HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

BÁO CÁO BÀI TẬP LỚN

MÔN: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

ĐỀ TÀI: LẬP TRÌNH ĐỒ HỌA 2D TRONG JAVA

DANH SÁCH NHÓM

1.Lê Ngọc Châu

2. Nguyễ Thị Thêu

3. Nguyễn Ngọc Minh

4. Đinh Thị Thanh Hương

5. Dương Ngọc Anh

Hà Nội 10/2019

Nội dung

A. Đồ họa 2D

I. Giới thiệu

1. Java 2D API

- Java 2D API cung cấp khả năng đồ họa 2D cho các chương trình java thông qua việc mở rộng Abstract Windowing Toolkit (AWT).
- Java 2D API cho phép phát triển giao diện người dùng ở mức phức tạp và phong phú hơn nhờ có đầy đủ các tính năng về đồ họa, văn bản và hình ảnh.
- Các API của Java Animation và Java Media Framework dựa trên API Java 2D để hỗ trợ kết xuất (rendering) nhằm tạo và hiển thị chuyển động.

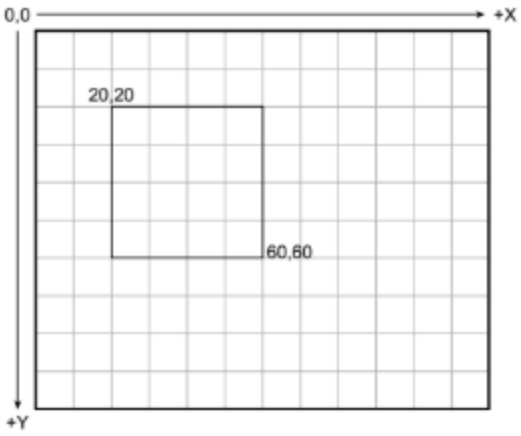
(hình bộ sơ đồ kế thừa của awt)

2. Tọa độ

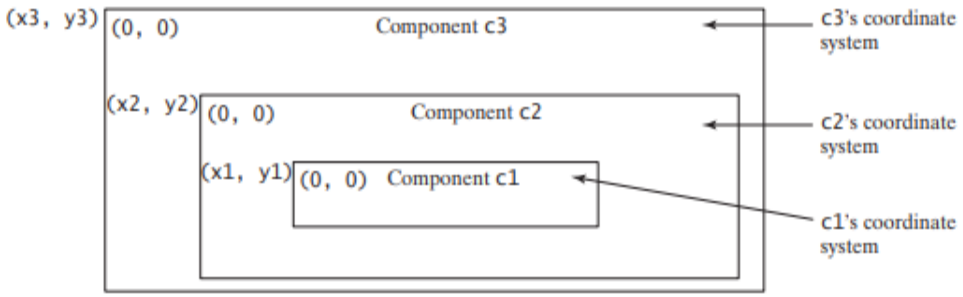
Java 2D API tồn tại song song 2 không gian tọa độ độc lập: *User space, Device space.*

User space: là không gian tọa độ sử dụng trong lập trình.

Chứa hệ tọa độ có gốc (0,0) ở góc trên cùng bên trái. Giá trị x tăng về bên phải và giá trị y tăng chiều hướng xuống.Đơn vị là các số nguyên pixel. Không có một phần hoặc một bộ phận pixel.



Hình minh họa



Device space: là không gian tọa độ của thiết bị ra, các thiết bị ra khác nhau như window, màn hình , máy in,... sẽ khác nhau.

Trong java việc chuyển đổi giữa *User space và Device space* sẽ được thực hiện tự động trong khi rendering. Những đối tượng java 2D tồn tại trên *User space,* khi được kết xuất lên màn hình sẽ chuyển sang *Device space*

3.Lớp Graphics2D

- Kế thừa lớp Graphics thuộc gói java.awt
- Graphics2D cung cấp 3 nhóm phương thức như sau :

1.Phương thức vẽ : draw...,fill....

draw fill

2.Phương thức làm thay đổi các thuộc tính của đối tượng Graphics2D (paint, front, stroke, transform, composite, clip): set...., rotate,...

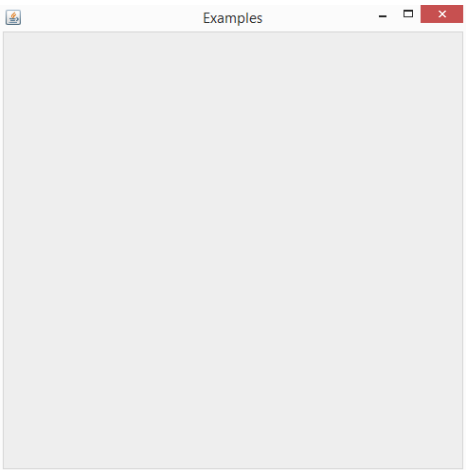
3.Phương thức trả về giá trị các thuộc tính hiện tại của đối tượng : get....

- Để sử dụng được các tính năng của Java 2D API cần phải ép kiểu đối tượng Graphic được truyền vào phương thức của container thành đối tượng Graphics2D

```
public void paint (Graphics g) {
        Graphics2D g2 = (Graphics2D) g;
            ...
}
```

4.Container

a. Frame



Là cửa sổ chính, giao diện chính của người dùng. Frame chứa các Panel, Canvas để đặt các đối tượng đồ họa 2D.

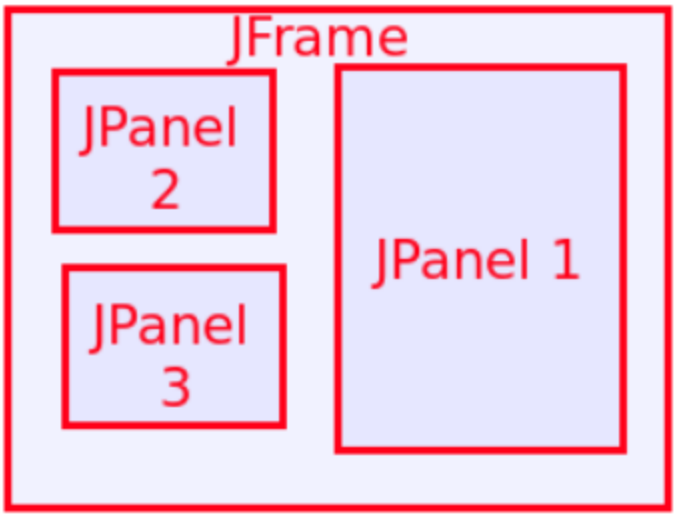
Tạo Frame bằng cách extends JFrame:

```
public class Example extends JFrame{

    private final Panel2 pp = new Panel2();
    private final Panel1 p = new Panel1();

    public Example() throws HeadlessException {
        initUI();
    }

    private void initUI() {
        setTitle("Examples");
        this.setSize(500, 500);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //add(pp);
    }

}
```

Hiển thị Frame :

```
public class Demo {
    public static void main(String[] args) {
        Example ex = new Example();
        ex.setVisible(true);
    }
}
```

b. Panel – Canvas



JPanel (hay Panel) là một container chứa các conpoment cũng như các đối tượng đồ họa 2D. Các Panel được bố trí trong frame theo một layout nào đó. Một panel cũng có thể chứa các panel khác trong nó.

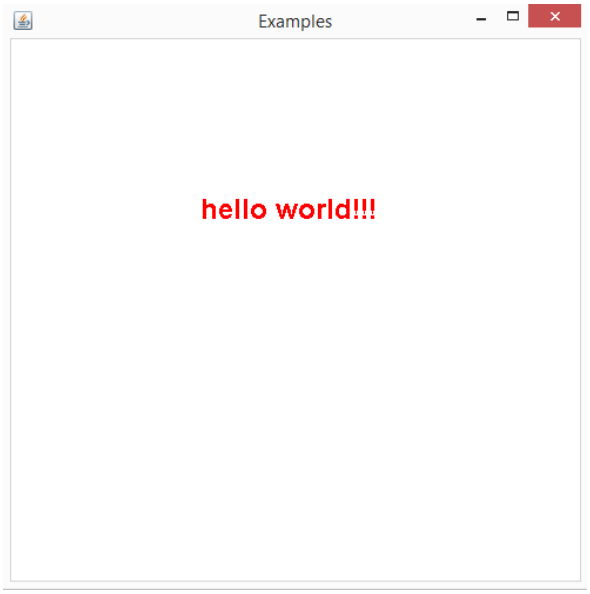
Tạo một panel tương tự tạo một frame:

```
public class MyPanel extends JPanel{

}
```

Panel được kế thừa các phương thức với tham số là một đối tượng graphic như: paint, repaint, paintComponent,... các thương thức này được override ở lớp cháu để hiển thị các đối tượng đồ họa cần được vẽ.

