KULAKOWSKI Fabienne
BROCCOLICCHI Jérôme

Projet LPMI

OFL- ML

Réalisation d'un éditeur permettant d'éditer graphiquement (UML étendu) une application conformément au modèle OFL

ANNEXES

Responsables : LAHIRE Philippe
CRESCENZO Pierre

Année 2001-2002

Table des matières

1. Classe AttributC

```
/**
* Class AttributC
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* TO SAVE AND LOAD ATTRIBUTES
*
*/
import java.util.Vector;

public class AttributC
{
	//attribut name
	private String aName;

	//attribut type
	private String aType;

	//attribut modifier
	private Vector attributModifierVector= new Vector();

	public AttributC(String name,String type,Vector modifier)
	{
		aName=name;
		aType=type;
		attributModifierVector=modifier;
	}

	/**
	* Return the vector which contains all the attributes
	*
	* @since JDK1.3
	*
	* @return attributModifierVector
	*
	*/
	public Vector getAttributModifierVector()
	{
		return attributModifierVector;
	}
}
```

2. Classe AttributeFrame

```
/**
* Class AttributeFrame
*
* Creates a frame which allows the user
* to enter the attribute's name,type and modifier(s).
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
* @since JDK1.3
*
* @Pre the attribute button in the description frame has been clicked (DescriptionFrame)
*
* @Post attributes' description can be created
*
*/

import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.BorderFactory;
import javax.swing.JFrame;
import javax.swing.JComboBox;
import javax.swing.JTextField;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTabbedPane;
import javax.swing.JScrollPane;
import javax.swing.JButton;
import java.util.Vector;

public class AttributeFrame extends JFrame
{


        private static Vector tmpModifierVector= new Vector();
        private Vector frameStringV = new Vector();
        private Vector frameAttCV = new Vector();

        private JComboBox typeList = new JComboBox (LoadObject.getTypeVector());
        private static JComboBox modifierJcomboBox;
        private JTextField attribName = new JTextField ();

        /**
        * Creates a frame with a tabbed panel which contains 2 others
        * panels : one for the description attributes (attribPane)
        * and one to enter the documentation about the attributes the user enters
        * (docPane).
        * This frame is created when you click on the button "attributes"
        * in the frame called "Description" (Class DescriptionFrame).
        *
        * The description attributes panel contains a scrollPane with a list in it.
        * (This list displays the attributes the user adds (<modifier>
        * attribName) by clicking the button "Add") and an other
        * pane to enter the attribute's name and modifier(s).
        *
        * The documentation panel only contains a text area
        *
        * @param attributDescription in order to initialize the frame display
```

```
    *
    * @see AttributsDescription
    * @see DescriptionFrame
    *
    * @since JDK1.3
    * @since JDK1.3
    *
    * @Pre the attribute button in the description frame has been clicked (DescriptionFrame)
    *
    * @Post attributes' description can be created
    *
    */
    public  AttributeFrame (final AttributsDescription attributDescription, String name)
    {
            frameStringV=(Vector)attributDescription.getStringVector().clone();
            frameAttCV=(Vector)attributDescription.getAttributCVector().clone();

            final JList attribList = new JList (frameStringV);



            modifierJcomboBox = new JComboBox(/*attributDescription.modifierVector*/);
            modifierJcomboBox.setPreferredSize(new Dimension(121,21));
            modifierJcomboBox.setEditable(false);
            modifierJcomboBox.setMaximumRowCount(5);
            typeList.setMaximumRowCount(5);

            JButton modifierButton = new JButton(" Choose ");

            JPanel attribPane = new JPanel ();
            JPanel docPane = new JPanel ();
            JPanel propertiesPane = new JPanel ();

            JLabel attribNameLabel = new JLabel ("Name :                    ");


            JLabel typeLabel;
            if(name.equals("attribut"))
                    typeLabel = new JLabel ("Attribute type :  ");
            else
                    typeLabel = new JLabel ("Parameter type :  ");

            JLabel modifLabel = new JLabel("Modifier :               ");

            JTabbedPane tabbedPane = new JTabbedPane();

            JScrollPane scroller = new JScrollPane(attribList);

            JButton deleteButton = new JButton(" Delete ");
            JButton addButton    = new JButton("  Add    ");
            JButton okButton     = new JButton("   Ok    ");
            JButton cancelButton = new JButton(" Cancel ");

            GridBagLayout gridbag = new GridBagLayout();
GridBagConstraints c = new GridBagConstraints();


            if(name.equals("attribut"))
                    attribName.setText ("AttributeName"); // the default text for the method's name
            else
                    attribName.setText ("ParameterName"); // the default text for the method's name


            attribList.setBackground (Color.white);


            // Set the backgound color and the size of the scroller
```

```
                scroller.setBackground (Color.white);
                scroller.setPreferredSize(new Dimension (300,200));

                propertiesPane.setPreferredSize(new Dimension (300,120));
                typeList.setPreferredSize(new Dimension (100,21));
                attribName.setPreferredSize(new Dimension (121,21));

                // Set the attribPane layout
        attribPane.setLayout (gridbag);

        c.anchor = GridBagConstraints.WEST;
                c.gridwidth = GridBagConstraints.RELATIVE;

                // Adds the scroller to the attribPane
                gridbag.setConstraints(scroller, c);
attribPane.add (scroller);

                // Adds the delete button to the attribPane
                c.insets = new Insets (0,30,0,0);
                c.gridwidth = GridBagConstraints.REMAINDER;
                gridbag.setConstraints(deleteButton, c);
attribPane.add (deleteButton);


                ////////////////////////////////////
                // Properties of the attribute     //
                //(name,type,modifier)             //
                ////////////////////////////////////
                c.insets = new Insets (10,-80,5,0);
                c.anchor = GridBagConstraints.WEST;
                c.gridwidth = GridBagConstraints.RELATIVE;

                propertiesPane.setLayout (gridbag);


                // Add "Name"
                gridbag.setConstraints(attribNameLabel, c);
                propertiesPane.add (attribNameLabel);

                // Add the texfield to enter the attribute's name
                c.gridwidth = GridBagConstraints.REMAINDER;
                c.insets = new Insets (10,-40,0,0);
                gridbag.setConstraints(attribName, c);
                propertiesPane.add (attribName);

                // Add "attribute type"
                c.gridwidth = GridBagConstraints.RELATIVE;
                c.insets = new Insets (10,-80,5,0);
                gridbag.setConstraints(typeLabel, c);
                propertiesPane.add (typeLabel);

                // Add a list to select the attribute's type
                c.gridwidth = GridBagConstraints.REMAINDER;
                c.insets = new Insets (10,0,5,0);
                gridbag.setConstraints(typeList, c);
                propertiesPane.add (typeList);


                // Add "Modifier"
                c.gridwidth = GridBagConstraints.RELATIVE;
                c.insets = new Insets (10,-80,10,0);
                gridbag.setConstraints(modifLabel, c);
                propertiesPane.add (modifLabel);

                // Add a list to select the attribute's modifier
                c.gridwidth = GridBagConstraints.RELATIVE;
                c.insets = new Insets (10,-40,10,10);
```

```
                gridbag.setConstraints(modifierJcomboBox, c);
                propertiesPane.add (modifierJcomboBox);


                c.gridwidth = GridBagConstraints.REMAINDER;
                c.insets = new Insets (10,0,10,10);
                gridbag.setConstraints(modifierButton, c);
                propertiesPane.add (modifierButton);


                // Add the propertiesPane to the attribPane
                c.gridwidth = GridBagConstraints.RELATIVE;
                gridbag.setConstraints(propertiesPane, c);
attribPane.add (propertiesPane);


    ////////////////////
    // THE BUTTONS  //
    ////////////////////
    // Add "Add" button to the attribPane
                c.insets = new Insets (0,30,0,0);
                c.gridwidth = GridBagConstraints.REMAINDER;
                gridbag.setConstraints(addButton, c);
attribPane.add (addButton);

                // Add "OK" button to the attribPane
    c.insets = new Insets (10,13,10,0);
    c.gridwidth = GridBagConstraints.RELATIVE;
                gridbag.setConstraints(okButton, c);
attribPane.add (okButton);

                // Add "Cancel" button to the attribPane
c.insets = new Insets (10,-105,10,10);
                c.gridwidth = GridBagConstraints.REMAINDER;
                gridbag.setConstraints(cancelButton, c);
attribPane.add (cancelButton);

                propertiesPane.setBorder(BorderFactory.createEtchedBorder());

                // Put methodPane and docPane in a tabbed pane
                if(name.equals("attribut"))
                        tabbedPane.addTab("Attributes", null, attribPane, null);
                else
                        tabbedPane.addTab("Parameters", null, attribPane, null);

tabbedPane.addTab("Documentation", null, docPane, null);
tabbedPane.setSelectedIndex(0);


getContentPane ().add (tabbedPane);
setSize (450,450);
setResizable (false);
setVisible (true);


                ////////////////////////
                // ACTION LISTENER  //
                ////////////////////////
                /**
                * to open the modifier frame
                *
                * @param e the action event
                *
                * @see ModifierFrame
                *
                * @since JDK1.3
                *
```

```
* @Post ModifierFrame is created
*
*/
modifierButton.addActionListener (new ActionListener ()
{
	public void actionPerformed (ActionEvent e)
	{
		ModifierFrame modif=new ModifierFrame(tmpModifierVector,"Attribut");
	}
});


// Add listener to the delete button
deleteButton.addActionListener (new ActionListener ()
{
	/**
	* To delete attributes from the description
	* The selected attributes are first removed from the attribList
	* and if the user clicks OK, the selected attributes are removed
	* from the attributes' vector in class AttributsDescription (else
	* nothing happens).
	*
	* @param e the action event
	*
	* @see AttributsDescription
	*
	* @since JDK1.3
	*
	* @Pre attributes are selected
	*
	* @Post selected attributes are removed
	*
	*/
	public void actionPerformed (ActionEvent e)
	{
		if(frameStringV.size()!=0)
		{
			Object [] selectedItems = attribList.getSelectedValues ();


			for (int i=0;i<selectedItems.length;i++)
			{
				frameStringV.removeElement(selectedItems[i]);
				attribList.updateUI();

				frameAttCV.removeElement(selectedItems[i]);
			}
		}
	}
});

// Add listener to the add button
addButton.addActionListener (new ActionListener ()
{
	/**
	* To add attributs to the description
	* The selected attributs are first added to the attribList
	* and if the user clicks OK, the selected attributs are added
	* to the attributs' vector in class AttributsDescription (else
	* nothing happens).
	*
	* @param e the action event
	*
	* @see AttributsDescription
	* @see AttributC
	*
	* @since JDK1.3
```

```
                        *
                        * @Post created attributes are added to the description
                        */
                        public void actionPerformed (ActionEvent e)
                        {
                                String desc = new String();

                                for (int i=0;i<tmpModifierVector.size();i++)
                                {
                                        desc =desc.concat((String)tmpModifierVector.elementAt(i));
                                        desc = desc.concat(" ");
                                }
                                desc = desc.concat((typeList.getSelectedItem()).toString());
                                desc = desc.concat(" ");
                                desc = desc.concat(attribName.getText());


                                frameStringV.addElement (desc);
                                attribList.updateUI();

                                AttributC attribut = new
AttributC(attribName.getText(),(typeList.getSelectedItem()).toString(),

        tmpModifierVector);

                                frameAttCV.addElement (attribut);


                                setJcomboBoxModifierText(new Vector());
                        }
                });

                // Add listener to the cancel button
                cancelButton.addActionListener (new ActionListener ()
                {
                        /**
                        * All the changes the user made
                        * are not treated. The attributes frame
                        * is just closed.
                        *
                        * @param e the action event
                        *
                        * @since JDK1.3
                        */
                        public void actionPerformed (ActionEvent e)
                        {
                                setVisible (false);


                        }
                });

                // Add listener to the ok button
                okButton.addActionListener (new ActionListener ()
                {
                        /**
                        * All the changes the user made
                        * are treated.
                        * The attributs are added or deleted.
                        * The attributs frame is closed.
                        *
                        * @param e the action event
                        *
                        * @see AttributsDescription
                        *
                        * @since JDK1.3
                        */
```

```
                        public void actionPerformed (ActionEvent e)
                        {
                                attributDescription.changeAttributC(frameAttCV,frameStringV);
                                setVisible (false);
                        }
                });
        }

        /**
        * Set the text in the modifier combo box by going through
        * the vector
        *
        * @param stringVector the Vector to cross
        *
        * @see ModifierFrame#okButton.addActionListener
        *
        * @since JDK1.3
        */
        public static void setJcomboBoxModifierText(Vector stringVector)
        {
                modifierJcomboBox.removeAllItems();

                for (int i=0;i<stringVector.size();i++)
                        modifierJcomboBox.addItem((String)stringVector.elementAt(i));

                tmpModifierVector=stringVector;
        }
}
```

