



Programare orientată pe obiecte

1. Clase interioare/imbricate
2. Tratarea evenimentelor în Java
3. Introducere în grafica Java



Clase interioare

- **Clasă interioară:** o definiție de clasă înăuntrul altei definiții de clasă
 - permite să *grupăm clasele* care aparțin d.p.d.v. logic una alteia și să *controlăm vizibilitatea* uneia în interiorul celeilalte
 - clasele interioare diferă de compoziție
- Pentru a crea o instanță a clasei interioare de oriunde altundeva decât din metode ne-**statice** a clasei exterioare, trebuie precizat tipul obiectului ca fiind *NumeClasaExterioara.NumeClasaInterioara*



Exemplu de clase interioare

- Tipic, o clasă exterioară are o metodă care returnează o referință la o clasă interioară

```

public class Pachet {
    class Continut {
        private int val = 10;
        public int valoare() { return val; }
    }
    class Destinatie {
        private String label;
        Destinatie(String dst) {
            label = dst;
        }
        String readLabel() { return label; }
    }
}

public Destinatie spre(String s) {
    return new Destinatie(s);
}
public Continut cont() {
    return new Continut();
}
public void trimite(String dest) {
    Continut c = cont();
    Destinatie d = spre(dest);
    System.out.println(d.readLabel());
}
public static void main(String[] args) {
    Pachet p = new Pachet();
    p.ship("Romania");
    Pachet q = new Pachet();
    // Definirea referintelor spre
    // clasele interioare:
    Pachet.Continut c = q.cont();
    Pachet.Destinatie d = q.spre("China");
}

```



Clase interioare

- Clasele interioare sunt de patru feluri:
 - Clase membre statice
 - Clase membre
 - Clase locale
 - Clase anonime
- O **clasă membră statică** este un membru static al clasei.
 - Are acces la toate metodele *statice* ale clasei exterioare care o conține

```

class Exterioara {
    public static class Interna {
        //...
        public void functie1() {}
        //alte functii
    }
}

```



Clase interioare

- O *clasă membru* este și ea definită în cadrul unei clase.
 - Spre deosebire de forma statică, clasa membru este *specifică instanței* și
 - are *acces la toate* metodele și membrii, chiar și la referința *this* a clasei care o înconjoară.

```
class Externa {
    private float variable = 0;
    public void faCeva() { //faceti ce e necesar }
    private class Interna {
        public void faCeva() { //faceti ce e necesar }
        public void functie() {
            // Pentru a apela o functie cu acelasi nume
            // din clasa care o contine
            Externa.this.faCeva();
        }
    }
}
```

OOP9 - M. Joldoș - T.U. Cluj

5



Clase interioare

- *Clasele locale* se declară într-un bloc de cod și sunt vizibile doar în acel bloc, exact ca variabilele unei metode.

```
interface Interfata {
    public String getInfo();
}
class Externa {
    Interfata current_object;
    public void setInterface(String info) {
        class Interna implements Interfata {
            private String info;
            public Interna(String inf) {info=inf;}
            public String getInfo() {return info;}
        }
        current_object = new Interna(info);
    }
}
```

OOP9 - M. Joldoș - T.U. Cluj

6



Clase interioare

- O *clasă anonimă* = clasă locală fără nume
- Clasele interioare sunt foarte utile la implementarea procedurilor *callback*.
 - Dacă se implementează o procedură callback fără a folosi o clasă interioară, atunci la implementarea interfeței `ActionListener` trebuie folosite o mulțime de `if` și `else if` pentru a determina care este obiectul pentru care a apărut evenimentul.
 - Folosirea claselor interne permite realizarea eficientă a unui bloc de cod separat pentru a gestiona funcția `actionPerformed` pentru fiecare obiect component.

OOP9 - M. Joldoș - T.U. Cluj

7



Ce este un *callback*

- *Callback* este o schemă folosită în programele conduse de evenimente, în care un program înregistrează o subrutină (numită "callback handler") pentru a trata un anumit eveniment.
- Programul nu apelează direct subrutina, ci, atunci când apare un eveniment, sistemul de execuție apelează subrutina și îi transmite de obicei argumente care descriu evenimentul.

