Aplicatii JAVA

9

JAVA
Utilizarea desenării în Java

Adrian Runceanu
www.runceanu.ro/adrian

Curs 9

Utilizarea desenării în Java

Utilizarea desenării în Java

1. Conceptul de desenare
2. Desenarea obiectelor - metoda paint()
3. Suprafete de desenare - clasa Canvas
4. Contextul grafic de desenare - clasa Graphics
    1. Proprietatile unui context grafic
    2. Primitive grafice
5. Desenarea textelor
6. Desenarea figurilor geometrice

1. Conceptul de desenare

- Un program **Java** care are interfata grafica cu utilizatorul trebuie sa deseneze pe ecran toate componentele sale care au o reprezentarea grafica vizuala.

- Aceasta desenare include componentele vizuale standard folosite în program precum si obiectele grafice definite de catre programator.

1. Conceptul de desenare

Desenarea componentelor se face automat si este un proces care se executa în următoarele situații:

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

1. Conceptul de desenare

Metodele care controlează procesul de desenare se găsescîn clasa **Component** și sunt următoarele:

1. **void paint(Graphics g)** - Desenează o componentă.

Este o metodă supradefinită de fiecare componentă în parte pentru a furniza reprezentarea sa grafică specifică.

Metoda este apelată de fiecare dată când conținutul componentei trebuie desenat sau redesenat și nu va fi apelată explicit.

1. Conceptul de desenare

2. **void update(Graphics g)** - Actualizeaza starea grafica a unei componente.

Actiunea acestei metode se realizeaza in trei pasi:

1. sterge componenta prin supradesenarea ei cu culoarea fundalului

2. stabileste culoarea (foreground) a componentei

3. apeleaza metoda paint pentru a redesena componenta

1. Conceptul de desenare

3. **void repaint()** - Executa explicit un apel al metodei update pentru a actualiza reprezentarea grafica a unei componente.

Dupa cum se observa, singurul argument al metodelor paint si update este un obiect de tip **Graphics**.

Acesta reprezinta *contextul grafic in care se executa desenarea componentelor*

Utilizarea desenării în Java

1. Conceptul de desenare
2. Desenarea obiectelor - metoda paint()
3. Suprafete de desenare - clasa Canvas
4. Contextul grafic de desenare - clasa Graphics
    1. Proprietatile unui context grafic
    2. Primitive grafice
5. Desenarea textelor
6. Desenarea figurilor geometrice

2. Desenarea obiectelor - metoda paint()

- Toate desenele care trebuie sa apara pe o suprafata de desenare se realizeaza în metoda **paint** a unei componente.

- Metoda **paint** este definita în superclasa **Component** însa nu are nici o implementare si, din acest motiv, orice obiect grafic care doreste sa se deseneze trebuie sa o supradefineasca pentru a-si crea propria sa reprezentare.

2. Desenarea obiectelor - metoda paint()

Componentele standard **AWT** au deja supradefinita aceasta metoda deci nu trebuie sa ne preocupe

desenarea lor, însa putem modifica reprezentarea lor grafica prin crearea unei subclase si supradefinirea metodei paint, având însa grija sa apelam si metoda superclasei care se ocupa cu desenarea efectiva a componentei.

2. Desenarea obiectelor - metoda paint()

In exemplul de mai jos, redefinim metoda paint pentru un obiect de tip **Frame**, pentru a crea o clasa ce instantiaza ferestre pentru o aplicatie demonstrativa (în coltul stânga sus este afisat textul "Aplicatie Grafica").

2. Desenarea obiectelor - metoda paint()

```
import java.awt.*;
class Fereastra extends Frame {
        public Fereastra(String titlu) {
                super(titlu);
                setSize(200, 100);
        }
        public void paint(Graphics g) {
                super.paint(g);  // apelez metoda paint a clasei Frame
                g.setFont(new Font("Arial", Font.BOLD, 11));
                g.setColor(Color.red);
                g.drawString("Aplicatie Grafica", 5, 35);
        }
}
public class TestPaint {
        public static void main(String args[]) {
                Fereastra f = new Fereastra("Test Paint");
                f.show();
        }
}
```

2. Desenarea obiectelor - metoda paint()

Se observa ca la orice redimensionare a ferestrei textul "Aplicatie Grafica" va fi redesenat.

Daca desenarea acestui text ar fi fost facuta oriunde în alta parte decât în metoda paint, la prima redimensionare a ferestrei acesta s-ar pierde.

