- Start-Tags von Elementen können Attribute enthalten, die Zusatz-Informationen für die "Darstellung" des Elements enthalten.
- Der Wert eines Attributs kann (u.a.) ein String sein.
- Das Attribut `href` des Elements `a` gibt eine Internet-Adresse an:

```
<a href="Basic.java">The source.</a>
```

# Einige Attribute des Elements `applet`:

- `codebase` — gibt das Verzeichnis an, in dem das Applet liegt;
- `documentbase` — gibt ein weiteres Verzeichnis an;
- `code` — gibt die Datei an, in der ausführbare Java-Code abgespeichert ist;
- `width` — gibt die Breite der verfügbaren Fläche an;
- `height` — gibt die Höhe der verfügbaren Fläche an.

... und jetzt das Applet selber:

```
import java.applet.Applet;
import java.awt.*;
public class Basic extends Applet {
   public void init() {
      showStatus("... no variables to initialize!");
   }
   public void start() {
      setBackground(Color.orange);
   }
   public void stop() {
      showStatus("... stopping the Basic Applet!");
   }
   public void destroy() {
      showStatus("... destroying the Basic Applet!");
   }
   ...
```

- Ein neues Applet erweitert die Klasse `java.applet.Applet`.
- Ein Applet braucht nicht über eine Klassen-Methode `main()` zu verfügen.
- Aufrufen des Applets durch das Element `applet` einer HTML-Seite führt die folgenden Aktionen aus:

  1. Auf der Seite wird dem Applet eine Fläche zur Verfügung gestellt, die entsprechend den Attribut-Werten `width` und `height` dimensioniert ist.
  2. Ein Objekt der Applet-Klasse (hier der Klasse: `Basic`) wird angelegt.
  3. Zur Initialisierung des Objekts wird die Objekt-Methode `init()` aufgerufen.
  4. Dann wird die Objekt-Methode `start()` aufgerufen.
  5. ...

```
    ...
    public void paint(Graphics g) {
       g.setColor(Color.red);
       g.fillRect(50,50,200,100);
       g.setColor(Color.blue);
       g.fillRect(100,100,200,100);
       g.setColor(Color.green);
       g.fillRect(150,150,200,100);
       showStatus("... painting the Basic Applet!");
    }
 } // end of Applet Basic
```

- Die Klasse `java.awt.Graphics` ist eine abstrakte Klasse, um Bilder zu erzeugen.
- Jede Implementierung auf einem konkreten System muss für diese Klasse eine konkrete Unterklasse bereitstellen ↑Portierbarkeit.
- Mit dem Applet verknüpft ist ein Objekt (der konkreten Unterklasse) der Klasse `Graphics`.
- Nach Ausführen der Methode `start()` wird das `Graphics`-Objekt `page` des Applets auf dem Fenster sichtbar gemacht.
- Auf dem sichtbaren `Graphics`-Objekt kann nun gemalt werden ...

# Achtung:

- Wenn die Applet-Fläche verdeckt wurde und erneut sichtbar wird, muss die Applet-Fläche neu gemalt werden. Dazu verwaltet Java eine Ereignis-Schlange (event queue).
- Verändern der Sichtbarkeit der Fläche erzeugt ein `AWTEvent`-Objekt, das in die Ereignis-Schlange eingefügt wird.
- Das erste Ereignis ist (natürlich :-) die Beendigung der `start()`-Methode ...
- Das Applet fungiert als Consumer der Ereignisse.
- Es consumiert ein Ereignis “Fenster wurde sichtbar(er)”, indem es den Aufruf `paint(page)` (für das aktuelle Applet) ausführt.

  Dadurch wird das Bild (hoffentlich) wiederhergestellt ...

# Weitere Ereignisse:

- Modifizieren des Browser-Fensters $\Longrightarrow$ die `start()`-Methode wird erneut aufgerufen.
- Verlassen der HTML-Seite $\Longrightarrow$ die `stop()`-Methode wird aufgerufen.
- Verlassen des Browsers $\Longrightarrow$ die `destroy()`-Methode wird aufgerufen.

