

Java-Programmierung



Thread-Programmierung und Animationen

Autor: Dipl.-Math. E. Engelhardt

Stand: 08. Juni 2009



Inhalt

- 1 Entwicklung des Multi-Tasking
- 2 Parallelität - Nebenläufigkeit
 - 2.1 Multitasking - Multiuser
 - 2.2 Prozesse - Threads
 - 2.3 Timesharing – Prozesswechsel
 - 2.4 Robin-Round-Steuerung
 - 2.5 Prozess-Zustände
 - 2.6 Serialisierung
- 3 Java-Threads
 - 3.1 Thread-Zustände (alt)
 - 3.2 Thread-Zustände
 - 3.3 Lebenszyklus des Applet
 - 3.4 Aufrufreihenfolge bei „repaint“
- 4 Stufen des Mond-Applet
 - 4.1 Schritte im DOS-Fenster
 - 4.2 HTML-Datei des Mond-Applet
 - 4.3 Grundgerüst des Mond-Applet
 - 4.4 Stufe 1:
Attribute einfügen
 - 4.5 Stufe 2:
Methode „init“
 - 4.6 Stufe 3:
Methoden „start“ und „stop“
 - 4.7 Stufe 4:
Methode „run“ des Thread
 - 4.8 Stufe 5:
Methoden „paint“ und „update“

Abkürzungen

- ◆ API Application Programming Interface
- ◆ CPU Central Processing Unit
- ◆ IPC Inter Process Communication
- ◆ JVM Java Virtual Machine
- ◆ URL Uniform Resource Locator

1 Entwicklung des Multi-Threading

- ◆ Ausschließliches Nutzungsrecht eines Nutzers mit einem Programm bei den ersten Rechnern (viel Zeit geht für Ergebnisanalyse und Fehlersuche verloren)
- ◆ ab 1955 Stapelbetrieb, d.h. Programm und Ergebnisse wurden auf Lochstreifen / Lochkarten gestanzt (Jobs)
- ◆ Einführung von Vorrechnern, die die Lochbänder / Lochkarten auf Magnetbänder übertrugen
- ◆ ab 1965 Mehrprogrammbetrieb (Multi-Programming) und dadurch bessere Ausnutzung des Prozessors
- ◆ Einsatz der Rechner für Echtzeitbetrieb
- ◆ Intelligente periphere Geräte ermöglichten Dialog- und Mehrnutzerbetrieb
- ◆ in den 80iger Jahren entstand das Thread-Konzept

2 Parallelität - Nebenläufigkeit

- ◆ Parallelität wird durch Hardware und Betriebssystem realisiert
- ◆ „echte“ Parallelität wird durch Einsatz mehrerer Prozessoren realisiert (z.B. Windows NT)
- ◆ bei Rechnern mit nur einem Prozessor wird Quasi-Parallelität (Pseudo-Parallelität) realisiert, indem mit hoher Frequenz zwischen den Programmen umgeschaltet wird (z.B. ab Windows 95)
- ◆ Das Betriebssystem MS-DOS ist ein „Single-Tasking“-Betriebssystem
- ◆ Mit dem grafischen Aufsatz „Win 3.x“ ist sogenanntes „kooperatives“ Multi-Tasking möglich, d.h. die Programme geben freiwillig die Steuerung zurück (sie verhalten sich kooperativ)

2.1 Multitasking - Multiuser

- ◆ Multitasking-Betriebssysteme sind Betriebssysteme mit „echter“ Parallelität (Präemptives Multitasking)
- ◆ Multi-User-Betriebssysteme sind Betriebssysteme, bei denen mehrere Nutzer **gleichzeitig** arbeiten können
- ◆ Voraussetzung für Multi-User ist die Multi-Tasking-Fähigkeit (Windows 9x und Windows NT Workstation sind **nicht** Multi-User-fähig)
- ◆ UNIX ist von Anfang an als Multi-User-Betriebssystem konzipiert
- ◆ Wenn mehrere Nutzer eingerichtet werden können, dann handelt es sich um ein Mehrbenutzer-System
- ◆ in den 80iger Jahren wurde UNIX um das Thread-Konzept ergänzt

