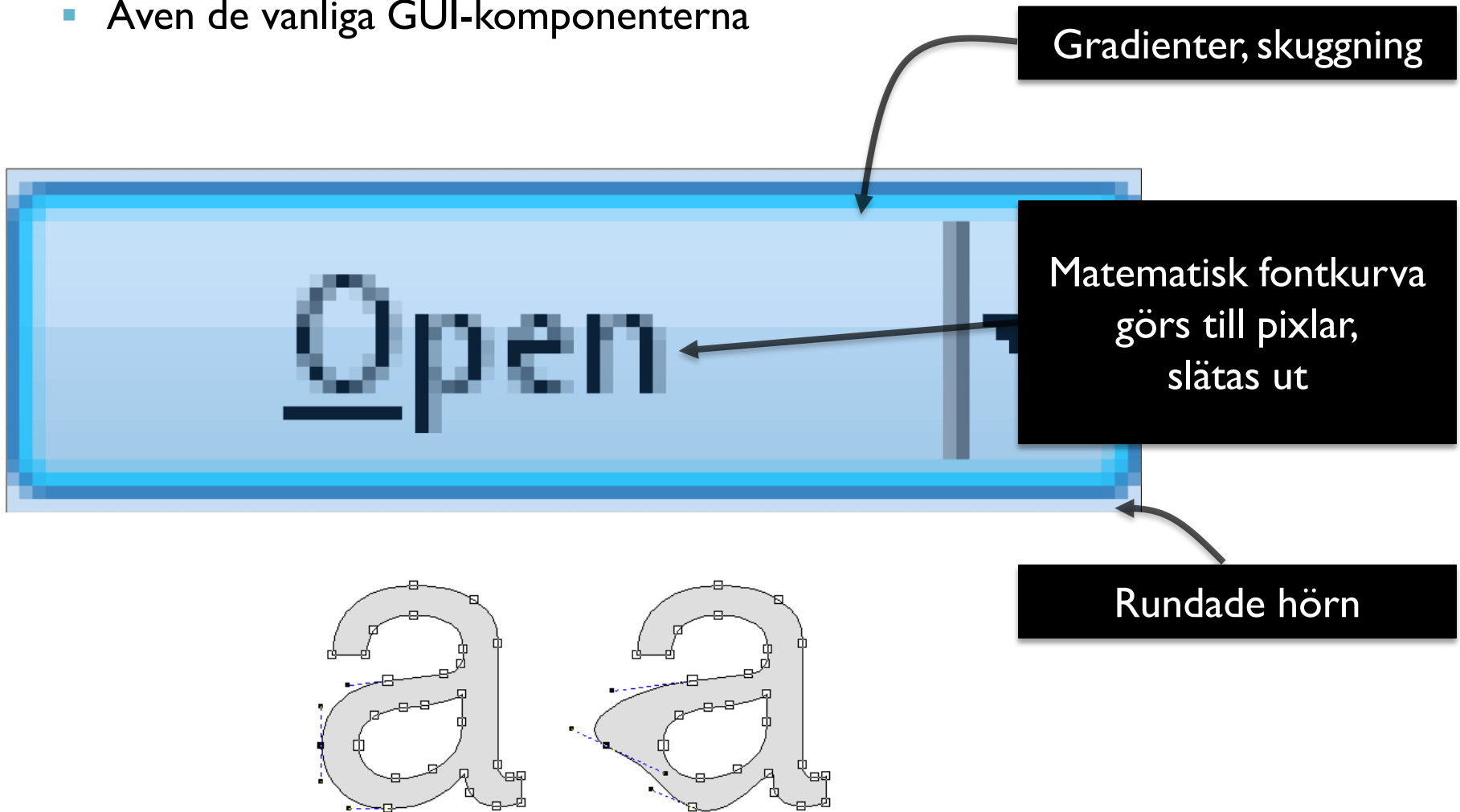




# Grafik:

## Att "rita" egna komponenter

- Allt på skärmen **ritas** som **pixlar**
  - Även de vanliga GUI-komponenterna

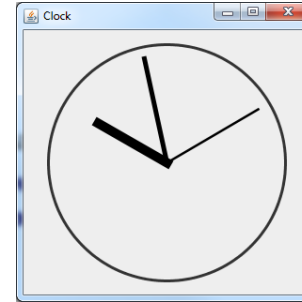


# Intro 2: Egna komponenter

- Detta behövs också för att skapa egna komponenter

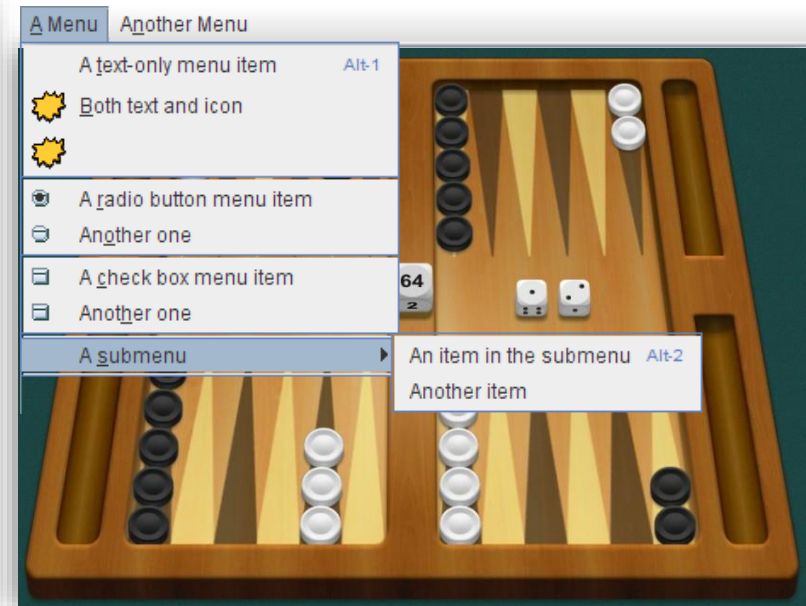
- Genomgående exempel: Analog klocka

- **public class** ClockComponent  
**extends** JComponent { ... }
- Bakgrund, cirklar, tre visare



