Java2D - Atributele grafice

Java 2D ofera capabilitati sporite pentru grafica bidimensionala si pentru lucrul cu texte si imagini prin extensii ale AWT. Acest pachet cuprinzator de redare suporta lucrul cu linii, texte si imagini intr-un cadru flexibil ce permite dezvoltarea de interfete utilizator complexe, de programe de desenare si editoare de imagini sofisticate.

Interfata de programare (API) Java 2D furnizeaza:

-          un model de redare uniform pentru dispozitive ecran si imprimante

-          o colectie bogata de primitive geometrice, cum ar curbe, dreptunghiuri si elipse precum si un mecanism pentru redarea oricaror forme geometrice

-          mecanisme pentru identificarea primitivelor grafice

-          un model de combinare a culorilor pixelilor la scrierea in memoria raster

-          suport sporit pentru culori

-          suport pentru tiparirea documentelor complexe

Java 2D extinde clasa Graphics cu clasa Graphics2D.Contextul grafic Graphics2D imbogateste contextul Graphics cu noi atributesi permite accesul la operatiile avansate de redare ale API-ului Java 2D.

Deoarece metoda care efectueaza afisarea pe suprafata unei componente primeste ca parametru un obiect Graphics, pentru a folosi caracteristicile API-ului Java 2D trebuie facuta conversia („cast”) de la obiectul Graphics trimis ca parametru la un obiect Graphics2D.

public void Paint (Graphics g)

3.1. Atributele grafice Java2D

Tabelul urmator sintetizeaza noile atribute grafice, care alcatuiesc un context grafic Graphics2D.

|  |  |  |
| --- | --- | --- |
| Atribut | Tip | Descriere |
| Culoarea | Color | Atribut mostenit de la clasa Graphics. Reprezinta culoarea folosita la afisarea liniilor si a contururilor. Poate fi definita folosind metoda (mostenita) Setcolor(). |
| Font-ul | Font | Atribut mostenit de la clasa Graphics. Sunt realizate toate fonturile sistem. |
| Regiunea de decupare | Shape | Atribut mostenit de la clasa Graphics. Regiunea de decupare poate avea o forma arbitrara. Este definita cu ajutorul functiei clip().   |
| Culoarea de fond | Color | Atribut mostenit de la clasa **Graphics**. Folosit numai de metoda **clearRect()**. Poate fi setat/interogat folosind setBackground() / getBackground().  |
| Stilul liniei   | Stroke | Specifica modul de desenare a liniilor si contururilor: latimea, sablonul, si altele. Poate fi specificat prin setStroke(). |
| Stilul interiorului | Paint | Specifica modul de generare a pixelilor interiori unei suprafete. Se poate specifica prin setPaint(). |
| Regula de combinare a culorilor | Composite | Specifica modul de combinare a culorii care trebuie inscrisa intr-un pixel cu culoarea curenta a pixelului. Tipul Composite este implementat de clasa AlphaComposite. |
| Transformarea | Java.awt.geom.AffineTransform | Defineste transformarea care se aplica coordonatelor transmise functiilor de afisare. Se defineste cu setTransform(), translate(), scale(), rotate(), shear(). |
| Calitatea redarii  | RenderingHints | Specifica preferinta privind modul de generare a imaginilor. De exemplu, se poate cere efectuarea operatiei de antialiasing la generarea primitivelor grafice. Se defineste cu setRenderingHints(), setRenderingHint() sau addRenderingHints(). |

Pentru a seta valoarea unui atribut al unui context Graphics2D, se folosesc metode de tip „set”: setStroke, setPaint, setComposite, etc. Aceste metode primesc ca parametru un obiect de tipul corespunzator atributului.Deexemplu, pentru a modifica atributul Stilul interiorului la o culoare gradient albastru-verde mai intai se construieste un obiect GradientPaint si apoi se apeleaza metoda setPaint.

GradientPaint gp = new GradientPaint(0f,0f,blue,0f,30f,green);

g2.setPaint(gp);

Intr-un context Graphics2D se pastreaza referinte la obiectele care descriu atributele sale. Atunci cand se modifica un obiect specificat ca atribut intr-un context Graphics2D este necesar sa se apeleze metoda „set” corespunzatoare atributului.

3.1.1. Fonturi

Formele utilizate de un font pentru reprezentarea caracterelor se numesc „*gliph-uri*”. Un caracter sau o combinatie de caractere pot fi reprezentate prin intermediul unuia sau a mai multor gliph-uri. De exemplu, á ar putea fi reprezentat cu ajutorul a doua gliph-uri, in timp ce  *fi* ar putea fi reprezentat de un singur gliph.

Un font poate fi imaginat ca o colectie de gliph-uri. Un font poate avea mai multe aspecte („fete”), cum ar fi normal, ingrosat, oblic, gotic. Toate „fetele” unui font au aceleasi caracteristici tipografice si pot fi recunoscute drept membre ale aceleiasi familii. Cu alte cuvinte, o colectie de gliph-uri cu un anumit stil formeza o „fata” a unui font; o colectie de „fete” de fonturi alcatuiesc o familie de fonturi iar o colectie de familii de fonturi formeaza setul de fonturi disponibile intr-un sistem.