2.2 Prozesse - Threads

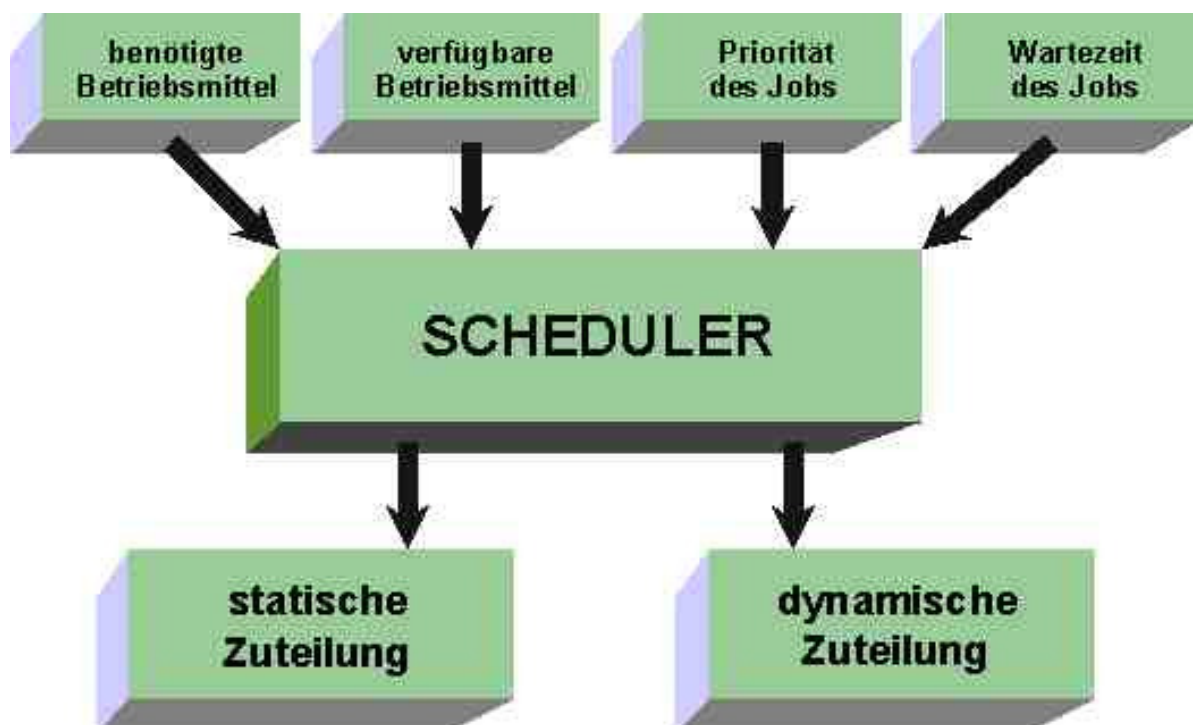
- ◆ Die Begriffe „Task“ und „Prozess“ sind gleichbedeutend, nicht aber „Thread“ und „Prozess“
- ◆ Programme (ausführbare Dateien) sind eingebettet in einen Prozess mit eigener Prozessumgebung (Adressraum, Prozess-Variablen, Tabelle offener Dateien mit Bearbeitungszuständen)
- ◆ Prozesse kommunizieren untereinander über „Inter Prozess Communication“ (IPC-Mechanismus: Pipes, Semaphoren, Shared-Memory, Signale, Queues)
- ◆ Threads sind Programmstücke (Fäden) ohne eigenen Adressraum (auch Leichtgewichtsprozesse)
- ◆ Das Umschalten zwischen Threads kann sehr einfach erfolgen (nur Retten der Register notwendig)
- ◆ alle Threads arbeiten mit den selben Daten