- Projekt: Ofta spel

- Ofta egen komponent för spel-GUI, plus standardkomponenter (menyer osv)

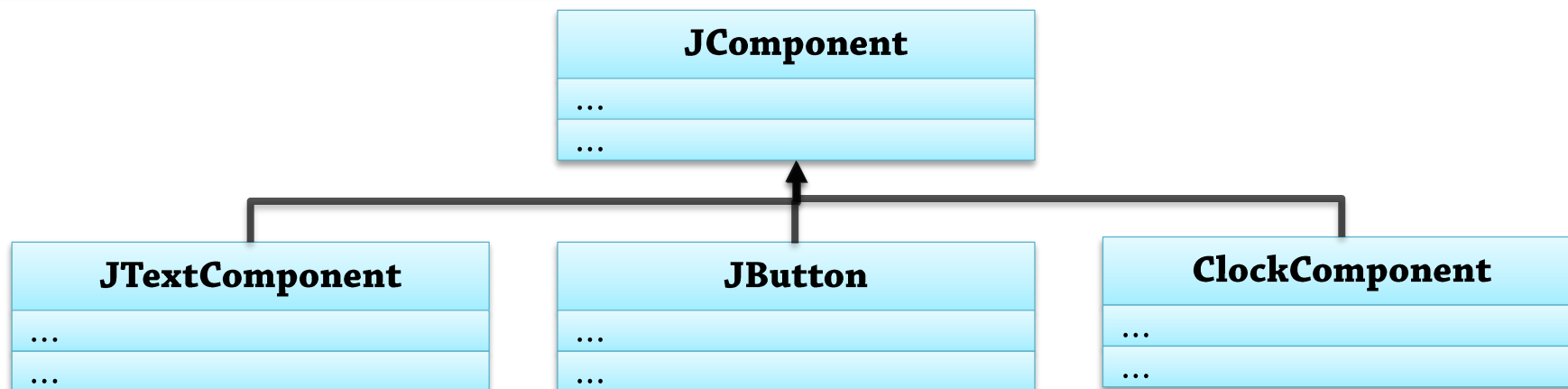
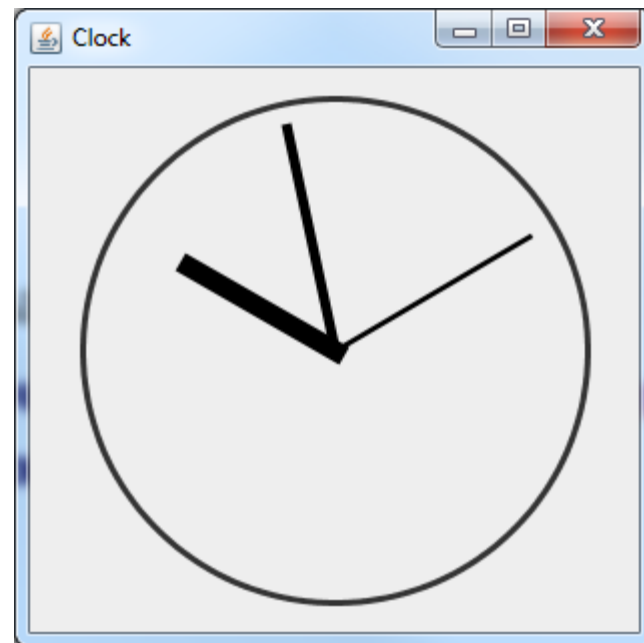


# Egna komponenter och `paintComponent()`

# Rita 1: Egen komponent

- Komponenter: Subklasser till **JComponent**

```
public class ClockComponent extends JComponent {  
    // Which time should be shown?  
    private LocalTime time;  
  
    // Construct a clock component  
    public ClockComponent(LocalTime time) {  
        this.time = time;  
    }  
    ...  
}
```

