Grafik in Java mit SWING

Ac 03-2018 bis 04-2021

Im folgenden Beitrag wird die Grafikerstellung mit **Swing** betrachtet, da Swing im Gegensatz zu dem neueren JavaFX auch in aktuellen Java-Versionen (***Stand 04-2021: Version 16***) enthalten ist . Programmierkenntnisse der Sprache JAVA werden vorausgesetzt !

Als Grundlage aller grafischen Darstellungen mit Swing verwenden wir den sogenannten **JFrame** (engl. frame = Rahmen). In diesem Rahmen bzw. Fenster finden alle Aktionen statt.

Ein JFrame lässt sich mit folgendem Mini-Programm erzeugen:

```
import javax.swing.JFrame;

public class JFrameInMain {
   public static void main(String[] args) {
      JFrame frame = new JFrame();
      frame.setVisible(true);
   }
}
```

Da wir mit einer bloßen Titelleiste nicht viel anfangen können, müssen wir noch einiges ergänzen.

```
import javax.swing.JFrame;

public class JFrameInMain {
   public static void main(String[] args) {
      JFrame frame = new JFrame();
      frame.setSize(200, 100);
      frame.setLocationRelativeTo(null);
      frame.setVisible(true);
   }
}
```

Der JFrame hat nun eine definierte Breite = 200 Pixels und Höhe = 100 Pixels, so dass unterhalb der Titelleiste Platz für Komponenten oder zum Zeichnen ist.

In der Praxis geht man jedoch etwas anders vor. Man erzeugt den JFrame durch "**Vererbung**" (die zu erstellende Klasse "erbt" von der Klasse JFrame mittels des Schlüsselworts `extends`) .
Die folgende Klasse JFrame0 liefert das selbe Ergebnis wie die Klasse JFrameInMain .

```
import javax.swing.JFrame;

public class JFrame0 extends JFrame {

   public static void main(String[] args) {
      new JFrame0();
   }

   // KONSTRUKTOR
   public JFrame0() {
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(200, 100);
      setLocationRelativeTo(null);
      setVisible(true);
   }
}
```

Das Fenster befindet sich in der Mitte des Bildschirms, was durch `setLocationRelativeTo(null);` bewirkt wurde. Durch Anklicken des Kreuzes in der rechten oberen Ecke kann es geschlossen werden.

Anmerkung:

Die Java-Konventionen empfehlen, dass JFrame-Aktionen in einem eigenen "Thread" ablaufen sollen. Die main-Methode sieht dann so aus:

```
   public static void main(String[] args) {
      EventQueue.invokeLater(new Runnable() {
         public void run() {
            try {
               Frame0 frame = new Frame0();
               frame.setVisible(true);
            } catch (Exception e) {
               e.printStackTrace();
            }
         }
      });
   }
```

Mit der sog. "Lambda-Schreibweise" geht es deutlich kürzer:

```
  public static void main(String[] args) {
      EventQueue.invokeLater(() -> {
         var frame = new Frame0();
         frame.setVisible(true);
      });
   }
```

Allerdings werden wir in den nächsten Beispielen auf diese "Thread-Methode" noch **verzichten** !

Man kann jetzt sog. Komponenten (Labels, Buttons, Panels, ...) auf den JFrame setzen.
Als Beispiel verwenden wir einen JButton (= Aktionsknopf) :

```
import javax.swing.JButton;
import javax.swing.JFrame;

public class JFrameMitButton extends JFrame {

   public static void main(String[] args) {
      new JFrameMitButton();
   }

   // KONSTRUKTOR
   public JFrameMitButton() {
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(200, 80);
      setLocationRelativeTo(null);
      JButton btn = new JButton("Knopf");
      add(btn);
      setVisible(true);
   }
}
```

Das Hauptthema in diesem Beitrag soll aber nicht Komponentenerstellung sein, sondern Zeichnen.

Es gibt verschiedene Methoden, in Java mittels Swing zu zeichnen:

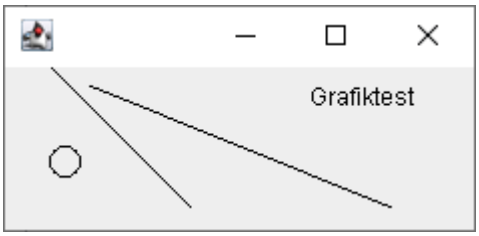
1) Zeichnen direkt auf einem JFrame

Ein JFrame (Fenster bzw. Rahmen) wird erzeugt; anschließend wird mit der **paint-Methode** , die einen Grafikkontext (d.h. eine Zeichenumgebung) zum JFrame liefert, darauf gezeichnet. Dazu muss diese paint-Methode überschrieben werden.

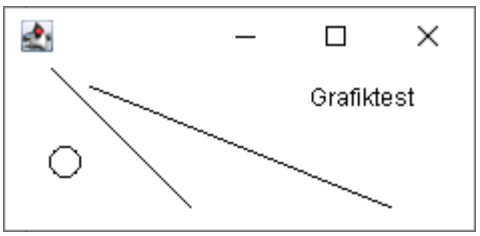
Ein sehr einfaches Swing-Programm ist das folgende:

```
import java.awt.Graphics;
import javax.swing.JFrame;

// Ac 24.03.2018;  verbessert:  16.04.2021

public class GrafikJFrame0 extends JFrame {
   private static final int FRAME_WIDTH = 250;
   private static final int FRAME_HEIGHT = 120;

   public static void main(String[] args) {
      new GrafikJFrame0();
   }

   // KONSTRUKTOR
   public GrafikJFrame0() {
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      setSize(FRAME_WIDTH, FRAME_HEIGHT);
      setLocationRelativeTo(null);
      setVisible(true);
   }

   public void paint(Graphics g) { // g ist der "Grafikkontext" (die Zeichenumgebung des Frames)
      g.drawLine(0, 0, 100, FRAME_HEIGHT - 5); // Linie zeichnen
      g.drawLine(50, 40, 200, 100);
      g.drawOval(30, 70, 15, 15); // Kreis zeichnen
      g.drawString("Grafiktest", 160, 50); // Text ausgeben
   }
}
```