In JDK versiunea 1.1 fonturile erau descrise de nume logice care erau mapate pe „fete” de fonturi diferite, in functie de fonturile disponibile pe o anumita platforma. Din motive de compatibilitate, Graphics2D suporta si specificarea fonturilor prin nume logice. Pentru a obtine o lista cu numele logice ale fonturilor se poata apela metoda java.awt.Toolkit.getFontList.

Pentru crearea unui obiect de tip Font se foloseste de obicei constructorul:

Font(String name, int style, int size)

Numele dat ca parametru este fie un nume logic (Dialog, DialogInput, Monospaced, Serif, SansSerif**,** sau Symbol) fie un nume de familie de fonturi. Daca name este null, numele noului font va fi setat la Default.

Stilul poate avea una dintre valorile urmatoare, specificate cu ajutorul unor „constante” declarate in clasa  Font:

         Font.PLAIN

         Font.ITALIC

         Font.BOLD

         Font.ITALIC+Font.BOLD

Dimensiunea unui Font se specifica in unitati ale spatiului de coordonate utilizator.

Exemplul urmator construieste un font apartinand familiei „Times New Roman”, cu stilul normal si dimensiunea 10.

Font font = new Font('Times New Roman', Font.PLAIN, 10);

Java2D permite utilizarea oricarui font memorat pe calculatorul pe care se executa aplicatia si suportat de catre implementarea de java utilizata. Pentru a determina fonturile disponibile se poate apela metoda:

String[]GraphicsEnvironment.getAvailableFontFamilyNames()

Acesta metoda intoarce un array de siruri de caractere care contin numele de familie ale fonturilor disponibile. Oricare dintre aceste nume de familie poate fi folosit pentru a crea un obiect Font nou:

GraphicsEnvironment gEnv =

    GraphicsEnvironment.getLocalGraphicsEnvironment();

String availableFontsNames[]

                         = gEnv.getAvailableFontFamilyNames();

Font font = new Font (availableFontsNames[0], Font.PLAIN, 10);

O alta metoda de a obtine o enumerare a fonturilor diponibile consta in folosirea metodei

Font[] GraphicsEnvironment.getAllFonts(),

care intoarce un array de obiecte de tip font, continand cate o instanta a fiecarui font disponibil in sistem, avand dimensiunea caracterelor de 1.

Pentru a obtine, plecand de la un obiect Font existent, un nou font cu stil sau dimensiuni diferite, se poate utiliza una dintre metodele deriveFont ale clasei Font.

Pentru controlul fontului folosit la redarea de text trebuie stabilit atributul font intr-un  context, inainte de redare, cu ajutorul metodei setFont(Font)a clasei Graphics2D. In urmatorul exemplu, se seteaza atributul font in contextul grafic, apoi se afiseaza un sir de caractere in centrul componentei folosind fontul specificat.

g2.setFont(font);

String string = 'font demo';

FontMetrics fm = g2.getFontMetrics();

int width = fm.stringWidth(string),

    height = fm.getHeight();

g2.drawString(string,                                                                                                                 getWidth()/2-width/2,getHeight()/2-height/2);

Se observa utilizarea unui obiect FontMetrics pentru a obtine informatii legate de dimensiunea ocupata de un sir de caractere cu fontul curent dintr-un context grafic.

Appletul prezentat in continuare permite schimbarea numelui, a stilului si a dimensiunii fontului folosit la afisarea textului.

import java.awt.\*;

import java.applet.Applet;

import java.awt.font.\*;

import java.awt.event.\*;

/\*

 Afiseaza un sir de caractere cu fontul selectat de

 utilizator,dintre toate fonturile disponibile

\*/

**public class FontSelection extends Applet implements ItemListener**

  }

  int[] sizes        = new int[] ;

  String[] styles    = new String[]                                                                                       ;

  int[] stylesValues = new int[]

                   ;

  FontComponent  fontComponent;

  Choice fontsChoice, sizesChoice, stylesChoice;

**public void init()**

**public void itemStateChanged(ItemEvent e)**

**void updateFont ()**

}

3.1.2. Regiunea de decupare

In Java 2D orice obiect de tip Shape poate fi utilizat pentru a specifica regiunea de decupare. In acest scop se poate folosi metoda setClip(Shape). De asemenea, se poate utiliza metoda clip(Shape), care va seta regiunea de decupare curenta ca fiind intersectia  dintre aceasta si obiectul Shape transmis ca parametru. Inainte de calculul intersectiei, obiectul Shape este transformat folosind atributul Transformare al contextului grafic.

3.1.3. Stilul liniei

Atributul Stilul liniei (stroke) defineste aspectul formelor redate prin intermediul metodei draw(). Caracteristicile unui stil de linie sunt incapsulate intr-un obiect BasicStroke. Acestea sunt: latimea liniei, stilul de imbinare a segmentelor unei polilinii, stilul de realizare a capetelor terminale („end-cap”) si stilul de linie punctata („dash style”).

Pentru stabilirea stilului de linie intr-un context grafic trebuie creat un obiect BasicStroke care este apoi transmis metodei setStroke() .