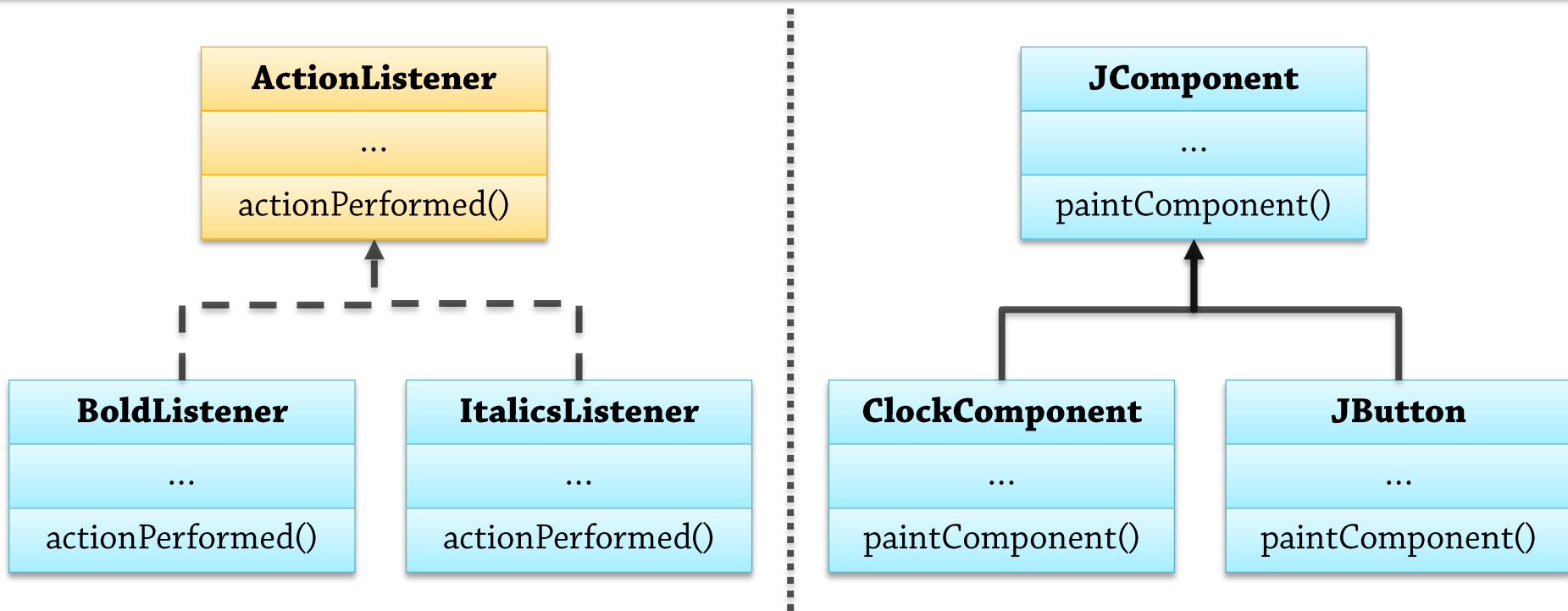


## Hur kan vi *rita* en ClockComponent?

**Liknar ActionListeners:**

**Vi implementerar en *callback*-metod (i ett objekt)**

**Swing anropar metoden vid behov – från händelsehanteringstråden**



**Skillnad: Metoden implementeras direkt i komponentklassen**

# Rita 3: paintComponent()

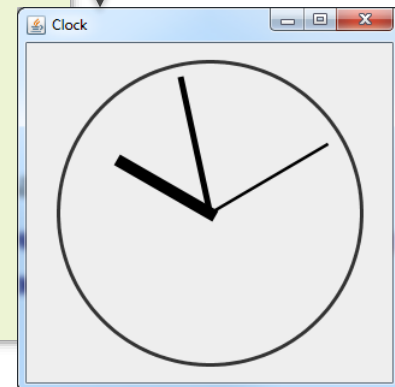


Swings händelsehanteringstråd anropar **paintComponent()** för att rita upp komponentens innehåll...

## Komponenten definierar hur den ska ritas

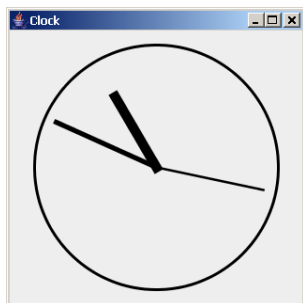
```
public class ClockComponent extends JComponent {  
    private LocalTime time;  
  
    public void paintComponent(Graphics g) {  
        // How should this component look?  
        g.drawOval(... coordinates for circle ...);  
        g.drawLine(... coordinates for hours ...);  
        g.drawLine(... coordinates for minutes ...);  
        g.drawLine(... coordinates for seconds ...);  
    } // Details to follow!  
}
```

Swing ger oss ett **Graphics**-objekt som kan rita på denna komponent

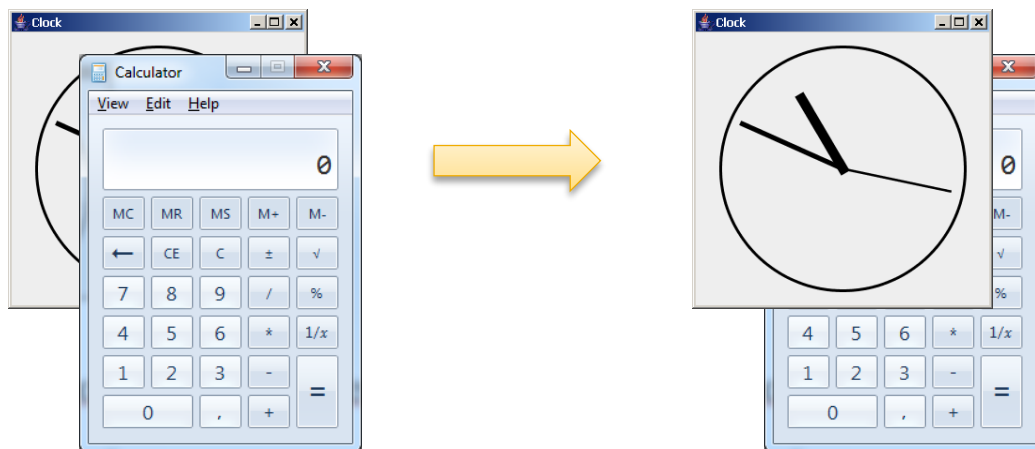


# Rita 4: Swing anropar `paintComponent()`

- Så Swing (inte vi!) anropar **`paintComponent()`**:
  - Den första gången fönstret visas



- När komponenten visas igen efter att ha varit övertäckt





# Rita 5: Vi anropar repaint()



- Om **vi** vill **uppdatera** innehållet:
  - Vi **ber** Swing att anropa **paintComponent()** åt oss

```
public class TextLabel extends JComponent {  
    public void setText(String text) {  
        this.text = text;  
        this.repaint();  
    }  
}
```

Min etikett ändrades →  
be Swing att måla om mig

```
public class GameMechanics {  
    private GameComponent gameComponent;  
    private void stepGameForward() {  
        movePlayers();  
        checkCollisions();  
        gameComponent.repaint();  
    }  
}
```

När vi vill stega framåt:  
be Swing att måla om  
spelkomponenten

Lägger en **begäran** om omritning i "att-göra"-listan  
Hanteras när det finns tid

# Rita 6: Hur stor är jag?



- Layouthanteraren kan bestämma komponentens storlek
  - Komponenten frågar: "Hur stor är jag?"

```
public class ClockComponent extends JComponent {  
    private LocalTime time;  
  
    public void paintComponent(Graphics g) {  
        int myWidth = this.getWidth();  
        int myHeight = this.getHeight();  
        ...  
    }  
}
```

