Chapitre 9

De C++ à Java

9.1 Introduction

Le langage *java* a été créé par Sun Micro-Systems dans les années 1990. L'objectif du projet *java* était la conception d'un langage simple pour des applications portables avec interface graphique utilisateur (GUI = *graphic user interface*), et interactif avec Internet et ses navigateurs ; pour ce dernier objectif deux nouveaux environnements ont été créés : les *applets* et le *javascript*.

Il y a ainsi trois modes pour exécuter du *java*, les deux précédents et le mode usuel comme tout autre langage de programmation. Dans les trois modes, la syntaxe est la même sauf que certaines fonctions ne sont pas disponibles dans certains modes. Ainsi, *javascript* est un tout petit sous-ensemble de *java* ; les *applets* ne sont limités quant aux appels des fonctions *java* que pour des problèmes de sécurité.

On commence par décrire le mode de base, puis on dira quelques mots sur le *javascript* et sur les *applets* avant de revenir sur la syntaxe générale du langage.

9.2 Un premier programme Java

Le programme suivant doit être sauvegardé sous forme de *texte simple* dans le fichier *hello.java*. Le nom est obligatoirement celui de la classe de base définie par le programmeur suivi de `.java`.

Listing 9.1 *(hello.java)*

```
public class hello{
  public static void main(String[] args) {
      System.out.println("Bonjour");
      }
  }
```

Ce programme écrit le mot `Bonjour` sur la console. Le principe est de surcharger la classe `main` qui est une classe virtuelle dans un environnement prédéfini par le langage. C'est la grande

différence entre *java* et `C /C++` : l'utilisateur modifie un «univers»pré-défini, il «hérite»de toute une structure de classe, ici simplement la classe `System`.

9.2.1 L'environnement de programmation

SUN Micro-Systems distribue gratuitement un compilateur *java* et un environnement de développement, appelé `JDK`, initiales de *Java Development Kit*.

L'utilisation du compilateur est standard : une fois le programme écrit dans un fichier `.java` (à l'aide d'un éditeur de texte quelconque), le compilateur `javac` est appelé par la commande :
`javac hello.java`
Le compilateur va générer un fichier exécutable dans l'environnement *java* de nom `hello.class`.
L'exécution sera lancée dans `JDK` par :

```
java hello
```

Comme résultat de l'exécution du programme, le texte `Bonjour` est affiché à l'écran.

Remarque 9.1

Il faut néanmoins s'assurer que les chemins d'accès aux programmes `javac, java, classes.zip` *et* `hello.java` *sont connus au moment de l'exécution. Sous Windows 98, il faut ajouter les lignes suivantes au fichier* `autoexec.bat` *:*

```
SET PATH=C:\jdk1.3.1\bin;C:\jdk1.3.1\lib;
```

Sous Windows 2000 et XP, cet ajout se fait par la modificiation de variable d'environnement dans le tableau de bord «System«(cliquer avec le bouton de droite sur l'icone «Poste de travail», puis chercher la section «avancé»). Bien entendu, on a supposé que la version 1.3.1 du `JDK` *a été installée sur le disque C (pour plus de details voir http ://java.sun.com/j2se/1.3/install-windows.html. Les versions antérieures avaient aussi besoin de la variable d'environnement* `CLASSPATH` *que l'on définissait par*

```
SET CLASSPATH=C:\jdk1.3.1\lib\classes.zip
```

Désormais cette variable est par défaut le dossier courant.

Le compilateur `javac` et l'interpréteur `java` sont disponibles[1] pour de multiples systèmes d'exploitation : Unix, Windows, Mac OS, etc. Pour assurer la compatibilité et la portabilité des programmes *java* d'une plate-forme à l'autre, `javac` produit un pseudo-code universel qui peut être interprété par un *moteur java*, faisant éventuellement partie du système d'exploitation de l'ordinateur.

En dehors de la portabilité des programmes *java*, deux autres caractéristiques du langage doivent être soulignées :

1. La capacité des navigateurs Internet (*Web browsers*) d'exécuter du pseudo-code *java*. Les programmes de type `applets` et `javascript` sont largement utilisés pour ce type d'applications.