# Nützliche Objekt-Methoden der Klasse `applet`:

- `public void showStatus(String status)` schreibt den String `status` in die Status-Zeile des Browsers.
- `public void setBackground(Color color)` setzt die Hintergrundsfarbe.
- `public Color getBackground()` liefert die aktuelle Hintergrundsfarbe.
- `public void setVisible(boolean b)` macht das `Graphics`-Objekt sichtbar bzw. unsichtbar.
- `public boolean isVisible()` teilt mit, ob Sichtbarkeit vorliegt.

## 22.1 Malen mit der Klasse `Graphics`

- Die Klasse `Graphics` stellt eine Reihe Methoden zur Verfügung, um einfache graphische Elemente zu zeichnen, z.B.:
  - `public void drawLine(int xsrc, int ysrc, int xdst, int ydst);` — zeichnet eine Linie von `(xsrc, ysrc)` nach `(xdst, ydst)`;
  - `public void drawRect(int x, int y, int width, int height);` — zeichnet ein Rechteck mit linker oberer Ecke `(x,y)` und gegebener Breite und Höhe;
  - `public void drawPolygon(int[] x, int[] y, int n);` — zeichnet ein Polygon, wobei die Eckpunkte gegeben sind durch `(x[0],y[0])`,..., `(x[n-1],y[n-1])`.

- Diese Operationen zeichnen nur die Umrisse.
- Soll auch das Innere gezeichnet werden, muss man statt `drawXX(...);` die entsprechende Methode `fillXX(...);` benutzen.

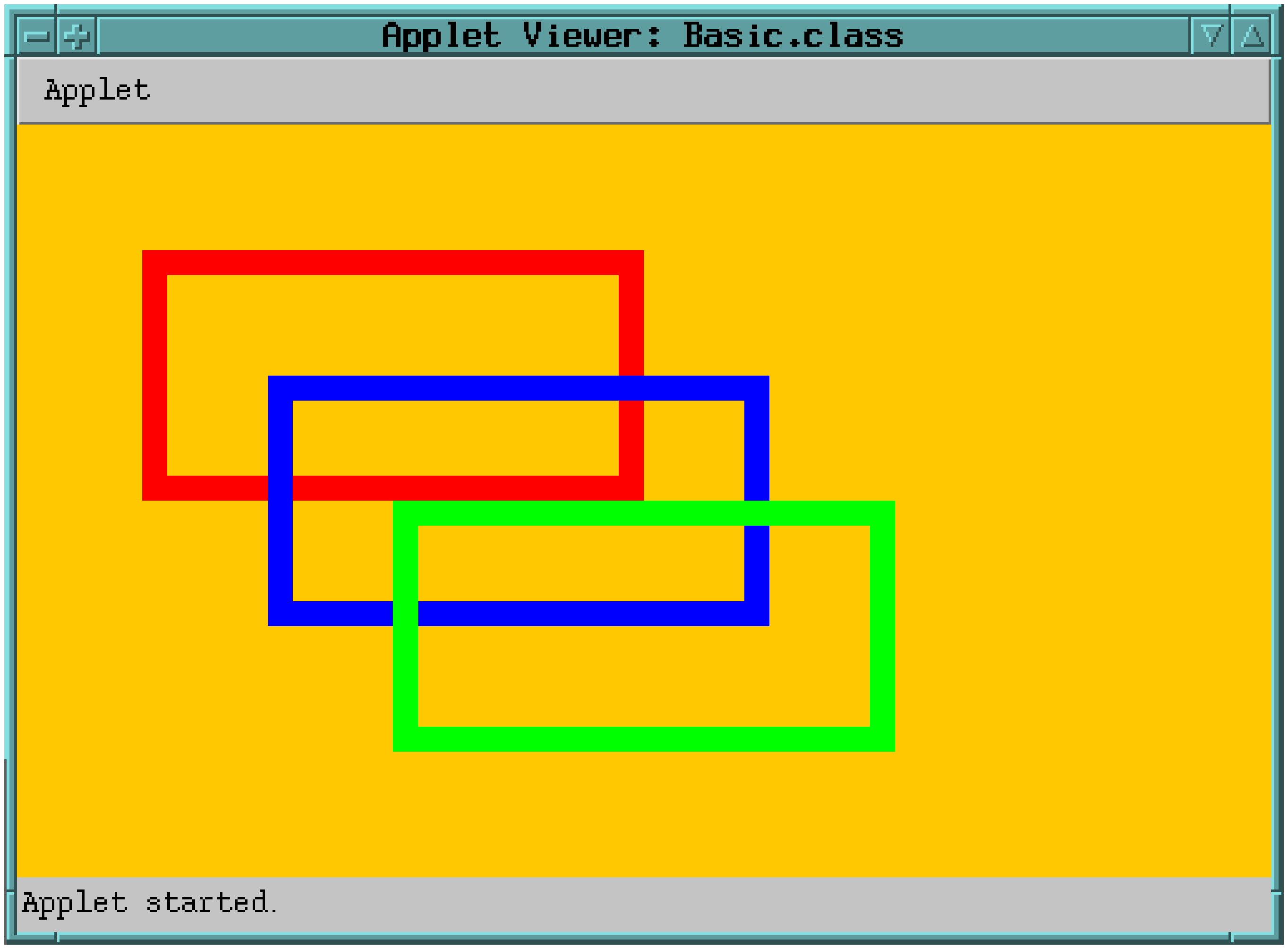
## Achtung:

- Die gemalten Elemente werden sequentiell vor dem Hintergrund des Applets abgelegt.
- Die Farbe, in der aktuell gemalt wird, muss mit der `Graphics`-Objekt-Methode `public void setColor(Color color);` gesetzt werden.

# Beispiel: Rechteck mit Rand

```
import java.awt.*;
public class AuxGraphics {
   private Graphics page;
   public AuxGraphics (Graphics g) { page = g;}
   public void drawRect(int x, int y,
                        int w, int h, int border) {
      page.fillRect(x,y,border,h);
      page.fillRect(x+border,y,w-2*border,border);
      page.fillRect(x+border,y+h-border,w-2*border,border);
      page.fillRect(x+w-border,y,border,h);
   }
}
```

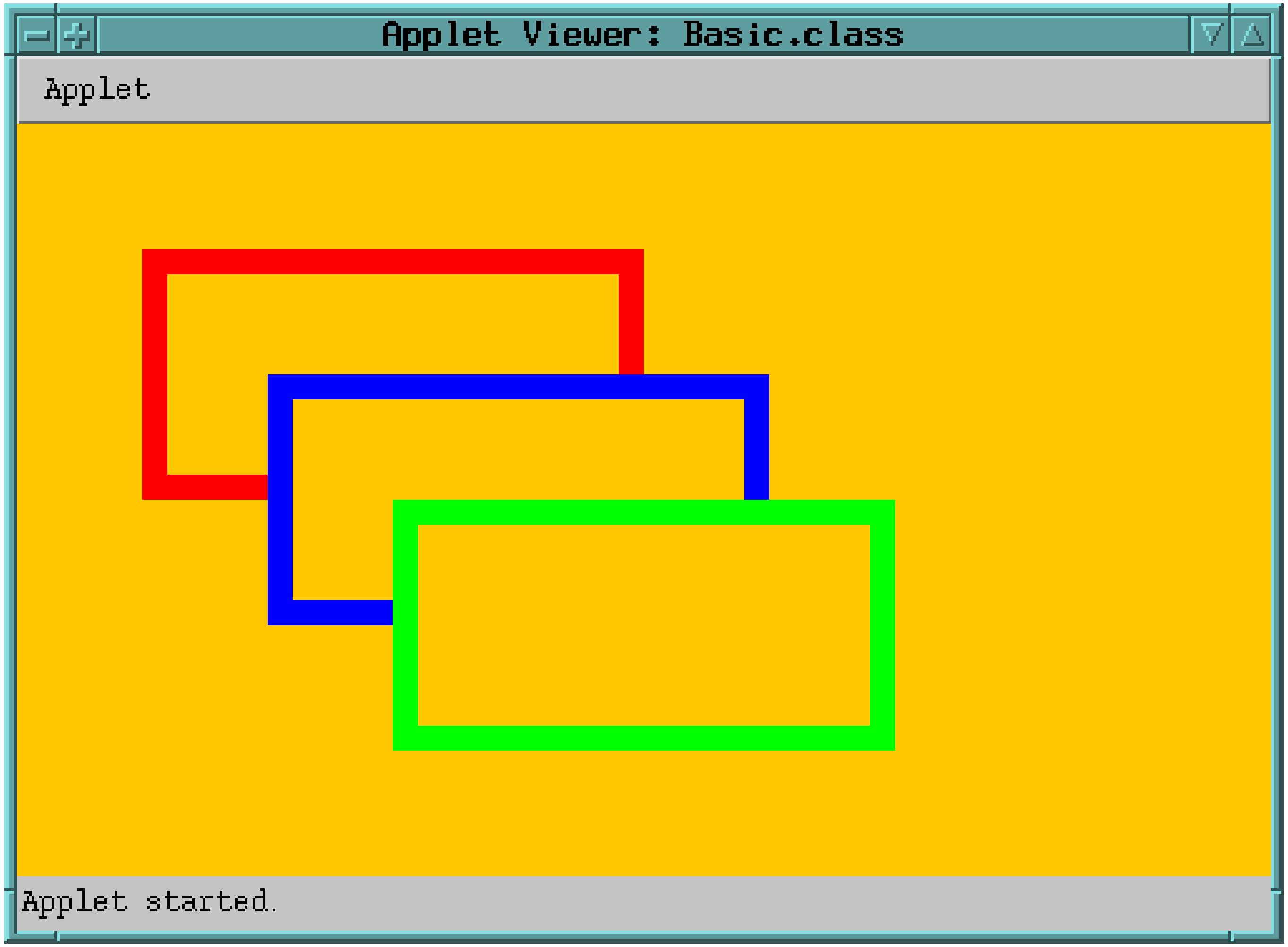
Der Rand wird aus vier kleineren Rechtecken zusammen gesetzt ...


Applet Viewer: Basic.class
Applet
Applet started.

- Man könnte auch auf die Idee kommen, das Innere des Rechtecks durch ein Rechteck in Hintergrunds-Farbe zu übermalen:

```
public void drawRect(int x, int y, int w, int h, int border) {
   Color col = page.getColor();
   page.fillRect(x, y, w, h);
   page.setColor(app.getBackground());
   page.fillRect(x+border,y+border,w-2*border,h-2*border);
   page.setColor(col);
}
```

... mit dem Effekt:


Applet Viewer: Basic.class
Applet
Applet started.

- Seit der Java-Version 1.2 ist das `Graphics`-Objekt, auf dem das Applet malt, aus der Unterklasse `Graphics2D`.
- Diese Klasse bietet tolle weitere Mal-Funktionen :-).
- Insbesondere macht sie den beim Malen verwendeten Strich (Stroke) der Programmiererin zugänglich.
- Statt dem Konzept "Farbe" gibt es nun das Interface `Paint`, das es z.B. gestattet, auch farbliche Abstufungen und Texturen zu malen :-))

## 22.2 Schreiben mit `Graphics`

Um (z.B. auf dem Bildschirm) schreiben zu können, benötigt man eine Schrift (Font).

Eine Schrift ...

- gehört zu einer Schrift-Familie;
- besitzt eine Ausprägung
- ... und eine Größe.