2/25 www.the5thwave.com

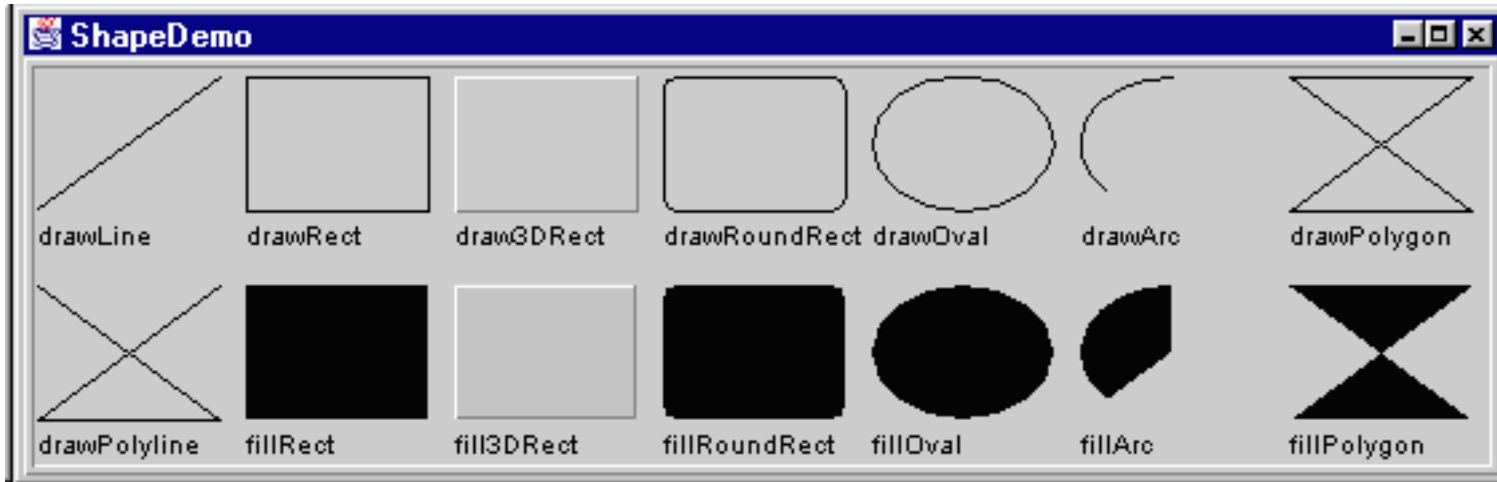
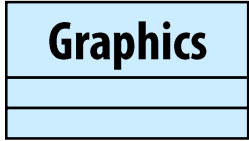
Jeez - that's impressive!  
Let's see that airbrush  
effect again.



Grundläggande  
grafik

# Grafik 1: Graphics-klassen

- **Graphics**: Begränsad repertoar, en metod per "form"



```
public class ClockComponent extends JComponent {  
    public void paintComponent(Graphics g) {  
        ...  
        g.drawOval(... coordinates for circle ...);  
        g.drawLine(... coordinates for hours ...);  
    }  
}
```

# Grafik 2: java.awt.Graphics2D

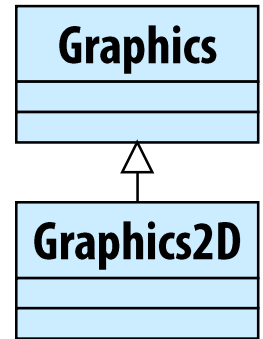


- Nyare subclass **Graphics2D**: Godtyckliga former

- Skapa ett **Shape-objekt**, och anropa:

- `g2.draw(Shape s)`
- `g2.fill(Shape s)`

För allt som implementerar gränssnittet **Shape**, även egna former!



- Likheter med Pythons grafikpaket – och olikheter

```
r = Rectangle(Point(20,20),Point(100,60))
r.draw(win)
```

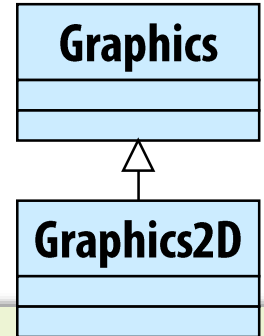
Formen ritar ut sig själv

```
public void paintComponent(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    Shape r = new Rectangle2D.Float(20, 20, 80, 40);
    g2.fill(r);
}
```

Graphics-objektet ritar formen

# Grafik 3: Exempel

- Rita med Graphics2D



```
public class ClockComponent extends JComponent {
    public void paintComponent(Graphics g) {
        ...
        ...
        Graphics2D g2 = (Graphics2D) g;

        Shape myCircle = new Ellipse2D.Double(
            center.x - radius, center.y - radius,
            2 * radius, 2 * radius);
        g2.draw(myCircle);
    }
}
```

Bakåtkompatibilitet:  
*Parameter av typ Graphics...*

Det verkliga värdet är Graphics2D,  
jag lovar!

Skapa ett Shape  
(parametrar av dubbel precision)

