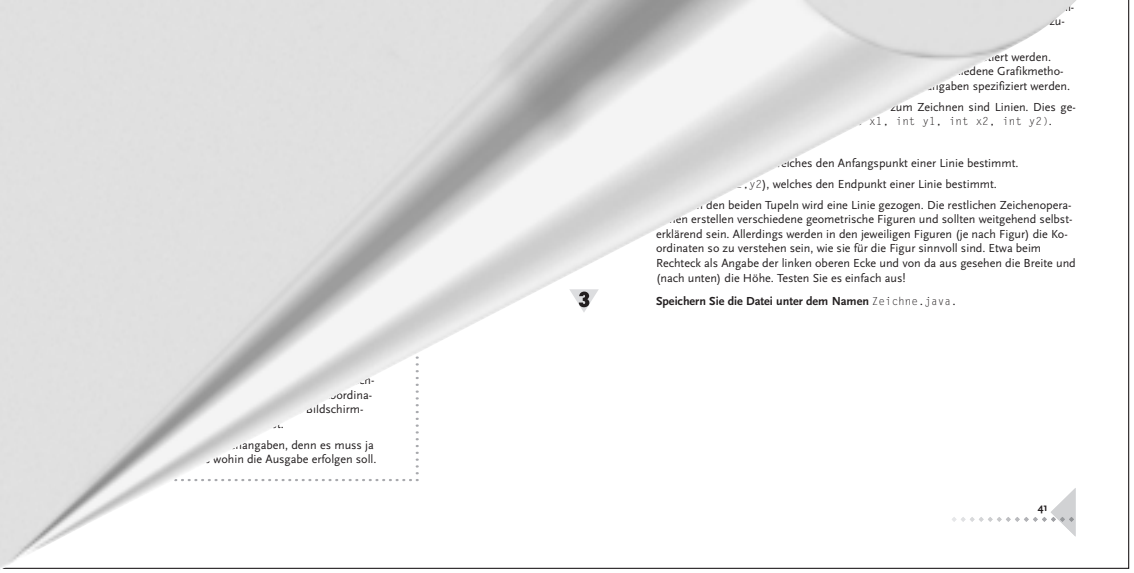
Projekt 3 Grundlegende Grafiktechniken

Eine der für viele Anwender wichtigsten Fähigkeiten von Java ist die Grafikfähigkeit. Dies betrifft sowohl Java-Applets als auch normale Programme. In dem vorangegangenen Projekt wurden ja bereits einige Techniken verwendet. In diesen Projekt sollen einige grafische Möglichkeiten von Java genauer beleuchtet werden. Dabei werden insbesondere die Grafikgrundlagen, einfache Grafikvorgänge, Farben und die Textausgabe mittels Grafik-Methoden behandelt.

1. Grafikgrundlagen
2. Farbangaben
3. Einfache Textausgabe mit grafischen Oberflächen
4. Komplexere Textausgabe über den Zeichnen-Modus



Lektion 1 Grafikgrundlagen

Java ist eine Programmiersprache mit recht eindrucksvollen grafischen Möglichkeiten. Die meisten der schon von Anfang an vorhandenen Grafikfähigkeiten von Java sind in der `Graphics`-Klasse untergebracht, welche ein Bestandteil des Paketes `java.awt` ist. Andere finden Sie in der `Image`-Klasse, welche ebenso zu diesem Paket gehört. Seit Java 2 gibt es noch zahlreiche weitere Pakete, die insbesondere für 2D-Java und Swing wichtig sind.

TIPP

2D-Java und Swing werden im folgenden Projekt behandelt.

Java äußert sich normalerweise bei einer Ausgabe nur im Befehlszeilenfenster. Wenn jedoch Grafik genutzt werden soll, muss eine grafische Bildschirmoberfläche extra angegeben werden. Ein einfacher Weg dazu ist die Verwendung von Applets, welche von Hause aus mit einer grafischen Oberfläche versehen sind. Deshalb sollen die ersten Experimente auch mit Applets durchgeführt werden. Dies ist jedoch nicht zwingend, denn auch jedes Programm mit einer grafischen Oberfläche kann Basis für Zeichenoperationen sein.

