NetBeans GUI-Erstellung mit dem GUI-Builder

Version 2.1, Nov. 2007

1 Voraussetzungen

Der Gui-Builder setzt NetBeans Version 5.0 (oder höher) und die JDK Version SE 5 (oder höher) voraus. Unter der JRE Version 5 muss sich zu Laufzeit die Jar-Datei *swing-layout-nnn.jar* (nnn bezeichnet eine Versionsnummer) im Klassenpfad befinden. Sie befindet sich im Verzeichnis *NetBeans_installation_folder/platform6/modules/ext.* Ab JDK/JRE Version 6 ist sie in der API-Distribution integriert.

2 Einstellen der Vorlage

Es ist anzuraten, in NetBeans folgende Einstellungen vorzunehmen:

Extras | *Vorlagen einrichten*
Öffnen im Fenster *Templates*: *Java GUI-Formular* und darin *JFrame-Formular* anwählen. Button *Im Editor öffnen* klicken. Das Template wie folgt ändern:

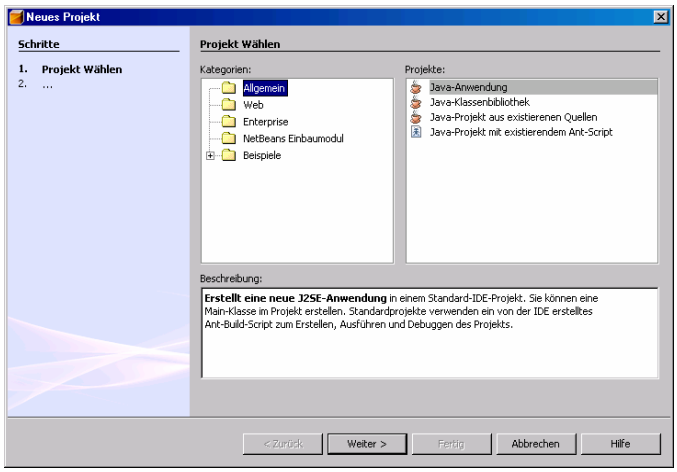
```
// __NAME__.java

import javax.swing.*;
import ch.aplu.util.*;

public class __NAME__ extends JFrame
{
  public __NAME__()
  {
    initComponents();
  }

    Generated Code
   public static void main(String[] args)
   {
    java.awt.EventQueue.invokeLater(new Runnable()
    {
      public void run()
      {
        new __NAME__().setVisible(true);
      }
    });
   }

    // Variables declaration - do not modify
    // End of variables declaration

}
```

Zum Speichern *Ctrl-S* drücken.

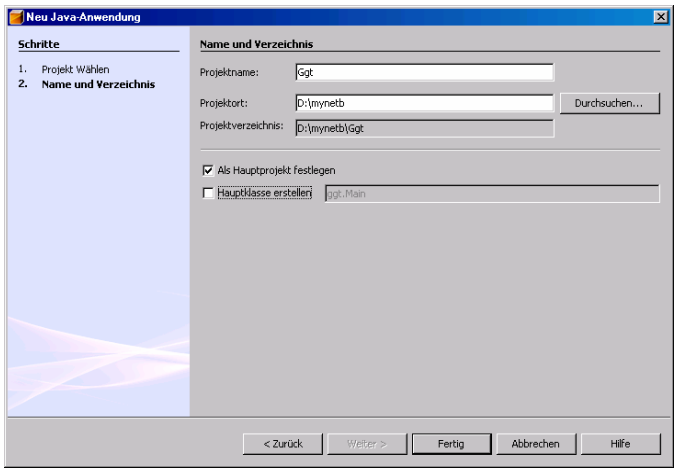
3 Einfaches Beispiel

Erstellen einer Applikation zum Einlesen von zwei Zahlen. Es wird der Ggt ausgeschrieben.