## ■ Fördefinierade former:

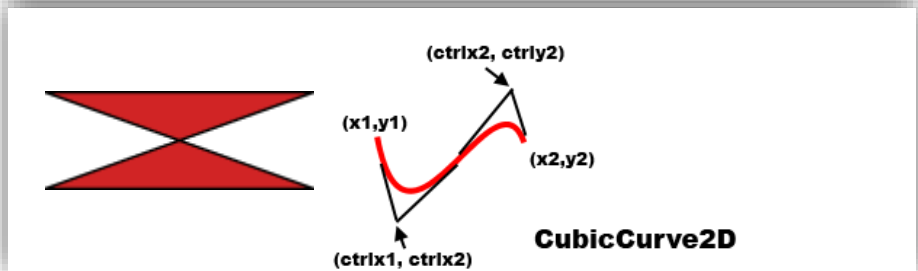
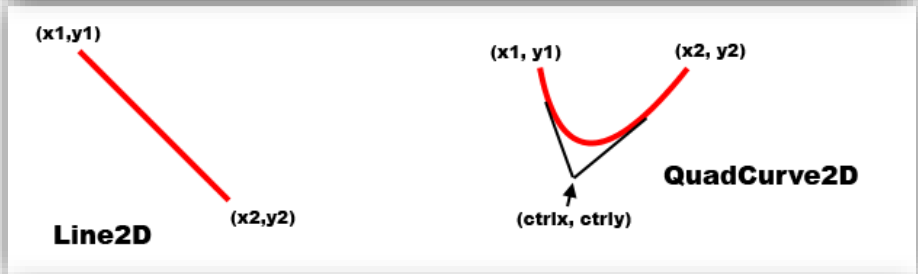
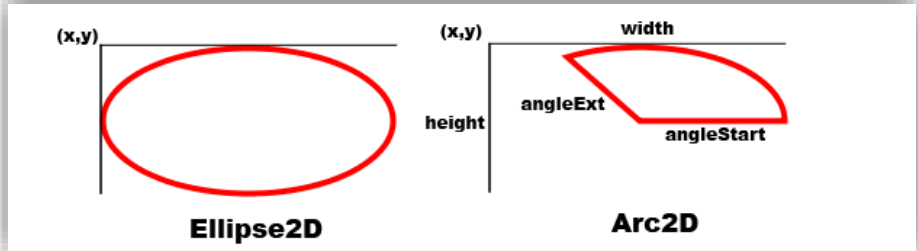
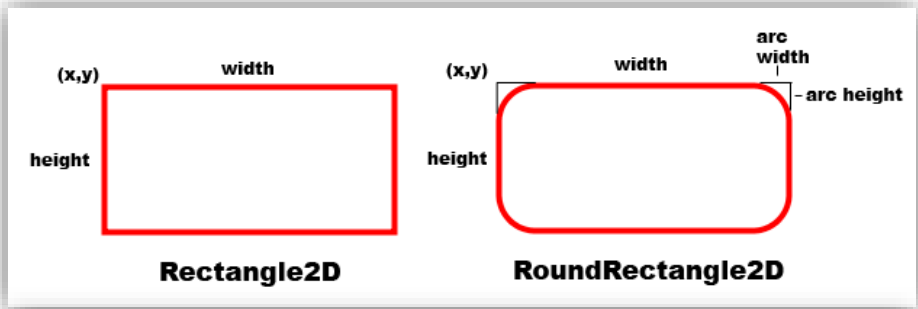
- Rectangle2D,  
RoundRectangle2D

Rectangle2D.Float och  
Rectangle2D.Double, etc.

- Arc2D,  
Ellipse2D

- Line2D,  
QuadCurve2D,  
CubicCurve2D

- Area,  
GeneralPath



- För att rita text:

- Välj en font (**java.awt.Font**)

- Plus *stil* och *punktstorlek*
- `Font myFont = new Font("Arial", Font.BOLD, 12);`  
`g2.setFont(myFont);`

- Använd gärna "logiska" fontnamn

- Dialog, DialogInput
- Serif, SansSerif
- Monospaced, Symbol

→ *mappas till*: Times, Arial? Palatino, Helvetica?



- Använd `drawString()`

- `int x = 100;`  
`int y = 100; // For the baseline!`  
`g2.drawString("Hello World", x, y);`





# Exempel 1: Klockkomponent



## ■ Klockkomponent:

```
class ClockComponent extends JComponent {
    private LocalTime time;

    public ClockComponent(LocalTime time) {
        this.time = time;
    }

    public LocalTime getTime() {
        return time;
    }

    public void setTime(LocalTime time) {
        this.time = time;
    }

    ...
}
```

# Exempel 2: Klockkomponent



```
public void paintComponent(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    Point center = new Point(getWidth() / 2, getHeight() / 2);
    // Fill most of the current size, but leave some margins
    int radius = (int) (0.45 * Math.min(getWidth(), getHeight()));

    // Paint the circle representing the clock
    g2.draw(new Ellipse2D.Double(center.x - radius, center.y - radius, 2 * radius, 2 * radius));

    final double  $\pi$  = Math.PI;
    drawHand(g2, center, 2 *  $\pi$  * (time.getHour() / 12.0),    radius * 2 / 3);
    drawHand(g2, center, 2 *  $\pi$  * (time.getMinute() / 60.0), radius * 9 / 10);
    drawHand(g2, center, 2 *  $\pi$  * (time.getSecond() / 60.0), radius * 9 / 10);
}

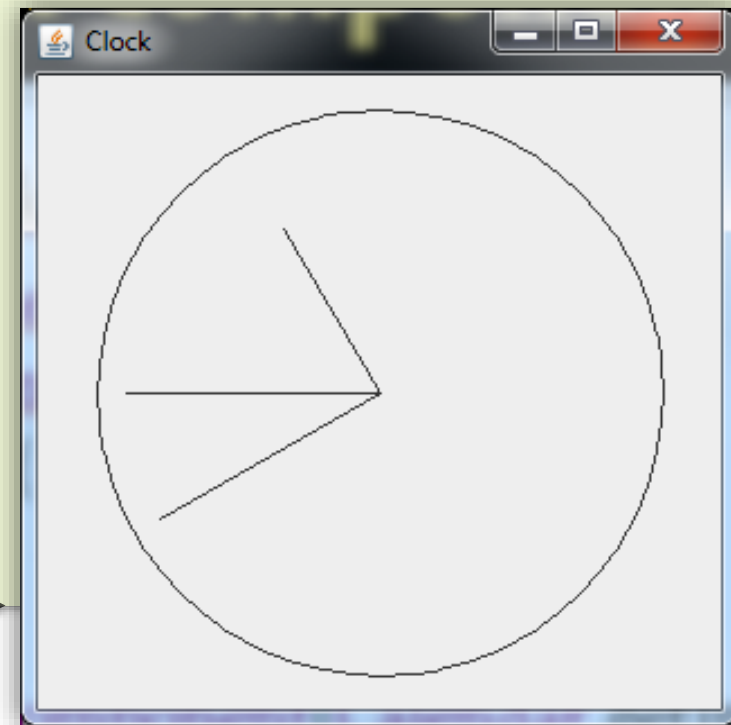
private void drawHand(Graphics2D g2, Point center, double angle, int radius) {
    g2.drawLine(center.x, center.y,
                (int) (center.x + radius * Math.sin(angle)),
                (int) (center.y - radius * Math.cos(angle)));
    // Or: g2.draw(new Line2D.Float(center.x, center.y, ..., ...));
}
```

"Hur hög och bred är jag, klockan,  
på skärmen?"

