

Desenarea

Programare Orientată pe Obiecte



Desenarea



- Conceptul de desenare
- Metoda paint
- Suprafețe de desenare
- Folosirea fonturilor
- Folosirea culorilor
- Folosirea imaginilor
- Mecanismul de "double-buffering"
- Tipărirea

Conceptul de desenare

- Un program Java care are interfață grafică cu utilizatorul trebuie să deseneze pe ecran toate componentele sale care au o reprezentare vizuală. Desenarea componentelor se face automat:
 1. la afișarea pentru prima dată;
 2. la operații de minimizare, maximizare, redimensionare a suprafeței de afișare;
 3. ca răspuns al unei solicitări explicite a programului.

Metode:

- **void paint(Graphics g)** - Desenează o componentă.
- **void update(Graphics g)** - Actualizează starea grafică a unei componente.
 - a. șterge componenta prin supradesenarea ei cu culoarea fundalului;
 - b. stabilește culoarea (foreground) componentei;
 - c. apelează metoda paint pentru a redesena componenta.
- **void repaint()** - Execută explicit un apel al metodei *update* pentru a actualiza reprezentarea grafică a unei componente.

Metoda paint

- toate desenele care trebuie să apară pe o suprafață de afișare se realizează în metoda paint
- Responsabilă cu desenarea unei componente

```
import java.awt.*;
import javax.swing.*;
class Fereastra extends JFrame {
    public Fereastra ( String titlu ) {
        super ( titlu );
        setSize ( 200 , 100 ) ;
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
    public void paint ( Graphics g ) {
        super.paint(g);
        g. setFont (new Font ( " Arial ", Font .BOLD , 11));
        g. setColor ( Color .red );
        g. drawString ( " Aplicatie DEMO " , 5, 35);
    }
}
public class TestPaint {
    public static void main ( String args []) {
        Fereastra f = new Fereastra ( " Test paint " );
        f. show ();
    }
}
```

Suprafețe de desenare

- Clasa Canvas:
 - Este o clasă generică din care se derivează subclase pentru crearea suprafețelor de desenare (planșe).
 - Planșele nu pot conține alte componente grafice, ele fiind utilizate doar ca suprafețe de desenat sau ca fundal pentru animație.
 - Desenarea pe o planșă se face prin supradefinirea metodei *paint* a acesteia.

```
class Plansa extends Canvas implements ...Listener {
    //Eventual, unul sau mai multi constructori
    public Plansa() { ... }
    // Metode de desenare a componentei
    public void paint(Graphics g) { ... }
    // Metodele folosite de gestionarii de pozitionare
    public Dimension getPreferredSize() {
        // Dimensiunea implicita a plansei
        return ...; }
    public Dimension getMinimumSize() { return ... }
    public Dimension getMaximumSize() { return ... }
    // Implementarea metodelor interfetelor de tip //Listener
    ...
}
```

Folosirea clasei Canvas

```
import java . awt . * ;
import java . awt . event . * ;
import javax . swing . * ;
class Plansa extends Canvas {
    Dimension dim = new Dimension ( 100 , 100 ) ;
    private Color color [] = { Color . red , Color . blue } ;
    private int index = 0 ;
    public Plansa () {
        this . addMouseListener ( new MouseAdapter () {
            public void mouseClicked ( MouseEvent e ) {
                index = 1 - index ;
                repaint () ;
            }
        } );
    }
    public void paint ( Graphics g ) {
        g . setColor ( color [ index ] ) ;
        g . drawRect ( 0 , 0 , dim . width , dim . height ) ;
        g . setColor ( color [ 1 - index ] ) ;
        g . fillOval ( 0 , 0 , dim . width , dim . height ) ;
    }
}
```

Folosirea clasei Canvas

```
public Dimension getPreferredSize () {  
    return dim ;  
}  
}
```

```
class Fereastra extends JFrame {  
    public Fereastra ( String titlu ) {  
        super ( titlu );  
        setSize ( 200 , 200 ) ;  
        setDefaultCloseOperation(EXIT_ON_CLOSE);  
        add ( new Plansa () , BorderLayout . CENTER ) ;  
    }  
}
```

```
public class TestCanvas {  
    public static void main ( String args []) {  
        new Fereastra ( " Test Canvas " ). show ();  
    }  
}
```

Contextul grafic de desenare

- Un context grafic este un obiect de tip *Graphics* folosit pentru desenare:
 - pe o porțiune de ecran
 - la imprimantă
 - într-o zonă virtuală de memorie.