Threads

- ◆ Betriebssysteme ohne Thread-Konzept realisieren Threads über Prozesse
- ◆ Ein Prozess enthält mindestens einen Thread, d.h. Threads sind immer Teil eines Prozesses
- ◆ Durch Threads lassen sich verschiedene Teilaufgaben eines Programms flexibel lösen
 - reagieren auf Ereignisse
 - Zeitgeber (z.B. für Uhr)
 - Berechnungen im Hintergrund ausführen
 - usw.
- ◆ Für die Synchronisation der Threads ist der Programmierer verantwortlich
 - Verhindern von Deadlocks
 - Zugriff auf Ressourcen

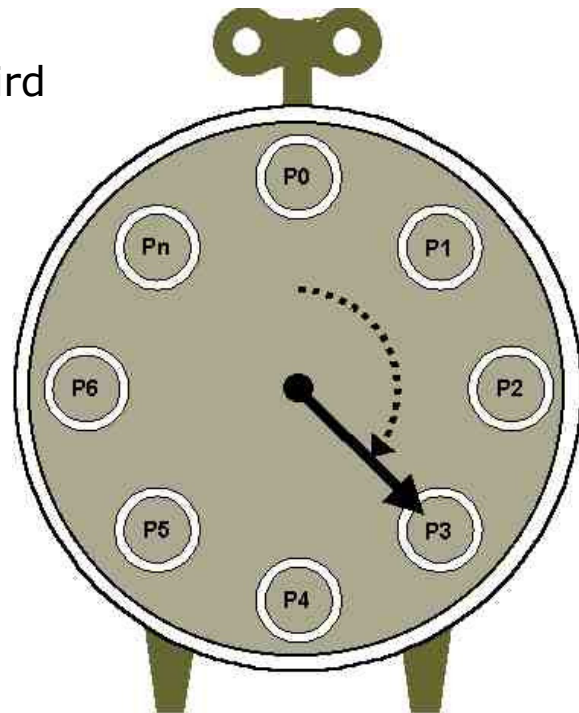
2.3 Timesharing - Prozesswechsel

- ◆ Der Scheduler ist Teil des Betriebssystems und zuständig für die Zuteilung der Betriebsmittel



2.4 Robin-Round-Steuerung

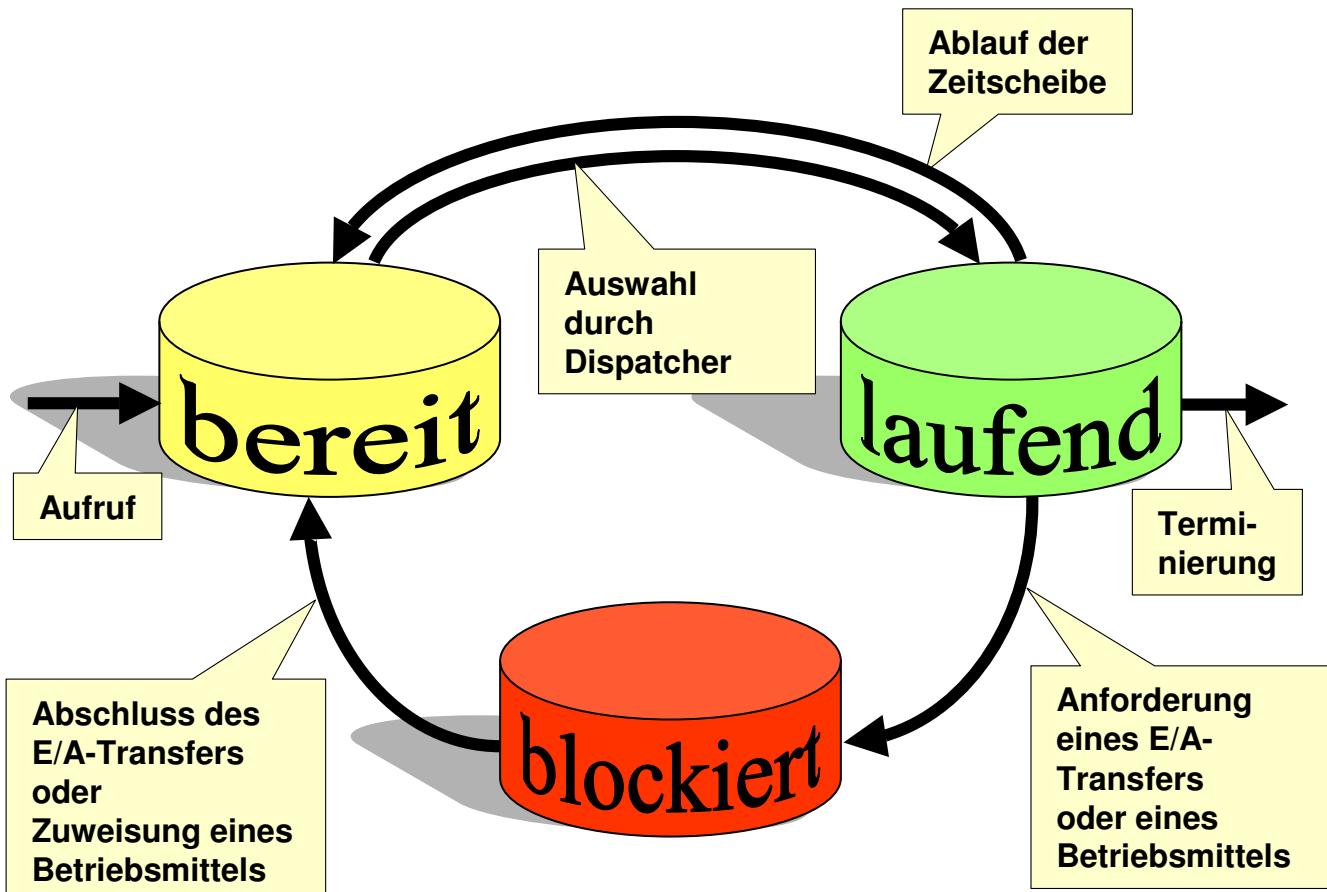
- ◆ Mit Hilfe der Zeitscheibensteuerung wird den Prozessen CPU-Zeit zugeteilt
- ◆ Diese Art der Zeitzuteilung wird meist mit Prioritätensteuerung gekoppelt