[1] On trouvera un didacticiel en anglais à l'adresse `http://java.sun.com/docs/books/tutorial`.

2. Le souci de garder l'efficacité et la vitesse d'exécution des programmes. Les compilateurs récents produisent des environnements d'exécution indépendants et des environnements intermédiaires, appelés `JIT` (*just in time compilers*), et sont capables de compiler le programme au fur et à mesure qu'ils l'interprètent. Avec cette technique, l'exécution pour la première fois d'un bloc d'instructions est lente, mais la vitesse d'exécution est très rapide à partir de la deuxième utilisation du code généré.

Le langage *java* est aujourd'hui très populaire grâce à l'existence de plusieurs *interfaces graphiques utilisateur* (GUI) et *plate-formes java* avec des environnements de développement des applications ; les plus utilisées sont `Visual J++` de Microsoft, `Java-café` de Symantec, `Java IDE` de Metrowerks.[2]

Java utilise une syntaxe `C++` simplifiée, basée sur l'utilisation intensive des classes dérivées. Pour des objectifs de clarté, *java* cache les pointeurs, l'allocation contrôlée de la mémoire dynamique (le destructeur `delete` n'existe pas) et la surcharge d'opérateurs.

Les concepteurs du langage ont considéré que les pointeurs et la gestion de la mémoire dynamique étaient les principaux concepts du `C` qui rendaient ce langage compliqué. Néanmoins, l'absence de la possibilité de surcharger les opérateurs est un réel inconvénient du langage pour l'analyse numérique, qui sera, paraît-il, pris en compte dans les prochaines versions de la norme *java*.

La vitesse d'exécution lente étant très pénalisante pour le calcul scientifique, les programmes *java* sont aujourd'hui utilisés exclusivement pour la partie interface utilisateur de programme et pour les applications Web. Mais les choses changent...

9.3 Javascript

Voici le programme *hello.java* en version *javascript* :

Listing 9.2 *(jscript.htm)*

```
 <HTML><BODY><SCRIPT language="javascript">
document.writeln("Bonjour, <B>Javascript</B> est support{e},")
</SCRIPT></BODY></HTML>
```

Ces sont des commandes à l'intérieur d'un fichier `html` (le langage des pages Web). Tout ce qui est entre les mots clefs <script> et </script> est du *javascript* ; on reconnaît des instructions assez proches du langage `C++` , en particulier // pour les commentaires.

Tous les navigateurs ne supportent pas *javascript*. Le programme suivant (fichier javascript.htm, écrit en collaboration avec Pascal Havé) effectue les opérations suivantes :

1. On commence par tester si *javascript* est pris en charge en faisant afficher la chaîne `Javascript est supporté, on peut continuer...` (avec le premier mot en gras et deux sauts de ligne, d'où les directives `html` <b> et
).

[2] On trouvera un environnement gratuit pour Windows et MacOSX (et Linux) à l'adresse `http://www.bluej.org`.

2. Ensuite, on démontre les possibilités mathématiques de *javascript* en affichant $\sqrt{2}$.
3. Puis on implémente une fonction qui a pour effet d'envoyer l'utilisateur sur la page Web du laboratoire Jacques-Louis Lions de Jussieu.
4. Enfin, on crée deux boutons, l'un qui appelle la fonction `show` et l'autre qui affiche un nombre aléatoire.
5. Pour finir, on imprime la date du jour.

Listing 9.3 *(javascript.htm)*

```
<HTML><BODY>
<SCRIPT language="javascript">
                //    On vérifie que le navigateur prend en charge javascript.
        //    Utiliser la classe correspondante à la fonction : ici 'document'
document.writeln("<B>Javascript</B> est support{e},"
                        + " on peut continuer...<BR><BR>")
document.write("Calcul de sqrt(2) = ") a = 2;
  //    Une variable document.writeln(Math.sqrt(a)) qui utilise la classe Math
                                //    Cette fonction va piloter le navigateur
function show() {
  location.replace ("http:                          //   www.ann.jussieu.fr");
  alert("On change de page Web : direction le LAN");
}
</SCRIPT><BR><BR>
<FORM> <!-- Utilisation de Javascript {a} partir de la class form--!>
<INPUT TYPE=button VALUE="Click here" onclick="javascript:show()">
<BR><BR>
<INPUT TYPE=button VALUE="Nombre Al{e}atoire"
      OnClick="form.num.value=parseInt(Math.random()*100)">
<INPUT TYPE="text" NAME="num" SIZE=10> </FORM><BR><HR>
<!  Cr{e}ation d'une nouvelle instance de la classe Date!>
<SCRIPT language=javascript> today = new Date();
    dateGMT = today.toGMTString();
    document.write("Date au format GMT:"+dateGMT);
</SCRIPT>
</BODY></HTML>
```