Ergebnis:



Der Hintergrund ist grau / beige, was für das Setzen von Komponenten auf den Frame in Ordnung ist. Für das Zeichnen wählen wir jedoch einen weißen Hintergrund. Ihn erhält man durch Einfügen von `import java.awt.Color;` zu Beginn sowie von `setBackground(Color.WHITE);` im "Konstruktor" :

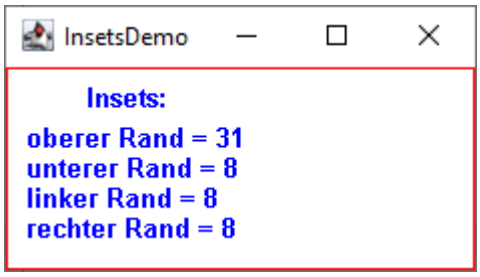


Wie man sieht, wird die erste Linie von der oberen Titelleiste teilweise verdeckt, denn (0,0) befindet sich in der oberen linken Ecke des Frames. Außerdem erreicht diese Linie auch nicht die y-Koordinate 115 am unteren Rand. Die Titelleiste nimmt eine Höhe von 31 Pixels ein (Windows 10), wie wir weiter unten noch sehen werden, außerdem gibt es noch links, rechts und unten (nicht sichtbare) Ränder.
Die Maße für diese Ränder lassen sich per Programm ermitteln. Dazu berechnet man die sog. "Insets" des JFrames, das sind die Breiten der Ränder oben, unten, links und rechts. Das, was die Ränder vom JFrame übriglassen, ist ein Rechteck, welches oben einen weißen Hintergrund zugewiesen bekam.

Das folgende Programm ermittelt die Ränder (Insets) und gibt die Maße der Insets auf dem JFrame aus. Zur besseren Übersicht wird noch das sichtbare Fenster als Rechteck gezeichnet:

```java
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Insets;
import javax.swing.JFrame;

// Ac 24.03.2018;  verbessert:  16.04.2021

public class GrafikJFrame1 extends JFrame {
   private static final int FRAME_WIDTH = 250;
   private static final int FRAME_HEIGHT = 140;

   public static void main(String[] args) {
      new GrafikJFrame1();
   }

   // KONSTRUKTOR
   public GrafikJFrame1() {
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(FRAME_WIDTH, FRAME_HEIGHT);
      setLocationRelativeTo(null);
      setBackground(Color.WHITE);
      setTitle("InsetsDemo");
      setVisible(true);
   }

   @Override
   public void paint(Graphics g) {
      Insets ins = getInsets();
      int obRand = ins.top, untRand = ins.bottom, liRand = ins.left, reRand = ins.right;
      // Text (in blauer Farbe) im JFrame ausgeben
      g.setColor(Color.BLUE);
      g.setFont(new Font("Arial", Font.BOLD, 13));
      g.drawString("Insets:", liRand + 40, obRand + 20);
      g.drawString("oberer Rand = " + obRand, liRand + 10, obRand + 40);
      g.drawString("unterer Rand = " + untRand, liRand + 10, obRand + 55);
      g.drawString("linker Rand = " + liRand, liRand + 10, obRand + 70);
      g.drawString("rechter Rand = " + reRand, liRand + 10, obRand + 85);
      // Rechteck einzeichnen
      g.setColor(Color.RED);
      g.drawRect(liRand, obRand, FRAME_WIDTH - reRand - liRand - 1, FRAME_HEIGHT - untRand - obRand -
                1);
   }
}
```

Ergebnis:

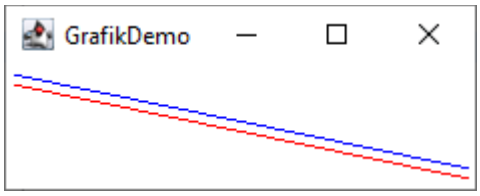


Wollen wir jetzt innerhalb des sichtbaren Bereichs Linien einzeichnen, so sind nur kleine Änderungen in der paint-Methode notwendig.

Hier der gesamte Quellcode für das Zeichnen von Linien:

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Insets;
import javax.swing.JFrame;

// Ac 24.03.2018;  verbessert:  16.04.2021

public class GrafikJFrameLinien extends JFrame {
   private static final int FRAME_WIDTH = 250;
   private static final int FRAME_HEIGHT = 100;

   public static void main(String[] args) {
      new GrafikJFrameLinien();
   }

   // KONSTRUKTOR
   public GrafikJFrameLinien() {
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(FRAME_WIDTH, FRAME_HEIGHT);
      setLocationRelativeTo(null);
      setBackground(Color.WHITE);
      setTitle("GrafikDemo");
      setVisible(true);
   }

   @Override
   public void paint(Graphics g) {
      // Insets für das Grafikfenster ermitteln
      Insets ins = getInsets();
      int obRand = ins.top, liRand = ins.left;

