**I**

**PENGANTAR KOMPUTER GRAFIK**

**1.1 Pengenalan Komputer Grafik Keperluan dalam belajar pemrograman Grafik**

Komputer grafik (grafika komputer): Proses pembuatan, manipulasi, penampilan grafik (2D/3D), citra, animasi dan sejenisnya ke display, layar komputer, printer, maupun devais lainnya.

Ada beberapa hal yang diperlukan dalam belajar pemrograman grafik, yang meskipun tidak mutlak, dapat membantu lebih mengerti algoritma grafik.

1. Akses atau abstraksi ke piksel dalam layar

2. Cross platform untuk melihat efek grafik dalam berbagai lingkungan (misal: efek tidak adanya font tertentu dalam sistem operasi tertentu, tidak adanya antialiasing dalam lingkungan tertentu)

3. Level menengah: tidak terlalu low level (sehingga terlalu kompleks untuk dipelajari), ataupun high level (sehingga semua sudah ditangani, dan tidak ada yang bisa dipelajari)

4. Tersedia secara gratis dan atau open source

**Alasan Penggunaan Java**

Lingkungan DOS sebenarnya sangat baik digunakan untuk memahami pembangunan grafik secara low level, namun usaha akan banyak diperlukan, dari sekedar menggambar titik, sampai berurusan dengan interrupt untuk mengakses device seperti mouse. Device-device yang lebih kompleks, seperti digital camera, sangat sulit untuk bisa diakses.

Lingkungan Windows memberikan fasilitas yang sangat banyak untuk melakukan pemrograman grafik, baik 2D maupun 3D, namun umumnya pemrograman grafik di Windows membuat kita tidak bisa melihat bagaimana algoritma sebenarnya bekerja, karena terlalu banyak hal yang sudah disediakan oleh Windows.

Pemrograman low level grafik (dengan C/C++) dan mengakses GDI Windows atau Direct X terlalu kompleks sedangkan pemrograman dengan bahasa visual seperti Visual Basic dan Delphi menyembunyikan terlalu banyak hal yang seharusnya dipelajari dalam kuliah grafik. Belajar pemrograman grafik di Windows juga akan mengikat kita pada satu platform tertentu, sedangkan banyak aplikasi grafik yang tidak berjalan di Windows.

**Java untuk belajar pemrograman Grafik**

Java merupakan bahasa yang cross platform, dan sudah menyediakan primitif grafik 2D dan secara opsional grafik 3D. Java cukup mudah dipelajari, dan bisa mengakses mode grafik dalam lingkungan manapun (X Window, GDI Windows, dll).

Java juga sudah digunakan sebagai sarana pembantu dalam banyak mata kuliah, termasuk juga pemrograman sistem terdistribusi dan sistem operasi.

**Ekstensi Grafik pada Java**

Selain operasi grafik dasar, Java juga mendukung pemrosesan grafik 2D melalui kelas-kelas Java2D dan pemrosesan grafik 3D melalui paket tambahan (bukan standar) Java3D. Dalam hal Java3D ini tidak dibahas di dalam diktat ini, namun informasi mengenai hal tersebut dapat dicari dalam dokumentasi yang disertakan.

**1.2 Shapes (Bentuk)**

Kelas Grafis mencakup sejumlah besar metode misalnya untuk menggambar berbagai bentuk, seperti garis, persegi panjang, dan oval. Bentuk ditentukan menggunakan sistem (x, y) koordinat dijelaskan di atas. Mereka tertarik dalam warna gambar saat ini dari konteks grafis. Warna gambar saat ini diatur dengan warna latar depan komponen ketika konteks grafis dibuat, tetapi dapat diubah setiap saat dengan menggunakan metode setcolor().

Berikut adalah daftar dari beberapa metode gambar yang paling penting. Dengan semua perintah, setiap gambar yang dilakukan di luar batas komponen diabaikan. Perhatikan bahwa semua metode ini di kelas Graphics, sehingga mereka semua harus dipanggil melalui sebuah objek dari tipe Grafis.

1. drawString(String str, **int** x, **int** y)

2. drawLine(**int** x1, **int** y1, **int** x2, **int** y2)

3. drawRect(**int** x, **int** y, **int** width, **int** height)

4. drawOval(**int** x, **int** y, **int** width, **int** height)

5. drawRoundRect(**int** x, **int** y, **int** width, **int** height,

**int** xdiam, **int** ydiam)

6. draw3DRect(**int** x, **int** y, **int** width, **int** height,

**boolean** raised)

7. drawArc(**int** x, **int** y, **int** width, **int** height,

**int** startAngle, **int** arcAngle)

8. fillRect(**int** x, **int** y, **int** width, **int** height)

9. fillOval(**int** x, **int** y, **int** width, **int** height)

10. fillRoundRect(**int** x, **int** y, **int** width, **int** height,

**int** xdiam, **int** ydiam)

11. fill3DRect(**int** x, **int** y, **int** width, **int** height,

**boolean** raised)

12. fillArc(**int** x, **int** y, **int** width, **int** height,

**int** startAngle, **int** arcAngle)

*Untuk lebih jelasnya kita akan mencoba beberapa program sederhana*

*pada bab selanjutnya.*

**II**

**PEMBENTUKAN GRAFIK PADA JAVA**

**2.1 Grafik dasar pada java**

Pada grafik java dasar, terdapat beberapa objek grafik yang nilai koordinat pada grafik itu adalah bilangan bulat (integer).

Koordinat grafik dalam komputer berbeda dengan koordinat grafik sehari-hari. Perhatikan gambar berikut:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Koordinat dalam grafik Komputer**  The image “http://zahra/~_ngAli/Ngajar/Programming/Java%20Programming/Java%20Unleashed,%20Second%20Edition/f19-2.gif” cannot be displayed, because it contains errors. | |  |  |  |  | | --- | --- | --- | --- | | ***Color*** | Red | Green | Blue | | White | 255 | 255 | 255 | | Black | 0 | 0 | 0 | | Light Gray | 192 | 192 | 192 | | Dark Gray | 128 | 128 | 128 | | Red | 255 | 0 | 0 | | Green | 0 | 255 | 0 | | Blue | 0 | 0 | 255 | | Yellow | 255 | 255 | 0 | | Purple | 255 | 0 | 255 |   **Warna dasar dalam grafik** |

**Gambar 1.**

Grafik dasar pada Java

Berikut ini contoh applet yang menggambar berbagai bentuk di grafik Java.

Contoh 2.1. Fungsi-fungsi Dasar Grafis :

//BasicGraphics.java

1 import java.applet.Applet;

2 import java.awt.Graphics;

3 import java.awt.Color;

4 public class BasicGraphics extends Applet {

5 public void paint (Graphics g) {

6 **int xPts[] = {5, 25, 50, 30, 15, 5};**

7 **int yPts[] = {10, 35, 20, 65, 40, 10};**

8 **g.drawString ("Java Graphics Basics",0,50);**

9 **g.drawLine(5, 100, 150, 250);**

10 **g.setColor(Color.red);**

11 **g.drawRect(50, 100, 150, 250);**

12 **g.drawRoundRect(50, 150, 150, 250, 6, 12);**

13 **g.drawPolygon(xPts, yPts, xPts.length);**

14 // cara lain membuat polygon

15 // **Polygon poly = new Polygon(xPts+100, yPts+100,**

**xPts.length);**

16 // **g.fillPolygon(poly);**

17 **g.drawOval(150, 100, 250, 55);**

18 **g.drawArc(50, 100, 150, 275, 95, 115);**

19 }

20 }

**Keterangan per-baris:**

1. Disini adalah meng*import* library applet, dimana diperlukan karena class ini meng-*extends* applet, note: jika ingin menggunakan JPanel atau *Frame* sebagai tempat kita untuk mengimplementasikan grafik, itu sangat diperbolehkan dan sudah didapatkan materinya saat pemrograman visual, untuk contoh sederhana maka digunakan applet.
2. dalam baris ini kita mengimport library **Graphics**, berfungsi untuk menampilkan beberapa objek grafik dasar pada class itu sendiri. Bisa kita liat pada metode **paint** terdapat pada baris 5>> “public static void paint **(Graphics g) { }**”, **g** disitu sebagain nama objek yang telah diinstasiasi, jadi pemanggilan dalam Graphics jadi seperti ini >> contoh: **g.drawLine**.
3. library yang berfungsi untuk memanggil warna-warna yang terdapat dalam grafik dasar.
4. nama class yang ber-extends Applet
5. metode **paint** (sudah dibahas pada nomor 2)
6. sebuah variabel array yang mempunyai type data integer (xPts [ ]).
7. sebuah variabel array yang mempunyai type data integer (yPts [ ]).
8. **g.drawString ("Java Graphics Basics",0,50);** >> *drawString(“String yang diinginkan”, posisi koordinat X, posisi koordinat Y);*
9. **g.drawLine(5, 100, 150, 250);** >> *drawLine( koordinat X, koordinat Y, koordinat X1, koordinat Y1);* >> di dalam **Line** tedapat dua titik gabungan antara X & Y lalu X1 & Y1 lalu akan membentuk sebuah garis.

Contoh gambar: **drawLine(2, 2, 7, 7);**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |

**Gambar 2.**

Pembentukan bidang garis

Diatas menerangkan dalam skala pixel yang merupakan gambar sebuah garis atau line.

