TEHNICI AVANSATE DE PROGRAMARE
LUCRARE DE LABORATOR 6

Applet–uri Java. Componentele interfeţei grafice. Evenimente generate de componentele AWT

I. SCOPUL LUCRĂRII

Lucrarea de faţă are rolul de a prezenta şi familiariza studentul cu modul de construire a unei interfeţe grafice utilizator si cu modul de tratare a evenimentelor generate de componentele grafice. Se vor prezenta câteva componente vizuale utile, împreună cu modul de creare şi dispunere a acestora pe suprafaţa unui applet.

La sfârşitul acestei lucrări, studentul va avea posibilitatea să scrie mini-aplicaţii Java în care să utilizeze noţiunile învăţate si să scrie applet-uri Java cu interfaţă grafică complet funcţională.

II. NOŢIUNI TEORETICE

Observaţie: Vezi noţiunile prezentate în cursul 5.

1. Construirea unei interfeţe grafice

Interfeţele grafice utilizator **GUI** (graphic user interface) sunt o parte importantă a oricărui program. Pachetul **java.awt** (Abstract Window Toolkit) furnizează funţionalităţi extinse în acest sens.

1.1 Componentele interfeţei grafice

Componentele Java sunt implementate printr-o serie de subclase ale superclasei **java.awt.Component** şi ale superclasei **java.awt.MenuComponent.**

O modalitate de a grupa componentele grafice este următoarea: componente vizuale, componente container şi componene meniu.

Există 11 componente vizuale. În cele ce urmează vom prezenta numai o parte dintre acestea: **Button, Checkbox, Choice, Label, List, TextArea, TextField**. Pentru a utiliza una dintre acestea într-o GUI, se va crea întâi o instanţă prin apelarea constructorului corespunzător, apoi se va adăuga componenta unui container.

Există 4 componente container: **Applet, Frame, Panel, Dialog** (vor fi prezentate în următoarele lucrări de laborator). Container-ele sunt componente capabile să conţină alte componente în cadrul lor.

Câteva metode sunt implementate de toate componentele vizuale şi container, în virtutea moştenirii din clasa **java.awt.Component**.

- ***getSize( )*** – returnează dimensiunea unei componente. Tipul returnat este **Dimension**, care are datele membru publice **height** şi **width**.
- ***setForeground( ), setBackground( )*** – setează culoarea de scriere a textului şi respectiv

culoarea de fond pentru o componentă. Fiecare primeşte ca argument o instanţă a clasei **java.awt.Color**. Dacă aceste culori nu se vor seta explicit, se vor utiliza culorile implicite ale containerului căruia îi aparţine componenta.

• ***setFont( )*** – determină font-ul pe care o componentă îl va utiliza la scrierea textului. Dacă nu se setează explicit un font, componenta va utiliza font-ul containerului său.

• ***setSize( ), setBounds( )*** – stabilesc dimensiunile unei componente. Metoda **setSize()** primeşte 2 argumente: width (lăţime) şi height (înălţime). Metoda **setBounds()** stabileşte atât poziţia cât şi dimensiunea. Poziţia este specificată relativ la containerul componentei, sau în cazul unei ferestre relativ la ecran. O formă a metodei are 4 argumente: x, y , width şi height.

• ***setVisible( )*** – primeşte un argument de tip Boolean şi stabileşte dacă componenta va fi vizibilă sau nu. Se utilizează în general pentru Frame-uri.

1.2 Layout Managers

Alte sisteme GUI încurajează programatorul să gândească în termenii unei specificări precise a dimensiunii şi locaţiei componentelor interfeţei grafice. Java, în schimb, furnizează o serie de 5 **layout managers**, fiecare având propria politică de dispunere a componentelor în containere.