On ouvre cette page avec un navigateur compatible *javascript* (Netscape ou Internet Explorer (IE), par exemple). Si le script a une erreur, elle s'affichera en bas de page ,sinon le programme s'exécutera ; on a ainsi rendu la page Web interactive.

Exercice 9.1

Modifier le programme précédent *javascript.htm* (listing 9.2) pour qu'il affiche le nombre premier suivant dans la liste des m nombres premiers, à chaque clique de souris.

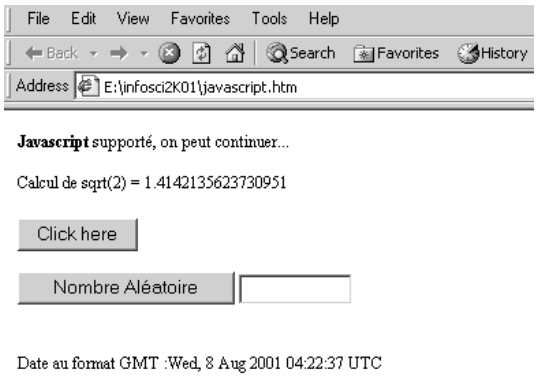

FIG. 9.1 – Zoom sur la fenêtre d'Internet Explorer, créee par le programme *javascript.htm* au démarrage.

9.4 Interface Utilisateur Java

L'assimilation du principe des classes dérivées du `C++` est essentielle pour la compréhension du fonctionnement du langage *java*.

Exercice 9.2

Traduire en `C++` le programme *hello.java* (listing 9.1) en écrivant une classe `system`.

Java fournit une série de bibliothèques contenant des fonctions virtuelles prédéfinies. L'utilisateur va spécialiser ces fonctions suivant ses besoins. Par conséquent, le succès des applications dépend de la qualité des bibliothèques. Actuellement, `swing` est la bibliothèque de base pour les applications *java* version 2 ; `java.awt` (*abstract window toolkit*) est une bibliothèque plus classique, celle du *java* version 1. Cette dernière bibliothèque contient toutes les fonctions nécessaires à la manipulation des fenêtres, menus et boutons.

Le programme suivant va ouvrir une fenêtre principale, ensuite une fenêtre *dérivée* avec trois boutons. Les fonctions `add`, `pack` et les classes pour toutes les variables sont définies dans la bibliothèque `java.awt`.

Listing 9.4 *(winmw.java)*

```
import java.awt.*;
import java.awt.event.*;
public class winmw extends Dialog{
    protected Button bouton1, bouton2, bouton3;
    public winmw(Frame parent, String title)
    {
        super(parent, title, false);              //    création de la fenêtre
        this.setLayout(new BorderLayout(100,100));
        bouton1 = new Button("OK");
        this.add("East",bouton1);
```

```
        bouton2 = new Button("Not OK");
        this.add("West",bouton2);
        bouton3 = new Button("Cancel");
        this.add("South",bouton3);
        this.pack();
        bouton1.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e)
            {   System.out.print("Yes");}
        });
        bouton2.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e)
            {   System.out.print("No");}
        });
        bouton3.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e)
            {   System.out.print("Cancel");System.exit(0);}
        });
}
     public static void main(String[] args){
        Frame f = new Frame("win Test");
        f.setSize(400,400);
        Label theline = new Label("From D. Flanagan 1996-99");
        f.add("Center",theline);
        f.show();
        winmw d = new winmw(f,"A window with 2 buttons");
         d.show();
    }
}
```

