JavaSpass

brew it, do it, enjoy it . . .

Java - weil es Spass macht

Aegidius Plüss
a.pluess@swissonline.ch

Die Sourcen der Programme und die Klassenbibliotheken befinden sich auf meiner Website: *www.aplu.ch/java*.

Die Türme von Hanoi

Die Legende geht auf unseren Kollegen, Edourd Lucas, Mathematiklehrer am Licée Saint-Louis (1883) zurück. Sie lautet:

Im grossen Tempel von Benares, der den Mittelpunkt der Welt bezeichnet, stehen drei diamantene Säulen. Auf einer von diesen hat der Herr zu Beginn der Zeiten einen Turm mit 64 goldenen Scheiben aufgebaut. Dieser Turm ist Brahma geweiht. Tag und Nacht sind die Tempelpriester damit beschäftigt, den Turm nach folgenden Regeln umzubauen:

- *Der Turm ist auf einer anderen Säule zu errichten*
- *Die Scheiben dürfen nur einzeln von Säule zu Säule verschoben werden*
- *Keine Scheibe darf je über eine kleinere zu liegen kommen.*

Wenn das Werk vollendet ist, zerfallen Turm und Priester zu Staub und das Ende der Welt ist gekommen.

Man formuliert bekanntlich den Turmumbau mit den vorgegebenen `n` Scheiben rekursiv, indem man ihn auf dasselbe Problem mit `n-1` Scheiben zurückführt. Dabei verschiebt man die `n-1` oberen Scheiben von der Quell-Säule auf die Hilfs-Säule unter Verwendung der Ziel-Säule, nachfolgend schiebt man die unterste Scheibe von der Quell-Säule auf die Ziel-Säule. Schliesslich verschiebt man die `n-1` Scheiben von der Hilfs-Säule auf die Ziel-Säule unter Verwendung der Quell-Säule. Wir verankern die Rekursion, wenn nur eine Scheibe zu verschieben ist.

Im Java-Programm, das wir als Swing-Applet mit einem eigenen Fenster konzipieren, modellieren wir die drei Säulen durch drei Stacks. Dabei handelt es sich bekanntlich, analog zu einem Bücherstapel, um einen Speicher nach dem Prinzip von **First-In-Last-Out** (**FILO**). Diese Datenstruktur ist unserem Problem perfekt angepasst. Mit der Methode `push()` werden Scheiben oben auf den Stack gebracht, mit `pop()` wird die oberste Scheibe wieder zurückgeholt. Wir führen die Rekursion in der Methode `go()` durch.

Nicht ganz so einfach ist es, das Applet so zu verfassen, dass die intensive und langdauernde Arbeit der Mönche nicht zu einer Beeinträchtigung der Reaktionszeiten des Computers führt. Da ein Applet im gleichen Thread wie der Browser läuft, vermeiden wir eine Beeinflussung des Browsers, indem wir die Baustelle in einen anderen Thread verlegen. Weil wir unsere Klasse `Hanoi` bereits aus `JApplet` ableiten müssen, bleibt uns wegen der Einfachvererbung von Java nichts anderes üblich, als `Runnable` zu implementieren. In der run-Methode des Threads rufen wir die rekursive Methode `go()` auf.

Jetzt bleibt uns noch die etwas heikle, aber wichtige Aufgabe, den Baustellenthread zu beenden, wenn der Benutzer mit dem Close-Button das Ende der Welt nicht abwarten will oder wenn er die Webseite verlässt. Da man einen Thread nicht einfach mir nichts - dir nichts stoppen („töten“) kann, setzen wir in beiden Fällen ein Flag `active` auf `false` und brechen dadurch sowohl die Rekursion wie die run-Methode und damit den Thread ab. Den Klick auf den Close-Button müssen wir durch die Implementierung eines `ExitListener`, welcher die Callbackmethode `notifyExit()` zur Verfügung stellt, abfangen. Dieser Callback wird von der Klasse `GPanel` aufgerufen, falls wir unseren `ExitListener` mit `addExitListener()` registrieren.

