Es gibt verschiedene Methoden, in Java Grafik zu erzeugen:

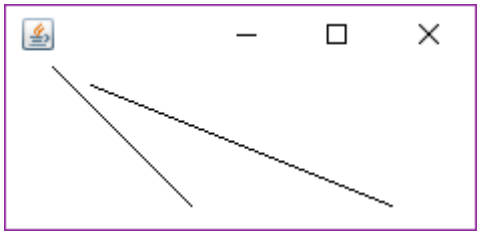
1) Grafik in einem Frame (hier: JFrame)

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

Das einfachste Programm dieser Art sieht so aus:

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

public class GrafikJFrame0 extends JFrame {

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

       public GrafikJFrame0() {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(250, 120);
        setVisible(true);
    }

    public void paint(Graphics g) { // g ist der "Grafikkontext" (Zeichenumgebung)
       g.drawLine(0, 0, 100, 100);
       g.drawLine(50, 40, 200, 100);
    }

}
```

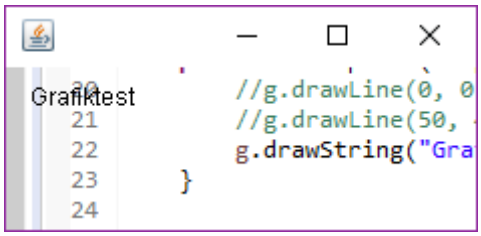
Ergebnis:



Wie man sieht, wird die Linie von der oberen Titelleiste verdeckt, denn (0,0) befindet sich in der oberen linken Ecke. Die Titelleiste nimmt eine Höhe von 31 Pixels ein, wie wir weiter unten noch sehen werden, außerdem gibt es noch links, rechts und unten (nicht sichtbare) Ränder.

Es kann auch noch ganz andere **Probleme** geben:
Ersetzt man nämlich die drawLine-Anweisungen durch z.B. `g.drawString("Grafiktest", 20, 50);`
so erhalten wir einen JFrame mit "eingebettetem" Windows-Hintergrund !!

Anmerkung: Nur bei JFrame, nicht so bei Frame (AWT).



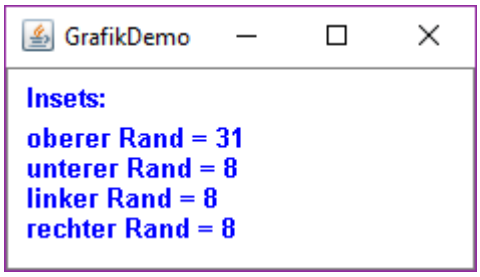
Das Problem lässt sich lösen, wenn man vor der drawString-Aktion entweder **super.paint(g);** aufruft (Nachteil: grauer Hintergrund) oder (besser) einen rechteckigen Rahmen auf dem JFrame zeichnen lässt.

Die Maße für den Rahmen 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 das Rechteck, welches einen Rahmen bekommen soll.

Das folgende Programm ermittelt die Ränder (Insets), zeichnet einen rechteckigen Rahmen und gibt die Maße der Insets auf dem JFrame aus:

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

public class GrafikJFrame extends JFrame{

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

	int frmWidth = 250, frmHeight = 140;

	public GrafikJFrame() {  // Konstruktor
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(frmWidth, frmHeight);
		setLocationRelativeTo(null);
		setTitle("GrafikDemo");
		setVisible(true);
	}

	@Override
	public void paint(Graphics g) {
	   // Insets für das Grafikfenster ermitteln; auf den 4 Randbereichen wird nicht gezeichnet !!
	   Insets ins = getInsets();
	   int obRand = ins.top, untRand = ins.bottom, liRand = ins.left, reRand = ins.right;

	   // Rahmen für das Grafikfenster zeichnen
	   g.setColor(Color.GRAY);
	   g.drawRect(liRand, obRand, frmWidth-liRand-reRand-1, frmHeight-obRand-untRand-1);

	   // Text im JFrame ausgeben
	   g.setColor(Color.BLUE);
	   g.setFont(new Font("Arial", Font.BOLD, 13));
	   g.drawString("Insets:", liRand + 10, 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);
	}
}
```