9.5 Applets

Il existe une troisième façon de travailler, en fabriquant une *applet*, c'est a dire une petite application qui est appelée par le navigateur Web (Netscape, Internet Explorer...). Voici à nouveau le programme `hello.java` resservi à la sauce *applet* :

Listing 9.5 *(HelloApplet.java)*

```
import java.applet.Applet; import java.awt.Graphics;
public class HelloApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("Bonjour!", 40, 40);
    }
}
```

On va compiler ce fichier par (attention aux majuscules) :

```
javac HelloApplet.java
```

mais pour l'exécuter il faut créer une page `HTML`

Listing 9.6 *(HelloApplet.htm)*

```
<HTML><BODY>
 <APPLET code="HelloApplet.class", width=200, height=200> </APPLET>
</BODY></HTML>
```

que l'on ouvrira avec un navigateur. Il apparaîtra alors `bonjour` sur fond gris, dans une fenêtre 200×200.

La bibliothèque `java.applet` a été conçue pour pouvoir lancer des applications *java* à partir d'un navigateur Web. Voici un deuxième exemple :

Listing 9.7 *(aplet.java)*

```
                                                    //    fichier: aplet.java
import java.applet.*; import java.awt.*;
public class aplet extends Applet {
    public void init() {    }
    public void paint( Graphics g ) {
        g.drawString( "Bonjour tous le monde!", 30, 130 );
        g.setColor(Color.pink);
        g.fillOval(10,10,100,100);
    }
}
```

Le programme doit être compilé avec la commande
`javac aplet.java`.
L'exécution peut être lancée à partir de tout navigateur Web, compatible *java*, et qui contient la ligne suivante :

```
<APPLET code="aplet.class", width=200, height=200> </APPLET>
```

Exemple
Le fichier *aplet.htm* suivant peut être ouvert par un navigateur directement pour tester l'*applet* `aplet.class`

Listing 9.8 *(aplet.htm)*

```
 <HTML><BODY>
 <APPLET code="aplet.class", width=200, height=200> </APPLET>
</BODY></HTML>
```

Le résultat est affiché sur la figure 9.2.

Pour bien comprendre la différence entre une *applet* et un programme autonome nous donnons la même application en programme autonome

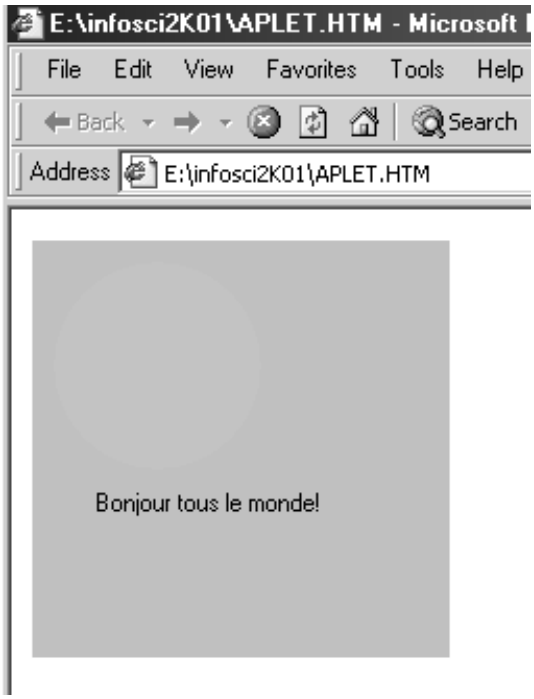

FIG. 9.2 – Zoom sur la fenêtre d'IE crée par le programme *aplet.htm* au démarrage.

Listing 9.9 *(apletmain.java)*

```
                                                    //    fichier : apletmain.java
import java.awt.*; public class apletmain  extends Frame{
    public apletmain(){
        super();
        resize(200,200);
        Graphics g  = this.getGraphics();
        g.drawString( "Bonjour tous le monde!", 30, 130 );
        g.setColor(Color.pink);
        g.fillOval(10,10,100,100);
    }
    public static void main(String[] args){
         apletmain f = new apletmain();
         f.show();
     }
}
```

Exercice 9.3 Traduire le programme du paragraphe 9.3 en *applet*.