1. **g.setColor(Color.red);** >> mengambil warna dasar dalam grafik komputer berwarna merah, dan untuk menampilkan warna ini dibutuhkan library **java.awt.Color**.
2. **g.drawRect(50, 100, 150, 250)** >> dalam objek grafik ini akan membentuk suatu Rectangle(persegi empat), terdapat 4 koordinat dalam **Rect**, yaitu *drawRect(koordinat X, koordinat Y, nilai a + koordinat X, nilai b + koordinat Y);*. Nilai a disini adalah **150**, dan nilai b adalah **250**, jadi jika masing ditambahkan sesuai dengan koordinat yang sudah ditentukan maka *(nilai a + koordinat X = panjang X) dan (nilai b + koordinat Y = panjang Y)*, lalu akan membentuk suatu persegi.
3. **g.drawRoundRect(50,100,150,250,6,12)**>> *g.drawRoundRect(koordinat X, koordinat Y, a + koordinat X, nilai b + koordinat Y, diameter X, diameter Y);*. Diameter X dan Y diukur dari tiap sudut persegi, lalu akan membentuk suatu persegi dengan lengkungan di tiap sudutnya.
4. **g.drawPolygon(xPts, yPts, xPts.length);** >> Polygon atau disebut juga segi banyak, di dalam gambar grafik yang satu ini kita harus menyiapkan dua variabel array, yang tujuannya untuk mengambil nilai dan menghasilkan beberapa titik, lalu dari titik-titik itu dihubungkan yang menjadi sebuah bentuk polygon. *g.drawPolygon(xPts, yPts, xPts.length) >> xPts = nilai variabel array untuk tiap koordinat X, yPts = nilai variabel array tiap koordinat Y, xPts.length = merupakan panjang array dari variabel xPts yang berfungsi sebagai patokan banyaknya titik yang akan dihubungkan.* Cara membaca coding pada Polygon adalah >> contoh :

*drawPolygon(xPts, yPts, xPts.length); 🡪 baca tiap per-indeks pada array, misal drawPolygon(xPts* ***[0]****, yPts* ***[0]****, xPts.length);*

*drawPolygon(xPts* ***[1]****, yPts* ***[1]****, xPts.length);*

*drawPolygon(xPts* ***[2]****, yPts* ***[2]****, xPts.length);*

dan seterusnya sampai length yang ditentukan (xPts.length), menentukan length disini juga kita bisa dibuat dalam bentuk bilangan bulat, contoh : drawPolygon(xPts [0], yPts [0], **6**);

1. hanya sebuah komentar
2. cara kedua pembuatan grafik polygon, dengan cara membuat objek dari Poygon.
3. Pemanggilan objek Polygon, yang sudah diinstasiasikan dan langsung dipanggil langsung pada drawPolygon.
4. **drawOval(150, 100, 250, 55);** >> drawOval(koordinat X, koordinat Y, nilai a + koordinat X, nilai b + koordinat Y);, disini hasil dari Oval adalah berupa lingkaran. **Oval** sama konsepnya dengan **Rect**, oval membutuhkan diameter jadi pada paramater ***“nilai a + koordinat X = diameter X”*** *dan* ***“nilai b + koordinat Y = diameter Y”****.*
5. **g.drawArc(50, 100, 150, 275, 95, 115);**  >> ***Arc*** *disebut juga dengan bentuk* ***Lengkung****, drawArc(****koordinat X****,* ***koordinat Y****,* ***diameter X****,* ***diameter Y****,* ***titik awal perputaran, perputaran lingkaran****);*. “Masih ada lanjutannya isitrahat dulu ya :D”

File HTML untuk memanggilnya adalah sbb:

<!-- BasicGraphics.html -->

<HTML><BODY>

<APPLET CODE="BasicGraphics.**class**" WIDTH = 500 HEIGHT=500> </APPLET>

</BODY></HTML>

Berikut ini contoh applet untuk menampilkan **String** yang terformat di dalam grafik.

Contoh 2.2. Menampilkan String terformat dalam Grafik

**import** java.applet.\*;

**import** java.awt.\*;

**public** **class** DrawText **extends** Applet

{

**public** **void** paint(**Graphics** g)

{

Font font = **new** Font("Helvetica", Font.BOLD +

Font.ITALIC, 22);

FontMetrics fm = g.getFontMetrics(font);

**String** str = **new** **String**("The highest result of

education is..");

g.setFont(font);

g.drawString(str, (size().width - fm.stringWidth

(str))/2,((size().height –

fm.getHeight()) / 2) +

fm.getAscent());

}

}

**III**

**MEMBANGUN GRAFIK PRIMITIF DENGAN JAVA**

**3.1 Kelas Graphics**

Dengan kemampuan dari kelas graphics yang dimiliki Java dalam *class libraries*, maka dimungkinkan untuk dapat membangun bidang lines, *shape*, character, warna dan image pada suatu applet. Anda tidak harus menciptakan sebuah instance dari Graphics yang tujuannya adalah untuk membangun sesuatu di applet; pada applet anda metode paint() menyediakan objek-objek dari kelas Graphics.

Kelas Graphics adalah bagian dari paket java.awt dengan demikian apabila Anda membangun suatu bidang pada applet, pastikan bahwa Anda harus melakukan import pada class tersebut dan diletakkan pada bagian awal dari file java, contoh :

Import java.awt.Graphics;

Public class GrafikSatu extends java.applet.Applet {

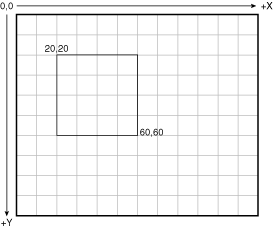
…

}

**Sistem Koordinat Grafik**

Untuk membangun sebuah objek pada layar, Anda harus memanggil salah satu dari metode yang tersedia didalam kelas Graphics. Pada semua metode drawing memiliki argumen yang mewakili titik-akhir, titik-sudut dan titik-awal dari sebuah objek yang berisi nilai-nilai dalam sistem koordinat yang membangun grafik, sebagai contoh : suatu garis dimulai dari titik-awal (10,10) dan titik-akhir (20,20).

*Sistem koordinat pada java selalu dimulai dari titik (0,0) yang diposisikan pada sudut kiri atas*. *Nilai x positif berada di kanan dan nilai y positif berada di bawah*. Semua nilai piksel bertipe integer (tidak ada piksel dengan nilai partial dan fractional).



*Catatan :*

*Sistem koordinat pada java berbeda dengan bahasa pemrograman yang lainnya, dimana dimulai dari sudut bawah. Apabila Anda belum terbiasa bekerja dengan posisi yang terbalik ini, maka disarankan agar Anda mulai memperbanyak latihan dengan sistem koordinat pada java.*

**Gambar 2.**

Sistem koordinat grafik pada java

**3.2 Membangun Bidang “Line”**

Untuk membangun bidang garis lurus, gunakan metode drawLine(). Metode drawLine memiliki 4 (empat) argumen, yaitu : koodinat x dan y pada titik-awal dan koordinat x dan y pada titik-akhir.

Contoh :

import java.awt.Graphics;

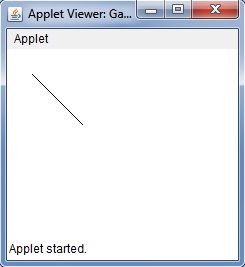
public class MyLine extends java.applet.Applet {

public void paint(Graphics g) {

g.drawLine(25,25,75,75);

}

}



**Gambar 3.**

Tampilan garis pada applet di BlueJ

**3.3 Membangun Bidang “Rectangle”**

Pemrograman grafik primitif pada Java menyediakan beberapa bentuk dari rectangles :

⮊ Segi empat biasa

⮊ Segi empat dengan sudut yang melengkung

⮊ Segi empat dengan tampilan tiga dimensi, yang dibentuk dengan membangun sudut yang berbayang.

Dari setiap segi empat ini, Anda akan mendapati 2 (dua) macam metode yang akan membentuk : *(1) metode yang akan membentuk segi empat dari sisi luar, (2) metode yang akan men-draw segi empat dengan warna didalamnya*.

**3.3.1 Segi Empat Biasa**

Agar dapat menggambar suatu bidang segi empat biasa, gunakan salah satu dari metode drawRect() atau fillRect().

Kedua metode ini memiliki argumen : *koordinat x dan y yang berada pada sudut kiri atas dari segi empat, adalah lebar dan tinggi dari segi empat yang akan di gambar.*

Contoh :

import java.awt.Graphics;

public class SegiEmpat1 extends java.applet.Applet {

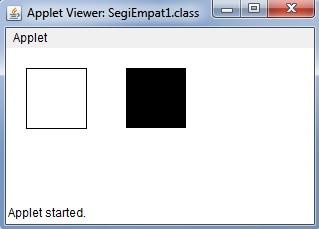
public void paint(Graphics g) {

g.drawRect(20,20,60,60);

g.fillRect(120,20,60,60);

}

}



**Gambar 4**

Bentuk segi empat biasa (kiri) dan bentuk segi empat yang dibentuk dari warna didalamnya (kanan)

**3.3.2 Segi Empat Membulat / Melengkung**

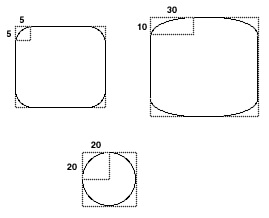
Segi empat membulat/melengkung yang dimaksudkan disini adalah segi empat lengkungan pada sudut-sudutnya. Adapun metode drawRoundRect() dan fillRoundRect(), metode ini mirip dengan metode yang digunakan untuk menggambar segi empat biasa, hanya saja perbedaannya terletak pada nilai lebar dan tinggi yang ditambahkan pada sudut segi empat.

Argumen

tambahan

g.drawRoundRect(20,20,60,60,10,10);

Kedua argumen tambahan (lebar dan tinggi) menjelaskan berapa panjang dari luas tepian pada segi empat yang dimana luas tepian ini akan berbentuk seperti “arc” akan mulai digambar.  
Titik pertama aka membangun arah horizontal dan titik kedua akan membangun garis vertikal. Besaran nilai yang ditambahkan pada setiap titik (lebar, tinggi) akan menciptakan lebih banyak lengkungan secara keseluruhan pada sudut bidang segi empat ; nilai yang sama pada setiap titik akan membentuk lingkaran.



**Gambar 5**

Sudut melengkung pada bidang segi empat

Contoh :

import java.awt.Graphics;

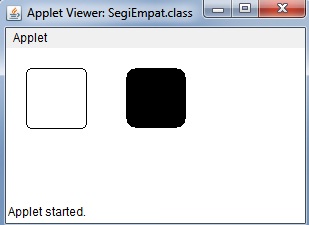
public class SegiEmpat extends java.applet.Applet {

public void paint(Graphics g) {

g.drawRoundRect(20,20,60,60,10,10);

g.fillRoundRect(120,20,60,60,20,20);

Hasil :



**Gambar 6**

Bidang segi empat dengan sudut melengkung (kiri), bidang segi empat dengan sudut melengkung yang diisi dengan warna hitam (kanan)

**3.3.3 Segi Empat Dalam Bentuk 3 (tiga) Dimensi**

Pada dasarnya yang dimaksud dengan bidang 3 dimensi disini adalah bidang segi empat yang memiliki efek berbayang seolah-olah segi empat tersebut berada diatas atau dibawah tampilan applet. *Segi empat ini memiliki 4 (empat) argumen yang diperuntukkan pada titik x dan y dari posisi titik awal dan lebar dan tinggi dari bidang segi empat. Argumen ke-5 berbentuk nilai boolean yang mengindikasikan apakah efek 3D ini dibangun diatas applet (true) atau dibawah applet (false).* Adapun metode yang digunakan untuk membangun bidang segi empat 3D ini adalah : draw3Drect() dan fill3Drect().

