BufferedImage (Bildverarbeitung in JAVA) Ac

Ein ***BufferedImage*** ist ein (unsichtbarer) Bereich im Speicher (RAM) des Computers, in den gezeichnet werden kann.

Das so erzeugte Bild kann dann in eine Komponente (Panel etc.) übertragen oder aber extern (z.B. Festplatte) gespeichert werden. Ebenso kann ein externes Bild (.bmp, .jpg, .png) in ein BufferedImage geladen werden. Die Oberklasse zu BufferedImage heißt ***Image*** und besitzt ähnliche Eigenschaften, jedoch **nicht** die Eigenschaft der externen Speicherung.

Ein BufferedImage ist also notwendig zum Laden und Speichern von Bildern, und auch zum Zwischenspeichern von Bildern im Arbeitsspeicher.

Im folgenden müssen noch einige „Packages“ importiert werden,
z.B. java.awt.Image, javax.imageio.ImageIO , etc .

Erzeugung eines BufferedImage (willkürliche Bezeichnung bufImg):

Mittels des folgenden Konstruktors wird ein Speicherbereich für ein Standard-RGB-Bild reserviert:

```
BufferedImage bufImg = new BufferedImage(90,60,BufferedImage.TYPE_INT_RGB);
```

Dies erzeugt ein (nicht sichtbares) Bild der Breite 90 und der Höhe 60 im Speicher des Computers.

Achtung: Die pure Deklaration `BufferedImage bufImg;` reserviert noch keinen Speicherbereich !

Zeichnen in ein BufferedImage:

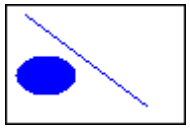
Für das Zeichnen in das BufferedImage verwendet man ein Graphics-Objekt.

```
Graphics gBuf = bufImg.createGraphics();
```

erzeugt ein solches Objekt, für das jetzt alle Methoden der Graphics-Umgebung bereit stehen. Es folgen 2 Beispiele:

```
gBuf.setColor(Color.WHITE);
gBuf.fillRect(1,1,88,58);
gBuf.setColor(Color.YELLOW);
gBuf.fillRect(10,5,70,50);
```

```
gBuf.setColor(Color.WHITE);
gBuf.fillRect(1, 1, 88, 58);
gBuf.setColor(Color.BLUE);
gBuf.drawLine(10, 5, 70, 50);
gBuf.fillOval(5, 25, 30, 20);
```



BufferedImage auf einem Panel sichtbar machen:

Will man ein BufferedImage sichtbar machen, so verwendet man die Methode drawImage(), mit der man das BufferedImage z.B. auf ein Panel zeichnen (oder aber in ein anderes BufferedImage bzw. Image übertragen) kann.

Wir definieren ein Panel mit den gleichen Maßen wie das BufferedImage und eine Methode, mit der wir das Bild übertragen können:

```
JPanel panelBufImg = new JPanel();
panelBufImg.setBounds(200,50,90,60);
contentPane.add(panelBufImg);

void bildZeigen() {
    panelBufImg.getGraphics().drawImage(bufImg, 0, 0, this);
}
```

Die Koordinaten 0, 0 geben dabei die linke, obere Ecke des Bildes innerhalb von panelBufImg an.
Als „ImageObserver“ wird hier „this“, eine Referenz auf die aufrufende Panel-Instanz, hier: panelBufImg, verwendet .

Auf diese Art wurden übrigens die obigen rechteckigen Bilder erzeugt !

Panel in ein BufferedImage übertragen:

Dies ist möglich, jedoch werden lediglich die **Komponenten** des Panels (Buttons, TextField, BackGround, etc.) übertragen !
Die Übertragung einer mit Graphics erstellten Zeichnung ist nicht möglich !
Hier muss erst in ein BufferedImage gezeichnet und dann die Zeichnung auf das Panel übertragen werden (s.o.).

BufferedImage Laden und Speichern:

Laden mittels ImageIO (das Bild möge unter "e:\bild01.jpg" gespeichert sein):

```
try {
     bufImg = ImageIO.read(new File("e:\bild01.jpg"))
}
catch (IOException ex) {
     ex.printStackTrace();
}
```

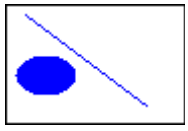
Speichern:

```
try {
     if(bufImg == null)
        System.out.println("kein Bild vorhanden");
     else
        ImageIO.write(bufImg,"jpeg",new File("e:\bild01.jpg"));
}
catch (IOException ex) {
     ex.printStackTrace();
}
```

Will man evtl. Wartezeiten vermeiden, so verwende man den sog. „MediaTracker" zum Überwachen des Transfers !

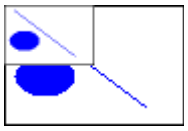
```
MediaTracker mt = new MediaTracker(this);
mt.addImage(bufImg,0);
try {
     mt.waitForID(0);
}
catch (InterruptedException ex){
     ex.printStackTrace();
}
```

BufferedImage skalieren:



Ein BufferedImage kann man folgendermaßen skalieren (vergrößern oder verkleinern).

```
Image sclImg = bufImg.getScaledInstance(45, 30, Image.SCALE_SMOOTH );
bufImg.getGraphics().drawImage(sclImg, 0, 0, null);
```

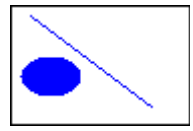


Es wird also die Klasse Image zur Zwischenspeicherung benötigt .
Rechts ist dargestellt, wie sich das Bild verändert.

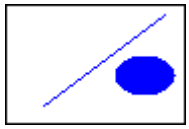
Will man aber ein ganz neues Bild erzeugen, so muss ein zweites BufferedImage deklariert werden:

```
BufferedImage bufImg2 = new BufferedImage(45, 30, BufferedImage.TYPE_INT_RGB);
Image sclImg = bufImg.getScaledInstance(45, 30, Image.SCALE_SMOOTH );
bufImg2.getGraphics().drawImage(sclImg, 0, 0, null);
```

BufferedImage spiegeln:

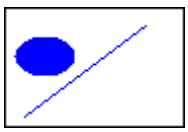


Zunächst wird wieder das zweite BufferedImage benötigt. Es dient als Ziel der Umwandlung.



```
public void bildSpiegelnVert() { // vertik. Achse;  aus Buf in Buf2 schreiben
     for (int i = 0; i < xmax; i++)
        for (int k = 0; k < ymax; k++)
           bufImg2.setRGB(xmax - 1 - i, k, bufImg.getRGB(i, k));
     }

public void bildSpiegelnHoriz() { // horiz. Achse;  aus Buf in Buf2 schreiben
   for (int i = 0; i < xmax; i++)
      for (int k = 0; k < ymax; k++)
         bufImg2.setRGB(i, ymax - 1 - k, bufImg.getRGB(i, k));
     }
```



Screenshot erstellen:

```
try { // Bildschirmgröße ermitteln
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize() ;
    Rectangle screenRect = new Rectangle(screenSize);
    Robot robbie = new Robot();  // Screenshot durchführen
    BufferedImage screenShot = robbie.createScreenCapture(screenRect) ;
}
catch(AWTException ex){}
```

Mit einem Rectangle-Objekt kann man auch nur einen Teil des ganzen Bildschirms beschreiben.