Fiecare componentă Java are o **dimensiune preferată**. Dimensiunea preferată este în general cea mai mică dimensiune necesară pentru a reprezenta componenta într-un mod vizual ce are sens. De exemplu, dimensiunea preferată a unui buton este dimensiunea etichetei sale, plus o mică margine de spaţiu liber din jurul textului, plus decoraţiunile care marchează marginea butonului. Dimensiunea preferată este dependentă de platformă, deoarece decoraţiunile marginii componentei variază de la un sistem la altul. Când un **layout manager** aşează componentele unui container, trebuie să ia în considerare 2 criterii: politica sa şi dimensiunea preferată a fiecărei componente. (Prima prioritate este de a respecta politica de aşezare a componentelor, ignorându-se dimesiunea preferată a componentelor.)

În această lucrare de laborator vom prezenta numai două dintre cele 5 **layout mangers**, şi anume **Flow Layout Manager** şi **Border Layout Manager**.

Flow Layout Manager aranjează componentele în rânduri orizontale. Este managerul implicit pentru applet-uri şi pentru panel-uri. Întotdeauna onorează dimensiunea preferată a componentelor.

Border Layout Manager este managerul implicit pentru frame-uri. Îşi divide teritoriul în 5 regiuni: NORTH, SOUTH, EAST, WEST şi CENTER. Fiecare regiune poate fi vidă sau poate conţine o singură componentă.

2. Exemple

```
// App1.java
import java.applet.Applet;
import java.awt.*;

public class App1 extends Applet
{
   public void init()
   {
     setLayout(new FlowLayout(FlowLayout.RIGHT));
     Label label=new Label("Introduceti text: ");
     add(label);

     TextField text1=new TextField("");
     add(text1);
```

```
      TextField text2=new TextField("textul 2");
      add(text2);
      Button but=new Button("OK");
      add(but);

   }
   public void paint(Graphics g)
   {
      g.drawString("Acesta este un exemplu",30,100);
   }
}


// App1.html
<html>
<body>
<applet code=App1.class width=250 height=400>
</applet>
</body>
</html>
```

Exerciţiu:

- Comentaţi prima linie de cod din metoda **init()** a applet-ului.
- Modificaţi în **App1.html** lăţimea applet-ului; setaţi **width=350** şi reîncărcaţi pagina html.

În ambele situaţii observaţi şi explicaţi rezultatele.

```
//App2.java
import java.applet.Applet;
import java.awt.*;

public class App2 extends Applet
{
  public void init()
  {
    setLayout(new BorderLayout());
    Panel toolbar=new Panel();
    toolbar.setLayout(new FlowLayout(FlowLayout.LEFT));
    toolbar.setBackground(Color.orange);
    toolbar.add(new Button("Buton 1"));
    toolbar.add(new Button("Buton 2"));
    add(toolbar,BorderLayout.NORTH);
    TextArea txA1=new TextArea();
    StringBuffer s=new StringBuffer("Acesta este un text \n mai
                                    lung. Ca sa observati ");
    s.append("\n plasarea \n sagetilor \n de defilare");
    txA1.setText(s.toString());
    txA1.setFont(new Font("Arial",Font.ITALIC,24));
    txA1.setBackground(Color.blue);
   txA1.setForeground(Color.white);
   add(txA1,BorderLayout.CENTER);

   TextArea txA2=new TextArea("Al doilea text.",5,20);
   txA2.setFont(new Font("Monospaced",Font.ITALIC|Font.BOLD,20));
   txA2.setBackground(Color.white);
   txA2.setForeground(Color.blue);
```

```
   txA2.setEditable(false);
   add(txA2,BorderLayout.SOUTH);
   Checkbox chk=new Checkbox("Bifati aici!");
   add(chk,BorderLayout.WEST);
   Panel option=new Panel();
   CheckboxGroup chgroup=new CheckboxGroup();
   option.add(new Checkbox("Prima",false,chgroup));
   option.add(new Checkbox("A doua",true,chgroup));
   add(option,BorderLayout.EAST);
  }
}

//App2.html
<html>
<body>
<applet code=App2.class width=400 height=400>
</applet>
</body>
</html>


//App3.java
import java.applet.Applet;
import java.awt.*;
public class App3 extends Applet
{
   public void init()
   {
     setBackground(Color.cyan);
     setFont(new Font("Arial",Font.BOLD,16));
     Label label1=new Label("Checkbox:");
     add(label1);
     Choice ch=new Choice();
     ch.addItem("Prima");
     ch.addItem("A doua");
     ch.addItem("A treia");
     ch.setForeground(Color.red);
     add(ch);
     Label label2=new Label("List:");
     add(label2);
     List list=new List(3,true);
     for(int i=1;i<10;i++)
       list.add("Floare "+i);
     list.setForeground(Color.blue);
     list.setFont(new Font("Courier new",Font.ITALIC,20));
     add(list);
   }
}

//App3.html
<html>
<body>
<applet code=App3.class width=500 height=200>
</applet>
</body>
</html>
```