Contoh :

import java.awt.Graphics;

public class Kotak3D extends java.applet.Applet

{

public void paint(Graphics g)

{

g.draw3DRect(20,20,60,60,true);

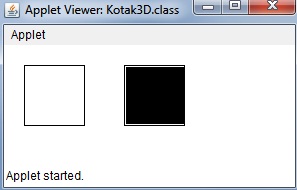
g.draw3DRect(120,20,60,60,false);

g.fill3DRect(122,22,58,57,true);

}

}

Hasil :

****

**Gambar 7**

Bidang segi empat 3D

**3.4 Membangun Bidang “Polygons”**

Polygon adalah bentuk suatu bidang dengan jumlah sisi yang tak tehingga. Untuk menggambar bidang polygon, membutuhkan sekelompok koordinat dari x dan y. Sebuah bidang polygon di gambar seperti halnya ada sejumlah garis lurus yang dimulai dari titik pertama ke titik kedua, titik kedua ke titik ke tiga, dan seterusnya.

Sama seperti segi empat, Anda dapat menggambar bentuk polygon menggunakan metode drawPolygon() dan fillPolygon(). *Anda dapat pula memilih bagaimana mengindikasikan sejumlah titik koordinat (dalam bentuk array dari koordinat x dan y) atau sebagai suatu instan dari kelas Polygon*.

**Menggunakan cara pertama** untuk menggambarkan bidang polygon, metode drawPolygon() dan fillPolygon() membutuhkan 3 (tiga) argumen :

⮊ array bertipe integer yang menjadi koordinat sumbu x

⮊ array bertipe integer yang menjadi koordinat sumbu y

⮊ integer untuk mengumpulkan sejumlah titik-titik yang dibentuk

*Elemen dari array x dan y, harus berada dalam jumlah yang sama*

Contoh :

import java.awt.Graphics;

public class MyPoly extends java.applet.Applet {

public void paint(Graphics g) {

int exes[] = { 39,94,97,142,53,58,26 };

int whys[] = { 33,74,36,70,108,80,106 };

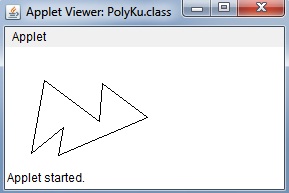
int pts = exes.length;

g.drawPolygon(exes,whys,pts);

}

}

Hasil



**Gambar 8**

Bidang polygon

*Catatan : Java tidak otomatis menutup bidang polygon; apabila ingin membangun bidang polygon secara lengkap, dilengkapi dengan menyertakan titik-awal dari polygon di akhir dari array. Membangun polygon dengan filledPolygon, bisa dikatakan menggabungkan titik awal dan akhir.*

**Cara yang kedua** dari pemanggilan metode drawPolygon() dan fillPolygon() adalah dengan menggunakan objek Polygon untuk mengembalikan nilai pada masing-masing titik dari polygon. Kelas dari Polygon dapat berguna apabila Anda secara intens menambahkan selalu menambahkan sejumlah sumbu kedalam bidang polygon atau Anda sedang mengembangkan bentuk polygon. Menggunakan kelas dari Polygon, dimungkinkan untuk menggunakan polygon sebagai objek dari pada Anda harus mengatur array pembentuk polygon.

Untuk memakai objek polygon, sebelumnya membuat objek kosong dari polygon.

Polygon poly = new Polygon();

Atau membuat polygon dari sejumlah titik dalam array bertipe integer, seperti contoh sebelumnya :

int exes[] = { 39,94,97,142,53,58,26 };

int whys[] = { 33,74,36,70,108,80,106 };

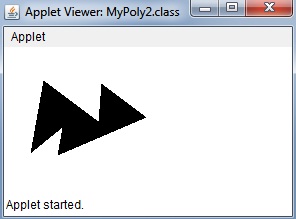
int pts = exes.length;

Polygon poly = new Polygon(exes,whys,pts);

Saat objek polygon sudah tercipta, Anda dapat menambahkan titik sesuai dengan yg diinginkan :

Poly.addPoint(20,35);

Kemudian, gunakan objek polygon tersebut sebagai argumen pada metode drawPolygon() atau fillPolygon().



**Gambar 9**

Polygon sebagai objek

**3.5 Membangun Bidang “Ovals”**

Oval digunakan untuk membetuk bidang elips atau lingkaran. Oval hampir sama dengan segi empat dengan sudut yang melengkung secara keseluruhan. Bidang oval dibangun menggunakan argumen : koordinat x dan y pada sudut atas, dan lebar dan tinggi pada bagian oval itu sendiri.

Seperti halnya metode yang digunakan pada bidang geometri lainnya, metode drawOval() untuk membentuk bidang oval dan metode fillOval() digunakan untuk membentuk bidang oval yang berisi terisi.

Contoh :

import java.awt.Graphics;

public class MyOval extends java.applet.Applet {

public void paint(Graphics g) {

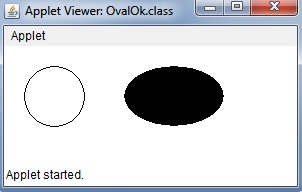
g.drawOval(20,20,60,60);

g.fillOval(120,20,100,60);

}

}

Hasil :



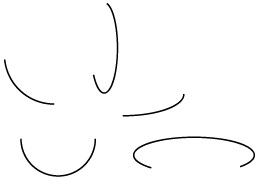
**Gambar 10**

Bidang lingkaran (kiri) dan Bidang elips (kanan)

*Catatan : jangan karena Anda mengira-ngira bahwa oval itu dibentuk dari beberapa jarak di kiri dan naik keatas dari garis oval itu sendiri. Tetapi anggaplah bahwa oval dibentuk dari sebuah kotak, dengan begitu Anda akan lebih mudah membangun bidang oval ini.*

**3.6 Membangun Bidang “Arcs”**

Dari semua bentuk yang dapat dibangun menggunakan metode dari kelas Graphics, arc (busur/lengkung) adalah bidang yang paling kompleks. Busur adalah bagian dari bidang oval; pada dasarnya, cara yang mudah memikirkan bentuk dari sebuah busur adalah bagian dari oval yang komplit.



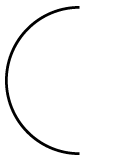
**Gambar 11**

bentuk-bentuk bidang arc

Metode drawArc() memiliki beberapa argumen : sudut awal, lebar dan tinggi, sebuah sudut untuk memulai bentuk arc, sebuah derajat sebagai titik henti. Adapun metode fillArc() digunakan untuk mengisi bidang arc; kedua titik ahir bergabung ke tengah-tengah dari sebuah lingkaran.

Hal terpenting yang harus dimengerti mengenai arc adalah Anda sebenarnya sedang memformulasi bentuk arc sebegai sebuah bidang oval dan hanya menggambarkan bagian itu saja. Posisi titik awal dan lebar dan tinggi, bukanlah membentuk sudut awal, tinggi dan lebar dari bentuk arc yang Anda lihat pada layar; mereka adalah bentuk penuh dari elips yang merupakan bagian dari sebuah arc.

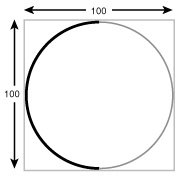
Bagian pertama ini (sudut awal, lebar dan tinggi) menentukan ukuran dan bentuk dari sebuah arc; dua argumen terakhir (derajat) menentukan titik-awal dan titik-akhir dari sebuah arc.



**Gambar 12**

Sebuah arc berbentuk huruf C

Untuk membangun metode yang membentuk arc, langkah awal adalah membayangkan arc sebagai lingkaran penuh. Kemudian Anda akan menemukan koordinat x dan y, dan titik yang mewakili lebar dan tinggi dari lingkaran tersebut. Keempat nilai tersebut adalah empat argumen yang pertama untuk menggambar menggunakan metode drawArc() atau fillArc().

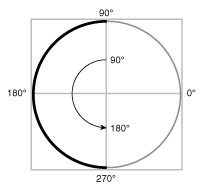


**Gambar 13**

Lingkaran yang akan membentuk sebuah arc

Untuk mendapatkan dua argumen terakhir, sudut mulai lengkung dan dearajat kelengkungan dapat dibayangkan seperti besaran sudut (derajat) putaran berlawanan arah dengan jarum jam. Nol derajat adalah posisi jam 03.00, 90 derajat pada posisi jam 12.00, 180 derajat pada posisi jam 09.00, dan 270 derajat pada posisi jam 06.00. Mulai lengkungan adalah nilai derajat dari bentuk huduf C pada 90 derajat sehingga argumen ke lima adalah 90.

Argumen terakhir atau argumen ke enam adalah nilai derajat lain yang menunjukkan seberapa jauh lingkaran menyapu beserta arahnya. Dalam hal ini, karena kita bergerak setengah lingkaran berarti melakukan penyapuan 180 derajat, sehingga 180 adalah argumen terakhir bentuk lengkung. Harus diingat bahwa arah sapuan 180 derajat adalah melawan arah jarum jam, yang diasumsikan arah positif dalam java. Bila kita bermaksud membuat bentuk lengkung C yang berbalikan arah maka kita menyapu180 derajat ke arah negatif sehingga argumen terakhir menjadi -180 derajat. Gambar berikut ini akan menjelaskan uraian diatas.



