

Interfețe

Revedeți cursul pentru detalii și exemple privind interfețele în Java.



Probleme

1. Ilustrați utilitatea interfețelor în Java și modul în care se rezolvă problema moștenirii multiple în Java cu ajutorul interfețelor
2. Ilustrați utilitatea interfețelor `java.lang.Comparable` și `java.util.Comparator`
3. Modificați clasa de listă astfel încât să permită parcurgerea folosind iterator. Parcurgeți o astfel de listă cu `for` pentru colecții (`for each`)

Interfețe grafice*



Exemplul 1

Ne propunem să creăm o interfață grafică pentru problema evaluării unei expresii cu operatori binari dată în formă prefixată (cu ajutorul arborelui binar asociat) care să arate astfel:

Expresie	<input type="text"/>
	<input type="button" value="Evalueaza"/>
Rezultat	<input type="text"/>

Pentru a crea o fereastră trebuie să scriem o clasă care extinde clasa `java.awt.Frame`. La această fereastră vom adăuga obiectele dorite: două etichete (de tip `java.awt.Label`), două câmpuri de text (de tip `java.awt.TextField`), și un buton (de tip `java.awt.Button`).

Pentru plasarea componentelor în fereastră, sau, mai general, în orice container, trebuie folosit un *gestionar de poziționare* (*layout manager*). În acest exemplu vom folosi `java.awt.FlowLayout`, **care plasează componentele în container de la stânga la dreapta și de sus în jos**. Gestionarul de poziționare se specifică prin invocarea metodei `setLayout` din clasa `Container`.

Pentru ca eticheta `Rezultat` să nu fie separată de câmpul de text corespunzător ei (să nu apară lângă butonul `Evalueaza`), cele două obiecte trebuie grupate înainte de a fi adăugate la fereastră. Pentru a grupa obiecte se folosește un obiect de tip `java.awt.Panel`.

Mai trebuie să specificăm acțiunile care dorim să se execute pentru fiecare control și anume: când scriem în câmpul de text expresia să anulăm rezultatul precedent, iar când apăsăm pe butonul `Evalueaza` să apară în câmpul de rezultat valoarea expresiei introduse; în plus la apăsarea butonului de închidere a ferestrei aplicația trebuie să se termine. Pentru a specifica o acțiune la apariția unui eveniment referitor la un control grafic, trebuie mai întâi să asociem un **gestionar (listener, handler)** controlului, folosind metoda `addTipListener` unde `tip` este tipul acțiunii pe care dorim să o controlăm.

Un handler (gestionar) este un obiect al unei clase ce implementează interfața `TipListener` sau extinde clasa `TipAdapter` (ambele din pachetul `java.awt.event`) corespunzătoare tipului de acțiune pe care dorim să o controlăm.

```
import java.awt.*;
import java.awt.event.*;

//clasa care extinde Frame
class ExpresieGrafica extends Frame{
    static final long serialVersionUID=1;
    TextField txtExpresie, txtRezultat;
    Label lblExpresie, lblRezultat;
    Button btnTransform;

    ExpresieGrafica(){
        super("Expresie");

        //cream obiectele
        lblExpresie=new Label("Expresie");
        lblRezultat=new Label("Rezultat");
        txtExpresie=new TextField(50);
        txtRezultat=new TextField(50);
        btnTransform=new Button("Evalueaza");

        //plasam componentele in fereastra, folosind gestionarul FlowLayout
        setLayout(new FlowLayout());

        //grupam eticheta cu campul de text corespunzator
        Panel p1=new Panel();
        p1.add(lblExpresie);
        p1.add(txtExpresie);

        Panel p2=new Panel();
        p2.add(lblRezultat);
        p2.add(txtRezultat);
```

```

        //adaugam obiectele in fereastra, in ordinea in care dorim sa apara
        add(p1);
        add(btnTransform);
        add(p2);

        /*asociem controalelor gestionare(handlere)
        pentru actiunile pe care vrem sa le controlam*/

        //un handler implementeaza interfata tipListener sau extinde clasa
        tipAdapter
        /*folosim aceeasi clasa ca handler pentru toate evenimentele
        asociate componentelor din fereastra pe care vrem sa le controlam*/

        ExprEventsHandler eeh=new ExprEventsHandler(this);

        txtExpresie.addTextListener(eeh);
        txtExpresie.addActionListener(eeh);
        btnTransform.addActionListener(eeh);

        //asociem si ferestrei un handler
        addWindowListener(new ExprWindowHandler());

        //campul de text rezultat va fi needitabil
        txtRezultat.setEditable(false);
    }

    public static void main(String ar[]){

        ExpresieGrafica eg=new ExpresieGrafica();

        eg.setSize(500,150);//al doilea parametru e inaltimea
        eg.setResizable(false);//fereastra nu se poate redimensiona
        eg.setVisible(true);
        eg.setLocation(200,200);//in ce punct apare fereastra
    }

}

class ExprWindowHandler extends WindowAdapter{

    //WindowListener are mai multe metode ( (de)activate, open ...)
    public void windowClosing(WindowEvent e){//!nu windowClosed
        System.exit(0);
    }

}

