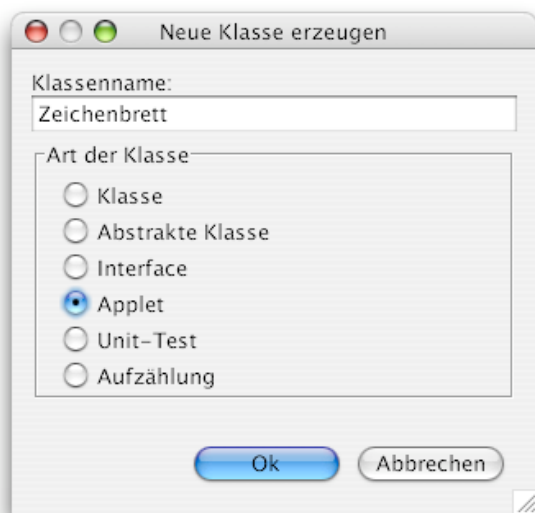


## Folge 5: Java-Applets

Einer der Gründe, warum so viele Leute mit Java arbeiten, ist sicherlich die Tatsache, dass man Java-Anwendungen in einem Browser laufen lassen kann, z.B. Safari oder Firefox. Allerdings kann man nicht jede Java-Anwendung in einem Browser anzeigen, sondern muss dazu ein so genanntes Java-Applet erstellen. Am besten lernen Sie dieses Vorgehen in einem kleinen Workshop.

### Schritt 1 - Ein neues Applet erstellen

Starten Sie BlueJ und legen Sie ein neues Projekt an. Speichern Sie dieses neue Projekt in einem neuen Ordner namens "Folge05" als "Version01". Klicken Sie dann auf "Neue Klasse" und achten Sie darauf, dass Sie in dem Dialogfenster "Neue Klasse erzeugen" den Klassentyp "Applet" ausgewählt haben:



5.1 Ein neues Applet erzeugen.

In unserem Beispiel hat das Applet den Namen "Zeichenbrett". Das hört sich ja schon einmal sehr verheißungsvoll an.

### Schritt 2 - Quelltext aufräumen

Wenn man den orangefarbenen Kasten anklickt, den BlueJ erzeugt hat, bekommt man zuerst einen kleinen Schock - so viel Quelltext. Dass das meiste davon völlig unnötig ist, zeigt folgender Minimal-Quelltext, den man erhält, wenn man den von BlueJ erzeugten Quelltext gehörig aufräumt. Vielleicht ist es aber auch besser, wenn Sie den Quelltext ganz löschen und den folgenden **Minimal-Quelltext** selbst eintippen.

```
import java.awt.*;
import javax.swing.*;

public class Zeichenbrett extends JApplet
{
    public void init()
    {
    }

    public void paint(Graphics g)
    {
        g.drawString("Beispiel-JApplet", 20, 20);
    }
}
```

## Schritt 3 - Analyse des Minimal-Quelltextes

Wenn Sie den Minimal-Quelltext auf den ersten Blick verstanden haben, können Sie diesen Schritt überspringen. Ich werde Ihnen hier den Quelltext Zeile für Zeile erläutern.

```
import java.awt.*;
```

Diese Zeile importiert zusätzliche Befehle in Ihre Java-Anwendung, die üblicherweise nicht zur Verfügung stehen. Zum Beispiel den Befehl `drawString()`, mit dem Sie einen Text in dem Applet ausgeben können.

```
import javax.swing.*;
```

Auch diese Zeile importierte weitere Befehle, die Ihr Applet benötigt.

```
public class Zeichenbrett extends JApplet
```