**Gambar 14**

Arc dalam lingkaran

**Contoh :**

import java.awt.Graphics;

public class MyOval extends java.applet.Applet {

public void paint(Graphics g) {

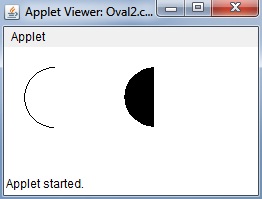
g.drawArc(20,20,60,60,90,180);

g.fillArc(120,20,60,60,90,180);

}

}

**Hasil :**



**Gambar 15**

Bidang arc membentuk huruf C (kiri) dan bidang arc dengan fill

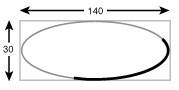
Lingkaran merupakan sarana yang mudah untuk menvisualkan lengkung pada lingkaran. Lengkung pada ellips sedikit lebih sulit.

**arc-elips1**

**Gambar 16**

Bidang ellips dalam arc (lengkung)

Bagian dari ellips membentuk lengkung yang kita sebut lengkung epileptik. Dengan menyempurnakan bentuk oval di mana lengkung merupakan bagiannya, kita akan mendapatkan titik awal serta lebar dan tinggi dalam argumen untuk metode drawArc() atau fillArc()

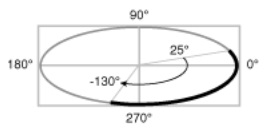


**Gambar 16**

Bentuk elips sebagai dasar

dari bentuk lengkung

Berdasar pada bentuk elips pada gambar diatas, kita dapat membuat bentuk lengkung dengan menentukan sudut mulai dan sudut penyapuannya.



**Gambar 17**

Derajat pembentuk elips

Seperti yang dilihat pada gambar diatas, lengkung dimulai dari 25 derajat. Kemudian menyapu ke arah jarum jam kira-kira 130 derajat. Setelah angka-angka sebagai argumen ditentukan, kita baru dapat menulis kode.

**Contoh :**

import java.awt.Graphics;

public class MyOval extends java.applet.Applet {

public void paint(Graphics g) {

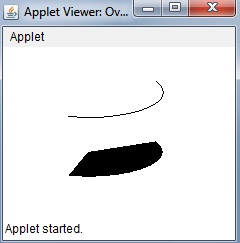
g.drawArc(10,20,150,50,25,-130);

g.fillArc(10,80,150,50,25,-130);

}

}

**Hasil :**



**Gambar 18**

2 buah bidang lengkung eliptikal

Untuk memudahkan langkah membuat bentuk lengkung, berikut ini ringkasan dari apa sebelumnya telah dijelaskan.

1. Asumsikan bahwa lengkung adalah potongan dari bentuk oval penuh.
2. Konstruksilah bentuk oval penuh dengan titik awal serta lebar dan tinggi.
3. Tentukan sudut mulai untuk titik awal lengkung.
4. Tentukan jauhnya sapuan bentuk lengkung serta arahnya Ilawan arah jarum jam artinya arah positif, searah jarum jam artinya arah negatif).

**3.7 Membangun Bidang Geometri Sederhana**

Berikut ini adalah contoh dari sebuah applet yang menggunakan banyak bentuk dari grafik primitif dengan bentuk-bentuk dasar.

1: import java.awt.\*;

2:

3: public class Lamp extends java.applet.Applet {

4:

5: public void paint(Graphics g) {

6: // platform dari objek

7: g.fillRect(0,250,290,290);

8:

9: // membetuk dasar dari objek

10: g.drawLine(125,250,125,160);

11: g.drawLine(175,250,175,160);

12:

13: // bentuk dari objek, tepi atas dan bawah

14: g.drawArc(85,157,130,50,-65,312);

15: g.drawArc(85,87,130,50,62,58);

16:

17: // tirai pada objek, sisi objek

18: g.drawLine(85,177,119,89);

19: g.drawLine(215,177,181,89);

20:

21: // bintik pada tirai objek

22: g.fillArc(78,120,40,40,63,-174);

23: g.fillOval(120,96,40,40);

24: g.fillArc(173,100,40,40,110,180);

25: }

26: }

**IV**

**TEKS & HURUF**

Dengan menggunakan kelas Graphics, kita dapat juga mencetak teks ke layar, dalam kaitannya dengan kelas Font (yang kadang-kadang disebut kelas FontMetrics). Kelas Font mewakili font tertentu – nama, gaya dan ukurannya. – sedangkan FontMetrics memberikan kita informasi tentang font tersebut (misalnya, tinggi atau lebar aktual dari karakter tertentu) sehingga kita dapat secara tepat mengatur teks dalam applet. Perlu diingat bahwa proses penggambaran teks disini dilakukan sekali dan dimaksudkan untuk tetap tampil dilayar.

4.1 Objek Font

Untuk menggambat teks ke layar, mula-mula kita perlu membuat *instance* dari kelas Font. Objek font mewakili font individual – yakni nama, gaya (bold, italic, plain) dan ukurannya. Nama font adalah string yang mewakili kelaurga font, misalnya TimesRoman, Courier atau Helvetica. Gaya font adalah konstanta yang ditentukan oleh kelas Font. Kita dapat mengambil daya font ini dengan menggunakan variabel kelas – misalnya Font.PLAIN, Font.BOLD, Font.ITALIC. terakhir adalah ukuran font yang ditentukan oleh font itu sendiri, bisa merupakan tinggi karakter, bisa juga bukan.

Diperlukan tiga argumen untuk konstruktor new dari kelas Font untuk membuat obejk font individual.

Font f = new Font (“TimesRoman”, Font.BOLD, 24);

Perintah diatas akan membuat objek font untuk font TimesRoman BOLD, dengan ukuran 24 titik. Harap diingat, sebagaimana kebanyakan kelas Java, kita harus mengimpor kelas java.awt.Font sebelum menggunakannya.

Font yang tersedia dalam applet bergantung pada font yang diinstal pada sistem dimana applet bekerja. Jika mengambil font untuk applet dan font ini tidak teredia pada sistem yang menjalankan applet, maka Java akan menggantinya dengan font *default* (biasanya courier). Kita dapat mengambil larik nama font yang tersedia dalam sisitem dengan perintah berikut ini :

String[] fontslist = this.getToolkit().getFontList();

Dari daftar tersebut kita dapat menentukan font yang akan kita gunakan dalam applet. Untuk hasil yang terbaik disarankan menggunakan font standar, misalnya TimesRoman, Helvetica, dan Courier.

4.2 Karakter dan String

Dengan objek font kita dapt menggambar teks pada layar dengan menggunakan metode drawChars() dan DrawString(). Pertama kita perlu mengatur font ke objek kita dengan menggunakan metode setFont().

Font kini adalah bagian dari grafis yang senantiasa dilacak oleh objek Graphics yang kita gunakan untuk menggambar. Setiap kali kita menggambar karakter atau string pada layar, Java menggambar teks tersebut pada font kini. Untuk mengubah font teks, kita harus mengubah font kini terlebih dahulu. Metode Paint() berikut ini menciptakan font baru, mengatur font kini ke font tersebut, dan menggambar string “ini font besar”, pada titik 10,100.

public void paint(Graphics g) {

Font f = new Font("TimesRoman", Font.PLAIN, 72);

g.setFont(f);

g.drawString("ini font besar.", 10, 100);

}

Dua argumen berikutnya dalam drawString() adalah menentukan titik dimana string akan dimulai. Nilai x adalah awal dari ujung kiri teks, y adalah garis dasar untuk seluruh string. Mirip dengan drawString() adalah metode drawChars(), yang menerima larik karakter sebagai argumen, metode drawChars() memiliki 5 argumen : larik karakter, integer yang mewakili karakter pertama dalam larik yang harus digambar, integer lain untuk karakter akhir dalam larik yang harus digambar (seluruh karakter antara yang pertama dan yang terakhir digambar), dan x dan y untuk titik awal. Dalam banyak keperluan, drawString() lebih bermanfaat dibandingkan drawChars().

Contoh:

import java.awt.Font;

import java.awt.Graphics;

public class ManyFonts extends java.applet.Applet {

public void paint(Graphics g) {

Font f = new Font("TimesRoman", Font.PLAIN, 18);

Font fb = new Font("TimesRoman", Font.BOLD, 18);

Font fi = new Font("TimesRoman", Font.ITALIC, 18);

Font fbi = new Font("TimesRoman",Font.BOLD+Font.ITALIC,18);

g.setFont(f);

g.drawString("This is a plain font", 10, 25);

g.setFont(fb);

g.drawString("This is a bold font", 10, 50);

g.setFont(fi);

g.drawString("This is an italic font", 10, 75);

g.setFont(fbi);

g.drawString("This is a bold italic font", 10, 100);

}

}



Gambar 19

Jenis-jenis gaya huruf

4.3 Metode Font

Kadang-kadang kita ingin dalam program ada pengambilan keputusan berdasar kualitas font kini – misalnya ukuran atau tinggi karakter. Informasi dasar tentang font dan objek font dapat kita peroleh dengan menggunakan metode-metode yang ada pada objek Graphics dan objek Font.

Tabel 1. Metode dalam kelas Graphics dan Objek Font

|  |  |  |
| --- | --- | --- |
| Nama Metode | Dalam Objek | Aksi |
| GetFamily() | Font | Mengembalikan nama famili font |
| GetFont() | Graphics | Mengembalikan objek font kini seperti yang telah diatur oleh setFont() |
| GetName() | Font | Mengembalikan nama font sebagai string |
| GetSize() | Font | Mengembalikan ukuran font kini (integer) |
| GetStyle | Font | Mengembalikan gaya font kini (gaya merupakan konstanta integer : 0 untuk palin, 1 untuk bold, 2 untuk italic, dan 3 untuk bold italic) |
| IsPlain() | Font | Mengembalikan nilai true atau false bila gaya font adalah plain |
| IsBold() | Font | Mengembalikan nilai true atau false bila gaya font adalah bold |
| IsItalic() | Font | Mengembalikan nilai true atau false bila gaya font adalah italic |

Bila kita menginginkan informas lebih detil tentang kualitas font kini (misalnya lebar dan tinggi karakter tertentu) kita harus bekerja dengan metrik font. Kelas FontMetrics memberikan informasi khusus untuk font tertentu : jarak antar baris, lebar dan tinggi tiap karakter, dan sebagaimananya. Agar dapat bekerja dengan nilai-nilai itu kita perlu membuat objek FontMetric berdasar pada font kini dengan menggunakan metode applet getFontMetric() :

Font f = new Font (“TimesRoman:, FONT.BOLD, 36);

FontMetrics fmetrics = getFontMetrics (f);

g.setFont(f);

Pada tabel berikut dibawah ini, akan menunjukkan beberapa hal yang bisa diperoleh dengan menggunakan font metrics. Semua metode yang digunakan harus dipanggil bersama dengan objek FontMetrics.