Das GUI besteht aus einem Textfeld, in dem man zu Beginn die Anzahl der Scheiben eingeben kann und einem Button, mit dem man die Simulation startet. Der aktuelle Zustand wird laufend grafisch darstellt.

```
// Hanoi.java
```

```
import java.awt.*;
import javax.swing.*;
import ch.aplu.util.*;
import java.awt.event.*;
import java.util.Stack;

public class Hanoi extends JApplet implements Runnable,
    ExitListener
{
// ------------------ Inner class Disk --------------------
  private class Disk
  {
    private int size;
    private Color color;

    public Disk(int diskSize)
    {
      size = diskSize;
      color = new Color((1 * size) % 256, (40 * size) % 256,
                        (40 * size) % 256);
    }
  }

// ----------- Inner class MyButtonListener ---------------
  private class MyButtonListener implements ActionListener
  {
    public void actionPerformed(ActionEvent evt)
    {
      if (play) // Inhibit button when playing
        return;
      nbDisks = Integer.parseInt(textField.getText());
      for (int i = 0; i < 3; i++)
        towers[i] = new Stack();
      for (int i = nbDisks; i > 0; i--)
        towers[0].push(new Disk(i));
      p.window(0, 6 * nbDisks, 0, 3 * nbDisks);
      play = true;
    }
  }

// ------------- Instance variables ----------------------
  private GPanel p;
  private int nbDisks;
  private int nbMoves;
  private Stack[] towers = new Stack[3];
  private JTextField textField = new JTextField(3);
  private JLabel label = new JLabel("Number of disks");
  private JButton goButton = new JButton("Go");
  private boolean play;
  private volatile boolean active;
```

```
  public void start()
  {
    p = new GPanel(GPanel.APPLETFRAME);
    p.addExitListener(this);
    p.title("Towers of Hanoi");
    p.bgColor(Color.yellow);
    p.add(label);
    p.add(textField);
    p.add(goButton);
    goButton.addActionListener(new MyButtonListener());
    p.validate();
    textField.setText("5"); // Default
    textField.requestFocus();
    nbMoves = 0;
    play = false;
    new Thread(this).start();
  }

  public void stop()
  {
    active = false;
    p.dispose();
  }

  public void run()
  {
    active = true;
    while (active)
    {
      if (play)
      {
        draw();
        go(towers[0], towers[1], towers[2], nbDisks);
        nbMoves = 0;
        play = false;
      }
    }
  }

  public void notifyExit()
  {
    active = false;
    p.dispose();
  }

  private void draw()
  {
    p.clear();
    p.color(Color.black);
    p.move(0, 0);
    p.text("Number of moves: " + nbMoves);
```

```
    for (int i = 0; i < 3; i++)
    {
      p.color(Color.black);
      p.move(nbDisks * (2 * i + 1), 1.5 * nbDisks);
      p.fillRectangle(0.2, nbDisks + 1);
    }
    for (int i = 0; i < 3; i++)
    {
      for (int k = towers[i].size() - 1; k >= 0; k--)
      {
        Disk current = (Disk)towers[i].elementAt(k);
        p.color(current.color);
        p.move((2 * i + 1) * nbDisks, nbDisks + k);
        p.fillRectangle(2 * current.size, 0.9);
      }
    }
  }

  private void go(Stack source, Stack help, Stack target,
                  int n)
  {
    if (!active)
      return;

    if (n == 1)
    {
      nbMoves++;
      target.push(source.pop());
      Console.delay(1000);
      draw();
    }
    else
    {
      go(source, target, help, n - 1);
      go(source, towers[0], target, 1);
      go(help, source, target, n - 1);
    }
  }

  public static void main(String[] args)
  {
    new Hanoi().start();
  }
}
```

Für die Herstellung des Archivs müssen wir mit

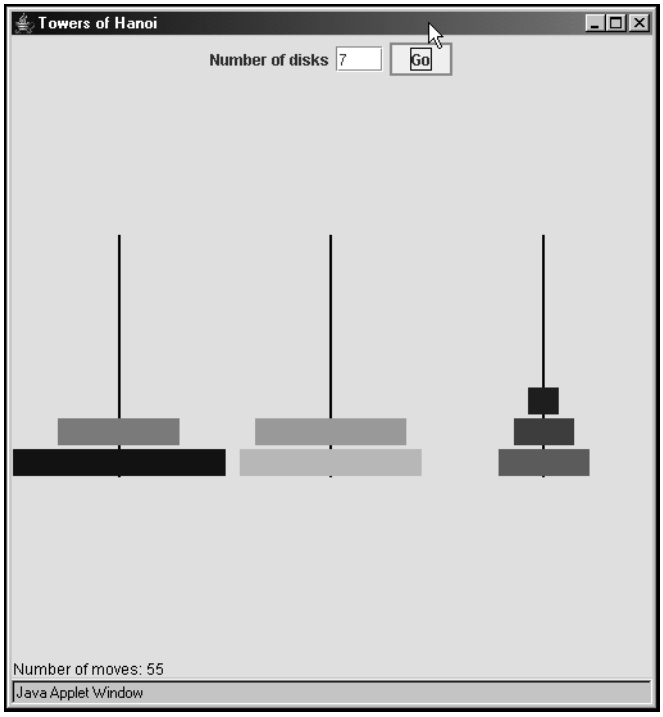
```
jar cvf hanoi.jar
   Hanoi.class Hanoi$Disk.class
```

`Hanoi$MyButtonListener.class`

auch die Class-Dateien der inneren Klassen in die jar-Datei `hanoi.jar` verpacken. Zur Ausführung genügt eine einfache HTML-Datei `Hanoi.html`.

```
<html>
  <body>
    <h2>Towers of Hanoi</h2>
    <applet code="Hanoi.class" archive="aplu.jar, hanoi.jar"
       width="0" height="0">
    </applet>
  </body>
</html>
```

Die Abbildung zeigt eine typische Situation anlässlich einer kurzen Verschnaufpause der Mönche. Das Applet kann unter *www.aplu.ch/jex/applets* gefunden und ausgeführt werden, vorausgesetzt dass das Java Runtime Environment (JRE) Version 1.4 oder höher installiert ist. Dem aufmerksamen Leser ist nicht entgangen, dass das Programm eine main-Methode enthält, die als Applet völlig überflüssig ist, aber auch nicht stört. Ohne jede Änderung kann man das Programm damit auch als gewöhnliche Applikation unabhängig von einem Web-Browser verwenden und hat damit zwei Fliegen auf einen Schlag erwischt.



Mit diesem Beitrag schliesse ich die Reihe *JavaSpass* ab. Aus all der Freude an Java gebe ich beim Oldenbourg-Verlag ein Lehrbuch mit dem Titel *Java exemplarisch* heraus, das im Herbst 2004 erscheinen wird (ISBN 3-486-20040-2). Es soll sowohl Anfänger wie Fortge-

schrittene in die 1001 Geheimnisse des Programmierens einführen, und zwar mit über 300 praxisbezogenen Programmen, die sich leicht an eigene Problemstellungen anpassen lassen.

Lassen Sie sich von Java verführen.