Folie 11 von 27

Threads und Animationen

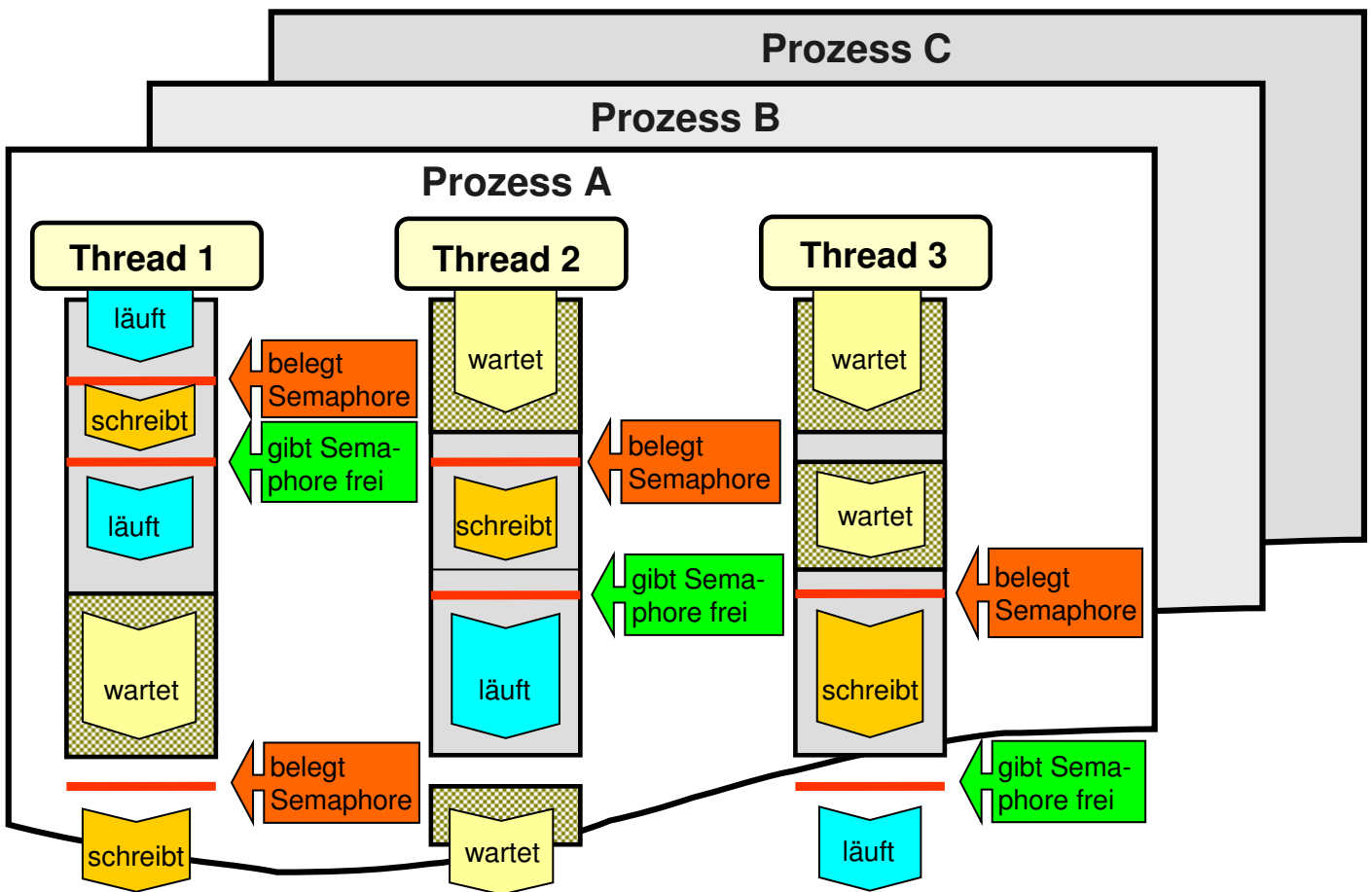
2.5 Prozess-Zustände



Folie 12 von 27

Threads und Animationen