Tabel 2. Metode Font metrics

|  |  |
| --- | --- |
| Nama Metode | Aksi |
| stringWidth(string) | Mengembalikan nama famili font |
| charWidth(char) | Diberikan karakter, dikembalikan lebar karakter tersebut |
| getAscent() | Mengembalikan panjang-naik (ascent) font, yakni jarak antara garis dasar font dengan pucak karakter. |
| getDescent() | Mengembalikan panjang-turun (descent) font, yakni jarak antara garis dasar font dengan dasar karakter (misalnya untuk karakter p dan q yang melewati garis dasar) |
| getLeading() | Mengembalikan jarak antara panjang-turun (descent) suatu baris dengan panjang-naik (ascent) baris berikutnya (leading) |
| getHeight() | Mengembalikan tinggi total font yang merupakan jumlah panjang-naik (ascent), panjang-turun (descent), dan leading. |

Contoh Menampilkan String terformat dalam Grafis:

**import** java.applet.\*;

**import** java.awt.\*;

**public** **class** DrawText **extends** Applet {

**public** **void** paint(**Graphics** g) {

Font font = **new** Font("Helvetica", Font.BOLD + Font.ITALIC,

22);

FontMetrics fm = g.getFontMetrics(font);

**String** str = **new** **String**("The highest result of education

is..");

g.setFont(font);

g.drawString(str, (size().width - fm.stringWidth(str)) / 2,

((size().height - fm.getHeight())

/ 2) + fm.getAscent());

} }

**V**

**WARNA**

Java menyediakan metode-metode dan perilaku untuk berurusan dengan warna secara umum melalui kelas Color, dan juga menyediakan metode untuk mengatur warna latar depan dan latar belakang.

Warna dalan java, menggunakan standar 24-bit, dimana warna diwakili oleh konbinasi merah, hijau dan biru. *Setiap komponen warna dapat memiliki nilai antara 0 dan 255. Nilai 0,0,0 adalah warna hitam, 255,255,255 adalah warna putih*. Ini berarti secara teoritis ada jutaan variasi warna yang dapat dihasilkan.

Model warna abstrak Java dipetakan ke model warna dimana platform Java berjalan, biasanya hanya 256 warna atau kurang yang dapat dipilih. Jika warna yang diminta dalam objek warna tidak tersedia, warna akan dipetakan kedalam warna lain, tergantung bagaimana browser melihat warna tersebut dan platform apa yang digunakan. Dengan kata lain, meskipun Java mampu mengelola jutaan warna, hanya sedikit yang dalam praktiknya bisa didapatkan.

**5.1 Pemanfaatan Objek Warna**

Untuk menggambar objek dengan warna tertentu, kita harus membuat *instance* dari kelas Color untuk mewakili warna itu. Kelas Color menentukan seperangkat objek warna standar, yang disimpan dalam variabel kelas, untuk dengan cepat mengambil objek warna. Sebagai contoh, Color.red mengembalikan objek Color yang mewakili *warna* *merah (nilai RGB 255,0,0)*, Color.white mengembalikan *warna putih (nilai RGB 255,255,255)*, dan seterusnya.

**Tabel 3.** Warna-warna standar

|  |  |
| --- | --- |
| **Color Name** | **RGB Value** |
| Color.white | 255,255,255 |
| Color.black | 0,0,0 |
| Color.lightGray | 192,192,192 |
| Color.gray | 128,128,128 |
| Color.darkGray | 64,64,64 |
| Color.red | 255,0,0 |
| Color.green | 0,255,0 |
| Color.blue | 0,0,255 |
| Color.yellow | 255,255,0 |
| Color.magenta | 255,0,255 |
| Color.cyan | 0,255,255 |
| Color.pink | 255,175,175 |
| Color.orange | 255,200,0 |

Jika warna yang ingin digambarkan tidak ada dalam objek Color standar, kita dapat membuat objek warna dengan kombinasi merah, hijau, biru (RBG), asalkan kita memiliki nilai RGB dari warna yang kita inginkan. Kita hanya perlu membuat objek warna baru, misalnya :

Color c = new Color (140,140,140);

Baris perintah di atas akan menghasilkan warna abu-abu tua. Kita dapat membuat warna apa saja dari gabungan nilai RGB. Sebagai alternatif, kita dapat membuat objek warna dengan menggunakan bilangan float dari 0,0 sampai 1, misalnya :

Color c = new Color(0.55, 0.55, 0.55);

**5.2 Pemeriksaan Dan Pengaturan Warna**

Untuk menggambar objek atau teks dengan menggunakan objek warna, kita harus mengatur warna kini menjadi warna objek. Metode setColor() (metode dari objek Graphics) dapat digunakan untuk keperluan itu.

g.setColor(Color.green);

Setelah mengatur warna kini, maka seluruh operasi penggambaran akan berlangsung dalam warna tersebut.

Selain pengaturan warna kini untuk grafik, kita dapat juga mengatur warna latar belakang dan latar depan applet dengan menggunakan metode setBackGround() dan setForeGround(). Kedua metode ini di definisikan dalam kelas **java.awt.Component**, dimana applet – dan juga kelas yang kita buat – secara otomatis menjadi pewarisnya. Untuk mengatur warna latar belakang applet, kita menggunakan metode setBackGround() yang nilai defaultnya adalah abu-abu terang (agar sesuai dengan latar belakang browser). Metode tersebut menerima satu argumen, yakni objek Color.

setBackGround(Color.white)

Metode setForeGround() juga menerima warna tunggal sebagai argumen dan akan mempengaruhi apa pun yang telah digambar. Kita dapat menggunakan setForeGround() untuk mengubah warna segala bentuk yang ada dalam applet sekaligus, tidak dengan menggambar ulang satu persatu.

setForeGround(Color.black);

Selain metode setColor(), setForeGround(), dan setBackGround() ada metode get yang memungkinkan kita menerima warna grafik, latar belakang, atau latar depan yang sedang aktif. Metode-metode itu adalah :

(1) getColor() 🡪 didefinisikan dalam objek Graphics,

(2) getForeGround() 🡪 didefinisikan dalam applet,

(3) getBackGround() 🡪 didefinisikan dalam applet

Kita dapat menggunakan metode-metode tersebut untuk memilih warna berdasar pada warna yang ada pada applet.

setForeGround(g.setColor());

Contoh berikut dibawah ini, adalah kode sumber untuk membuat kotak-kotak bujur sangkar yang diberi warna secara random, sehingga setiap kali di refresh kotak yang sama akan memiliki yang (bisa) berbeda. Program ini dibuat sedemikian rupa sehingga mampu secara otomatis mengisi bidang applet dengan jumlah kotak yang tepat.

1: import java.awt.Graphics;

2: import java.awt.Color;

3:

4: public class ColorBoxes extends java.applet.Applet {

5: public void paint(Graphics g) {

6: int rval, gval, bval;

7: for (int j = 30; j < (size().height -25); j += 30)

8: for (int i = 5; i < (size().width -25); i += 30) {

9: rval = (int)Math.floor(Math.random() \* 256);

10: gval = (int)Math.floor(Math.random() \* 256);

11: bval = (int)Math.floor(Math.random() \* 256);

12: g.setColor(new Color(rval,gval,bval));

13: g.fillRect(i, j, 25, 25);

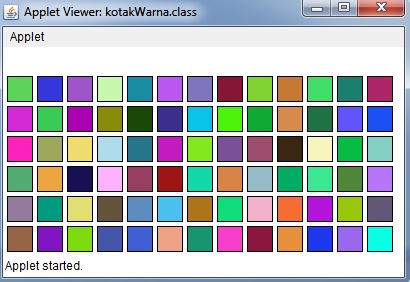
14: g.setColor(Color.black);

15: g.drawRect(i-1, j-1, 25, 25);

16: }

17: }

18: }



**Gambar 20**

Kotak warna-warni

*Catatan : dua buah loop for menjadi jantung dari program diatas, yang pertama menggambar baris, dan kedua menggambar kotak-kotak dalam tiap baris. Saat kotak digambar, warna acak dihitung dahulu kemudian kotaknya digambar. Garis rangka hitam digunakan pada tiap-tiap kotak.*

**VI**

**PEMROGRAMAN GRAFIS JAVA 2D**

Application Programming Interface (API) pada Java ada kesatuan dari kelas-kelas yang dapat digunakan untuk menciptakan grafis dengan kualitas tinggi. Java 2D adalah termasuk bagian utama dari kelas-kelas dalam platform Java 2. Dalam java 2D memiliki beberapa paket-paket, yaitu :

* java.awt
* java.awt.image
* java.awt.color
* java.awt.font
* java.awt.geom
* java.awt.print
* java.awt.image.renderable
* com.sun.image.codec.jpeg

**5.1 Apa saja yang dapat Java 2D lakukan ?**

Java 2D dirancang untuk dapat melakukan apa saja yang anda inginkan (dengan komputer grafik tentunya). Tetapi pada versi awal Java 2D sebelumnya, penanganan grafis AWT (Abstract Windowing Toolkit) menemui beberapa batasan yang serius, diantaranya :

* Semua garis digambar dengan ketebalan satu piksel
* Ketersediaan *font* yang terbatas
* AWT tidak menawarkan banyak kontrol atas gambar. Sebagai contoh, Anda tidak bisa memanipulasi bentuk karakter individu.
* Jika Anda ingin memutar atau skala apa pun, Anda harus melakukannya sendiri.
* Jika Anda ingin *fill* khusus, seperti *gradien* atau *patterns*, Anda harus membuat mereka sendiri.
* Dukungan gambar yang sempurna.
* Pengendalian transparansi yang canggung.