Latimea liniei reprezinta grosimea liniei masurata perpendicular pe traiectoria sa. Este specificata printr-o valoare float in coordonate utilizator. Atunci cand se foloseste transformarea implicita, latimea liniei este masurata in pixeli. Valoarea implicita este 1.0.

Stilul de imbinare se refera la modul de desenare a imbinarii dintre doua segmente de linie. BasicStroke accepta trei modalitati de imbinare:

JOIN\_BEVEL    JOIN\_MITER      JOIN\_ROUND

Stilul „end-cap” se refera la modul de realizare a capetelor terminale ale segmentelor de linie. BasicStroke accepta trei stiluri „end-cap”:

CAP\_BUTT    CAP\_ROUND    CAP\_SQUARE

Stilul de linie punctata („dash style”) descrie sablonul de portiuni opace si transparente care se aplica de-a lungul liniei atunci cand este generata in spatiul discret. El este definit prin parametrii „dash array” (un vector care specifica lungimile portiunilor succesive din linie, in coordonate utilizator) si „dash phase” (un deplasament in sablonul „dash”, de asemenea in coordonatelor utilizator). Elementul 0 din „dash array” reprezinta prima portiune opaca  din linie, elementul 1 primul spatiu, s.a.m.d. De exemplu, pentru a specifica o linie intrerupta in care atat segmentele de linie opaca cat si cele transparente (spatiile dintre segmente) sunt de 20 de unitati se poate folosi vectorul

new float[]

O linie intrerupta cu segmentele opace inegale poate fi specificata astfel:

new float[] ;

Parametrul „faza” specifica deplasamentul in vectorul sablon de unde se doreste inceperea aplicarii sablonului.

Valorile implicite ale atributelor care definesc stilul de linie sunt: linie solida (continua) de latime 1.0, CAP\_SQUARE, JOIN\_MITER, „miter limit”=10.0.

Clasa **BasicStroke** are mai multi constructori care permit specificarea numai a unora dintre atribute, pentru restul considerandu-se valorile implicite.

Exemple:

/\*Se afiseaza un dreptunghi cu linie de latime 5 unitati,folosind valorile implicite ale celorlalte atribute \*/

Graphics2D g;

Shape dreptunghi=newRectangle2D.Float(50,50,200,100);

g.setStroke(new BasicStroke(5.0f));

g.draw(dreptunghi);

g.setStroke(new BasicStroke(5.0f, //latimea liniei

           BasicStroke.CAP\_ROUND, //stilul „end-cap”

           BasicStroke.JOIN\_ROUND)); // stilul „join”

Stroke s = new BasicStroke(7.0f, // latimea

                        BasicStroke.CAP\_SQUARE, // stilul „end-cap”

                        BasicStroke.JOIN\_MITER, //  stilul „join”

                        10.0f, //”miter limit”

                        new float[] , sablonul

                        0.0f); //deplasamentul in sablon

1.1.4. Stilul interiorului suprafetelor

Acest atribut defineste sablonul folosit la generarea interiorului suprafetelor. Sablonul este o instanta a uneia dintre clasele care implementeaza interfata Paint. Pentru stabilirea stilului de interior intr-un context Graphics2D se apeleaza metoda setPaint().

Interfata Paint este implementata de clasele Color, GradientPaint, si TexturePaint. GradientPaint si TexturePaint sunt noi in JDK 1.2.

Stilul de interior uzual este cel compact (cu toti pixelii de aceeasi culoare). Pentru stabilirea acestui stil intr-un context grafic se apeleaza setPaint() cu un obiect al clasei Color.

Pentru a crea un GradientPaint trebuie specificate o pozitie de inceput, P1, si o culoare, C1, precum si pozitie de sfarsit, P2, si o culoare, C2. Culoarea gradient variaza liniar intre cele doua pozitii (figura 3.2). Pozitiile se specifica in coordonate utilizator. Punctele aflate pe dreapta ce trece prin P1 si P2, dar in afara segmentului P1-P2 pot fi colorate in 2 moduri:

- daca gradientul este ciclic, atunci culorile punctelor de pe dreapta P1-P2 se modifica neted intre C1 si C2, apoi inapoi catre C1 si asa mai departe;

- daca gradientul este aciclic, culoarea ramane fixata la C1 pentru punctele aflate in fata lui P1 si la C2 pentru punctele aflate dupa P2.

Exemplu:

Graphics2D g;

Paint p = new GradientPaint(P1, Color.white, P2, Color.blue, false);//aciclic

Paint p1 = new GradientPaint(P1, Color.white, P2, Color.blue, true);//ciclic

                        

Figura 3.2 Sablon GradientPaint.                                           Figura 3.3. Sablonul TexturePaint

Un sablon de tip TexturePaint este o textura definita printr-un obiect BufferedImage. Pentru a crea un sablon TexturePaint se specifica imaginea care contine sablonul (obiectul BufferedImage)si un dreptunghi folosit pentru a ancora si duplica sablonul. Dreptunghiul este specificat in coordonate utilizator (un obiect java.awt.geom.Rectangle2D). La aplicarea sablonului, imaginea sablon este mapata pe suprafata dreptunghiului, care este multiplicat in toate directiile, in spatiul utilizator, pana la marginile suprafetei pe care se aplica sablonul (figura 3.3.). In mod simplificat se poate specifica un dreptunghi cu coltul stanga-sus in (0,0), avand latimea si inaltimea imaginii textura.