Ví dụ: vẽ String "hello world" màu đỏ lên màn hình tại vị trí (w/3, h/3)

```
public class Panel1 extends JPanel{

    public Panel1() {
       innit();
    }

    private void innit(){
         this.setBackground(Color.WHITE);
    }


    @Override
    public void paintComponent( Graphics g){
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D)g;
        g2d.setColor(Color.red);
        Font f = new Font( "TimesRoman", Font.BOLD, 24);
        g2d.setFont(f);
        g2d.drawString("hello world!!!", this.getWidth()/3, this.getHeight()/3);

    }
}
```



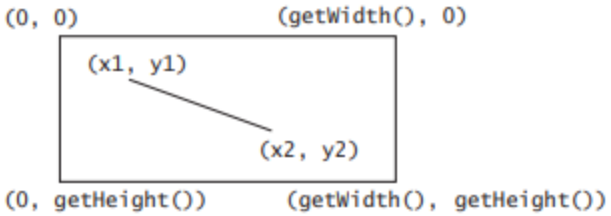
II. Drawing và Filling

Lớp Graphics2D cung cấp nhiều phương thức để lập trình đồ họa, bao gồm các đường (lines) , hình chữ nhật (rectangles), đa giác (polygons), hình bầu dục (ovals) và vòng cung (arcs).

1. Đường (line):

Để vẽ các đường thẳng, sử dụng phương thức drawLine. drawLine có bốn đối số: x và y tọa độ của điểm bắt đầu và tọa độ x và y của điểm kết thúc.

```
@Override
public void paint( Graphics g){
    Graphics2D g2d = (Graphics2D) g;
    g2d.drawLine(25,25,75,75);

}
```



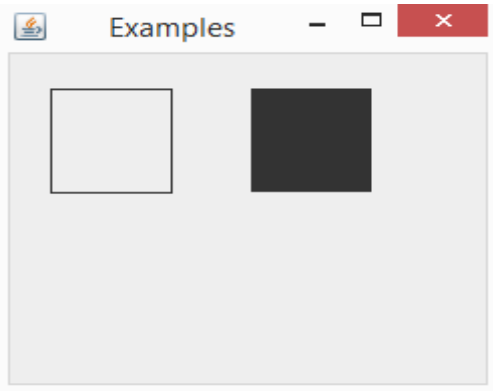
2. Hình chữ nhật (Rectangle):

Đồ họa java cơ bản cung cấp 3 kiểu tam giác :

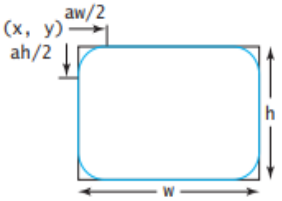
- Hình chữ nhật trơn
- Hình chữ nhật bo góc
- Hình chữ nhật 3 chiều đổ bóng

Đối với mỗi hình chữ nhật trơn, bạn có hai phương pháp để chọn: một phương pháp vẽ hình chữ nhật ở dạng phác thảo (drawRect) và một phương pháp vẽ hình chữ nhật chứa màu sắc (fillRect). Cả hai đều có bốn đối số: tọa độ x và y của góc trên cùng bên trái của hình chữ nhật, và chiều rộng và chiều cao của hình chữ nhật để vẽ.

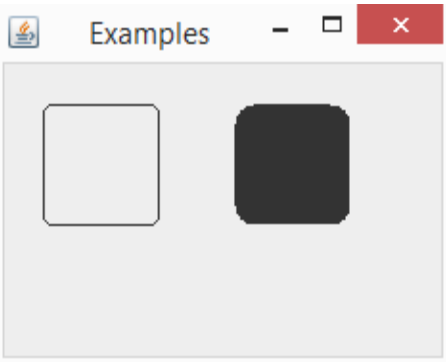
```
@Override
public void paint( Graphics g){
    Graphics2D g2d = (Graphics2D) g;
    g2d.drawRect(20, 20, 60, 60);
    g2d.fillRect(120, 20, 60, 60);

}
```



Đối với hình chữ nhật bo góc được vẽ tương tự như hình chữ nhật thông thường với phương thức drawRoundRect và fillRoundRect để vẽ cần bổ xung thêm hai đối số phụ cho chiều rộng và chiều cao của góc của các đỉnh. Hai đối số đó xác định khoảng cách dọc theo các cạnh của hình chữ nhật, vòng cung cho góc sẽ bắt đầu, cái thứ nhất cho góc theo mặt phẳng ngang, cái thứ hai cho chiều dọc.

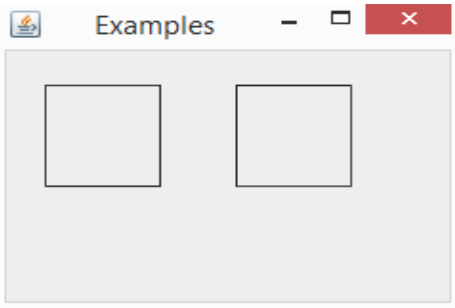



```
@Override
public void paint( Graphics g){
    Graphics2D g2d = (Graphics2D) g;
   // g2d.drawRoundRect(x, y, w, h, aw, ah);
    g2d.drawRoundRect(20, 20, 60, 60, 10, 10);
    g2d.fillRoundRect(120, 20, 60, 60, 20 , 20);
}
```



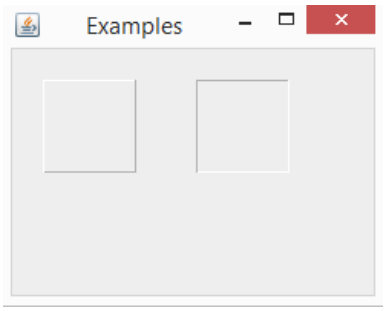
Đối với hình chữ nhật ba chiều, những hình chữ nhật này không thực sự 3D. Thay vào đó, chúng có hiệu ứng đổ bóng khiến chúng có vẻ như được nâng lên hoặc thụt vào từ bề mặt của panel. Hình chữ nhật ba chiều có bốn đối số cho x và y của vị trí bắt đầu và chiều rộng và chiều cao của hình chữ nhật. Đối số thứ năm là một boolean cho biết hiệu ứng 3D là nâng hình chữ nhật (true) hay thụt lề (false). Cũng như các hình chữ nhật khác, cũng có các phương pháp khác nhau để vẽ và điền: draw3DRect và fill3DRect.

```
@Override
public void paint( Graphics g){
    Graphics2D g2d = (Graphics2D) g;
    g2d.draw3DRect(20, 20, 60, 60, true);
    g2d.draw3DRect(120, 20, 60, 60, false);

}
```



Trên thực tế kết quả hiện thị rất khó để thấy hiệu ứng 3D trên hình chữ nhật . Để thấy được hiệu ứng ta có thể setcolor cho đối tượng graphics2D sang màu trắng.

```
@Override
public void paint( Graphics g){
    Graphics2D g2d = (Graphics2D) g;
    g2d.setColor(Color.WHITE);
    g2d.draw3DRect(20, 20, 60, 60, true);
    g2d.draw3DRect(120, 20, 60, 60, false);

}
```



3. Polygon (đa giác)

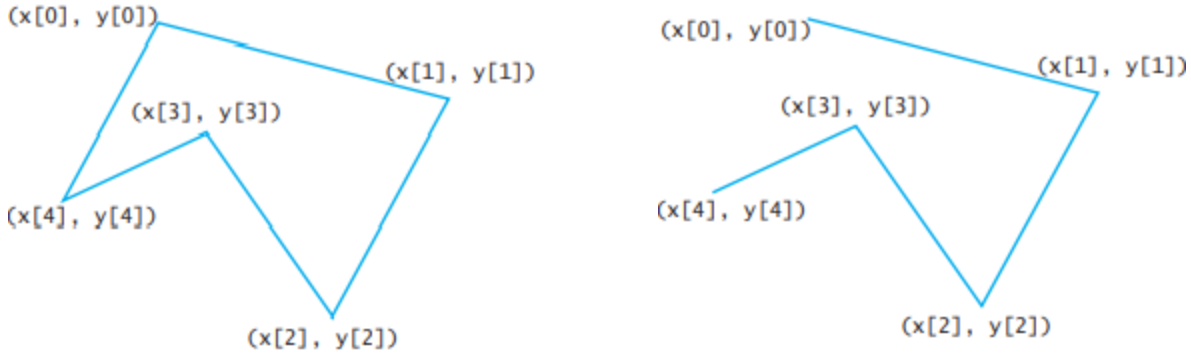
Đa giác là hình dạng với số lượng cạnh không giới hạn. Để vẽ đa giác, bạn cần một tập hợp tọa độ x và y, và phương thức vẽ sau đó bắt đầu tại một điểm, vẽ một đường thẳng đến điểm thứ hai, sau đó là một đường thẳng đến đường thứ ba, v.v.