```

```
class ExprEventsHandler implements ActionListener,TextListener{
    ExpresieGrafica f;

    ExprEventsHandler(ExpresieGrafica f){
        this.f=f;
    }

    public void actionPerformed(ActionEvent e){//ActionListener
        Expresie obExp=new Expresie(f.txtExpresie.getText());
        f.txtRezultat.setText(obExp.eval()+" Tema ");
    }

    public void textValueChanged(TextEvent e){//TextListener
        f.txtRezultat.setText("");
    }
}

class Expresie{
    String expresie;
    Expresie(String exp){
        expresie=exp;
    }

    public double eval(){
        return 0;
    }
}
```

Exemplul 2.

Să presupunem că dorim ca interfața grafică de la exemplul 1 să arate astfel

Expresie	<input type="text"/>	<input type="button" value="Evalueaza"/>
Rezultat	<input type="text"/>	

Pentru aceasta nu vom folosi un gestionar de poziție existent în Java (vom seta gestionarul utilizat ca fiind `null` - `setLayout(null)`), ci vom seta la fiecare componentă în parte punctul de unde dorim să fie desenată (colțul stânga-sus al ei), lățimea și înălțimea, folosind metoda `setBounds`

Constructorul ferestrei va deveni în acest caz

```

ExpresieGrafica() {
    ExprEventsHandler eeh=new ExprEventsHandler(this);
    setTitle("Expresie");
    setLayout(null);

    lblExpresie=new Label("Expresie");
    txtExpresie=new TextField(50);
    lblRezultat=new Label("Rezultat");
    txtRezultat=new TextField(50);
    btnTransform=new Button("Evalueaza");

    lblExpresie.setBounds(10,50,50,20);
    txtExpresie.setBounds(70,50,300,20);
    lblRezultat.setBounds(10,80,50,20);
    txtRezultat.setBounds(70,80,300,20);
    btnTransform.setBounds(380,60,100,20);

    add(lblExpresie);
    add(txtExpresie);
    add(lblRezultat);
    add(txtRezultat);
    add(btnTransform);

    txtExpresie.addTextListener(eeh);
    txtExpresie.addActionListener(eeh);
    btnTransform.addActionListener(eeh);

    addWindowListener(new ExprWindowHandler());

    txtRezultat.setEditable(false);
}

```

- **Controale grafice (componente elementare)**

Exemplul 3.

Modificăm exemplul 1, astfel încât să avem posibilitatea de a alege dacă vrem să evaluăm expresia sau să afișăm expresia în formă infixată. Pentru aceasta vom adăuga două obiecte de tip `Checkbox`, grupate pentru a nu putea fi selectate simultan.

Astfel,

1. se vor adăuga clasei câmpurile

```

Checkbox ckEval, ckIn;
CheckboxGroup ckgForma;

```

2. se vor crea în constructor obiectele corespunzătoare

```

ckgForma=new CheckboxGroup();
ckEval =new Checkbox("Evaluare",ckgForma,false);
ckIn=new Checkbox("Infix",ckgForma,true);

```

3. se grupează cele două obiecte de tip `Checkbox` într-un `Panel` `p3`

```
Panel p3=new Panel();
p3.add(ckEval);
p3.add(ckIn);
```

4. se adaugă panelul `p3` la fereastră înaintea butonului `btnTransform`


```
add(p3);
```