3.1.5. Transformarea coordonatelor

Atributul Transformare dintr-un context Graphics2D specifica transformarea pe care metodele ce definesc operatii grafice o aplica asupra coordonatelor inainte de afisare. De aceea, se poate vorbi de existenta a doua sisteme de coordonate: *sistemul coordonatelor utilizator*-in care sunt specificate coordonatele transmise functiilor de afisare, si *sistemul coordonatelor dispozitiv*, propriu dispozitivului de afisare. Sistemul de coordonate dispozitiv are originea in coltul stanga-sus al suprafetei de afisare, axa OX orientata spre dreapta si axa OY in jos. Sistemul coordonatelor utilizator este un sistem cartezian 2D oarecare, in care coordonatele pot fi numere reale. Transformarea implicita este transformarea identica, deci in acest caz sistemul coordonatelor utilizator coincide cu sistemul coordonatelor dispozitiv.

Atributul Transformare reprezinta o transformare afina, care conserva liniile drepte si paralelismul liniilor. Ea este o transformare liniara care se reprezinta matricial in coordonate omogene. Forma generala a matricii de transformare este:

            sx  shx tx

      M=            shy sy  ty

                           0   0   1

iar transformarea coordonatelor utilizator in coordonate dispozitiv este exprimata astfel:

        xd                   xu

        yd     =  M\*    yu

         1                     1

 unde

- (xu, yu) reprezinta un punct in coordonate utilizator

- (xd, yd) reprezinta un punct in coordonate dispozitiv

- sx si sy sunt factorii de scalare, shx si shy sunt factorii de forfecare (shearing) iar (tx, ty) exprima  translatia.

In capitolul „Transformari geometrice 2D” sunt prezentate detalii despre diferitele tipuri de transformari geometrice 2D.

Java2D ofera functii care permit specificarea transformarilor de scalare, rotatie, forfecare si translatie si functii de interogare si setare a transformarii curente. Astfel, functiile rotate(), scale(), shear() si translate(), apelate cu parametrii specifici transformarii corespunzatoare, modifica matricea curenta a transformarii din contextul grafic prin acumularea transformarii. Aceasta inseamna ca,  daca matricea de transformare setata in contextul grafic este MG si matricea de transformare specificata prin parametri este MT, atunci dupa executia uneia dintre functiile mentionate transformarea setata in contextul grafic va fi definita prin matricea M=MG\*MT. Deci, prima transformare aplicata coordonatelor este ultima specificata.

**Exemple:**

Graphics2D g;

g.drawLine(100,100,300,100);//deseneaza o linie orizontala

                           //intre coordonatele dispozitiv

                           //  (100,100)-(300,100)

g. translate(0.0,200.0);

g.drawLine(100,100,300,100);//deseneaza o linie orizontala

                           //intre coordonatele dispozitiv

                           //  (100,300)-(300,300)

g. translate(0.0,-200.0);

g.scale(2.0, 2.0);

g.drawLine(50,50,150,50);//deseneaza o linie orizontala intre

                         //coordonatele dispozitiv

                        //  (100,100)-(300,100)

g.scale(0.5, 0.5);//se revine la factori de scalare egali cu 1

g.rotate(Math.PI/4);

g.drawString(„Text rotit cu 45 grade fata de originea

             implicita”, 200,200);

g.rotate(-Math.PI/4);

Rotatia se aplica atat pozitiei de afisare a textului (200, 200) cat si liniei de baza a textului. Rotatia cu un unghi pozitiv se efectueaza dinspre directia pozitiva a axei OX spre directia pozitiva a axei OY, in cazul de fata in jos.

Acelasi efect se poate obtine cu secventa:

AffineTransform t=g.getTransform();//se obtine transformarea

                                   //curenta

g.rotate(Math.PI/4);//modifica transformarea

g.drawString(„Text rotit cu 45 grade fata de originea

             implicita”, 200,200);

g.setTransform(t);//reface vechea transformare

Functia transform()a clasei Graphics2D permite specificarea unei transformari care se compune cu transformarea curenta din contextul grafic:

public abstract void transform(AffineTransform Tx)

Alte functii care permit crearea si manipularea transformarilor sunt definite in clasa AffineTransform. Clasa contine mai multi constructori care primesc ca parametri elementele matricei de transformare. Astfel este:

public AffineTransform(float m00, float m10, float m01,

                       float m11,float m02, float m12);

unde mij sunt elementele matricei de transformare, primul indice reprezentand randul iar al doilea coloana. De exemplu, apeland constructorul ca mai jos,

AffineTransform t= AffineTransform( 2.0, 0.0, 0.0, 2.0,

                                   0.0,0.0);

se creaza un obiect **AffineTransform** care contine o transformare de scalare cu factorul 2 pe ambele axe.