OOP9 - M. Joldoș - T.U. Cluj

8



Clase interioare

- Clasele interioare pot fi create într-o metodă sau chiar într-un bloc arbitrar.
- Când se folosesc clasele interne:
 - La implementarea unei interfețe pentru a permite crearea și returnarea unei referințe.
 - La rezolvarea unei probleme complicate în care clasa interioară se creează pentru a ajuta la rezolvare și nu se dorește ca această să fie disponibilă public.
- Fiind membri ai clasei, clasele interioare pot fi făcute *private* sau *protected*, ceea ce nu se poate cu clasele normale (ne-interne)



Evenimente, surse și ascultători de eveniment

- Toate acțiunile utilizatorilor aparțin unui set abstract de lucruri numite *evenimente*.
- Un eveniment *descrie*, suficiente de detaliat, o anumită *acțiune* a utilizatorului.
- Sistemul de execuție Java *notifică* programul la apariția unui eveniment de interes pentru acesta.
- Programele care gestionează interacțiunea cu utilizatorul în acest fel sunt numite programe *conduse de evenimente* (*event driven*).
- Evenimentele din interfața cu utilizatorul includ apăsări de taste, deplasări ale mouse, clic-uri, ș.a.m.d.
- Un program poate indica faptul că îl interesează numai *anumite* evenimente



Evenimente, surse și ascultători de eveniment

- *Ascultătorul (listener)* de eveniment:
 - Este notificat la apariția unui eveniment
 - Aparține unei clase furnizate de către programatorul de aplicație
 - Metodele sale descriu *acțiunile* de efectuat la apariția unui eveniment
 - Un program *indică ce anume evenimente* are nevoie să primească prin instalarea *obiectelor ascultătoare de evenimente*
- *Sursa* unui eveniment:
 - Sursele de evenimente *raportează* asupra evenimentelor
 - La apariția unui eveniment, sursa evenimentului *notifică toți ascultătorii de evenimente*



Evenimente, surse și ascultători de eveniment

- Exemple: folosim componente `JButton` pentru butoane; atașăm fiecărui buton un `ActionListener`
- Interfața `ActionListener`:


```
public interface ActionListener {
    void actionPerformed(ActionEvent event);
}
```
- Trebuie să furnizăm o clasă a cărei metodă `actionPerformed` conține instrucțiunile de efectuat la clic pe un buton
- Parametrul `event` conține detalii despre eveniment, cum sunt momentul la care a apărut evenimentul
- Construim un obiect ascultător și îl atașăm butonului:


```
ActionListener listener = new ClickListener();
button.addActionListener(listener);
```



Exemplu (+BlueJ Demo)

```

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
/**
 * Un ascultător care tipărește un mesaj*/
public class ClickListener implements
    ActionListener
{
    public void
    actionPerformed(ActionEvent event)
    {
        System.out.println("M-ai apasat.");
    }
}
/*-----*/
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
/**
 * Programul demonstreza cum se instaleaza
 * un ascultator pentru o actiune.
 */
public class ButtonTester {
    private static final int
        FRAME_WIDTH = 100;
    private static final int
        FRAME_HEIGHT = 60;
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        JButton button = new
            JButton("Apasa aici!");
        frame.add(button);
        ActionListener listener = new
            ClickListener();
        button.addActionListener(listener);
        frame.setSize(FRAME_WIDTH,
            FRAME_HEIGHT);
        frame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}

```

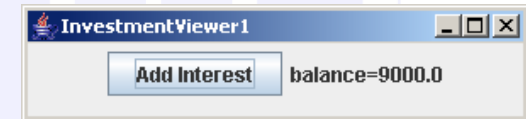
OOP9 - M. Joldoș - T.U. Cluj

13



Construirea aplicațiilor cu butoane

- Exemple: program de vizualizare a investiției; ori de câte ori se apasă pe buton (clic) se adaugă dobânda și se afișează noul sold:



- Construim un obiect al clasei `JButton`:
`JButton button = new JButton("Add Interest");`
- Avem nevoie de o componentă de interfață cu utilizatorul care afișează un mesaj:

```
JLabel label=new JLabel("balance="+account.getBalance());
```

OOP9 - M. Joldoș - T.U. Cluj

14



Construirea aplicațiilor cu butoane

- Folosim un container `JPanel` pentru a grupa mai multe componente de interfață utilizator:
`JPanel panel = new JPanel();`
`panel.add(button);`
`panel.add(label);`
`frame.add(panel);`
- Clasa ascultător adaugă dobânda și afișează noul sold:

```

class AddInterestListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        double interest = account.getBalance() *
            INTEREST_RATE / 100;
        account.deposit(interest);
        label.setText("balance=" + account.getBalance());
    }
}

```

- Adăugăm `AddInterestListener` ca clasă internă astfel încât ea are acces la variabile `final` din clasa externă (`account` și `label`). (BlueJ demo: `InvestmentViewer1`).