3. Classe AttributsDescription

```
/**
* Class AttributsDescription
*
* Draw the description's attributes'
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* @Pre OFLPanel and ClassDescription have been instancied
*
* @Post the panel which represents the description's atributes is created
*
*/

import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.Insets;
import javax.swing.JLabel;
import javax.swing.JPanel;
import java.util.Vector;
import javax.swing.JList;


public class AttributsDescription extends JPanel
{

	private Vector attributCVector;

	// for the display
	private int nb;

	private Vector stringVector;

	private JLabel desc = new JLabel ();

	/**
	* Creates 2 Vectors : one which will contain the string representation
	* of the methods (stringVector), and the other one which will contain ...
	*
	* @since JDK1.3
	*/
	public AttributsDescription ()
	{
		stringVector    = new Vector();
		attributCVector = new Vector();
	}

	/**
	* Sets the layout and the background color of the panel.
	* Adds the attributs to the panel by going trough stringVector.
	* If it's empty than a "" is added to the panel,
	* else each attribut is added.
	*
	* @since JDK1.3
	*
	* @Pre it's a new description. The description is created and not modified
	*/
	public void initAttributsDescription()
	{
		removeAll();
		GridBagLayout gridbag = new GridBagLayout();
```

```
GridBagConstraints c = new GridBagConstraints();

		// Set the frame background color
		setBackground (Color.white);

		// Set the frame layout
	setLayout (gridbag);

	// Set the constraints properties
	   c.anchor = GridBagConstraints.WEST;
		c.gridwidth = GridBagConstraints.REMAINDER;
			c.insets = new Insets (0,0,0,5);

		// Set  text
		if(stringVector.size()==0)
		{
			c.insets = new Insets (0,0,5,5);
			desc.setText(" ");
		   gridbag.setConstraints(desc, c);
			add(desc);
			nb=0;
		}
		else
		{
			for(int i=0;i<stringVector.size();i++)
			{

				if(i==stringVector.size()-1)
					c.insets = new Insets (0,0,0,5);

				desc=addattribut((stringVector.elementAt(i)).toString());
			gridbag.setConstraints(desc, c);
				add(desc);
			}

			nb=stringVector.size();
		}
	}

	/**
	* return the number of attributs in the vector
	*
	* @return nb the number of attributes
	*
	* @since JDK1.3
	*
	*/
	public int getnbAttributs ()
	{
		return nb;
	}

	/**
	* Adds an attribut to a Vector
	*
	* @param attr the attribut to add
	*
	* @return descAttrib JLabel to be added to the Vector
	*
	* @since JDK1.3
	*
	* @Pre attributes have been defined
	*
	* @Post attributes are added to the panel
	*
	*/
	public JLabel addattribut(String attr)
```

```
	{
		JLabel descAttrib = new JLabel ();
		descAttrib.setText(attr);

		return  descAttrib;
	}

	/**
	* @return attributCVector return a vector which contains all the attributes created
	*
	* @since JDK1.3
	*
	* @Pre attributes have been created
	*/
	public Vector getAttributCVector()
	{
		return attributCVector;
	}

	/**
	* @return stringVector retrun the string representation of a method
	*
	* @since JDK1.3
	*
	* @Pre stringVector exists
	*/
	public Vector getStringVector()
	{
		return stringVector;
	}


	/**
	* if an attribute has been modified, change attributCVector and stringVector
	*
	* @param vector
	*        stringVector
	*
	* @since JDK1.3
	*/
	public void changeAttributC(Vector vector,Vector stringvector)
	{
		attributCVector =(Vector)vector.clone();
		stringVector    =(Vector)stringvector.clone();

		initAttributsDescription();

	}
}
```

4. Classe ClassDescription

```
/**
* Class ClassDescription
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* Allows the draw of the description in the panel (OFLPanel)
*
* @Pre user clicked on OK button in DescriptionFrame
*
* @Post the new description is drawn
*/

import java.awt.Color;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.Insets;
import javax.swing.JPanel;
import javax.swing.BorderFactory;
import java.awt.Graphics;
import java.awt.Dimension;
import java.util.Vector;
import java.lang.Object;
import javax.swing.border.TitledBorder;
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import java.awt.Cursor;
import java.awt.event.MouseMotionListener;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import java.awt.Font;

public class ClassDescription extends JPanel
{

	private int posiX,posiY;
	private int larg, haut;
	private int valAttrib;
	private NameDescription nameD;
	private AttributsDescription attributsD;
	private MethodsDescription methodsD;
	private ImageIcon moinsIcon;
	private ImageIcon plusIcon;

	// Type of border
	private TitledBorder type;

	private JLabel descriptionNameLabel = new JLabel ();
	private JLabel descriptionModifierLabel = new JLabel ();

	private boolean attributsBool;
	private boolean methodsBool;

	// Configure the layout
	private GridBagLayout gridbag = new GridBagLayout();
	private GridBagConstraints c = new GridBagConstraints();


	/**
	* To initialize a new description
	*
	* Class extends JPanel which contains the description draw, composed by
	* the draw of :
```

```
	*		NameDescription class
	*   AttributsDescription class
	*   MethodsDescription class
	*
	* It contains the coordinates of a description in OFLPanel, and its height
	*
	* @since JDK1.3
	*
	* @see OFLPanel
	*/
	public ClassDescription (NameDescription name,AttributsDescription attrib, MethodsDescription
methods)
	{
		//**********
		nameD=name;
		attributsD=attrib;
		methodsD=methods;
		attributsBool=true;
		methodsBool=true;

	}

	/**
	* The draw of a new description
	* Creates the UML representation of a description
	*
	* @since JDK1.3
	*/
	public void ClassDescription2()
	{
	   // buttons to show or hide methods or attributes
		moinsIcon = new ImageIcon ("moins.gif") ;
		plusIcon = new ImageIcon ("plus.gif") ;

		setBackground (Color.white);

		c.anchor = GridBagConstraints.WEST;
		c.gridwidth = GridBagConstraints.REMAINDER;

		setLayout (gridbag);

		//**********
		type =
BorderFactory.createTitledBorder(BorderFactory.createMatteBorder(1,0,0,0,(Color.black)),nameD.getType());

		setBorder(type);

		descriptionNameLabel=nameD.getNameLabel();
		descriptionModifierLabel=nameD.getModifierLabel();


		descriptionNameLabel.setFont(new
Font(descriptionNameLabel.getFont().getName(),Font.BOLD,descriptionNameLabel.getFont().getSize()));

		c.insets = new Insets (0,5,0,5);
		gridbag.setConstraints(descriptionNameLabel, c);
		add(descriptionNameLabel);


		c.insets = new Insets (0,10,5,5);
		gridbag.setConstraints(descriptionModifierLabel, c);
		add(descriptionModifierLabel);


		c.insets = new Insets (0,5,10,0);
		gridbag.setConstraints(attributsD, c);
		add(attributsD);
```

```
		c.insets = new Insets (5,0,0,0);
		gridbag.setConstraints(methodsD, c);
		add(methodsD);

		//**********
		larg=(int)getPreferredSize().getWidth();
		haut=(int)getPreferredSize().getHeight();

		if((larg-10)<(nameD.getTypeLabel().getPreferredSize().getWidth()))
		{
			larg=(int)nameD.getTypeLabel().getPreferredSize().getWidth()+15;
		}

		valAttrib=56+10+(17*(attributsD.getnbAttributs ()));

		if(attributsD.getnbAttributs ()==0)
			valAttrib=valAttrib+19;

	}

	/**
	* if a description is modified or moved by the user,
	* redraw the description
	*
	* @since JDK1.3
	*
	* @see DescriptionFrame
	*/
	public void ClassDescription3()
	{
		type =
BorderFactory.createTitledBorder(BorderFactory.createMatteBorder(1,0,0,0,(Color.black)),nameD.getType());
		setBorder(type);

		//**********
		larg=(int)getPreferredSize().getWidth();
		haut=(int)getPreferredSize().getHeight();

		if((larg-10)<(nameD.getTypeLabel().getPreferredSize().getWidth()))
		{
			larg=(int)nameD.getTypeLabel().getPreferredSize().getWidth()+15;
		}
	}

	public void paintComponent(Graphics g)
	{
		 valAttrib=56+10+(17*(attributsD.getnbAttributs ()));

		if(attributsD.getnbAttributs ()==0)
			valAttrib=valAttrib+19;

		super.paintComponent(g) ;

	   if(attributsBool==true && methodsBool==true)
	   {
			int xPoints[] ={2,2,larg-2,larg-2};
		int yPoints[] ={8,haut-2,haut-2,8};

		g.drawPolyline(xPoints,yPoints,xPoints.length);

			g.drawLine (2,56,(larg-3),56) ;
			g.drawLine (2,valAttrib+2,(larg-3),valAttrib+2) ;

			g.drawImage (moinsIcon.getImage(), -1,50, null) ;
			g.drawImage (moinsIcon.getImage(), -1,valAttrib-4, null) ;
```

```
		}
		else if(attributsBool==false && methodsBool==false)
		{
			int xPoints[] ={2,2,larg-2,larg-2};
		int yPoints[] ={8,haut-7,haut-7,8};

		g.drawPolyline(xPoints,yPoints,xPoints.length);


			g.drawImage (plusIcon.getImage(), -1,50, null) ;
			g.drawImage (plusIcon.getImage(), 10,50, null) ;

		}
		else if(attributsBool==true && methodsBool==false)
	{
			int xPoints[] ={2,2,larg-2,larg-2};
		int yPoints[] ={8,haut-2,haut-2,8};

		g.drawPolyline(xPoints,yPoints,xPoints.length);

			g.drawLine (2,56,(larg-3),56) ;

			g.drawImage (moinsIcon.getImage(), -1,50, null) ;
			g.drawImage (plusIcon.getImage(), 10,50, null) ;
		}
		else if(attributsBool==false && methodsBool==true)
		{

			int xPoints[] ={2,2,larg-2,larg-2};
		int yPoints[] ={8,haut-2,haut-2,8};

		g.drawPolyline(xPoints,yPoints,xPoints.length);

			g.drawLine (2,56,(larg-3),56) ;

			g.drawImage (plusIcon.getImage(), -1,50, null) ;
			g.drawImage (moinsIcon.getImage(), 10,50, null) ;
		}
}

/**
*
* @Pre
*
* @Post
*
*/
public void setBoolAttributs()
{
		if(attributsBool==true)
		{
			attributsBool=false;

			c.insets = new Insets (0,20,5,5);
			gridbag.setConstraints(descriptionModifierLabel, c);

			c.insets = new Insets (0,0,0,0);
			gridbag.setConstraints(attributsD, c);

			attributsD.setVisible(false);
		}
		else
		{
			attributsBool=true;

			if(methodsBool==true)
			{
```

```
                    c.insets = new Insets (0,10,5,5);
                    gridbag.setConstraints(descriptionModifierLabel, c);
                }

                c.insets = new Insets (0,5,10,0);
                gridbag.setConstraints(attributsD, c);

                attributsD.setVisible(true);
        }

        larg=(int)getPreferredSize().getWidth();
        haut=(int)getPreferredSize().getHeight();

        if((larg-10)<(nameD.getTypeLabel().getPreferredSize().getWidth()))
        {
                larg=(int)nameD.getTypeLabel().getPreferredSize().getWidth()+15;
        }
}

/**
*
* @Pre
*
* @Post
*
*/
public void setBoolMethods()
{


        if(methodsBool==true)
        {
                methodsBool=false;

                c.insets = new Insets (0,20,5,5);
                gridbag.setConstraints(descriptionModifierLabel, c);

                c.insets = new Insets (0,0,0,0);
                gridbag.setConstraints(methodsD, c);

                methodsD.setVisible(false);
        }
        else
        {

                methodsBool=true;

                if(attributsBool==true)
                {
                        c.insets = new Insets (0,10,5,5);
                        gridbag.setConstraints(descriptionModifierLabel, c);
                }
                c.insets = new Insets (5,0,0,0);
                gridbag.setConstraints(methodsD, c);

                methodsD.setVisible(true);
        }

        larg=(int)getPreferredSize().getWidth();
        haut=(int)getPreferredSize().getHeight();

        if((larg-10)<(nameD.getTypeLabel().getPreferredSize().getWidth()))
        {
                larg=(int)nameD.getTypeLabel().getPreferredSize().getWidth()+15;
        }
}
```

```
/**
*
* @Pre
*
* @Post
*
*/
public void buildString()
{
	nameD.buildString();
}

/**
* @param
* @param
* @param
*
* @Pre
*
* @Post
*
*/
public void initNameDescription (String name,String type,String modif)
{
	nameD.initNameDescription(name,type,modif);
}

/**
* @param
* @param
* @param
*
* @Pre
*
* @Post
*
*/
public void reinitNameDescription (String name,String type,String modif)
{
	nameD.reinitNameDescription (name,type,modif);
}

/**
*
* @Pre
*
* @Post
*
*/
public void initAttributsDescription()
{
	attributsD.initAttributsDescription();
}

/**
*
* @Pre
*
* @Post
*
*/
public void initMethodsDescription ()
{
	methodsD.initMethodsDescription();
}
```

```
/**
*
* @Pre
*
* @Post
*
*/
public boolean  getBoolAttributs()
{
	return attributsBool;
}

public NameDescription getNameDescription()
{
	return nameD;
}


public AttributsDescription getAttributsDescription()
{
	return attributsD;
}


public MethodsDescription getMethodsDescription()
{
	return methodsD;
}


public boolean getBoolMethods()
{
	return methodsBool;
}

public int getHauteur()
{
	return haut;
}

public int getLargeur()
{
	return larg;
}

public int getPositionX()
{
	return posiX;
}

public int getPositionY()
{
	return posiY;
}

public int getPositionXmax()
{
	return (larg+posiX);
}

public int getPositionYmax()
{
	return (haut+posiY);
}

public int getValAttrib()
{
```

```
                return valAttrib;
        }

        public void setPositionX(int valx)
        {
                posiX=valx;
        }

        public void setPositionY(int valy)
        {
                posiY=valy;
        }
}
```