Dies ist der Kopf einer Klasse, es wird also eine neue Klasse deklariert. Allerdings keine "normale" Klasse, sondern eine Applet-Klasse. Was ist denn das schon wieder, eine Applet-Klasse. Ich fürchte, wir müssen jetzt ganz kurz über Vererbung sprechen. Unter Vererbung versteht man in der objektorientierten Programmierung eine bestimmte Beziehung zwischen zwei Klassen. Sie kennen ja bereits die HAT-Beziehung, wenn also eine Klasse Objekte einer anderen Klasse als Attribute hat. Die Vererbung ist dagegen eine IST-Beziehung. Die Klasse **Zeichenbrett** ist eine Tochterklasse der Klasse **JApplet**. Das heißt, alle Attribute und Methoden von **JApplet** sind auch in **Zeichenbrett** vorhanden. Somit IST ein Zeichenbrett-Objekt gleichzeitig auch ein JApplet-Objekt. **Zeichenbrett** ist aber noch mehr als nur ein JApplet. Der Benutzer - also Sie - kann weitere Methoden und Attribute festlegen, die nur für die Klasse **Zeichenbrett** gelten. Über Vererbung werden wir in einem späteren Kapitel noch mehr sprechen, für das Erste soll es reichen. Programmiertechnisch erreicht man eine solche Vererbung über das Schlüsselwort `extends`. Der Satz "`Zeichenbrett extends JApplet`" ist also so zu verstehen: Die Klasse **Zeichenbrett** ist eine Erweiterung der bereits vorhandenen Klasse **JApplet**.

```
{  
    public void init()  
    {  
    }  
}
```

Diese Zeilen enthalten den Quelltext der Methode `init()`. Zur Zeit enthält die Methode noch keinen Quelltext, später aber wird sich dies ändern. Auf jeden Fall sollte man die Methode `init()` bei einem Applet nicht löschen.

```
    public void paint(Graphics g)  
    {  
        g.drawString("Beispiel-JApplet", 20, 20);  
    }  
}
```

Damit wären wir auch schon bei der wichtigsten Methode eines Applets. Die `paint()`-Methode ist dafür verantwortlich, das der Benutzer überhaupt etwas sehen kann in dem Java-Applet. Hier können Sie Texte ausgeben, Kreise, Rechtecke und Linien malen und so weiter. In unserem Beispiel-Programm schreibt die Methode `paint()` lediglich den Text "Beispiel-JApplet" in das Applet, und zwar an der Position  $x = 20$ ,  $y = 20$ . Diese Zahlen sind Pixel-Koordinaten. Der String "Beispiel-JApplet" erscheint daher an der Position (20,20) in dem Applet.

## Schritt 4 - Erstes Kompilieren und Testen

Klicken Sie nun auf den "Übersetzen"-Button von BlueJ. Das Applet müsste fehlerfrei übersetzt werden. Klicken Sie jetzt bitte mit der rechten Maustaste auf die Klasse **Zeichenbrett** und wählen Sie den Befehl "Applet ausführen" (Abbildung 5.2).

Es erscheint nun ein Dialogfenster. Übernehmen Sie einfach die Standard-Einstellungen. Danach wird das Applet nicht im Web-Browser angezeigt, sondern in dem Applet-Viewer. Das ist ein kleines Programm, das den Browser ersetzen soll. Es dauert nämlich immer recht lange, bis der Browser gestartet ist und dann das Applet anzeigt, der Applet-Viewer ist wesentlich schneller. Außerdem können Sie hier die Größe des Applets frei wählen. Standardmäßig ist eine Größe von 500 mal 500 Pixeln eingestellt. Klicken Sie also einfach auf O.K., um die Standardeinstellungen zu übernehmen.

Es erscheint dann ein Fenster auf dem Bildschirm, das tatsächlich ungefähr 500 mal 500 Pixel groß ist. Links oben in dem Applet kann man dann den Schriftzug "Beispiel-JApplet" lesen.

Damit wäre unser erster Test gelungen.