OOP9 - M. Joldoș - T.U. Cluj

15



Prelucrarea intrării text

- Folosim componente `JTextField` pentru a permite acceptarea de date de la utilizator

```

final int FIELD_WIDTH = 10; // In caractere
final JTextField rateField = new JTextField(FIELD_WIDTH);

```

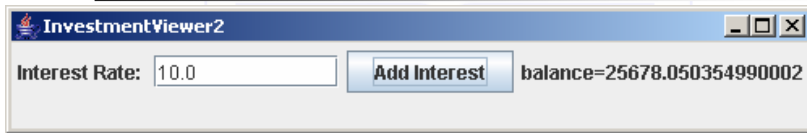
- Punem o `JLabel` lângă fiecare câmp text
`JLabel rateLabel = new JLabel("Interest Rate: ");`
- Furnizăm un buton pe care utilizatorul să-l apese pentru a indica că a terminat de introdus datele

OOP9 - M. Joldoș - T.U. Cluj

16



Prelucrarea intrării text



- Metoda `actionPerformed` a butonului citește intrarea din câmpurile text (cu `getText`)

```
class AddInterestListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        double rate=Double.parseDouble(rateField.getText());
        . . .
    }
}
```

BlueJ Demo: InvestmentViewer2



Evenimente legate de mouse

- Folosim un ascultător de mouse pentru a reacționa la evenimentele legate de mouse
- Implementăm interfața `MouseListener`:

```
public interface MouseListener {
    void mousePressed(MouseEvent event);
    // Apelata la apăsarea unui buton al mouse pe o componenta
    void mouseReleased(MouseEvent event);
    // Apelata la eliberarea unui buton al mouse pe o componenta
    void mouseClicked(MouseEvent event);
    // Apelata atunci când s-a dat clic pe o componenta
    void mouseEntered(MouseEvent event);
    // Apelata atunci când indicatorul mouse intra pe o componenta
    void mouseExited(MouseEvent event);
    // Apelata atunci când indicatorul mouse iese
    // de pe o componenta
}
```



Evenimente legate de mouse

- `mousePressed`, `mouseReleased`: apelate atunci când s-a apăsat sau eliberat un buton al mouse
- `mouseClicked`: dacă s-a apăsat și eliberat în succesiune rapidă un buton și mouse nu s-a mișcat
- `mouseEntered`, `mouseExited`: mouse a intrat sau ieșit din zona componentei
- Adăugăm un ascultător de mouse la o componentă apelând metoda `addMouseListener`:

```
public class MyMouseListener implements MouseListener
{
    // Implements five methods
}
MouseListener listener = new MyMouseListener();
component.addMouseListener(listener);
```



Evenimente legate de mouse

- Program exemplu: la clic mută componenta dreptunghi
- Apelăm metoda `repaint` atunci când modificăm formele pe care le desenează metoda `paintComponent`:


```
box.setLocation(x, y);
repaint();
```
- Ascultătorul de mouse: dacă se apasă un buton al mouse, ascultătorul mută dreptunghiul la locul unde se află indicatorul mouse
- BlueJ demo: `RectangleMover.java`



Evenimente legate de mouse

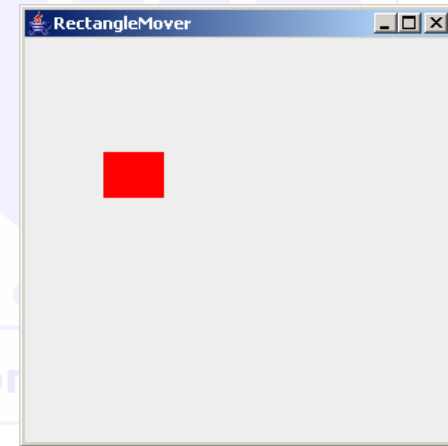
```
class MousePressListener implements MouseListener
{
    public void mousePressed(MouseEvent event) {
        int x = event.getX();
        int y = event.getY();
        component.moveTo(x, y);
    }
    // Do-nothing methods
    public void mouseReleased(MouseEvent event) {}
    public void mouseClicked(MouseEvent event) {}
    public void mouseEntered(MouseEvent event) {}
    public void mouseExited(MouseEvent event) {}
}
```

- **Toate cele cinci metode ale interfeței trebuie implementate; metodele nefolosite pot fi vide**



Evenimente legate de mouse. Exemplu

- BlueJ Demo: RectangleMover



Sisteme grafice ale Java

- SDK Java SDK conține două sisteme grafice diferite
 - Abstract Windowing Toolkit (AWT), sistemul grafic original al Java
 - Pachetul Swing, un sistem grafic mai nou și mai flexibil
- Vom trata doar grafica cu Swing



Componente și containere

- Cele două feluri principale de obiecte grafice sunt **Containerele** și **Componentele**
- **Componentă**: obiect vizual care conține text sau grafică
- **Container**: obiect grafic care poate păstra *componente* sau alte *containere*
 - Containerul principal este un cadru (**Frame**). El reprezintă o parte din ecran înconjurată de margini (*borders*) și bare de titlu (*title bars*).