5. se adaugă celor două obiecte un gestionar `eeh` folosind metoda `addItemListener`

```
ckEval.addItemListener(eeh);
ckIn.addItemListener(eeh);
```

modificând clasa `ExprEventsHandler` astfel încât aceasta să implementeze și interfața `ItemListener` (care are metoda `itemStateChanged`), pentru a șterge textul din câmpul de rezultat când se modifică opțiunea utilizatorului privind acțiunea pe care o dorește și modificând implementarea metodei `actionPerformed`, astfel încât să țină cont de opțiunea bifată.

```
class ExprEventsHandler implements ActionListener, TextListener,
ItemListener{
    ExpresieGrafica f;
    ExprEventsHandler(ExpresieGrafica f){
        this.f=f;
    }
    public void actionPerformed(ActionEvent e){//ActionListener
        Expresie obExp=new Expresie(f.txtExpresie.getText());
        if(f.ckEval.getState())
            f.txtRezultat.setText(obExp.eval()+" Tema ");
        else
            f.txtRezultat.setText("Expresia infix tema ");
    }
    public void textValueChanged(TextEvent e){//TextListener
        f.txtRezultat.setText("");
    }
    public void itemStateChanged(ItemEvent e){
        f.txtRezultat.setText("");
    }
}
```

 **Exemplul 4.** Vom reprezenta grafic două funcții pozitive ($x/3$ și $x*x/2-4*x+9$) pe intervalul $[0, 8]$.

Pentru aceasta să ne reamintim câteva lucruri legate de suprafețe grafice în Java (revedeți cursul). **Modul standard de creare a unei suprafețe grafice constă în**

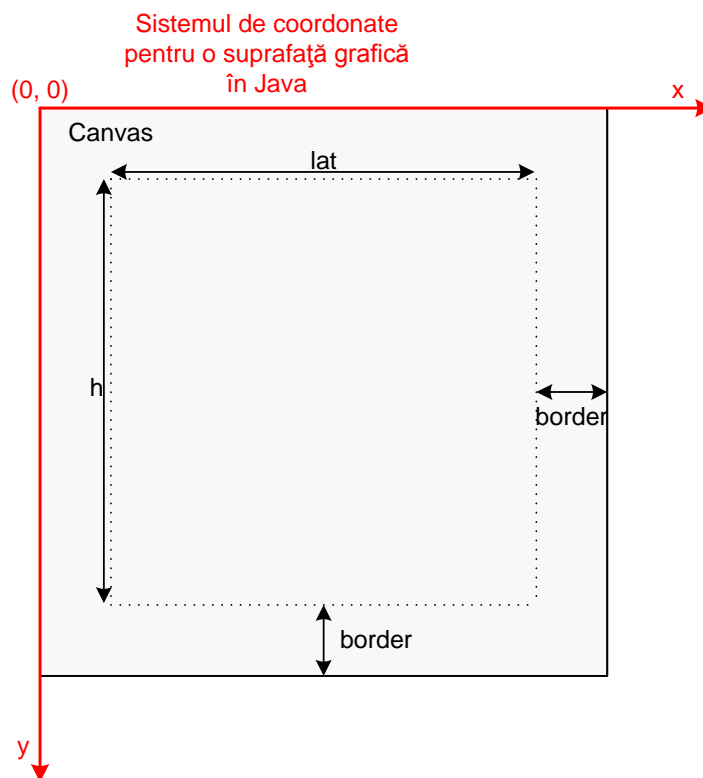
extinderea clasei `Canvas` (din pachetul `java.awt`), **redefinind metoda `paint` ce afișează componenta**. Modificările ulterioare devin vizibile prin invocarea metodei `repaint`.

O suprafață grafică reprezintă un dreptunghi de pixeli, **cu colțul din stânga sus având coordonatele (0,0)** (axa O_x fiind orizontală, orientată de la stânga la dreapta, iar axa O_y fiind verticală, orientată de sus în jos). Este folosit un *context de desenare*; vom folosi contextul de desenare standard disponibil, pus la dispoziție la invocarea metodei `paint` (de tip `Graphics`):

```
public paint(Graphics g)
```

Astfel, pentru a desena funcțiile vom crea o suprafață grafică – clasa `GraficFunctie` care extinde clasa `Canvas`. Vom redefini metoda `paint` pentru a desena axele de coordonate O_x și O_y (partea pozitivă, pe care vom evidenția punctele 1, 2, ..., 9) și a reprezenta grafic funcția f curentă (dată în clasa care implementează interfața `IFunctie`).

Pe suprafața de desenare vom considera o bordură de lungime `border=50`, iar suprafața pe care vom desena graficul are lățimea `lat=400` și înălțimea `h=400` (deci suprafața va avea lățimea `lat+2*border` și înălțimea `h+2*border`)



Am considerat scala pentru Ox **scalex=40** și pentru Oy **scaley=40**.

Pasul cu care sunt considerate puncte din intervalul [a, b] pentru realizarea graficului funcției este dat de variabila `pas`.