5. Classe DescriptionFrame

```
/**
* Class DescriptionFrame
*
* Allows the user to enter the name,
* the attributes and the methods of the description
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* @Pre user clicked on description-component in the tree
*
* @Post description can be drawn
*
*/

import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.Insets;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import java.util.Vector;
import java.awt.Dimension;

public class DescriptionFrame  extends JFrame
{
	private JTextField descName = new JTextField (11);
	private JComboBox typeList;
	private static JComboBox modifierJcomboBox;
	private boolean typeAddBool=false;

	/**
	* Initialize all the parts of the descripion frame :
	*		-the description's name
	*		-the description's modifier(s)
	*		-the buttons to add Attributes, Methods, Choose modifier
	*
	* @since JDK1.3
	*
	* @Pre user clicked on description-component in the tree
	*
	* @Post description can be drawn
	*/
	public DescriptionFrame (final ClassDescription classDescription,final int index,final boolean newClass)
	{
		typeList= new JComboBox (LoadObject.getDescriptionTypeVector());

		JPanel namePane = new JPanel ();
		JPanel buttonPane = new JPanel ();

		JLabel descLabel = new JLabel ("Description name :             ");
		JLabel typeLabel = new JLabel ("Choose the description type :  ");
		JLabel modifLabel = new JLabel (" Modifier :                ");

		modifierJcomboBox=new
JComboBox(classDescription.getNameDescription().getModifierVector());
```

```
                modifierJcomboBox.setPreferredSize(new Dimension(121,21));
                modifierJcomboBox.setEditable(false);
                modifierJcomboBox.setMaximumRowCount(5);

                JButton modifierButton     = new JButton(" Choose ");

                JButton okButton     = new JButton("  OK  ");
                JButton cancelButton = new JButton("Cancel");

                JButton attribButton = new JButton(" Attributes ");
                JButton methodsButton= new JButton(" Methods  ");

                GridBagLayout gridbag = new GridBagLayout();
GridBagConstraints c = new GridBagConstraints();
GridBagLayout gridbagPane = new GridBagLayout();
GridBagConstraints cPane = new GridBagConstraints();

                // Default text for the description name
                descName.setText (classDescription.getNameDescription().getNameString());

                typeList.setSelectedItem(classDescription.getNameDescription().getType());


                // Set the frame layout
getContentPane().setLayout (gridbag);

                // Set the gribag properties
                c.insets = new Insets (40,10,20,10);
                c.anchor = GridBagConstraints.WEST;
                c.gridwidth = GridBagConstraints.RELATIVE;

                // Set the gribagPane properties
                cPane.insets = new Insets (10,10,10,10);
                cPane.anchor = GridBagConstraints.WEST;
                cPane.gridwidth = GridBagConstraints.RELATIVE;

                // Set the namePane layout
                namePane.setLayout (gridbagPane);

                // Set the namePane border
                namePane.setBorder(BorderFactory.createEtchedBorder());

                // Add descLabel to the gridbagPane constraints and to the namePane
                gridbagPane.setConstraints(descLabel, cPane);
                namePane.add (descLabel);

                // Add descName to the gridbagPane constraints and to the namePane
                cPane.gridwidth = GridBagConstraints.REMAINDER; // to go to the next line
                gridbagPane.setConstraints(descName, cPane);
                namePane.add (descName);

                // Add typeLabel to the gridbagPane constraints and to the namePane
                cPane.gridwidth = GridBagConstraints.RELATIVE;
                gridbagPane.setConstraints(typeLabel, cPane);
                namePane.add (typeLabel);

                // Add typeList to the gridbagPane constraints and to the namePane
                cPane.gridwidth = GridBagConstraints.REMAINDER;
                gridbagPane.setConstraints(typeList, cPane);
                namePane.add (typeList);

                // Add modifLabel to the gridbagPane constraints and to the namePane
                cPane.gridwidth = GridBagConstraints.RELATIVE;
                gridbagPane.setConstraints(modifLabel, cPane);
                namePane.add (modifLabel);
```

```
cPane.insets = new Insets (10,-115,10,10);
// Add modifList to the gridbagPane constraints and to the namePane
cPane.gridwidth = GridBagConstraints.RELATIVE;
gridbagPane.setConstraints(modifierJcomboBox, cPane);
namePane.add (modifierJcomboBox);

cPane.gridwidth = GridBagConstraints.REMAINDER;
cPane.insets = new Insets (10,-110,10,10);
gridbagPane.setConstraints(modifierButton, cPane);
namePane.add (modifierButton);

cPane.insets = new Insets (10,10,10,10);

// Add namePane to the gridbag constraints and to the frame
gridbag.setConstraints(namePane, c);
getContentPane().add (namePane);


// Attributes button
c.gridwidth = GridBagConstraints.REMAINDER;
buttonPane.setLayout (gridbagPane);
buttonPane.setBorder(BorderFactory.createEtchedBorder());

cPane.insets = new Insets (24,10,10,10);
cPane.gridwidth = GridBagConstraints.REMAINDER;
gridbagPane.setConstraints(attribButton, cPane);
buttonPane.add (attribButton);


// Methods button
cPane.insets = new Insets (10,10,24,10);
gridbagPane.setConstraints(methodsButton, cPane);
buttonPane.add (methodsButton);

gridbag.setConstraints(buttonPane, c);
getContentPane().add (buttonPane);

// OK Button
c.insets = new Insets (30,100,20,20);
c.gridwidth = GridBagConstraints.RELATIVE;

gridbag.setConstraints(okButton, c);
getContentPane().add(okButton);

// Cancel Button
c.insets = new Insets (30,0,20,20);
c.gridwidth = GridBagConstraints.REMAINDER;

gridbag.setConstraints(cancelButton, c);
getContentPane().add(cancelButton);


// Set the frame properties
setTitle ("Description");
setResizable (false);
pack ();
setVisible (true);

// Add listener to the cancel button
/**
* if the user cliks on Cancel, all the changes he made are
* cancelled.
*
* the name of the description which is added in the vector which contains
* the types is removed from.
*
```

```
		* @see #addActionListener
		* @see LoadObject
		*/
		cancelButton.addActionListener (new ActionListener ()
		{
			public void actionPerformed (ActionEvent e)
			{
				if(typeAddBool==true)
					LoadObject.deleteClasstoTypeVector();

				cancel_actionPerformed(e);
			}
		});

		// Add listener to the modifier button
		/**
		* if the user clicks on modifier, ModifierFrame is created and
		* the window, which is implemented by, is opened
		*
		* @since JDK1.3
		*
		* @see ModifierFrame
		*
		* @Post modifiers are added
		*/
		modifierButton.addActionListener (new ActionListener ()
		{
			public void actionPerformed (ActionEvent e)
			{
				ModifierFrame modif=new
ModifierFrame(classDescription.getNameDescription().getModifierVector(),"description");
			}
		});

		// Add listener to the ok button
		/**
		* if the user clicks on OK, call getText method
		*
		* @since JDK1.3
		*
		* @see #getText
		*/
		okButton.addActionListener (new ActionListener ()
		{
			public void actionPerformed (ActionEvent e)
			{
				getText(classDescription,index,newClass);
			}
		});

		// Add listener to the attributes button
		/**
		* if the users clicks on Attributes :
		*  - if newClass is true (means that the class is not modified)
		*  - and if typeAddBool is false
		*  - and if the description's name is not contained in the vector
		*    which contains all the types (so that descriptions can't have the same name)
		* then :
		*  -  the name of the description is added to the vector (so that
		*     attribute's type can be a description)
		*  - window to create attributes is opened
		*
		* if newClass is false (means that the description is being modified),
		* the name of the description is also modified in the vector
		* @since JDK1.3
		*
		* @see LoadObject#getTypeVector
```

```
		* @see LoadObject#addClasstoTypeVector
		*/
		attribButton.addActionListener (new ActionListener ()
		{
			public void actionPerformed (ActionEvent e)
			{
				if(newClass==true && typeAddBool==false &&
(LoadObject.getTypeVector().contains(descName.getText())==false))
				{
					LoadObject.addClasstoTypeVector(descName.getText());
					typeAddBool=true;

					AttributeFrame attrib = new
AttributeFrame(classDescription.getAttributsDescription(),"attribut");
				}
				else if((newClass==false) && (typeAddBool==false) &&
(LoadObject.getTypeVector().contains(descName.getText())==false))
				{

	LoadObject.changeClasstoTypeVector(classDescription.getNameDescription().getNameString(),descNa
me.getText());
					typeAddBool=true;

					AttributeFrame attrib = new
AttributeFrame(classDescription.getAttributsDescription(),"attribut");
				}
				else if(newClass==true && typeAddBool==true)
				{
					Vector test= new
Vector((Vector)LoadObject.getTypeVectorPart().clone());

					if(test.contains(descName.getText())==false)
					{
						AttributeFrame attrib = new
AttributeFrame(classDescription.getAttributsDescription(),"attribut");
					}
				}
				else if((newClass==false) && (typeAddBool==false))
				{
					Vector test= new
Vector((Vector)LoadObject.getTypeVectorPart(index).clone());

					if(test.contains(descName.getText())==false)
					{
						AttributeFrame attrib = new
AttributeFrame(classDescription.getAttributsDescription(),"attribut");
					}
				}
			}
		});


		// Add listener to the methods button
		/**
		* Same processus as attributs
		*
		* @since JDK1.3
		*
		* @see LoadObject#getTypeVector
		* @see LoadObject#addClasstoTypeVector
		*/
		methodsButton.addActionListener (new ActionListener ()
		{
			public void actionPerformed (ActionEvent e)
			{
```

```
                                        if(newClass==true && typeAddBool==false &&
(LoadObject.getTypeVector().contains(descName.getText())==false))
                                        {
                                                LoadObject.addClasstoTypeVector(descName.getText());
                                                typeAddBool=true;

                                                MethodFrame method = new
MethodFrame(classDescription.getMethodsDescription());
                                        }
                                        else if((newClass==false) && (typeAddBool==false) &&
(LoadObject.getTypeVector().contains(descName.getText())==false))
                                        {

        LoadObject.changeClasstoTypeVector(classDescription.getNameDescription().getNameString(),descNa
me.getText());
                                                typeAddBool=true;

                                                MethodFrame method = new
MethodFrame(classDescription.getMethodsDescription());
                                        }
                                        else if(newClass==true && typeAddBool==true  )
                                        {
                                                Vector test= new
Vector((Vector)LoadObject.getTypeVectorPart().clone());

                                                if(test.contains(descName.getText())==false)
                                                {
                                                        MethodFrame method = new
MethodFrame(classDescription.getMethodsDescription());
                                                }
                                        }
                                        else if((newClass==false) && (typeAddBool==false) )
                                        {
                                                Vector test= new
Vector((Vector)LoadObject.getTypeVectorPart(index).clone());

                                                if(test.contains(descName.getText())==false)
                                                {
                                                        MethodFrame method = new
MethodFrame(classDescription.getMethodsDescription());
                                                }
                                        }
                                }
                        });
        }


        // Cancel Listener
        /**
        * Close the window
        */
        void cancel_actionPerformed (ActionEvent e)
        {
                setVisible (false);
        }

        // OK Listener
        /**
        * get the name of the description and add it in the vector in class LoadObject
        */
        void getText(ClassDescription classDescription,int index,boolean newClass)
        {
                if((descName.getText().length()!=0))
                {
                        classDescription.buildString();

                        if(newClass==true )
```

```
                {
                        boolean good=false;

                        if (typeAddBool==true &&
(LoadObject.getTypeVector().contains(descName.getText())==false))
                        {
                                LoadObject.deleteClasstoTypeVector();
                                typeAddBool=false;
                                good=true;
                        }
                        else if(typeAddBool==true)
                        {
                                Vector test= new Vector((Vector)LoadObject.getTypeVectorPart(
).clone());

                                if(test.contains(descName.getText())==false)
                                        good=true;
                        }

                        if(typeAddBool==false &&
(LoadObject.getTypeVector().contains(descName.getText())==false))
                        {
                                LoadObject.addClasstoTypeVector(descName.getText());
                                typeAddBool=true;
                                good=true;
                        }

                        if(good==true)
                        {

        classDescription.initNameDescription(descName.getText(),(typeList.getSelectedItem()).toString(),(classD
escription.getNameDescription().getModifierString()));
                                classDescription.initAttributsDescription();
                                classDescription.initMethodsDescription();
                                classDescription.ClassDescription2();

                                setVisible (false);

                                OFLDescriptionPanel.addDescription      (classDescription);

                                Main.mainAddEnd(classDescription);
                        }
                }
                else if(newClass==false)
                {
                        if((descName.getText() !=
classDescription.getNameDescription().getNameString()))
                        {


        if((LoadObject.getTypeVector().contains(descName.getText())==false))

        LoadObject.changeClasstoTypeVector(classDescription.getNameDescription().getNameString(),descNa
me.getText());
                        }

                        Vector test= new
Vector((Vector)LoadObject.getTypeVectorPart(index).clone());

                        if(test.contains(descName.getText())==false)
                        {

        classDescription.reinitNameDescription(descName.getText(),(typeList.getSelectedItem()).toString(),(clas
sDescription.getNameDescription().getModifierString()));

                                classDescription.initAttributsDescription();
                                classDescription.initMethodsDescription();
```

```
                                        classDescription.ClassDescription3();
                                        setVisible (false);

                                        OFLDescriptionPanel.addDescription          (classDescription);


                                        Main.mainSet(index,classDescription);
                                }
                        }
                }
        }

        /**
        * Set the modifiers
        *
        * @param stringVector to initialize the modifiers
        */
        public static void setJcomboBoxModifierText(Vector stringvector)
        {
                modifierJcomboBox.removeAllItems();

                for (int i=0;i<stringvector.size();i++)
                        modifierJcomboBox.addItem((String)stringvector.elementAt(i));
        }
}
```

6. Classe DescriptionRelation

```
/**
* Class DescriptionRelation
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* TO DRAW RELATIONS BETWEEN DESCRIPITONS
* THIS CLASS IS NOT COMPLETLY IMPLEMENTED
*/

import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import java.awt.Graphics;
import javax.swing.JPanel;
import java.awt.Color;
import java.awt.Dimension;

public class DescriptionRelation extends JPanel
{
	ClassDescription source,dest;
	private static int larg, haut;

	public DescriptionRelation (ClassDescription source, ClassDescription dest)
	{
		this.source = source;
		this.dest = dest;
		setPreferredSize (new Dimension(source.getPositionX(),dest.getPositionY()));
		setVisible(true);
	}

	public void paintComponent(Graphics g)
	{
		super.paintComponent(g) ;

		g.drawLine
(source.getPositionX(),source.getPositionY(),dest.getPositionX(),dest.getPositionY());
	}
}
```

7. Classe EditMenu

```
/**
* Class EditMenu
*
* Contains a menu and the event which can happend on
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* @Pre MainMenu has been created
*
* @Post a new menu is added
*
*/

import javax.swing.ImageIcon;
import javax.swing.JMenu;
import javax.swing.JMenuItem;

public class EditMenu extends JMenu
{
	private  JMenuItem menuUndo, menuRedo,menuDelete, menuResize, menuMove;

	/**
	* Creates 3 menu items : Undo, Redo, Delete (to delete a description),
	* Resize (to resize a description)
	*
	* These menu items are added to the menu
	*
	* @param menuName the name of the menu
	*
	* @since JDK1.3
	*
	* @Pre MainMenu has been created
	*
	* @Post a new menu is added
	*
	*/
	public EditMenu (String menuName)
	{
		setText (menuName);

		// Create menus item
		menuUndo = new JMenuItem("Undo");
		menuRedo = new JMenuItem("Redo");
		menuDelete = new JMenuItem("Delete");
		menuResize = new JMenuItem("Resize");
		menuMove = new JMenuItem("Move");

		// Add icons to the items
		menuUndo.setIcon (new ImageIcon (Windows.class.getResource ("Undo.gif")));
		menuRedo.setIcon (new ImageIcon (Windows.class.getResource ("Redo.gif")));
		menuDelete.setIcon (new ImageIcon (Windows.class.getResource ("Delete.gif")));

		// Add items to the Edit menu
		add(menuUndo);
		add(menuRedo);
		add(menuDelete);
		add(menuResize);
		add(menuMove);
	}
}
```