TIPP

Es muss zur Ausgabe von Grafik nicht zwingend die paint()-Methode verwendet werden. Diese ist nur eine komfortable Möglichkeit, ohne besonderen Aufwand Grafik-Aktionen durchzuführen.

Zeichnen und neu zeichnen

Allgemeine Zeichenvorgänge in Java erfolgen, indem eine beliebige Grafikmethode innerhalb einer Methode aufgerufen wird. Eine besonders sinnvolle Möglichkeit ist die Methode `paint()`, welche unter anderem in allen Applets automatisch vorhanden ist.

TIPP

Sie können Java veranlassen, ein Applet oder Fenster unabhängig von den beschriebenen Standardereignissen nachzuzeichnen. Dies erfolgt über die repaint()-Methode, die – evtl. mit geringen Verzögerungen – die paint()-Methode ausführt.

Java-Applets bzw. Programme mit grafischer Oberfläche zeichnen sich – sofern Grafikaktionen notwendig sind – selbst neu, indem Sie die `paint()`-Methode aufrufen. Dies ist etwa immer beim ersten Aufruf eines Applets oder Programms der Fall, aber auch jedes Mal dann, wenn das Fenster verschoben oder zwischenzeitlich von einem anderen Fenster überlagert wurde.

Der `paint()`-Methode wird eine Instanz der `Graphics`-Klasse weitergegeben, die für das Zeichnen verschiedener Formen und Bilder zahlreiche Methoden bereitstellt. Die allgemeine Syntax sieht so aus:

```
public void paint(Graphics g)
```

Zum Test der grundlegenden Grafikvorgänge von Java wird nun ein Beispiel erstellt.

Das Koordinatensystem

Damit grafische Ausgaben möglich sind, muss die Position der Ausgabe über ein Koordinatensystem von (in 2-dimensionalem Fall) zwei Werten (sogenannte Vektoren, welche vom Ursprung des Koordinatensystems ausgehen) eindeutig festgelegt werden. Um in dem bei Java verwendeten einfachen, 2-dimensionalen, kartesischen Koordinatensystem einen Punkt festzulegen, sind zwei Angaben notwendig – ein x-Wert und ein y-Wert.

Die obere linke Ecke des Bildschirms (genauer: des Bildschirmbereichs eines Applets oder dem Fenster einer Applikation) wird mit (0, 0) – also dem Koordinatensystem-Ursprung – abgebildet. Der x-Wert ist die Anzahl der Bildschirm-Pixel von links ausgehend, also in der Waagerechten, und y ist die Zahl der Pixel von oben angefangen, also in der Senkrechten. Das Koordinaten-Tupel (42,10) beschreibt z.B. einen Punkt, der 42 Pixel vom linken Rand des Bildschirmbereichs und 10 Pixel vom oberen Rand des Bildschirmbereichs entfernt ist.

Jede Grafik-Methode von Java verwendet irgendwelche Koordinatenangaben, denn es muss ja festgelegt werden, wo eine Ausgabe beginnt und oft noch bis wohin die Ausgabe erfolgen soll.

1 **Starten Sie einen Editor.**

2 **Erstellen Sie die nachfolgende Datei.**

Zeichen-Methoden von Java

```
import java.awt.*;
public class Zeichne extends java.applet.Applet {
public void paint(Graphics g) {
// Zeichne Linien
g.drawLine(100,5,200,200);
g.drawLine(10,5,200,200);
g.drawLine(10,50,180,250);
// Zeichne gefüllte Rechtecke
g.fillRect(300, 300, 100, 100);
g.fillRect(300, 20, 40, 100);
// Zeichne eine Ellipsen/Kreise
g.fillOval(50, 250, 250, 150);
g.fillOval(250, 100, 100, 100);
// Zeichne gefüllte Bogen
g.fillArc(50, 15, 100, 100, 90, 90);
g.fillArc(150, 150, 50, 50, 120, 180);
}}
```

TIPP

Im Quelltext finden Sie Kommentare, die mit // beginnen und die jeweilig folgende Aktion genauer erklären.

