Curs 7
Desenarea

- Conceptul de desenare
- Desenarea obiectelor - metoda paint()
- Suprafete de desenare - clasa Canvas
- Contextul grafic de desenare - clasa Graphics
  - Proprietatile unui context grafic
  - Primitive grafice
    - Desenarea textelor
    - Desenarea figurilor geometrice
- Folosirea fonturilor
  - Clasa Font
  - Clasa FontMetrics
- Folosirea culorilor
- Folosirea imaginilor
  - Incarcarea unei imagini dintr-un fisier
  - Afisarea imaginilor
  - Monitorizarea încarcarii imaginilor - interfata ImageObserver
  - Crearea imaginilor în memorie - clasa MemoryImageSource
- Tiparirea

Conceptul de desenare

Un program Java care are interfata grafica cu utilizatorul trebui 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. Desenarea componentelor se face automat si este un proces care se executa în urmatoarele situatii:

- la afisarea pentru prima data a unei componente
- ca raspuns al unei solicitari explicite a programului
- la operatii de minimizare, maximizare, redimensionare a suprafetei de afisare pe care este plasata o componenta

Metodele care controleaza procesul de desenare se gasesc în clasa `Component` si sunt prezentate în tabelul de mai jos:

`void` **`paint`**`(Graphics g)`	Deseneaza o componenta. Este o metoda supradefinita de fiecare componenta în parte pentru a furniza reprezentarea sa grafica specifica. Metoda este apelata de fiecare data când continutul componentei trebuie desenat (redesenat) - la afisarea pentru prima data a componentei, la operatii de redimensionare, etc. Nu se apeleaza explicit.
`void` **`update`**`(Graphics g)`	Actualizeaza starea grafica a unei componente. Actiunea acestei metode se realizeaza în trei pasi: • sterge componenta prin supradesenarea ei cu culoarea fundalului • stabileste culoarea (foreground) a componentei

	• apeleaza metoda paint pentru a redesena complet componenta Nu se apeleaza explicit.
`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 obiect reprezinta *contextul grafic* în care se executa desenarea componentelor (vezi "Contextul grafic de desenare - clasa Graphics").
Toate desenele care trebuie sa apara pe o suprafata de desenare se realizeaza în metoda `paint` a unei componente, în general apelata intern sau explicit cu metoda `repaint`, ori de câte ori componenta respectiva trebuie redesenata.

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.
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.

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 DEMO").

```
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 DEMO", 5, 35);
        }
}

public class TestPaint {
        public static void main(String args[]) {
                Fereastra f = new Fereastra("Test Paint");
                f.show();
        }
}
```

Observati ca la orice redimensionare a ferestrei textul "Aplicatie DEMO" 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.

Curs 7
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. 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`.
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 implcite 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.

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

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

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 . . .;
        }
```
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);
        }
        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();
        }
}
```

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.

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.

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.

Proprietatile contextului grafic

Curs 7
La orice tip de desenare parametrii legati de culoare, font, etc. sunt specificati de contextul grafic în care se face desenarea. In continuare, enumeram aceste proprietati si 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 prin :
- `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`

Primitive grafice

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

Desenarea textelor

Desenarea textelor de face cu metodele **drawString, drawBytes, drawChars** în urmatoarele formate:

```
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.

Desenarea figurilor geometrice

Enumeram în continuare figurile geometrice ce pot fi desenate în Java si metodele folosite pentru aceasta:

- 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`
- 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)`

Metodele care încep cu "fill" vor desena figuri geometrice care au interiorul colorat, adica "umplut" cu culoarea curenta a contextului de desenare.

Folosirea fonturilor

Dupa cum vazut, pentru a scrie un text pe ecran avem doua posibilitati. Prima dintre acestea este sa folosim o componenta orientata-text cum ar fi `Label`, `TextField` sau `TextArea`, iar a doua 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)**.

Cei mai importanti parametri ce caracterizeaza un font sunt:

- numele fontului: Helvetica Bold, Arial Bold Italic, etc
- familia din care face parte fontul: Helvetica, Arial, etc
- dimensiunea fontului: înaltimea sa
- stilul fontului: **îngrosat (bold)**, *înclinat (italic)*
- metrica fontului

Clasele care ofera suport pentru lucrul cu fonturi sunt **Font** si **FontMetrics**. In continuare sunt prezentate modalitatile de lucru cu aceste doua clase.

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 argumene numele fontului, dimensiunea si stilul acestuia.

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

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);
```