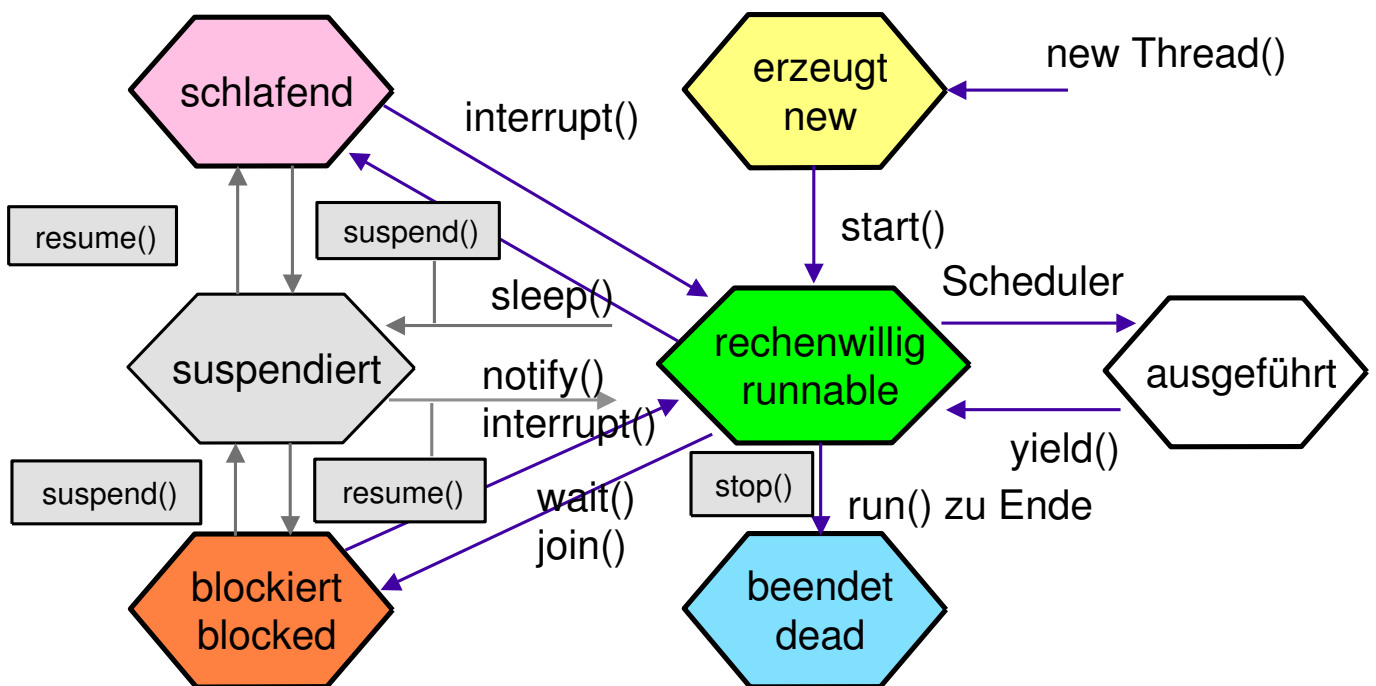
2.6 Serialisierung



Folie 13 von 27

Threads und Animationen

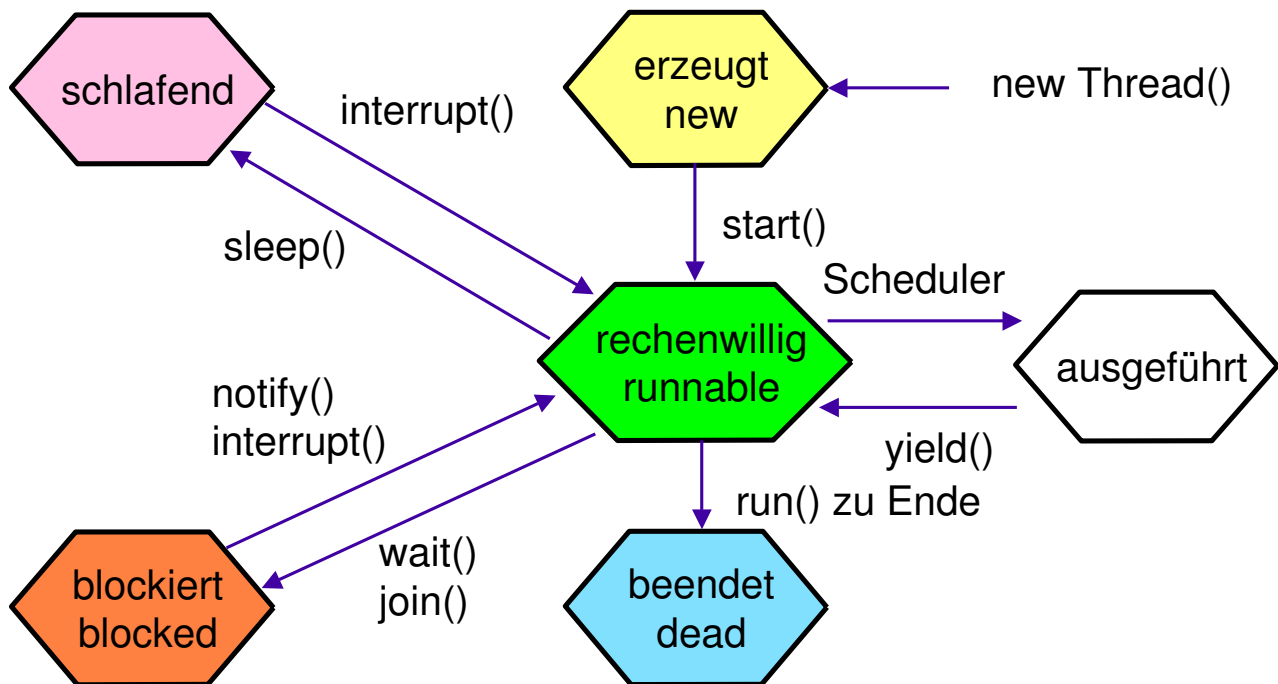
3 Java-Threads 3.1 Thread-Zustände (alt)