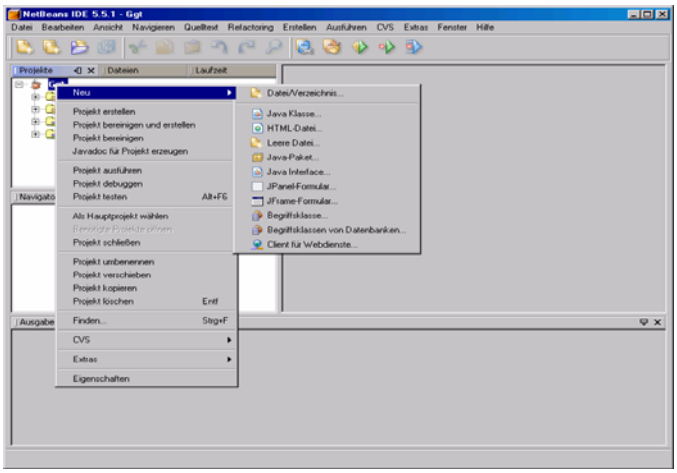
Datei | Neues Projekt
- Kategorien: Allgemein
-Projekte: Java-Anwendung



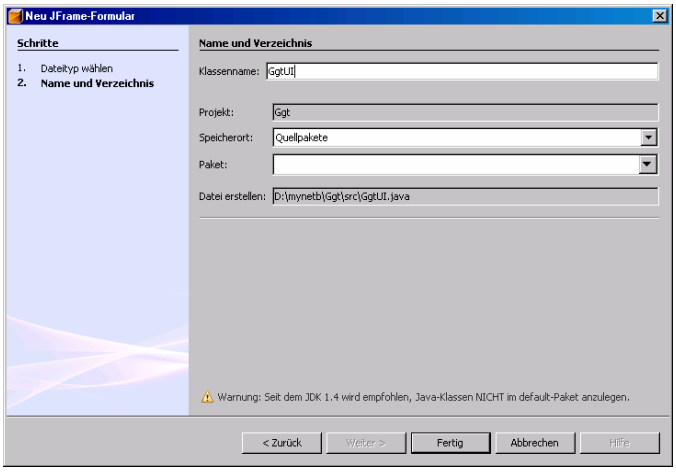
Weiter klicken. Projektname einsetzen, beispielsweise *Ggt*. Projektort einsetzen, z.B. *D:\mynetb*. *Hauptklasse erstellen* abwählen.



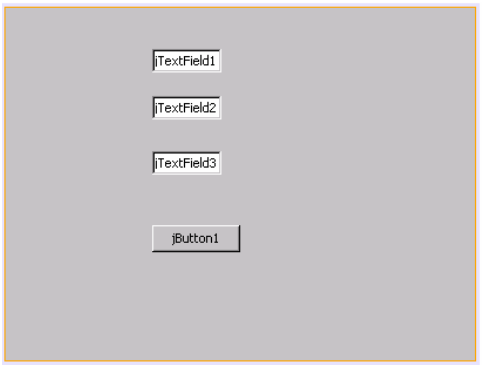
Fertig klicken. Im Fenster *Projekte* rechter Mausklick auf *Ggt*. Unter *Neu, JFrame-Formular* wählen:



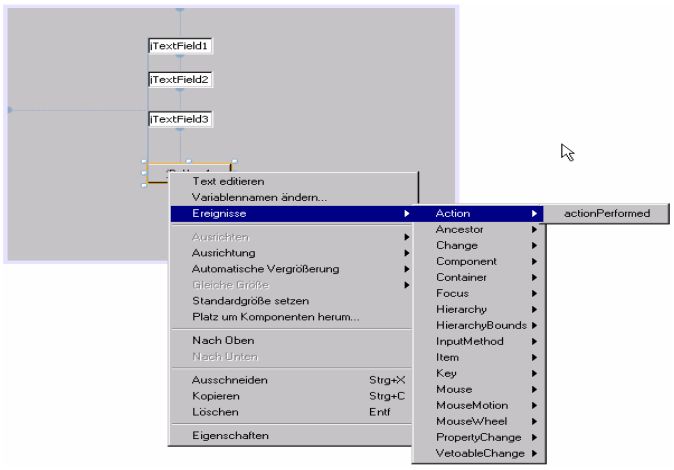
Im Dialog *Klassenname* eingeben, z.B. *GgtUI*:



Fertig klicken. Der GUI-Builder wird angezeigt.
Im Fenster Palette auf *JTextField* klicken, in das Formular gehen an gewünschter Stelle ablegen. Dasselbe mit zwei weiteren *JTextField* und einem *JButton*. Das Formular sieht etwa so aus:



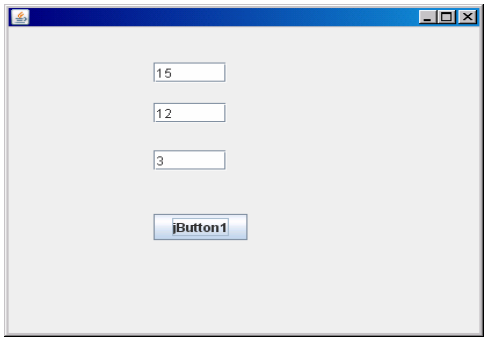
Rechter Mausklick auf den Button *jButton1*. Pulldown-Menüs *Ereignisse* | *Action* | *actionPerformed* wählen.