Như với hình chữ nhật, bạn có thể vẽ một phác thảo hoặc một đa giác đầy (tương ứng là các phương thức drawPolygon và fillPolygon). Bạn cũng có một lựa chọn về cách bạn muốn chỉ ra danh sách các tọa độ, hoặc là các mảng của tọa độ x và y.

Các phương thức drawPolygon và fillPolygon có ba đối số:

- Mảng số nguyên biểu thị các tọa độ.
- Mảng số nguyên biểu thị tọa độ y.
- Một số nguyên cho tổng số điểm.

(chú ý : 2 mảng tọa độ cần có cùng số phần tử)




```
@Override
public void paint( Graphics g){
    Graphics2D g2d = (Graphics2D) g;

   int [] X = {  0, 75, 100, 125, 200, 150, 160, 100, 40 , 50};
   int [] Y = {  85, 75, 10 , 75, 85, 125, 190, 150, 190, 125};
   int pts = X.length;
   g2d.drawPolygon(X, Y, pts);
```

Sau khi tạo đủ ba đối số, công việc bây giờ là nối từng điểm lại để tạo thành đa giác mong muốn. Tuy nhiên, java không tự động đóng đa giác vì vậy cần bao gồm điểm bắt đầu của đa giác ở cuối mảng.

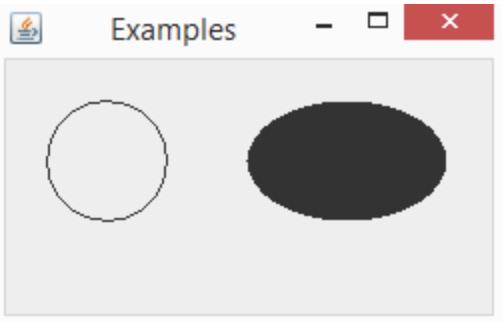
4. Hình bầu dục (Oval)

Hình bầu dục có thể sử dụng để vẽ hình elip hoặc hình tròn. Hình bầu dục tương đương với hình chữ nhật với các góc rất tròn.

Thật vậy, hình bầu dục cũng được vẽ bằng 4 đối số tương tự như hình chữ nhật gồm tọa độ góc trên cùng bên trái vài chiều rộng, chiều cao (2 độ dài trục).

Cũng như các thao tác vẽ khác, phương thức drawOval vẽ đường viền của hình bầu dục và phương thức fillOval vẽ hình bầu dục đầy.

```
@Override
public void paint( Graphics g){
    Graphics2D g2d = (Graphics2D) g;
    g2d.drawOval(20, 20, 60, 60);
    g2d.fillOval(120, 20, 100, 60);
}
```

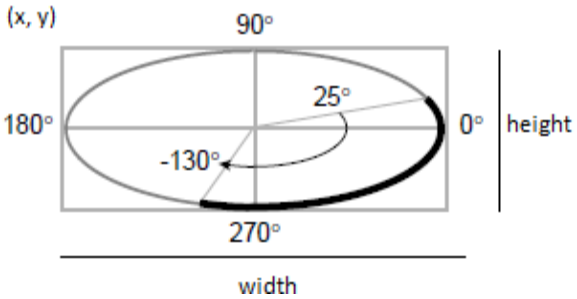


5. Arc (hình vòng cung)

Trong số các thao tác vẽ, vòng cung là phức tạp nhất .Một vòng cung, cách dễ hiểu nhất là coi nó như một phần của hình bầu dục.

Phương thức drawArc có sáu tham số: trong đó 4 tham số đầu tương tự như tạo 1 hình oval hoàn chỉnh, 2 đối số còn lại lần lượt là giá trị góc bắt đầu quét đến giá trị góc kết thúc quét hình(đơn vị độ). Dấu của tham số cuối cùng quyết định hướng quét của hình (nếu âm quét theo chiều kim đồng hồ, giá trị dương thì ngược lại.

```
@Override
public void paint( Graphics g){
    Graphics2D g2d = (Graphics2D) g;
    g2d.drawArc(10, 20, 150, 50, 25, -130);

}
```



6. Copying và clearing

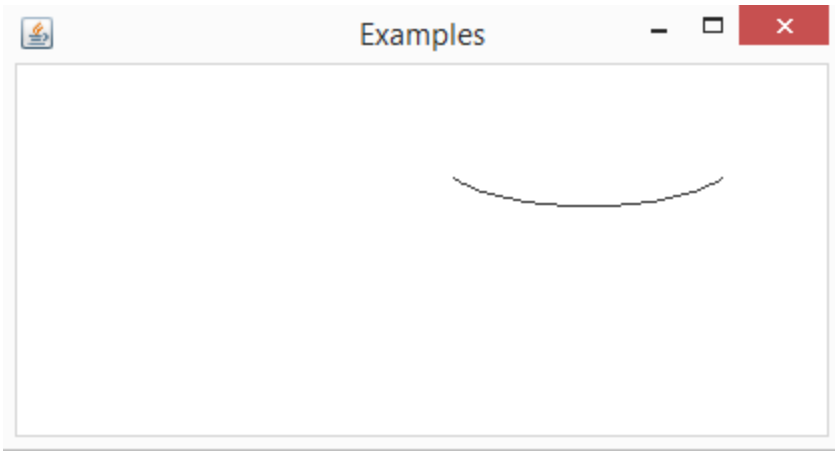
Khi đã vẽ một vài thứ trên màn hình, nếu muốn di chuyển chúng xung quanh hoặc xóa toàn bộ panel. Lớp Graphics2D cung cấp các phương thức để thực hiện cả hai điều này.

Phương thức copyArea sao chép một khu vực hình chữ nhật của màn hình sang một khu vực khác của màn hình. copyArea có sáu đối số: x và y của góc trên cùng của hình chữ nhật để sao chép, chiều rộng

và chiều cao của hình chữ nhật đó và khoảng cách theo hướng x và y để sao chép nó.

Để xóa một khu vực hình chữ nhật, sử dụng phương thức clearRect. clearRect, dùng bốn đối số như các phương thức drawRect và fillRect, lấp đầy hình chữ nhật đã cho bằng màu nền hiện tại của frame.

```
@Override
public void paint( Graphics g){
    Graphics2D g2d = (Graphics2D) g;
    // ve hinh
    g2d.drawArc(10, 20, 150, 50, -25, -130);
    // copy sang phai
   g2d.copyArea(0, 0, 200, 100, 200, 0);
   // xoa hinh ben trai
   g2d.clearRect(0, 0, 170, 100);
}
```



Để xóa toàn bộ có thể sử dụng clearRect với các đối số như sau :

g2d.clearRect (0, 0, this.size (). width, this.height ());

III. Các lớp hình học

1. Shape

Trên thực tế, lớp Graphic2D không quan tâm nhiều đến việc vẽ một hình cơ bản cụ thể như : đường, chữ nhật, oval. lớp Graphic2D quam tâm đến việc vẽ một Shape (dạng hình học). Nó có thể vẽ một shape bằng phương thức draw() và fill(). Đây là điểm tạo ra biệt lớn giữa lớp Graphic và lớp Graphic2D.