8. Classe ExplorerPanel

```
/**
* Class ExplorerPanel
*
* Creates a panel with a tree, which represents all the classes and relations, in it
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* @Pre Windows ahs been created
*
* @Post A tree which contains component-language, descriptions-components,
*       and relationships-components, is created
*
*/

import java.awt.BorderLayout;
import java.awt.Dimension;
import javax.swing.JPanel;
import javax.swing.JScrollPane;

public class ExplorerPanel extends JPanel
{
            /**
            * Creates a Nodes contained by a scroller
            * Sets the scroller size, layout and background color.
            *
            * @see Nodes
            *
            * @Pre Windows ahs been created
            *
            * @Post A tree which contains component-language, descriptions-components,
            *       and relationships-components, is created
            *
            */
            public ExplorerPanel ()
            {
                        Nodes nodes = new Nodes ();

                        JScrollPane scroller = new JScrollPane(nodes.getLanguageTree());

      scroller.setPreferredSize(new Dimension(200,500));

      setLayout(new BorderLayout());

      add(scroller,BorderLayout.CENTER);
            }
}
```

9. Classe FileMenu

```
/**
* Class FileMenu
*
* Contains a menu and the event which can happend on
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* @Pre MainMenu has been created
*
* @Post a new menu is added
*
*/

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.ImageIcon;
import javax.swing.JFileChooser;
import javax.swing.JMenu;
import javax.swing.JMenuItem;


public class FileMenu extends JMenu
{
	private  JMenuItem menuNew, menuOpen, menuSave, menuSaveAs, menuQuit;

	/**
	* Creates 3 menu items : New (to create a new document),
	* Open (to open an existant document), Save, Save As (to save a document),
	* Quit (to quit the application).
	* These menu items are added to the menu
	*
	* @param menuName the name of the menu
	*
	* @since JDK1.3
	*
	* @Pre MainMenu has been created
	*
	* @Post a new menu is added
	*
	*/
	public FileMenu (String menuName)
	{
		setText (menuName);

		// Create menus item
		menuNew = new JMenuItem("New");
		menuOpen = new JMenuItem("Open");
		menuSave = new JMenuItem("Save");
		menuSaveAs = new JMenuItem("Save As");
		menuQuit = new JMenuItem("Quit");

		// Add icons to the items
		menuNew.setIcon (new ImageIcon (Windows.class.getResource ("New.gif")));
		menuOpen.setIcon (new ImageIcon (Windows.class.getResource ("Open.gif")));
		menuSave.setIcon (new ImageIcon (Windows.class.getResource ("Save.gif")));
		menuQuit.setIcon (new ImageIcon (Windows.class.getResource ("Exit.gif")));

		// Add items to the File menu
		add(menuNew);
		add(menuOpen);
		add(menuSave);
		add(menuSaveAs);
```

```
		add(menuQuit);

		// Add listeners to the items
		menuOpen.addActionListener (new ActionListener () {
			public void actionPerformed (ActionEvent e)
			{
				open_actionPerformed (e);
			}
		});

		menuSave.addActionListener (new ActionListener () {
			public void actionPerformed (ActionEvent e)
			{
				save_actionPerformed (e);
			}
		});

		menuQuit.addActionListener (new ActionListener () {
			public void actionPerformed (ActionEvent e)
			{
				quit_actionPerformed (e);
			}
		});

	}

	// Open Listener
	/**
	* if the user clicks on "OPEN" in the file menu
	*
	* @param e the action event
	*
	* @Post an open dialog window appears
	*
	*/
	void open_actionPerformed (ActionEvent e)
	{
		JFileChooser chooser = new JFileChooser ();

		chooser.showOpenDialog (this);
	}

	// Save Listener
	/**
	* @param
	*
	* @Pre
	*
	* @Post
	*
	*/
	void save_actionPerformed (ActionEvent e)
	{
	}

	// Quit Listener
	/**
	* @param e the action event
	*
	* @Post quit OFL-ML
	*
	*/
	void quit_actionPerformed (ActionEvent e)
	{
		System.exit (0);
	}
}
```

10. Classe LoadObject

```
import java.util.Vector;

/**
* Class LoadObject
*
* initialize every vector used in the programm
* in order to load them
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* @Pre
*
* @Post
*
*/
public class LoadObject
{
	private static String nameLanguage;

	private static Vector descriptionTypeVector= new Vector();
	private static Vector descriptionUseVector = new Vector ();
	private static Vector descriptionImportVector = new Vector ();

	private static Vector modifierVector= new Vector();
	private static Vector typeVector= new Vector();
	private static int nbTypeLoad;

	/**
	* Initialize the vector which contains the tree nodes.
	* Initialize the vector which contains the modifiers.
	*
	* @since JDK1.3
	*
	* @Pre  Node has been created
	*
	* @Post The names of Language-component, relationships-component and
	*       descriptions-components are initialized
	*
	*/
	public  LoadObject ()
	{
		nameLanguage="Java";

		descriptionTypeVector.addElement("Class");
		descriptionTypeVector.addElement("Interface");

		descriptionUseVector.addElement("Agregate");
		descriptionUseVector.addElement("Compose");

		descriptionImportVector.addElement("Extends");
		descriptionImportVector.addElement("Implements");

		modifierVector.addElement("public");
		modifierVector.addElement("private");
		modifierVector.addElement("protected");
		modifierVector.addElement("static");
		modifierVector.addElement("final");

		typeVector.addElement("Integer");
		typeVector.addElement("String");
		typeVector.addElement("boolean");
		typeVector.addElement("");
```

```
		nbTypeLoad=typeVector.size();
}

/*
* @return nameLanguage the language-component name
*/
public static String getNameLanguage()
{
		return nameLanguage;
}

/*
* @return descriptionTypeVector the description-components vector
*/
public static Vector getDescriptionTypeVector()
{
		return descriptionTypeVector;
}

/*
* @return modifierVector the modifier vector
*/
public static Vector getModifierVector()
{
		return modifierVector;
}

/*
* @return typeVector the type vector
*/
public static Vector getTypeVector()
{
		return typeVector;
}

/*
* @return descriptionUseVector the description use relation vector
*/
public static Vector getDescriptionUseVector()
{
		return descriptionUseVector;
}

/*
* @return descriptionImportVector the description import relation vector
*/
public static Vector getDescriptionImportVector()
{
		return descriptionImportVector;
}

/**
*if a new description is created
*	her name is added in a vector in order to be added too in the
*	tree
*
* @param nameClass the name of the new description
*
* @see Nodes#addDescription(String)
*
*/
public static void addClasstoTypeVector(String nameClass)
{
		typeVector.addElement(nameClass);
		Nodes.addDescriptionTree(nameClass)	;
}
```

```
/**
*if a description is deleted
*       her name is removed from the vector in order to be removed too from the
*       tree
*
* @see Nodes#deleteDescriptionTree()
*/
public static void deleteClasstoTypeVector()
{
            typeVector.removeElementAt(typeVector.size()-1);
            Nodes.deleteDescriptionTree();
}

/**
*
* if a description name is changed
*       her name is changed in the vector in order to be modified too in the
*       tree
*
* @param nameClass the first description's name
* @param newname the new description's name
*
* @see Nodes#changedDescriptionTree(int,String)
*/
public static void changeClasstoTypeVector(String nameClass,String newname)
{
            int index = typeVector.indexOf(nameClass);
            typeVector. set(index,newname);

            Nodes.changeDescriptionTree((index-nbTypeLoad),newname);
}

/**
* to remove an element from typeVector
*
* @param the index of the element to remove
*
* @return tmp the new vector
*/
public static Vector getTypeVectorPart(int eltToRemove)
{
            Vector tmp =new Vector((Vector) typeVector.clone());
            tmp.removeElementAt(eltToRemove+nbTypeLoad);
            return tmp;
}

/**
* to remove the last element of typeVector
*
* @return tmp the new vector
*/
public static Vector getTypeVectorPart()
{
            Vector tmp =new Vector((Vector) typeVector.clone());
            tmp.removeElementAt(typeVector.size()-1);
            return tmp;
}

/**
* @return nbTypeLoad the number of type inTypeVector
*
*/
public static int getNbTypeLoad()
{
            return nbTypeLoad;
} }
```

11. Classe Main

```
/**
* Class Main
*
* Contains the main function
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
*
* @Post Windows class is created
*
*/
import java.util.Vector;
import java.lang.Object;
import javax.swing.UIManager;

public class Main
{
	 static Vector vector=new Vector();

	/**
	* Creates a Windows with the OS look
	*
	* @see Windows
	*
	* @since JDK1.3
	*
	* @Post Windows class is created
	*
	*/
	public static void main (String args [])
	{
		try
		{
			//to have the Windows style
			UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
		}
		catch(Exception e)
		{
			e.printStackTrace();
		}

		LoadObject loadobject =new LoadObject();

		// Create a new window
		Windows fenetre = new Windows();
	}

	/**
	* Adds a ClassDescription to the end of the vector in order
	* to keep them in memory
	*
	* @param x the ClassDescription to add
	*
	* @see ClassDescription
	* @since JDK1.3
	*
	* @Pre Almost one ClassDescription object must have been instancied
	*
	* @Post a new ClassDescription is added to the vector
	*
	*/
	static public void mainAddEnd(ClassDescription x)
```

```
		{
			vector.addElement (x);
		}

		/**
		* Return the size of the vector which contains the ClassDescription
		*
		* @return int the vector size
		*
		* @see ClassDescription
		*
		* @since JDK1.3
		*
		* @Pre the vector exists
		*
		*/
	static public int mainSizeVector()
	{
			return vector.size();
		}

	/**
		* Return the ClassDescription which is in a vector at a precise place
		*
		* @return ClassDescription the ClassDescription contains in the vector at index
		*
		* @param index the place of the ClassDescription in the vector
		*
		* @see ClassDescription
		*
		* @since JDK1.3
		*
		* @Pre ClassDescription and vector exist
		*
		* @Post
		*
		*/
		static public ClassDescription mainEltAt(int index)
		{
			return (ClassDescription)vector.elementAt(index);
		}

		/**
		* @return vector the vector which contains the ClassDescription's objects
		*
		*/
		static public Vector getMainVector()
		{
			return vector;
		}


		/**
		* If the user changes the description's name, it is also changed in the vector
		*
		* @param index the place of the ClassDescription in the vector
		* @param newname the new name of the description
		*
		* @Post the description's name is modified
		*
		*/
		static public void mainSet(int index,Object newname)
		{
			vector.set(index,newname);
		}
}
```

12. Classe MainMenu

```
/**
* Class MainMenu
*
* Creates a menu bar with 2 menus "File" and "Edit"
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* @Pre Windows has been created
*
* @Post a menu bar is added to the window
*
*/

import javax.swing.JMenuBar;

public class MainMenu extends JMenuBar
{
	/**
	* Creates 2 menus and adds them to the menu bar
	*
	* @see FileMenu
	* @see EditMenu
	*
	* @since JDK1.3
	*
	* @Pre Windows has been created
	*
	* @Post a menu bar is added to the window
	*
	*/
	public MainMenu()
	{
		// Create 2 menus (File and Edit)
		FileMenu menuFile= new FileMenu("File");
		EditMenu menuEdit= new EditMenu("Edit");

		// Add the menus to the menu bar
		add (menuFile);
		add (menuEdit);
	}


}
```

13. Classe MethodC

```
/**
* Class MethodC
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* TO SAVE AND LOAD METHODS
*
*/
import java.util.Vector;

public class MethodC
{
	//method name
	private String mName;

	//method type
	private String mType;

	private Vector methodModifierVector= new Vector();

	private Vector attributCVector= new Vector();


	public MethodC(String name,String type,Vector methodmodifiervector,Vector attributcvector)
	{
		mName=name;
		mType=type;
		methodModifierVector=methodmodifiervector;
		attributCVector=attributcvector;
	}
}
```