Der Source-Code wird sichtbar. (Sonst auf Toolbar Button *Quelltext* klicken.) Folgenden Code eintragen:

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
  String value1 = jTextField1.getText();
  String value2 = jTextField2.getText();
  int a = Integer.parseInt(value1);
  int b = Integer.parseInt(value2);

  jTextField3.setText(Integer.toString(ggt(a, b)));
}

private int ggt(int a, int b)
{
  if (b == 0)
    return a;
  return ggt(b, a % b);
}
```

F6 klicken. Das Programm wird compiliert und ausgeführt. In die oberen Edit-Felder zwei Zahlen eingeben und auf den Button klicken. Das Resultat erscheint an der Stelle des dritten Edit-Feldes.



Dann *Datei* | *Beenden*.

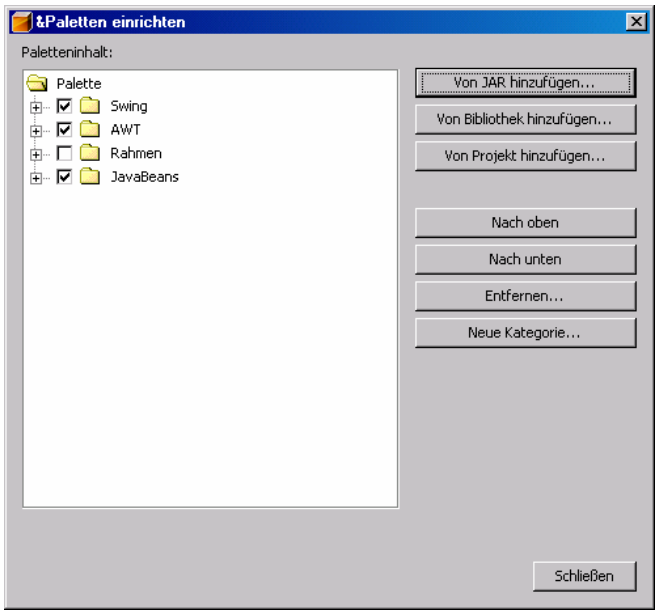
Verbesserungen: Den Text auf den Komponenten entfernen. Dazu auf Komponente klicken und im Eigenschaftseditor *text* auf leer stellen, und *colums* auf 10 setzen. Man kann auch einen Exit-Button hinzufügen, und *System.exit(0)* in der Event-Methode aufrufen:

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{
  System.exit(0);
}
```

Der vom Gui-Builder erzeugte Code ist blau unterlegt und kann nicht verändert werden. Der vollständige erzeugte Initialisierungscode wird sichtbar, wenn man auf das + in der Zeile *Erzeuger Quelltext* klickt. Dieser kann mit Kopieren & Einfügen in jede andere Java-IDE übernommen werden.

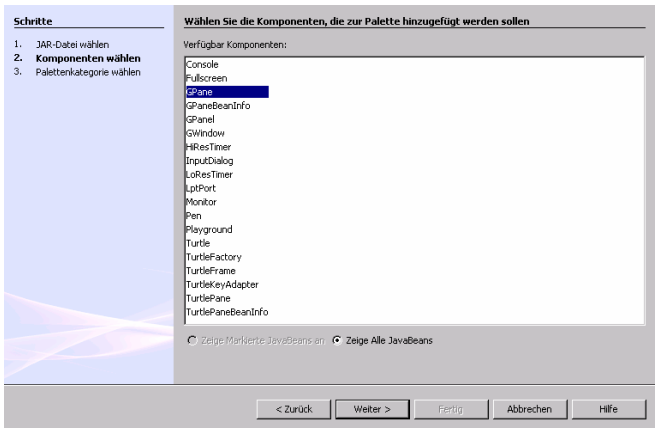
4 Verwendung eines GPanels als Komponente