Asadar, desenarea în Java trebuie sa se faca doar în cadrul metodelor paint ale componentelor grafice.

Utilizarea desenării în Java

1. Conceptul de desenare
2. Desenarea obiectelor - metoda paint()
3. Suprafete de desenare - clasa Canvas
4. Contextul grafic de desenare - clasa Graphics
    1. Proprietatile unui context grafic
    2. Primitive grafice
5. Desenarea textelor
6. Desenarea figurilor geometrice

3. Suprafete de desenare - clasa Canvas

- In afara posibilitatii de a utiliza componente grafice standard, **Java** ofera si posibilitatea controlului la nivel de punct (pixel) pe dispozitivul grafic, respectiv desenarea a diferite forme grafice direct pe suprafata unei componente.
- Desi este posibil, în general nu se deseneaza la nivel de pixel direct pe suprafata ferestrelor sau a altor suprafete de afisare.
- In **Java** a fost definit un tip special de componenta numita **Canvas** (pânza de pictor), a carui scop este de a fi extinsa pentru a implementa obiecte grafice cu o anumita înfatisare.

3. Suprafete de desenare - clasa Canvas

- Asadar clasa **Canvas** este o clasa generica din care se deriveaza subclase pentru crearea suprafetelor de desenare (planse).

- Plansele nu pot contine alte componente grafice, ele fiind utilizate doar ca suprafete de desenat sau ca fundal pentru animatie.

- Desenarea pe o plansa se face prin supradefinirea metodei **paint**.

3. Suprafete de desenare - clasa Canvas

- Concret, *o plansa este suprafata dreptunghiulara de culoare alba pe care se poate desena*.
- Implicit dimensiunile plansei sunt 0 si, din acest motiv, gestionarii de pozitionare nu vor avea la dispozitie dimensiuni implicite pentru afisarea unui obiect de tip **Canvas**.
- Pentru a evita acest neajuns este recomandat ca o plansa sa redefineasca si metodele **getMinimumSize**, **getMaximumSize**, **getPreferredSize** pentru a-si specifica dimensiunile implicite.

3. Suprafete de desenare - clasa Canvas

Etapele care trebuie parcurse pentru crearea unui desen, sau mai bine zis, a unui obiect grafic cu o anumita înfatisare sunt:

1. crearea unei planse de desenare, adica o subclasa a clasei **Canvas**
2. redefinirea metodei **paint** din clasa respectiva
3. redefinirea metodelor **getMinimumSize**, **getMaximumSize**, **getPreferredSize**
4. desenarea efectiva a componentei în cadrul metodei **paint**
5. adaugarea plansei la un container cu metoda **add**.
6. interceptarea evenimentelor de tip **FocusEvent**, **KeyEvent**, **MouseEvent**, **ComponentEvent** si tratarea lor (daca este cazul).

3. Suprafete de desenare - clasa Canvas

Definirea generica a unei planse are urmatorul format:

```
class Plansa extends Canvas {
    public void paint(Graphics g) {
    . . .
    // desenarea
    }
    public Dimension getMinimumSize() {
    return . . .
    }
    public Dimension getMaximumSize() {
    return . . .
    }
    public Dimension getPreferredSize() {
    return . . .;
}
```

3. Suprafete de desenare - clasa Canvas

Exemplu: Sa definim o plansa pe care desenam un patrat si cercul sau circumscris. Plansa o vom afisa apoi pe o fereastra.

```
import java.awt.*;
class Plansa extends Canvas {
        Dimension canvasSize = new Dimension(100, 100);
        public void paint(Graphics g) {
                g.setColor(Color.red);
                g.drawRect(0, 0, 100, 100);
                g.setColor(Color.blue);
                g.drawOval(0, 0, 100, 100);
        }
```

3. Suprafete de desenare - clasa Canvas

```
        public Dimension getMinimumSize() {
                return canvasSize;
        }
        public Dimension getPreferredSize() {
                return canvasSize;
        }
}
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[]) {
                                Fereastra f = new Fereastra("Test Paint");
                                f.show();
                }
}
```

Utilizarea desenării în Java

1. Conceptul de desenare
2. Desenarea obiectelor - metoda paint()
3. Suprafete de desenare - clasa Canvas
4. Contextul grafic de desenare - clasa Graphics
    1. Proprietatile unui context grafic
    2. Primitive grafice
5. Desenarea textelor
6. Desenarea figurilor geometrice

4. Contextul grafic de desenare - clasa **Graphics**

- Inainte ca utilizatorul sa poata desena el trebuie sa obtina un ***context grafic*** de desenare pentru suprafata careia îi apartine regiunea pe care se va desena.