Exercice 9.4 Traduire le programme *ailette.cpp* (listing 1.1) en *java*.

Exercice 9.5 Traduire le programme *ailette.cpp* (listing 1.1) en *java-applet* en ignorant la partie lecture/écriture de fichiers.

9.6 Une application plus compliquée

Nous allons réaliser ici une interface avec un programme externe, `nsc2ke`[3] qui calcule les écoulements fluides turbulents dans des configurations 2D. Le programme `nsc2ke` a besoin pour fonctionner de lire un fichier de données nommé `DATA` et un fichier contenant un maillage (triangulation).

L'interface *java* permettra à l'utilisateur :
- de modifier les données dans `DATA` ;
- de spécifier le fichier de maillage qui sera utilisé pour le calcul ;
- de visualiser et explorer (zoom) le maillage indiqué ;
- et, finalement, de lancer le programme externe `nsc2ke`, avec le nouveau set de données.

On commence donc par lire le fichier `DATA` et afficher son contenu dans des champs éditables ; l'utilisateur peut ensuite modifier ces champs et les nouvelles valeurs seront écrites dans `DATA` pour un nouveau calcul. La lecture et l'affichage de la triangulation se font suivant les techniques exposées dans le chapitre 4.

Remarque 9.2 ‖ *Nous n'avons pas utilisé l'interface* applet *car nous devons lire des fichiers.*

L'interface utilisateur possède trois boutons :
– Le bouton «load mesh»qui lorsqu'il est activé appelle le dialogue de choix d'un fichier et affiche la triangulation contenue.
– Le bouton «reset zoom»qui permet de réafficher la triangulation avec les valeurs initiales des paramètres.
– Le bouton «run job»qui écrit dans `DATA` les valeurs des paramètres (éventuellement modifiés par l'utilisateur), lance `nsc2ke` et termine ce programme *java*.

Une fois la triangulation sélectionnée, un zoom peut être obtenu en traînant la souris, bouton enfoncé, d'un point P0 de l'écran à un autre point P1. Le contenu du rectangle de sommet supérieur gauche P0 et de sommet inférieur droit P1 sera affiché.

Listing 9.10 *(nsc2ke.java)*

```
                                                    //    fichier nsc2ke2.java
import java.awt.*;
import java.io.*;
import java.awt.event.*;
public class nsc2ke2 extends Dialog implements MouseListener
{
    String dfname = "DATA";                  //    pour communiquer avec nsc2ke
    int nv, nt;
    Triangle t[];
    Vertex v[];
    int offset = 80, hmax =380 , vmax =180, downoffset = 40 + offset;
    double scale, oldscale, xmin,xmax,ymin,ymax,oldxmin,oldxmax,oldymin
          ,oldymax,zoom0x,zoom1x,zoom0y,zoom1y;
    Button loadtriau, resetzoom, runjob;
    boolean mesh=false;
    TextField textfield0,textfield1, textfield2, textfield3, textfield4, textfield5;
    Label reynolds, mach, angle, cfl, nstep, ftime;
```

