
15:  ////////////////////////////////////
16:  // attributes
17:
18:
19:/**
20: * De lijst met producten en hun prijs ...
21: */
22:    private Properties products;
23:
24:  ////////////////////////////////////
25:  // operations
26:
27:
28:/**
29: * Constructor. the remote object instance will need to be "exported".
30: * Exporting a remote object makes it available to accept incoming
31: * remote method requests, by listening for incoming calls to the
32: * remote object on an anonymous port. The class will be exported
33: * automatically upon creation.
34: *
35: * @param _products De lijst met producten
36: */
37:    public WebStoreImpl(Properties _products) throws RemoteException {
38:        super();
39:        products = _products;
40:
41:    } // end WebStoreImpl
42:
43:/**
44: * Levert een properties object met producten op.
45: *
46: * @return Properties een collectie met producten
47: */
48:    public Properties getProducts() throws RemoteException {
49:
50:        return products;
51:
52:    } // end getProducts
53:
54:/**
55: * Uitrekenen wat de order kost ...
56: *
57: * @return double De totale prijs
58: * @param _order De Order welke de producten met hun waarde bevat
59: */
60:    public double getCalculatedPrice(Order _order) {
61:
62:        double totalPrice = 0;
63:
64:        //Loop de lijst met producten uit de order langs en
65:        //reken uit wat de totale prijs is.
66:        if (_order != null) {
```

```
67:
68:             Collection products = _order.getProducts();
69:             Iterator i = products.iterator();
70:             while (i.hasNext()) {
71:
72:                 Product product = (Product) i.next();
73:                 totalPrice += product.getCount() * product.getPrice();
74:             }
75:         }
76:
77:         return totalPrice;
78:     } // end getCalculatedPrice
79:
80:/**
81: * Creeer het object dat als remote object beschikbaar gesteld
82: * wordt en registreer het bij een naming service.
83: */
84:     public static void main(String[] args) {
85:
86:         try {
87:
88:             //Het propertiesobject inladen
89:             Properties products = new Properties();
90:             try {
91:                 FileInputStream input = new FileInputStream("products.data");
92:                 products.load(input);
93:                 input.close();
94:             }
95:             catch (IOException e) {
96:                 System.out.println(e);
97:                 System.exit(0);
98:             }
99:
100:            // The server needs to create an instance of the remote object
101:            // implementation, or Servant.
102:            WebStoreImpl webStoreImplRef = new WebStoreImpl(products);
103:
104:            // Systeem properties instellingen
105:            Properties p = new Properties(System.getProperties());
106:            p.setProperty("java.naming.factory.initial","com.sun.jndi.cosnaming.CNCtxFactory");
107:            p.setProperty("java.naming.provider.url","iiop://localhost:1050");
108:            System.setProperties(p);
109:
110:            // The following code binds the name "WebStore" to a reference
111:            // for the remote object:
112:            Context initialNamingContext = new InitialContext();
113:            initialNamingContext.rebind("WebStore", webStoreImplRef );
114:
115:            System.out.println("WebStore Server: Ready...");
116:
117:        } catch (Exception e) {
118:            System.out.println("Trouble: " + e);
119:            e.printStackTrace();
120:        }
121:
122:    } // end main
123:
124: } // end WebStoreImpl
```

PRODUCT.JAVA

```
1:import java.io.*;
2:import java.util.*;
3:
4:/**
5: *
6: */
7:public class Product implements Serializable {
8:
9:  ////////////////////////////////////
10:  // attributes
11:
12:    private String name;
13:    private int count;
14:    private double price;
15:
16:
17:  ////////////////////////////////////
18:  // operations
19:
20:
21:/**
22: * Constructor
23: *
24: * @param _name
25: * @param _count
26: * @param _price
27: */
28:    public  Product(String _name, int _count, double _price) {
29:
30:        name  = _name;
31:        count = _count;
32:        price = _price;
33:
34:    } // end Product
35:
36:
37:    public String getName() {
38:        return name;
39:    } // end getName
40:
41:    public int getCount() {
42:        return count;
43:    } // end getCount
44:
45:    public double getPrice() {
46:        return price;
47:    } // end getPrice
48:
49:
50:} // end Product
```

ORDER.JAVA

```
  1:import java.io.*;
  2:import java.util.*;
  3:
  4:/**
  5: * Een order bevat een collectie met producten.