- *Un context grafic este, de fapt, un obiect prin intermediul caruia putem controla procesul de desenare a unui obiect.*

In general desenarea se poate face:

- ✓ pe o portiune de ecran
- ✓ la imprimanta sau
- ✓ într-o zona virtuala de memorie

4. Contextul grafic de desenare - clasa **Graphics**

- Un context grafic este specificat prin intermediul obiectelor de tip **Graphics** primite ca parametru în metodele **paint** si **update**.

- In functie de dispozitivul fizic pe care se face afisarea (ecran, imprimanta, plotter, etc) metodele de desenare au implementari interne diferite, transparente utilizatorului.

4. Contextul grafic de desenare - clasa **Graphics**

Clasa **Graphics** pune la dispozitie metode pentru:

- **primitive grafice**: desenarea de figuri geometrice, texte si imagini

- **stabilirea proprietatilor unui context grafic**, adica:
  - ✓ stabilirea culorii si fontului curente cu care se face desenarea
  - ✓ stabilirea originii coordonatelor suprafetei de desenare
  - ✓ stabilirea suprafetei în care sunt vizibile componentelor desenate
  - ✓ stabilirea modului de desenare

4. Contextul grafic de desenare - clasa **Graphics**

Proprietatile contextului grafic

- La orice tip de desenare parametrii legati de culoare, font, etc. sunt specificati de contextul grafic în care se face desenarea.

- Enumeram aceste proprietati si metodele asociate lor în clasa **Graphics**.

4. Contextul grafic de desenare - clasa **Graphics**

proprietati	metodele asociate lor în clasa **Graphics**
culoarea curenta de desenare	**Color getColor()** **void setColor(Color c)**
fontul curent cu care vor fi scrise textele	**Font getFont()** **void setFont(Font f)**
originea coordonatelor - poate fi modificata	**translate(int x, int y)**
zona de decupare: zona în care sunt vizibile desenele	**Shape getClip()** **void setClip(Shape s)** **void setClip(int x, int y, int width, int height)**
modul de desenare	**void setXorMode(Color c1) - desenare "sau exclusiv"** **void setPaintMode(Color c1) - supradesenare**

4. Contextul grafic de desenare - clasa **Graphics**

Primitive grafice

Prin primitive grafice ne vom referi în continuare la metodele clasei **Graphics** care permit desenarea de figuri geometrice si texte.

4. Contextul grafic de desenare - clasa **Graphics**

Desenarea textelor

Desenarea textelor de face cu metodele:

drawString(String str, int x, int y)
drawBytes(bytest data, int offset, int length, int x, int y)
drawChars(charst data, int offset, int length, int x, int y)

unde x si y reprezinta coltul din stânga-jos al textului.
Textul desenat va avea culoarea curenta a contextului grafic.

4. Contextul grafic de desenare - clasa **Graphics**

Desenarea figurilor geometrice

proprietati	metodele asociate lor în clasa **Graphics**
linii	**drawLine(int x1, int y1, int x2, int y2)** **drawPolyline(intst xPoints, intst yPoints, int nPoints)**
dreptunghiuri simple	**drawRect(int x, int y, int width, int height)** **fillRect(int x, int y, int width, int height)** **clearRect(int x, int y, int width, int height)**
dreptunghiuri cu chenar "ridicat" sau "adâncit"	**draw3DRect(int x, int y, int width, int height, boolean raised)** **fill3DRect(int x, int y, int width, int height, boolean raised)**
dreptunghiuri cu colturi rotunjite	**drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)** **fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight**

4. Contextul grafic de desenare - clasa **Graphics**

Desenarea figurilor geometrice (continuare)

proprietati	metodele asociate lor în clasa **Graphics**
ovaluri	**drawOval(int x, int y, int width, int height)** **fillOval(int x, int y, int width, int height)**
arce circulare sau eliptice	**drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)** **fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)**
poligoane	**drawPolygon(intst xPoints, intst yPoints, int nPoints)** **drawPolygon(Polygon p)** **fillPolygon(intst xPoints, intst yPoints, int nPoints)** **fillPolygon(Polygon p)**