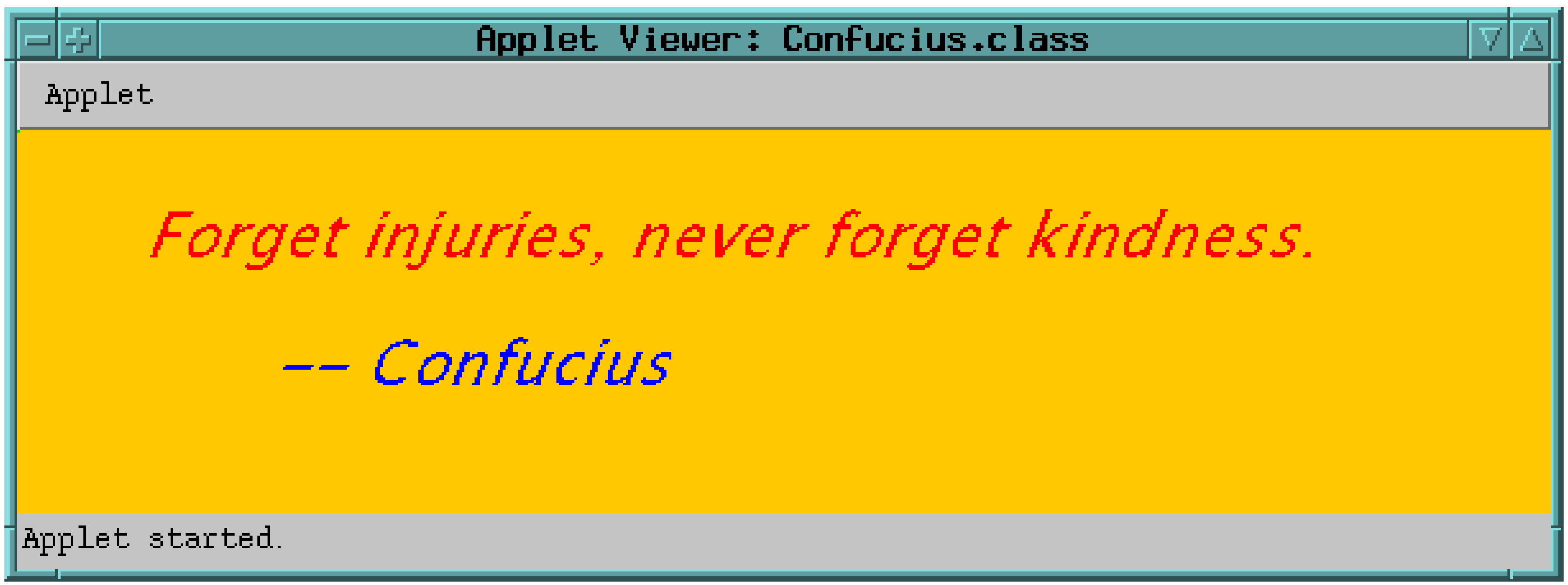
Betrachten wir erstmal ein Applet, das Zeichen mit dem voreingestellten Font darstellt ...

# Beispiel: Das Confucius-Applet

```
import java.applet.Applet;
import java.awt.*;
public class Confucius extends Applet {
   public void paint (Graphics page) {
      setBackground(Color.orange);
      page.setColor(Color.red);
      page.drawString ("Forget injuries, never forget kindness.",
                       50, 50);
      page.setColor(Color.blue);
      page.drawString ("-- Confucius", 70, 70);
   } //     method paint()
}   //      end of class Confucius
```

- `public void drawString(String str, int x, int y);` ist eine Objekt-Methode der Klasse `Graphics`, die den String `str` ab der Position `(x, y)` in der aktuellen Farbe auf den Bildschirm malt.
- Der Effekt:

- Die Qualität der Wiedergabe ist so schlecht, weil
  - die Zeichen klein sind im Verhältnis zur Größe der Pixel;
  - der Screenshot für die Folie skaliert wurde :-)
- Wollen wir ein anderes Erscheinungsbild für die Zeichen des Texts, müssen wir einen anderen Font wählen ...