Afișarea graficii Java

- Pentru a afișa grafică Java:
 1. Creăm componenta sau componentele de afișat
 2. Creăm cadrul care să păstreze componentele și plasăm componentele în cadru (frame).
 3. Creăm un obiect "ascultător" pentru a detecta și răspunde la clic-uri pe mouse și asignăm ascultătorul cadrului.
- Vom folosi acum componente de clasa `JPanel`, și containere de clasa `JFrame`



Afișarea graficii Java

Pachete necesare

Creăm "ascultătorul"

Creăm componenta

Creăm cadrul

Adăugăm ascultătorul și componenta la cadru

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TestJPanel {
    public static void main(String s[]) {
        // Create a Window Listener to handle "close" events
        MyWindowListener l = new MyWindowListener();
        // Create a blank yellow JPanel to use as canvas
        JPanel c = new JPanel();
        c.setBackground( Color.yellow );
        // Create a frame and place the canvas in the center
        // of the frame.
        JFrame f = new JFrame("Test JPanel ...");
        f.addWindowListener(l);
        f.add(c, BorderLayout.CENTER);
        f.pack();
        f.setSize(400,400);
        f.setVisible(true);
    }
}
(DisplayGraphicsEx1)
```



Ascultători

- O clasă "ascultător" ascultă evenimentele clic pe mouse sau apăsarea de taste pe o componentă sau un container și răspunde la apariția acestor evenimente
 - Vom folosi un ascultător pentru ferestre ("Window" listener) pentru a detecta apăsările pe mouse și a termina programul

Interceptăm clic în zona "Close Window" și ieșim din program dacă apare

```
import java.awt.event.*;
public class MyWindowListener extends WindowAdapter {
    // This method implements a simple listener that detects
    // the "window closing event" and stops the program.
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
}
```



Afișarea graficii pe o componentă

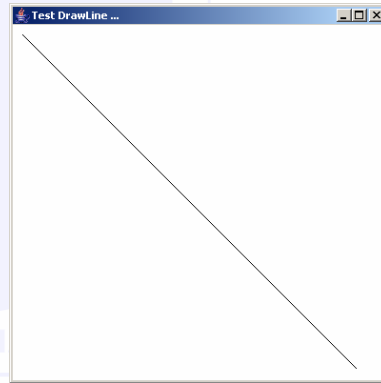
- Metoda `paintComponent` se folosește pentru a desena grafică pe o componentă.
 - Apelul său este:


```
paintComponent( Graphics g )
```
 - Obiectul grafic trebuie convertit (cast) imediat la un obiect `java.awt.Graphics2D` înainte de a putea fi folosit cu grafica Swing
 - O dată acest lucru făcut, se pot folosi toate clasele din `java.awt.geom` pentru a desena grafică pe componentă



Exemplu: desenarea unei linii

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
public class DrawLine extends JPanel {
    public void paintComponent ( Graphics g ) {
        // Converteste obiectul la Graphics2D
        Graphics2D g2 = (Graphics2D) g;
        // Seteaza culoarea de fundal
        Dimension size = getSize();
        g2.setColor( Color.white );
        g2.fill(new Rectangle2D.Double(0,0,
            size.width,size.height));
        // Draw line
        g.setColor( Color.black );
        Line2D line = new Line2D.Double (10., 10., 360.,
            360.);
        g2.draw(line);
    }
    ...
    puneți aici metoda main
    ...
```



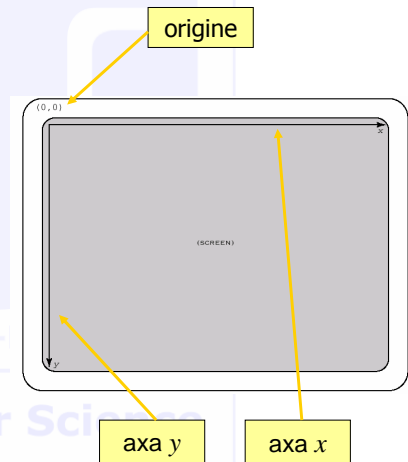
Crează un obiect Line2D

Desenează linia reprezentată de obiect



Sistemul de coordonate pentru grafică

- Java folosește pentru grafică un sistem de coordonate cu originea (0,0) în colțul din stânga sus
 - axa x este pozitivă spre dreapta
 - axa y este pozitivă în jos
- Implicit unitatea de măsură este *pixelul*
 - Sunt 72 pixeli / țol (inch)
- Unitatea de măsură poate fi schimbată



Clasele Line2D

- Există două clase concrete pentru crearea liniilor: `Line2D.Float` și `Line2D.Double`. Ele diferă prin tipul parametrilor de apel.
- Constructorii:


```
Line2D.Double( double x1, double y1,
                double x2, double y2 )
Line2D.Float( float x1, float y1,
                float x2, float y2 )
```
- Aceste clase creează linii de la (x_1, y_1) la (x_2, y_2)



Controlul culorii unui obiect

- Culoarea unui obiect grafic se controlează cu metoda `setColor` din `Graphics2D`.
- Culoarea poate fi orice obiect de clasa `java.awt.Color`, inclusiv următoarele valori predefinite:

<code>Color.black</code>	<code>Color.magenta</code>
<code>Color.blue</code>	<code>Color.orange</code>
<code>Color.cyan</code>	<code>Color.pink</code>
<code>Color.darkGray</code>	<code>Color.red</code>
<code>Color.green</code>	<code>Color.white</code>
<code>Color.lightGray</code>	<code>Color.yellow</code>



Controlul lăţimii şi stilului de linie

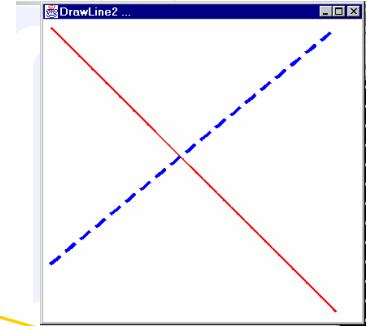
- Lăţimea şi stilul liniei este controlat cu ajutorul unui obiect `BasicStroke`
- Constructori au forma:


```
BasicStroke(float width);
BasicStroke(float width, int cap, int join,
float miterlimit,
float[] dash, float dash_phase);
```
- Pot controla lăţimea liniei, stilul capetelor, stilul de unire a liniilor şi şablonul de întrerupere (dashing)



Exemplu: Setarea culorii şi stilului

```
public void paintComponent ( Graphics g ) {
    BasicStroke bs; // Ref to BasicStroke
    Line2D line; // Ref to line
    float[] solid = {12.0f,0.0f}; // Solid line style
    float[] dashed = {12.0f,12.0f}; // Dashed line style
    // Cast the graphics object to Graph2D
    Graphics2D g2 = (Graphics2D) g;
    ...
    // Set the Color and BasicStroke
    g2.setColor(Color.red);
    bs = new BasicStroke( 2.0f, BasicStroke.CAP_SQUARE,
        BasicStroke.JOIN_MITER, 1.0f,
        solid, 0.0f );
    g2.setStroke(bs);
    // Draw line
    line = new Line2D.Double (10., 10., 360., 360.);
    g2.draw(line);
    // Set the Color and BasicStroke
    g2.setColor(Color.blue);
    bs = new BasicStroke( 4.0f, BasicStroke.CAP_SQUARE,
        BasicStroke.JOIN_MITER, 1.0f,
        dashed, 0.0f );
    g2.setStroke(bs);
    // Draw line
    line = new Line2D.Double (10., 300., 360., 10.);
    g2.draw(line);
}
```



Setează culoarea

Define apăsarea (stroke)

Setează apăsarea

Desenează linia



Clasele `Rectangle2D`

- Există două clase pentru crearea dreptunghiurilor: `Rectangle2D.Float` şi `Rectangle2D.Double`. Diferenţa este la tipul parametrilor.
- Constructori:


```
Rectangle2D.Double( double x, double y,
double w, double h )
Rectangle2D.Float( float x, float y,
float w, float h )
```
- Aceste clase creează dreptunghiuri cu originea în (x,y) , lăţimea w şi înălţimea h



Clasele `RoundRectangle2D`

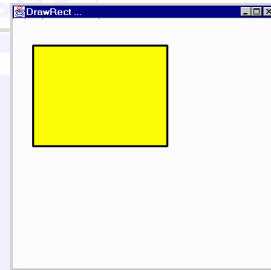
- Există două clase pentru crearea dreptunghiurilor rotunjite: `RoundRectangle2D.Float` şi `RoundRectangle2D.Double`. Diferenţa este la tipul parametrilor.
- Constructori:


```
RoundRectangle2D.Double( double x, double y,
double w, double h, double arcw, double arch )
RoundRectangle2D.Float( float x, float y,
float w, float h, float arcw, float arch )
```
- Aceste clase creează dreptunghiuri cu originea (x,y) , de lăţime w , înălţime h , lăţimea arcului $arcw$, şi înălţimea arcului $arch$

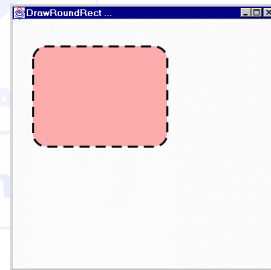


Exemple: crearea unui dreptunghi și a unui dreptunghi rotunjit

```
float[] solid = {12.0f, 0.0f}; // Solid line style
bs = new BasicStroke( 3.0f, BasicStroke.CAP_SQUARE,
    BasicStroke.JOIN_MITER, 1.0f,
    solid, 0.0f );
g2.setStroke(bs);
Rectangle2D rect = new Rectangle2D.Double
(30., 40., 200., 150.);
g2.setColor(Color.yellow);
g2.fill(rect);
g2.setColor(Color.black);
g2.draw(rect);
```



```
float[] dashed = {12.0f, 12.0f}; // Dashed line style
bs = new BasicStroke( 3.0f, BasicStroke.CAP_SQUARE,
    BasicStroke.JOIN_MITER, 1.0f,
    dashed, 0.0f );
g2.setStroke(bs);
RoundRectangle2D rect = new RoundRectangle2D.Double
(30., 40., 200., 150., 40., 40.);
g2.setColor(Color.pink);
g2.fill(rect);
g2.setColor(Color.black);
g2.draw(rect);
```