      // Linien im JFrame ausgeben
      g.setColor(Color.BLUE);
      g.drawLine(liRand + 4, obRand + 4, liRand + 230, obRand + 50);
      g.setColor(Color.RED);
      g.drawLine(liRand + 4, obRand + 9, liRand + 230, obRand + 55);
   }
}
```

Ergebnis:

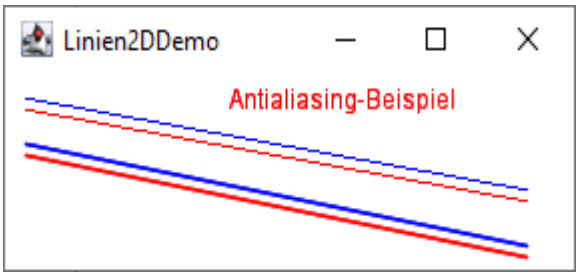


Sollen die Linien dicker werden, dann geht das nur unter Verwendung von **Java2D** bzw. **Graphics2D**. Dazu muss das Graphics-Objekt der paint-Methode in ein Graphics2D-Objekt "gecastet" (umgewandelt) werden. Fortan können alle Vorteile von Java2D genutzt werden, u.a. auch die Verwendung von gestrichelten Linien oder das sog. "Antialiasing" (= Kantenglättung).

Im folgenden wird der entsprechende Quellcode angegeben:

```java
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Insets;
import java.awt.RenderingHints;
import javax.swing.JFrame;

// Ac 24.03.2018; verbessert: 16.04.2021

public class Grafik2DJFrame extends JFrame {
   private static final int FRAME_WIDTH = 300;
   private static final int FRAME_HEIGHT = 140;

   public static void main(String[] args) {
      new Grafik2DJFrame();
   }

   public Grafik2DJFrame() { // KONSTRUKTOR
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(FRAME_WIDTH, FRAME_HEIGHT);
      setLocationRelativeTo(null);
      setBackground(Color.WHITE);
      setTitle("Linien2DDemo");
      setVisible(true);
   }

   public void paint(Graphics g) {
      Graphics2D g2d = (Graphics2D) g;
      Insets ins = getInsets();
      int obRand = ins.top, liRand = ins.left;

      // 2 Linien normal zeichnen
      g2d.setColor(Color.BLUE);
      g2d.drawLine(liRand + 10, obRand + 10, liRand + 260, obRand + 50);
      g2d.setColor(Color.RED);
      g2d.drawLine(liRand + 10, obRand + 15, liRand + 260, obRand + 55);

      // Anti-Aliasing (Kantenglättung) einschalten; vermeidet Treppenstufen
      g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
      g2d.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
                                          RenderingHints.VALUE_TEXT_ANTIALIAS_ON);

      // Dicke der Linien festlegen
      float dicke = 1.5f;
      g2d.setStroke(new BasicStroke(dicke, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND));

      // 2 Linien dick und mit Antialiasing zeichnen
      g2d.setColor(Color.BLUE);
      g2d.drawLine(liRand + 10, obRand + 30, liRand + 260, obRand + 75);
      g2d.setColor(Color.RED);
      g2d.drawLine(liRand + 10, obRand + 35, liRand + 260, obRand + 80);
      // Text
      g2d.drawString("Antialiasing-Beispiel", 120, obRand + 15);
   }
}
```

Ergebnis:



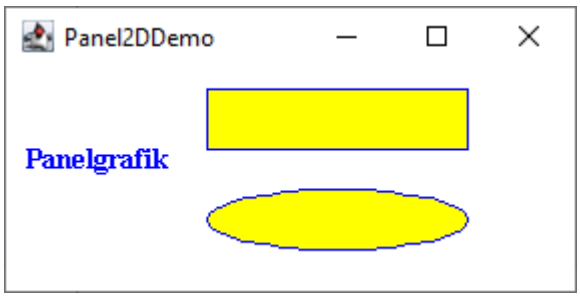
2) Zeichnen auf einem JPanel

Eine bessere Alternative zum Zeichnen in Java ist ein JPanel (Platte, Tafel), das auf die sogenannte "**contentPane**" (InhaltsScheibe) des JFrames gesetzt wird. Damit umgeht man die Probleme mit den Insets, denn die Maße des JPanels sind jetzt entscheidend ! Das JPanel fungiert dann als "Leinwand", auf die mittels seiner **paintComponent-Methode** gezeichnet wird.
Das JPanel wird als eigene Klasse deklariert.

Ein mögliches Java-Programm ist das folgende:

```
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class Grafik2DJPanel extends JFrame {
   private static final int FRAME_WIDTH = 300;
   private static final int FRAME_HEIGHT = 150;
   Leinwand malPanel = new Leinwand();

   public static void main(String[] args) {
      new Grafik2DJPanel();
   }

   public Grafik2DJPanel() { // KONSTRUKTOR
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(FRAME_WIDTH, FRAME_HEIGHT);
      setLocationRelativeTo(null);
      setTitle("Panel2DDemo");

      malPanel.setPreferredSize(new Dimension(FRAME_WIDTH - 20, FRAME_HEIGHT - 45));
      malPanel.setBackground(Color.WHITE);
      getContentPane().add(malPanel);
      setVisible(true);
   }
}

class Leinwand extends JPanel {

   public void paintComponent(Graphics g) {
      Graphics2D g2d = (Graphics2D) g;
      super.paintComponent(g2d); // "säubert" das Panel

      g2d.setColor(Color.YELLOW);
      g2d.fillRect(100, 10, 130, 30);
      g2d.fillOval(100, 60, 130, 30);
      g2d.setColor(Color.BLUE);
      g2d.drawRect(100, 10, 130, 30);
      g2d.drawOval(100, 60, 130, 30);

