

```

import java.util.ArrayList;
public class Polynomial
{
    private ArrayList<Double> alCoefficients = new ArrayList<>();

    public void add(double pCoeff)
    {
        alCoefficients.add(pCoeff);
    }

    public int size()
    {
        return alCoefficients.size();
    }

    public double getCoefficient(int i)
    {
        if (i >= alCoefficients.size())
            return 0;
        else
            return alCoefficients.get(i);
    }

    public String toString()
    {
        String res = "";
        String exp;
        double c;

        /*
        // Version simple:
        for (int i = 0; i < alCoefficients.size(); i++)
        {
            res = "+"+(alCoefficients.get(i)+"x"+i)+res;
        }

        */

        // Version complexe: gestion des + et - des coefficients
        for (int i = alCoefficients.size() - 1; i >= 0; i--)
        {
            c = alCoefficients.get(i);

            // dans tous les cas on ajoute l'exposant aux x
            // x^0, x^1, x^2, x^3, ...
            // les coefficients 1 / -1 et 0 sont affichés également
            exp = "x" + i;

            if (c < 0)
            {
                if (res.equals(""))
                    res = "-";
                else
                    res = res + " - ";
                res = res + Math.abs(c) + exp;
            }
            else
            {
                if (res.equals(""))
                    res = c + exp;
                else
                    res = res + " + " + c + exp;
            }
        }

        // Cas spécial si tout les coefficients sont nuls
        if (res.equals(""))
            res = "0.0x^0";

        return "y = " + res;
    }

    public Object[] toArray()
    {
        return alCoefficients.toArray();
    }

    public double evaluateNaive(double pX)
    {
        double sum = 0;
        for (int i = 0; i < alCoefficients.size(); i++)
        {
            sum = sum + alCoefficients.get(i) * Math.pow(pX, i);
        }
        return sum;
    }

    public double evaluateHorner(double pX)
    {
        double sum = 0;
        for (int i = alCoefficients.size() - 1; i >= 0; i--)
        {
            sum = sum * pX + alCoefficients.get(i);
        }
        return sum;
    }
}

```

```

public void computeDerivative()
{
    // Version 1: utiliser une liste temporaire

    // ne rien faire si pas de polynôme de départ
    if (alCoefficients.isEmpty())
        return;

    // Attention:
    // le polynôme original est remplacé par la dérivée!!

    // Dériver  $c \cdot x^n \rightarrow c \cdot n \cdot x^{(n-1)}$ 
    // si  $n == 0$  on ne fait rien... car le résultat donne  $0 \cdot x^{(-1)}$ 
    // donc on ajoute à la liste temporaire le calcul de la position n
    ArrayList<Double> alTemp = new ArrayList<>();

    for (int i = 1; i < alCoefficients.size(); i++)
    {
        alTemp.add(alCoefficients.get(i) * i);
    }

    // Recopier le résultat de la dérivée dans le tableau des coefficients original
    alCoefficients.clear();
    for (int i = 0; i < alTemp.size(); i++)
    {
        alCoefficients.add(alTemp.get(i));
    }

    // !! dans de nombreux cas pas besoin de recopier:
    // alCoefficients = alTemp;
    // une affectation suffit

    // Si le polynôme dérivé final est vide il faut ajouter le coefficient 0 pour  $x^0$ 
    if (alCoefficients.isEmpty())
        alCoefficients.add(0.0);
}

public void computeDerivative2()
{
    // Version 2: sans utiliser une liste temporaire

    // ne rien faire si pas de polynôme de départ
    if (alCoefficients.isEmpty())
        return;

    // Attention:
    // le polynôme original est remplacé par la dérivée!!

    // Dériver  $c \cdot x^n \rightarrow c \cdot n \cdot x^{(n-1)}$ 
    // si  $n == 0$  on ne fait rien... car le résultat donne  $0 \cdot x^{(-1)}$ 
    // donc on remplace le coefficient à la
    // position (n-1) par celui du calcul de la position n
    for (int i = 1; i < alCoefficients.size(); i++)
    {
        alCoefficients.set(i - 1, alCoefficients.get(i) * i);
    }

    // Enlever le coefficient à la position n
    if (alCoefficients.size() > 0)
        alCoefficients.remove(alCoefficients.size() - 1);

    // Si le polynôme dérivé final est vide il faut ajouter le coefficient 0 pour  $x^0$ 
    if (alCoefficients.isEmpty())
        alCoefficients.add(0.0);
}
}

```

```

public class MainFrame extends javax.swing.JFrame
{
    private Polynomial poly = new Polynomial();

    public void updateView()
    {
        coeffsList.setListData(poly.toArray());

        polyLabel.setText(poly.toString());

        naiveLabel.setText("-");
        hornerLabel.setText("-");
    }

    public MainFrame()
    {
        initComponents();
        updateView();
    }
// Skipped: ... initComponents { ... }
    private void addButtonActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_addButtonActionPerformed
        String cVal = coeffTextField.getText();
        if (cVal.equals(""))
            return; // pas de coeff tapé

        poly.add(Double.valueOf(cVal));
        updateView();
        coeffTextField.setText("");
    }GEN-LAST:event_addButtonActionPerformed

    private void newButtonActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_newButtonActionPerformed
        poly = new Polynomial();
        updateView();
    }GEN-LAST:event_newButtonActionPerformed

    private void evaluateButtonActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_evaluateButtonActionPerformed
        String xVal = xTextField.getText();
        if (xVal.equals(""))
            return; // pas de x tapé

        double x = Double.valueOf(xVal);
        double y;

        y = poly.evaluateNaive(x);
        naiveLabel.setText("y = " + Double.toString(y));
        y = poly.evaluateHorner(x);
        hornerLabel.setText("y = " + Double.toString(y));
    }GEN-LAST:event_evaluateButtonActionPerformed

    private void computeButtonActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_computeButtonActionPerformed
        poly.computeDerivative();
        updateView();
    }GEN-LAST:event_computeButtonActionPerformed

    private void coeffTextFieldActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_coeffTextFieldActionPerformed

        String cVal = coeffTextField.getText();
        if (cVal.equals(""))
            return; // pas de coeff tapé

        poly.add(Double.valueOf(cVal));
        updateView();
        coeffTextField.setText("");
    }GEN-LAST:event_coeffTextFieldActionPerformed
// Skipped: ... Look & Feel
// Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton addButton;
    private javax.swing.JTextField coeffTextField;
    private javax.swing.JList coeffsList;
    private javax.swing.JButton computeButton;
    private javax.swing.JButton evaluateButton;
    private javax.swing.JLabel hornerLabel;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JLabel naiveLabel;
    private javax.swing.JButton newButton;
    private javax.swing.JLabel polyLabel;
    private javax.swing.JTextField xTextField;
// End of variables declaration//GEN-END:variables
}

```