Clasele Ellipse2D

- Există două clase pentru crearea cercurilor și elipselor: **Ellipse2D.Float** și **Ellipse2D.Double**. Singura diferență între ele este tipul pentru parametri de apel.
- Constructorii:


```
Ellipse2D.Double( double x, double y,
                double w, double h);
Ellipse2D.Float( float x, float y,
                float w, float h);
```
- Aceste clase creează elipsa care încapă într-un dreptunghi cu originea (x,y) , cu lățimea w și înălțimea h
- Exemplu: crearea unei elipse


```
Ellipse2D rect = new Ellipse2D.Double
                (30., 40., 200., 150.);
g2.setColor(Color.black);
g2.fill(rect);
```



Clasele Arc2D

- Există două clase pentru crearea arcelor: **Arc2D.Float** și **Arc2D.Double**.
- Constructorii:


```
Arc2D.Double( double x, double y, double w, double h,
                double start, double extent, int type );
Arc2D.Float( float x, float y, float w, float h,
                float start, float extent, int type );
```
- Aceste clase creează arce care încap într-o zonă dreptunghiulară cu originea (x,y) , cu lățimea w și înălțimea h . Arcul începe la *start* grade și se întinde pe *extent* grade.
- Tipul de arc este **Arc2D.OPEN**, **Arc2D.CHORD**, sau **Arc2D.PIE**



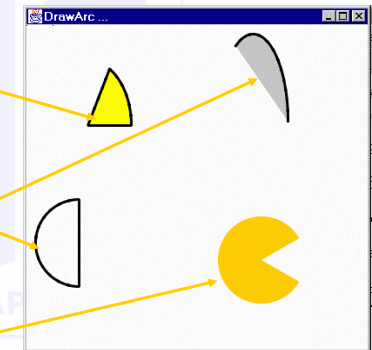
Exemplu: crearea arcelor

```
// Defineste arc1
Arc2D arc = new Arc2D.Double (20., 40., 100., 150.,
    0., 60., Arc2D.PIE);
g2.setColor(Color.yellow);
g2.fill(arc); g2.setColor(Color.black); g2.draw(arc);

// Defineste arc2
arc = new Arc2D.Double (10., 200., 100., 100.,
    90., 180., Arc2D.CHORD);
g2.setColor(Color.black); g2.draw(arc);

// Defineste arc3
arc = new Arc2D.Double (220., 10., 80., 200.,
    0., 120., Arc2D.OPEN);
g2.setColor(Color.lightGray);
g2.fill(arc); g2.setColor(Color.black); g2.draw(arc);

// Defineste arc4
arc = new Arc2D.Double (220., 220., 100., 100.,
    -30., -300., Arc2D.PIE);
g2.setColor(Color.orange); g2.fill(arc);
```



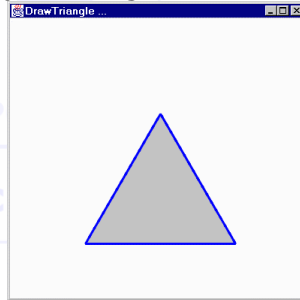
Notă: Nu puneți mai multe instrucțiuni pe o linie!



Clasa GeneralPath

- Permite construirea de forme arbitrare.
- Constructor: `GeneralPath()`;
- Câteva metode (in documentație sunt mult mai multe):
`moveTo(float x, float y); // Deplaseaza la (x,y) fara linie`
`lineTo(float x, float y); // Deseneaza o linie de aici pana la (x,y)`
`quadTo(float x1, float y1, float x2, float y2); // Deseneaza curba`
`closePath(); // Incheie forma`
- Creează o formă generală ca o serie de linii și curbe legate.
- Exemplu:

```
GeneralPath p = new GeneralPath();
p.moveTo(100.0f,300.0f);
p.lineTo(300.0f,300.0f);
p.lineTo(200.0f,127.0f);
p.closePath();
g2.setColor( Color.lightGray );
g2.fill(p);
g2.setColor( Color.blue );
g2.draw(p);
```