      g2d.setFont(new Font("SERIF", Font.BOLD, 14));
      g2d.drawString("Panelgrafik", 10, 50);
   }
}
```

Ergebnis:



Anmerkung:
Als Komponenten, auf die gezeichnet werden kann, eignen sich z.B. auch **JLabel** oder **JButton**.

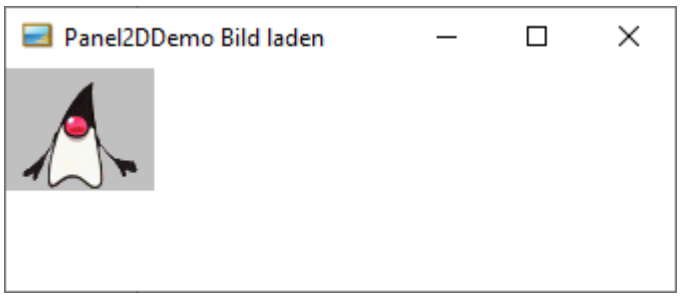
Bilder in ein JPanel laden

Das JPanel ist nicht nur dazu geeignet, als Zeichenbrett zu dienen, sondern es kann selbstverständlich auch fertige Bilder darstellen, die z.B. von der Festplatte geladen werden.
Java stellt zum Laden von Bildern die Klasse **ImageIO** zur Verfügung.
Hat man nun ein Bild (etwa eine png-Datei namens "duke.png") in einem Verzeichnis gespeichert, so kann man es mittels **bild = ImageIO.read(new File("duke.png");** in einen Bereich im Speicher laden, der vorher als **Image bild;** deklariert wurde.
Voraussetzung ist, dass die Datei "duke.png" sich im aktuellen Verzeichnis befindet.

Zusätzlich eingebautes "Schmankerl": Wir ersetzen das Java-Logo "Kaffetasse" durch ein neues Icon !

```
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

public class Grafik2DJPanelBild extends JFrame {
   private static final int FRAME_WIDTH = 350;
   private static final int FRAME_HEIGHT = 150;
   Leinwand0 malPanel = new Leinwand0();

   public static void main(String[] args) {
      new Grafik2DJPanelBild();
   }

   public Grafik2DJPanelBild() { // KONSTRUKTOR
      ImageIcon frameIcon = new ImageIcon("frameIcon.png"); // ein Bildsymbol
      setIconImage(frameIcon.getImage());

      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(FRAME_WIDTH, FRAME_HEIGHT);
      setLocationRelativeTo(null);
      setTitle("Panel2DDemo Bild laden");
      setVisible(true);

      malPanel.setPreferredSize(new Dimension(FRAME_WIDTH - 20, FRAME_HEIGHT - 45));
      malPanel.setBackground(Color.WHITE);
      getContentPane().add(malPanel);
   }

}

class Leinwand0 extends JPanel {
   private Image bild = null;

   public Leinwand0() {
      ladeBild();
   }

   public void paintComponent(Graphics g) {
      Graphics2D g2d = (Graphics2D) g;
      super.paintComponent(g2d);
      if (bild != null)
         g2d.drawImage(bild, 0, 0, this);
   }

   public void ladeBild() {
      try {
         bild = ImageIO.read(new File("duke.png"));
      } catch (IOException e) {
         JOptionPane.showMessageDialog(null, "Fehler beim Öffnen der Datei!");
      }
   }
```

Ergebnis:



Soll das Bild vorher auf der Festplatte gesucht werden, so ist ein deutlich höherer Aufwand erforderlich.

```
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FileDialog;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

public class Grafik2DJPanelBild extends JFrame {
   int FRAME_WIDTH = 350, FRAME_HEIGHT = 150;
   Leinwand0 malPanel = new Leinwand0();

   public static void main(String[] args) {
      new Grafik2DJPanelBild();
   }

   public Grafik2DJPanelBild() { // KONSTRUKTOR
      ImageIcon frameIcon = new ImageIcon("frameIcon.png"); // ein Bildsymbol
      setIconImage(frameIcon.getImage());

      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(FRAME_WIDTH, FRAME_HEIGHT);
      setLocationRelativeTo(null);
      setTitle("Panel2DDemo Bild laden");

      malPanel.setPreferredSize(new Dimension(FRAME_WIDTH - 20, FRAME_HEIGHT - 45));
      malPanel.setBackground(Color.WHITE);
      getContentPane().add(malPanel);
      setVisible(true);
   }
}


class Leinwand0 extends JPanel {
   private Image bild = null;

   public Leinwand0() {
      ladeBild();
   }

   public void paintComponent(Graphics g) {
      Graphics2D g2d = (Graphics2D) g;
      super.paintComponent(g2d);
      if (bild != null)
         g2d.drawImage(bild, 0, 0, this);
   }

   JFrame dateiDialogFrame = new JFrame();
   String bildDateiListe = "*.jpg;*.bmp;*.gif;*.png";