Folie 14 von 27

Threads und Animationen

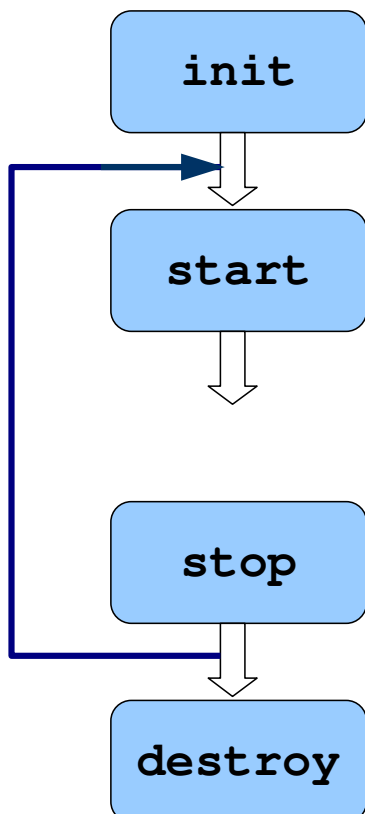
3.2 Thread-Zustände



Folie 15 von 27

Threads und Animationen

3.3 Lebenszyklus des Applet



Wird vom Browser aufgerufen wenn Applet zum ersten Mal geladen wird (ähnlich Konstruktor bei Applikationen)

nach Ausführung von „init“ wird automatisch „start“ ausgeführt und damit das Applet gestartet; wird auch aufgerufen, wenn Applet wieder sichtbar wird

hält Applet an, wenn HTML-Seite verlassen wird oder anderes Programm in den Vordergrund gesetzt wird (Applet wird vorübergehend unsichtbar)

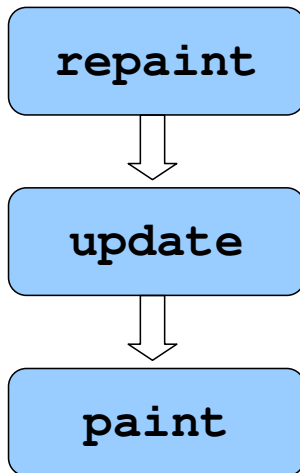
Aufruf, wenn neue Seite geladen wird oder wenn Browser geschlossen wird

Folie 16 von 27

Threads und Animationen

3.4 Aufruffreihenfolge bei „repaint“

- ◆ Die Methode „paint“ ist bei allen Flächen, auf denen gezeichnet werden kann, vorhanden (Frame, Panel, Canvas, Applet, ...).
- ◆ Sie wird automatisch aufgerufen, wenn sich irgend etwas an der Zeichenfläche ändert (Größenänderung, Minimieren und danach wieder herstellen, Überdecken und wieder sichtbar machen, ...)
- ◆ Durch den Programmierer wird „paint“ indirekt über „repaint“ aufgerufen (dazwischen liegt der Aufruf von „update“)



Erzwingt Neuzeichnen (oft bei Animationen)

Durch „update“ wird u.a. der Bildschirmhintergrund gelöscht (was manchmal nicht gewünscht wird). Wenn etwas anderes getan werden soll, muss „update“ überschrieben werden

Es wird das gezeichnet, was in der Methode „paint“ implementiert ist

4 Stufen des Mond-Applets

- ◆ Grundgerüst: leeres Applet-Fenster
- ◆ Stufe 1: Attribute einfügen
- ◆ Stufe 2: Methode „init“ des Applet
- ◆ Stufe 3: Methoden „start“ und „stop“ des Applet
- ◆ Stufe 4: Methode „run“ des Thread
- ◆ Stufe 5: Methoden „paint“ und „update“ des Applet, um Flackern zu verhindern

4.1 Schritte im DOS-Fenster

```
...> edit Mond.html  
(nur einmal erforderlich)
```

```
...> edit Mond.java
```

```
...> javac Mond.java
```