Zuerst wird die neue Komponente in die Palette der Swing-Komponenten aufgenommen. Dazu *Extras | Palette einrichten | Swing, AWT-Komponenten* wählen. Button *Von JAR hinzufügen* klicken

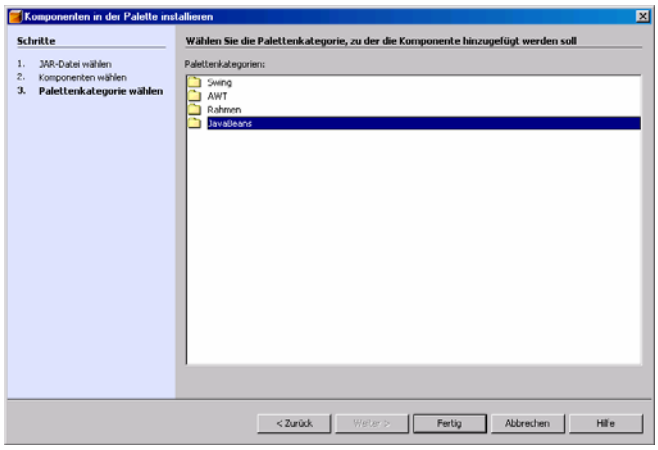


Im erscheinenden Dialog die Datei *aplu5.jar* angeben, die sich beispielsweise auf *c:\jars* befindet. (Download von *www.aplu.ch/download*).

Es werden die verfügbaren Komponenten angezeigt. Die Komponente *GPane* wählen (Achtung nicht mit *GPanel* verwechseln.)



Weiter drücken. Im Fenster *Palettenkategorien* die Zeile *JavaBeans* auswählen



Fertig und dann *Schließen* drücken.

Wie oben beschrieben ein neues GUI Projekt erstellen. Im Entwurfsmodus erscheint in der Palette in der Kategorie *JavaBeans* die neue Komponente *GPane*. Man kann diese auf das Formular ziehen. Als Eigenschaften können die Hintergrundfarbe, die Strichfarbe, sowie die Windowkoordinaten eingestellt werden. Diese Größen sind aber auch im Code wählbar.

Öffnet man den Quelltext, so erkennt man als neue Komponente *gPane1* vom Typ *GPane*. Um in das GPanel zu zeichnen, muss eine Eventmethode oder ein eigener Thread verwendet werden (also nicht im Konstruktor). Beispielsweise setzt man einen Button aufs Formular und schreibt in der Eventmethode

```
private void
  jButton1ActionPerformed(java.awt.event.ActionEvent evt)
  {
    gPane1.move(0.5, 0.5);
    gPane1.fillCircle(0.3);
  }
```

Dabei darf nicht vergessen werden, das Package *ch.aplu.util.** zu importieren.

Startet man die Applikation, so muss beim Klicken auf den Button der schwarze Kreis sichtbar werden. Unter Eigenschaften kann die Hintergrund- und die Strichfarbe geändert

werden, am einfachsten, indem man auf den rechten Button klickt (es geht ein Farbdialog auf). Ebenfalls lässt sich die Größe und die Lage des *GPane* auf dem Formular verändern.

5 Verwendung eines eigenen Threads bei länger dauernden Aktionen

In den Eventmethoden sollten bekanntlich keine länger dauernden Aktionen ausgeführt werden, da in dieser Zeit das GUI einfriert. Um dies zu vermeiden, erstellt man einen "WorkerThread", der die Aktion abarbeitet. Meist verwendet man dazu eine innere Klasse, damit auf alle Instanzvariablen der umgebenden Klasse zugegriffen werden kann. Als Beispiel schreiben wir als innere Klasse:

```
class WorkerThread extends Thread
{
  public void run()
  {
    while (true)
    {
      while (!isRunning) {}
      gPane1.clear();
      gPane1.move(0.5, 0.5);
      gPane1.fillCircle(0.3);
      .......Hier weiterer länger dauernder Code ...........
      isRunning = false;
    }
  }
}
```

und deklarieren in der Hauptklasse eine boolesche Variable

```
private volatile boolean isRunning = false;
```

(*volatile*, weil sie von zwei Threads verwendet wird.)

