POA

RAPPORT PROJET : COURBES

CORLOUER FABRICE
&
BELLAMINE ELAMINE

1. Le but du projet:

On veut construire une application permettant de visualiser des fonctions numériques à une variable et dans un intervalle donné, par exemple :
x -> x sin(1/x) entre *-Pi* et *Pi*.

Pour cela, on représente les variations d'une fonction *f* dont l'argument et le résultat sont de type double, sur un intervalle d'étude $[x_1, x_2]$, à l'aide d'une instance de Variations, qui implémente l'interface FunctionVariations.
Cette représentation permet de calculer une approximation de l'intégrale de *f* entre x_1 et x_2.
Le but du TD était de l'utiliser pour visualiser ces variations sur une fenêtre graphique à l'aide de segments successifs.

Après avoir récupérer la correction des TD précédents le but maintenant est de satisfaire le cahier des charges suivant:

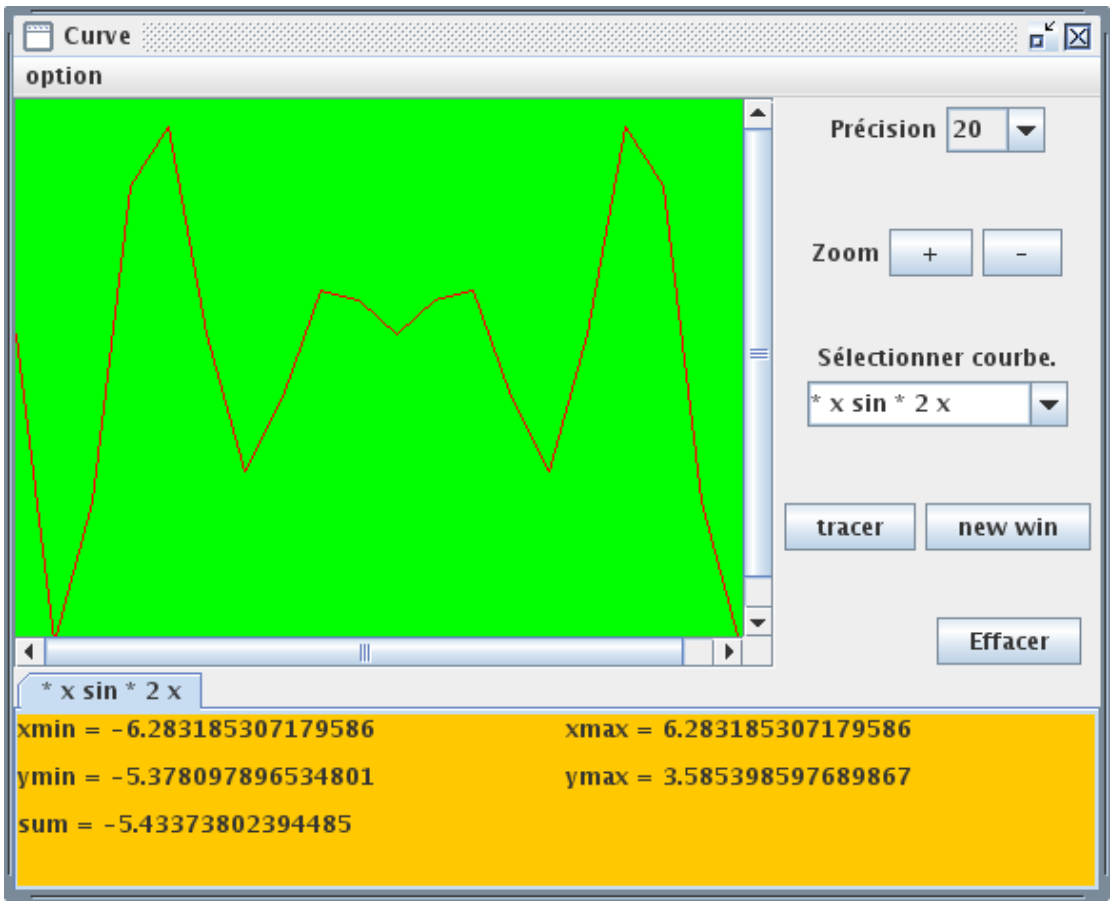
- implémenter une fonctionnalité de ``zoom" (avant et arrière) ;
- autoriser l'affichage simultané de plusieurs fonctions ;
- permettre l'utilisation de plusieurs fenêtres ;
- définir une fonction dans un fichier ;
- ...

2.Comment utiliser le logiciel :

Le projet a été compressé en format « .jar » pour lancer le logiciel il faut taper la commande :

java -jar projet.jar dans la console ou en important le fichier .jar dans eclipse.

une fenêtre s'ouvre comme ceci:



Pour tracer la courbe d'une fonction il faut choisir l'une des deux options:

- sélectionner une fonction qui appartient déjà la liste.
- écrire une fonction en mettant l'opérateur avant les opérandes.

Si on veux tracer une courbe dans une autre fenêtre il faut sélectionner une fonction et appuyer sur tracer, cela va générer une autre fenêtre avec la courbe sélectionné.