Utilizarea desenării în Java

7. Folosirea fonturilor
    1. Clasa Font
    2. Clasa FontMetrics
8. Folosirea culorilor
9. Folosirea imaginilor
    1. Incarcarea unei imagini dintr-un fisier
    2. Afisarea imaginilor
    3. Monitorizarea încarcarii imaginilor - interfata ImageObserver
    4. Crearea imaginilor în memorie - clasa MemoryImageSource
10. Tiparirea

7. Folosirea fonturilor

Pentru a scrie un text pe ecran avem doua posibilitati:

1. Prima dintre acestea este sa folosim o componenta orientata-text cum ar fi Label, TextField sau TextArea
2. A doua sa este sa apelam la metodele clasei **Graphics** de desenare a textelor: drawString, drawChars, drawBytes

Indiferent de modalitatea aleasa, putem specifica prin intermediul fonturilor cum sa arate textul respectiv, acest lucru realizându-se prin metoda clasei **Component**, respectiv **Graphics**:

setFont(Font f)

7. Folosirea fonturilor

Cei mai importanti parametri ce caracterizeaza un font sunt:

1. numele fontului: Helvetica Bold, Arial Bold Italic, etc
2. familia din care face parte fontul: Helvetica, Arial, etc
3. dimensiunea fontului: înaltimea sa
4. stilul fontului: îngrosat (bold), înclinat (italic)
5. metrica fontului

Clasele care ofera suport pentru lucrul cu fonturi sunt **Font** si **FontMetrics**.

7. Folosirea fonturilor

7.1. Clasa Font

Un obiect de tip **Font** încapsuleaza informatii despre toti parametrii unui font, mai putin despre metrica acestuia.

Constructorul uzual al clasei este cel care primeste ca argumente numele fontului, dimensiunea si stilul:

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

7. Folosirea fonturilor

Stilul unui font este specificat prin intermediul constantelor:

Font.PLAIN - normal
Font.BOLD - îngrosat
Font.ITALIC - înclinat

Exemple:

```
new Font("Arial", Font.BOLD, 12);
new Font("Times New Roman", Font.ITALIC, 14);
new Font("Courier New", Font.PLAIN, 10);
```

7. Folosirea fonturilor

Folosirea unui obiect de tip **Font** se realizeaza uzual astfel:

1. pentru componente etichetate

```
Label label = new Label("Text Java");
label.setFont(new Font("Arial", Font.BOLD, 12));
```

2. in metoda **paint(Graphics g)**

```
g.setFont(new Font("Times New Roman", Font.ITALIC, 14));
g.drawString("Text Java", 0, 0);
```

7. Folosirea fonturilor

O platforma de lucru are instalate, la un moment dat, o serie întreaga de fonturi care sunt disponibile pentru scrierea textelor.

Lista acestor fonturi se poate obtine cu metoda getAllFonts a clasei **GraphicsEnvironment** astfel:

```
Font[] fonturi =
GraphicsEnvironment.getLocalGraphicsEnvironment().getAllFonts();
```

7. Folosirea fonturilor

Exemplul urmator afiseaza lista primelor 20 de fonturi disponibile pe platforma curenta de lucru.
Textul fiecarui nume de font va fi scris cu fontul sau corespunzator.

```
import java.awt.*;
class Fonturi extends Canvas {
        private Font[] fonturi;
        Dimension canvasSize = new Dimension(400, 400);
        public Fonturi() {
                setSize(canvasSize);
                fonturi = GraphicsEnvironment.
                getLocalGraphicsEnvironment().getAllFonts();
        }
```

7. Folosirea fonturilor

```
    public void paint(Graphics g) {
        String nume;
        for(int i=0; i < 20; i++) {
          nume = fonturi[i].getFontName();
          g.setFont(new Font(nume, Font.PLAIN, 14));
          g.drawString(nume, 20, (i + 1) * 20);
        }
    }
    public Dimension getMinimumSize() {
         return canvasSize;
    }
    public Dimension getPreferredSize() {
         return canvasSize;
    }
}
```

7. Folosirea fonturilor

```
class Fereastra extends Frame {
        public Fereastra(String titlu) {
                super(titlu);
                add(new Fonturi(),BorderLayout.CENTER);
                pack();
        }
}
public class TestAllFonts {
        public static void main(String args[]) {
                Fereastra f = new Fereastra("All fonts");
                f.show();
        }
}
```

7. Folosirea fonturilor

7.2. Clasa FontMetrics

- La afisarea unui sir cu metoda **drawString** trebuie sa specificam pozitia la care sa apara sirul pe ecran.
- In momentul în care avem de afisat mai multe siruri trebuie sa calculam pozitiile lor de afisare în functie de lungimea si înaltimea în pixeli a textului fiecarui sir.
- Pentru aceasta este folosita clasa **FontMetrics**.