TIPP

Es kann auch direkt eine einzelne Klasse importiert werden.

Die erste Anweisung `import java.awt.*;` bedeutet das Importieren von einem Java-Paket. Dies ist nichts weiter als die Möglichkeit, im Laufe des folgenden Quelltextes mit einer verkürzten Schreibweise auf Klassen innerhalb dieses Paketes zugreifen zu können.

WIE BITTE?

Ein Paket ist die Zusammenfassung von mehreren Java-Klassen.

Die Klasse `Graphics` müsste sonst mit `java.awt.Graphics` notiert werden. Über das aus dieser Klasse erzeugt Objekt `g` kann auf verschiedene Grafikmethoden zugegriffen werden, die allesamt mit Koordinatenangaben spezifiziert werden.

Die einfachste Technik der `Graphics`-Klasse zum Zeichnen sind Linien. Dies geschieht mit der Methode `drawLine(int x1, int y1, int x2, int y2)`.

Sie hat zwei Koordinatenpaare:

- das Tupel (`x1,y1`), welches den Anfangspunkt einer Linie bestimmt,
- das Tupel (`x2,y2`), welches den Endpunkt einer Linie bestimmt.

Zwischen den beiden Tupeln wird eine Linie gezogen. Die restlichen Zeichenoperationen erstellen verschiedene geometrische Figuren und sollten weitgehend selbsterklärend sein. Allerdings werden in den jeweiligen Figuren (je nach Figur) die Koordinaten so zu verstehen sein, wie sie für die Figur sinnvoll sind. Etwa beim Rechteck als Angabe der linken oberen Ecke und von da aus gesehen die Breite und (nach unten) die Höhe. Testen Sie es einfach aus!

3 **Speichern Sie die Datei unter dem Namen** `Zeichne.java`.

4 **Kompilieren Sie die Datei in DOS-Fenster.**

Da es sich um Applet handelt, wird noch eine HTML-Datei zum Aufruf benötigt.

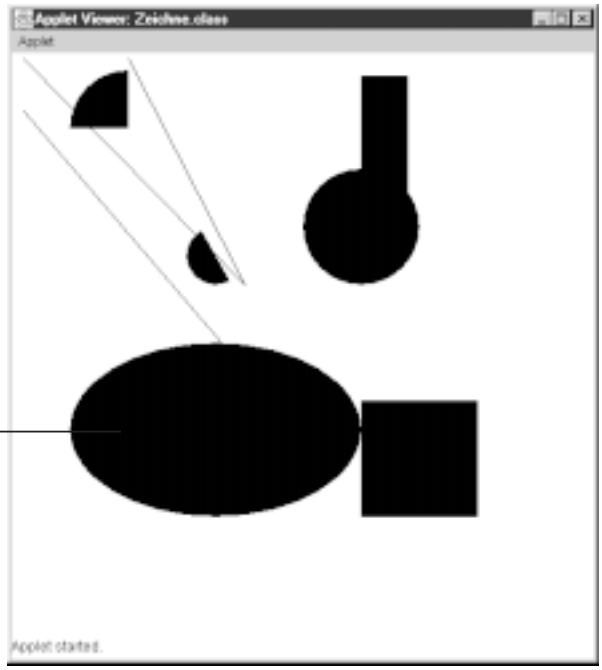
5 **Erstellen Sie die nachfolgende Datei.**

```
<HTML>
<BODY>
<APPLET CODE = "Zeichne.class" WIDTH = 500 HEIGHT = 500>
</APPLET>
</BODY>
</HTML>
```

Die Applet-Referenz

6 **Speichern Sie diese Datei unter dem Namen** `Zeichne.HTML`

7 **Starten Sie die Datei im Browser oder Appletviewer.**



Ausgabe der Grafikmethoden

Wie Ihnen sicher auffällt, werden alle Zeichenvorgänge in einer einzigen (Default-) Farbe erstellt. Dies ist natürlich nicht zwingend. Deshalb folgt als nächstes die Technik zum Zugriff auf Farben unter Java.