  6: * Elk product heeft een naam, aantal en prijs (per stuk).
  7: */
  8:public class Order implements Serializable {
  9:
 10:  ////////////////////////////////////
 11:  // attributes
 12:
 13:    private String name;
 14:    private String address;
 15:    private String city;
 16:    private String postcode;
 17:    private Collection products; // of type Product
 18:
 19:
 20:  ////////////////////////////////////
 21:  // operations
 22:
 23:/**
 24: * Constructor. Creeer een nieuwe order ...
 25: *
 26: *
 27: * @param _name
 28: * @param _address
 29: * @param _city
 30: * @param _postcode
 31: * @param _products
 32: * @return
 33: */
 34:    public  Order(String _name, String _address, String _city, String _postcode, Collection
_products) {
 35:
 36:        name     = _name;
 37:        address  = _address;
 38:        city     = _city;
 39:        postcode = _postcode;
 40:        products = _products;
 41:
 42:    } // end Order
 43:
 44:/**
 45: * Does ...
 46: *
 47: *
 48: * @return
 49: */
 50:    public Collection getProducts() {
 51:
 52:        return products;
 53:
 54:    } // end getProducts
 55:
 56: } // end Order
```

WebStoreClientJApplet.java

```
  1:/*
  2: * WebStoreClientJApplet.java
  3: *
  4: * Created on September 20, 2003, 1:31 AM
  5: */
  6:import java.util.Properties;
  7:import java.util.Hashtable;
  8:import java.util.Enumeration;
  9:import javax.swing.*;
 10:import javax.swing.event.*;
 11:
 12:import java.rmi.RemoteException;
 13:import java.net.MalformedURLException;
 14:import java.rmi.NotBoundException;
 15:import javax.rmi.*;
 16:import java.util.*;
 17:import java.lang.*;
 18:import java.text.*;
 19:import javax.naming.NamingException;
 20:import javax.naming.InitialContext;
 21:import javax.naming.Context;
 22:import javax.swing.table.*;
 23:/**
 24: *
 25: * @author  rob
 26: */
 27:public class WebStoreClientJApplet extends javax.swing.JApplet {
 28:
 29:    private DefaultTableModel orderInTable;
 30:    private WebStore          webStore;
 31:    private Properties        products = new Properties();
 32:
 33:
 34:    /** Initializes the applet WebStoreClientJApplet */
 35:    public void init() {
 36:
 37:        Context    ic;
 38:        Object     objref;
 39:        Hashtable  enviroment;
 40:
 41:        // Look&Feel instellen. De look van het systeem pakken
 42:        try {
 43:            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
 44:        } catch (Exception e) { }
 45:
 46:        // Systeem variabelen zetten voor de verbinding
 47:        enviroment = new Hashtable();
 48:        enviroment.put("java.naming.factory.initial","com.sun.jndi.cosnaming.CNCtxFactory");
 49:        enviroment.put("java.naming.provider.url","iiop://" + getCodeBase().getHost() +
":1050");
 50:        // -Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory
 51:        // -Djava.naming.provider.url=iiop://localhost:1050
 52:
 53:        // GUI initialiseren
 54:        initComponents();
 55:
 56:
 57:        // Probeer verbing te maken met de ORB
 58:        try {
 59:            ic = new InitialContext(enviroment);
 60:
 61:            // Get the Object reference from the Name Service using JNDI call.
