Programare orientată pe obiecte

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

Clase interioare

- *Clasă interioară*: o definiţie de clasă înlăuntrul altei definiţii de clasă
  - permite să *grupăm clasele* care aparţin d.p.d.v. logic una alteia şi să *controlăm vizibilitatea* uneia în interiorul celeilalte
  - clasele interioare diferă de compoziţie
- Pentru a crea o instanţă 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 referinţă 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

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

- O *clasă membră statică* este un membru static al clasei.
  - Are acces la toate metodele *statice* ale clasei exterioare care o conţine

Clase interioare

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

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


Clase interioare

- *Clasele locale* se declară într-un bloc de cod şi 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);
  }
}
```

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 interfeţei `ActionListener` trebuie folosite o mulţime de `if` şi `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 funcţia `actionPerformed` pentru fiecare obiect component.

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 execuţie apelează subrutina şi îi transmite de obicei argumente care descriu evenimentul.

Clase interioare

- Clasele interioare pot fi create într-o metodă sau chiar într-un bloc arbitrar.
- Când se folosesc clasele interne:
  - La implementarea unei interfeţe pentru a permite crearea şi returnarea unei referinţe.
  - La rezolvarea unei probleme complicate în care clasa interioară se creează pentru a ajuta la rezolvare şi nu se doreşte 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 şi ascultători de eveniment

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

Evenimente, surse şi ascultători de eveniment

- *Ascultătorul (listener)* de eveniment:
  - Este notificat la apariţia unui eveniment
  - Aparţine unei clase furnizate de către programatorul de aplicaţie
  - Metodele sale descriu *acţiunile* de efectuat la apariţia 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 apariţia unui eveniment, sursa evenimentului *notifică toţi ascultătorii de evenimente*

Evenimente, surse şi ascultători de eveniment

- Exemple: folosim componente `JButton` pentru butoane; ataşăm fiecărui buton un `ActionListener`
- Interfaţa `ActionListener`:

```
public interface ActionListener {
   void actionPerformed(ActionEvent event);
}
```

- Trebuie să furnizăm o clasă a cărei metodă `actionPerformed` conţine instrucţiunile de efectuat la clic pe un buton
- Parametrul `event` conţine detalii despre eveniment, cum sunt momentul la care a apărut evenimentul
- Construim un obiect ascultător şi îl ataşăm butonului:

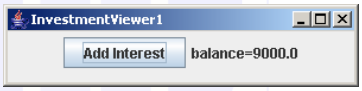
```
ActionListener listener = new ClickListener();
button.addActionListener(listener);
```

Exemplu (+BlueJ Demo)

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
/**
   Un ascultător care tipăreşte 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);
   }
}
```


Construirea aplicaţiilor cu butoane

- Exemple: program de vizualizare a investiţiei; ori de câte ori se apasă pe buton (clic) se adaugă dobânda şi se afişează noul sold:



- Construim un obiect al clasei **`JButton`**:

```
JButton button = new JButton("Add Interest");
```

- Avem nevoie de o componentă de interfaţă cu utilizatorul care afişează un mesaj:

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


Construirea aplicaţiilor cu butoane

- Folosim un container **`JPanel`** pentru a grupa mai multe componente de interfaţa utilizator:

```
JPanel panel = new JPanel();
panel.add(button);
panel.add(label);
frame.add(panel);
```

- Clasa ascultător adaugă dobânda şi afişează 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());
   }
}
```

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

Prelucrarea intrării text

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

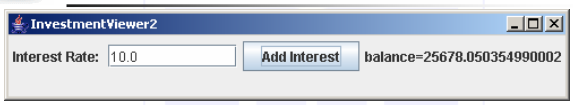
```
final int FIELD_WIDTH = 10; // In caractere
final JTextField rateField = new JTextField(FIELD_WIDTH);
```

- 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

Prelucrarea intrării text



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

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

BlueJDemo: InvestmentViewer2

Evenimente legate de mouse

- Folosim un ascultător de mouse pentru a reacţiona la evenimentele legate de mouse
- Implementăm interfaţa **`MouseListener`**:

```
public interface MouseListener {
   void mousePressed(MouseEvent event);
   // Apelata la apasarea 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 şi eliberat în succesiune rapidă un buton şi mouse nu s-a mişcat
- **mouseEntered, mouseExited**: mouse a intrat sau ieşit 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