Folosirea unui obiect de tip Font se realizeaza uzual astfel:

```
//pentru componente etichetate
                Label label = new Label("Text Java");
                label.setFont(new Font("Arial", Font.BOLD, 12));
//in metoda paint(Graphics g)
                g.setFont(new Font("Times New Roman", Font.ITALIC, 14));
                g.drawString("Text Java", 0, 0);
```

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();
```

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();
        }

        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;
        }
}

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();
        }
}
```

Clasa FontMetrics

Curs 7

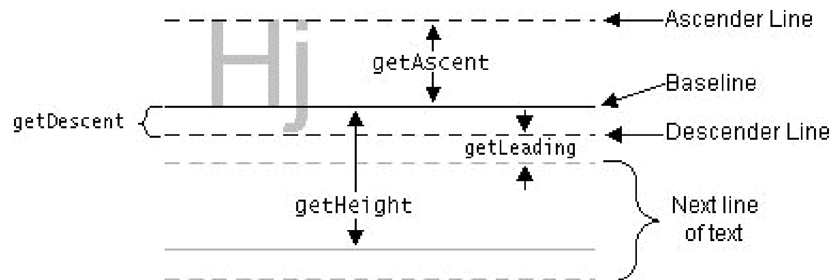
La afisarea unui sir cu metoda `drawString` trebuie sa specificam pozitia la care sa apara sirul pe ecran. In momentul în care avem de afisat mai multe siruri trebuie sa calculam pozitiile lor de afisare în functie de lungimea si înaltimea în pixeli a textului fiecarui sir. Pentru aceasta este folosita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

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

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



Reamintim ca la metoda `drawString(String s, int x, int y)` argumentele x si y semnifica coltul din **stânga-jos** al textului. Ca sa fim mai precisi, y reprezinta pozitia liniei de baza a textului care va fi scris.

Constructorul clasei `FontMetrics` creeaza un obiect ce încapsuleaza informatii despre un anumit font: `FontMetrics(Font f)`.

```
        Font f = new Font("Arial", Font.BOLD, 11);
        FontMetrics fm = new FontMetrics(f);
```

Un context grafic pune la dispozitie o metoda speciala `getFontMetrics` de creare a unui obiect de tip `FontMetrics`, pornind de la fontul curent al contextului grafic:

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

Cele mai uzuale metode ale clasei `FontMetrics` sunt cele pentru:

- aflarea înaltimii unei linii pe care vor fi scrise caractere ale unui font: **getHeight**,
- aflarea latimii totale în pixeli a unui sir de caractere specificat: **stringWidth**
- aflarea latimii unui anumit caracter din font: **charWidth**

Exemplu: afisarea unor texte pe ecran (zilele saptamânii, lunile anului si mesajul "Hello FontMetrics!") folosind clasa `FontMetrics`.

```
import java.awt.*;
class Texte extends Canvas {
        Dimension canvasSize = new Dimension(700, 300);
        private Stringst zile={"Luni", "Marti", "Miercuri", "Joi", "Vineri",
                "Sambata", "Duminica"};
        private Stringst luni = { "Ianuarie", "Februarie", "Martie", "Aprilie",
                "Mai", "Iunie", "Iulie", "August",
                "Septembrie", "Octombrie", "Noiembrie", "Decembrie"};

        public Texte() {
                setSize(canvasSize);
        }

        public void paint(Graphics g) {

                FontMetrics fm;
                int x,y;
                String enum_zile = "Zilele saptamanii:",
                        enum_luni="Lunile anului:", text;

                //alegem un font si aflam metrica sa
                g.setFont(new Font("Arial", Font.BOLD, 20));
                fm = g.getFontMetrics();
                x = 0;
                y = fm.getHeight();
                g.drawString(enum_zile, x, y);
                x += fm.stringWidth(enum_zile);

                for(int i=0; i < zile.length; i++) {
                        text = zilesit;
                        if (i < zile.length - 1) text += ", ";
                        g.drawString(text, x, y);
                        x += fm.stringWidth(text);
                }
                //schimbam fontul
                g.setFont(new Font("Times New Roman", Font.PLAIN, 14));
                fm = g.getFontMetrics();
                x = 0;
                y += fm.getHeight();
                g.drawString(enum_luni, x, y);
                x += fm.stringWidth(enum_luni);

                for(int i=0; i < luni.length; i++) {
                        text = lunisit;
                        if (i < luni.length - 1) text += ", ";
                        g.drawString(text, x, y);
                        x += fm.stringWidth(text);
                }
                //schimbam fontul curent
                g.setFont(new Font("Courier New", Font.BOLD, 60));
                fm = g.getFontMetrics();
                x = 0;
                y += fm.getHeight();
                g.drawString("Hello FontMetrics!", x, y);

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

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

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).
O culoare este reprezentata printr-o instanta a clasei **Color** sau a subclasei sale **SystemColor**. Pentru a crea o culoare avem doua posibilitati:

- sa folosim una din constantele definite într-un din cele doua clase
- sa folosim unul din constructorii clasei Color.

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. Folosirea acestor constante se face ca în exemplele de mai jos:

```
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.
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);
```

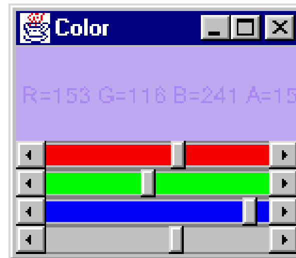
Metodele cele mai folosite ale clasei `Color` sunt:

`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)

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:




```
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; }
}

//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);
                        }
                });
        }

        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);

                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);

                pack();

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

        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();
        }
}

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

```
                f.show();
        }
}
```

Folosirea imaginilor

Aceasta este o imagine:

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).

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)`

Pentru a obtine un obiect de tip `Toolkit` se va folosi metoda `getDefaultToolkit`, ca în exemplul de mai jos:

```
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.
Dintre metodele clasei `Image` cele mai des folosite sunt cele pentru determinarea dimensiunilor unei imagini:

```
int getHeight(ImageObserver observer)
int getWidth(ImageObserver observer)
```

unde parametrul `observer` este uzual `this`. (despre interfata `ImageObserver` se va discuta ulterior)

Curs 7
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)
```

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

In exemplul urmator afisam aceeasi imagine de trei ori

```
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);
```

Metoda `drawImage` returneaza `true` daca imaginea a fost afisata în întregime si `false` în caz contrar, cu alte cuvinte metoda nu astepta ca o imagine sa fie complet afisata ci se termina imediat ce procesul de afisare a început. Sa detaliem putin acest aspect.
In cazul în care se afiseaza o imagine care se gaseste pe Internet sau imaginea afisata este de dimensiuni mari se va observa ca aceasta nu apare complet de la început ci este desenata treptat fara interventia programatorului. Acest lucru se întâmpla deoarece metoda `drawImage` nu face decât sa declanseze procesul de încarcare/afisare a imaginii, dupa care reda imediat controlul apelantului, lucru deosebit de util întrucât procesul de încarcare a unei imagini poate dura mult si nu este de dorit ca în acest interval de timp (pâna la încarcarea completa a imaginii) aplicatia sa fie blocata.
Ca urmare, la apelul metodei `drawImage` va fi desenata numai portiunea de imagine care este disponibila la un moment dat si care poate fi incompleta. De aceea trebuie sa existe un mecanism prin care componenta sa fie redesenata în momentul în care au mai sosit informatii legate de imagine. Acest mecanism este realizat prin intermediul interfetei **ImageObserver**, implementata de clasa `Component` si deci de toate componentele. Aceasta interfata specifica obiecte care au început sa utilizeze o imagine incompleta si care trebuie anuntate de noile date obtinute în legatura cu imaginea respectiva.

Monitorizarea încarcarii imaginilor - interfata ImageObserver

Interfata are o singura metoda **imageUpdate** apelata periodic de firul de executie (creat automat) care se ocupa cu încarcarea imaginii. Formatul acestei metode este:

```
boolean imageUpdate (Image img, int flags, int x, int y, int w, int h )
```

Implementarea implicita consta într-un apel la metoda `repaint` pentru dreptunghiul specificat la apel si care reprezinta zona din imagine pentru care se cunosc noi informatii. Intregul `flags` furnizeaza informatii despre starea transferului. Aceste informatii pot fi aflate prin intermediul constantelor definite de interfata. Acestea sunt :