14. Classe MethodFrame

```
/**
* Class MethodFrame
*
* Creates a frame which allows the user
* to enter the method's name, parameters, return type and modifier(s).
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* @Pre the method button in the description frame has been clicked (DescriptionFrame)
*
* @Post the prototype of the method's description can be created
*
*/
import java.awt.Color;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.Dimension;
import javax.swing.BorderFactory;
import javax.swing.JFrame;
import javax.swing.JComboBox;
import javax.swing.JTextField;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTabbedPane;
import javax.swing.JScrollPane;
import javax.swing.JButton;
import java.util.Vector;

public class MethodFrame extends JFrame
{

	private static Vector tmpModifierVector= new Vector();
	private Vector methodsVector = new Vector();
	private Vector frameMethCV = new Vector();

	private AttributsDescription parametreDescription = new AttributsDescription ();

	private JComboBox typeList= new JComboBox (LoadObject.getTypeVector());
	private static JComboBox modifierJcomboBox;
	private JTextField methName = new JTextField ();

	/**
	* Creates a frame with a tabbed panel which contains 2 others
	* panels : one for the description methods (methodPane)
	* and one to enter the documentation about the methods the user enters
	* (docPane).
	* This frame is created when you click on the button "Methods"
	* in the frame called "Description" (Class DescriptionFrame).
	*
	* The description methods panel contains a scrollPane with a list in it.
	* (This list displays the methods the user adds (<modifier> <return type>
	* methodName (<parameters>) by clicking the button "Add") and an other
	* pane to enter the method's name, modifier(s) and return type.
	*
	* The documentation panel only contains a text area
	*
	* @param methodsDescription
	*
```

```
    * @see MethodsDescription
    * @see DescriptionFrame
    *
    * @since JDK1.3
    *
    * @Pre the method button in the description frame has been clicked (DescriptionFrame)
    *
    * @Post the prototype of the method's description can be created
    *
    */
    public  MethodFrame (final MethodsDescription methodsDescription)
    {
                methodsVector=(Vector)methodsDescription.getStringVector().clone();
                final JList methodList = new JList (methodsVector);

                frameMethCV=(Vector)methodsDescription.getMethodCVector().clone();

                modifierJcomboBox = new JComboBox();
                modifierJcomboBox.setPreferredSize(new Dimension(121,21));
                modifierJcomboBox.setEditable(false);
                modifierJcomboBox.setMaximumRowCount(5);
                typeList.setMaximumRowCount(5);

                JButton modifierButton = new JButton(" Choose ");

                JPanel methodPane = new JPanel ();
                JPanel docPane = new JPanel ();
                JPanel propertiesPane = new JPanel ();

                JLabel methNameLabel = new JLabel ("Name :                    ");
                JLabel typeLabel = new JLabel ("Method type :  ");
                JLabel modifLabel = new JLabel("Modifier :               ");

                JTabbedPane tabbedPane = new JTabbedPane();

                JScrollPane scroller = new JScrollPane(methodList);

                JButton deleteButton   = new JButton(" Delete ");
                JButton addButton      = new JButton("  Add    ");
                JButton parametreButton= new JButton(" Add Parameter ");
                JButton okButton       = new JButton("   Ok   ");
                JButton cancelButton   = new JButton(" Cancel ");

                GridBagLayout gridbag = new GridBagLayout();
GridBagConstraints c = new GridBagConstraints();


                methName.setText ("MethodName"); // the default text for the method's name

                methodList.setBackground (Color.white);

                // Set the backgound color and the size of the scroller
                scroller.setBackground (Color.white);
                scroller.setPreferredSize(new Dimension (300,200));

                // Set the methodPane layout
    methodPane.setLayout (gridbag);

    c.anchor = GridBagConstraints.WEST;
                c.gridwidth = GridBagConstraints.RELATIVE;

                // Adds the scroller to the methodPane
                gridbag.setConstraints(scroller, c);
methodPane.add (scroller);

                ////////////////////
    // THE BUTTONS //
```

```
    /////////////////

                // Adds the delete button to the methodPane
                c.insets = new Insets (0,30,0,0);
                c.gridwidth = GridBagConstraints.REMAINDER;
                gridbag.setConstraints(deleteButton, c);
methodPane.add (deleteButton);



                ////////////////////////////////////
                // Properties of the method       //
                //(param,name,return)             //
                ////////////////////////////////////
                c.insets = new Insets (10,10,10,10);
                c.anchor = GridBagConstraints.WEST;
                c.gridwidth = GridBagConstraints.RELATIVE;

                propertiesPane.setLayout (gridbag);
                propertiesPane.setBorder(BorderFactory.createEtchedBorder());
                propertiesPane.setPreferredSize(new Dimension (300,120));
                typeList.setPreferredSize(new Dimension (100,21));
                methName.setPreferredSize(new Dimension (121,21));



                // Add "Name"
                c.insets = new Insets (10,-80,5,0);
                gridbag.setConstraints(methNameLabel, c);
                propertiesPane.add (methNameLabel);

                // Add the texfield to enter the method's name
                c.gridwidth = GridBagConstraints.REMAINDER;
                c.insets = new Insets (10,-40,0,0);
                gridbag.setConstraints(methName, c);
                propertiesPane.add (methName);

                // Add "Method type"
                c.gridwidth = GridBagConstraints.RELATIVE;
                c.insets = new Insets (10,-80,5,0);
                gridbag.setConstraints(typeLabel, c);
                propertiesPane.add (typeLabel);

                // Add a list to select the method's type
                c.gridwidth = GridBagConstraints.REMAINDER;
                c.insets = new Insets (10,0,5,0);
                gridbag.setConstraints(typeList, c);
                propertiesPane.add (typeList);

                // Add "Modifier"
                c.gridwidth = GridBagConstraints.RELATIVE;
                c.insets = new Insets (10,-80,10,0);
                gridbag.setConstraints(modifLabel, c);
                propertiesPane.add (modifLabel);

                // Add a list to select the method's modifier
                c.gridwidth = GridBagConstraints.RELATIVE;
                c.insets = new Insets (10,-40,10,10);
                gridbag.setConstraints(modifierJcomboBox, c);
                propertiesPane.add (modifierJcomboBox);

                c.gridwidth = GridBagConstraints.REMAINDER;
                c.insets = new Insets (10,0,10,10);
                gridbag.setConstraints(modifierButton, c);
                propertiesPane.add (modifierButton);

                // Add the propertiesPane to the methodPane
```

```
                c.gridwidth = GridBagConstraints.RELATIVE;
                gridbag.setConstraints(propertiesPane, c);
methodPane.add (propertiesPane);

    /////////////////
    // THE BUTTONS //
    /////////////////
    // Add "Add" button to the methodPane
                c.insets = new Insets (-170,10,0,0);
                c.gridwidth = GridBagConstraints.REMAINDER;
                gridbag.setConstraints(parametreButton, c);
methodPane.add (parametreButton);




                // Add "OK" button to the methodPane
    c.insets = new Insets (10,13,10,0);
    c.gridwidth = GridBagConstraints.RELATIVE;
                gridbag.setConstraints(okButton, c);
methodPane.add (okButton);

                // Add "Cancel" button to the methodPane
c.insets = new Insets (10,-105,10,10);
                c.gridwidth = GridBagConstraints.RELATIVE;
                gridbag.setConstraints(cancelButton, c);
methodPane.add (cancelButton);


                // Add "Add" button to the methodPane
                c.insets = new Insets (-190,-100,0,0);
                c.gridwidth = GridBagConstraints.REMAINDER;
                gridbag.setConstraints(addButton, c);
methodPane.add (addButton);

                // Put methodPane and docPane in a tabbed pane
    tabbedPane.addTab("Methods ", null, methodPane, null);
tabbedPane.addTab("Documentation", null, docPane, null);
tabbedPane.setSelectedIndex(0);


getContentPane ().add (tabbedPane);
setSize (480,450);
setResizable (false);

setVisible (true);

                ////////////////////
                // ACTION LISTENER //
                ////////////////////

                /**
                * to open the modifier frame
                *
                * @param e the action event
                *
                * @see ModifierFrame
                *
                * @since JDK1.3
                *
                * @Post ModifierFrame is created
                *
                */
                modifierButton.addActionListener (new ActionListener ()
                {
                        public void actionPerformed (ActionEvent e)
```

```
                {
                        ModifierFrame modif=new ModifierFrame(tmpModifierVector,"Method");

                }
        });


        // Add listener to the delete button
        deleteButton.addActionListener (new ActionListener ()
        {
                /**
                * To delete methods from the description
                * The selected methods are first removed from the methodList
                * and if the user clicks OK, the selected methods are removed
                * from the methods' vector in class MethodsDescription (else
                * nothing happens).
                *
                * @param e the action event
                *
                * @see MethodsDescription
                *
                * @since JDK1.3
                *
                * @Pre methods are selected
                *
                * @Post selected methods are removed
                *
                */
                public void actionPerformed (ActionEvent e)
                {
                        Object [] selectedItems = methodList.getSelectedValues ();
                        int i;

                        for (i=0;i<selectedItems.length;i++)
                        {
                                methodsVector.removeElement(selectedItems[i]);
                                methodList.updateUI();

                                frameMethCV.removeElement(selectedItems[i]);

                        }
                }
        });


        // Add listener to the parametre button
        parametreButton.addActionListener (new ActionListener ()
        {
                /**
                * to add parameters to a method
                * the parameter frame appears (AttributeFrame)
                *
                * @param e the action event
                * @see AttributeFrame
                * @since JDK1.3
                *
                */
                public void actionPerformed (ActionEvent e)
                {
                        AttributeFrame attrib = new
AttributeFrame(parametreDescription,"parametre");
                }
        });

        // Add listener to the add button
        addButton.addActionListener (new ActionListener ()
        {
```

```
                        /**
                        * To add methods to the description
                        * The selected methods are first added to the methodList
                        * and if the user clicks OK, the selected methods are added
                        * to the methods' vector in class MethodsDescription (else
                        * nothing happens).
                        *
                        * @param e the action event
                        *
                        * @see MethodsDescription
                        * @see MethodC
                        *
                        * @since JDK1.3
                        *
                        * @Post created methods are added to the description
                        *
                        */
                        public void actionPerformed (ActionEvent e)
                        {


                                String desc = new String();

                                for (int i=0;i<tmpModifierVector.size();i++)
                                {
                                        desc =desc.concat((String)tmpModifierVector.elementAt(i));
                                        desc = desc.concat(" ");
                                }

                                desc = desc.concat((typeList.getSelectedItem()).toString());
                                desc = desc.concat(" ");
                                desc = desc.concat(methName.getText());
                                desc = desc.concat("(");


                                for (int i=0;i<parametreDescription.getStringVector().size();i++)
                                {
                                        desc
=desc.concat((String)parametreDescription.getStringVector().elementAt(i));
                                        desc = desc.concat(" ");
                                        if (i+1 < parametreDescription.getStringVector().size())
                                         desc = desc.concat(", ");
                                }



                                desc = desc.concat(")");
                                methodsVector.addElement (desc);

                                methodList.updateUI();

                                //*********
                                MethodC method = new
MethodC(methName.getText(),(typeList.getSelectedItem()).toString(),tmpModifierVector,parametreDescription.ge
tAttributCVector());

                                frameMethCV.addElement (method);

                                parametreDescription = new AttributsDescription ();
                                setJcomboBoxModifierText(new Vector());
                        }
                });

                // Add listener to the cancel button
                cancelButton.addActionListener (new ActionListener ()
                {
                        /**
```

```
                        * All the changes the user made
                        * are not treated. The method frame
                        * is just closed.
                        *
                        * @param e the action event
                        *
                        * @since JDK1.3
                        *
                        */
                        public void actionPerformed (ActionEvent e)
                        {
                                setVisible (false);
                        }
                });

                // Add listener to the ok button
                okButton.addActionListener (new ActionListener ()
                {
                        public void actionPerformed (ActionEvent e)
                        {
                                /**
                                * All the changes the user made
                                * are treated.
                                * The methods are added or deleted.
                                * The method frame is closed.
                                *
                                * @param e the action event
                                *
                                * @see MethodsDescription
                                *
                                * @since JDK1.3
                                */

                                methodsDescription.changeMethodC(frameMethCV,methodsVector);
                                setVisible (false);
                        }
                });
        }

        /**
        * Set the text in the modifier combo box by going through
        * the vector
        *
        * @param stringVector the Vector to cross
        *
        * @see ModifierFrame#okButton.addActionListener
        *
        * @since JDK1.3
        */
        public static void setJcomboBoxModifierText(Vector stringVector)
        {
                modifierJcomboBox.removeAllItems();

                for (int i=0;i<stringVector.size();i++)
                        modifierJcomboBox.addItem((String)stringVector.elementAt(i));


                tmpModifierVector=stringVector;
        }
}
```

15. Classe MethodsDescription

```
/**
* Class MethodsDescription
*
* Draw the description's methods'
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* @Pre OFLPanel and ClassDescription have been instancied
*
* @Post the panel which represents the description's methods is created
*
*/

import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.Insets;
import javax.swing.JLabel;
import javax.swing.JPanel;
import java.util.Vector;

public class MethodsDescription extends JPanel
{
	private Vector methodCVector;
	private Vector stringVector;
	private JLabel desc = new JLabel ();

	/**
	* Creates 2 Vectors : one which will contain the string representation
	* of the methods (stringVector), and the other one which will contain
	* every informations about a method...
	*
	* @since JDK1.3
	*/
	public MethodsDescription ()
	{

		methodCVector = new Vector();
		stringVector  = new Vector();
	}

	/**
	* Sets the layout and the background color of the panel.
	* Adds the methods to the panel by going trough stringVector.
	* If it's empty than a "" is added to the panel,
	* else each method is added.
	*
	* @since JDK1.3
	*
	* @Pre it's a new description. The description is created and not modified
	*/
	public void initMethodsDescription ()
	{
		removeAll();

		GridBagLayout gridbag = new GridBagLayout();
      GridBagConstraints c = new GridBagConstraints();

		// Set the frame background color
		setBackground (Color.white);
```

```
		// Set the frame layout
setLayout (gridbag);

// Set the constraints properties
    c.anchor = GridBagConstraints.WEST;
		c.gridwidth = GridBagConstraints.REMAINDER;
		c.insets = new Insets (0,5,0,5);

		// Set  text
		if(stringVector.size()==0)
		{
			desc.setText(" ");
			gridbag.setConstraints(desc, c);
			add(desc);
			//nb=0;
		}
		else
		{
			for(int i=0;i<stringVector.size();i++)
			{
				if(i==stringVector.size()-1)
					c.insets = new Insets (0,5,5,5);

				desc = addMethod((stringVector.elementAt(i)).toString());
			    gridbag.setConstraints(desc, c);
				add(desc);
			}
		}
}

/**
* Adds a method to a Vector
*
* @param meth the method to add
*
* @return descMeth JLabel to be added to the Vector
*
* @since JDK1.3
*
* @Pre methods have been defined
*
* @Post methods are added to the panel
*
*/
public JLabel addMethod(String meth)
{
		JLabel descMeth = new JLabel ();
		descMeth.setText(meth);
		return  descMeth;
}


/**
* @return methodCVector return a vector which contains all the methods created
*
* @since JDK1.3
*
* @Pre methods have been created
*/
public Vector getMethodCVector()
{
		return methodCVector;
}

/**
* @return stringVector retrun the string representation of a method
*
```

```
	* @since JDK1.3
	*
	* @Pre stringVector exists
	*/
	public Vector getStringVector()
	{
		return stringVector;
	}

	/**
	* if a method has been modified, change methodCVector and stringVector
	*
	* @param vector
	*	stringVector
	*
	* @since JDK1.3
	*/
	public void changeMethodC(Vector vector,Vector stringvector)
	{
		methodCVector =(Vector)vector.clone();
		stringVector  =(Vector)stringvector.clone();
	}
}
```