In der Eventmethode setzen wir das Flag auf true:

```
private void
  jButton1ActionPerformed(java.awt.event.ActionEvent evt)
  {
     isRunning = true;
  }
```

Den Thread erstellen und starten wir im Konstruktor:

```
public Demo()
{
  initComponents();
  new WorkerThread().start();
}
```

Der Thread wartet geduldig in der hängenden Schleife `while (!isRunning) {}` bis der Button geklickt wird. Nach Ende des Zeichnens kehrt man wegen des `while (true)` wieder dorthin zurück und die run-Methode wird nie abgebrochen. Das *gPane1.clear()* bewirkt, dass die Grafik immer neu gezeichnet wird. Ist man sich des Schreibens von *gPane1* überdrüssig, so kann im Eigenschaftsfenster des *GPane* unter der Karte *Quelltext* der Variablenname geändert werden, beispielsweise in *g*.

Als Ergänzung können weiter Felder ins Formular aufgenommen werden, beispielsweise ein Eingabefeld für Integer. Oft muss man die Eingabe überprüfen, damit sich kein Laufzeitfehler ergibt. Am einfachsten macht man dies durch Abfangen der *IllegalArgumentException* beim Umwandeln des Integers in einen String:

```
private void
  goBtnActionPerformed(java.awt.event.ActionEvent evt)
  {
    boolean isError = false;
    nb = 100000;
    try
    {
      nb = Integer.parseInt(countTField.getText());
    }
    catch (IllegalArgumentException ex)
    {
      isError = true;
    }
    if (nb < 1000)
      isError = true;

    if (isError)
       ...
   else
     ...
  }
```

Man beachte, dass grundsätzlich Swing-Methoden nur im Event Dispatch Thread (EDT) aufgerufen werden dürfen. Fügt man beispielsweise auf dem Formular ein *JLabel* mit dem Namen *statusLbl* als Statusfeld hinzu, so darf *statusLbl.setText()* nur entweder in einer Eventmethode oder aber über *invokeLater()* aufgerufen werden. Im zweiten Fall muss der Aufruf in einen Thread eingebaut werden. Am einfachsten ist es, wenn dieser anonym gemacht wird:

```
private void showStatus(String msg)
{
  SwingUtilities.invokeLater(
    new Runnable()
    {
      public void run()
      {
        statusLbl.setText(msg);
```

```
        }
    });
}
```

Im Package ch.aplu.util ist ab Version 1.54 eine Hilfsklasse *JRunner* enthalten. Sie ist in Analogie zu der Klasse *SwingWorker* aus dem Java API erstellt und vereinfacht das oben dargestellte Verfahren. Um eine Methode, beispielsweise *foo()* in einem eigenen Thread laufen zu lassen, erstellt man zuerst mit

```
JRunner jRunner = new JRunner(this);
```

eine Instanz und übergibt ihr eine Referenz auf die Klasse, in der *foo()* deklariert ist. Um sie vom neuen Thread ausführen zu lassen, ruft man

```
jRunner.run("foo");
```

auf. *foo()* darf keine Parameter haben. Manchmal benötigt man eine Notifikation, wenn die Methode zu Ende gelaufen ist. Dazu deklariert man eine Methode

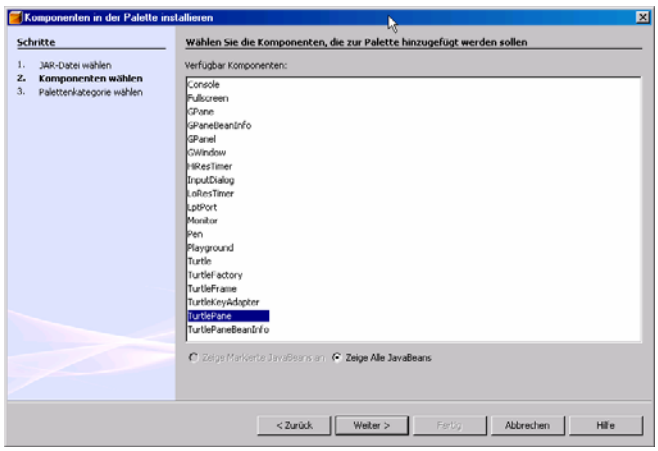
```
void done(String methodName)
{
  ...
}
```

Diese wird automatisch aufgerufen, wenn *foo()* zu Ende läuft. Dabei wird der Name der Methode zurückgegeben, damit man bei mehreren Aufrufen von *run()* die verschiedenen Methoden unterscheiden kann. Zudem ist gewährleistet, dass *done()* im Event Dispatch Thread (EDT) ausgeführt wird, damit sich Swing-Aufrufe direkt (ohne *SwingUtilities.invokeLater()* o.ä.) einbauen lassen.