Dan sebagai solusinya, maka pada versi Java 2D yang sekarang ini telah menyediakan kemudahan untuk menciptakan beberapa objek grafis, seperti :

**Shapes;** Bentuk-bentuk geometris dengan baik dapat diwakili oleh kombinasi garis lurus dan kurva. API 2D ini juga menyediakan *tools* yang berguna dari bentuk standar, seperti persegi panjang, busur, dan elips.

**Stroking;** Garis dan bentuk garis dapat digambar sebagai garis padat atau putus-putus dengan lebar yang bermacam-macam, proses ini disebut sebagai *stroking*. Anda dapat mendefinisikan pola grais-bertitik dan menentukan bagaimana bentuk dari sudut dan bagaimana seharusnya garis akhir itu ditarik(dibentuk).

**Filling;** suatu bentuk dalam diisi menggunakan warna yang solid, dalam bentuk yang berpola, warna gradient, atau hal lainnya yang mungkin ada bayangkan.

**Transformation;** apapun yang digambar dengan API 2D di rentangkan, dirapatkan dan diputar. API ini dapat membentuk *shapes*, text, dan image. Anda menyiapkan transformasi seperti apa yang Anda inginkan selanjutnya 2D API ini akan menangani semuanya.

**Alpha compositing;** adalah proses penambahan elemen-elemen baru pada operasi menggambar. API 2D akan memberikan fleksibilitas yang cukup dengan menggunakan aturan komposit Porter-Duff

**Clipping;** adalah proses membatasi jangkauan operasi menggambar. Sebagai contoh, menggambar pada suatu *window* biasanya dijepit ke batas-batas window. Dalam 2D API, bagaimanapun, Anda dapat menggunakan bentuk apapun untuk *clipping*.

**Antialiasing;** adalah suatu tehnik yang mengurangi *jagged* di tiap tepi dari gambar.

**Text;** API 2D menggunakan font *TrueType* atau *TrueType 1* yang terinstall di sistem. Anda dapat membuat string, mengambil bentuk string individu atau beberapa huruf, dan memanipulasi teks dalam cara yang sama dengan suatu *shape* dimanipulasi.

**Color;** sulit untuk menunjukkan warna dengan benar. API 2D termasuk kelas dan metode yang dengan caranya sendiri mendukung wujud dari warna, tidak bergantung pada perangkat keras tertentu atau kondisi dari perangkat tampilan.

**Images;** dukungan API 2D dapat melakukan hal-hak apik seperti halnya membentuk suatu *shape* dan teks. Khususnya, Anda dapat mengubah suatu *images*, menggunakan *clipping shapes*, dan menggunakan *shape compositing* pada *image*. API 2D dalam Java 2, juga mencakup satu set kelas untuk membuka dan menyimpan gambar (dalam format JPEG).

**Dimana Bisa Mendapatkan Grafis 2D ?**

*Shapes*, teks, dan *Images*, semuanya digambar dengan menggunakan objek Graphics2D. Tetapi yang menjadi pertanyaannya, darimana Graphics2D itu berasal?

1. **Drawing on Components**

Setiap Component yang ditampilkan AWT dilayar memiliki metode paint(). Dalam JDK 1.1 maupun yang terbaru, Components dapat digambar dengan meng-overriding metode paint() dan menggunakan kelas Graphics untuk menggambar berbagai bentuk.

Yang membedakan proses kerja grafik Java 2 dengan Java 2D adalah pada Java 2D tersedia kelas Graphics2D yang *dilempar* ke dalam metode paint(). Untuk dapat menggunakan seluruh fitur-fitur dalam Java 2D, haruslah melakukan proses *casting* didalam metode paint(), seperti contoh berikut ini :

public void paint(Graphics g)

{

Graphics2D g2 = (Graphics2D) g;

}

*Catatan : apabila ingin bekerja dengan component Swing, Anda harus dapat mengimplementasikan metode paintComponent() bukan paint(). Swing menggunakan metode paint() untuk mengurai component. Implementasi Swing dari metode paint() memanggil metode paintComponent() untuk menggambar componen itu sendiri. Anda bisa saja mengabaikan metode paintComponent() dan memakai metode paint(), dengan begitu maka resiko yang mungkin didapati adalah objek (outline) tidak digambarkan dengan baik.*

1. **Drawing on Images**

Anda dapat menggunakan kelas Graphics atau Graphics2D untuk menggambar pada *Image*. *Image* yang diciptakan sendiri, dialamatkan oleh Graphics dengan memangil metode createGraphics(), seperti :

public void drawOnImage(Image i)

{

Graphics g = i.getGraphics();

// Now draw on the image using g.

}

Listing ini hanya dapat dijalankan untuk semua gambar yang diciptakan, bukan untuk Image yang di-*load* dari file.

Apabila ingin menggunakan BufferedImage(), akan diperoleh dengan menggunakan Graphics2D(), seperti :

public void drawOnBufferedImage(BufferedImage bi)

{

Graphics2D g2 = bi.createGraphics();

// Now draw on the image using g2.

}

**5.2 Graphics2D**

Rendering adalah suatu proses mengambil beberapa kumpulan dari *shapes*, teks dan images dan mengatur bagaimana tampilannya muncul dilayar atau saat dicetak. *Shapes*, teks dan image disebut juga *graphics primitives*; layar dan printer disebut sebagai *ouput devices*. kinerja dari *rendering engine* menunjukkan; pada 2D API, kelas Graphics2D adalah *rendering engine*-nya. Seperti gambah dibawah ini, menunjukkan bahwa proses ini ada pada bagian teratas. *Rendering engine* 2D ini mengatur bagaimana setiap detail dari perangkat yang digunakan dan dengan akurat memproduksi penggambaran suatu geometri dan pewarnaan, tanpa terpengaruh oleh perangkat yang menampilkan geometri dan warna.



**Gambar 21.** Proses Rendering

Terpisah dari bagian sebagai *rendering engine*, Graphics2D juga mewakili *drawing surface*, dimana hal ini merupakan kumpulan dari piksel, tiap piksel akan dapat diisi dengan warna. *Drawing surface* bisa menjadi bagian dari tampilan di window atau halaman yang tercetak oleh printer. Setiap saat anda menggambar sesuatu yang baru, maka elemen baru tersebut ditambahkan kedalam gambar yang ada melalui Graphics2D.

*Rendering pipeline* disebut juga beberapa operasi penggambaran dimana grafik primitif dapat langsung menuju ke *rendering engine*. Gambar 21 dibawah ini akan menunjukkan bagaimana tujuh elemen dari *internal state* pada Graphics2D digunakan didalam rendering pipeline. Adapun dalam gambar 21 menunjukkan empat operasi dasar dari Graphics2D :

* Anda dapat mengisi *shape* dengan memasukkan metode fill(). Dalam API 2D, yang dimaksud dengan *shape* adalah implementasi dari paket java.awt.Shape
* Anda dapat menggambak bentuk-bentuk dari dengan memanggil metode draw().
* Ilustrasi teks dapat dilakukan dengan memanggil metode drawString() dari kelas Graphics2D
* Anda dapat menggambar suatu Image dengan memasukkan paket java.awt.Image kedalam metode drawImage().



**Gambar 21.** Rincian *rendering pipeline*

**5.2.1 Geometri**

Java 2D memnginjinkan kita untuk memciptakan bentuk apa saja sebagai kombinasi dari segmentasi gais lurus atau kurva. Beberapa topik yang akan di bahas disini adalah : (1) kelas-kelas yang akan membentuk *point* (titik); (2) dua antarmuka utama untuk bentuk geometry : Shape dan PahIterator; (3) alat bantu pembentuk 2D dari paket java.awt.geom; (4) dukungan 2D untuk mengkombinasi *shape* dengan yang lainnya.

**Points**

Kelas pada paket java.awt.geom.Point2D mengenkapsulasikan satu titik (x dan y) di area kerja user. Ini adalah kelas paling dasar dari Java 2D dan digunakan bersam dengan API. Sebuah point, tidak meimiliki area, jadi tidak dapat di-*render.* Points digunakan untuk membentuk empat persegi panjang atau *shape* (bentuk) atau lainnya yang mempunyai area dan dapat di-*render.*

**

**Gambar 22.** Family tree of Point2D classes

*Point2D mewakili bentuk titik pada area user, tetapi tidak menspesifikasikan bagaimana titik koordinat disimpan. Sub-kelasnya menyediakan tingkat ketelitian yang berbeda untuk menyimpan titik-titik koordinat.*

**Shape**

Seperti yang sudah dibahas sebelumnya, bahwa Graphics2D adalah rendering engine untuk API Java 2D. Dua operasi dasarnya yaitu *filling shapes* dan menggambar suatu garis. Tetapi Graphics2D tidak terlalu banyak menangani operasi geometri. Karena pada faktanya, Graphics2D hanya dapat melakukan satu operasi saja : java.awt.Shape. Shape ini merupakan antarmuka yang menampung bentuk dari geometri, salah satunya menangani pembuatan garis dan lainnya mengatur interior. Dengan Graphics2D, dapat menggambar bagian tepi dari suatu bentuk menggunakan draw() dan juga dapat melakukan proses fill didalam *shape* menggunakan fill().

Paket java.awt.Geom adalah toolbox dari kelas-kelas yang sangat membantu implementasi dari antarmuka Shape. Adapun kelas-kelas didalamnya yang membentuk *ellipses, arcs, rectangles* dan *lines*.



**Gambar 23.** Family tree dari *Shape*

Semua kelas-kelas geometrik di Java2D menjalankan operasi pada antarmuka Shape. hal ini berarti bahwa kelas-kelas tersebut dalam dimasukkan langsung ke dalam metode-metode dalam Graphics2D 🡪 draw() dan paint().

**Catatan:**

Untuk diingat, bahwa kelas Graphics2D memiliki metode untuk operasi *drawing* dan *filling* yang sederhana: drawRect(), drawOval(), fillRect(), fillOval(), fillArc(), dll. Karena Graphics2D adalah sub-kelas dari Graphics, Anda masih bisa memanggil metode ini untuk me-*render* suatu *shape*. Dalam beberapa kasus, dengan memanggil salah satu dari metode diatas akan memudahkan operasi menggambar, karena Anda dapat melakukan proses render dari suatu shape dalam satu langkah saja.

Disisi lain, metode-metode ini hanya menggunakan integer untuk koordinat-nya, dan metode-metode ini tidak memperbolehkan shape tersebut dipakai kembali.