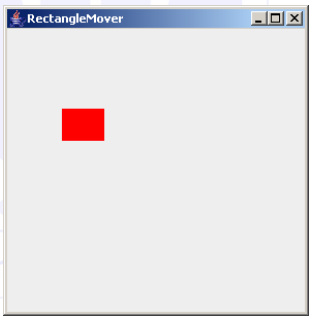
Evenimente legate de mouse

```
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

- BlueJDemo: RectangleMover



Sisteme grafice ale Java

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

Componente şi containere

- Cele două feluri principale de obiecte grafice sunt **Containerele** şi **Componentele**
- **Componentă**: obiect vizual care conţine 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*) şi bare de titlu (*title bars*).

Afişarea graficii Java

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

Afişarea graficii Java

Pachete necesare

Creăm "ascultătorul"

Creăm componenta

Creăm cadrul

Adăugăm ascultătorul şi 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 şi răspunde la apariţia acestor evenimente
  - Vom folosi un ascultător pentru ferestre ("Window" listener) pentru a detecta apăsările pe mouse şi a termina programul

Interceptăm clic în zona "Close Window" şi ieşim 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);
  };
}
```

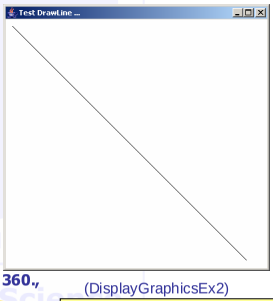

Afişarea graficii pe o componentă

- Metoda `paintComponent` se foloseşte 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);
  }
...
  puneţi aici metoda main
...
```



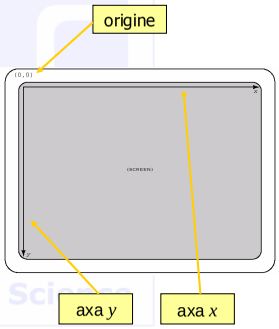
(DisplayGraphicsEx2)

Creează un obiect Line2D

Desenează linia reprezentată de obiect

Sistemul de coordonate pentru grafică

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



Clasele `Line2D`

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

```
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:

`Color.black`	`Color.magenta`
`Color.blue`	`Color.orange`
`Color.cyan`	`Color.pink`
`Color.darkGray`	`Color.red`
`Color.green`	`Color.white`
`Color.lightGray`	`Color.yellow`

Controlul lăţimii şi stilului de linie

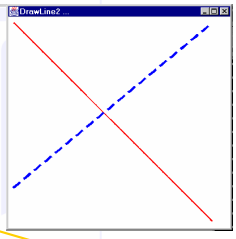
- Lăţimea şi stilul liniei este controlat cu ajutorul unui obiect `BasicStroke`
- Constructori au forma:

```
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);
}
```



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`

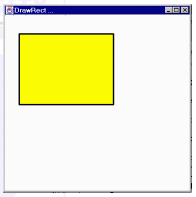
- Există două clase pentru crearea dreptunghiurilor rotunjite: `RoundRectangle2D.Float` şi `RoundRectangle2D.Double`. Diferenţa este la tipul parametrilor.
- Constructori:

```
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 )
```

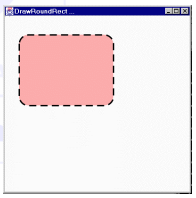
- Aceste clase creează dreptunghiuri cu originea (x,y), de lăţime w, înălţime h, lăţimea arcului *arcw*, şi înălţimea arcului *arch*

Exemple: crearea unui dreptunghi şi 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 şi elipselor: `Ellipse2D.Float` şi `Ellipse2D.Double`. Singura diferenţă între ele este tipul pentru parametri de apel.
- Constructori:

```
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 încape într-un dreptunghi cu originea (*x,y*), cu lăţimea *w* şi înălţimea *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` şi `Arc2D.Double`.
- Constructori:

```
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 lăţimea *w* şi înălţimea *h*. Arcul începe la *start* grade şi se întinde pe *extent* grade.
  - Tipul de arc este `Arc2D.OPEN`, `Arc2D.CHORD`, sau `Arc2D.PIE`

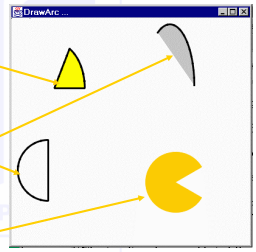
Exemplu: crearea arcelor

```
// 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 puneţi mai multe instrucţiuni pe o linie!

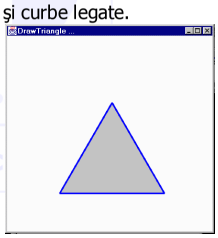
Clasa `GeneralPath`

- Permite construirea de forme arbitrare.
- Constructor: `GeneralPath();`
- Câteva metode (in documentaţie 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();                // Inchide forma
```

- Creează o formă generală ca o serie de linii şi 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);
```



Afişarea textului

- Textul se afişează folosind metoda `drawString` din `Graphics2D`. Forme:

```
drawString(String s, int x, int y);
drawString(String s, float x, float y);
```

- Acesta metode scriu `String s` pe componentă. Punctul (*x, y*) specifică colţul din *stânga jos* al cutiei text în cadrul componentei.
  - *Observaţi că aceasta diferă de convenţia pentru alte obiecte grafice2D, unde (x, y) este colţul din stânga sus!*
- Exemplu:

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

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 combinaţie a lor)
  - `size` dimensiunea fontului în puncte
- Se poate folosi orice font din sistem, dar există unele care sunt prezente pe orice sistem

Nume standard pentru fonturi

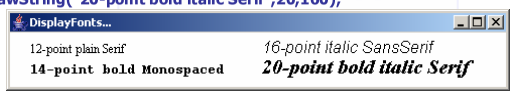
- Următoarele fonturi standard sunt prezente în orice implementare Java:

Numele fontului	Descriere
`Serif`	Font serif standard pentru un anumit sistem. Exemple: Times şi Times New Roman.
`SansSerif`	Font sansserif standard pentru un anumit sistem. Exemple: Helvetica şi Arial.
`Monospaced`	Font monospaced (cu spatiere egală) pentru un anumit sistem. Exemple: Courier şi Courier New.
`Dialog`	Font standard pentru cutii de dialog (*dialog boxes*).
`DialogInput`	Font standard pentru intrări dialog (*dialog inputs*) pe un anumit sistem.

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)



Obţinerea informaţiei despre fonturi

- Clasa `java.awt.FontMetrics` se poate folosi pentru a obţine informaţii despre fonturi
- Constructor:

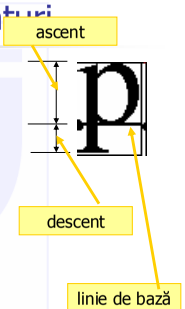
```
FontMetrics fm = new FontMetrics( Font f );
FontMetrics fm = g2.getFontMetrics();
```

- Metode:

Numele metodei	Descriere
`public int getAscent()`	Returnează distanţa în pixeli între marginea superioară şi linia de bază.
`public int getDescent()`	Returnează distanţa în pixeli între marginea inferioară şi linia de bază.
`public int getHeight()`	Întoarce înălţimea fontului în pixeli.
`public int getLeading()`	Returnează distanţa în pixeli dintre două linii de text consecutive.

Terminologie despre fonturi

- Ascent (partea de sus a literei faţă de linia de bază):
  - Distanţa nominală în pixeli de la linia de bază la marginea inferioară a liniei de text precedente.
- Descent:
  - Distanţa 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 spaţiere interlinie: spaţiul rezervat între marginea superioară a unei linii şi marginea inferioară a liniei următoare.
  - Metrica height include acest spaţiu.
- Height (înălţime):
  - distanţa dintre linia de bază a două linii de text adiacente.
  - Suma: leading + ascent + descent.

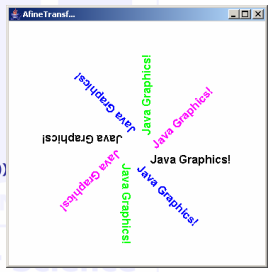
Transformata afină

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

Numele metodei	Descriere
`rotate(double theta)`	Roteşte datele cu theta radiani. Unghiurile pozitive corespund rotaţiei *în sens orar*.
`rotate(double theta, double x, double y)`	Roteşte datele cu theta radiani în jurul punctului *(x, y)*. Unghiurile pozitive corespund rotaţiei *în sens orar*.
`scale(double sx, double sy)`	Scalează (multiplică) axele *x* şi *y* prin cantităţile specificate.
`void shear(double shx, double shy)`	Retează axele *x* şi *y* prin cantităţile specificate..
`translate(double tx, double ty)`	Concateneaza trasformata cu o translaţie

Exemplu: folosirea transformătelor afine 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)

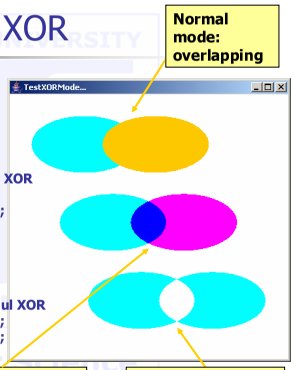
Modul XOR (sau-exclusiv)

- 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 aceeaşi 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);
```

(TestXORMode)



Rezumat

- Clase interne
  - feluri
  - utilizare
- Evenimente,
  - surse ale evenimentelor
  - ascultători pentru evenimente
- Aplicaţii 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 şi stilului
  - Fonturi: setare, obţinerea info despre
  - Transformata afină
  - Modul XOR