Lektion 2 Farbangaben

Java setzt Farben aus so genannten Primärfarben des Lichts zusammen. Dies sind die Farben Rot, Grün und Blau. Man nennt dieses Farbmodell das RGB-Modell (Red-Green-Blue). Das bedeutet, dass Farben aus der Summe von Anteilen aus rotem, grünem und blauem Licht zusammengesetzt werden.

Sie definieren eine Farbe im RGB-Modell, indem Sie angeben, wie viel rotes, grünes und blaues Licht in der Farbe enthalten ist. Sie können dies entweder mit einer Zahl zwischen 0 und 255 oder mit Gleitkommazahlen zwischen 0,0 und 1,0 angeben. Um ein Objekt in einer bestimmten Farbe zeichnen und darstellen zu können, können Sie eine Instanz der `Color`-Klasse erzeugen. Hier sollen drei Arten behandelt werden, mit denen Sie ein Farbobjekt erzeugen können.

TIPP

Schwarz entsteht, wenn es kein Licht gibt, weiß wird mit der Kombination aller Primärfarben gebildet: Rot + grün + blau (in gleicher Menge) = weiß. Dies ist genau umgekehrt zu dem pigment-basierenden Modell.

- `public Color(int red, int green, int blue)`. Damit wird eine Farbe mit Rot-, Grün- und Blau-Werten zwischen 0 und 255 erzeugt.
- `public Color(int rgb)`. Damit wird eine Farbe mit Rot-, Grün- und Blau-Werten zwischen 0 und 255 erzeugt, bei der jedoch alle Werte zu einem einzigen Integerwert kombiniert werden. Die Bits 16-23 enthalten den Rot-Wert, die Bits 8-15 enthalten den Grün-Wert und die Bits 0-7 enthalten den Blau-Wert. Diese Werte werden normalerweise in hexadezimaler Schreibweise notiert, damit Sie die Farbwerte ganz einfach sehen können. `0x123456` würde beispielsweise einen Rot-Wert von 0x12 (18 dezimal) ergeben, einen Grün-Wert von 34 (52 dezimal)) und einen Blau-Wert von 56 (96 dezimal). Jede Farbe in hexadezimaler Schreibweise nimmt genau 2 Zeichen ein.
- `public Color(float red, float green, float blue)`. Damit wird eine Farbe mit Rot, Grün und Blau-Werten von 0.0 und 1.0 erzeugt.

TIPP

Gängige Farben können Sie über in der Color-Klasse definierte Standardfarbobjekte verwenden, auf die Sie per Punktnotation zugreifen können.

Eine Instanz der `Color`-Klasse erstellen Sie also z.B. so:

```
Color pinkColor = new Color(255, 192, 192);
Color MeineFarbe = new Color(0.0, 1.0, 0.25);
```

Wenn Sie einmal eine Farbe erstellt haben oder ein Standardfarbobjekt verwenden wollen, können Sie die Zeichenfarbe mit der Methode `setColor(Color c)` verändern. Als Parameter geben Sie das gewünschte Farbobjekt an.

1 **Starten Sie einen Editor.**

2 **Erstellen Sie die nachfolgende Datei.**

Erstellung von Farbobjekten

```
import java.awt.*;
public class Zeichne2 extends java.applet.Applet {
public void paint(Graphics g) {
// Erzeugen von Farbobjekten
Color farbe1 = new Color(255, 100, 92);
Color farbe2 = new Color(155, 192, 192);
Color farbe3 = new Color(55, 192, 192);
Color farbe4 = new Color(5, 12, 120);
Color farbe5 = new Color(255, 192, 192);
g.setColor(farbe1);
g.fillRect(5, 5, 150, 250);
// Zeichne eine Ellipse
g.setColor(farbe2);
g.fillOval(50, 15, 250, 150);
// Zeichne einen Kreis
g.setColor(farbe3);
g.fillOval(250, 150, 150, 150);
// Zeichne einen gefüllten Bogen
g.setColor(farbe4);
g.fillArc(50, 15, 100, 100, 90, 90);
// Zeichne einen weiteren gefüllten Bogen in einer anderen
Farbe
g.setColor(farbe5);
g.fillArc(150, 150, 50, 50, 120, 180);
// direkte Verwendung einer Farbkonstanten
g.setColor(Color.green);
g.drawRect(40,40,100,200);
}}
```

3 **Speichern Sie diese Datei unter dem Namen** `Zeichne2.java`.