16. Classe ModifierFrame

```
/**
* Class ModifierFrame
*
* Creates the frmae to select the modifiers for the
* attributes, the methods
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* @Pre the modifier button in the description frame has been clicked (DescriptionFrame)
*
* @Post the description's modifier(s) can be choose
*
*/

import java.awt.BorderLayout;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.Insets;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import java.util.Vector;
import java.awt.Dimension;
import javax.swing.JList;
import javax.swing.JScrollPane;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ModifierFrame  extends JFrame
{

	private Vector tmpVector= new Vector();
	private Vector retVector= new Vector();

	/**
	* Creates a frame with 2 scroll panels and 2 JList in them.
	* One JList (list) displays every modifiers (LoadObject.getModifierVector()).
	* The other one (personalList) displays the ones which we have selected and
	* are written in a vector (tmpVector).
	*
	* It creates four others button :
	*      - ">>" button : to add a modifier
	*      - "<<" button : to delete a modifier
	*      - "OK" button
	*      - "CANCEL" button
	*
	* @param modifierVector the vector where the modifier are saved and loaded
	* @param classString to know if it's an attribut, a method or a
	*        description modifier
	*
	* @since JDK1.3
	*
	* @Pre the modifier button in the description frame has been clicked (DescriptionFrame)
	*
	* @Post the description's modifier(s) can be choose
	*
	*/
	public ModifierFrame(Vector modifierVector,final String classString)
```

```
    {
            JPanel modifierPane = new JPanel ();
            JPanel button_AD_Pane = new JPanel ();
            JPanel button_OC_Pane = new JPanel ();

            retVector=(Vector)modifierVector;


            tmpVector=(Vector)modifierVector.clone();

            final JList list = new JList ((Vector)LoadObject.getModifierVector().clone());
            final JList personalList = new JList (tmpVector);

            JScrollPane scroll = new JScrollPane(list);
            scroll.setPreferredSize(new Dimension (150,200));

            JScrollPane personalscroll = new JScrollPane(personalList);
            personalscroll.setPreferredSize(new Dimension (150,200));

            JButton addButton   = new JButton(">>");
            JButton deleteButton= new JButton("<<");

        JButton okButton     = new JButton("  OK  ");
            JButton cancelButton = new JButton("Cancel");

            GridBagLayout gridbag = new GridBagLayout();
GridBagConstraints constraint = new GridBagConstraints();

            getContentPane().setLayout (gridbag);
            modifierPane.setLayout (gridbag);
            button_AD_Pane.setLayout (gridbag);
            button_OC_Pane.setLayout (gridbag);

            button_AD_Pane.setBorder(BorderFactory.createEtchedBorder());

            //set button_AD_Pane
            // add addButton to button_AD_Pane
            constraint.anchor = GridBagConstraints.WEST;
            constraint.gridwidth = GridBagConstraints.REMAINDER;
            constraint.insets = new Insets (20,5,20,5);
            gridbag.setConstraints(addButton, constraint);
            button_AD_Pane.add (addButton);

            // add deleteButton to button_AD_Pane
            constraint.insets = new Insets (10,5,20,5);
            gridbag.setConstraints(deleteButton, constraint);
            button_AD_Pane.add (deleteButton);


            // add scroll to modifierPane
            constraint.anchor = GridBagConstraints.WEST;
            constraint.gridwidth = GridBagConstraints.RELATIVE;
            constraint.insets = new Insets (20,10,20,270);
            gridbag.setConstraints(scroll, constraint);
            modifierPane.add (scroll);

            // add scroll to modifierPane
            constraint.insets = new Insets (20,-245,20,0);
            constraint.gridwidth = GridBagConstraints.RELATIVE;
            gridbag.setConstraints(button_AD_Pane, constraint);
            modifierPane.add (button_AD_Pane);


            // add personalscroll to modifierPane
            constraint.gridwidth = GridBagConstraints.REMAINDER;
            constraint.insets = new Insets (0,-160,0,0);
            gridbag.setConstraints(personalscroll, constraint);
```

```
modifierPane.add (personalscroll);


//set button_OC_Pane
constraint.anchor = GridBagConstraints.WEST;
constraint.gridwidth = GridBagConstraints.RELATIVE;
constraint.insets = new Insets (0,50,10,10);
gridbag.setConstraints(okButton, constraint);
button_OC_Pane.add (okButton);

constraint.gridwidth = GridBagConstraints.REMAINDER;
constraint.insets = new Insets (0,90,10,0);
gridbag.setConstraints(cancelButton, constraint);
button_OC_Pane.add (cancelButton);


constraint.anchor = GridBagConstraints.WEST;
constraint.gridwidth = GridBagConstraints.REMAINDER;
constraint.insets = new Insets (0,0,0,0);

gridbag.setConstraints(modifierPane, constraint);
getContentPane().add(modifierPane);


constraint.insets = new Insets (0,53,0,13);
gridbag.setConstraints(button_OC_Pane, constraint);
getContentPane().add(button_OC_Pane);

// set frame title, size
setTitle (" Choose Modifier ");
setSize(new Dimension (445,305));

// the frame can't be resized
setResizable (false);

// set the frame visible
setVisible (true);

// Add listener to the cancel button
cancelButton.addActionListener (new ActionListener ()
{
	/**
	* All the changes the user made
	* are not treated. The method frame
	* is just closed.
	*
	* @param e the action event
	*
	* @since JDK1.3
	*
	*/
	public void actionPerformed (ActionEvent e)
	{
		setVisible (false);
	}
});

// Add listener to the ok button
okButton.addActionListener (new ActionListener ()
{
	/**
	* If ModifierFrame has been opened for a description
	* it changes the modifiers in the JCombobox in DescriptionFrame.
	* If ModifierFrame has been opened for attributs
	* it changes the modifiers in the JCombobox in AttributeFrame.
	*
	* The frame is then closed
```

```
		*
		* @param e the action event
		*
		* @since JDK1.3
		*/
		public void actionPerformed (ActionEvent e)
		{
			retVector=(Vector)tmpVector.clone();

			if(classString.equals("description"))
				// init modifier Jcombobox of DescriptionFrame
				DescriptionFrame.setJcomboBoxModifierText(retVector);
			else if(classString.equals("Attribut"))
			{
				// init modifier Jcombobox of AttributeFrame
				AttributeFrame.setJcomboBoxModifierText(retVector);
			}
			else if(classString.equals("Method"))
				MethodFrame.setJcomboBoxModifierText(retVector);

			setVisible (false);
		}
});

// Add listener to the add button
addButton.addActionListener (new ActionListener ()
{
		/**
		* To add modifiers : they are added in the
		* personalList and in the vector
		*
		* @param e the action event
		*
		* @since JDK1.3
		*
		* @Pre modifiers has been choosen
		*
		* @Post choosen modifiers are added in the personalList
		*
		*/
		public void actionPerformed (ActionEvent e)
		{
			Object [] selectedItems = list.getSelectedValues ();

			for (int i=0;i<selectedItems.length;i++)
			{
				if(tmpVector.contains(selectedItems[i])==false)
				{
					tmpVector.addElement(selectedItems[i]);
					personalList.updateUI();
				}
			}
		}
});

// Add listener to the delete button
deleteButton.addActionListener (new ActionListener ()
{
		/**
		* To delete modifiers : they are deleted from the
		* personalList and from the vector
		*
		* @param e the action event
		*
		* @since JDK1.3
		*
		* @Pre modifier to be deleted are selected
```

```
                *
                * @Post modifiers are deleted
                *
                */
                public void actionPerformed (ActionEvent e)
                {
                        Object [] selectedItems = personalList.getSelectedValues ();


                        for (int i=0;i<selectedItems.length;i++)
                        {
                                tmpVector.removeElement(selectedItems[i]);
                                personalList.updateUI();
                        }
                }
        });
    }
}
```

17. Classe NameDescription

```
/**
* Class NameDescription
*
* Handles the description drawing
*
* String nameD : for the description name
* String typeD : for the description type
* String modifierD : concatenation of every string in stringVector for the display
* Vector stringVector : contains the differents modifiers
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* @Pre OFLPanel and ClassDescription have been instancied
*
* @Post the panel which represents the description's name is created
*
*/

import java.awt.Font;
import java.awt.Color;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.Insets;
import javax.swing.JLabel;
import javax.swing.JPanel;
import java.awt.Dimension;
import java.util.Vector;

public class NameDescription extends JPanel
{
	private	String descriptionName;
	private	String descriptionType;
	private	String descriptionModifier;
	private	Vector descriptionModifierVector ;

	private	JLabel descriptionNameLabel = new JLabel ();
	private	JLabel descriptionTypeLabel = new JLabel ();
	private	JLabel descriptionModifierLabel = new JLabel ();

	/**
	* Sets the default text for the description's name, type and modifier
	*
	* @since JDK1.3
	*
	* @Pre it's a new description. The descripition is created and not modified
	*/
	public NameDescription()
	{
		descriptionName ="DescriptionName";
		descriptionType = Nodes.getNodeSelect();
		descriptionModifier =" ";
		descriptionModifierVector= new Vector();
	}

	/**
	* For a new description
	* Set the class display properties : her layout, her background color,
	* and add the labels (descName,descType,descModifier) to it.
	*
	* @param name the description name
	* @param type the description type
	* @param modif the description modifier
```

```
        *
        * @since JDK1.3
        *
        * @Pre it's a new description. The description is created and not modified
        */
        public void initNameDescription (String name,String type,String modif)
        {
                GridBagLayout gridbag = new GridBagLayout();
    GridBagConstraints c = new GridBagConstraints();


                descriptionName =name;
                descriptionType=type;

                if(descriptionModifierVector.size()==0)
                        descriptionModifier =" ";
                else
                        descriptionModifier=modif;

    setBackground (Color.white);

                // Set the frame layout
        setLayout (gridbag);
        c.anchor = GridBagConstraints.WEST;
                c.gridwidth = GridBagConstraints.REMAINDER;

                // Set the default text for the description name, type and modifier
                descriptionNameLabel.setText(descriptionName);
                descriptionNameLabel.setFont(new
Font(descriptionNameLabel.getFont().getName(),Font.BOLD,descriptionNameLabel.getFont().getSize()));

                descriptionTypeLabel.setText(descriptionType);
                descriptionModifierLabel.setText(descriptionModifier);

                // Add descName to the gridbag constraint and to the panel
                c.insets = new Insets (0,0,0,5);

                // Add descModifier to the gridbag constraint and to the panel
                c.insets = new Insets (0,15,0,5);
        }

        /**
        * Sets the description's name, type and modifier
        * if the description already exists
        *
        * @param name the description name
        * @param type the description type
        * @param modif the description modifier
        *
        * @since JDK1.3
        *
        * @Pre the description has been modified and the display of the descripiton is
        *      modified to
        *
        * @Post the name, the type and the mofifiers of the description are updated
        *
        */
        public void reinitNameDescription (String name,String type,String modif)
        {

                descriptionName =name;
                descriptionType=type;

                if(descriptionModifierVector.size()==0)
                        descriptionModifier =" ";
                else
                        descriptionModifier=modif;
```

```
		descriptionNameLabel.setText(descriptionName);
		descriptionTypeLabel.setText(descriptionType);


		descriptionModifierLabel.setText(descriptionModifier);
}

/**
* @return descriptionName return a String which represents the description name
*
*
* @since JDK1.3
*
* @Pre descriptionName exists
*/
public String getNameString()
{
		return descriptionName;
}

/**
* @return descriptionType return a String which represents the description type
*
* @since JDK1.3
*
* @Pre descriptionType exists
*/
public String getType()
{
		return descriptionType;
}

/**
* @return descriptionModifier return a String which represents the description modifier
*
*
* @since JDK1.3
*
* @Pre descriptionModifier exists
*/
public String getModifierString()
{
		return  descriptionModifier;
}

/**
* @return descriptionModifierVector return the Modifier Vector
*
* @since JDK1.3
*
* @Pre descriptionModifierVector exists
*/
public Vector getModifierVector()
{
		return descriptionModifierVector;
}


/**
* @return descriptionTypeLabel return the JLabel which represents the description type
*
* @since JDK1.3
*/
public JLabel getTypeLabel()
{
```

```
		return descriptionTypeLabel;
	}

	/**
	* @return descriptionNameLabel return the JLabel which represents the description name
	*
	* @since JDK1.3
	*/
	public JLabel getNameLabel()
	{
		return descriptionNameLabel;
	}

	/**
	* @return descriptionModifierLabel return the JLabel which represents the description modifier
	*
	* @since JDK1.3
	*/
	public JLabel getModifierLabel()
	{
		return descriptionModifierLabel;
	}

	/**
	* Create a string whiwh represents the description's modifiers
	* according to the descriptionModifierVector vector
	*
	* @since JDK1.3
	*/
	public void buildString()
	{
		String desc = new String("");

		for (int i=0;i<descriptionModifierVector.size();i++)
		{
			desc =desc.concat((String)descriptionModifierVector.elementAt(i));
			desc = desc.concat(" ");
		}

		// init modifier of namedescription
		descriptionModifier= desc;

	}

}
```

18. Classe Node

```
/**
* Class Node
*
* Creates nodes
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* @Pre Nodes has been instancied
*
* @Post Node class is created
*
*/

import javax.swing.tree.DefaultMutableTreeNode;
import java.lang.Object;

public class Node extends DefaultMutableTreeNode
{
	/**
	* @param nodeName the node's name
	*
	* @see DefaultMutableTreeNode#setUserObject(Object)
	*
	* @since JDK1.3
	*/
	public Node (Object nodeName)
	{
	  setUserObject (nodeName);
	}
}
```