**5.2.2 Lines**

Kelas java.awt.geom.Line2D membentuk sebuah garis yang dimana koordinat-koordinatnya dapat diterima/benilai doubles. Seperti halnya Point2D, Line2D adalah abstrak. Sub-kelasnya dapat menyimpan koordinat dengan cara apa saja yang diinginkan.



**Gambar 23.** Hirarki Line2D

Kelas Line2D mewakili segment koordinat dari suatu garis. Sub-kelas dari Line2D.Float dan Line2d.Double menentukan presisi garis dengan nilai *float* dan *double*.

//draw Line2D.Double

g2.draw(new Line2D.Double(x1,y1,x1,y2));

Pada kelas ini sudah termasuk metode setLine() yang menjelaskan titik-akhir dari suatu garis. Alternalif lainnya, titik-akhir dari garis dapat dispesifikasikan dengan menggunakan konstruktor dari kelas Line2D.Float seperti :

• Line2D.Float(float X1, float Y1, float X2, float Y2)

• Line2D.Float(Point2D p1, point p2)

**5.2.3 Curve**

Paket java.awt.geom memperbolehkan untuk menciptakan suatu *quadratic* atau *cubic curve*.

**Quadratic Curve Segmen**

Kelas Quadcurve2D adalah mengimplementasikan antarmuka dari Shape. Kelas ini mengekspresikan segmen parametrik kurva quadratik pada koordinat (x,y). Sub-kelas dari QuadCurve2D.Float dan QuadCurve2D.Double secara presisi menentukan bentuk kurva quadratik dalam *float* dan *double*.

Beberapa metode setCurve() digunakan untuk menentukan 2 titik-akhir dan titik-kendali dari kurva, yang koordinatnya dapat didefinisikan secara langsung oleh titik-titik yang lain dan dengan menggunakan array yang diberikan. Sebuah metode yang berguna, setCurve(QuadCurve2D c), menge-set kurva quadratic dengan titik-akhir yang sama dan titik-kontrol sebagai kurva yang tersediakan. Sebagai contoh :

*//membuat objek untuk Quadratic2D.Float*

QuadCurve2D q = new QuaadCurve2D.Float();

//menggambar QuadCurve2D.Float dengan satu set koordinat

q.setCurve(x1,y1,ctrlx, ctrly, x2, y2);

g2.draw(q);

**Gambar 24.** Garis QuadCurve2D

**Cubic Curve Segment**

Kelas dari CubicCurve2D juga mengimplementasikan antarmuka dari Shape. Kelas ini merupakan representasi dari segmentasi dari kurva kubik-parametrik dalam ruang koordinat (x,y). Sub-kelas dari CubicCurve2D.Float dan CubicCurve2D.Double menentukan keakuratan kurva kubik dalam nilai *float* dan *double*.

Kelas CubicCurve2D memiliki metode yang sama untuk mengatur kurva sebagai kelas QuadraticCurve2D, selain dengan titik kontrol kedua. Sebagai contoh :

*// membuat objek baru dari CubicCurve2D.Double*

CubicCurve2D c = new CubicCurve2D.Double();

*// menggambar CubicCurve2D.Double dengan satu set koordinat*

c.setCurve(x1, y1, ctrlx1, ctrly1, ctrlx2, ctrly2, x2, y2);

g2.draw(c);



**Gambar 25.** Garis QuadCurve2D

**5.2.4 Rectangles**

****

**Gambar 26.** Family tree of Rectangle2D

Kelas-kelas yang merujuk kepada bentuk primitif diwakilkan dalam contoh adalah turunan dari kelas RectangularShape, yang mengimplementasikan antarmuka *Shape* dan memasukkan beberapa metode-nya sendiri.

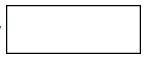
Metode-metode ini membolehkan untuk mendapatkan informasi mengenai lokasi dan ukuran dari *shape*, untuk mengecek titik-pusat dari *rectangle*, dan mengeset titik-batas dari *shape*

**Kelas Rectangle2D** mengilustrasikan sebuah *rectangle* yang didefinisikan dalam suatu *lokasi (x,y)* dan *dimensi (w x h).*

Sub-kelas Rectangle2D.Float dan Rectangle2D.Double menentukan keakurasian sebuah *rectangle* dalam nilai *float* dan *double*. Seperti contoh :

*// menggambar Rectangle2D.Double*

g2.draw(new Rectangle2D.Double(x, y, rectwidth, rectheight));



**Gambar 27.** Rectangle2D

**Kelas RoundRectagle2D** mengilustrasikan sebuah rectangle dengan sudut melengkung yang didefinisikan dengan *lokasi (x,y), dimensi ( w x h) dan dalam lebar dan tinggi dari sudut berbentuk arc.*

Sub-kelas RoundRectangle2D.Float dan RoundRectangle2D.Double membentuk keakurasian sebuah *rectangle* dengan sudut melengkung dalam nilai *float* dan *double.*

Rounded rectangle dispesifikasikan dengan parameter dibawah ini :

• Lokasi

• Lebar

• Tinggi

• Lebar dari sudut berbentuk *arc*

• Tinggi dari sudut berbentuk *arc*

Untuk mengeset lokasi, ukuran, dan objek berbentuk arc dari RoundRectangle2D, gunakan metode setRoundRect(double a, double y, double w, double h, double arcWidth, double arcHeight). Sebagai contoh :

*// draw RoundRectangle2D.Double*

g2.draw(new RoundRectangle2D.Double(x, y, rectwidth,

rectheight,10,10));

**Gambar 28.** RoundRectangle

**5.2.5. Ellipses**

Kelas Ellipse2D membentuk sebuah ellipse yang didefinisikan melalui sebuah rectangle yang membulat.

Sub-kelas Ellipse2D.float dan Ellipse2d.Double didefinisikan dalam nilai float dan double dengan akurat.

Ellips secara penuh didefinisikan oleh lokasi, lebar dan tinggi. Sebagai contoh :

*// draw Ellipse2D.Double*

g2.draw(new Ellipse2D.Double(x, y, rectwidth,

 rectheight));

**Gambar 29.** Ellips2D

**5.2.6 Arcs**

Untuk menggambar sebuah *ellips*, gunakanlah kelas Arc2D. Kelas ini mewakili sebuah *arc* yang didefinisikan sebagai sebuah kotak (*rectangle*) dengan sudut yang melengkung, sudut-awal, luas-sudut, type-tutupan. Sub-kelas Arc2D.Float dan Arc2D.Double menspesifikasikan keakurasian sebuah ellips dalam nilai *float* dan *double*.

Kelas Arc2D didefinisikan dalam tiga tipe dari arcs, direpresentasikan oleh konstanta yang berkaitan dalam kelas ini adalah : OPEN, PIE dan CHORD

Beberapa metode mengeset ukuran dan parameter dari arc :

• Secara langsung, melalui koordinat

• Dengan menyuplai Point2D dan Dimension2D

• Dengan meng-copy Arc2D yang ada.

Selain itu juga, Anda dapat menggunakan metode setArcByCenter untuk menspesifikasikan sebuah arc dari titik-tengah, yang diberikan oleh koordinat itu sendiri dan sebuah radius.

// draw Arc2D.Double

g2.draw(new Arc2D.Double(x, y, rectwidth, rectheight, 90, 135,

Arc2D.OPEN));



**Gambar 30.** Arc2D

**VII**

**AREA KONTRUKSI GEOMETRI**

Area kontruksi geometri adalah penamaan untuk pengabungan shapes. Dua shape atau lebih dapat dikombinasikan menggunakan aturan atau operasi yang berbeda, banyak sejumlah cara yang dapat dikombinasikan dalam sebuah persamaan. API 2D mendukung empat operasi yang berbeda untuk mengkombinasikan area dari dua *shapes* : **(1) Union,** penambahan dari dua *shapes* dari suatu area yang tertampil oleh salah satu atau kedua *shape* secara bersamaan; **(2) Intersection,** sebuah titik potong dari dua *shapes*, adalah sebuah area yang mencakupi oleh kedua *shape* secara simultan; **(3) Subtraction,** hasil mengurangkan satu *shape* dari *shape* yang lain adalah daerah yang dicakup oleh salah satu *shape* yang tidak tercakup oleh *shape* yang lain; **(4) Exclusive or,** sebuah exclusive or operator adalah kebalikan dari operator perpotongan. Dengan kata lain, sebuah *exclusive or* dari dua *shapes* merupakan area yang dicakupi oleh salah satu *shapes* atau *shapes* lainnya. Akan tetapi operasi ini tidak termasuk area yang dicakupi oleh keduanya.



**Gambar 31.** Bentuk-bentuk kombinasi dari dua shapes

Tidak mudah untuk menvisualisasikan operator-operator yang ada hanya dengan membaca. Gambar 31 menunjukkan hasil dari meng-*apply* operator-operator ini kedalam dua *overlapping shapes*.

Di dalam Java 2D, kelas java.awt.geom.Area mendukung area konstruksi geometri. Pada kelas ini memberikan dua konstruktor :

public Area()

Konstruktor ini membuat area yang kosong. Anda dapat mengakumulasikan sebuah area menggunakan metode add().

public Area(Shape g)

konstruktor ini membuat sebuah Area menggunakan interior (bagian isi) dari shape yang telah di suplai.

Adapun empat operator dari area yang terimplementasi sebagai metode dari Area :

public void add(Area rhs)

gunakan metode ini untuk menambahkan area yg diberikan, rhs, ke Area tesebut.

public void intersect(Area rhs)

metode ini memodifikasi Area ini untuk merepresentasikan proses pemotongan dari bentuk shape awal dan area yang tersuplai.

public void subtract(Area rhs)

gunakan metode ini untuk mengurangi bagian area yang direpresentasikan oleh rhs dari Area ini.

public void excusiveOr(Area rhs)

metode ini memodifikasi Area sehingga metode ini merepresentasikan sebuh exclusive or dari shape yang asli dan area yang diberikan.

Area itu sendiri mengimplementasikan antar muka dari Shape. Sesaat setelah Area itu dibuat seperti yang diinginkan, Anda dapat menggambarnya, Mengisinya (fill) dan bahkan memanipulasinya seperti halnya Shape lainnya.