7. Folosirea fonturilor

- Un obiect din aceasta clasa se construieste pornind de la un obiect de tip **Font** si pune la dispozitie informatii despre dimensiunile în pixeli pe care le au caracterele fontului respectiv.
- Asadar, un obiect de tip **FontMetrics** încapsuleaza informatii despre metrica unui font, cu alte cuvinte despre dimensiunile în pixeli ale caracterelor sale.

- *Utilitatea principala a acestei clase consta în faptul ca permite pozitionarea precisa a textelor pe o suprafata de desenare, indiferent de fontul folosit de acestea.*

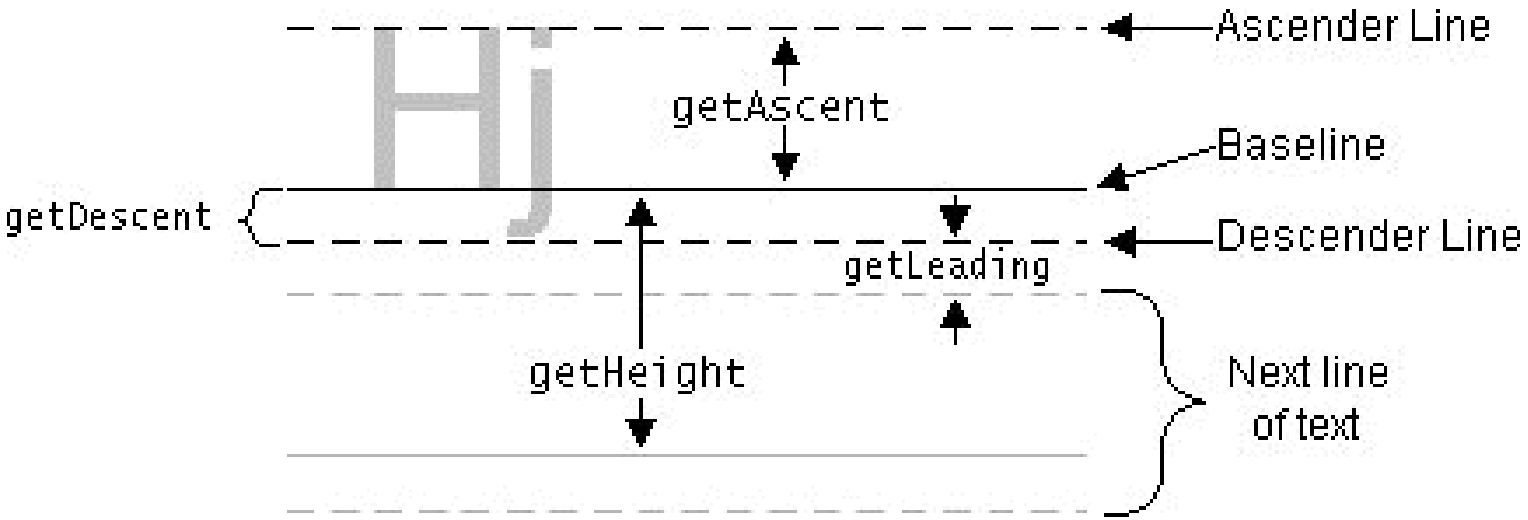
7. Folosirea fonturilor

Metrica unui font consta în urmatoarele atribute pe care le au caracterele unui font:

- ✓ linia de baza: este linia dupa care sunt aliniate caracterele unui font
- ✓ linia de ascendenta: linia superioara pe care nu o depaseste nici un caracter din font
- ✓ linia de descendenta: linia inferioara sub care nu coboara nici un caracter din font
- ✓ ascendentul: distanta între linia de baza si linia de ascendenta
- ✓ descendentul: distanta între linia de baza si linia de descendenta
- ✓ latimea: latimea unui anumit caracter din font
- ✓ înaltimea: distanta între liniile de baza
- ✓ distanta între linii ("leading"): distanta optima între doua linii de text scrise cu acelasi font