Metode:

- primitive grafice: desenarea de figuri geometrice, texte și imagini
- stabilirea proprietăților contextului grafic:
 - culoare, font
 - originea coordonatelor
 - suprafața vizibilă
 - modul de desenare

Proprietățile contextului grafic

Proprietate	Metode
Culoarea de desenare	<code>Color getColor()</code> <code>void setColor(Color c)</code>
Fontul de scriere a textelor	<code>Font getFont()</code> <code>void setFont(Font f)</code>
Originea coordonatelor	<code>translate(int x, int y)</code>
Zona de decupare	<code>Shape getClip()</code> <code>void setClip(Shape s)</code>
Modul de desenare	<code>void setXorMode(Color c)</code> <code>void setPaintMode(Color c)</code>

Primitive grafice

- Desenarea textelor - drawString
drawString("Hello", 10, 20);
- Desenarea figurilor geometrice

Linie: drawLine, drawPolyline

Dreptunghi simplu: drawRect, fillRect,
clearRect

Dreptunghi cu chenar: draw3DRect

Dreptunghi cu chenar "ridicat" sau "adâncit":
fill3DRect

Dreptunghi cu colțuri: drawRoundRect

Dreptunghi cu colțuri rotunjite: fillRoundRect

Poligon: drawPolygon, fillPolygon

Oval (Elipsă) drawOval, fillOval

Arc circular sau eliptic: drawArc, fillArc

Folosirea fonturilor



Parametrii unui font

- Numele fontului: Helvetica Bold, Arial, Bold Italic, etc.
- Familia din care face parte fontul: Helvetica, Arial, etc.
- Dimensiunea fontului (înălțimea sa)
- Stilul fontului: îngroșat (bold), înclinat (italic);
- Metrica fontului.

Clase: Font, FontMetrics

Stabilirea unui font: setFont

Clasa Font

- Incapsulează toate informațiile fontului, mai puțin metrica sa.
- Font(String name, int style, int size)
 - new Font("Dialog", Font.PLAIN, 12);
 - new Font("Arial", Font.ITALIC, 14);
 - new Font("Courier", Font.BOLD, 10);
- Pentru componente etichetate
Label label = new Label("Un text");
label.setFont(new Font("Dialog", Font.PLAIN, 12));
- In metoda paint(Graphics g)
g.setFont(new Font("Courier", Font.BOLD, 10));
g.drawString("Alt text", 10, 20);
- Lista fonturilor instalate:
Font[] fonturi = GraphicsEnvironment.
 getLocalGraphicsEnvironment().getAllFonts();

Lucrul cu fonturi

```
import java . awt . * ;
import javax.swing . * ;
class Fonturi extends Canvas {
    private Font [] fonturi ;
    Dimension canvasSize = new Dimension ( 400 , 400 ) ;
    public Fonturi () {
        fonturi = GraphicsEnvironment .
            getLocalGraphicsEnvironment () . getAllFonts () ;
        canvasSize . height = ( 1 + fonturi . length ) * 20 ;
    }
    public void paint ( Graphics g ) {
        String nume ;
        for ( int i = 0 ; i < fonturi . length ; i ++ ) {
            nume = fonturi [ i ] . getFontName () ;
            g . setFont ( new Font ( nume , Font . PLAIN , 14 ) ) ;
            g . drawString ( i + " . " + nume , 20 , ( i + 1 ) * 20 ) ;
        }
    }
}
```

Lucrul cu fonturi

```
    public Dimension getPreferredSize () {
        return canvasSize ;
    }
}

class Fereastra extends JFrame {
    public Fereastra ( String titlu ) {
        super ( titlu );
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        ScrollPane sp = new ScrollPane ();
        sp.add(new Fonturi());
        sp. setSize (200 , 200) ;
        add (sp , BorderLayout . CENTER );
        pack ();
    }
}

class TestAllFonts {
    public static void main ( String args []) {
        new Fereastra ("All fonts "). show ();
    }
}
```