4 **Wechseln Sie in das DOS-Fenster, und kompilieren Sie die Datei.**

```
javac Zeichne2.java.
```

5 **Erstellen Sie die nachfolgende HTML-Datei.**

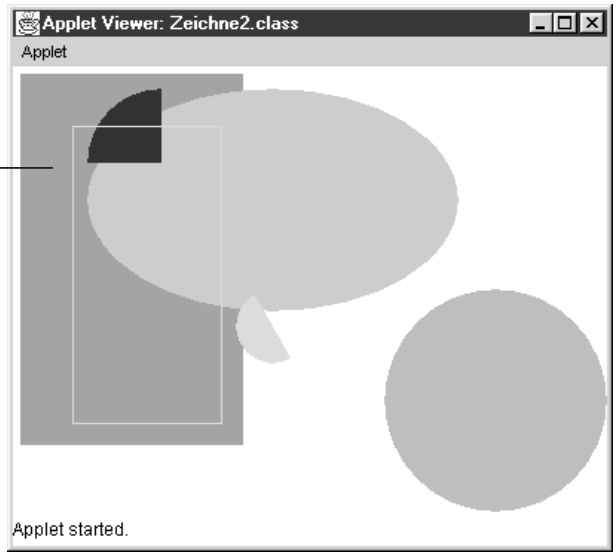
Applet-Referenz

```
<HTML>
<BODY>
<APPLET CODE = "Zeichne2.class" WIDTH = 400 HEIGHT = 300>
</APPLET>
</BODY>
</HTML>
```

6 **Speichern Sie diese Datei unter dem Namen** `Zeichne2.HTML`.

7 **Starten Sie die Datei im Browser oder Appletviewer.**

Farbige Grafikobjekte



Farbangaben sind nicht nur für Zeichenaktionen einstellbar. Ebenso können Sie mit einem Farbobjekt Hintergrundfarben und Vordergrundfarben pauschal setzen. Dazu gibt es die Methoden `setBackground(Color c)` und `setForeground(Color c)`. Sie gehören zur Klasse `java.awt.Component`. Als Parameter geben Sie wieder das gewünschte Farbobjekt an.

Das Abrufen von Farbinformationen funktioniert mit einer ganzen Reihe von Methoden, welche allesamt mit `get` beginnen. Z.B. liefert die Methode `public int getRed()` den Rotanteil einer Farbe als Integer-Wert zurück, `public int getGreen()` den Grünanteil oder die `public int getBlue()`-Methode den Blauanteil.

3 Lektion — Einfache Textausgabe auf grafischen Oberflächen

Die `Graphics`-Klasse verfügt nicht nur über verschiedene Arten von Methoden zum Zeichnen von Grafiken, sondern ist auch die Basisklasse für die Textausgabe auf grafischen Oberflächen. Sie enthält Methoden zum Zeichnen von Text in verschiedenen Schriften und Schriftstilen.

Wie bitte?

Unter einer grafischen Oberfläche werden auch Texte gezeichnet, nicht geschrieben.

1 **Starten Sie einen Editor.**

2 **Erstellen Sie die nachfolgende Datei.**

Textausgabe auf einer grafischen Oberfläche

```
import java.awt.Graphics;
public class Zeichne1 extends java.applet.Applet {
public void paint(Graphics g) {
g.drawString("Die Anwort ist 42!", 70, 40);
}}
```

3 **Speichern Sie diese Datei unter dem Namen** `Zeichne1.java`.

Um nun mit der `Graphics`-Klasse Text auszugeben, rufen Sie `drawString (String str, int x, int y)` auf und übergeben als String die Zeichenkette, sowie die x- und y-Koordinaten für den Anfang der darzustellenden Zeichen.