7. Folosirea fonturilor

Figura de mai jos prezinta o imagine grafica asupra metricii unui font:



Utilizarea desenării în Java

7. Folosirea fonturilor
    1. Clasa Font
    2. Clasa FontMetrics
8. Folosirea culorilor
9. Folosirea imaginilor
    1. Incarcarea unei imagini dintr-un fisier
    2. Afisarea imaginilor
    3. Monitorizarea încarcarii imaginilor - interfata ImageObserver
    4. Crearea imaginilor în memorie - clasa MemoryImageSource
10. Tiparirea

8. Folosirea culorilor

Orice culoare este formata prin combinatia culorilor standard rosu (Red), verde (Green) si albastru (Blue), la care se adauga un anumit *grad de transparenta (Alpha)*.

Fiecare din acesti patru parametri poate varia într-un interval:

- cuprins fie între 0 si 255 (daca dorim sa specificam valorile prin numere întregi)
- fie între 0.0 si 1.0 (daca dorim sa specificam valorile prin numere reale)

8. Folosirea culorilor

O culoare este reprezentata printr-o instanta a clasei **Color** sau a subclasei sale **SystemColor**.

Pentru a crea o culoare avem doua posibilitati:

1. sa folosim una din constantele definite într-un din cele doua clase

2. sa folosim unul din constructorii clasei **Color**.

8. Folosirea culorilor

Sa vedem mai întâi care sunt constantele definite în aceste clase:

Color	SystemColor
black	activeCaption
blue	activeCaptionBorder
cyan	activeCaptionText
darkGray	control
gray	controlHighlight
green	controlShadow
lightGray	contolText
magenta	desktop
orange	menu
pink	text
red	textHighlight
white	window
yellow	. . .

Observati ca în clasa **Color** sunt definite culori uzuale din paleta standard de culori, în timp ce în clasa **SystemColor** sunt definite culorile componentelor standard (ferestre, texte, meniuri, etc) ale platformei curente de lucru.

8. Folosirea culorilor

Folosirea acestor constante se face astfel:

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

Daca nici una din aceste culori predefinite nu corespunde preferintelor noastre atunci putem crea noi culori prin intermediul constructorilor clasei **Color**:

```
Color(float r, float g, float b)
Color(float r, float g, float b, float a)
Color(int r, int g, int b)
Color(int r, int g, int b, int a)
Color(int rgb)
```

unde r, g, b, a sunt valorile pentru rosu, verde, albastru si transparenta (alpha) iar parametrul "rgb" de la ultimul constructor reprezinta un întreg format din: bitii 16-23 rosu, 8-15 verde, 0-7 albastru.

8. Folosirea culorilor

- Valorile argumentelor variaza între 0-255 pentru tipul int, respectiv 0.0-1.0 pentru tipul float.
- Valoarea 255 (sau 1.0) pentru transparenta specifica faptul ca respectiva culoare este complet opaca,
- iar valoarea 0 (sau 0.0) specifica transparenta totala. Implicit, culorile sunt complet opace.

Exemple de folosire a constructorilor:

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

8. Folosirea culorilor

Metodele cele mai folosite ale clasei **Color** sunt:

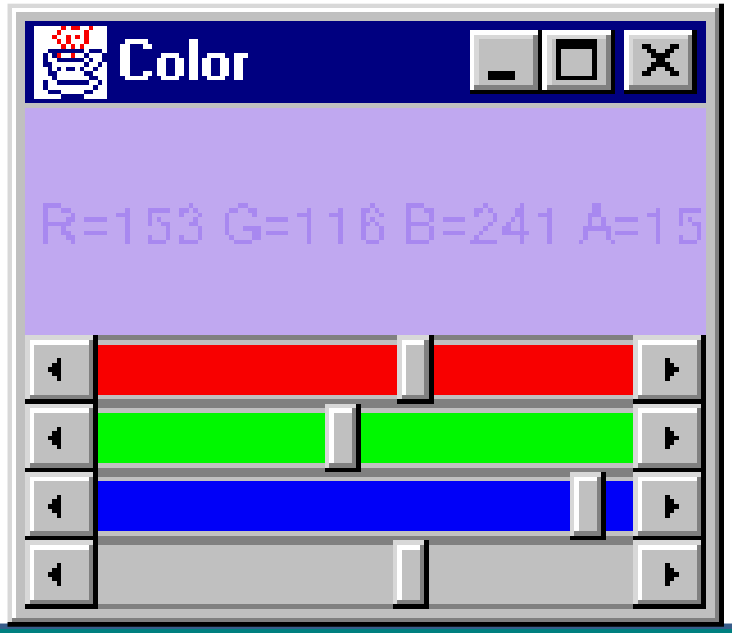
`Color brighter()` `Color darker()`	Creeaza o noua versiune a culorii curent mai deschisa / închisa