Interface java.awt.Shape được sử dụng phổ biến trong lập trình graphic2d, nó quy định các phương thức :

// trả về bao đóng hình chữ nhật của shape

getBounds()

// trả về true nếu điểm thuộc hình fale nếu ngược lại

contains(Point2D p)

// trả về true nếu một phần hình chữ nhật nằm trong shape

intersects(Rectangle2D r)

//

getPathIterator().

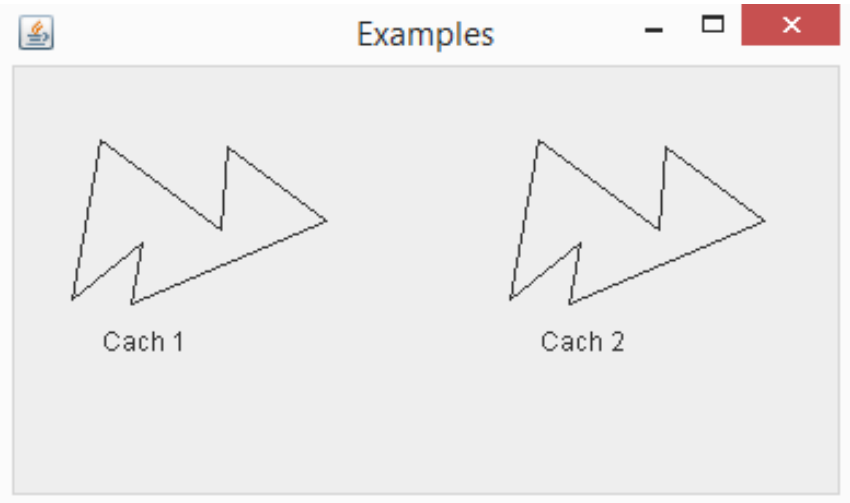
2. Các lớp hình học cơ bản

Gói Java.awt.geom chứa các lớp Java 2D và giao diện liên quan đến shape và hình học cơ bản. Hầu hết các lớp trong gói này đều implements java.awt.Shape có thể được draw hoặc fill.

Gói Java.awt.geom cung cấp các lớp quản lí các đối tượng hình học cơ bản như là: Point2D, Line2D, Rectangle2D, Ellipse2D, Arc2D,....Ngoài ra còn có các lớp quản lí đối tượng hình phức tạp hơn như : CubicCurve2D , QuadCuve2D,....

Ví dụ: vẽ một polygon bằng 2 cách

```
@Override
public void paint( Graphics g){
    Graphics2D g2d = (Graphics2D) g;
    // cach 1
   int [] X = { 39 ,94 ,97 ,142 ,53 ,58 ,26 };
   int [] Y = { 33 ,74 ,36 ,70 ,108 ,80 ,106 };
   int pts = X.length;
   g2d.drawPolygon(X, Y, pts);
   g2d.drawString("Cach 1", 40, 130);
   // cach 2
   Polygon poly = new Polygon();
   for( int i = 0 ; i < pts; i++)
       poly.addPoint(X[i] + 200, Y[i]);
   g2d.draw(poly);
   g2d.drawString("Cach 2", 240, 130);
}
```



Sử dụng đối tượng đồ họa có nhiều ưu điểm hơn cách thông thường. Nó giúp việc lập trình dễ dàng reuse hoặc dùng để tính toán các đối tượng khác vì thế mà mã nguồi trở nên sáng rõ hơn. Ngoài ra nó còn có thể đáp ứng các nhu cầu lập trình phức tạp hơn.

3. Kết hợp hình với lớp java.awt.geom.Area

Các cách để kết hợp 2 hình:

Tương ứng với các phương thức add(Area rhs), intersect(Area rhs), subtract(Area rhs), exclusiveOr(Area rhs).

Bản chất Area là một Shape vì thế có thể draw hay fill nó tương tự như các shape khác.

IV. Draw String

Lớp Đồ họa cũng cho phép in văn bản trên màn hình, kết hợp với lớp Front, và đôi khi là cả lớp FontMetric. Lớp Font đại diện cho một phông chữ đã cho, tên, kiểu, và kích thước . FontMetric cung cấp thông tin về phông chữ đó (ví dụ: chiều cao hoặc chiều rộng của một ký tự cho trước) để có thể bố trí chính xác văn bản trong panel của mình.

Lưu ý: rằng văn bản ở đây là văn bản tĩnh, được vẽ lên màn hình một lần và ở lại đó.

1. Font Object

Để vẽ văn bản lên màn hình, trước tiên cần tạo một thực thể của lớp Font. Các đối tượng phông chữ đại diện cho một phông chữ riêng

lẻ đó là, tên, kiểu của nó (đậm, in nghiêng) và kích thước. Tên phông chữ là các chuỗi đại diện cho họ của phông chữ.

Ví dụ: "TimesRoman" , "courier", hoặc" Helicaica".

Các kiểu phông chữ là các hằng số được xác định bởi lớp Font; bạn có thể lấy chúng bằng cách sử dụng các biến lớp.

Ví dụ: *Font.PLAIN, Font.BOLD hoặc Font.ITALIC.*

Cuối cùng, kích thước điểm là kích thước của phông chữ, như được xác định bởi chính phông chữ; kích thước có thể hoặc không thể là chiều cao của các ký tự.

Để tạo một đối tượng phông chữ riêng lẻ, hãy sử dụng ba đối số này cho lớp xây dựng mới của lớp Font:

Ví dụ: *Front f = new Front("TimesRoman", Front.BOLD, 24);*

Các kiểu phông chữ thực sự là các hằng số nguyên có thể được thêm vào để tạo các kiểu kết hợp.

Ví dụ: tạo ra một phông chữ in đậm và in nghiêng

Font.BOLD + Font.ITALIC

Lưu ý: Các phông chữ có thể sử dụng được phụ thuộc vào hệ thống mà chương trình đang chạy. Trong trường hợp không tìm thấy font cần sử dụng, java sẽ thay thế bằng một font chữ mặc định có thể là Courier.

2. Draw Character và String

Vẽ văn bản trên màn hình bằng các phương thức drawChars và drawString với một đối tượng phông chữ đã tạo trước. Trước tiên, cần đặt phông chữ hiện tại cho đối tượng phông chữ của vừa tạo bằng phương thức setFont.

Font chữ hiện tại là một phần của trạng thái đồ họa được theo dõi bởi đối tượng Đồ họa. Mỗi lần vẽ một ký tự hoặc một chuỗi lên màn hình, văn bản đó được vẽ bằng cách sử dụng pont chữ hiện tại. Để thay đổi phông chữ của văn bản, trước tiên hãy thay đổi phông chữ hiện tại.

```
@Override
public void paint( Graphics g){
    Graphics2D g2d = (Graphics2D) g;
    Font f = new Font("TimesRoman", Font.PLAIN, 72);
    g2d.setFont(f);
    g2d.drawString("Hello world!!!", 10, 100);
}
```

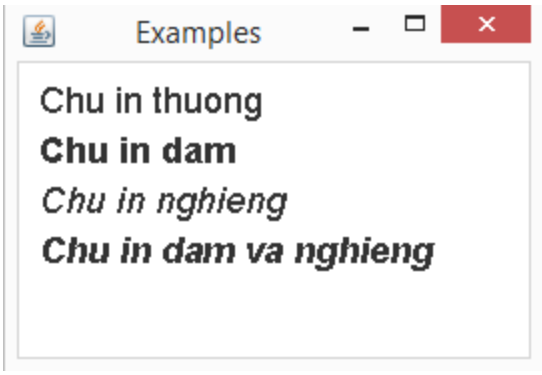
Hai tham số sau để drawString xác định điểm mà chuỗi sẽ bắt đầu. Giá trị x là bắt đầu của cạnh ngoài cùng bên trái của văn bản, y là đường cơ sở cho toàn bộ chuỗi.

Tương tự như drawString là phương thức drawChars, thay vì lấy một chuỗi làm đối số, sẽ lấy một mảng các ký tự. drawChars có năm đối số: mảng các ký tự, một số nguyên n đại diện cho ký tự đầu tiên trong mảng cần vẽ, một số nguyên khác cho ký tự cuối cùng trong mảng cần vẽ (tất cả các ký tự giữa đầu tiên và cuối cùng được vẽ) và x và y cho điểm bắt đầu.