 62:            objref = ic.lookup("WebStore");
 63:
 64:            // Narrow the object reference to the concrete type and invoke the method.
 65:            webStore = (WebStore) PortableRemoteObject.narrow(objref, WebStore.class);
```

```
 66:             products = webStore.getProducts();
 67:
 68:         } catch( Exception e ) {
 69:                 JOptionPane.showMessageDialog(this,
 70:                 "Kan geen verbinding met de server opzetten",
 71:                 "Netwerkfout", JOptionPane.ERROR_MESSAGE);
 72:             System.err.println( "Exception " + e + " Caught" );
 73:             e.printStackTrace( );
 74:             return;
 75:         }
 76:
 77:         // De combobox vullen met de producten
 78:         cmbProducts.addItem("< kies product >");
 79:         if (products != null) {
 80:             Enumeration enum = products.propertyNames();
 81:             while (enum.hasMoreElements()) {
 82:                 cmbProducts.addItem((String)enum.nextElement());
 83:             }
 84:         }
 85:
 86:         // aan het model van de tabel worden straks de producten toegevoegd
 87:         orderInTable = (DefaultTableModel) tblOrder.getModel();
 88:     }
 89:
 90:     /** This method is called from within the init() method to
 91:      * initialize the form.
 92:      * WARNING: Do NOT modify this code. The content of this method is
 93:      * always regenerated by the Form Editor.
 94:      */
 95:     private void initComponents() {//GEN-BEGIN:initComponents
 96:         java.awt.GridBagConstraints gridBagConstraints;
 97:
 98:         jScrollPaneWebStore = new javax.swing.JScrollPane();
 99:         panelWebStoreClient = new javax.swing.JPanel();
100:         panelAddress = new javax.swing.JPanel();
101:         panelAddress2 = new javax.swing.JPanel();
102:         txtName = new javax.swing.JTextField();
103:         txtAddress = new javax.swing.JTextField();
104:         txtPostcode = new javax.swing.JTextField();
105:         txtCity = new javax.swing.JTextField();
106:         lblName = new javax.swing.JLabel();
107:         lblAddress = new javax.swing.JLabel();
108:         lblPostcode = new javax.swing.JLabel();
109:         lblCity = new javax.swing.JLabel();
110:         panelBestelling = new javax.swing.JPanel();
111:         panelBestelling2 = new javax.swing.JPanel();
112:         ScrollPaneOrder = new javax.swing.JScrollPane();
113:         tblOrder = new javax.swing.JTable();
114:         panelAddButtons = new javax.swing.JPanel();
115:         cmbProducts = new javax.swing.JComboBox();
116:         txtQuantity = new javax.swing.JTextField();
117:         btnAdd = new javax.swing.JButton();
118:         panelKnoppen = new javax.swing.JPanel();
119:         btnReset = new javax.swing.JButton();
120:         btnSend = new javax.swing.JButton();
121:
122:         setName("PanelWebStoreClient");
123:         panelWebStoreClient.setLayout(new java.awt.GridBagLayout());
124:
125:         panelAddress.setLayout(new java.awt.FlowLayout(java.awt.FlowLayout.LEFT, 15, 10));
126:
127:         panelAddress.setBorder(new javax.swing.border.TitledBorder(null, "Adresgegevens",
javax.swing.border.TitledBorder.DEFAULT_JUSTIFICATION,
javax.swing.border.TitledBorder.DEFAULT_POSITION, new java.awt.Font("Dialog", 1, 11)));
128:         panelAddress2.setLayout(new java.awt.GridBagLayout());
129:
130:         txtName.setFont(new java.awt.Font("DialogInput", 0, 11));
131:         txtName.setMinimumSize(new java.awt.Dimension(160, 19));
132:         txtName.setPreferredSize(new java.awt.Dimension(200, 19));
133:         gridBagConstraints = new java.awt.GridBagConstraints();
134:         gridBagConstraints.gridx = 1;
```

```
135:         gridBagConstraints.gridy = 0;