`int getAlpha()`
`int getRed()`
`int getGreen()`
`int getBlue()`
	Determina parametrii din care este alcatuita culoarea

`int getRGB()`	Determina valoarea ce reprezinta culoarea respectiva (bitii 16-23 rosu, 8-15 verde, 0-7 albastru)

8. Folosirea culorilor

Sa consideram o aplicatie cu ajutorul careia putem vizualiza dinamic culorile obtinute prin diferite combinatii ale parametrilor ce formeaza o culoare. Aplicatia va arata astfel:



8. Folosirea culorilor

```
import java.awt.*;
import java.awt.event.*;
class Culoare extends Canvas {
		public Color color = new Color(0, 0, 0, 255);
		Dimension canvasSize = new Dimension(150, 50);
		public Culoare() { setSize(canvasSize);
}
public void paint(Graphics g) {
		g.setColor(color);
		g.fillRect(0, 0, canvasSize.width, canvasSize.height);
		String text = "";
		text += " R=" + color.getRed();
		text += " G=" + color.getGreen();
		text += " B=" + color.getBlue();
		text += " A=" + color.getAlpha();
		g.drawString(text, 0, 30);
		}
public Dimension getPreferredSize() {		return canvasSize;	}
}
```

8. Folosirea culorilor

```
// fereastra principala
class Fereastra extends Frame implements AdjustmentListener {
        private Scrollbar rValue, gValue, bValue, aValue;
        private Culoare culoare;
        public Fereastra(String titlu) {
                super(titlu);
                this.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                System.exit(0);
                }
                });
}
```

8. Folosirea culorilor

```
public void initializare() {
Panel rgbValues = new Panel();
rgbValues.setLayout(new GridLayout(4, 1));

rValue = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, 0, 256);
rValue.setBackground(Color.red);

gValue = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, 0, 256);
gValue.setBackground(Color.green);

bValue = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, 0, 256);
bValue.setBackground(Color.blue);
```

8. Folosirea culorilor

```
aValue = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, 0, 256);
aValue.setValue(255);
aValue.setBackground(Color.lightGray);
rgbValues.add(rValue);
rgbValues.add(gValue);
rgbValues.add(bValue);
rgbValues.add(aValue);
rgbValues.setSize(200, 100);
add(rgbValues, BorderLayout.CENTER);
culoare = new Culoare();
add(culoare, BorderLayout.NORTH);
```

8. Folosirea culorilor

```
        pack();
        rValue.addAdjustmentListener(this);
        gValue.addAdjustmentListener(this);
        bValue.addAdjustmentListener(this);
        aValue.addAdjustmentListener(this);
}
```

8. Folosirea culorilor

```
public void adjustmentValueChanged(AdjustmentEvent
e) {
    int r = rValue.getValue();
    int g = gValue.getValue();
    int b = bValue.getValue();
    int a = aValue.getValue();
    Color c = new Color(r, g, b, a);
    culoare.color = c;
    culoare.repaint();
    }
}
```

8. Folosirea culorilor

```
// clasa principala

public class TestColor{
        public static void main(String args[]) {
                Fereastra f = new Fereastra("Color");
                f.initializare();
                f.show();
        }
}
```

Utilizarea desenării în Java

7. Folosirea fonturilor
    1. Clasa Font
    2. Clasa FontMetrics
8. Folosirea culorilor
9. Folosirea imaginilor
    1. Incarcarea unei imagini dintr-un fisier
    2. Afisarea imaginilor
    3. Monitorizarea încarcarii imaginilor - interfata ImageObserver
    4. Crearea imaginilor în memorie - clasa MemoryImageSource
10. Tiparirea

9. Folosirea imaginilor

In **Java AWT** este posibila folosirea imaginilor create extern în format gif sau jpeg.

Orice imagine este o instanta a clasei **Image**.

Aceasta nu este o clasa de componente (nu extinde clasa **Component**) ci implementeaza obiecte care pot fi desenate pe suprafata unor componente cu metode specifice unui context grafic pentru componenta respectiva (similar modului cum se deseneaza o linie sau un cerc).