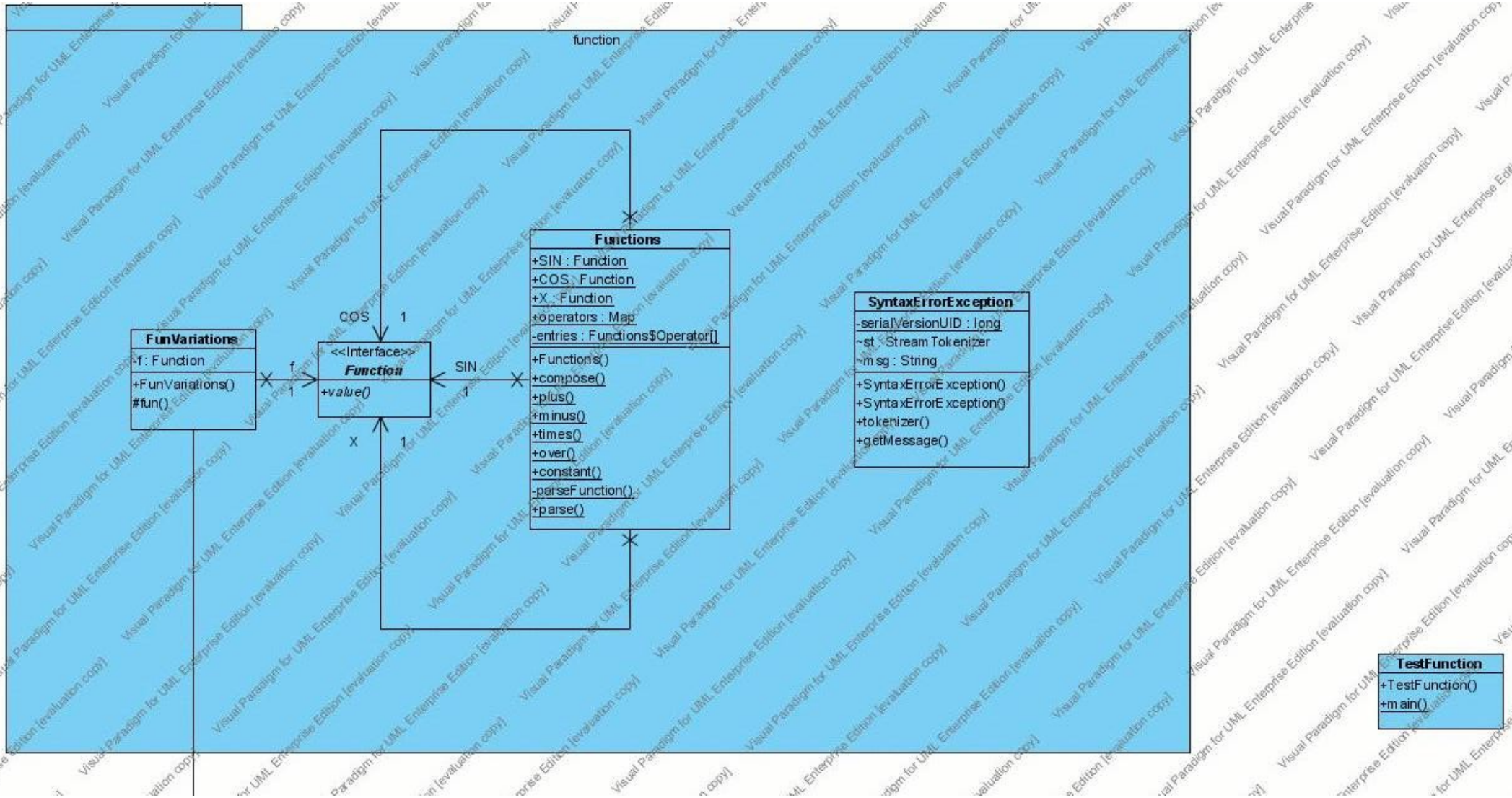
Si on veux tracer une courbe dans la même fenêtre il faut choisir la fonction et appuyer sur tracer .cela va générer un nouvel onglet avec en dessous les informations relatives à la fonction qu'on vient de tracer.

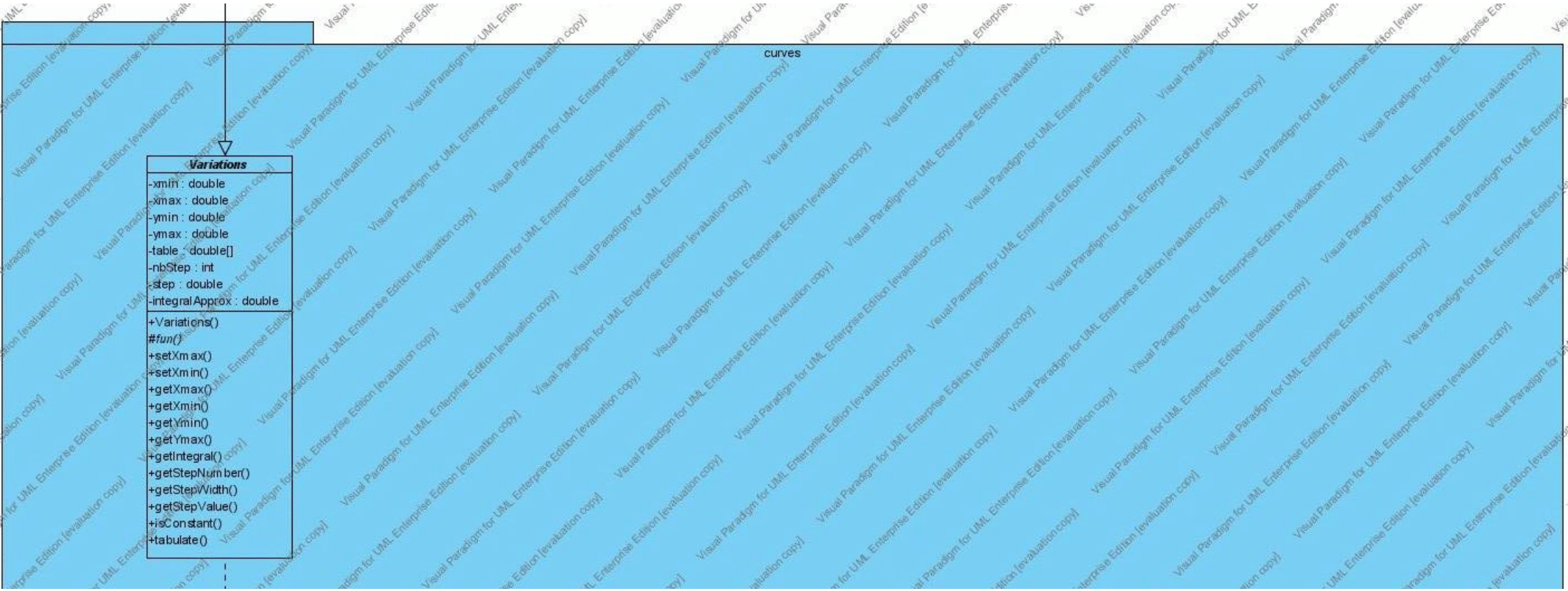
Après avoir tracer une courbe dans une autre fenêtre on peut toutefois la fermer en appuyant sur la croix en haut à droite ou si on le souhaite en allant sur option -> quitter on peut quitter le programme.