136:         gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
137:         panelAddress2.add(txtName, gridBagConstraints);
138:
139:         txtAddress.setFont(new java.awt.Font("DialogInput", 0, 11));
140:         txtAddress.setPreferredSize(new java.awt.Dimension(200, 19));
141:         gridBagConstraints = new java.awt.GridBagConstraints();
142:         gridBagConstraints.gridx = 1;
143:         gridBagConstraints.gridy = 1;
144:         gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
145:         panelAddress2.add(txtAddress, gridBagConstraints);
146:
147:         txtPostcode.setFont(new java.awt.Font("DialogInput", 0, 11));
148:         txtPostcode.setPreferredSize(new java.awt.Dimension(100, 19));
149:         gridBagConstraints = new java.awt.GridBagConstraints();
150:         gridBagConstraints.gridx = 1;
151:         gridBagConstraints.gridy = 2;
152:         gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
153:         panelAddress2.add(txtPostcode, gridBagConstraints);
154:
155:         txtCity.setFont(new java.awt.Font("DialogInput", 0, 11));
156:         txtCity.setPreferredSize(new java.awt.Dimension(200, 19));
157:         gridBagConstraints = new java.awt.GridBagConstraints();
158:         gridBagConstraints.gridx = 1;
159:         gridBagConstraints.gridy = 3;
160:         gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
161:         panelAddress2.add(txtCity, gridBagConstraints);
162:
163:         lblName.setFont(new java.awt.Font("Dialog", 0, 11));
164:         lblName.setText("Naam");
165:         lblName.setToolTipText("De volledige naam");
166:         gridBagConstraints = new java.awt.GridBagConstraints();
167:         gridBagConstraints.gridx = 0;
168:         gridBagConstraints.gridy = 0;
169:         gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
170:         gridBagConstraints.insets = new java.awt.Insets(0, 0, 0, 10);
171:         panelAddress2.add(lblName, gridBagConstraints);
172:
173:         lblAddress.setFont(new java.awt.Font("Dialog", 0, 11));
174:         lblAddress.setText("Adres");
175:         lblAddress.setToolTipText("Straatnaam inclusief huisnummer en toevoeging");
176:         gridBagConstraints = new java.awt.GridBagConstraints();
177:         gridBagConstraints.gridx = 0;
178:         gridBagConstraints.gridy = 1;
179:         gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
180:         gridBagConstraints.insets = new java.awt.Insets(0, 0, 0, 10);
181:         panelAddress2.add(lblAddress, gridBagConstraints);
182:
183:         lblPostcode.setFont(new java.awt.Font("Dialog", 0, 11));
184:         lblPostcode.setText("Postcode");
185:         lblPostcode.setToolTipText("Postcode");
186:         gridBagConstraints = new java.awt.GridBagConstraints();
187:         gridBagConstraints.gridx = 0;
188:         gridBagConstraints.gridy = 2;
189:         gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
190:         gridBagConstraints.insets = new java.awt.Insets(0, 0, 0, 10);
191:         panelAddress2.add(lblPostcode, gridBagConstraints);
192:
193:         lblCity.setFont(new java.awt.Font("Dialog", 0, 11));
194:         lblCity.setText("Plaats");
195:         lblCity.setToolTipText("De aflever plaats");
196:         gridBagConstraints = new java.awt.GridBagConstraints();
197:         gridBagConstraints.gridx = 0;
198:         gridBagConstraints.gridy = 3;
199:         gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
200:         gridBagConstraints.insets = new java.awt.Insets(0, 0, 0, 10);
201:         panelAddress2.add(lblCity, gridBagConstraints);
202:
203:         panelAddress.add(panelAddress2);
204:
205:         gridBagConstraints = new java.awt.GridBagConstraints();
```

```
206:         gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