6 Verwendung von TurtlePane als Komponente

Zur Einführung in eventgesteuerte Programme eignet sich die Turtle gut, insbesondere wenn man mit ihr rekursiv definierte Muster erzeugt.

Dazu wird wie bei *GPane* vorgegangen und zuerst die Komponente *TurtlePane* in die Palette der Swing-Komponenten aufgenommen. Diese ist ebenfalls in *aplu5.jar* enthalten.



Nach dem Einfügen der Komponente unter den Swing-Komponenten erstellt man wie üblich ein neues GUI Projekt. Im Entwurfsmodus erscheint in der Palette die neue Komponente *TurtlePane*. Man zieht diese auf das Formular. Als Eigenschaften kann nur die Hintergrundfarbe gewählt werden. Man beachte, dass es sich bei den Turtle-Koordinaten eigentlich um Pixelkoordinaten mit dem Nullpunkt in der Mitte der *TurtlePane* handelt. In einer *TurtlePane* in Standardgröße ist der Koordinatenbereich -200..200 in x- und y-Richtung. Verändert man die Größe der *TurtlePane* im Gui-Builder, so ändert sich der Koordinatenbereich entsprechend.

Öffnet man den Quelltext, so erkennt man als neue Komponente *turlePane1* vom Typ *TurtlePane*. Man fügt als erstes den Import des Package *ch.aplu.turtle.** hinzu. Um eine Turtle in die neue TurtlePane zu setzen, wird der Konstruktor *Turtle(TurtlePane tp)* verwendet. Man deklariert beispielsweise eine Instanzvariable

```
private Turtle t;
```

und initialisiert *t* im Konstruktor der Applikationsklasse

```
public GuiTurtle()
{
  initComponents();
  t = new Turtle(turtlePane1);
}
```

Da die *turtlePane1* erst in *initComponents()* initialisiert wird, darf die Initialisierung von *t* nicht bereits bei ihrer Deklaration erfolgen, also **nicht**

```
private Turtle t = new Turtle(turtlePane1);
```

Turtle-Aktionen sollten wie beim *GPanel* nicht im Konstruktor der Applikationsklasse ausgeführt werden, da sonst die Auswirkungen erst nach dem Beenden des Konstruktors sichtbar werden. Vielmehr wird entweder eine Eventmethode oder ein WorkerThread verwendet. Soll die Turtle bei jedem Klick auf den Button lediglich ein Segment eines Kreises (Vielecks) zeichnen, so setzt man den kurzen Code in die Eventmethode:

```
private
  void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
  {
    t.fd(20).lt(20);
  }
```

Größe und Hintergrundfarbe des *TurtlePane*s können im Entwurfsmodus des Gui-Builders verändert werden. (Die Farbänderung ist in der Formularansicht nicht sichtbar.)

Soll ein länger dauerndes rekursives Muster gezeichnet werden, so muss dies in einem WorkerThread erfolgen. Als Beispiel soll der bekannte Binärbaum gezeichnet werden. Man verwendet dazu die bekannte Methode *baum()*:

```
void baum(int s)
 {
```

```
  if (s < 16)
    return;
  t.forward(s);
  t.left(45);
  baum(s / 2);
  t.right(90);
  baum(s / 2);
  t.left(45);
  t.back(s);
}
```

Wie oben beschrieben, deklariert man eine innere Klasse

```
class WorkerThread extends Thread
{
  public void run()
  {
    while (true)
    {
      while (!isRunning) {}
      t.clear();
      baum(128);
      isRunning = false;
    }
  }
}
```

Im Konstruktor wird die Turtle auf die Anfangsposition gesetzt und der Thread losgelassen:

```
public GuiTurtle()
{
  initComponents();
  t = new Turtle(turtlePane1);
  t.setPos(0, -64);
  new WorkerThread().start();
}
```

Das boolesche Flag

```
private volatile boolean isRunning = false;
```

steuert den Ablauf. In der Eventmethode setzen wir das Flag auf *true*:

```
private void
  jButton1ActionPerformed(java.awt.event.ActionEvent evt)
  {
     isRunning = true;
  }
```

Am Ende des Zeichnens eines Baums wird es wieder auf *false* gesetzt und der Thread wartet in einer Endlosschleife.