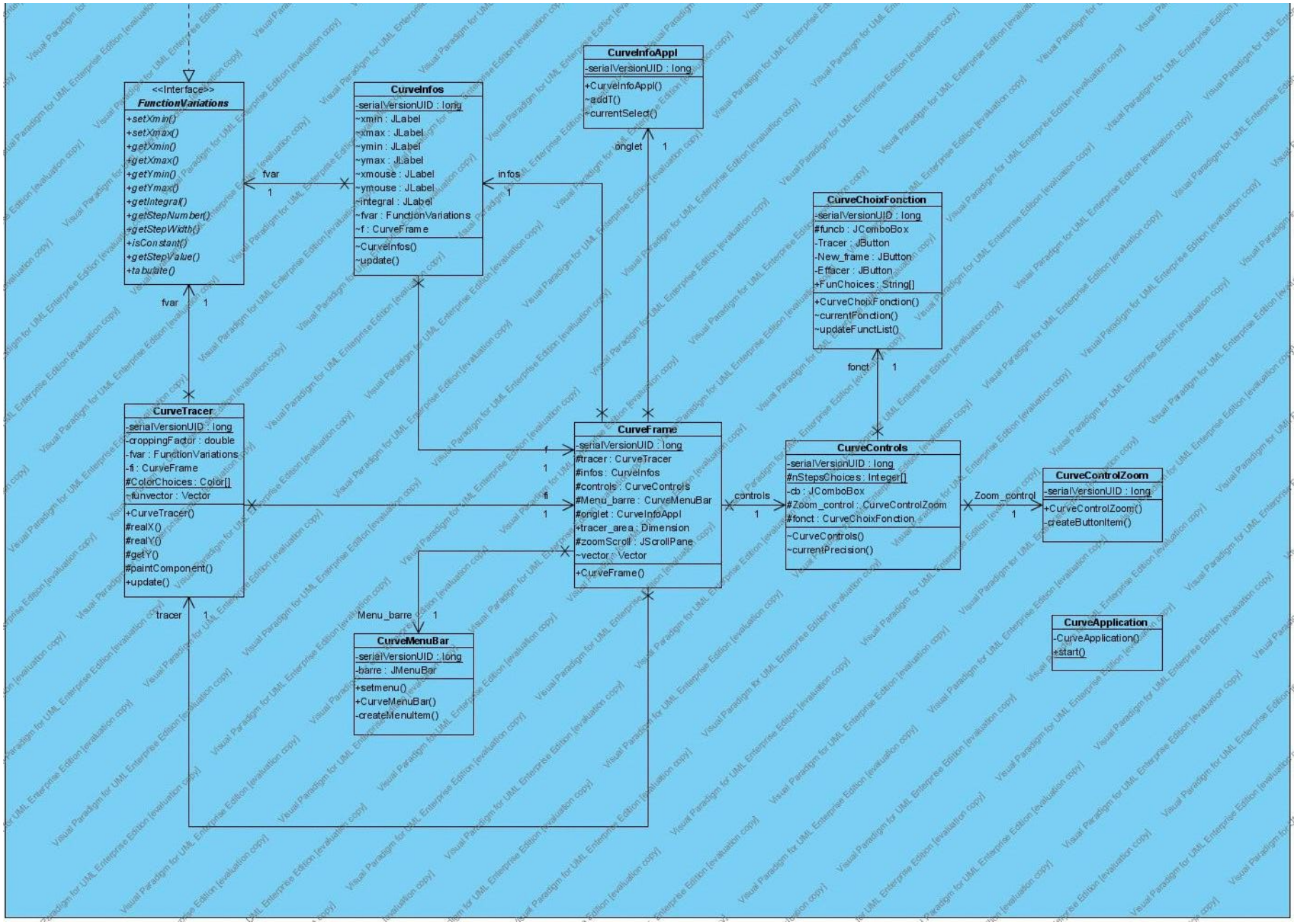
Après avoir tracer plusieurs courbes dans une fenêtre on peut également effacer celle qu'on souhaite, il suffit pour cela cliquer sur l'onglet de la courbe qu'on veux effacer et appuyer sur le bouton effacer.

a n'importe quel moment on peut effectuer un zoom de-zoom ou changer la précision.

3.Diagramme UML(diagramme UML complet- dans diagrammeUML.jpg)


function
FunVariations
-f : Function
+FunVariations()
#fun()
f
1
<<Interface>>
Function
+value()
COS
1
SIN
1
X
1
Functions
+SIN : Function
+COS : Function
+X : Function
+operators : Map
-entries : Functions$Operator[]
+Functions()
+compose()
+plus()
+minus()
+times()
+over()
+constant()
-parseFunction()
+parse()
SyntaxErrorException
-serialVersionUID : long
~st : StreamTokenizer
~msg : String
+SyntaxErrorException()
+SyntaxErrorException()
+tokenizer()
+getMessage()
TestFunction
+TestFunction()
+main()


curves
Variations
-xmin : double
-xmax : double
-ymin : double
-ymax : double
-table : double[]
-nbStep : int
-step : double
-integralApprox : double
+Variations()
#fun()
+setXmax()
+setXmin()
+getXmax()
+getXmin()
+getYmin()
+getYmax()
+getIntegral()
+getStepNumber()
+getStepWidth()
+getStepValue()
+isConstant()
+tabulate()


<<Interface>>
FunctionVariations
+setXmin()
+setXmax()
+getXmin()
+getXmax()
+getYmin()
+getYmax()
+getIntegral()
+getStepNumber()
+getStepWidth()
+isConstant()
+getStepValue()
+tabulate()
CurveInfos
-serialVersionUID : long
~xmin : JLabel
~xmax : JLabel
~ymin : JLabel
~ymax : JLabel
~xmouse : JLabel
~ymouse : JLabel
~integral : JLabel
~fvar : FunctionVariations
~f : CurveFrame
~CurveInfos()
~update()
CurveInfoAppl
-serialVersionUID : long
+CurveInfoAppl()
~addT()
~currentSelect()
CurveChoixFonction
-serialVersionUID : long
#funcb : JComboBox
-Tracer : JButton
-New_frame : JButton
-Effacer : JButton
+FunChoices : String[]
+CurveChoixFonction()
~currentFonction()
~updateFunctList()
CurveTracer
-serialVersionUID : long
-croppingFactor : double
-fvar : FunctionVariations
-fi : CurveFrame
#ColorChoices : Color[]
~funvector : Vector
+CurveTracer()
#realX()
#realY()
#getY()
#paintComponent()
+update()
CurveFrame
-serialVersionUID : long
#tracer : CurveTracer
#infos : CurveInfos
#controls : CurveControls
#Menu_barre : CurveMenuBar
#onglet : CurveInfoAppl
+tracer_area : Dimension
#zoomScroll : JScrollPane
~vector : Vector
+CurveFrame()
CurveControls
-serialVersionUID : long
#nStepsChoices : Integer[]
-cb : JComboBox
#Zoom_control : CurveControlZoom
#fonct : CurveChoixFonction
~CurveControls()
~currentPrecision()
CurveControlZoom
-serialVersionUID : long
+CurveControlZoom()
-createButtonItem()
CurveMenuBar
-serialVersionUID : long
-barre : JMenuBar
+setmenu()
+CurveMenuBar()
-createMenuItem()
CurveApplication
-CurveApplication()
+start()
fvar
1
infos
1
onglet
1
fonct
1
controls
1
Zoom_control
1
tracer
1
Menu_barre
1

4.Implémentation des Classes.

(fichier source en annexe formatés en a2ps)

1,CurveFrame.java

Cette Classe a été légèrement modifiée afin de pouvoir accueillir des nouvelles fonctionnalités.