207:         gridBagConstraints.insets = new java.awt.Insets(5, 8, 5, 8);
208:         panelWebStoreClient.add(panelAddress, gridBagConstraints);
209:
210:         panelBestelling.setLayout(new java.awt.FlowLayout(java.awt.FlowLayout.LEFT, 15, 10));
211:
212:         panelBestelling.setBorder(new javax.swing.border.TitledBorder(null, "Bestelling",
javax.swing.border.TitledBorder.DEFAULT_JUSTIFICATION,
javax.swing.border.TitledBorder.DEFAULT_POSITION, new java.awt.Font("Dialog", 1, 11)));
213:         panelBestelling2.setLayout(new java.awt.GridBagLayout());
214:
215:         ScrollPaneOrder.setPreferredSize(new java.awt.Dimension(450, 160));
216:         tblOrder.setFont(new java.awt.Font("Dialog", 0, 11));
217:         tblOrder.setModel(new javax.swing.table.DefaultTableModel(
218:             new Object [][] {
219:
220:             },
221:             new String [] {
222:                 "Product", "Aantal", "Prijs per stuk"
223:             }
224:         ) {
225:             Class[] types = new Class [] {
226:                 java.lang.String.class, java.lang.Integer.class, java.lang.String.class
227:             };
228:             boolean[] canEdit = new boolean [] {
229:                 false, false, false
230:             };
231:
232:             public Class getColumnClass(int columnIndex) {
233:                 return types [columnIndex];
234:             }
235:
236:             public boolean isCellEditable(int rowIndex, int columnIndex) {
237:                 return canEdit [columnIndex];
238:             }
239:         });
240:         tblOrder.getColumnModel().getColumn(0).setPreferredWidth(200);
241:         tblOrder.setToolTipText("Producten op bestellijst");
242:         tblOrder.setPreferredScrollableViewportSize(null);
243:         tblOrder.setTableHeader(tblOrder.getTableHeader());
244:         ScrollPaneOrder.setViewportView(tblOrder);
245:
246:         gridBagConstraints = new java.awt.GridBagConstraints();
247:         gridBagConstraints.gridx = 0;
248:         gridBagConstraints.gridy = 1;
249:         gridBagConstraints.gridwidth = java.awt.GridBagConstraints.REMAINDER;
250:         panelBestelling2.add(ScrollPaneOrder, gridBagConstraints);
251:
252:         panelAddButtons.setLayout(new java.awt.FlowLayout(java.awt.FlowLayout.LEFT, 0, 5));
253:
254:         cmbProducts.setFont(new java.awt.Font("Dialog", 0, 11));
255:         cmbProducts.setToolTipText("Selecteer een product");
256:         cmbProducts.setPreferredSize(new java.awt.Dimension(172, 24));
257:         panelAddButtons.add(cmbProducts);
258:
259:         txtQuantity.setToolTipText("Aantal te bestellen producten");
260:         txtQuantity.setMinimumSize(new java.awt.Dimension(40, 25));
261:         txtQuantity.setPreferredSize(new java.awt.Dimension(40, 25));
262:         panelAddButtons.add(txtQuantity);
263:
264:         btnAdd.setFont(new java.awt.Font("Dialog", 0, 11));
265:         btnAdd.setText("Toevoegen");
266:         btnAdd.setPreferredSize(new java.awt.Dimension(110, 25));
267:         btnAdd.addActionListener(new java.awt.event.ActionListener() {
268:             public void actionPerformed(java.awt.event.ActionEvent evt) {
269:                 btnAddActionPerformed(evt);
270:             }
271:         });
272:
273:         panelAddButtons.add(btnAdd);
274:
```

```
275:         gridBagConstraints = new java.awt.GridBagConstraints();
276:         gridBagConstraints.gridx = 0;
277:         gridBagConstraints.gridy = 0;
278:         gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
279:         panelBestelling2.add(panelAddButtons, gridBagConstraints);
280: 
281:         panelBestelling.add(panelBestelling2);
282: 
283:         gridBagConstraints = new java.awt.GridBagConstraints();