Clasa FontMetrics

Informații despre metrica unui font.

```
public void paint(Graphics g) {
```

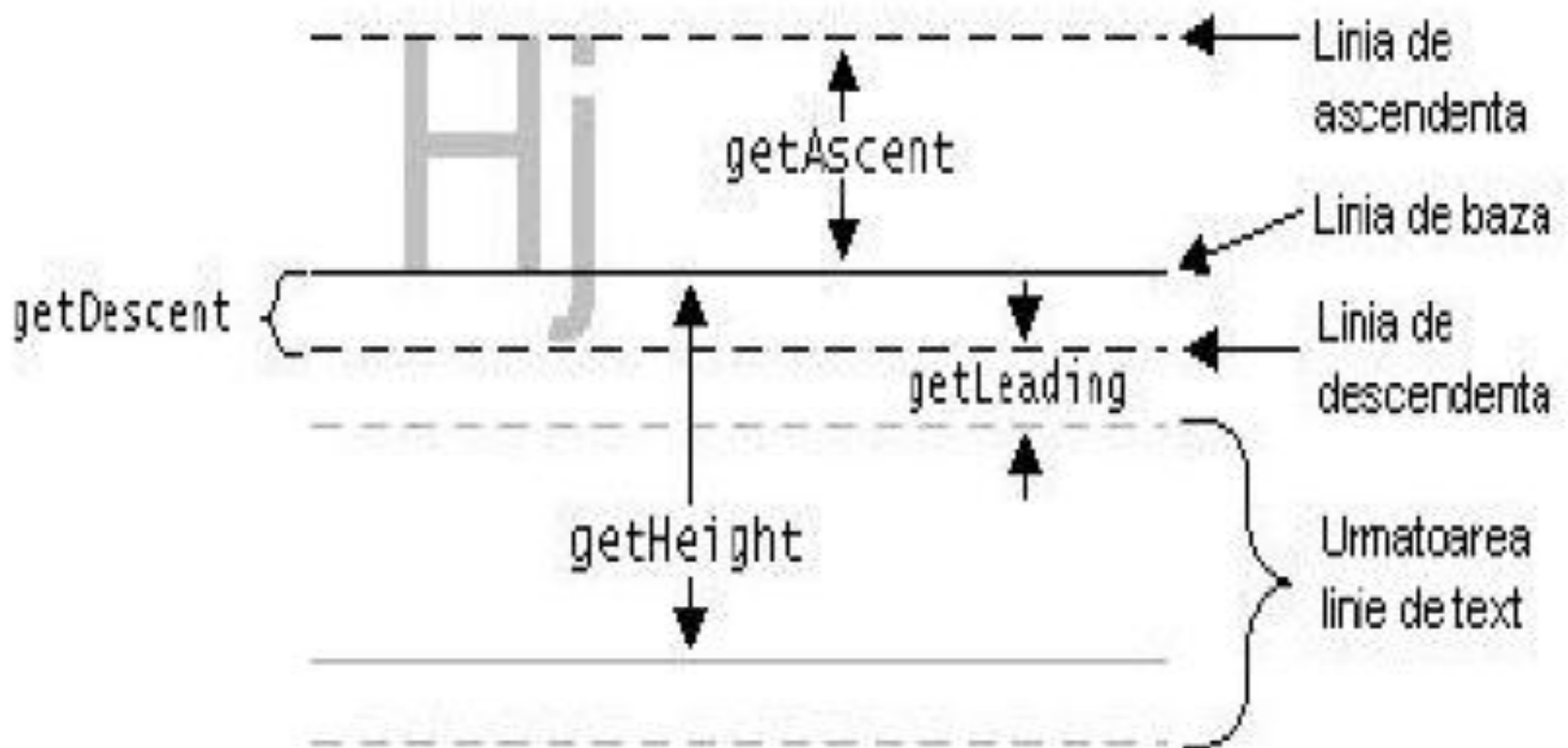
```
    ...
```

```
        FontMetrics fm = g.getFontMetrics();
```

```
}
```

