




Tehnici avansate de programare

Curs -

Cristian Frăsinaru

`acf@infoiasi.ro`

Facultatea de Informatică
Universitatea "Al. I. Cuza" Iași



Desenarea (Java 2D)

Cuprins

- **Conceptul de desenare**
- **Metoda paint**
- **Desenarea componentelor**
- **Context grafic de desenare**
- **Primitive grafice de desenare**
- **Fonturi**
- **Culori**
- **Imagini**
- **Tipărirea**

Grafică pe calculator

"Grafică pe calculator = Reprezentarea și gestionarea de conținut vizual

Platforma Java oferă:

- API de tip "infrastructură" care permite lucrul cu: imagini, culori, fonturi, dimensiuni, etc.
- Grafică 2D → Java2D API
- Grafică 3D → Java3D API
- Grafică în modul *full-screen*

Conceptul "clasic" de desenare

Interfețele grafice sunt construite folosind componente. Desenarea acestora se face automat:

- la afișarea pentru prima dată;
- la operații de minimizare, maximizare, redimensionare a suprafeței de afișare;
- ca răspuns al unei solicitări explicite a programului.

Metodele de bază sunt în superclasa `Component`:

- **`void paint(Graphics g)`**
- **`void update(Graphics g)`**
- **`void repaint()`**

Metoda *paint*

Metoda **paint** din clasa `Component` oferă reprezentarea grafică a unei componente (descrie desenul acesteia).

```
public class MyFrame extends Frame {  
    public MyFrame(String title) {  
        super(title);  
        setSize(200, 100);  
    }  
  
    public void paint(Graphics g) {  
        super.paint(g); // Apelam metoda paint a clasei Frame  
        g.setFont(new Font("Arial", Font.BOLD, 11));  
        g.setColor(Color.red);  
        g.drawString("Aplicatie DEMO", 5, 35);  
    }  
}
```

Metoda *paintComponent*

In Swing, componentele sunt obiecte de tip *Container*, deci pot avea fii. Metoda **paintComponent** descrie desenul componentei propriu-zise.

Metoda **paint** va face următoarele invocări:

- **paintComponent**
- **paintBorder**
- **paintChildren**

Crearea unei componente proprii

- Extindem o clasă suport care nu are reprezentare vizuală: `java.awt.Canvas`, `javax.swing.Panel`, etc.
- Definim metoda `paint` sau `paintComponent`
- Stabilim dimensiunile *preferată, minimă, maximă*

```
class MyComponent extends JPanel {  
  
    public void paintComponent(Graphics g) { ... }  
  
    // Metodele folosite de gestionarii de pozitionare  
    public Dimension getPreferredSize() { return ... };  
    public Dimension getMinimumSize() { return ... }  
    public Dimension getMaximumSize() { return ... }  
}
```


Contextul grafic de desenare

Un **context grafic** este un obiect de tip *Graphics* sau *Graphics2D* folosit pentru desenare într-un spațiu de coordonate *virtual*, numit *spațiul utilizator*. Spațiul utilizator este apoi mapat la un spațiu concret, *specific unui dispozitiv*:

- ecran, imprimantă, memorie, etc.

Un context grafic de desenare oferă:

- **primitive grafice**: desenarea de figuri geometrice, texte, imagini, etc.
- modalități de configurare a **proprietăților**
culoare, font, originea coordonatelor, suprafața vizibilă, modul de desenare, etc.

Proprietățile contextului grafic

Culoarea de desenare	<i>Color</i> <i>Paint</i>
Fontul de scriere a textelor	<i>Font</i>
Grosimea peniței	<i>Stroke</i>
Preferințe de desenare	<i>RenderingHints</i>
Zona de decupare	<i>Clip</i>
Modul de desenare	<i>XorMode</i>
Modul de compunere	<i>Composite</i>
Modul de transformare	<i>Transform</i>
...	...

Primitive grafice

Text	<code>drawString</code>
Imagine	<code>drawImage</code>
Linie	<code>drawLine</code> <code>drawPolyline</code>
Dreptunghi	<code>drawRect, fillRect</code> <code>draw3DRect, fill3DRect</code> <code>drawRoundRect, fillRoundRect</code>
Poligon	<code>drawPolygon, fillPolygon</code>
Oval (Elipsă)	<code>drawOval, fillOval</code>
Arc de cerc	<code>drawArc, fillArc</code>
...	...
Formă geometrică	<code>draw(Shape), fill(Shape)</code>
Operații	<code>clip, rotate, translate,</code> <code>scale, shear</code>

Folosirea fonturilor

- **Font** = colecție de simboluri (*glyphs*).
- Fonturi înrudite formează o **familie**: Helvetica, Arial, etc.
- **Numele** fontului identifică exact un font recunoscut de sistem: Helvetica Bold, Arial Bold Italic, etc.
- Fonturile pot fi:
 - **fizice**: unul din fonturile instalate în SO
 - **logice**: abstracte, specifice platformei Java - sunt mapate corespunzător cu fonturi fizice la runtime.
(Dialog, DialogInput, Monospaced, Serif, SansSerif)
- Un font are parametri: **înălțime** și **stil**
- Dimensiunile caracterelor: **metrica fontului**.

Clasa *Font*

Clasa `Font` încapsulează toate informațiile fontului, mai puțin metrica sa.