[3] `http://www-rocq.inria.fr/gamma/cdrom/www/nsc2ke/fra.htm`

```
    Checkbox checkbox0, checkbox1, checkbox2,checkbox3;
    CheckboxGroup fluidcheckboxes, timecheckboxes;
    double dreynolds=1000.0, dmach=0.95, dangle=0.1, dcfl=1.0, dtime=2.0;
    int dwhat0=1, dwhat1=1, dnstep=200;
public class Vertex { int where; double x,y; void Vertex(){where=0; x=0; y=0;}}
public class Triangle { int where; int[] v; void Triangle(){where=0; }}
public  nsc2ke2(Frame parent)
{
        super(parent, "nsc2ke", false);
        this.setSize(hmax+2*offset, vmax+4*offset);
    }
public class  getfilename extends Frame
{
  FileDialog fd;
  public String fname;
    public getfilename() { }
    public String getthename()
    {
        fd = new FileDialog(this, "Choose a mesh",FileDialog.LOAD);
        fd.show();
        fname = fd.getFile();
        return fname;
    }
}
public  void readtriau(String fname){
        try{
        FileInputStream filename = new FileInputStream(fname);
        StreamTokenizer file = new StreamTokenizer(filename);
      file.parseNumbers();
         file.nextToken(); nv = (int)file.nval;
         file.nextToken(); nt= (int)file.nval;
         t = new Triangle[nt];
         v = new Vertex[nv];
         for(int i = 0; i<nv;i++)
         {
            v[i] = new Vertex();
            file.nextToken(); v[i].x = file.nval;
            file.nextToken(); v[i].y = file.nval;
            file.nextToken(); v[i].where = (int)file.nval;
         }
         for(int i = 0; i<nt;i++)
         {
            t[i] = new Triangle(); t[i].v = new int[3];
            for(int j=0;j<3;j++)
            {
                file.nextToken();
                t[i].v[j] = (int)file.nval -1;
            }
            file.nextToken(); t[i].where = (int)file.nval;
         }
         xmin=v[0].x; xmax = v[0].x; ymin=v[0].y; ymax = v[0].y;
         for(int i = 1; i<nv;i++)
         {
            if(v[i].x < xmin) xmin = v[i].x;
            if(v[i].x > xmax) xmax = v[i].x;
            if(v[i].y < ymin) ymin = v[i].y;
```

```
            if(v[i].y > ymax) ymax = v[i].y;
          }
          scale = hmax / (xmax - xmin);
          if( vmax/ (ymax - ymin) < scale ) scale = vmax/ (ymax - ymin);
          oldscale = scale;
          oldxmin = xmin; oldxmax = xmax;
          oldymin = ymin; oldymax = ymax;
         }
         catch (IOException Ex)
         {
             System.out.println(Ex.getMessage());
         }
    }
public void rline( Graphics g, double x0, double x1, double y0, double y1)
{
                int h0 = offset+(int)(scale * (x0 - xmin));
                int h1 = offset+(int)(scale * (x1 - xmin));
                int v0 = downoffset+(int)(scale * (y0 - ymin));
                int v1 = downoffset+(int)(scale * (y1 - ymin));
                g.drawLine(h0,v0,h1,v1);
}
public void plottriau( Graphics g)
    {
        g.setColor(Color.blue);
         for(int k = 0; k<nt;k++)
            for(int j = 0; j<3; j++)
            {
                int i = t[k].v[j];
                int jp = j+1; if(jp==3) jp = 0;
                int ip = t[k].v[jp];
                rline(g, v[i].x, v[ip].x, v[i].y, v[ip].y);
            }
    }
public void myinterface(){
        this.setLayout(new FlowLayout());
        loadtriau = new Button("Load Mesh");this.add(loadtriau);
        resetzoom =new Button("Reset Zoom");this.add(resetzoom);
        runjob = new Button("Run Job");     this.add(runjob);
        reynolds = new Label("Re");
        textfield0 = new TextField(Double.toString(dreynolds),5);
        this.add(reynolds);                 this.add(textfield0);
        mach = new Label("Mach");
        textfield1 = new TextField(Double.toString(dmach),2);
        this.add(mach);                     this.add(textfield1);
        angle = new Label("Angle ");
        textfield2 = new TextField(Double.toString(dangle),2);
        this.add(angle);                    this.add(textfield2);
        cfl = new Label("CFL");
        textfield3 = new TextField(Double.toString(dcfl),2);
        this.add(cfl);                      this.add(textfield3);
        nstep = new Label("nStep");
        textfield4 = new TextField(Integer.toString(dnstep),3);
        this.add(nstep);                    this.add(textfield4);
        ftime = new Label("Time");
        textfield5 = new TextField(Double.toString(dtime),3);
        this.add(ftime);                    this.add(textfield5);
```

```
        fluidcheckboxes = new CheckboxGroup();