## Schritt 5 - Wir malen einen Kreis

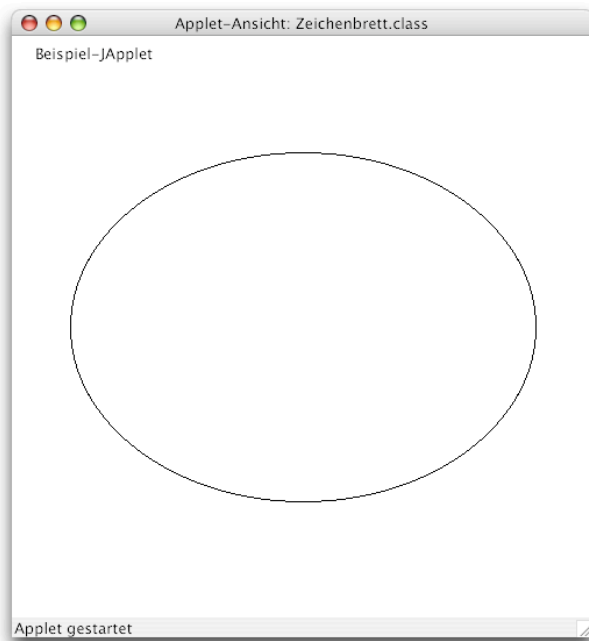
Zum Zeichnen von Kreisen gibt es einen einprägsamen Befehl: **drawOval**. Der folgende Quelltext zeigt Ihnen, wie dieser Befehl benutzt wird.

```
public void paint(Graphics g)
{
    g.drawString("Beispiel-JApplet", 20, 20);
    g.drawOval(50,100,400,300);
}
```

Sie sehen hier den Quelltext der **paintO**-Methode des Applets, die anderen Zeilen des Applets wurden nicht verändert.

Zunächst einmal muss man sagen, dass das Applet selbst keinen **drawStringO**- oder **drawOvalO**-Befehl kennt. Es gibt innerhalb der Klasse **JApplet** bzw. der davon abstammenden Klasse **Zeichnung** keine Methode, die **drawStringO** oder **drawOvalO** heißt. Wenn man genauer hinschaut, sieht man, dass der Befehl zum Zeichnen eines Ovals auch nicht "drawOvalO" heißt, sondern "g.drawOvalO". Was ist dieses "g" nun schon wieder? Bei g handelt es sich um ein Objekt der Klasse **Graphics**. Diese Klasse wurde durch die Import-Zeilen am Anfang des Applets in das Applet eingebunden. Die Klasse **Graphics** stellt viele Befehle zum Zeichnen zur Verfügung, z.B. den Befehl zum Zeichnen von Ovalen.

Und so sollte das Applet aussehen, wenn es kompiliert und ausgeführt wurde:

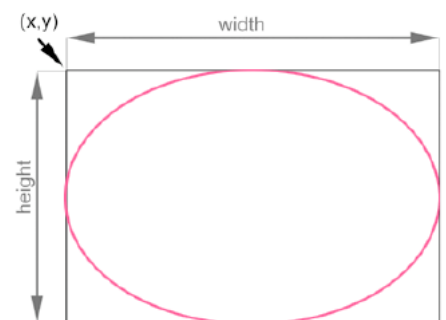


5.2 Das Applet mit dem Oval

### Der drawOval-Befehl

Schauen wir uns den **drawOvalO**-Befehl näher an. Es werden vier Parameter erwartet. Um die Bedeutung dieser Parameter zu verstehen, betrachten Sie die Abbildung 5.3. Stellen Sie sich das Oval in ein Rechteck eingebettet vor. Die beiden ersten Parameter legen dann die linke obere Ecke dieses Rechtecks fest. Es handelt sich also um x-y-Parameter.

Die beiden anderen Parameter bestimmen die Breite und die Höhe (width, height) des Rechtecks. Einen Kreis erhalten Sie, wenn für Breite und Höhe gleiche Werte gewählt werden.