Ví dụ:

```
char[] hello = {'h', 'e', 'l' , 'l' , 'o' , ' ' , 'w' , 'o', 'r', 'l', 'd', '!'};
    g2d.drawChars(hello, 0, 12, 10, 200);
```

Ví dụ: một số kiểu chữ.



```
@Override
public void paint( Graphics g){
    Graphics2D g2d = (Graphics2D) g;
    Font f1 = new Font("TimesRoman", Font.PLAIN, 18);
    Font f2 = new Font("TimesRoman", Font.BOLD, 18);
    Font f3 = new Font("TimesRoman", Font.ITALIC, 18);
    Font f4 = new Font("TimesRoman", Font.BOLD + Font.ITALIC, 18);
    g2d.setFont(f1);
    g2d.drawString("Chu in thuong", 10, 25);
    g2d.setFont(f2);
    g2d.drawString("Chu in dam", 10, 50);
    g2d.setFont(f3);
    g2d.drawString("Chu in nghieng ", 10, 75);
    g2d.setFont(f4);
    g2d.drawString("Chu in dam va nghieng", 10, 100);
}
```

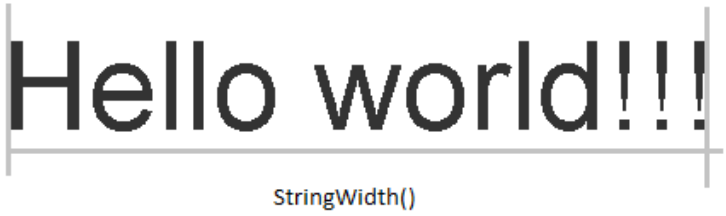
3. FontMetric

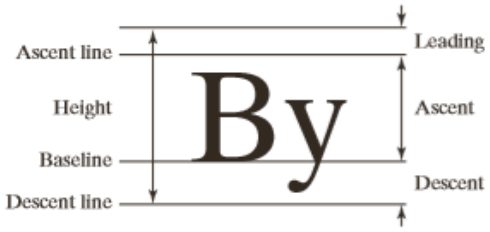
Để kiểm tra các thuộc tính của một Font có thể dùng một số các phương thức đơn giản cung cấp sẵn bởi lớp Font.

Method Name	In Object	Action
`getFont()`	Graphics	Returns the current font object as previously set by `setFont()`
`getName()`	Font	Returns the name of the font as a string
`getSize()`	Font	Returns the current font size (an integer)
`getStyle()`	Font	Returns the current style of the font (styles are integer constants: `0` is plain, `1` is bold, `2` is italic, `3` is bold italic)
`isPlain()`	Font	Returns `true` or `false` if the font's style is plain
`isBold()`	Font	Returns `true` or `false` if the font's style is bold
`isItalic()`	Font	Returns `true` or `false` if the font's style is italic

Tuy nhiên, khi cần kiểm tra chi tiết hơn về từng font chữ (ví dụ chiều dài chiều cao). FontMetric có thể cho biết các thuộc tính sau đây:

Method Name	Action
`stringWidth()`	Given a string, returns the full width of that string, in pixels
`charWidth()`	Given a character, returns the width of that character
`getAscent()`	Returns the ascent of the font, that is, the distance between the font's baseline and the top of the characters
`getDescent()`	Returns the descent of the font—that is, the distance between the font's baseline and the bottoms of the characters (for characters such as p and q that drop below the baseline)
`getLeading()`	Returns the leading for the font, that is, the spacing between the descent of one line and the ascent of another line
`getHeight()`	Returns the total height of the font, which is the sum of the ascent, descent, and leading value





V. Color

Java cung cấp các phương thức và behavior để xử lý màu nói chung thông qua lớp Color và cũng cung cấp các phương thức để thiết lập background và foreground để bạn có thể vẽ bằng màu bạn đã tạo. Mô hình màu trừu tượng Java sử dụng 24 bit màu, trong đó một màu được biểu diễn dưới dạng kết hợp của các giá trị đỏ, lục và lam. Mỗi thành phần của màu có thể có một số từ 0 đến 255. 0,0,0 là màu đen, 255,255,255 là màu trắng và Java cũng có thể đại diện cho hàng triệu màu. Mô hình thường chỉ có 256 màu hoặc ít hơn để chọn. Nếu một màu được yêu cầu trong một đối tượng Color không có sẵn để hiển thị, màu kết quả có thể được chuyển sang màu khác hoặc kết hợp màu sắc, tùy thuộc vào nền tảng chạy chương trình.

1. Color Object

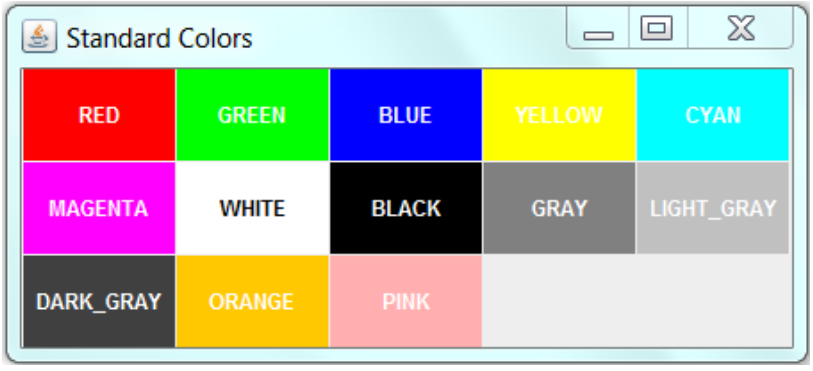
Để vẽ một đối tượng trong một màu cụ thể, bạn phải tạo một thực thể của lớp Color để thể hiện màu đó. Lớp Color định nghĩa một tập hợp các đối tượng màu tiêu chuẩn, được lưu trữ trong các biến của lớp, cho phép bạn nhanh chóng có được một đối tượng màu cho một số màu phổ biến hơn.

Ví dụ:

Color.red cung cấp cho bạn một đối tượng Color đại diện cho màu đỏ (giá trị RGB là 255, 0 và 0),

Color.white cung cấp cho bạn màu trắng (giá trị RGB là 255, 255 và 255), v.v.

Các standard colors được cung cấp sẵn bởi lớp Color :



Nếu màu mong muốn vẽ không phải là một trong những đối tượng màu tiêu chuẩn. Lớp color có thể tạo một đối tượng màu cho bất kỳ sự kết hợp nào của màu đỏ, xanh lá cây và xanh dương, miễn là có các giá trị của màu mong muốn.

Ví dụ: tạo đối tượng màu dark gray.

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

Ngoài ra, cũng có thể tạo một đối tượng màu bằng ba phao từ 0,0 đến 1,0

Ví dụ:

Color c = new Color(0.34,1.0,0.25);

2. Kiểm tra và thiết lập màu

Để vẽ một đối tượng hoặc văn bản bằng một đối tượng màu, bạn phải đặt màu hiện tại thành đối tượng màu đó, giống như bạn phải đặt phông chữ hiện tại thành phông chữ mà bạn muốn vẽ. Sử dụng phương thức setColor().Sau khi thiết lập màu hiện tại, tất cả các thao tác vẽ sẽ hiển thị theo màu đó.