Die HTML-Datei kann mit einem Browser geöffnet werden. Dabei werden allerdings auftretende Fehler eventuell nicht angezeigt! Außerdem sollten Applets zumindest mit dem zum sdk mitgelieferten Programm fehlerfrei ausführbar sein. Das garantiert allerdings nicht, dass das Applet dann auch mit allen Browsern und Viewern fehlerfrei läuft. Deshalb besser:

```
...> appletviewer Mond.html
```

4.2 HTML-Datei des Mond Applet

```
<html>  
<head>  
<title>Mond Applet</title>  
</head>  
<body bgcolor=white>  
<applet code="Mond.class" height="300" width="400">  
</applet>  
</body>  
</html>
```

4.3 Grundgerüst des Mond-Applet

```
import java.awt.*;
import java.applet.*;

public class Mond extends Applet      4a
{
    1 // Attribute

    public void init() {} ← 2
    public void start() {} ← 3a
    public void stop() {} ← 3b

    4b // Methode "run" des Thread

    public void paint(Graphics g) {} ← 5a

    5b // Methode "update"
}
```

4.4 Stufe 1: Attribute einfügen

1

```
// Attribute
boolean runFlag; // steuert Thread

int x, y, Hoehe, Breite; // des Applet-Fensters

Image Bild; // Bildspeicher
Graphics g; // Grafik-Kontext

Color NachtFarbe = new Color(0, 0, 102);
Color MondFarbe = new Color(216, 202, 207);

Thread t = null; // Thread-Deklaration
```

4.5 Stufe 2: Methode „init“

2

```
public void init()
{
    Dimension d = getSize(); // aus HTML-Datei

    Breite = d.width;        // Breite und Hoehe
    Hoehe = d.height;       // des Applet-Fensters
    Bild = createImage(Breite, Hoehe);
    g = Bild.getGraphics();

    x = Breite / 2;         // Mitte des Fensters
    y = Hoehe / 2;

}
```

4.6 Stufe 3: Methoden „start“ und „stop“

3a

```
public void start()
{
    if (t == null)
    {
        t = new Thread(this);

        t.start(); // Thread wird gestartet
    }             // Methode „run“ wird
                 // ausgeführt
}
```

3b

```
public void stop()
{
    if (t != null)
    {
        runFlag = false;
        t = null;
    }
}
```

4.7 Stufe 4: Methode „run“ des Thread

4b

```
public void run ()
{
    runFlag = true;

    while (runFlag)
    {
        g.setColor(NachtFarbe);
        g.fillRect(0, 0, Breite, Hoehe);
        g.setColor(MondFarbe);
        g.fillArc(x, y-25, 150, 150, 270, 180);
        repaint(); // Bild neu zeichnen
        x += 1; // ein Pixel pro Schritt
        if (x > (Breite + 50)) x = -50;

        try { t.sleep(100); } // 100 Millisekunden
        catch (InterruptedException e) {}
    }
}
```

4a

implements Runnable

```
// "run" des Thread
// zur Threadsteuerung

// Bild neu zeichnen
// ein Pixel pro Schritt
// 100 Millisekunden
catch (InterruptedException e) {}
```

4.8 Stufe 5: Methoden „paint“ und „update“

5a

```
public void paint(Graphics g)
{
    if (Bild != null)
        g.drawImage(Bild, 0, 0, null);
}
```

5b

```
// Methode „update“ wird überschrieben
public void update(Graphics g)
{
    paint(g); // Bildhintergrund löschen
              // wird verhindert
}
```

Literaturangaben

- [1] Daniel Basler: *Anwendungsentwicklung mit Java*,
Computer & Literatur Verlag GmbH,
ISBN 3-932311-68-x
- [2] Rainer Oechsle: *Parallele Programmierung mit Java Threads*,
Fachbuchverlag Leipzig im Carl Hanser Verlag
ISBN 3-446-21780-0
- [3] **Java 1.1** *Fortgeschrittene Programmierung*,
HERDT-Verlag für Bildungsmedien GmbH, Nackenheim