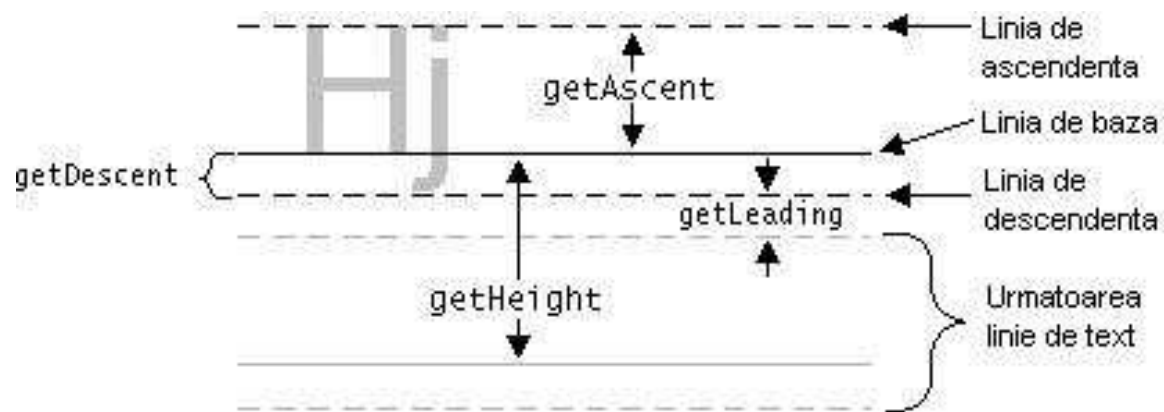
```
Font(String name, int style, int size)

new Font("Dialog", Font.PLAIN, 12);
new Font("Arial", Font.ITALIC, 14);
new Font("Courier", Font.BOLD, 10);

// Pentru componente
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);
```

Clasa *FontMetrics*



```
public void paint(Graphics g) {  
    Font f = new Font("Arial", Font.BOLD, 11);  
    FontMetrics fm = g.getFontMetrics();  
  
    int height = fm.getHeight();  
    int helloWidth = fm.stringWidth("Hello");  
    int xWidth = fm.charWidth('x');  
}
```

Folosirea culorilor

Interfața Paint

- **Color:** *Red Green Blue Alpha* (0 – 255, 0.0 – 1.0)

```
Color rosu = Color.red;  
Color negru = new Color(0, 0, 0);  
Color rosuTransparent = new Color(255, 0, 0, 128);
```

- **SystemColor:** `SystemColor.desktop`

- **GradientColor**

```
GradientPaint(Point2D p1, Color c1, Point2D p2, Color c2)
```

- **TexturePaint**

```
TexturePaint(BufferedImage txtr, Rectangle2D anchor)
```

Folosirea imaginilor

- **java.awt.Image** (abstractă)

- **java.awt.BufferedImage**

- Formate: *GIF, PNG, JPEG, etc.*

- Incărcarea unei imagini:

```
BufferedImage image = ImageIO.read(new File("hello.jpg"))
```

- Crearea unei imagini:

```
BufferedImage bi = new BufferedImage(w, h, type);
```

```
Graphics g = bi.getGraphics();
```

- Desenarea unei imagini: `graphics.drawImage`

- Salvarea unei imagini într-un fișier: `ImageIO.write`

Desenarea unei imagini



```
BufferedImage img = ImageIO.read(  
    new URL("http://www.remoteServer.com/hugeImage.jpg"));
```

```
public void paint(Graphics g) {  
    g.drawImage(img, 0, 0, this);  
    g.drawImage(img, 0, 200, 100, 100, this);  
}
```

```
//Formatul cel mai general al metodei drawImage  
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
```

```
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'

Eliminarea efectului de "flickering".

```
// Supradefinim update pentru a elimina stergerea desenului
public void update(Graphics g) {
    paint(g);
}
public void paint(Graphics g) {
    // Desenam in memorie intr-un obiect de tip BufferedImage
    BufferedImage offImage =
        new BufferedImage(100, 200, BufferedImage.TYPE_INT_ARGB);
    Graphics2D g2 = offImage.getGraphics();
    // Realizam desenul folosind g2
    g2.setColor(...); g2.fillOval(...); ...

    // Transferam desenul din memorie pe ecran,
    // desenand de fapt imaginea creata
    g.drawImage(offImage, 0, 0, this);
    g2.dispose();
}
}
```

Tipărirea

java.awt.print

1. Crearea unei sesiuni de tipărire:

`PrinterJob.getPrinterJob`

2. Specificarea obiectului care va fi tipărit: `setPrintable`
Acesta trebuie să implementeze interfața *Printable*

3. Opțional, inițierea unui dialog cu utilizatorul pentru precizarea unor parametri legați de tipărire:

`printDialog;`

4. Tipărirea efectivă: `print.`

Interfața *Printable*



```
public int print(Graphics g, PageFormat pf, int pageIndex)
    throws PrinterException {

    if (ceva nu este in regula) {
        return Printable.NO_SUCH_PAGE;
    }

    // Descrierea grafica a componentei in vederea tiparirii
    paint(g); // poate fi, dar nu neaparat
    g.drawString("Numai la imprimanta", 200, 300);

    return Printable.PAGE_EXISTS;
}
```



Java 3D

- Independent de platformă
- Interfață peste *OpenGL* sau *DirectX*
- *Graful scenei* - arbore de obiecte 3D
- *Vizualizare* dinamică
- *Umbre*
- *Sunet "spațial"*
- Suport pentru formate ca VRML
- ..

Nu este inclus kit-ul standard (JDK) (instalare separată)

Java Tutorial



- Trail: **2D Graphics**

<http://docs.oracle.com/javase/tutorial/2d/index.html>

- Lesson: **Full-Screen Exclusive Mode API**

<http://docs.oracle.com/javase/tutorial/extra/fullscreen/index.html>

- Trail: **Sound**

<http://docs.oracle.com/javase/tutorial/sound/index.html>