OOP9 - M. Joldoș - T.U. Cluj

41



Afișarea textului

- Textul se afișează folosind metoda `drawString` din `Graphics2D`. Forme:
`drawString(String s, int x, int y);`
`drawString(String s, float x, float y);`
- Aceste metode scriu `String s` pe componentă. Punctul (x, y) specifică colțul din *stânga jos* al cutiei text în cadrul componentei.
 - Observați că aceasta diferă de convenția pentru alte obiecte grafice2D, unde (x, y) este colțul din stânga sus!*
- Exemplu:

```
g2.setColor( Color.black );
g2.drawString("Acesta este un test!", 20, 40);
```

OOP9 - M. Joldoș - T.U. Cluj

42



Setarea fonturilor

- Fonturile sunt create folosind clasa `java.awt.Font`
- Constructor:
`Font(String s, int style, int size)`
 - `s` – numele fontului de utilizat.
 - `style` – stilul (`Font.PLAIN`, `Font.BOLD`, `Font.ITALIC`, sau o combinație a lor)
 - `size` dimensiunea fontului în puncte
- Se poate folosi orice font din sistem, dar există unele care sunt prezente pe orice sistem

OOP9 - M. Joldoș - T.U. Cluj

43



Nume standard pentru fonturi

- Următoarele fonturi standard sunt prezente în orice implementare Java:

Numele fontului	Descriere
<code>Serif</code>	Font serif standard pentru un anumit sistem. Exemple: Times și Times New Roman.
<code>SansSerif</code>	Font sansserif standard pentru un anumit sistem. Exemple: Helvetica și Arial.
<code>Monospaced</code>	Font monospaced (cu spațiere egală) pentru un anumit sistem. Exemple: Courier și Courier New.
<code>Dialog</code>	Font standard pentru cutii de dialog (<i>dialog boxes</i>).
<code>DialogInput</code>	Font standard pentru intrări dialog (<i>dialog inputs</i>) pe un anumit sistem.

OOP9 - M. Joldoș - T.U. Cluj

44



Exemplu: definirea fonturilor

```
Font f1 = new Font("Serif",Font.PLAIN,12);
Font f2 = new Font("SansSerif",Font.ITALIC,16);
Font f3 = new Font("Monospaced",Font.BOLD,14);
Font f4 = new Font("Serif",Font.BOLD+Font.ITALIC,20);
// Display fonts
g2.setColor( Color.black );
g2.setFont(f1);
g2.drawString("12-point plain Serif",20,40);
g2.setFont(f2);
g2.drawString("16-point italic SansSerif",20,80);
g2.setFont(f3);
g2.drawString("14-point bold Monospaced",20,120);
g2.setFont(f4);
g2.drawString("20-point bold italic Serif",20,160);
```

(DefineFontsDemo)



Obținerea informației despre fonturi

- Clasa `java.awt.FontMetrics` se poate folosi pentru a obține informații despre fonturi

- Constructor:

```
FontMetrics fm = new FontMetrics( Font f );
FontMetrics fm = g2.getFontMetrics();
```

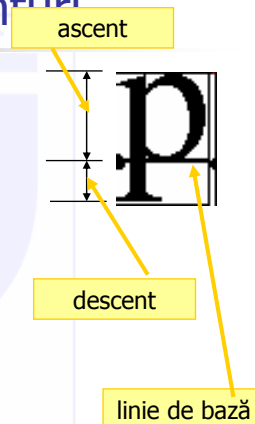
- Metode:

Numele metodei	Descriere
<code>public int getAscent()</code>	Returnează distanța în pixeli între marginea superioară și linia de bază.
<code>public int getDescent()</code>	Returnează distanța în pixeli între marginea inferioară și linia de bază.
<code>public int getHeight()</code>	Întoarce înălțimea fontului în pixeli.
<code>public int getLeading()</code>	Returnează distanța în pixeli dintre două linii de text consecutive.



Terminologie despre fonturi

- Ascent (partea de sus a literei față de linia de bază):
 - Distanța nominală în pixeli de la linia de bază la marginea inferioară a liniei de text precedente.
- Descent:
 - Distanța nominală în pixeli de la linia de bază la marginea superioară a liniei de text următoare.
- Unele glife se pot extinde dincolo de ascent/descent.



Terminologie despre fonturi

- Leading, sau spațiere interlinie: spațiul rezervat între marginea superioară a unei linii și marginea inferioară a liniei următoare.
 - Metrica height include acest spațiu.
- Height (înălțime):
 - distanța dintre linia de bază a două linii de text adiacente.
 - Suma: leading + ascent + descent.