Tout d'abord , nous créons un vecteur de fonctions *vector,* afin d'y stocker toutes les fonctions que nous allons tracer et traiter. Pour pouvoir effectuer des zooms nous allons utiliser un JscrollPane « zoomScroll » ayant pour paramètre *tracer* dans lequel nous allons afficher nos courbes .Ainsi lorsque l'on effectuera un zoom important *zoomScroll* se comportera comme une fenêtre qui se déplacera sur le tracer de nos courbes.

Pour la classe CurveInfos, on ne fait aucune modification.

On ajoute aussi deux containers supplémentaire onglet « CurveInfoAppl.java » qui va nous permettre de créer un onglet pour chaque courbe tracée avec dedans les informations relatives a cette courbe, et Menu_barre « CurveMenuBar.java » qui va ajouter un menu dans la barre avec 2 fonctionnalités supplémentaires.

En ce qui concerne l'initialisation de la frame principale *mainPane* on ajoute désormais 3 container: *zoomScroll, onglet, et controls* la plupart des fonctionnalités seront engendrées par ces 3 containers.

2, CurveControls.java

Pour cette classe, on passe désormais en paramètre du constructeur un vecteur de fonctions et non plus un objet de type « FunctionVariations », ainsi lors d'un changement de précision toutes les fonctions continues dans le vecteur vont être recalculées (fonction tabulate « FunctionVraitions.java »).De plus nous allons ajouter 2 Jpanels supplémentaire pour pouvoir afficher des boutons en plus *Zoom_control* « CurveConrolZoom.java » pour pouvoir effectuer un zoom et de-zoom puis *fonct* « CurveChoixFonction » qui va nous permettre de saisir ou sélectionner une fonction .Dans cette classe est aussi intégré les boutons permettant de tracer, supprimer une courbe et de tracer la courbe dans une nouvelle fenêtre.

3,CurveControlZoom.java

Zoom_control est de type Jpanel il va contenir deux boutons JButton pour pouvoir exécuter le zoom ou de-zoom lors d'un clic sur un de ces boutons, la taille du Jcomponent *tracer* va être augmenter (ou réduite) ce qui aura pour effet de tracer une courbe avec une nouvelle échelle. Grâce au JScrollPane *zoomScroll* si la grille du tracer est plus grande que l'espace qui lui est attribuée dans la frame principale on va pouvoir l'observer à travers la fenêtre que zoomScroll crée. Les actions sont détectées par des évènements nous implémentons donc dans la construction d'un bouton les méthodes actionPerformed(ActionEvent ev) et ActionListener().

4,CurveChoixFonction.java

Funct est lui aussi une classe fille de JPanel et va contenir un JCombobox éditable *funcb* que nous initialisons avec une liste de fonction pré-enregistrée et qui va nous permettre de sélectionner ou saisir une fonction pour cela lors d'un clic sur celle-ci on va récupérer la fonction sélectionnée ou saisie et vérifier si elle est dans la liste fournie (« FunChoices ») dans le cas contraire et si elle a une syntaxe correcte elle sera ajoutée.

Nous avons ensuite un JButton *Tracer* qui lors d'une détection de clic va ajouter la fonction sélectionnée dans la liste de fonction *funvector* de la frame principale, puis retracer les courbes contenu dans ce vecteur, de plus elle va ajouter un onglet supplémentaire qui contiendra les informations de la nouvelle courbe tracée.

Le JButton New_frame va nous permettre d'ouvrir une nouvelle fenêtre dans laquelle sera tracée la courbe sélectionnée dans *funcb*.

Le JButton Effacer supprime l'onglet sélectionné et sa courbe correspondante pour se faire on agit directement sur le vecteur de fonction de la frame principale *vector* et son *onglet* .

5,CurveInfoAppl,java

C'est ce container qui va nous permettre d'afficher dans des onglets les informations de chaque courbes nommé ici *onglet* de Type JtabbedPane. Cette classe contient un constructeur avec un *super(JTabbedPane.TOP, JtabbedPane.SCROLL_TAB_LAYOUT);*

qui appel la classe mère JtabbedPane avec 2 paramètre qui permettent pour le premier d'afficher les onglets en haut et le deuxième permet d'avoir scroll lorsque le nombre d'onglets est trop important.

6,CurveTracer,java

Ici la classe est très peu modifiée on lui met désormais en paramètre lors de l'appel au constructeur (Vector<FunctionVariations> vec , CurveFrame f) les modification sont dans la méthode paintComponent ou on ajoute ;

```
for (int j = 0; j < funvector.size() ; j++){
        double step = ((double) getWidth()) /
funvector.elementAt(j).getStepNumber();
        if(j>=ColorChoices.length)
                g.setColor(ColorChoices[j-ColorChoices.length]);
        else
                g.setColor(ColorChoices[j]);
```

Ce qui nous permet de tracer toutes les fonctions contenues dans le vecteur *funvector*

et de donner une couleur a chaque courbe. De plus on rajoute une méthode *void update()*

qui est appeler lorsque l'on veut changer la taille du JComponent *Tracer (lors d'un appel de zoom ou de-zoom);*

7,CurveMenuBar.java

Cette classe est très courte elle permet de créer une barre de menu dans laquel on place 2 item *: quitter* qui permet de quitter le programme, un message de confirmation s'affichera afin de ne pas quitter par erreur et *Nouvelle Fenêtre* qui ouvre une nouvelle fenêtre initialisée comme si on venait de lancer le programme.

5.Dysfonctionnements constatés

La fonction zoom met du temps à réagir sur le tracé (à peu près 5 sec),mais quand on bouge les deux JScrollPane ça permet de rafraîchir et ainsi le zoom réagit beaucoup plus vite. Quand on trace une nouvelle courbe dans la même fenêtre on obtient un onglet avec la courbe sélectionné ,à ce moment la on a plus les variations des coordonnées du curseurs , elles ne s'affichent que dans le premier onglet.

6.Les réutilisations possible :

Ce petit projet peut être intégrer à un grand projet de développement pour pouvoir visualiser les différentes variations des fonctions interactivement, je site ainsi les variations de temps de calcul des processeurs,mémoire RAM... (d'ailleurs il existe quelques widgets pour cet effet) mais cela n'est qu'un exemple ,on peut aussi réfléchir à l'intégrer à un projet de développement des calculs scientifiques (calculatrice par exemple).
Quelques améliorations restent possible pour rendre notre programme plus performant et plus rapide .Dans l'état actuel il fonctionne mise a part quelques dysfonctionnements énumérés avant , on aurait pu ajouter une fonction pour enregistrer le tracer d'une courbe dans un fichier .jpg par exemple.