19. Classe Nodes

```
/**
* Class Nodes
*
* Creates a tree and handles the events which can append on
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*/

import java.awt.Color;
import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.Point;
import java.awt.Toolkit;
import java.awt.event.MouseEvent;
import java.awt.event.InputEvent;
import javax.swing.JPanel;
import javax.swing.JTree;
import javax.swing.tree.TreeNode;
import javax.swing.event.MouseInputAdapter;
import javax.swing.tree.TreeSelectionModel;
import java.awt.event.MouseListener;
import java.util.Vector;

public class Nodes implements MouseListener
{
	private static JTree languageTree;
	private static String boolCursor;
	private static Node root = new Node ("Language : " + LoadObject.getNameLanguage());
	private static Node newDescription = new Node ("Descriptions");

	/*
	* Creates a JTree, 3 main Node and add them in the JTree.
	* The other Node are be dynamically added (LoadObject)
	* The JTree is displayed unwind, and only one Node can be selected
	* at a time.
	* The JTree also handles mouse events which append on
	*
	* @see JTree
	* @see LoadObject
	*
	* @since JDK1.3
	*
	* @Pre ExplorerPanel has been instancied
	*
	* @Post a tree which represents the langauge component (root of thr tree),
	*       the descripitons, ant the relationships components is created
	*
	*/
	public Nodes ()
	{
		int i=0;

		// set the name of the language
		Node root = new Node ("Language : " + LoadObject.getNameLanguage());

		Node oflDesc = new Node ("Descriptions-Components");
		Node oflRel = new Node ("Relationships-Components");

		root.add (oflDesc);

		// Add nodes to the tree by crossing a vector
		for (i=0;i<LoadObject.getDescriptionTypeVector().size();i++)
```

```
		{
			Node tmp = new Node (LoadObject.getDescriptionTypeVector().elementAt(i));
			oflDesc.add (tmp);
		}


		Node oflUse = new Node ("Use");
		for (i=0;i<LoadObject.getDescriptionUseVector().size();i++)
		{
			Node tmp1 = new Node (LoadObject.getDescriptionUseVector().elementAt(i));
			oflUse.add (tmp1);
		}

		Node oflImport = new Node ("Import");
		for (i=0;i<LoadObject.getDescriptionImportVector().size();i++)
		{
			Node tmp2 = new Node (LoadObject.getDescriptionImportVector().elementAt(i));
			oflImport.add (tmp2);
		}

		languageTree = new JTree(root);

		// construct the tree
		oflRel.add (oflUse);
		oflRel.add (oflImport);

		root.add (oflRel);
		root.add (newDescription);

		// the tree will be displayed unwinded
		languageTree.setLargeModel(true);

		// only one node can be selected at the same time
		languageTree.getSelectionModel().setSelectionMode
(TreeSelectionModel.SINGLE_TREE_SELECTION);

		// add a treelistener
		languageTree.addMouseListener(new TreeListener());
	}

	/**
	* Intern classe which manages the mouse events
	*
	* @Pre Nodes has been instancied
	*
	* @Post Nodes classes handles the mouse events whiwh append on
	*
	*/
	class TreeListener extends MouseInputAdapter
	{
		private Node node;
		private Object nodeInfo;


		public void mouseClicked (MouseEvent e) {}
		public void mouseDragged (MouseEvent e){}
		public void mouseReleased (MouseEvent e){}

		/**
		* when the mouse is pressed, calss mousePressedOrClicked method
		*
		* @param e the MouseEvent
		*/
		public void mousePressed (MouseEvent e)
		{
			mousePressedOrClicked( e);
		}
```

```
		/**
		* when the mouse is entered, if boolCursor equals "",
		* set the cursor to default
		*
		* @param the MouseEvent
		*/
		public void mouseEntered (MouseEvent e)
		{
			if(boolCursor=="")
			{
				languageTree.setCursor(new Cursor (Cursor.DEFAULT_CURSOR));
			}
		}

		/**
		* If a leaf node is selected, creates a new mouse pointer like a
		* description representation.
		* boolCursor is true in order to have the same mouse pointer
		* in ExplorerPanel, and OFLPanel.
		*
		* @param e		the event that occurs on the mouse
		*
		* @see OFLPanel#setCursorTrue()
		*
		* @since JDK1.3
		*
		* @Pre the mouse has been pressed
		*
		* @Post the cursor is changed
		*
		*/
		private void mousePressedOrClicked(MouseEvent e)
		{
			if ((e.getModifiers() & InputEvent.BUTTON1_MASK) != 0)
			{
				node = (Node) languageTree.getLastSelectedPathComponent();

			    if (node == null)
				return;

			    if ((node.isLeaf()) && (getNodeFolder()!="Use") && (getNodeFolder()!="Import"))
			    {
				node.getUserObject();

				Cursor c = languageTree.getToolkit().createCustomCursor
(languageTree.getToolkit().createImage ("Description.gif"),new Point (),"");

				languageTree.setCursor (c);
				OFLPanel.setCursor("Description");
				boolCursor = "Description";
			    }

			    if ((node.isLeaf()) && (getNodeFolder()=="Use"))
			    {
				node.getUserObject();

				Cursor c = languageTree.getToolkit().createCustomCursor
(languageTree.getToolkit().createImage ("FirstUse.gif"),new Point (),"");

				languageTree.setCursor (c);
				///languageTree.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
				OFLPanel.setCursor("Use");
				boolCursor = "Use";
			    }

			    if ((node.isLeaf()) && (getNodeFolder()=="Import"))
```

```
                                {
                                        node.getUserObject();

                                        Cursor c = languageTree.getToolkit().createCustomCursor
(languageTree.getToolkit().createImage ("FirstImport.gif"),new Point (),"");

                                        languageTree.setCursor (c);
                                        //languageTree.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
                                        OFLPanel.setCursor("Import");
                                        boolCursor = "Import";
                                }
                        }
                        else if(boolCursor != "")
                        {
                                boolCursor = "";
                                OFLPanel.setCursor("");
                                languageTree.setCursor(new Cursor (Cursor.DEFAULT_CURSOR));
                        }
                }
        }

        /**
        * When the mouse is entered, if boolCursor is false
        * set the cursor to default
        *
        * @param e        the event that occurs on the mouse
        */
        public void mouseEntered (MouseEvent e)
        {
                if(boolCursor=="")
                {
                        languageTree.setCursor(new Cursor (Cursor.DEFAULT_CURSOR));
                }
        }

        public void mouseReleased (MouseEvent e){}
        public void mousePressed (MouseEvent e){}
        public void mouseClicked (MouseEvent e) {}
        public void mouseExited (MouseEvent e){}


        /**
        * Gets the node name on which we have clicked
        *
        * @return the node name on which we have clicked
        *
        * @Pre a node has been selected
        */
        public static String getNodeSelect()
        {
                Node node;
                node = (Node) languageTree.getLastSelectedPathComponent();
           return (String) node.getUserObject();
        }

        /**
        * Retrieve the name's parent folder of the selected node
        *
        * @return the parent folder
        *
        * @Pre a node has been selected
        */
        public static String getNodeFolder()
        {
                Node node;
                TreeNode parent;
```

```
		node = (Node) languageTree.getLastSelectedPathComponent();
		parent = node.getParent();

		return parent.toString();

}

/**
* set bool cursor to ""
*
*/
public static void setCursorFalse()
{
		boolCursor = "";
}

/**
* @return languageTree (it's the tree which has been created)
*/
public JTree getLanguageTree()
{
		return languageTree;
}


/**
* @param nameClass the name of the no de to add to the new description folder
*
* @Pre a new descripion has been created
*
* @Post the new description is added to the descriptions' folder
*
*/
public static void addDescriptionTree(String nameClass)
{
		newDescription.add (new Node (nameClass));

		languageTree.updateUI();
}

/**
* @Pre a description has been deleted
*
* @Post the description's name is took off the description's folder
*
*/
public static void deleteDescriptionTree()
{
		newDescription.remove (newDescription.getLastLeaf());

		languageTree.updateUI();
}

/**
* @param index the index of the descripiton in the descripion node
*	newname the newname of the descripition
*
* @Pre a description's name has been changed
*
* @Post the descriptions' folder is updated
*
*/
public static void changeDescriptionTree(int index,String newname)
{
		newDescription.remove(index);
		languageTree.updateUI();
		newDescription.insert(new Node (newname),index);
```

```
                languageTree.updateUI();
            }
        }
```

20. Classe OFLDescriptionPanel

```
/**
* Class OFLDescriptionPanel
*
* Creates the panel where the descritpion will be drawn
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* @Pre A Window class object has been created
*
* @Post OFLPanel class object is created
*
*/

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.Insets;
import javax.swing.JPanel;
import javax.swing.JScrollPane;


public class OFLDescriptionPanel extends JPanel
{
	private static OFLPanel paneArea = new OFLPanel ();

	/**
	* creates a scrollpanel with a OFLPanel in it,
	* sets his size and his layout
	*
	* @see JScrollPane
	* @see OFLPanel
	* @see GridBagLayout
	* @see GridBagConstraints
	*
	* @since JDK1.3
	*/
	public OFLDescriptionPanel ()
	{
		GridBagLayout gridbag = new GridBagLayout();
		GridBagConstraints c = new GridBagConstraints();

		JScrollPane scroller = new JScrollPane(paneArea);

      scroller.setPreferredSize(new Dimension(500,500));

	c.insets = new Insets (0,0,50,50);
	c.anchor = GridBagConstraints.NORTHWEST;
		c.gridwidth = GridBagConstraints.REMAINDER;

      setLayout(new BorderLayout());

      add(scroller,BorderLayout.CENTER);
	}

	/*
	* Adds a ClassDescription to the OFLPanel
	* at x-coordinate x and y-coordinate y, and sets her bounds
	* The coordinates are retrieved from OFLPanel
	*
	* @parameters desc the ClassDescription to add
	*
```

```
	* @see OFLPanel#getPanelX()
	* @see OFLPanel#getPanelY()
	* @see ClassDescription#getLargeur()
	* @see ClassDescription#getHauteur()
	*
	* @since JDK1.3
	*
	* @Pre a ClassDescription's object has been created
	*
	* @Post a new description is drawn/added in the OFLPanel
	*
	*/
	static public  void addDescription(ClassDescription desc)
	{
		desc.setPositionX(paneArea.getPanelX());
		desc.setPositionY(paneArea.getPanelY());


	desc.setBounds(paneArea.getPanelX(),paneArea.getPanelY(),desc.getLargeur(),desc.getHauteur());

		paneArea.add (desc);

		paneArea.updateUI();
	}

	/*
	* Adds a DescriptionRelation to the OFLPanel
	* at x-coordinate x and y-coordinate y.
	*
	* @parameters desc the DescriptionRelation to add
	*
	* @see OFLPanel#getPanelX()
	* @see OFLPanel#getPanelY()
	* @since JDK1.3
	*
	* @Pre a DescriptionRelation's object has been created
	*
	* @Post a new relation is drawn/added in the OFLPanel
	*
	*/
	public static void addRelation (DescriptionRelation rel)
	{
		rel.setBounds(paneArea.getPanelX(),paneArea.getPanelY(),200,200);

		paneArea.add (rel);
		paneArea.updateUI();
	}

	/**
	* @param desc : the description to be updated
	*
	* @Pre a description has been modified
	*
	* @Post the display area is updated
	*
	*/
	public static void update(ClassDescription desc)
	{

	desc.setBounds(paneArea.getPanelX(),paneArea.getPanelY(),desc.getLargeur(),desc.getHauteur());

		paneArea.updateUI();
	}


}
```