Ví dụ:

g2d.setColor(Color.red);

g2d.setColor(255, 0, 0);

Ngoài việc đặt màu hiện tại cho đồ họa, cũng có thể đặt background và foreground cho chính frame, panel bằng cách sử dụng các phương thức setBackground và setForeground. Cả hai phương thức này đều được định nghĩa trong lớp java.awt.Component do đó các frame, panel và lớp kế thừa 2 lớp trên của bạn tự động kế thừa phương thức này

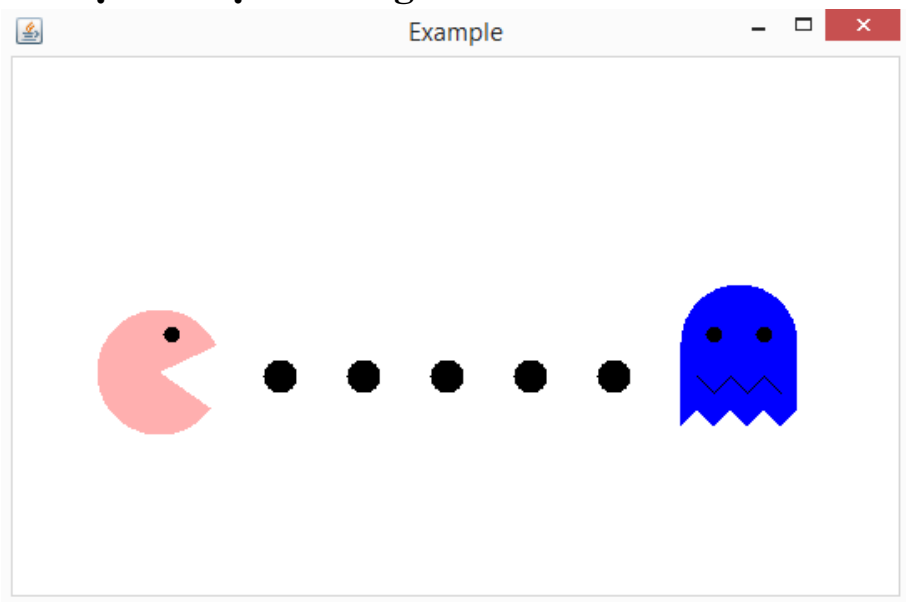
Thông thường, màu background mặc định là màu xám đậm còn foreground mặc định là màu đen.

Ngoài các phương thức setColor, setForeground và setBackground, còn có các phương thức get...() tương ứng cho phép bạn truy xuất màu đồ họa, nền hoặc tiền cảnh hiện tại. Các phương thức đó là getColor (được xác định trong các đối tượng Đồ họa), getForeground (được xác định trong compoment) và getBackground (cũng trong compoment).

Ví dụ:

setForeground (g2d.getColor);

Một ví dụ đồ họa đơn giản




```
public class Example1 extends JPanel {


    public void paint(Graphics  g){

        Graphics2D g2d =  (Graphics2D) g;

        // pacman
        g2d.setColor(Color.pink);
        g2d.fillArc(50, 150, 75, 75, 25, 300);
        g2d.setColor(Color.black);
        g2d.fillOval(90, 160, 10, 10);

        //dot
        for( int i = 0 ; i < 5; i++)
            g2d.fillOval(150 + 50*i, 180, 20, 20);

        //ghost
        int X[] = {400, 470, 470,460,450,440,430,420,410,400, 400};
        int Y[] = {10 +160, 10+160, 50+160,60+160,50+160,60+160,
                    50+160,60+160,50+160,60+160, 10+160};

        g2d.setColor(Color.blue);
        Area than = new Area( new Polygon(X, Y, X.length));
        Area dau = new Area(new Ellipse2D.Double(400, 160-25, 70, 70));
        than.add(dau);
        g2d.fill(than);

        g2d.setColor(Color.black);
        g2d.fillOval(415, 160, 10, 10);
        g2d.fillOval(445, 160, 10, 10);

        int XX[] = { 460,450,440,430,420,410};
        int YY[] = { 60+140,50+140,60+140,50+140,60+140,50+140};

        for( int i = 0 ; i < XX.length-1; i++)
            g2d.drawLine(XX[i], YY[i], XX[i+1], YY[i+1]);


    }
}
```

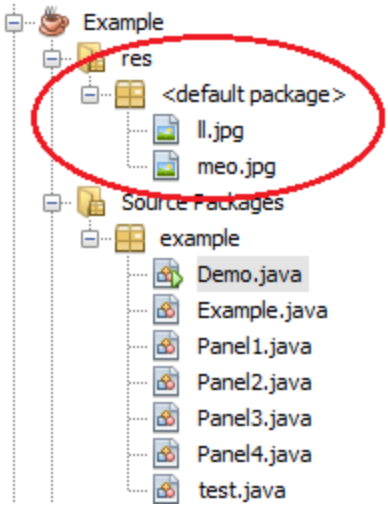
VI. Image

Một Image là mảng 2 chiều các ô màu. Mỗi phần tử là một pixel.

Xử lí hình ảnh cơ bản trong java khá đơn giản. Lớp trong java.awt cung cấp các thuộc tính trừu tượng đại diện cho biểu hiện ảnh thông thường. Đặc biệt, các phương thức định nghĩa trong panel và Graphics2D cho phép thực hiện load hay hiển thị ảnh dễ như vẽ một hình chữa nhật vậy.

1. Lấy hình ảnh

Để hiển thị một hình ảnh trong frame của bạn, trước tiên cần phải tải hình ảnh vào file mã nguồn chương trình Java đang thực hiện. Hình ảnh được lưu trữ dưới dạng các tệp riêng biệt từ các tệp lớp Java, vì vậy cần phải cho Java biết nơi để tìm chúng.



Bước tiếp theo là load ảnh và tạo một thực thể của lớp Image trong gói java.awt. Trong trường hợp ảnh tải lên không thành công đối tượng image sẽ nhận giá trị null vì thế khi thực hiện draw không hiện hình ảnh nào.

Ví dụ: tải file ảnh meo.jpg vào đối tượng image.

```
Image image = this.getToolkit().getImage("res/meo.jpg");
```

Hoặc

```
Image image = new ImageIcon("res/meo.jpg").getImage();
```

Chú ý: Java hỗ trợ hình ảnh ở định dạng GIF và JPEG

2. Vẽ hình ảnh

Lớp Graphics cung cấp hai phương thức drawImage để làm vẽ hình ảnh.

Phiên bản đầu tiên của drawImage có bốn đối số: hình ảnh để hiển thị, vị trí x và y của góc trên cùng bên trái và this(mặc định). Phương thức này vẽ hình ảnh theo kích thước ban đầu của ảnh tại vị trí góc trên trái (x,y) . Trong trường hợp ảnh quá lớp so với kích thước frame thì ảnh chỉ hiển thị một phần lên frame.

Ví dụ:

g2d.drawImage(image, 100, 100, this);

Dạng drawImage thứ hai có sáu đối số: hình ảnh cần vẽ, tọa độ x và y, giới hạn chiều rộng , chiều cao của hình ảnh và this (mặc định).

Nếu giới hạn chiều rộng , chiều cao của hình ảnh nhỏ hơn hoặc lớn hơn hình ảnh thực tế, hình ảnh sẽ tự động được điều chỉnh cho phù hợp. Sử dụng các đối số bổ sung đó cho phép ép và mở rộng hình ảnh vào bất kỳ không gian nào cần để phù hợp (tuy nhiên, hình ảnh có thể bị biến dạng do việc thu nhỏ nó hoặc lớn hơn kích thước thật của nó).

Ví dụ:

g2d.drawImage(image, 200, 200, 200, 200, this);

Việc chia tỉ lệ hiển thị phù hợp sẽ giúp tránh tránh biến dạng hình ảnh theo chiều rộng hoặc chiều cao. Lớp Image cung cấp 2 phương thức getWidth () và getHeight () với đối số this sẽ trả về chiều rộng chiều cao của ảnh (đơn vị pixel).

```
public class Panel5 extends JPanel{

    private Image im ;
    public Panel5() {
        innit();
    }

    private void innit(){
        this.setBackground(Color.WHITE);
        this.setSize(500, 500);
        im = this.getToolkit().getImage("res/meo.jpg");
    }

    @Override
    public void paintComponent( Graphics g){
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        // vẽ hình mèo
        int w = im.getWidth(this);
        int h = im.getHeight(this);

        // 25% x, 25%y
            g2d.drawImage(im, 100, 100, w/4, h/4,  this);
        // 50%x, 25%y
            g2d.drawImage(im, 110 + w/4, 100, w/2, h/4 , this);


    }
}
```

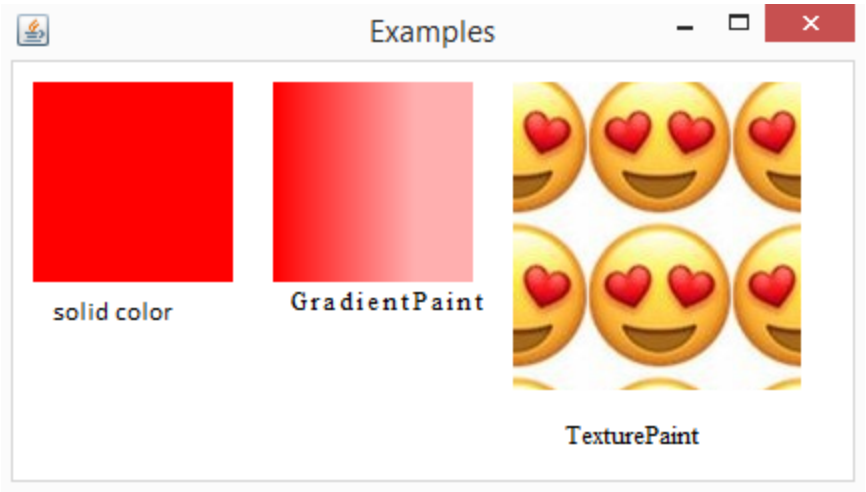
Đối số this (mặc định) khi gọi phương thức drawImage() trên thực tế là một ImageObserver. Lớp Component đã implements ImageObserver interface vì thế panel5 cũng là một ImageObserver.