7, Annexes

```
package curves;

import javax.swing.JFrame;

public class CurveApplication {

        private CurveApplication() {
        }

        public static void start(FunctionVariations fv) {
                final FunctionVariations fvar = fv;
                javax.swing.SwingUtilities.invokeLater(new Runnable() {
                        public void run() {
                                JFrame.setDefaultLookAndFeelDecorated(true);
                                CurveFrame cv = new CurveFrame(fvar);
                                cv.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE
);
                        }
                });
        }

}
```

```
package curves;
import function.FunVariations;
import function.Function;
import function.Functions;
import function.SyntaxErrorException;
import java.io.IOException;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class CurveChoixFonction extends JPanel {

        private static final long serialVersionUID = 1L;
        protected JComboBox funcb;
        private Function fp;
        private JButton Tracer;
        private JButton New_frame;
        private JButton Effacer;
        public boolean zoom=false;
        public String FunChoices[] = {"* x sin * 2 x","sin x", "cos x" , "x" , "* x x","* x sin x","+ cos x sin x"};

        public CurveChoixFonction(final FunctionVariations var,final CurveFrame f){

                setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));



        JPanel ed= new JPanel();
        JPanel boutton = new JPanel();


        JLabel titrecourbe = new JLabel("Sélectionner courbe.");
        titrecourbe.setAlignmentX(Component.CENTER_ALIGNMENT);
            titrecourbe.setAlignmentY(Component.CENTER_ALIGNMENT);
        funcb = new JComboBox(FunChoices);

        funcb.setToolTipText("Selectionner la fonction a tracer ou taper en une");
                funcb.setEditable(true);
                funcb.addItemListener((ItemListener) (new ItemListener() {
                        public void itemStateChanged(ItemEvent e)  {
                                if (e.getStateChange() == ItemEvent.SELECTED) {

                                        try {

                                                fp = Functions.parse((String)funcb.getSelectedItem());//FunChoices[funcb.getSelectedIndex()]);
                                                if(updateFunctList((String)funcb.getSelectedItem())==false)
                                                        funcb.addItem((String)funcb.getSelectedItem());

                                                f.controls.repaint();
                                                f.repaint();
                                        } catch (SyntaxErrorException e1) {
                                                JOptionPane.showMessageDialog(new JFrame(),
                                                                " Mauvaise syntaxe \n
Usage : <fonction> <arguments> \n                       \n       (Exemple : * x x -> x²)");
                                        } catch (IOException e1) {
```



```
                                                e1.printStackTrace();
                                        }

                                }


                        }}));


                add(titrecourbe);
                ed.add(funcb);
                ed.setAlignmentX(Component.CENTER_ALIGNMENT);
            ed.setAlignmentY(Component.CENTER_ALIGNMENT);
                add(ed);

                Tracer = new JButton("tracer");
                Tracer.setToolTipText("Cliquer sur ce bouton pour tracer la courbe dans cette fenêtre ");
                Tracer.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent ev) {

                                FunVariations fv = new FunVariations(fp, -2 * Math.PI,2 * Math.PI);
                                fv.tabulate(f.controls.currentPrecision());

                                f.tracer.funvector.add(fv);
                                f.tracer.repaint();
                                f.infos.update();
                                if(zoom==false)
                                        f.onglet.addT(fv,currentFonction());

                                f.repaint();}

                });
                boutton.add(Tracer);



                New_frame = new JButton("new win");
                New_frame.setToolTipText("Cliquer sur ce bouton pour tracer la courbe dans une nouvelle fenetre");
                New_frame.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent ev) {
                                if(FunChoices[funcb.getSelectedIndex()] == null)
                                        JOptionPane.showMessageDialog(new JFrame(),
                                                        "Selectionner une fonction ou ecriver en une \n");
                                else{


                                        FunVariations fv = new FunVariations(fp, -2 * Math.PI,2 * Math.PI);

                                                new CurveFrame((FunctionVariations) fv);

                                }
                        }
                });
                boutton.add(New_frame);
                boutton.setAlignmentX(Component.CENTER_ALIGNMENT);
            boutton.setAlignmentY(Component.CENTER_ALIGNMENT);
```

```
                add(boutton);

                Effacer = new JButton("effacer");
                Effacer.setToolTipText("Cliquer sur ce bouton pour effacer la courbe");
                Effacer.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent ev) {
                                if(f.vector.size() == 1)
                                        JOptionPane.showMessageDialog(new JFrame
(),
                                        "il ny a rien a effacer \n");
                                else{


                                        f.vector.remove(f.onglet.currentSelect()
);
                                        f.onglet.remove(f.onglet.currentSelect()
);
                                        f.repaint();

                                }
                                }
                });
                add(Effacer);


        }
        String currentFonction() {
                return (String)funcb.getSelectedItem();
        }

        boolean updateFunctList(String element){
                for(int i=0;i<FunChoices.length;i++){
                        if (FunChoices[i].equals(element))
                                return true;
                }
                return false;
        }


}
```

```
package curves;
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import java.util.Vector;
/**
 * @author casteran
 */

class CurveControls extends JPanel {

        private static final long serialVersionUID = 1L;

        protected final static Integer nStepsChoices[] = { 1, 2, 3, 4, 5, 10, 20
,
                        40, 80, 160, 320, 640 };

        private JComboBox cb;

        protected CurveControlZoom Zoom_control;
        protected CurveChoixFonction fonct;


        CurveControls(final Vector<FunctionVariations> vec, final CurveFrame f)
{
                super();

                setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
                JPanel precision = new JPanel();
                JLabel title = new JLabel("Précision");
                cb = new JComboBox(nStepsChoices);

                precision.add(title);
                precision.add(cb);
                cb.setToolTipText("CHoisissez la precision de la courbe");
                precision.setAlignmentX(Component.CENTER_ALIGNMENT);
            precision.setAlignmentY(Component.CENTER_ALIGNMENT);

                add(precision);
                cb.addItemListener((ItemListener) (new ItemListener() {
                        public void itemStateChanged(ItemEvent e) {