   public void ladeBild() {
       try {
```

```
            FileDialog d = new FileDialog(dateiDialogFrame, "Bild laden", FileDialog.LOAD);
            d.setFile(bildDateiListe);
            d.setVisible(true);
            if (d.getFile() == null)
               return;
            String pfadName = d.getDirectory() + d.getFile();
            File bildDatei = new File(pfadName);
            bild = ImageIO.read(bildDatei);
         } catch (IOException e) {
            JOptionPane.showMessageDialog(null, "Fehler beim Öffnen der Datei!");
         }
      }
}
```

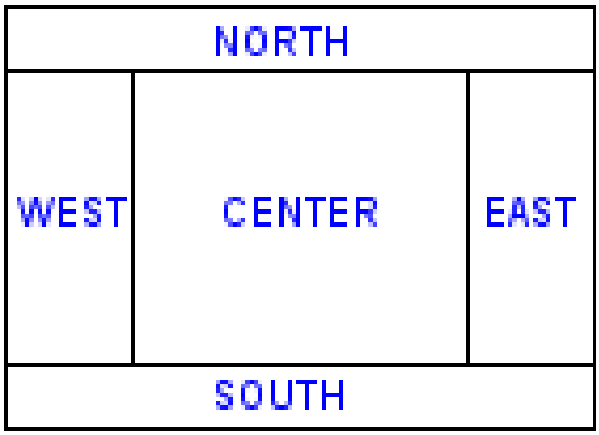
Ergebnis (Fenster angepasst !) :

Ereignisgesteuert zeichnen

In den meisten Anwendungen reicht eine bloße Grafikausgabe nicht aus, sondern man will Aktionen per Knopfdruck auslösen. Z.B. soll per Knopfdruck (Button) eine Grafik gezeichnet, bei Bedarf aber auch wieder per Knopfdruck gelöscht werden.

Da jetzt weitere Komponenten Verwendung finden (JButton etc.), ist es ratsam, die grafische Oberfläche per "Layout-Manager" aufzubauen.

Ein oft verwendeter Layout-Manager ist das **Border-Layout** mit den 5 Bereichen NORTH, SOUTH, WEST, EAST, CENTER.



Dieser Manager passt nach Größenänderungen des Fensters die Inhalte automatisch an .

Wir legen fest, dass CENTER die Zeichenfläche aufnimmt und NORTH die JButtons zur Steuerung. Die restlichen 3 Bereiche bleiben ungenutzt.

Die beiden JButtons erhalten je einen "ActionListener" (Lauscher), so dass die Zeichenaktion (Grafik zeichnen, Grafik löschen) per Boolean-Variable gesteuert werden kann. Damit der Boolean-Wert an die paintComponent-Methode des Zeichenpanels problemlos weitergeleitet werden kann, wird das Zeichenpanel namens "Leinwand2" als "innere Klasse" (Klasse innerhalb der Hauptklasse) deklariert.

Nach jeder Zustandsänderung (= Knopfdruckänderung) muss die Zeichnung per **repaint()** - Anweisung neu ausgeführt werden.

Beim Löschen wird nichts gezeichnet, sondern nur das Zeichenpanel per **super**-Anweisung "gesäubert" .

Zusätzlich wird das Hauptprogramm (main) gemäß Java-Konvention in einem **eigenen Thread** (EventQueue) abgearbeitet.

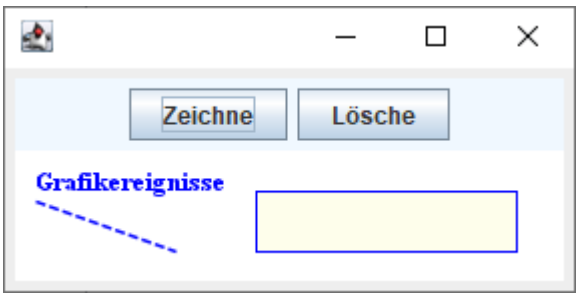
Das Java-Programm:

```
import java.awt.BasicStroke;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.geom.Line2D;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
```

```
public class Grafik2DEreignisgesteuert extends JFrame {
   private static final int FRAME_WIDTH = 300;
   private static final int FRAME_HEIGHT = 150;

   public static void main(String[] args) {
      EventQueue.invokeLater(new Runnable() {
         public void run() {
            try {
               Grafik2DEreignisgesteuert frame = new Grafik2DEreignisgesteuert();
               frame.setLocationRelativeTo(null);
               frame.setVisible(true);
            } catch (Exception e) {
               e.printStackTrace();
            }
         }
      });
   }

   private JPanel contentPane;
   boolean zeichnen = true;

   public Grafik2DEreignisgesteuert() {
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(FRAME_WIDTH, FRAME_HEIGHT);
      contentPane = new JPanel();
      contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
      contentPane.setLayout(new BorderLayout(0, 0));
      setContentPane(contentPane);

      Leinwand2 malPanel = new Leinwand2();
      malPanel.setPreferredSize(new Dimension(getWidth() - 20, getHeight() - 45));
      contentPane.add(malPanel, BorderLayout.CENTER);

      JPanel pnlSteuerung = new JPanel();
      pnlSteuerung.setBackground(new Color(240, 248, 255));
      contentPane.add(pnlSteuerung, BorderLayout.NORTH);

      JButton btnZeichne = new JButton("Zeichne");
      btnZeichne.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent arg0) {
            zeichnen = true;
            malPanel.repaint();
         }
      });
      pnlSteuerung.add(btnZeichne);

      JButton btnClear = new JButton("Lösche");
      btnClear.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent arg0) {
            zeichnen = false;
            malPanel.repaint();
         }
      });
      pnlSteuerung.add(btnClear);
   }

   class Leinwand2 extends JPanel { // innere Klasse

      Leinwand2() { // Konstruktor
         this.setBackground(Color.WHITE);
      }

      public Color farbeMitAlpha(Color f, int alpha) {
         return new Color(f.getRed(), f.getGreen(), f.getBlue(), alpha);
      }

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

         if (zeichnen) {
            g2d.setColor(farbeMitAlpha(Color.YELLOW, 20));
            g2d.fillRect(120, 20, 130, 30);
            g2d.setColor(Color.BLUE);
```

```
            g2d.drawRect(120, 20, 130, 30);

            float[] dash = { 5, 2 }; // gestrichelt ***** ***** ******
            Line2D.Double linie2d = new Line2D.Double(10, 25, 80, 50);
            g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                                 RenderingHints.VALUE_ANTIALIAS_ON);
            g2d.setStroke(new BasicStroke(1.5f, BasicStroke.CAP_BUTT,
                                 BasicStroke.JOIN_ROUND, 1, dash, 0));
            g2d.draw(linie2d);

            g2d.setFont(new Font("SERIF", Font.BOLD, 13));
            g2d.drawString("Grafikereignisse", 10, 20);