5.3 Der Befehl drawOvalO

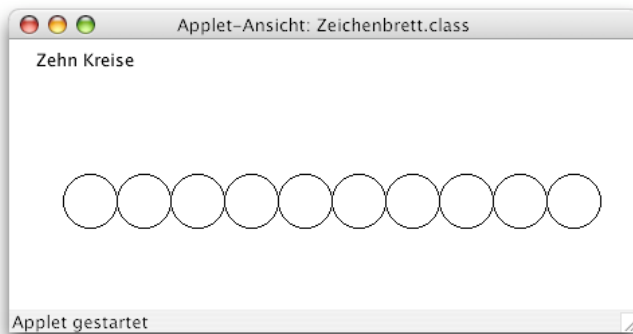
## Schritt 6 - Wir malen zehn Kreise

Wir wollen jetzt nebeneinander zehn kleine Kreise in das Applet malen. Natürlich könnte man dazu zehnmal den Befehl **drawOval()** hinschreiben und jedes mal die Parameter neu bestimmen. Aber wozu haben wir eigentlich in der Folge 4 über for-Schleifen gesprochen? Wir wollen jetzt diese for-Schleifen mal für etwas Sinnvolles einsetzen. Hier die entsprechende **paint()**-Methode:

```
public void paint(Graphics g)
{
    g.drawString("Zehn Kreise", 20, 20);

    for (int i = 1; i <= 10; i++)
        g.drawOval(40*i, 100, 40, 40);
}
```

So schwer war das doch gar nicht. Wir lassen die for-Schleife zehnmal durchlaufen. In jedem Schleifendurchgang wird ein Kreis gezeichnet. Die zehn Kreise unterscheiden sich nur in ihren x-Koordinaten. Die y-Koordinate hat immer den Wert 100, und Breite und Höhe des umgebenden Rechtecks bleiben auch gleich. Also müssen wir nur die x-Koordinate in jedem Schleifendurchgang neu berechnen. Dazu werten wir einfach die Laufvariable *i* der for-Schleife aus. Die x-Koordinate soll einfach das 40fache der Laufvariable sein. Betrachten wir nun das Ergebnis:



5.4 Zehn Kreise nebeneinander

## Schritt 7 - Sie malen viele Kreise

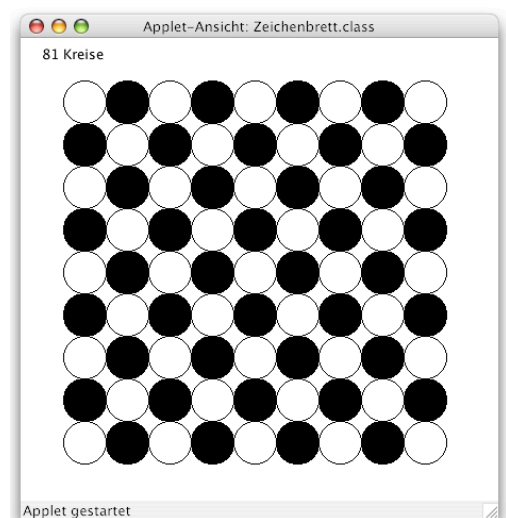
Richtig gelesen, jetzt sind Sie dran. Kommen wir also zu den ersten Übungen der Folge 5

### Übung 5.1 (2 Punkte)

Lesen Sie im Theorieteil der Folge 4 (for-Schleifen) nach, was man unter einer geschachtelten for-Schleife versteht und ergänzen Sie dann das Java-Applet so, dass zehn Reihen mit je zehn Kreisen gezeichnet werden, also insgesamt 100 Kreise.

### Übung 5.2 (4 Punkte)

Betrachten Sie nun die Abbildung 5.5. Hier sind 81 Kreise dargestellt (9 x 9), die abwechselnd weiß und schwarz gezeichnet sind. Zum Zeichnen eines schwarz gefüllten Kreises verwendet man nicht den Befehl **g.drawOval()**, sondern den Befehl **g.fillOval()**. Die Parameter sind die gleichen wie bei **g.drawOval()**. Das Problem bei dieser Übung ist es, die beiden for-Schleifen so zu verändern, dass immer abwechselnd ein weißer Kreis und ein schwarzer Kreis gezeichnet werden. Ich habe bei mir das Problem so gelöst, dass ich in der Methode **paint()**



5.5 81 Kreise