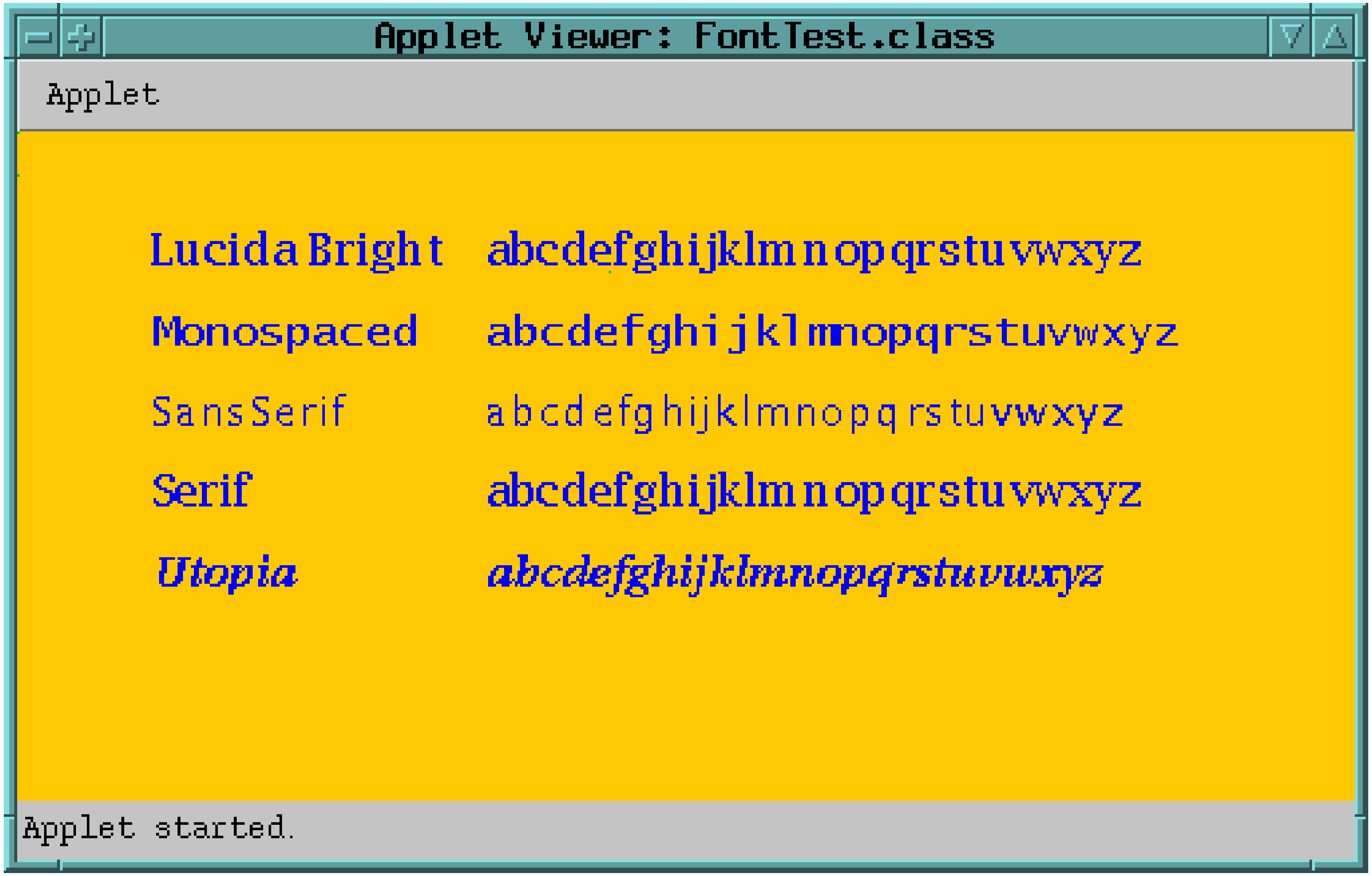
```
import java.applet.Applet;
import java.awt.*;
public class Confucius extends Applet {
   private Font font = new Font("SansSerif",Font.ITALIC,24);
   public void init() {
      setBackground(Color.orange);
      }
   public void paint (Graphics page) {
      page.setColor(Color.red);
      page.setFont(font);
      page.drawString ("Forget injuries, never forget kindness.",
                        50, 50);
      page.setColor(Color.blue);
      page.drawString ("-- Confucius", 100, 100);
   } //      method paint()
}    //      end of class Confucius
```

- Ein Java-Font wird repräsentiert durch ein Objekt der Klasse `Font` (wer hätte das gedacht? ;-)
- Eine Schrift-Familie fasst eine Menge von Schriften zusammen, die gewisse graphische und ästhetische Eigenschaften gemeinsam haben.
- SansSerif zum Beispiel verzichtet auf sämtliche Füßchen und Schwänzchen ...
- Einige Schrift-Familien, die mein Java-System kennt:

  Lucida Bright, Utopia,

  Monospaced, SansSerif, Serif

- Die untere Reihe enthält logische Familien, die verschiedenen konkreten (vom System zur Verfügung gestellten) Familien entsprechen können.