21. Classe OFLPanel

```
/**
* Class OFLPanel
*
* Creates the panel where the description will be drawn
* and handles the mouse events which happen on description.
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* @Pre A OFLDescriptionPanel class object has been created
*
* @Post OFLPanel is created, descripiton's drawing
*       and events on description can be handle
*
*/
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Cursor;
import java.awt.Point;
import java.awt.Dimension;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.Insets;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.event.MouseEvent;
import java.awt.event.InputEvent;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import java.util.Vector;
import java.lang.Object;


public class OFLPanel extends JPanel implements MouseListener, MouseMotionListener
{

        private boolean    press = false;
        private int ecartX=0,ecartY=0,vectorPlace=0,classPlace1,classPlace2;
        private int X,Y,X1,Y1;

        private static JPanel panetest = new JPanel ();
        private static boolean boolCursor;
        private static String enterCursor;
        private static boolean firstClickRelCursor=false;

        /**
        * Creates a panel, sets his size, his layout and his background color
        * Handles mouse listener and mouse motion listener
        *
        * @see  MouseListener
        * @see  MouseMotionListener
        * @see  JPanel
        *
        * @since JDK1.3
        *
        */
        public OFLPanel()
        {
                setPreferredSize(new Dimension(1500,1500));

                setBackground (Color.white);
```

```
                setLayout (null);

                addMouseListener (this) ;
                addMouseMotionListener (this) ;

        }

        /**
        * When the mouse left button is entered, if boolCursor is true,
        * creates a new mouse pointer like a description or a
        * relation representation.
        *
        * @param e        the event that occurs on the mouse
        *
        * @see Cursor
        *
        * @since JDK1.3
        *
        * @Pre the mouse has been entered first in the ExplorerPanel
        *
        */
        public void mouseEntered (MouseEvent e)
        {
                if(enterCursor=="Description")
                {
                        Cursor c = getToolkit().createCustomCursor (getToolkit().createImage
("Description.gif"),new Point (),"");

                        setCursor (c);

                }
                else if (enterCursor=="Use")
                {
                        Cursor c = getToolkit().createCustomCursor (getToolkit().createImage
("FirstUse.gif"),new Point (),"");
                        setCursor (c);
                }
                else if (enterCursor=="Import")
                {
                        Cursor c = getToolkit().createCustomCursor (getToolkit().createImage
("FirstImport.gif"),new Point (),"");
                        setCursor (c);
                }
                else
                {
                        setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
                }
        }

        /**
        * When the mouse left button is pressed, if boolCursor is true,
        * creates NameDescription, AttributsDescription, MethodsDescription, ClassDescription,
        * DescriptionFrame.
        * Sets the mouse pointer to default, and gets the x and y coordinate from where we clicked.
        * These are the ClassDescription coordinates (where it will be drawn/displayed on the
        * OFLPanel).
        *
        * @param e        the event that occurs on the mouse
        *
        * @see NameDescription
        * @see AttributsDescription
        * @see MethodsDescription
        * @see ClassDescription
        * @see DescriptionFrame
        * @see Nodes
        *
        * @since JDK1.3
```

```
	*
	* @Pre enterCursor must be equal to "Description"
	*      or "Use" or "Import" and the left mouse button must be clicked
	*
	* @Post NameDescription, AttributsDescription, MethodsDescription,
	*       ClassDescription and DescriptionFrame are created.
	*       The window to create description appears.
	*
	*/
	public void mouseClicked (MouseEvent e)
	{

		if(enterCursor=="Description"&& (e.getModifiers() & InputEvent.BUTTON1_MASK) != 0)
		{
			X=e.getX()-ecartX;
			Y=e.getY()-ecartY;

			boolCursor = false;
			enterCursor = "";
			Nodes.setCursorFalse();

			NameDescription nameDescription = new NameDescription ();
			AttributsDescription attributDescription = new AttributsDescription();
			MethodsDescription methodsDescription = new MethodsDescription();
			ClassDescription classDescr = new ClassDescription
(nameDescription,attributDescription,methodsDescription);

			DescriptionFrame desc;

			desc = new DescriptionFrame (classDescr,0,true);
			setCursor (new Cursor (Cursor.DEFAULT_CURSOR));
		}
		// For the relations
		else if ((enterCursor=="Use") || (enterCursor=="Import"))
		{
			for(int j=0;j<Main.mainSizeVector();j++)
			{
				if(((e.getX() > Main.mainEltAt(j).getPositionX() ) && (e.getX() <
Main.mainEltAt(j).getPositionXmax()))
				&& ((e.getY() > Main.mainEltAt(j).getPositionY() )&& (e.getY() <
Main.mainEltAt(j).getPositionYmax())))
				{
					if (firstClickRelCursor == false)
					{
						firstClickRelCursor = true;
						Cursor c = getToolkit().createCustomCursor
(getToolkit().createImage ("SecondUse.gif"),new Point (),"");
						setCursor (c);

						X1 = Main.mainEltAt(j).getPositionXmax();
						Y1 = e.getY();

						classPlace1 = j;
					}
					else
					{
						firstClickRelCursor = false;
						classPlace2 = j;
						RelationFrame relFrame = new RelationFrame
(Main.mainEltAt(classPlace1),Main.mainEltAt(classPlace2));
					}
				}
			}
		}
	}

	/**
```

```
	* When the left mouse button (press is true) is released,
	* get the x and y coordinate, from where we released the mouse
	* and put in ClassDescription (which is in the vector (class Main) at the indice
	* vectorPlace) these coordinates.
	* Sets the mouse pointer to default.
	*
	* @param e		the event that occurs on the mouse
	*
	* @see Main#mainEltAt(int)
	*
	* @since JDK1.3
	*
	* @Pre the left mouse button has been clicked
	*
	* @Post the new description X and Y are added in a vector
	*
	*/
	public void mouseReleased (MouseEvent e)
	{
		if(press == true)
		{
			press = false;
			Main.mainEltAt(vectorPlace).setPositionX(e.getX()-ecartX);
		   Main.mainEltAt(vectorPlace).setPositionY(e.getY()-ecartY);
			setCursor(new Cursor ( Cursor.DEFAULT_CURSOR ));
			vectorPlace=0;
		}
	}



	/**
	* When the left mouse button (press is true) is released,
	* get the x and y coordinate, from where we released the mouse
	* and put in ClassDescription (which is in the vector (class Main) at the indice
	* vectorPlace) these coordinates.
	* Sets the mouse pointer to default.
	*
	* @param e		the event that occurs on the mouse
	*
	* @see Main#mainEltAt(int)
	*
	* @since JDK1.3
	*
	* @Pre the user has clicked on a description and then dragg the mouse
	*
	* @Post the description is moved and her coordinates are changed in the vector
	*
	*/
	public void mouseDragged (MouseEvent e)
	{
		if(press == true)
		{
			X=e.getX()-ecartX;
			Y=e.getY()-ecartY;


			panetest.setBounds((e.getX()-ecartX),(e.getY()-
ecartY),Main.mainEltAt(vectorPlace).getLargeur(),Main.mainEltAt(vectorPlace).getHauteur());
		}

	}

	/**
	* When the mouse is pressed, verifies if the button pressed
	* is the left one.
	* If so, call leftMousePressed function
```

```
	*
	* @param e		the event that occurs on the mouse
	*
	* @see #leftMousePressed(e)
	*
	* @since JDK1.3
	*
	*/
	public void mousePressed (MouseEvent e)
	{

		if ((e.getModifiers() & InputEvent.BUTTON1_MASK) != 0)
		{
			leftMousePressed (e);
		}
	}

	/**
	* If the mouse pointer is in a description (which is in the vector at element i)
	* and if we are keeping the left mouse pressed, the cursor is changed into move
	* cursor, and the space between the place where we clicked and the description's top left
	* corner is calculated for the cursor to stay at the same place in the descripiton.
	*
	* If we double click in a desctipion, her properties window appears.
	*
	* @param e		the event that occurs on the mouse
	*
	* @see Main#mainEltAt(int)
	* @see Main#mainSizeVector()
	* @see DescriptionFrame
	* @see Cursor
	* @see ClassDescription#getPositionX()
	* @see ClassDescription#getPositionY()
	* @see ClassDescription#getPositionXmax()
	* @see ClassDescription#getPositionYmax()
	*
	* @since JDK1.3
	*
	* @Pre the left mouse button has been pressed
	*
	* @Post the description is moved
	*
	*/
	private void  leftMousePressed (MouseEvent e)
	{
		for(int i=0;i<Main.mainSizeVector();i++)
		{

			if(((e.getX() > Main.mainEltAt(i).getPositionX() ) && (e.getX() <
Main.mainEltAt(i).getPositionXmax()))
			 && ((e.getY() > Main.mainEltAt(i).getPositionY() )&& (e.getY() <
Main.mainEltAt(i).getPositionYmax())))
			{
				ecartX=e.getX()-Main.mainEltAt(i).getPositionX();
				ecartY=e.getY()-Main.mainEltAt(i).getPositionY();
				vectorPlace=i;
				panetest=Main.mainEltAt(i);
				X=e.getX()-ecartX;
				Y=e.getY()-ecartY;


				if(((e.getX() > Main.mainEltAt(i).getPositionX()-1 ) && (e.getX() <
Main.mainEltAt(i).getPositionX()+9))
				&& ((e.getY() > Main.mainEltAt(i).getPositionY()+50 )&& (e.getY() <
Main.mainEltAt(i).getPositionY()+60)))
				{
```

```
                                        Main.mainEltAt(i).setBoolAttributs();
                                        OFLDescriptionPanel.update(Main.mainEltAt(i));

                                }
                                else if(Main.mainEltAt(i).getBoolMethods()== true &&
Main.mainEltAt(i).getBoolAttributs()== true
                                &&(e.getX() > Main.mainEltAt(i).getPositionX()-1  && e.getX() <
Main.mainEltAt(i).getPositionX()+9)
                                && ((e.getY() >
Main.mainEltAt(i).getPositionY()+Main.mainEltAt(i).getValAttrib()-5 )
                                && (e.getY() <
Main.mainEltAt(i).getPositionY()+Main.mainEltAt(i).getValAttrib()+5 )))
                                {

                                        Main.mainEltAt(i).setBoolMethods();
                                        OFLDescriptionPanel.update(Main.mainEltAt(i));
                                }
                                else if((Main.mainEltAt(i).getBoolAttributs()== false ||
(Main.mainEltAt(i).getBoolMethods()== false && Main.mainEltAt(i).getBoolAttributs()== true))
                                &&      ((e.getX() > Main.mainEltAt(i).getPositionX()+9 ) && (e.getX() <
Main.mainEltAt(i).getPositionX()+19))
                                &&      ((e.getX() > Main.mainEltAt(i).getPositionX()+9 ) && (e.getX() <
Main.mainEltAt(i).getPositionX()+19))
                                && ((e.getY() > Main.mainEltAt(i).getPositionY()+50 )&& (e.getY() <
Main.mainEltAt(i).getPositionY()+60)))
                                {
                                        Main.mainEltAt(i).setBoolMethods();
                                        OFLDescriptionPanel.update(Main.mainEltAt(i));
                                }
                                else
                                {
                                        if(press == false)
                                        {
                                                press = true;
                                                setCursor(new Cursor ( Cursor.MOVE_CURSOR ));

                                        }

                                        if(e.getClickCount()==2)
                                        {
                                                DescriptionFrame desc = new DescriptionFrame
(Main.mainEltAt(i),i,false);
                                        }
                                }
                        }
                }
        }


        public void mouseMoved (MouseEvent e){}
        public void mouseExited (MouseEvent e){}

        /**
        * Call in OFLDescriptionPanel
        *
        * @return X ClassDescription x-coordinate
        *
        * @since JDK1.3
        *
        */
        public int getPanelX()
        {
                return X;
        }

        /**
        * Call in OFLDescriptionPanel
```

```
	*
	* @return Y ClassDescription y-coordinate
	*
	* @since JDK1.3
	*
	*/
	public int getPanelY()
	{
		return Y;
	}

	/**
	* set boolCursor to true
	*
	* @since JDK1.3
	*/
	public static void setCursorTrue()
	{
		boolCursor = true;
	}

	/**
	* set boolCursor to false
	*
	* @since JDK1.3
	*/
	public static void setCursorFalse()
	{
		boolCursor = false;
	}

	/**
	* @return boolCursor
	*
	* @since JDK1.3
	*/
	public static boolean getPanelCursor()
	{
		return boolCursor;
	}

	/**
	* set enterCursor
	*
	* @param s the value to set enterCursor
	*		Values can be : "Description", "Use" or "Import"
	*
	* @since JDK1.3
	*/
	public static void setCursor(String s)
	{
		enterCursor = s;
	}
}
```

22. Classe RelationFrame

```
/**
* Class RelationFrame
*
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
* @Pre user selected relations in the tree
*
* @Post relations can be added
*
*/

import java.awt.Font;
import java.awt.Dimension;
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.BorderFactory;
import java.util.Vector;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class RelationFrame extends JFrame
{
	/**
	* creates a panel, sets his size, his layout and his background color
	* handles mouse listener and mouse motion listener
	*
	* allows the user to modify the source and the destination of a relation
	* between 2 descriptions
	*
	* @since JDK1.3
	* @Pre user selected relations in the tree
	*
	* @Post relations can be added
	*/
	public  RelationFrame (final ClassDescription source, final ClassDescription destination)
	{
		int i=0;
		Vector sourceVect = new Vector();
		Vector destVect = new Vector();
		final JComboBox sourceList;
		final JComboBox destList;
		final JComboBox typeList;

		JPanel mainPanel = new JPanel ();
		JPanel buttonPanel = new JPanel ();

		// the name of the source and the destination
		NameDescription sourceName = source.getNameDescription();
		NameDescription destName = destination.getNameDescription();


		JLabel sourceLabel = new JLabel ("Source :         ");
		sourceLabel.setFont(new Font
(source.getFont().getName(),Font.BOLD,source.getFont().getSize()));

		JLabel destinationLabel = new JLabel ("Destination : ");
```

```
                destinationLabel.setFont(new Font
(destination.getFont().getName(),Font.BOLD,destination.getFont().getSize()));

                JLabel typeLabel = new JLabel ("Type :         ");
                typeLabel.setFont(new Font
(typeLabel.getFont().getName(),Font.BOLD,typeLabel.getFont().getSize()));

                JButton okButton = new JButton ("OK");
                JButton cancelButton = new JButton ("Cancel");

                for(i=LoadObject.getNbTypeLoad();i<LoadObject.getTypeVector().size();i++)
                {
                        sourceVect.addElement(LoadObject.getTypeVector().elementAt(i));
                        destVect.addElement(LoadObject.getTypeVector().elementAt(i));
                }

                sourceList = new JComboBox (sourceVect);
                sourceList.setMaximumRowCount(5);
                sourceList.setSelectedItem (sourceName.getNameString());

                destList = new JComboBox (destVect);
                destList.setMaximumRowCount(5);
                destList.setSelectedItem (destName.getNameString());

                typeList = new JComboBox (LoadObject.getDescriptionUseVector());

                // Add elements to the panels
                mainPanel.setLayout(null);
                mainPanel.setBorder(BorderFactory.createEtchedBorder());
                sourceLabel.setBounds (10,10,80,15);
                sourceList.setBounds (100,10,180,20);
                destinationLabel.setBounds (10,50,80,15);
                destList.setBounds (100,50,180,20);
                typeLabel.setBounds (10,90,80,15);
                typeList.setBounds (100,90,180,20);

                mainPanel.add (sourceLabel);
                mainPanel.add (sourceList);
                mainPanel.add (destinationLabel);
                mainPanel.add (destList);
                mainPanel.add (typeLabel);
                mainPanel.add (typeList);

                buttonPanel.add (okButton,BorderLayout.WEST);
                buttonPanel.add (cancelButton,BorderLayout.EAST);

                // Add panels to the frame
                getContentPane().setLayout(null);
                mainPanel.setBounds(10,10,300,115);
                mainPanel.setBorder(BorderFactory.createEtchedBorder());
                buttonPanel.setBounds (60,135,150,35);

                getContentPane().add(mainPanel);
                getContentPane().add(buttonPanel);

                okButton.addActionListener (new ActionListener ()
                {
                        public void actionPerformed (ActionEvent e)
                        {
                                if (!(sourceList.getSelectedItem().equals(destList.getSelectedItem())))
                                {
                                        setVisible (false);
                                        int sourceIndex = (LoadObject.getTypeVector()).indexOf
(sourceList.getSelectedItem());
                                        int destIndex = (LoadObject.getTypeVector()).indexOf
(destList.getSelectedItem());
```

```
                                        OFLDescriptionPanel.addRelation (new DescriptionRelation
(Main.mainEltAt(sourceIndex-LoadObject.getNbTypeLoad()),Main.mainEltAt(destIndex-
LoadObject.getNbTypeLoad())));
                                        }
                                }
                        });

                        cancelButton.addActionListener (new ActionListener ()
                        {
                                public void actionPerformed (ActionEvent e)
                                {
                                        setVisible(false);
                                }
                        });

                        setTitle ("Add a relation");
                        setSize(359,210);
                        setVisible(true);
                }
}
```

23. Classe Windows

```
/**
* Class Windows
*
* Creates the main program windows which contains the explorer panel and the
* OFL description panel
*
* @author  Jerome BROCCOLICCHI and Fabienne KULAKOWSKI
* @version %I%, %G%
* @since JDK1.3
*
*/

import java.awt.Dimension;
import java.awt.Toolkit;
import javax.swing.JFrame;
import javax.swing.JSplitPane;
import java.awt.event.WindowEvent;
import java.awt.event.WindowAdapter;

public class Windows extends JFrame
{

	/**
	* Creates :
	*	the main frame
	*	the explorer panel
	*	the OFL description panel
	*	a split pane to put in the explorer and the OFL description panels
	*	and make the frame visible in the middle of the screen
	*
	* @see ExplorerPanel
	* @see OFLDescriptionPanel
	*
	* @since JDK1.3
	*
	*/
	public Windows ()
	{
		// Create the ExplorerPanel
		ExplorerPanel explorer = new ExplorerPanel ();

		// Create the OFLDescriptionPanel
		OFLDescriptionPanel oflDesc = new OFLDescriptionPanel();

		/*
		* Create a split pane with ExplorerPanel and OFLDescriptionPanel
		*  the frame will be split horizontally (HORIZONTAL_SPLIT)
		*/
		JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,true,explorer,
oflDesc);

		// Get the screen size
		Toolkit tk = Toolkit.getDefaultToolkit();
		Dimension screenSize = tk.getScreenSize();

		// Set the split pane properties
		splitPane.setOneTouchExpandable(true);  // display whether the explorer or the OFl descripition
pane by one click
		splitPane.setDividerSize (8);			// the divider width
		splitPane.setDividerLocation(200);		// where the divider will appear

		explorer.setMinimumSize (new Dimension (0,0));
		oflDesc.setMinimumSize (new Dimension (0,0));
		splitPane.setPreferredSize(new Dimension(700, 500));
```

```
                // Display the frame in the middle of the screen
                setBounds((screenSize.width/2)-350,(screenSize.height/2)-250,700,500);

                // Set window title
                setTitle ("OFL-ML");

                // Set a menu bar
                setJMenuBar (new MainMenu());

                // Add the split pane
                getContentPane().add("Center",splitPane);

                pack ();

                // Add window listener
                // to close the window
                addWindowListener (new WindowAdapter ()
                {
                        public void windowClosing (WindowEvent e)
                        {
                                System.exit(0);
                        }
                });

                // make the window visible
                setVisible (true);
        }
}
```