Metode

- getHeight
- stringWidth
- charWidth



Clasa FontMetrics

- Metrica unui font constă în următoarele atribute pe care le au caracterele sale:
- Linia de bază: este linia după care sunt aliniate caracterele unui font;
- Linia de ascendență: linia superioara pe care nu o depaseste nici un caracter din font
- Linia de descendență: linia inferioară sub care nu coboară nici un caracter din font;
- Ascendentul: distanța între linia de bază și linia de ascendență;
- Descendentul: distanța între linia de bază și linia de descendență;
- Lățimea: lățimea unui anumit caracter din font;
- Distanța între linii ("leading"): distanța optimă între două linii de text scrise cu același font.
- Inălțimea: distanța dintre liniile de bază (leading+ascent+descent);

exemplu

Folosirea culorilor

- Red Green Blue Alpha (0–255,0.0 – 1.0)
- Clase: Color, SystemColor
- Constante

Color rosu = Color.red;

Color galben = Color.yellow;

Color fundal = SystemColor.desktop;

- Constructori

// Exemple de folosire a constructorilor:

Color alb = new Color(255, 255, 255);

Color negru = new Color(0, 0, 0);

Color rosuOpac = new Color(255, 0, 0);

Color rosuTransparent = new Color(255, 0, 0, 128);

- Metode

brighter, darker, getRed, etc

Folosirea culorilor

```
int r = rValue.getValue();
int g = gValue.getValue();
int b = bValue.getValue();
int a = aValue.getValue();
color = new Color(r, g, b, a);
repaint();
...
public void paint(Graphics g) {
    g.setColor(Color.black);
    g.setFont(new Font("Arial", Font.BOLD, 12));
    String text = "";
    text += " R=" + color.getRed();
    text += " G=" + color.getGreen();
    text += " B=" + color.getBlue();
    text += " A=" + color.getAlpha();
    g.drawString(text, 0, 30);
    g.setColor(color);
    g.fillRect(0, 0, canvasSize.width, canvasSize.height);
}
```



Folosirea imaginilor

Aceasta este o imagine:



Formate permise: gif sau jpeg

Clasa: Image

Afișarea unei imagini:

1. Crearea unui obiect de tip Image;
2. Afișarea propriu-zisă într-un context grafic;

Folosirea imaginilor

- Crearea unui obiect Image

```
Toolkit toolkit = Toolkit.getDefaultToolkit();
```

```
Image image1 = toolkit.getImage("poza.gif");
```

```
Image image2 = toolkit.getImage(  
    new URL("http://www.infoiasi.ro/~acf/poza.gif"));
```

- Afișarea unei imagini

```
Image img =
```

```
    Toolkit.getDefaultToolkit().getImage("taz.gif");
```

```
g.drawImage(img, 0, 0, this);
```

```
g.drawImage(img, 0, 200, 100, 100, this);
```

```
g.drawImage(img, 200, 0, 200, 400, Color.yellow, this);
```

```
//Formatul cel mai general:
```

```
boolean drawImage(Image img, int x, int y, int width,  
    int height, Color bgcolor, ImageObserver observer)
```

Monitorizarea încărcării imaginilor

- Interfața ImageObserver

```
    boolean imageUpdate (Image img, int flags, int
x, int y, int w, int h )
```
- flags: ABORT, ALLBITS, ERROR, HEIGHT, WIDTH, PROPERTIES

```
    // Imaginea este completa
    (flags & ALLBITS) != 0
    // Eroare sau transfer intrerupt
    (flags & ERROR | ABORT ) != 0
```
- ```
public boolean imageUpdate(Image img, int flags,
 int x, int y, int w, int h) {
 // Desenam doar daca toti bitii sunt disponibili
 if ((flags & ALLBITS) != 0) repaint();
 // Daca sunt toti bitii nu mai sunt necesare
 // noi update-uri
 return (flags & (ALLBITS | ABORT)) == 0);
}
```

# Mecanismul de "double-buffering"

- implică realizarea unui desen în memorie și apoi transferul său pe ecran, pentru a elimina efectul neplăcut de "clipire" ("flickering") rezultat atunci când sunt efectuate redesenări repetate la intervale mici de timp (crearea de animații).