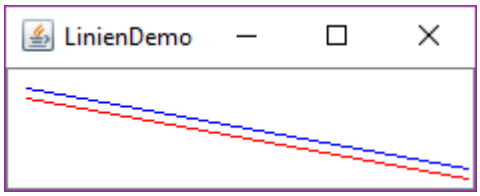
Ergebnis:



Zum Zeichnen von Linien sind nur kleine Änderungen in der paint-Methode notwendig:

```
	// Linien im JFrame ausgeben
	g.setColor(Color.BLUE);
	g.drawLine(links + 10, oben + 10, links + 230, oben + 50);
	g.setColor(Color.RED);
	g.drawLine(links + 10, oben + 15, links + 230, oben + 55);
```

Ergebnis:

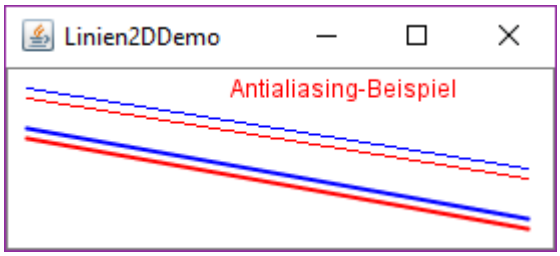


Sollen die Linien dicker werden, dann geht das nur unter Verwendung von **Java2D** bzw. **Graphics2D**.
Dazu muss das Graphics-Objekt der paint-Methode in ein Graphics2D-Objekt "gecastet" werden.
Dann können alle Vorteile von Java2D genutzt werden, u.a. auch "Antialiasing" (= Kantenglättung).

Die paint-Methode könnte so aussehen:

```
public void paint(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;

     // Insets für das Grafikfenster ermitteln
       ...  usw.
       ...

     // 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
     // erst für die Linien
     g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
     // dann für den Text
     g2d.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,RenderingHints.VALUE_TEXT_ANTIALIAS_ON);

     // Dicke der Linien festlegen
     float dicke = 2.0f;
     g2d.setStroke(new BasicStroke(dicke, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND));

     // 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:



Anmerkung:

Experten raten davon ab, direkt auf ein JFrame zu zeichnen, weil es zu Komplikationen kommen kann.

Besser ist es , ein JPanel zu verwenden (s. unten) !

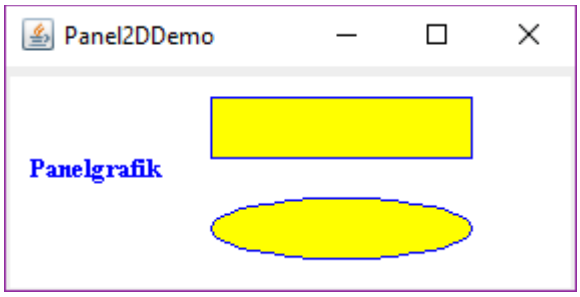
2) Grafik in einem JPanel

Eine bessere Alternative zum Zeichnen von Grafik ist ein JPanel, das in ein JFrame "eingehängt" wird.
Mit der **paintComponent-Methode** des Panels wird gezeichnet. 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 {

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

	int frmWidth = 300, frmHeight = 150;
	Leinwand malPanel = new Leinwand();
	JPanel contentPane;

	public Grafik2DJPanel() {  // Konstruktor
	    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	    setSize(frmWidth, frmHeight);
	    setLocationRelativeTo(null);
	    setTitle("Panel2DDemo");
	    contentPane = new JPanel();
	    setContentPane(contentPane);

	    malPanel.setPreferredSize(new Dimension(frmWidth - 20, frmHeight - 45));
	    malPanel.setBackground(Color.WHITE);
	    contentPane.add(malPanel);
	    setVisible(true);
	}
}

class Leinwand extends JPanel {

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

	    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("Arial", Font.BOLD, 13));
	    g2d.drawString("Panelgrafik", 10, 50);
	}
}
```

