R3.04 - TD 4 : Binôme

Objectifs
Ce TD a pour objectif de vous montrer l'intérêt d'un travail de maintenance par retro conception s'appuyant sur le langage UML.

Principaux concepts discutés:
- Approche de transformation de programme à l'aide de rétro-conception et travail au niveau du modèle ;
- Comment l'architecture peut anticiper et faciliter les évolutions.

L'application Binome

L'application *Binome* est fournie sous forme de code. Il est demandé d'identifier les modifications à apporter pour que cette application gère aussi le calcul des racines complexes des polynômes du second degré à discriminant négatif. Pour parvenir à ce résultat, il faudra mettre en oeuvre la démarche suivante :

1. comprendre comment fonctionne l'application actuelle,
2. identifier les modifications à apporter pour répondre aux nouvelles exigences du cahier des charges,
3. transmettre de façon non-ambigüe ces modifications au programmeur.

Rappel sur les polynômes du second degré :

On appelle **polynôme (ou trinôme) du second degré** toute expression pouvant se mettre sous la forme :

$P(x)=ax^2+bx+c$

où a, b et c sont des réels avec $a \neq 0$

Le nombre $\Delta=b^2-4ac$ s'appelle le ***discriminant*** du trinôme ax^2+bx+c

Racines d'un polynôme du second degré

L'équation $ax^2+bx+c=0$

- n'a aucune solution réelle si $\Delta < 0$
- a une solution unique (appelée racine double) $x_0=-b/2a$ si $\Delta=0$
- a deux solutions si $\Delta > 0$

$$x_1 = \frac{-b-\sqrt{\Delta}}{2a} \quad \text{et} \quad x_2 = \frac{-b+\sqrt{\Delta}}{2a}$$

Question 1

A l'aide du code ci-dessous

a) construire le diagramme de classes de conception ;

b) puis expliquer le statut de la méthode **`create`** ;

c) Enfin pourquoi le constructeur **`Binome`** est déclaré *protected* ?

Question 2

Pour comprendre le fonctionnement de cette application, modéliser à l'aide d'un diagramme de séquence complet la méthode **`main()`** de la classe **`Bin .`**

Question 3 (à finir en TP)

Modifier le diagramme de classes de conception pour qu'il tienne compte de binômes à racines complexes en faisant en sorte que l'on puisse réutiliser dans une autre application le concept de *Complexe*. Cela signifie qu'il faut **définir une classe *Complexe dans un autre package*** qui représente un nombre complexe et sa manipulation.

Modifier le code en conséquence.

Rappel sur les racines complexes :

Un nombre complexe z est composé d'une partie réelle a et d'une partie imaginaire b. Il s'écrit $z = a + ib$, i étant le nombre imaginaire dont le carré est -1.

Un discriminant négatif signifie que l'équation $az^2 + bz + c = 0$ admet deux solutions complexes conjuguées dans l'ensemble C des complexes :

$$z_1 = \frac{-b - i\sqrt{|\Delta|}}{2a} \text{ et } z_2 = \frac{-b + i\sqrt{|\Delta|}}{2a}.$$

Question 4

Le code d'une classe cliente de la classe *Binome* est-il affecté par l'ajout de cette nouvelle fonctionnalité ?

Code de l'application

```
import binome.Binome;

/**
* launcher class of the application
*/
public class Bin {

    public static void main(String args[]) {

        Binome abin;
        abin = Binome.create (1.0, 0.0, 1.0);
        abin.computeRoots();
        abin.displayRoots();
        abin = Binome.create (1.0, 0.0, -1.0);
        abin.computeRoots();
        abin.displayRoots();
        abin = Binome.create (1.0, 2.0, 1.0);
        abin.computeRoots();
        abin.displayRoots();
    } // end main

} // end Bin
```

```
package binome;

public abstract class Binome {

    protected double a; // X2 coefficient
    protected double b; // X coefficient/
    protected double c; // constant of the binome
    protected Discriminant aDisc; // The binome discriminant

    protected Binome (double cx2, double cx, double cons, Discriminant delta) {
        a = cx2 ; b = cx ; c = cons ;
        aDisc = delta ;
    }

    public static Binome create(double a, double b, double c) {
        Discriminant d;         // a discriminant instance
        double delta;           // the discriminant value
        Binome aBin;            // the instance of Binome

        d = new Discriminant (a, b, c);
        delta = d.value();
        if ( delta == 0.0 ) {
            aBin = new BinomeWithOne (a, b, c, d);
        } else if ( delta > 0.0 ) {
            aBin = new BinomeWithTwo (a, b, c, d);
        } else {
            aBin = new BinomeWithout (a, b, c, d);
        }
        return aBin;
    } // end create


    public abstract void computeRoots();
    public abstract void displayRoots();
} // end Binome
```

```
package binome;

/**
 *  Binome object with one root
 */
class BinomeWithOne extends Binome {

    private double theRoot;

    protected BinomeWithOne( double cx2,double cx, double cons,
                                                Discriminant delta) {
        super (cx2, cx, cons, delta);
    }

    /**
     *  Computes the roots
     */
    public void computeRoots() {
        theRoot = (-b / (2 * a));
    }

    /**
     *  Displays the roots
     */
    public void displayRoots() {
        System.out.println ("Une racine double : " + theRoot);
    }

} // end BinomeWithOne
```

```
package binome;

/**
 *  Binome object with two roots
 */
class BinomeWithTwo extends Binome {

    private double firstRoot, secondRoot;

    protected BinomeWithTwo ( double cx2,double cx, double cons,
                                            Discriminant delta) {
        super (cx2, cx, cons, delta);
    }

    public void computeRoots() {
        firstRoot = (-b + Math.sqrt (aDisc.value()) / (2.0 * a));
        secondRoot = (-b - Math.sqrt (aDisc.value()) / (2.0 * a));
    }


    public void displayRoots() {
        System.out.println ("Deux racines distinctes : \n\tx1 = " +
                            firstRoot);
        System.out.println ("\tx2 = " + secondRoot);
    }

} // end BinomeWithTwo
```

```
package binome;

/**
 *  Binome object without roots
 */
class BinomeWithout extends Binome {

    protected BinomeWithout ( double cx2,double cx, double cons,
                                                    Discriminant delta) {
        super (cx2, cx, cons, delta);
    }


    public void computeRoots() {
     // nothing to do here
    }

    public void displayRoots() {
        System.out.println ("Pas de racine");
    }

} // end BinomeWithout
```

```
package binome;

/**
 *  Handle the computation of the discriminant
 */
public class Discriminant {

    private double delta;

    /**
     *  Dicriminant constructor: computes its value
     */
    public Discriminant(double a, double b, double c) {
        delta = (b * b) - (4.0 * a * c);
    }

    /**
     *  Return the discriminant value
     */
    public double value() {
        return delta;
    }

} // end Discriminant
```