Statt der uneleganten und ressourcen-verbrauchenden Endlosschleife kann man den Thread mit *wait()* in Schlaf versetzen und ihn in der Eventmethode mit *notify()* wieder aufwecken. Am einfachsten verwendet man dazu die Methoden *putSleep()* und *wakeUp()* aus der Klasse *ch.aplu.util.Monitor*. Man importiert sie

```
import ch.aplu.util.Monitor;
```

und ändert die run-Methode von WorkerThread:

```
class WorkerThread extends Thread
{
  public void run()
  {
    while (true)
    {
      Monitor.putSleep();
      t.clear();
      baum(128);
    }
  }
}
```

Die Eventmethode weckt mit *wakeUp*() den schlafenden Thread:

```
private
  void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
  {
    Monitor.wakeUp();
  }
```

Das boolesche Flag *isRunning* ist überflüssig. Es ist nicht jedermanns Sache, eigene Threads zu deklarieren und den wait()-notify()-Mechanismus einzusetzen. Die Klasse *JRunner* übernimmt diese Aufgabe. Zuerst verlagert man das Zeichnen in eine neue parameterlose Methode, beispielsweise

```
void draw()
{
  t.clear();
  baum(64);
}
```

Als nächstes deklariert und initialisiert man mit

```
private JRunner jRunner = new JRunner(this);
```

eine JRunner-Referenz und ruft deren run-Methode in der Eventmethode des Buttons auf:

```
private
  void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
  {
    jRunner.run("draw");
  }
```

Das ist bereits alles, was zu tun ist. Will man sogar auf eine Eventmethode verzichten, so kann *jRunner.run("draw")* auch bereits im Konstruktor aufgerufen werden.

Um ein sauberes GUI zu erhalten, sollte man verhindern, dass der Button gedrückt wird, solange die Turtle noch am Zeichnen ist. Am einfachsten inaktiviert man ihn in der Eventmethode mit *setEnabled(false)*:

```
private
  void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
  {
    jButton1.setEnabled(false);
    jRunner.run("draw");
  }
```

Jetzt hat man aber das Problem, dass er am Ende der Zeichnung wieder aktiviert werden muss. Dazu setzt man die Notifikationmethode *done()* ein und deklariert

```
private void done(String methodName)
{
  jButton1.setEnabled(true);
}
```

Der Parameter *methodName* wird dabei nicht verwendet. Das Programmlisting ist im Anhang enthalten.

7 Weitere Dokumentationen und Hilfen

www.netbeans.org/kb/50/quickstart-gui.html : Einführung in den GUI-Builder
www.netbeans.org/kb/articles/gui-functionality.html : Einführung über Event-Methoden
wiki.netbeans.info/wiki/view/NetBeansUserFAQ : NetBeans FAQ (unter GUI Editor)

7 Anhang: Programmlisting

Die vom Gui-Builder erzeugte Methode *initComponents()* ist nicht aufgeführt.

```
// GuiTurtle.java

import ch.aplu.turtle.*;
import ch.aplu.util.JRunner;

public class GuiTurtle extends javax.swing.JFrame
{
  public GuiTurtle()
  {
    initComponents();
```

```
    t = new Turtle(turtlePane1);
    t.setPos(0, -64);
  }

  private void draw()
  {
    t.clear();
    baum(128);
  }

  void baum(int s)
  {
    if (s < 16)
      return;
    t.forward(s);
    t.left(45);
    baum(s / 2);
    t.right(90);
    baum(s / 2);
    t.left(45);
    t.back(s);
  }

  private void done(String methodName)
  {
    jButton1.setEnabled(true);
  }


  // Vom Gui-Builder erzeugt
  private void initComponents()
  {
    ...
  }

  public static void main(String args[])
  {
    java.awt.EventQueue.invokeLater(new Runnable()
    {
      public void run()
      {
        new GuiTurtle().setVisible(true);
      }
    });
  }


  private javax.swing.JButton jButton1;
  private ch.aplu.turtle.TurtlePane turtlePane1;
  private Turtle t;
  private JRunner jRunner = new JRunner(this);
}
```