Ergebnis:

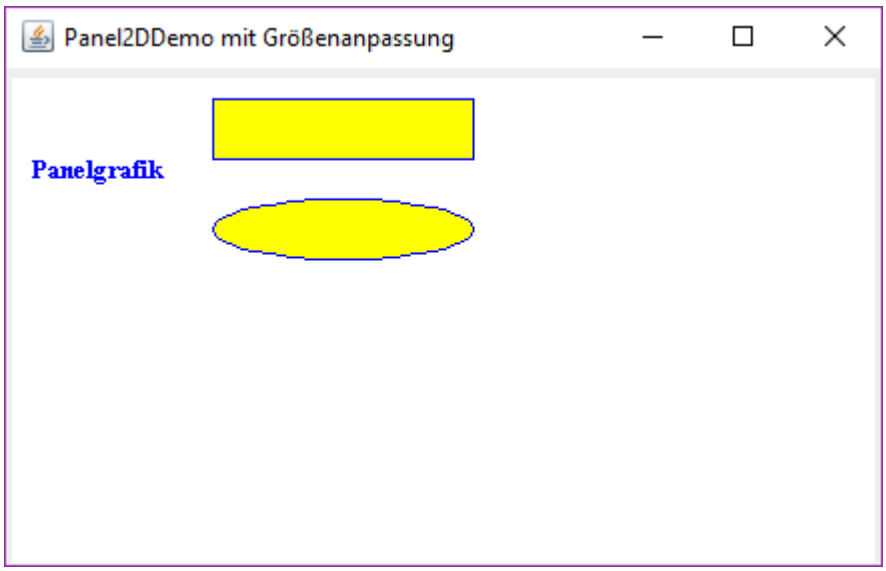


Störend ist nur, dass beim Ändern der Größe des JFrame das JPanel statisch bleibt.
Dies kann man aber mittels "componentResized()" ändern .

Es ist lediglich ein zusätzlicher ComponentListener im Konstruktor vonnöten:

```
public Grafik2DJPanel() {  // Konstruktor
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(frmWidth, frmHeight);
        setLocationRelativeTo(null);
        setTitle("Panel2DDemo mit Größenanpassung");
        contentPane = new JPanel();
        setContentPane(contentPane);

        addComponentListener(new ComponentAdapter() {
                @Override
                public void componentResized(final ComponentEvent arg0) {
                   malPanel.setPreferredSize(new Dimension(getWidth() - 20, getHeight() - 45));
                }
        });

        malPanel.setPreferredSize(new Dimension(frmWidth - 20, frmHeight - 45));
        malPanel.setBackground(Color.WHITE);
        contentPane.add(malPanel);
        setVisible(true);
}
```

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.

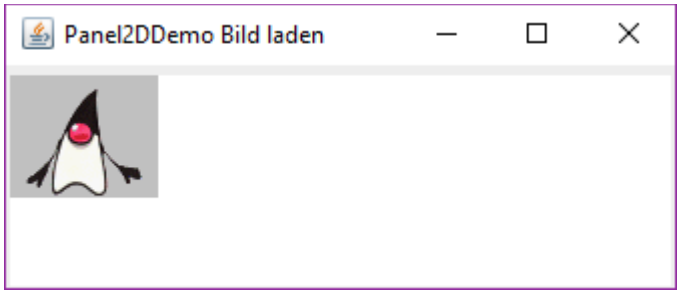
Java stellt zum Laden von Bildern die Klasse ImageIO zur Verfügung.
Hat man nun ein Bild (etwa eine png-Datei namens "duke.png") in einem Verzeichnis gespeichert, so kann man es mittels **bild = ImageIO.read(new File("duke.png");** in einen Bereich im Speicher laden, der vorher als **Image bild;** deklariert wurde.