3.1.6. Combinarea culorilor

Clasa Graphics contine atributul Mod raster care, prin setXORMode(), poate fi setat pentru a produce o combinare simpla si specializata a culorilor. Java 2D generalizeaza regula de combinare prin interfata java.awt.Composite si implementarea sa java.awt.AlphaComposite.

Regulile de compozitie definite in clasa AlphaComposite se bazeaza pe notiunea de culoare transparenta. Atunci cand o culoare sursa, Cs, este afisata peste o culoare destinatie, Cd, cele doua culori sunt combinate pentru a produce o noua culoare destinatie, Cd’, calculata cu formula:

Cd’ = Cs\*α + Cd\*(1- α )

unde α este o valoare reala cuprinsa intre 0 si 1, numita *factorul de transparenta*. Formula se aplica separat pentru componentele red, green, blue ale celor doua culori, care sunt si ele reprezentate prin valori reale cuprinse intre 0 si 1. Daca α=1, culoarea Cs este opaca si Cd’=Cs iar daca α=0, Cs este complet transparenta si atunci Cd’ =Cd. Valori intermediare ale lui α corespund unor culori sursa translucide care produc o culoare finala Cd’ ca o combinatie intre Cs si Cd.

Regulile de combinare definite deAlphaCompositereprezinta un subset din regulile de combinare „Porter-Duff”. Ele sunt prezetate succint in tabelul de mai jos.

Formula de combinare prezentata mai sus corespunde regulii SRC\_OVER. Pentru regula DST\_OVER, se interschimba rolul celor doua culori in cadrul formulei.

Formula de combinare se generalizeaza la forma

Cd’ = Cs\*Fs + Cd\*Fd

unde Fs si Fd sunt specifici fiecareia dintre cele 8 reguli de combinare suportate de clasa  AlphaComposite**.**

|  |  |
| --- | --- |
| http://www.rasfoiesc.com/files/informatica/java/8_poze/image014.gif | Source-over (SRC\_OVER)Pixelii din obiectul desenat (sursa) sunt scrisi peste pixelii  existenti (destinatie). Culoarea rezultata depinde de transparenta culorii sursa:αs=1: sursa (opaca) inlocuieste destinatia αs=0:sursa transparenta, destinatia ramane neschimbataFs= αs, Fd=1- αs |
| http://www.rasfoiesc.com/files/informatica/java/8_poze/image015.gif | Source-in (SRC\_IN)Afiseaza culoarea sursa folosind transparenta culorii destinatie:αd=1: destinatia opaca, se inlocuieste cu culoarea sursa opacaαd=0: destinatia transparenta, ramane transparenta-destinatia translucida, se inlocuieste cu o versiune translucida a sursei. Culoarea destinatie nu este combinata cu cea a sursei.Fs= αd,  Fd=0 |
| http://www.rasfoiesc.com/files/informatica/java/8_poze/image016.gif | Source-out (SRC\_OUT)Este inversa regulii SRC\_IN. αd=1: destinatia opaca devine transparentaαd=0: destinatia transparenta, este inlocuita cu versiunea opaca a surseiFs=1- αd, Fd=0 |
| http://www.rasfoiesc.com/files/informatica/java/8_poze/image017.gif | Destination-over (DST\_OVER)Culoarea finala depinde de transparenta culorii destinatie:αd=1: destinatia (opaca) ramane neschimbataαd=0: destinatia transparenta, se afiseaza culoarea sursaFs=1- αd, Fd= αd |
| http://www.rasfoiesc.com/files/informatica/java/8_poze/image018.gif | Destination-in (DST\_IN)Regula ignora culoarea sursei dar modifica destinatia pe baza transparentei sursei:αs=1: sursa opaca, destinatia ramane neschimbataαs=0: sursa transparenta, destinatia devine transparentaFs=0, Fd= αs |
| http://www.rasfoiesc.com/files/informatica/java/8_poze/image019.gif | Destination-out (DST\_OUT)Este inversa regulii DST\_IN.αs=1: sursa opaca, destinatia devine transparentaαs=0: sursa transparenta, destinatia ramane nemodificataFs=0, Fd= 1-αs |
| http://www.rasfoiesc.com/files/informatica/java/8_poze/image020.gif | Clear (CLEAR)Ignora atat culoarea sursa cat si culoarea destinatie.Fs=0, Fd=0Pixelii destinatiei sunt setati la culoarea negru transparent  |
| http://www.rasfoiesc.com/files/informatica/java/8_poze/image014.gif | Source (SRC)Culoarea sursa inlocuieste culoarea destinatie. Destinatia este ignorata si cele doua culori nu sunt combinate.Fs=1, Fd=0 |

Pentru a schimba regula de combinare dintr-un context Graphics2D, se foloseste metoda setComposite (AlphaComposite).

Clasa AlphaComposite nu ofera constructori, in schimb include campuri membre statice de tipul AlphaComposite pentru fiecare din regulile implementate: AlphaComposite.SrcOver, AlphaComposite.SrcIn, etc. De asemenea, este oferita si o metoda statica

public static AlphaComposite getInstance(int rule, float alfa)

care intoarce una dintre aceste instante statice.