3. Clasele Eveniment

Subclasele clasei **java.awt.AWTEvent** reprezintă diferite tipuri de evenimente care pot fi generate de diferitele componente AWT. Aceste subclase conţin toate informaţiile necesare cu privire la activitatea care a declanşat evenimentul. Iată care sunt:

- **ActionEvent** – generat de activarea unei componente
- **AdjustmentEvent** – generat prin ajustarea componentelor ajustabile cum ar fi "scroll bars"
- **ContainerEvent** – generat când componentele sunt adăugate sau scoase dintr-un container
- **FocusEvent** – generat când o componentă primeşte sau pierde focus-ul asupra ei
- **ItemEvent** – generat când o opţiune este selectată dintr-o listă (Choice, List) sau Checkbox
- **KeyEvent** – generat de activitatea tastaturii
- **MouseEvent** - generat de activitatea cu mouse-ul
- **PaintEvent** - generat când o componentă este desenată
- **TextEvent** – generat când o componentă text este modificată
- **WindowEvent** – generat de activitatea cu fereastra (iconificarea, deziconificarea)

4. Event Listeners

Există două moduri de a trata evenimentele prezentate mai sus. Primul este de a delega tratarea evenimentului unui obiect care "ascultă" (**listens**). Al doilea este de a permite explicit componentei care generează evenimentele să-şi manipuleze propriile evenimente.

Iată ce presupune prima variantă. Să considerăm următorul exemplu. Fie un buton într-un applet. Când butonul este apăsat, un eveniment **ActionEvent** va fi trimis unei instanţe a clasei **MyActionListener**. Clasa **MyActionListener** implementează interfaţa **ActionListener**, garantându-se astfel prezenţa metodei **actionPerformed()**.

```
class MyActionListener implements ActionListener  {
       public void actionPerformed(ActionEvent e)  {
                System.out.println("Action performed.");
        }
 }

public class ListenerTest extends Applet {
          public void init()  {
               Button btn=new Button("OK");
               MyActionListener myAl=new MyActionListener();
               btn.addActionListener(myAl);
               add(btn);
          }
}
```

Putem rezuma tehnica de lucru în patru paşi:

1. Se creează o clasă „listener" care implementează interfaţa **ActionListener** (pt. exemplul nostru).
2. Se construieşte componenta.
3. Se construieşte o instanţă a clasei „listener".

4. Se apelează metoda **addActionListener()** pentru componentă, transmiţându-i ca parametru obiectul „listener”.

Există 11 tipuri de “listeners”, fiecare reprezentat de o interfaţă. În continuare vom prezenta numai o parte dintre acestea, împreună cu metodele din interfaţă şi cu metodele **addXXXListener()** corespunzătoare.

(Pentru celelalte interfeţe, anume AdjustmentListener, ComponentListener, ContainerListener, FocusListener, TextListener, WindowListener se vor căuta informaţii în documentaţia jsdk).

Interfaţă	**Metodele din interfaţă**	**Metoda add**
ActionListener	actionPerformed(ActionEvent e)	addActionListener( )
ItemListener	itemStateChanged(ItemEvent e)	addItemListener( )
KeyListener	keyPressed(keyEvent e) keyReleased(keyEvent e) keyTyped(keyEvent e)	addKeyListener( )
MouseListener	mouseClicked(MouseEvent e) mouseEntered(MouseEvent e) mouseExited(MouseEvent e) mousePressed(MouseEvent e) mouseReleased(MouseEvent e)	addMouseListener( )
MouseMotionListener	mouseDragged(MouseEvent e) mouseMoved(MouseEvent e)	addMouseMotionListener( )