284:         gridBagConstraints.gridx = 0;
285:         gridBagConstraints.gridy = 1;
286:         gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
287:         gridBagConstraints.insets = new java.awt.Insets(5, 8, 5, 8);
288:         panelWebStoreClient.add(panelBestelling, gridBagConstraints);
289: 
290:         panelKnoppen.setLayout(new java.awt.GridLayout(1, 0));
291: 
292:         btnReset.setFont(new java.awt.Font("Dialog", 0, 11));
293:         btnReset.setText("Reset");
294:         btnReset.addActionListener(new java.awt.event.ActionListener() {
295:             public void actionPerformed(java.awt.event.ActionEvent evt) {
296:                 btnResetActionPerformed(evt);
297:             }
298:         });
299: 
300:         panelKnoppen.add(btnReset);
301: 
302:         btnSend.setFont(new java.awt.Font("Dialog", 0, 11));
303:         btnSend.setText("Verzenden");
304:         btnSend.setPreferredSize(new java.awt.Dimension(110, 25));
305:         btnSend.addActionListener(new java.awt.event.ActionListener() {
306:             public void actionPerformed(java.awt.event.ActionEvent evt) {
307:                 btnSendActionPerformed(evt);
308:             }
309:         });
310: 
311:         panelKnoppen.add(btnSend);
312: 
313:         gridBagConstraints = new java.awt.GridBagConstraints();
314:         gridBagConstraints.gridx = 0;
315:         gridBagConstraints.gridy = 2;
316:         gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
317:         gridBagConstraints.insets = new java.awt.Insets(5, 8, 5, 8);
318:         panelWebStoreClient.add(panelKnoppen, gridBagConstraints);
319: 
320:         jScrollPaneWebStore.setViewportView(panelWebStoreClient);
321: 
322:         getContentPane().add(jScrollPaneWebStore, java.awt.BorderLayout.CENTER);
323: 
324:     }//GEN-END:initComponents
325: 
326:     private void btnSendActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_btnSendActionPerformed
327:         // De order verzenden en het resultaat tonen
328:         double totalPrice = 0;
329: 
330:         // Melding als er geen producten in de lijst staan
331:         if (orderInTable.getRowCount() == 0) {
332:             JOptionPane.showMessageDialog(this,
333:                 "Er staan geen producten in de order.\nNiet verstuurd!",
334:                 "Order leeg", JOptionPane.ERROR_MESSAGE);
335:             return;
336:         }
337: 
338:         // De producten in een collection proppen. Een productregel bevat een
339:         // naam, aantal en prijs.
340:         Collection theProducts = new HashSet();
341:         for (int i=0; i < orderInTable.getRowCount(); i++ ) {
342:             String productName  = (String) orderInTable.getValueAt(i, 0);
343:             int productCount    = ((Integer) orderInTable.getValueAt(i,1)).intValue();
344:             double productPrice = ((Double) orderInTable.getValueAt(i,2)).doubleValue();
```

```
345:            Product product = new Product(productName, productCount, productPrice);
346:            theProducts.add(product);
347:        }
348:
349:        // De order met de producten samenstellen
350:        Order order = new Order(txtName.getText(), txtAddress.getText(), txtCity.getText(),
txtPostcode.getText(), theProducts);
351:
352:        // Probeer order te versturen
353:        try {
354:            totalPrice = webStore.getCalculatedPrice(order);
355:            JOptionPane.showMessageDialog(this,
356:                "De order is verstuurd\nTotale prijs: " + totalPrice,
357:                "Order verstuurd", JOptionPane.INFORMATION_MESSAGE);
358:            // Scherm schoonvegen
359:            resetForm();
360:
361:        }
362:        // Versturen van de order mislukt
363:        catch (Exception e) {
364:            System.out.println("Kan order niet versturen, netwerkprobleem: " + e);