Voraussetzung ist, dass die Datei "duke.png" sich im aktuellen Verzeichnis befindet.

Die entsprechende class Leinwand sieht so aus:

```
class Leinwand extends JPanel {
        private Image bild;

        public Leinwand() { // Konstruktor
           try {
               bild = ImageIO.read(new File("duke.png"));
           } catch (IOException e) { }
        }

        public void paintComponent(Graphics g) {
               super.paintComponent(g);
               g.drawImage(bild, 0, 0, this);
        }
}
```

Ergebnis:



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

```
class Leinwand extends JPanel {
        private Image bild = null;

        public Leinwand() {
            ladeBild();
        }

        public void paintComponent(Graphics g) {
               super.paintComponent(g);
               if (bild != null)
                    g.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!");
            }
        }
}
```

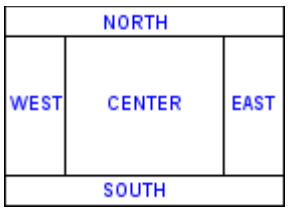
Ergebnis:

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. Dann kann auch der ComponentListener entfallen !

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



Wir legen fest, dass CENTER die Zeichenfläche aufnimmt und NORTH die JButtons.
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" .

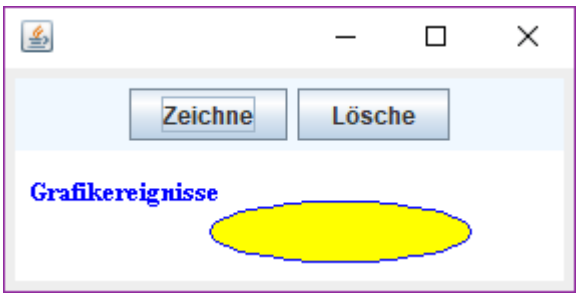
Zusätzlich wird das Hauptprogramm (main) gemäß Java-Konvention in einem eigenen Thread abgearbeitet.

Das Java-Programm:

```
import java.awt.BorderLayout;
import java.awt.EventQueue;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JButton;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

@SuppressWarnings("serial")
public class Grafik2DEreignisgesteuert extends JFrame {

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

	private JPanel contentPane;
```

```
	boolean zeichnen = true;

	public Grafik2DEreignisgesteuert() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setBounds(100, 100, 300, 150);
		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 void paintComponent(Graphics g) {
			Graphics2D g2d = (Graphics2D) g;
			super.paintComponent(g2d);

			if (zeichnen) {
				g2d.setColor(Color.YELLOW);
				g2d.fillOval(100, 20, 130, 30);
				g2d.setColor(Color.BLUE);
				g2d.drawOval(100, 20, 130, 30);
				g2d.setFont(new Font("SERIF", Font.BOLD, 13));
				g2d.drawString("Grafikereignisse", 10, 20);
			}

		}
	}

}
```

Ergebnis:



Ereignisgesteuert zeichnen (2)

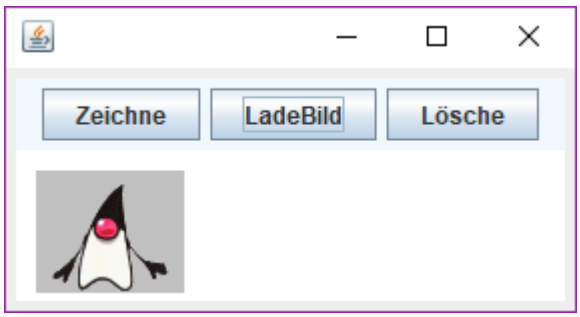
Das Programm wird nun um das Laden eines Bildes erweitert , was einen zusätzlichen Button erfordert.
Die Boolean-Variable wird wegen der 3 zu unterscheidenden Möglichkeiten (zeichnen, laden, löschen) ersetzt durch eine Aufzählungsvariable "enum" .