# Exempel 3: Använda klockkomponent

```
public class Clock {  
    private JFrame frame;  
    private ClockComponent clock;  
  
    public Clock() {  
        // LocalTime.now(): Static method returning the current time  
        clock = new ClockComponent(LocalTime.now());  
  
        frame = new JFrame("Clock");  
        frame.setLayout(new BorderLayout());  
        frame.add(clock, BorderLayout.CENTER);  
        frame.setSize(320, 320);  
        frame.setVisible(true);  
    }  
    public static void main(String[] args) {  
        new Clock();  
    }  
}
```

- Defaultfärg
- Tunna linjer
- "Hackiga" linjer
- ...



**Rita med olika stilar**

# Stilar 1: I Graphics-objektet



- Python's grafikpaket: **Stilar** är del av **formen**

```
r = Rectangle(Point(20,20),Point(100,60))
r.setFill('green')
r.draw(win)
```

- Java: Sätt **stilen** i **Graphics-objektet**

- Gäller för allt som målas därefter

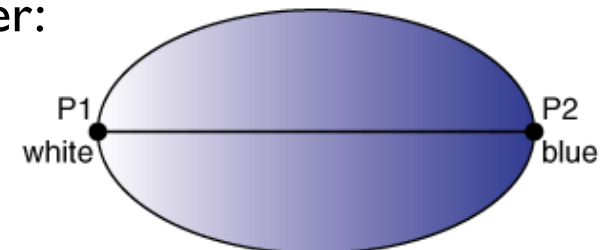
```
public void paintComponent(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    g2.setColor(...);
    g2.setStroke(new BasicStroke(3.5f));           // Pennans tjocklek, default 1
    g2.fill(new Rectangle2D.Float(20, 20, 80, 40));
}
```

- Färger och fyllmönster implementerar **Paint: setPaint(...)**
  - **Color**: Solid färg
    - Statiska fält för några standardfärger: `Color.RED`, `Color.GREEN`, `Color.BLACK`, ...



- `Color cyan1 = new Color(0.0, 1.0, 1.0); // Between 0.0 and 1.0`  
`Color cyan2 = new Color(0, 255, 255); // Between 0 and 255`
- **GradientPaint** anger en gradient mellan två färger:

**GradientPaint**



# En modifierad klocka

```
public void paintComponent(Graphics g) {  
    ...  
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,  
                        RenderingHints.VALUE_ANTIALIAS_ON);  
    g2.setColor(Color.GREEN);  
    g2.setStroke(new BasicStroke(3.0f));  
    g2.draw(new Ellipse2D.Double(center.x - radius, center.y - radius, 2*radius, 2*radius));  
    ...  
    ...  
    drawHand(g2, radius * 0.08f, center, 2 *  $\pi$  * (time.getHour() / 12.0), radius * 2 / 3);  
    drawHand(g2, radius * 0.04f, center, 2 *  $\pi$  * (time.getMinute() / 60.0), radius * 9 / 10);  
    drawHand(g2, radius * 0.02f, center, 2 *  $\pi$  * (time.getSecond() / 60.0), radius * 9 / 10);  
}  
  
private void drawHand(Graphics2D g2, float width, Point center,  
                      double angle, int radius) {  
    g2.setColor(Color.BLACK);  
    g2.setStroke(new BasicStroke(width));  
    ...  
}
```