4 **Kompilieren Sie die Datei in der DOS-Box.**

```
javac Zeichne1.java
```

5 **Erstellen Sie die nötige HTML-Datei im Editor.**

Die Applet-Referenz

```
<HTML>
<BODY>
<APPLET CODE = "Zeichne1.class" WIDTH = 400 HEIGHT = 300>
</APPLET>
</BODY>
</HTML>
```

6 **Speichern Sie die Datei unter dem Namen** `Zeichne1.HTML`.

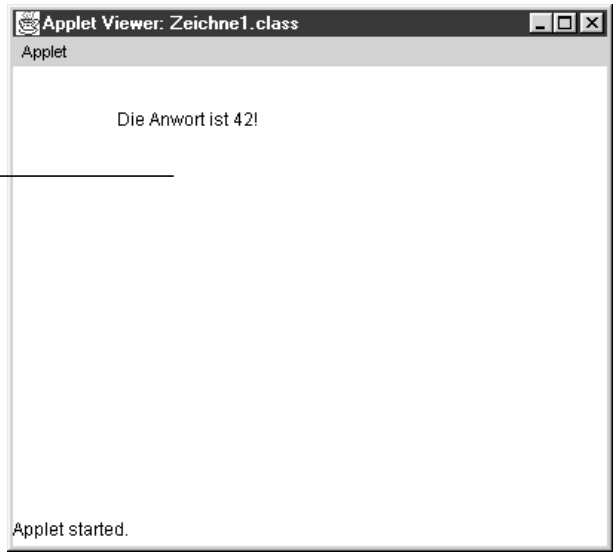
7 **Starten Sie die Datei z.B. im Browser.**

Dies geht wie üblich mit einem Doppelklick zum Anzeigen im Browser oder natürlich auch im Appletviewer.

Wie bitte?

Die Koordinaten definieren genau genommen den Beginn einer gedachten Grundlinie, auf der der darzustellende Text aufsitzt.

8 **Kontrollieren Sie das Ergebnis.**



Gezeichneter Text

4 Lektion Komplexere Textausgabe über den Zeichnen-Modus

Neben der `Graphics`-Klasse mit Methoden zum Textzeichnen spielen bei der Textausgabe die `Font`-Klasse und die `FontMetrics`-Klasse eine Rolle. Die `Font`-Klasse stellt bestimmte Fonts dar (Name, Stil, Punktgröße), während `FontMetrics` Informationen über den Font enthält wie die tatsächliche Höhe und Breite eines bestimmten Zeichens.

Eine Möglichkeit, den Text formatiert auf dem Bildschirm ausgeben zu können, besteht darin, zuerst eine Instanz der Klasse `Font` zu erstellen. Fontobjekte stellen einen einzelnen Font dar. Fontnamen sind Zeichenketten, welche die Familie der Schrift bezeichnen (Arial, Times Roman, ...).

Fontstile sind Konstanten, welche in der `Font`-Klasse definiert sind und die Sie über die übliche Punktnotation ansprechen können. Die Punktgröße ist die Größe der betreffenden Schriftart entsprechend der im Schrifttyp enthaltenen Definition. Wenn Sie eine Schrift auswählen, müssen Sie die Punktgröße der Schrift angeben.

Punktgröße

Punktgröße ist ein Begriff aus dem Druckwesen, der sich auf die Größe der Schrift bezieht. Es gibt 100 Punkte pro Inch (dots per inch = dpi) bei einem Drucker; diese Größe gilt aber nicht zwangsläufig auch für den Bildschirm. Ein typischer Punktgrößenwert für Drucktext ist 12 oder 14. Die Punktgröße zeigt nicht die Anzahl der Pixel in der Höhe oder der Breite an. Es handelt sich vielmehr um einen vergleichenden Begriff. Eine Punktgröße von 24 ist zweimal so groß wie die von 12.