                                if (e.getStateChange() == ItemEvent.SELECTED) {
                                        for (int j =0 ; j < vec.size(); j++)
                                                vec.elementAt(j).tabulate(curren
tPrecision());

                                        f.infos.update();
                                        f.infos.setVisible(true);
                                        f.repaint();
                                }
                        }

                }));
                cb.setSelectedIndex(nStepsChoices.length / 2);

                Zoom_control = new CurveControlZoom(f);
                Zoom_control.setAlignmentX(Component.CENTER_ALIGNMENT);
            Zoom_control.setAlignmentY(Component.CENTER_ALIGNMENT);
            add(Zoom_control);
                fonct= new CurveChoixFonction(vec.elementAt(0),f);
                fonct.setAlignmentX(Component.CENTER_ALIGNMENT);
            fonct.setAlignmentY(Component.CENTER_ALIGNMENT);
```



```
                add(fonct);

        }
        int currentPrecision() {
                return nStepsChoices[cb.getSelectedIndex()];
        }
}
```

```
package curves;

import java.awt.event.*;
import javax.swing.*;
import java.awt.*;

class CurveControlZoom extends JPanel {

        private static final long serialVersionUID = 1L;

        public CurveControlZoom(final CurveFrame f) {
                super();
                JLabel subtitle = new JLabel("Zoom");
                add(subtitle);

                createButtonItem(this, "+", "Zoomer",
                                new ActionListener() {
                                        @SuppressWarnings("deprecation")
                                        public void actionPerformed(ActionEvent
ev) {

                                                f.dimx *= 2;
                                                f.dimy *= 2;

                                                f.tracer.setPreferredSize(new Di
mension(f.dimx * 2,
                                                                f.dimy * 2));

                                                f.zoomScroll.repaint();
                                                f.tracer.update();
                                                f.tracer.repaint();
                                                f.zoomScroll.repaint();
                                                f.repaint();

                                        }
                                });

                createButtonItem(this, "-",
                                "De-Zoomer", new ActionListener() {
                                        public void actionPerformed(ActionEvent
ev) {
                                                f.dimx /= 2;
                                                f.dimy /= 2;
                                                f.tracer.setPreferredSize(new Di
mension(f.dimx / 2,
                                                                f.dimy / 2));

                                                f.tracer.update();
                                                f.tracer.repaint();
                                                f.zoomScroll.repaint();
                                                f.repaint();
                                        }
                                });

        }

        private void createButtonItem(JPanel but, String im, String ToolTip,
                        ActionListener action) {
                JButton nu = new JButton(im);
                //nu.setPreferredSize(new Dimension(30, 30));
                nu.setToolTipText(ToolTip);
                nu.addActionListener(action);
```



```
                but.add(nu);

        }

}
```

```
package curves;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;
import javax.swing.*;
import java.util.Vector;
import javax.swing.JFrame;
import javax.swing.JPanel;



/**
 * A class  to represent the variations of some function  in some interval.
 * Allows some control on the accuracy of this representation
 */

/**
 * @author casteran
 */
public class CurveFrame extends JFrame{
	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	/** the graphic part of the display */
	protected CurveTracer tracer;
	/** information bar */
	protected CurveInfos infos;

	/** various commands */
	protected CurveControls controls;
	protected CurveMenuBar Menu_barre;
	protected CurveInfoAppl onglet;
	public int dimx = 400;
	public int dimy=300;

	protected JScrollPane zoomScroll;
	/**
	 * Builds a top-level window from the variations of a function
	 *
	 * @see FunctionVariations
	 *
	 */

	final Vector<FunctionVariations> vector;


	public CurveFrame(FunctionVariations fvar) {

		super("Curve");
		vector = new Vector<FunctionVariations>();
		vector.add(fvar);

		tracer = new CurveTracer(vector,this);
		tracer.update();
		zoomScroll= new JScrollPane(tracer);
		zoomScroll.setPreferredSize(new Dimension(dimx,dimy));
		zoomScroll.revalidate();
```



```
		infos = new CurveInfos(fvar);

		controls = new CurveControls(vector, this);
		onglet=new CurveInfoAppl(infos,controls.fonct);
		Menu_barre = new CurveMenuBar(this);
		setJMenuBar(Menu_barre.setmenu());


		JPanel mainPane = new JPanel(new BorderLayout());


		mainPane.add(zoomScroll, BorderLayout.CENTER);
		mainPane.add(onglet, BorderLayout.SOUTH);
		mainPane.add(controls, BorderLayout.EAST);


		tracer.addMouseListener(new MouseAdapter() {
			CurveFrame cf = CurveFrame.this;

			public void mouseEntered(MouseEvent e) {
				cf.infos.xmouse.setText("x = " + cf.tracer.realX(
e.getX()));
				cf.infos.ymouse.setText("y = " + cf.tracer.realY(
e.getY()));
				cf.infos.repaint();
			}

			public void mouseExited(MouseEvent e) {
				cf.infos.xmouse.setText("");
				cf.infos.ymouse.setText("");
				cf.infos.repaint();
			}
		});


		tracer.addMouseMotionListener(new MouseMotionAdapter() {
			CurveFrame cf = CurveFrame.this;

			public void mouseMoved(MouseEvent e) {
				cf.infos.xmouse.setText("x = " + cf.tracer.realX(
e.getX()));
				cf.infos.ymouse.setText("y = " + cf.tracer.realY(
e.getY()));
				cf.infos.repaint();
			}

			public void mouseDragged(MouseEvent e) {
				cf.infos.xmouse.setText("x = " + cf.tracer.realX(
e.getX()));
				cf.infos.ymouse.setText("y = " + cf.tracer.realY(
e.getY()));
				cf.infos.repaint();
			}
		});

		fvar.tabulate(controls.currentPrecision());
		setContentPane(mainPane);
		pack();
		infos.update();
```

```
                setVisible(true);
        }
}
```

```
package curves;
import javax.swing.JTabbedPane;


import javax.swing.*;


/**
 *
 * @author casteran
 *
 */


public class CurveInfoAppl extends  JTabbedPane{

        private static final long serialVersionUID = 1L;




        public CurveInfoAppl (CurveInfos in ,CurveChoixFonction choix) {

         super();
         this.addTab(choix.currentFonction(), new ImageIcon("curves/courbe.gif"), in,
 "Does nothing");

        }
        void addT(FunctionVariations f,String choix){
                CurveInfos inf = new CurveInfos(f);
                this.addTab(choix, new ImageIcon("curves/courbe.gif"), inf, "Does nothin
g");
                inf.update();
        }



        int currentSelect(){
                return this.getSelectedIndex();
        }


}
```

```
package curves;

import java.awt.Color;
import java.awt.GridLayout;


import javax.swing.JLabel;
import javax.swing.JPanel;

/**
 *
 * @author casteran
 *
 */

class CurveInfos extends JPanel {
        /**
         *
         */
        private static final long serialVersionUID = 1L;

        JLabel xmin, xmax, ymin, ymax;

        JLabel xmouse, ymouse;

        JLabel integral;

        FunctionVariations fvar;
        CurveFrame f;


        CurveInfos(     FunctionVariations fvar) {
                this.fvar =fvar;


                setLayout(new GridLayout(0, 2, 10, 10));
                xmin = new JLabel();