`ABORT`	Incarcarea imaginii a fost întrerupta, înainte de completarea ei.
`ALLBITS`	Imaginea a fost încarcata complet
`ERROR`	A aparut o eroare în timpul încarcarii imaginii
`FRAMEBITS`	Totii bitii cadrului curent sunt disponibili
`HEIGHT`	Inaltimea imaginii este disponibila
`PROPERTIES`	Proprietatile imaginii sunt disponibile
`SOMEBITS`	Au fost receptionati noi pixeli ai imaginii
`WIDTH`	Latimea imaginii este disponibila

Prezenta în `flags` a unui bit de valoare 1 pe pozitia reprezentata de o constanta înseamna ca respectiva conditie este îndeplinita.

```
        //Exemple
        (flags & ALLBITS) != 0
                imaginea este completa
        (flags & ERROR | ABORT ) != 0
                a aparut o erorare sau transferul imaginii a fost întrerupt
```

Metoda `imageUpdate` poate fi redefinta pentru a personaliza afisarea imaginii. Pentru aceasta implementam clasei de tip `Canvas`, folosita pentru afisarea imaginii, metoda `imageUpdate`, care va fi apelata asincron de fiecare data când sunt disponibili noi pixeli.

```
public boolean imageUpdate(Image img, int flags, int x, int y, int w, int h) {
        //se deseneaza imaginea numai daca toti bitii sunt disponibili
        if (( flags & ALLBITS) != 0) { repaint(); }

        //daca am toti bitii nu mai sunt necesare noi update-uri
        return ( (flags & (ALLBITS | ABORT)) == 0);
}
```

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. 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

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
```

Tiparirea

Tiparirea in Java este tratata in aceeasi maniera ca si desenarea, singurul lucru diferit fiind contextul grafic in care se executa operatiile. Pachetul care ofera suport pentru tiparire este **java.awt.print**, iar clasa principala care controleaza tiparirea este **PrinterJob**. O aplicatie va apela metode ale acestei clase pentru:

- crearea unei sesiuni de tiparire (job)
- invocarea dialogului cu utilizatorul pentru specificarea unor parametri legati de tiparire
- tiparirea efectiva

Orice componenta care poate fi afisata pe ecran poate fi si tiparita. In general, orice informatii care trebuie atât afisate cât si tiparite, vor fi incapsulate într-un obiect grafic - componenta, care are o reprezentare vizuala descrisa de metoda **paint** si care va specifica si modalitatea de reprezentare a sa la imprimanta.
Un obiect care va fi tiparit trebuie sa implementeze interfata **Printable** care contine o singura metoda **print** responsabila cu descrierea modalitatii de tiparire a obiectului. In cazul când imaginea de pe ecran coincide cu imaginea de la imprimanta, codurile metodelor `paint` si `print` pot fi identice. In general, metoda `print` are urmatorul format:

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

                //descrirea imaginii obiectului ce va fi afisata la imprimanta
                //poate fi un apel la metoda paint: paint(g)

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

                return Printable.PAGE_EXISTS;
        }
}
```

Pasii care trebuie efectuati pentru tiparirea unui obiect sunt:

1. Crearea unei sesiuni de tiparire	`PrinterJob.getPrinterJob`
2. Specificarea obiectului care va fi tiparit; acesta trebuie sa implementeze interfata `Printable`	`setPrintable`
3. Optional, initierea unui dialog cu utilizatorul pentru precizarea unor parametri legati de tiparire	`printDialog`
4. Tiparirea efectiva	`print`

In exemplul urmator vom defini un obiect care are aceeasi reprezentare pe ecran cât si la imprimanta (un cerc circumscris unui patrat, însotit de un text) si vom tipari obiectul respectiv.

```
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 FlowLayout(FlowLayout.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 e) {
                                System.out.println("Exceptie la tiparire!");
                                e.printStackTrace();
```

```
                        }
                }
        }

}

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

Tiparirea textelor

O alta varianta pentru tiparirea de texte este deschiderea unui flux catre dispozitivul special reprezentat de imprimanta si scrierea informatiilor, linie cu linie, pe acest flux. In Windows, imprimanta poate fi referita prin **"lpt1"**, iar în Unix prin **"/dev/lp"**. Observati ca aceasta abordare nu este portabila, deoarece necesita tratare speciala în functie de sistemul de operare folosit.

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

class TestPrint {
        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();
        }
}
```