        }
    }
  }
}
```

Ergebnis:



Ereignisgesteuert zeichnen (2)

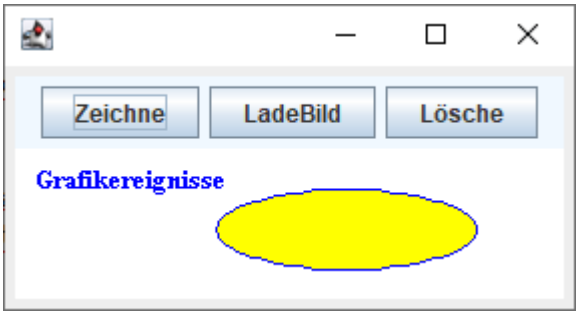
Das Programm wird nun um das Laden eines Bildes erweitert , was einen zusätzlichen Button erfordert. Die Boolean-Variable wird wegen der 3 zu unterscheidenden Möglichkeiten (zeichnen, laden, löschen) ersetzt durch eine Aufzählungsvariable "enum" .

Das Java-Programm:

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

public class Grafik2D3Ereignisse extends JFrame {
   private JPanel contentPane;

   public static void main(String[] args) {
      EventQueue.invokeLater(new Runnable() {
         public void run() {
            try {
               Grafik2D3Ereignisse frame = new Grafik2D3Ereignisse();
               frame.setLocationRelativeTo(null);
               frame.setVisible(true);
            } catch (Exception e) {
               e.printStackTrace();
            }
         }
      });
   }

   enum Aktion {
      ZEICHNE, LADE, LOESCHE
   };

   Aktion aktion = Aktion.ZEICHNE; // vordefiniert

   public Grafik2D3Ereignisse() {
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(300, 160);
      contentPane = new JPanel();
      contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
      contentPane.setLayout(new BorderLayout(0, 0));
      setContentPane(contentPane);

      Leinwand3 malPanel = new Leinwand3();
      malPanel.setPreferredSize(new Dimension(getWidth(), getHeight()));
      contentPane.add(malPanel, BorderLayout.CENTER);

      JPanel pnlSteuerung = new JPanel();
      pnlSteuerung.setBackground(new Color(240, 248, 255));
      contentPane.add(pnlSteuerung, BorderLayout.NORTH);

      JButton btnZeichne = new JButton("Zeichne");
      btnZeichne.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent arg0) {
            aktion = Aktion.ZEICHNE;
            malPanel.repaint();
         }
      });
```

```
        pnlSteuerung.add(btnZeichne);

        JButton btnLadebild = new JButton("LadeBild");
        btnLadebild.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent e) {
              aktion = Aktion.LADE;
              malPanel.repaint();
           }
        });
        pnlSteuerung.add(btnLadebild);

        JButton btnClear = new JButton("Lösche");
        btnClear.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent arg0) {
              aktion = Aktion.LOESCHE;
              malPanel.repaint();
           }
        });
        pnlSteuerung.add(btnClear);
     }

     class Leinwand3 extends JPanel { // innere Klasse

        Leinwand3() { // KONSTRUKTOR
           this.setBackground(Color.WHITE);
        }

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

           if (aktion == Aktion.ZEICHNE) {
              g2d.setColor(Color.YELLOW);
              g2d.fillOval(100, 20, 130, 40);
              g2d.setColor(Color.BLUE);
              g2d.drawOval(100, 20, 130, 40);
              g2d.setFont(new Font("SERIF", Font.BOLD, 13));
              g2d.drawString("Grafikereignisse", 10, 20);
           } else if (aktion == Aktion.LADE) {
              Image bild = null;
              try {
                 bild = ImageIO.read(new File("duke.png"));
              } catch (IOException e) {
              }
              if (bild != null)
                 g2d.drawImage(bild, 10, 10, this);
           } else
              g2d.clearRect(0, 0, getWidth(), getHeight());
        }
     }
}
```

Ergebnisse:



Der Nachteil ist, dass es mit dieser Konstruktion nicht möglich ist, nach dem Laden des Bildes noch zusätzlich eine Zeichnung auf das JPanel zu setzen, weil paintComponent immer erst die ganze Zeichnung löscht.

3) Zeichnen mittels "BufferedImage" in einem JPanel

Die flexibelste Möglichkeit zur Arbeit mit Grafik in Java besteht in der Verwendung eines Zwischenspeichers (Bildpuffers) im RAM, einem sog. "BufferedImage".

Z.B. legt die Anweisung

BufferedImage bufImg = new BufferedImage(200, 100, BufferedImage.*TYPE_INT_RGB*);

einen Bildpuffer mit 200 mal 100 Pixels im RAM des Computers an.

Damit man mit diesem Speicher arbeiten kann, muss man sich von ihm einen "Grafikkontext" , also eine Zeichenumgebung, besorgen (ähnlich wie "Graphics g" bei paintComponent bzw. bei paint).

Dies geschieht durch **Graphics2D grBufImg = bufImg.createGraphics();**

Man geht nun so vor, dass alle Grafikarbeiten zunächst in dem BufferedImage erledigt werden. Anschließend wird durch **g2d.drawImage(bufImg, 0, 0, this);** in der paintComponent() - Methode der Inhalt des Puffers auf das Grafikpanel "this" übertragen.

Alle Grafik-Routinen müssen in der Klasse Leinwand definiert werden. Dazu gehören u.a. das Löschen des gesamten BufferedImage, das Zeichnen von Linien und Flächen, das Ausgeben von Text, das Laden (und ggfs. auch Speichern) von Bildern, etc.

Außerdem muss eine Möglichkeit geschaffen werden, den BufferedImage-Bereich zu vergrößern (wenn das Panel größer wird, oder wenn ein großes Bild geladen wird), ohne dass die darin befindliche Grafik gelöscht wird. Zu diesem Zweck kopiert man das BufferedImage in ein temporäres BufferedImage, definiert das alte BufferedImage neu mit veränderten Maßen, und kopiert anschließend die Grafik aus dem temporären Bereich zurück. Zur Registrierung der Größenänderung des JPanels ist ein sog. ComponentListener erforderlich, der mittels eines ComponentResizers die Größe des BufferedImage vornimmt.