Applet Viewer: FontTest.class
Applet
Lucida Bright abcdefghijklmnopqrstuvwxyz
Monospaced abcdefghijklmnopqrstuvwxyz
SansSerif abcdefghijklmnopqrstuvwxyz
Serif abcdefghijklmnopqrstuvwxyz
Utopia abcdefghijklmnopqrstuvwxyz
Applet started.

- Als Ausprägungen unterstützt Java Normalschrift, **Fettdruck**, *Schrägstellung* und ***fette Schrägstellung***.
- Diesen entsprechen die (`int`-) Konstanten `Font.PLAIN`, `Font.BOLD`, `Font.ITALIC` und `(Font.BOLD + Font.ITALIC)`.
- Als drittes benötigen wir die Größe der Schrift (in Punkten).
- Der Konstruktor `public Font(String family, int style, int size);` erzeugt ein neues `Font`-Objekt.
- Die Objekt-Methoden:
  - `public Font getFont();`
  - `public void setFont(Font f);`
  - `public void drawString(String str, int x, int y);`

... der Klasse `Graphics` erlauben es, die aktuelle Schrift abzufragen bzw. zu setzen und in der aktuellen Schrift zu schreiben (`(x, y)` gibt das linke Ende der Grundlinie an)

## Achtung:

- Für die Positionierung und den Zeilen-Umbruch ist der Programmierer selbst verantwortlich :-)
- Dazu sollte er die Dimensionierung seiner Schrift-Zeichen bzw. der damit gesetzten Worte bestimmen können ...
- Dafür ist die abstrakte Klasse `FontMetrics` zuständig ...

## Ein Beispiel:

```
import java.applet.Applet;
import java.awt.*;
public class FontTest extends Applet {
   private String[] fontNames = {
      "Lucida Bright", "Monospaced",
      "SansSerif", "Serif", "Utopia"
      };
   private Font[] fonts = new Font[5];
   public void init() {
      for(int i=0; i<5; i++)
         fonts[i] = new Font(fontNames[i],Font.PLAIN,16);
   }
   public void start() {
      setBackground(Color.orange);
      }
   ...
```

```
    ...
    public void paint (Graphics page) {
        page.setColor(Color.blue);
        String length; FontMetrics metrics;
        for(int i=0; i<5; i++) {
            page.setFont(fonts[i]);
            page.drawString(fontNames[i],50,50+i*30);
            page.setFont(fonts[3]);
            page.drawString(":",175,50+i*30);
            metrics = page.getFontMetrics(fonts[i]);
            length = Integer.toString(metrics.stringWidth
                     ("abcdefghijklmnopqrstuvwxyz"));
            page.drawString(length,230,50+i*30);
        }
    }   //      method paint
}       //      class FontTest
```

- Die Objekt-Methoden

  ```
  public FontMetrics getFontMetrics();
  public FontMetrics getFontMetrics(Font f);
  ```