        checkbox0 = new Checkbox("Euler",fluidcheckboxes,dwhat0==0);
        checkbox1 = new Checkbox("NS",fluidcheckboxes,dwhat0==1);
         this.add(checkbox0);                this.add(checkbox1);
        timecheckboxes = new CheckboxGroup();
        checkbox2 = new Checkbox("Unsteady",timecheckboxes,dwhat1==0);
        checkbox3 = new Checkbox("Steady",timecheckboxes,dwhat1==1);
        this.add(checkbox2);                this.add(checkbox3);
        this.pack();
        Graphics g = this.getGraphics();
        loadtriau.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e)
            {
                String name;
                getfilename ff = new getfilename();
                name = ff.getthename();
                readtriau(name);
                mesh = true;
                repaint();
            }
        });
        resetzoom.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e)
            {
                xmin = oldxmin;
                xmax = oldxmax;
                ymin = oldymin;
                ymax = oldymax;
                scale = oldscale;
                repaint();
            }
        });
        runjob.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e)
            {
                    savemyparam();
                    Runtime a = Runtime.getRuntime();
                    try{ a.exec("nsc2ke");}
                        catch (IOException Ex)
                            {   System.out.println(Ex.getMessage()); }
                    System.exit(0);
              }
        });
        this.addMouseListener(this);
        addMouseListener(this);
}
public void getrunparam()
    {
        try{
            FileInputStream filename = new FileInputStream(dfname);
            StreamTokenizer file = new StreamTokenizer(filename);
            file.parseNumbers();
            file.nextToken(); dwhat0 = (int)file.nval;
            file.nextToken(); dreynolds = file.nval;
            file.nextToken(); dmach = file.nval;
            file.nextToken(); dangle = file.nval;
```

```
            file.nextToken(); dwhat1 = (int)file.nval;
            file.nextToken(); dcfl = file.nval;
            file.nextToken(); dnstep = (int)file.nval;
            file.nextToken(); dtime = file.nval;
        }
        catch (IOException Ex)
        {
            System.out.println(Ex.getMessage());
        }
    }
public void savemyparam()
    {   String dreynolds = textfield0.getText();
        String dmach = textfield1.getText();
        String dangle = textfield2.getText();
        String dcfl = textfield3.getText();
        String dnstep = textfield4.getText();
        String dtime = textfield5.getText();
        String dwhat0, dwhat1;
        try{
            FileOutputStream filename= new FileOutputStream(dfname);
            PrintStream ffile = new PrintStream(filename);
            if(checkbox0.getState()) ffile.println("0");
                else ffile.println("1");
            ffile.println(dreynolds);
            ffile.println(dmach);
            ffile.println(dangle);
            if(checkbox2.getState()) ffile.println("0");
                else ffile.println("1");
            ffile.println(dcfl);
            ffile.println(dnstep);
            ffile.println(dtime);
        }
        catch (IOException Ex)
        {
            System.out.println(Ex.getMessage());
        }
    }
public void paint( Graphics g ) {
        if(mesh) plottriau(g);
    }
public void mousePressed(MouseEvent e)
    {   int x=e.getX(), y=e.getY();
        zoom0x = (x-offset)/scale+xmin;
        zoom0y = (y-downoffset)/scale+ymin;
    }
public void mouseReleased(MouseEvent e)
    {   int x=e.getX(), y=e.getY();
        zoom1x = (x-offset)/scale+xmin;
        zoom1y = (y-downoffset)/scale+ymin;
        double w0 =zoom0x-zoom1x;
        if(w0 < 0)  w0 = - w0;
        if(zoom0y-zoom1y < 0) w0 -= zoom0y-zoom1y; else w0 += zoom0y-zoom1y;
        if(w0 > 1)
        {
            xmin = zoom0x;
            xmax = zoom1x;
```

```
            ymin = zoom0y;
            ymax = zoom1y;
            scale = hmax / (xmax - xmin);
        }
        repaint();
    }
public void mouseEntered(MouseEvent e) {;}
public void mouseExited(MouseEvent e)  {;}
public void mouseClicked(MouseEvent e) {;}
public static void main(String[] args){
        Frame f = new Frame("NSC2KE Java Interface");
        nsc2ke2 b = new nsc2ke2(f);
        b.getrunparam();
        b.myinterface();
        b.setSize(640,400);
        b.show();
    }
}
```