**VIII**

**PAINTING AND STROKING**

**7.1 Painting**

Pada pembahasan sebelumnya, kita telah melihat bagaimana bentuk-bentuk dari bidang grafis yang dibangun dengan API 2D. Tetapi apa yang akan anda lakukan dengan bentuk bidang grafis tersebut ?

⮊ *Painting* adalah proses *filling* (pengisian) pada tampilan objek grafis dengan warna, *color gradient* (pencampuran warna), atau *texture* (warna berkontur).

⮊ *Stroking* adalah proses dari penggambaran sebuah garis bidang grafis. Anda dapat menggambar sebuah garis menggunakan lebar garis, jenis garis dan warna garis yang berbeda-beda.

*Painting* dan *stroking* bisa dikatakan berhubungan sangat erat. *Stroking*, pada faktanya, hanyalah suatu proses dari menciptakan sebuah *shape* yang merepresentasikan sebuah garis dan mengisinya. Anda dapat menggambar sebuah garis menggunakan berbagai macam tipe dari *paint*. Perhatikan gambar dibawah ini.



**Gambar 32. Contoh tampilan dari painting dan stroking**

Di sebelah kiri, ada sebuah lingkaran yang telah terisi dengan *gradien* *color*. Ditengah, sebuah lingkaran yang dimana garisnya diberi warna yang penuh (*solid color*) yang membuat telihat tebal. Dikanan, sebuah lingkaran dengan garis yang digambar dalam bentuk garis yang sama hanya saja garis tersebut di beri efek *color gradient*.

Adapun kelas-kelas yang memproduksi gambar diatas menggunakan metode dan kelas seperti : setPaint(), setStroke(), GradientPaint dan BasicStroke.

**7.1.1 Solid Color**

Kelas java.awt.Color diimplementasikan ke antarmuka Paint. Dengan demikian jenis Color apa saja dapat digunakan untuk mengisi bidang dari sebuah *shape*. penanganan dari proses pewarnaan adalah kemungkinan besar yang dapat terus dikembangkan. Pada saat ini, perhatikan kelas-kelas dari Color yang menyertakan beberapa penamaan warna sebagai *variabel static* :

public static final Color white;

public static final Color lightGray;

public static final Color gray;

public static final Color darkGray;

public static final Color black;

public static final Color red;

public static final Color pink;

public static final Color orange;

public static final Color yellow;

public static final Color green;

public static final Color magenta;

public static final Color cyan;

apabila anda tidak menemui warna yang disukai, maka kita dapat membuat warna yang baru dengan menspesifikasikan *values* dari warna merah, hijau dan biru. Warna yang diciptakan memalui cara ini merupakan bagian dari warna standard RGB yang disebut dengan isitilah **sRGB**.

Anda dapat menciptakan warna menggunakan *nilai integer atau float* :

Public Color (int r, int g, int b)

*Konstruktor ini menciptakan Color yang baru menggunakan nilai yang dispesifikasikan untuk red, green dan blue. Nilai tersebut haruslah dalam rentang dari dan sampai 255.*

Public Color(float r, float g, float b)

Konstruktor ini menciptakan *Color* yang baru dengan menggunakan nilai dari red, green dan blue. Nilai yang akan dipakai haruslah berada dalam rentang dari 0,0 sampai 1,0.

Contoh berikut ini, membentuk sebuah pie-bentuk dari arc yang diisi dengan warna biru.

import java.awt.\*;

import java.awt.geom.\*;

public class SolidPaint extends ApplicationFrame

{

public static void main(String[] args)

{

SolidPaint f = new SolidPaint();

f.setTitle("SolidPaint v1.0");

f.setSize(200, 200);

f.center();

f.setVisible(true);

}

public void paint(Graphics g)

{

Graphics2D g2 = (Graphics2D)g;

Arc2D pie = new Arc2D.Float(0, 50, 150, 150,

-30, 90, Arc2D.PIE);

g2.setPaint(Color.blue);

g2.fill(pie);

}

}

Anda mungkin akan mulai mengingat bahwa setColor(), didefinisikan didalam Graphics, memungkinkan digunakan untuk memberi efek pada pewarnaan yang diisi pada sebuah shape. Pada API 2D, merupakan suatu metode yang sangat bagus yaitu : penggunaan dari setColor(c) dalam Graphics2D adalah sama dengan penggunaan setPaint(c).

**7.1.2 Swing Color Chooser Dialog**

Apabila anda menginginkan agar dapat mengkostumisasi pemilihan warna dalam sebuah aplikasi, Swing telah memiliki kemampuan untuk menangani hal ini. Nama dari kelas yang dimaksud adalah javax.swing.JcolorChooser. Anda dapat menggunakan kelas tersebut untuk di gabungkan dengan baris kode yang lainnya, dengan menerapkan metode static berikut ini :

Public static Color showDialog(Component component,

String title, Color initialColor)

*Metode ini menampilkan dialog dari pilihan warna. Component yang disuplai digunakan sebagai komponen parent dari dialog tersebut. Dialog yang ada akan mensuplai title; pengaturan ini akan diinisialisasikan agar dpat menunjukkan initialColor yang sudah diberikan.*

Contoh berikut ini mendemontrasikan penggunaan dari sebuah dialog. Anda dapat melakukan proses invoke pada warna yang dipilih dengan menekan tombol pilihan. Setelah anda melakukan pemilihan, latar belakang dari frame pada window dirubah sesuai dengan warna yang dipilih.

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class RedTown extends JFrame

{

public static void main(String[] args)

{

new RedTown();

}

public RedTown()

{

super("RedTown v1.0");

createUI();

setVisible(true);

}

protected void createUI()

{

setSize(400, 400);

setLocation(100, 100);

getContentPane().setLayout(new GridBagLayout());

JButton colorButton = new JButton("Choose a

color...");

getContentPane().add(colorButton);

colorButton.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent ae)

{

Color c = JColorChooser.showDialog

(RedTown.this, "Choose a color...",

getBackground());

if (c != null) getContentPane().

setBackground(c);

}

});

addWindowListener(new WindowAdapter()

{

public void windowClosing(WindowEvent we)

{

System.exit(0);

}

});

}

}

**7.1.3 GradientPaint**

Gradient adalah sebuah transisi dari color yang satu ke warna yang lainnya. API 2D menyediakan sebuah pengimplementasian dari *proses gradient* warna biasa, yaitu java.awt.GradientPaint. kelas ini mendefinisikan warna gradient menggunakan dua point dan dua warna. Sebuah proses gradient dengan baik berpindah dari satu warna menjadi warna yang lain saat and bergerak sepanjang garis yang menghubungkan kedua point, yang dimana hal ini disebut ***gradient line***. *GradientPaint* menciptakan sejumlah area warna secara paralel, mengikuti penyesuaikan yang dilewati oleh *gradient line*. Pada gambar dibawah ini menunjukkan proses *gradient* yang berjalan dari putih ke hitam abu-abu. Juga menunjukkan *gradient line* dan *endpoints* dari *gradient* itu sendiri.



**Gambar 33. An acyclic linear gradient**

GradientPaint bisa jadi cyclic atau acyclic. Pada sebuah acyclic GradientPint, titik mana saja diantara titik akhir dari gradient line adalah warna yang sama seperti titik akhir dari garis (gambar 33). Sedangkan cyclic GradientPaint, pewarnaan terus berlanjut atau berganti diantara akhir dari titik gradient line, dalam hal ini segmentasi dari gradient line merupakan replika dari pembentukan sebelumnya, lihat pada gambar dibawah ini.



**Gambar 34. A cyclic linear gradient**

**7.1.4 TexturePaint**

Jenis ketiga dari Paint adalah *texture fill*. Pada API 2D, suatu tekstur melalui penggunaan dari sebuah *image* yang terus tunjukkan secara terus menerus, selayaknya lantai ruangan dengan desain keramik kotak-kotak.

Kelas java.awt.TexturePaint merepresentasikan sebuah tekstur. Anda dapat membangun sebuah TexturePaint dengan dua buah informasi berikut :

1. Image yang digunakan haruslah disuplai memalui proses BufferedImage. Ilustrasikanlah BufferedImage akan menampilkan gambar dalam kotak-kotak.

2. Rectangle2D menspesifikasikan bagaimana *image* tersebut akan di-replika-kan untuk membentuk tekstur.

*Image* tersebut di-skala-kan untuk muat didalam kotak-kotak yang ditentukan. Kotak-kotak ini diproyeksikan kembali untuk membentuk sebuah tekstur. Gambar dibawah ini, menampilkan TexturePaint sebagai tekstur dalam kotak-kotak. *Catatan, image yang digunakan-dilukiskan/ditempatkan dengan tepat untuk mengisi tekstur dalam kotak.*



**Gambar 35. Texture image**

Gambar 36 dibawah ini, menunjukkan TexturePaint dibangun dari image yang sama. Hanya saja, dalam hal ini, kotak-kotak yang berisi tekstur lebih kecil – dan image tersebut telah diskalakan agar muat dengan frame.



**Gambar 36. A texture with a small texture rectangle**

Menciptakan texturePaint adalah semudah menspesifikasikan suatu image dan sebuah kotak :

Public TexturePaint(BufferedImage txtr, Rectangle2D anchor)

Konstruktor ini menciptakan TextureImage yang mrupakan replika dari image yang disuplai menggunakan kotak Anchor

Anda dapat mendapatkan sebuah image dan kotak dari TexturePaint melalui metode berikut ini :

Public BufferedImage get Image()

Pulic Rectangle2D getAnchorRect()

7.2 Stroking

7.2.1 Basic Stroke

- End Styles

- Join Styles

- Dashes

- Constructor

**IX**

**RENDERING & ANGLE UNITS**

8.1 Transforming

8.2 Compositing

8.3 Clipping

8.4 Rendering Hints

**X**

**TEXT AND LAYOUT**

9.1 Fonts

9.2 TextLayout Class

9.3 Close To Metal

**XI**

**IMAGE**

10.1 Menampilkan Image

10.2 Menggambar Image

10.3 Double Buffering

10.4 Class Pendukung

**XII**

**ANIMASI DENGAN THREAD (PERFORMA GRAFIS)**