9. Folosirea imaginilor

Incarcarea unei imagini dintr-un fisier

Crearea unui obiect de tip Image se face folosind o imagine dintr-un fisier fie aflat pe masina pe care se lucreaza, fie aflat la o anumita adresa (URL) pe Internet.

Metodele pentru încarcarea unei imagini dintr-un fisier se gasesc în clasele **Applet** si **Toolkit**, având însa aceeasi denumire **getImage** si urmatoarele formate:

Applet	Toolkit
`public Image getImage(URL url)` `public Image getImage(URL url, String fisier)`	`public Image getImage(URL url)` `public Image getImage(String fisier)`

9. Folosirea imaginilor

Pentru a obtine un obiect de tip **Toolkit** se va folosi metoda getDefaultToolkit, ca în exemplu:

```
Toolkit toolkit = Toolkit.getDefaultToolkit();
Image image1 = toolkit.getImage("imageFile.gif");
Image image2 = toolkit.getImage(
new URL("http://java.sun.com/graphics/people.gif"));
```

Metoda getImage nu verifica daca fisierul sau adresa specificata reprezinta o imagine valida si nici nu încarca efectiv imaginea în memorie, aceste operatiuni fiind facute abia în momentul în care se va realiza afisarea imaginii pentru prima data.

Metoda nu face decât sa creeze un obiect de tip Image care face referinta la o anumita imagine externa.

9. Folosirea imaginilor

Afisarea imaginilor

Afisarea unei imagini într-un context grafic se realizeaza prin intermediul metodei drawImage din clasa **Graphics** si, în general, se realizeaza în metoda paint a unui obiect de tip Canvas.

Cele mai uzuale formate ale metodei sunt:

boolean drawImage(Image img, int x, int y, ImageObserver observer)

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

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

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

9. Folosirea imaginilor

unde:

- img este obiectul ce reprezinta imaginea
- x si y sunt coordonatele stânga-sus la care va fi afisata imaginea, relative la spatiul de coordonate al contextului grafic
- observer este un obiect care "observa" încaracarea imaginii si va fi informat pe masura derularii acesteia; de obicei se specifica this.
- width, heigth reprezinta înaltimea si latimea la care trebuie scalata imaginea
- bgColor reprezinta culoarea cu care vor fi colorati pixelii transparenti ai imaginii

9. Folosirea imaginilor

In exemplul urmator afisam aceeasi imagine de trei ori

```
Image img =
Toolkit.getDefaultToolkit().getImage("cars.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);
```

9. Folosirea imaginilor

Crearea imaginilor în memorie – clasa MemoryImageSource

- In cazul în care dorim sa folosim o anumita imagine creata direct din program si nu încarcata dintr-un fisier vom folosi clasa MemoryImageSource, aflata in pachetul **java.awt.image**.
- Pentru aceasta va trebui sa definim un vector de numere întregi în care vom scrie valorile întregi (RGB) ale culorilor pixelilor ce definesc imaginea noastra.

9. Folosirea imaginilor

Dimensiunea vectorului va fi înaltimea înmultita cu latimea în pixeli a imaginii.
Constructorul clasei MemoryImageSource este:

MemoryImageSource(int w, int h, intst pixeli, int off, int scan)

unde:

- w, h reprezinta dimensiunile imaginii (latimea si înaltimea)
- pixeli[] este vectorul cu culorile imaginii
- off, scan reprezinta modalitatea de construire a matricii imaginii pornind de la vectorul cu pixeli, normal aceste valori sunt off = 0, scan = w

9. Folosirea imaginilor

In exemplul urmator vom crea o imagine cu pixeli de culori aleatorii si o vom afisa pe ecran:

```
int w = 100;
int h = 100;
intst pix = new intsw * ht;
int index = 0;
for (int y = 0; y < h; y++) {
            for (int x = 0; x < w; x++) {
                        int red = (int) (Math.random() * 255);
                        int green = (int) (Math.random() * 255);
                        int blue = (int) (Math.random() * 255);
                        pixsindex++t = new Color(red, green, blue).getRGB();
            }
}
img = createImage(new MemoryImageSource(w, h, pix, 0, w));
g.drawImage(img, 0, 0, this);
//g este un context grafic
```

Referinte

- Curs practic de Java, Cristian Frasinaru – capitolul Desenarea

Întrebări?