De exemplu, obiectul AlphaComposite implicit intr-un context Graphics2Deste un obiect creat astfel:

AlphaComposite.getInstance(AlphaComposite.SRC\_OVER, 1.0f);

Clasa Color din Java 2D include constructori noi care pot fi folositi pentru crearea de culori translucide. De exemplu:

Graphics2D g;

Color c = new Color (0.0f, 0.0f, 1.0f, 0.5f);//albastru cu α=0.5

g.setPaint(c);

g.filRect(20,50,100,50);

Acelasi efect se poate obtine cu secventa:

Graphics2D g;

Color c = new Color (0.0f, 0.0f, 1.0f);//albastru opac, α=1.0

g.setPaint(c);

g.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC\_OVER, 0.5f);

g.filRect(20,50,100,50);

Daca se modifica regula de combinare existenta intr-un context grafic Graphics2D este recomandabil ca dupa incheierea redarii sa fie refacut stilul initial, care poate fi obtinut cu getComposite()**.** Aceasta este o regula general valabila pentru modificari efectuate asupra oricarui atribut al unui context grafic.

3.1.7. Calitatea redarii (rendering hints)

Cu ajutorul acestui atribut se pot specifica optiuni in realizarea compromisului viteza/calitate: daca se doreste ca redarea obiectelor sa se efectueze cat mai repede sau cu o calitate cat mai ridicata.

Atributul este de tipul RenderingHints si poate include mai multe sugestii ale programatorului cu privire la modul in care el doreste sa fie generata  imaginiea. De exemplu, cu sau fara antialiasing. Nu toate implementarile Java 2D tin cont de setarile existente in acest atribut si totodata in diferite implementari valorile implicite pot fi diferite la fel ca si interpretarea diferitelor tipuri de sugestii.

Informatiile incluse intr-un obiect RenderingHints sunt urmatoarele:

         Alpha interpolation  – default, quality sau speed

         Antialiasing              – default, on sau off

         Color rendering        – default, quality sau speed

         Dithering                  – default, disable sau enable

         Fractional metrics    – default, on sau off

         Interpolation             – nearest-neighbor, bilinear sau bicubic

         Rendering                 – default, quality sau speed

         Text antialiasing       – default, on sau off

         Stroke control           – default, normalized sau pure

Cand o astfel de informatie este setata la valoarea default, va fi folosita valoarea implicita de pe platforma pe care va rula programul.

Fiecare dintre aceste informatii este identificata printr-o cheie definita drept camp membru static final in clasa RenderingHints, spre exemplu RenderingHints**.** KEY\_ALPHA\_INTERPOLATION. Valorile permise pentru fiecare informatie sunt de asemenea definite in acelasi mod, spre exempluRenderingHints. VALUE\_ANTIALIAS\_ON.

Pentru accesarea atributului “rendering hints” al unui obiect Graphics2D sunt disponibile urmatoarele metode:

         RenderingHints getRenderingHints()

         void setRenderingHints(Map hints)

         Object getRenderingHint(RenderingHints.Key hintKey)

         void setRenderingHint(RenderingHints.Key hintKey,

     Object hintValue)

Cu ajutorul primelor metode poate fi obtinut/modificat intregul set de informatii, in timp ce ultimele metode permit obtinerea/modificarea doar a unei singure informatii.

Spre exemplu, urmatoarea instructiune seteaza preferinta pentru utilizarea antialiasingului, daca este posibil:

g2.setRenderingHint (RenderingHints.KEY\_ANTIALIASING,

RenderingHints.VALUE\_ANTIALIAS\_ON);

3.2. Operatiile grafice Java2D

Java2D permite definirea de forme arbitrare. Acestea pot fi contururi de suprafete, linii drepte sau curbe. O forma generala Java2D este referita ca „path” deoarece descrie calea pe care ar urma-o un creion pentru a desena forma. Formele sunt abstractizate in interfata java.awt.Shape. Multe dintre formele uzuale sunt implementate in pachetul ava.awt.geom. Acestea sunt definite in clasele:  Arc2D, Area, CubicCurve2D, Ellipse2D, GeneralPath, Line2D, QuadCurve2D, Rectangle2D, RectangularShape, RoundRectangle2D.

In plus, pachetul java.awt.geom implementeaza si clasele Point2D si Dimension2D**.**

Interfata Shape defineste o serie de metode generale pentru operatii cu formele geometrice. Astfel sunt:

- getBounds() si getBounds2D(), care intorc dreptunghiul incadrator al formei;

- metodele contains(), care testeaza daca o forma include un punct sau o suprafata dreptunghiulara;

- metodele intersect(), care testeaza intersectia unei forme cu un dreptunghi.

3.2.1. Linii

Liniile sunt definite in clasele Line2D.Float si Line2D.Double din java.awt.geom. Ambele sunt derivate din clasa abstracta Line2D. Pot fi afisate folosind metoda draw(). De exemplu, pentru a desena un segment orizontal intre x1 si x2 se apeleaza:

g2D.draw(new Line2D.Float(x1,0.f,x2,0.f));

Clasa Line2D contine mai multe metode care implementeaza operatii geometrice, cum ar fi: intersectia a doua linii drepte (intersectsLine() si linesIntersect()), distanta de la un punct la o linie dreapta (ptLineDist()) si altele.