Hübscher Nebeneffekt:

Die JButtons lassen sich auf einfache Art und Weise mit Icons dekorieren.
Liegen die betreffenden Icons im aktuellen Verzeichnis, so genügt für einen JButton names "btnZeichne" folgende Anweisung, um ihn mit dem Icon "zeichnen.png" zu versehen:

```
btnZeichne.setIcon(new ImageIcon("zeichnen.png"));
```

Das Java-Programm:

```
package panelGrafikExtended;

import java.awt.BasicStroke;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.FileDialog;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Line2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
```

```java
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import panelGrafikExtended.MeineLeinwand.LinienArt;

// Ac 27.3.2018; verbessert: 16.04.2021

public class Grafik2DBuf3Ereignisse extends JFrame {

   public static void main(String[] args) {
      EventQueue.invokeLater(new Runnable() {
         public void run() {
            try {
               Grafik2DBuf3Ereignisse frame = new Grafik2DBuf3Ereignisse();
               frame.setVisible(true);
            } catch (Exception e) {
               e.printStackTrace();
            }
         }
      });
   }

   private JPanel contentPane;
   MeineLeinwand malPanel;

   public Grafik2DBuf3Ereignisse() {
      setTitle("PanelGrafik  Ac V1.0");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setBounds(100, 100, 500, 350);
      contentPane = new JPanel();
      contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
      contentPane.setLayout(new BorderLayout(0, 0));
      setContentPane(contentPane);

      JPanel pnlSteuerung = new JPanel();
      pnlSteuerung.setBackground(new Color(240, 248, 255));
      contentPane.add(pnlSteuerung, BorderLayout.NORTH);

      JButton btnZeichne1 = new JButton("Zeichne1");
      btnZeichne1.setIcon(new ImageIcon("zeichnen1 24x24.png"));
      btnZeichne1.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            zeichneBild1();
         }
      });
      pnlSteuerung.add(btnZeichne1);

      JButton btnZeichne2 = new JButton("Zeichne2");
      btnZeichne2.setIcon(new ImageIcon("zeichnen2 24x24.png"));
      btnZeichne2.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            zeichneBild2();
         }
      });
      pnlSteuerung.add(btnZeichne2);

      JButton btnLadebild = new JButton("LadeBild");
      btnLadebild.setIcon(new ImageIcon("laden 24x24.png"));
      btnLadebild.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            malPanel.ladeBild();
         }
      });
      pnlSteuerung.add(btnLadebild);

      JButton btnClear = new JButton("Clear");
      btnClear.setIcon(new ImageIcon("abbrechen 24x24.png"));
      btnClear.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            malPanel.clearBufImg();
         }
      });
```

```
      pnlSteuerung.add(btnClear);

      malPanel = new MeineLeinwand(getWidth(), getHeight());
      malPanel.setPreferredSize(new Dimension(getWidth(), getHeight()));
      contentPane.add(malPanel, BorderLayout.CENTER);

   } // Ende Konstruktor Hauptprogramm

   // Methoden Hauptprogramm

   void zeichneBild1() { // auf das Panel zeichnen
      LinienArt pen = LinienArt.SOLID;
      malPanel.zeichneLinie2D(10, 10, 200, 240, 1.0f, pen, Color.BLUE);
      malPanel.zeichneEllipse2D(110, 80, 100, 60, 2.0f, pen, Color.BLUE, Color.YELLOW, true);
      malPanel.zeichneKreisMr(300, 80, 40, 1.5f, pen, Color.RED, Color.ORANGE, true);
      malPanel.zeichneKreisMr(300, 140, 5, 1.5f, pen, Color.RED, Color.BLUE, false); // Kringel ohne
         Füllung
      malPanel.zeichneKreisMr(315, 140, 5, 1.5f, pen, Color.RED, Color.BLUE, true); // Kringel mit
         Füllung
      Font font = new Font("SERIF", Font.BOLD, 14);
      malPanel.schreibeText("Leinwand-Test", 70, 15, font, Color.MAGENTA);
   }

   void zeichneBild2() { // auf das Panel zeichnen
      LinienArt pen = LinienArt.DASH;
      malPanel.zeichneLinie2D(20, 120, 200, 10, 2.0f, pen, Color.RED);
      for (int i = 0; i < 20; i += 2)
         malPanel.zeichnePixel(50 + i, 20 + i, Color.BLACK);
      malPanel.zeichneLinie2D(100, 20, 200, 480, 3.0f, pen, Color.RED);
   }

} // Ende Hauptprogramm

//=======================================
//        Klasse MeineLeinwand
//=======================================

class MeineLeinwand extends JPanel {
   private int breiteBuf;
   private int hoeheBuf;

   private BufferedImage bufImg; // Grafikpuffer im RAM definieren
   private Graphics2D grBufImg; // der "Grafikkontext" des Puffers
   private BufferedImage bufTmp; // zum Vergrößern
   private Graphics2D grBufTmp;

   public enum LinienArt {
      SOLID, DASH
   } // definiert Klasse LinienArt

   float[] dash = { 5, 2 }; // gestrichelt ***** ***** *****
   // public LinienArt pen = LinienArt.SOLID;

   public MeineLeinwand(int breite, int hoehe) { // KONSTRUKTOR
      this.breiteBuf = breite;
      this.hoeheBuf = hoehe;
      // System.out.println(hoeheBuf + " " + this.getHeight());
      // hoeheBuf ist zu groß gewählt, da = getHeight() vom JFrame ; ebenso breiteBuf

      addComponentListener(new ComponentAdapter() {
         @Override