Reversul metodei **addXXXListener()** este metoda **removeXXXListener()** care înlătură pentru o componentă obiectul ataşat ei care “asculta” evenimentele generate de aceasta. Pentru exemplul anterior avem:

```
btn.removeActionListener(myAl);
```

Aşa cum deja am menţionat, o a doua tehnică de lucru permite explicit componentei care generează evenimentele să-şi manipuleze propriile evenimente. În continuare vom prezenta câteva exemple sugestive.

```
// App4.java
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class App4 extends Applet implements ActionListener
{
   Button but1,but2;
   int x=0,y=0;
   public void init()
   {
     but1=new Button("Deseneaza");
     add(but1);
     but2=new Button("Sterge");
     add(but2);
     but1.addActionListener(this);
     but2.addActionListener(this);
```

```
   }

   public void actionPerformed(ActionEvent e)
   {
        Button b=(Button)e.getSource();
        if (b==but1)
          {
             x=150;y=150;
             repaint();
          }
        else if(b==but2)
          {
             x=y=0;
             repaint();
          }
   }
   public void paint(Graphics g)
   {
       setBackground(Color.blue);
       setForeground(Color.magenta);
       if(x>0)
           g.fillOval(x,y,100,100);
   }
}



// App4.html
<html>
<body>
<applet code=App4.class width=500 height=400>
</applet>
</body>
</html>

//App5.java
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;

public class App5 extends Applet
                  implements ActionListener, MouseListener
{
   Button but;
   Vector points;
   public void init()
   {
     but=new Button("Clear all");
     add(but);
     but.addActionListener(this);
     addMouseListener(this);
     points=new Vector();
   }
   public void actionPerformed(ActionEvent e)
   {
        Button b=(Button)e.getSource();
        if (b==but)
          {
```

```
            points.setSize(0);
            repaint();
         }

   }
   public void mouseClicked(MouseEvent e)
   {
      e.consume();
      points.addElement(new Point(e.getPoint()));
      repaint();
   }
   public void mouseEntered(MouseEvent e)
   {
   }
   public void mouseExited(MouseEvent e)
   {
   }
   public void mousePressed(MouseEvent e)
   {
   }
   public void mouseReleased(MouseEvent e)
   {
   }

   public void paint(Graphics g)
   {
       setBackground(Color.orange);
       setForeground(Color.black);
       for(int i=0;i<points.size();i++)
       {

           Point p=(Point)points.elementAt(i);
           g.fillRect(p.x,p.y,70,30);
       }
   }
}

//App5.html
<html>
<body>
<applet code=App5.class width=500 height=500>
</applet>
</body>
</html>

//App6.java
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;

public class App6 extends Applet
                  implements ActionListener, MouseListener
{
   Button but;
   Vector lines;
   int x1,y1,x2,y2;
   public void init()
   {
```

```
      but=new Button("Clear all");
      add(but);
      but.addActionListener(this);
      addMouseListener(this);
      lines=new Vector();
   }
   public void actionPerformed(ActionEvent e)
   {
        Button b=(Button)e.getSource();
        if (b==but)
          {
             lines.setSize(0);
             repaint();
          }
   }
   public void mouseClicked(MouseEvent e)
   {
   }

   public void mouseEntered(MouseEvent e)
   {
   }
   public void mouseExited(MouseEvent e)
   {
   }
   public void mousePressed(MouseEvent e)
   {
       e.consume();
       x1=e.getX();
       y1=e.getY();
   }
   public void mouseReleased(MouseEvent e)
   {
      e.consume();
      x2=e.getX();
      y2=e.getY();
      lines.addElement(new Rectangle(x1,y1,x2,y2));
      repaint();
   }

   public void paint(Graphics g)
   {
       for(int i=0;i<lines.size();i++)
       {

           Rectangle r=(Rectangle)lines.elementAt(i);
           g.drawLine(r.x,r.y,r.width,r.height);
       }
   }
}

//App6.html
<html>
<body>
<applet code=App6.class width=500 height=500>
</applet>
</body>
</html>
```

```
//App7.java
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class App7 extends Applet
                  implements ActionListener, ItemListener
{
   Button but;
   Choice ch;
   String color;

   public void init()
   {
     but=new Button("Draw");
     add(but);
     but.addActionListener(this);
     ch=new Choice();
     ch.addItem("red");
     ch.addItem("yellow");
     ch.addItem("pink");
     add(ch);
     ch.addItemListener(this);
     color=new String();
     setFont(new Font("Arial",Font.PLAIN,14));
   }

   public void actionPerformed(ActionEvent e)
   {
        Button b=(Button)e.getSource();
        if (b==but)
          {
             repaint();
          }
   }

   public void itemStateChanged(ItemEvent e)
   {
      color=ch.getSelectedItem();
   }

   public void paint(Graphics g)
   {
       setBackground(Color.black);
       if(color.compareTo("red")==0)
             g.setColor(Color.red);
       else if(color.compareTo("yellow")==0)
             g.setColor(Color.yellow);
       else if(color.compareTo("pink")==0)
             g.setColor(Color.pink);

       g.drawString(color,70,150);
   }
}

//App7.html
<html>
<body>
<applet code=App7.class width=200 height=300>
```

```
</applet>
</body>
</html>

//App8.java
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;



public class App8 extends Applet implements ActionListener
{
   Button but;
   TextField tx;
   myFrame f;

   public void init()
   {
     setFont(new Font("Arial",Font.PLAIN,14));
     tx=new TextField("",10);
     add(tx);
     but=new Button("Open window");
     add(but);
     but.addActionListener(this);
   }

   public void actionPerformed(ActionEvent e)
   {
        Button b=(Button)e.getSource();
        if (b==but)
          {
             f=new myFrame("Little window",tx.getText());
             f.setSize(300,300);
             f.setVisible(true);
          }
   }
}

class myFrame extends Frame implements ActionListener
{
   String text;
   Button buton;
   public myFrame(String s,String tx)
   {
      super(s);
      text=tx;
      setLayout(new FlowLayout());
      buton=new Button("Close");
      add(buton);
      buton.addActionListener(this);
   }
   public void actionPerformed(ActionEvent e)
   {
        Button b=(Button)e.getSource();
        if (b==buton)
          {
            dispose();
```

```
            }
    }
    public void paint(Graphics g)
    {
        g.drawString(text,50,100);
    }
}
```

```
//App8.html
<html>
<body>
<applet code=App8.class width=300 height=300>
</applet>
</body>
</html>
```