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;

@SuppressWarnings("serial")
public class Grafik2D3Ereignisse extends JFrame {

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

	private JPanel contentPane;
	enum Aktion {ZEICHNE, LADE, LOESCHE};
	Aktion aktion = Aktion.ZEICHNE; // vordefiniert

	public Grafik2D3Ereignisse() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setBounds(100, 100, 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() - 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) {
				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) {
			Graphics2D g2d = (Graphics2D) g;
			super.paintComponent(g2d);

			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);
			}
		}
	}

}
```

Ergebnis:



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

3) Grafik 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.

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:

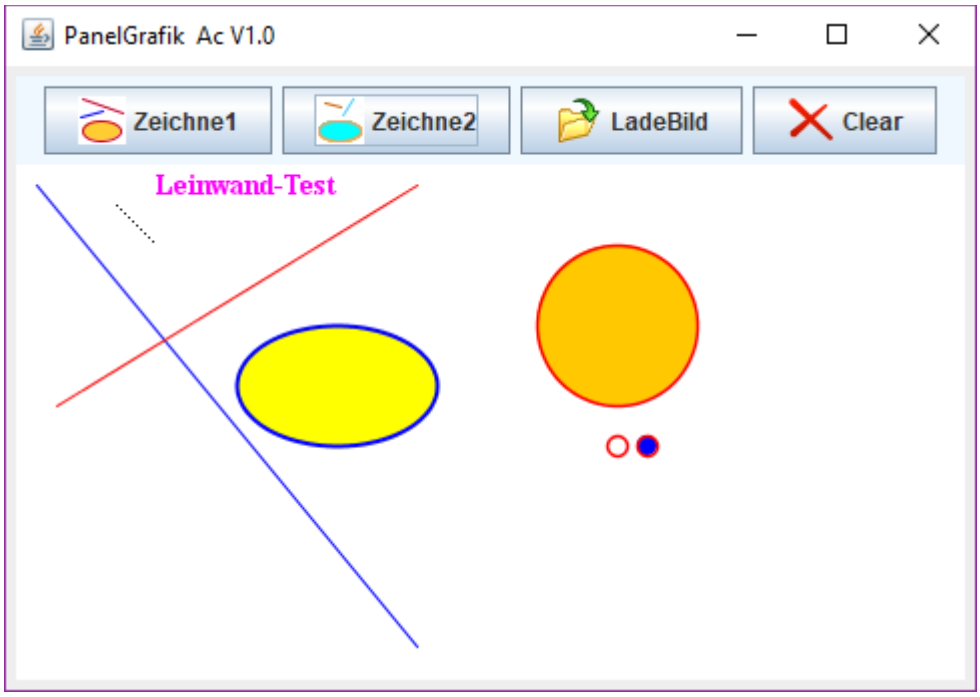
```
import java.awt.BasicStroke;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.FileDialog;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.ActionEvent;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

```
public class Grafik2DBuf3Ereignisse extends JFrame {  // Ac 27.3.2018 bis 28.3.2018

	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;
	Leinwand malPanel = new Leinwand();

	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) {
	           zeichneBild();
	       }
	    });
	    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) {
	           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) {
	           loeschePanel();
	       }
	    });
	    pnlSteuerung.add(btnClear);

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

	}  // Ende Konstruktor Hauptprogramm

	// Methoden Hauptprogramm

	void zeichneBild() { // auf den Puffer zeichnen
```

```
            malPanel.zeichneLinie2D(10, 10, 200, 240, Color.BLUE);
            malPanel.zeichneEllipse2D(110, 80, 100, 60, 2.0f, Color.BLUE, Color.YELLOW, true);
            malPanel.zeichneKreisMr(300, 80, 40, 1.5f, Color.RED, Color.ORANGE, true);
            malPanel.zeichneKreisMr(300, 140, 5, 1.5f, Color.RED, Color.BLUE, false); // ohne Füllung
            malPanel.zeichneKreisMr(315, 140, 5, 1.5f, Color.RED, Color.BLUE, true); // mit Füllung
            Font font = new Font("SERIF", Font.BOLD, 14);
            malPanel.schreibeText("Leinwand-Test", 70, 15, font, Color.MAGENTA);
        }

        void zeichneBild2() { // auf den Puffer zeichnen
            malPanel.zeichneLinie2D(20, 120, 200, 10, Color.RED);
            for (int i = 0; i < 20; i += 2)
               malPanel.zeichnePixel(50 + i, 20 + i, Color.BLACK);
        }

        void ladeBild() {
            malPanel.ladeBild();
        }

        void loeschePanel() {
            malPanel.clearBufImg();
        }

} // Ende Hauptprogramm


//==================================
//          Klasse Leinwand
//==================================

class Leinwand extends JPanel {
        int breiteBuf;
        int hoeheBuf;

        BufferedImage bufImg; // Grafikpuffer im RAM definieren