                xmax = new JLabel();
                ymin = new JLabel();
                ymax = new JLabel();
                xmouse = new JLabel();
                ymouse = new JLabel();
                integral = new JLabel();

                add(xmin);
                add(xmax);
                add(ymin);
                add(ymax);
                add(integral);
                add(new JLabel());
                add(xmouse);
                add(ymouse);
                setBackground(Color.orange);
        }

        void update() {

                xmin.setText("xmin = " + fvar.getXmin());
                xmax.setText("xmax = " + fvar.getXmax());
                ymin.setText("ymin = " + fvar.getYmin());
                ymax.setText("ymax = " + fvar.getYmax());
                xmouse.setText("");
```



```
                ymouse.setText("");
                integral.setText("sum = " + fvar.getIntegral());
        }


}
```

```
package curves;

import java.awt.event.*;

import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;

import function.FunVariations;
import function.Function;
import function.Functions;

public class CurveMenuBar extends JFrame {
        private static final long serialVersionUID = 1L;
        private JMenuBar barre;


        public JMenuBar setmenu(){
                return this.barre;
        }


        public CurveMenuBar( final CurveFrame fi){

                this.barre = new JMenuBar();
                setJMenuBar(barre);
                JMenu option = new JMenu ("option");

                barre.add(option);


                createMenuItem(option, "Nouvelle fenÃªtre","Ouvrir une nouvelle fenêtre vierge"
, new ActionListener() {
                        public void actionPerformed(ActionEvent ev) {

                                Function func = Functions.X;
                                CurveApplication.start(new FunVariations(func, -
2 * Math.PI,
                                                2 * Math.PI));
                        }
                });

                createMenuItem(option, "quitter","Fermer cette fenetre", new ActionListen
er() {
                        public void actionPerformed(ActionEvent ev) {
                                fi.dispose();
                        }
                });


                setVisible(true);
        }

        private void createMenuItem(JMenu menu, String name,String ToolTip , Act
ionListener action) {
                JMenuItem menuItem = new JMenuItem(name);
                menuItem.setToolTipText(ToolTip);
                menuItem.addActionListener(action);
                menu.add(menuItem);
        }
}
```

```
package curves;
import java.util.Vector;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;

import javax.swing.JComponent;

/**
 *
 * @author casteran
 *
 */

class CurveTracer extends JComponent {
        /**
         *
         */
        private static final long serialVersionUID = 1L;

        FunctionVariations fvar;

        /**
         * part of the height of the canvas devoted to the drawing of the curve
         */
        protected final static Color ColorChoices[] = { Color.red, Color.black,
Color.orange, Color.white, Color.pink, Color.cyan,Color.yellow,
                Color.magenta };
        private double croppingFactor = 0.9 ;
        CurveFrame fi;


        Vector<FunctionVariations> funvector;

        public CurveTracer(Vector<FunctionVariations> vec , CurveFrame f) {

                this.fi=f;
                this.fvar = vec.elementAt(0);
                this.funvector = vec;

                //setPreferredSize(new Dimension(f.dimx, f.dimy));
                setOpaque(true);
                setBackground(Color.green);
                //setForeground(Color.red);

        }

        /**
         * converts the mouse's x into some value of the considered interval.
         * correction 29/03/1999
         */

        protected double realX(int mouseX) {
                double xmin = fvar.getXmin();
                return xmin + (mouseX * (fvar.getXmax() - xmin) / getWidth());

        }

        /**
         * converts the mouse's y into the corresponding value in the function's
         * range
         */
```



```
        protected double realY(int mouseY) {
                if (fvar.isConstant())
                        return fvar.getYmax();
                int h = getHeight();
                double ymax = fvar.getYmax();
                return ymax + (fvar.getYmin() - ymax)
                                * (mouseY - h * (1 - croppingFactor) * 0.5) / h
                                / croppingFactor;
        }

        /** converts some y in the fun's range into a mouse y coordinate */

        protected int getY(double y) {
                int h = getHeight();
                if (fvar.isConstant())
                        return (int) h / 2;
                else {
                        double ymax = fvar.getYmax();
                        return (int) ((h * (1 - croppingFactor) / 2.0) + h * cro
ppingFactor
                                        * (y - ymax) / (fvar.getYmin() - ymax));
                }
        }

        protected void paintComponent(Graphics g) {
                int width = getWidth();

                // Paint background if we're opaque.
                if (isOpaque()) {
                        g.setColor(getBackground());
                        g.fillRect(0, 0, width, getHeight());
                }

                g.setColor(getForeground());
                for (int j = 0; j < funvector.size() ; j++){
                        double step = ((double) getWidth()) / funvector.elementA
t(j).getStepNumber();
                        g.setColor(ColorChoices[j]);
                double lastX = 0;
                double newX;

                int lastY = getY(funvector.elementAt(j).getStepValue(0));
                int newy;
                for (int i = 0; i < funvector.elementAt(j).getStepNumber() - 1;
i++) {
                        newy = getY(funvector.elementAt(j).getStepValue(i + 1));
                        newX = lastX + step;
                        g.drawLine((int) lastX, lastY, (int) newX, newy);
                        lastY = newy;
                        lastX = newX;
                }
                if (lastX < (double) width) {
                        g.drawLine((int) lastX, lastY, width, lastY);
                }
        }
}
        public void update(){

                this.setPreferredSize(new Dimension(fi.dimx,fi.dimy));
                this.repaint();
```

```
        }
}
```

```
package function;

public interface Function {
        public double value(double x);
}
```

```
package function;

import java.io.IOException;
import java.io.StreamTokenizer;
import java.io.StringReader;
import java.util.HashMap;
import java.util.Map;

public class Functions {
	public static final Function SIN = new Function() {
		public double value(double x) {
			return Math.sin(x);
		}

	};

	public static final Function COS = new Function() {
		public double value(double x) {
			return Math.cos(x);
		}
	};

	public static final Function X = new Function() {
		public double value(double x) {
			return x;
		}
	};

	public static final Function compose(final Function f1, final Function f
2) {
		return new Function() {
			public double value(double x) {
				return f1.value(f2.value(x));
			}
		};
	}

	public static final Function plus(final Function f1, final Function f2)
{
		return new Function() {
			public double value(double x) {
				return f1.value(x) + f2.value(x);
			}
		};
	}

	public static final Function minus(final Function f1, final Function f2)
 {
		return new Function() {
			public double value(double x) {
				return f1.value(x) - f2.value(x);
			}
		};
	}

	public static final Function times(final Function f1, final Function f2)
 {
		return new Function() {