Un obiect de tip `Canvas` se adaugă în fereastră ca orice altă componentă (cu metoda `add`).

```
import java.awt.*;
import java.awt.event.*;

interface IFuncție{
    double f(double x);
}

class F2 implements IFuncție{//implementare pentru functia f
    public String toString(){
        return "x*x/2-4*x+9";
    }

    public double f(double x){
        return x*x/2-4*x+9;
    }
}

class FLinie implements IFuncție{//alta implementare pentru functia f
    public double f(double x){
        return x/3;
    }

    public String toString(){
        return "x/3";
    }
}

class GraficFuncție extends Canvas{//suprafata de desenare
    static final long serialVersionUID=1; //optional, pentru warning

    IFuncție fct;

    double a=0,b=8;
    int lat=400,h=400;
    int border=50;

    GraficFuncție(IFuncție fct){
        this.fct=fct;
        setSize(lat+2*border,h+2*border);
    }
}
```


/*metode care fac corespondenta dintre coordonatele x respectiv y din sistemul ortogonal 'clasic' pentru reprezentarea grafica a functiei si coordonatele reale pentru Canvas, calculate raportat la coltul din stanga sus (si la axele evidentiata cu rosu in figura anterioara)*/

```

int yGrafic(double y){
    return h+border-(int)y;
}
int xGrafic(double x){
    return (int)x+border;
}

public void paint(Graphics g){
    g.drawLine(border,h+border,lat+border,h+border); //ox
    g.drawString("O",border-10,h+border+10); //eticheta O
    g.drawString("x",lat+border-10,h+border+10); //eticheta x
    g.drawLine(border,h+border,border,border); //oy
    g.drawString("y",border-10,border+10);
    double scalex,pas=0.01,scaley;
    scaley=scalex=40;

    //numerotare puncte ox
    for(int x=1;x<lat/scalex;x++){
        int x1=xGrafic(x*scalex);
        g.drawLine(x1,h+border-2,x1,h+border+2);
        g.drawString(x+"",x1-3,h+border+15);
    }

    //numerotare puncte oy
    for(int y=1;y<h/scaley;y++){
        int y1=yGrafic(y*scaley);
        g.drawLine(border-2,y1,border+2,y1);
        g.drawString(y+"",border-10,y1+3);
    }

    for(double x=a;x<b;x+=pas){
        int x1=xGrafic(x*scalex);
        int x2=xGrafic((x+pas)*scalex);

        int y1=yGrafic(fct.f(x)*scaley);
        int y2=yGrafic(fct.f(x+pas)*scaley);

        g.drawLine(x1,y1,x2,y2);
    }
}

```

```

class FereastragraficFunctie extends Frame{
    static final long serialVersionUID=1L;
    GraficFunctie gf;
    Checkbox ckf1,ckf2;
    CheckboxGroup ckgForma;
    FLinie f1;
    F2 f2;

    void setFunctie(IFunctie fct){
        gf.fct=fct;
    }

    FereastragraficFunctie(){
        EventHandler eeh=new EventHandler(this);

        f1=new FLinie();
        f2=new F2();

        setLayout(new FlowLayout());

        gf=new GraficFunctie(f1);

        ckgForma=new CheckboxGroup();
        ckf1=new Checkbox(f1.toString(),ckgForma,true);
        ckf2=new Checkbox(f2.toString(),ckgForma,false);

        ckf1.addItemListener(eeh); ckf2.addItemListener(eeh);

        add(gf); add(ckf1);add(ckf2);

        addWindowListener(new WindowHandler());
    }

    public static void main(String ar[]){
        FereastragraficFunctie eg=new FereastragraficFunctie();
        eg.setSize(500,600);
        //eg.setResizable(false);
        eg.setVisible(true); eg.setLocation(100,100);
    }
}

class WindowHandler extends WindowAdapter{
    public void windowClosing(WindowEvent e){
        System.exit(0);
    }
}

```

```
class EventHandler implements ItemListener{
    FereastraGraficFunctie f;

    EventHandler(FereastraGraficFunctie f){
        this.f=f;
    }
    public void itemStateChanged(ItemEvent e){
        if(f.ckf1.getState())
            f.setFunctie(f.f1);
        else
            f.setFunctie(f.f2);
        f.gf.repaint();
    }
    //redeseneaza suprafata grafica pentru noua functie, apeleaza paint
}
```

Exercitii

1. Studiați și exemplele din curs.
2. Realizați o interfață grafică pentru oricare dintre problemele scrise la laborator.
3. Realizați o interfață grafică având cât mai multe tipuri de controale.