De exemplu:

Line2D.Float l= new Line2D.Float(x1,y1,x2,y2);

if(l.intersectsLine(0.f, y11, 0.f, y21))

            g2D.draw(l);

este echivalent cu:

if(Line2D.linesIntersect(x1, y1, x2, y2,0.f,y11,0.f,y21))

  g2D.draw(new Line2D.Float(x1,y1,x2,y2));

3.2.2. Forme dreptunghiulare, eliptice si circulare

Formele dreptunghiulare sunt obiecte definite prin clasele Rectangle2D, RoundRectangle2D, Arc2D, Ellipse2D. Ele sunt ilustrate in figura 3.4.



Figura 3.4.

Toate sunt specificate prin dreptunghiul incadrator. De exemplu, pentru a desena o elipsa in interiorul unui dreptunghi se poate utiliza codul:

Graphics2D g;

Shape dreptunghi=new Rectangle2D.Float(100.0f,100.0f,300.0f,200.0f);

Shape elipsa = new Ellipse2D.Float(100.0f,100.0f,300.0f,200.0f);

g.draw(dreptunghi);

g.draw(elipsa);

3.2.3. Poligoane

Poligoanele sunt definite in clasa java.awt.Polygon. Clasa ofera constructori pentru definirea de poligoane precum si metodele care implementeaza operatii geometrice cu poligoane (metodele contains(), getBounds(), intersects()si altele ). De exemplu, pentru definirea unui poligon se poate apela constructorul:

public Polygon(int[] xpoints, int[] ypoints, int npoints)

Un exemplu mai cuprinzator de afisare a  formelor folosind metodele draw() si fill() este prezentat in 3.3.1.

3.2.4. Curbe

Pachetul java.awt.geom contine clase care permit afisarea de curbe Bézier cuadrice (QuadCurve2D.Float si QuadCurve2D.Double)si cubice (CubicCurve2D.Float  si  CubicCurve2D.Double) in planul XOY.

O curba cuadratica este definitia prin extremitatile sale si un punct de control (figura 3.5.):

public QuadCurve2D.Float(float x1,float y1,

                         float ctrlx,float ctrly,

                         float x2,float y2)

Exemplu:

Graphics g;

g.draw(new QuadCurve2D.Float(-1.f, 0.f, 0.f, 1.f, 1.f, 0.f));

 O curba cubica este definita prin cele doua puncte terminale si doua puncte de control (figura 3.5.):

public CubicCurve2D.Float(float x1,float y1,

                          float ctrlx1,float ctrly1,

                          float ctrlx2,float ctrly2,

                         float x2,float y2)



Figura 3.5.

In paragraful 3.3.2 sunt prezentate doua applet-uri care permit definirea si modificarea interactiva a curbelor Bezier folosind clasele QuadCurve2D si CubicCurve2D.

3.2.5. Forme generale

Clasa GeneralPath defineste o forma geometrica alcatuita din segmente de dreapta si  curbe Bézier cuadratice sau cubice. De exemplu, forma redata in figura 3.6. poate fi creata din trei segmente de linie si o curba cubica.



Figura 3.6.

Pentru crearea unei forme generale se apeleaza mai intai unul dintre constructorii clasei, de exemplu constructorul:

public GeneralPath(int rule, int initialCapacity)

Primul parametru specifica regula pe baza careia se determina interiorul unei forme de tip GeneralPath. Valorile posibile sunt:

WIND\_EVEN\_ODD: Un punct este interior unei forme daca numarul de intersectii dintre o semidreapta care pleaca din punctul respectiv si conturul formei este impar WIND\_NON\_ZERO: Un punct este interior unei forme daca ducand o semidreapta din punctul respectiv, numarul de intersectii ale conturului formei cu raza de la stanga la dreapta este diferit de numarul de intersectii ale conturului formei cu raza de la dreapta la stanga.

Al doilea parametru specifica numarul estimat de puncte prin care va fi definita forma. Pe masura adaugarii de segmente la traseul formei, memoria alocata este extinsa.

Dupa crearea formei, ea este definita iterativ, adaugand segmente de dreapta sau de curba. In acest scop se folosesc metodele:

public void moveTo(float x, float y)

Adauga un punct la forma, care devine ultimul punct pe traseul formei.

public void lineTo(float x, float y)

Adauga un punct la forma si traseaza segment de dreapta din ultimul punct pana in punctul adaugat.

public void quadTo(float ctrlx, float ctrly,float x2, float y2)

Adauga un segment de curba Bezier cuadrica. Curba are ca extremitati ultimul punct de pe traseul formei si punctul (x2,y2), iar ca punct de control punctul (ctrlx, ctrly).

public void curveTo(float ctrlx1, float ctrly1, float ctrlx2, float ctrly2, float x3, float y3)

Adauga un segment de curba Bezier cubica. Curba are ca extremitati ultimul punct de pe traseul formei si punctul (x3,y3), iar ca puncte de control punctele (ctrlx1, ctrly1) si (ctrlx2, ctrly2).