			public double value(double x) {
				return f1.value(x) * f2.value(x);
			}
		};
```



```
	}

	public static final Function over(final Function f1, final Function f2)
{
		return new Function() {
			public double value(double x) {
				return f1.value(x) / f2.value(x);
			}
		};
	}

	public static final Function constant(final double c) {
		return new Function() {
			public double value(double x) {
				return c;
			}
		};
	}

	public static final Map<String, Operator> operators = new HashMap<String
, Operator>();

	private static abstract class Operator {
		final String name;

		final int arity;

		public Operator(String name, int arity) {
			this.name = name;
			this.arity = arity;
		}

		public abstract Function eval(Function... args);
	}

	private static Operator[] entries = { new Operator("x", 0) {
		public Function eval(Function... functions) {
			return X;
		}
	}, new Operator("sin", 1) {
		public Function eval(Function... args) {
			return compose(SIN, args[0]);
		}
	}, new Operator("cos", 1) {
		public Function eval(Function... args) {
			return compose(COS, args[0]);
		}
	}, new Operator("+", 2) {
		public Function eval(Function... args) {
			return plus(args[0], args[1]);
		}
	}, new Operator("-", 2) {
		public Function eval(Function... args) {
			return minus(args[0], args[1]);
		}
	}, new Operator("*", 2) {
		public Function eval(Function... args) {
			return times(args[0], args[1]);
		}
	}, new Operator("/", 2) {
		public Function eval(Function... args) {
			return over(args[0], args[1]);
```

```
                }
        } };

        static {
                for (Operator e : entries) {
                        operators.put(e.name, e);
                }
        }

        private static final Function parseFunction(StreamTokenizer st)
                        throws SyntaxErrorException, IOException {
                switch (st.ttype) {
                case StreamTokenizer.TT_NUMBER:
                        return constant(st.nval);
                case StreamTokenizer.TT_WORD:
                        Operator op = operators.get(st.sval);
                        if (op == null) {
                                throw new SyntaxErrorException(st, "Unknown operato
r");
                        }
                        Function[] args = new Function[op.arity];
                        for (int i = 0; i < op.arity; ++i) {
                                st.nextToken();
                                args[i] = parseFunction(st);
                        }
                        return op.eval(args);
                default:
                        throw new SyntaxErrorException(st);
                }
        }

        public static final Function parse(String s) throws SyntaxErrorException
,
                        IOException {

                StreamTokenizer st = new StreamTokenizer(new StringReader(s));
                st.wordChars('!', '~');
                st.eolIsSignificant(false);
                st.nextToken();
                Function f = parseFunction(st);
                st.nextToken();
                if (st.ttype != StreamTokenizer.TT_EOF) {
                        throw new SyntaxErrorException(st);
                }
                return f;
        }

}
```

```
package curves;

/**
 * represents the (discretized) variations of a numeric function .
 *  @see ExampleTrigo
 *  @see Escalier
 */
/**
 *
 * @author casteran
 *
 */
public interface FunctionVariations {
        /** sets the leftmost point of the considered interval */
        public void setXmin(double xmin);

        /** sets the rightmost point of the considered interval */
        public void setXmax(double xmax);

        /** gets the leftmost point of the considered interval */
        public double getXmin();

        /** gets the rightmost point of the considered interval */
        public double getXmax();

        /** gets the minimum value of the function */
        public double getYmin();

        /** gets the maximum value of the function */
        public double getYmax();

        /** returns an approximation of the integral of fun */
        public double getIntegral();

        /** returns the number of steps in the representation */
        public int getStepNumber();

        /** returns the width of a step */
        public double getStepWidth();

        /** checks wether the (DISCRETE) function fun has a unique value */
        public boolean isConstant();

        /** returns the ith step of fun's variations */
        public double getStepValue(int i);

        /** computes the array of the variations of fun */
        public void tabulate(int nbStep);

}
```

```
package function;

import curves.Variations;

public class FunVariations extends Variations {
        private Function f;

        public FunVariations(Function f, double xmin, double xmax) {
                this.f = f;
                setXmin(xmin);
                setXmax(xmax);
        }

        protected double fun(double x) {
                return f.value(x);
        }
}
```

```
package function;

import java.io.StreamTokenizer;

public class SyntaxErrorException extends Exception {

        private static final long serialVersionUID = 1035562010426719163L;

        StreamTokenizer st = null;

        String msg;

        public SyntaxErrorException(StreamTokenizer st) {
                this.st = st;
        }

        public SyntaxErrorException(StreamTokenizer st, String msg) {
                this.st = st;
                this.msg = msg;
        }

        public StreamTokenizer tokenizer() {
                return st;
        }

        public String getMessage() {
                return "Syntax Error " + st + ((msg == null) ? "" : " " + msg);
        }
}
```

```
import java.io.IOException;

import curves.CurveApplication;
import function.FunVariations;
import function.Function;
import function.Functions;
import function.SyntaxErrorException;


public class TestFunction {
        public static void main(String[] args) throws SyntaxErrorException, IOEx
ception {
                //Function f = times(X, comp(SIN, times(constant(2.), X)));
                Function f = Functions.parse("* x sin * 2 x");

                CurveApplication.start(new FunVariations(f, -2 * Math.PI, 2 * Ma
th.PI));

        }
}
```

```
package curves;

/**
 A standard implementation of FunVariations.
 Needs to be extended by a definition of fun to  work effectively.
 * @see FunVariations
 * @see ExampleTrigo
 * @see Escalier
 */

/**
 *
 * @author casteran
 *
 */

public abstract class Variations implements FunctionVariations {
	private double xmin, xmax, ymin, ymax;

	private double[] table;

	private int nbStep;

	private double step;

	protected abstract double fun(double x);

	private double integralApprox;

	public final void setXmax(double xmax) {
		this.xmax = xmax;
	}

	public final void setXmin(double xmin) {
		this.xmin = xmin;
	}

	public final double getXmax() {
		return xmax;
	}

	public final double getXmin() {
		return xmin;
	}

	public double getYmin() {
		return ymin;
	}

	public double getYmax() {
		return ymax;
	}

	public final double getIntegral() {
		return integralApprox;
	}

	public final int getStepNumber() {
		return nbStep;
	}

	public double getStepWidth() {
```



```
		return step;
	}

	public double getStepValue(int i) {
		return table[i];
	}

	public boolean isConstant() {
		return ymin == ymax;
	}

	public final void tabulate(int nbStep) {
		this.nbStep = nbStep;
		table = new double[nbStep + 1];
		step = (xmax - xmin) / nbStep;
		table[0] = ymin = ymax = fun(xmin);
		integralApprox = step * table[0];
		double x = xmin + step;
		for (int i = 1; i <= nbStep; i++, x+=step) {
			double val = fun(x);
			table[i] = val;
			integralApprox += step * val;
			if (val < ymin)
				ymin = val;
			else if (val > ymax)
				ymax = val;
		}
	}

}
```