VII. Painting

Painting là quá trình điền đầy phần bên trong của một shape trước khi shape được fill. Có ba cách để painting một shape

1. Solid Colors : đổ màu lì bằng phương thức setColor() với đối số là một Color (như trên)
2. GradientPaint : tạo hiệu ứng chuyển từ mày này sang một màu khác dọc theo hướng của một đường thẳng sử dụng phương thức setPaint() với đối số là một GradientPaint
3. TexturePaint : thực hiện điền đầy một hình bằng ảnh sử dụng phương thức setPaint() với đối số là một TexturePaint

```
@Override
public void paintComponent( Graphics g){
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D) g;

//Solid Colors
    g2d.setColor(Color.red);
    g2d.fillRect(10, 10, 100, 100);
//GradientPaint
    g2d.setPaint(new GradientPaint(130, 10, Color.red, 200, 10, Color.pink));
    g2d.fillRect(130, 10, 100, 100);
//  TexturePaint
    g2d.setPaint(new TexturePaint(im, new Rectangle2D.Double(0, 0, im.getWidth(), im.getHeight())));
    g2d.fillRect(250, 10, im.getWidth()*2, im.getHeight()*2);
```



Đối với cách sử dụng TexturePaint, trong trường hợp ảnh nhỏ hơn hình cần được đổ màu hình sẽ được điền đầy bởi nhiều hình.

VIII. Stroking

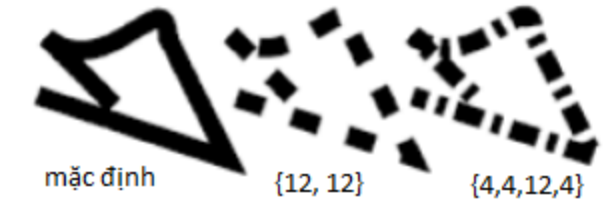
Stroking là quá trình vẽ đường viền (đường phác thảo) cho hình tương tự như paiting.

Graphic2D sử dụng thuộc tính Stroke để định dạng cụ thể hình được vẽ như thế nào. Phương thức setStroke() chấp nhận đối số là một đối tượng implement java.awt.Stroke , thông dụng nhất là lớp BasicStroke. Tương tự như Paint, Stroke không thể thay đổi sau khi đã được khởi tạo. Một đối tượng BasicStroke chứa các thông tin về độ rộng đường viền, kiểu nối, kiểu kết thúc (end-cap style), kiểu nét đứt (dash style).

Đối với join style BasicStroke hỗi trợ 3 kiểu như sau :

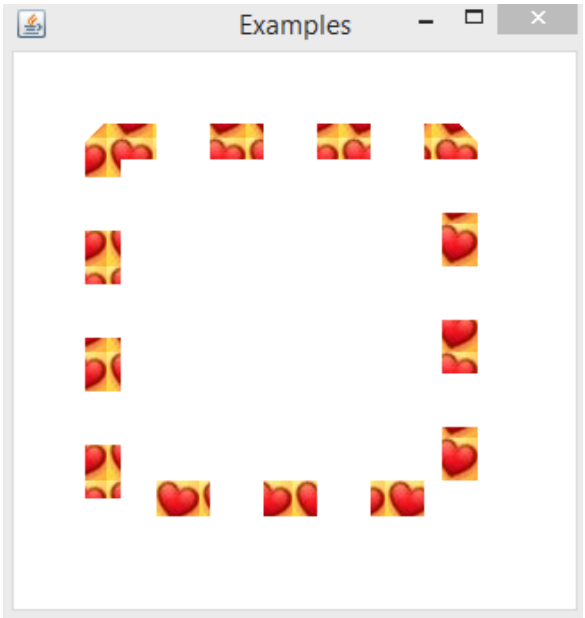
Đối với end style BasicStroke hỗi trợ 3 kiểu như sau :

Dash style là một mảng quy định độ dài của một nét vẽ và khoảng cách giữa chúng



Bản chất một stroke cũng là một shape vì thế nó cũng có thể được paint, fill tương tự như một shape.

```
    @Override
public void paintComponent( Graphics g){
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D) g;
    Rectangle2D r = new Rectangle2D.Double(50, 50, 200, 200);
    Stroke stroke = new BasicStroke(20,
                        BasicStroke.CAP_BUTT, BasicStroke.JOIN_BEVEL, 0,
                        new float[] { 30, 30 }, 0);
    g2d.setStroke(stroke);
    Shape strokedOutline = stroke.createStrokedShape(r);
    g2d.setPaint(new TexturePaint(tim,
        new Rectangle2D.Double(0, 0, tim.getWidth(), tim.getHeight())));
    g2d.fill(strokedOutline);
    //g2d.draw(r);
}
```

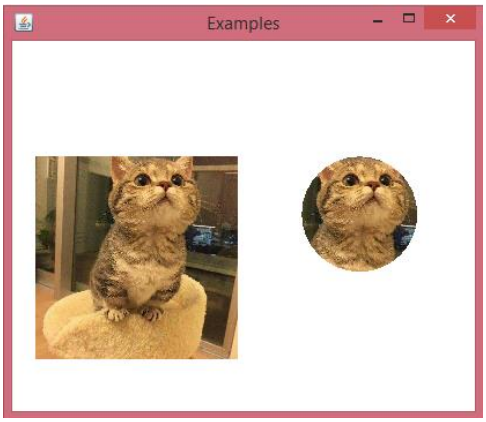


Trong trường hợp vừa dùng Stroke vừa dùng paint thì kết quả hình ảnh phụ thuộc vào thứ tự dùng thuộc tính.

IX. Clipping

Clipping là kĩ thuật giới hạn nội dung vẽ. Nó có tác dụng trong việc vẽ ảnh mà muốn cắt đi một phần của ảnh. Graphic2D cho phép sử dụng một vài shape để chip hoặc cũng có thể dùng hình text.

```
    @Override
public void paintComponent( Graphics g){
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D) g;

    // trc khi clip
    g2d.drawImage(meo, 20, 100, meo.getWidth(this)/4,
                  meo.getHeight(this)/4,  this);
    //clip
    Ellipse2D e = new Ellipse2D.Float(250, 100, 100, 100);
    g2d.clip(e);
    g2d.drawImage(meo, 200, 100, meo.getWidth(this)/4,
                  meo.getHeight(this)/4,  this);

}
```



B. Creating Animation in Java (Tạo hoạt cảnh trong Java)

Animation trong Java bao gồm hai bước: xây dựng frame của animation và sau đó yêu cầu Java vẽ frame đó. Lặp lại việc đó để tạo hiệu ứng chuyển động. Về cơ bản thì bước đầu tiên đã được giới thiệu ở các phần trước, công việc còn lại là làm sao để Java vẽ một frame.

I. Painting và Repainting

Phương thức paint () được Java gọi khi JPanel cần được vẽ. Tuy nhiên, bạn cũng có thể yêu cầu Java tạo lại frame tại một thời điểm bạn muốn. Vậy để thay đổi giao diện của màn hình thì bạn phải thiết lập hình ảnh hoặc "frame" mà bạn muốn vẽ, sau đó yêu cầu Java vẽ frame này. Nếu bạn lặp lại việc này nhiều lần và đủ nhanh thì bạn sẽ có được animaton trong Java JPanel của mình.