         public void componentResized(final ComponentEvent arg0) {
            vergroessereBufImg(getWidth(), getHeight());
         }
      });

      initBuffer();
   }

   @Override
   public void paintComponent(Graphics grPnl) { // zeichnet auf das Panel "MeineLeinwand"
      Graphics2D g2d = (Graphics2D) grPnl;
      // super.paintComponent(g2d); wird automatisch ausgeführt !
      g2d.drawImage(bufImg, 0, 0, this); // zeichnet bufImg(Puffer) auf "this"=JPanel
```

```
      repaint();
   }

   public void initBuffer() {
      bufImg = new BufferedImage(breiteBuf, hoeheBuf, BufferedImage.TYPE_INT_RGB);
      grBufImg = bufImg.createGraphics();
      grBufImg.setBackground(Color.WHITE);
      clearBufImg();
      // Anti-Aliasing einschalten; vermeidet Treppenstufen
      grBufImg.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
      grBufImg.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
                        RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
   }

   public void clearBufImg() {
      grBufImg.clearRect(0, 0, breiteBuf, hoeheBuf); // füllt bufImg mit (weißer) Hintergrundfarbe
   }

   public void vergroessereBufImg(int breite, int hoehe) {
      if (breite > breiteBuf || hoehe > hoeheBuf) { // nur, falls Leinwand vergrößert wird !
         if (breite > breiteBuf)
            breiteBuf = breite;
         if (hoehe > hoeheBuf)
            hoeheBuf = hoehe;
         // temporären Puffer erzeugen
         bufTmp = new BufferedImage(breiteBuf, hoeheBuf, BufferedImage.TYPE_INT_RGB);
         grBufTmp = bufTmp.createGraphics();
         grBufTmp.setBackground(Color.WHITE);
         grBufTmp.clearRect(0, 0, breiteBuf, hoeheBuf);
         grBufTmp.drawImage(bufImg, 0, 0, null); // Bild von bufImg in bufTmp sichern
         grBufTmp.dispose(); // temporären Speicher freigeben
         // bufImg mit neuer Grenze erzeugen
         initBuffer();
         grBufImg.drawImage(bufTmp, 0, 0, null); // Grafik in bufImg zurückkopieren
      }
   }

   // die geometrischen Figuren

   public void setzeLinienStil(float dicke, LinienArt pen) {
      switch (pen) {
      case SOLID:
         grBufImg.setStroke(new BasicStroke(dicke, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND));
         break;
      case DASH:
         grBufImg.setStroke(new BasicStroke(dicke, BasicStroke.CAP_BUTT, BasicStroke.JOIN_ROUND, 1,
            dash, 0));
         break;
      }
   }

   public void zeichnePixel(int sp, int ze, Color farbe) {
      bufImg.setRGB(sp, ze, farbe.getRGB());
   }

   public void zeichneLinie2D(int x1, int y1, int x2, int y2, float dicke, LinienArt pen, Color col) {
      // Strecke von (x1,y1) nach (x2,y2)
      setzeLinienStil(dicke, pen);
      // Linien-Shape definieren
      Line2D.Double linie2d = new Line2D.Double(x1, y1, x2, y2);
      grBufImg.setColor(col);
      grBufImg.draw(linie2d);
      grBufImg.setStroke(new BasicStroke(1.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND));
   }

   public void zeichneEllipse2D(int x1, int y1, int dx, int dy, float dicke, LinienArt pen,
         Color randFarbe, Color fuellFarbe, boolean gefuellt) {
      setzeLinienStil(dicke, pen);
      // erst das Shape definieren
      Ellipse2D.Double ellip2d = new Ellipse2D.Double(x1, y1, dx, dy);
      if (gefuellt) {
         grBufImg.setColor(fuellFarbe);
         grBufImg.fill(ellip2d);
      }
```

```
        grBufImg.setColor(randFarbe);
        grBufImg.draw(ellip2d);
        // Liniendicke zurücksetzen
        grBufImg.setStroke(new BasicStroke(1.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND));
    }

    public void zeichneKreisMr(int xM, int yM, int radius, float dicke, LinienArt pen, Color randFarbe,
                  Color fuellFarbe, boolean gefuellt) {
        // zeichnet den Kreis bei bekanntem Mittelpunkt und Radius !!
        zeichneEllipse2D(xM - radius, yM - radius, 2 * radius, 2 * radius, dicke, pen, randFarbe,
                         fuellFarbe, gefuellt);
    }

    public void schreibeText(String text, int xpos, int ypos, Font fontTxt, Color farbe) {
        grBufImg.setFont(fontTxt);
        grBufImg.setColor(farbe);
        grBufImg.drawString(text, xpos, ypos);
    }

    // Bild laden

    public void ladeBild() { // mit LadeMenü
        JFrame dateiDialogFrame = new JFrame();
        String bildDateiListe = "*.jpg;*.bmp;*.gif;*.png";
        try {
            FileDialog d = new FileDialog(dateiDialogFrame, "Bild laden", FileDialog.LOAD);
            d.setFile(bildDateiListe); // nur Bilder zulassen
            d.setVisible(true);
            if (d.getFile() == null)
                return;
            String pfadName = d.getDirectory() + d.getFile();
            File bildDatei = new File(pfadName);
            BufferedImage bufTmp = ImageIO.read(bildDatei);
            vergroessereBufImg(bufTmp.getWidth(), bufTmp.getHeight());
            grBufImg.drawImage(bufTmp, 0, 0, null);
        } catch (IOException e) {
            JOptionPane.showMessageDialog(null, "Fehler beim Öffnen der Datei!");
        }
    }

} // Ende Klasse MeineLeinwand
```

Ergebnis (Fenster nachträglich angepasst !):