365:            JOptionPane.showMessageDialog(this,
366:                "Kan order niet versturen ivm netwerkprobleem",
367:                "Netwerkprobleem", JOptionPane.ERROR_MESSAGE);
368:        }
369:    }//GEN-LAST:event_btnSendActionPerformed
370:
371:    private void btnResetActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_btnResetActionPerformed
372:        resetForm();
373:    }//GEN-LAST:event_btnResetActionPerformed
374:
375:    private void btnAddActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_btnAddActionPerformed
376:        // Er is op "toevoegen" gedrukt. Het item toevoegen aan de lijst
377:
378:        // Eerst controleren of het geselecteerde product en het aantal geldig zijn
379:        Integer quantity = null;
380:        try {
381:            quantity = new Integer(txtQuantity.getText());
382:        }
383:        catch (NumberFormatException e) {
384:            JOptionPane.showMessageDialog(this,
385:                "Het aantal te bestellen producten is niet gespecificeerd",
386:                "Fout bij toevoegen", JOptionPane.ERROR_MESSAGE);
387:            return;
388:        }
389:
390:        // Geen product geselecteerd
391:        if (cmbProducts.getSelectedIndex() <= 0) {
392:            JOptionPane.showMessageDialog(this,
393:                "Er is geen product om toe te voegen geselecteerd",
394:                "Fout bij toevoegen", JOptionPane.ERROR_MESSAGE);
395:            return;
396:        }
397:
398:        //Bestelling van dit product verwerken als het aantal geldig is
399:        if ((quantity != null) &&
400:            (cmbProducts.getSelectedIndex() > 0)) {
401:            String productName  = (String) cmbProducts.getSelectedItem();
402:            Double productPrice = new Double(products.getProperty(productName));
403:            orderInTable.addRow(new Object[] {productName, quantity, productPrice});
404:        }
405:
406:    }//GEN-LAST:event_btnAddActionPerformed
407:
408:    // Reset. De velden en de order leegmaken
409:    private void resetForm() {
410:        txtName.setText("");
411:        txtAddress.setText("");
412:        txtPostcode.setText("");
```

```
413:         txtCity.setText("");
414:         txtQuantity.setText("");
415: 
416:         while (orderInTable.getRowCount() > 0) {
417:             orderInTable.removeRow(0);
418:         }
419:         cmbProducts.setSelectedIndex(0);
420:     }
421: 
422: 
423: 
424:     // Variables declaration - do not modify//GEN-BEGIN:variables
425:     private javax.swing.JScrollPane ScrollPaneOrder;
426:     private javax.swing.JButton btnAdd;
427:     private javax.swing.JButton btnReset;
428:     private javax.swing.JButton btnSend;
429:     private javax.swing.JComboBox cmbProducts;
430:     private javax.swing.JScrollPane jScrollPaneWebStore;
431:     private javax.swing.JLabel lblAddress;
432:     private javax.swing.JLabel lblCity;
433:     private javax.swing.JLabel lblName;
434:     private javax.swing.JLabel lblPostcode;
435:     private javax.swing.JPanel panelAddButtons;
436:     private javax.swing.JPanel panelAddress;
437:     private javax.swing.JPanel panelAddress2;
438:     private javax.swing.JPanel panelBestelling;
439:     private javax.swing.JPanel panelBestelling2;
440:     private javax.swing.JPanel panelKnoppen;
441:     private javax.swing.JPanel panelWebStoreClient;
442:     private javax.swing.JTable tblOrder;
443:     private javax.swing.JTextField txtAddress;
444:     private javax.swing.JTextField txtCity;
445:     private javax.swing.JTextField txtName;
446:     private javax.swing.JTextField txtPostcode;
447:     private javax.swing.JTextField txtQuantity;
448:     // End of variables declaration//GEN-END:variables
```