Vậy tất cả những việc trên diễn ra ở đâu? Chúng không có trong phương thức paint (). Tất cả những gì phương thức paint () làm chỉ là tạo những điểm (dots) trên màn hình. Nói cách khác, paint () chỉ tạo ra frame hiện tại của animation tại một thời điểm và việc thay đổi các frame để tạo animation nằm ở phần khác. Tại đó, bạn sẽ thiết lập frame (đặt biến cho paint() để sử dụng, tạo màu hoặc phông chữ hoặc các objects khác mà pain () sẽ cần) và sau đó gọi phương thức repaint(), đây là cách khiến Java gọi paint () và vẽ frame.

Khi bạn vẽ trên JPanel, bạn sẽ chỉ định thông tin như vị trí, chiều rộng, chiều cao bằng các tham số.

Ví dụ: vẽ hình chữ nhật bạn sẽ sử dụng :

Graphics.drawRect(x,y,width,height);.

Bằng cách tăng các biến x hoặc y, vị trí của hình bạn vẽ sẽ thay đổi. Tuy nhiên hình cũ sẽ không được xóa bằng phương thức repaint hay paint, để chỉ repaint mỗi hình đó ở vị trí mới mà không hiển thị hình ở vị trí cũ thì bạn sẽ phải repaint lại nền như sau: cập nhật biến x và y, repaint nền, repaint lại đối tượng với các giá trị x và y mới của nó.

```
public void update() {
    x += xSpeed;
    y += ySpeed;
    if (x > CANVAS_WIDTH - size || x < 0) {
        xSpeed = -xSpeed;
    }
    if (y > CANVAS_HEIGHT - size || y < 0) {
        ySpeed = -ySpeed;
    }
}

@Override
public void actionPerformed(ActionEvent e) {
    update();   // cap nhat vi tri x va y
    repaint();  // ve lai
}
```

II. Thực hiện Animations

Bước quan trọng nhất để tạo một Animations chính là khởi tạo các framework một cách chính xác. Ngoại trừ các Animations đáp ứng cho các sự kiện mở rộng (ví dụ như người dung kéo một đối tượng trên màn hình), một chương trình thực hiện Animations cần có một vòng lặp của nó.

Cách đơn giản nhất để tạo một Animation đó là di chuyển một hình ảnh trên màn hình. Để làm được việc đó chúng ta sẽ sử dụng Swing Timer. Vậy công dụng của nó là gì? Một timer được sử dụng như một cách để kiểm tra dữ liệu theo định kỳ. Nếu ta ứng dụng nó trong game, nó sẽ rất có ích. Một vài yếu tố của trò chơi phải được kiểm tra liên tục và một số nhiệm vụ thường phải được thực hiện liên tục như phát hiện va chạm và vẽ các đơn vị lên màn hình.

Một swing timer có thể sử dụng để lên lịch các tác vụ được mô tả trên các khoảng thời gian đều đặn. Việc này rất hoàn hảo cho Animation và những tác vụ liên quan đến GUI. Swing Timer được sử dụng cho một cái gì đó như cập nhật một hình ảnh được vẽ ngay lập tức vào màn hình khi khởi động. Việc gọi Timer bao nhiêu lần mỗi giây tùy thuộc vào những gì bạn muốn làm, nếu bạn đang kiểm tra phát hiện va chạm hoặc vẽ một cái gì đó lên màn hình, nói chung là nếu bạn muốn cập nhập nhiều lần trong một giây thì khoảng thời gian 15mili giây tương đương với 66 FPS là hợp lí.

Timer có hàm khởi tạo theo cấu trúc: public Timer(int delay, ActionListener listener). Bạn sẽ phải ghi đè phương thức actionPerformed() của ActionListener để chỉ định hành vi của tác vụ. Timer sẽ kích hoạt một ActionEvent sau độ trễ ban đầu và sau đó là đều đặn theo độ trễ. Bạn có thể bắt đầu và dừng Timer thông qua các phương thức start() và stop() của Timer.

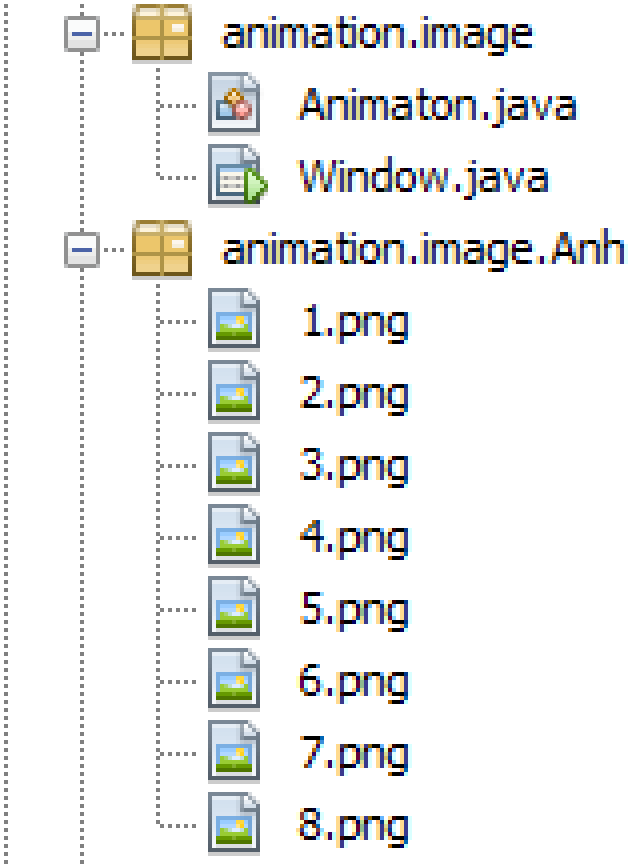
```
public class RectangleAnimationPanel extends JPanel implements ActionListener{
    Timer tm = new Timer(5,this);
    int x = 0, velx = 2;
    public void paintComponent(Graphics g){
        super.paintComponent(g);

        g.setColor(Color.red);
        g.fillRect(x, 30, 50, 20);

        tm.start();
    }

    @Override
    public void actionPerformed(ActionEvent ActE){
        if(x<0||x>550){
            velx = - velx;
        }
        x = x + velx;
        repaint();
    }
}
```

Bạn có thể sử dụng phương thức setRepeats(false) để đặt Timer chỉ kích hoạt một lần sau độ trễ. Bạn cũng có thể đặt độ trễ ban đầu thông qua setInitialDelay() và độ trễ thông thường bằng setDelay().

Một Timer có thể kích hoạt ActionEvent tới nhiều ActionListeners. Bạn có thể đăng ký thêm ActionListener thông qua phương thức addActionListener().

III. Tạo Animations bằng cách sử dụng các hình ảnh

Tạo animations bằng cách sử dụng ảnh cũng giống như cách tạo ảnh bằng cách sử dụng fonts, màu hoặc shapes, cách thực hiện về painting, repainting cũng tương tự. Điều khác biệt duy nhất đó là bạn phải xếp những ảnh đó lại để hiển thị ra chứ không phải là một tập hợp các phương thức painting. Trước khi bắt tay vào tạo một ảnh động thì bạn nên có tất cả các hình ảnh để tạo nên ảnh động đó, lưu trữ nó vào một file riêng.



Ý tưởng cơ bản đó là sử dụng những hình ảnh đó và hiển thị chúng một cách nhanh chóng để tạo chuyển động. Cách đơn giản nhất đó là bạn lưu trữ những hình ảnh này vào một mảng và sau đó khởi tạo một biến để lưu địa chỉ của hình ảnh hiện tại. Bởi vì bạn cần phải theo dõi

tọa độ x và y của ảnh hiện tại, vậy nên cần thêm 2 biến x và y. Tùy thuộc vào mục đích của bạn mà x và y có thể thay đổi hoặc không.

```
private Image[] nekopics = new Image[9];
private Image currentimg;
private int x, y;

public void innit(){

    String nekosrc[] = { "right1", "right2", "stop", "yawn", "scratch1",
        "scratch2","sleep1", "sleep2","awake" };

    for( int i = 0; i < nekosrc.length; i++ )
        nekopics[i] = (new ImageIcon("Anh/"+ nekosrc[i]+ ".gif")).getImage();
}
```