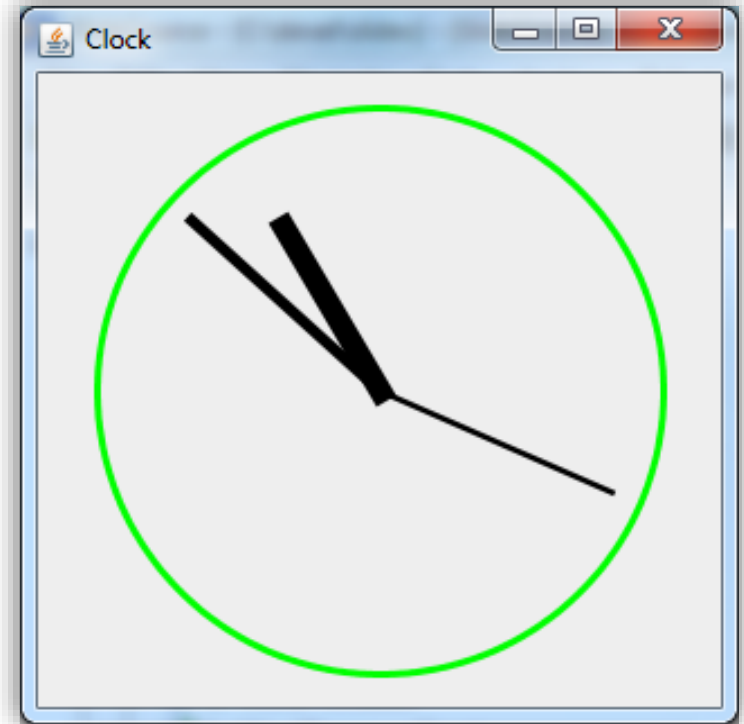
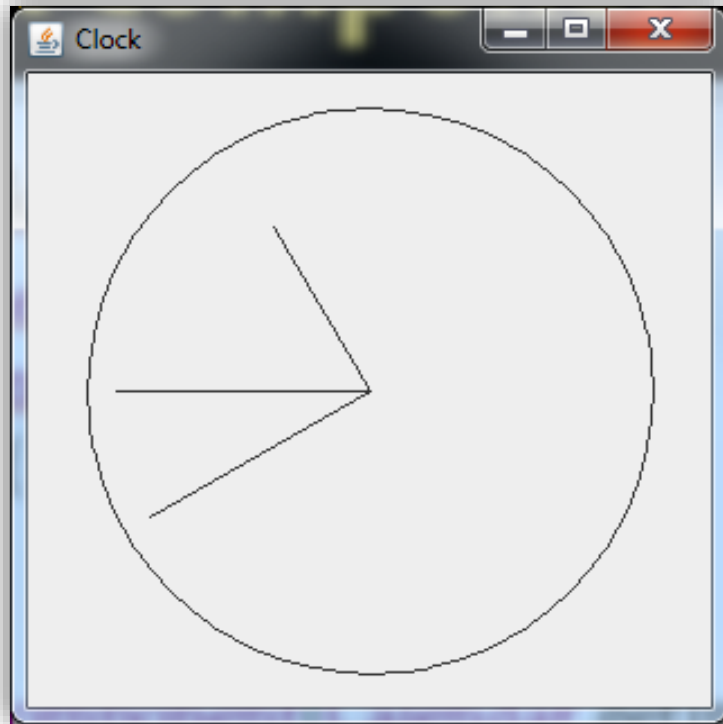
Kantutjämning på

Grön cirkel, tjockare penna

Ange linjetjocklek

# En modifierad klocka (2)

- Resultat:



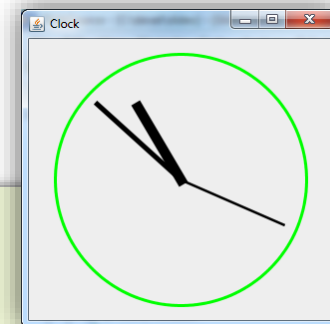


# Regelbundna uppdateringar

# Regelbundna uppdateringar 1



- Än så länge står klockan stilla – även om den ritas om
  - För att uppdatera regelbundet: Skapa uppdateringslyssnare



```
// Skapa en klocka
```

```
clock = new ClockComponent(LocalTime.now());
```

```
// Skapa en handling som uppdaterar klockan om och när den anropas
```

```
final ActionListener updater = new ActionListener() {  
    public void actionPerformed(ActionEvent event) {  
        clock.setTime(LocalTime.now());  
        clock.repaint();  
    }  
};
```

Eller uppdatera ett helt speltillstånd: Flytta spelare ett steg, testa kollisioner, ...

```
// Var 500:e millisekund, be Swing att "anropa updater så snart som möjligt"
```

```
// Sker i händelsehanteringstråden
```

```
clockTimer = new javax.swing.Timer(500 /*milliseconds*/, updater);
```

```
// Om vi är långsamma och en begäran redan ligger i kön: Hoppa över
```

```
clockTimer.setCoalesce(true);
```

```
clockTimer.start();
```

# Regelbundna uppdateringar 2



```
import javax.swing.Timer;
public class Clock02 {
    private JFrame frame;
    private ClockComponent clock;
    private Timer clockTimer;
    public Clock02() {
        clock = new ClockComponent(LocalTime.now());
        frame = new JFrame("Clock");
        frame.setLayout(new BorderLayout());
        frame.add(clock, BorderLayout.CENTER);
        frame.setSize(320, 320);
        frame.setVisible(true);

        final ActionListener updater = new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                clock.setTime(LocalTime.now());
                clock.repaint();
            }
        };
        clockTimer = new Timer(500 /*milliseconds*/, updater);
        clockTimer.setCoalesce(true);
        clockTimer.start();
    }
}
```

Olika paket har klasser med samma namn!

javax.swing.Timer → använder Swings  
händelsehanteringstråd (viktigt i GUI)!

# Regelbundna uppdateringar 3



```
import javax.swing.Timer;
public class Clock02 {
    private JFrame frame;
    private ClockComponent clock;
    private Timer clockTimer;
    public Clock02() {
        clock = new ClockComponent(LocalTime.now());
        frame = new JFrame("Clock");
        frame.setLayout(new BorderLayout());
        frame.add(clock, BorderLayout.CENTER);
        frame.setSize(320, 320);
        frame.setVisible(true);

        final ActionListener updater = event -> {
            clock.setTime(LocalTime.now());
            clock.repaint();
        };

        clockTimer = new Timer(500 /*milliseconds*/, updater);
        clockTimer.setCoalesce(true);
        clockTimer.start();
    }
}
```

Överkurs:  
Enklare med lambdafunktion

# Mer under "vanliga lösningar"



jonkv@ida

https://www.ida.liu.se/~TDDD78/labs/2017/tasks/#images

140% Search

## Jag vill...

- Göra något med strängar
- Göra något med listor
- Skapa en mappning, som *dict*
- Skapa menyer
- Fråga användaren något
- Reagera på tangenttryckningar
- Stänga ett fönster
- Rita ut en bitmap-bild
- Måla med texturer
- Måla genomskinligt
- Måla med kantutjämning
- Hitta typsnitt
- Upptäcka kollisioner på skärmen
- Spela upp ett ljud
- Spara eller skicka hela objekt

## Rita ut en bitmap-bild

Java har flera olika sätt att hantera bitmapbilder (JPEG, PNG med mera), beroende på hur avancerad funktionalitet man behöver.