Um nun ein Fontobjekt zu erstellen, verwenden Sie beispielsweise die drei beschriebenen Fonteigenschaften als Argumente einer Konstruktormethode für ein solches Objekt. Sie können also eine Schrift zuweisen, indem Sie den Schriftnamen, die Schriftformatierung und die Punktgröße auswählen. Es gilt folgende Syntax:

```
Font f = new Font([Fontname],[ Fontstil],[Punktgrösse])
```

Sie können in Java aus einer Vielzahl verschiedener Schriften auswählen. Es handelt sich um die auf den meisten Plattformen unterstützten Zeichensätze. Daneben gibt es mehrere Schriftformatierungen. Z.B. `Font.PLAIN` (normale Schrift), `Font.BOLD` (fette Schrift) und `Font.ITALIC` (kursive Schrift). Diese Formatierungen können auch kombiniert werden, so dass Sie eine Schrift fett und kursiv machen können: `Font.BOLD + Font.ITALIC`.

Die folgende Deklaration erstellt eine Times Roman-Schrift, die fett und kursiv ist und eine Punktgrösse von 12 hat:

```
Font myFont =
new Font("TimesRoman", Font.BOLD + Font.ITALIC, 12);
```

1 **Starten Sie einen Editor.**

2 **Erstellen Sie die nachfolgende Datei.**

Erzeugung von Font-Objekten

```
import java.awt.*;
 public class Zeichne3 extends java.applet.Applet {
 public void paint(Graphics g) {
Font schrift1 = new Font("TimesRoman", Font.PLAIN, 18);
 Font schrift2 = new Font("TimesRoman", Font.BOLD, 14);
 Font schrift3 = new Font("TimesRoman", Font.ITALIC, 22);
 Font schrift4 = new Font("TimesRoman", Font.BOLD +
 Font.ITALIC, 24);
 Font schrift5 = new Font("Arial", Font.PLAIN, 10);
 Font schrift6 = new Font("Courier", Font.BOLD, 12);
 Font schrift7 = new Font("Arial", Font.ITALIC, 34);
 Color farbe1 = new Color(255, 100, 92);
 g.setFont(schrift1);
 g.drawString("Normaler (plain) Font - TimesRoman", 10, 25);
 g.setFont(schrift2);
 g.drawString("Fetter (bold) Font - TimesRoman", 10, 50);
 g.setFont(schrift3);
 g.drawString("Kursiver (italic) Font - TimesRoman", 10, 75);
 g.setFont(schrift4);
 g.drawString("Fetter und kursiver (bold italic) Font -
 TimesRoman", 10, 85);
 g.setFont(schrift5);
 g.drawString("Normaler (plain) Font - Arial", 10, 125);
 g.setFont(schrift6);
 g.drawString("Fetter Font - Courier", 10, 150);
 g.setColor(farbe1);
 g.setFont(schrift7);
 g.drawString("Farbig, gross und kursiv - Arial", 10, 200);
 }
}
```

TIPP

Vor der letzten Ausgabe wird dem Grafik-Objekt zusätzlich eine Farbe zugewiesen.

3 **Speichern Sie die Datei unter** `Zeichne3.java`.

4 **Kompilieren Sie die Datei.**

5 **Erstellen Sie die notwendige HTML-Datei.**

```
<HTML>
<BODY>
<APPLET CODE = "Zeichne3.class" WIDTH = 550 HEIGHT = 250>
</APPLET>
</BODY>
</HTML>
```

6 **Speichern Sie die Datei unter** `Zeichne3.HTML`.

7 **Starten Sie die Datei im Browser oder Appletviewer.**

Text in verschiedenen Ausprägungen und teilweise sogar überlagert

Farbiger Text

Fragen

Testen Sie Ihr Wissen!

- In welcher Klasse befinden sich die grundlegenden Grafikfähigkeiten von Java?
- Welche Methode ist eine besonders sinnvolle Möglichkeit für die Grafikausgabe unter Java?
- Wo liegt der Nullpunkt des Koordinatensystems bei grafischen Ausgaben?
- Wozu dient die Anweisung `import`?
- Welche Klasse wird verwendet, um ein Objekt in einer bestimmten Farbe darstellen zu können?
- Was ist das RGB-Modell?
- Mit welcher Methode kann man auf einer grafischen Oberfläche Text ausgeben?
- Welche Klassen stellen erweiterte Möglichkeiten zur Formatierung der Textausgabe bereit?