La figure 9.3 montre l'interface utilisateur après lecture du fichier maillage `NACA.MSH`.

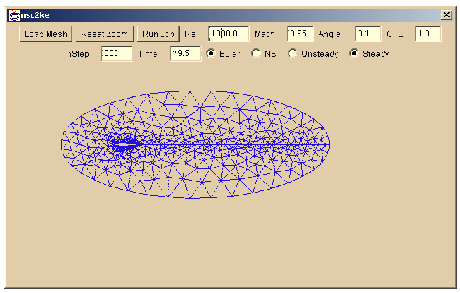

FIG. 9.3 – La fenêtre créée par le programme `nsc2ke.java` après lecture d'une triangulation.

Exercice 9.6

Rajouter une fenêtre pour afficher la triangulation dans une fenêtre différente de celle des constantes.

Exercice 9.7

Écrire une *applet* pour tracer une fonction. Tester avec la fonction $y = sin(x)$.

Exercice 9.8

Continuer les exercices 9.2 et 9.3 pour qu'ils affichent les résultats sous forme graphique.

9.7 Compléments

9.7.1 Communications java C/C++

Pour appeler du *java* une fonction `C` , il faut procéder en cinq étapes. Voici un exemple avec un programme principal `CditBonjour.java` qui appelle une fonction `C` définie dans le fichier `C bonjourEnC.c`. Pour réaliser la communication *java-C*, il faut :

1. compiler le fichier source *java* par la commande :
   `javac CditBonjour.java`
2. créer un fichier d'interface `CditBonjour.h` avec l'utilitaire *javah* par la commande :
   `javah -jni CditBonjour`
3. compiler le fichier source `C` par la commande :
   `gcc -c bonjourEnC.c`
   ce qui produit l'objet `bonjourenC.o` [4]
4. Traduire l'objet `bonjourenC.o` en librairie dynamique (sur le PC : une *dll*), par l'une des commandes :
   – Sur PC-Cygwin : `gcc -shared -o bonjour.dll bonjourenC.o`
   – Sur MacOSX : `gcc -bundle -o libbonjour.jnilib bonjourenC.o`
   – Sur PC-Linux :
5. lancer le programme par la commande :
   `java CditBonjour`.
   La chaîne
   `Bonjour tout le monde!`
   doit s'afficher sur la console.

Remarque 9.3 *En fin de compilation les fichiers suivants sont présents :*

```
CditBonjour.class  CditBonjour.java  bonjourEnC.c
CditBonjour.h      bonjour.dll       bonjourEnC.o
```

Analysons le mécanisme, en commençant par le programme principal *java* :

Listing 9.11 *(CditBonjour.java)*

```
                                                    //    fichier CditBonjour.java
class CditBonjour {
    public native void ditBonjour();
    static {
        System.loadLibrary("bonjour");
    }
    public static void main(String[] args) {
        new CditBonjour().ditBonjour();
    }
}
```

[4] ou bien par exemple, `gcc -I/System/Library/Frameworks/JavaVM.framework/Headers -c bonjourEnC.c` si la librairie *java* n'est pas trouvée.

Il fait appel à une librairie `bonjour` chargée par la fonction `loadLibrary`, et qui suppose qu'un fichier associé (ici de nom `bonjour.dll`) soit présent. Celui-ci est construit à partir du programme `C` :

Listing 9.12 *(bonjourEnC.c)*

```
                                                    //    fichier bonjourEnC.c
#include <jni.h>
#include "CditBonjour.h"
#include <stdio.h>
JNIEXPORT void JNICALL
Java_CditBonjour_ditBonjour(JNIEnv *env, jobject obj)
{
    printf("Bonjour tout le monde!\n");
    return;
}
```

Remarquez la présence du fichier `CditBonjour.h` ; il est généré automatiquement par l'utilitaire *javah* normalement présent sur toute distribution *java*. S'il y avait un passage de paramètre c'est ici qu'il serait géré (voir [Campione et Walrath-1996],[Daconta-1996] ou [Flanagan-1996] par exemple).

Exercice 9.9

Faire une interface *java* qui appelle le programme *ailette.cpp* (listing 1.1) sous forme de fonction `C` (lire la documentation *java* pour savoir comment appeler une fonction `C` du *java*).

Exercice 9.10

Même exercice que ci-dessus mais avec en plus un calcul de sensibilité par rapport à T_0 comme dans l'exercice 8.3.