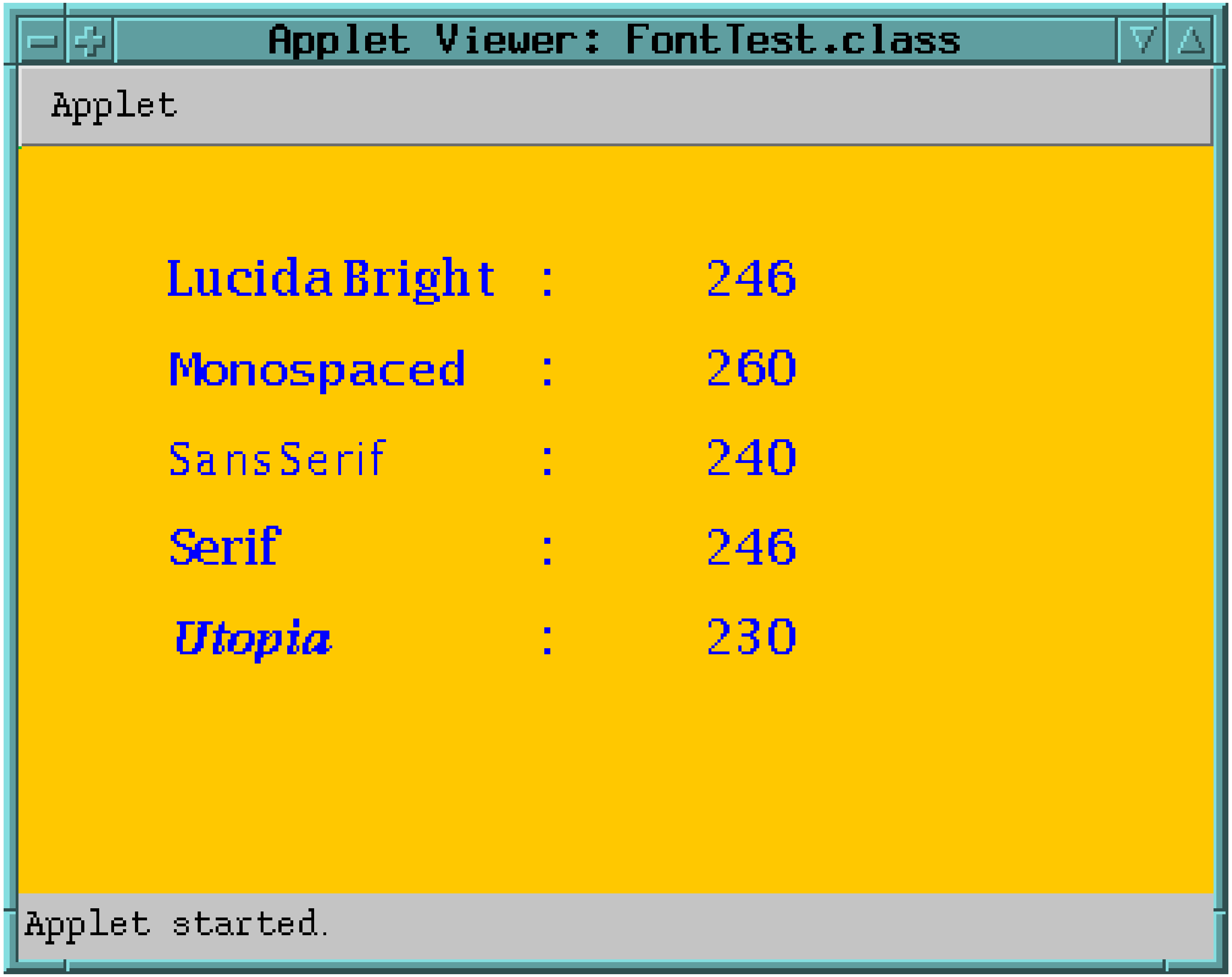
  der Klasse `Graphics` liefern das `FontMetrics`-Objekt für die aktuelle Schrift bzw. die Schrift `f`.

- Die Objekt-Methode `public int stringWidth(String string);` liefert die Länge von `string` in der aktuellen Font-Metrik ...

- Die Klassen `Font` und `FontMetrics` bieten noch viele weitere Einzelheiten bzgl. der genauen Größen-Verhältnisse :-)

- Die Objekt-Methode `public int getHeight();` liefert z.B. die maximale Höhe eines Worts in der aktuellen Schrift.

- Tja, und so sieht dann das Applet im `appletviewer` aus ...


Applet Viewer: FontTest.class
Applet
Lucida Bright : 246
Monospaced : 260
SansSerif : 240
Serif : 246
Utopia : 230
Applet started.