Pentru inchiderea traseului unei forme se poate folosi metoda:

public void closePath()

Forma este inchisa prin trasarea unui segment de dreapta intre ultimul punct al traseului si ultimul punct adaugat cu moveTo().

Exemplul Shapes2Ddemo prezentat in paragraful 3.3.1. foloseste GeneralPath pentru a desena poligoane in forma de clepsidra.

Exemplul urmator foloseste GeneralPath pentru a crea forma arbitrara din figura 3.7.



Figura 3.7.

import java.awt.\*;

import java.awt.event.\*;

import java.awt.geom.\*;

import java.awt.image.BufferedImage;

**class OddPanel extends Panel**

**public static void main(String s[])**

  });

    Panel oddPanel = new OddPanel();

    oddPanel.setBackground(Color.white);

    f.add(oddPanel);

    f.pack();

    f.setSize(new Dimension(350, 250));

    f.setVisible(true);

  }

 **public void paint(Graphics g)**

}

3.2.6. Forme definite prin operatii booleene

Procesul de construire a unor forme geometrice prin efectuarea de operatii booleene cu forme deja existente este cunoscut sub numele de “Constructive area geometry” (CAG). In Java 2D exista o clasa numita Area, care permite construirea de forme arbitrare prin operatii booleene de reuniune, intersectie, scadere si sau-exclusiv (xor) intre obiecte de tip Shape.

Un obiect Area poate fi construit din orice obiect de tip Shape, apoi poate fi folosit pentru efectuarea unei operatii booleene intre el si un alt obiect Area transmis ca parametru metodei care implementeaza operatia.

Prezentam in continuare un exemplu de construire de suprafete prin operatiile boleene ale clasei Area. Appletul construieste suprafetele afisate in fereastra sa (figura 3.8.).



Figura 3.8.

import java.awt.\*;

import java.awt.geom.\*;

import java.applet.Applet;

**public class AreaDemo extends java.applet.Applet**

, 0);

  public void init ()

**public void paint (Graphics g)**

  ;

    for (int i=0; i<4; i++)

    g2.setFont(savedFont);

  }

}

3.2.7. Texte

Pentru afisarea unui text poate fi folosita o componenta specializata, de exemplu una dintre cele oferite de Swing. O astfel de componenta ofera o multime de facilitati gata implementate – de exemplu obiectele JTextComponent furnizeaza suport pentru „hit testing” si afisarea de text „internationalizat”. Daca in schimb se doreste doar afisarea unui sir de caractere static, acesta poate fi redat folosind metoda drawString() din Graphics2D:

public abstract void drawString(String str, int x, int y)

Textul, specificat prin primul parametru, este afisat folosind atributele de tip Font,Paint, Composite, Clip si Transform ale contextului grafic. Linia de baza a primului caracter incepe din pozitia (x, y) in spatiul coordonatelor utilizator.

Pentru controlul amplasarii textului se poate folosi clasa TextLayout.

Daca se doreste control complet asupra modului in care textul este modelat si afisat, se pot construi propriile obiecte GlyphVector si fiecare GlyphVector poate fi redat cu ajutorul metodei drawGlyphVector() din Graphics2D:

public abstract void drawGlyphVector(GlyphVector g, float x, float y)

(x, y) este pozitia in care va fi afisat textul definit in vectorul g.

Prezentam in continuare cateva detalii privind modul de amplasare si redare a unui paragraf de text cu „stil” folosind TextLayout si LineBreakMeasurer.

Pentru formatarea unui paragraf de text intr-o anumita latime se poate folosi LineBreakMeasurer, care permite ca textul sa fie impartit in linii formate din cuvinte. Metodele getAscent si getDescent din TextLayout intorc informatii despre fontul utilizat, pentru pozitionarea liniilor de text in interiorul componentei. Textul este stocat ca un AttributedCharacterIterator, pentru ca atributele de font, dimensiune, etc. ale fiecarui caracter sa poate fi luate in consideratie.

Urmatoarea secventa creaza un iterator dintr-un sir de caractere. Dupa obtinerea inceputului si sfarsitului iteratorului, se creaza un obiect nou LineBreakMeasurer.

String string = 'test';

AttributedCharacterIterator paragraph = vanGogh.getIterator();

paragraphStart = paragraph.getBeginIndex();

paragraphEnd = paragraph.getEndIndex();

lineMeasurer = new LineBreakMeasurer(paragraph,

                   new FontRenderContext(null, false, false));

Dimensiunile ferestrei sunt folosite pentru a determina cand trebuie intrerupta o linie de text, si pentru fiecare linie a paragrafului este creat un obiect TextLayout.

Dimension size = getSize();

float formatWidth = (float) size.width;

float drawPosY = 0;

lineMeasurer.setPosition(paragraphStart);

while (lineMeasurer.getPosition() < paragraphEnd)

   else

   layout.draw(graphics2D, drawPosX, drawPosY);

   drawPosY += layout.getDescent() + layout.getLeading();

}

|  |
| --- |
|  |