// Supradefinim update pentru a elimina stergerea desenului

```
public void update(Graphics g) {
 paint(g);
}
```

```
public void paint(Graphics g) {
 // Desenam in memorie pe un obiect de tip Image
 // w si h sunt dimensiunile desenului
 Image img = createImage(w, h);
 Graphics gmem = img.getGraphics();
 // Realizam desenul folosind gmem
 gmem.setColor(...);
 gmem.fillOval(...); ...
 // Transferam desenul din memorie pe ecran
 // desenand de fapt imaginea creata
 g.drawImage(img, 0, 0, this);
 gmem.dispose();
}
```

# Tipărirea componentelor

- java.awt.print
- Interfața Printable

```
public int print(Graphics g, PageFormat pf, int pageIndex)
 throws PrinterException {
 // Descrierea imaginii obiectului
 // Poate fi un apel la metoda paint: paint(g)
 if (ceva nu este in regula) {
 return Printable.NO_SUCH_PAGE;
 }
 return Printable.PAGE_EXISTS;
}
```

Etapele tipăririi:

1. Crearea unei sesiuni de tipărire: `PrinterJob.getPrinterJob`
2. Specificarea obiectului care va fi tipărit: `setPrintable`;
3. Opțional, inițierea unui dialog cu utilizatorul pentru precizarea unor parametri legați de tipărire: `printDialog`;
4. Tipărirea efectivă: `print`.

# Tipărirea unei componente

```
import java .io .*;
import java . awt .*;
import java . awt. event .*;
import java . awt. print .*;
import javax.swing.*;
class Plansa extends Canvas implements Printable {
 Dimension d = new Dimension (400 , 400) ;
 public Dimension getPreferredSize () { return d; }
 public void paint (Graphics g) {
 g. drawRect (200 , 200 , 100 , 100) ;
 g. drawOval (200 , 200 , 100 , 100) ;
 g. drawString (" Hello ", 200 , 200) ;
 }
 public int print (Graphics g, PageFormat pf , int pi) throws
 PrinterException {
 if (pi >= 1) return Printable . NO_SUCH_PAGE ;
 paint (g);
 g. drawString (" Numai la imprimanta ", 200 , 300) ;
 return Printable . PAGE_EXISTS ;
 }
}
```



# Tipărirea unei componente

```
class Fereastra extends JFrame implements ActionListener {
 private Plansa plansa = new Plansa ();
 private JButton print = new JButton (" Print ");
 public Fereastra (String titlu) {
 super (titlu);
 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 add (plansa , BorderLayout . CENTER);
 JPanel south = new JPanel ();
 south.setLayout(new FlowLayout());
 south .add(print);
 add (south , BorderLayout . SOUTH);
 print . addActionListener (this);
 pack ();
 }
 public void actionPerformed (ActionEvent e) {
 // 1. Crearea unei sesiuni de tiparire
 PrinterJob printJob = PrinterJob . getPrinterJob ();
 // 2. Stabilirea obiectului ce va fi tiparit
 printJob . setPrintable (plansa);
 }
}
```

# Tipărirea unei componente

```
// 3. Initierea dialogului cu utilizatorul
if (printJob . printDialog ()) {
 try {
 // 4. Tiparirea efectiva
 printJob . print ();
 } catch (PrinterException ex) {
 System . out . println (" Exceptie la tiparire !");
 ex . printStackTrace ();
 }
}
}
}
class TestPrint {
 public static void main (String args []) throws
 Exception {
 Fereastra f = new Fereastra (" Test Print ");
 f . show ();
 }
}
```

# Tipărirea textelor

- Flux către "lpt1" sau "/dev/lp".

```
import java .io .*;
import java . awt .*;
class TestPrintText {
 public static void main (String args []) throws Exception{
 // pentru Windows
 PrintWriter imp = new PrintWriter (new FileWriter
 ("lpt1"));
 /* pentru UNIX
 PrintWriter imp = new PrintWriter (new FileWriter (
 "/dev/lp "));*/
 imp . println (" Test imprimanta ");
 imp . println (" ABCDE ");
 imp . close ();
 }
}
```