        Graphics2D grBufImg; // der "Grafikkontext" des Puffers

        public Leinwand() { // Konstruktor
            setBackground(Color.WHITE);
        }

        public void initPanel() {
            breiteBuf = this.getPreferredSize().width;
            hoeheBuf = this.getPreferredSize().height;
            bufImg = new BufferedImage(breiteBuf, hoeheBuf, BufferedImage.TYPE_INT_RGB);
            grBufImg = bufImg.createGraphics();
            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);
        }

        @Override
        public void paintComponent(Graphics grPnl) { // zeichnet auf das Panel
            Graphics2D g2d = (Graphics2D) grPnl;
            super.paintComponent(g2d);
            g2d.drawImage(bufImg, 0, 0, this); // zeichnet bufImg(Puffer) auf "this"=JPanel
            repaint();
        }

        public void clearBufImg() {
            grBufImg.setColor(Color.WHITE);
            grBufImg.fillRect(0, 0, breiteBuf, hoeheBuf); // weißer Hintergrund für Puffer
        }

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

        public void zeichneLinie2D(double x1, double y1, double x2, double y2,  Color farbe) {
            // Strecke von (x1,y1) nach (x2,y2)
            // Linien-Shape definieren
```

```
        Line2D.Double linie2d = new Line2D.Double(x1, y1, x2, y2);
        grBufImg.setColor(farbe);
        grBufImg.draw(linie2d);
    }

    public void zeichneEllipse2D(int x1, int y1, int dx, int dy, float dicke, Color randFarbe,
                        Color fuellFarbe, boolean gefuellt) {
        grBufImg.setStroke(new BasicStroke(dicke, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND));
        // 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 und -art zurücksetzen
        grBufImg.setStroke(new BasicStroke(1.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND));
    }

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

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

    public void ladeBild() { // mit LadeMenü
        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);
           vergroessereBuf(bufTmp.getWidth(), bufTmp.getHeight());
           grBufImg.drawImage(bufTmp, 0, 0, null);
        } catch (IOException e) {
           JOptionPane.showMessageDialog(null, "Fehler beim Öffnen der Datei!");
        }
    }

    public void vergroessereBuf(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
           BufferedImage bufTmp = new BufferedImage(breiteBuf, hoeheBuf,
                                                        BufferedImage.TYPE_INT_RGB);
           Graphics2D grBufTmp = bufTmp.createGraphics();
           grBufTmp.drawImage(bufImg, 0, 0, null); // Bild von bufImg in bufTmp sichern
           grBufTmp.dispose(); // temporären Speicher freigeben
           // bufImg mit neuer Grenze erzeugen
           bufImg = new BufferedImage(breiteBuf, hoeheBuf, BufferedImage.TYPE_INT_RGB);
           grBufImg = bufImg.createGraphics();
           grBufImg.drawImage(bufTmp, 0, 0, null); // Grafik in bufImg zurückkopieren
        }
    }
} // Ende Klasse Leinwand
```

Ergebnis:


PanelGrafik Ac V1.0
Zeichne1
Zeichne2
LadeBild
Clear
Leinwand-Test