Transformata afină

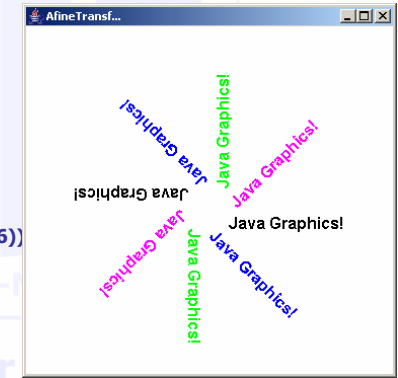
- *Transformata afină* este o transformată care deplasează, scalează, rotește și înclină o formă păstrând paralelismul liniilor.
- Constructor:
`AffineTransform at = new AffineTransform();`
- Metode (toate sunt public void):

Numele metodei	Descriere
<code>rotate(double theta)</code>	Rotește datele cu theta radiani. Unghiurile pozitive corespund rotației <i>în sens orar</i> .
<code>rotate(double theta, double x, double y)</code>	Rotește datele cu theta radiani în jurul punctului (x, y). Unghiurile pozitive corespund rotației <i>în sens orar</i> .
<code>scale(double sx, double sy)</code>	Scalează (multiplică) axele x și y prin cantitățile specificate.
<code>void shear(double shx, double shy)</code>	Retează axele x și y prin cantitățile specificate..
<code>translate(double tx, double ty)</code>	Concatenează transformata cu o translație



Exemplu: folosirea transformărilor afină pentru a roti textul

```
public void paintComponent ( Graphics g )
{
    super.paintComponent(g);
    // Tipul obiectului convertit la Graphics2D
    Graphics2D g2 = (Graphics2D) g;
    // Creaza transformata afina
    AffineTransform at = new AffineTransform();
    Color colorArray[] = new Color[] {
        Color.blue, Color.cyan, Color.magenta,
        Color.black, Color.blue, Color.cyan,
        Color.magenta, Color.black };
    g2.setFont(new Font("SansSerif",Font.BOLD,16))
    for ( int i = 0; i < 8; i++)
    {
        at.rotate(Math.PI/4, 180, 180);
        g2.setTransform(at);
        g2.setColor(colorArray[i]);
        g2.drawString("Java Graphics!", 200, 200);
    }
    super.setBackground( Color.white );
}
```



(AfineTransfDemo)



Modul XOR (sau-exclusiv)

- În mod normal, la suprapunerea a două obiecte, cel de dedesubt este ascuns de cel de deasupra.
- Metoda `setXORMode` din `Graphics2D` modifică acest comportament – regiunea de suprapunere apare cu culoare diferită.
- Apelul metodei:
`g2.setXORMode (Color c);`
unde `c` este culoarea pentru regiunea de suprapunere *dacă cele două obiecte au aceeași culoare*. În caz contrar, `c` este ignorat.

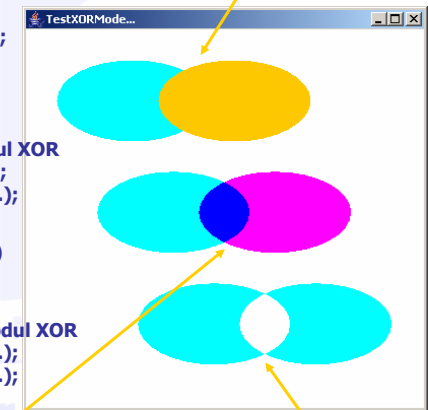


Exemplu: modul XOR

```
// Doua elipse desenate in modul normal
ell1 = new Ellipse2D.Double (30., 30., 150., 80.);
ell2 = new Ellipse2D.Double (130., 30., 150., 80.);
g2.setColor(Color.cyan);
g2.fill(ell1);
g2.setColor(Color.orange);
g2.fill(ell2);

// Doua elipse de culori diferite desenate in modul XOR
ell1 = new Ellipse2D.Double (70., 140., 150., 80.);
ell2 = new Ellipse2D.Double (170., 140., 150., 80.);
g2.setXORMode(Color.white);
g2.setColor(Color.cyan);
g2.fill(ell1);
g2.setColor(Color.magenta);
g2.fill(ell2);

// Doua elipse de aceeasi culoare desenate in modul XOR
ell1 = new Ellipse2D.Double (110., 250., 150., 80.);
ell2 = new Ellipse2D.Double (210., 250., 150., 80.);
g2.setXORMode(Color.white);
g2.setColor(Color.cyan);
g2.fill(ell1);
g2.setColor(Color.cyan);
g2.fill(ell2);
```



Normal mode: overlapping

XOR Mode w/different colors

XOR Mode w/same colors



Rezumat

- Clase interne
 - feluri
 - utilizare
- Evenimente,
 - surse ale evenimentelor
 - ascultători pentru evenimente
- Aplicații cu butoane
- Prelucrarea intrării text
- Evenimente de la mouse
- Grafica:
 - Componente, containere
 - Grafica pe o componentă
 - Sistemul de coordonate grafic
 - Desenarea:
 - liniilor drepte, a dreptunghiurilor, arcelor, elipselor, liniilor compuse
 - textului
 - Controlul culorii și stilului
 - Fonturi: setare, obținerea info despre
 - Transformata afină
 - Modul XOR