

Desenarea

- Conceptul de desenare
- Metoda paint
- Suprafețe de desenare
- Folosirea fonturilor
- Folosirea culorilor
- Folosirea imaginilor
- Mecanismul de ”double-buffering”
- Salvarea desenelor în format JPEG
- 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

Conceptul de desenare

Desenarea componentelor 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.

Metode

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

Metoda paint

Responsabilă cu desenarea unei componente. Definită în superclasa **Component**.

Listing 1: Supradefinirea metodei paint

```
import java.awt.*;
class Fereastră extends Frame {
    public Fereastră(String titlu) {
        super(titlu);
        setSize(200, 100);
    }

    public void paint(Graphics g) {
        // Apelăm metoda paint a clasei Frame
        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[]) {
        Fereastră f = new Fereastră("Test paint");
        f.show();
    }
}
```

Suprafețe de desenare

Clasa Canvas

```
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
    ...
}
```

Listing 2: Folosirea clasei Canvas

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

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);
    }

    public Dimension getPreferredSize() {
        return dim;
    }
}

class Fereastra extends Frame {
    public Fereastra(String titlu) {
        super(titlu);
        setSize(200, 200);
        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 sau
- 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

Figură geometrică	Metode
Linie	drawLine drawPolyline
Dreptunghi simplu	drawRect fillRect clearRect
Dreptunghi cu chenar ”ridicat” sau ”adâncit”	draw3DRect fill3DRect
Dreptunghi cu colțuri retunjite	drawRoundRect 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: **îngrosat (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();
```

Listing 3: Lucrul cu fonturi

```
import java.awt.*;

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);
        }
    }

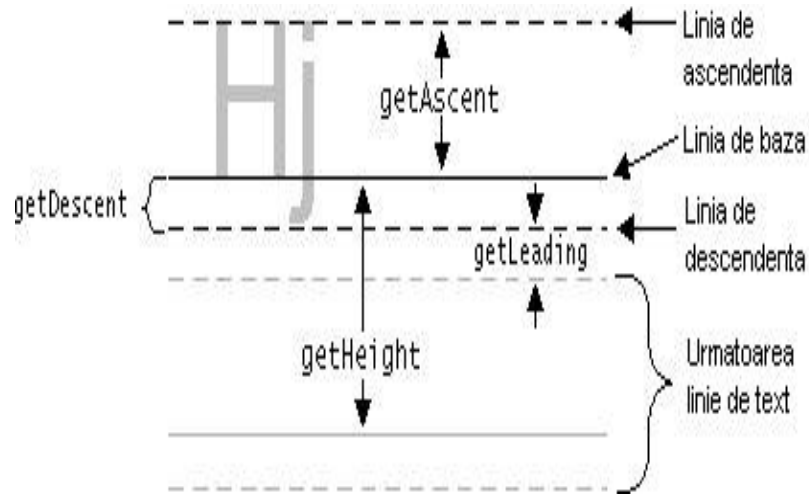
    public Dimension getPreferredSize() {
        return canvasSize;
    }
}

class Fereastră extends Frame {
    public Fereastră(String titlu) {
        super(titlu);
        ScrollPane sp = new ScrollPane();
        sp.setSize(400, 400);
        sp.add(new Fonturi());
        add(sp, BorderLayout.CENTER);
        pack();
    }
}

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

Clasa FontMetrics

Informații despre **metrica** unui font.



```
public void paint(Graphics g) {  
    Font f = new Font("Arial", Font.BOLD, 11);  
    FontMetrics fm = g.getFontMetrics();  
}
```

Metode

- `getHeight`
- `stringWidth`
- `charWidth`

Folosirea culorilor

Red Green Blue Alpha (0 – 255,
0.0 – 1.0)

Clase: **Color**, **SystemColor**

Constante

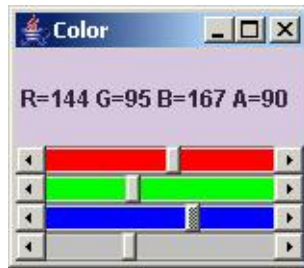
```
Color rosu = Color.red;  
Color galben = Color.yellow;  
Color fundal = SystemColor.desktop;
```

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, get...



```
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;

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"));
```

Afişarea 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"

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 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();
}
}
```

Salvarea desenelor în format JPEG

```
import com.sun.image.codec.jpeg.*;
...
class JPEGWriter {
    static float quality = 0.9f; //intre 0 si 1
    public static void write(BufferedImage img,
        String filename) {
        try {
            FileOutputStream out =
                new FileOutputStream(filename);
            JPEGImageEncoder encoder =
                JPEGCodec.createJPEGEncoder(out);
            JPEGEncodeParam jep =
                encoder.getDefaultJPEGEncodeParam(img);
            jep.setQuality(quality, false);
            // Folosim setarile de codare jpeg implicite
            encoder.setJPEGEncodeParam(jep);
            encoder.encode(img);
            out.close();
        } catch( Exception e ) {
            e.printStackTrace();
        }
    }
}
```

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`.

Listing 4: Tipărirea unei componente

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

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;
    }
}

class Fereastra extends Frame implements ActionListener {
    private Plansa plansa = new Plansa();
    private Button print = new Button("Print");

    public Fereastra(String titlu) {
        super(titlu);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });

        add(plansa, BorderLayout.CENTER);
    }
}
```

```

Panel south = new Panel();
south.setLayout(new BorderLayout(BorderLayout.CENTER));
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);

    // 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();
        }
    }
}

}

}

public class TestPrint {
    public static void main(String args[]) throws Exception {
        Fereastra f = new Fereastra("Test Print");
        f.show();
    }
}

```

Tiparirea textelor

Flux către "lpt1" sau "/dev/lp".

Listing 5: Tipărire textelor

```
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();
    }
}
```

Java 2D - Concepte

- *Shape* - forme geometrice
- *Paint* - culori de tip gradient
- *Composition* - compunerea desenării
- *Texture* - texturi
- *Transforms* - rotații, translații, etc.
- *Stroke* - creare de "penițe"
- Optimizări: *anti-aliasing*
- Interoperabilitate cu OpenGL

/demo/jfc/Java2D/Java2D.jar

Java 3D - Concepte

- Independent de platformă
- Interfață peste *OpenGL* sau *Direct3D*
- *Graficul scenei* - arbore de obiecte 3D
- *Vizualizare* dinamică
- *Umbre*
- *Sunet "spațial"*
- Suport pentru formate ca VRML
- etc.

Nu este inclus in JDK 6.0 (download separat)