När Java introducerades, i mitten av 90-talet, fanns klassen `java.awt.Image`. Det finns numera också varianter som `BufferedImage` som låter oss manipulera bilder i minnet. Men `Image` anpassades till användning i applets, Java-program som kör på websidor, och till dåtidens långsamma uppkopplingar. Därför kan den läsa in en bildfil från nätet, och den läser alltid *asynkront* och *inkrementellt* så man kan rita ut den del som har kommit fram hittills. Det gör också att de här klasserna är lite komplicerade att använda.

Istället fokuserar vi på `javax.swing.ImageIcon`, som implementerar `Icon`-gränssnittet. Den är enklare att hantera och klarar allt man behöver för typiska kursprojekt. En `ImageIcon` kan läsas in från minnet, från disk eller via HTTP när man anger en `URL`:

```
new ImageIcon(byte[] data)
new ImageIcon(String filename)
new ImageIcon(URL location)
```

Användbara metoder inkluderar:

```
int getIconHeight()
int getIconWidth()
void paintIcon(Component c, Graphics g, int x, int y)
```

Vi börjar med ett enkelt exempel. Notera att en bild läses in en gång för alla, och en pekare lagras i ett fält i klassen. Vi vill ju inte läsa in bilden varje gång den ska ritas ut! I `paintComponent()` väljer vi att slå på `antialiasing` och sedan rita ut bilden på position (50,50). Resten av koden, i `main()`, öppnar helt enkelt ett fönster och lägger in en `IconPainter01`-komponent där.

Resurser

# Resurser 1: Data på samma plats som kod



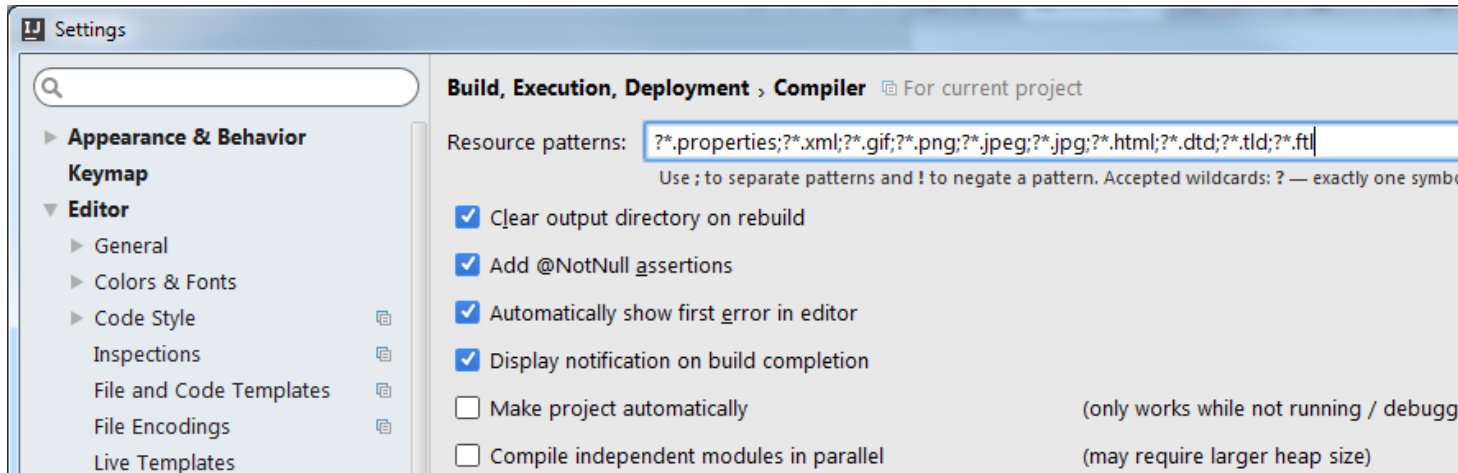
- Hur kan programmet **hitta sina egna** bildfiler, ljudfiler, osv?
  - Program kan installeras på **olika platser**, särskilt med olika OS
  - Hela programmet kan **packas** i en JAR-fil
    - Java Archive, i princip en ZIP-fil
- **Resurser** laddar *datafiler* på samma sätt som *kod*
  - Se till att filerna finns på *samma plats* som den *kompilerade* koden
  - Be *klassladdaren* att hitta filerna åt dig

Allt i en enda JAR-fil:  
mypackage/Main.class  
mypackage/ShowImage.class  
...  
data/bin.dat  
img/test.png

# Resurser 2: Få med datafiler i kompileringen



- I IDEA kan man ange vilka filer som är resurser
  - 1) Placera under *source*-mappen; ange *namnmönster* (?\*.jpg;?\*.\*png)



- 2) Placera i annan mapp; högerklicka; "mark directory as / resources root"
- Vid kompilering *inkluderar* IDEA de utvalda filerna
  - Annat kommer inte med i programmet



# Resurser 3: Användning



- **package** mypackage;

```
public class Main {
```

```
    public void getImage() {
```

```
        String name = "/img/test.png";
```

```
        URL url = ClassLoader.getSystemResource(name);
```

**Klassladdaren** kan tala om hur man hittar resursfiler på samma ställe som klasser

Resultat: URL som pekar ut resursen. Exempel:

```
"jar:file:/C:/devel/teach/java/out/artifacts/SlideSource_jar/SlideSource.jar!/img/test.png"
```

```
        ImageIcon img = new ImageIcon(url);
```

```
    }
```

```
}
```

Låt ImageIcon-klassen läsa från given URL

```
mypackage/Main.class  
mypackage/ShowImage.class  
...  
data/bin.dat  
img/test.png
```

```
▪ package mypackage;  
public class Main {  
    public void getData() {  
        String name = "/data/bana1.json";  
        URL url = ClassLoader.getSystemResource(name);  
        InputStream is = url.openStream();  
        // ...  
    }  
}
```

Alternativ: Öppna, och läs  
som från en vanlig fil

```
mypackage/Main.class  
mypackage/ShowImage.class  
...  
data/bin.dat  
img/test.png
```