III. MODUL DE LUCRU

- Clasic:

1. Se editează codul sursă al programului Java folosind un editor de text disponibil (de ex., se poate utiliza Notepad).
2. Se salvează fişierul cu extensia **.java**.
3. Compilarea aplicaţiei Java se va face din linia de comandă:
   javac nume_fişier_sursă.java
   În cazul în care programul conţine erori acestea vor fi semnalate şi afişate.
4. Pentru rularea aplicaţiei Java, se lansează interpretorul Java:
   java nume_clasă_care_conţine_main

- Se foloseşte utilitarul disponibil în laborator J2SDK Net Beans IDE.

IV. TEMĂ

1. Se vor parcurge toate exemplele prezentate în platforma de laborator testându-se practic şi explicând rezultatele obţinute.
2. Scrieţi un applet Java care să conţină următoarele: o zonă de editare TextArea cu 5 linii si 10 coloane, un buton cu eticheta OK şi un grup de componente CheckBox. Setaţi textul zonei de editare la un anume şir de caractere. Utilizaţi diverse culori şi font-uri.
3. Scrieţi un applet Java care să conţină un grup de 2 componente CheckBox etichetate “red” şi “blue”, şi un buton etichetat “Clear all”. Utilizatorul va bifa una din opţiunile “red” sau “blue”. La efectuarea unui click cu mouse-ul pe suprafaţa aapplet-ului se va desena un cerc colorat roşu sau albastru în funcţie de opţiunea aleasă. La apăsarea butonului “Clear all” se vor şterge toate cercurile desenate până în acel moment.